# Development and Evaluation of a Holistic, Cloud-driven and Microservices-based Architecture for Automated Semantic Annotation of Web Documents

**Oluwasegun Adedokun Adedugbe**

**Staffordshire University**

**A Doctoral Thesis submitted in partial fulfilment of the requirements of Staffordshire University for the degree of Doctor of Philosophy**

**August 2019**

# Abstract

The Semantic Web is based on the concept of representing information on the web such that computers can both understand and process them. This implies defining context for web information to give them a well-defined meaning. Semantic Annotation defines the process of adding annotation data to web information for the much-needed context. However, despite several solutions and techniques for semantic annotation, it is still faced with challenges which have hindered the growth of the semantic web. With recent significant technological innovations such as Cloud Computing, Internet of Things as well as Mobile Computing and their various integrations with semantic technologies to proffer solutions in IT, little has been done towards leveraging these technologies to address semantic annotation challenges. Hence, this research investigates leveraging cloud computing paradigm to address some semantic annotation challenges, with focus on an automated system for providing semantic annotation as a service. Firstly, considering the current disparate nature observable with most semantic annotation solutions, a holistic perspective to semantic annotation is proposed based on a set of requirements. Then, a capability assessment towards the feasibility of leveraging cloud computing is conducted which produces a Cloud Computing Capability Model for Holistic Semantic Annotation. Furthermore, an investigation into application deployment patterns in the cloud and how they relate to holistic semantic annotation was conducted. A set of determinant factors that define different patterns for application deployment in the cloud were identified and these resulted into the development of a Cloud Computing Maturity Model and the conceptualisation of a "Cloud-Driven" development methodology for holistic semantic annotation in the cloud. Some key components of the "Cloud-Driven" concept include Microservices, Operating System-Level Virtualisation and Orchestration. With the role Microservices Software Architectural Patterns play towards developing solutions that can fully maximise cloud computing benefits; **CloudSea**: a holistic, cloud-driven and microservices-based architecture for automated semantic annotation of web documents is proposed as a novel approach to semantic annotation. The architecture draws from the theory of "Design Patterns" in Software Engineering towards its design and development which subsequently resulted into the development of twelve Design Patterns and a Pattern Language for Holistic Semantic Annotation, based on the CloudSea architectural design. As proof-of-concept, a prototype implementation for CloudSea was developed and deployed in the cloud based on the "Cloud-Driven" methodology and a functionality evaluation was carried out on it. A comparative evaluation of the CloudSea architecture was also conducted in relation to current semantic annotation solutions; both proposed in academic literature and existing as industry solutions. In addition, to evaluate the proposed Cloud Computing Maturity Model for Holistic Semantic Annotation, an experimental evaluation of the model was conducted by developing and deploying six instances of the prototype and deploying them differently, based on the patterns described in the model. This empirical investigation was implemented by testing the instances for performance through series of API load tests and results obtained confirmed the validity of both the "Cloud-Driven" methodology and the entire model.

**Keywords**: Semantic Web, Semantic Annotation, Holistic Semantic Annotation, Cloud Computing, Cloud Computing Capability Model, Cloud Computing Maturity Model, Cloud-Driven, CloudSea, Microservices, Design Patterns.

# Acknowledgements

Firstly, I will like to give God thanks for His many blessings and the opportunity to have embarked on this academic journey as well as the grace to bring it to completion. Father in Heaven may your name be praised forever, Amen.

Sincere gratitude goes to my principal supervisor Professor Elhadj Benkhelifa for his immense contribution to the success of this work. Your support, encouragement and advice are so priceless. I also wish to express my sincere appreciation to my second supervisor Dr. Russell Campion for his continued support in diverse ways all through this journey. I wish to express my gratitude as well to the authorities at Staffordshire University for the scholarship I received towards tuition for the entire period of my PhD studies. Many thanks!

Furthermore, I wish to thank my family; my darling wife for all her support and ‘holding forth’ while I pursued this degree – thanks sweetheart, I appreciate it all! Thanks to my sweet mother for all her prayers and words of encouragement. You’re such a wonderful mummy! Thanks to my lovely daughter; Michelle and dear son; Jayden. Thanks to my dear sisters, brother, sisters-in-law and brothers-in-law for all their support and encouragement. Thanks to my Parents-in-law for their support, encouragement and prayers.

Thanks to my family at large, to friends and well-wishers too numerous to mention in every way you have offered some support!

# Dedication

This is dedicated to my late Father; Prince Johnson Adedapo Adedugbe who passed on to glory on 22nd September 2018, less than a year to the completion of my PhD studies. His desire for me to obtain this degree is beyond what my words can describe, and he so much looked forward to celebrating its completion with me but that was not meant to be.

Your passion for quality education and decency of lifestyle are great values that I will always remember and appreciate. Thanks for being such a great daddy, Dad.

# List of Publications

## Conferences:

Adedugbe, O., Benkhelifa, E. and Campion, R., 2017, October. Towards Cloud Driven Semantic Annotation. In *2017 IEEE/ACS 14th International Conference on Computer Systems and Applications (AICCSA)* (pp. 1378-1384). IEEE.

Adedugbe, O., Benkhelifa, E. and Campion, R., 2018, October. A Cloud-Driven Framework for a Holistic Approach to Semantic Annotation. In *2018 Fifth International Conference on Social Networks Analysis, Management and Security (SNAMS)* (pp. 128-134). IEEE.

Benkhelifa, E., Rowe, E., Kinmond, R., Adedugbe, O.A. and Welsh, T., 2014, August. Exploiting Social Networks for the Prediction of Social and Civil Unrest: A Cloud-Based Framework. In *2014 International Conference on Future Internet of Things and Cloud* (pp. 565-572). IEEE.

## Journals:

Adedugbe, O., Benkhelifa, E. and Campion, R., Jayawickrama, U., 2019. Leveraging Cloud Computing for the Semantic Web: Review and Trends. *Soft Computing Journal* (Submitted)

# Table of Contents

# List of Figures

# List of Tables

# List of Abbreviations

| | |
|---|---|
| ABAC | Attribute-Based Access Control |
| API | Application Programming Interface |
| ASMOV | Automated Semantic Matching of Ontologies with Verification |
| CPU | Central Processing Unit |
| CREAM | Creating Metadata |
| CRF | Conditional Random Fields |
| C-PANKOW | Context-driven Pattern-based Annotation through Knowledge on the Web |
| CSP | Cloud Service Provider |
| CSS | Cascading Style Sheets |
| DDD | Domain Driven Design |
| DL | Description Logic |
| DSR | Design Science Research |
| ESB | Enterprise Service Bus |
| FOAM | Framework for Ontology Alignment and Mapping |
| GATE | General Architecture for Text Engineering |
| GDPR | General Data Protection Regulation |
| HTML | HyperText Markup Language |
| HTTP | HyperText Transfer Protocol |
| HVM | Hardware-assisted Virtual Machine |
| IBM | International Business Machines |
| IA | Intelligent Agents |
| IAM | Identity and Access Management |
| IER | Information Extraction and Retrieval |
| IoT | Internet of Things |
| IRPS | Inter-Cloud Resource Provisioning System |
| IT | Information Technology |
| JAPE | Java Annotation Pattern Engine |
| JSON | JavaScript Object Notation |
| JSON-LD | JSON for Linked Data |
| KIM | Knowledge and Information Management |
| LDAP | Lightweight Directory Access Protocol |
| NCBO | National Center for Biomedical Ontology |
| NER | Named Entity Recognition |
| NLP | Natural Language Processing |
| OS | Operating System |
| OUL | Ontology Update Language |
| OWL | Web Ontology Language |
| PDO | PHP Data Objects |
| PHP | PreHypertext Processor |
| QOM | Quick Ontology Mapping |
| QoS | Quality of Service |
| RDF | Resource Description Framework |
| RDFa | Resource Description Framework in Attributes |
| RDFS | Resource Description Framework Schema |
| REST | Representational State Transfer |
| RSS | Really Simple Syndication |
| SLA | Service Level Agreement |
| SNOMED-CT | Systematised Nomenclature of Medicine – Clinical Terms |
| SOA | Service Oriented Architecture |
| SPARQL | Sparql Protocol and RDF Query Language |

| | |
|---|---|
| SSL | Secure Sockets Layer |
| SVM | Support Vector Machine |
| SQL | Structured Query Language |
| TDD | Test Driven Design |
| TPM | Trusted Platform Module |
| UI | User Interface |
| UML | Unified Modelling Language |
| URI | Uniform Resource Identifier |
| URL | Uniform Resource Locator |
| VMM | Virtual Machine Monitor |
| W3C | World Wide Web Consortium |
| WSDL | Web Services Description Language |
| XML | eXtensible Markup Language |
| YAML | Yet Another Markup Language |

# Chapter 1: Introduction

This chapter introduces the investigation into a cloud-driven solution for automated semantic annotation of documents on the Web and describes the research motivation. The aim and objectives of the research are explained as well as the research hypothesis. Furthermore, it presents the methodology adopted and research contributions to the body of knowledge. The chapter concludes with an outline for the structure of the thesis.

## 1.1 Research Motivation

The pre-web era of the Internet was mainly managed by service providers who facilitated data sharing across the Internet through computer internet interfaces for users. However, this approach gave rise to issues around content control, user behaviour and service interoperability thereby creating the need for adopting a more open platform. This facilitated the emergence of the World Wide Web in 1989 (Bratt, 2008); a facility which has come a very long way since then to become an integral part of the society today. The first version of the Web, commonly referred to as Web 1.0 started off as a "Static Web" with web documents being purely informational and "read-only" in nature. It was a document-centric web and focused on the creation of static websites via interlinked, hypertext documents containing information that could be accessed, searched and read by web users, over the Internet (Sheth & Thirunarayan, 2012). The static nature of the web then meant it offered little functionality in terms of interaction with, and contribution to content (Getting, 2007). Web 1.0 documents were also created using static HTML which often required frequent updating. Users had no influence or contribution in content creation, neither could they interact much with content. The major protocols or standards used for creating websites and content were HTTP, HTML and URL (Prasad et al., 2013).

Efforts in addressing these challenges facilitated the emergence of a Web 2.0, commonly referred to as the "Social Web" based on the dimension of interactivity and interconnectedness that it brought to the environment. With Web 2.0, the static nature of websites began to disappear as user-generated content became a prominent feature; facilitated by web technologies such as blogs, wikis, podcasts, RSS feeds, online web services and social networking (Prabhu., 2017). Websites became web applications; with the ability for end users to contribute and modify content. Interactions with other web users greatly enhanced the overall experience and fostered effective information collection and sharing (Kollmann et al., 2016). Some practical examples of features it enhanced include e-commerce; the ability to buy and sell online, e-learning; the ability to obtain an academic degree from a remote institution via online lessons, teaching and examination, e-tourism; the ability to book a flight and make hotel reservations online for a holiday trip and several others across health, education, finance, entertainment and all other industry sectors.

Furthermore, with advancements in other areas of Information and Communications Technology; such as Internet of Things, Cloud Computing and Mobile Computing, the amount of information on the web grew very drastically. Peer research statistics by Marr (2018) and sponsored by Forbes suggested that 90% of data on the Web was only added within the last 2 years. Likewise, research by Gunelius (2014) and published by ACI Group stated that every sixty seconds; over 72 hours of video content is uploaded to YouTube; over 4 million search queries are executed on Google and over 2.5 million content shares are made on Facebook. While these massive amounts of data make the web a more resourceful environment, it comes with an "Information Processing" challenge. With lots of data available to users to read, process, manage and utilise in diverse ways, it poses a challenge, as identifying and managing relevant data for an activity becomes cumbersome. Fensel (2011) classified these challenges as follows:

- Information Finding; which refers to issues with information searches that are predominantly keyword-based (Tate, 2018). Examples include issues with synonyms such as study/learn; issues with homonyms such as Jaguar (automobile) / Jaguar (animal); issues with spelling mistakes and variants such as Virtualisation / Virtualization.
- Information Extraction; which refers to issues with standardising methods for extracting information from web documents considering their usually unstructured nature as well as differences in formats and syntaxes (Kollmann et al., 2016).
- Information Representation; which refers to issues with being able to define context for data on the web in order to prevent misrepresentation (Sanchiz et al., 2017).
- Information Interpretation; which refers to the subjective interpretation of data based on user's understanding for various reasons (Sanchiz et al., 2017).
- Information Combination; which refers to issues with the disparate nature of web documents which prevents aggregating information from different sources together as a single piece (Zannettou et al., 2019).

The semantic web which is also referred to as Web 3.0 is designed to address the "Information Processing" challenge of humans with Web 2.0. This is because the semantic web defines one in which information processing is done by machines and not humans. So, the cumbersome task of processing massive amounts of data from various sources that humans experience with Web 2.0 is conducted by machines in Web 3.0. This is facilitated by means of adding machine-readable and hence, processable annotation data to documents on the web. The schema and format (such as RDF) for the annotation data is derived from ontologies; which are readable and understandable by machines; explaining why they can process them as well.

However, the evolvement of this "Semantic Web" has been faced with challenges as well which includes automated processes for annotating web documents with the required annotation data; a process known as "Semantic Annotation". Hence, the motivation for this research is to investigate semantic web challenges and more specifically, the automation challenge for semantic annotation in a bid to address the challenge and greatly foster the emergence of the "Semantic Web" – which has been elusive, or at best very slowly. This would involve exploring the use of software engineering concepts such as "Design Patterns" and "Microservices" as underpinning theories for developing a cloud computing solution to address the automation challenge of semantic annotation.

## 1.2 Research Aim and Objectives

Based on the motivation described in Section 1.1, the aim of this research is to design, develop and evaluate a holistic, cloud-driven and microservices-based architecture for automated semantic annotation of web documents. The objectives for achieving the aim are as follows:

1. To conduct a comprehensive literature review of both semantic web technologies and cloud computing paradigm, with a critical analysis of the state-of-the-art and beyond.
2. To investigate, identify and propose a set of requirements towards addressing automated semantic annotation challenges.
3. To conduct a feasibility study of leveraging cloud computing for an automated semantic annotation process.
4. To explore the use of software engineering concepts such as "Design Patterns" and "Microservices" as underpinning theories for developing a cloud computing solution to address automated semantic annotation challenges.
5. To propose and develop an architecture for automated semantic annotation based on the requirements identified earlier.

6. To validate the proposed architecture using a prototype implementation.
7. To evaluate the proposed architecture as well as the prototype implementation
8. To provide recommendations for further research in the area

## 1.3 Research Hypothesis

The hypothesis that forms the basis for this research is as follows: Cloud Computing can be fully leveraged as a paradigm to address the automation challenge of providing machine-understandable contextual data for semantically annotating documents on the web.

## 1.4 Research Methodology

Research can be referred to as a set of activities which contribute to a better understanding of an existing reality, comprising of a set of various behaviours of defined entities which researchers with common interests find engaging (Vaishnavi et al., 2017). Hanid (2014) defined research as the process of investigating which is systematic in nature by means of examining resources to extend the existing scope of knowledge within a domain. Furthermore, research can be characterised as a process that involves a systematic and controlled approach, distinguishable from gaining experience, which is usually an uncontrolled activity (Cohen & Manion, 1994). Research provides empirical results which requires validation by external entities (Vaishnavi et al., 2017). Research differs from reasoning in that the latter can operate in an abstract world that is highly separated from the reality. However, unlike experience and reasoning, research can be viewed as self-correcting; providing a rigorous testing with public access to the methods and results obtained for validation. Research techniques and methodologies are activities which researchers consider to be important and appropriate in understanding a research domain (Hevner & Chatterjee, 2010). Different means and methods are employed in information systems to conduct research and develop knowledge. Developing the understanding for an information systems domain being researched is positivist in nature as it allows the prediction of behaviour in relation to features or aspects of a phenomenon (Vaishnavi et al., 2017).

### 1.4.1 Design Science Research in Information Systems

Design Science Research (DSR) is a research methodology in Information Systems (IS) and describes techniques for generating novel constructs for solving problems or addressing challenges across various industry sectors and contributing to theories within the application domain (Hanid 2014). DSR can also be viewed as a methodology that aims to explore new solutions and alternatives to solving problems; explaining the exploratory process and improving the problem-solving processes for specific purposes (Thornhill et al., 2009). Over the years, the methodology has become well established in the field of IS (Drechsler & Hevner, 2016; Iivari, 2010). DSR focuses on developing valid knowledge to support organisational problem-solving within a given field. The support can be direct, indirect or instrumental. Aken (2004) noted that the mission of DSR is to develop scientific knowledge to support the design of novel and innovative artefacts by professionals and to emphasise its knowledge-orientation; in which focus is on developing new knowledge that will facilitate actions, such as design rather than on design itself.

Gregor & Hevner (2013) also noted that in IS context, DSR concerns the development of a wide range of socio-technical artefacts, such as decision support systems, modelling tools, methods for IS evaluation, IS modification interventions and governance strategies. Furthermore, they noted that DSR has evolved to become a crucial and reliable research model in Information Systems based on its unique features. These includes the fact that it is focused on proffering solutions and its nature for the outcome of a research problem. In line with that, Iivari (2010) proposed twelve theses and three levels of research which can be defined for Information Systems with the potential role of DSR for each of the levels. This is illustrated by Figure 1.1.

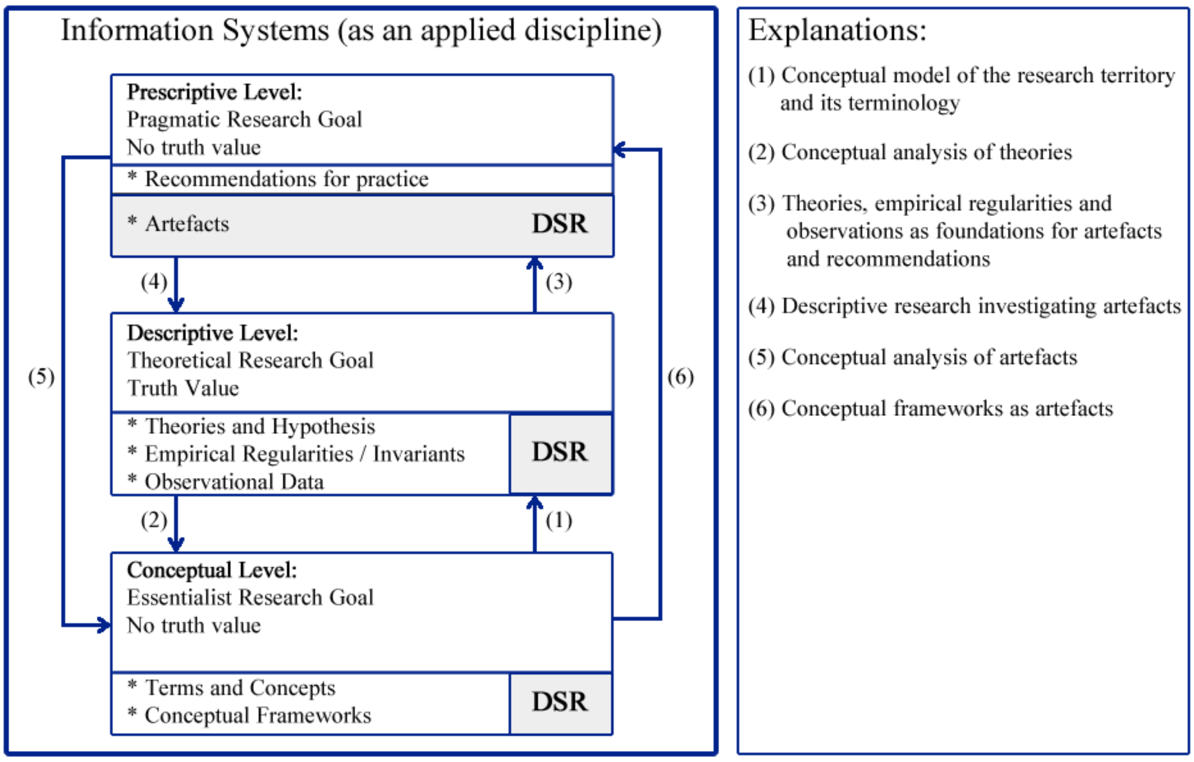

Information Systems (as an applied discipline)
Prescriptive Level:
Pragmatic Research Goal
No truth value
* Recommendations for practice
* Artefacts
DSR
(4)
(3)
Descriptive Level:
Theoretical Research Goal
Truth Value
* Theories and Hypothesis
* Empirical Regularities / Invariants
* Observational Data
DSR
(5)
(6)
(2)
(1)
Conceptual Level:
Essentialist Research Goal
No truth value
* Terms and Concepts
* Conceptual Frameworks
DSR
Explanations:
(1) Conceptual model of the research territory and its terminology
(2) Conceptual analysis of theories
(3) Theories, empirical regularities and observations as foundations for artefacts and recommendations
(4) Descriptive research investigating artefacts
(5) Conceptual analysis of artefacts
(6) Conceptual frameworks as artefacts

Figure 1. 1 - Levels of Research in Information Systems and DSR Role (Adapted From: Iivari, 2010)

According to Kuechler & Vaishnavi (2012), the common understanding of DSR in IS (DSRIS) is continually evolving as a process of acquiring knowledge through design principles and practices. Currently, the discussion is on what is to be built and the approach used in building it, that is, the DSRIS artefact and methodology. They further noted that the relationship of DSRIS to theory is forming a crucial part of the current discussion about DSR paradigm in the context of IS. Peffers et al. (2007) also emphasised the role DSR currently plays in IS towards developing and managing IS solutions and technologies as well as embodying goals and creativity which facilitates the creation of valid artefacts, hence, becoming pivotal to how IS researchers and practitioners apply, evaluate, create or improve various IS solutions and technological artefacts. The artefacts comprise of systems which are created with the aim of supporting management activities, decision-making processes, analysis and the operations of an organisation.

### 1.4.2 Philosophical Ground of DSR

Peffers et al. (2007) noted that information system researchers started developing interest in DSR in the early 1990s. This research methodology differed from other paradigms in terms of theory building, testing and interpretive research. To distinguish between research in design science, natural sciences and social sciences, Peffers et al. (2007) remarked that DSR attempts to create solutions, objects or models that serve human purposes. In that period, various researchers introduced DSR to the IS community. Some researchers advocated for integrating systems development with the research process through an approach that facilitates the development of theories and systems, supported with appropriate experiments (Nunamaker et al., 1990). It was further stated that DSR could effectively enhance the practicality of information systems research through the method of handling challenges experienced by IS practitioners.

Hanid (2014) noted that any process of knowledge creation starts with a substantive field of inquiry, commonly referred to as philosophy. In a philosophical inquiry, theories, facts and alternatives in the ideals are brought together and assessed in the creation and legitimisation of knowledge. In addition, there are a series of philosophical approaches to thinking, which are commonly divided into interpretivism, realism, positivism, critical theory, phenomenology, and hermeneutics (Scotland, 2012). The selection of a research strategy and methods for research activities depends on its philosophical stance. However, Hanid (2014) noted that such divisions did not distinguish another research paradigm centred toward practical problem solving, which is Design Science Research (DSR). Table 1.1 presents a comparison of DSR with some other well-known research perspectives.

Table 1. 1 - Design Science Research Perspective (Vaishnavi et al., 2017)

| **Basic Belief** | **Research Perspectives** | | |
|---|---|---|---|
| | **Positivist** | **Interpretive** | **Design** |
| Ontology | A single knowable and probabilistic reality | The construction of multiple realities in a social manner. | Multiple realities with contextually situated alternative world-states. Socio-technologically enabled. |
| Epistemology | Objective; dispassionate. Detached observer of truth | Subjective, i.e. values and knowledge emerge from the researcher-participant interaction. | Knowing through making: objectively constrained construction within a context. Iterative circumscription reveals meaning. |
| Methodology | Observational in nature with quantitative and statistical measures | Participation; qualitative. Hermeneutical, dialectical. | Developmental in nature with an impact measurement of artefact on compound systems |
| Axiology | Truth: universal and beautiful; prediction | Understanding: situated and description | Control; creation; progress (i.e. improvement); understanding |

From Table 1.1, it can be observed that DSR offers a multi-dimensional perspective from both the ontological and epistemological beliefs. The methodological belief is also developmental in nature which is well suited for Information Systems domain. Furthermore, from the Axiological belief, DSR provides control of the environment while offering improvements for the research process. These provide a basis for the adoption of DSR as a philosophical stance in this research as it constitutes an appropriate match considering the pragmatic nature of the research objectives.

### 1.4.3 DSR Three and Four-Cycle Views

Drechsler & Hevner (2016) evaluated a widely cited model visualisation of the DSR paradigm called the Hevner's three cycle view of DSR, which comprises of rigour, design and relevance cycles as comprehensively conceptualising the major perspectives of a DSR project. The cycles represent a series of activities which are closely related and can be described as follows;

- The relevance cycle provides a set of requirements based on the investigation domain for the research and integrates research outputs into environment field testing.
- The rigor cycle provides grounding theories and methods along with domain experience and expertise from the foundations of knowledge base into the research and adds new knowledge generated by the research to the increasing knowledge base.
- The central design cycle focuses on implementing the research methodology and accompanying activities to produce and evaluate a set of research outputs.

However, Drechsler & Hevner (2016) noted that the model lacks a key dynamic perspective on how DSR projects relate to the organisational context in which it is embedded. Due to the strong links to a real-world problem or situation, the researcher is not necessarily controlling the DSR project's speed, unlike in other research paradigms. For example, rapidly changing environmental conditions may require quick and short design cycles to maintain artefact utility. In effect, quick design cycles may leave only limited opportunities to draw on and grow extensive theoretical knowledge bases in the rigor cycle (Drechsler & Hevner 2016). Based on these, a four-cycle view was proposed, which adds "Change and Impact" as a fourth view to the cycle. This is presented in Figure 1.2.

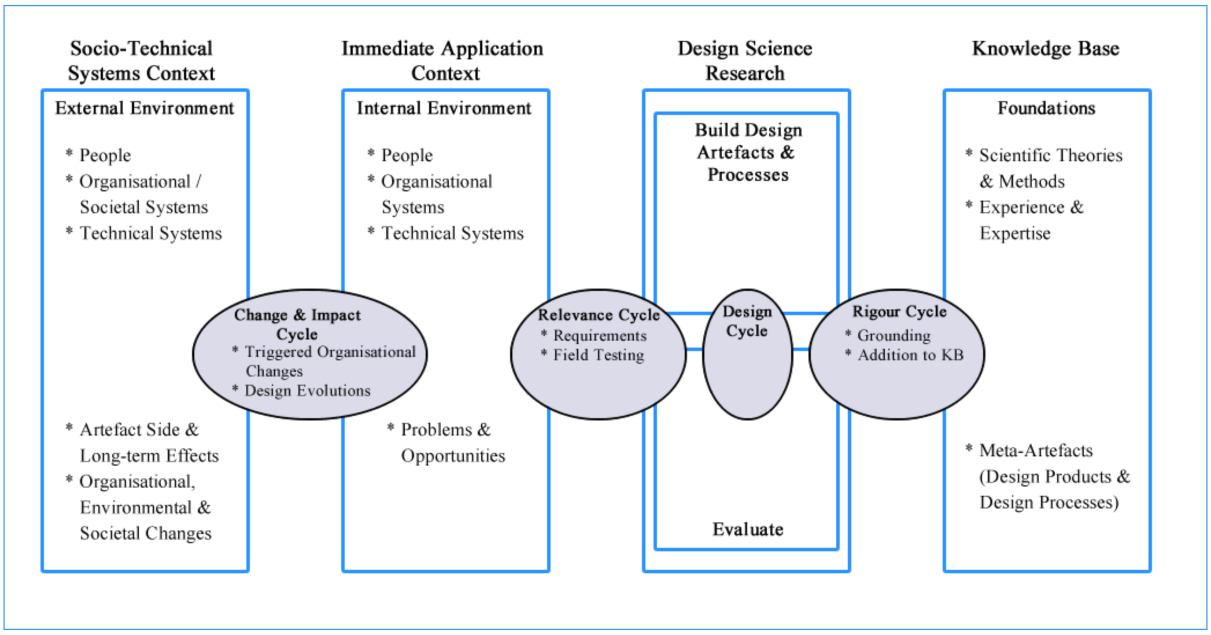

Socio-Technical Systems Context
Immediate Application Context
Design Science Research
Knowledge Base
External Environment
* People
* Organisational / Societal Systems
* Technical Systems
* Artefact Side & Long-term Effects
* Organisational, Environmental & Societal Changes
Internal Environment
* People
* Organisational Systems
* Technical Systems
* Problems & Opportunities
Build Design Artefacts & Processes
Evaluate
Foundations
* Scientific Theories & Methods
* Experience & Expertise
* Meta-Artefacts (Design Products & Design Processes)
Change & Impact Cycle
* Triggered Organisational Changes
* Design Evolutions
Relevance Cycle
* Requirements
* Field Testing
Design Cycle
Rigour Cycle
* Grounding
* Addition to KB

Figure 1. 2 – The Four-Cycle View of DSR (Adapted From: Drechsler & Hevner, 2016)

This research is in line with the four-cycle view proposed by Drechsler & Hevner (2016) and the context for each of the views in this research is as follows:

- Relevance Cycle: From the critical review of existing literature and identification of research gaps, this research proposes a set of requirements for the contextual domain; which is automated semantic annotation.
- Rigour Cycle: Based on the DSR methodology, this research investigates and employs software engineering concepts as underpinning theories, alongside the existing body of knowledge to develop new domain knowledge in the form of research artefacts and in accordance with research output types in DSR. These include Design Patterns and Microservices.
- Central Design Cycle: From the underpinning theories employed in the Rigour Cycle, this research implements DSR methodology to apply the theories within series of research activities towards producing research outputs.
- Change and Impact Cycle: This research considers the dynamic context that impacts research artefacts. Such contexts include societal and organisational systems from the external environment which influence research activities and outputs. Regarding this, outputs from the research are geared towards portability and interoperability to facilitate the needed flexibility that emanates from the external environment. Furthermore, the outputs foster standardisation within the domain which enhances a seamless integration and synchronisation with these external factors.

The identification of these cycles in a research study distinguishes DSR from other research paradigms (Hevner, 2007). Furthermore, advancements in the study and application of DSR topics introduces new dimensions for DSR processes to cope with dynamic and time-related aspects in research. The sources of such dynamics often lie in the wider environment outside the artefacts' immediate application context and therefore outside the three cycles of the original model. The new four-cycle view elevates the dynamic issues to the same level as refining the artefact in the design cycle or ensuring a research contribution in the rigor cycle.

### 1.4.4 DSR Process Model

The model of a DSR follows a multiplicity of described variants according to how they are practiced. The DSR process model utilised in this research was adapted by Vaishnavi et al. (2017) from a process model developed by Takeda et al. (1990). While the different phases present in both models appear to be similar, it is worth noting that activities carried out in each model are totally different from each other. Differences arise from the contribution of novel knowledge which is required to be the focal point in DSR. There are several key areas in a DSR model. Figure 1.3 presents the DSR process model adapted for this research and the application of each process step follows.

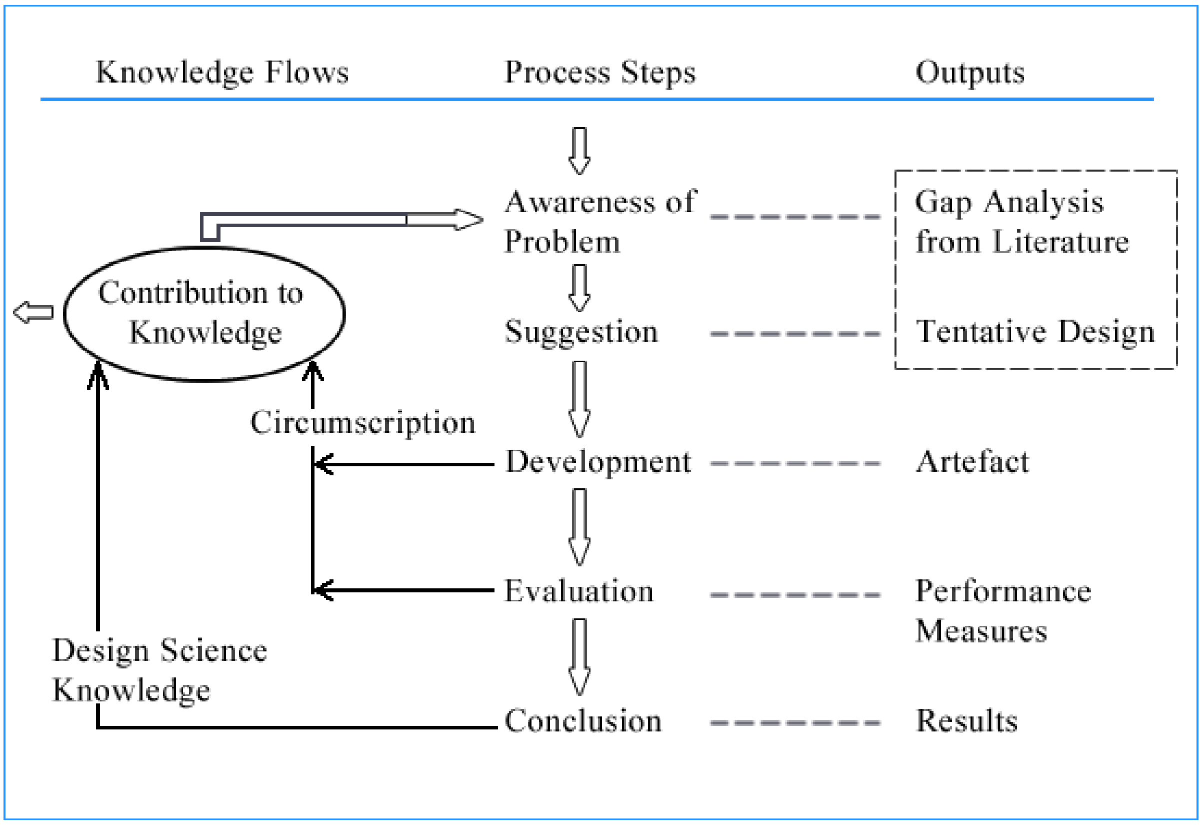


Figure 1. 3 – DSR Process Model (Adapted from Vaishnavi et al., 2017)

**Awareness of Problem**

This step requires an understanding of the domain and identification of a problem (or problems) to be solved. The understanding and identification process can be carried out in diverse ways. Multiple sources may result to an awareness of research problems that are of interest to a researcher. The sources may comprise of new and emerging developments in a specific industry or identifying the problems within a specified discipline. In this case, a secondary research through the critical review of existing literature was utilised. Upon completion of the critical review, an analysis of research findings based on identified gaps was conducted which provided a comprehensive awareness of the problem. This is detailed in Chapter 2.

**Suggestion**

Suggestion is also a crucial step within the DSR model. After a comprehension of the domain and review of the state-of-the-art, it becomes imperative to propose a solution or solutions. For this research, the step suggested addressing the challenge of automated semantic annotation holistically. The holistic perspective also necessitated the identification of requirements for its facilitation. While most of the requirements came from research findings, a few were novel; having been conceptualised from an analysis of research findings and the adoption of a multi-disciplinary approach to implementing the model. Furthermore, the suggestion step included the proposal of leveraging cloud computing paradigm for the holistic semantic annotation which was conducted through a feasibility study and resulted in a cloud computing capability model for holistic semantic annotation. This step is detailed in Chapter 3.

**Development**

This phase involves the actual development of an artefact or a set of artefacts for the identified problem(s). The artefact is developed to provide specified solutions and meet its objectives. DSR artefacts can belong to any of the output categories listed in Table 1.2.

Table 1. 2 - Types of Research Outputs for DSR. (Vaishnavi et al., 2017)

| | **Output** | **Description** |
|---|---|---|
| 1 | Constructs | The conceptual vocabulary of a domain |
| 2 | Models | Sets of propositions or statements expressing relationships between constructs |
| 3 | Frameworks | Real or conceptual guides to serve as support or guide |
| 4 | Architectures | High level structures of systems |
| 5 | Design Principles | Core principles and concepts to guide design |
| 6 | Methods | Procedures for implementing tasks |
| 7 | Instantiations | Situated implementations in certain environments that do or do not operationalise constructs, models, methods, and other abstract artefacts; in the latter case such knowledge remains tacit. |
| 8 | Design Theories | A prescriptive set of statements on how to do something to achieve a certain objective. A theory usually includes other abstract artefacts such as constructs, models, frameworks, architectures, design principles and methods. |

They can also include innovations for other fields, and they can be technical or informational in nature. The development of new explanatory theories, innovative design as well as development models and implementation processes or methods are all valid forms of artefact with DSR. An artefact's functionality and architecture for its creation are also valid outputs (March & Storey, 2008). This research has several outputs for the 'Development' step, and these are presented across Chapters 3 to 6.

## Evaluation

An evaluation phase is part of the DSR methodology activities. Since design artefacts are very fundamental to a DSR methodology, the evaluation phase needs to be both rigorous and appropriately implemented, providing a basis for demonstrating that an artefact meets its aim and objectives. This involves observing its capability to solve problems based on a set of requirements, by drawing comparisons between the requirements specification and results obtained. Within DSR, a design artefact is required to contain knowledge relating to the domain where the artefact is expected to function (Gregor & Hevner, 2013). The available evaluation methods in DSR are presented in Table 1.3. This research utilises all the described methods over the course of the evaluation processes. These can be found in Chapter 7.

Table 1. 3 – Design Science Research Evaluation Methods

| Evaluation Method | Description |
|---|---|
| Observational | The monitoring of the artefact within native environments to observe its outputs. |
| Analytical | The demonstration of the qualities of the artefact and how it is fit for use, as intended. |
| Experimental | The execution of an experiment on the artefact within a controlled environment, and possibly with some dummy data. |
| Testing | The execution of the different interfaces of artefacts to detect errors and functionality issues. |
| Descriptive | Construction of an informed argument for the artefact's usability based on information from knowledge base. Development of scenarios too for artefact's usability. |

## Conclusion

The last step of the model is the 'Conclusion' which defines the overall position of the research and how it has met the stated aim and objectives, based on its outputs, with adequate information and knowledge on disciplinary culture. It communicates the identified problem, its significance, utility, design rigour, relevant audience, artefact and novelty (Offermann et al., 2009). This phase should also provide recommendations for further research in the area. This is in line with the Vaishnavi et al. (2017) DSR methodology.

However, the Peffers et al. (2007) DSR methodology concludes with a 'Communication' phase which is concerned with publishing research findings. For this research, the "Conclusion" and "Communication" are integrated into the final phase of the research process. While the 'Conclusion' is detailed in Chapter 8, 'Communication' cuts across Chapters 2 to 8. Table 1.4

presents a summary of the DSR model adoption in this research in terms of the process steps and corresponding chapters.

Table 1. 4 – DSR Process Steps and Research Corresponding Chapters

| **DSR Process Model Steps** | **Corresponding Chapter(s)** | **Outputs** |
|---|---|---|
| Awareness of Problem | Chapter 2 | Gaps Analysis from the investigation and review of problem area domains |
| Suggestion | Chapter 3 | Proposal of a holistic perspective and requirements specification for it, including novel requirements |
| | | Cloud Computing Capability Model for Holistic Semantic Annotation |
| | Chapter 4 | Cloud Computing Maturity Model for Holistic Semantic Annotation |
| | Chapter 5 | Design Patterns and Pattern Language for Holistic Semantic Annotation |
| | | Holistic, Cloud-driven and Microservices-based Architecture for Automated Semantic Annotation of Web Documents |
| Development | Chapter 6 | Description of implementation approach, techniques and supporting technologies. |
| Evaluation | Chapter 7 | Functional Evaluation of prototype implementation |
| | | Comparative Evaluation for any developed artefacts such as models and architectures |

| | | |
|---|---|---|
| | | Experimental Evaluation for any implementations for performance measurement |
| Conclusion | Chapter 8 | Research Summaries |
| | | Research Contributions |
| | | Recommendations for future work in the domain |
| Communication | Chapters 2 to 8 | Research Thesis |
| | | Publications |

### 1.4.5 DSR Cognitive Stages

According to Vaishnavi et al. (2017), the DSR process model comprises of three cognitive stages; Abduction, Deduction and Reflection. The "Suggestion" phase of the DSR process corresponds to the Abduction cognitive stage whereby researchers utilise a research-driven approach to proffer technical solutions based on new or existing knowledge from a problem area. The Development and Evaluation phases of DSR correspond to the Deduction cognitive stage, in which an implementation; either partial or full is evaluated based on a set of pre-defined requirements. The Evaluation results to making deductions which are either in line with the initial hypothesis or not. If they are, these are further utilised in the next stage of the cognitive process; Reflection. On the other hand, if they are not, the deductions still constitute knowledge as "Circumscriptions". The Conclusion phase corresponds to the Reflection or Abstraction cognitive stage which indicates the end of a research cycle with adequate reflection and communication of new knowledge; referred to as Design Science Knowledge in this context. This stage is very crucial as it forms the basis upon which one can understand the entire DSR process. The means of communicating the knowledge can be through a research thesis, publications, conferences, workshops, seminars and lots more. The DSR process model by Peffers et al. (2007) implicitly identifies these cognitive stages as well in its description of the DSR methodology phases. Figure 1.4 illustrates the cognitive stages and their corresponding mappings within the DSR phases.

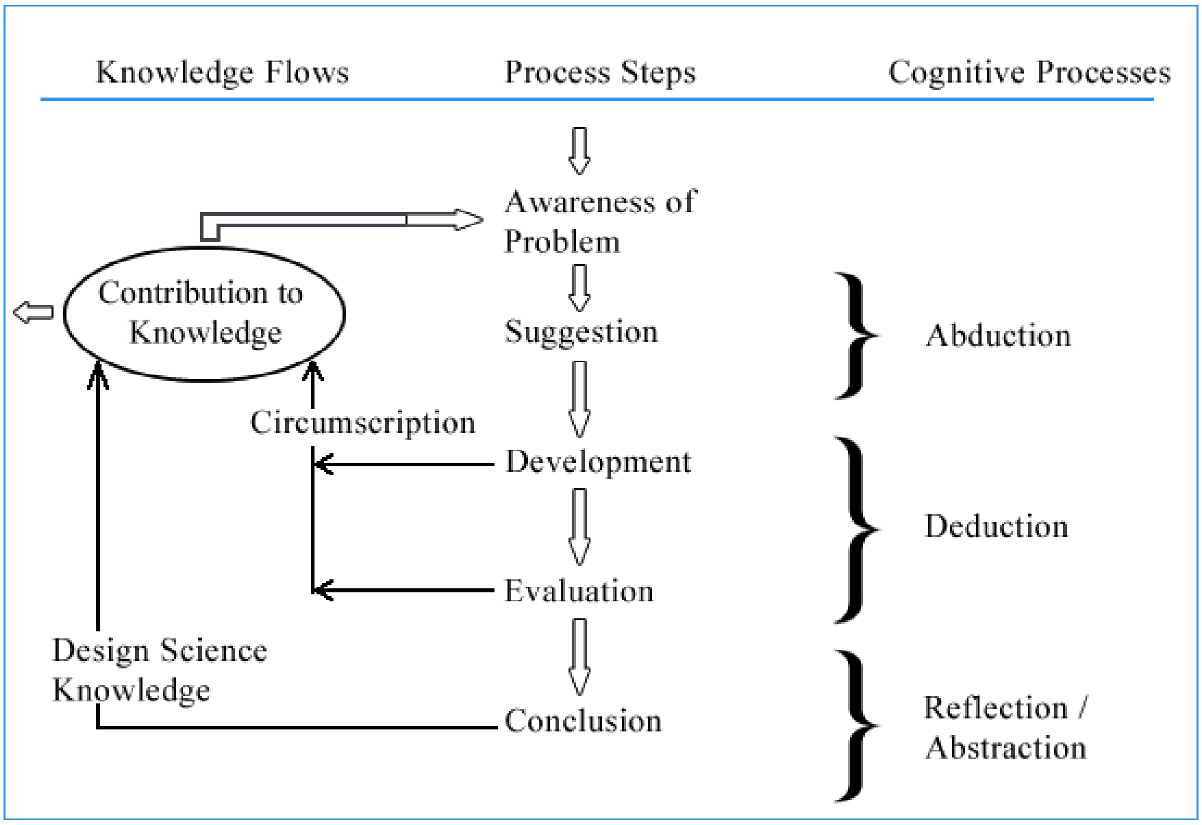


Figure 1. 4 – DSR Cognitive Stages. (Adapted from: Vaishnavi et al., 2017)

### 1.4.6 DSR Knowledge Contribution Framework

DSR requires its outputs to be contributions to the body of knowledge within a domain. Gregor & Hevner (2013) identified four major types of knowledge contribution in DSR. These are Invention, Improvement, Adaptation and Routine Design. Invention refers to knowledge contribution based on inventing new knowledge or providing solutions to a new problem area within a domain. Improvement refers to the development of new knowledge or solutions to a well-known problem area. The Adaptation type of knowledge refers to an innovative utilisation of known knowledge or solutions to address a new problem area. This might involve adapting concepts from other disciplines to provide a multi-disciplinary solution in a domain. Lastly, the Routine Design is based on utilising existing knowledge or solutions for well-known problem areas. Vaishnavi et al. (2017) noted that a single DSR has the capability of utilising more than one of the types of knowledge. Figure 1.5 presents the knowledge contribution framework for DSR adapted from Gregor & Hevner (2013).

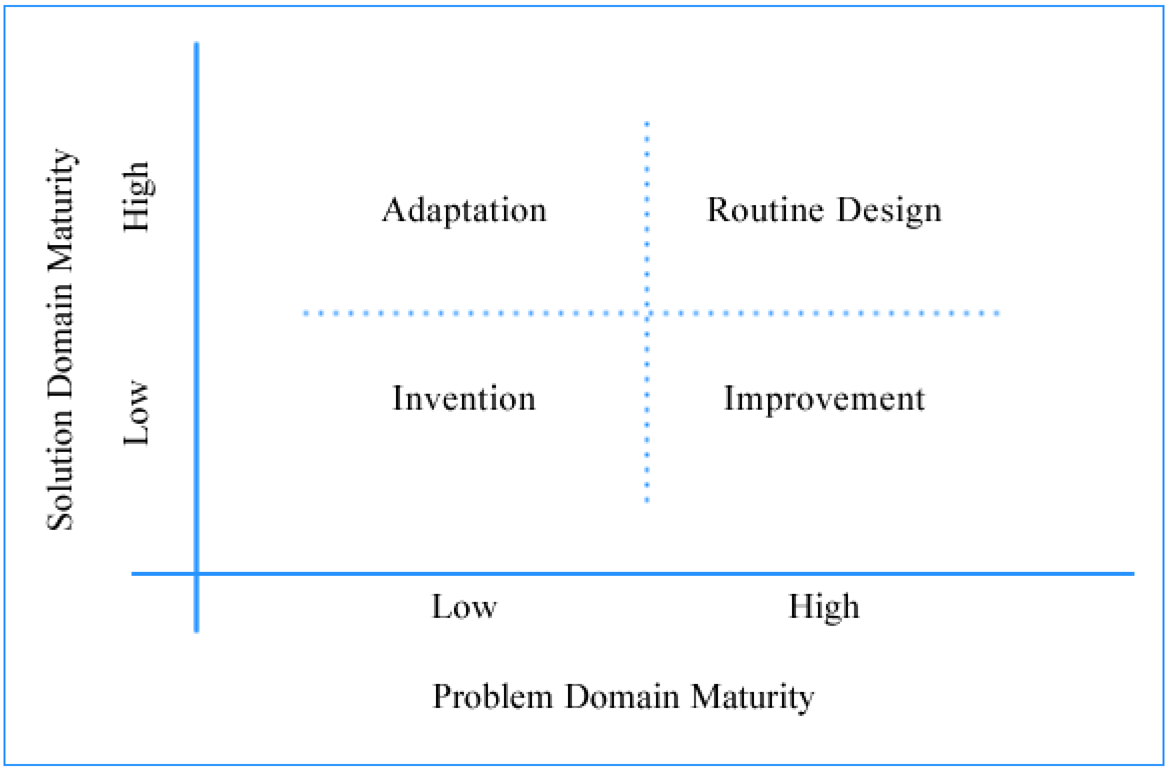


Figure 1. 5 - DSR Knowledge Contribution Framework. (Adapted from: Gregor & Hevner, 2013)

Based on the DSR Knowledge Contribution Framework in Figure 1.5, the contributions to knowledge from this research fall under the "Improvement" category by means of developing significant new knowledge and solutions for known challenges with automated semantic annotation and the overall evolvement of a semantic web.

## 1.5 Research Contributions to Knowledge

The contributions to knowledge from this research are as follows:

1. A comprehensive literature review on semantic web technologies and cloud computing paradigm, including the interaction types between both.
2. The proposal of a holistic approach to addressing automated semantic annotation challenges with an identification of a set of requirements and additional novel requirements to facilitate a holistic semantic annotation process.
3. The development of a Cloud Computing Capability Model for Holistic Semantic Annotation which assesses and defines cloud computing mechanisms for facilitating a holistic semantic annotation process.

4. The development of a Cloud Computing Maturity Model for Holistic Semantic Annotation which provides maturity levels for holistic semantic annotation solution deployment in the cloud based on a set of well evaluated metrics.
5. Design Patterns and Pattern Language for Cloud-Driven, Holistic Semantic Annotation which details technical solutions towards meeting the holistic semantic annotation requirements.
6. The development of **CloudSea**: A Microservices-Based Architecture for Cloud-Driven, Holistic Semantic Annotation which provides a full-fledged software architecture design for holistic semantic annotation.
7. The development of a prototype for CloudSea as proof-of-concept.
8. Functional and Comparative Evaluation of CloudSea.
9. The development of multiple CloudSea prototypes based on different software architectural patterns as well as different application deployment patterns in the cloud.
10. Experimental Evaluation of the Cloud Computing Maturity Model for Holistic Semantic Annotation based on the multiple CloudSea prototypes.
11. The provision of detailed recommendations towards further research in the area.

## 1.6 Thesis Structure

The thesis is organised into eight chapters and a summary for each one is presented as follows:

**Chapter 1: Introduction**

This chapter introduces the research, which includes a background and motivation, research aim and objectives, research questions and contributions to knowledge. It also includes the methodology adopted for the research which is based on the Design Science Research (DSR) paradigm. The chapter concludes with a breakdown of the thesis structure and a summary for each chapter.

**Chapter 2: Literature Review**

This chapter investigates the domains of semantic web technologies and cloud computing paradigm, with a comprehensive review of their state-of-the-art as well as how they integrate to proffer IT solutions. Furthermore, the chapter analyses findings from research to identify challenges with the evolution of a semantic web.

**Chapter 3: Requirements for a Holistic Semantic Annotation Process**

This chapter proposes a holistic perspective to automated semantic annotation based on findings from Chapter 2. Furthermore, requirements for the holistic perspective are identified from literature and additional novel requirements were proposed. The chapter goes on to evaluate the feasibility of leveraging cloud computing for holistic semantic annotation and proposes a cloud computing capability model for holistic semantic annotation based on the evaluation.

**Chapter 4: Cloud Computing Maturity Model for Holistic Semantic Annotation**

This chapter investigates different application deployment patterns in the cloud with respect to holistic semantic annotation. The defining metrics for different patterns of cloud application deployments were identified and critically evaluated. Based on the metrics, different deployment patterns for holistic semantic annotation in the cloud were proposed as maturity levels. The different patterns constitute a cloud computing maturity model for holistic semantic annotation; as a guide towards maximising cloud computing benefits for holistic semantic annotation. Based on the highest maturity level within the model, 'CloudSea: Cloud-Driven Semantic Annotation' was proffered.

**Chapter 5: Design Patterns for Holistic, Microservices-Based CloudSea**

This chapter developed twelve design patterns and a pattern language for holistic semantic annotation based on the set of requirements identified in Chapter 2. These patterns are facilitated by microservices software architectural pattern and constitute technical solutions for the holistic semantic annotation requirements. Furthermore, they constitute re-usable knowledge for the solutions they proffer in line with Design Patterns paradigm. The chapter goes on to propose CloudSea; A Holistic, Cloud-Driven and Microservices-Based Architecture for Automated Semantic Annotation of Web Documents as a full-fledged software architecture implementable and deployable in the cloud for automated, web-scale semantic annotation.

**Chapter 6: CloudSea Prototype Implementation**

This chapter provides implementation steps and techniques for the developed CloudSea prototype, with details regarding supporting technologies for its development and deployment in the cloud.

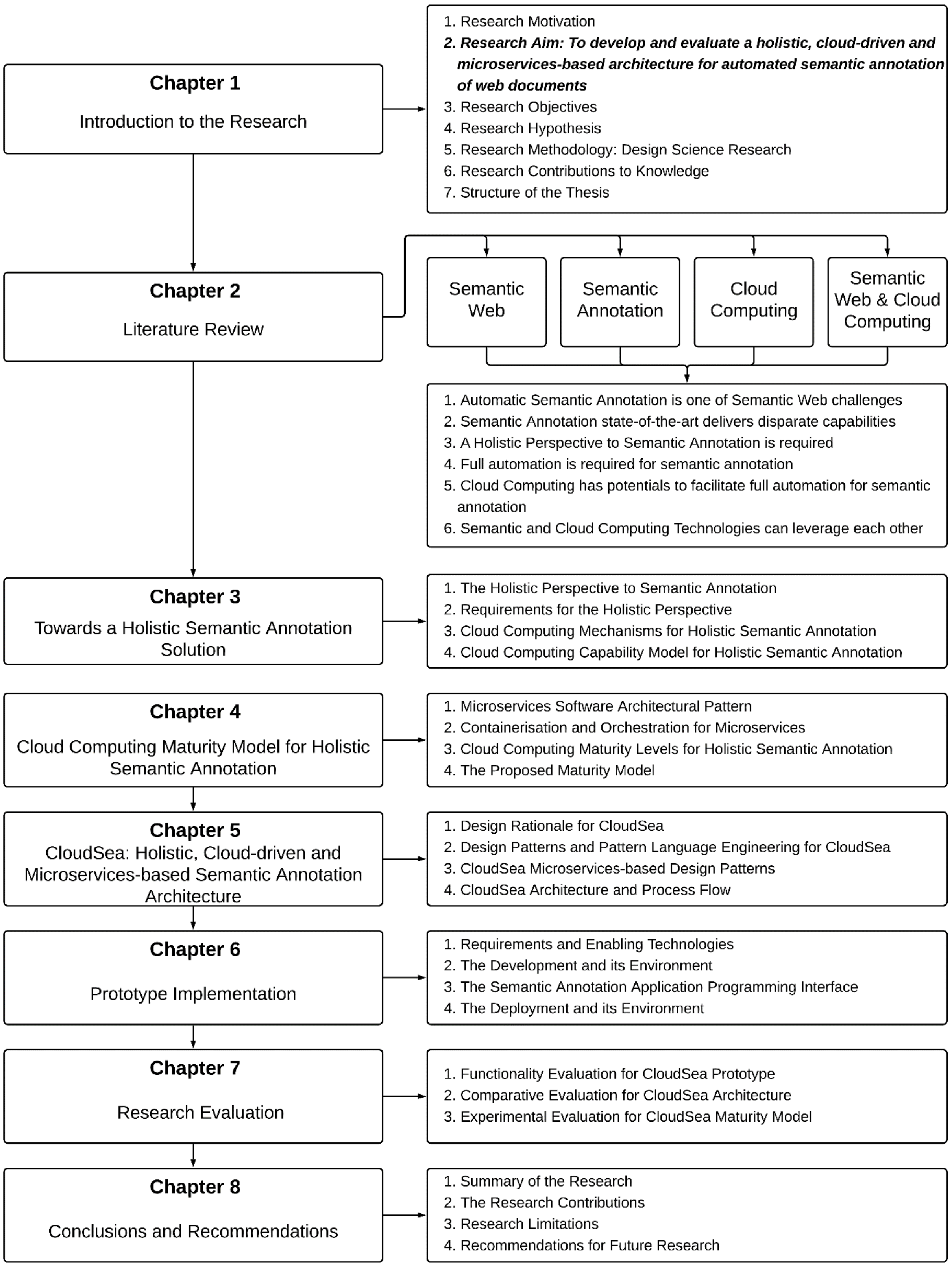
**Chapter 7: Research Evaluation**

This chapter is in three phases, as follows:

- It provides a functionality evaluation for CloudSea prototype described in Chapter 6; describing its features and how it delivers automated semantic annotation with capabilities for large-scale utilisation.
- It provides a comparative evaluation for CloudSea Architecture proposed in Chapter 5; with a detailed comparison and qualitative evaluation against existing semantic annotation solutions presented in the literature review chapter.
- It provides experimental evaluation for the multiple CloudSea prototypes implemented based on the differences in their software architectural pattern and their deployment patterns. This evaluated the Cloud Computing Maturity Model for holistic semantic annotation proposed in Chapter 4.

**Chapter 8: Conclusion and Recommendations**

This chapter summarises the research outcomes and provides recommendations for future work in the area. Figure 1.6 further illustrates the structure of the thesis.



Chapter 1
Introduction to the Research
1. Research Motivation
2. Research Aim: To develop and evaluate a holistic, cloud-driven and microservices-based architecture for automated semantic annotation of web documents
3. Research Objectives
4. Research Hypothesis
5. Research Methodology: Design Science Research
6. Research Contributions to Knowledge
7. Structure of the Thesis
Chapter 2
Literature Review
Semantic Web
Semantic Annotation
Cloud Computing
Semantic Web & Cloud Computing
1. Automatic Semantic Annotation is one of Semantic Web challenges
2. Semantic Annotation state-of-the-art delivers disparate capabilities
3. A Holistic Perspective to Semantic Annotation is required
4. Full automation is required for semantic annotation
5. Cloud Computing has potentials to facilitate full automation for semantic annotation
6. Semantic and Cloud Computing Technologies can leverage each other
Chapter 3
Towards a Holistic Semantic Annotation Solution
1. The Holistic Perspective to Semantic Annotation
2. Requirements for the Holistic Perspective
3. Cloud Computing Mechanisms for Holistic Semantic Annotation
4. Cloud Computing Capability Model for Holistic Semantic Annotation
Chapter 4
Cloud Computing Maturity Model for Holistic Semantic Annotation
1. Microservices Software Architectural Pattern
2. Containerisation and Orchestration for Microservices
3. Cloud Computing Maturity Levels for Holistic Semantic Annotation
4. The Proposed Maturity Model
Chapter 5
CloudSea: Holistic, Cloud-driven and Microservices-based Semantic Annotation Architecture
1. Design Rationale for CloudSea
2. Design Patterns and Pattern Language Engineering for CloudSea
3. CloudSea Microservices-based Design Patterns
4. CloudSea Architecture and Process Flow
Chapter 6
Prototype Implementation
1. Requirements and Enabling Technologies
2. The Development and its Environment
3. The Semantic Annotation Application Programming Interface
4. The Deployment and its Environment
Chapter 7
Research Evaluation
1. Functionality Evaluation for CloudSea Prototype
2. Comparative Evaluation for CloudSea Architecture
3. Experimental Evaluation for CloudSea Maturity Model
Chapter 8
Conclusions and Recommendations
1. Summary of the Research
2. The Research Contributions
3. Research Limitations
4. Recommendations for Future Research

Figure 1. 6 – Thesis Structure

# Chapter 2: Literature Review

This chapter aims to review and examine existing literature in the domains of semantic web technologies and cloud computing; with a view to critically analyse the state-of-the-art for both domains as well as their integrations in various dimensions. Findings from the review and subsequent analysis would be described as well, in line with how they provide some insights and direction for the subsequent chapters.

## 2.1 The Semantic Web

The Semantic Web also commonly referred to as Web 3.0 ushered in an extension of web via expression of content not only using natural language, but by other means that provide comprehension, interpretation and usability abilities to software agents (Ye et al., 2015). This makes finding, sharing and aggregating information from multiple sources easier, laying groundwork for the evolution of what is called "The Data Web" - the publishing of structured data records to the web to enable remote reusability and querying (Khalili et al., 2016). With Web 3.0, data integration and interoperability of applications achieved a new level, providing interlinked open access to data as web pages and creating the path towards a full semantic web (Verspoor et al., 2015). As a result, web documents can become context-aware using annotation data, for various processing and management capabilities, facilitated by intelligent agents (Rudman & Bruwer, 2016). These intelligent agents are software programs designed to enable collection of information according to user interactions on the web, in order to perform automated tasks for users. This is facilitated via languages that offer information description that machines, and intelligent agents can understand (Ye et al., 2015).

The semantic web concept can also be perceived as involving the provision of a general framework that adds a semantic layer to the web for facilitating and allowing machines to read, understand and interpret web content (Bourgonje et al., 2016; Gutierrez et al., 2019). The aim of this is to enable data sharing and reuse across diverse applications and systems. The thought behind its emergence is the conversion of structured and semi-structured web documents into a 'web of data' that allows expression of basic semantics in a way machines can process and understand (Ye et al., 2015). The machine-readable data can be produced through the creation of schema comprising of marked and interlinked characteristics such as defined terms, properties and formal relationships of web documents by developers (Gutierrez et al., 2019). Furthermore, the creation of such a schema creates the need for a semantic structure determining the attachment of these characteristics to certain instances, and the representation of statements collected in a formal set of relationships known as ontology (Akgun & Ayvaz, 2018). An ontology provides the definition for the rules of representation and the establishment of relationship hierarchies (Narula et al., 2018). This allows for the contextualisation of data points by linked data through the supply of additional information on data, thereby making provision for easy comprehension of information by machines (Giri, 2011; Halford et al., 2013). Several technologies play a role in facilitating the semantic web in diverse ways. Semantic technologies refer to a set of programming languages and standards with common exchange protocols and data formats to support a web of data across several domains, employing formal semantics to provide context for digital documents (Coronado et al., 2015). While some are generic to information systems, others are quite specific for the semantic web. Figure 2.1 presents the semantic web technology stack.

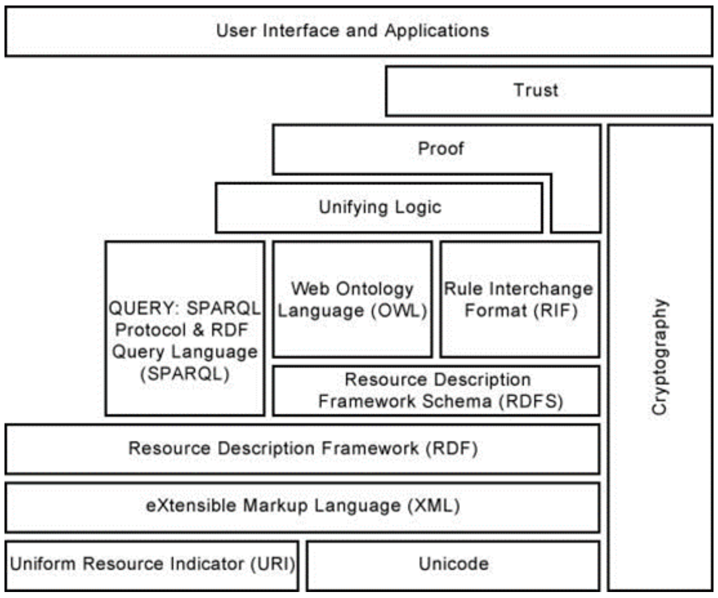

User Interface and Applications
Trust
Proof
Unifying Logic
QUERY: SPARQL Protocol & RDF Query Language (SPARQL)
Web Ontology Language (OWL)
Rule Interchange Format (RIF)
Resource Description Framework Schema (RDFS)
Resource Description Framework (RDF)
eXtensible Markup Language (XML)
Uniform Resource Indicator (URI)
Unicode
Cryptography

Figure 2. 1 – The Semantic Web Technology Stack (Adapted from: Gezer & Bergweiler, 2016)

## 2.1.1 Supporting Technologies

A wide range of technologies and standards constitute the semantic web technology stack as represented in Figure 2.1. These are utilised for varying functionalities across the implementation of a semantic web and can be described as follows:

- The Unicode and URI Layers define standards for recognising semantic web objects and confirming the use of international character sets for data representation (Alam et al., 2015).

- The XML (eXtensible Markup Language) Layer which also comprises of namespaces and XML Schema helps in integrating semantic web and XML-based standards. XML is responsible for the provision of surface syntax applicable to structured documents without imposing semantic constraints on what the documents stand for (Ye et al., 2015). XML Schema focuses on the schema for XML documents; defining a strict structure for elements contained within them.
- RDF is a simple data model for referring to objects and their relationships. It facilitates the portability of annotation data across multiple platforms. XML and RDF technologies complement each other in building an intelligent web (Gutierrez et al., 2019). The RDF (Resource Description Framework) layer alongside RDFS (Resource Description Framework Schema) and RDFa (Resource Description Framework with attributes) facilitates the schematic and syntactic definition of vocabularies to be referenced by Uniform Resource Identifiers (Ye et al., 2015). The resources, semantic relations, links and services are also defined in this layer. It makes provision for a directed graph formalisation, with nodes representing resources and arcs representing properties.
- RDFS represents a vocabulary to describe properties and classes of RDF resources, including semantics for generalisation-hierarchies of such properties and classes at various abstraction levels (Ye et al., 2015).
- The Ontology Layer is based on the description of concepts, properties and relations within ontologies. It also outlines the traits between various concepts which helps in vocabulary evolution (Basu, 2019, Wang et al., 2015). OWL (Web Ontology Language) is a prominent standard on this layer and it offers a description of roles for ontological components and how they relate with one another.
- RIF (Rule Interchange Format) is for rule exchanges within the web while SPARQL is a recursive acronym for Sparkle Protocol and RDF Query Language and is used to query semantic graph databases for data in formats such as RDF or JSON (Ye et al., 2015).
- The Cryptography Layer ensures data security by means of encryption across the different standards available within the semantic web stack (Alam et al., 2015).
- The Unifying Logic Layer authors rules for the semantic web while the Proof Layer implements them while the Trust Layer collaborates with the Proof Layer to evaluate

application mechanism and validate the implementation of rules by the Proof Layer (Alam et al., 2015).

### 2.1.2 Challenges of the Semantic Web

Despite promises offered by the semantic web, it has been faced with several challenges. According to Buscaldi et al. (2018), major challenges for the semantic web include content availability; the evolution, availability and development of ontology; scalability; multilingualism, and standardisation for semantic web technologies, suggesting that only little content is available on the semantic web with the need to facilitate context-awareness for web documents at large through semantic annotations towards a fully semantic web. The need to create general ontologies for the semantic web as well as the means to develop and manage changes that come with their evolution and the annotations that refer to them was also a vital challenge put forward.

The research opined that the organisation of the semantic web content including the storage and the appropriate mechanisms to search for them are required with the need to exhibit scalability in preparation for a massive semantic web growth. There is also the challenge of accessing information in various languages such that there is no access to the content by providers and users in their native language. Based on the continued increase in the quantity of information on the semantic web, there is the issue of easy recognition of relevant content by those who access information. The usual hypertext structure visualisation of the current web needs to be improved upon for better visualisation. More technologies need to be provided to make the semantic web languages stable. However, several other dimensions of identifying semantic web challenges can be noticed from literature that encompass the classification by Buscaldi et al. (2018) and goes beyond those to identify other challenges. These have been classified into Execution-Related, Implementation-Related and General Challenges, with an analysis of each as follows.

#### 2.1.2.1 Execution-Related Challenges

Semantic Web challenges relating to its execution are many and diversified. There are issues with methods of Information Extraction and Retrieval (IER) from web documents (Niklaus et al., 2018). While some semantic annotation tools utilise traditional IER methods such as N-Gram Analysis or Hidden Markov's Model, another school of thought subscribes to the development and use of Web IER methods (Yates, 2007). These create a wide range of processes for semantic annotation based on the IER method deployed. Issues relating to services and trust also exist, in which there are no W3C-compliant standards usable and deployable by the public for web documents (Tjoa et al., 2005). Furthermore, there are issues with scalability, as most existing semantic annotation tools have been developed to utilise specific ontologies or run on a specific platform, thereby limiting its scope of adoption for wide-scale semantic annotation (Kulesza et al., 2018). For the web to become truly semantic in nature, a semantic annotation tool that is portable in nature, easily accessible and robust enough to accept different ontologies for semantic annotation is required.

#### 2.1.2.2 Implementation-Related Challenges

The major challenges in this area are focused on content for the semantic web and means of managing their life cycle. The additional content that semantic web technologies add to the web are ontology-generated metadata which are used to annotate web data for the provision of an underlying meaning and context. Based on available tools and methods, the generation and availability of these metadata has been a daunting task due to challenges relating to the development and engineering of ontologies and metadata (Narula et al., 2018). Integrating ontologies is vital for the aggregation of resources both within same domain and across multiple domains and is a challenge within Ontology Engineering (Da Silva & Cavalcanti, 2014). Ontology Integration can be implemented in several ways such as ontology mapping, merging, alignment, elucidation, optimisation and self-learning. The other major area is the means of adding these additional content (i.e. metadata) when available. There are manual, semi-automatic and automatic semantic annotation methods and they all present strengths and weaknesses which are analysed later in section 2.2.

#### 2.1.2.3 General Challenges

Some of the general semantic web challenges identified in research include multilingualism (Piao et al., 2015; Gracia et al., 2012) and social issues (Kirrane et al., 2018). Multilingualism refers to issues relating to the translation of ontologies and ontology-generated annotations from one language to another. The task of developing ontologies for different languages is time-consuming and cumbersome, considering the need for regular updates as well. While translating ontologies from one language to another is favoured over developing new ones for each language, existing translation standards are not matured enough currently and results into inaccurate translations (Arcan et al., 2016). Social challenges also exist such as consensus on taxonomies or data dictionaries for specific knowledge domains between experts in the field.

These taxonomies or data dictionaries are required for ontology development. Figure 2.2 presents a summary of semantic web challenges based on the classification described.

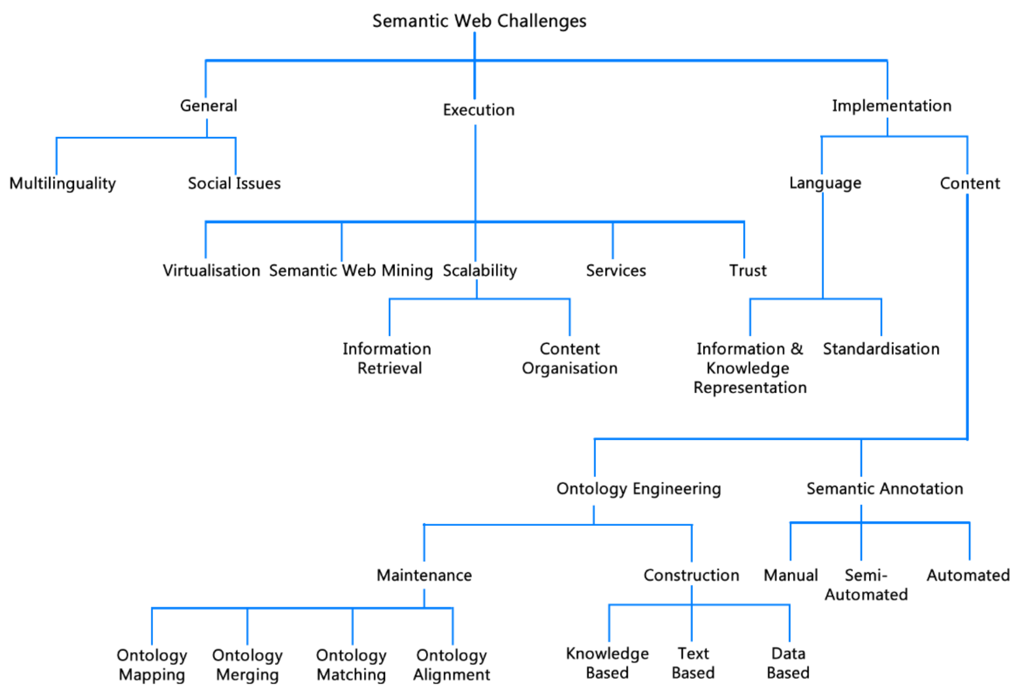


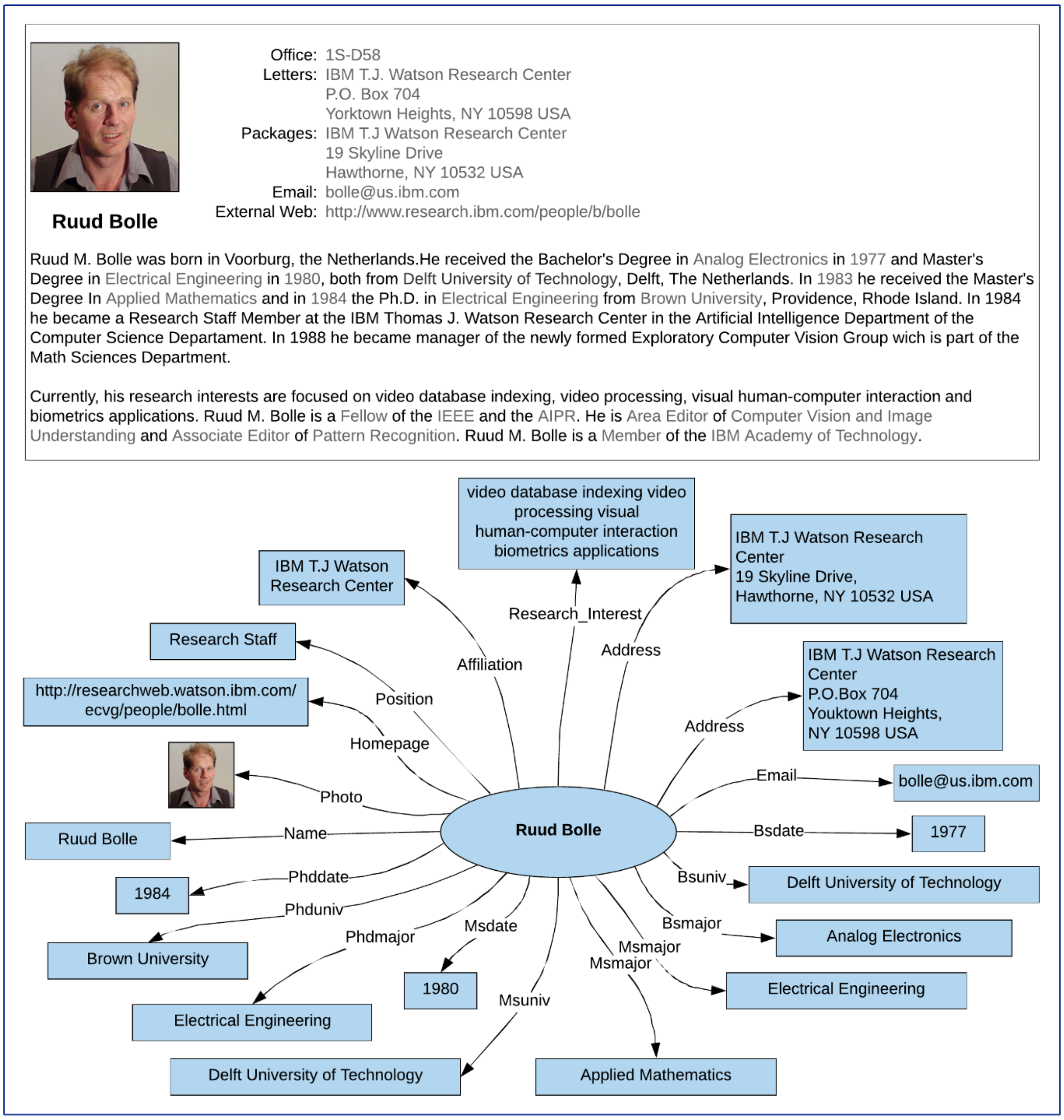
Figure 2. 2 – A Summary of Semantic Web Challenges

## 2.2 Semantic Annotation

As described in Section 2.1, the concept of a semantic web is based on the ability of computers to understand and process documents on the web by means of the application of context to these documents through an annotation process. Annotation is defined in the Merriam-Webster online dictionary as “a note added by way of comment or explanation”. Annotation is regarded as both an object which is added to a document, and the activity that produces this object (Yordanova & Kruger, 2018). Semantic Annotation defines the process of applying contextual information to web documents or more specifically, to contents of web documents. Ontology-based annotation data in formats such as RDF and JSON-LD are utilised to provide the much-needed context for web document content. Generally, the success of the semantic web depends to a great deal on the spread of semantically annotated web documents. Annotating web documents helps to improve search efficiency by introducing well-defined concepts described by the search domain’s ontology (Slimani, 2013). Semantic Annotation also entails merging semantic concepts with natural language through the introduction of annotation data (concepts of an ontology, such as classes, instances, properties and relations) in web documents, in order to define context (Oliveira & Rocha, 2013).

In addition, semantic annotation is a necessity for enabling widespread intelligence in new and existing web content and making the semantic web vision a reality (Tang et al., 2012), with application in a wide range of content-oriented areas (Tulasi et al., 2017). It is also used in the support of information visualisation, reasoning about web resources and advanced information search (Sajja & Akerkar, 2016). Semantic annotation is applicable to any sort of text, web pages, regular (non-web) documents, text fields in databases; providing annotations for mapping instances of ontology classes to the actual ontology classes (Brank et al., 2018).

Furthermore, Semantic Annotation fosters a great level of automation on the web. This includes automated data linkage based on semantic relationships between annotated web documents. Likewise, the integration or aggregation of data from multiples sources, presented to a user as a single piece of data is an automation feature which is both beneficial for the user; in terms of convenience and timesaving, as well as computing resources; in terms of resources optimisation. In addition, an automated lifecycle for data on the web can be fostered with semantic annotation, providing a management mechanism for web data from creation until the data becomes obsolete and is archived. Figure 2.3 presents an example of a web document being semantically annotated using data from a knowledge graph.



Office: 1S-D58
Letters: IBM T.J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598 USA
Packages: IBM T.J Watson Research Center
19 Skyline Drive
Hawthorne, NY 10532 USA
Email: bolle@us.ibm.com
External Web: http://www.research.ibm.com/people/b/bolle

**Ruud Bolle**

Ruud M. Bolle was born in Voorburg, the Netherlands.He received the Bachelor's Degree in Analog Electronics in 1977 and Master's Degree in Electrical Engineering in 1980, both from Delft University of Technology, Delft, The Netherlands. In 1983 he received the Master's Degree In Applied Mathematics and in 1984 the Ph.D. in Electrical Engineering from Brown University, Providence, Rhode Island. In 1984 he became a Research Staff Member at the IBM Thomas J. Watson Research Center in the Artificial Intelligence Department of the Computer Science Departament. In 1988 he became manager of the newly formed Exploratory Computer Vision Group wich is part of the Math Sciences Department.

Currently, his research interests are focused on video database indexing, video processing, visual human-computer interaction and biometrics applications. Ruud M. Bolle is a Fellow of the IEEE and the AIPR. He is Area Editor of Computer Vision and Image Understanding and Associate Editor of Pattern Recognition. Ruud M. Bolle is a Member of the IBM Academy of Technology.

Ruud Bolle
video database indexing video processing visual human-computer interaction biometrics applications
Research_Interest
IBM T.J Watson Research Center
Affiliation
IBM T.J Watson Research Center 19 Skyline Drive, Hawthorne, NY 10532 USA
Address
Research Staff
Position
http://researchweb.watson.ibm.com/ ecvg/people/bolle.html
Homepage
IBM T.J Watson Research Center P.O.Box 704 Youktown Heights, NY 10598 USA
Address
Photo
Email
bolle@us.ibm.com
Ruud Bolle
Name
Bsdate
1977
1984
Phddate
Phduniv
Bsuniv
Delft University of Technology
Brown University
Phdmajor
Msdate
Bsmajor
Analog Electronics
Msmajor
Msmajor
Electrical Engineering
1980
Electrical Engineering
Msuniv
Delft University of Technology
Applied Mathematics

Figure 2. 3 – An Example Web Document Semantic Annotation (Adapted From: Tang et al., 2012)

### 2.2.1 Manual Annotation

Manual annotation follows strict guidelines and changes the current syntactic resources to interlinked knowledge structures by adding information to some level of document that incorporates metadata (Slimani, 2013). It can offer more precision when compared to semi-automatic and automatic annotation methods due to the full human involvement in the process which means the annotations can be monitored and appropriately implemented by the user. Tools like OntoMat and SHOE provide an integrated environment for concurrently annotating and authoring text or documents (Reeve & Han, 2005). However, it is very labour intensive and due to the use of human annotators, the system is exposed to errors caused by factors like complex schemas, volume of required training and sometimes the annotators familiarity with the domain can pose a challenge.

Manual annotation does not take into consideration multiple perspectives of a data source, that requires multiple ontologies of which can be valuable in supporting the needs of diverse users (Tang et al., 2012). This also implies that manual annotation for documents requiring very high levels of precision and aggregation of multiple resources for the annotation can quickly become a daunting task due to the complexities inherent in such processes. With the scale of the web, running billions of web documents, a manual process for applying semantic annotations to web documents is not feasible (El-Ghobashy et al., 2014). Table 2.1 presents a list of some manual semantic annotation tools.

Table 2. 1 – Common Manual Semantic Annotation Tools

| **Manual Tools / Research Artefacts** | **Source** |
|---|---|
| Thresher, AKTiveMedia | Oliveira & Rocha (2013) |
| Annotea, Yawas | Nacer & Aissani (2014), Belloze et al. (2012) |
| SHOE, Smore, Meteor-S | Nacer & Aissani (2014) |
| Annozilla, Melita, Knowtator, RDFace, MnM, Melita | Belloze et al. (2012) |
| RadiantWeb | Guttula (2012) |
| Amaya, Mangrove, Vannotea | Uren et al. (2006) |
| WebAnno | Yimam et al. (2014) |

### 2.2.2 Semi-Automatic Annotation

With this type of annotation, while some processes are automated, the overall semantic annotation still requires a significant level of intervention by humans. The weight of how much automation or manual work involved varies from one approach to another based on the different tools available (Dammak et al., 2013). The automated and non-automated tasks across current tools also vary. While semi-automatic annotation of web documents has been proposed in different quarters to overcome the challenges with manual annotations and is the method mostly used in current systems (Tang et al., 2012), it still faces most of the challenges of manual annotation (such as being time-consuming, tedious and cumbersome) and cannot be adopted as the means for annotating over 7 billion web pages available on the web today. The need for human intervention is still a limitation, especially with regards to scalability (El-Ghobashy et al., 2014). There are also a wide range of variations in architectural patterns, techniques for text analysis, ontology engineering tasks, annotation data storage methods as well as automation levels among current semi-automatic tools; posing a selection and service type challenge for users (Slimani, 2013). Table 2.2 presents a list of some popular semi-automatic semantic annotation tools.

Table 2. 2 – Common Semi-Automatic Semantic Annotation Tools

| **Semi-Automatic Tools / Research Artefacts** | **Source** |
|---|---|
| Cerno, Lixto, Semantic MediaWiki, Zemanta, AeroDAML, Armadillo, KnowWe, CREAM | Oliveira & Rocha (2013) |
| Semantator | Tao et al. (2013) |
| Analec | Landragin et al. (2012) |
| Argo | Batista-Navarro et al. (2016) |
| OntoMAT | Gawich et al. (2012) |
| Autometa, | Belloze et al. (2012) |
| Visual OntoBridge | Grcar & Mladenic (2009) |
| GonTongle | Giannopoulos et al. (2010), Belloze et al. (2012), Oliveira & Rocha (2013) |
| GateCloud | Belloze et al. (2012), Gawich et al. (2012) |
| Aatos | Tamper et al. (2017) |
| PANKOW, Muse, Amilcare, S-CREAM | Reeve & Han (2005) |
| Ontea | Laclavık et al. (2006) |
| Domeo | Ciccarese et al. (2012) |
| RDFa Editor | Duma (2011) |
| Marcinczuk et al. (2012), Dammak et al. (2013), Neveol et al. (2011), Liu et al. (2009), Davis et al. (2009) | |

### 2.2.3 Automatic Annotation

Automatic Annotation defines a fully automated approach to semantic annotation. In this case the annotation process is not expected to require human intervention, instead is to be fully implemented by computing systems. This eradicates issues associated with both manual and semi-automatic semantic annotation such as being time-consuming, tedious and cumbersome. Automation for semantic annotation remains the major viable means of annotating existing web data as well as the vast amount added to it on a momentary basis (Liu et al., 2017). Automated annotation also provides the scalability needed for existing documents on the web and reduces the burden of annotating new ones. Other potential benefits are consistently applying ontologies and using multiple ontologies to annotate documents through an automated system (Tulasi et al., 2017). An all-in-one automatic semantic annotation platform that is scalable enough for the web and captures the entire process of semantic annotation, with additional requirements such as ontology engineering tasks and annotation data lifecycle management to provide an automated system for semantic annotation is currently unavailable to the best of the writer's knowledge. SemTag, which uses a Seeker engine for automatic semantic annotation is one of the few with a sizeable level of scalability. However, it is still vastly limited in many ways, more predominantly because its annotation source is a single taxonomy (known as TAP) as compared to the flexibility of integration with multiple OWL-compliant ontologies (Oliveira & Rocha, 2013).

In comparison with manual and semi-automatic semantic annotation, the challenge to automatic annotation however is the potential of the annotation not being fully accurate and with the human intervention removed from the process, such inaccuracies are not addressed immediately. Having said that, the availability of annotation accuracy measurement indexes means the automation can be tested for accuracy with widely accepted indexes such as Precision, Recall and F-Measure (Liu et al., 2017). With global knowledge graphs such as DBpedia and Linked Open Data Cloud receiving detailed attention and large communities for their management, annotation accuracy is expected to continue to increase significantly. A list of existing semantic annotation tools with a high level of automation can be found on Table 7.1 in section 7.2. Table 2.3 presents a comparison of the three types of semantic annotation described.

Table 2. 3 – Classification for Semantic Annotation Methods

| Methods | Description | Advantages | Disadvantages |
|---|---|---|---|
| **Manual Annotation** | The process in which humans manually add metadata to data. | 1. Provides a very high level of accuracy. | 1. Tedious and cumbersome<br>2. Time-Consuming<br>3. Long-Winded<br>4. Not applicable for web-scale |
| **Semi-Automatic Annotation** | The process of automating certain tasks within the annotation process of data. One major task often automated is information extraction and retrieval using methods such as Named Entity Recognition, Co-reference Resolution, etc. | 1. Provides a reasonably high level of accuracy.<br>2. It also provides a means for humans to make corrections where necessary. | 1. Quite tedious and time-consuming<br>2. Not ideal for annotating the vast amount of web data. |
| **Automatic Annotation** | The process of eliminating the requirement of a | 1. Quick. | 1. Processing-Overload: It requires a lot of machine |

| | | | |
|---|---|---|---|
| | human intervention in the annotation process for web data, making it fully machine processed. | 2. Feasible as a solution for web data annotation.<br>3. A self-learning iterative annotation process would provide a very high level of accuracy. | processing capabilities and dynamism for machine resources allocation for on-demand semantic annotation. |

## 2.2.4 The Role of Ontologies

An Ontology is defined as an "explicit, formal specification of a shared conceptualisation". The term emanated from philosophy where it refers to a logical account of existence (Gruber, 2007). It defines a representation for a knowledge domain; providing a formal description of concepts and their relationships resulting in a shared understanding (Munir & Anjum, 2018). With the current evolvement of a semantic web, the need for standards to facilitate it is very vital. Ontologies provide this by means of defining data model schemas, which are utilised by annotation data in the semantic annotation process (Luczak-Rosch et al., 2014). With ontologies being developed using scientific programming languages, it also implies that machines can easily understand the annotation they provide to web documents and further assist humans with information usage, extraction and retrieval on the web. They consist of terminologies or vocabulary within the specified domain and the relationships between them. While some generic ontologies exist, most ontologies are domain specific. Figure 2.4 presents a visual representation of an ontology for a 'Food Product', which is a natural domain.

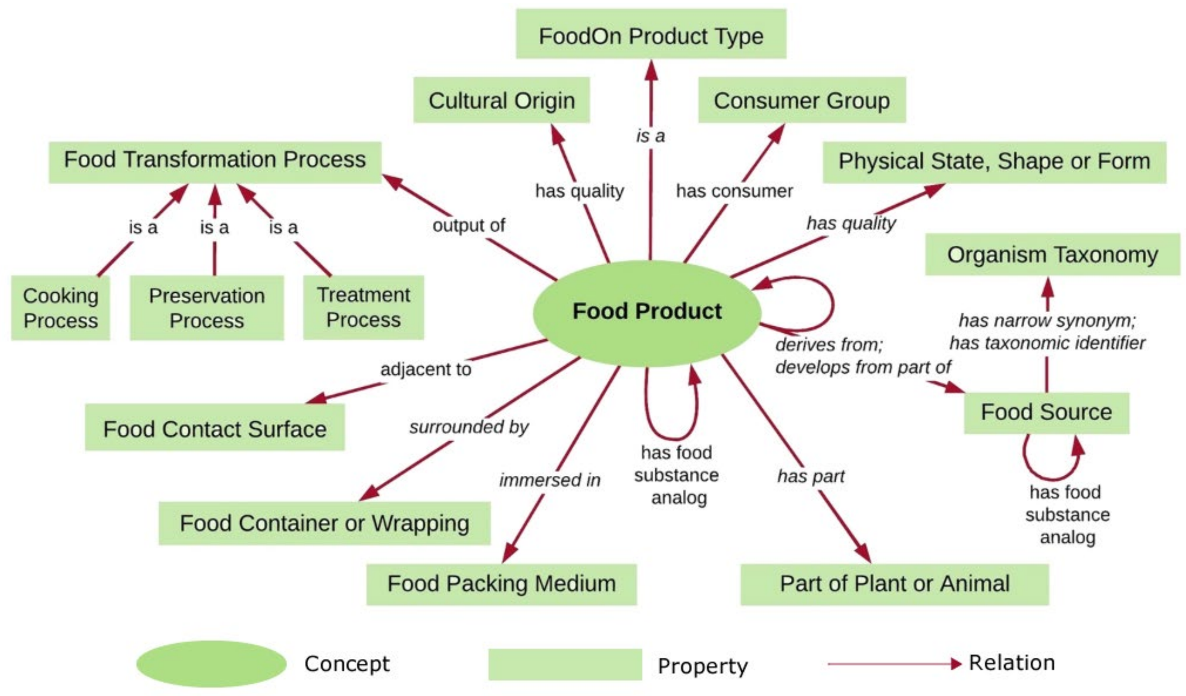


Figure 2. 4 – A Visual Representation of an Ontology (Adapted From: Dooley et al., 2018)

There are different classification perspectives for ontologies in research. Slimani (2014) proposed a classification based on the problems that ontologies have been developed to solve; defining Terminological Ontologies as based on terms utilised in the construction of knowledge representation repositories, Information Ontologies for specifying the schema of data in data storage media such as files and databases, as well as Knowledge Modelling Ontologies which are developed for describing and conceptualising knowledge in specific domains.

However, Fensel (2011) classified ontologies based on their generality and role in building knowledge-based systems, proposing domain, metadata, generic, representational and task ontologies. Similarly, Guarino & Musen (2015) proposed ontology classifications based on the level of generality of ontologies, defining top-level, domain, task and application ontologies. In a similar way to the availability of varying classifications for ontologies, several methodologies exist for engineering them as well. Some of these are MENELAS, IDEF5, Methontology and DILIGENT (Iqbal et al., 2013). Ontology engineering tools include Protégé, OntoEdit, OilEd, Ontolingua, Chimaera, SymOnto, WebOnto and Integrated Ontology Development Environment (Slimani, 2015). Furthermore, some primary uses of ontologies across several industries such as health, education, engineering and IT as identified by Vegetti et al. (2016) and Slimani (2015) include the following:

- To share a common representation of knowledge among both computers and humans

- To facilitate knowledge reuse within a specific domain. This also prevents re-inventing the wheel and introduces standards that will consequently facilitate interoperability
- To provide clear and concise context within domains
- To separate domain and operational knowledge.

The lifecycle of developing ontologies as well as other related tasks and activities associated with them is referred to as Ontology Engineering. Figure 2.5 presents a classification for different areas of ontology engineering.

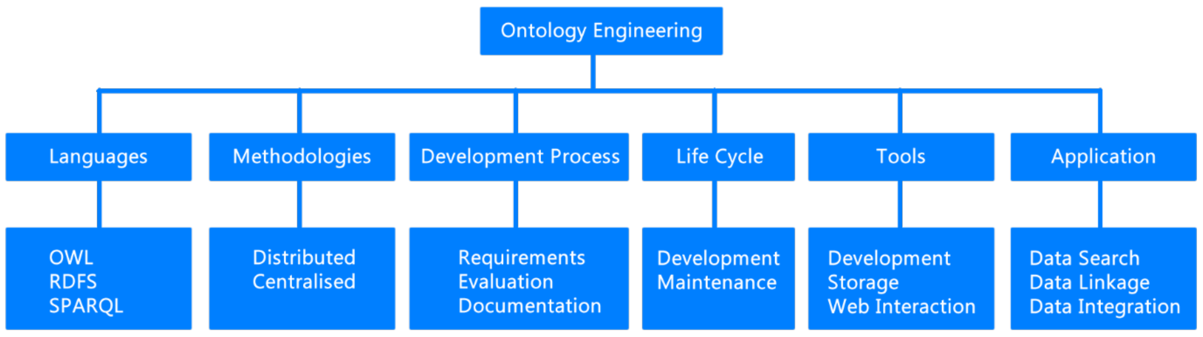


Figure 2. 5 – Classification of Ontology Engineering Activities

Ontology Languages refers to scientific programming languages for building ontologies, such as OWL and RDFS. Some variations across these include their level of expressivity and standardisation (Akgun & Ayvaz, 2018). Regarding development processes, default software engineering steps such as requirements gathering, evaluation and documentation also exist with developing ontologies, with different ontology methodologies defining development and maintenance lifecycles. The building or compiling of a comprehensive taxonomy with well-defined relationships between concepts and relations for use in developing ontologies is also a major phase required for the development (Vegetti et al., 2016).

The analysis and evaluation of ontologies requires consensus on the entire representation among knowledge experts within the domain being modelled. After the development of ontologies, a maintenance phase is required to ensure that the concepts and relations defined within the ontology remain accurate based on updates to ontologies and their repositories. Furthermore, ontology engineering tasks can be classified under phases; management, pre-development, development, post-development and support. This is illustrated with Figure 2.6.

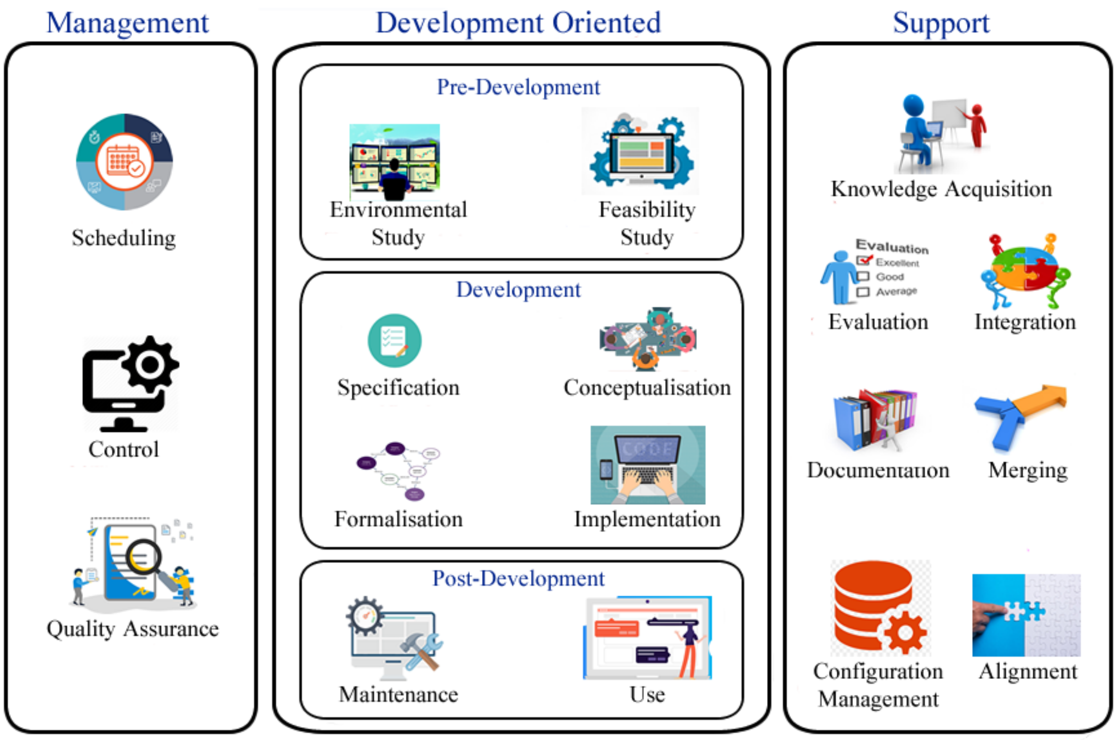


Figure 2. 6 – Phases of Ontology Engineering (Adapted from SlideWiki, 2016)

As it can be observed from Figure 2.6, the support phase involves tasks to integrate multiple ontologies together. Ontology Integration tasks are fuelled by different reasons. However, a prominent reason is to address the overlap which exists within different ontologies, in which their aggregation would result to a wider scope of formal representation for information in the domain (Johnson et al., 2012). The representation of concepts within ontologies vary from one to another and integrating them can help identify and resolve such cases. Several factors can be identified which leads to a single concept being represented in different ways within ontologies. These include the use of a single term to describe different concepts referred to as ambiguity, the use of different terms to describe a single concept referred to as redundancy and the use of different types of representation for describing concepts within ontologies.

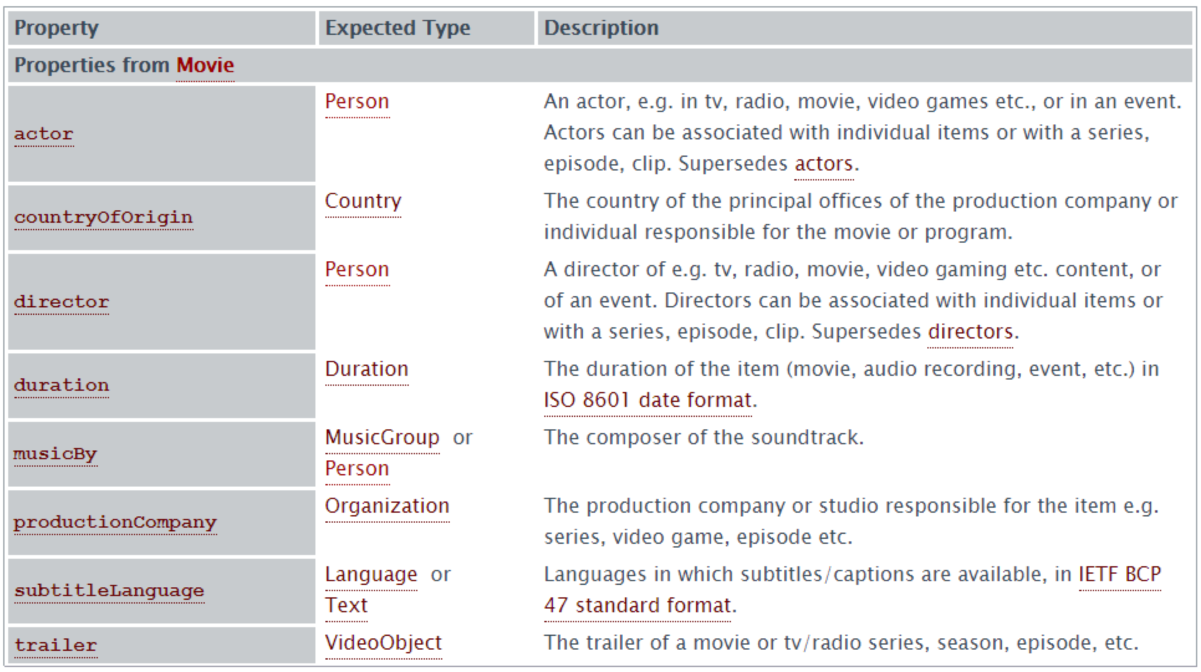
Furthermore, ontologies can be developed to different levels of granularity or depth and from different perspectives; factors which can also result into different formalisms of concepts across ontologies. Likewise, there are challenges common to the different approaches or methods of integrating ontologies (Harrow et al., 2019). These are actively being engaged in research and it will be required that results/findings from research are taken into consideration for any method adopted. According to Da Silva & Cavalcanti (2014), some of these include:

- **Different naming conventions** – naming conventions for concepts and terms within ontologies vary and these variations will imply difficulties in being able to match two or more terms that mean the same together without some additional input into recognising such similarities.
- **Lack of reliable textual definitions** – some knowledge domains do not have existing comprehensive and standard textual definitions for terms. A comprehensive vocabulary is required for such domains first before ontologies can be used or integrated.
- **Lack of formal categorisation** – the principles applied in classifying terms belonging to same sub-domains requires consistency in order to ensure that the relationships between different terminologies are well defined and established.
- **Different theories of a single domain** (such as modern medical knowledge and pre-scientific representations of the human body) - For instance, the term 'Cold' can be defined within different domains as meaning a sensory perception, an upper respiratory viral infection or a pulmonary diagnosis known as Chronic Obstructive Lung Disease. This creates ambiguity in defining the term within ontologies and subsequently mapping or matching it correctly with similar terms as required.

## 2.3 Schema.org Markup

Schema.org provides syntax for annotating parts of documents with metadata based on definitions provided to objects in the Schema.org repository. The objects belong to different schemas, referred to as 'Types', with over 600 Types containing hundreds of properties and enumeration values at the time of writing (Schema.org, 2019). Some common 'Types' include places, people, things and organisations. This markup makes search results more meaningful to users based on the additional interpretation it provides to small chunks of data on web pages. These small chunks are referred to as Google Rich Snippets on the Google search engine (Guha et al, 2016). An example is a Google search for a product to buy online and the Google search engine returning relevant results that contain information about the price of the product, the number of reviews for the product, and so on. These snippets enable users make better, timely and informed decisions for results from a web search (Ambiah & Lukose, 2012). With

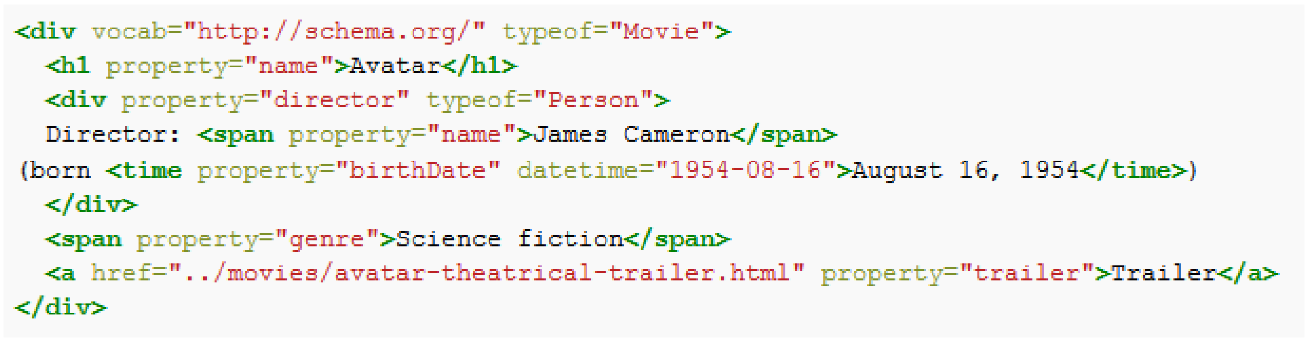
Schema.org, markup is added to existing HTML tags within web documents. Figure 2.7 displays a property table from Schema.org.



| Property | Expected Type | Description |
|---|---|---|
| **Properties from Movie** | | |
| actor | Person | An actor, e.g. in tv, radio, movie, video games etc., or in an event. Actors can be associated with individual items or with a series, episode, clip. Supersedes actors. |
| countryOfOrigin | Country | The country of the principal offices of the production company or individual responsible for the movie or program. |
| director | Person | A director of e.g. tv, radio, movie, video gaming etc. content, or of an event. Directors can be associated with individual items or with a series, episode, clip. Supersedes directors. |
| duration | Duration | The duration of the item (movie, audio recording, event, etc.) in ISO 8601 date format. |
| musicBy | MusicGroup or Person | The composer of the soundtrack. |
| productionCompany | Organization | The production company or studio responsible for the item e.g. series, video game, episode etc. |
| subtitleLanguage | Language or Text | Languages in which subtitles/captions are available, in IETF BCP 47 standard format. |
| trailer | VideoObject | The trailer of a movie or tv/radio series, season, episode, etc. |

Figure 2. 7 – A Sample Property Table from Schema.org Vocabulary (Schema.org, 2019)

Schema.org uses standards such as Microdata, RDFa and JSON-LD for the markup in web documents (Navarrete & Lujan-Mora, 2018). It extends the expressive power of HTML, though the expressiveness is still limited. Figure 2.8 presents an example of Schema.org markup using RDFa.



```
<div vocab="http://schema.org/" typeof="Movie">
  <h1 property="name">Avatar</h1>
  <div property="director" typeof="Person">
  Director: <span property="name">James Cameron</span>
(born <time property="birthDate" datetime="1954-08-16">August 16, 1954</time>)
  </div>
  <span property="genre">Science fiction</span>
  <a href="../movies/avatar-theatrical-trailer.html" property="trailer">Trailer</a>
</div>
```

Figure 2. 8 – An Example of Schema.org Markup using RDFa (Schema.org, 2019)

While the Schema.org project has been a success so far for adding markup to existing HTML tags within documents thereby extending the expressiveness of HTML and postulating an evolutionary rather than a revolutionary web, it is not semantic annotation and requires technical expertise for the addition of structured metadata to web content. Its primary use is for the convenience of a group of search engines; in order to better understand web content and return search results based on this understanding.

## 2.4 Cloud Computing

Cloud Computing refers to a model of enabling on-demand and convenient access to a shared pool of computing resources, such as networks, applications, servers, services and storage (Jula et al., 2014). This definition highlights the ability of computing resources to be provisioned and released with minimal interaction between the user and the cloud service provider. Senyo et al. (2018) reiterates this by drawing attention to the ability of the cloud to enable a simple way of provisioning information technology as a service to users, rather than as a product through the Internet. Some popular cloud service providers include Google, Amazon, Microsoft, and Yahoo. Cloud users are abstracted from computing resources and do not actually know where data or applications are stored in the cloud (Namasudra et al., 2017). Recently, many organisations have been migrating to the cloud to enjoy the benefits offered by the technology, such as: affordable access, high performance resources, low maintenance costs and enhanced security (Zhang et al., 2017).

### 2.4.1 Cloud Computing Delivery Models

There are three main delivery models in cloud computing – software-as-a-service, platform-as-a-service and infrastructure-as-a-service. Other categories of delivery models may exist as a cross between any or all the three main delivery models. Software-as-a-service (SaaS) involves provisioning of applications on the cloud by a service provider, to be used by several customers in a multi-tenacity setup (Mell & Grance 2011). SaaS offers software to cloud consumers on-demand. Common examples of SaaS solutions include web-based emails such as Gmail and Yahoo Mail, Google Apps, Dropbox, and Salesforce CRM (Namasudra et al., 2017). In most cases, SaaS services are accessed via simple client interface such as web browsers. The application provider is responsible for managing technology resources and the solution's performance. SaaS services are billed based on a pattern which can be hourly, weekly, monthly or yearly.

Platform as a Service (PaaS) is another cloud computing delivery model that allows clients to access various capabilities, such as programming languages, services, tools and libraries; provisioned and managed by providers (Mell & Grance 2011). As a result, the user transfers the responsibility of controlling cloud operational platforms and infrastructure, such as network, operating systems, memory and servers to a cloud vendor. The PaaS delivery model reduces the cost and complexity of purchasing and supporting both software, hardware and hosting resources (Diaby & Rad, 2017). However, the role of deploying and supporting the application remains with the client. Some popular examples of PaaS include Azure and Aneka.

Infrastructure as a Service (IaaS) involves utilisation of computing resources such as: processing, networks and memory from a vendor (Rittinghouse & Ransome, 2017). Users can deploy their platforms such as operating systems and different applications without worrying about the necessary cloud infrastructure. In effect, the control for processing, networking, and storage is transferred to the vendor. Some examples of IaaS solutions are GoGrid and Amazon EC2.

### 2.4.2 Cloud Computing Deployment Models

There are four main deployment models in cloud computing – private cloud, community cloud, public cloud, and hybrid cloud (Mell & Grance, 2011). The private cloud model involves provisioning cloud resources for use by a single entity with several units or consumers (Mell & Grance, 2011). A private cloud can be owned and controlled by the enterprise or outsourced to a third-party service provider. It can either be on premise or externally hosted (Diaby & Rad, 2017). Some of the benefits of this cloud deployment model include enhanced security, dedicated resources and improved customisation (Diaby & Rad 2017). For community clouds, they are provisioned for utilisation by a group of consumers referred to as a community (Liu et al. 2011). In most cases, enterprises forming a community have a shared factor, such as security requirements and compliance considerations (Mell & Grance, 2011). A community cloud can be managed by either one of the organisations in the group or outsourced to a service provider.

On the other hand, public clouds are provisioned for public use (Liu et al. 2011). An example is a cloud service provided by academic institutions or government organisations. Liu et al. (2011) noted that a public cloud is also owned by a vendor selling cloud services to several diverse users. Benefits of this deployment model include flexible environment, freedom of self-service, pay-per-use, increased availability and reliability (Diaby & Rad, 2017). For hybrid clouds, two or more cloud deployment models can be combined using standardised or proprietary strategies to create it (Liu et al., 2011). The advantages of a hybrid cloud include optimal use, data centre consolidation, enhanced availability, and risk transfer (Diaby & Rad 2017).

### 2.4.3 Cloud Computing Essential Characteristics

There are five main characteristics that help in the understanding of what cloud computing is, what it does and how best to maximise potential benefits of the cloud as defined by NIST (National Institute of Standards and Technology) (Mell & Grance, 2011). Firstly, with On-Demand Service, cloud services such as storage, hardware and operating systems can be provisioned automatically by cloud consumers when needed, providing opportunity for the reservation and release of IT resources in an independent manner to meet their needs. This involves what is referred to as 'provisioning' and 'decommissioning' (Liu et al., 2011). The provisioning for these functionalities can be through graphical user interfaces (GUIs) or command line interfaces (CLIs). They can also be through application programming interfaces (APIs) meant for use in automation. This feature allows a resource changing procedure that involves updating configuration parameters as well as adding new nodes to the cloud environment.

Broad Network Access is another of the five essential characteristics. With cloud computing, services are provisioned by vendors located in diverse geographic locations and accessed through global reach capability (Diaby & Rad, 2017). The broad network access feature ensures that cloud computing services are available via the Internet and accessible through diverse devices such as tablets, workstation computers, laptops, and smartphones. The next one is Resource Pooling in which the cloud service users are abstracted from mechanisms that facilitate resources provisioning, creating the impression that resources are from a single blended resource. This capability enables providers offer a set of real and virtual resources dynamically (Diaby & Rad, 2017). A cloud service provider can pool computing resources to serve several users on a multi-tenant model, with diverse computing resources, both physical and virtual that are dynamically assigned on-demand.

Furthermore, there is Rapid Elasticity which refers to the ability of cloud computing technology to scale resources up or down based on demand from users (Rittinghouse & Ransome, 2017). It is a crucial characteristic of cloud computing since it implies that organisations can rapidly provision or de-provision resources without user interaction. As cloud users have different workload requirements, there is a "levelling" so that unused resources by some users are readily deployed to users with higher demands at that moment. In addition to being able to distribute workload among independent resources, the elasticity has the capability to free up resources flexibly when they become under-utilised (Liu et al., 2018). Lastly, there is Measured Service, which refers to the ability to automatically monitor, optimise and provide reports for the utilisation or consumption of different cloud resources over a period. It allows for cloud computing service usage metering, to enable cloud consumers pay only for what they use (Saxena & Pushkar, 2016). This charge per use strategy helps optimise resources utilisation.

Saxena & Pushkar (2016) further stated some cloud computing features such as Mobile Accessibility and Shared Infrastructure as essential for information systems today. Mobile Accessibility referring to the ability of users to access cloud resources while mobile and without any degradation in the service or its accessibility. Shared Infrastructure on the other hand, referring to the ability of cloud providers to deploy a virtualised software model to enable sharing of physical computing resources, and dynamic provisioning that involves automatic allocation of resources based on demand.

### 2.4.4 Cloud Computing Benefits

Cloud Computing offers a wide range of benefits, which includes the ability to rapidly provision computing resources; both hardware and software as well as the potential to dynamically scale resources based on user demands; a feature which goes a long way in optimising cloud computing resources, delivers proven results and saves cost (Namasudra et al., 2017). Cloud providers are responsible for supporting and maintaining IT resources, effectively relieving users of the burden of purchasing expensive computing components such as servers, operating systems, networking tools and so on (Mell & Grance, 2011). Another benefit of cloud computing is elasticity (Jula et al., 2014). Users can request for computing resources depending on their requirements, anytime, anywhere, if there is network connectivity.

Business continuity is also achieved using cloud technology (Namasudra et al., 2017). An enterprise can deploy cloud backup solutions for storing systems and crucial information. In case of natural disasters, the information will not be affected. Moreover, cloud providers invest in security controls which ensures that the business is not adversely affected by cybercriminal activities. Cloud computing offers flexible work practices and speed (Rittinghouse & Ransome, 2017). Users can access systems and data from any location via a network connection. This feature enhances productivity and mobility, since employees can work at home or while travelling.

### 2.4.5 Cloud Computing Issues

Security and privacy are some of the crucial challenges faced while using cloud computing services (Veloudis & Paraskakis, 2016). Since systems and data are hosted in remote servers, there is the risk of exposure to hackers and other unauthorised users attempting to gain access to the cloud resources for various reasons such as financial gain and other malicious purposes. Another issue for cloud computing service users is downtime (Namasudra et al., 2017). In case the service is down, or the network has issues, users cannot access applications or data from cloud locations. Moreover, cloud clients have limited control over their information and systems since they are stored or running on remote servers owned by vendors. Finally, data and application interoperability remain a crucial issue in the cloud (Saxena & Pushkar, 2016). It is vital that cloud data and systems use standard interfaces to allow interoperability regardless of the delivery or deployment model adopted.

## 2.5 Semantic Web and Cloud Computing

This section focuses on the review of literature for research efforts which involves an integration of two or more standards across both semantic web and cloud computing technologies. With the possibility of leveraging each other, section 2.5.1 reviews literature on cloud computing leveraging semantic web technologies while section 2.5.2 reverses the order, by reviewing semantic web technologies leveraging cloud computing. For both instances, it can be observed that there is a high level of interaction between them as the following sections analyse.

### 2.5.1 Leveraging Semantic Technologies for Cloud Computing

Several semantic technologies and solutions have been implemented for cloud platforms, making them semantic in nature. A semantic cloud fosters the efficiency of a cloud platform with respect to services across the different delivery models (IaaS, PaaS and SaaS). This is usually implemented using semantic technologies such as RDF to model data for cloud services and OWL or RDFS to develop ontologies for cloud models. Currently, ontologies exist for providing metadata for cloud entities' description. However, most of these still require further enhancement and enrichment (Rodriguez-García et al., 2014). From the review of existing literature on semantic cloud, a classification for the utilisation of semantic technologies for cloud platforms is presented and evaluated as follows:

### 2.5.1.1 Cloud Interoperability and Portability

While interoperability between applications at the software-as-a-service level in cloud computing presents challenges, semantic technologies have increasingly been used to address them and overcome some of the barriers. The work by Rezaei et al. (2014) asserted that semantic technologies are the fundamental prerequisites towards achieving interoperability in the cloud. The implementation of semantic interoperability frameworks is critical for software-as-service systems within the cloud. In addition, semantic technologies are used to provide comprehensive service specification across various abstraction levels and service categories. For instance, Fang et al. (2016) proposed a fuzziness-embedded and agility-oriented semantic model that captures cloud interactions and details across different abstraction levels including SaaS, PaaS and IaaS. The model can be used to reveal multiple agile interactions among the resources and services within a cloud computing environment. Figure 2.9 presents the agility-oriented ontology design of the work by Fang et al. (2016).

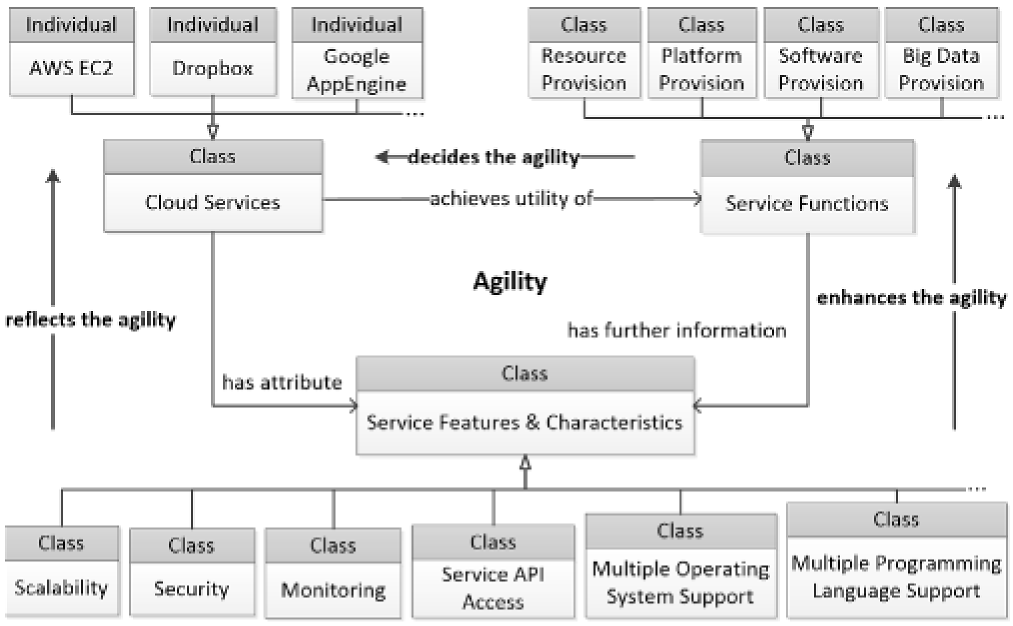


Figure 2. 9 - Agility-Oriented Semantic Model for Cloud Services (Fang et al., 2016)

In addition to enhancing the interoperability of cloud-based applications and inter-cloud policies, semantic technologies are also critical in resource scheduling and provisioning in an inter-cloud environment. Particularly, ontology-based resource description helps in solving inter-cloud interoperability problems making it possible for proper resource provisioning from different cloud service providers. The work by Di Martino & Esposito (2016) proposed an Inter-Cloud Resource Provisioning System (IRPS) that allows for semantic description of tasks and resources. The system also facilitates the storage of such tasks and resources using resource ontology which in turn facilitates proper resource allocation based on a semantic scheduler and inference rules. Similarly, Somasundaram et al. (2012) developed a broker-based architectural model aimed at the problem of interoperability between OpenNebula and Eucalyptus. This framework integrates semantic-based resource discovery, capacity-based resource provisioning and selection mechanism. Overall, semantic technologies have been observed to be critical in solving inter-cloud interoperability issues, enhancing scheduling and provisioning success rate, increasing the cloud resources' efficiency and enhancing the throughput of cloud-based applications.

#### 2.5.1.2 Discovery, Selection and Utilisation of Cloud Services

Another important application of semantic technologies in the cloud is in their use for the discovery, selection and utilisation of cloud services (Rekik et al., 2015). One of such uses is for the enhancement of topic coherence. Without the use of semantic technologies such as RDFS, OWL and SPARQL, it is difficult to discern whether a set of annotation data is related to a certain topic. Considering this, a cloud transformation model was developed by Zhang et al. (2015) which not only determines the relationship between annotation data and a topic but also integrates the annotation data into the necessary topic model. This model enhances performance and reduces noise while integrating semantic knowledge into Tag-LDA model. Furthermore, semantic technologies have proven to be vital for testing reproducibility of scientific experiments. The reproducibility of results obtained from scientific experiments is regarded as the cornerstone of any scientific method (Zhang et al., 2015). Unlike the conventional techniques of addressing reproducibility, semantic tools provide scientists with a platform for sharing and capturing valuable knowledge regarding computational experiments' equipment, enabling them to capture the execution environment under which the scientific experiments are performed and share them through the cloud. Specifically, Santana-Perez et al. (2017) proposed a novel approach that describes scientific workflows' execution in the cloud using semantic vocabularies.

The work by Alti et al. (2015) proposed a multi-level ontology-based architecture referred to as OntoSmart which can be used in enhancing "the high level of context concepts abstraction for heterogeneous service sources and profiles using a top-level ontology". This architecture can be beneficial in overcoming the barrier resulting from the heterogeneity and diversity of cloud-based service sources and profiles. In addition, semantic technologies are critical in making the cloud more scalable so that it can interconnect a vast number of servers while supporting a variety of online services within the cloud environment. This can be observed from the work of Hua et al. (2014) which presented a scalable and distributed data-centric system referred to as “Antelope” for cloud computing data centres. The system can be used to overcome the possibility of mismatching between the data placement and the network architecture, taking into consideration the data placement’s optimisation as well as the network architecture’s property. Its underlying concept is leveraging the precomputation-based data cube to enhance online services hosted on cloud platforms.

In addition, semantic technologies enhance the capabilities of cloud-based platforms in terms of management and knowledge representation, bridging semantic resources together in distributed cloud-based platforms and allowing for the interconnection of cloud-based heterogeneous services with respect to flexibility and interoperability in a virtual organisational schema (Pileggi et al., 2013). Semantic technologies are also critical in enhancing service access and discovery within cloud environments. The work by Cortazar et al. (2012) proposed a cloud computing ontology that allows for semantic access, identification and discovery of cloud-based services. Additionally, semantic technologies help in the integration of cloud services among different cloud-based platforms. In addition, Trajanov et al. (2012) proposed a framework referred to as “Semantic Sky” as a platform that allows the integration of many cloud-based services via semantic technologies. This system is capable of automatically discovering the user’s cloud context and offering the necessary actions which can be executed with the data within the context. It also automates the process of executing users’ tasks and thus resulting in improvements to the users’ efficiency, information exchange and productivity. Similarly, Dautov et al. (2013) proposed an approach that allows for self-managing capabilities of cloud application platforms. The approach perceives cloud computing platforms as “networks of distributed data centres”.

#### 2.5.1.3 Cloud Security

Semantic-based approaches have also been deployed to enhance security mechanisms within a cloud environment. Privacy and security concerns have made organisations reluctant to shift their respective business operations to the cloud (Veloudis & Paraskakis, 2016). Hendre & Joshi (2015) also stated that cloud providers are required to adhere to necessary privacy and security policies towards ensuring users’ data are kept secure and confidential. Their research led to the development of a semantics-enabled application that allows cloud users to identify the necessary cloud compliance and policy statements required for their organisations. The application also facilitates the identification of privacy and security threats within a cloud computing environment and the compliance and security models against such threats.

Semantic technologies are also used for retrieving encrypted data from cloud environments. Yang (2015) emphasised the vital role encryption plays in protecting the privacy and security of data before and after transfer to a cloud platform. As such, semantic technologies are used to overcome the limitations of traditional data retrieval methods such as keyword search. The work by Xia et al. (2014) further stated that there are various searchable encryption techniques for performing searches on secure outsourced data. Boneh et al. (2004) also proposed a "Public Key Encryption Scheme with keyword search (PEKS)" as a solution to the problem of searching on encrypted data. Similarly, Attribute-Based Encryption (ABE) schemes enhance the flexibility of accessing confidential data as well as the ease of sharing such data, as illustrated by Figure 2.10 which presents a search model of encrypted data in cloud.

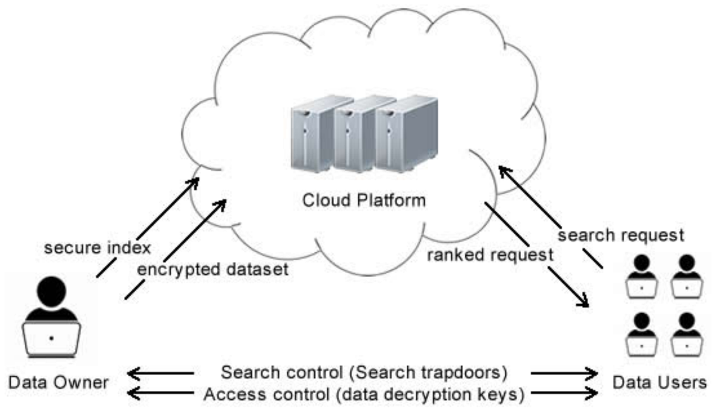


Figure 2. 10 – A Search Model over Encrypted Data in Cloud. (Zhang et al., 2015)

### 2.5.1.4 Description of Cloud Resources and Services

Contextual description of resources and services in the cloud greatly enhances their effective use. One of the uses of semantic technologies for cloud is the monitoring of systems. Ward & Barker (2012) proposed "a scalable distributed data collection system" as a tool for monitoring cloud systems. This monitoring includes server resources utilisation levels such as processing power consumption, disc storage usage, etc. Data collected from the monitoring is used to provide appropriate, real-time statistical status for cloud resources. The system is based on RDF rather than flat files or relational databases. Chernyshov et al. (2016) defined RDF as a straightforward way of describing instance data in the subject-object relation using resource identifiers. The RDF's vocabulary is extensible through other schemas that facilitate the generation of comprehensive ontologies to represent any problem domain. Unlike other non-semantic enabled cloud monitoring tools that utilises flat files or relational databases, Ward & Barker's (2012) proposed system employs RDF which provides storage for all machine-readable information, providing a means for computers to be able to understand and process the data accordingly.

### 2.5.2 Leveraging Cloud Computing for the Semantic Web

The semantic web is based on the utilisation of technologies from the semantic web stack to provide context-awareness for web documents. This section provides an investigation and assessment of a semantic web that is driven by cloud computing. The focus is on leveraging cloud computing for the semantic web by maximising the full benefits of cloud computing based on its nature and characteristics. These includes the use of cloud computing mechanisms to analyse, query and reason with the massive amounts of metadata for the semantic web. Metadata management for the semantic web requires a very high level of automatic scalability which cloud computing can provide. The implementation of a solution such as Hadoop MapReduce in a cloud environment for semantic metadata operations also provides a basis for leveraging cloud computing for the semantic web.

One of the key issues with the semantic web is scalability. It arises when organising, storing and retrieving semantic metadata for the vast amount of web documents accessed concurrently at any given time (Manzoor et al., 2014). The semantic web forms a global graph where SPARQL is used to retrieve these links. These SPARQL queries may be required to navigate through several web and database servers joining the links within RDF databases. Retrieving a link and accessing a web document requires proof and trust processes as well. All these demands high computational resources due to the number of documents and servers involved. The access operation in normal circumstances means instantaneously handling millions of user requests which would result in extended processing times if handled in this manner. To eliminate this issue, cloud computing mechanisms can be implemented for the provision of high-performance computing which subsequently, reduces the processing time and high computational cost that would have ensued (Erl et al., 2015). While semantic web technologies provide standards for defining context for data and relationships between them, cloud services have the capability to provide a platform where these data can be stored and processed accordingly.

The research by Mika (2008) considered web semantics in the cloud, providing an overview of semantic web technologies and three various aspects that have to do with semantic web technologies in the cloud. Firstly, cloud computing for web data was discussed; these web data includes metadata obtained from applications that run on the web and computational data produced via search engines. Due to the large amount of data generated from the web, the use of cloud services was proposed as an effective way of handling these data. Technologies such as MapReduce and Hadoop were considered for processing these large datasets. Secondly, the use of Yahoo! Pig to process huge amounts of RDF data was discussed. An overview of Yahoo! Pig for querying large volumes of information in a batch processing mode utilising clusters of several machines, without evident challenges in scalability was analysed. Throughout the examination it was seen that Yahoo! Pig's information model and change language are like the relational representations of RDF and the SPARQL query language. The authors stretched out Pig for processing RDF queries. A limitation of using the model was highlighted as it provides solutions for only the offline batch processing task. The author recommends that more algorithms be included into the MapReduce framework to address the issue of scalability. The scope of the research is also supported by Kim et al. (2010) with a proposition for e-portfolio designs based on a Private-Public data index system that integrates cloud computing applications and storage with semantic web architecture.

Similarly, Husain et al. (2011) addressed the issue of complex queries and scalability for large semantic web data. Leveraging cloud technologies, a scalable semantic framework was built to handle queries regarding RDF dataset which was becoming very large and complex. The reason for this work was because the authors stated that the existing solutions that has been provided, though they handle large RDF dataset, they are usually not scalable, or they do not scale adequately. The authors devised a novel algorithm to handle complex queries, that is queries with optional blocks which their previous work did not include and basic graph queries. Hadoop framework was also utilised to store the RDF data and MapReduce was used for the query answering system, although some algorithms were used for modification to achieve scalability. The system was tested using SP2B dataset and the desired result was achieved. The article did not dwell much on the performance of the queries when applied to larger datasets. However, the system was stated to possess capabilities for maintaining scalability and efficiency in such cases.

Likewise, Amato et al. (2012) proposed a system that leverages semantic web technologies for document composition such as editing or composing aiding services; exploiting hardware and software functionalities (service model) provided by the cloud service provider. The system can be applicable to various domains, but the testing of the system was streamlined to the health domain. The paper proposed CloSe, a cloud software as a service system for document semantic composition. CloSe depicts a development in the cloud computing domain for record handling and is dependent on semantic methodologies. The framework exploits data and information contained in suitable document bases, gathered from heterogeneous sources, for appropriate recommended fragments to be embedded in the document. The outcome demonstrated that the framework improves the archive structure; helping and providing preliminary results about the viability of the semantic recovery methods, in view of precision, recall and f-measure metrics. In any case, performance assessments of the framework were not revealed.

Furthermore, Hsu & Cheng (2015) proposed a cloud service model called Semantic Agent as a Service (SAaaS) which involves the integration of a semantic web and software agents as a typical approach to access cloud resources consistently. SAaaS was developed using UML but it was enhanced to use SAUML; Semantic-based Agent UML. The proposed model was associated with an existing cloud service to encourage the improvement of resourceful cloud computing applications. In line with this, the work by Dessi et al. (2016) proposed the use of semantic web technologies in relation to bioinformatics. The authors discussed the concerns of technologies been suitable for promoting the prerequisites of a cooperative environment where a research community share and develop information regarding the biomedicine discipline. The authors proposed COWB (Collaborative Workspaces in Biomedicine), a system which underpins collective knowledge management with regards to biomedical communities to address this issue. The framework was a cloud service model based on PaaS (Platform as a Service), displaying an elective method to knowledge management and utilising cloud platform to share information aggregately. It also enabled storage of knowledge bases across several machines and accessible to a wide range of users.

Corradi et al. (2016) also towed the Platform as a Service (PaaS) model, proposing a mobile cloud infrastructure for extracting semantic data from speech recognition within social care domains. The system proposed was MoSSCa, a mobile cloud empowered speech recognition system that can give semantic-enhanced text recognition, which is challenging on cell phones without a portable, supporting cloud architecture. The study did exhibit a system and an architecture, with a survey across various accents, queries and high levels of concurrent user requests. The architecture aided the processing and management of vast amounts of information in a Big Data environment (Herrera et al., 2017).

Across the different research efforts analysed, it can be observed that both cloud computing and semantic web technologies are required for efficiently handling large amount of heterogenous data that is currently available on the web. While developing a framework to ensure data is stored and retrieved with respect to the desired domain or challenge, scalability, efficiency and a high-performance rate are some of the vital requirements. The roles played by ontologies and natural language processing are also prominent, especially in querying data either via speech or text form. Table 2.4 presents a summary of the reviewed literature, their concerns, tools used to achieve those concerns with regards to cloud computing models and semantic web technologies as well as their scope during the analysis of the review.

Table 2. 4 - Summary of research projects on leveraging cloud computing for semantic-based applications

| **Authors** | **Domain** | **Cloud Model** | **Semantic Tools** | **Scope** |
|---|---|---|---|---|
| Mika (2008); Kim et al. (2010) | Web | PaaS (Hadoop, MapReduce) | RDF, SPARQL | Overview of technologies. |
| Husain et al. (2011) | Web | PaaS (Hadoop, MapReduce) | RDF, SPARQL | Did not handle complex queries that involves optional blocks |
| Husain et al. (2011) | Web | Hadoop, MapReduce | RDF, SPARQL | Performance was not evaluated with respect to complex queries that involved optional blocks |
| Amato et al. (2012) | E-Government and E-Health | SaaS, IaaS, PaaS | NLP, OWL, SPARQL | Processing semantic documents in the cloud |
| Hsu & Cheng (2015) | Web | SaaS, IaaS, PaaS | RDF, OWL, SWRL | SAUML profile only addressed SAaaS modelling. |
| Dessi et al. (2016) | Bioinformatics (Health) | PaaS | RDF, OWL | Considers only domain knowledge from ontology. |
| Corradi et al. (2016) | M-Health, Social Care | PaaS | NLP, OWL | Although performance and scalability were |

| | | | | |
|---|---|---|---|---|
| | | | | considered in the research, the issue of data security was not covered. |

## 2.6 Chapter Summary

This chapter provided a comprehensive review of literature for the technological paradigms central to this research; semantic web and cloud computing. With semantic annotation being very pivotal to the semantic web, it was critically reviewed alongside how ontologies impact on the varying types of challenges it faces today. Cloud computing delivery and deployment models; characteristics, benefits and issues were also reviewed. Furthermore, the role each of the two paradigms plays in facilitating the other were reviewed and analysed. For each of the sections, a gap analysis is conducted by elucidating the challenges therein. With focus on the automation challenge for semantic annotation, the next chapter starts with the proposal of requirements to address this holistically.

# Chapter 3: Towards a Holistic Semantic Annotation Solution

In this chapter, a holistic perspective to automated semantic annotation is proposed. This is based on the automation challenge of semantic annotation described in section 2.2 of Chapter 2 which analysed the state-of-the-art approaches for automatic semantic annotation. From the analysis, a disparate approach to semantic annotation was pointed out which has hindered automatic processes for semantic annotation. To address this, transformation from a disparate approach to a holistic one is proposed. The holistic perspective is believed to be capable of addressing the automation challenge. The case for a holistic perspective is presented in section 3.1 while section 3.2 identifies and analyses requirements for the holistic perspective. In section 3.3, a feasibility of cloud computing facilitating the holistic view is assessed by evaluating cloud computing mechanisms and their potential impacts on the requirements. These results in a cloud computing capability model for holistic semantic annotation in section 3.4 with a chapter summary in section 3.5.

## 3.1 The Holistic Perspective to Semantic Annotation

From the semantic annotation challenges presented and analysed in section 2.2 of Chapter 2 and the subsequent focus on the automatic semantic annotation challenge, several issues mitigating against it can be observed with existing semantic annotation tools from literature. Firstly, the scope and depth of ontologies that describe concepts and relations are vital factors which determine the level of precision obtainable with annotation data (Faria et al., 2014). While domain-specific ontologies can be very useful, they are still limited in most cases. The use of a single ontology for describing concepts and relations would result to a low level of accuracy and completeness for annotation data generated based on such ontologies. This can be observed with some existing semantic annotation tools such as SemTag (Dill et al., 2003) and KIM (Malik et al., 2010) that utilise a single ontology. With ontology engineering activities such as ontology mapping, merging and alignment, it is pertinent to aggregate resources from multiple ontologies for automatic semantic annotation processes in order to expand the scope and depth of concept/relations construction.

In addition, the storage mechanism for annotation data can be observed to vary widely across existing semantic annotation solutions. While some embed these within web documents; such as CREAM, OpenCalais and MnM, some others such as Cerno and KIM store them remotely (Oliveira & Rocha, 2013). The format for storing annotation data across both methods can also be observed to vary, with portable and well-structured formats such as XML and JSON as well as less structured formats such as HTML and XHTML among others. With focus on web documents which are mostly based on a client-server architecture over the Internet, annotation data stored remotely and served to corresponding web documents using portable formats would offer greater benefits. The remote storage mechanism also makes it easier to integrate the process with ontology engineering activities to facilitate a synchronisation between both. This is also in line with managing the evolvement of ontologies. Ontologies can evolve by adding new concepts and relations to them through ontology population (Petasis et al., 2011), or through an update to the structure of the ontology either by means of changes in the structure of the model or changes based on the development language; such as OWL evolving to OWL2 (Bayoudhi et al., 2017). The impact of such evolution needs to be managed alongside automatic semantic annotation of web documents.

Furthermore, when ontologies are mapped or aligned to expand their scope and depth, techniques adopted for obtaining optimal results from the process are vital, as well as having a continuous synchronisation between mappings generated and the ontologies that provided them (Mittra & Ali, 2017). These processes within ontologies; evolution, mapping, alignment, update among others directly impact on annotation data, hence, the need for a seamless communication between them. In addition, web documents are very dynamic and change often. With web documents hosted on web servers possibly remotely from their corresponding annotation data, a real time consistency between web documents and annotation data is required. Based on these; web, annotation as well as ontology servers need to establish a data communication stream continuously for up-to-date data sharing. Figure 3.1 illustrates the communication required between these.

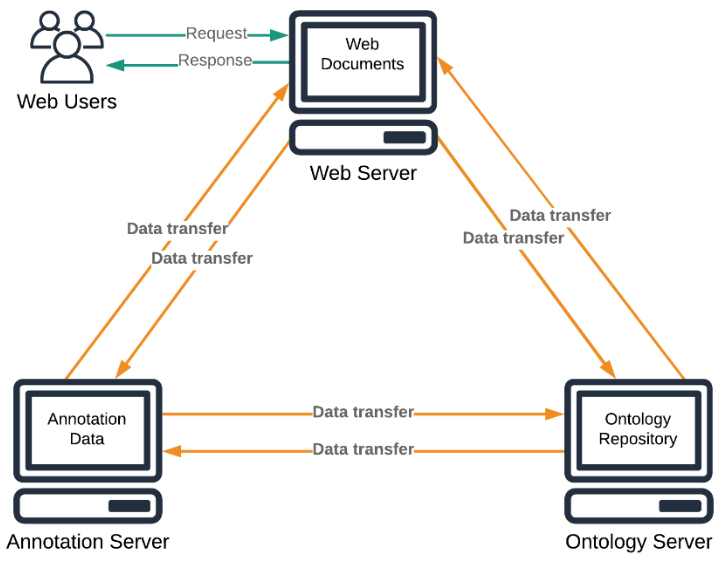


Figure 3. 1 – Communication Stream between Web, Annotation and Ontology Servers

Likewise, the optimisation of data at set intervals is a well-established concept in information systems (Wang et al., 2018). This applies to annotation data as well because schematic and structural changes to ontologies and annotation data formats over time has the potential to alter the structure of annotation data. An optimisation process that will trigger when necessary requires synchronisation with ontologies and other associated third-party computing resources. Currently, the optimisation of annotation data has not been observed to be a topic of discussion in literature. Another concept which has not been observed to receive attention from the state-of-the-art and which the holistic perspective can foster for automated semantic annotation is the co-location of annotation data and web documents. Lastly, while some tools are presented as providing automatic semantic annotation, the level of automation can be realised to be low; requiring expertise level involvement in the semantic annotation process for end users. This eventually constitutes a challenge for utilising such solutions. In this context, automation is opined as not requiring any expertise from end users for the semantic annotation process. Figure 3.2 illustrates transformation from a disparate state of semantic annotation to a holistic one; which provides a synchronisation of several independent solutions as well as considers additional functionalities required for a fully automated semantic annotation process.

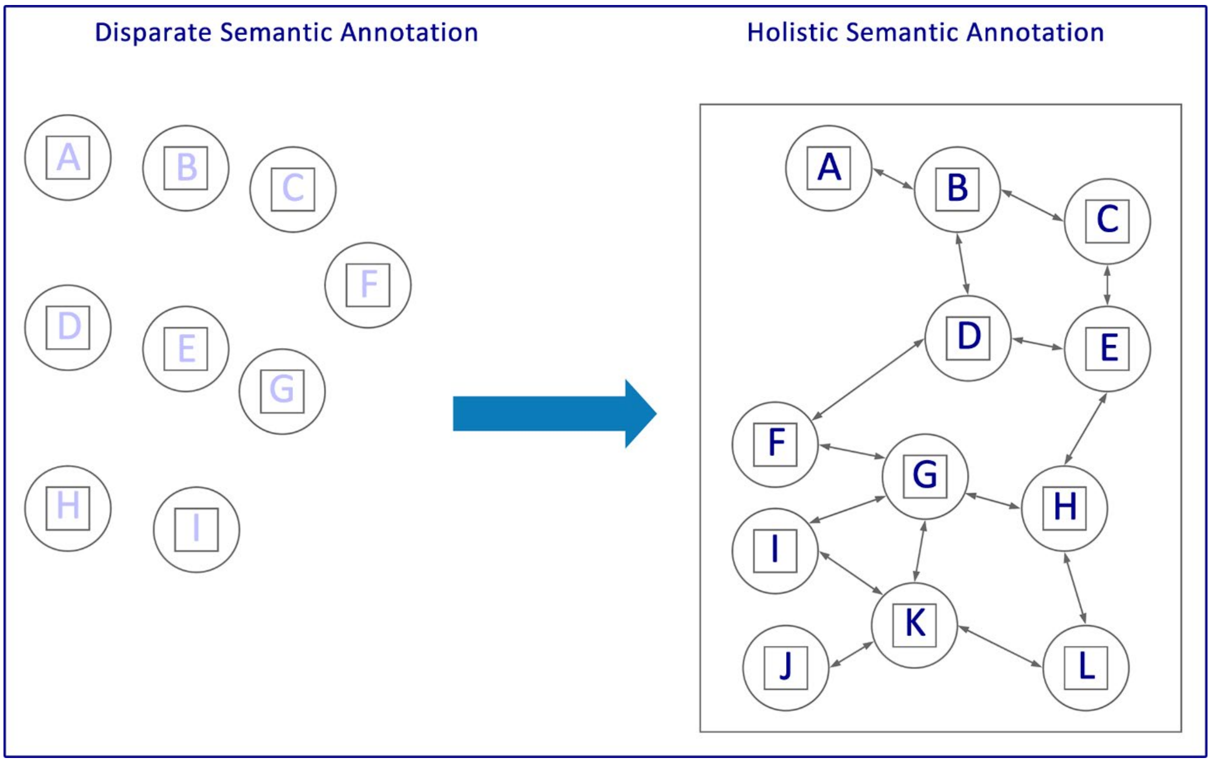


Figure 3. 2 – From Disparate to Holistic Perspective for Semantic Annotation

These can be observed to be contributing factors to the challenge of automatic semantic annotation and to the best of our knowledge, no existing semantic annotation tool provides a solution that addresses all these holistically. While some of these can be observed to be addressed disparately, a holistic perspective to semantic annotation, which investigates all of these for facilitating a fully automated semantic annotation solution is proposed in this research. A solution based on this perspective is believed to possess the potential of transforming the current state of semantic annotation by means of a holistic approach that investigates the various automation challenges and offers an integrated solution; capable of delivering the service as a single unit while also providing a continuous lifecycle of annotation data generation, delivery, management and evolution for web documents with the capability to scale and be deployed in multiple clusters across the web. The next section reviews and analyses identified requirements for a holistic semantic annotation process.

## 3.2 Requirements for the Holistic Perspective

From the holistic perspective to semantic annotation described in section 3.1, several requirements can be identified, which are also well supported in literature for semantic annotation generally. While the technique and extent of implementation for these vary across some existing tools, the understanding of what they are as requirements are quite generic. Furthermore, two additional requirements are added to the ones adapted from literature. These are Annotation Data Optimisation and Annotation Data Colocation.

Table 3. 1 – Holistic Semantic Annotation Requirements

| **Requirement** | **Summary Description** | **Support for Requirement** |
|---|---|---|
| Concept Extraction | Extraction, disambiguation and interlinking of instances of concepts and relations from text corpus and their storage in RDF graph databases | Dou et al., (2015), Grobe-Bolting et al., (2015), Martinez-Rodriguez et al., (2018) |
| Ontology Population | Addition of concepts and relation models into the structures of existing ontologies | Petasis et al. (2011), Faria et al. (2014), Tomaz et al. (2012), Cheatham et al. (2019), Makki (2017) |
| Ontology Selection | Evaluation and selection of appropriate ontologies for a semantic annotation process | Park et al. (2011), Hooi et al. (2015), Sabou et al. (2006), Dhingra & Bhatia (2012) |
| Ontology Mapping | Mapping same or similar concepts across multiple ontologies together | Luczak-Rosch et al. (2014), Johnson et al. (2012), Xiangmei & Chunli (2013), Kumar & Harding (2013), Mittra & Ali (2017), Wang et al. (2015), Jean-Mary et al. (2009) |
| Annotation Data Storage | Generation and storage of annotation data based on a decoupled approach | Uren et al. (2006), Zou & Ozsu (2017), Jie et al. (2018), De Virgilio (2017) |
| Annotation On-the-fly | Online, real time and automated generation, storage and annotation of web documents with contextual data | Based on traditional client-server architecture in computing (Mainetti et al., 2015) |
| Annotation Data Reuse | Use of annotation data instance multiple times for a web document | Uren et al. (2006), Zou & Ozsu (2017), Jie et al. (2018) |
| Annotation Data Sharing | Use of annotation data instance by multiple web documents | Uren et al. (2006), Oliveira & Rocha (2013) |
| Annotation Data Auto-Update | Automatically updating annotation data to maintain consistency with corresponding web documents | Uren et al. (2006), Oliveira & Rocha (2013) |
| Ontology Auto-Update | Automatically updating ontologies to ensure accuracy of annotation data | Sassi et al. (2016), Imam (2016), Losch et al. (2009), Sangers et al. (2012), Flahive et al. (2015) |

| | | |
|---|---|---|
| Annotation Data Optimisation | Periodic optimisation of annotation data based on schematic changes to ontologies. | Novel; based on the computing concept of data optimisation (Wang et al., 2018) |
| Annotation Data Colocation | Locating web documents and annotation data close to each other to minimise data transfer issues such as network latency | Novel; based on the computing concept of co-location of resources (Wilder, 2012) |

Figure 3.3 also presents the holistic semantic annotation requirements based on a classification of three phases for the semantic annotation process; preparatory, annotation and maintenance phases.

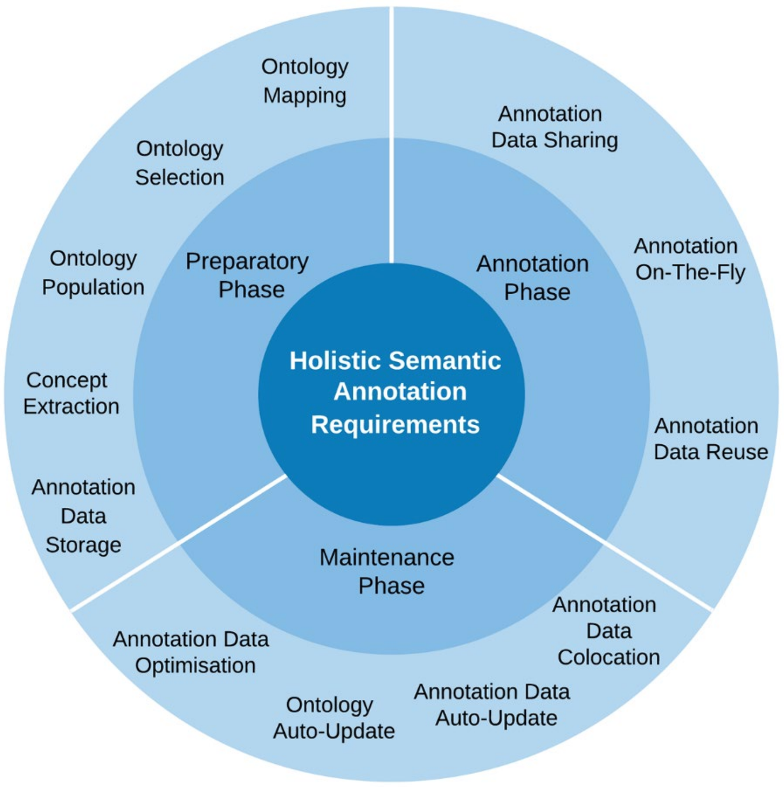


Figure 3. 3 – Holistic Semantic Annotation Requirements and Phases

### 3.2.1 Preparatory Phase Requirements

This phase comprises of requirements that defines series of processes needed prior to a semantic annotation instance. These are: Concept Extraction, Ontology Population, Ontology Selection, Ontology Mapping and Annotation Data Storage.

### 3.2.1.1 Concept Extraction

Concept Extraction refers to the process of extracting instances of concepts and relations from different sources to develop annotation data for web documents (Dou et al., 2015). While this process can be implemented in diverse ways and using different techniques, a generalised procedure, which defines the major steps can be utilised. Figure 3.4 presents the general approach to concept extraction for semantic annotation.

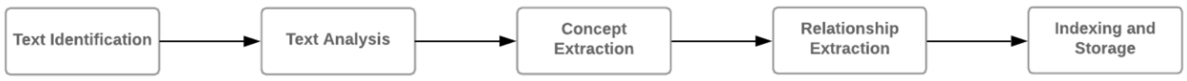


Figure 3. 4 – Generic Approach to Concept Extraction for Semantic Annotation

From Figure 3.4, it can be observed that the first step is Text Identification. This requires identifying documents containing instances of concepts and relations for extraction. The source of the document needs to be identified and both its availability and accessibility confirmed. Once the document has been identified and accessed, the extraction of text follows using scientific techniques such as web scrapping, TextRunner and KnowitAll (Niklaus et al., 2018). This is known as Information Extraction and Retrieval (IER) (Vlahovic, 2011). Several methods of IER exist today but they can be broadly categorised into pattern-based and machine learning-based methods. The pattern-based methods are dependent on specific patterns from data or rules defined and include Hearst Pattern, JAPE (Java Annotation Pattern Engine), Pattern Discovery, etc. The machine learning-based are either dependent on probabilistic or induction theories. Examples are Hidden Markov's Model, LP2 (Lifted Probabilistic Logic Programming) and N-Gram Analysis (Reeve & Han, 2005).

The next phase; Text Analysis involves the use of specific algorithms for splitting sentences and the identification of different concepts such as people, things, places, organisations and events. This relies on the use of Natural Language Processing (NLP). NLP describes the process of utilising computerised techniques to read, decipher and analyse natural languages, such as English and French to come up with meaningful information (Young et al., 2018). This analysis involves the identification of concepts and relationships from natural text. Some NLP techniques include Automatic Summarisation, Co-Reference Resolution, Discourse Analysis, Machine Translation, Morphological Segmentation, Named Entity Recognition, Optical Character Recognition and Part-of-Speech Tagging (Khurana et al. 2017). The role of Automatic Summarisation is the production of a comprehensive shorter form of a larger piece of text such that the overall context of the initial text is still intact. Co-reference Resolution has to do with the identification of different objects within a piece of text that refers to a specific named entity within the text. Such references include the use of pronouns such as ‘he’, ‘she’, ‘it’, ‘I’ and ‘me’ in reference to the named entity. The function of Discourse Analysis is the identification of the communication structure for a set of inter-related text. Machine Translation automatically translates text from one natural language to another one. Morphological Segmentation deals with the separation of words into separate contextual morphemes and identification of the types of morphemes. As for Optical Character Recognition (OCR), it offers an image that represents printed text, thereby helping to determine associated text for the corresponding images. Part-of-speech Tagging offers sentence description and the determination of the part of speech for every word (Khurana et al. 2017).

A typical NLP system helps with the manipulation of an input text in a progressively complex manner. NLP deals with words and considers words as carriers of textual meanings. Hence, it requires a pre-processing step before additional analysis for delimiting individual word tokens making up a text, which is known as Tokenisation. Tokens (words) are deciphered according to the context of their use in a pre-trained classification model that studies parts of speech (Singh, 2019). A simple approach is the splitting of the text based on whitespaces or punctuations. Similarly, Sentence Splitting in most cases has to do with utilising basic heuristics such as searching for the typical end of sentence punctuation (period or question mark) followed by a capital letter (Verspoor & Cohen 2013).

It is sometimes essential to establish the relevant relations among words in a text. Part-of-Speech Tagging is the most fundamental in determining the part of speech of a word. Shallow parsing analyses sentences by identifying and recognising words, or a sequence of words as belonging to a part of speech. However, deep parsing is required to identify specific grammatical relationships among them and their roles within the sentences, as shallow parsing does not do this. (Verspoor & Cohen 2013). Named Entity Recognition (NER) is used in NLP for locating and classifying named entities in word tokens into categories predefined in NLP models (Marrero & Urbano, 2018). These categories include names of people, places, organisations, monetary values and time expressions; utilised in solving real-world problems such as specific products mentioned in a certain review or names and location of people mentioned in documents.

From the classification, the next step involves the disambiguation of recognised concepts (that is, definition as people, organisations, places, events and so on) based on one or more domain-specific ontologies. For example, 'Leopard' is classified as an animal and further disambiguated as 'Leopard: Cat' not 'Leopard: Pisces'. This step is very crucial as it gives text the ability to be processed and become understandable pieces of data through linkage to a broader set of already existing data. Next, the Relationship Extraction step, which identifies relationships between extracted concepts and links them with related external or internal domain knowledge (Martinez-Rodriguez et al., 2018). The fifth step; Indexing and Storage deals with indexing and storing the instances in a semantic graph database, usually in RDF format. At this stage, the data within the graph database can be queried using SPARQL to generate contextual data for a web document based on the web document content.

#### 3.2.1.2 Ontology Population

Ontology Population refers to the population of domain-specific ontologies with new concepts and relations as additions to the existing knowledge modelled within such ontologies. It is also sometimes referred to as Ontology Enrichment (Petasis et al., 2011). In comparison with Concept Extraction which involves the extraction of instances of concepts and relations, Ontology Population provides a representation for concepts and relations and not their actual realisations as utilised in text corpus. Invariably, Concept Extraction utilises the structure defined for concepts and relations within an ontology to define context and relationships for instances of such concepts. The addition of more concepts and relations to ontologies implies increasing the scope of contextual data described by the ontology which subsequently results into the generation of more meaningful annotation data for web documents (Tomaz et al., 2012).

Furthermore, with the dynamic and transformational nature of information resulting to new data being added to the body of knowledge daily, the need for ontology population is very vital to cater for newly constructed or re-defined concepts and relations in order to avoid mis-representation of information within annotation data and facilitate their accuracy at all times. Ontology Population processes result into structural changes for the ontology as it can require modifications to the hierarchy of concepts or to taxonomic relations (Cheatham et al., 2019). A typical ontology population process usually requires a mechanism, often regarded as an extraction engine to identify concepts and relations from several document types (Faria et al., 2014). An initial ontology is then input into the system alongside the newly extracted concepts and relations to construct a taxonomy from the existing concepts within the initial ontology as well as the newly extracted ones. Thereafter, semantic relations are extracted and defined for the new taxonomy. These results into an enriched ontology which undergoes an evaluation to resolve any inconsistencies. Tomaz et al. (2012) proposed an iterative, unsupervised approach for this process. Upon inconsistencies resolution, a populated ontology emerges. The overall process can be repeated for several text corpus and either based on specified intervals with new document sources or based on the availability of a required level of new concept/relation pairs within the domain of an ontology. Figure 3.5 presents an ontology population process as defined by Petasis et al. (2011).

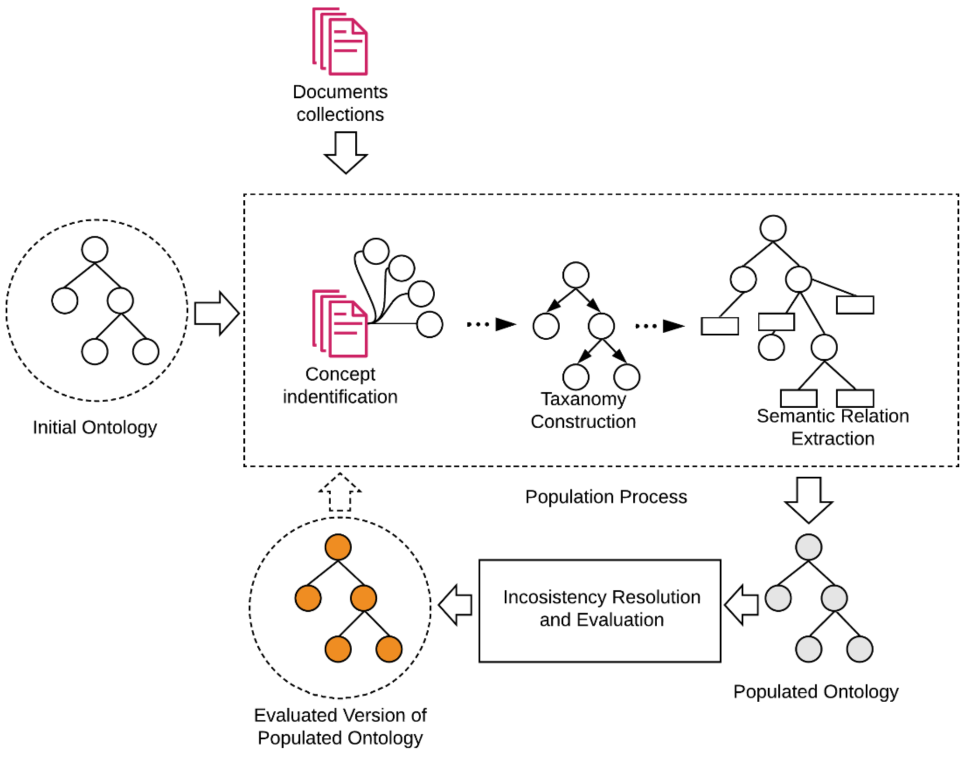


Figure 3. 5 – Ontology Population Process (Petasis et al., 2011)

### 3.2.1.3 Ontology Selection

Ontology Selection defines the process of identifying ontologies that meet a set of criteria for a specific purpose (Park et al, 2011). The selection of an appropriate ontology or multiple ontologies for semantic annotation is very vital in determining the quality of annotation data produced. This is because selecting appropriate, multiple ontologies to generate RDF annotation data for a web document provides a better means of covering the required scope and depth of contextual data required by the web document. Often, selection criteria for ontologies are dependent on the web documents that the ontologies are to be used for. The domain, scope, size, depth and standardisation for ontologies are all key factors in a selection process (Hooi et al., 2015).

The selection process also constitutes an evaluation for the ontologies by means of considering if they meet the required criteria for selection. Once the evaluation metrics are met by one or more ontologies, they can then be selected for use. Sabou et al. (2006) defined an approach to ontology selection which involves three criteria; popularity, richness of content and scope. The popularity of ontologies defines how high they rank; using ontology ranking algorithms, in comparison with other similar ones. AktiveRank (Dhingra & Bhatia, 2012) is one of such algorithms popularly used for the ranking process. It defines a five-stage algorithmic approach to calculate a popularity index for ontologies by analysing the structure of concepts defined within them. The stages are class match measure, centrality measure, density measure, semantic similarity measure and total score. The Ontology Scope criteria defines knowledge domains that the ontology will be representing. The granularity of the ontology is also required to be identified, as this impacts on its intended use. A list of competency questions can be developed to provide a well-defined scope for the ontology. Other measures such as number of concepts and relations contained within the ontology are also means of evaluating and selecting appropriate ontologies which can be utilised in other ontology engineering processes such as ontology mapping and ontology alignment among others (Petasis et al., 2011).

#### 3.2.1.4 Ontology Mapping

Ontology Mapping refers to the matching and alignment process for concepts and relations across multiple ontologies with same or similar contextual information (Luczak-Rosch et al, 2014). For the holistic semantic annotation process being proposed, ontology mapping is required sequel to ontology selection. The goal of the mapping process is to address issues such as semantic ambiguities and redundancies (Johnson et al., 2012) which, if well addressed would lead to the generation of better and rich-content annotation data for web documents by fostering a means of aggregating resources from multiple ontologies at the same time without conflicts in their schema for concepts and relations. Mapping ontologies together makes it possible to update the different ontologies independent of each other and add more ontologies as the need arises without having to undergo any major overhaul (Xiangmei & Chunli, 2013). Several factors can be identified which leads to a single data or concept being represented in different ways within different ontologies. These include:

- The use of a single term to describe different concepts – ambiguity.

- The use of different terms to describe a single concept – redundancy.
- The use of different types of representation for describing concepts within ontologies.
- Developing ontologies to different levels of depth in terms of scope.
- Developing domain-specific ontologies from different perspectives.

While several approaches exist to utilising multiple ontologies for a web document, ontology mapping is very crucial as it enables the individual ontologies to continue evolving independently irrespective of earlier defined mappings, with updates to the mappings when one or more ontologies evolve (He et al., 2010). Ontology Merging is one of such other approaches. However, ontology merging results into an integration between two or more ontologies to constitute a single unit. Table 3.2 details the differences between both.

Table 3. 2 – Comparison of Ontology Mapping and Ontology Merging

| **Criteria** | **Ontology Merging** | **Ontology Mapping** |
|---|---|---|
| Language | The ontologies need to have been developed using same language | Does not require ontologies to be of the same language before their concepts can be mapped |
| Upgrade | This is cumbersome, as each initial ontology still evolves independently | Ontologies are upgraded separately, hence, less cumbersome |
| Purpose | Becomes restricted to uses permissible by the merging | This is flexible as individual ontologies can still be utilised separately |
| Maintenance | It becomes more cumbersome, due to the volume as well as differences in maintenance techniques | Relatively less cumbersome due to their independence |

The current approaches to ontology mapping deploy a wide range of methods which was classified by Kumar & Harding (2013) as follows:

- Linguistic Methods: These exploit linguistic labels of concepts within ontologies to be mapped. Similarities between labels are identified using techniques such as Hamming Distance or some specialised domain knowledge. Once these similarities have been identified, the concepts represented by the labels can be mapped together as representing a single meaning.
- Statistical Methods: These define mappings between ontological concepts based on the existence of statistical correlations between them. Such methods, however, are heavily

reliant on the availability of large numbers of instances within RDF graphs utilising such ontologies.

- Structural Methods: These utilise the internal structures of ontologies to identify similarities between different ontologies. Structural methods however cannot be used alone and is usually alongside either a linguistic or statistical method.
- Logical Methods: These utilise the logical formalisms within the ontology. However, ontologies with a low level of semantic structure would not provide a great deal of formalism. Hence, this can be utilised alongside either a linguistic or statistical method as well.

Ontology mapping frameworks such as ASMOV (Automated Semantic Mapping of Ontologies with Validation), FOAM (Framework for Ontology Alignment and Mapping) and QOM (Quick Ontology Mapping) are commonly used. There is the need however, to ensure that updates are made to mappings after upgrading an ontology to a newer version (Adachi & Fukuta, 2017). Mappings can be between local, merged and remote ontologies (Wang et al., 2015). The ASMOV ontology framework also provides a semantic verification step for mappings generated as a means of validation (Jean-Mary et al., 2009; Mittra & Ali, 2017). It uses lexical and structural characteristics of ontologies for an iterative calculation of a similarity measure index between them to generate an alignment. These alignments go through a verification process to ensure that there are no semantic inconsistencies. ASMOV exploits four major characteristics of an ontology to match pairs of entities. These are the lexical information, internal structure, external structure and individuals (Jean-Mary et al., 2009; Mittra & Ali, 2017).

#### 3.2.1.5 Annotation Data Storage

Annotation data storage refers to the storage mechanism of contextual data for annotating a web document. These can be domiciled within the web document which is classified as a monolithic approach or they can be stored separately from the web document, known as a decoupled approach (Uren et al., 2006). For web-scale, holistic semantic annotation, the decoupled approach is believed to be more efficient and effective for their intended purposes. Table 3.3 presents a comparison of these two approaches.

Table 3. 3 - Comparison of Monolithic and Decoupled Annotation Data Storage Mechanisms

| Monolithic Annotation Data Storage | Decoupled Annotation Data Storage |
|---|---|
| Changes in a web document automatically makes the annotation data invalid. | Changes in a web document automatically triggers a re-generation of annotation data for the document. |
| Annotation data is not sharable among multiple web documents. | Annotation data can be shared by multiple web documents |
| Annotation data is usable only by the web document it is embedded in. | Annotation data is re-usable by multiple web documents |
| It does not foster collaboration | It promotes collaboration |
| Requires a 1:1 mapping for web documents and annotation data | Fosters a 1: N mapping for web documents and annotation data |
| Annotation data is always available with the web document | Annotation data and web document exist separately |
| Annotation data can be edited locally, which can lead to errors | Annotation data is not locally available for editing; hence its integrity is preserved. |

With the dynamic nature of the web, web documents and ontologies are frequently evolving and requiring updates. The decoupled approach will facilitate automatic updating of annotation data; triggered when either corresponding web documents or ontologies change as a result of an update to content, context or structure. Considering the structure of annotation data which can be in portable formats such as XML, JSON or YAML, changes within a web document would require a re-generation of its annotation data which needs to be an automated process without a user's direct access to the annotation data. However, with the monolithic approach, annotation data is exposed to users, hence the chances of changes to it by non-experts which will subsequently render the web document semantic annotation invalid. A scheduled server script (such as cron jobs on Linux kernels) can be set up to monitor web documents evolution and initiate annotation data re-generation when such documents evolve.

### 3.2.2 Annotation Phase Requirements

This phase involves the actual requirements and different scenarios that would result into the semantic annotation of a web document. It requires that annotation data has already been generated and stored for the corresponding document. The requirements involved in this phase are: Annotation On-The-Fly, Annotation Data Reuse and Annotation Data Sharing.

#### 3.2.2.1 Annotation On-the-fly

Annotation On-the-fly refers to the ability of a holistic semantic annotation process to provide online, real time and automated semantic annotation for web documents. Upon the receipt of a request by a web server for a web document, annotation data can be generated for the web document content by running SPARQL queries on a semantic graph database for contextual data regarding the web document. The generated data; which is annotation data is then stored as well as referenced by the web document instantaneously. This is based on a typical client-server architecture for web applications on the Internet. A service-based model which web documents can subscribe to for such a level of automation is not known of to the best of the writer's knowledge and is believed to constitute a major stride towards the successful evolvement of the semantic web. Such a service would require a high level of automation to enable non-experts utilise it; possibly by means of an API (Application Programming Interface) call within web documents.

Furthermore, considering the wide-scale use of web content management systems and frameworks which can utilise a single header file for multiple web documents (hundreds or even thousands), the API call would only be needed once within a header file and that would facilitate semantic annotation for all the web documents. Furthermore, considering this requirement alongside Annotation Data Storage; web content editors, web administrators and several other categories of non-technical web content managers can have full and free access to web content without any chances of altering annotation data content or structure.

#### 3.2.2.2 Annotation Data Reuse

The adoption of a decoupled approach to annotation data storage means it can be re-used by one or more web documents. Annotation data remains valid if the web document content remains the same. Hence, the same instance of annotation data can be served to a web document multiple times without requiring a re-generation of the data if the web document has not changed. This provides a means of optimising computing resources usage, especially considering that some web documents do not change frequently. A mechanism for facilitating the requirement would involve mapping web document URLs to annotation data IDs within a database such that once a web server receives request for a web document, a record in the database table assigning annotation data to the web document is checked for. The assigned annotation data would then be reused for the document if such a record exist. Otherwise, annotation data is generated for the document real time.

#### 3.2.2.3 Annotation Data Sharing

Annotation data generated for a web document and stored on an annotation server can be shared with other similar web documents of the same domain and containing similar content. This will foster optimisation for the usage of computing resources as it means the same annotation data is not replicated across a server multiple times for different web documents, thereby consuming more storage and requiring increased processing power to access them. Furthermore, an update to the annotation data is immediately implemented for all web documents referencing it, providing a cascading effect over multiple web documents all at once. Access to an instance of annotation data for sharing can be managed based on consumer or document attributes using an attribute store for entities. Entities in the context will be annotation data, web documents and users. With Attribute-Based Access Control (ABAC), annotation data attributes are signed to security tokens which matches them with document or consumer attributes to either grant or deny access to an instance of annotation data (Talukdar et al., 2017). However, this is still based on the validity of a web document utilising an instance of annotation data previously generated for another document.

### 3.2.3 Maintenance Phase Requirements

The requirements in this phase are based on ensuring a consistent and continuous semantic annotation lifecycle for web documents. This includes by scheduling and running processes that foster a holistic process for synchronising different tasks together towards service delivery. The requirements in this phase are: Annotation Data Auto-Update, Ontology Auto-Update, Annotation Data Optimisation and Annotation Data Colocation.

#### 3.2.3.1 Annotation Data Auto-Update

The dynamicity of annotation data is very crucial due to the ever-changing nature of web documents (Oliveira & Rocha, 2013). Annotation data once generated, would require a re-generation whenever either the web document or domain-specific ontologies generating the annotation data is updated. This is to ensure that consistency is maintained within the three components; web documents, annotation data and domain-specific ontologies. Updates to the annotation data are implemented by queries sent to both the web server (containing web documents) and ontology server (containing the ontologies) from the annotation server. The response from either of the servers determine if an update is required for the annotation data. Annotation data can be stored using XML or JSON which are portable and interoperable formats usable on the web for storage and data transfer across different platforms (Petasis et al., 2011). With the decoupled approach to annotation data storage, which stores annotation data separately from web documents, it is pertinent to ensure that an up-to-date annotation data is always served to web documents, hence this process focuses on automatically updating annotation data when required. Scheduled tasks can be set to iteratively select web document URLs from a database and read their contents. If there have been any chances to a web document content from the previous running instance of the scheduled job, then annotation data for the specific document would have to be regenerated. The regeneration would require invoking some other processes; Ontology Selection, Ontology Mapping and Annotation Data Storage.

#### 3.2.3.2 Ontology Auto-Update

Ontologies usually require updates or upgrades based on changes within concepts and relations modelled within them (Sassi et al., 2016). These can be at a low level, such as adding domain-specific sub-concepts or at a high level, such as adding middle or upper level concepts to cover new areas within the domain (Imam, 2016). Updating ontologies based on such changes maintains consistency between them, data from ontology mapping processes and annotation data. Maintaining this consistency also implies that annotation data for web documents remain accurate. An automated means of updating or upgrading ontologies stored in ontology servers is crucial for holistic semantic annotation. Update to an ontology initiates a re-mapping of two or more ontologies and subsequently, the re-generation of annotation data for one or more web documents.

Some research literatures propose techniques for automatically updating ontologies (Flahive et al., 2015; Sassi et al., 2016; Losch et al., 2009). The work by Losch et al. (2009) which proposed an event-triggered Ontology Update Language (OUL) to eliminate manual ontology updating through the provision of a means of defining sets of SPARQL update rules. OUL is based on SQL-triggers in database management systems. It carries out the process of updating a list of ontologies with the aid of an Event-Condition-Action model as triggered by event occurrences through what is known as change handlers. Furthermore, Sangers et al. (2012) adapted the mechanism to propose OULx which provides prefixes and negation language features as well as various update execution processes with a prototype implementation.

#### 3.2.3.3 Annotation Data Optimisation

As the structure of ontologies evolve over time, structural changes might be required for corresponding annotation data. With annotation data stored in formats such as XML, JSON or

YAML which are highly structured document types, optimising the structure of annotation data in order to keep the integrity of the contextual information it stores is perceived as a needed requirement. While Annotation Data Auto-Update is about updating annotation data itself, Annotation Data Optimisation is about optimising the structure of annotation data documents, which invariably may result in minor changes to the actual contextual data stored in it. This will also constitute a form of maintaining consistency between ontologies and annotation data by eliminating any disparities between them, which is very vital for the accuracy of contextual information within annotation data files. The optimisation process would need to be automated and scheduled to be run for annotation data when its supporting ontology or one of its multiple supporting ontologies has gone through a structural change due to an upgrade or evolvement.

#### 3.2.3.4 Annotation Data Colocation

With the decoupled approach to annotation data storage described in section 3.1.1.5 which is based on storing annotation data separately from web documents, the co-location of both becomes a factor of interest. Annotation Data Colocation defines a mechanism in which annotation data is stored as closely as possible to the corresponding web document. This is based on the technological concept of colocation which in simple terms means placing resources together, or close to each other for one or more reasons (Wilder, 2012). These reasons include to foster faster communication and reduce network latencies as communication between system nodes is faster when the nodes are closer to each other than when the nodes are away from each other (Saeed et al., 2015).

While the location of web documents is dependent on the document owners or authors, with a global presence for most application hosting providers, annotation data can be stored in the same geographical zone as its corresponding web document and close to it as much as possible. Alongside the benefit for web users and web document owners, it will also constitute a huge benefit for hosting solution providers by greatly minimising computing overhead for semantically annotating web documents globally. Furthermore, in cases whereby web document location can be influenced by hosting solution providers, demand and usage statistics for such documents becomes more important as these can be utilised to select the most appropriate location for web documents and their associated annotation data based on proximity to the target audience.

## 3.2.4 Holistic Requirements Summary

The holistic perspective described earlier implies the need for a distributed system that can facilitate it. With the different requirements needed to be implemented and demanding a high level of coherence between them to provide a continuous semantic annotation service, there is the need for a computing platform or paradigm which will be well suited for this purpose. From the requirements analysed earlier, it can be observed that automation is key to establishing and maintaining a coherent workflow between them. With automation being very central to cloud computing, this research reviews and analyses the feasibility of leveraging cloud computing to deliver holistic semantic annotation. This is illustrated with a high-level conceptual diagram in Figure 3.6.

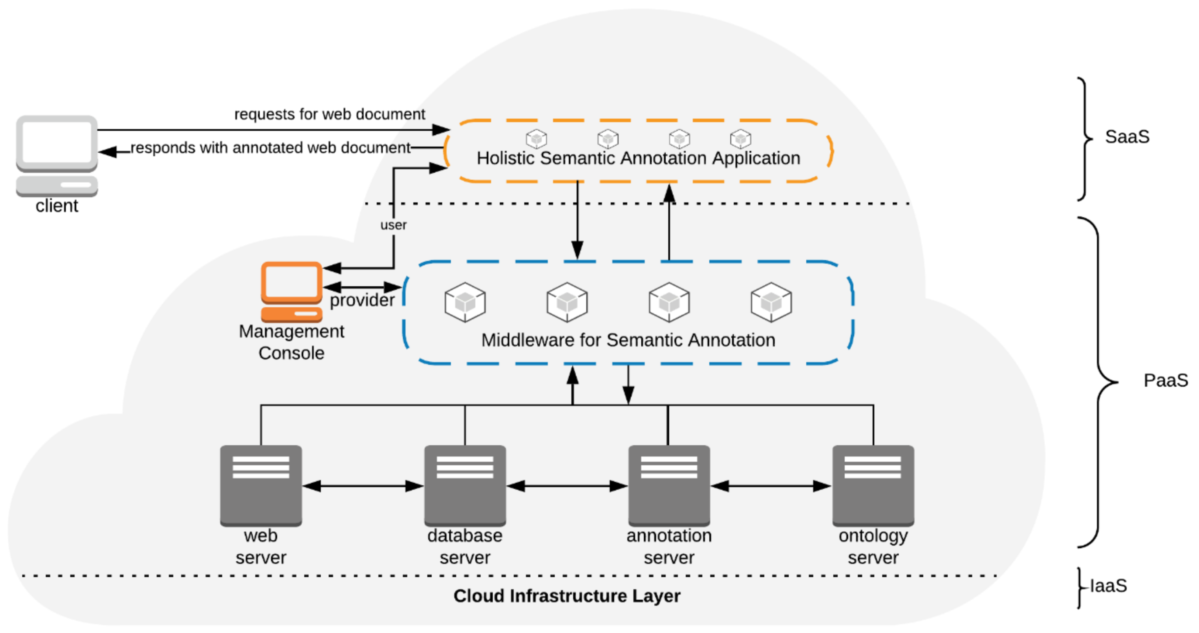

requests for web document
responds with annotated web document
client
Holistic Semantic Annotation Application
SaaS
user
provider
Management Console
Middleware for Semantic Annotation
PaaS
web server
database server
annotation server
ontology server
Cloud Infrastructure Layer
IaaS

Figure 3. 6 – Leveraging Cloud Computing for Holistic Semantic Annotation (High Level)

Furthermore, web documents and data are generally being migrated to cloud platforms due to the benefits inherent in adopting cloud computing such as: better insights and visibility, collaboration, supporting diverse business needs, allowing for rapid development and provisioning of new products and services through automated systems (Namasudra et al., 2017). The need for very high level of computer processing power for data storage, processing and management is also a factor that supports cloud computing adoption for holistic semantic annotation. With the vast amount of data on the web and its ever-increasing nature, coupled with the generation of equally large amounts of annotation data, high performance computing would be required to effectively store, process and manage the entire data and their lifecycle.

Cloud Computing offers this level of high performance and can be leveraged for the same purpose (Husain et al., 2011).

Likewise, the need for delivering semantic annotation holistically: as a service (SaaS), via a platform (PaaS), on an infrastructure (IaaS) for web documents is crucial. These are models which cloud computing offers (Mell & Grance, 2011). In addition, the need to automate the processes of: deploying (rapid provisioning), scaling (dynamic scalability) and monitoring (usage monitoring) the overall process to meet, maintain and manage the web-scale need and demand for semantic annotation positions cloud computing in good stead for adoption (Rodriguez & Buyya, 2019). Lastly, the need for standardisation which: provides a common language and foundation, fosters collaboration and best-of-breed solutions, facilitates a simpler development and deployment experience which is required for the semantic web, can be facilitated by cloud computing (Erl et al., 2015). Sections 3.3 and 3.4 focus on an assessment of cloud computing paradigm and its potential adoption for facilitating a holistic semantic annotation process.

## 3.3 Cloud Computing Mechanisms for Holistic Semantic Annotation

Since its inception, cloud computing has grown tremendously, and indications suggest that its growth trajectory will continue increasing (Namasudra et al., 2017). The growth has been motivated by factors such as adopting new business models and realising economies of scale due to the several cloud computing benefits. The fundamental building blocks of a cloud environment are based on its nature as a paradigm; characterised by computing technology concepts which make up cloud computing essential characteristics and more broadly, cloud computing mechanisms (Chiregi & Navimipour, 2017). Cloud computing mechanisms form primary artefacts that, in turn, constitute the fundamental cloud technology architecture (Erl et al., 2013). These mechanisms facilitate the design of various cloud applications that are reliable, scalable and secure in nature, ensuring that cloud consumers can trust the services offered by cloud providers. In general, cloud computing uses different approaches to achieve the same cloud services. The implementation of cloud computing mechanisms does not only standardise proven practices and solutions, it creates a common foundation on which higher-level systems can be built and enhances the development of standard software libraries and frameworks (Arcitura Education, 2018). Erl et al. (2013), proposed a classification for cloud computing mechanisms based on their characteristics and the technical solutions they offer.

This research has adopted the classification for providing an analysis of the roles and responsibilities for cloud computing mechanisms mapped with the holistic semantic annotation requirements for the semantic web. The classification helps define a scope and context for each of the cloud computing mechanisms with respect to facilitating holistic semantic annotation. The classifications; cloud infrastructure, cloud management, cloud security and specialised cloud mechanisms are analysed in this section.

### 3.3.1 Cloud Infrastructure Mechanisms

The first classification is for cloud infrastructure mechanisms which, considering a cloud environment as an infrastructure, constitutes IT solutions on which the infrastructure is built. These include mechanisms for resource clustering, resource replication, failover systems and geotagging which are analysed and evaluated in terms of their potential impact on holistic semantic annotation.

#### 3.3.1.1 Resource Clustering

Resource clustering refers to grouping multiple instances of cloud computing resources together for operation as a single entity (Erl et al., 2013). This is often required based on the geographical diversity of resources within a cloud infrastructure. The diversity requires a logical combination of resources into groups for the improvement of their allocation and usage which leads to an increased total computational capacity and availability of the clustered resources (Cui et al., 2016). The architecture on which this mechanism works requires real-time communication between nodes for an effective synchronisation of computing systems. This is usually orchestrated by a cluster management service which monitors and manages resources across geographical zones and aggregates them into a logical unit when required (Erl et al., 2013).

As such, Cao et al. (2016) proposed an innovative service model called Cluster as a Service (ClaaS). The model is said to suit the needs of medium-sized data centres with the aim of virtualising cluster environments for distributed application frameworks. A prototype of ClaaS called Docklet was implemented using lightweight containers to ascertain its feasibility. For a web scale holistic semantic annotation service, the need for clustering cloud resources; infrastructure, platform and service across geographical zones to meet on-demand requests which can greatly fluctuate is perceived as vital to delivering such a service. This can include for aggregating RDF graph storage across multiple geographical zones to facilitate annotation data generation.

#### 3.3.1.2 Resource Replication

Resource replication refers to creating multiple instances of a computing resource, in this context, within a cloud environment. Software agents responsible for this depend on virtualisation technologies such as hypervisor-level and operating system-level virtualisation for its implementation and usually results in automatically scaling the computing resources being replicated (Shahapure & Jayarekha, 2015). It is also important to note that these can be hardware or software resources. Replicating resources not only enhances their availability, it improves their reliability and consistency as multiple instances are created within same or different environments. Furthermore, it can foster shared bandwidth for software resources, thereby lowering associated access costs and decreasing delay time (Makhsous et al., 2016). Furthermore, it ensures that users have seamless and transparent access to computing resources even in the event of issues such as cyber-attacks or system failure.

For hypervisor-level virtualisation, a hypervisor can access the virtual image of a server and use this to create multiple instances. This includes replicating and deploying ready cloud environments based on technologies such as “infrastructure-as-code” to run one or more applications, allowing cloud users to access different instances of same physical resource in real-time (Shahapure & Jayarekha, 2015). Unlike on-site infrastructure where an organisation only has access to the available resources, resource replication works with resource clustering to provide cloud users with the ability to abstract data and IT resources with various types of hardware across multiple sites, thereby increasing productivity rates.

#### 3.3.1.3 Failover System

Failover Systems help to ensure that cloud resources are reliable and available with the aid of established clustering technologies for providing redundant instances of infrastructures (Mohammed et al., 2017). A failover system is programmed to automatically switch over to a redundant or standby cloud resource instance when there is a system failure and the currently active resource is no longer available (Erl et al., 2013). A typical failover system is generally utilised for mission-critical applications or ones requiring very high levels of availability. They also work across different geographical regions in such a way that every location hosts one or more redundant instances of the same resource (Mohammed et al., 2017). Furthermore, it is dependent on resource replication for provisioning redundant resource instances with active monitoring to detect errors and downtimes which will initiate switching over to a redundant instance. Failover Systems ensure a continuous service delivery and a requirement for high-end applications in the cloud such as a semantic annotation solution for web documents which can be required to serve up to millions of client semantic annotation requests daily. Figure 3.7 illustrates a failover system in the cloud.

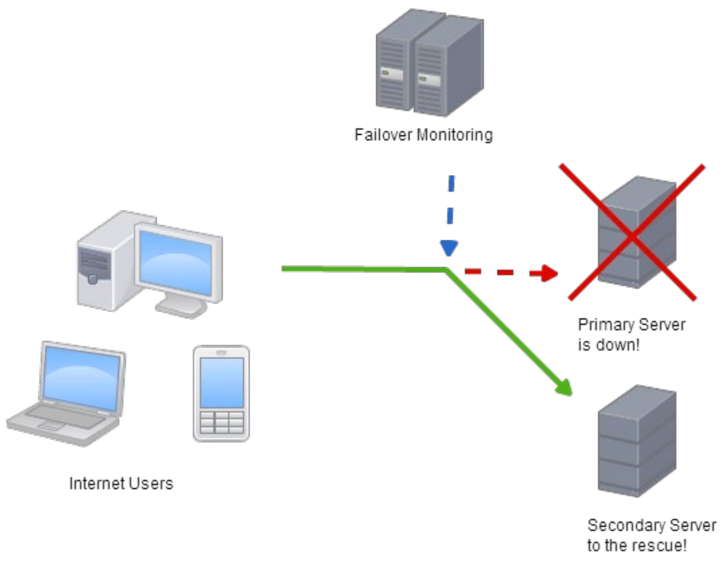


Figure 3. 7 – A Failover System for Cloud Computing Services  (HowToExpert, 2019)

#### 3.3.1.4 Geotagging

Geotagging refers to adding metadata about geographical location or zone to data. It can be implemented as a "data receptacle in a trusted platform module (TPM)" containing geolocation properties with the capability to tag data during the provisioning of a cloud resource (Erl et al., 2013). This gives users the opportunity to have a specified location for the placement of a workload and verification of the geographical location in which virtual hosts and workloads run. A geotag can make provisions for extensions to trusted cloud resource pools, which allows hardware pooling when it is being provided in the same geolocation (Samet et al. 2014). The mechanism aids cloud resource provisioning as it defines and presents statistical data about demand for resources across geographical zones.

With high-level annotation data processing, storage and management for holistic semantic annotation, Geotagging will provide a basis for defining geographic restrictions for some instances of annotation data in line with data confidentiality and privacy concerns. It is commonly used in different application areas such as in web search engines to define geographical tags for data such as images and videos. In addition, studies on other ways of using its features have included the location of tweets for localisation on the web (Jonathan & Mokbel, 2017).

### 3.3.2 Specialised Cloud Mechanisms

There are specialised cloud mechanisms which, unlike several other cloud computing mechanisms inherited from traditional computing models or other computing paradigms such as grid computing and utility computing, are native to cloud computing paradigm and were standardised as technical solutions to specific IT challenges in cloud environments. These include mechanisms for automated scaling, cloud usage monitoring, load balancing and cloud workload scheduling which are analysed in this section.

#### 3.3.2.1 Automated Scaling

Automated Scaling refers to a mechanism for automating the number of instances of a cloud resource running at every point in time, providing a means for cloud resources to dynamically scale in proportion to real time volume of demands (Novak et al., 2019). It can be implemented via a listener which acts as a service agent responsible for monitoring and tracking communication between users and cloud services being accessed in order to achieve dynamic scalability. The determination of workloads is possible by the volume of server processing demands initiated by user requests (Erl et al., 2013). Therefore, the major aim of using an automated scaling mechanism is to enhance the automatic adjustment of cloud resources which effectively minimises costs while still meeting service level agreements (Jiang et al. 2013). Figure 3.8 illustrates an auto-scaling mechanism for a cloud resource defining a minimum size, desired capacity and maximum size for scaling out as needed.

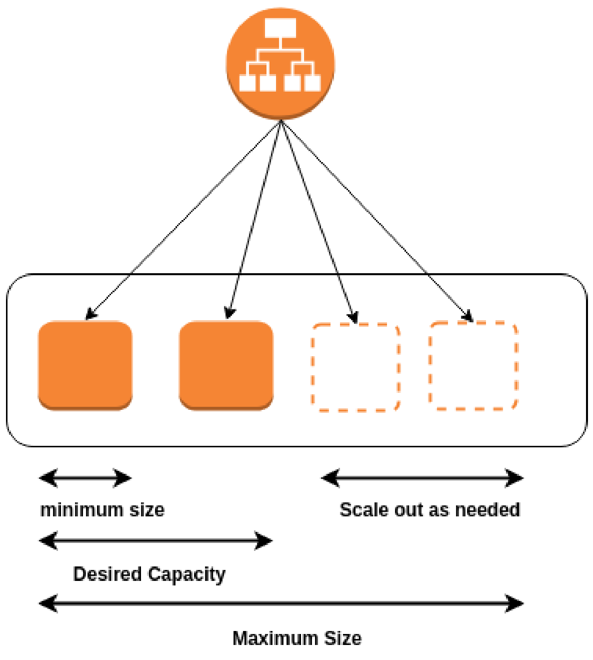
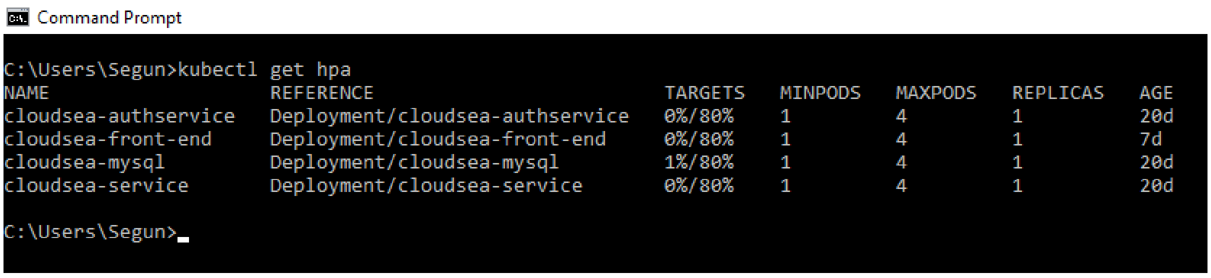


Figure 3. 8 – Automated Scaling Mechanism (Amazon AWS, 2018)

Furthermore, the mechanism eases burden on cloud administrators by means of the automation; requiring no human intervention once implemented. With such mechanisms, software agents can run automatically based on pre-defined metrics and thresholds. Thresholds can be set such that when the observed performance metric exceeds or falls below a specified value, there is an addition or subtraction of predefined number of resource instances accordingly. This type of automation helps to enhance the merits of cloud dynamic scalability, providing the opportunity to have additional resources for handling increased workloads and shutting down redundant resource instances (Novak et al., 2019). Some of the performance metrics used when cloud auto-scaling mechanisms are deployed include CPU and memory utilisation, disk operation, bandwidth usage and so on. These metrics are based on the performance of resources and they help to indicate system utilisation information (Papadopoulos et al., 2016). Figure 3.9 shows automated scaling parameters for containerised applications running on Kubernetes orchestration platform. The containers in this case have been configured to a minimum size of 1, desired capacity of 1 and dynamic scalability of up to 4 instances each. The threshold for initiating a new instance has been set to CPU utilisation of 80% of the allocated CPU usage for each container.



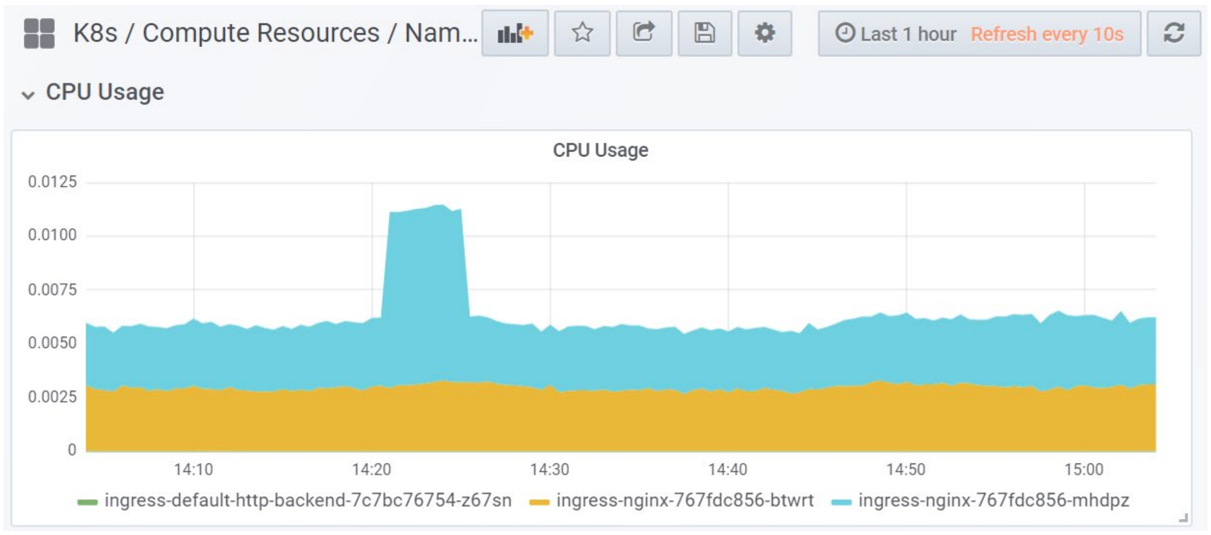
```
C:\Users\Segun>kubectl get hpa
NAME                   REFERENCE                         TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
cloudsea-authservice   Deployment/cloudsea-authservice   0%/80%    1         4         1          20d
cloudsea-front-end     Deployment/cloudsea-front-end     0%/80%    1         4         1          7d
cloudsea-mysql         Deployment/cloudsea-mysql         1%/80%    1         4         1          20d
cloudsea-service       Deployment/cloudsea-service       0%/80%    1         4         1          20d

C:\Users\Segun>
```

Figure 3. 9 – Horizontal Pod Autoscaling in a Kubernetes Cluster

With such features, auto-scaling for individual holistic semantic annotation capabilities based on the requirements defined in section 3.1 would be vital to processing fluctuating client requests while also optimising computing resources usage and computational overhead.

#### 3.3.2.2 Cloud Usage Monitoring

With the wide range of resources available in cloud computing across the different delivery models, it is vital to implement adequate monitoring mechanisms for diverse purposes such as computing resources utilisation, issue tracking, audit trail and analytics among others (Leach et al, 2017). Some commonly monitored metrics in cloud computing include CPU, memory, bandwidth, storage and database utilisation levels. Monitoring mechanisms ingest data from several sources and process them accordingly. According to Arcitura Education (2018), three types of agents function together in cloud computing for monitoring cloud resources usage; monitoring, resource and polling agents. A monitoring agent is based on events and an intermediary service agent, existing along the paths of communication in order to provide transparent monitoring and analysis of the flow of data. The resource agent helps in the collection of usage data through event-driven communications with certain specialised resource software while a polling agent is a processing module known for the collection of cloud service usage data by polling IT resources. Figure 3.10 presents an example of cloud usage monitoring within a Kubernetes container orchestration cluster for CPU utilisation by resources within a namespace using open source tools Prometheus and Grafana.



K8s / Compute Resources / Nam...
Last 1 hour
Refresh every 10s
CPU Usage
CPU Usage
0.0125
0.0100
0.0075
0.0050
0.0025
0
14:10
14:20
14:30
14:40
14:50
15:00
ingress-default-http-backend-7c7bc76754-z67sn
ingress-nginx-767fdc856-btwrt
ingress-nginx-767fdc856-mhdpz

Figure 3. 10 – Cloud Usage Monitoring in Kubernetes using Prometheus & Grafana

The several benefits of cloud usage monitoring mechanisms apply to both cloud providers and consumers. As described in the study of Shao et al. (2010), one of the important features of cloud computing services is the ability of cloud providers to have monitoring mechanisms for allocated resources. This helps in providing efficient services to cloud consumers by handling future requests based on statistical monitoring data. In addition, cloud consumers benefit from the analysis of resources requirements as well as being able to obtain value for the cost incurred in cloud resources usage (Dhingra et al., 2012).

### 3.3.2.3 Load Balancing

Load Balancing is executed by agents known as load balancers which work on runtime and are programmed with basic logic aimed at employing horizontal scaling. This is to ensure uniform distribution of workload across multiple instances of a cloud resource, fostering increased performance and capacity not obtainable with a single instance for same workload (Milani & Navimipour, 2016). This is achieved by algorithms that divide roles with a variety of techniques for their implementation across different cloud platforms. Some of the popularly utilised algorithms include Round Robin, Weighted Round Robin, Carton, HoneyBee and Throttled Load Balancing among others (Aslam & Shah, 2015). Furthermore, balancing requests for resources allow load balancers prevent an instance from being a single point of failure, subsequently, improving availability of applications and fostering an increased level of responsiveness (Ghomi et al., 2017). Some other capabilities of load balancers include ability to direct traffic which is based on transport layer protocols or based on data obtained from the network; content switching which is the ability to reroute decisions depending on data from the application layer and attributes like SSL, session ID and HTTP; and load balancing in global servers which is the applicability of the aforementioned capabilities in achieving load balancing within server farms across multiple geographic distributions (Erl et al., 2013). Figure 3.11 illustrates a load balancing scenario within a cloud environment.

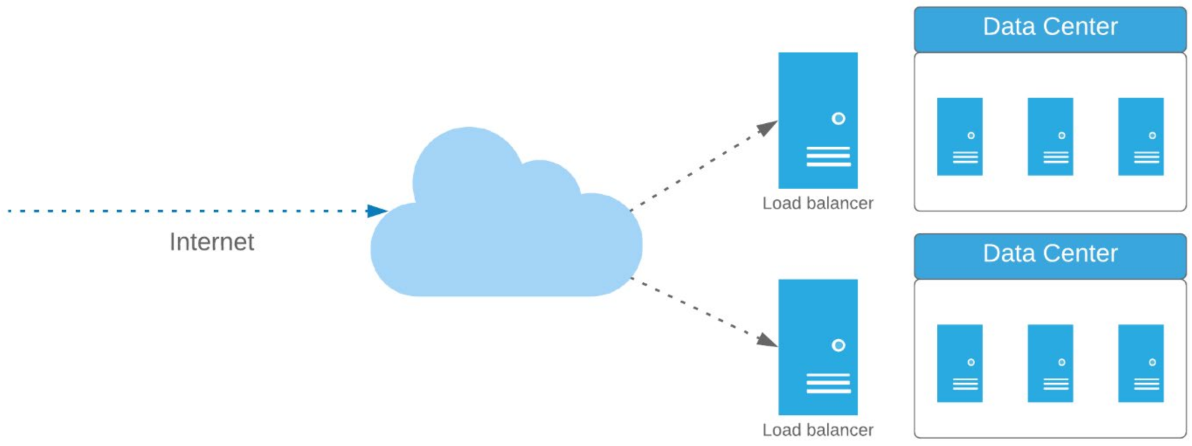


Figure 3. 11 – Load Balancing in Cloud Computing

#### 3.3.2.4 Cloud Workload Scheduling

Workload in the cloud refers to the amount of processing assigned to a cloud computing resource over a given period (Agarwal & Jain, 2014). Workload scheduling is implemented by software agents known as schedulers and refers to the process of automating and controlling several processes or workflows within a cloud infrastructure with their allocation to available resources as each process requires (Erl et al., 2013). With this capability, cloud platforms can maximise available computing resources and ensure they are allocated for optimal consumption. A typical task for workload schedulers is to provision new services such that they are placed in hosts with relatively few active services running (Santhosh & Ravichandran, 2013). They also define policies which are set as default for assigning resources and runtime to different processes.

The policies determine strategies to follow when scheduling services especially if an administrative runtime input is not available. Runtime decisions made by the scheduler is based on key resources such as throughput which can be measured based on the total number of processes that complete their execution per unit time (Agarwal & Jain, 2014). Likewise, latency is another of such resources which can be measured as the total turnaround time between submission of a process and its completion. Furthermore, response time is essential for measuring user experience; and can be measured as the amount of time between the submission of a request and receipt of a response on the user's device (Kakadia et al., 2013). Workload Scheduling is a very crucial capability for holistic semantic annotation. With a distributed system requiring a high level of automation, workload scheduling to manage resources for application functionality is inevitable. The automation and controlling of processes would be a much-needed functionality for web-scale holistic semantic annotation. For instance, the consistency between web documents and annotation data would require a scheduling task which monitors web documents against their corresponding annotation data at specified intervals.

### 3.3.3 Cloud Management Mechanisms

Cloud management mechanisms can be described as focusing on management tasks required within a cloud environment. Such tasks include the set-up, configuration, maintenance and monitoring of IT systems. These include mechanisms for Billing Management Systems, Resource Management Systems and SLA Management Systems.

#### 3.3.3.1 Billing Management System

Billing Management Systems in cloud computing are designed to collect and process data relating to cloud resources utilisation by cloud consumers for accounting and billing purposes. They utilise monitoring mechanisms for gathering required usage runtime data (Erl et al., 2013). The billing management system makes it possible to define various pricing policies, including price models suited for different categories of users as well as facilities for limited or unlimited usage. Most providers implement a system that bills cloud users on a pay-per-use basis which monitors and acquires usage data at runtime and stores them in repositories (Sui et al., 2018). The system uses these to generate billing and invoice reports, thereby pricing users based on their usage and fostering an effective billing model.

The dynamism of such billing models is well suited for cloud computing in which billing is a function of instantaneous cloud loads combined with pricing information acquired depending on the service provider’s billing specifications. It also entails identifying cloud loads throughout usage history, determining the load-based entities that correspond to the service provider’s pricing details (Iwashita & Tanimoto, 2013). A modern billing model, for instance, bills for resources such as memory and CPU utilisation, storage and network traffic, among others. Some billing systems can dynamically adjust their billing rates in a given period depending on infrastructure and load conditions. This enables providers to apply computing resources and services more efficiently (Li et al., 2019).

#### 3.3.3.2 SLA Management System

Service Level Agreements (SLAs) determine the type of services and resources a cloud consumer can expect from a cloud service provider. These in turn determines costs for using such services over a specified period between the consumer and the service provider. They assist cloud service consumers determine service instances they want to be provided with, the level of the service instances and the involved cost usually for a defined period (Singh et al., 2017). SLA Management Systems are responsible for maintaining contractual agreements between both parties with respect to service delivery (Zhao et al., 2013).

An SLA monitoring mechanism aids specific observation of runtime performance for cloud services to ensure the fulfilment of what is required based on contractual QoS (Quality of Service) as stated in SLAs. This is achieved through the collection of data by an SLA monitoring agent and its subsequent processing by an SLA management system for aggregation into reporting tools (Zhao et al., 2013). Based on reports analysis, cloud services performance can be enhanced to meet up with the required levels (Erl et al., 2013). Some factors which could lead to an SLA violation includes system or application malfunctioning and variations in workload conditions (Mehmood & Umar, 2017). The work by Singh et al. (2017) which focused on provisioning dedicated cloud services that can avoid SLA violations was based on the opinion that cloud systems need self-management services; requiring mechanisms that can manage resources automatically based on QoS requirements, thereby avoiding SLA violations quite well.

#### 3.3.3.3 Resource Management System

Resources management in the cloud provides a means of coordinating cloud resources based on actions and events from both cloud consumers and providers. Some of the tasks that the resource management system mechanism is able to automate and implement are management of cloud resource templates used for creating pre-built instances; allocation and release of cloud resources into available physical infrastructure; and coordination of resources based on interactions with other mechanisms (Erl et al., 2013). Others include enforcing usage and security policies for cloud service instances; and monitoring the operational status of resources. With factors such as complexity, geographical span as well as unending and unpredictable levels of demand for resources, cloud resource management requires complex decisions and policies to achieve objectives towards resources optimisation (Liaqat et al., 2017).

### 3.3.4 Cloud Security Mechanisms

With security being a prominent component of any computing infrastructure, this section analyses some cloud security mechanisms as they relate to the assessment of cloud computing capabilities for holistic semantic annotation. These are attribute-based access control, digital certification, digital signatures and identity & access management.

#### 3.3.4.1 Attribute-Based Access Control

Attribute-Based Access Control (ABAC) in computing systems were designed to enhance confidentiality of data by means of authentication and authorisation based on a set of pre-defined attributes and policies (Joshi et al., 2017). These include consumer, resource and environmental attributes as well as a set of policies specified in terms of the attributes. Consumer in this context is a person or non-person entity, such as a service or device with the ability to request resources or perform operations on resources. The entity being managed by ABAC is known as a ‘resource’ while policies spell out rules and relationships for determining the eligibility of a consumer to access a resource based on resource and environmental attributes (Erl et al., 2013). Environmental attributes provide operational or situational context for requests. Anytime a consumer requests for an operation or access, ABAC can investigate the required specifications in the access control rules and match them with the current value of consumer, resource and environmental attributes to determine eligibility (Dan et al., 2012). The ability of ABAC to define different matches of attributes and policies within a distributed system provides a fine-grained level of authorisation. This is perceived as a vital technology for defining different levels of authorisation based on a wide range of consumer types with varying attributes for holistic semantic annotation on the web.

#### 3.3.4.2 Digital Certification

With cloud technologies introducing additional IT security challenges, the main goal of digital certification is to build acceptance, trust and transparency for cloud users regarding data and application security within a cloud computing environment (Manjusha & Ramachandran, 2015). A typical cloud digital certification process should include extensive auditing for infrastructure and evaluation for services and contracts (Lins et al., 2016). These require satisfying specifications such as legal, security, contractual and functional requirements to ensure that consumers completely trust the reliability of a service before they adopt it.

Building such trust requires cloud providers to obtain digital certifications relating to security, which among others includes public key and validation certificates. Besides, most users adopt cloud technologies to access computing resources for processing and storing data. However, geographical locations of consumers and the physical infrastructure hosting their data are vital regarding legal and privacy compliance. Cloud users would experience lesser challenges adopting cloud services provided CSPs (Cloud Service Providers) include certifications for observing local and international compliance regulations and standards (Lins et al., 2016). Such include GDPR, European Union Data Protection Directive, US Patriot Act. Certifications for preserving digital information such that it can be accessed and processed without interruptions go a long way in reassuring cloud users.

#### 3.3.4.3 Digital Signature

Digital Signature is a mechanism which works with digital certification for protecting the integrity of data through authentication and non-repudiation mechanisms (Sonali et al., 2015). A digital signature makes provision for confirming the message authenticity by comparing it with that created by its original sender. It achieves this by assigning a digital signature before messages are transmitted, rendering the message invalid if it finds out that there has been any unauthorised modification to the data. The mechanism makes use of hashing and asymmetrical encryption to create a digital signature which is in the form of a message digest from the encryption of a private key and appended to the original message. The message receiver verifies that the signature is valid by using the corresponding public key for decrypting the digital signature's encrypted hash, which produces the message digest. The function of the hashing mechanism is to produce the message digest. The integrity of the message is intact when results from the two different processes are identical (Erl et al., 2013).

#### 3.3.4.4 Identity and Access Management

Identity and Access Management (IAM) mechanisms provide features and policies required for controlling and tracking identities of users and access privileges for cloud resources, environments and systems (Werner et al., 2017). Four basic features of IAM mechanisms include authentication, authorisation, user management and credential management (Indu et al., 2018). The general forms of authenticating user credentials include username and password profile with support for other security features such as attribute services which helps with the definition of attributes for controlling access. Authorisation on the other hand, defines resources that specific users can access after being authenticated for a resource. The user management feature manages user administration while credential management establishes and manages identities using credential issuance. The main reason for utilising IAM mechanisms is to administer authorisation, denial of service and overlapping trust boundary threats (Erl et al., 2013).

Access to holistic semantic annotation requirements such as annotation data storage, sharing and reuse can require identity and access management mechanisms depending on permission rights on the data. While some annotation data might be available from public RDF graph repositories such as the Linked Open Data Cloud, others might require authorisation and other security mechanisms to access them. Such scenarios might be obtainable with confidential data in sectors such as health, finance and military among others.

## 3.4 Cloud Computing Capability Model for Holistic Semantic Annotation

Having identified the requirements for holistic semantic annotation (in section 3.2) and cloud computing mechanisms required for their implementation earlier in this section, a mapping between both is presented. These mappings are based on the technical processes for implementing the holistic requirements and the technical functionalities of each of the cloud computing mechanisms mapped against them. In each case, the technical specification for the requirement is reviewed and examined against each of the cloud computing mechanisms in relation to their technical functionalities. Hence, a mapping is established if the cloud computing mechanism's technical functionalities proffer a solution for the implementation of a requirement's technical specification. The mapping table is presented in Table 3.4.

Table 3. 4 - Mapping Cloud Computing Mechanisms with Holistic Semantic Annotation Requirements

| | **Concept Extraction** | **Annotation On-the-fly** | **Annotation Data Auto-Update** | **Annotation Data Sharing** | **Annotation Data Reuse** | **Annotation Data Optimisation** | **Annotation Data Storage** | **Ontology Auto-Update** | **Annotation Data Co-Location** | **Ontology Mapping** | **Ontology Population** | **Ontology Selection** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Resource Cluster | | ✓ | | | ✓ | | ✓ | | ✓ | | ✓ | |
| Resource Replication | | ✓ | | | ✓ | | ✓ | | ✓ | | ✓ | |
| Failover System | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Geotag | | ✓ | | ✓ | | | ✓ | | ✓ | | ✓ | |
| Automated Scaling Listener | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Cloud Usage Monitor | ✓ | ✓ | ✓ | | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ | ✓ |
| Load Balancer | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Cloud Workload Scheduler | ✓ | ✓ | ✓ | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | |
| Billing Management System | ✓ | ✓ | ✓ | | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | |
| SLA Management System | ✓ | ✓ | | ✓ | | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Resource Management System | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Attribute-Based Access Control | ✓ | | ✓ | ✓ | | | ✓ | ✓ | | | ✓ | ✓ |
| Digital Certification | ✓ | ✓ | | ✓ | ✓ | | | | ✓ | | | |
| Digital Signature | ✓ | ✓ | | ✓ | ✓ | | | | ✓ | | | |
| Identity and Access Management | | | | ✓ | ✓ | | ✓ | | ✓ | | ✓ | ✓ |

From Table 3.4, it can be observed that the requirements for holistic semantic annotation on the web can be facilitated using a cloud architecture which has been designed for this purpose and that considers the specific needs for each of the requirements. These mechanisms are required to be implemented as a core suite for this objective alongside other necessary cloud computing mechanisms for deploying applications in the cloud. Different cloud architectural models emphasise cloud characteristics to varying degrees and deploy cloud design patterns and mechanisms accordingly to meet the requirements of the cloud characteristics (Erl et al., 2015). This implies that patterns and mechanisms can be implemented for specific solutions in order to facilitate or enhance application functionality. Hence, from Table 3.4, a cloud computing capability model for holistic semantic annotation is proposed, as presented in Figure 3.12.

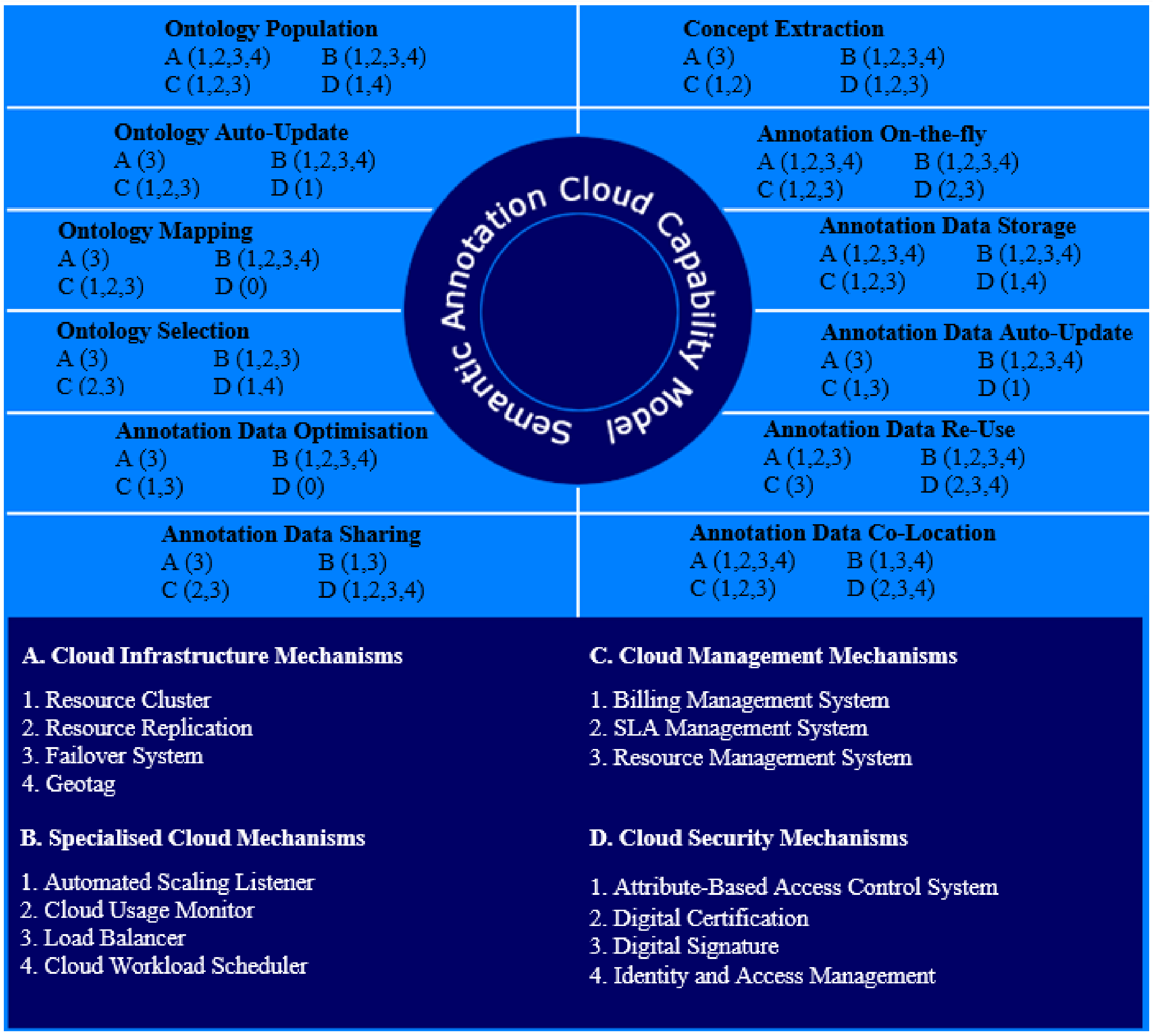


Figure 3. 12 – Cloud Capability Model for Holistic Semantic Annotation

From Figure 3.12, it can be observed that cloud computing has a significant role to play in the realisation of a truly semantic web through the facilitation of a holistic semantic annotation solution; one in which documents and resources on the web can be provided with the required rich-content semantic annotation for data dynamically and continuously, based on a cloud hosted solution. The semantic annotation of web documents thereby, results in a context-aware web, in which web documents and resources are processed as “things” rather than as “strings”. While several applications run from a cloud environment, different application deployment patterns in the cloud exist. Furthermore, differences between the patterns offer and leverage cloud computing benefits to varying degrees. Furthermore, the scope and utilisation scale for applications play a role in determining and selecting an application deployment pattern in the cloud. With the web-scale nature for semantic annotation, it becomes more pertinent to investigate different application deployment patterns in the cloud and adopt or adapt a suitable option. The investigation might also suggest the design of a novel application deployment pattern for holistic semantic annotation solution in the cloud. The model presented in Figure 3.12 aims to implement an application deployment pattern in the cloud that best meets the requirements for holistic semantic annotation and that fully maximises cloud computing characteristics and benefits.

## 3.5 Chapter Summary

In this chapter, a holistic perspective to the process of semantically annotating documents on the web based on the varying challenges and disparity observed within existing semantic annotation solutions was proposed. The holistic perspective is with the aim to address challenges relating to automatic semantic annotation and it is based on the refinement and formulation of a set of requirements from research and the addition of some novel ones too. These requirements were further classified into three categories; pre-annotation, annotation and maintenance phases. Furthermore, the nature of a holistic process requires a technological paradigm to drive it and a case for the use of the cloud computing paradigm was presented. The nature of cloud computing, which also defines its benefits to software applications was fundamental to the choice. In the next chapter, different application deployment patterns in the cloud would be investigated and analysed based on a set of determinant metrics. This results into the development of a Cloud Computing Maturity Model for Holistic Semantic Annotation.

# Chapter 4: Cloud Computing Maturity Model for Holistic Semantic Annotation

Having identified and proposed a set of requirements for holistic semantic annotation and assessed how to leverage cloud computing capabilities for a holistic process, this chapter provides an investigation into different application deployment patterns in the cloud for holistic semantic annotation. There are varying degrees to maximising cloud computing benefits and these are dependent on a set of factors some of which are software architectural pattern (for cloud software layer), the implementation and configuration of technological artefacts for delivering cloud computing capabilities (for cloud platform layer), the physical cloud infrastructure (for cloud infrastructure layer) as well a few others. These are critically evaluated in terms of the various techniques for leveraging them and obtaining maximum benefits for the deployed application.

Based on the critical analysis and evaluation, a cloud computing maturity model for holistic semantic annotation is proposed. The model defines maturity levels for holistic semantic annotation in the cloud from which a novel approach is developed for delivering web-scale automated semantic annotation as a cloud service. To achieve this, the chapter investigates software architectural patterns in section 4.1, virtualisation patterns in section 4.2, containerisation and microservices in section 4.3, container orchestration technologies in section 4.4 and application automation lifecycle in section 4.5 prior to developing the cloud computing maturity model in sections 4.6 and 4.7.

## 4.1 Software Architectural Patterns

A Software Architecture Pattern can be defined as a system's basic organisation found within its components and their inter-relationship (Franchitti, 2019). A system's architecture offers description of key components within it, the way such components relate (structures), and their interactions (Richards, 2015). Solms (2012) describes a component as an encapsulated feature of a software system characterised by an interface, serving as a building block for the system's structure and its representation at the programming language level can be as a module, class, object or as a collection of related functions. The software architectural level provides a medium for analysing quality features such as reliability or usability, which would not be possible to do at the code level (Franchitti, 2019). Software architecture helps stakeholders to communicate; stakeholders are those involved in the making and use of a software system. It also stands for design decisions made at the early stages of the development of a software (Hao et al., 2017). Due to the architecture being the system's highest level of decomposition, it is utilised as a work-breakdown structure; thereby dictating units to plan, schedule and budget. It can also be used to check whether the system will exhibit its required features even before its development and deployment. Software architecture is an abstraction that can be transferred, thereby promoting large scale reuse (Franchitti, 2019).

There are some commonly used architectural patterns and styles in software design. Some of them include client-server, component-based, data-centric, event-driven, layered, microservices, service-oriented, space-based and more (Richards, 2015). The Event-Driven Architecture promotes, produces, detects and reacts to events. The Space-Based Architecture is mostly used to achieve linear scalability of stateful, high-performance applications with the tuple space paradigm. Its principles are quite similar with those of Representational State Transfer (REST) and Service-Oriented Architecture (SOA) (Jacob & Mani, 2018). The Component-Based Architecture works based on reuse to define, implement and compose loosely coupled independent components into systems. With SOA, it works by providing services to other components within an application, through a communication protocol over a network; its basic principles do not depend on vendors, products and technologies. A service is a distinct unit with individual functions accessible remotely, acted upon and updated independently (Richards, 2015). The Client-Server Architecture uses a structure for the partitioning of tasks or workloads between providers of a resource or service, called servers, and service requesters, called clients (Mainetti et al., 2015).

The Microservices Software Architecture is seen in most quarters as a fork of SOA (Richards, 2015). However, with Microservices Architecture, sub-systems (hereafter referred to as microservices) are focused on implementing a specific task and the protocols are lightweight (Newman, 2015). This architecture makes it possible to decompose an application into distinct smaller units thereby improving modularity and offering several benefits such as easy comprehension of an application, easy development and testing, including resilience to architectural erosion (Newman, 2015; Richardson, 2015). With Microservices Architecture, there is parallelisation of development by allowing independent development, deployment and scaling of services by small autonomous teams. In addition, Microservices Architecture enables continuous delivery and deployment (Carneiro & Schmelmer, 2016). The following sections would present a critical evaluation of the Microservices Architecture as well as comparison with some other popular software architectural patterns.

### 4.1.1 Microservices Software Architecture

Due to advancements in technological innovations, various factors have arisen that required the evolvement of software architectures in order to cope with solution requirements across several industry sectors. Dynamic and interactive applications demand availability and scalability from a software architecture point of view (Aderaldo et al., 2017). The Microservices Software Architecture is an evolving one which provides such capabilities. The concept of Microservices requires splitting applications into smaller services such that each service can be tested, scaled, implemented, monitored and deployed independently. Such a deconstruction allows all functionality components to be deployed or updated without affecting other components of the application (Florio & Di Nitto, 2016; Newman, 2015). It presents an approach for software service design, delivery and development that focuses software application's development processes on well-established modularisation concepts and emphasises on technical boundaries (Thones, 2015). Each microservice is developed and deployed independently, utilising a well-established network interface to provide access to its interior data and logic. As a result, since every microservice is an independent unit of design, deployment, development, scaling and versioning, software agility improves and increases significantly (Daya et al., 2016). Figure 4.1 presents a simplistic illustration of a microservices architecture.

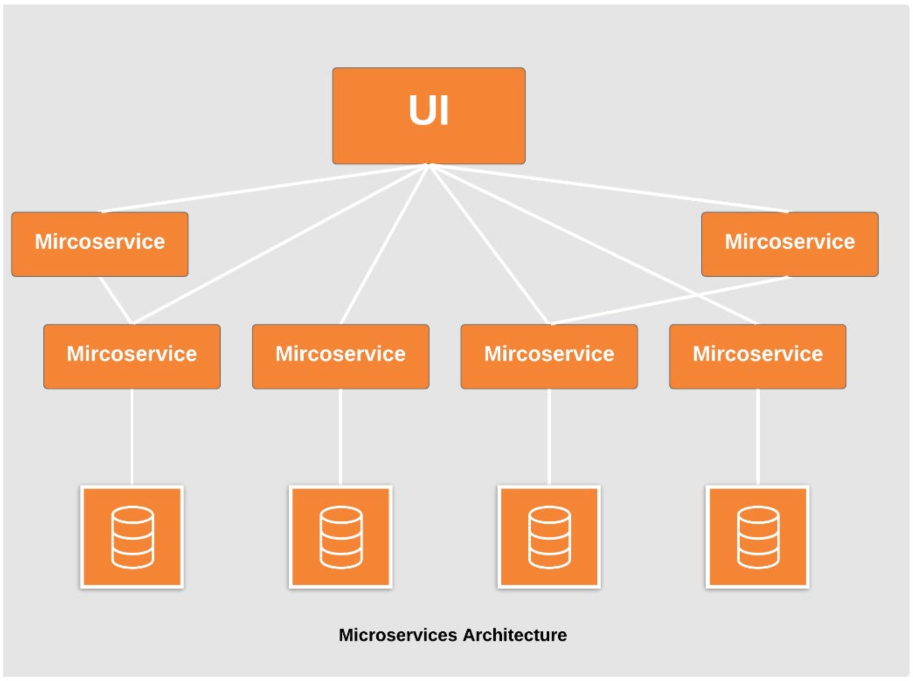


Figure 4. 1 – A Simplistic Example of Microservices Software Architecture

Microservices Architecture was introduced to eliminate the demerits of the Monolithic/N-Tier Architecture that is characterised by application logic within one deployable unit (Richardson, 2015). Monolithic/N-Tier systems are more suitable for small systems and likewise, possible to achieve availability and scalability to reasonably high degrees through auto-scaling and load balancing mechanisms. However, with an increasing growth of such systems, challenges such as code complexities, increase in deployment time, scalability for loads that are data intensive as well as the gradual appearance of a long-term commitment to a technology stack are bound to occur (Balalaie et al., 2016). Hence, Microservices aid the provision of small services characterised by easy comprehension, capable of independent deployment and scalability and can run on different technology stacks (Bakshi, 2017). The focus of each microservice is the completion of just one task thoroughly; and in most cases, the single task stands for a small business capability. Furthermore, it is possible to implement the development of microservices with a technology stack different from that of other microservices within an application (Newman, 2015). This is known as polyglot programming which advocates for using the most appropriate programming language for each microservice while polyglot persistence involves the use of multiple dedicated and different storage systems for each microservice, as appropriate (Wilder, 2012). Each microservice makes use of the most appropriate language and/or storage mechanism based on its requirements (Gilbert, 2018). The communication between microservices which is commonly referred to as inter-process communication is through a neutral API language like REST. For very large, data-intensive applications, messaging platforms such as Apache Kafka, RabbitMQ, ZeroMQ, etc are usually utilised based on their capacity to process massive amounts of messages at once. Another feature of Microservices Architecture is the Bounded Context which implies that microservices within an application need not have any knowledge about the basic implementation or architecture of other microservices (Zimmermann, 2017).

Furthermore, each microservice should constitute an independent software, with its own separate code base; having personal delivery pipelines for builds and deployments. The independence between microservices makes them loosely coupled which ensures frequent and rapid deployments and it gives consumers the opportunity to obtain features and capabilities they really require (Richardson, 2015; Zimmermann, 2017). While several research efforts identify and discuss a range of key principles for Microservices Architecture with slightly varying opinions, most of the principles are observed to be uniform across multiple sources.

Table 4.1 presents an investigation of Microservices Architecture key principles as identified from different literature sources.

Table 4. 1 – Microservices Key Principles Across Literature

| | **MICROSERVICES KEY PRINCIPLES** | **Newman (2015)** | **Daya et al. (2016)** | **Jaramillo et al. (2016)** | **Jung et al. (2016)** | **Open Group (2016)** | **Microsoft (2019)** | **Amazon AWS (2019)** | **Baskarada et al. (2018)** | **Shadija et al. (2017)** | **Carnell (2017)** |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Model Around Business Concepts | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| 2 | Adopt a Culture of Automation | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ | ✓ | | |
| 3 | Hide Internal Implementation Details | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| 4 | Decentralise All the Things (including Polyglot Programming) | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| 5 | Independently Deployable | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| 6 | Failure Isolation | ✓ | ✓ | ✓ | | | ✓ | | ✓ | | ✓ |
| 7 | High Observability | ✓ | | ✓ | ✓ | | | ✓ | | | |
| 8 | Loose Coupling and High Cohesion | | | | | | ✓ | | ✓ | ✓ | |
| 9 | Service Discoverability | | ✓ | | | | ✓ | | | | ✓ |
| 10 | Well-Defined and Published API Interface | | | | | ✓ | ✓ | | ✓ | | |
| 11 | Offloading Cross-Cutting Concerns to Gateway | | | | | | ✓ | | | | |

From Table 4.1, it can be observed that while some of the key principles' cuts across the different literature sources, a few are not so commonly defined. However, with detailed analysis of the more common ones, the others can be observed to be inclusive within them. This suggests a high level of uniformity in terms of the required key principles for Microservices Architecture. Furthermore, successful microservices architectural designs and deployments can be traced to current software engineering models and present-day progress in web application development approaches that promote the adoption of standards and best practices, also known as DevOps (Garg & Garg, 2019).

### 4.1.2 Benefits of Microservices

From the overview of Microservices Software Architecture in section 4.1, it can be observed that the architectural pattern provides several benefits, such as the following:

- First and foremost, it provides faster 'time to market' for applications and software products by speeding up processing time as separate development teams can work on different microservices of the same application in parallel (Taibi et al., 2017). In addition, Dragoni et al. (2017) opines that most organisations today use microservices to become more agile in responding to market changes. With the ability to leverage both hypervisor-level and operating system-level virtualisation for applications and deploy them on a cloud platform, supported by a software integration that is fully automated, the speed of implementing new software features is greatly increased.
- Microservices Architecture is also beneficial since each microservice is its own unit of scaling. This implies that at runtime, the different requirements for each microservice dictates the level of scaling needed (Dragoni et al., 2017). An independent scaling mechanism not only reduces the time required to scale in or out depending on the application's demand for resources, but it also ensures that each component only scales for the needed resources, thereby optimising computing resources utilisation.
- Furthermore, each microservice is an independent unit of development, thus allowing development teams to introduce new features without affecting the functionality of other sub-systems. Moreover, the independence of microservices enable ease of frequent deployments as compared to other traditional development practices (Newman, 2015). A specific part of an application can be developed, modified or deployed as many times as possible without affecting the normal functionality of the overall system.
- Microservices require different development teams to design, develop and test their own respective codes which are independent from others. This facilitates improved composability, maintainability and reusability of code, thereby fostering the development of quality software and application products.
- Within a Microservices Architecture, failure isolation can be easily implemented. The architecture promotes techniques such as caching, health-checking and circuit breakers (Richardson, 2015). These are important since they minimise the impact of a failing sub-system which in turn leads to the overall improved availability of a given software system.
- Lastly, Microservices Architecture facilitates the design and development of applications according to business domains. Rather than writing a bulky code to satisfy a certain technological need, they enable a business to create applications that are

focused on improving business functionalities within specified domains. The processes of a microservice can thereby be adapted for use in multiple instances. A functionality can be reused in multiple business processes or in differing business channels, depending on needs and requirements (Tanasseri & Rai, 2017).

### 4.1.3 Drawbacks of Microservices

While the Microservices Software Architecture provides solutions to some of the challenges faced with other software architectural patterns such as Monolithic/N-Tier Architectures, it poses a few drawbacks of its own. It is common practice for trade-offs to exist with different software architectures and these applies for Microservices Architectures as well. The major drawbacks obtainable with the architectural pattern are as follows:

- With a lot of moving parts; in terms of the existence of multiple microservices, it leads to an increase in operational complexity. Fowler (2015) supports this point, suggesting that it is necessary to have a sound operations team for the management of the numerous microservices characterised by regular deployment.
- It also leads to an increase in operating overhead, in most cases. With a Microservice Architecture, there may be need for more resources for deployment just as there exists numerous deployments (Wu, 2017). With Microservices, there is a frequent introduction of multiple databases, message brokers, data caches, and closely related services requiring maintenance, clustering and preservation (Daya et al., 2016). Therefore, this often leads to the need for additional time and effort for the creation of the infrastructure.
- While Microservices aid the provision of small autonomous services characterised by easy comprehension, capable of independent deployment and scalability, they still form part of a distributed system. According to Fowler (2015), the demerits of distributed systems are that they are quite difficult to program, as it takes time to achieve remote calls and the vulnerability to failure is constant.

Despite the drawbacks identified for Microservices Software Architecture, the benefits far outweigh them, hence the reason why its adoption rate is very high currently across different sectors in the society (Di Francesco et al., 2017). The drawbacks identified for Microservices can be mitigated by adopting the concept of DevOps (Zhu et al., 2016; Balalaie et al., 2016). DevOps is a term which stands for "Development and Operations" and it is based on the combination of practices, tools and cultural changes to provide automation of systems and increase the ability of an organisation to deliver services and applications at a higher velocity (Zhu et al., 2016; Garg & Garg, 2019). It works through the evolution of a product and provides improvements much quicker than traditional methods of software development.

### 4.1.4 Monolithic / N-Tier vs. Microservices

This section presents a comparison between Microservices Software Architecture and Monolithic / N-Tier Architectures. The comparison is based on the innovativeness of Microservices Architecture and the popularity of Monolithic / N-Tier Architectures. Monolithic / N-Tier architectures are a sharp contrast to Microservices in that all the code's components in the former are designed to work in unison, where the failure of one component impacts the entire application. The application's components work together as a single cohesive unit and they share the same memory space. A software product developed using such architectures is self-contained and all its components are interdependent and interconnected to each other (Villamizar et al., 2015). In addition, if the development team desires to implement new changes or create new updates, they may need to rebuild and redeploy the entire application from ground up (Escobar et al., 2016). The same case applies for scalability in that a single component scaling out will result in the entire application scaling as well. It is also difficult to add new technologies and using a new framework or platform may necessitate the application's code to be re-written. Figure 4.2 illustrates differences between monolithic and microservices architectures.

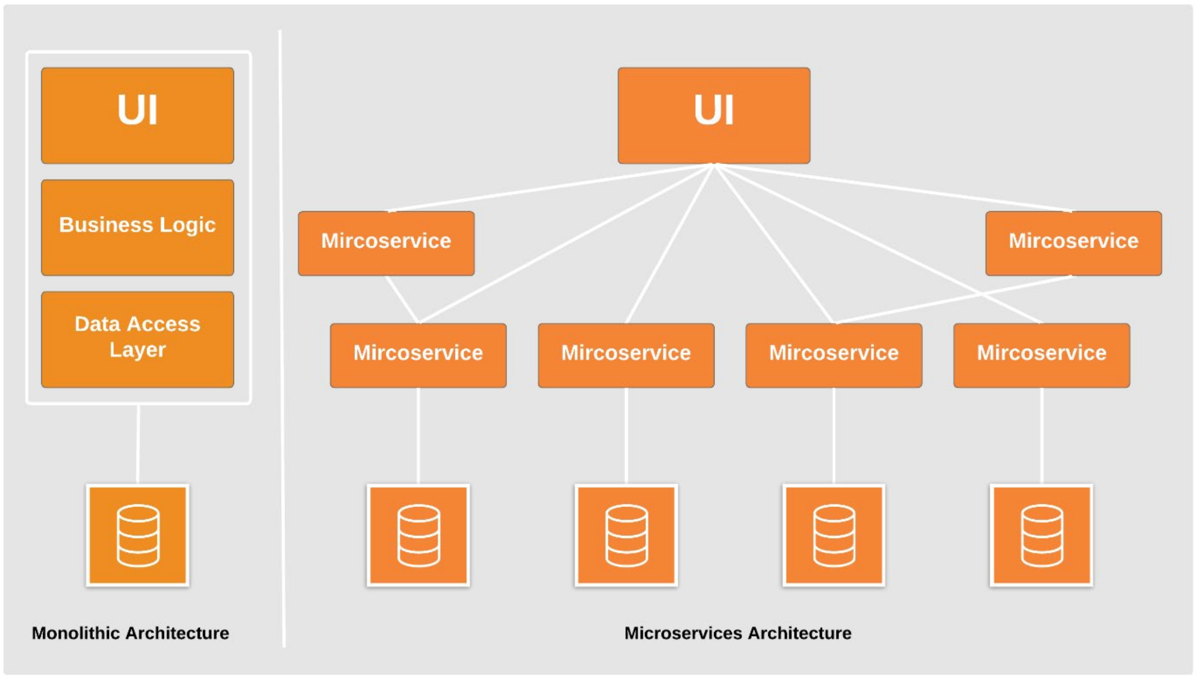


Figure 4. 2 – Monolithic / N-Tier Architecture vs. Microservices Architecture

These two can be compared based on several factors. These include Code, Understandability, Reusability, Deployment, Reliability, Adaptability, Scaling, Programming Language and Testing. Table 4.2 presents a comparison between them based on these factors.

Table 4. 2 – Comparison of Monolithic / N-Tier and Microservices Software Architectures

| **Factor** | **Monolithic / N-Tier Architecture** | **Microservices Architecture** |
|---|---|---|
| **Code** | Since the application is developed as a unit it comprises of a single code base. | Since there are multiple small services that make up a large application, there are multiple code bases with each microservice having its own. |
| **Understandability** | As the code base grows over time, it becomes more difficult to understand the entire process and make updates. Maintenance also becomes difficult, due to its size and complexity. | This is more understandable as it is micro in nature. It becomes easy to investigate each microservice to obtain any information especially while troubleshooting. Maintenance is relatively easier as well. |
| **Reusability** | Due to its size and complexity, it is not easy to reuse. | Any part of the entire application can be reused since they are independent in nature. |
| **Deployment** | Any update means redeployment of the entire application. Continuous deployment becomes difficult. It is complex to deploy within restricted maintenance windows and scheduled downtimes. | It is easier to re-deploy each microservice independently, with minimal or zero downtime. This promotes the possibility of continuous deployment for complex applications. |
| **Reliability** | When there is an issue with a module, it has the potential to bring down the entire application. Furthermore, troubleshooting the entire application could be very cumbersome and time-consuming. | The level of reliability is higher since each unit is on its own. It is easier to troubleshoot a microservice independently without any impact on the others within the application. |
| **Adaptability** | The adoption of emerging technologies is usually very difficult | Emerging technologies are easier to adopt for specific microservices and |

| | | |
|---|---|---|
| | as transforming from a technology stack to another would probably imply re-developing the entire application from scratch. This will be quite expensive in both time and cost. | with polyglot programming, microservices can run based on different technology stacks, giving developers the opportunity to adopt innovations quicker. |
| **Testing** | Testing is easier here and can be through end-to-end testing by launching the application. | There is a level of complexity in the process of testing this architecture due to the multiple microservices available. |
| **Technology** | The entire development is typically based on a specific technology stack. | Each microservice can be based on a separate technology stack from the others. |
| **Scaling** | The entire application needs to be scaled since it's a single unit. While horizontal scaling is simple when multiple instances run behind a load balancer, this implies a very high computational overhead and cost. | The entire application does not need to be scaled. Scaling can be automated for each microservice based on pre-defined thresholds, hence optimising computational overhead and cost. |

### 4.1.5 SOA vs. Microservices

Service Oriented Architecture (SOA) is an architectural pattern where components of systems receive services from other systems through a communication network. Communication is usually for data transfer and co-ordination of connected services. It usually comprises of service consumers and a service provider with the user's interface to the SOA being the consumer layer and the services provided by the SOA as the provider layer (Richards, 2015). While Microservices and Software-Oriented Architectures are similar in their reliance on independent services that have clear and well-defined boundaries, they also possess some notable differences (Cerny et al., 2017). It is possible to deploy and operate services independently in Microservices as opposed to SOA. In addition, SOA strongly relies on products like enterprise service buses and other similar heavyweight middleware whereas Microservices can utilise much lightweight technologies. As a result, application development with Microservices Architecture is much simpler and devoid of complex architectural requirements as is the case with SOA (Bogner et al., 2018). In SOA, a single point of failure can negatively affect the entire system since the communication is made through an Enterprise Service Bus (ESB), such that when one service is down, it leads to an overall communication breakdown within the application (Xiao et al., 2016). This is opposed to microservices which are built to be more fault tolerant.  For instance, if there is a memory failure in one of the services in a microservices architecture, only that service is affected, while other services continue to run (Richards, 2015). Figure 4.3 illustrates the difference between the two software architectures.

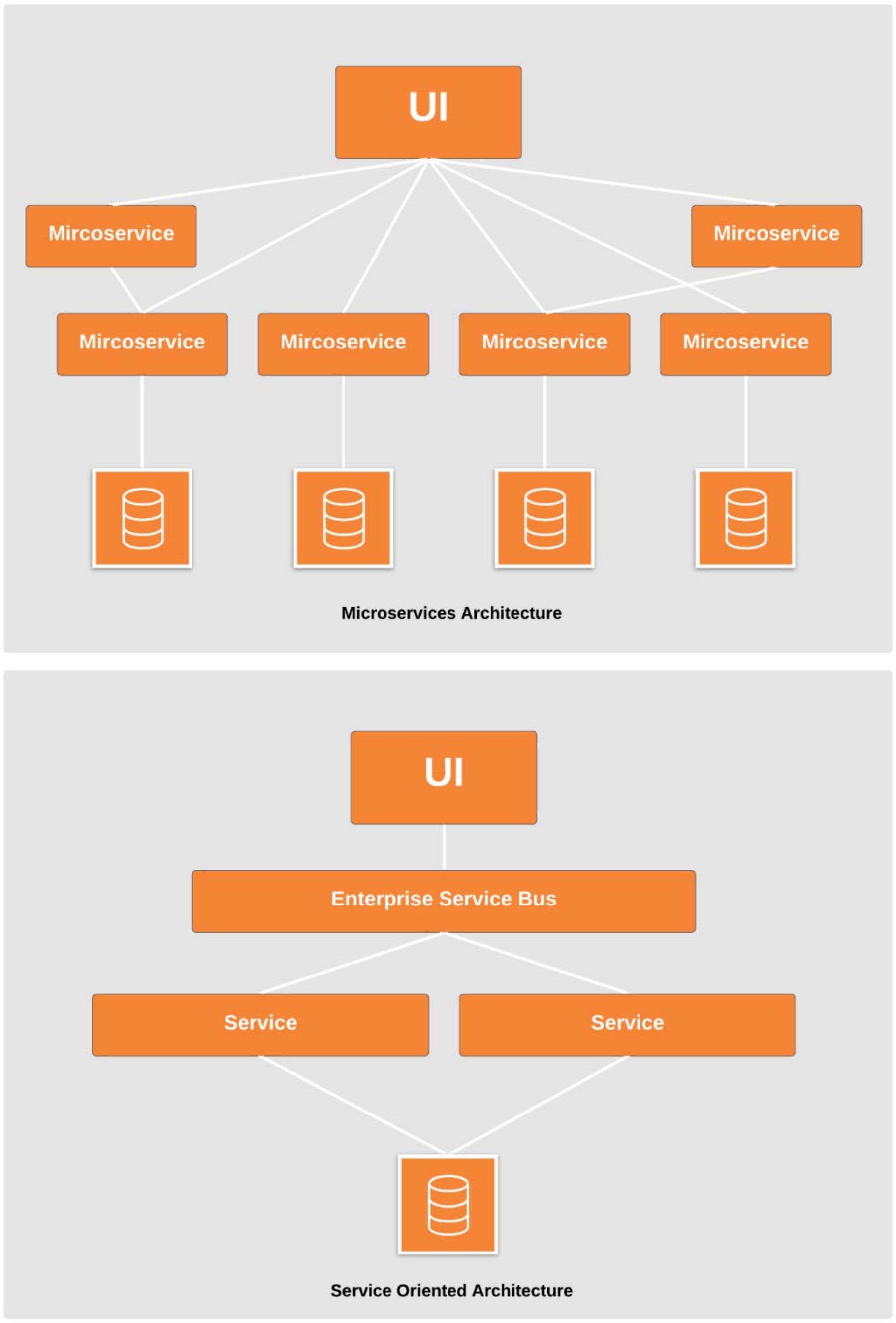


Figure 4. 3 - Service Oriented Architecture (SOA) vs. Microservices Architecture

Furthermore, from Figure 4.3 it can be observed that services in SOA share a single database. However, each service should have an independent storage with microservices. Sharing storage has risks of loss and hinders the independence required for individual services. In terms of scope and size, microservices are smaller compared to SOA (Xiao et al., 2016). According to Newman (2015), Microservices stand for how to do SOA right as the approach to its development has come from real world usage, requiring the use of better comprehension of systems and architecture in doing SOA well.

There are wide-ranging and extensive challenges attempted to be solved by SOA. This further provides an opportunity to observe differences between both. The attempt by SOA is to make services available to those who are interested in their use. However, the focus of microservices is established and the goals are limited, which is acting as a part of a single distributed system (Daya et al., 2016). Microservices are not about serving multiple systems at the same time. The existence of microservices is frequently implicit unlike with SOA. Furthermore, the discovery of microservices does not happen at runtime and there is no need for mediation as is the case with SOA (Cerny et al., 2017). Table 4.3 details the comparison between SOA and Microservices.

Table 4. 3 - Comparison of SOA and Microservices Software Architectures

| **Factor** | **Service-Oriented Architecture** | **Microservices Architecture** |
|---|---|---|
| **Component Sharing** | The development is based on the concept of a “share-as-much-as-possible” architectural style. This helps to eliminate the issue of business functionality duplication. This has the tendency to tightly couple components and increase the overall risk associated with change. | It utilises the concept of Bounded Context to hide implementation details from other microservices by coupling services and their associated data together as a single closed unit having minimal dependencies. |
| **Service Orchestration** | Service Orchestration is about coordinating multiple services through a centralised mediator. This is managed via the messaging middleware component of SOA by calling multiple services based on a request, giving SOA the tendency to | Microservices architecture utilises Service Choreography because of the absence of a centralised middleware component in the architecture topology. This makes its development, testing, |

| | | |
|---|---|---|
| | be slower than Microservices. It requires more time and effort for its development, testing, deployment and maintenance. | deployment and maintenance to be faster. |
| **Access to Remote Services** | There are no limitations to the protocol for accessing remote services. The availability of different types of remote-access protocols is one of the major distinctive features of SOA. | There is tendency to depend on REST as the main remote-access protocol. However, other messaging systems are available for simple messaging such as Redis and more robust, data-intensive messaging using message solutions such as Apache Kafka, ZeroMQ, RabbitMQ, etc. |
| **Application scope** | It is suitable for enterprise-wide systems requiring integration with several other software and web services. It suits applications having numerous shared components. | It is suitable for large-scale, web-based systems which are open for public consumption. |
| **Heterogeneous Interoperability** | It has the capability for the integration of multiple heterogeneous systems and services. | It attempts to achieve the simplification of the architectural pattern and corresponding implementation through the reduction of the number of choices to integrate services. |

## 4.2 Virtualisation Patterns

Virtualisation was first used in mainframe computers in the 1960s, as a method that logically divides resources of systems within a variety of application software (Naeem et al., 2016). The concept is based on the abstraction of computing resources and involves creating multiple instances of a resource from a physical one (Naeem et al., 2016; Carroll et al., 2012). Resources that can be involved include an operating system, a storage device, a server, computer network or an application. Several organisations utilise virtualisation to solidify their workloads as a means of dividing and sharing computing resources into multiple environments for execution by applying technologies such as time-sharing, quality of service and partitioning of computing resources (Ranganathan, 2018). In software development and the deployment of virtual infrastructure, the use of virtualisation is prevalent and non-disruptive. For software development, virtualisation facilitates the delivery of application software to many clients on demand. For virtual infrastructure, it provides the advantage of managing pooled resources across an enterprise, allowing a higher level of responsiveness to dynamic organisational needs and better leverage investment in infrastructure (Ranganathan, 2018; Jain & Choudhary, 2016).

Furthermore, virtualisation provides several advantages for the deployment of application software in a cloud environment. Firstly, it provides dynamic scalability; the scaling up and down of resources on-demand, which helps to minimise the infrastructure cost for an organisation by optimising computing resources usage (Morabito, 2017). In addition, it ensures faster provisioning and deployment of servers. With the agility and effectiveness for backing up data, it improves disaster recovery that could occur within a system, ensuring a high level of productivity for computing systems and resources. Several virtualisation types exist, these include server virtualisation, client and desktop virtualisation, services and applications virtualisation, network virtualisation and storage virtualisation (Naeem et al., 2016). For application deployment in the cloud, hypervisor-level and operating-system-level virtualisation are very relevant (Jain & Choudhary, 2016). These are analysed in sections 4.2.1 and 4.2.2 respectively.

### 4.2.1 Hypervisor-Level Virtualisation

Hypervisor-based Virtualisation is the most prominent virtualisation method in computing. It is based on the use of a hypervisor which is a computer software that allows abstraction from the hardware layer by intercepting the operating system call to the hardware (Eder, 2016). A hypervisor performs the function of a middleware between the physical host and operating system. Furthermore, it is commonly referred to as a Virtual Machine Monitor (VMM) because it creates and runs virtual machines and can either be a software, hardware or firmware (Gkortzis et al., 2016). A virtual platform is created on the computer (host machine), on top which various guest operating systems are run and monitored. A computer which runs multiple virtual machines on a hypervisor is referred to as a host machine and each virtual machine is called a guest machine, i.e. the hardware that runs the hypervisor is referred to as the host, and its operating system as the host operating system while all virtual machines running on it are referred to as guests and their operating systems as guest operating systems (Eder, 2016;

Morabito, 2017). There are three essential characteristics of a hypervisor; the provision of an environment identical to that of a real machine for programs; provision of appropriate performance for programs running in virtualised environments and the control of all system resources in the host machine or system (Ranganathan, 2018).

Furthermore, hypervisor-level virtualisation can be categorised into; Type 1 or Type 2 hosted hypervisor. The Type 1, native or bare metal hypervisor architecture runs on the host machine or hardware directly, controlling the hardware and managing the guest operating system (virtual machine). In this case, the hypervisor interfaces directly with the memory, devices and CPU on the host hardware, positioned between the hardware and the guest operating system. The type 1 hypervisor is more efficient and recommended than the type 2 hypervisor (Gkortzis et al., 2016). Some examples of the type 1 hypervisor architecture include; Xen and Oracle VM. The Type 2 or hosted hypervisor architecture runs as a program on the host system and is utilised for software virtualisation. It does not have same level of priority as the type 1 hypervisor and does not access the hardware directly since it runs as a program. A major advantage is that it can be installed on various host systems without modifying the system (Morabito, 2017). VMware workstation, VirtualBox and Parallels Desktop for Mac are some examples of this type of architecture. Figure 4.4 illustrates the two different types of hypervisor-based virtualisation.

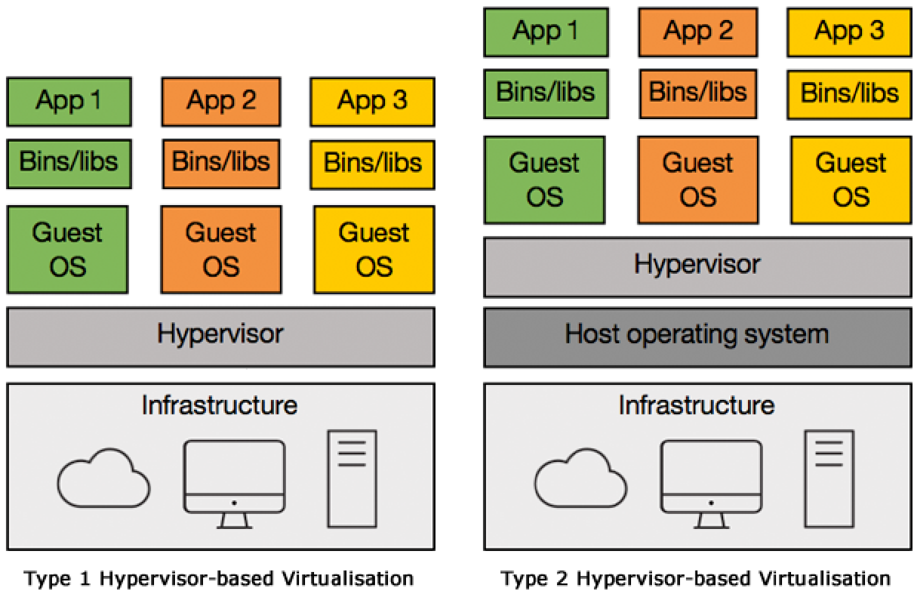


Figure 4. 4 – Hypervisor-Level Virtualisation (Infoworld, 2019)

### 4.2.2 Operating System-Level Virtualisation

Virtualisation at the operating system level, also known as containerisation or container-based virtualisation is a lightweight alternative to hypervisor-based virtualisation. In this case, there are no hypervisors involved and host operating system level virtualisation is performed (Eder, 2016). The host OS (Operating System) kernel is shared by all virtualised instances since there are no hypervisors involved. This largely reduces the runtime overhead and sharing the same operating system also reduces the storage overhead (Taherizadeh & Stankovski, 2018). The virtualisation layer is positioned between the operating system and application programs running on it. Sets of applications or software written for an OS being virtualised is run by the virtual machine (Jain & Choudhary, 2016). Each application runs in what is known as 'Containers'. Containers provide a level of abstraction on top of host operating system kernel, allowing each instance of a container to behave as an independent system with isolation. While the OS provides a platform for the containers to be deployed, the containers are packaged with an application as well as all binaries and libraries it requires to run and perform efficiently on the OS. Application Software running in containers need to be compatible with the host system's kernel and CPU architecture (Eder, 2016). Some examples of operating-system level virtualisation tools are Open VPN, Solaris and Docker. Figure 4.5 presents the architecture for operating-system level virtualisation using containers.

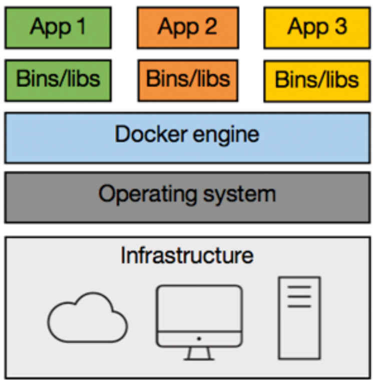


Figure 4. 5 – Operating-System Level Virtualisation Architecture (Infoworld, 2019)

Hypervisor-based and Container-based Virtualisation have their various trade-offs; hence they are used to achieve different goals. Based on these trade-offs, they both have individual strengths and weaknesses. However, they also have some similarities; both are virtualisation technologies and they involve some levels of abstraction from the hardware. In addition, both types of virtualisation can be easily migrated to and they enable better use of resources, which results in reduced computational overhead; saving costs and energy (Eder, 2016). Table 4.4 presents a detailed comparison of both types of virtualisation technologies.

Table 4. 4 – Comparison of Hypervisor-Level and OS-Level Virtualisation

| **Hypervisor-Level Virtualisation** | **Operating-System Level Virtualisation** |
|---|---|
| Emulates the underlying physical hardware and creates new virtual hardware for each guest OS. This implies that resources (RAM, CPU, Storage space) of the physical hardware are shared with the virtual hardware. | No emulation of physical hardware, it uses kernel features which creates an environment that is isolated for the processes. Hence, they utilise the resources of the host system. |
| Duplication of functionalities exist due to the sharing of physical hardware of the host system, this reduces the performance of the system. | Duplication of functionalities do not exist, so it provides improved performance rate. |
| It provides a high level of isolation (complete isolation) of applications running on different guest operating systems. | It provides a weak level of isolation because the base operating system is shared among several applications. Hence, it provides an awareness of all processes (applications) running on the base machine. |
| Increased runtime and storage overhead. | Reduced runtime overhead due to the absence of hypervisors. Storage overhead is also reduced since applications share same operating system. |
| Various operating systems may share hardware virtualised resources, and they can run on same physical device | All instances (containers) share a single operating system. |
| Even though deployments, provisioning, backups etc. are facilitated, this type of virtualisation consumes more resources when compared to operating system level virtualisation | This is a lightweight alternative to the hypervisor-based virtualisation. Deployments, provisioning, backups etc. are relatively quicker. |
| It provides a higher level of flexible as different operating systems can be deployed on a single host. | Less flexibility as all containerised applications run on the same operating system. |
| Some examples are Xen, Oracle VM, VMware workstation, VirtualBox. | Some examples are Open VPN, Solaris, Docker, OpenVZ. |

Bearing in mind the nature of the web with respect to its scale and the comparison between hypervisor-level and operating-system level virtualisation, leveraging both techniques provides better results. Benefits of operating-system level virtualisation such as reduced runtime overhead, reduced storage overhead, reduced computational costs and increased application agility are all vital for holistic semantic annotation as well as the ability to run multiple virtual images from the same physical server. These technologies become more important based on the scale required for a holistic semantic annotation solution. A public-facing web application on the Internet with the potential to attract hundreds of millions of traffic daily will require the utmost benefits of overhead reduction in every area applicable. The next section, 4.3 focuses on ‘Containerisation’ and how it greatly facilitates ‘Microservices’.

## 4.3 Containerisation and Microservices

As described in section 4.2, Containerisation refers to the encapsulation of a software application in a container with all the binaries and libraries that it requires to run on any host machine’s operating system, hence possessing the ability to be deployed on another operating system running on a different host without the need for code changes or re-configuration (Kang et al., 2016). Containerisation and Microservices have a seamless interaction. Containerisation greatly enables running and deploying distributed microservices-based web applications without launching the full virtual environment in which the application is deployed. The containerisation of a microservices-based web application allows all the microservices to run on the same host and to access the same operating system (Fazio et al., 2016; Kang et al., 2016). In addition, it allows such web applications to be managed using their unique namespaces and requiring relatively minimal computing resources in comparison with utilising only hypervisor-level virtualisation. With the nature of microservices-based applications implying that each microservice can be updated, developed, modified and equipped with new features without affecting the other ones; the ability to deploy each within a container greatly facilitates their independence while maintaining the communication between them, thereby enhancing the “loosely-coupled and highly cohesive” key principle for microservices-based applications (Newman 2015; Richards, 2015). Besides, containerisation does not present the overhead problems that are common with other virtualisation management options (Guo et al., 2016). As a result, it supports more applications all within the same infrastructure. Figure 4.6 demonstrates the use of containerisation technology to deploy different microservices of the same application in a polyglot environment whereby the microservices run on varying technology stacks.

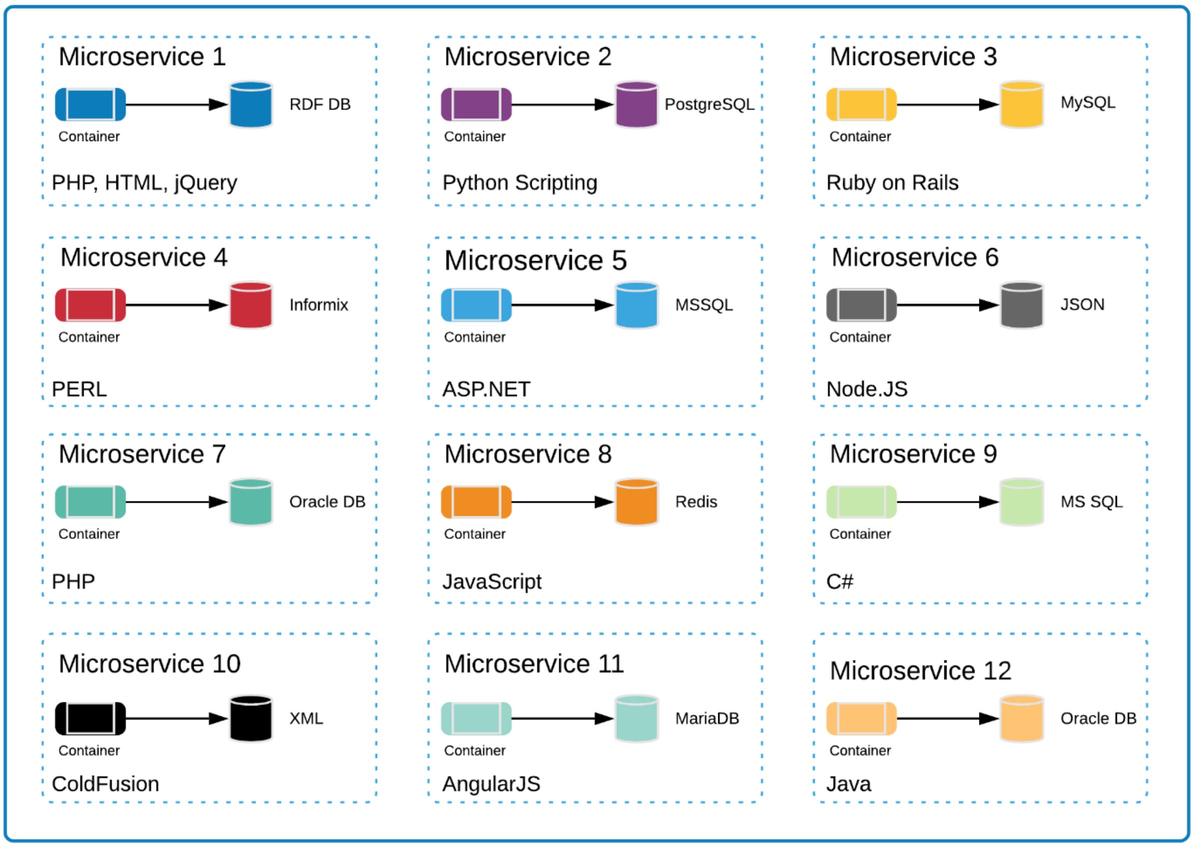


Figure 4. 6 – Polyglot Programming for Microservices

Furthermore, it supports portability across different platforms and operating systems, as containerised microservices can run on different platforms and even across multiple cloud environments, maintaining an inter-process communication medium via a messaging channel such as is available with Apache Kafka, ZeroMQ and RabbitMQ (Eder, 2016). In addition, the containerisation of microservices-based applications also greatly increases application agility. With the use of methodologies such as the twelve-factor app to build software as a service, organisations can quickly respond to customer feedback and carry out rapid iterations for deploying software updates and increasing their time-to-market (Pahl et al., 2017). Containerisation has helped to enable agility within software development lifecycles. Often, microservices-based web applications contain clusters of containerised service instances with some of their key characteristics being the ability to withstand fault, availability, automatic scalability based on user demand and ability to disperse geographically (Khan, 2017). This advantage is evident in the rapid rise of the adoption of containerisation and microservices for developing and deploying business-critical applications. The isolation of an application and the components that it depends on to run effectively in a self-contained unit operating in the cloud is a very significant benefit of containerisation and microservices which will greatly foster holistic semantic annotation. The individual requirements of the holistic semantic annotation can be developed into capabilities based on microservices architecture and deployed to run from a cloud platform with the ability to meet user demands either independently or as a collection of two or more; delivering application functionality in a holistic manner.

## 4.4 Container Orchestration Technology

With the emergence of containerisation technologies came the need to develop orchestration platforms for managing the deployment of container clusters, hence the term "Container Orchestration" (Paladi et al., 2018). Container Orchestration builds on the concept of "Orchestration" in computing which defines the process of automation for configuring, co-ordinating and managing a collection of computing systems and software harmoniously (Khan, 2017). Based on these, Container Orchestration refers to the process of automating the deployment, scaling, monitoring and management of containerised applications within a cloud computing environment, integrating and managing containers at enterprise level (Rodriguez & Buyya, 2019). With Container Orchestration, several containers can be managed as an entity; aiding their availability, networking and scaling while simplifying their overall management. Some key functions of a container orchestration platform include cluster state management and scheduling, high availability and fault tolerance, security, service discovery and monitoring among others (Khan, 2017). Orchestration tools utilise containerised application's configuration file, usually in JSON or YAML format to execute processes for the application. The configuration file is machine-readable and defines rules for the orchestration platform with respect to the successful running of each containerised application. These includes rules for establishing communication between containers, mounting storage volumes, locating container images, storing container logs and lots more (Rodriguez & Buyya, 2019). Based on these rules, containers are deployed onto host machines typically in groups that are replicated. On further request to deploy a new container into the cluster, the container orchestration platform schedules the deployment by an assessment of the most suitable host based on computing resources needs of the container in relation to the available resources across the host machines within the cluster (Hoque et al., 2017). Upon successful deployment, containers lifecycles are managed based on the configuration data. This management usually includes a self-healing mechanism for the containers if found to be corrupt or destroyed. The self-healing mechanism could require an automatic re-start or re-scheduling (Paladi et al., 2018). Figure 4.7 presents a simplified illustration of a container orchestration platform.

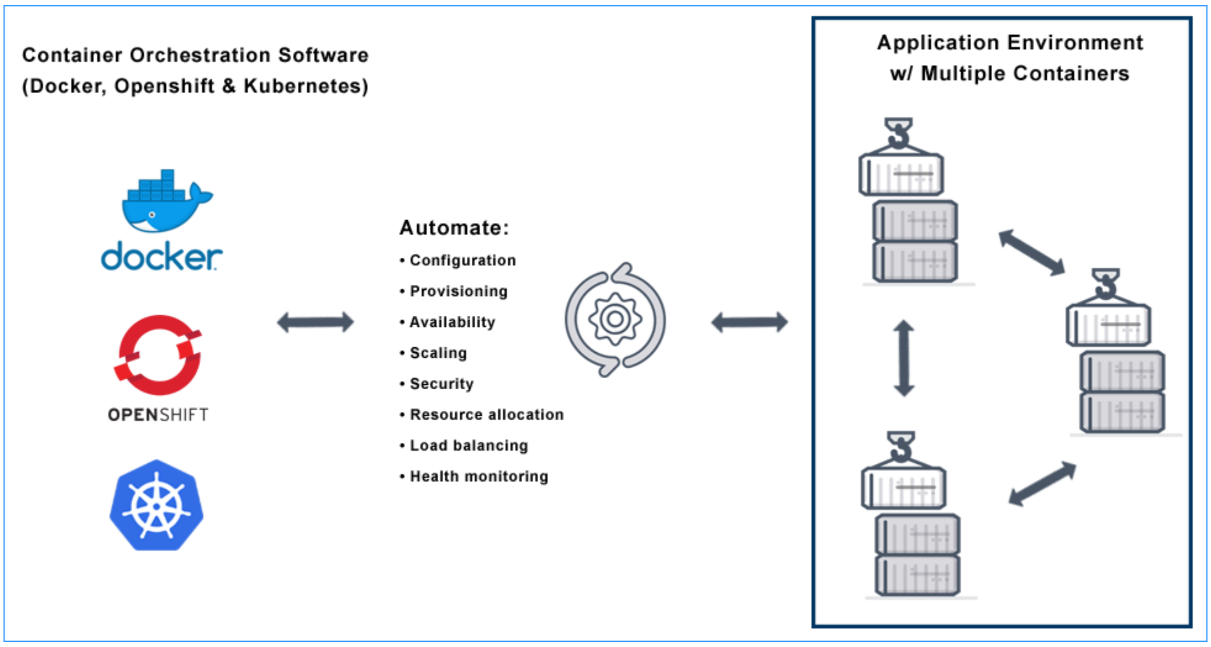


Figure 4. 7 – Simplistic View of Container Orchestration Cloud Environment (Avi Networks, 2019)

Furthermore, the design of container orchestration systems includes for scheduling workloads of containerised applications varying from one to many types. There are varying features and requirements for different applications. While some require a very high level of availability and long-running jobs, others could require mission-critical batched jobs or latency-sensitive jobs for instance (Rodriguez & Buyya, 2019). In addition, an application could require a combination of two or more of these. According to Khan (2017), container orchestration platforms make provision for an enterprise-level framework to integrate and manage containers at scale. With the capability to leverage hypervisor-level virtualisation from the virtual host layer and operating system-level virtualisation from the container layer, container orchestration provides a high level of agility and flexibility for applications (Hoque et al., 2017). Some of the commonly used container orchestration platforms include Kubernetes originally developed by Google and now open source, Elastic Container Service by Amazon, Mesosphere by Apache, Docker Swarm, Azure from Microsoft supporting Kubernetes, Rancher and Nomad (Khan, 2017).

## 4.5 Continuous Integration Mechanism

A continuous integration mechanism refers to the ability of delivering an automated and continuous stream of updates to a software based on a workflow. Such updates can be as often as possible; including several times daily. Container Orchestration Engines can facilitate this by an integration with third-party solutions for the purpose (Mun, 2017). This is believed to be a required facility for holistic semantic annotation of web documents based on the dynamic and ever-changing nature of web documents and the web environment itself. Updates for software to ensure and maintain the required level of accuracy and consistency between components of a holistic semantic annotation solution (such as ontologies, RDF graph databases, web documents and annotation data) requires an automated means that will facilitate the persistent nature of such updates.

The continuous integration workflow merges several sets of local changes to a shared code repository, allowing multiple developers define their respective services that will be used during application development. The workflow permits different developers to develop and update their services concurrently. These are then merged to the source code repository. The source code repository is a storage location for hosting the code developed when updating the microservices. Predefined configuration files in the source code repository are used to create new microservice images after updates have been pushed to the repository. For example, with docker containers, the YAML "docker-compose.yml" configuration file instructs the tools present within the orchestration environment to create a new image of the containerised microservice, establish networks for communication and a storage location for storing log files generated during orchestration (Stahl & Bosch, 2016). The new microservice image is then tested to identify any errors.

Once the microservice images pass the test, the source codes are combined with their respective dependencies to create a new instance. The new instance is a runnable file of the containerised application being updated for re-deployment. Like the microservice image, the health of the new instance must be tested. Testing identifies problems or errors with the application functionality or with the integration process. These tests need to be automated, based on a test plan developed by the development teams, and their main purpose is to validate the software's behaviour and functionality (Rathod & Surve, 2015). The time used for testing the health of an instance depends on the its size, complexity and scope of the testing process. A healthy instance is one whose code is bug-free, and the developed functions work as intended. In some cases, the test results may show an unhealthy instance in which case, the new instance would be discarded, and the previous instance is left to continue running while identified bugs and errors in the new instance are fixed. Once they are fixed, the new instance is restarted, and it is subjected to the same testing process. The need for testing a new instance until it is 100% healthy is to ensure that reproducible errors do not reach the targeted end users. Most large-scale projects run tests using several stages. These can start with a smoke test, which is designed to perform sanity tests for end to end integration of the entire project and from a user's perspective (Sachdeva, 2016). Testing the health of new instances expose issues that developers were not aware of when writing codes for building the software. It is necessary for the workflow to provide quick feedbacks for developers to maintain the stream. Figure 4.8 depicts the persistent deployment mechanism for a continuous stream of updates for containerised microservices applications.

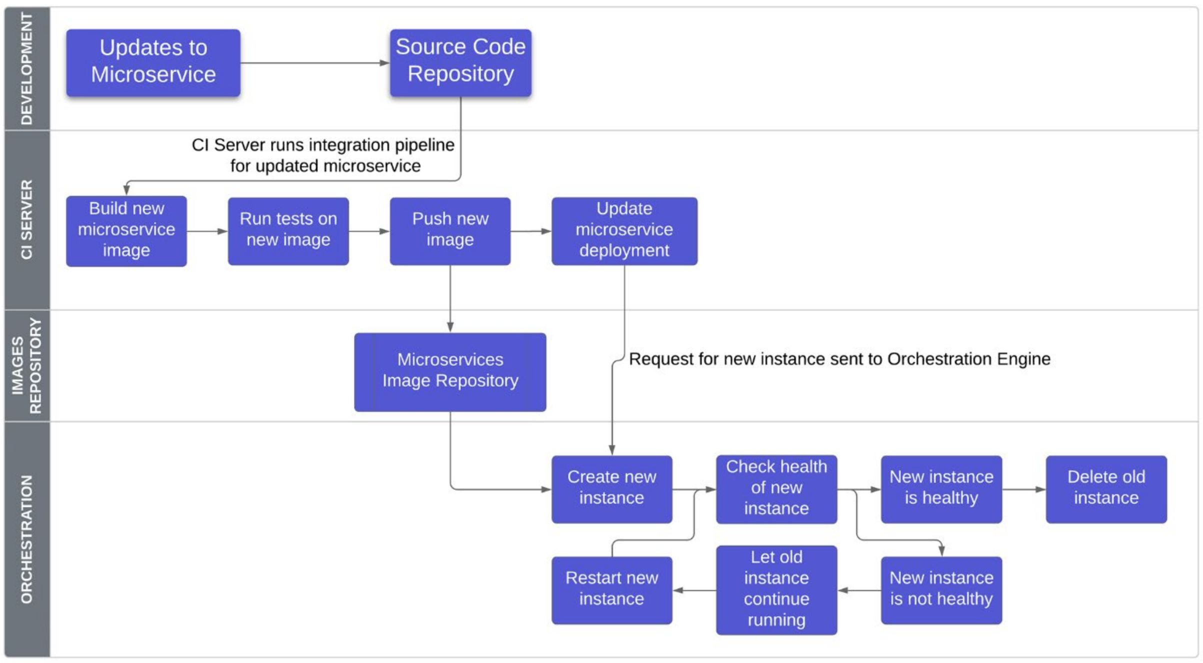

DEVELOPMENT
Updates to Microservice
Source Code Repository
CI SERVER
CI Server runs integration pipeline for updated microservice
Build new microservice image
Run tests on new image
Push new image
Update microservice deployment
IMAGES REPOSITORY
Microservices Image Repository
Request for new instance sent to Orchestration Engine
ORCHESTRATION
Create new instance
Check health of new instance
New instance is healthy
Delete old instance
Restart new instance
Let old instance continue running
New instance is not healthy

Figure 4. 8 – Continuous Integration Mechanism for Holistic Semantic Annotation

## 4.6 Cloud Computing Maturity Levels for Holistic Semantic Annotation

Having reviewed and analysed different factors that impact on application deployment in the cloud with an evaluation of each, Table 4.5 presents a summary of the factors and their potential impacts with respect to application deployment in the cloud.

Table 4. 5 – Determinant Factors for Holistic Semantic Annotation Deployment in the Cloud

| **Factor** | **Impact** |
|---|---|
| Software Architectural Pattern | A higher level of decomposability of software architectural components fosters software agility by being lightweight and requiring lesser computing resources to meet user demands. It also greatly enhances automatic scaling. |
| Virtualisation Pattern | A software's ability to leverage both hypervisor-level and operating system-level virtualisation increases its performance and efficiency |

|  |  |
|---|---|
| Containerisation and Microservices | Containerisation greatly enables microservices by encapsulating them with the required files and binaries needed for them to be independently deployable |
| Container Orchestration | It provides a platform deploying containerised application; automating their deployment, scaling, monitoring and management which subsequently enhances software agility. |
| Persistent Deployment Mechanism | It provides an automated mechanism for deploying updates and new features for software, reducing "time-to-market" for software products. Its integration with microservices architecture produces optimal results. |

The different factors summarised in Table 4.5 form the basis for defining different patterns for holistic semantic annotation deployment in the cloud. These are described in the following sections.

### 4.6.1 Cloud-Based Monolithic Maturity Level

The Cloud-Based Monolithic Maturity Level depicts a scenario whereby an application is hosted in a cloud computing environment either by migrating an existing application from a non-cloud environment or by hosting a newly developed application directly in a cloud environment. In either case, no changes to the application's code or architecture is required and such applications do not fully leverage any of the features illustrated in Table 4.5. Such applications are developed using a monolithic or n-tier software architectural pattern which hinders its ability to scale maximally as well as limits its capability based on the several drawbacks identified for such architectural patterns in section 4.1.

Furthermore, while the application can be configured to leverage hypervisor-level virtualisation such as by deploying it on a virtual host with automatic scaling capabilities, it does not leverage operating system-level virtualisation which provides a higher level of software agility and increases efficiency and productivity when combined with hypervisor-level virtualisation. In addition, such applications are not containerised, hence does not require a container orchestration platform. Furthermore, based on the nature of the monolithic or n-tier software architectural pattern, automating the application lifecycle for continuous integration and delivery is quite cumbersome as the entire application might need to be re-deployed after minimal updates. Hence, the ability to implement code changes and deploy new software updates would be greatly hampered. With this pattern, applications are mainly leveraging cloud computing infrastructure layer with little or no leverage from both platform and software layers. This is depicted with Figure 4.9.

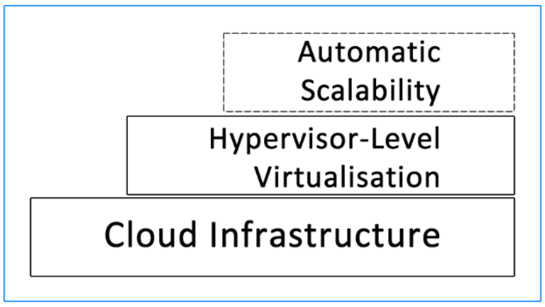


Figure 4. 9 – Cloud-Based Monolithic Maturity Level for Holistic Semantic Annotation

### 4.6.2 Cloud-Based Microservices Maturity Level

This is like the Cloud-Based Monolithic Maturity Level. However, it is based on a microservices software architecture which is the difference between both. While the software architectural pattern has the potential to boost its performance, it still cannot fully leverage the other factors described in Table 4.6. Microservices provide the greater level of software agility based on the architectural pattern's potential to leverage several other technologies that monolithic architectures might not be able to leverage or might not leverage as much as microservices facilitates. Hence, deploying a microservices-based application on a virtual host directly without containerisation and orchestration capabilities does not take advantage of the microservices architectural pattern. This is depicted with Figure 4.10.

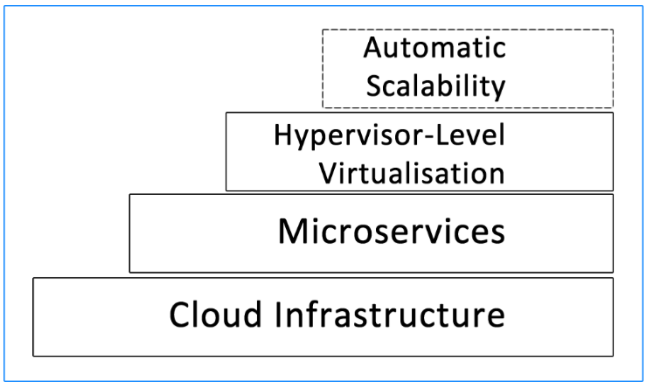


Figure 4. 10 – Cloud-Based Microservices Maturity Level for Holistic Semantic Annotation

### 4.6.3 Cloud-Optimised Maturity Level

The Cloud-Optimised Maturity Level has similarities with the Cloud-Based Monolithic in that they are both based on a monolithic or n-tier software architectural pattern. However, unlike the Cloud-Based, the Cloud-Optimised is modified (or optimised as the name implies) to ensure a higher level of leverage from the cloud computing environment. While the architectural pattern remains unchanged, the application is 'optimised' by leveraging containerisation which enables it to utilise both hypervisor-level and operating system-level virtualisation techniques. This implies a minimal code change for such applications to implement the encapsulation of the application in software containers; otherwise referred to as containerisation. Additionally, due to the application containerisation, the containers that ensue can benefit from deployment in an orchestration platform. As described in section 4.4, container orchestration fosters automation for deploying, scaling, monitoring and managing applications. The orchestration platform has the capability to provide several managed services for such applications like auto-scaling, job scheduling, configuration management and lots more. This presents a significant leap from the Cloud-Based Monolithic.

However, due to the limitations with the software architectural pattern, the operating system-level virtualisation in this case requires the entire application to be automatically scaled when demanded rather than scaling just a component of the entire application that requires it, thereby leading to an increase in computational overhead by consuming more computing resources as well as requiring more processing time, hence reducing software agility. Furthermore, the software architectural pattern still impacts on the application automation lifecycle as is the case with the Cloud-Based Monolithic by requiring a re-deployment or re-compiling of the entire application even for minimal changes which impacts on the frequency at which software updates can be implemented and can lead to extended downtimes when an update fails or produces unexpected results. So, while both infrastructure and platform layers of cloud computing are leveraged with this maturity level, the software architectural pattern would still inhibit maximum application performance. Figure 4.11 depicts the factors leveraged by this pattern.

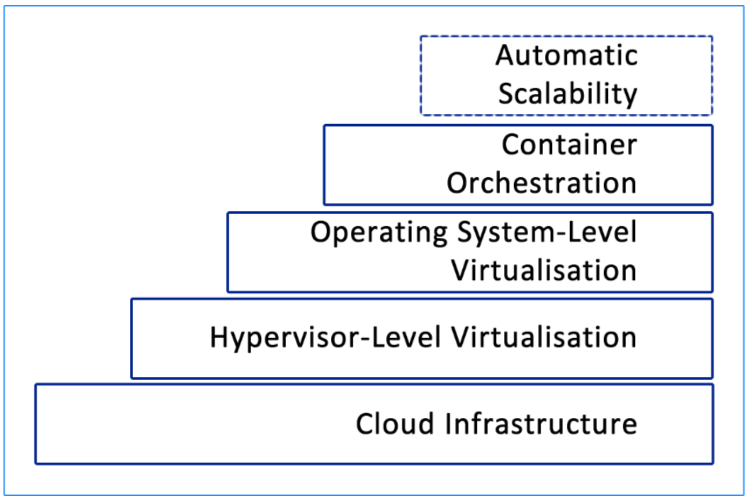

Automatic Scalability
Container Orchestration
Operating System-Level Virtualisation
Hypervisor-Level Virtualisation
Cloud Infrastructure

Figure 4. 11 – Cloud-Optimised Maturity Level for Holistic Semantic Annotation

### 4.6.4 Cloud-Native Maturity Level

The Cloud-Native Maturity Level, as the name suggests, is based on developing applications that are well-suited for maximising cloud computing benefits by deploying a development process which natively ensures cloud computing characteristics and benefits are fully maximised. According to Wilder (2012), the key to maximising cloud computing benefits for software applications does not lie with the cloud infrastructure alone, but also with the architecture of a software deployed in the cloud. This opinion is elaborated on by Leymann et al. (2016) and Gilbert (2018). The basis of the opinion focuses on the development of applications utilising a software architectural pattern that enables the decomposition of an application into loosely-coupled but highly-cohesive modules such that each module is independently deployable, scalable, updateable and lots more; which are features central to the fundamental principles of microservices software architecture (as described in sections 4.1 and 4.2). With the microservices architecture, it implies that automatic scaling is at the level of each microservice rather than scaling an entire application due to the auto-scaling needs of a single module, thereby making judicious use of computing resources and limiting the computational overhead involved. Leymann et al. (2016) provides a detailed definition of cloud native where the term implies using containerised open-source software stacks such that the development process assigns every part of an application its own container to ensure each part is dynamically orchestrated. Figure 4.12 illustrates this maturity level.

So, with containerised microservices, leveraging both hypervisor-level and operating system-level virtualisations with an orchestration platform to deploy them, the Cloud-Native provides a significant leap from the Cloud-Optimised. However, while it fosters such potential benefits for holistic semantic annotation solution in the cloud, the automation of application lifecycle by means of a continuous integration and delivery is very vital for a solution requiring the level of dynamism, agility and efficiency as semantic annotation for web documents. While the Cloud-Native has the potential to deliver very significantly, it will still present bottlenecks without a continuous integration mechanism. Based on this, an enhanced version, referred to as Cloud-Driven is proposed in the next section.

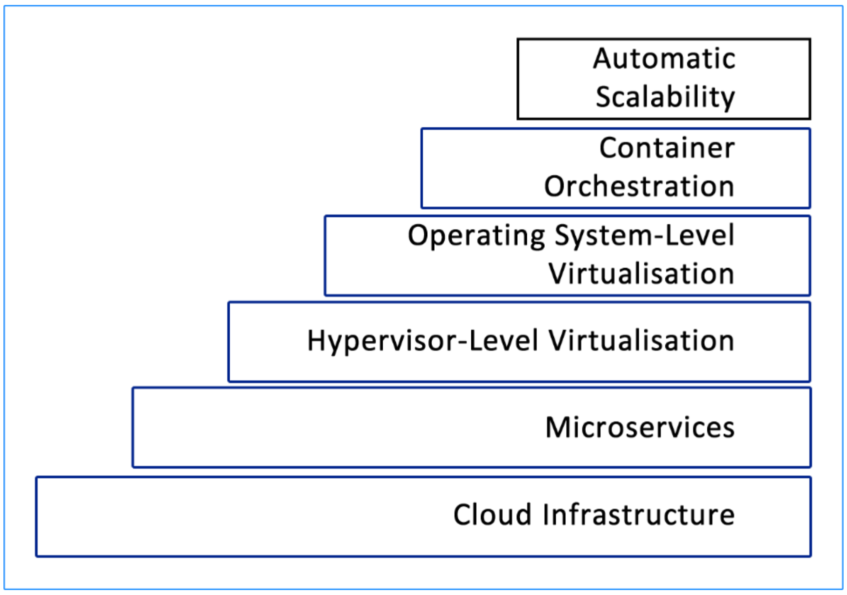


Figure 4. 12 – Cloud-Native Maturity Level for Holistic Semantic Annotation

### 4.6.5 Enhanced Cloud-Native: Cloud-Driven Maturity Level

The proposed "Enhanced Cloud-Native Maturity Level", defined as "Cloud-Driven" provides all the "Cloud-Native" features as well as a "Continuous Integration Mechanism" as described in section 4.5. It is believed that a holistic semantic annotation solution developed and deployed based on this model would be equipped with the necessary capabilities to deliver automated semantic annotation to web documents online, real time, continuously and holistically by ensuring that updates across different microservices are agile in nature, seamlessly integrated and dynamically synthesised with the entire system.

Generically, some of the automated processes would include the initiation of code builds, testing procedures, and deployment. Specifically, required software updates due to evolving ontologies (by means of ontology updates, upgrades, population, etc.), optimisation of annotation data, web documents evolution and several other mechanisms within the holistic sphere of a semantic annotation solution can be quickly implemented without impacting on the overall functionality of the solution, hence, providing a complete lifecycle of semantic annotation process workflow and management for web documents at large. The integration of a continuous integration mechanism to foster a transformation from "Cloud-Native" to "Cloud-Driven" is believed to have the potential for delivering application solution that is entirely driven by cloud computing with a full automation lifecycle; one that is envisioned as the required development methodology for web-scale semantic annotation effectively and efficiently. This implies that with "Cloud-Driven", the following features are implemented:

- Development of modularised functionalities for the holistic semantic annotation requirements using microservices architecture.
- Encapsulation of the modularised "holistic semantic annotation" functionalities in software containers (otherwise referred to as Containerisation) for operating system-level virtualisation using container software such as Dockers.
- Configuration of the containerised functionalities for both hypervisor-level and operating system-level virtualisation.
- Orchestration for automating deployment, scaling, monitoring and management of the containerised functionalities using container orchestration software such as Kubernetes, Docker Swarm or Amazon Elastic Container Service, among others.
- Application Automation Lifecycle for continuous integration and delivery using a continuous integration mechanism.

As described in section 4.5, the automation is not only applicable to software but to hardware as well. This is through automation for rapidly provisioning cloud infrastructure using document templates containing machine-readable configuration data rather than physically configuring hardware or using graphical configuration tools which require human intervention and subsequently, a level of potential errors. The template files can be developed using data portable formats such as YAML or JSON. Figure 4.13 illustrates the enhanced cloud-native; the cloud-driven maturity level for holistic semantic annotation.

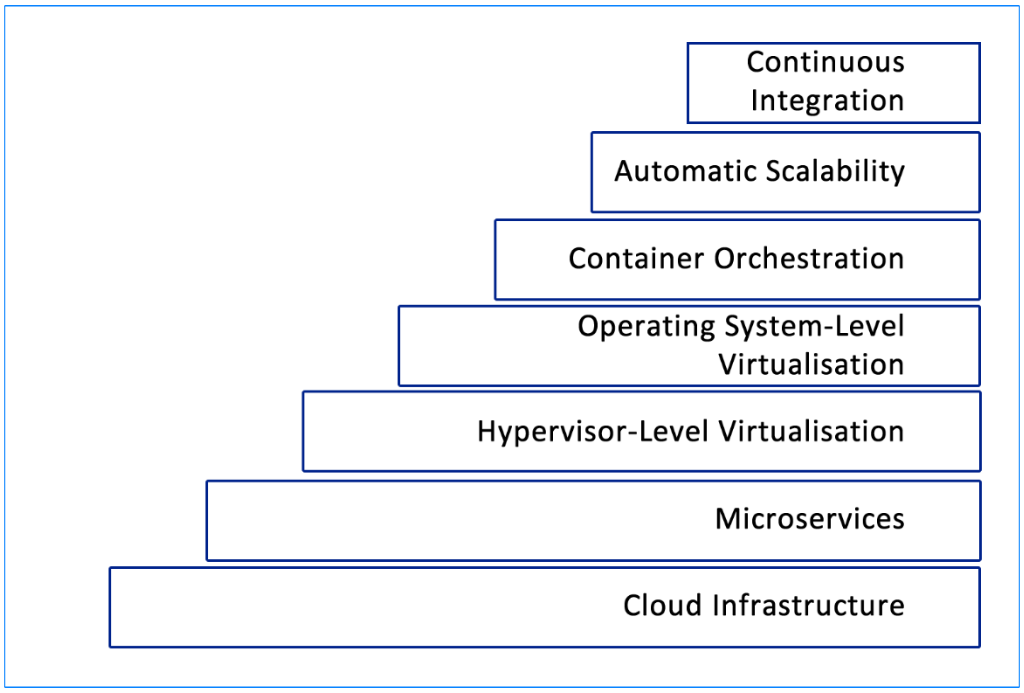
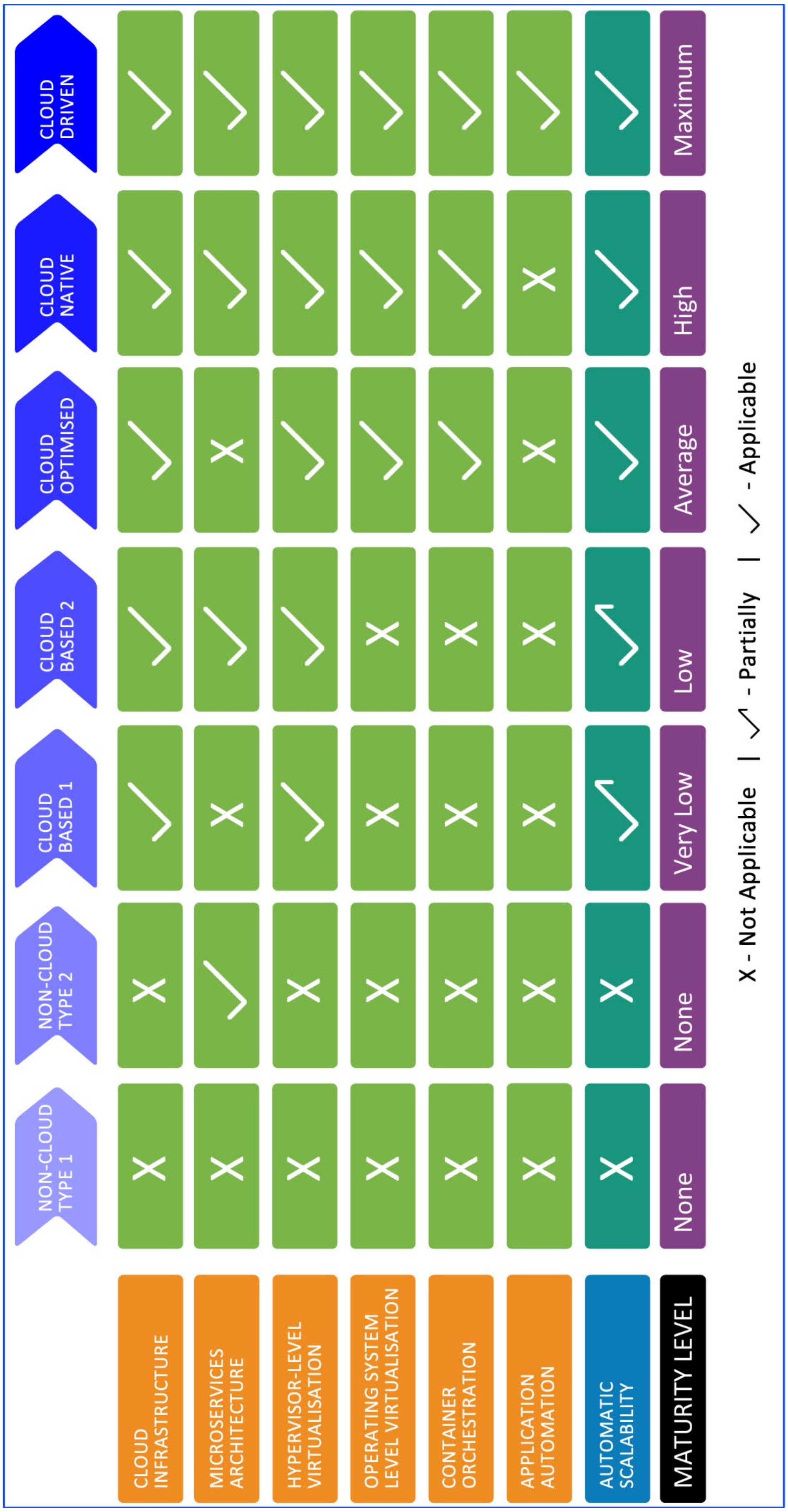


Figure 4. 13 – Cloud-Driven Maturity Level for Holistic Semantic Annotation

## 4.7 The Proposed Maturity Model

The different deployment patterns for holistic semantic annotation presented in section 4.6 define various deployment approaches in which a set or requirements can be evaluated against each pattern to select an appropriate approach; which forms the basis for developing and selecting "cloud-driven" as the appropriate choice for holistic semantic annotation. Due to a perceived nature that will need evaluating requirements against the different patterns to guide a selection, they are integrated into a single unit for this purpose. This is in line with the concept of models in Design Science Research, in which they are described as a set of propositions in a problem/solution scenario; defining how things are or should be (March & Smith, 1995). The different patterns provide solutions for application deployment in the cloud based on varying requirements. Hence, Figure 4.14 presents a Cloud Computing Maturity Model for Holistic Semantic Annotation.



| | NON-CLOUD TYPE 1 | NON-CLOUD TYPE 2 | CLOUD BASED 1 | CLOUD BASED 2 | CLOUD OPTIMISED | CLOUD NATIVE | CLOUD DRIVEN |
|---|---|---|---|---|---|---|---|
| CLOUD INFRASTRUCTURE | X | X | ✓ | ✓ | ✓ | ✓ | ✓ |
| MICROSERVICES ARCHITECTURE | X | ✓ | X | ✓ | X | ✓ | ✓ |
| HYPERVISOR-LEVEL VIRTUALISATION | X | X | ✓ | ✓ | ✓ | ✓ | ✓ |
| OPERATING SYSTEM LEVEL VIRTUALISATION | X | X | X | X | ✓ | ✓ | ✓ |
| CONTAINER ORCHESTRATION | X | X | X | X | ✓ | ✓ | ✓ |
| APPLICATION AUTOMATION | X | X | X | X | X | X | ✓ |
| AUTOMATIC SCALABILITY | X | X | √ | √ | ✓ | ✓ | ✓ |
| MATURITY LEVEL | None | None | Very Low | Low | Average | High | Maximum |

X - Not Applicable | √ - Partially | ✓ - Applicable

Figure 4. 14 – Cloud Computing Maturity Model for Holistic Semantic Annotation

From the model presented in Figure 4.14, this research defines the name; "**CloudSea**" which stands for **Clou**d-**D**riven **Se**mantic **A**nnotation based on the "cloud-driven" pattern of the model. The technique for application deployment described and analysed by the pattern will be utilised for the design and development of the holistic, automated semantic annotation solution: **CloudSea**.

## 4.8 Software Architectural Layers for CloudSea

From the review and analysis of Microservices Software Architecture in section 4.1 of this chapter and its comparison with SOA and monolithic architectures in section 4.2, it is adopted for use in this research. From the comparisons in section 4.2, it can be observed that it provides better software agility and performance as compared to the others. It also has enormous economic and technical benefits due to reduced time to market and the ease of making and deploying software updates. Furthermore, and very importantly; the key to maximising cloud computing benefits does not lie with the cloud infrastructure alone, but also with the architecture of a software deployed in the cloud (Gilbert 2018; Wilder 2012). The Microservices Software Architecture is natively for the cloud as its fundamental principles help maximise cloud computing resources and hence, benefits. Figure 4.15 presents the CloudSea Microservices Software Architectural Layers with a description of the SaaS layer. The PaaS layer has been covered in sections 4.3 to 4.5 of this chapter while the IaaS layer is based on public cloud infrastructures.

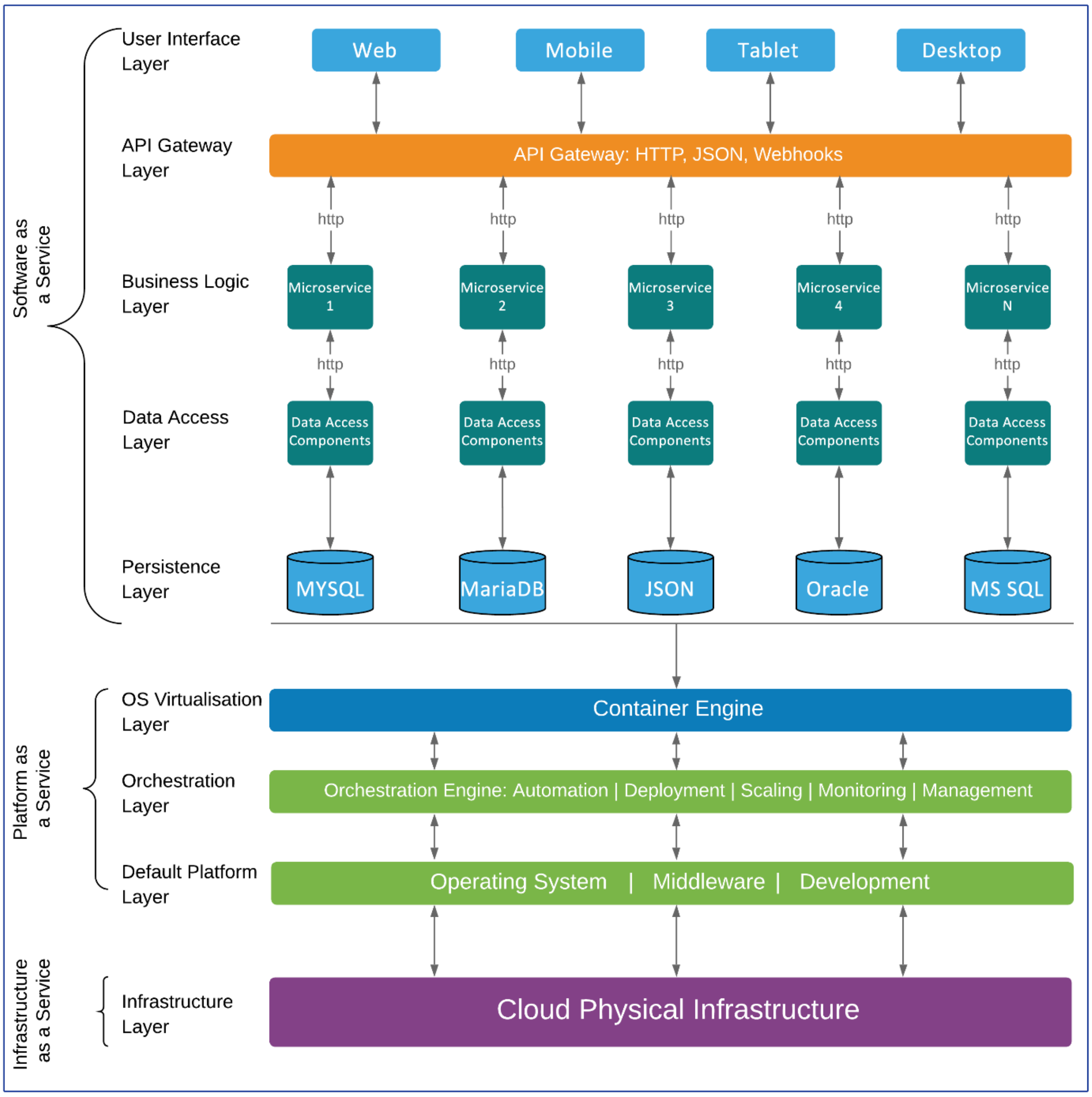


Figure 4. 15 – Software Architectural Layers for CloudSea

## 4.8.1 The User Interface Layer

The User Interface Layer is a critical element of any software architecture as it allows for effective communication between the physical components and the users of the system. It provides interfaces via personal computers, laptops, mobile devices, kiosks and lots more; condensing and formatting data for the application users and helping with the acquisition and validation of data from them (Richards, 2015). It provides access to the various functionalities implemented within an application. For web-based applications, this layer usually comprises of a web client software known as web browsers to access applications running on the web. The web browsers capture data from the users and transfer same to the application as a request. The response to the request is also presented to the user via the same medium. Some popularly used technologies for developing the user interface layer includes HTML (HyperText Markup Language), CSS (Cascading Style Sheets) and JavaScript. With Microservices, this layer is ideally the only public-facing component of the entire application; presented as an interface for interacting with the several microservices within the application. With CloudSea, this layer would be web-based, hence, accessible via web browsers for access to different application functionalities such as annotation on-the-fly, annotation data generation for web documents semantic annotation and lots more.

### 4.8.2 The API Gateway Layer

This layer helps with the implementation of one entry point for all users via the User Interface Layer. Different types of requests received via the User Interface Layer are channelled through the API Gateway Layer to the different microservices that will process each of the requests. A common technique for the channelisation is known as ‘Service Discovery’. This involves the use of a Service Registry to keep the identities of the different microservices and their locations within the system, in terms of their IP addresses in a networked environment. So, once a request or series of requests are received, the address of the microservice to process each one is looked up and the requests are channelled accordingly (Zhao et al., 2018). Furthermore, the API Gateway is usually utilised for the implementation of security mechanisms such as authentication and authorisation for users (Joshi et al., 2017). It helps with the insulation of clients from the details on the decomposition of the application into microservices and brings about a reduction in the number of requests or the number of cycles to search through for a request (Newman, 2015). That is, users can obtain data from several microservices in one cycle, thereby enhancing less overhead and the improvement of user experience. The hosting of all the API services characterised by clear business functionality is through the API Gateway (Zhao et al., 2018). Figure 4.16 further illustrates the role of an API Gateway Layer in a microservices architecture.

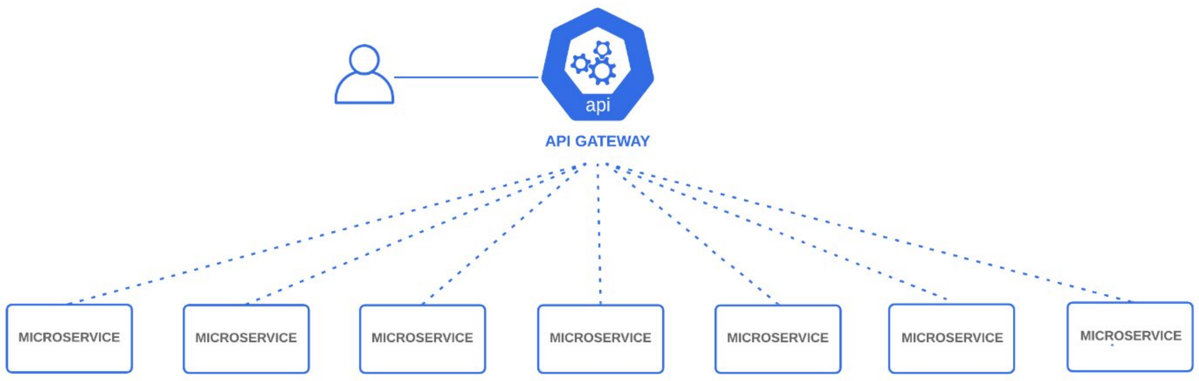


Figure 4. 16 – API Gateway Implementation for Microservices

### 4.8.3 The Application Logic Layer

This layer is also referred to as the Business Logic Layer and is made up of the components representing the core of the application and helps with the implementation of business rules. It is the major aspect of an enterprise application, presenting various rules and activities of a given business domain that defines the relationship and properties of various business data. It forms the basis of tackling various problems that the entire application aims to solve, consisting of microservices that have limited scopes, concentrate on specific tasks and are independent. Some other types of components on this layer can be classes, functions, modules and so on (Autili et al., 2019). For CloudSea, this layer comprises of the microservices that make up the entire application. Requests received via the API Gateway Layer are directed to the microservices based on the user request type. In this regard, the microservices receive pieces of information from the API Gateway and manipulate the information as required. After manipulation, the results are relayed back to the User Interface Layer via the API Gateway for end users. While some web application frameworks merge this layer with the User Interface Layer, enterprise and web-scale solutions often implement them separately (Hnatkowska & Kasprzyk, 2009). So, API Gateways act as an intermediary between the Application Logic Layer and the User Interface Layer. Several technologies exist for implementing this layer; from open source standards to proprietary solutions. The PHP (Hypertext PreProcessor) programming language is intended for use primarily for this layer in CloudSea, alongside any other languages or standards which might be more appropriate for a specific microservice. PHP is a very common scripting language for web applications. In addition, it forms the bedrock of several web application frameworks popularly used in IT today for web development. Examples of such frameworks include CodeIgniter, Zend, Yii and Laravel.

### 4.8.4 The Data Access Layer

The Data Access Layer is critical for any application since data processing and retrieval are common tasks within an application. Within a Microservices Architecture, the layer consists of microservices data abstraction for performing various functions to ensure persistence of data; focusing on system information structures such as databases, connections, SQL queries, triggers and result sets (Ahmed & Kurnaz, 2019). It offers simplified access to stored data in various forms of persistent storage. The Data Access Layer allows for the creation of user modules with a high level of abstraction such that connectivity to multiple databases either at once or at different stages is possible without any need for application code change in the

Application Logic layer. Functions such as creating, recording, updating and deleting (CRUD) data in the database are typical events within the layer (Taibi et al., 2018). This approach facilitates making queries into the database by abstracting various database calls. This layer also offers a central point for various queries directed to multiple databases making porting various applications to different database systems practical. In other words, the Data Access Layer allows an application to interact with various databases within the system. Therefore, the best designs for the Data Access Layer should consist of appropriate data access technologies that match the type of data to be handled. Similarly, appropriate Data Access Layers allow for easier configuration and maintenance of applications by centralising data access functionalities (Taibi et al., 2018). PDO (PHP Data Objects) is a common PHP library that provides an abstraction layer for PHP, enabling connectivity to different types of databases using the scripting language. Figure 4.17 provides an illustration of the Data Access Layer with PHP programming language and access to multiple database types via a data access layer.

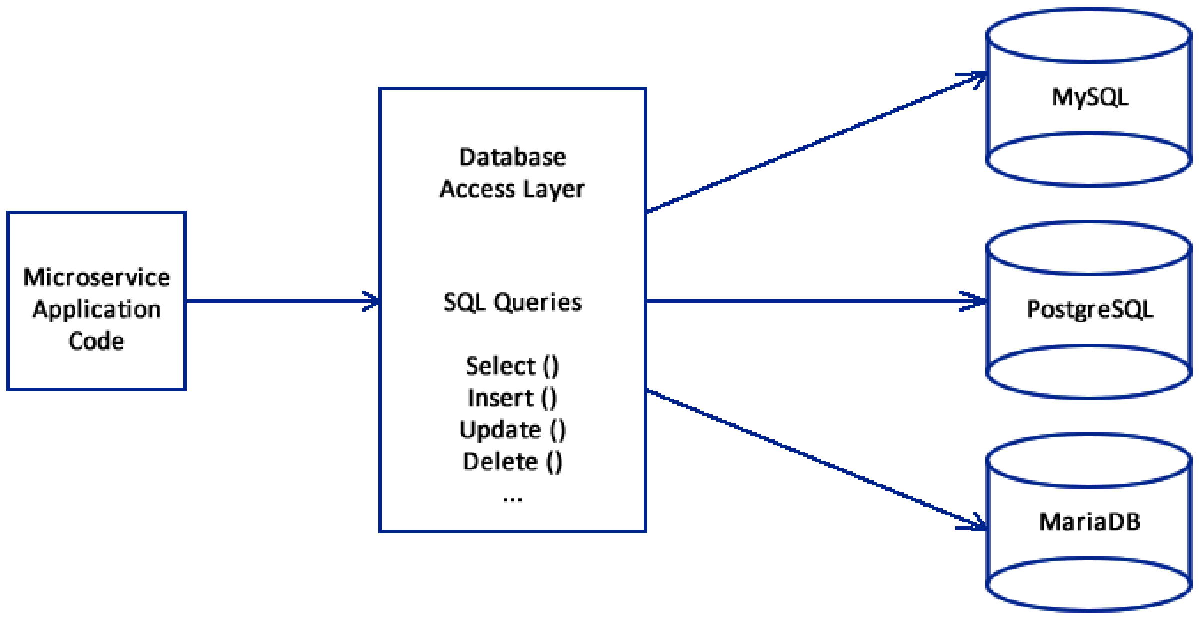


Figure 4. 17 – Data Access Layer with Connectivity to Multiple Databases

### 4.8.5 The Persistence Layer

This is the database layer responsible for the storage of all application data. It is very pivotal to the operations of an application as applications are driven by data. The abstraction defined in the Data Access Layer provides access to data for the Application Logic Layer which in turn manipulates data for several types of transactions (Ahmed & Kurnaz, 2019). Examples of databases commonly used for application development today include MySQL, SQL Server, Oracle, PostgreSQL and lots more. With a microservices architecture, an application can consists of several types of databases, with individual microservices implementing the most appropriate database server for its internal operations. In addition, access to a database is only by the microservice owning it, with others being able to utilise such data via a common messaging channel for the entire application (Richardson, 2015). With CloudSea, the polyglot persistence is inevitable as several types of databases would be required across multiple microservices. Some of these include MySQL and RDF Graph databases. Table 4.6 presents a breakdown of the components for each of the CloudSea software architectural layers discussed above.

Table 4. 6 – Components for CloudSea Software Architectural Layers

| **Architectural Layer** | **Components** |
|---|---|
| User Interface | End Users, Web Client and Front-End Interface |
| API Gateway | Service Registry and Messaging Stream |
| Application Logic | CloudSea Microservices and their components |
| Data Access | PHP Programming |
| Persistent (Data Storage) | MySQL and JSON |

## 4.9 Chapter Summary

In this chapter, a comprehensive analysis towards the development of the "Cloud-Driven" concept for holistic semantic annotation was presented with an in-depth analytical approach that reviewed and critically evaluated all the determinant factors towards developing the concept. While the "Cloud-Driven" concept for semantic annotation is novel, the overall approach towards defining different deployment patterns for holistic semantic annotation in the cloud led to the development of a Cloud Computing Maturity Model which is beneficial specifically in this context as well as other related domains in information systems. Furthermore, the holistic perspective to semantic annotation from Chapter 3 and the 'cloud-driven' maturity level from this chapter will constitute the basis for developing CloudSea; a microservices-based architecture for automated semantic annotation as a cloud service. The design rationale for the architecture will be based on the theory of "Design Patterns" from software engineering. These will be covered in the following chapter.

# Chapter 5: CloudSea – Holistic, Cloud-driven and Microservices-based Semantic Annotation Architecture

In this chapter, a holistic, cloud-driven and microservices-based architecture for automated semantic annotation of web documents is proposed. The concept of the architecture draws from the research efforts described in Chapter 3 (Requirements and Cloud Computing Capability Model for Holistic Semantic Annotation) and Chapter 4 (Cloud-Driven Pattern of Cloud Computing Maturity Model). Furthermore, its conceptual design and presentation of technical solutions adopts the theory of architectural designs based on “Design Patterns and Pattern Language” from Software Engineering. Section 5.1 describes the design rationale for CloudSea. Section 5.2 focuses on the engineering methodology for CloudSea Design Patterns. Section 5.3 describes the CloudSea Pattern Language. In section 5.4, the CloudSea Design Patterns are described in detail while Section 5.5 presents the proposed architecture. The chapter concludes with a summary in Section 5.6.

## 5.1 Design Rationale for CloudSea

The design of CloudSea architecture is based on the concept of “Design Patterns”. Design Patterns are well-documented and structured data for providing solutions to recurring problems within a specific domain (Erl et al., 2015). According to Edwin (2014), Design Patterns stand for solutions to problems arising from the development of software within a context; encapsulating the static and dynamic structure including the collective efforts in software designs thereby ensuring the facilitation of the reuse of successful software architectures and designs. The concept of design patterns was initially created for city planning as well as construction design. Christopher Alexander, an architect, was dissatisfied with contemporary architectural projects and believed that a rigid architectural practice had resulted in the predominance of impracticable resolutions (Alexander, 1977). His motivation was drawn from primeval cultures that had developed towns and building designs for many years and realised that there existed recurrent structures or "patterns." Accordingly, with his associates, he printed 253 patterns for building architecture and urban planning for reuse in new projects (Goodyear & Retalis, 2010). Over the years, the concept has become multi-disciplinary and adopted by other sectors including computing, for software design purposes. Thus, in computing, Design Patterns are commonly arising patterns in design which are recurrent and generalist enough to be written down and called software design constructs which all may usually identify and apply.

A pattern entails defined fields, which include Forces, Context, Problem and Solution. Patterns are utilised in an architectural ‘Context’ and take into consideration a repetitive design ‘Problem’ in such contexts. The pattern takes focus on the ‘Forces’ which the designer encounters prior to explaining a ‘Solution’ – a recommended method to the ‘Context’ that solves the pressures amongst ‘Forces’. According to his book, “A Place to Wait Pattern" the ‘Context’ defines a situation in which people wait for something like a medic's surgical procedure (Alexander, 1977). The concept of ‘Design Patterns’ for physical architectural design has become multi-disciplinary and utilised in ‘Design Patterns’ for software architectural design (Harrer et al., 2017). In Cloud Computing, Design Patterns have been greatly adopted to define an approach for providing solutions or addressing issues relating to the development and deployment of applications in the cloud. They cover potential problems that can be encountered when designing, building and managing cloud applications (Fehling et al., 2014). This is evident from the catalogues of cloud computing design patterns from several leading cloud service providers such as Amazon AWS, Google, Microsoft and IBM which are well proven to be very beneficial for their intended uses. Figure 5.1 presents Amazon AWS cloud computing design patterns which are utilised for designing and deploying varying solutions on the Amazon AWS cloud computing platform.

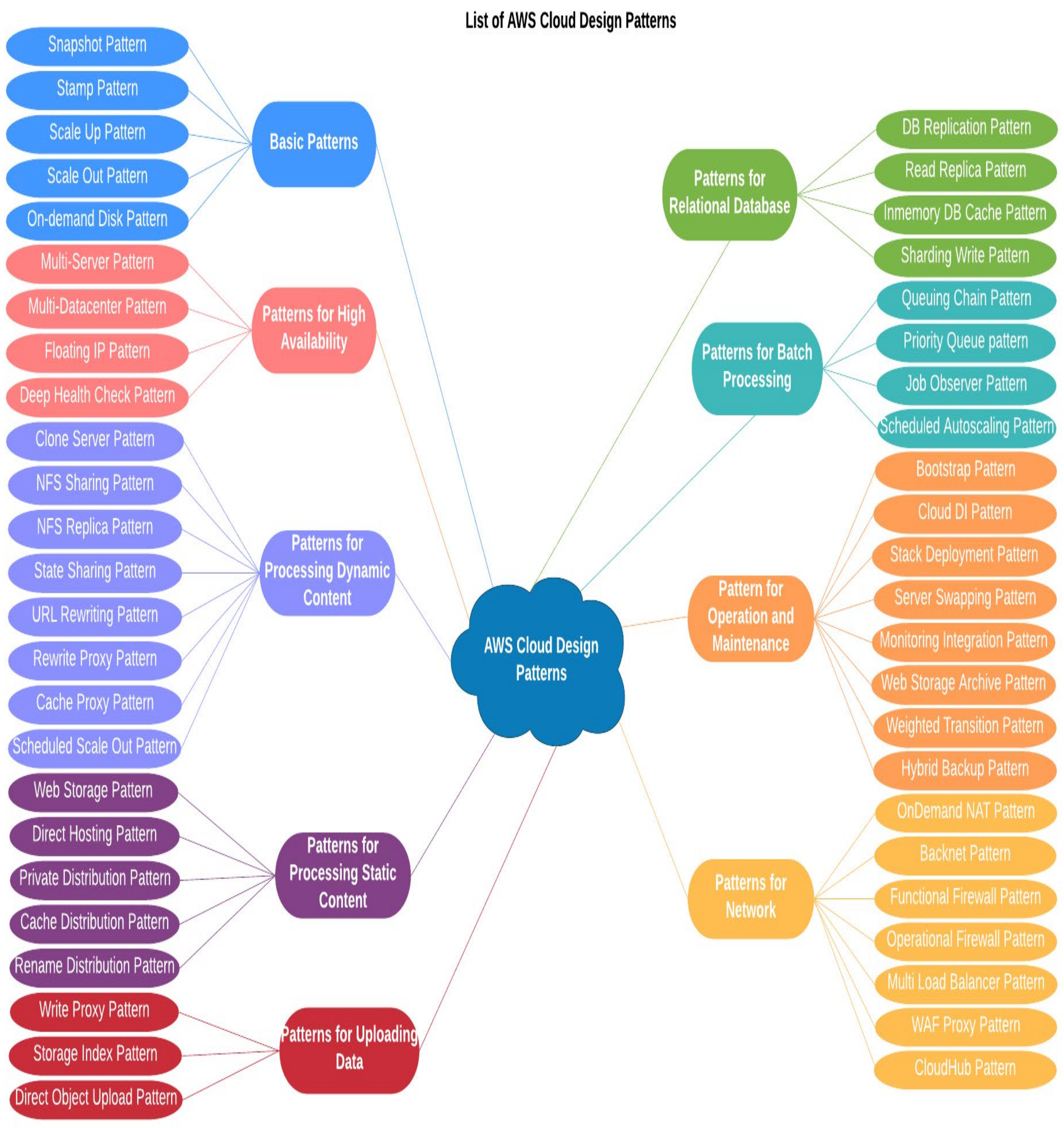


Figure 5. 1 - Amazon AWS Cloud Design Patterns

Furthermore, the use of design patterns provides a means of ensuring best practices in the design of application systems. Some examples are design patterns for typical application architecture, information systems (Hukerikar & Engelmann, 2017), security systems (Delessy et al., 2007), software development (Pautasso et al., 2016) and Internet of Things (Chandra, 2016). One of the features of design patterns is abstractness and independency of the programming language involved or runtime infrastructure to come up with the ageless knowledge that is applicable in various IT environments (Fehling et al. 2014). Table 5.1

presents a survey of several other research efforts in IT that is based on the utilisation of 'Design Patterns' within several other domains.

Table 5. 1 - Survey of the use of Design Patterns across different IT domains

| **Publication(s)** | **Domain** |
|---|---|
| Braga et al. (1999), Meszaros & Brown (1997), Manolescu (1997), Hamza & Fayad (2002), Mulyar & van der Aalst (2005), Schneider & Matthes (2015), Harrer et al. (2017), Haimes et al. (2016), Hukerikar & Engelmann (2017) | Information Systems |
| Lehtonen & Parssinen (2001), Fernandez & Pan (2001), Kodituwakku et al. (2001), Fernandez & Sorgente (2005), Delessy et al. (2007) | Security Systems |
| Richardson (2001), Rossi et al. (1996), Re et al. (2001), Avgeriou et al. (2004), Paris et al. (2003) | Web Applications |
| Mahemoff & Johnston (1999), Beedle et al. (1999), Kendall et al. (1997), Silva et al. (1996), E Silva et al. (2005), Zhao et al. (2008), Schummer (2003), Evitts & Hinchcliffe (2000), Eloranta et al. (2010), Hentrich & Zdun (2009), Weiss (2003), Molin & Ohlsson (1996), Pyarali et al. (2000), Pautasso et al. (2016) | Software Development |
| Ben-Yehuda (1997), Guerra et al. (2009) | Framework Development |
| Tidwell (1997), Mahemoff & Johnston (1998) | Human Computer Interface Design |
| Stepney (2012) | Computer Simulations |
| Fehling et al. (2012) | Cloud Applications |
| Keller & Coldewey (1996) | Database Management |
| Byun et al (2002), Amoretti & Zanichelli (2018), Lascano (2017) | Systems Networking |
| Chandra (2016) | Internet of Things |

The use of design patterns makes it possible to leverage characteristics of flexible design and design reuse embedded in models that are established based on collaboration for providing adaptability and reuse in the implementation (VanHilst & Notkin 1996). They also provide a common vocabulary for professionals across multiple sub-domains in information technology to enhance the documentation of software designs (Erl et al., 2015). Furthermore, they bring together static and dynamic structures and over time, are made up of collaborations of successful solutions to problems coming from when building applications in a domain (Brezillon, 2003). Section 5.2 presents the engineering process for holistic semantic annotation design patterns in this research, based on the work of Fehling (2015).

## 5.2 Design Patterns Engineering for CloudSea

Design Patterns Engineering defines the process of identifying, authoring and applying design patterns within a specified domain. Design Patterns identified within a domain evolve over time. Hence, the overall engineering process repeats indefinitely to identify new patterns and refine existing ones constantly. Each of the phases; Pattern Identification, Pattern Authoring and Pattern Application are also iterative. This is because decisions and assumptions made in the first iteration of a stage should be accordingly reviewed and adjusted. All phases of the pattern engineering process are addressed by a user who conducts the steps each phase prescribes. Every user role may be met by a person or a group of individuals (Fehling et al., 2015). Figure 5.2 defines the different user roles for each of the engineering phases.

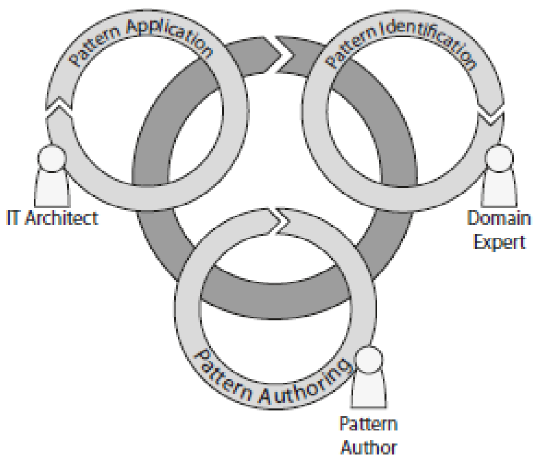


Figure 5. 2 - User Roles for Design Patterns Engineering (Fehling et al., 2015)

### 5.2.1 Pattern Identification Phase

Fehling et al. (2015) described the pattern identification phase as a phase for the structuring and collection of information applicable to a domain where the patterns will be identified. Specifically, in this phase, the aim is to come up with a well-structured domain that will identify the patterns. The definition of the terminology and graphical elements utilised in describing patterns are also established for use in every pattern and with the documentation of the solution in a uniform manner. The steps become necessities in the event of the coordination of large teams of pattern researchers. The process of identifying patterns in the identification phase is handled by a domain expert. The responsibility of the domain expert here is to define the domain and basic characteristic features. There are also negotiations on the collection of information on solutions at hand; provider documentation, detailed structure, and collection format. Specifically, this is important in situations where several persons are responsible for the collection of information in this phase to achieve homogeneous outputs. With this research, a list of requirements for holistic semantic annotation were identified in Chapter 3. The need to fulfil the requirements necessitated defining the role of each one of them within the holistic semantic annotation perspective. The role definition also implied specifying capabilities for each one of them, as can be seen in Chapter 3. Hence, the pattern identification phase in this context was fulfilled in Chapter 3 by the identification of necessary requirements for holistic semantic annotation. A summary of the identified design patterns is presented in Table 5.2.

Table 5. 2 - Identified Design Patterns for Holistic Semantic Annotation

| **Design Pattern** | **Summary Description** |
|---|---|
| Concept Extraction | Provides a mechanism for extracting instances of concepts such as people, places, organisations, etc. from textual documents to populate a RDF graph database within the repository. |
| Ontology Population | Adding newer concepts and relations to ontologies from textual documents, defining a larger scope or completeness for the ontology. |
| Ontology Selection | Provides a procedure for selecting the appropriate ontologies from the repository for an annotation data generation process requiring mapping of two or more ontologies |
| Ontology Mapping | Executes a mapping process (using ASMOV algorithm) for selected ontologies towards annotation data generation |
| Annotation Data Storage | Provides a decoupled storage mechanism for generated annotation data, storing the annotation data separately from the web document |
| Annotation On-the-fly | Provides online, real time annotation for web documents by querying and fetching corresponding annotation data from the RDF graph database. |
| Annotation Data Re-Use | Provides a direct mapping between web document and annotation data such that they can always be paired upon request anytime |
| Annotation Data Sharing | Provides authorised access to annotation data for multiple web documents with same content and within the same domain |
| Annotation Data Auto-Update | Updating annotation data based on ontology or ontology language evolution to ensure consistency between web documents and their annotation data. |
| Ontology Auto-Update | Providing updates to ontologies based on updates to ontology schema |
| Annotation Data Optimisation | Optimises stored annotation data based on processes such as schematic evolution of ontology or ontology language to ensure accuracy of annotation data |

| | |
|---|---|
| Annotation Data Co-Location | Storing or migrating annotation data to locations with close proximity to the corresponding web documents to optimise computing resources. |

### 5.2.2 Pattern Authoring Phase

During this phase, patterns are established according to the information obtained based on the identification of similar existing solutions (Fehling et al. 2015). A pattern author is required in this phase for the analysis of the information collected to identify recurring patterns. The pattern author then drafts and revises these in several iterations. Pattern documents are produced by following a specific structure as well as referencing other pattern documents with the aid of sound interrelations. The study of Meszaros & Brown (1997) gives a description for the best way to achieve pattern authoring, specifically on how to write pattern documents. The best practices described in their study are captured as patterns, being in the form of a pattern language. Their study also discussed the subject pattern document structure which possesses the capability to influence pattern language metamodel. In addition, their focus was on how to name patterns as well as how to name the references that come with each pattern in such a way that the terms used are understandable.

The terms used are also such that they structure the pattern language sufficiently as regards the design process utilised within the domain. Just as the names given to design patterns should portray the reason for their existence and should be easy to use within sentences for contributions to the normal language of architects, naming conventions have been influencing all cloud computing patterns in a significant manner. Particularly, the use of nouns for naming patterns is more promoted than the use of verbs for easy reference to patterns as entities in sentences (Fehling 2015). Furthermore, the structure of design patterns is of huge importance during the authoring phase. Design Patterns across different areas and domains can have slightly varying structures which are defined based on the context in which they are being authored. The structure across multiple domains in IT is quite uniform, although with some variations. Table 5.3 presents a survey of design pattern structures across varying literature sources in computing.

Table 5. 3 - Design Patterns Structure across Literature in Computing

| | **Context** | **Problem** | **Forces** | **Solution** | **Resulting Context** | **Example** | **Known Uses** | **Related Patterns** | **Variations** | **Others** |
|---|---|---|---|---|---|---|---|---|---|---|
| Braga et al. (1999), Manolescu (1997) | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ | |
| Lehtonen & Parssinen (2001), Guerra et al. (2009) | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | ✓ | | |
| Fernandez & Pan (2001), Fehling et al. (2012), Eloranta et al. (2010), Schneider & Matthes (2015) | ✓ | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ | | |
| Richardson (2001), Amoretti & Zanichelli (2018) | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | |
| Mahemoff & Johnston (1999), Ben-Yehuda (1997), Tidwell (1997), Silva et al. (1996), Meszaros & Brown (1997) | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | | | Related Issues - Ben-Yehuda (1997), Notes - Tidwell (1997), |
| Beedle et al. (1999) | ✓ | ✓ | ✓ | ✓ | ✓ | | ✓ | | | |
| Kendall et al. (1997) | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ | | | |
| Rossi et al. (1996) | ✓ | ✓ | | ✓ | ✓ | | ✓ | ✓ | | Participants, Collaboration, Implementation – Rossi et al. (1996) |
| Stepney (2012) | ✓ | ✓ | | | ✓ | | | ✓ | | |
| Kodituwakku et al. (2001) | ✓ | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ | | |
| Fernandez & Sorgente (2005), Lascano (2017) | ✓ | ✓ | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | |
| Re et al. (2001) | ✓ | ✓ | ✓ | ✓ | | ✓ | | ✓ | ✓ | |
| E Silva et al. (2005) | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | Implementation – E Silva (2005) |
| Massingill et al. (2001) | ✓ | ✓ | | | ✓ | ✓ | ✓ | ✓ | | Applicability, Implementation – Massingill et al. (2001) |
| Delessy et al. (2007) | ✓ | ✓ | | | ✓ | | ✓ | ✓ | | Implementation – Delessy et al. (2007) |

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Avgeriou et al. (2004), Schummer (2003) | ✓ | ✓ | ✓ | ✓ | | | ✓ | ✓ | | Participants, Rationale, Danger Spot – Schummer (2003) |
| Zhao et al. (2008) | ✓ | ✓ | | ✓ | ✓ | ✓ | | | | Limitations, What's Next – Zhao et al. (2008) |
| Evitts & Hinchcliffe (2000), Weiss (2003), Molin & Ohlsson (1996), Pyarali et al. (2000), Hukerikar & Engelmann (2017) | ✓ | ✓ | ✓ | ✓ | ✓ | | | | | Discussion – Evitts & Hinchcliffe (2000) |
| Paris et al. (2003) | ✓ | ✓ | | ✓ | | | ✓ | ✓ | | User Category – Paris et al. (2003) |
| Hentrich & Zdun (2009) | ✓ | ✓ | | ✓ | | ✓ | ✓ | | | |
| Byun et al. (2002) | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | ✓ | |
| Mahemoff & Johnston (1998), Hamza & Fayad (2002) | ✓ | ✓ | ✓ | ✓ | | ✓ | | | | |
| Mulyar & van der Aalst (2005), Pautasso et al. (2016) | ✓ | ✓ | ✓ | ✓ | ✓ | | ✓ | | ✓ | |
| Keller & Coldewey (1996) | ✓ | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ | ✓ | |
| Harrer et al. (2017) | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | | Liabilities, Shared Challenge, Unresolved Forces, Benefits For – Harrer et al. (2017) |
| Haimes et al. (2016) | ✓ | ✓ | | ✓ | | ✓ | | | ✓ | |

From Table 5.3, it can be observed that while some differences exist in the structure of different collections of design patterns, these differences are only minimal, with most of them having very similar structures. From the table, they all define a 'Context' for the design patterns as well as a specific 'Problem' it is designed to solve. While the 'Forces' section is missing in some of the research efforts, this is because the impacting factors that constitute scenarios leading to the problems is single and already described alongside the 'Problem' statement. In similar vein, some of the research efforts define 'Solutions' to the stated problems under the 'Resulting Context' section. This explains the reason why the 'Solution' field is missing in some of them.

Furthermore, some of the design patterns missing the 'Examples' section makes up for this in several other ways. These include the use of diagrams or other representations such as UML (Unified Modelling Language) with class, sequence or use case diagrams. The inclusion of the

‘Known Uses’ section is dependent on if there are any known uses or not, hence the reason why it is missing in some design patterns. In some other cases, rather than this field, the ‘Variations’ field is more relevant. Lastly, some design patterns have also defined additional sections such as ‘Participants’, ‘Collaboration’, ‘Liabilities’, etc. to provide completeness based on the domain involved. Conclusively, the table reveals the widely accepted sections required within design patterns and the level of flexibility with them which could imply omitting one or more sections. Based on these, the sections relevant to this research have been carefully selected and are described as follows:

**Pattern Name** - A unique name identifies a design pattern. The name specifies the design pattern’s purpose. The patterns name further identifies the entity present in the application’s architecture. The design pattern normally describes the entity.

**Context** - For every created design pattern, it is necessary to specify the conditions which describe the problem that it should solve. A Pattern Author may specify the context such that it refers the design pattern to other patterns if necessary. The context is a very crucial component especially in describing cloud offering and cloud types. Describing the context in a design pattern format significantly simplifies the environment where the design pattern’s application architecture can apply. The context eases the description of the design pattern requirements in relation to the cloud environment.

**Problem** - As mentioned earlier, design patterns are used to find solutions for recurring problems in various IT fields. Detailed design patterns used in cloud computing have a small descriptive question at the beginning which describes the problem that the pattern will solve. The driving question is important since developers creating cloud applications use a design pattern’s catalogue to look for solutions for the patterns’ driving questions. Thus, the driving question eases the process for identifying the required patterns.

**Solution** - The solution makes up a complete format of a design pattern. In this section, the author specifies instructions which the design pattern can use to address all the identified challenges. The design pattern gives the solution in the form of small steps. The system follows the steps to address the challenges. The solution is usually supported by at least a graphical representation. It depicts the functionality of the resulting architecture after the application of the pattern.

**Structure -** This section provides an illustration of the technical solution described in the "Solution" field. This could be in the form of a diagram, flowchart or any other suitable means of representing the technical solution.

**Resulting Context** - This section describes the effects achieved by following the steps described in the solution. It also provides details about the implementation of the design pattern. The identified design patterns for holistic semantic annotation presented in Table 5.2 and formulated from the holistic semantic annotation requirements defined in Chapter 3 of this thesis have been authored based on these 'Pattern Authoring' process as described in the section.

### 5.2.3 Pattern Application Phase

After the pattern identification and authoring phases, the next phase is its application. That is, the newly discovered solutions in form of design patterns are used by any IT professional (such as a software architect or software developer) who finds them useful. In this phase, there are various ways of making the design patterns and pattern language available through the provision of guidance to users of the design patterns while considering the utilisation of the patterns in solving their problems. Fehling (2015) suggested that the pattern application process consists of the search for applicable design as well as its application, defining a "pattern search and recommendation" process which involves recommending patterns to users according to the structure of the pattern language. Navigation from one pattern to another is carried out with the aid of references existing between them. The references depict the interrelationships that exist between the design patterns, as they form building blocks for solutions within a domain and constitute a 'Pattern Language' together. Following navigation is finding and selecting an initial set of patterns based on their categorisations. The decision to use a specific pattern should be based on its description and relevance to a challenge according to a need. The design patterns identified and authored in this research are utilised in section 5.5 for the proposed architecture.

## 5.3 Design Pattern Language for CloudSea

An individual design pattern may lead to augmented reuse, although the greatest benefits emerge once patterns are coarsely grained together to form a pattern language. A pattern language of a specific domain is the collection of available design patterns within the domain, their interrelations, and rules for combining them (Erl et al., 2015). Thus, the pattern language deals with the common problems in the domain for the purposes of guiding a design process.

After the application of the resolution of a certain design pattern, a fresh context emerges whereby further complex problems call for a solution. More design patterns may be developed for capturing the problem-solving process integral in the new context (Goodyear & Retalis 2010). Consequently, the organisation of a group of design patterns into a set-up of co-dependent patterns creates a pattern language, particularly in which design patterns of greater level produce situations that are solved by design patterns which are more comprehensive. This enables a software architect or developer to use the pattern language in a generic manner, starting with a situation, and following all the applicable design patterns to implement a solution. The Pattern Language is composed of references among design patterns of distinct types for enabling navigation. Fehling et al. (2011) affirm that the patterns' order to be considered is subsequently described in an implicit manner in the pattern language.

This is also applicable for design patterns in the cloud computing domain such that cloud design patterns possess an implicit ordering for their consideration. To introduce further help to pattern users, the common tasks of creating a new cloud application may be made more explicit. The design patterns' implicit ordering along with the refinement stages are defined as an explicit process that should be followed throughout IT architecture design. During every phase of the process, a collection of patterns is given which is normally used. Such patterns might be applied as points of entry to the pattern language with the aim of identifying the use case for specific pattern compositions by following the references amongst patterns (Fehling et al. 2014; Fehling 2015). A pattern-based design technique for cloud applications is required to explain the general order in which the cloud computing pattern language ought to be considered in the process of designing a new application. From the design patterns developed for holistic semantic annotation, Figure 5.3 represents the proposed pattern language; defining the inter-relations between the twelve design patterns for semantic annotation.

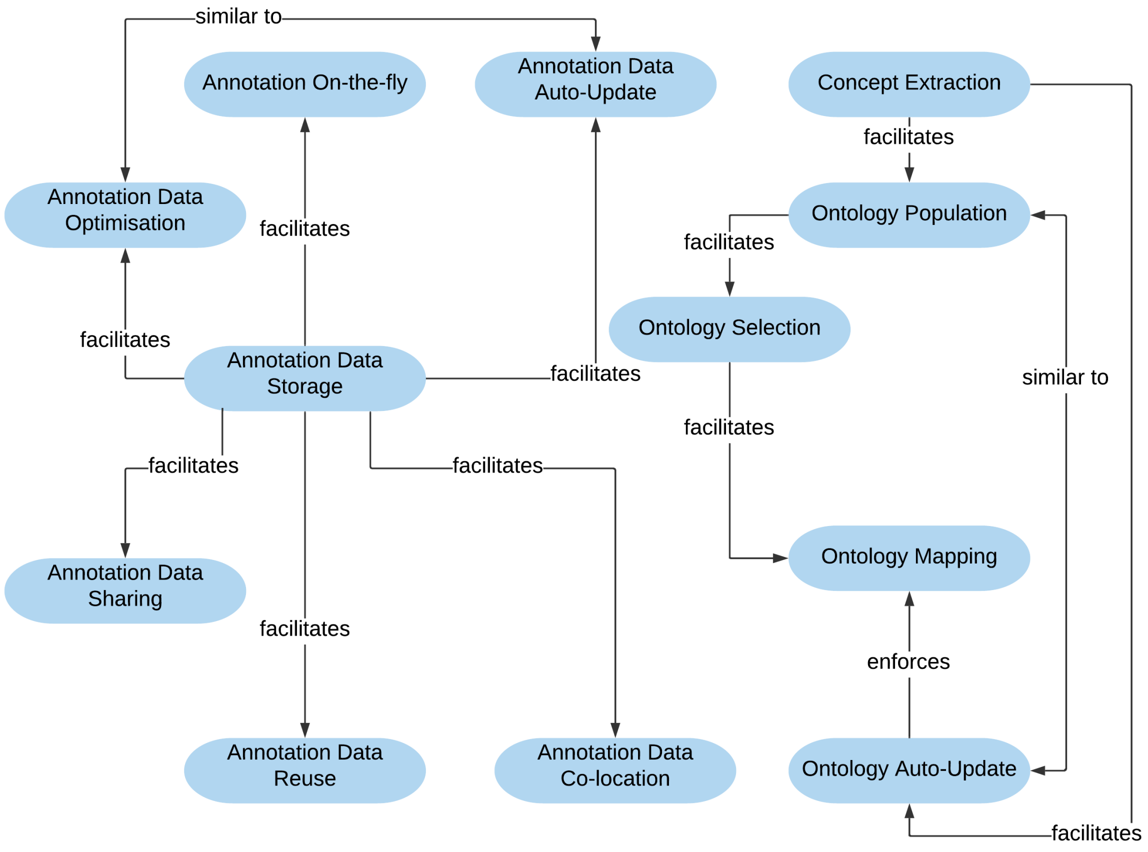


Figure 5. 3 - Holistic Semantic Annotation Pattern Language

Furthermore, a pattern language provides several benefits for software development, right from the architectural design stage. Firstly, they provide a common foundation, on which to build higher-level systems. Transiting from a pattern language to an architectural pattern, usually offering a description of an overall pattern followed by an entire system is a very logical design flow (Edwin, 2014). In addition, according to Opdyke (1990), design patterns and pattern languages make provision for a means of facilitating the reorganisation or refactoring of class hierarchies. Pattern languages also provide a common workspace, which fosters best-of-breed solutions and ensures consistency in how software systems are designed and built. As described by Gamma et al. (2000), design patterns and pattern languages serve as building blocks for the construction of more complex designs, which makes them considerable as microarchitectures that impact on the overall system architecture. Based on continuous efforts to build more complex computer systems, the challenges encountered are from the construction phase rather than from analysis. Hence, the challenges coming from the development of systems are solved by coming up with programming solutions that are based on the context of the computer application being developed. Some of the challenges continue to come up several times across a wide range of different computer applications. Evidently, design patterns and pattern languages can provide solutions to these challenges as they offer a generic solution to such repeating problems, and such solutions can be adapted to other specific needs for an application development process.

Furthermore, they enhance the development of standard software libraries and frameworks. The development of design patterns and pattern languages to provide solutions for recurring challenges in software development fosters the compilation of software libraries and frameworks developed as a result of their utilisation. Such libraries and frameworks evolve over time and often constitute open source solutions to the IT community. In addition, they provide a simpler development and deployment experience, by making good practices easier to adopt. Generally, pattern documentation offers the description of a context in which it is possible to utilise it, the problem the pattern solves, and the solution it proffers (Amoretti & Zanichelli, 2018), with the opportunity to focus on an object-oriented design through the description of when it applies, whether it is applicable in view of other design constraints, and the consequences and trade-offs of its use. Lastly, they organise design intelligence into a standardised and easily referenced format, providing a description that discusses details of a design decision (Hukerikar & Engelmann, 2017). Design patterns consist of established standards that are accessible through software libraries and frameworks, organisable in such a way that software architects and developers reference them for finding solutions to design and deployment challenges.

## 5.4 CloudSea Microservices-based Design Patterns

The pattern language for holistic semantic annotation presented in Figure 5.3 represents a high-level conceptualisation of the CloudSea Microservices Architecture. In the following section, each design pattern is decomposed into its various components and functionalities of each one of them is described. Furthermore, the components of each of the other layers of the architectural pattern that make up the architecture is defined. The microservices for the architecture have been classified into two categories; Service Microservices and Experience Microservices. While the Service Microservices constitute the core of the application, the Experience Microservices are auxiliary in nature, providing addon value for the application architecture. Table 5.4 presents a mapping of the CloudSea Design Patterns to their Microservices names as would be utilised later in the chapter for the CloudSea Architecture.

Table 5. 4 – Mapping of CloudSea Design Patterns to Microservices

| CloudSea Design Pattern | Corresponding CloudSea Microservice |
|---|---|
| Concept Extraction | Concept Extractor |
| Ontology Population | Population Engine |
| Ontology Selection | Selection Engine |
| Ontology Mapping | Mapping Engine |
| Annotation Data Storage | Annotation Data Storage |
| Annotation On-the-Fly | Annotation On-the-Fly |
| Annotation Data Reuse | Annotation Data Reuse |
| Annotation Data Sharing | Annotation Data Sharer |
| Annotation Data Auto-Update | Annotation Data Auto-Updater |
| Ontology Auto-Update | Ontology Auto-Updater |
| Annotation Data Optimisation | Annotation Data Optimiser |
| Service Co-Location | Service Co-Locator |

### 5.4.1 The Service Microservices-Based Design Patterns for CloudSea

The Service Microservices refers to the core sub-systems of the architecture which implement some fundamental processes for the overall application functionality. This is on contrast to microservices that offer auxiliary roles within the entire system architecture. Service Microservices within CloudSea comprises of the holistic semantic annotation requirements identified in Chapter 3 of this thesis and are discussed in this section.

#### 5.4.1.1 Concept Extraction Design Pattern

**Context**

The Concept Extraction Design Pattern is based on the “Concept Extractor” microservice of CloudSea, as stated in Table 5.4 and is responsible for the extraction of entities and their relationships from diverse textual sources and adding them to a semantic graph database. It publishes “semantic graph” datasets to the messaging stream which is utilised by the “Annotation Data Storage” microservice for generating annotation data instances for web documents.

**Problem**

How can semantic graph databases be managed within a microservices architecture with activities such as extraction, interlinking and storage among others?

**Solution**

A "Concept Extractor" microservice is developed with the implementation and integration of the following components:

- **Text Identifier:** Textual information exists in diverse types of repositories and documents. These need to be identified and extracted. This component is for identifying text from diverse sources, including from both structured and non-structured sources as well as extracting them for further processing.
- **Text Processor:** For analysing the extracted text using NLP algorithms and identifying concepts such as places, organisations, people and dates.
- **Entity Extractor:** For classifying extracted concepts into specific categories and addressing any ambiguities based on keyword matching across different domains.
- **Relationship Extractor:** For identifying and building relationships between the different extracted entities. This is done by defining a "predicate" values for interlinking entities. Relationships are also established with existing entities within the graph database.
- **Entity Processor:** For indexing the new dataset and storing them in a semantic graph database for subsequent querying towards web documents semantic annotation.

**Structure**

Figure 5.4 presents a flowchart to illustrate the process flow for the Concept Extraction Design Pattern.

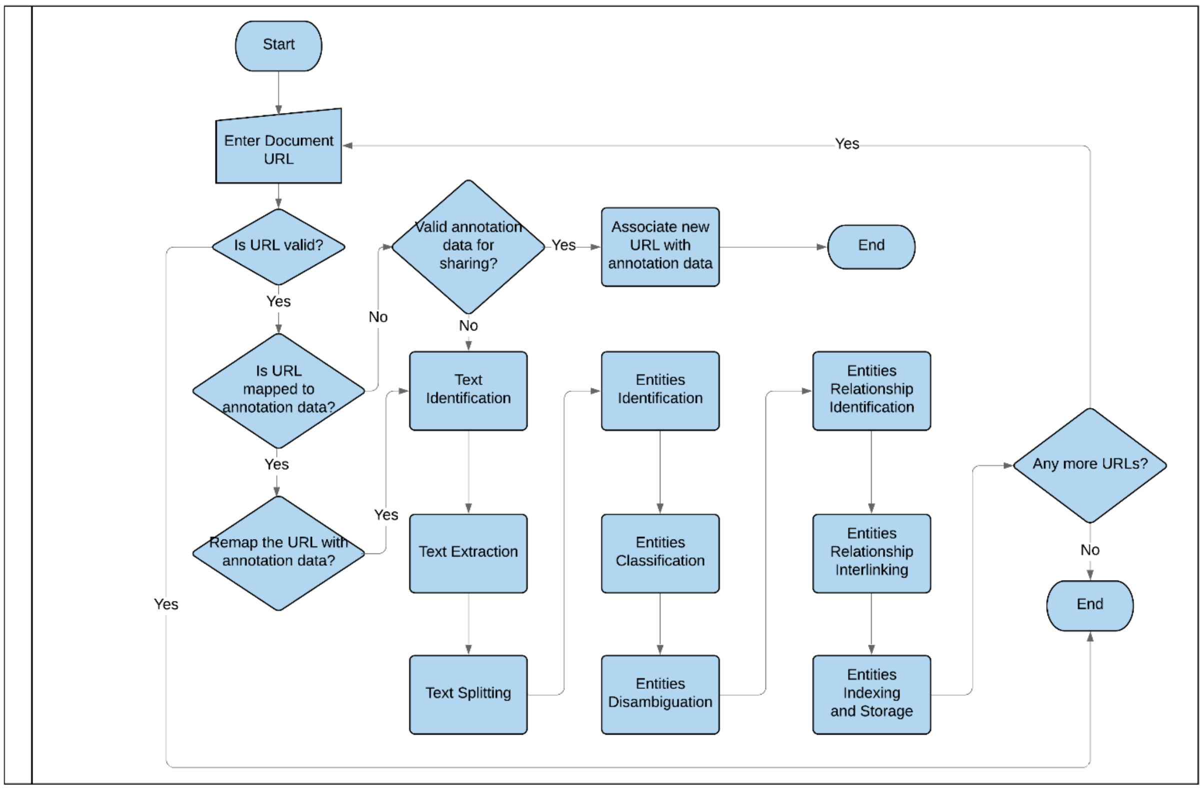

Start
Enter Document URL
Yes
Is URL valid?
Valid annotation data for sharing?
Yes
Associate new URL with annotation data
End
Yes
No
No
Is URL mapped to annotation data?
Text Identification
Entities Identification
Entities Relationship Identification
Any more URLs?
Yes
Yes
Remap the URL with annotation data?
Text Extraction
Entities Classification
Entities Relationship Interlinking
No
Yes
End
Text Splitting
Entities Disambiguation
Entities Indexing and Storage

Figure 5. 4 – Concept Extraction Design Pattern Flowchart

**Resulting Context**

Annotation data repository, in the form of a graph database is populated with additional entities and their relationships with each other. This is queried to obtain annotation data instances for web documents.

### 5.4.1.2 Ontology Population Design Pattern

**Context**

The Ontology Population Design Pattern is based on the "Population Engine" microservice of CloudSea, as stated in Table 5.4. It is responsible for the addition of new concepts and relations extracted from various textual sources to ontologies. It comprises of a data store which holds records of ontologies and enough descriptive data about each to inform an effective selection process. Furthermore, it subscribes to "ontology updates" dataset from the "Ontology Auto-Updater" microservice while it publishes "ontology profiles" dataset for the "Selection Engine" microservice.

**Problem**

How can ontologies be populated with new concepts and relations from several streams of information and aggregated within a microservices architecture?

**Solution**

A "Population Engine" microservice is developed which provides a mechanism for the population process. This is based on the implementation and integration of the following components:

- **Concepts Extractor:** For the extraction process of concepts from one or more textual sources based on a technique capable of extracting from different sources and formats.
- **Relations Extractor:** For identifying and extracting relations between different concepts. These relations will be applied to the schema within an ontology to define relationships between different concepts; both existing and newly added ones.
- **OWL Generator:** For generating an OWL document from the new data (concepts and relations) for subsequent addition to the appropriate ontology.
- **OWL Exporter:** The generated OWL document, comprising of new concepts and relations is exported for importing into the corresponding ontology.
- **Ontology Populator:** For adding new concepts and relations to the existing schema within the ontology; identifying the required level of hierarchy and relationships between new and old concepts.

**Structure**

Figure 5.5 presents a flowchart to illustrate the process flow for the Concept Extraction Design Pattern.

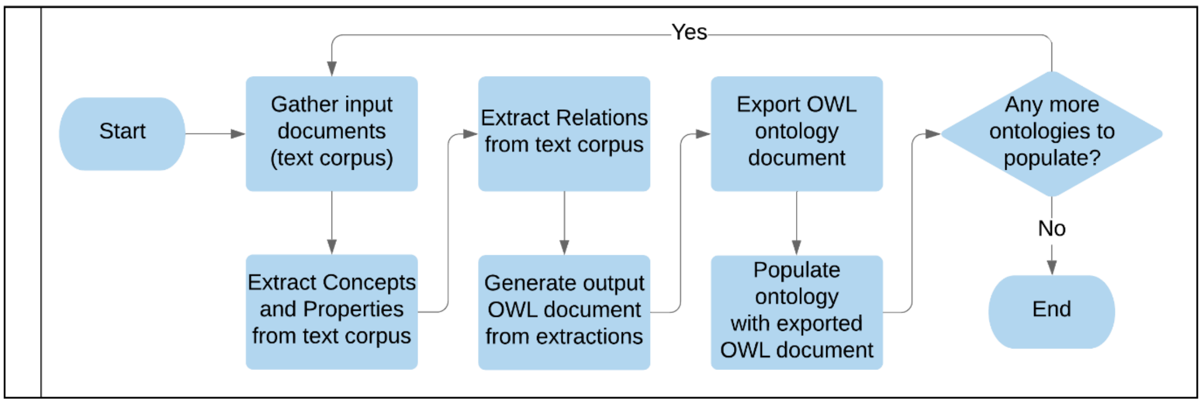

Yes
Start
Gather input documents (text corpus)
Extract Concepts and Properties from text corpus
Extract Relations from text corpus
Generate output OWL document from extractions
Export OWL ontology document
Populate ontology with exported OWL document
Any more ontologies to populate?
No
End

Figure 5. 5 – Ontology Population Design Pattern Flowchart

**Resulting Context**

New concepts and relations are added to ontologies within a microservices architecture, from multiple textual sources and for rich-content annotation data processing.

#### 5.4.1.3 Ontology Selection Design Pattern

**Context**

The Ontology Selection Design Pattern is based on the "Selection Engine" microservice of CloudSea, as stated in Table 5.4 and is responsible for the selection of appropriate ontologies from a repository for an ontology mapping process. The microservice holds records of ontology selections and their mappings to corresponding URIs. Furthermore, it subscribes to the "ontology profile" dataset from the "Population Engine" microservice; which provides the necessary data about ontologies towards an efficient selection process.

**Problem**

How can appropriate ontologies be selected for aggregation of resources towards ontology engineering tasks such as ontology mapping to facilitate rich-content semantic annotation within a microservices architecture?

**Solution**

A "Selection Engine" microservice is developed which defines parameters and mechanisms for an effective selection process. This is based on the implementation and integration of the following components:

- **URI Validator:** For receiving URI of web document that requires semantic annotation. Upon its receipt, the URI is parsed to validate its syntax.
- **URI Scope Definer:** For identifying and defining a scope for the URI. This provides information regarding the ontologies required to be selected from the repository for the URI. The scope definition would require defining the domain a URI belongs to, across multiple levels, such as top-level domain, middle-level domain and lower-level domain.
- **Ontology Browser:** For facilitating access to the "ontology profile" dataset from the "Population Engine" microservice. The dataset is queried for ontologies with a profile that matches the scope defined for the URI.
- **Ontology Selector:** For the selection of appropriate ontologies for a URI based on the result of the dataset query by the "Ontology Browser". The selection is required for an ontology mapping process by the "Mapping Engine" microservices.

- **URI Associator:** For storing URIs and associated ontologies in its data store.

**Structure**

Figure 5.6 presents a flowchart to illustrate the process flow of the Ontology Population Design Pattern.

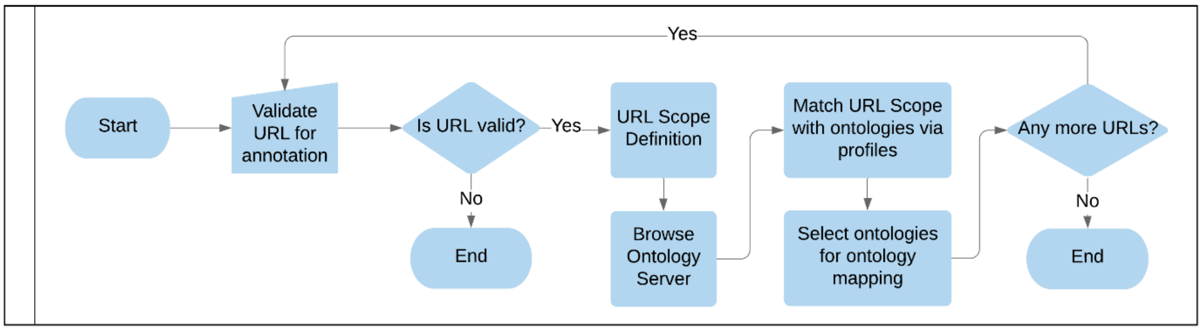

Start
Validate URL for annotation
Is URL valid?
Yes
No
End
URL Scope Definition
Browse Ontology Server
Match URL Scope with ontologies via profiles
Select ontologies for ontology mapping
Any more URLs?
Yes
No
End

Figure 5. 6 – Ontology Selection Design Pattern Flowchart

**Resulting Context**

A set of appropriate ontologies are selected for a URI for use in an ontology mapping process towards the generation of rich-content semantic annotation for the corresponding web document.

### 5.4.1.4 Ontology Mapping Design Pattern

**Context**

The Ontology Mapping Design Pattern is based on the "Mapping Engine" microservice of CloudSea, as stated in Table 5.4 and is responsible for generating mappings between different concepts across multiple ontologies. The mapping provides an aggregation of resources across the ontologies; providing an extension to the scope and possible context derivable for entities. The ASMOV technique to ontology mapping is adopted as it applies semantic verification for the mappings generated, hence, providing a high level of accuracy for them (Mittra & Ali, 2017). The microservice stores these mappings in its data store in JSON format, utilising its subscription to "selection records" dataset of the "Selection Engine" microservice and publishing "ontology mappings" dataset to the messaging stream.

### Problem

How can contextual data schemas across multiple ontologies be utilised together for an ontology mapping process in a microservices architecture towards the generation of annotation data for web documents?

### Solution

A "Mapping Engine" microservice is developed which adapts the ASMOV algorithm, based on the implementation and integration of the following components:

- **Lexical Matcher:** For performing matchings between concepts of two ontologies based on keyword patterns. While this is iterative, it takes two ontologies for each iteration.
- **Similarity Calculator:** For calculating similarities between matched concepts. This is done by an assessment of degrees of equivalence between the pair and utilising a benchmark to assess the weight of the similarity index. The output is a similarity matrix.
- **Semantic Verifier:** Based on the similarity matrix obtained, a pre-alignment of concepts is done using a greedy algorithm. The pre-alignment then undergoes a process of semantic verification based on the ontology schemas, in which unverifiable matches are removed.
- **Mapping Extractor:** Upon successful verification and finalisation of the pre-alignments, the mappings obtained are extracted from the various sources and stored in the data store. The process can be repeated between other pairs of ontologies.

### Structure

Figure 5.7 presents a flowchart to illustrate the process flow of the Ontology Mapping Design Pattern.

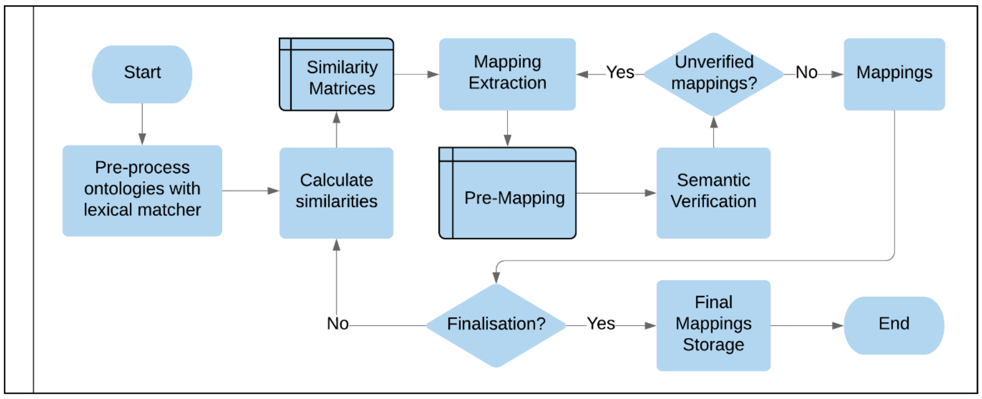


Figure 5. 7 – Ontology Mapping Design Pattern Flowchart

**Resulting Context**

Concepts and relations across multiple ontologies are mapped, thereby providing a means of integration between the ontologies and their aggregated use to facilitate the generation of rich-content annotation data for web documents.

#### 5.4.1.5 Annotation Data Storage Design Pattern

**Context**

The Annotation Data Storage Design Pattern is based on the "Annotation Data Storage" microservice of CloudSea, as stated in Table 5.4 and is responsible for the generation and storage of annotation data for URIs. It does both file-based and server-based storage for generated annotation data. The microservice is subscribed to two datasets; mapping data from the "Mapping Engine" microservice and entities from the "Concept Extraction" microservice. On the other hand, it publishes annotation data to the messaging stream for several microservices, such as "Service Colocator" and "Annotation On-the-Fly" among others subscribed to the dataset.

**Problem**

How can annotation data be stored separately to avoid the challenges of storing them with their corresponding web documents within a microservice architecture?

**Solution**

An "Annotation Data Storage" microservice is developed for the decoupled approach to annotation data storage and is facilitated based on the implementation and integration of the following components:

- **Mapping Fetcher:** For fetching mapping data generated by the "Ontology Mapping" microservice and the associative URIs to the data. The URIs define web documents that the mapping data have been generated for and the data is received in JSON format.
- **Annotation Data Generator:** For the generation of annotation data for specific URIs utilising the mapping data alongside the required ontologies to query the semantic graph database for the required annotation data, which is received as output in JSON format.
- **Annotation Data Processor:** For saving a copy of the generated annotation data and entering a record of it in the data store. The file is saved based on a naming convention.

Processing details, such as corresponding URI, process status, JSON filename, timestamp, etc. are also stored as part of its record in the data store.

**Structure**

Figure 5.8 presents a flowchart to illustrate the process flow of the Annotation Data Storage Design Pattern.

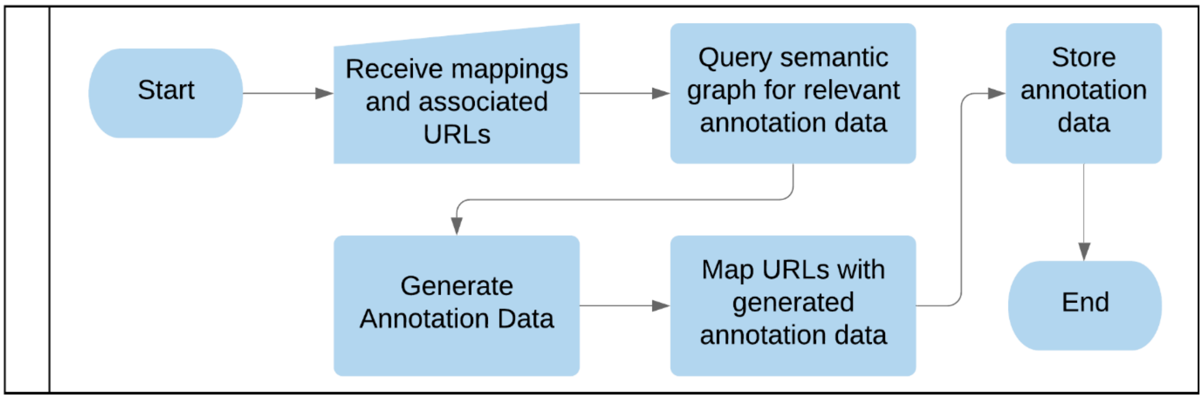

Start
Receive mappings and associated URLs
Query semantic graph for relevant annotation data
Store annotation data
Generate Annotation Data
Map URLs with generated annotation data
End

Figure 5. 8 – Annotation Data Storage Design Pattern Flowchart

**Resulting Context**

Annotation Data is stored separately from the web documents they annotate, ensuring that both the data and web documents can evolve and be processed independently while still maintaining consistency between them.

### 5.4.1.6 Annotation On-The-Fly Design Pattern

**Context**

The Annotation On-the-Fly Design Pattern is based on the "Annotation On-the-Fly" microservice of CloudSea, as stated in Table 5.4 and is responsible for online, real-time semantic annotation of web documents without requiring a human intervention. It subscribes to the "annotation data" dataset of the "Annotation Data Storage" microservice and has no data store of its own.

**Problem**

How can web documents be semantically annotated online, real time without requiring human intervention by means of a microservices architecture?

**Solution**

An "Annotation On-the-Fly" microservice is developed which will trigger three other microservices; "Selection Engine", "Mapping Engine" and "Annotation Data Storage". Furthermore, it will require the implementation and integration of the following components:

- **Annotation Fetcher:** For fetching the generated annotation data which has just been saved on file with a record of it in the data store as well. The file data is published by the "Annotation Data Storage" microservice and accessible from the messaging stream.
- **Annotator:** For the actual semantic annotation which requires parsing objects from the annotation data JSON file and matching them with corresponding strings within the web document for annotation, utilising string offsets; 'start offset' and 'end offset' for the matching.

**Structure**

Figure 5.9 presents a flowchart to illustrate the process flow of the Annotation On-the-Fly Design Pattern.

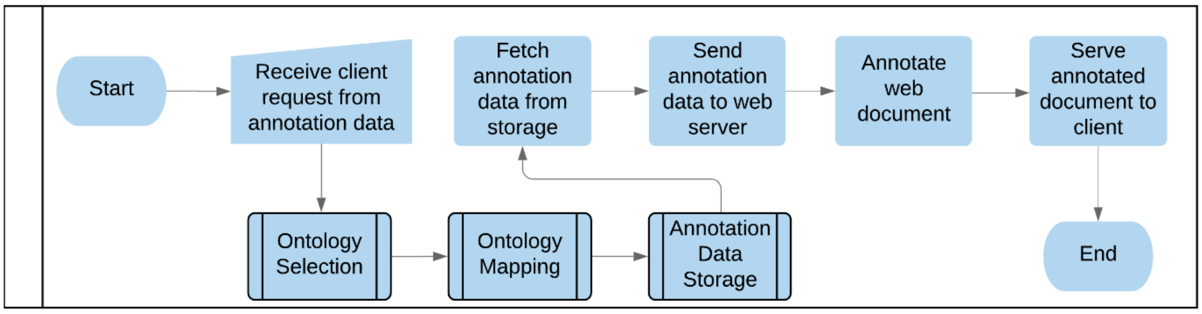

Start
Receive client request from annotation data
Fetch annotation data from storage
Send annotation data to web server
Annotate web document
Serve annotated document to client
Ontology Selection
Ontology Mapping
Annotation Data Storage
End

Figure 5. 9 – Annotation On-the-Fly Design Pattern Flowchart

**Resulting Context**

Web documents can have an automated process for the delivery of continuous and up-to-date annotation data for their content dynamically.

### 5.4.1.7 Annotation Data Reuse Design Pattern

**Context**

The Annotation Data Reuse Design Pattern is based on the "Annotation Data Reuse" microservice of CloudSea, as stated in Table 5.4 and is responsible for the reuse of an instance of annotation data by a web document. This would be the case if the web document has not evolved from the time the annotation data was generated for it. It subscribes to “annotation data” and “URI repository” datasets from the “Annotation Data Storage” and “Annotation Data Sharing” microservices respectively. Its own data store would hold data of statistical reuse of annotation data of different URIs.

### Problem

How can computing resources be optimised within a microservices architecture by reusing a web document annotation data it the web document content has not changed?

### Solution

An “Annotation Data Reuse” microservice is developed which is based on the implementation and integration of the following components:

- **URI Verifier:** For parsing a URI to validate its syntax. It also verifies the URI’s attributes and its mapping to a specific annotation data instance based on reuse statistical data in the data store.
- **Annotator:** For fetching the URI’s annotation data and carrying out same functionality as the “Annotator” component of the “Annotation On-the-Fly” microservice for the actual semantic annotation process.

### Structure

Figure 5.10 presents a flowchart to illustrate the process flow of the Annotation Data Reuse Design Pattern.

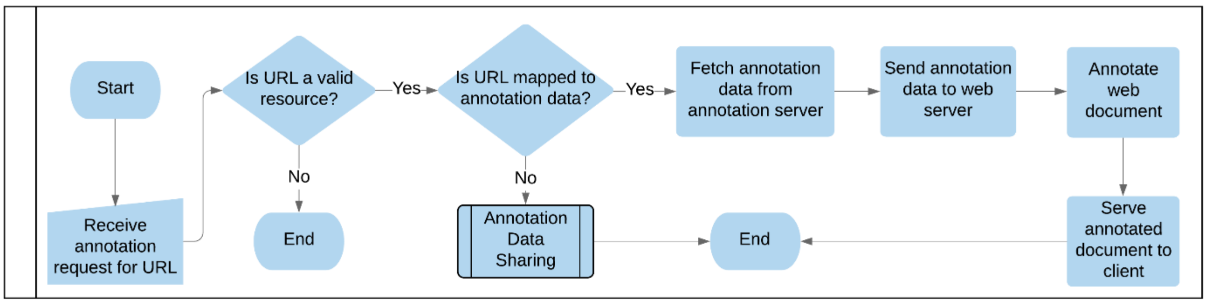

Start
Receive annotation request for URL
Is URL a valid resource?
Yes
No
End
Is URL mapped to annotation data?
Yes
No
Annotation Data Sharing
Fetch annotation data from annotation server
Send annotation data to web server
Annotate web document
Serve annotated document to client
End

Figure 5. 10 – Annotation Data Reuse Design Pattern Flowchart

### Resulting Context

A web document can reuse annotation data generated for it if the web document content has not changed after the generation of the annotation data.

#### 5.4.1.8 Annotation Data Sharing Design Pattern

**Context**

The Annotation Data Sharing Design Pattern is based on the "Annotation Data Sharer" microservice of CloudSea, as stated in Table 5.4 and is responsible for the utilisation of an instance of annotation data to semantically annotate multiple web documents of the same content. It requires and is subscribed to "annotation data" datasets from the "Annotation Data Storage" microservice and publishes "URI repository" datasets which provides data regarding URI-to-annotation data mappings to the messaging stream. Its dataset is subscribed to by the "Annotation Data Reuse" microservice.

**Problem**

How can an instance of annotation data be shared by multiple web documents with the same piece of content within a microservices architecture?

**Solution**

An "Annotation Data Sharer" microservice is developed with the implementation and integration of the following components:

- **URI Validator:** This component provides same functionality as the "URI Validator" component of the "Selection Engine" microservice which is based on parsing URIs to validate their syntax.
- **Mapping Browser:** For browsing through URI and annotation data mappings based on the scope for the validated URI to identify annotation data within same low-level scope.
- **Similarity Calculator:** A similarity calculation compares content within the validated URI and URI mapped to annotation data instances. A similarity index would be defined as a requirement before annotation data instance can be shared. This would be expected to be a value of 100%.
- **URI Mapper:** Based on the attainment of the required similarity index, the URI Mapper maps the validated URI with the corresponding annotation data and keeps a record of it within the microservice data store.

**Structure**

Figure 5.11 presents a flowchart to illustrate the process flow of the Annotation Data Sharing Design Pattern.

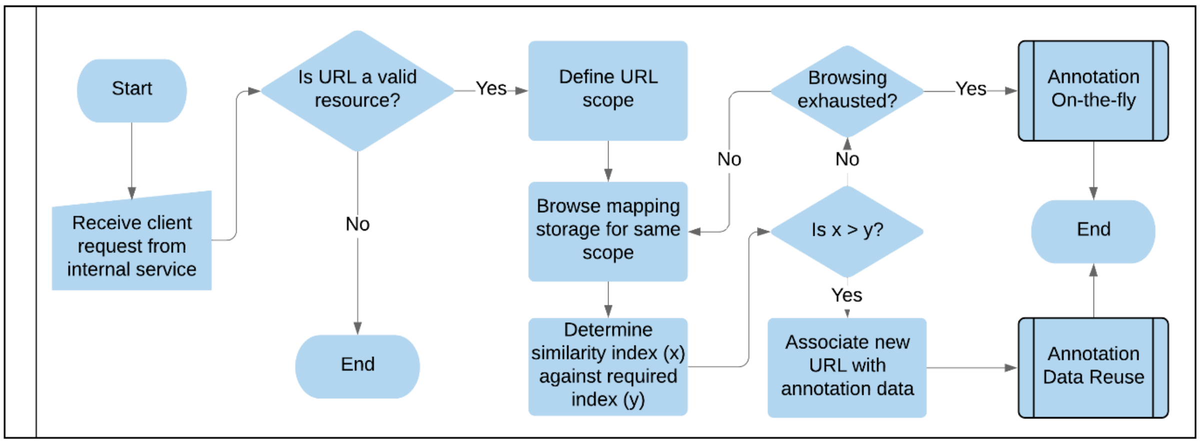

Start
Receive client request from internal service
Is URL a valid resource?
Yes
No
End
Define URL scope
Browse mapping storage for same scope
Determine similarity index (x) against required index (y)
Browsing exhausted?
No
Yes
No
Is x > y?
Yes
Associate new URL with annotation data
Annotation On-the-fly
End
Annotation Data Reuse

Figure 5. 11 – Annotation Data Sharing Design Pattern Flowchart

#### Resulting Context

An instance of annotation data is shared between web documents containing same piece of information. However, this would be applicable only when the web documents content is exactly same as a slight difference can alter the context of a document.

### 5.4.1.9 Annotation Data Auto-Update Design Pattern

#### Context

The Annotation Data Auto-Update Design Pattern is based on the “Annotation Auto-Updater” microservice of CloudSea, as stated in Table 5.4 and is responsible for maintaining consistency between web documents and their corresponding annotation data. This is by automatically updating annotation data of a web document when the document evolves. It requires a tracking mechanism for web document evolution and publishes logs of updates to annotation data from its own data store to the messaging stream. The “Annotation Data Storage” microservice is subscribed to its dataset.

#### Problem

How can consistency between web documents and annotation data be maintained within a microservices architecture considering the dynamic nature of web documents?

#### Solution

An “Annotation Auto-Updater” microservice is developed to maintain consistency between web documents and annotation data, utilising a mechanism for tracking changes within a web document and triggering a corresponding update to its annotation data, such as web scraping.

These is facilitated through the implementation and integration of an "**Annotation Update Manager**" which will read web document content and track any updates to initiate a re-generation of annotation data if an update has occurred. To re-generate annotation data for a web document, it will trigger actions for the "Selection Engine", "Mapping Engine" and "Annotation Data Storage" microservices based on the functionality provided by each one of them.

**Structure**

Figure 5.12 presents a flowchart to illustrate the process flow of the Annotation Data Auto-Update Design Pattern

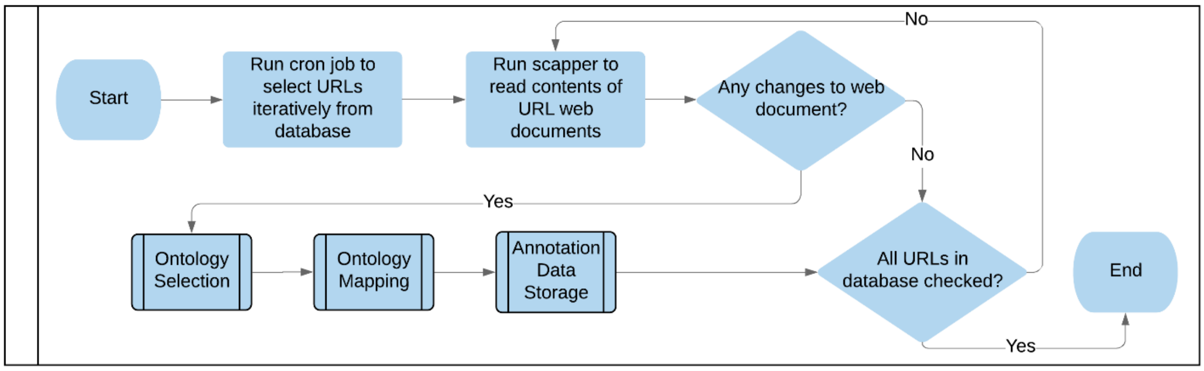

Start
Run cron job to select URLs iteratively from database
Run scapper to read contents of URL web documents
Any changes to web document?
No
No
Yes
Ontology Selection
Ontology Mapping
Annotation Data Storage
All URLs in database checked?
Yes
End

Figure 5. 12 – Annotation Data Auto-Update Design Pattern Flowchart

**Resulting Context**

Annotation data is re-generated for a web document once its content has evolved, maintaining the required level of consistency between them to guarantee annotation data accuracy always.

### 5.4.1.10 Ontology Auto-Update Design Pattern

**Context**

The Ontology Auto-Update Design Pattern is based on the "Ontology Auto-Updater" microservice of CloudSea, as stated in Table 5.4 and is responsible for ensuring up-to-date ontology schema to maintain accuracy of instances of annotation data based on such schemas and subsequently, maintaining consistency between web documents and annotation data by enforcing annotation data re-generation when necessary.

**Problem**

How can ontology schema changes be managed to ensure consistency with web documents and annotation data is maintained in a microservices architecture?

**Solution**

An "Ontology Auto-Updater" microservice is developed with the implementation and integration of an "**Ontology Update Manager**" which will track schematic changes to ontologies and its implications on instances of annotation data. A scheduled task (such as a cron job on Linux platforms) monitors ontologies for any schematic changes and triggers the "Concept Extractor", "Selection Engine", "Mapping Engine" and "Annotation Data Storage" microservices for web document(s) annotation data re-generation when such changes occur.

**Structure**

Figure 5.13 presents a flowchart to illustrate the process flow of the Ontology Auto-Update Design Pattern.

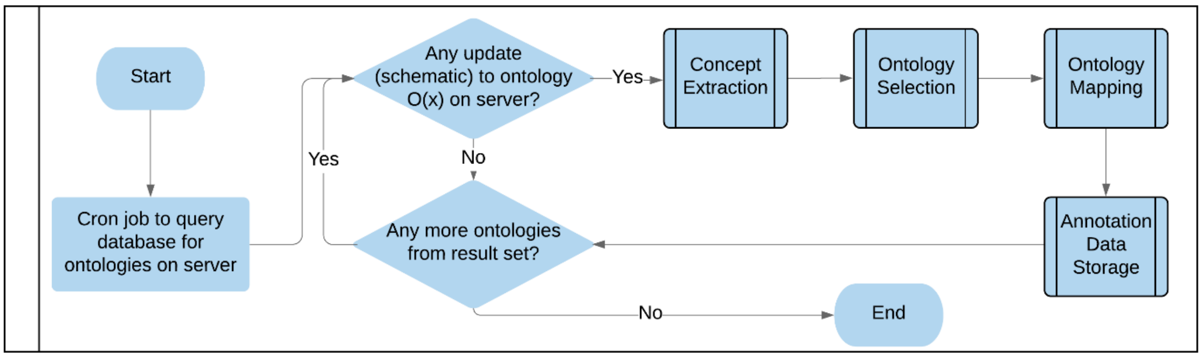

Start
Cron job to query database for ontologies on server
Any update (schematic) to ontology O(x) on server?
Yes
Concept Extraction
Ontology Selection
Ontology Mapping
Annotation Data Storage
No
Any more ontologies from result set?
Yes
No
End

Figure 5. 13 – Ontology Auto-Update Design Pattern Flowchart

**Resulting Context**

Ontology schemas are kept up-to-date and this triggers other actions to ensure consistency of the updates with web documents and annotation data utilising such ontologies.

### 5.4.1.11 Annotation Data Optimisation Design Pattern

**Context**

The Annotation Data Optimisation Design Pattern is based on the "Annotation Data Optimiser" microservice of CloudSea, as stated in Table 5.4 and is responsible for managing upgrades to ontologies, based programming language or standards evolution. It holds a datastore comprising of “optimisation data”. It also subscribes to “ontology upgrades” dataset from the “Ontology Auto-Updater” microservice and “annotation data” datasets from the “Annotation Data Storage” microservice and publishes required “optimisation data” datasets to the messaging stream; which is utilised by the “Annotation Data Storage” microservice.

**Problem**

How can the accuracy of annotation data be maintained despite ontological upgrades resulting from programming language or standards evolution?

**Solution**

An “Annotation Data Optimisation” microservice is developed with the implementation and integration of an “**Ontology Upgrade Manager**” which will track ontological evolutions based on upgrades to the ontology development language (such as version 1.0 to version 2.0) or evolution of one or more standards-based specifications (such as W3C standards for OWL). A scheduled task such as Linux “cron jobs” or Windows “schtasks” monitors ontologies and standards specifications for any evolution and triggers the “Selection Engine”, “Mapping Engine” and “Annotation Data Storage” microservices to re-generate annotation data instances that utilise the evolved ontology.

**Structure**

Figure 5.14 presents a flowchart to illustrate the process flow of the Annotation Data Optimisation Design Pattern.

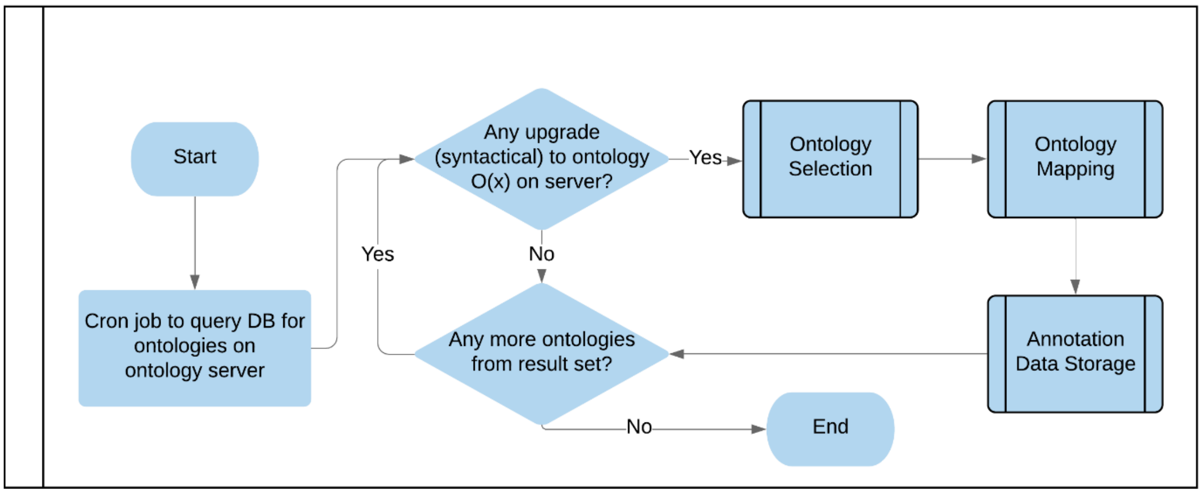

Start
Cron job to query DB for ontologies on ontology server
Any upgrade (syntactical) to ontology O(x) on server?
Yes
Ontology Selection
Ontology Mapping
Annotation Data Storage
No
Yes
Any more ontologies from result set?
No
End

Figure 5. 14 – Annotation Data Optimisation Design Pattern Flowchart

**Resulting Context**

Annotation data is optimised to maintain consistency with web documents and continue to provide the required level of accuracy for semantic annotation. Computing resources are also optimised by removing redundant data from storage.

#### 5.4.1.12 Service Colocation Design Pattern

**Context**

The Service Co-Location Design Pattern is based on the "Service Co-Locator" microservice of CloudSea, as stated in Table 5.4 and is responsible for maintaining proximity between annotation data node and web document node. A closer proximity between both enhances performance by reducing network latency for the semantic annotation process. While web document node cannot be easily influenced, annotation data node can be flexible to ensure the highest possible level of proximity to the web document node. The design pattern subscribes to the "user profiles and settings" dataset of the "User Management" microservice and holds a data store which maps web document nodes with annotation data nodes.

**Problem**

How can communication between web documents and annotation data nodes be more efficient to reduce potential network latencies?

**Solution**

A "Service Co-locator" microservice is developed with the implementation and integration of the following components:

- **Similarity Calculator:** For managing proximity between web document node and annotation data node by utilising the IP address location of the former to place annotation data in a node with the closest possible proximity to the web document node. A benchmark is defined for acceptable proximity index and this is measured for each web document-annotation data pairing to set annotation data node.
- **Geotagger:** For geographical metadata specification of annotation data. Annotation data geolocation is obtained from the "Proximity Calculator" and tagged to the corresponding web document by updating web document geodata with its annotation data geodata as additional metadata for the web document.

**Structure**

Figure 5.15 presents a flowchart to illustrate the process flow of the Service Co-locator Design Pattern.

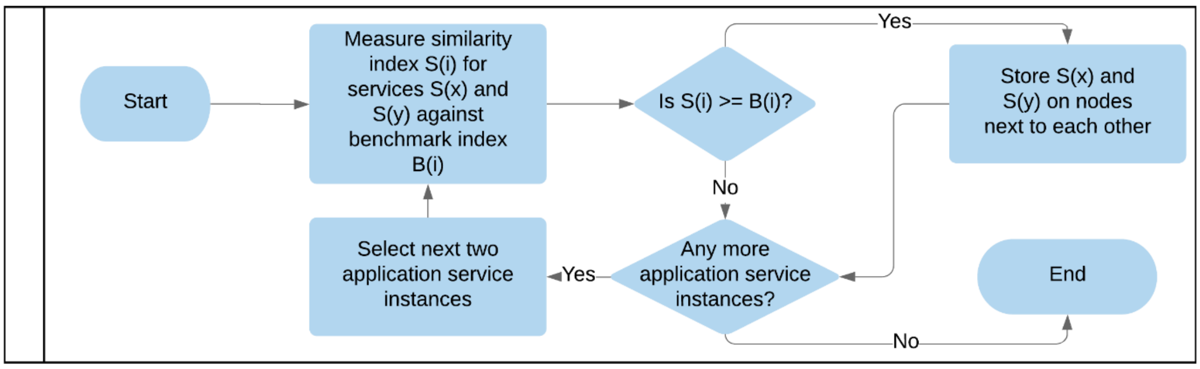

Start
Measure similarity index S(i) for services S(x) and S(y) against benchmark index B(i)
Is S(i) >= B(i)?
Yes
Store S(x) and S(y) on nodes next to each other
No
Any more application service instances?
Yes
Select next two application service instances
No
End

Figure 5. 15 – Service Co-locator Design Pattern Flowchart

**Resulting Context**

Annotation data nodes are placed as close to web document nodes as possible to foster faster communication and reduce potential network latencies, thereby responding to semantic annotation user requests in a timely manner.

## 5.4.2 The Experience Microservices

The Experience Microservices are not core to the application architecture. However, they are required as auxiliary microservices to cater for secondary tasks that are needed while implementing the primary ones. The two experience microservices covered here are: Frontend and User Management.

### 5.4.2.1 The User Interface Design Pattern

**Context**

The User Interface Microservice is the gateway to the application from end users. End Users would access the application via the Internet using a graphical user interface. This interface holds no data of its own. Rather it receives user requests and passes them to the API Gateway. Subsequently, the API Gateway forwards the requests to the Service Registry. The Service Registry defines and categorises the requests and forwards them to the appropriate Service API that will process them. Upon the completion of the processing, the response is sent back to the User Interface Microservice via the API Gateway for the consumption of the end user that initiated the request. Since the User Interface is not directly involved with inter-process communication, it does not need to publish or subscribe to data, rather it functions as an interface between the application and its end users.

**Problem**

How would users interact with the different application logic implemented within the various microservices?

**Solution**

This is by means of a user interface layer that provides several interfaces for interacting with application logic. The required user interface types can be classified as follows:

- **User Services Interfaces:** This function would be responsible for serving interfaces relating to user account management tasks. Such tasks include new user registration, user login, user password reset, user profile management, user annotations management, user-generated annotation process and lots more.
- **Administrative Services Interfaces:** While both user and administrative services share some interfaces in common, access rights are still limited for user accounts. Hence, this function utilises Identity and Access Management (IAM) to define access rights to resources within the application databases. For instance, a logged-in user would only have access to their own annotation records from a specific URL while the same URL would grant an admin user access to annotation records for multiple users.
- **Operational Interfaces:** This function would be responsible for serving interfaces to both users and administrators for operational tasks. This function would still work in unison with the Administrative Services Interface function to define access to different operations for both users and administrators.

**Resulting Context**

Users can interact with different application logic through the corresponding user interface and access rights for each made available.

#### 5.4.2.2 The User Management Design Pattern

**Context**

The User Management Microservice is responsible for dealing with application logics relating to user accounts. These includes user registration, login, password reset, profile update and lots more. The Microservice holds a database of its own which contains user data for the different application logics. Such data includes user profiles, settings and system data. These data are published to the messaging channel for inter-process communication (such as Apache Kafka) for the consumption of some other microservices. It will comprise of the following functions:

**Problem**

How would user data be managed appropriately, with the required level of authentication, authorisation and permission settings?

**Solution**

A user management mechanism would be implemented which would process user data as appropriate. The basic modules that will be inclusive within the mechanism are as follows:

- **User Registration:** This function would be responsible for accepting data from end users submitted via the Frontend Microservice to register an account. The data collected would include username, firstname, lastname, password, email address, etc.
- **User Login:** This function would be responsible for authentication and authorisation for end users. Authentication involves the process of identifying a user based on a set of parameters they have presented. Once the parameters have been validated correctly, then the user becomes authorised. Authorisation implies granting access to certain resources based on pre-defined account settings within the system.
- **Profile Manager:** This function would be responsible for the management of user profiles. This would include updating user data such as password, email address, contact details, etc.
- **Password Reset:** This function would be responsible for assisting users to reset their password. The function would require and utilise mail server settings to send a 'password reset' email to the user's email address registered on file. The mail server settings that would be required include port number, incoming and outgoing mail server addresses and mail protocol.

**Resulting Context**

User data is managed accordingly with the required mechanisms and permission settings. The user data management would also imply users can access services within the system, maintaining confidentiality and integrity.

### 5.4.3 The Inter-Process Communication: Apache Kafka

Inter-Process Communication is a vital component of a microservices architecture. With application functionalities already mapped to business domains and developed independently as microservices with an API interface, communication and data exchange within the multiple microservices ensures that they meet the overall integrated solution for a specific application. While the inter-process communication can be synchronous or asynchronous, the asynchronous is the ideal option for data-intensive applications (Le Noac'H et al., 2017). Apache Kafka is one of several options for providing asynchronous messaging between microservices. It has been described as a messaging system characterised by several messaging patterns, including the distributed publish-subscribe messaging pattern; known for the handling of huge data volumes (Shaheen, 2017). It acts as a robust queue and allows the passage of messages from one endpoint to another. When it first emerged, its description was a "distributed commit log"; however, it has become a major messaging solution, with is adoption by several organisations for asynchronous messaging within highly distributed systems (D'silva et al., 2017). Its design is such that it addresses the challenges of the management of continuous data flows within data-intensive applications.

The Publish-subscribe messaging pattern works by the sender of a message not specifically directing it to a receiver, but there is classification of the message by the publisher with the receiver subscribing to receiving some classes of messages. A broker and a central point for the publication of messages facilitate this process. Figure 5.16 illustrates the publish-subscribe messaging system of Apache Kafka:

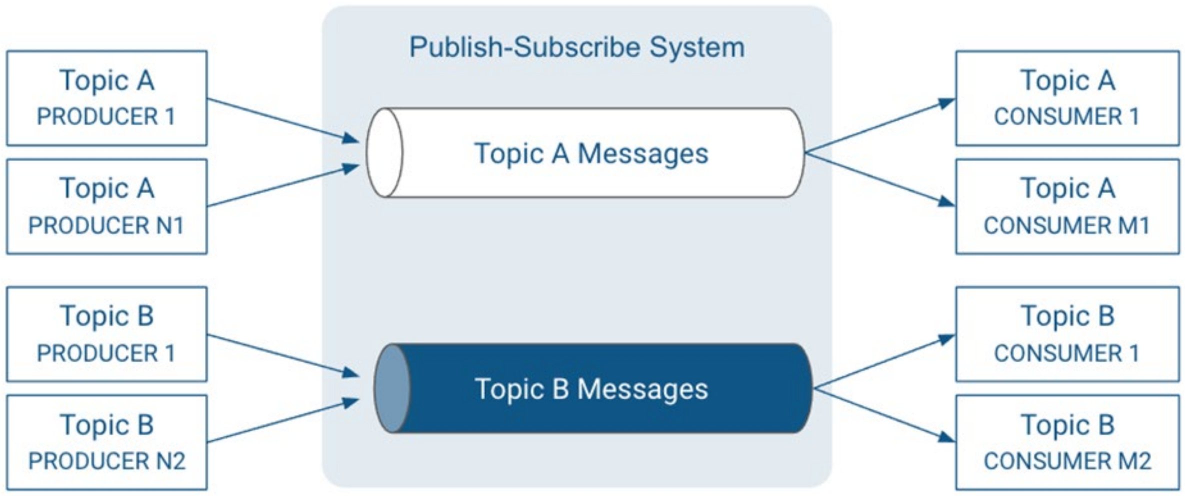


Figure 5. 16 - Apache Kafka Pub-Sub for Microservices Inter-Process Communication (Apache Documentation, 2019)

Durable recording of every transaction for replaying to consistently build the system state is through a filesystem or database commit log. In addition, there is durable storage of data within Kafka, in order, and can be read in a deterministic manner. In addition, there could be data distribution within the system for the provision of additional protection against failures, including remarkable opportunities to scale performance (Wang et al., 2015). The applicative pattern designed for the holistic semantic annotation microservices architecture is the former, whereby streaming data pipelines can be accessed by the microservices based on the publish-subscribe pattern. Some of the benefits of Apache Kafka include reliability, scalability, durability and performance. It is reliable due to its distribution, partitioning, replication and fault-tolerant features. It achieves scalability as a messaging system, without downtime (Shaheen, 2017). Its durability is based on persistence of messages on disk at a speedy rate using "distributed commit log". Furthermore, it does message publishing and subscription with a high throughput, providing a high level of stability in its performance even with the storage of huge number of messages. Several datasets are defined within CloudSea, with each belonging to a specific microservice and distributed across the system based on the Apache Pub-Sub Messaging Pattern. Table 5.5 presents a list of the datasets and their distribution pattern across the overall architecture.

Table 5. 5 - CloudSea Data Distribution based on Pub-Sub Messaging

| **Data Sets** | **Text Corpus** | | **Frontend** | | **User Management** | | **Entity Extraction** | | **Ontology Population** | | **Ontology Selection** | | **Ontology Mapping** | | **Annotation Data Storage** | | **Annotation On-the-fly** | | **Annotation Data Reuse** | | **Annotation Data Sharing** | | **Annotation Data Auto-Update** | | **Ontology Auto-Update** | | **Annotation Data Optimisation** | | **Service Colocation** | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Publish | Subscribe | Publish | Subscribe | Publish | Subscribe | Publish | Subscribe | Publish | Subscribe | Publish | Subscribe | Publish | Subscribe | Publish | Subscribe | Publish | Subscribe | Publish | Subscribe | Publish | Subscribe | Publish | Subscribe | Publish | Subscribe | Publish | Subscribe | Publish | Subscribe |
| Concepts and Relations | ✓ | | | | | | | | | ✓ | | | | | | | | | | | | | | | | | | | | |
| Ontology Updates | | | | | | | | | | ✓ | | | | | | | | | | | | | | ✓ | ✓ | | | | | |
| Entities | ✓ | | | | | | | ✓ | | | | | | | | | | | | | | | | | | | | | | |
| Ontology Profiles | | | | | | | | | ✓ | | | ✓ | | | | | | | | | | | | | | | | | | |
| Ontology Selections | | | | | | | | | | | ✓ | | | ✓ | | | | | | | | | | | | | | | | |
| Mapping Data | | | | | | | | | | | | | ✓ | | | ✓ | | | | | | | | | | | | | | |
| Annotation Data | | | | | | | | | | | | | | | ✓ | | | ✓ | | ✓ | | ✓ | | ✓ | | | | ✓ | | ✓ |
| Optimisation Data | | | | | | | | | | | | | | | | ✓ | | | | | | | | | | | ✓ | | | |
| User Data | | | | ✓ | ✓ | | | | | | | | | | | | | | | | | | | | | | | | | ✓ |
| User Request | | | | ✓ | ✓ | | | | | | | | | | | | | | | | | | | | | | | | | |
| User Response | | | | ✓ | ✓ | | | | | | | | | | | | | | | | | | | | | | | | | |

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| URI Repository | | | | | | | | | | | | | | | | | | | ✓ | ✓ | | | | | | | | | | |
| Annotation Update Logs | | | | | | ✓ | | | | | | | | | | | | | | | | ✓ | | | | | | | | |
| RDF Data Repository | | | | | | | ✓ | | | | | | | | | ✓ | | | | | | | | | | | | | | |
| Consumer Statistics | | | | | | | | | | | | | | | | | | | | | | | | | | | | | ✓ | |
| Reuse Statistics | | | | | | ✓ | | | | | | | | | | | | | ✓ | | | | | | | | | | | |

## 5.4.4 Service Discovery

Service discovery refers to the process of determining the host present in the container orchestrator that should be scheduled next. In microservices architectures, it ascertains that microservices applications process requests effectively such that it can handle workload changes (Khan, 2017). Service discovery can be accomplished through two main ways. These are a client-side discovery and server-side discovery technique. In a client-side discovery, a client determines the location of the networks hosting available services and determines the load balancing requests. The client then queries the service registry which hosts a database containing instances of available services. A load balancer is then used to select the appropriate service instance and the client’s request is channelled to the instance (Ghomi et al., 2017). On the other hand, server-side discovery is a process where a client uses the load balancer to make a request for services. The load balancer then queries for the availability of service instances in the service registry and each request is routed wherever a service instance is available (Milani & Navimipour, 2016). The similarity between the two options is that the service registry registers and deregisters service instances. With a microservices architecture comprising of multiple micro-applications like CloudSea does, the Service Discovery capability in the orchestration of containers is very vital. From the comparison of both client-side and server-side approaches to service discovery, the server-side approach is proposed based on the scale of requests that the system is built for with availability of more computing resources for the server-side approach. With the holistic semantic annotation architecture, it works with a service registry to keep the identities (names) of the different micro-applications and their IP address. So, once a request is channelled towards a specific micro-application, the service discovery mechanism looks up the address of the micro-application and directs the request to it accordingly.

## 5.5 CloudSea: The Proposed Holistic, Cloud-Driven, Microservices-Based Architecture for Automated Semantic Annotation of Web Documents

Section 5.4 has provided detailed descriptions of the architectural components for CloudSea; from the design patterns developed for each of the CloudSea microservices. The overall architecture encapsulates the novel concepts developed from this thesis. The requirement specifications for a holistic semantic annotation proposed in Chapter 3 of this thesis have been assessed and transformed into capabilities. These capabilities are demonstrated by the design patterns of section 5.4. In addition, the design patterns are microservices-based and derived from the cloud-driven pattern of the proposed Holistic Semantic Annotation Cloud Computing Maturity Model in Chapter 4 of this thesis. Therefore, this research proposes a holistic, cloud-driven and microservices-based architecture for automated semantic annotation of web documents, as presented in Figure 5.17.

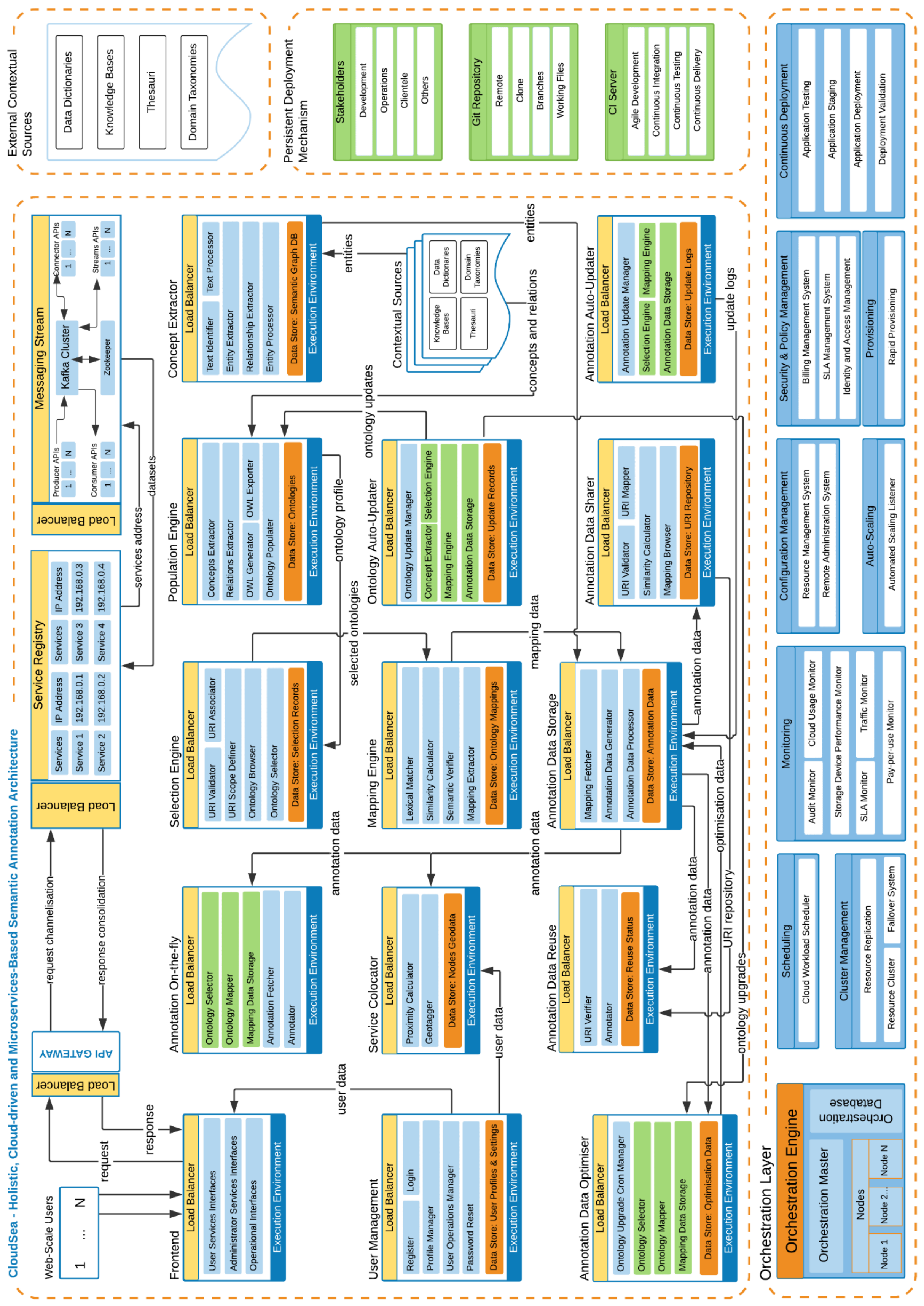


Figure 5. 17 – CloudSea: Holistic, Cloud-driven and Microservices-based Architecture for Automated Semantic Annotation

From Figure 5.17, the orchestration layer offers several capabilities for the overall running of the CloudSea architecture in a cloud computing environment. The orchestration engine comprises of a database for storing data and several nodes for provisioning and deploying the CloudSea microservices. With Cluster Management capabilities, a central system operates as the server for the orchestration engine, managing the entire workflow for the layer. Such management capabilities include resource replication, resource clustering and creating failover systems. The scheduling of tasks such as running scheduled processes (like cron jobs or schtasks) or assigning resources for processes is also a key capability of the orchestration engine and this is commonly handled by a Cloud Workload Scheduler.

Furthermore, several monitoring activities are engaged in within the orchestration layer. This includes monitoring for the usage of cloud resources, traffic monitoring and SLA monitoring. Third-party solutions can be integrated to enhance monitoring in diverse ways, such as processing the ingestion of data and providing reporting tools via enhanced visualisation interfaces. Results obtained from monitoring metrics can also influence configuration parameters. For instance, a microservice can be configured to automatically scale to multiple instances when CPU utilisation reaches 80%. Based on performance monitoring metrics, this value may need to be adjusted, either by increasing or reducing it. Furthermore, regarding the automatic scaling of microservices, it provides a basis for large-scale service delivery which is directly proportional to real time client request traffic, hence, optimising cloud computing resources utilisation.

Capabilities for the management of security and policy issues are also facilitated by the orchestration layer; providing mechanisms such as identity and access management, attribute-based access control and SLA management systems. In addition, with the need for an automated and dynamic means of a continuous stream of updates to several components within the overall architecture, the orchestration layer facilitates continuous integration through a synchronisation with a remote CI (Continuous Integration) server that pushes updates to the microservices as often as needed. These foster the holistic perspective proposed for an automated semantic annotation cloud service for web documents at large-scale which fully maximises cloud computing benefits based on the cloud-driven pattern of deployment that it is built on. Furthermore, the process flow for CloudSea Architecture is demonstrated with the flowchart in Figure 5.18.

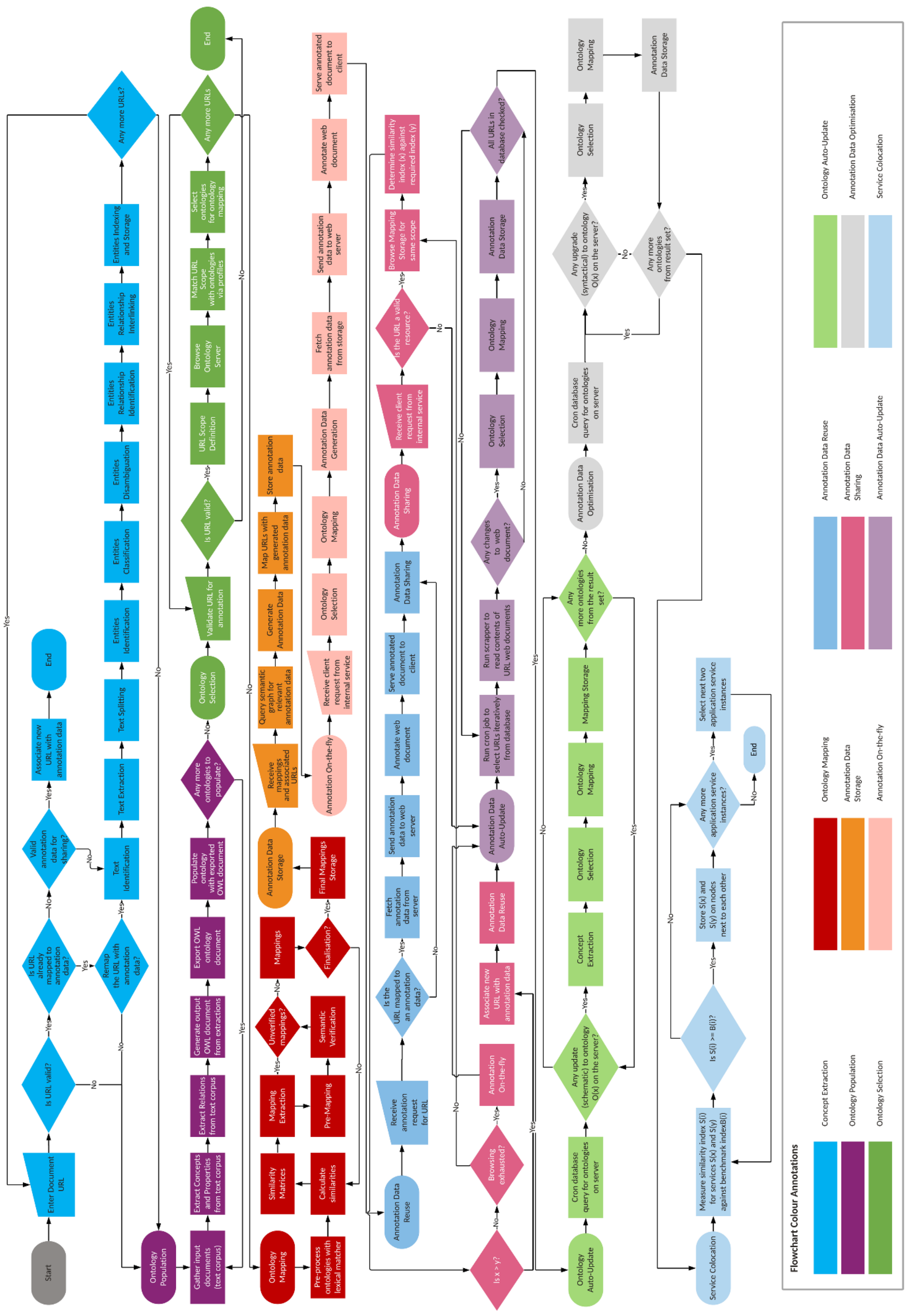

Start
Enter Document URL
Is URL valid?
Is URL already mapped to annotation data?
Valid annotation data for sharing?
Associate new URL with annotation data
End
Remap the URL with annotation data?
Text Identification
Text Extraction
Text Splitting
Entities Identification
Entities Classification
Entities Disambiguation
Entities Relationship Identification
Entities Relationship Interlinking
Entities Indexing and Storage
Any more URLs?
Ontology Population
Gather input documents (text corpus)
Extract Concepts and Properties from text corpus
Extract Relations from text corpus
Generate output OWL document from extractions
Export OWL ontology document
Populate ontology with exported OWL document
Any more ontologies to populate?
Ontology Selection
Validate URL for annotation
Is URL valid?
URL Scope Definition
Browse Ontology Server
Match URL Scope with ontologies via profiles
Select ontologies for ontology mapping
Any more URLs
End
Ontology Mapping
Similarity Matrices
Mapping Extraction
Unverified mappings?
Mappings
Annotation Data Storage
Receive mappings and associated URLs
Query semantic graph for relevant annotation data
Generate Annotation Data
Map URLs with generated annotation data
Store annotation data
Pre-process ontologies with lexical matcher
Calculate similarities
Pre-Mapping
Semantic Verification
Finalisation?
Final Mappings Storage
Annotation On-the-fly
Receive client request from internal service
Ontology Selection
Ontology Mapping
Annotation Data Generation
Fetch annotation data from storage
Send annotation data to web server
Annotate web document
Serve annotated document to client
Annotation Data Reuse
Receive annotation request for URL
Is the URL mapped to an annotation data?
Fetch annotation data from server
Send annotation data to web server
Annotate web document
Serve annotated document to client
Annotation Data Sharing
Annotation Data Sharing
Receive client request from internal service
Is the URL a valid resource?
Browse Mapping Storage for same scope
Determine similarity index (x) against required index (y)
Is x > y?
Browsing exhausted?
Annotation On-the-fly
Associate new URL with annotation data
Annotation Data Reuse
Annotation Data Auto-Update
Run cron job to select URLs iteratively from database
Run scrapper to read contents of URL web documents
Any changes to web document?
Ontology Selection
Ontology Mapping
Annotation Data Storage
All URLs in database checked?
Ontology Auto-Update
Cron database query for ontologies on server
Any update (schematic) to ontology O(x) on the server?
Concept Extraction
Ontology Selection
Ontology Mapping
Mapping Storage
Any more ontologies from the result set?
Annotation Data Optimisation
Cron database query for ontologies on server
Any upgrade (syntactical) to ontology O(x) on the server?
Ontology Selection
Ontology Mapping
Annotation Data Storage
Any more ontologies from result set?
Service Colocation
Measure similarity index S(i) for services S(x) and S(y) against benchmark indexB(i)
Is S(i) >= B(i)?
Store S(x) and S(y) on nodes next to each other
Any more application service instances?
Select next two application service instances
End
Yes
No
Flowchart Colour Annotations
Concept Extraction
Ontology Population
Ontology Selection
Ontology Mapping
Annotation Data Storage
Annotation On-the-fly
Annotation Data Reuse
Annotation Data Sharing
Annotation Data Auto-Update
Ontology Auto-Update
Annotation Data Optimisation
Service Colocation

Figure 5. 18 – Process Flow for CloudSea Architecture

## 5.5 Chapter Summary

This chapter provided detailed description for one of the major contributions of the research. With the conceptualisation of CloudSea in Chapter 4. Chapter 5 started with a design rationale for CloudSea; lending the theory of ‘Design Patterns’ from software engineering. The engineering methodology for CloudSea Design Patterns was also described. In addition, CloudSea Pattern Language; which provides an inter-relationship between the different design patterns and how each constitutes a building block in the overall conceptual system was developed. These set the stage for developing the CloudSea Design Patterns, as ‘Microservices-Based’; in line with the background concepts developed in Chapter 4 as well as the Cloud Computing Maturity Model. The technical details for each of the microservices-based design patterns was also detailed based on ‘Design Patterns’ standard structure and format. The CloudSea Architecture for automated semantic annotation of web documents resulted from the design patterns and was presented. Furthermore, a process flow for the architecture was presented in the form of a flowchart.

# Chapter 6: Prototype Implementation

This chapter provides details of a prototype implementation for CloudSea. This is based on the development and deployment of functionalities for the Annotation On-the-Fly and Annotation Data Storage microservices of CloudSea Architecture presented in Figure 5.17. The Annotation On-the-Fly microservice utilises an API call for service delivery. The prototype implementation is based on the "Cloud-Driven" model proposed in Chapter 4 of this thesis, which, alongside leveraging microservices software architecture, also leverages hypervisor-level and operating system-level virtualisations based on Amazon AWS virtual hosts and Docker Containers respectively, container orchestration using Kubernetes platform as well as the persistent deployment mechanism proposed in Chapter 4; based on a synchronisation between a private GitHub repository and CircleCi to provide pipeline for a continuous stream of updates and upgrades to the microservices within the architecture.

## 6.1 Requirements

This section provides the detailed requirements, both software and platform requirements for the prototype implementation. The software requirements specify the features and behaviour for the prototype while the platform requirements specify the features for the deployment environment which is set up on Amazon AWS.

### 6.1.1 Software Requirements

- The prototype should allow users register for an account
- The prototype should allow users manage their account; such as logging in, updating user profile, password reset and logging out
- The prototype should allow users submit URLs for annotation data generation
- The prototype should allow users view a list of all URLs they've submitted for semantic annotation
- The prototype should allow users to re-submit a URL or URLs for annotation data generation
- The prototype should automatically provide semantic annotation to a web document with a call to the Annotation API via its endpoint
- The prototype should provide automatic semantic annotation to a web document for every instance of the document being accessed online

- The prototype should provide a categorisation of 'Entity Types' on web documents being semantically annotated
- The prototype should allow users turn 'on' or 'off' any of the 'Entity Types' annotated within a document
- The prototype should allow users define the text on a web document that they want to have semantically annotated
- The prototype should be developed based on Microservices Software Architecture
- The prototype needs to be responsive in nature; being usable on diverse types of devices such as PCs, laptops, tablets, mobile devices, etc.
- The prototype should fulfil the 'Cloud-Driven' Methodology
- The prototype should be built using the PHP framework 'Laravel'
- The prototype should conform to good programming standards and best practices

### 6.1.2 Platform Requirements

- The prototype should be deployed within a cloud computing environment
- The prototype should leverage both hypervisor-level and operating system-level virtualisation
- The prototype should leverage a container orchestration platform in the cloud
- The prototype should have a full application automation lifecycle for continuous integration and delivery
- The prototype should be scalable to adequately respond to huge amounts of client requests over a period
- Facilities should be available to manage the prototype remotely after deployment in a cloud environment
- Facilities should be available to monitor the prototype after deployment in a cloud environment

## 6.2 The Development Environment

This section describes the development environment for the actual build of the prototype. A local workstation running Windows 10 operating system was utilised for developing the prototype. The development required installing and configuring some software. A stack of technologies known as XAMPP, bundled together and available online as a WinZip file was downloaded and installed. The components of the bundle are as follows:

- Apache: A very popular web server for running web applications and responding to client requests over protocols such as HTTP and HTTPS for data via the applications running on it. It is developed and maintained by the Apache Software Foundation and runs on up to 67% of web servers on the Web (CBROnline, 2019).
- MySQL: It stands for My Structured Query Language and is an open-source relational database management system for storing data. It is commonly used as the database for web applications, either within content management systems (such as Drupal, Joomla, Magento and WordPress) or for custom-built applications. It utilises SQL queries for carrying out CRUD (Create, Record, Update, Delete) functions on its database tables.
- PHP: PHP stands for Hypertext Pre-Processor and is an open-source server-side scripting language popularly used on the Web. It is regarded as a scripting language because its code is interpreted at runtime, offering a rich set of in-built functions for processing and managing data transactions on the Web. PHP code is either processed by its interpreter which runs as a module on a web server (in this case Apache Web Server) or as a CGI (Common Gateway Interface) executable. It also offers a data access layer which enables it to be ported with different types of databases, such as MySQL, PostgreSQL, SQL Server and lots more.

## 6.3 Enabling Technologies

A wide range of enabling technologies were utilised for developing the prototype. These includes programming languages, libraries, frameworks, open source software and lots more, utilised for functionalities across the different architectural layers of the prototype. These are detailed in the section, as follows.

### 6.3.1 Laravel Framework

The prototype has been developed using Laravel Framework. Laravel is an open source PHP framework which is very popular for developing high-end, custom-built web applications. It uses a model-view-controller design pattern and benefits from a rich list of components commonly built into standard PHP frameworks. Laravel offers a platform for developing quick, scalable and efficient web applications with its rich set of features. It has a set of system requirements which needs to be met before running Laravel, most of which are required PHP extensions (such as PDO, JSON and MbString PHP extensions) as well as a PHP engine. Laravel 5.7 was installed, configured and utilised for the implementation and it required PHP version 7.1.3 or higher as well as several other dependencies.

### 6.3.2 The Frontend Tools

Frontend refers to the web graphical user interface for interacting with the prototype application logic. This was developed as a microservice on its own, in line with MVC design pattern and microservices key principles. The development of the frontend was based on standard technologies such as HTML, CSS, JavaScript and jQuery.

### 6.3.3 Redis

Redis is an acronym that stands for Remote Dictionary Server. It acts like a dictionary as it stores data in "key-value" pairs within an in-memory data store for both strings and abstract data types. It is very portable and a viable option for small to medium-sized distributed systems. While Apache Kafka was proposed for CloudSea due to its benefits and robustness to manage extremely large datasets, the prototype implementation was done using Redis in its place. This is due to the minimal data being managed within the prototype in comparison with what is expected for an actual product based on the CloudSea architecture. Implementing Apache Kafka with its steep learning curve and vast requirement for resources for the prototype implementation was observed to be quite imbalanced. Redis was deemed to be an appropriate alternative for the purpose of a prototype implementation. It was utilised for inter-process communication between the microservices, storing and transmitting data such as user sessions, cookies and authentication tokens.

### 6.3.4 JavaScript Object-Notation (JSON)

JSON is a lightweight and portable file format which is commonly used for storing and transferring data across multiple systems. The file format is interoperable which makes it a versatile option for use in sharing and transporting data amongst several different types of vendor-specific applications. Data within JSON files are stored as objects, comprising of an "attribute-value" pair like the same type of pairing with HTML format. JSON files also contain array data types for the data objects. JSON has been utilised for storing annotation data retrieved from DBpedia knowledge graph. Objects within the JSON files constitute contextual data which is utilised to provide semantic annotation for the corresponding web documents.

## 6.4 The Annotation API

While a web interface was developed for users' registration, generation of annotation data for URLs and their management, for the purpose of demonstrating the implemented functionality, the annotation API approach to the semantic annotation process was the focus; in which case a guest can still utilise the service and need not sign up for an account before being able to do so. Please refer to Appendix B for screenshots demonstrating the web interface functionalities. A request to the API service is required to provide semantic annotation by means of inserting the API call between the <head> tags of a web document to be semantically annotated. The request call invokes the Annotation on-the-fly microservice which processes the request on the business logic layer of the microservice architecture as well as managing transactions with other layers and with external components such as the DBpedia graph database. It responds by providing the required semantic annotation to the web document. Figure 6.1 presents a flowchart for the semantic annotation process either as a registered or guest user.

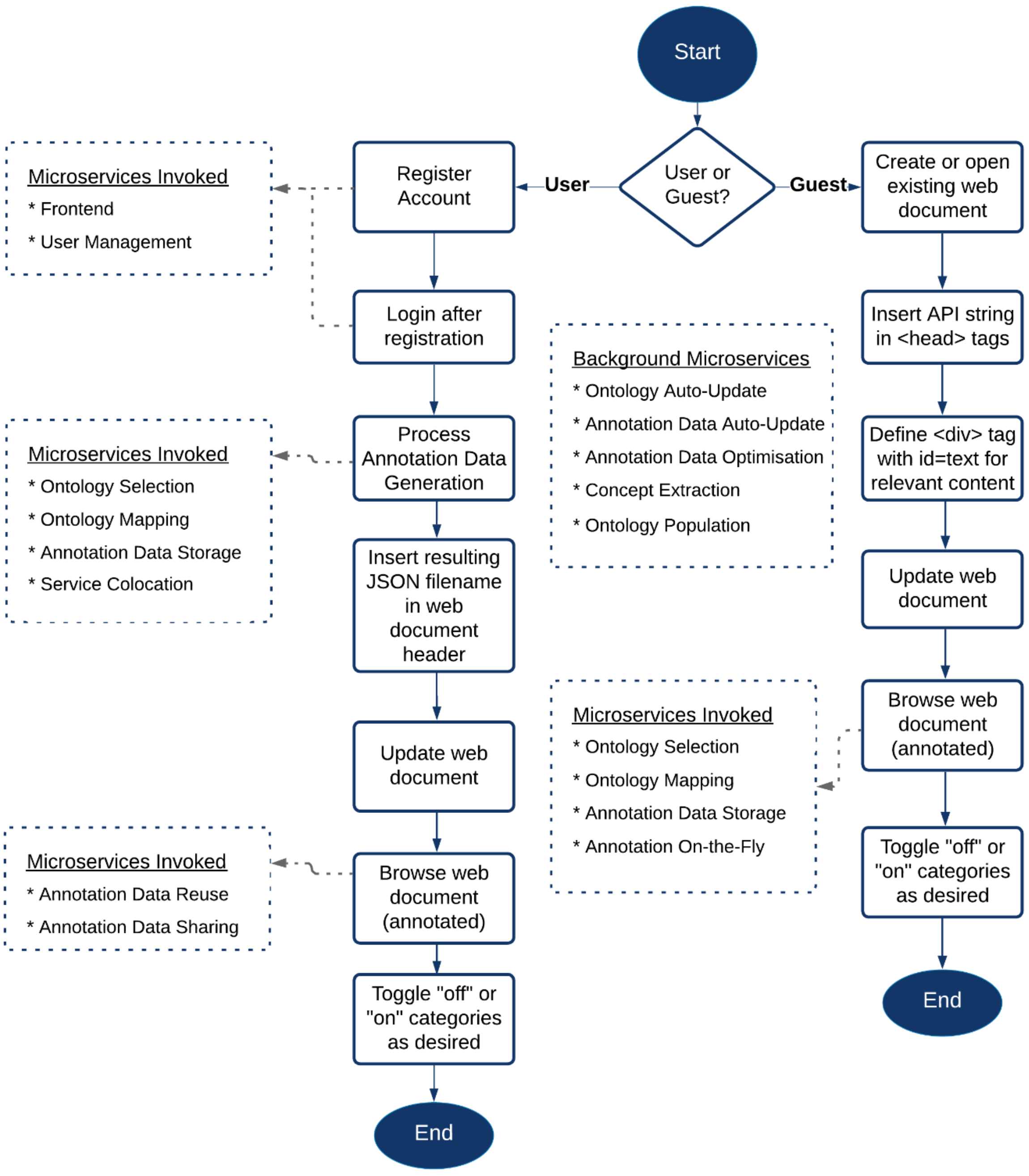
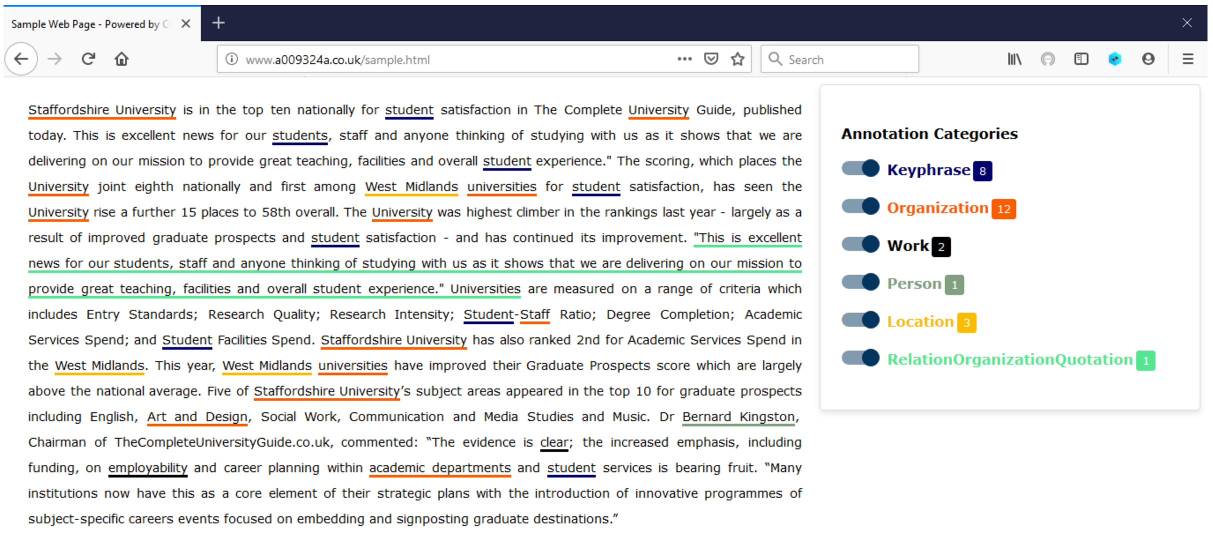

Start
User or Guest?
User
Guest
Register Account
Login after registration
Microservices Invoked
* Frontend
* User Management
Process Annotation Data Generation
Microservices Invoked
* Ontology Selection
* Ontology Mapping
* Annotation Data Storage
* Service Colocation
Insert resulting JSON filename in web document header
Update web document
Browse web document (annotated)
Microservices Invoked
* Annotation Data Reuse
* Annotation Data Sharing
Toggle "off" or "on" categories as desired
End
Background Microservices
* Ontology Auto-Update
* Annotation Data Auto-Update
* Annotation Data Optimisation
* Concept Extraction
* Ontology Population
Create or open existing web document
Insert API string in <head> tags
Define <div> tag with id=text for relevant content
Update web document
Browse web document (annotated)
Microservices Invoked
* Ontology Selection
* Ontology Mapping
* Annotation Data Storage
* Annotation On-the-Fly
Toggle "off" or "on" categories as desired
End

Figure 6. 1 – User Perspective for CloudSea Semantic Annotation Process

Please refer to section 7.1.2 for further explanation of the semantic annotation process. Furthermore, Figure 6.2 presents a screenshot of a sample web page semantically annotated using the Annotation API.


Sample Web Page - Powered by C

www.a009324a.co.uk/sample.html

Staffordshire University is in the top ten nationally for student satisfaction in The Complete University Guide, published today. This is excellent news for our students, staff and anyone thinking of studying with us as it shows that we are delivering on our mission to provide great teaching, facilities and overall student experience." The scoring, which places the University joint eighth nationally and first among West Midlands universities for student satisfaction, has seen the University rise a further 15 places to 58th overall. The University was highest climber in the rankings last year - largely as a result of improved graduate prospects and student satisfaction - and has continued its improvement. "This is excellent news for our students, staff and anyone thinking of studying with us as it shows that we are delivering on our mission to provide great teaching, facilities and overall student experience." Universities are measured on a range of criteria which includes Entry Standards; Research Quality; Research Intensity; Student-Staff Ratio; Degree Completion; Academic Services Spend; and Student Facilities Spend. Staffordshire University has also ranked 2nd for Academic Services Spend in the West Midlands. This year, West Midlands universities have improved their Graduate Prospects score which are largely above the national average. Five of Staffordshire University's subject areas appeared in the top 10 for graduate prospects including English, Art and Design, Social Work, Communication and Media Studies and Music. Dr Bernard Kingston, Chairman of TheCompleteUniversityGuide.co.uk, commented: "The evidence is clear; the increased emphasis, including funding, on employability and career planning within academic departments and student services is bearing fruit. "Many institutions now have this as a core element of their strategic plans with the introduction of innovative programmes of subject-specific careers events focused on embedding and signposting graduate destinations."

**Annotation Categories**

- Keyphrase 8
- Organization 12
- Work 2
- Person 1
- Location 3
- RelationOrganizationQuotation 1

Figure 6. 2 – Automated Semantic Annotation using CloudSea API

## 6.5 The Deployment Environment

After developing the prototype on a local workstation, it was transferred to the deployment environment. In line with the platform requirements in section 6.1.2 of this chapter, the prototype has been developed to run from a cloud computing environment, hence, it had to be deployed accordingly. While the prototype could have been deployed on any classification of cloud platforms; public, private, hybrid or community clouds, it was deployed in a public cloud; Amazon AWS. This is due to its proposed purpose, which is for a global and public access to documents on the web and via the Internet. In addition, of the several available public cloud service providers such as Google, Amazon, Microsoft, IBM, DigitalOcean, etc., any of them could have been utilised. However, due to a significantly higher market share and the level of technical details readily available on the web regarding their products, services and models for offering these services, Amazon AWS was the preferred choice as a deployment environment. Figure 6.3 presents a survey by CBN Insights (2019) which shows the market share for the top cloud service providers in the market as at February 2019.

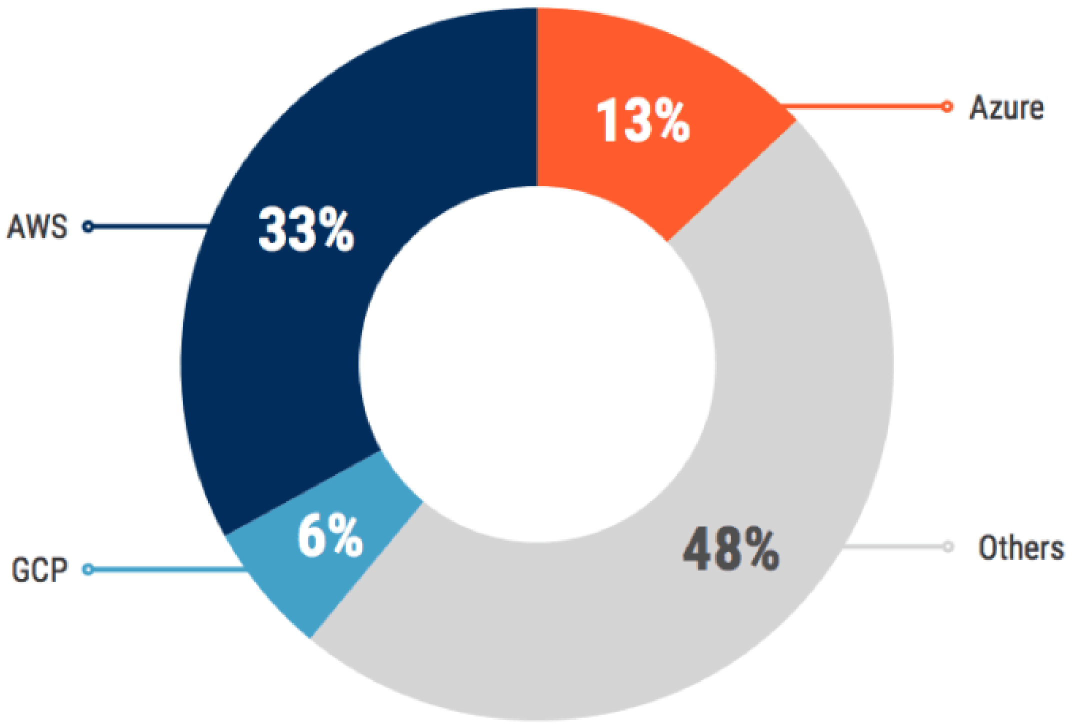


Figure 6. 3 – Market Share for Top Cloud Service Providers (CBN Insights, 2019)

### 6.5.1 Amazon AWS

After completing the development, a deployment environment was set-up on Amazon AWS for the prototype. To set up the deployment environment, the following steps were taken:

- The creation of an Amazon AWS account via: https://aws.amazon.com
- The setting up and configuration of four EC2 instances for the deployment and its orchestration using Kubernetes cluster. One of the instances was set up for the Kubernetes master node and the other two for the worker nodes. The specification for the EC2 instances is presented in Table 6.1.

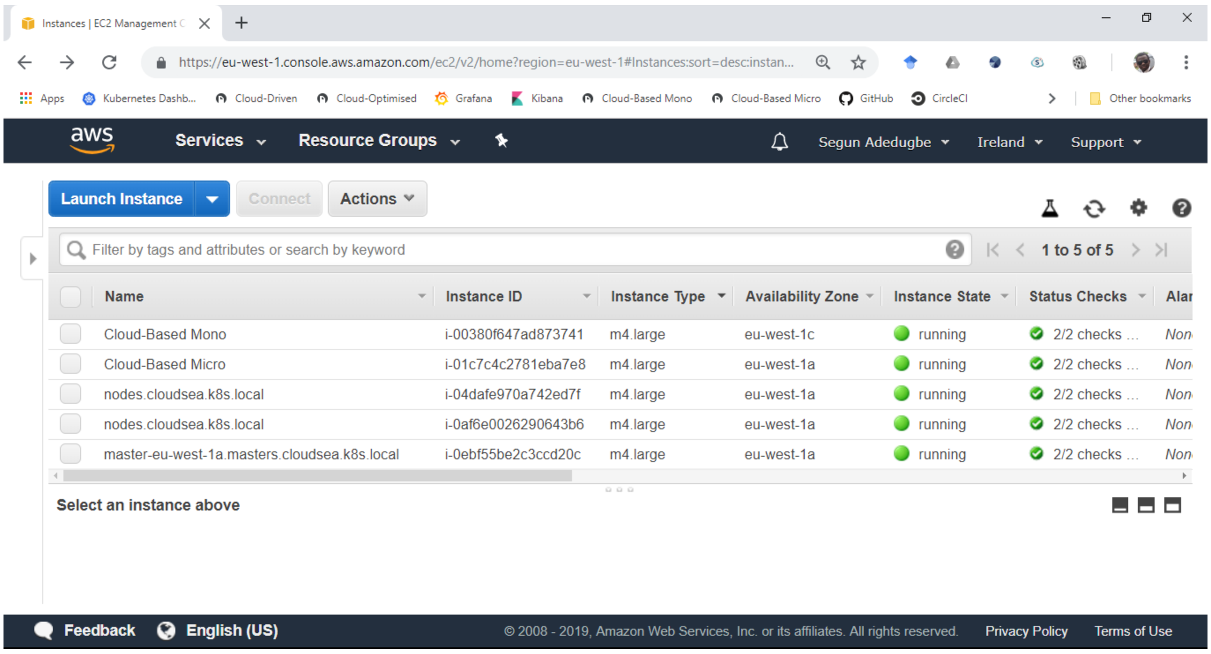
Table 6. 1 – System Specification of EC2 Instances for CloudSea Kubernetes Cluster

| **Specification** | **Value** |
|---|---|
| Model | M4 Large |
| Virtual CPUs | 2 (Each is a thread of either an Intel Xeon core or an AMD EPYC core) |
| Memory | 8 GB |
| Storage | Amazon EBS (Elastic Block Store) |
| AMI | Debian Jessie AMD64 |

| | |
|---|---|
| Hypervisor-Level Virtualisation | HVM |
| Dedicated EBS Bandwidth | 450 Mbps |
| Network Performance | Moderate |

- The installation and configuration of Apache Web Server, PHP and MySQL. For detailed installation, configuration and deployment steps, please refer to Appendix C.

Figure 6.4 shows a screenshot of the Amazon AWS account with five EC2 instances. Three for the CloudSea Kubernetes master node with two worker nodes. The fourth and fifth EC2 instances were for deployments of the prototype based on the 'Cloud-Based Monolithic' and 'Cloud-Based Microservices' maturity levels of the Cloud Computing Maturity Model respectively. These would be utilised in the next chapter for empirical investigation.



| Name | Instance ID | Instance Type | Availability Zone | Instance State | Status Checks | Alar |
|---|---|---|---|---|---|---|
| Cloud-Based Mono | i-00380f647ad873741 | m4.large | eu-west-1c | running | 2/2 checks ... | Non |
| Cloud-Based Micro | i-01c7c4c2781eba7e8 | m4.large | eu-west-1a | running | 2/2 checks ... | Non |
| nodes.cloudsea.k8s.local | i-04dafe970a742ed7f | m4.large | eu-west-1a | running | 2/2 checks ... | Non |
| nodes.cloudsea.k8s.local | i-0af6e0026290643b6 | m4.large | eu-west-1a | running | 2/2 checks ... | Non |
| master-eu-west-1a.masters.cloudsea.k8s.local | i-0ebf55be2c3ccd20c | m4.large | eu-west-1a | running | 2/2 checks ... | Non |

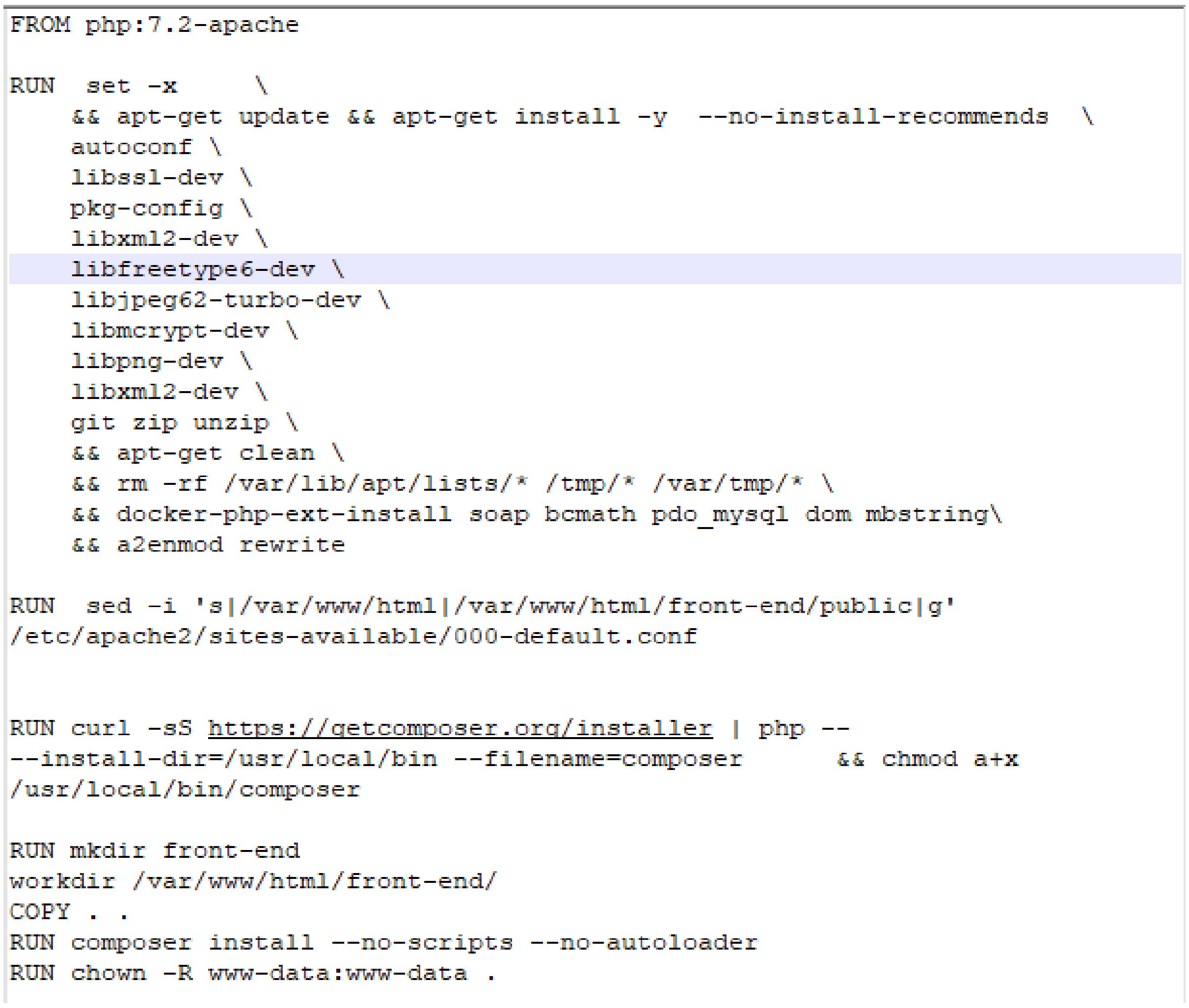
Select an instance above

Feedback
English (US)
© 2008 - 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.
Privacy Policy
Terms of Use

Figure 6. 4 – Amazon AWS account running EC2 instances for CloudSea

### 6.5.2 Prototype Containerisation

As detailed in Chapter 5, containerisation is a type of virtualisation which allows applications to run independent of the virtual host operating system, hence easily deployable and scalable across platforms. The prototype microservices were containerised for operating system-level virtualisation using Docker containers. A Dockerfile was created for each of the microservices to build docker images for them. These Docker images are required to build the actual containerised applications and includes the necessary configuration parameters and dependencies for each microservice to be deployable and runnable as a container within the orchestration platform. Figure 6.5 presents a screen shot of the “Frontend” microservice containerisation configuration parameters using Dockerfile.



```
FROM php:7.2-apache

RUN  set -x     \
    && apt-get update && apt-get install -y  --no-install-recommends  \
    autoconf \
    libssl-dev \
    pkg-config \
    libxml2-dev \
    libfreetype6-dev \
    libjpeg62-turbo-dev \
    libmcrypt-dev \
    libpng-dev \
    libxml2-dev \
    git zip unzip \
    && apt-get clean \
    && rm -rf /var/lib/apt/lists/* /tmp/* /var/tmp/* \
    && docker-php-ext-install soap bcmath pdo_mysql dom mbstring\
    && a2enmod rewrite

RUN  sed -i 's|/var/www/html|/var/www/html/front-end/public|g'
/etc/apache2/sites-available/000-default.conf


RUN curl -sS https://getcomposer.org/installer | php --
--install-dir=/usr/local/bin --filename=composer      && chmod a+x
/usr/local/bin/composer

RUN mkdir front-end
workdir /var/www/html/front-end/
COPY . .
RUN composer install --no-scripts --no-autoloader
RUN chown -R www-data:www-data .
```

Figure 6. 5 – Screen shot of the Dockerfile for CloudSea Frontend Microservice

### 6.5.3 Installing, Configuring and Running an Orchestration Platform: Kubernetes

With the prototype microservices already containerised using Docker containers and the cloud computing infrastructure layer resources in place, the next step was to set up an orchestration platform for the containers. While there are several container orchestrators available, Kubernetes was the preferred choice for this implementation. Some of the other available options include Docker Swarm, Apache Mesos Marathon, Amazon Elastic Container Service, Microsoft Azure Container Service and HashiCorp Nomad. The choice of Kubernetes was based on the extensive list of features it provides and supports; with a very large support community as an open source software. In addition, the technical documentation available for Kubernetes was observed to be very comprehensive for use either as a beginner, intermediate or an expert. Furthermore, it is very popular and widely adopted for use in the industry compared to the others. For instance, Docker provides Docker Swarm for container orchestration. However, it has adopted Kubernetes as the official container orchestrator for its docker containers (Handy, 2019).

Kubernetes is an open source software that automates the deployment, scaling, monitoring and management of containerised applications (Kubernetes.io, 2019). It manages application container operations across clusters of hosts; helping to replicate containers, providing automatic scaling for cluster containers, facilitating load balancing across multiple containers, rolling application container upgrades, rescheduling failed containers, controlling the exposure of network ports to external systems and lots more (Wu, 2017). The fundamental building block of Kubernetes is called a pod. The function of a pod is to encapsulate containers that are tightly coupled and are co-located, sharing the same set of resources. Some other resources that pods encapsulate include storage resources and network IPs. During the creation of a pod, it is necessary to specify the amount of CPU, memory, and ephemeral storage it requires (Kubernetes.io, 2019). The specifications are then available to the scheduler for decisions on where to place pods within the cluster. Figure 6.6 shows the popularity among different container orchestrators, with Kubernetes being significantly more popular than the others.

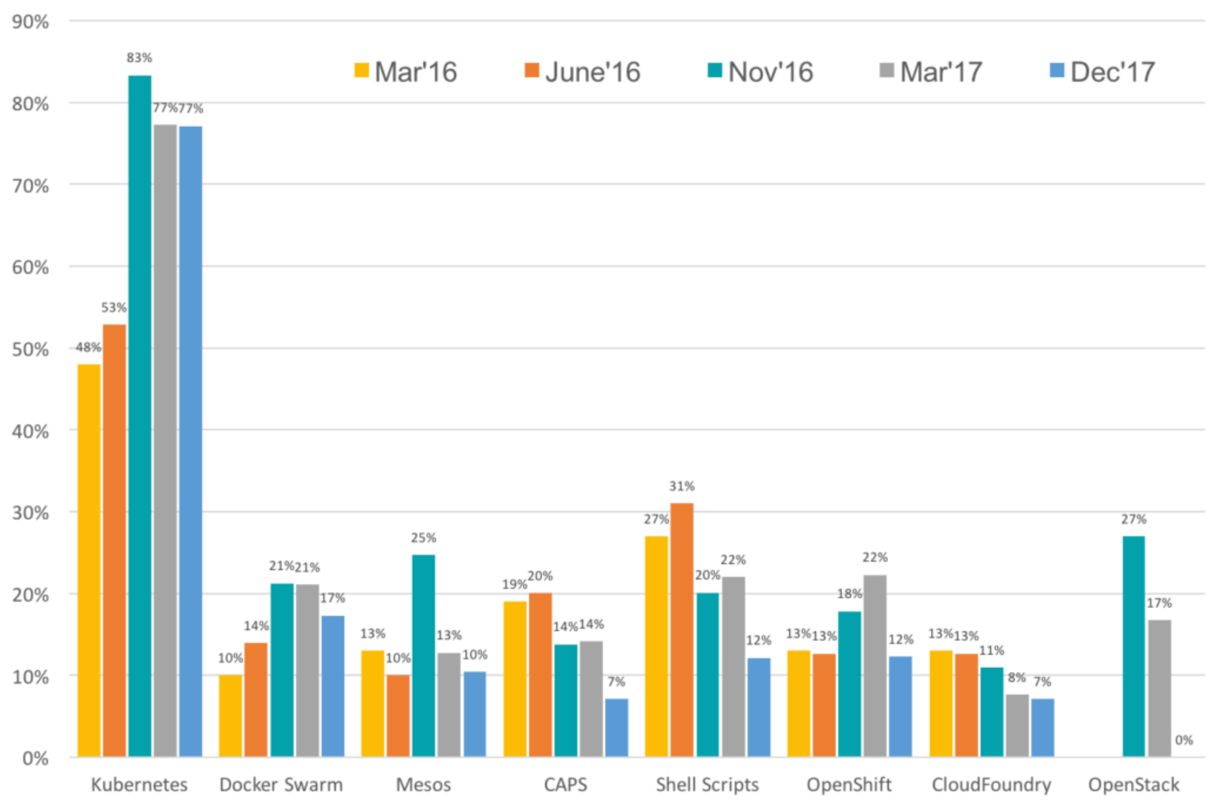


Figure 6. 6 – Container Orchestration Engines Popularity (CNCF.io, 2018)

Kubernetes was installed and configured as a cluster, with the master node running on an EC2 instance and two worker nodes each running on separate EC2 instances. For the CloudSea Kubernetes cluster, some of the most prominent components include the nodes, namespaces, deployments, pods and addons. These are described as follows.

### 6.5.3.1 The Kubernetes Nodes

The CloudSea Kubernetes cluster comprises of three nodes; a master node and two worker nodes. The master node is on an EC2 instance and runs the control plane of the Kubernetes cluster. A Kubernetes cluster control plane is the central point of the cluster as it contains components that control, manage and provide needed functionalities for the entire operations of the cluster. The components involved are as follows:

- API Server: It provides an interface for managing the state of the cluster. This includes querying and modifying component data over a RESTful API, providing the ability to execute CRUD (Create, Read, Update and Delete) functions on different cluster components. Kubectl; a command-line interface utility makes use of the API server for its operations as well. This was installed and configured on a local machine for interacting with the Kubernetes API Server.

- Scheduler: It assigns pods to specific worker nodes and updates the pod definition within the API server accordingly. The pod-to-node assignment is based on the availability of computing resources for running each of the pods. These includes both system and application pods.
- Controller Manager: It comprises of several types of controllers executing operations for a specific type of resource. Some of these controllers include pod, deployment, replication, namespace and service controllers. They carry out various functions for these resources by observing any changes within the resources and carrying out operations in response to any observations to ensure the continuous optimal functioning of the resources.
- Etcd: It is the database for the cluster; storing stateful data as well as both configuration data and metadata, allowing each of the nodes within the cluster to read and write data to it.

The worker nodes each comprise of a Kubelet, Kube Proxy and Pods. Their roles are as follows:

- Kubelet: It functions as the engine room for each worker node by managing and monitoring activities. It registers the node with the API server on the master node and then listens on the API server for any pods that have been scheduled to run on it. Once pods have been scheduled to run on a node, the kubelet assigns the necessary resources including the container runtime to run the pod and continuously monitors metrics such as status, health, etc. of each pod and transmits this back to the API server.
- Kube Proxy: This is responsible for establishing connections between clients and the services they have requested for. It keeps a record of all running pods and their locations within the node. When a service request has multiple pods running for it, the Kube Proxy performs load balancing to distribute requests among the different pods.
- Pods: These are the smallest units of deployment on the cluster. A pod would contain one or more containers that are deployed together. The CloudSea prototype runs as pods across the two worker nodes and are automatically distributed by the master node based on the availability of computing resources.

The architecture of the CloudSea Kubernetes cluster is presented in Figure 6.7.

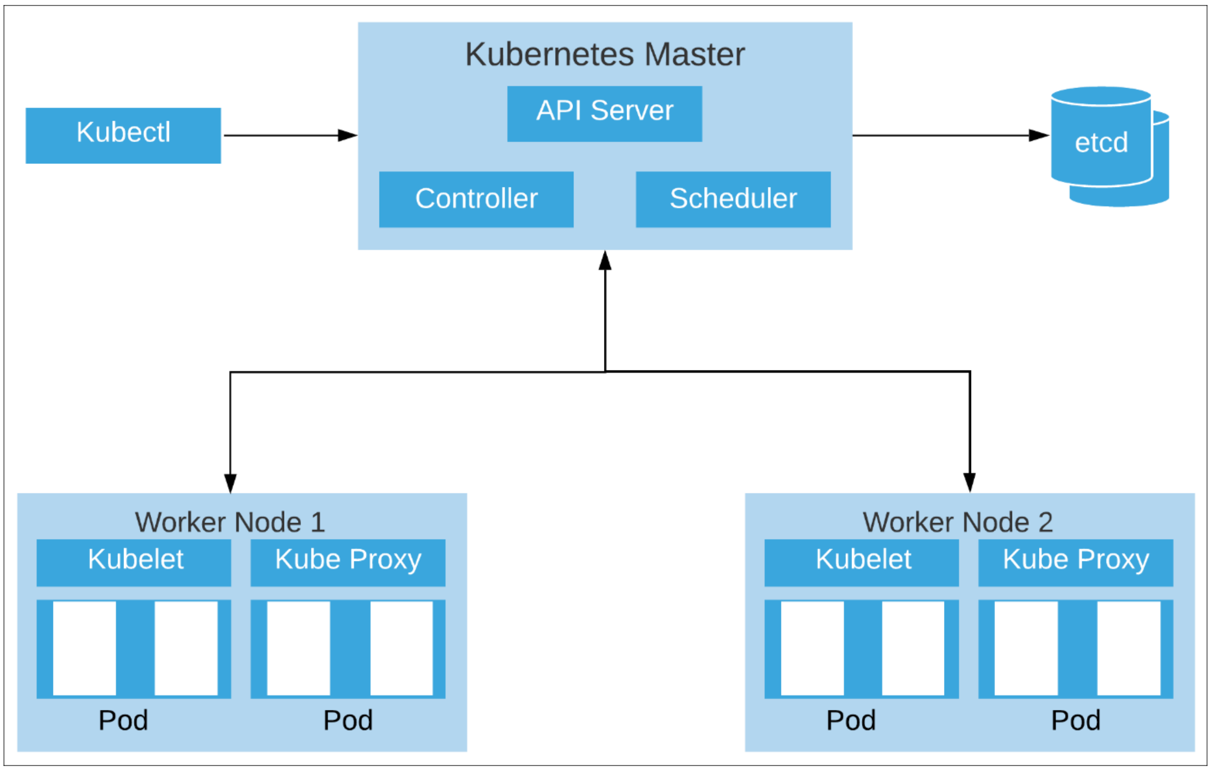
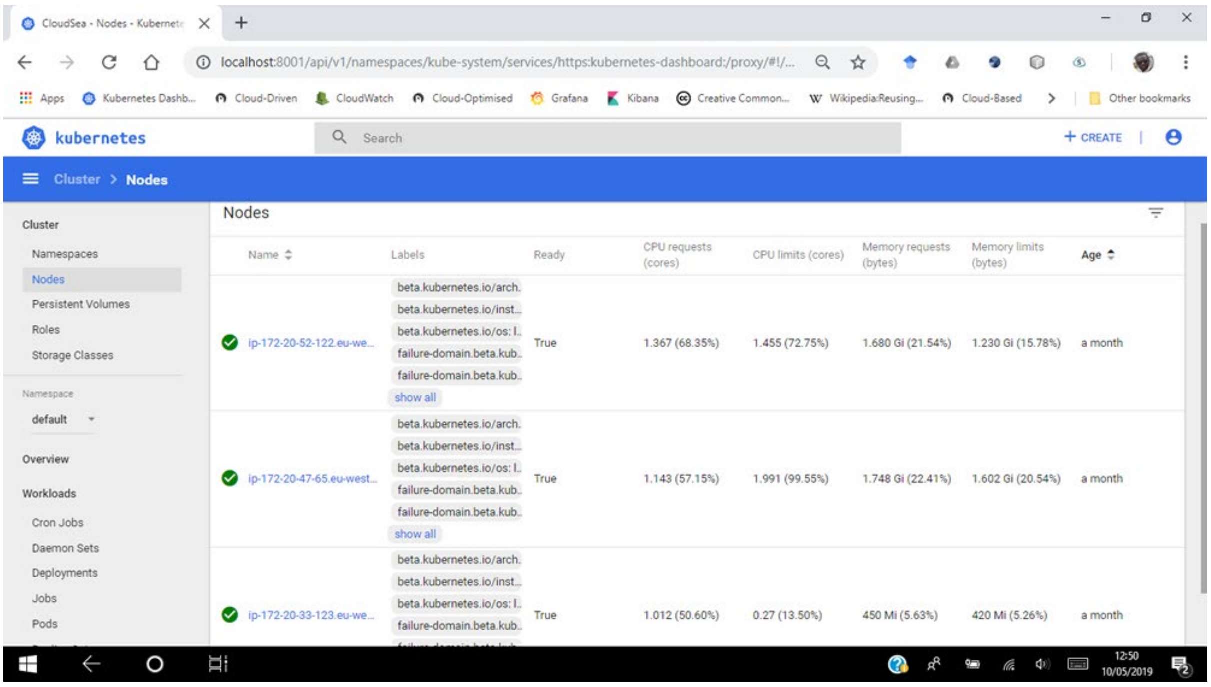


Figure 6. 7 – Architecture of CloudSea Kubernetes Cluster

Figure 6.8 also provides a screenshot of the CloudSea Kubernetes nodes via the Kubernetes Dashboard.

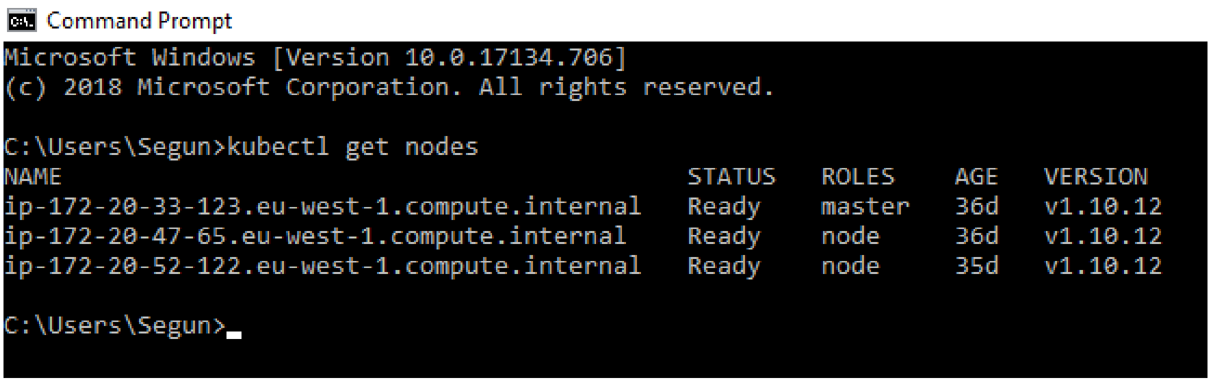


Nodes

| Name | Labels | Ready | CPU requests (cores) | CPU limits (cores) | Memory requests (bytes) | Memory limits (bytes) | Age |
|---|---|---|---|---|---|---|---|
| ✓ ip-172-20-52-122.eu-we... | beta.kubernetes.io/arch.<br>beta.kubernetes.io/inst...<br>beta.kubernetes.io/os: l...<br>failure-domain.beta.kub...<br>failure-domain.beta.kub...<br>show all | True | 1.367 (68.35%) | 1.455 (72.75%) | 1.680 Gi (21.54%) | 1.230 Gi (15.78%) | a month |
| ✓ ip-172-20-47-65.eu-west... | beta.kubernetes.io/arch.<br>beta.kubernetes.io/inst...<br>beta.kubernetes.io/os: l...<br>failure-domain.beta.kub...<br>failure-domain.beta.kub...<br>show all | True | 1.143 (57.15%) | 1.991 (99.55%) | 1.748 Gi (22.41%) | 1.602 Gi (20.54%) | a month |
| ✓ ip-172-20-33-123.eu-we... | beta.kubernetes.io/arch.<br>beta.kubernetes.io/inst...<br>beta.kubernetes.io/os: l...<br>failure-domain.beta.kub... | True | 1.012 (50.60%) | 0.27 (13.50%) | 450 Mi (5.63%) | 420 Mi (5.26%) | a month |

Figure 6. 8 – Screenshot of CloudSea Kubernetes Master and Worker Nodes

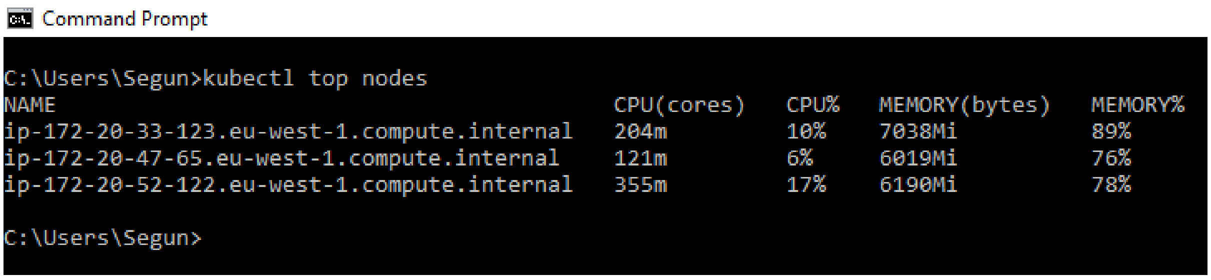
Upon installing and configuring Kubectl locally, the command **$ kubectl proxy** is used to launch the dashboard from the command-line interface. The CloudSea Kubernetes nodes can also be checked from the command-line with the command: **$ kubectl get nodes** as shown in Figure 6.9. While most of the CloudSea Kubernetes cluster resources can be accessed via either the graphical user interface; the dashboard or command-line interface utilising Kubectl, most of the subsequent descriptions would be via Kubectl; the command-line utility for Kubernetes.



```
Command Prompt
Microsoft Windows [Version 10.0.17134.706]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\Segun>kubectl get nodes
NAME                                         STATUS   ROLES    AGE   VERSION
ip-172-20-33-123.eu-west-1.compute.internal  Ready    master   36d   v1.10.12
ip-172-20-47-65.eu-west-1.compute.internal   Ready    node     36d   v1.10.12
ip-172-20-52-122.eu-west-1.compute.internal  Ready    node     35d   v1.10.12

C:\Users\Segun>
```

Figure 6. 9 – Screenshot of CloudSea Kubernetes Master and Worker Nodes from CLI

System metrics such as real time CPU and memory utilisation of the Kubernetes nodes can also be checked. Figure 6.10 shows a screenshot of real time CPU and memory utilisation of the CloudSea Kubernetes nodes using the **$ kubectl get top nodes** command.



```
Command Prompt

C:\Users\Segun>kubectl top nodes
NAME                                         CPU(cores)   CPU%   MEMORY(bytes)   MEMORY%
ip-172-20-33-123.eu-west-1.compute.internal  204m         10%    7038Mi          89%
ip-172-20-47-65.eu-west-1.compute.internal   121m         6%     6019Mi          76%
ip-172-20-52-122.eu-west-1.compute.internal  355m         17%    6190Mi          78%

C:\Users\Segun>
```

Figure 6. 10 – Real time CPU & Memory Usage of CloudSea Kubernetes Nodes

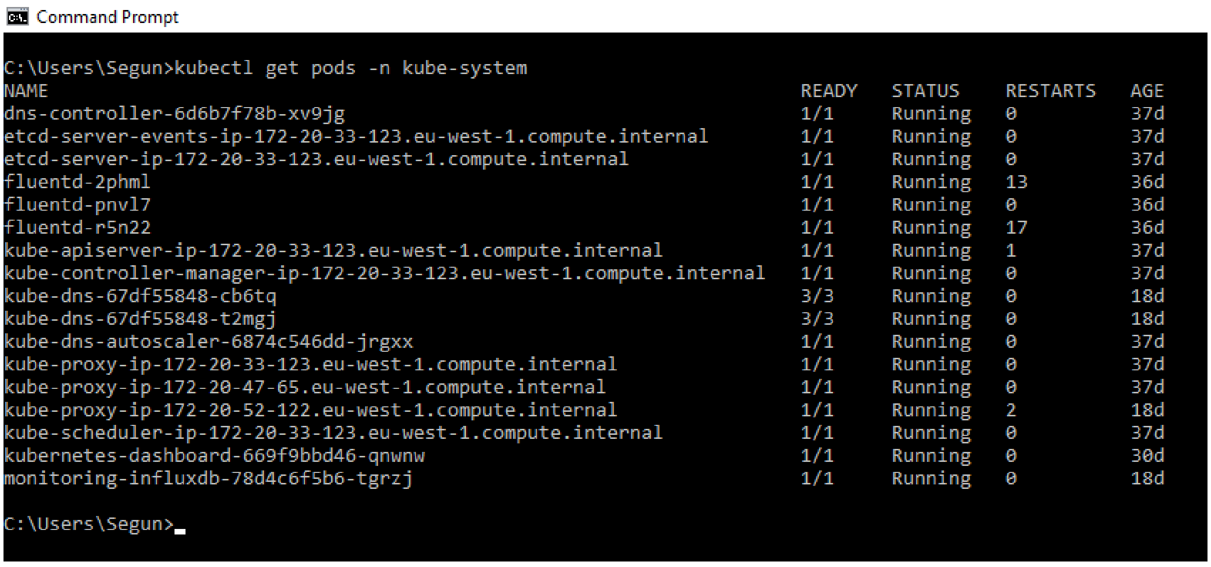
#### 6.5.3.2 The Kubernetes Namespaces

Namespaces in Kubernetes provides a means of categorising resources within the cluster into different groups. While this is an optional feature, it can be helpful in managing and monitoring resources as these can be done based on the namespace such resources belong to. CloudSea Kubernetes was configured with the following namespaces:

- Default: As the name implies, it is a default namespace which is used for objects not assigned to a specific namespace. For the CloudSea Kubernetes cluster, the prototype microservices were not assigned to any specific namespace during their deployment, hence they all belong to the ‘Default’ namespace.

- Kube-ingress: This namespace caters for resources relating to ‘ingress’; the mechanism for exposing specific resources to the public by defining routing rules for each service being exposed for access from outside of the cluster.
- Kube-public: This was created automatically by the Kubernetes system and is generally assigned for objects within the cluster that are meant to be publicly accessible within the cluster.
- Kube-system: This namespace is for system-generated objects such as pods for kube-proxy, kube-dns and others.
- Monitoring: It was set up for the monitoring resources implemented for the cluster. This includes Prometheus, Grafana and the ELK (Elasticsearch, Logstash and Kibana) stack.

As an example, Figure 6.11 shows a screenshot of the CloudSea Kubernetes cluster pods within the ‘Kube-system’ namespace with the number of instances running, their status, restarts and age (i.e. number of days since their creation) using the **$ kubectl get pods -n kube-system** command.

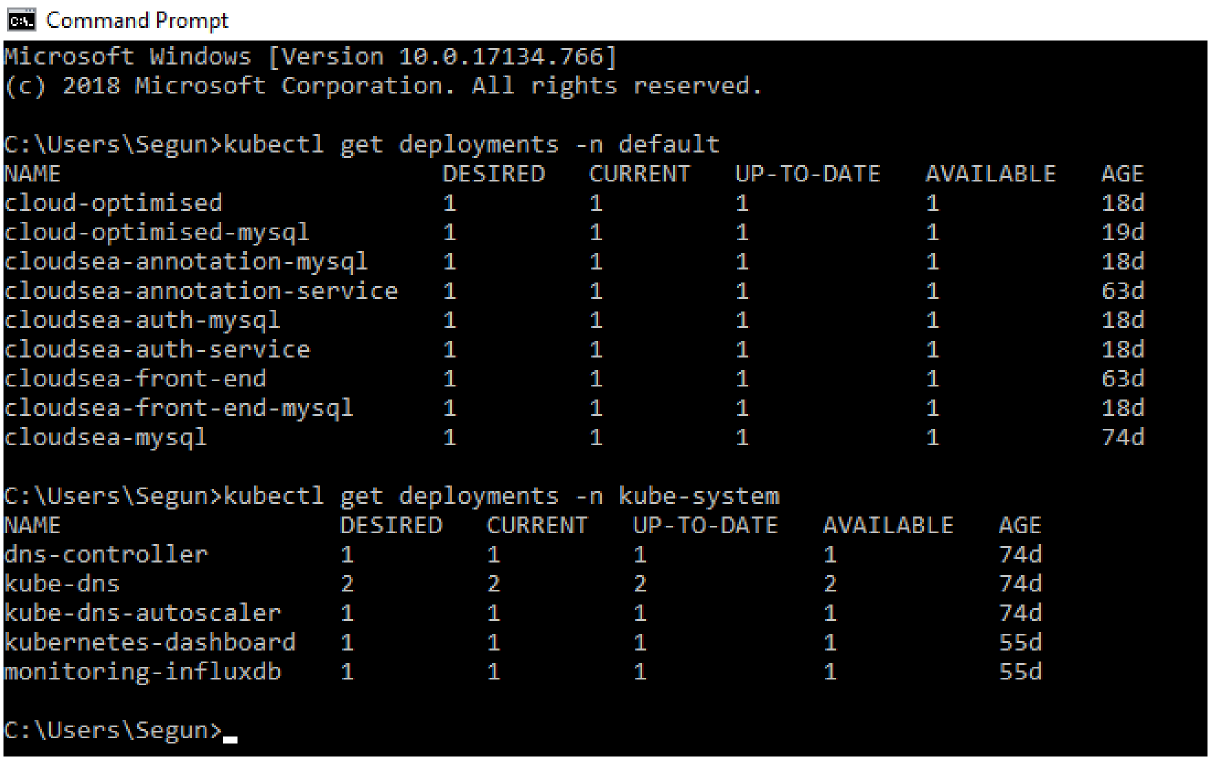


```
Command Prompt

C:\Users\Segun>kubectl get pods -n kube-system
NAME                                                              READY   STATUS    RESTARTS   AGE
dns-controller-6d6b7f78b-xv9jg                                    1/1     Running   0          37d
etcd-server-events-ip-172-20-33-123.eu-west-1.compute.internal    1/1     Running   0          37d
etcd-server-ip-172-20-33-123.eu-west-1.compute.internal           1/1     Running   0          37d
fluentd-2phml                                                     1/1     Running   13         36d
fluentd-pnvl7                                                     1/1     Running   0          36d
fluentd-r5n22                                                     1/1     Running   17         36d
kube-apiserver-ip-172-20-33-123.eu-west-1.compute.internal        1/1     Running   1          37d
kube-controller-manager-ip-172-20-33-123.eu-west-1.compute.internal   1/1     Running   0          37d
kube-dns-67df55848-cb6tq                                          3/3     Running   0          18d
kube-dns-67df55848-t2mgj                                          3/3     Running   0          18d
kube-dns-autoscaler-6874c546dd-jrgxx                              1/1     Running   0          37d
kube-proxy-ip-172-20-33-123.eu-west-1.compute.internal            1/1     Running   0          37d
kube-proxy-ip-172-20-47-65.eu-west-1.compute.internal             1/1     Running   0          37d
kube-proxy-ip-172-20-52-122.eu-west-1.compute.internal            1/1     Running   2          18d
kube-scheduler-ip-172-20-33-123.eu-west-1.compute.internal        1/1     Running   0          37d
kubernetes-dashboard-669f9bbd46-qnwnw                             1/1     Running   0          30d
monitoring-influxdb-78d4c6f5b6-tgrzj                              1/1     Running   0          18d

C:\Users\Segun>
```

Figure 6. 11 - Screenshot of CloudSea Kubernetes Kube-System Namespace Pods

### 6.5.3.3 The Kubernetes Deployments

Deployments in Kubernetes refers to containers that have been instantiated to run within the Kubernetes cluster. These could be application containers; for the application developed to run on Kubernetes or system containers; for the effective running of the cluster itself which could be a native Kubernetes container or for any third-party software installed and configured to run within the cluster. The CloudSea Kubernetes cluster has several application and system containers deployed on it and Figure 6.12 shows a screenshot of all deployments under the default and kube-system namespaces using the **$ kubectl get deployments -n default** and **kubectl get deployments -n kube-system** commands respectively.



```
Command Prompt
Microsoft Windows [Version 10.0.17134.766]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\Segun>kubectl get deployments -n default
NAME                          DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
cloud-optimised               1         1         1            1           18d
cloud-optimised-mysql         1         1         1            1           19d
cloudsea-annotation-mysql     1         1         1            1           18d
cloudsea-annotation-service   1         1         1            1           63d
cloudsea-auth-mysql           1         1         1            1           18d
cloudsea-auth-service         1         1         1            1           18d
cloudsea-front-end            1         1         1            1           63d
cloudsea-front-end-mysql      1         1         1            1           18d
cloudsea-mysql                1         1         1            1           74d

C:\Users\Segun>kubectl get deployments -n kube-system
NAME                   DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
dns-controller         1         1         1            1           74d
kube-dns               2         2         2            2           74d
kube-dns-autoscaler    1         1         1            1           74d
kubernetes-dashboard   1         1         1            1           55d
monitoring-influxdb    1         1         1            1           55d

C:\Users\Segun>_
```

Figure 6. 12 – Screenshot of CloudSea Kubernetes Deployments by Namespaces

A brief description of all the various deployments by their namespaces is as follows:

- Default Namespace
  - cloud-optimised: for the deployed monolithic and containerised prototype version to enable experimental evaluation comparison (detailed in Chapter 7).
  - cloud-optimised-mysql: the mysql database deployment for “cloud-optimised”
  - cloudsea-annotation-service: for the annotation-on-the-fly microservice, allowing users to semantically annotate web documents by embedding the API within the corresponding web documents.
  - cloudsea-annotation-mysql: the mysql database deployment for “cloud-annotation-service”
  - cloudsea-auth-service: for the user-management microservice, allowing registration, user login, password reset and log out functions.
  - cloudsea-auth-service-mysql: the mysql database deployment for “cloudsea-auth-service”

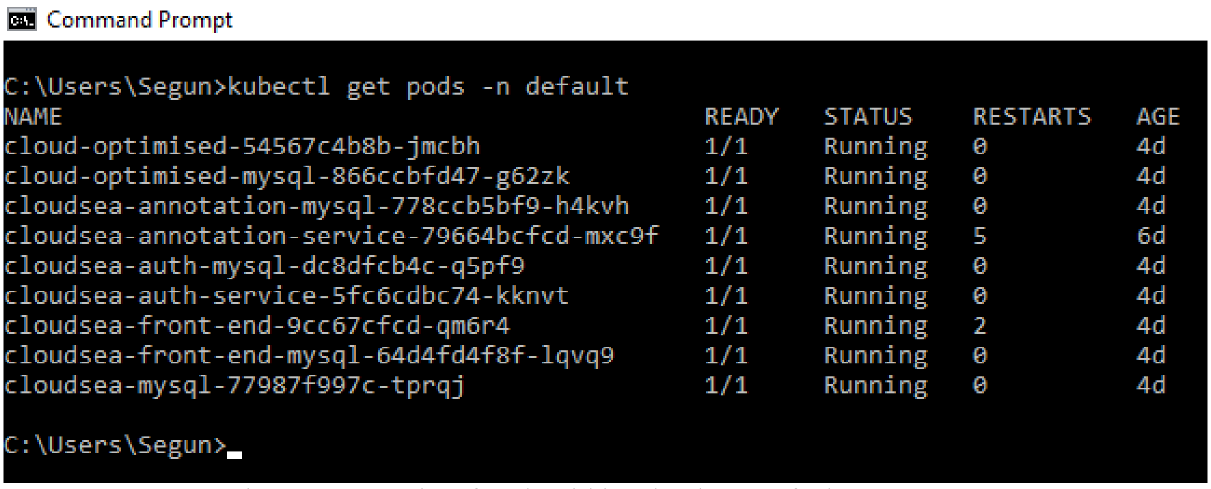
    - cloudsea-front-end: for the frontend microservice, providing a graphical user interface for the web application.
    - cloudsea-front-end-mysql: mysql database for "cloudsea-front-mysql". However, as the microservice does not require a database, this was not utilised
    - cloudsea-mysql: this was also not utilised but was kept as a backup for the mysql databases.
- Monitoring Namespace
    - cerebro: It is a web admin tool for Elasticsearch (Elasticsearch is a component of the ELK stack). It comes with the ELK stack.
    - es-coordinating: An Elasticsearch node for the distribution of search queries to different nodes and the aggregation of all received search results.
    - es-ingest: It is an Elasticsearch component for pre-processing documents before their indexing.
    - grafana: A visualisation tool for displaying application performance metrics in a web graphical user interface.
    - kibana: A visualisation tool for displaying log metrics in a web graphical user interface.
    - kube-state-metrics: For listening to the Kubernetes API server and generating metrics for deployments, nodes and pods. It is focused on the healthy running of these components.
    - prometheus-adapter: For gathering custom metrics which are utilised by the HPA (Horizontal Pod Auto-Scaling) for auto-scaling pods when necessary. Some of such custom metrics includes CPU and memory utilisation.
    - prometheus-operator: For creating, configuring and managing the Prometheus installation. It acts as a central system to ensure the effective running of the different instances of Prometheus objects running on the cluster.
- Kube System Namespace
    - dns-controller: It creates DNS records for objects within the Kubernetes cluster so they can be accessed based on the record.
    - kube-dns: It is the DNS service of the Kubernetes cluster, converting hostnames into IP addresses for specifying and identifying the location of different objects within the cluster.

- kube-dns-autoscaler: It is responsible for the automatic horizontal scaling of the Kubernetes DNS service based on the number of requests at every moment.
- kubernetes-dashboard: It is responsible for the web user interface of Kubernetes, through which cluster objects can be monitored and updated.
- monitoring-influxdb: It is responsible for storing time-series data in relation to monitoring metrics and events within the kube-system namespace, providing a high-speed data ingestion and compression.

- Kube Ingress Namespace
    - ingress-default-http-backend: It defines a default backend for any routes (such as HTTP or HTTPS requests) that are not specified for a service within the cluster.
    - ingress-nginx: It is responsible for exposing routes such as ‘HTTP’ and ‘HTTPS’ from outside the cluster to specific services within the cluster. For the CloudSea cluster, this has been utilised to expose deployments such as the ‘Frontend’ microservice of the prototype to a domain name.

#### 6.5.3.4 The Kubernetes Pods

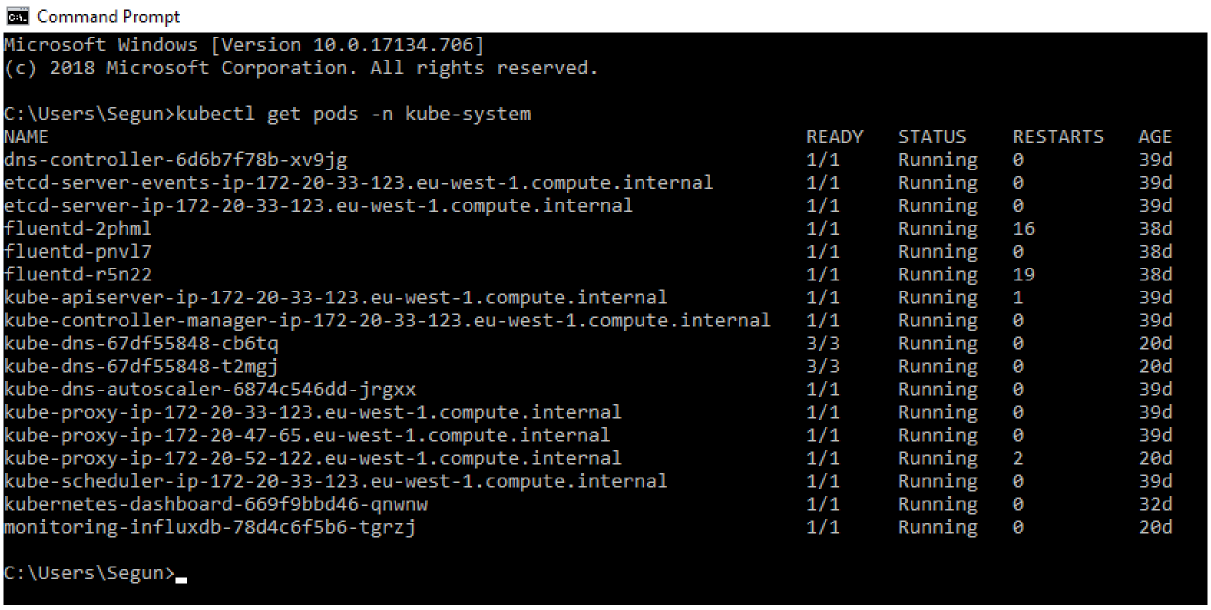
Each CloudSea Kubernetes namespace has several pods running under it. With Kubernetes, pods contain one or more containers and are the smallest units of deployment. Figure 6.13 shows a screenshot of all the running pods within the ‘Default’ namespace of the cluster using the **$ kubectl get pods -n default** command.



```
C:\Users\Segun>kubectl get pods -n default
NAME                                          READY   STATUS    RESTARTS   AGE
cloud-optimised-54567c4b8b-jmcbh              1/1     Running   0          4d
cloud-optimised-mysql-866ccbfd47-g62zk        1/1     Running   0          4d
cloudsea-annotation-mysql-778ccb5bf9-h4kvh    1/1     Running   0          4d
cloudsea-annotation-service-79664bcfcd-mxc9f  1/1     Running   5          6d
cloudsea-auth-mysql-dc8dfcb4c-q5pf9           1/1     Running   0          4d
cloudsea-auth-service-5fc6cdbc74-kknvt        1/1     Running   0          4d
cloudsea-front-end-9cc67cfcd-qm6r4            1/1     Running   2          4d
cloudsea-front-end-mysql-64d4fd4f8f-lqvq9     1/1     Running   0          4d
cloudsea-mysql-77987f997c-tprqj               1/1     Running   0          4d

C:\Users\Segun>
```

Figure 6. 13 – List of Pods within CloudSea Default Namespace

To check running pods within any other namespace, the namespace would have to be specified in the command, as shown in Figure 6.14 which lists all running pods within the 'kube-system' namespace using the **$ kubectl get pods -n kube-system** command.



```
Command Prompt
Microsoft Windows [Version 10.0.17134.706]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\Segun>kubectl get pods -n kube-system
NAME                                                              READY   STATUS    RESTARTS   AGE
dns-controller-6d6b7f78b-xv9jg                                    1/1     Running   0          39d
etcd-server-events-ip-172-20-33-123.eu-west-1.compute.internal    1/1     Running   0          39d
etcd-server-ip-172-20-33-123.eu-west-1.compute.internal           1/1     Running   0          39d
fluentd-2phml                                                     1/1     Running   16         38d
fluentd-pnvl7                                                     1/1     Running   0          38d
fluentd-r5n22                                                     1/1     Running   19         38d
kube-apiserver-ip-172-20-33-123.eu-west-1.compute.internal        1/1     Running   1          39d
kube-controller-manager-ip-172-20-33-123.eu-west-1.compute.internal   1/1     Running   0          39d
kube-dns-67df55848-cb6tq                                          3/3     Running   0          20d
kube-dns-67df55848-t2mgj                                          3/3     Running   0          20d
kube-dns-autoscaler-6874c546dd-jrgxx                              1/1     Running   0          39d
kube-proxy-ip-172-20-33-123.eu-west-1.compute.internal            1/1     Running   0          39d
kube-proxy-ip-172-20-47-65.eu-west-1.compute.internal             1/1     Running   0          39d
kube-proxy-ip-172-20-52-122.eu-west-1.compute.internal            1/1     Running   2          20d
kube-scheduler-ip-172-20-33-123.eu-west-1.compute.internal        1/1     Running   0          39d
kubernetes-dashboard-669f9bbd46-qnwnw                             1/1     Running   0          32d
monitoring-influxdb-78d4c6f5b6-tgrzj                              1/1     Running   0          20d

C:\Users\Segun>_
```

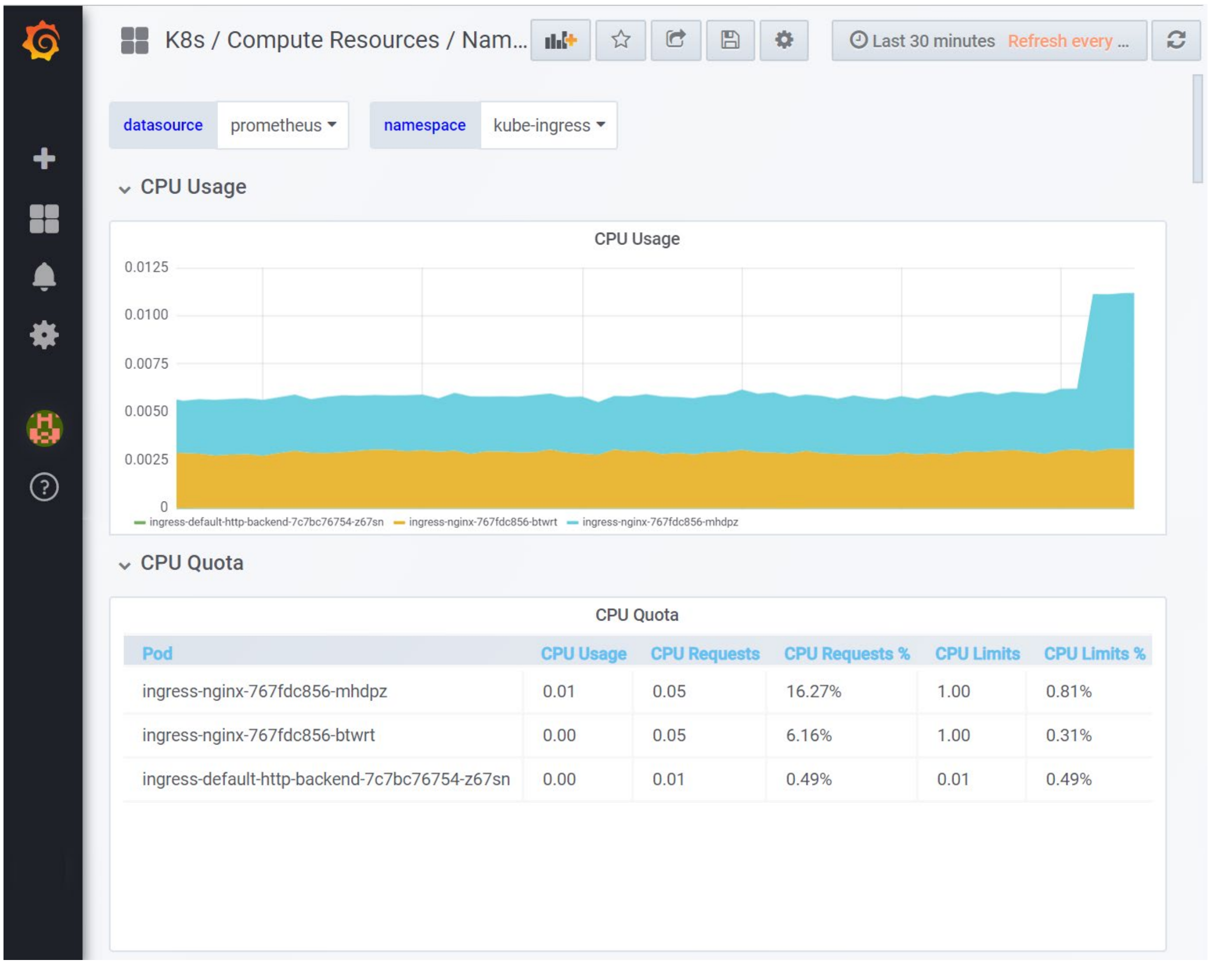
Figure 6. 14 – List of Pods under 'kube-system' namespace of CloudSea Kubernetes

### 6.5.4 Installing, Configuring and Running Prometheus + Grafana for Monitoring

Some additional software were installed and configured for the CloudSea Kubernetes cluster. Kubernetes provides a means of integrating with several other third-party applications to enhance application performance in various ways. Some areas through which other solutions can be integrated with Kubernetes include co-ordination, service discovery, remote procedure call, service proxy and so on (IBM Cloud Education, 2019). For CloudSea Kubernetes; Prometheus, Grafana and ELK (Elasticsearch, Logstash and Kibana) stack were installed and configured as well.

The CloudSea Kubernetes cluster utilises Prometheus + Grafana for monitoring various resources within the cluster. Prometheus is an open source software which is used for this purpose with respect to the deployment of applications in the cloud. Prometheus can be installed on a virtual host as a stand-alone or within a Kubernetes cluster (Prometheus.io, 2019). For CloudSea, it runs within the Kubernetes cluster and is responsible for monitoring several metrics such as CPU utilisation, memory utilisation, network activity and lots more for deployments, nodes and pods across different namespaces.

Grafana on the other hand is a visualisation tool which can be utilised with several data stores to present metrics of application data in visual forms; utilising several graphical representations (Grafana Labs, 2019). The combination of Prometheus and Grafana implies that while the former pulls application metrics from the Kubernetes cluster, the latter renders these data in compelling visualisation formats. Just like Prometheus, Grafana is an open source tool. Figure 6.15 is a screenshot of Grafana running as a node port over a secure socket layer (SSL) and displaying real time CPU utilisation of the pods under the 'kube-ingress' namespace.



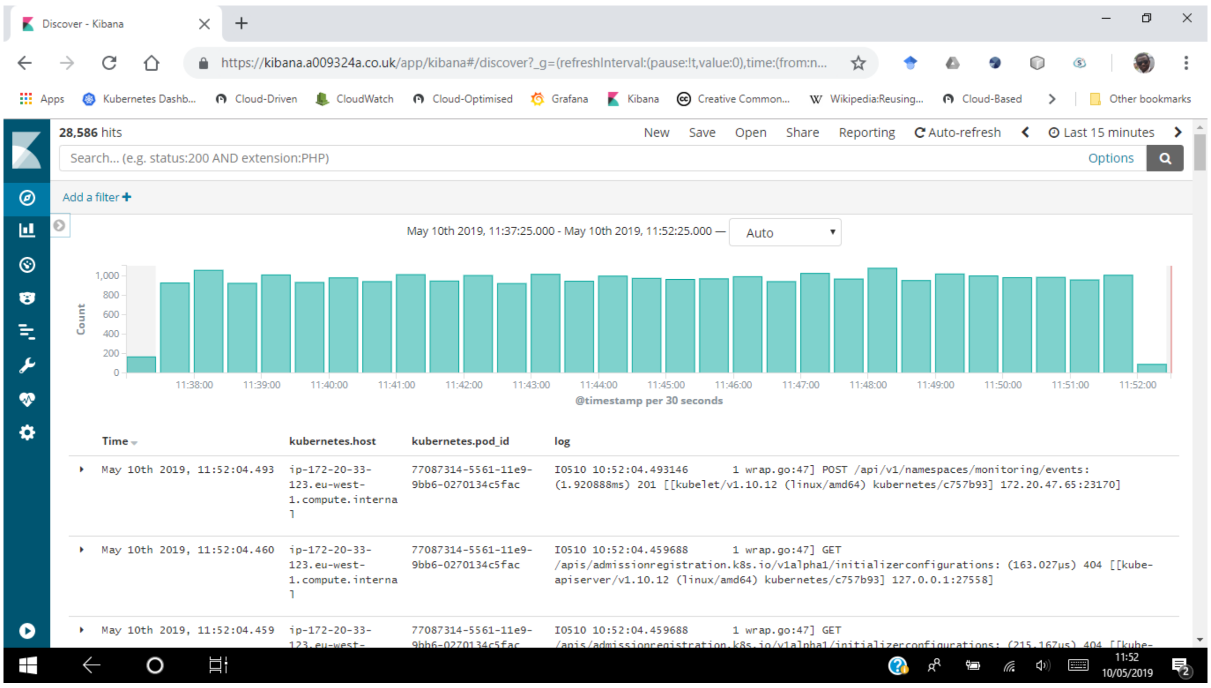
| Pod | CPU Usage | CPU Requests | CPU Requests % | CPU Limits | CPU Limits % |
|---|---|---|---|---|---|
| ingress-nginx-767fdc856-mhdpz | 0.01 | 0.05 | 16.27% | 1.00 | 0.81% |
| ingress-nginx-767fdc856-btwrt | 0.00 | 0.05 | 6.16% | 1.00 | 0.31% |
| ingress-default-http-backend-7c7bc76754-z67sn | 0.00 | 0.01 | 0.49% | 0.01 | 0.49% |

Figure 6. 15 – Monitoring Metrics for 'kube-ingress' Namespace via Grafana

### 6.5.5 Installing, Configuring and Running the ELK Stack for Logging

ELK is an acronym that stands for the different open source technologies that make up a stack. These are Elasticsearch, Logstash and Kibana (Elastic.co, 2019). Elasticsearch is a search and analytics engine which is built on Apache Lucene. It is used for several purposes such as search and log analytics (Elastic.co; Elasticsearch, 2019). For this implementation, Elasticsearch has been utilised for searching and aggregating logs across the entire Kubernetes cluster. The different sources of logs include the namespaces (kube-system, kube-ingress, monitoring, default, etc.), pods (both system and application pods) as well as the several containers running with the cluster.

Logstash is a tool for data ingestion, with the capability to ingest data from multiple data sources, transform the data based on pre-defined or default configuration values and transfer the processed data to specified destinations (Elastic.co; Logstash, 2019). It receives the different logs generated by Elasticsearch for this implementation, transforms and transfers the data to Kibana. Kibana sits on top of the stack, providing visualisation for the aggregated logs received (Elastic.co; Kibana, 2019). The visualisation ensures that these logs can be explored, reviewed and analysed accordingly. Figure 6.16 shows a screenshot of Kibana with aggregated log analytics from the CloudSea Kubernetes cluster over a 15-minute period. Furthermore, please refer to Appendix E for sample log entries fetched from Kibana for application pods within the cluster.



```
Time                          kubernetes.host      kubernetes.pod_id      log

May 10th 2019, 11:52:04.493   ip-172-20-33-        77087314-5561-11e9-    I0510 10:52:04.493146       1 wrap.go:47] POST /api/v1/namespaces/monitoring/events:
                              123.eu-west-         9bb6-0270134c5fac      (1.920888ms) 201 [[kubelet/v1.10.12 (linux/amd64) kubernetes/c757b93] 172.20.47.65:23170]
                              1.compute.interna
                              l

May 10th 2019, 11:52:04.460   ip-172-20-33-        77087314-5561-11e9-    I0510 10:52:04.459688       1 wrap.go:47] GET
                              123.eu-west-         9bb6-0270134c5fac      /apis/admissionregistration.k8s.io/v1alpha1/initializerconfigurations: (163.027µs) 404 [[kube-
                              1.compute.interna                           apiserver/v1.10.12 (linux/amd64) kubernetes/c757b93] 127.0.0.1:27558]
                              l

May 10th 2019, 11:52:04.459   ip-172-20-33-        77087314-5561-11e9-    I0510 10:52:04.459688       1 wrap.go:47] GET
                              123.eu-west-         9bb6-0270134c5fac      /apis/admissionregistration.k8s.io/v1alpha1/initializerconfigurations: (215.167µs) 404 [[kube-
```

Figure 6. 16 – Kibana Log Aggregation Sample Data for CloudSea Kubernetes cluster

## 6.7 Chapter Summary

This chapter has provided a description of steps taken towards the implementation of a prototype for the CloudSea Architecture proposed in Chapter 5 up to its deployment and running from a public cloud infrastructure. The development environment, supporting technologies and deployment environment were also described. The next chapter will focus on functional, comparative and experimental evaluation for CloudSea Architecture and the prototype implementation.

# Chapter 7: Research Evaluation

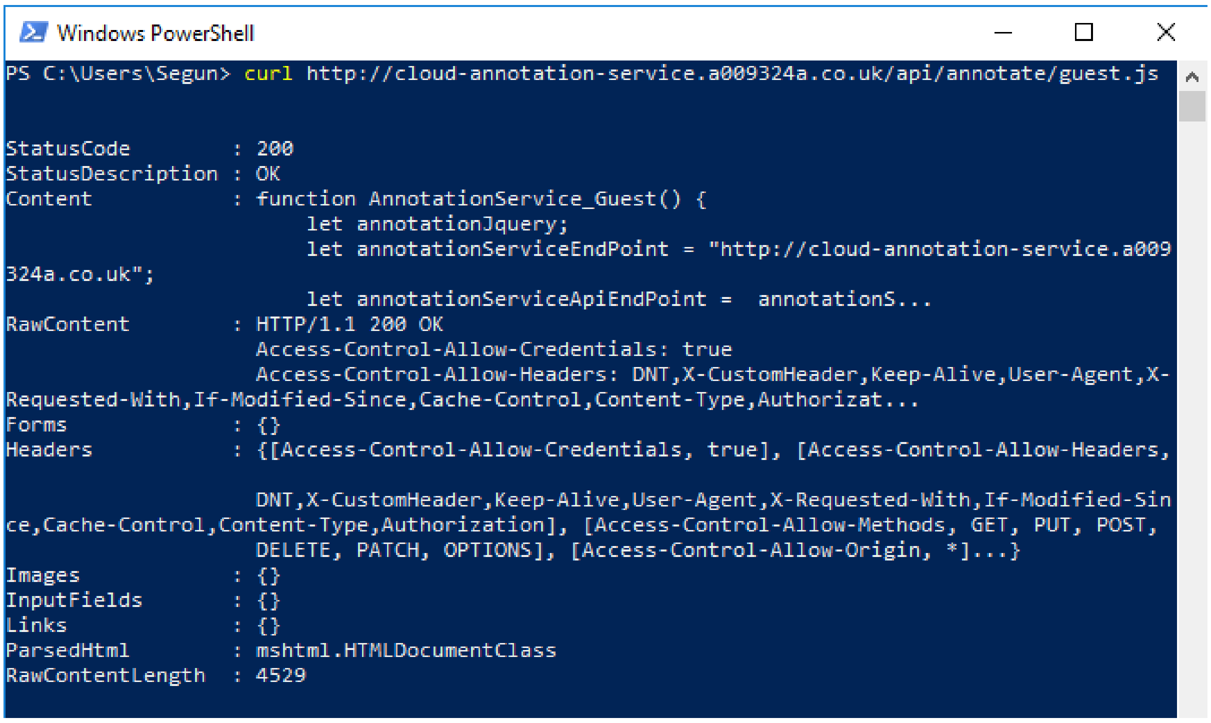
This chapter provides evaluation for the research in three different contexts. Firstly, a functionality evaluation is conducted for CloudSea prototype implementation as a demonstration of its core functionalities. Secondly, a comparative evaluation of CloudSea Architecture is conducted to demonstrate its novelty through a rigorous comparison with existing solutions in the same domain. Thirdly, an experimental evaluation of the proposed Cloud Computing Maturity Model for Holistic Semantic Annotation is conducted by carrying out empirical investigation on different instances of the prototype, based on the patterns defined in the model.

## 7.1 Functionality Evaluation

This section provides an evaluation for the core functionality of CloudSea prototype implementation. Firstly, an API status check is conducted to confirm its availability. Subsequently, a demonstration of its usability for automated semantic annotation and its capability to automatically scale to multiple instances of same microservice in response to client request counts is conducted.

### 7.1.1 API Status Check

The availability of an API can be checked by confirming its status through a command or series of commands. One popularly used command to check an API availability is known as cURL. cURL is an acronym for Client for URLs, and it is a software utility command tool that is used for transferring data to or from a network server using one of several supported protocols such as HTTP, HTTPS, FTP, FTPS, Telnet, LDAP and lots more. It comprises of two sub-utilities; cURL and libcurl. cURL provides the command-line for receiving or sending resources such as files using the URL web protocol. libcurl on the other hand, is a portable utility that provides support for client-side transfer library protocols (Stenberg, 2019). An API call using cURL provides a code to confirm the status of the API in terms of its availability and validating for providing functionality. Status code ‘200’ confirms the availability and validity of an API endpoint. Figure 7.1 provides a screen shot of the cURL command with CloudSea API endpoint which confirms its validity.



```
PS C:\Users\Segun> curl http://cloud-annotation-service.a009324a.co.uk/api/annotate/guest.js


StatusCode        : 200
StatusDescription : OK
Content           : function AnnotationService_Guest() {
                        let annotationJquery;
                        let annotationServiceEndPoint = "http://cloud-annotation-service.a009
324a.co.uk";
                        let annotationServiceApiEndPoint =  annotationS...
RawContent        : HTTP/1.1 200 OK
                    Access-Control-Allow-Credentials: true
                    Access-Control-Allow-Headers: DNT,X-CustomHeader,Keep-Alive,User-Agent,X-
Requested-With,If-Modified-Since,Cache-Control,Content-Type,Authorizat...
Forms             : {}
Headers           : {[Access-Control-Allow-Credentials, true], [Access-Control-Allow-Headers,

                    DNT,X-CustomHeader,Keep-Alive,User-Agent,X-Requested-With,If-Modified-Sin
ce,Cache-Control,Content-Type,Authorization], [Access-Control-Allow-Methods, GET, PUT, POST,
                    DELETE, PATCH, OPTIONS], [Access-Control-Allow-Origin, *]...}
Images            : {}
InputFields       : {}
Links             : {}
ParsedHtml        : mshtml.HTMLDocumentClass
RawContentLength  : 4529
```

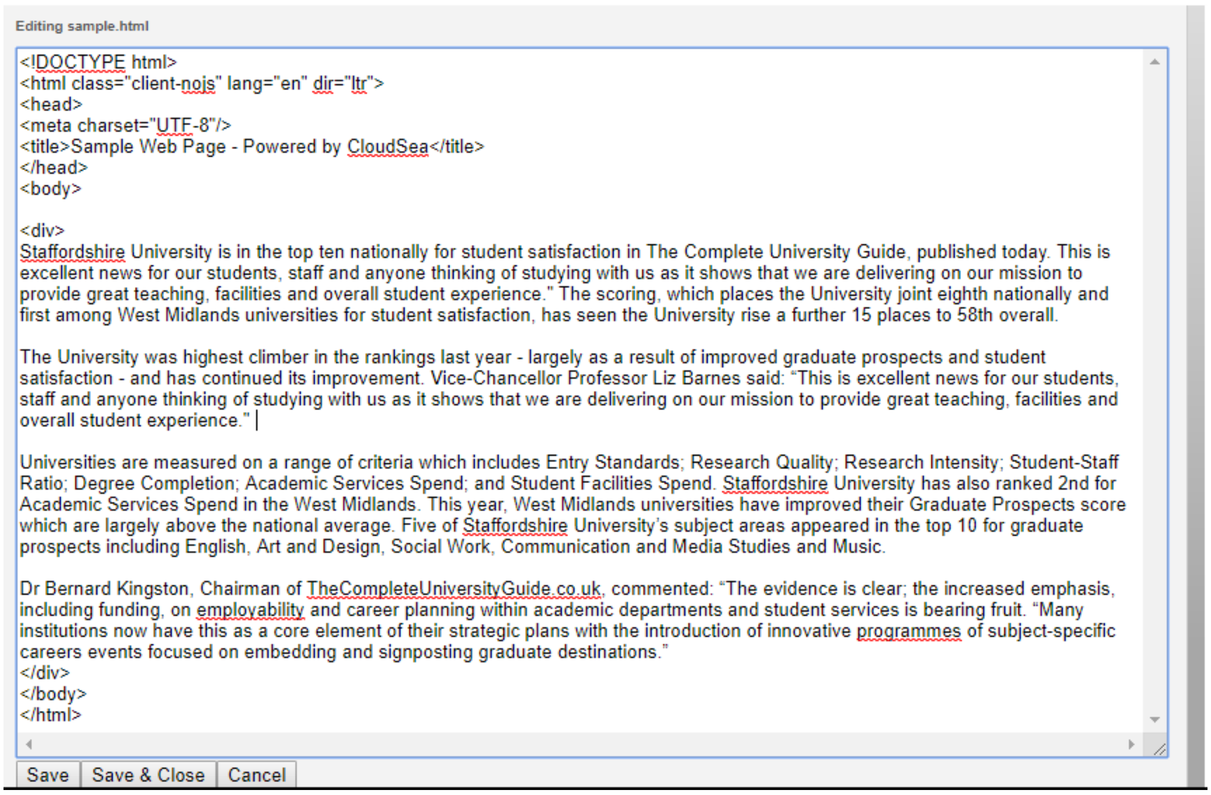
Figure 7. 1 – API Status Check for CloudSea API endpoint

From Figure 7.1, it can be observed that the status code for the CloudSea API endpoint is ‘200’ which confirms that a call to the API is successful.

### 7.1.2 Annotation API Functionality

In this section, further evaluation of the CloudSea Annotation API is provided by describing the procedure for utilising it to provide online, real time and automatic semantic annotation to web documents. The procedure is a very straightforward one, making it as easy as possible for web documents semantic annotation. The process can be implemented by several categories of people such as website administrators, website content editors, website designers, web developers and lots more. It is as follows:

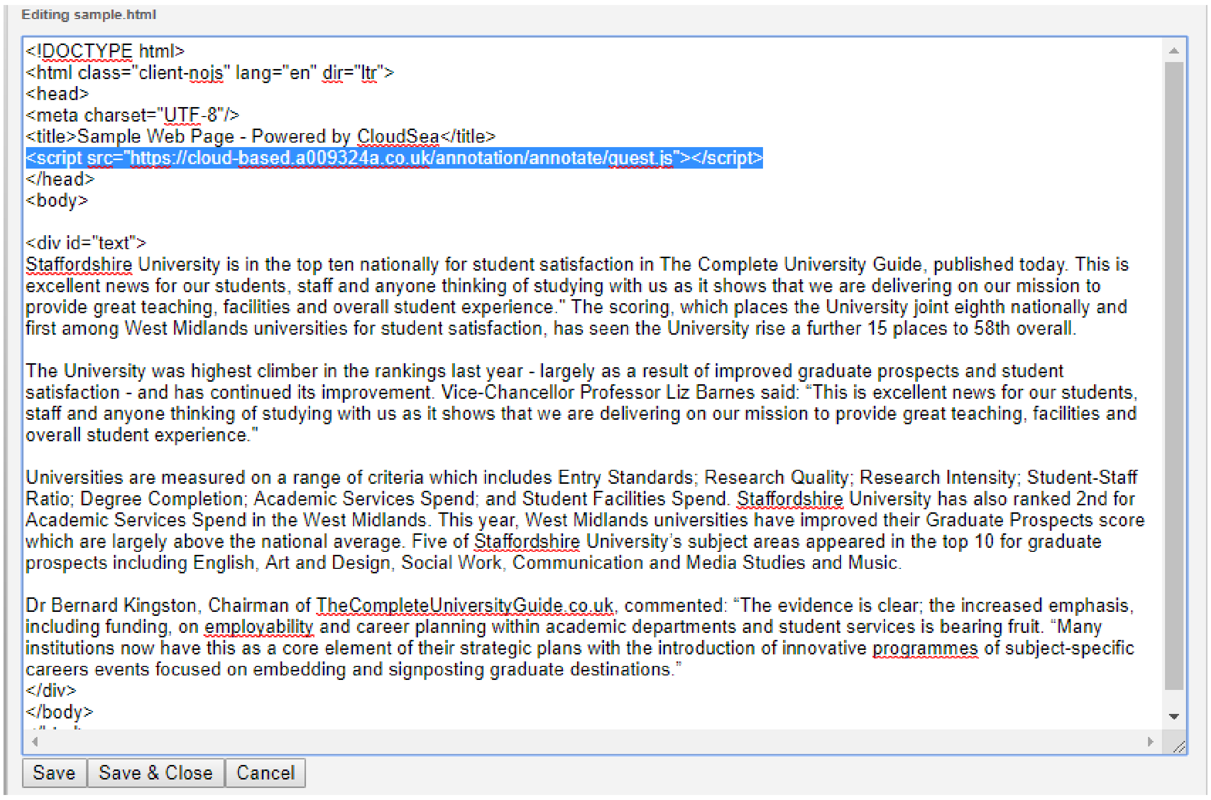
1. Create a web page or open an existing one for editing: This use case will be done with an existing web page as a web page creation method varies across several types of web applications, frameworks or content management systems. A simplistic method of creating and utilising a physical web file on a server is demonstrated here. Figure 7.2 is an example of a web page in which the API is to be called.



```
<!DOCTYPE html>
<html class="client-nojs" lang="en" dir="ltr">
<head>
<meta charset="UTF-8"/>
<title>Sample Web Page - Powered by CloudSea</title>
</head>
<body>

<div>
Staffordshire University is in the top ten nationally for student satisfaction in The Complete University Guide, published today. This is
excellent news for our students, staff and anyone thinking of studying with us as it shows that we are delivering on our mission to
provide great teaching, facilities and overall student experience." The scoring, which places the University joint eighth nationally and
first among West Midlands universities for student satisfaction, has seen the University rise a further 15 places to 58th overall.

The University was highest climber in the rankings last year - largely as a result of improved graduate prospects and student
satisfaction - and has continued its improvement. Vice-Chancellor Professor Liz Barnes said: "This is excellent news for our students,
staff and anyone thinking of studying with us as it shows that we are delivering on our mission to provide great teaching, facilities and
overall student experience."

Universities are measured on a range of criteria which includes Entry Standards; Research Quality; Research Intensity; Student-Staff
Ratio; Degree Completion; Academic Services Spend; and Student Facilities Spend. Staffordshire University has also ranked 2nd for
Academic Services Spend in the West Midlands. This year, West Midlands universities have improved their Graduate Prospects score
which are largely above the national average. Five of Staffordshire University's subject areas appeared in the top 10 for graduate
prospects including English, Art and Design, Social Work, Communication and Media Studies and Music.

Dr Bernard Kingston, Chairman of TheCompleteUniversityGuide.co.uk, commented: "The evidence is clear; the increased emphasis,
including funding, on employability and career planning within academic departments and student services is bearing fruit. "Many
institutions now have this as a core element of their strategic plans with the introduction of innovative programmes of subject-specific
careers events focused on embedding and signposting graduate destinations."
</div>
</body>
</html>
```

Figure 7. 2 – A sample web document source code before API call insertion

2. Insert the API call within the <head> tags of the web page and define a <div> tag with id set as: "text" around the text you wish to annotate. The API endpoint needs to be inserted into web page source code before the closing <head> tag of the file. This is to call the Annotation On-the-fly Microservice to provide semantic annotation for the web document. The "id" attribute is defined with value as "text" for the content within the web document that is to be semantically annotated. Figure 7.3 displays the same web page now with the API call inserted into it and a user specification for the web document content to be semantically annotated.

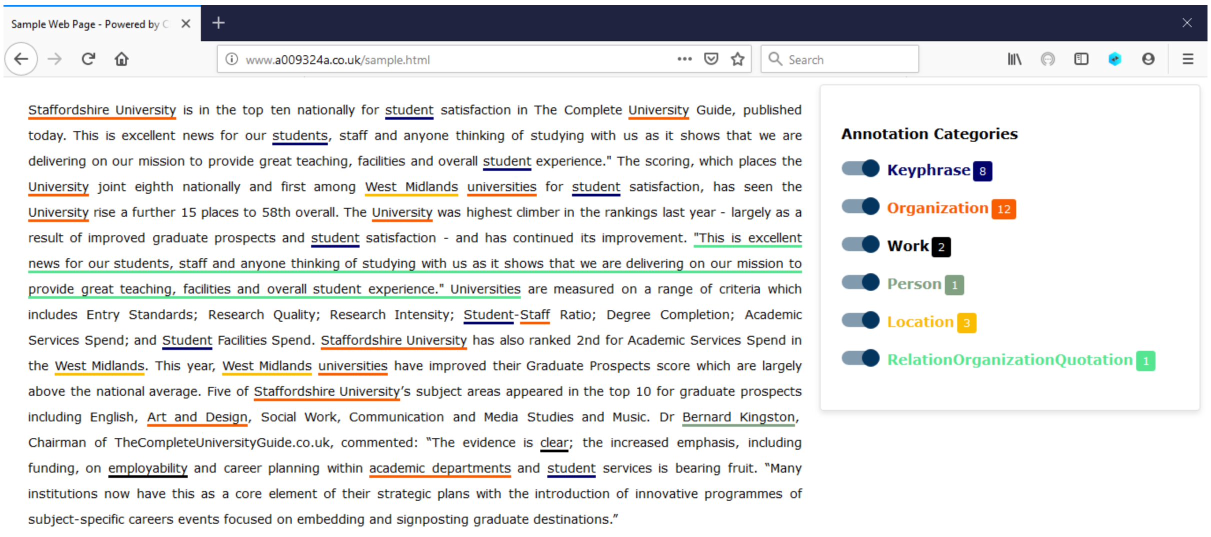

Editing sample.html

```
<!DOCTYPE html>
<html class="client-nojs" lang="en" dir="ltr">
<head>
<meta charset="UTF-8"/>
<title>Sample Web Page - Powered by CloudSea</title>
<script src="https://cloud-based.a009324a.co.uk/annotation/annotate/quest.js"></script>
</head>
<body>

<div id="text">
Staffordshire University is in the top ten nationally for student satisfaction in The Complete University Guide, published today. This is
excellent news for our students, staff and anyone thinking of studying with us as it shows that we are delivering on our mission to
provide great teaching, facilities and overall student experience." The scoring, which places the University joint eighth nationally and
first among West Midlands universities for student satisfaction, has seen the University rise a further 15 places to 58th overall.

The University was highest climber in the rankings last year - largely as a result of improved graduate prospects and student
satisfaction - and has continued its improvement. Vice-Chancellor Professor Liz Barnes said: "This is excellent news for our students,
staff and anyone thinking of studying with us as it shows that we are delivering on our mission to provide great teaching, facilities and
overall student experience."

Universities are measured on a range of criteria which includes Entry Standards; Research Quality; Research Intensity; Student-Staff
Ratio; Degree Completion; Academic Services Spend; and Student Facilities Spend. Staffordshire University has also ranked 2nd for
Academic Services Spend in the West Midlands. This year, West Midlands universities have improved their Graduate Prospects score
which are largely above the national average. Five of Staffordshire University's subject areas appeared in the top 10 for graduate
prospects including English, Art and Design, Social Work, Communication and Media Studies and Music.

Dr Bernard Kingston, Chairman of TheCompleteUniversityGuide.co.uk, commented: "The evidence is clear; the increased emphasis,
including funding, on employability and career planning within academic departments and student services is bearing fruit. "Many
institutions now have this as a core element of their strategic plans with the introduction of innovative programmes of subject-specific
careers events focused on embedding and signposting graduate destinations."
</div>
</body>
```

Figure 7. 3 – Web page with the API call and specification for content to be annotated

3. Save the changes to the web page

4. Access the web page via any web browser of choice. Figure 7.4 presents a screen shot of the semantically annotated web page on-the-fly.


Sample Web Page - Powered by C
www.a009324a.co.uk/sample.html
Search

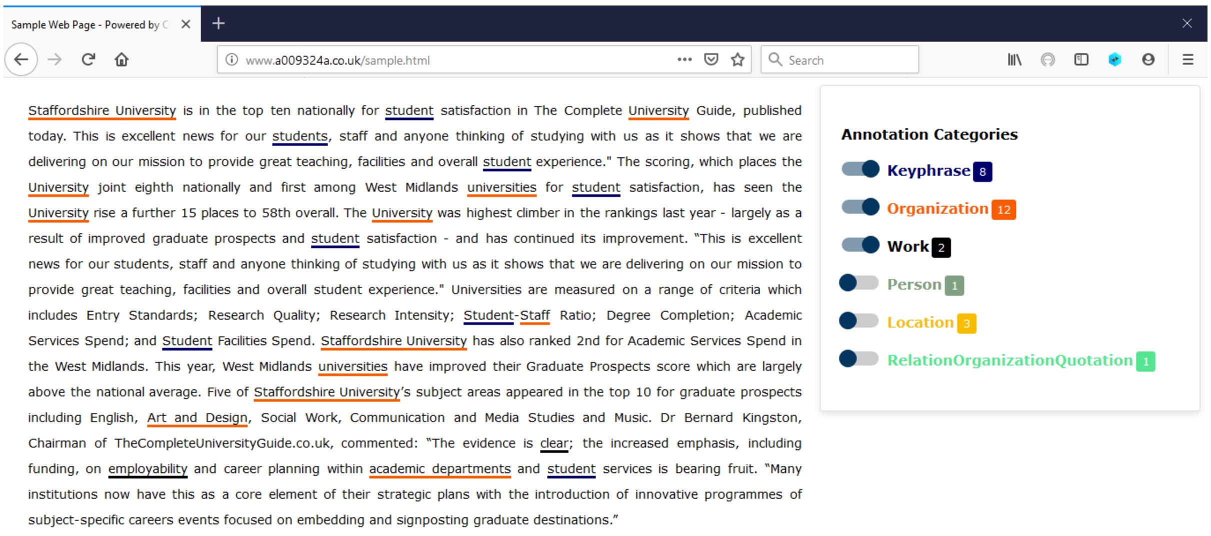
Staffordshire University is in the top ten nationally for student satisfaction in The Complete University Guide, published today. This is excellent news for our students, staff and anyone thinking of studying with us as it shows that we are delivering on our mission to provide great teaching, facilities and overall student experience." The scoring, which places the University joint eighth nationally and first among West Midlands universities for student satisfaction, has seen the University rise a further 15 places to 58th overall. The University was highest climber in the rankings last year - largely as a result of improved graduate prospects and student satisfaction - and has continued its improvement. "This is excellent news for our students, staff and anyone thinking of studying with us as it shows that we are delivering on our mission to provide great teaching, facilities and overall student experience." Universities are measured on a range of criteria which includes Entry Standards; Research Quality; Research Intensity; Student-Staff Ratio; Degree Completion; Academic Services Spend; and Student Facilities Spend. Staffordshire University has also ranked 2nd for Academic Services Spend in the West Midlands. This year, West Midlands universities have improved their Graduate Prospects score which are largely above the national average. Five of Staffordshire University's subject areas appeared in the top 10 for graduate prospects including English, Art and Design, Social Work, Communication and Media Studies and Music. Dr Bernard Kingston, Chairman of TheCompleteUniversityGuide.co.uk, commented: "The evidence is clear; the increased emphasis, including funding, on employability and career planning within academic departments and student services is bearing fruit. "Many institutions now have this as a core element of their strategic plans with the introduction of innovative programmes of subject-specific careers events focused on embedding and signposting graduate destinations."

Annotation Categories

| | |
|---|---|
| Keyphrase | 8 |
| Organization | 12 |
| Work | 2 |
| Person | 1 |
| Location | 3 |
| RelationOrganizationQuotation | 1 |

Figure 7. 4 – A screenshot of the semantically annotated web document using the API

5. Toggle 'on' or 'off' any annotation types you are not interested in. The different categories of annotations on the web document can be toggled 'on' or 'off' based on user preferences as shown in Figure 7.5.


Sample Web Page - Powered by C
www.a009324a.co.uk/sample.html
Search

Staffordshire University is in the top ten nationally for student satisfaction in The Complete University Guide, published today. This is excellent news for our students, staff and anyone thinking of studying with us as it shows that we are delivering on our mission to provide great teaching, facilities and overall student experience." The scoring, which places the University joint eighth nationally and first among West Midlands universities for student satisfaction, has seen the University rise a further 15 places to 58th overall. The University was highest climber in the rankings last year - largely as a result of improved graduate prospects and student satisfaction - and has continued its improvement. "This is excellent news for our students, staff and anyone thinking of studying with us as it shows that we are delivering on our mission to provide great teaching, facilities and overall student experience." Universities are measured on a range of criteria which includes Entry Standards; Research Quality; Research Intensity; Student-Staff Ratio; Degree Completion; Academic Services Spend; and Student Facilities Spend. Staffordshire University has also ranked 2nd for Academic Services Spend in the West Midlands. This year, West Midlands universities have improved their Graduate Prospects score which are largely above the national average. Five of Staffordshire University's subject areas appeared in the top 10 for graduate prospects including English, Art and Design, Social Work, Communication and Media Studies and Music. Dr Bernard Kingston, Chairman of TheCompleteUniversityGuide.co.uk, commented: "The evidence is clear; the increased emphasis, including funding, on employability and career planning within academic departments and student services is bearing fruit. "Many institutions now have this as a core element of their strategic plans with the introduction of innovative programmes of subject-specific careers events focused on embedding and signposting graduate destinations."

**Annotation Categories**

- Keyphrase 8
- Organization 12
- Work 2
- Person 1
- Location 3
- RelationOrganizationQuotation 1

Figure 7. 5 – A screenshot of the web document with some annotation types toggled off

It is also noteworthy to mention that the API can be utilised with web documents with dynamic content, in which case it will provide semantic annotation for the text resulting from processing data at the application logic and data access layers. Please refer to Appendix B for functionality evaluation of the web interface features through screenshots for the interfaces.

With the web-scale audience for utilising such a service, the ability to automatically scale in and out based on user requests is very vital. Based on this, the microservices were configured for automatic scaling. This provides the application with the capability to meet user demands and maximise computing resources usage. Within the CloudSea Kubernetes cluster, four deployments were configured for auto-scaling from the command-line using kubectl. To display the auto-scaling configurations defined for each deployment, the following command is utilised: ***$ kubectl get hpa.*** The command executed for auto-scaling is as follows:

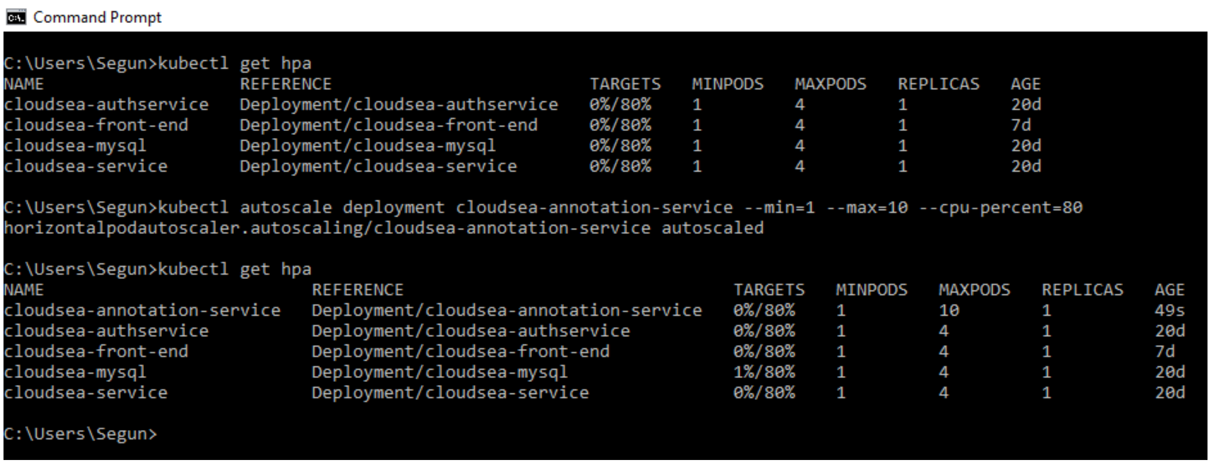
***$ kubectl autoscale deployment <deployment-name> --min=1 --max=10 --cpu-percent=80***

The components of the command are as follows:

- *kubectl: the command-line utility that sends an executable statement to the Kubernetes master node*
- *autoscale: the command being sent to the Kubernetes master node*
- *deployment: specifying the type of resource that the command is meant for*

- *<deployment-name>: where 'deployment-name' is the name of the deployment to be auto-scaled*
- *min: specifies the minimum count of instances for the deployment*
- *max: specifies the maximum count that the deployment should auto-scale to*
- *cpu-percent: specifies the CPU percentage utilisation threshold before auto-scaling*

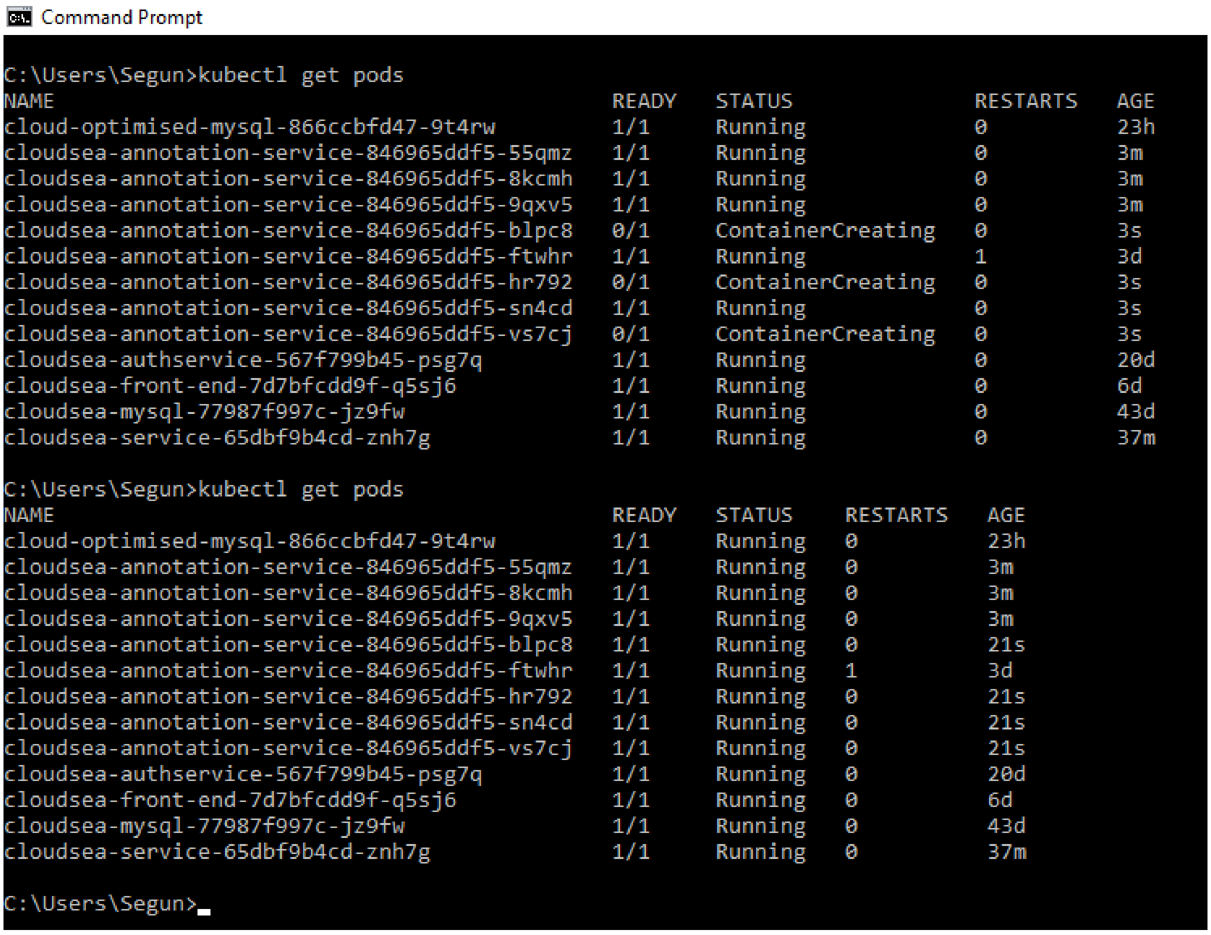
Figure 7.6 shows a screenshot of the auto-scaling configurations for each of the deployments as the first command executed. The next command executed is to configure auto-scaling for another deployment: 'cloudsea-annotation-service'. The third command from the screenshot shows the first command executed again and then the new auto-scaling configuration which has been added to the previously configured ones.



```
C:\Users\Segun>kubectl get hpa
NAME                  REFERENCE                        TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
cloudsea-authservice  Deployment/cloudsea-authservice  0%/80%    1         4         1          20d
cloudsea-front-end    Deployment/cloudsea-front-end    0%/80%    1         4         1          7d
cloudsea-mysql        Deployment/cloudsea-mysql        0%/80%    1         4         1          20d
cloudsea-service      Deployment/cloudsea-service      0%/80%    1         4         1          20d

C:\Users\Segun>kubectl autoscale deployment cloudsea-annotation-service --min=1 --max=10 --cpu-percent=80
horizontalpodautoscaler.autoscaling/cloudsea-annotation-service autoscaled

C:\Users\Segun>kubectl get hpa
NAME                         REFERENCE                               TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
cloudsea-annotation-service  Deployment/cloudsea-annotation-service  0%/80%    1         10        1          49s
cloudsea-authservice         Deployment/cloudsea-authservice         0%/80%    1         4         1          20d
cloudsea-front-end           Deployment/cloudsea-front-end           0%/80%    1         4         1          7d
cloudsea-mysql               Deployment/cloudsea-mysql               1%/80%    1         4         1          20d
cloudsea-service             Deployment/cloudsea-service             0%/80%    1         4         1          20d

C:\Users\Segun>
```

Figure 7. 6 – Configuration for auto-scaling a microservice when CPU utilisation hits 80%

Furthermore, Figure 7.7. shows the "cloudsea-annotation-service" automatically scaling through operating system-level virtualisation to eight instances based on a load test of 100,000 user requests over a period of 10 mins.



```
C:\Users\Segun>kubectl get pods
NAME                                           READY   STATUS              RESTARTS   AGE
cloud-optimised-mysql-866ccbfd47-9t4rw         1/1     Running             0          23h
cloudsea-annotation-service-846965ddf5-55qmz   1/1     Running             0          3m
cloudsea-annotation-service-846965ddf5-8kcmh   1/1     Running             0          3m
cloudsea-annotation-service-846965ddf5-9qxv5   1/1     Running             0          3m
cloudsea-annotation-service-846965ddf5-blpc8   0/1     ContainerCreating   0          3s
cloudsea-annotation-service-846965ddf5-ftwhr   1/1     Running             1          3d
cloudsea-annotation-service-846965ddf5-hr792   0/1     ContainerCreating   0          3s
cloudsea-annotation-service-846965ddf5-sn4cd   1/1     Running             0          3s
cloudsea-annotation-service-846965ddf5-vs7cj   0/1     ContainerCreating   0          3s
cloudsea-authservice-567f799b45-psg7q          1/1     Running             0          20d
cloudsea-front-end-7d7bfcdd9f-q5sj6            1/1     Running             0          6d
cloudsea-mysql-77987f997c-jz9fw                1/1     Running             0          43d
cloudsea-service-65dbf9b4cd-znh7g              1/1     Running             0          37m

C:\Users\Segun>kubectl get pods
NAME                                           READY   STATUS    RESTARTS   AGE
cloud-optimised-mysql-866ccbfd47-9t4rw         1/1     Running   0          23h
cloudsea-annotation-service-846965ddf5-55qmz   1/1     Running   0          3m
cloudsea-annotation-service-846965ddf5-8kcmh   1/1     Running   0          3m
cloudsea-annotation-service-846965ddf5-9qxv5   1/1     Running   0          3m
cloudsea-annotation-service-846965ddf5-blpc8   1/1     Running   0          21s
cloudsea-annotation-service-846965ddf5-ftwhr   1/1     Running   1          3d
cloudsea-annotation-service-846965ddf5-hr792   1/1     Running   0          21s
cloudsea-annotation-service-846965ddf5-sn4cd   1/1     Running   0          21s
cloudsea-annotation-service-846965ddf5-vs7cj   1/1     Running   0          21s
cloudsea-authservice-567f799b45-psg7q          1/1     Running   0          20d
cloudsea-front-end-7d7bfcdd9f-q5sj6            1/1     Running   0          6d
cloudsea-mysql-77987f997c-jz9fw                1/1     Running   0          43d
cloudsea-service-65dbf9b4cd-znh7g              1/1     Running   0          37m

C:\Users\Segun>_
```

Figure 7. 7 – Auto-Scaling for Annotation Microservice through OS-level Virtualisation

## 7.2 Comparative Evaluation

The purpose of this section is to provide a comparison of the CloudSea architecture with existing automatic semantic annotation solutions, both from academic publications and the industry. The basis of comparison are two folds; firstly, based on the set of requirements identified as necessary for addressing the automatic semantic annotation challenge which have been comprehensively reviewed, analysed and developed into capabilities in this thesis. Secondly, from the evaluation of the existing solutions as well as through the arguments for the solutions presented by CloudSea in this thesis, some other factors emerge which constitute vital components for drawing comparisons. A detailed search was conducted to identify the list of solutions utilised for the comparative evaluation; both proposed in academic publications and available as industry tools.

From the comparative evaluation conducted, it was observed that no existing solution meets all the identified requirements to foster an automated system that delivers semantic annotation holistically and as a service to web documents, with most of the requirements unavailable in existing solutions. While the set of requirements for holistic semantic annotation identified in this thesis are proven from research, it was further observed that none of the existing solutions provides any other perspective; either holistic or non-holistic that presents a solution for addressing automatic semantic annotation challenges as CloudSea presents. From the evaluation, annotation data management; to ensure consistency between web documents and annotation data appears to be one of the major areas uncovered among most existing solutions such as Alec et al. (2016), Brank et al. (2018), Albukhitan et al. (2018) and Fiorelli et al. (2015). It is opined that the unavailability of the "Cloud-Driven" pattern proposed in this thesis or any similar pattern among existing solutions might be a major factor for such. This is because annotation data management for a scale as large as the web implies a very high level of computing processing and resources optimisation which can be a daunting task to achieve without the adoption of cloud computing and a mechanism for optimising and fully maximising cloud computing benefits, as "Cloud-Driven" presents. In addition, ontology management to ensure accuracy of annotation data and consistency with web documents can be observed to be an issue among some solutions (Gao et al., 2017; Espinoza & Melga, 2015; Salih, 2013). Furthermore, the general level of automation within different components, as CloudSea presents, to foster a dynamic and continuous delivery of automated semantic annotation as a service is not observable within any of the existing solutions.

It is worth mentioning as well however, that most industry solutions are not focused on semantic annotation, rather on text analytics using semantic technologies and provide wide ranges of solutions based on these technologies. Hence, while a comparison with such solutions was conducted, it is quite important to recognise the goal of such solutions which then impacts on the semantic annotation capabilities they foster. Considering the instance-based nature of text analytics and its scope which is often enterprise-level, in comparison to semantic annotation of web documents which is not necessarily instance-based (as some web document content may not change over very long periods of time) and has a much larger scope; global-level, the unavailability of some semantic annotation requirements in the text analytic tools is understandable.

In addition to facilitating only some of the identified holistic semantic annotation requirements and, in most cases, not leveraging cloud computing or only doing so minimally, some other factors were observed. For instance, some existing solutions were observed to provide semantic annotation for plain text or locally imported documents, without a facility to do so for web documents via a URL. Some of these include Brank et al. (2018) and Yosef et al. (2011). Some others also exist as a web browser plugin (Fiorelli et al., 2015) which neither fosters a universal usage nor ensures consistency, with an array of possible challenges such as compatibility with browsers when they are upgraded, differing requirements from one browser to another, the need to install the plugin among others. Overall, this comparative evaluation demonstrates the novelty of the proposed CloudSea architecture and its capability to address the automation challenge for semantic annotation of web documents at large based on its holistic perspective, cloud-driven methodology, microservices-based architecture as well as other novel specifications presented in this thesis. Table 7.1 presents a summary of the comparative evaluation.

Table 7. 1 – Summary Comparison of CloudSea with Existing Systems

| **Tools** | **Concept Extraction** | **Ontology Population** | **Ontology Selection** | **Ontology Mapping** | **Annotation Data Storage** | **Annotation On-the-Fly** | **Annotation Data Reuse** | **Annotation Data Sharing** | **Annotation Data Auto-Update** | **Ontology Auto-Update** | **Annotation Data Optimisation** | **Annotation Data Colocation** | **Other Remarks** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **The Proposed Requirements for Holistic Semantic Annotation** | | | | | | | | | | | | |
| Alec et al. (2016) | ✓ | ✓ | ✗ | ✗ | ✗ | ✓ | ✗ | ✗ | ✗ | ✓ | ✗ | ✗ | A |
| Brank et al. (2018) | ✓ | ✗ | ✗ | ✗ | ✗ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | A, B |
| Fiorelli et al. (2015) | ✓ | ✓ | ✗ | ✗ | ✓ | ✗ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | A, C |
| Gao et al. (2017) | ✓ | ✓ | ✗ | ✗ | ✗ | ✓ | ✗ | ✗ | ✗ | ✓ | ✗ | ✗ | A |
| Albukhitan et al. (2018) | ✓ | ✓ | ✓ | ✓ | ✗ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | A |
| Da Silva & Cavalcanti (2014) | ✓ | ✓ | ✓ | ✓ | ✗ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | A |
| Yosef et al. (2011) | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | A, B |
| Espinoza & Melga (2015) | ✓ | ✗ | ✗ | ✗ | ✓ | ✓ | ✓ | ✗ | ✓ | ✗ | ✗ | ✗ | A |
| Salih (2013) | ✓ | ✗ | ✗ | ✗ | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | A |
| DBpedia Spotlight (Michel et al., 2018) | ✓ | ✗ | ✗ | ✗ | ✗ | ✓ | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ | G |
| Aylien (Aylien, 2019; Michel et al., 2018) | ✓ | ✓ | ✗ | ✗ | ✗ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | D, E |
| Ambiverse (Ambiverse, 2019; Michel et al., 2018) | ✓ | ✗ | ✗ | ✗ | ✗ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | D, F |
| Cogito API | ✓ | ✓ | ✗ | ✗ | ✗ | ✓ | ✗ | ✗ | ✗ | ✓ | ✗ | ✗ | D, E |
| Dandelion (Michel et al., 2018) | ✓ | ✓ | ? | ? | ✗ | ✓ | ✗ | ✗ | ✗ | ✓ | ✗ | ✗ | D, G |

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| GateCloud (Gate Cloud, 2019; Tablan et al., 2013) | ✓ | ✓ | ✗ | ✗ | ✗ | ✓ | ✓ | ✓ | ✗ | ? | ✗ | ✗ | D, I |
| OntoText (OntoText, 2019) | ✓ | ✓ | ✗ | ✗ | ✓ | ✓ | ✓ | ✓ | ✗ | ✗ | ? | ✗ | D, E |
| Open Calais (Michel et al., 2018) | ✓ | ? | ? | ? | ✗ | ✓ | ✗ | ✗ | ✗ | ? | ✗ | ✗ | D, G, H |
| PoolParty (PoolParty, 2019) | ✓ | ✓ | ✓ | ✓ | ✓ | ✗ | ✓ | ✗ | ✗ | ? | ✓ | ✗ | D, E |
| TextRazor (Michel et al., 2018) | ✓ | ✗ | ✓ | ✓ | ✗ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | D, E |
| Proposed CloudSea | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | J |

| **Description for Symbols in “Other Remarks” Column** |
|---|
| **A** – It does not leverage cloud computing in any context |
| **B** - It only annotates copied text or loaded from a document. It does not provide annotation via web document URL |
| **C** - It exists as a Mozilla Firefox plugin, hence, not universally usable on the Web. |
| **D** – It focuses on text analysis using NLP and leveraging semantic technologies. Hence, the semantic annotation capability is only accessible programmatically; requiring a programmer or developer for implementation. |
| **E** - Leverages cloud computing but exact context not clear. |
| **F** - No longer in service |
| **G** - No reference to leveraging cloud computing. |
| **H** - No adequate documentation available online. Hence, some features are not determinable. |
| **I** – Only leverages cloud computing at the "Cloud-Based" maturity level. |
| **J** - The proposed architecture by this thesis, which is holistic, cloud-driven and microservices-based for an automated system that provides managed services for continuous delivery of automatic semantic annotation to web documents at large. |

## 7.3 Experimental Evaluation

An experiment is an empirical investigation that examines fundamental relations and processes. Experimentation in software engineering involves the gathering of evidence, through measurements and experiments that involves software systems such as products, processes and resources (Menzies et al., 2016). The data obtained is intended to be used as the basis of theories about specific processes within information systems. These theories backed up by data is a fundamental tenet of scientific methods. Several research groups primarily use empirical and experimental techniques for conducting research. Data analysis is an important component of experimental evaluation and is based on a design template for the experiment. While the analysis summarises data collected and treatment of the data, it is also important to note that the analysis is not expected to interpret results and data should be analysed in accordance with the design. Furthermore, the outcomes of an experiment need to be properly analysed to generate knowledge. The adoption of good practices towards finding explanations for results and following standard reporting guidelines will help in generating this knowledge (Jedlitschka et al., 2008).

Chapter 4 of this thesis proposed a Cloud Computing Maturity Model for Holistic Semantic Annotation which defined different maturity levels for application deployment in the cloud generally and specifically for holistic semantic annotation. This section presents an experimental evaluation for the different instances of the prototype developed in this thesis. The differences are based on the maturity levels identified within the cloud computing maturity model. The objective is to gather, present and analyse performance metrics for each one of them. The analysis would be based on the results obtained and how they relate to the research hypothesis and critical evaluations made over the course of this research. For this experimental evaluation, several series of iterative load tests were carried out on the API endpoints for each of the different versions of the prototype. The API endpoint in each case is the medium through which the application provides automatic and holistic semantic annotation for corresponding web documents based on the 'Annotation On-the-Fly' microservice. Testing API endpoints is a standard and well-accepted means of evaluating API functionality. According to Wang et al. (2017), load tests for API endpoints provides a relatively accurate means of evaluating them. This is also supported by Stahlin et al. (2016) and Bangare et al. (2012). One of the major reasons for this is because it provides a precise and stream-lined test in comparison with testing via a web document that calls an API. For the latter, the different requests made by the web document other than the API request would also constitute determinants in the overall values obtained, creating an additional task of separating API request results from non-API request results. Several types of requests are commonly made by web documents via protocols such as HTTP, HTTPS, database query requests and lots more.

Furthermore, the load tests were carried out using a very popular cloud-based API endpoints load testing tool known as Loader.io (Loader.io, 2019) and accessible via https://loader.io as at the time of writing. Loader.io provides cloud-based load and scalability testing via the SaaS cloud model for web applications and API endpoints. It is a product of SendGrid Labs; an Amazon AWS partner and providers of an AWS email service; SendGrid Email Delivery Service, a leading industry product delivering efficient and scalable, cloud-based email services (AWS Marketplace, 2019). The tests were carried out based on a set of test configuration parameters which define the settings and requirements for each of them. Table 7.2 presents the test configuration settings.

Table 7. 2 – Load Tests Configuration for Experimental Evaluation

| **Description** | **Value** |
|---|---|
| Environment | Loader.io (Cloud-Based API Endpoints Load Testing) |
| Test Type | Clients per test |
| Clients Type | Virtual |
| Clients Count | Ranging from 10,000 to 100,000 |
| Test Duration | Ranging from 1 to 10 minutes |
| Error Threshold | 50% |
| Timeout | 30 seconds |
| Iteration Count | 7 |
| Test Command | **GET Requests**<br>Non-Cloud Monolithic<br>http://www.a009324a.co.uk/mono/public/annotation/annotate/guest.js |
|  | Non-Cloud Microservices<br>http://a009324a.co.uk/micro/annotation-service/public/api/annotate/guest.js |
|  | Cloud-Based Monolithic<br>http://cloud-based.a009324a.co.uk/annotation/annotate/guest.js |
|  | Cloud-Based Microservices<br>http://cloud-based-micro.a009324a.co.uk/api/annotate/guest.js |
|  | Cloud-Optimised<br>http://cloud-optimised.a009324a.co.uk/annotation/annotate/guest.js |
|  | Cloud-Driven<br>http://cloud-annotation-service.a009324a.co.uk/api/annotate/guest.js |
| Scenario 1 | 10,000 requests over 10 minutes |
| Scenario 2 | 20,000 requests over 10 minutes |
| Scenario 3 | 40,000 requests over 10 minutes |
| Scenario 4 | 100,000 requests over 10 minutes |
| Scenario 5 | 100,000 requests over 5 minutes |
| Scenario 6 | 100,000 requests over 1 minute |
| Metric 1 | Average Response Time (in milliseconds) |
| Metric 2 | Successful Responses (count) |
| Metric 3 | Timeout Errors (count) |
| Metric 4 | Network Errors (count) |
| Metric 5 | 400/500 Errors (count) |
| Metric 6 | Average Error Rate (percentage) |

| | |
|---|---|
| Average API Response Time (in milliseconds) | 200 |

From Table 7.2, the different metrics listed are described below:

**Metric 1: Average Response Time**

Response Time in the context of client/server architecture refers to the time between a server receiving client request and the time client receives a response to the request from the server (Loader.io, 2019). For each test, the response time for every request made is monitored by Loader.io. The average response time is the average value of time taken to get responses to all requests made for each iteration and is specified in milliseconds.

**Metric 2: Successful Responses**

For each test, several requests will be made via Loader.io load generators and each request is expected to receive a response from the API endpoint. This parameter would count the number of successful responses to requests made for each iterative test. This value would be expected to be as close to 100% as possible and closely observed to note when the percentage success rate starts dropping.

**Metric 3: Timeout Errors**

Timeout refers to the amount of time a client making a request will wait for a response before giving up on receiving it. If the requesting client has waited for the specified timeout period and has not received any response, it stops waiting and constitutes a timeout for the specific request. So, the timeout error is defined by instances when the threshold for the requesting client to wait for a response passed without any response being received. The threshold defined for the configuration settings was 50%.

**Metric 4: Network Errors**

A network error implies that the application or server making a request cannot reach the application/server that is meant to process and respond to the request. Several reasons could be responsible for a network error. This includes DNS resolution errors, TCP connection problems or the server closing/resetting the connection with no response (Loader.io). In this scenario, a network error can occur if the domain name assigned for the specific application version does not resolve to the Amazon AWS IP address or if there is a TCP connectivity issue. It could also be that the API is unavailable or unable to process the request due to the size of the load to process at that point in time which could be excessive for the application or the server based on available computing resources such as CPU and memory. In such cases, there will be no response at all from the API. This will be measured as counts, in terms of the number of network errors obtained during a specific test iteration.

### Metric 5: 400/500 Errors

400/500 errors refer to a series of error codes obtainable over a request such as HTTP in a client-server model. Error codes starting with '4' implies a client-side error while ones starting with '5' implies a server-side error. While several error codes belong to this classification, Table 7.3 presents the most common ones over HTTP as obtained from Oracle Corporation (2018).

Table 7. 3 – Common 400/500 Error Codes for Web Server Responses

| Error Code | Description |
|---|---|
| 400 | Bad input parameter. This should be further described by the error message. |
| 401 | Unauthorised. This implies that an invalid authorisation token was passed by the client. |
| 404 | Not Found. This implies that the resource being requested for cannot be found. |
| 405 | Method Not allowed. This implies that the HTTP verb is unsupported. |
| 409 | Conflict. This implies that there is a conflict in the request. |
| 411 | Length Required. This implies that a length is required to be specified for the content header |
| 412 | Pre-condition failed. |
| 429 | Too many requests. This implies requests are too many for the server to handle at the time. |
| 500 | Internal Server Error. A wide range of issues could cause this and would need further troubleshooting. |
| 503 | Service Unavailable. This implies that the service being requested for is unavailable currently. |

### Metric 6: Average Error Rate

This returns the average value of errors obtained from timeout, network and 400/500 errors. As these parameters are based on counts, the average value across the three is obtained and presented as a percentage value. This value would be expected to be as close as possible to 0.00%. It would be vital to observe when this value starts rising for the tests.

**GET Command**

The GET command is a HTTP request method for data transfer between a client and a server over a network. The command can take one or more parameters which constitutes data sent to a server. The format and data type are expected to be understandable by the server and able to be processed. It is a standard method for making requests to API endpoints over HTTP (Loader.io). The remainder of this section will be focused on the different load experimental tests, their objectives, specific settings, data gathering and presentation, analysis of the data and comparisons where applicable.

### 7.3.1 Non-Cloud Monolithic Maturity Level

An instance of the developed monolithic version of CloudSea prototype was set up on a traditional web hosting account which is a non-cloud computing environment. While this instance has same fundamental functionality as the CloudSea prototype, it runs from a non-cloud computing environment and is based on the monolithic software architectural pattern and design. Hence, the purpose for setting this up is to carry out load tests on it and observe its performance. The performance metrics can then be evaluated accordingly. There is no access to server specification or performance monitoring tools for the public hosting platform. However, the load test environment is cloud-based, and the test configuration is as presented in Table 7.2.

#### 7.3.1.1 Scenario 1

Scenario 1 (as defined in Table 7.2) experiments conducted for this maturity level provided results as presented in Table 7.4.

Table 7. 4 – Scenario 1 Results for Non-Cloud Monolithic

| | Average Response Time | Successful Reponses | Timeout Errors | Network Errors | 400/500 Errors | Average Error Rate |
|---|---|---|---|---|---|---|
| Test 1 | 442 | 9987 | 11 | 0 | 0 / 0 | 0.11 % |
| Test 2 | 871 | 9966 | 1 | 0 | 0 / 27 | 0.28 % |
| Test 3 | 333 | 9997 | 0 | 0 | 0 / 0 | 0.0 % |
| Test 4 | 832 | 9929 | 2 | 0 | 0 / 0 | 0.02 % |
| Test 5 | 418 | 9997 | 0 | 0 | 0 / 0 | 0.0 % |
| Test 6 | 501 | 9996 | 0 | 0 | 0 / 0 | 0.0 % |
| Test 7 | 487 | 9996 | 0 | 0 | 0 / 0 | 0.0 % |
| | 555 | 99.81 % | 2 | 0 | 0 / 4 | 0.06 % |

From Table 7.4, it can be noted that seven load tests were carried out for 'Non-Cloud Monolithic' with 10,000 client requests for each iteration. The tests were based on settings defined in Table 7.2. From the results, it can be observed that average response time for the API was 555 milliseconds for 10,000 requests distributed over a period of 10 minutes; across 7 iterations. It can also be observed that 99.81% of the responses were successful over the series of tests. The total timeout count was fourteen while there were twenty-seven '400/500' error codes, with all of them being '500' error codes. This implies that the errors were server-side; pointing to possible issues with availability of adequate server resources to process the requests. The overall error rate was approximately 0.06%. The average response time was quite high (at 555 milliseconds) and both timeout and '500' errors are values which signalled some level of stress on the API based on the available computing resources. Figure 7.8 presents a graph of average values for response time and error rates.

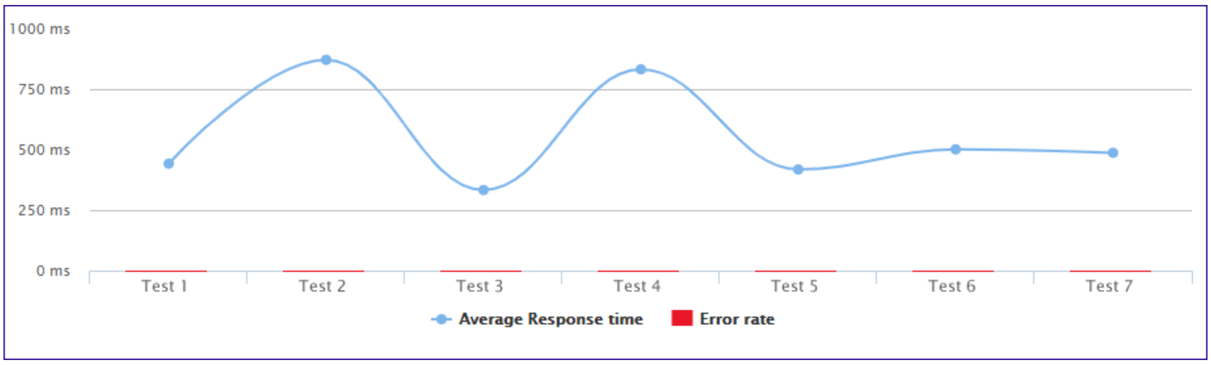


Figure 7. 8 - Average Response Time and Error Rate for Non-Cloud Monolithic Scenario 1

With values obtained for "Scenario 1", it was necessary to conduct "Scenario 2" in which the client request count doubled from 10,000 to 20,000.

#### 7.3.1.2 Scenario 2

For "Scenario 2" experiments with this maturity level, results obtained are presented in Table 7.5.

Table 7. 5 - Scenario 2 Results for Non-Cloud Monolithic

| | **Average Response Time** | **Successful Reponses** | **Timeout Errors** | **Network Errors** | **400/500 Errors** | **Average Error Rate** |
|---|---|---|---|---|---|---|
| Test 1 | 4241 | 17811 | 168 | 0 | 0 / 1974 | 10.71 % |
| Test 2 | 4424 | 17776 | 175 | 0 | 0 / 1879 | 10.27 % |
| Test 3 | 6899 | 16750 | 170 | 0 | 0 / 2982 | 15.76 % |
| Test 4 | 6357 | 17164 | 166 | 0 | 0 / 2343 | 12.55 % |
| Test 5 | 5369 | 16767 | 182 | 0 | 0 / 2885 | 15.43 % |
| Test 6 | 4853 | 17122 | 169 | 0 | 0 / 2373 | 12.71 % |
| Test 7 | 7800 | 17501 | 229 | 0 | 0 / 2151 | 11.90 % |
| Average | 5706 | 86.35 % | 180 | 0 | 0 / 2370 | 12.76 % |

From Table 7.5, a further decline in the application performance can be noticed. There was an increase of up to 700% in the average response time; from 555 to 5,706 milliseconds. In addition, the successful response rate dropped from 99.81% to 86.35%. Similar trends can be noticed across the timeout and '400/500' error rates. It can be generally observed that the application struggled to cope with the 20,000-client request count for "Scenario 2". Figure 7.9 presents average values for response time and error rates for these.

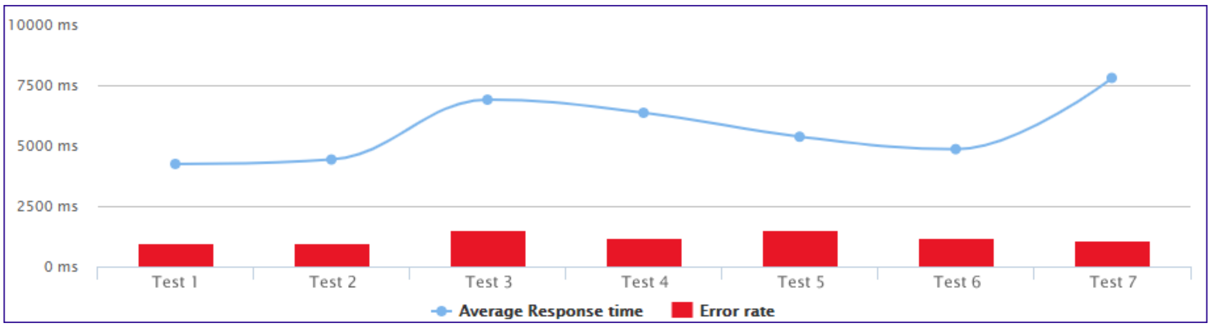


Figure 7. 9 - Average Response Time and Error Rate for Non-Cloud Monolithic Scenario 2

#### 7.3.1.3 Analysis

From test results obtained for this maturity level, while the server specification is unknown and would vary across different web hosting solutions available, it shows that essential cloud characteristics such as elasticity, resource pooling, rapid provisioning and lots more are very vital with applications intended for large-scale utilisation on the web. As the hosting environment for this instance is non-cloud, cloud computing features such as auto-scaling and load balancing could not be configured to provide cloud computing characteristics and enable the application cope with the number of requests. The next section presents results for the 'Non-Cloud Microservices' Maturity Level which is deployed on same hosting platform.

### 7.3.2 Non-Cloud Microservices Maturity Level

An instance of CloudSea prototype based on the 'Non-Cloud Microservices' Maturity Level of the Holistic Semantic Annotation Cloud Computing Maturity Model presented in Chapter 4 was also set up in a non-cloud computing environment and load tested for experimental evaluation. It was set up on the same hosting platform as the 'Non-Cloud Monolithic' described in section 7.3.1. With these two running from the same platform, it provides a very good basis for comparing their performance. Hence, the purpose for this experiment is to compare the same web application built using two different software architectural patterns; monolithic and microservices architecture. The comparisons are based on the differences drawn between them within the model in Chapter 4 and summarised in Figure 4.14. Two series of load tests were carried out in this case too; Scenario 1 and 2, with each one comprising of seven iterations. The test configuration is same, as described in Table 7.2.

#### 7.3.2.1 Scenario 1

Results obtained for Scenario 1 for this maturity level are presented in Table 7.6.

Table 7. 6 - Scenario 1 Results for Non-Cloud Microservices

| | **Average Response Time** | **Successful Reponses** | **Timeout Errors** | **Network Errors** | **400/500 Errors** | **Average Error Rate** |
|---|---|---|---|---|---|---|
| Test 1 | 279 | 10000 | 0 | 0 | 0 / 0 | 0.0 % |
| Test 2 | 276 | 10000 | 0 | 0 | 0 / 0 | 0.0 % |
| Test 3 | 268 | 10000 | 0 | 0 | 0 / 0 | 0.0 % |
| Test 4 | 261 | 10000 | 0 | 0 | 0 / 0 | 0.0 % |
| Test 5 | 274 | 10000 | 0 | 0 | 0 / 0 | 0.0 % |
| Test 6 | 267 | 10000 | 0 | 0 | 0 / 0 | 0.0 % |
| Test 7 | 268 | 10000 | 0 | 0 | 0 / 0 | 0.0 % |
| Average | 270 | 100% | 0 | 0 | 0 / 0 | 0.0 % |

From Table 7.6, it can be observed that this pattern performed better than its monolithic version for Scenario 1. While the average response time for 'Non-Cloud Monolithic' was 555 milliseconds, it was 270 milliseconds for the 'Non-Cloud Microservices', constituting over 105% better response time. In addition, improvements can be noticed with the successful response and error rates, with this maturity level reporting 100% and 0.0% for them respectively; implying a complete successful response and no errors. Figure 7.10 presents the average values for response time and error rates for these.

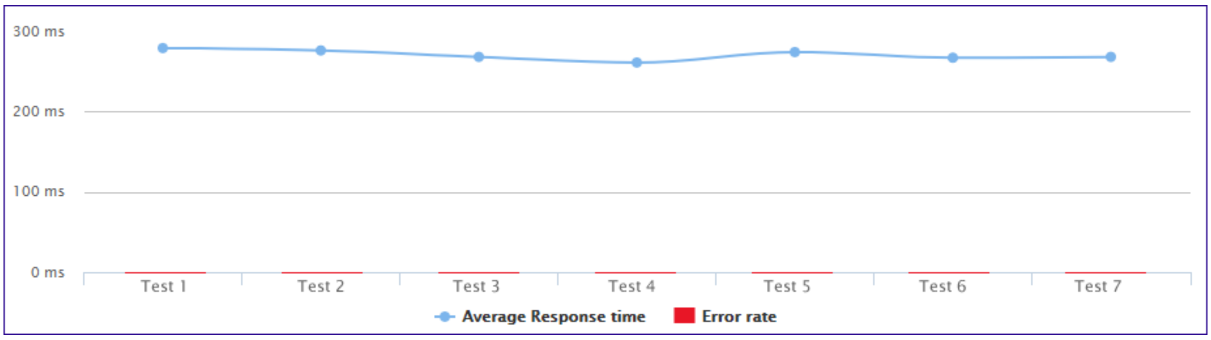


Figure 7. 10 - Average Response Time and Error Rate for Non-Cloud Microservices Scenario 1

#### 7.3.2.2 Scenario 2

With the impressive set of results obtained in the first series (Scenario 1), the second series of tests (Scenario 2) for this maturity level were conducted and results obtained are presented in Table 7.7.

Table 7. 7 - Scenario 2 Results for Non-Cloud Microservices

| | **Average Response Time** | **Successful Reponses** | **Timeout Errors** | **Network Errors** | **400/500 Errors** | **Average Error Rate** |
|---|---|---|---|---|---|---|
| Test 1 | 444 | 19968 | 31 | 0 | 0 / 0 | 0.16 % |
| Test 2 | 588 | 19967 | 0 | 0 | 0 / 0 | 0.0 % |
| Test 3 | 2589 | 18524 | 142 | 0 | 0 / 1027 | 5.85 % |
| Test 4 | 8151 | 17380 | 190 | 0 | 0 / 2065 | 11.28 % |
| Test 5 | 6883 | 17613 | 208 | 0 | 0 / 2139 | 11.74 % |
| Test 6 | 6631 | 17207 | 158 | 0 | 0 / 2282 | 12.20 % |
| Test 7 | 8165 | 16479 | 156 | 0 | 0 / 3154 | 16.55 % |
| Average | 4778 | 90.81 % | 126 | 0 | 0 / 1524 | 8.25 % |

From Table 7.7, it can be observed that in a similar way to the ‘Non-Cloud Monolithic’, this maturity level showed a decline in performance from Scenario 1 to Scenario 2. It can be observed that the average response time increased from 270 milliseconds to 4778 milliseconds. In addition, the successful response rate dropped from 100% to 90.81%. The same trends can be noticed with timeout and ‘400/500’ errors which were zero with its values for Scenario 1. Figure 7.11 presents a graph of the average response time and error rates for these series of tests.

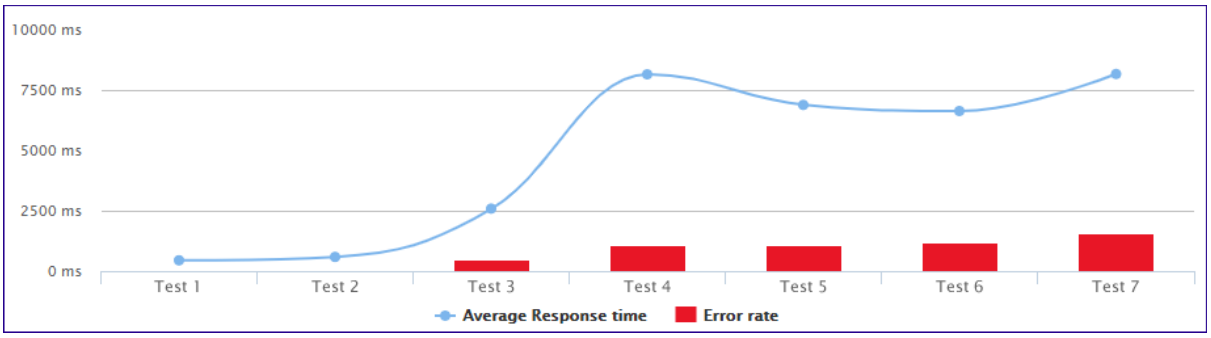


Figure 7. 11 - Average Response Time and Error Rates for Non-Cloud Microservices Scenario 2

#### 7.3.2.3 Comparative Analysis

Over the two series of tests for both 'Non-Cloud Monolithic' and 'Non-Cloud Microservices' maturity levels, while their performances declined from Scenario 1 to 2, the results obtained for the 'Non-Cloud Microservices' were observed to be better across the different performance metrics in comparison with those of the 'Non-Cloud Monolithic'. Based on the huge performance decline after Scenario 2 for both versions, no further tests were conducted for them. While the 'Non-Cloud Microservices' performed better than the 'Non-Cloud Monolithic', they are both still inhibited with being hosted in a non-cloud environment which does not facilitate capabilities such as auto-scaling, load balancing, elasticity and lots more. However, it provides a valid basis for concluding that the microservices architectural pattern provides better software agility than monolithic / N-tier architectures as concluded with the model summarised in Figure 4.14. Furthermore, as these two could not be configured for automatic scaling to cater for the additional requests, since they are deployed in a non-cloud computing environment, it provides a level of support to the concept of leveraging cloud computing to deliver automated semantic annotation on a large-scale to web documents.

### 7.3.3 Cloud-Based Monolithic Maturity Level

This instance is based on the 'Cloud-Based Monolithic' Maturity Level of the Holistic Semantic Annotation Cloud Computing Maturity Model presented in Chapter 4. As the name suggests, it is hosted in a cloud environment; Amazon AWS. However, it is based on a monolithic architectural pattern and focuses on leveraging the IaaS model of cloud computing. The deployment technique is a direct migration from a non-cloud environment to a virtual host in the cloud without any changes to the application code or architectural pattern. Hence, the purpose of this test is in two phases. Firstly, to provide a comparison between this maturity level and the 'Non-Cloud Monolithic' as the comparison would be between same application deployed in different environments. Cloud computing elasticity features such as auto-scaling is not configured for the 'Cloud-Based Monolithic' as doing so would give it a competitive advantage over the 'Non-Cloud Monolithic'. Secondly, to provide a comparison between this maturity level and the 'Non-Cloud Microservices', in which case, results would be observed for any progression in performance, with cognisance to the differences in their architectural patterns. Some specification about the virtual host for this deployment is detailed in Table 7.8.

Table 7. 8 - Virtual Host Specification for Cloud-Based Monolithic

| **Description** | **Value** |
|---|---|
| Instance Type | m4.large |
| Virtual CPUs | 2 |
| RAM | 4 GB |
| Amazon Image Type | Canonical, Ubuntu, 18.04 LTS, AMD64 bionic image |
| Platform | Ubuntu |
| Virtualisation | HVM |

With this maturity level, Scenarios 1 and 2 (of Table 7.2) were conducted. Scenario 3 was not conducted due to the observable stress after Scenario 2 as was the case with the previously tested maturity levels; Non-Cloud Monolithic and Non-Cloud Microservices.

### 7.3.3.1 Scenario 1

Table 7.9 presents results obtained from Scenario 1 experiments for this maturity level.

Table 7. 9 - Scenario 1 Results for Cloud-Based Monolithic

| | **Average Response Time** | **Successful Reponses** | **Timeout Errors** | **Network Errors** | **400/500 Errors** | **Average Error Rate** |
|---|---|---|---|---|---|---|
| Test 1 | 108 | 9996 | 0 | 0 | 0 / 0 | 0.0 % |
| Test 2 | 92 | 9997 | 0 | 0 | 0 / 0 | 0.0 % |
| Test 3 | 88 | 9997 | 0 | 0 | 0 / 0 | 0.0 % |
| Test 4 | 89 | 9998 | 0 | 0 | 0 / 0 | 0.0 % |
| Test 5 | 85 | 9997 | 0 | 0 | 0 / 0 | 0.0 % |
| Test 6 | 87 | 9996 | 0 | 0 | 0 / 0 | 0.0 % |
| Test 7 | 89 | 9997 | 0 | 0 | 0 / 0 | 0.0 % |
| Average | 91 | 99.97 % | 0 | 0 | 0 / 0 | 0.0 % |

Results obtained for these tests as presented in Table 7.9 were impressive; with an average response time of 91 milliseconds, average successful response rate of 99.97% and an infinitesimal error rate. Figure 7.12 presents the average response time and error rate.

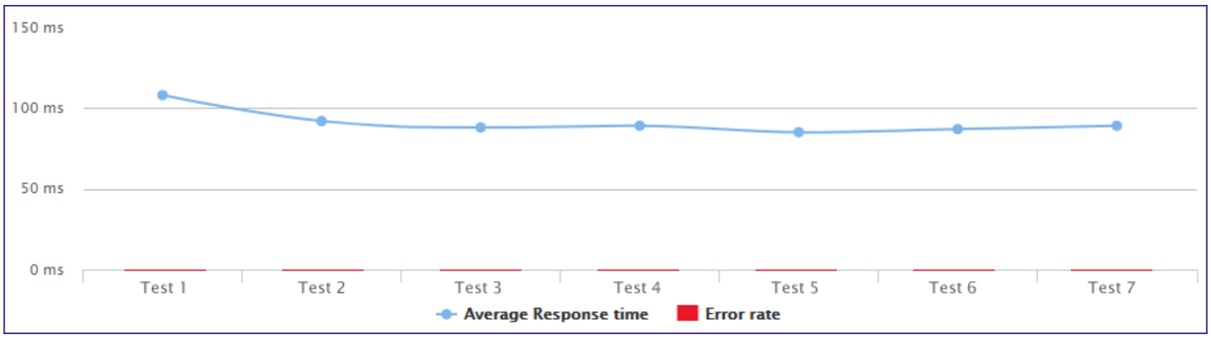


Figure 7. 12 - Average Response Time and Error Rate for Cloud-Based Monolithic Scenario 1

### 7.3.3.2 Scenario 2

With the relatively good performance with Scenario 1, Scenario 2 was conducted, and results obtained are presented in Table 7.10.

Table 7. 10 - Scenario 2 Results for Cloud-Based Monolithic

| | **Average Response Time** | **Successful Reponses** | **Timeout Errors** | **Network Errors** | **400/500 Errors** | **Average Error Rate** |
|---|---|---|---|---|---|---|
| Test 1 | 452 | 19852 | 144 | 0 | 0 / 0 | 0.72 % |
| Test 2 | 616 | 19812 | 184 | 0 | 0 / 0 | 0.92 % |
| Test 3 | 755 | 19653 | 336 | 0 | 0 / 0 | 0.17 % |
| Test 4 | 928 | 19483 | 345 | 0 | 0 / 0 | 0.17 % |
| Test 5 | 543 | 19837 | 148 | 0 | 0 / 0 | 0.74 % |
| Test 6 | 615 | 19822 | 169 | 0 | 0 / 0 | 0.85 % |
| Test 7 | 678 | 19784 | 185 | 0 | 0 / 0 | 0.93 % |
| Average | 655 | 98.75 % | 215 | 0 | 0 / 0 | 0.64 % |

From the results obtained as presented in Table 7.10, a decline in performance can be observed for this too from Scenario 1 to 2. There was a drastic effect of the increased load as the average response time increased from 91 milliseconds to 655 milliseconds, representing an increase of about 620%. In addition, timeout errors can be noticed from the results which suggests inadequate computing resources for the application to respond to client requests before the timeout threshold of 30 seconds. With the values obtained from Table 7.10, no further load tests were carried out on this deployment based on the emergence of a significant percentage of errors in the context of server responses to client requests. Figure 7.13 presents the average response time and error rate.

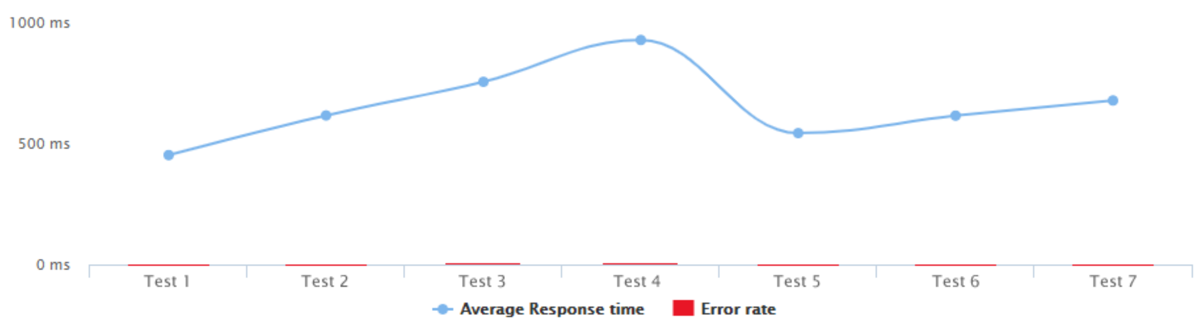


Figure 7. 13 - Average Response Time and Error Rate for Cloud-Based Monolithic Scenario 2

#### 7.3.3.3 Comparative Analysis

Results obtained for this maturity level can be observed to be better than those of the 'Non-Cloud Monolithic' across both Scenarios 1 and 2. Considering that they are exactly same application, hosted in different environments, this deployment can be observed to have leveraged cloud computing to perform better and produce results accordingly. For comparison with the 'Non-Cloud Microservices', while this deployment runs on a monolithic architecture, it still had a better overall performance than the 'Non-Cloud Microservices'. From the results obtained for Scenario 1, this deployment had a 197% better response time; at 91 milliseconds compared to 270 milliseconds for the 'Non-Cloud Microservices'.

It was a wider margin for Scenario 2 at an average of 655 to 4778 milliseconds, representing a 630% difference. These suggest that the 'Cloud-Based Monolithic' benefitted from being hosted in a cloud computing environment, even in comparison with a microservices application. Overall, results obtained with this deployment and their comparison with the previous two are in line with the Holistic Semantic Annotation Cloud Computing Maturity Model presented in Chapter 4. From the results comparison, even though microservices offers better software agility, it was observed that leveraging cloud computing can greatly enhance application performance due to resources capability for high performance and data processing, in comparison with a non-cloud computing environment.

### 7.3.4 Cloud-Based Microservices Maturity Level

This maturity level demonstrates an instance based on the microservices software architecture and hosted in the cloud. However, despite its potentials to leverage operating system-level virtualisation due to the architectural pattern, it has been deployed directly on a virtual host, like with the 'Cloud-Based Monolithic'. Hence, the comparison in this case would be with the

'Cloud-Based Monolithic' to observe if same pattern noticed with the two non-cloud deployments would emerge as well. Scenarios 1,2 and 3 were conducted for this deployment and results obtained are analysed as follows.

#### 7.3.4.1 Scenario 1

Results obtained for Scenario 1 with this deployment are presented in Table 7.11.

Table 7. 11 – Scenario 1 Results for Cloud-Based Microservices

| | **Average Response Time** | **Successful Reponses** | **Timeout Errors** | **Network Errors** | **400/500 Errors** | **Average Error Rate** |
|---|---|---|---|---|---|---|
| Test 1 | 94 | 9995 | 0 | 0 | 0 / 0 | 0.0 % |
| Test 2 | 80 | 9995 | 0 | 0 | 0 / 0 | 0.0 % |
| Test 3 | 80 | 9994 | 0 | 0 | 0 / 0 | 0.0 % |
| Test 4 | 77 | 9995 | 0 | 0 | 0 / 0 | 0.0 % |
| Test 5 | 79 | 9995 | 0 | 0 | 0 / 0 | 0.0 % |
| Test 6 | 82 | 9994 | 0 | 0 | 0 / 0 | 0.0 % |
| Test 7 | 82 | 9993 | 0 | 0 | 0 / 0 | 0.0 % |
| Average | 82 | 99.94 % | 0 | 0 | 0 / 0 | 0.0 % |

From Table 7.11, an impressive set of results can be observed; with an average response rate of 82 milliseconds, successful response rate of 99.94% and quite an infinitesimal amount of errors. Figure 7.14 further presents the average response time and error rate.

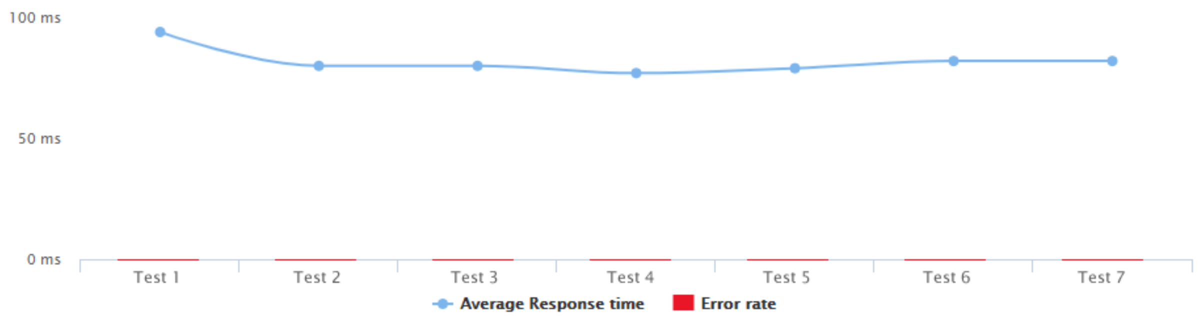


Figure 7. 14 – Average Response Time and Error Rate for Cloud-Based Microservices Scenario 1

#### 7.3.4.2 Scenario 2

Scenario 2 results for this deployment are presented in Table 7.12.

Table 7. 12 – Scenario 2 Results for Cloud-Based Microservices

| | **Average Response Time** | **Successful Reponses** | **Timeout Errors** | **Network Errors** | **400/500 Errors** | **Average Error Rate** |
|---|---|---|---|---|---|---|
| Test 1 | 191 | 19995 | 0 | 0 | 0 / 0 | 0.0 % |
| Test 2 | 251 | 19983 | 0 | 0 | 0 / 0 | 0.0 % |
| Test 3 | 290 | 19973 | 24 | 0 | 0 / 0 | 0.12 % |
| Test 4 | 384 | 19852 | 126 | 0 | 0 / 0 | 0.63 % |
| Test 5 | 391 | 19946 | 31 | 0 | 0 / 0 | 0.15 % |
| Test 6 | 141 | 19999 | 0 | 0 | 0 / 0 | 0.0 % |
| Test 7 | 147 | 19997 | 0 | 0 | 0 / 0 | 0.0 % |
| Average | 256 | 99.82 % | 26 | 0 | 0 / 0 | 0.13 % |

From Table 7.12, while the overall application performance declined in comparison with the Scenario 1 results, the average response time, at 256 milliseconds is relatively alright even though the average error rate is slightly high in this context at 0.13%. Figure 7.15 further presents the average response time and error rate.

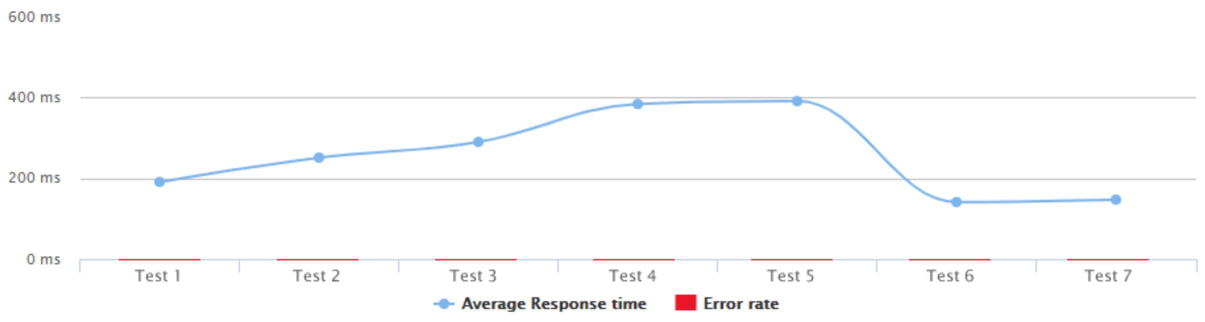


Figure 7. 15 – Average Response Time and Error Rate for Cloud-Based Microservices Scenario 2

With the error rate from Scenario 2 at 0.13% compared to the 0.64% obtained for same tests with the 'Cloud-Based Monolithic', Scenario 3 was conducted, and results obtained are presented in section 7.3.4.3.

#### 7.3.4.3 Scenario 3

Scenario 3 results obtained for this deployment are presented in Table 7.13.

Table 7. 13 – Scenario 3 Results for Cloud-Based Microservices

| | **Average Response Time** | **Successful Reponses** | **Timeout Errors** | **Network Errors** | **400/500 Errors** | **Average Error Rate** |
|---|---|---|---|---|---|---|
| Test 1 | 8081 | 30348 | 8846 | 0 | 0 / 0 | 22.57 % |
| Test 2 | 11647 | 25412 | 13633 | 0 | 0 / 0 | 34.92 % |
| Test 3 | 8123 | 30119 | 8991 | 0 | 0 / 0 | 22.99 % |
| Test 4 | 5858 | 33417 | 6099 | 0 | 0 / 0 | 15.43 % |
| Test 5 | 13880 | 22555 | 16574 | 0 | 0 / 0 | 42.36 % |
| Test 6 | 12821 | 24264 | 15185 | 0 | 0 / 0 | 34.89 % |
| Test 7 | 16232 | 17937 | 17986 | 0 | 0 / 0 | 50.07 % |
| Average | 10949 | 65.73 % | 12473 | 0 | 0 / 0 | 31.89 % |

The Scenario 3 results obtained for this deployment, as presented in Table 7.13 demonstrates a significant level of stress on the deployment with the 40,000 client requests count per iteration; producing a much higher value of 31.89% error rate. Figure 7.16 further presents the average response time and error rate.

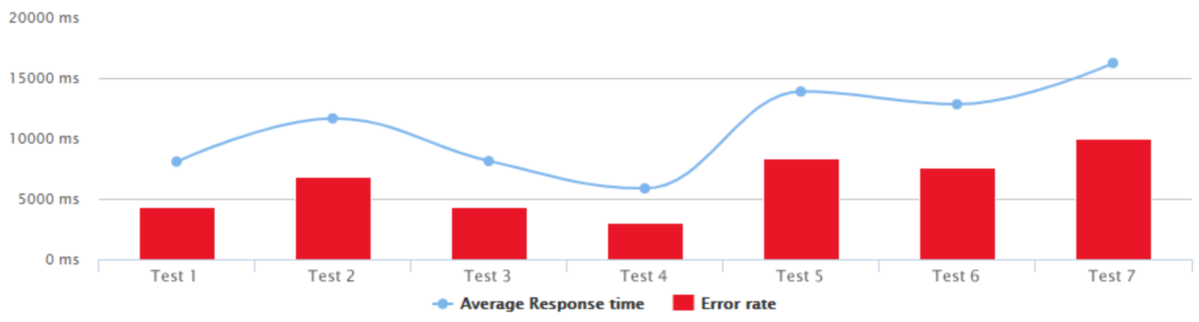


Figure 7. 16 – Average Response Time and Error Rate for Cloud-Based Microservices Scenario 3

#### 7.3.4.4 Comparative Analysis

The results obtained for this deployment can be observed to have been better than those of the 'Cloud-Based Monolithic'. Even though they are hosted on virtual hosts with same specification and capabilities, the 'Cloud-Based Microservices' yielded better results. This suggests that the latter is leveraging its microservices software architecture, which is the only difference between both. With an average response rate of 655 milliseconds for the 'Cloud-Based Monolithic' at Scenario 2, this pattern returned at less than half of that value averagely; 256 milliseconds. A similar pattern was observed between the two non-cloud deployments. Overall, two significant patterns can be observed from the 'Cloud-Based Microservices' results. Firstly, in comparison with the 'Cloud-Based Monolithic', it produced better results which suggests that the microservices architecture provided better software agility. This same pattern was observed between the 'Non-Cloud Monolithic' and 'Non-Cloud Microservices' as well. Secondly, the cloud-based deployments can be observed to have performed better than the non-cloud ones; in line with the relatively high-performance that can be obtained from cloud computing.

### 7.3.5 Cloud-Optimised Maturity Level

This instance represents the 'Cloud-Optimised' Maturity Level of the Holistic Cloud Computing Maturity Model and as described in Figure 4.14. It is deployed in the cloud with operating system-level virtualisation using Docker containers and running it on Kubernetes Container Orchestration platform as a monolith application. The experiments in this case aim to demonstrate the impact of operating system-level virtualisation and subsequently, the implementation of orchestration. So, even though this deployment runs on a monolithic architecture, it is "optimised" for cloud environment through the OS-level virtualisation and orchestration. Hence, this maturity level would be compared with the 'Cloud-Based Monolithic' which is same application but neither leverages OS-level virtualisation nor orchestration. It would also be compared with the 'Cloud-Based Microservices' to observe if there is the expected maturity, in terms of progressive performance. The test configuration is still same as defined in Table 7.2. A summary overview of the specification for the deployment platform is provided in Table 7.14.

Table 7. 14 – CloudSea Kubernetes Cluster Summary Overview

| **Description** | **Value** |
|---|---|
| Number of EC2 Instances | 3 |
| Instance Type | m4.large |
| Kubernetes Master Node | 1 EC2 Instance |
| Kubernetes Worker Nodes | 2 EC2 Instances |
| Virtual CPUs (per instance) | 2 |
| RAM (per instance) | 4 GB |
| Virtualisation | Operating System Level |
| Amazon Image Type | Kubernetes 1.10 Base Image - Debian Jessie amd64 |
| OS-level Virtualisation Engine | Docker Containers |
| Orchestration Engine | Kubernetes |

## 7.3.5.1 Scenario 1

From the experiments conducted for this deployment based on "Scenario 1", results obtained are presented in Table 7.15.

Table 7. 15 - Scenario 1 Results for Cloud-Optimised

| | **Average Response Time** | **Successful Reponses** | **Timeout Errors** | **Network Errors** | **400/500 Errors** | **Average Error Rate** |
|---|---|---|---|---|---|---|
| Test 1 | 151 | 9996 | 0 | 0 | 0 / 0 | 0.0 % |
| Test 2 | 128 | 9994 | 0 | 0 | 0 / 0 | 0.0 % |
| Test 3 | 132 | 9997 | 0 | 0 | 0 / 0 | 0.0 % |
| Test 4 | 173 | 9996 | 0 | 0 | 0 / 0 | 0.0 % |
| Test 5 | 129 | 9998 | 0 | 0 | 0 / 0 | 0.0 % |
| Test 6 | 132 | 9997 | 0 | 0 | 0 / 0 | 0.0 % |
| Test 7 | 156 | 9998 | 0 | 0 | 0 / 0 | 0.0 % |
| Average | 143 | 99.97 % | 0 | 0 | 0 / 0 | 0.0 % |

From Table 7.15, the average response time for Scenario 1 in this case can be observed to be 143 milliseconds, with an average success rate of 99.97% and an infinitesimal error rate. While these values are impressive, a quick comparison with Scenario 1 results for both 'Cloud-Based' maturity levels show a decline. However, further tests would be conducted before a comparative analysis. Figure 7.17 presents the average response and error rates for these tests.

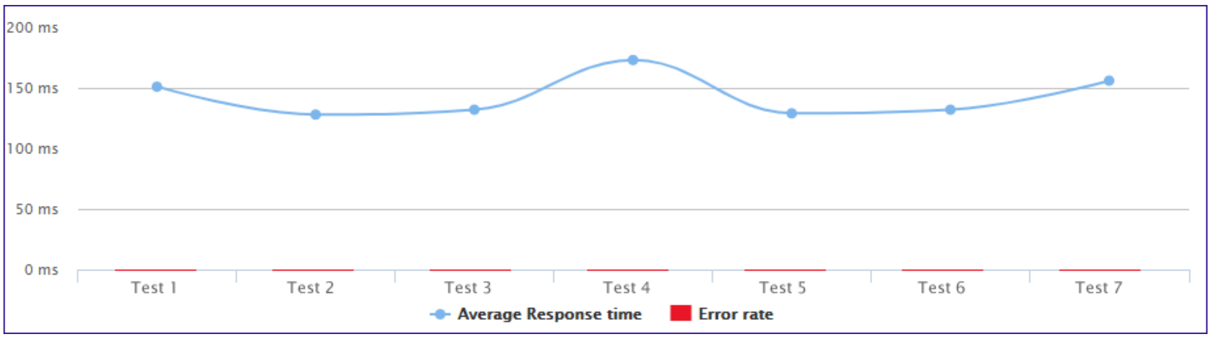


Figure 7. 17 - Average Response Time and Error Rate for Cloud-Optimised Scenario 1

### 7.3.5.2 Scenario 2

Scenario 2 results for this deployment are presented in Table 7.16.

Table 7. 16 - Scenario 2 Results for Cloud-Optimised

| | **Average Response Time** | **Successful Reponses** | **Timeout Errors** | **Network Errors** | **400/500 Errors** | **Average Error Rate** |
|---|---|---|---|---|---|---|
| Test 1 | 205 | 20000 | 0 | 0 | 0 / 0 | 0.0 % |
| Test 2 | 202 | 20000 | 0 | 0 | 0 / 0 | 0.0 % |
| Test 3 | 128 | 20000 | 0 | 0 | 0 / 0 | 0.0 % |
| Test 4 | 170 | 20000 | 0 | 0 | 0 / 0 | 0.0 % |
| Test 5 | 212 | 20000 | 0 | 0 | 0 / 0 | 0.0 % |
| Test 6 | 255 | 20000 | 0 | 0 | 0 / 0 | 0.0 % |
| Test 7 | 131 | 20000 | 0 | 0 | 0 / 0 | 0.0 % |
| Average | 186 | 100 % | 0 | 0 | 0 / 0 | 0.0 % |

From Table 7.16, it can be observed that the pattern still performed well with Scenario 2, returning an average response time of 186 milliseconds, a 100% successful response rate and no errors at all. While the average response time increased from Scenario 1 to Scenario 2, a value of 186 milliseconds is still regarded as healthy. In addition, the average success rate can be observed to have been better. Figure 7.18 presents a graph of the average response time and error rates for these tests.

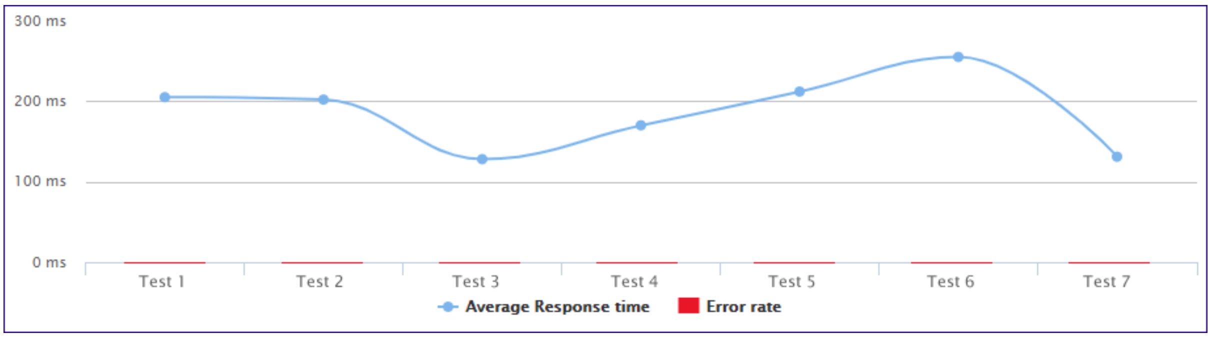


Figure 7. 18 - Average Response Time and Error Rates for Cloud-Optimised Scenario 2

#### 7.3.5.3 Scenario 3

The third series of tests; Scenario 3 was conducted based on results obtained from Scenario 2. The results are presented in Table 7.17.

Table 7. 17 - Scenario 3 Results for Cloud-Optimised

| | Average Response Time | Successful Reponses | Timeout Errors | Network Errors | 400/500 Errors | Average Error Rate |
|---|---|---|---|---|---|---|
| Test 1 | 133 | 40000 | 0 | 0 | 0 / 0 | 0.0 % |
| Test 2 | 144 | 40000 | 0 | 0 | 0 / 1 | 0.0 % |
| Test 3 | 134 | 40000 | 0 | 0 | 0 / 0 | 0.0 % |
| Test 4 | 148 | 40000 | 0 | 0 | 0 / 0 | 0.0 % |
| Test 5 | 133 | 40000 | 0 | 0 | 0 / 0 | 0.0 % |
| Test 6 | 138 | 39999 | 0 | 0 | 0 / 0 | 0.0 % |
| Test 7 | 136 | 40000 | 0 | 0 | 0 / 0 | 0.0 % |
| Average | 138 | 100% | 0 | 0 | 0 / 1 | 0.0 % |

From Table 7.17, Scenario 3 results for this maturity level can be observed to have been even better than its Scenario 1 and 2 results despite the increase to 40,000-client request count in this case. The successful response rate was approximately 100% with only one 500 code error out of the entire 280,000 requests over the seven iterations. Figure 7.19 presents the average response time and error rates for these.

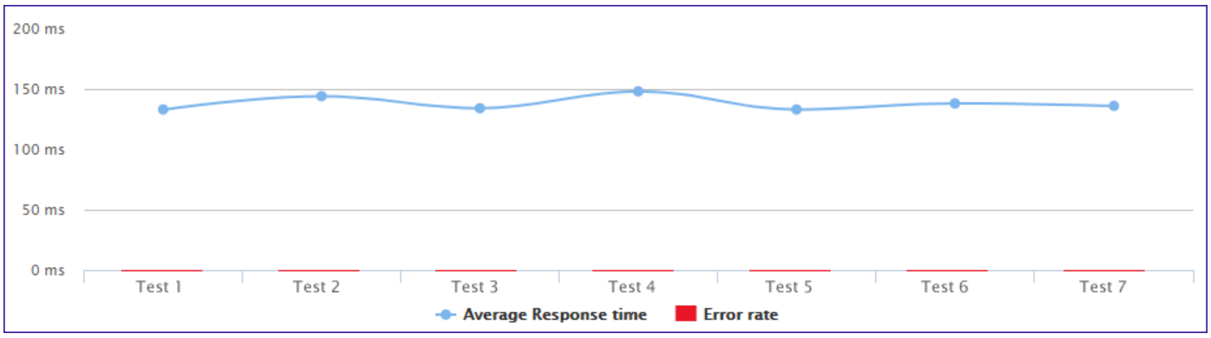


Figure 7. 19 - Average Response Time and Error Rate for Cloud-Optimised Scenario 3

#### 7.3.5.4 Scenario 4

With the Scenario 3 results obtained, Scenario 4 was conducted for this maturity level and results obtained are presented in Table 7.18.

Table 7. 18 - Scenario 4 Results for Cloud-Optimised

| | **Average Response Time** | **Successful Reponses** | **Timeout Errors** | **Network Errors** | **400/500 Errors** | **Average Error Rate** |
|---|---|---|---|---|---|---|
| Test 1 | 13030 | 4491 | 596 | 0 | 0 / 4394 | 52.63 %* |
| Test 2 | 11252 | 2191 | 159 | 0 | 0 / 2033 | 50.01 %* |
| Test 3 | 9668 | 2213 | 37 | 0 | 0 / 2290 | 51.26 %* |
| Test 4 | 6928 | 478 | 0 | 0 | 0 / 2853 | 85.65 %* |
| Test 5 | 6741 | 233 | 0 | 0 | 0 / 3094 | 93.0 %* |
| Test 6 | 12751 | 839 | 0 | 0 | 0 / 1775 | 67.9 %* |
| Test 7 | 10219 | 1277 | 0 | 0 | 0 / 1412 | 52.51 %* |
| Average | 10084 | 1.68 % | 113 | 0 | 0 / 2550 | 64.71 % |
| *Tests were aborted automatically after reaching the 50% error threshold defined in test configuration. Hence, the average error rates were at the point of test abortion. | | | | | | |

A drastic difference in performance can be observed with the ‘Cloud-Optimised’ maturity level from results obtained for Scenario 3 to 4, as presented in Table 7.18. While the tests were automatically aborted before processing all requests for each iteration due to the 50% error threshold, the impact of 100,000 client requests per iteration can be observed to have greatly impacted the results negatively. Figure 7.20 presents a graph of the average response time and error rates for these.

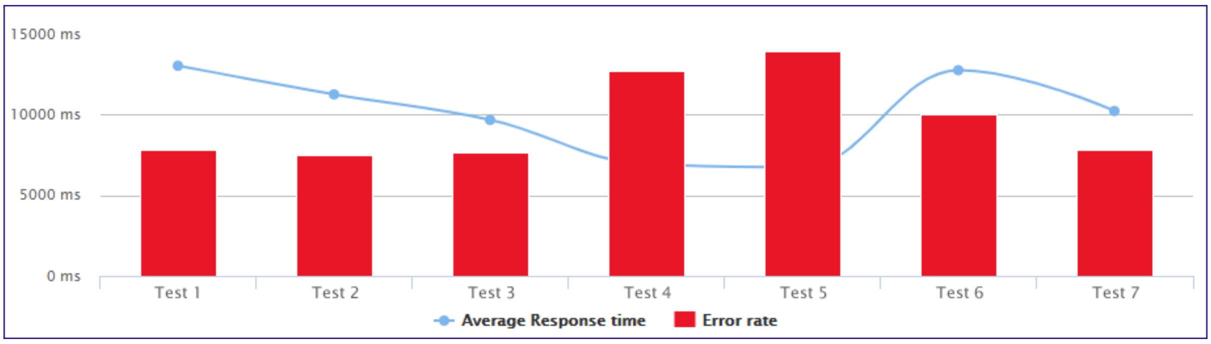


Figure 7. 20 - Average Response Time and Error Rates for Cloud-Optimised Scenario 4

#### 7.3.5.5 Comparative Analysis

Results obtained for Scenarios 1 and 2 can be observed to have been better with the ‘Cloud-Based Monolithic’ than the ‘Cloud-Optimised’. However, while performance for the former declined drastically with higher load stress in Scenario 3, the ‘Cloud-Optimised’ can be observed to have produced even better results, suggesting the OS-level virtualisation influence; a factor the ‘Cloud-Based Monolithic’ does not leverage. In comparison with the ‘Cloud-Based

Microservices', a similar pattern can be observed with Scenarios 1 and 2, in which 'Cloud-Based Microservices' performed better but declined drastically for Scenario 3. The suggestion from that pattern is that while both 'Cloud-Based' maturity levels struggled with increased loads, the 'Cloud-Optimised' was able to leverage OS-level virtualisation and cope with the increased loads accordingly.

Overall, the cloud-optimised version was observed to have performed very well until the client request count was increased to 100,000 requests over 10 minutes for Scenario 4. Based on the features described in Figure 4.14 for the 'Cloud-Optimised' maturity level, it is important to note that with OS-level virtualisation and container orchestration, additional resources could still be configured for the deployment to automatically scale and be able to process the 100,000 requests per 10 minutes which it could not do with Scenario 4. However, for the purpose of these experiments, an upper limit of resources substantial enough for the evaluation was set.

### 7.3.6 Cloud-Driven Maturity Level

This instance is based on the 'Cloud-Driven' Maturity Level of the Holistic Semantic Annotation Cloud Maturity Model of Figure 4.14. As described in Chapter 4, it is an enhanced 'Cloud-Native' Maturity Level as they are same application. However, the 'Cloud-Driven' Maturity Level involves the integration of a Continuous Integration Mechanism (as presented in Figure 4.12) to facilitate an automated semantic annotation application life cycle. Hence, the tests in this section applies to both 'Cloud-Native' and 'Cloud-Driven' Maturity Levels. Like the 'Cloud-Optimised', these also leverage OS-level virtualisation using Docker containers and run on the Kubernetes container orchestration platform. However, unlike the monolithic 'Cloud-Optimised', these are based on microservices architecture. It is also worth reiterating that the prototype implementation in Chapter 6 is based on the 'Cloud-Driven' maturity level. Cloud-Driven, as defined in this research requires an automated process for continuous integration and continuous delivery to a cloud-native application; in which case provides an automation lifecycle required for data-intensive and large-scale use applications on the web.

The tests in this case would enable a comparison with the monolithic 'Cloud-Optimised' as well as observe if the performance progression of the maturity model persists. The test configuration is still same, as presented in Table 7.2.

#### 7.3.6.1 Scenario 1

From Scenario 1 for this maturity level, the results obtained are presented in Table 7.19.

Table 7. 19 - Scenario 1 Results for Cloud-Driven

| | **Average Response Time** | **Successful Reponses** | **Timeout Errors** | **Network Errors** | **400/500 Errors** | **Average Error Rate** |
|---|---|---|---|---|---|---|
| Test 1 | 125 | 9996 | 0 | 0 | 0 / 0 | 0.0 % |
| Test 2 | 124 | 9994 | 0 | 0 | 0 / 0 | 0.0 % |
| Test 3 | 125 | 9997 | 0 | 0 | 0 / 0 | 0.0 % |
| Test 4 | 142 | 9996 | 0 | 0 | 0 / 0 | 0.0 % |
| Test 5 | 130 | 9998 | 0 | 0 | 0 / 0 | 0.0 % |
| Test 6 | 125 | 9997 | 0 | 0 | 0 / 0 | 0.0 % |
| Test 7 | 121 | 9998 | 0 | 0 | 0 / 0 | 0.0 % |
| Average | 127 | 99.97 % | 0 | 0 | 0 / 0 | 0.0 % |

From Table 7.19, an impressive set of results can be observed for this maturity level; with an average response time of 127 milliseconds, a 99.97% successful response rate and no errors. These results can also be observed to represent a better performance than those obtained with Scenario 1 results for 'Cloud-Optimised' as presented in Table 7.15. Figure 7.21 presents the average response time and error rates for these.

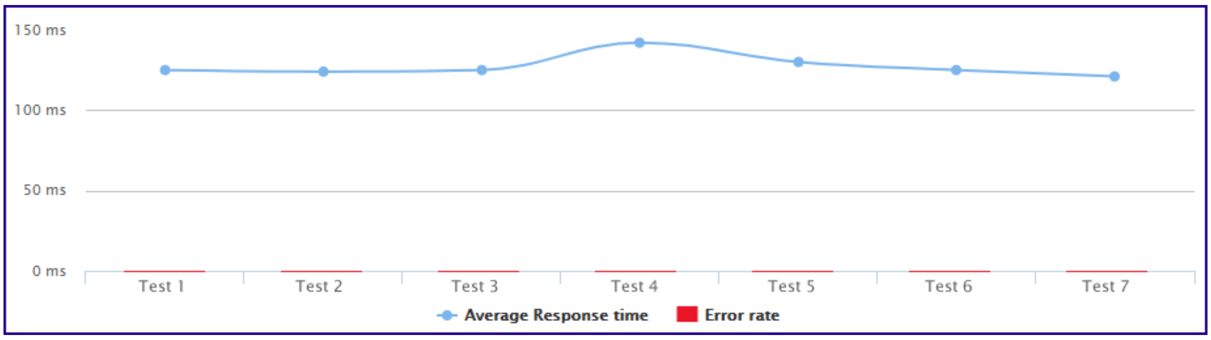


Figure 7. 21 - Average Response Time and Error Rates for Cloud-Driven Scenario 1

#### 7.3.6.2 Scenario 2

From Scenario 2 for this maturity level, the results obtained are presented in Table 7.20.

Table 7. 20 - Scenario 2 Results for Cloud-Driven

| | **Average Response Time** | **Successful Reponses** | **Timeout Errors** | **Network Errors** | **400/500 Errors** | **Average Error Rate** |
|---|---|---|---|---|---|---|
| Test 1 | 124 | 20000 | 0 | 0 | 0 / 0 | 0.0 % |
| Test 2 | 124 | 19997 | 0 | 0 | 0 / 0 | 0.0 % |
| Test 3 | 123 | 20000 | 0 | 0 | 0 / 0 | 0.0 % |
| Test 4 | 121 | 20000 | 0 | 0 | 0 / 0 | 0.0 % |
| Test 5 | 122 | 19999 | 0 | 0 | 0 / 0 | 0.0 % |
| Test 6 | 120 | 20000 | 0 | 0 | 0 / 0 | 0.0 % |
| Test 7 | 143 | 20000 | 0 | 0 | 0 / 0 | 0.0 % |
| Average | 125 | 99.997 % | 0 | 0 | 0 / 0 | 0.0 % |

The data presented in Table 7.20 for Scenario 2 here shows a similar pattern with Scenario 1 results, in Table 7.19, with a slightly better performance in this case. Figure 7.22 further presents the average response time and error rates for these.

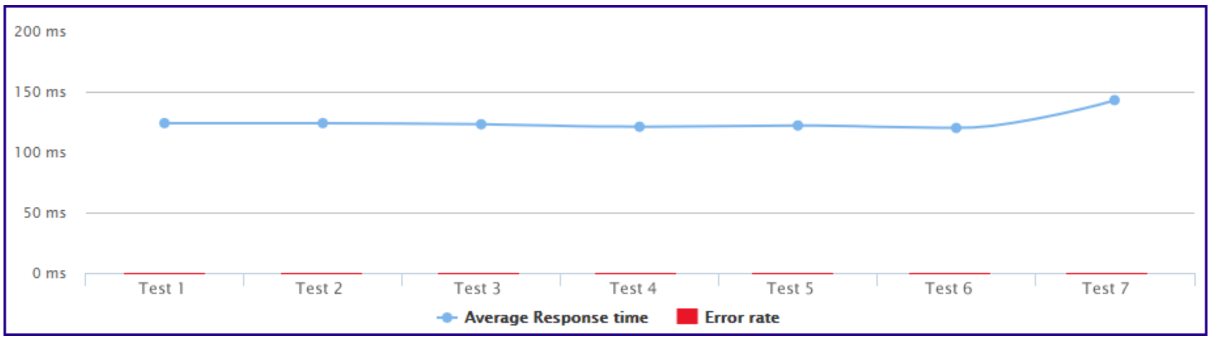


Figure 7. 22 - Average Response Time and Error Rates for Cloud-Driven Scenario 2

### 7.3.6.3 Scenario 3

Scenario 3 results for this maturity level are presented in Table 7.21.

Table 7. 21 - Scenario 3 Results for Cloud-Driven

| | **Average Response Time** | **Successful Reponses** | **Timeout Errors** | **Network Errors** | **400/500 Errors** | **Average Error Rate** |
|---|---|---|---|---|---|---|
| Test 1 | 129 | 40000 | 0 | 0 | 0 / 0 | 0.0 % |
| Test 2 | 126 | 40000 | 0 | 0 | 0 / 0 | 0.0 % |
| Test 3 | 121 | 39999 | 0 | 0 | 0 / 0 | 0.0 % |
| Test 4 | 124 | 40000 | 0 | 0 | 0 / 0 | 0.0 % |
| Test 5 | 126 | 40000 | 0 | 0 | 0 / 0 | 0.0 % |
| Test 6 | 127 | 39998 | 0 | 0 | 0 / 0 | 0.0 % |
| Test 7 | 122 | 39998 | 0 | 0 | 0 / 0 | 0.0 % |
| Average | 125 | 100 % | 0 | 0 | 0 / 0 | 0.0 % |

The Scenario 3 results also demonstrate an impressive performance, with yet a slightly better performance than Scenario 2 results. With the two maturity levels that leverage OS-level virtualisation, a pattern of better performance from Scenarios 1 to 3 can be observed, which as suggested earlier seems to demonstrate the impact of OS-level virtualisation for both. Figure 7.23 further presents the average response time and error rates for these.

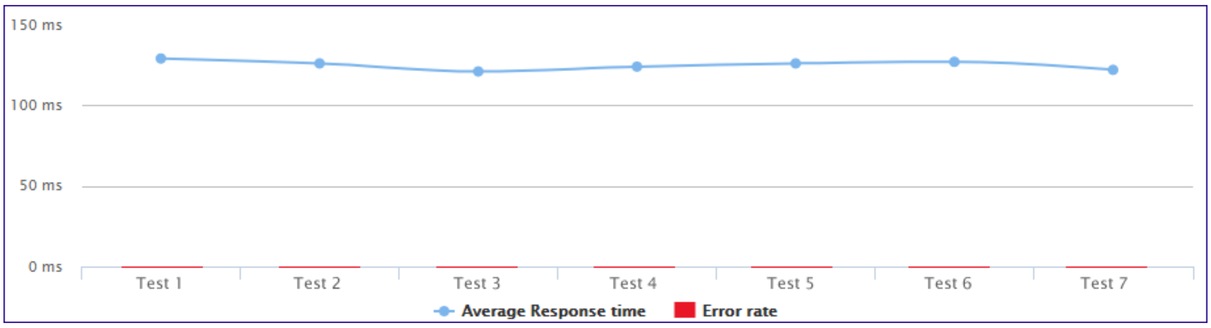


Figure 7. 23 - Average Response Time and Error Rates for Cloud-Driven Scenario 3

#### 7.3.6.4 Scenario 4

Based on the Scenario 3 results obtained, Scenario 4 was also conducted, and the results are presented in Table 7.22.

Table 7. 22 - Scenario 4 Results for Cloud-Driven

| | **Average Response Time** | **Successful Reponses** | **Timeout Errors** | **Network Errors** | **400/500 Errors** | **Average Error Rate** |
|---|---|---|---|---|---|---|
| Test 1 | 125 | 100000 | 0 | 0 | 0 / 0 | 0.0 % |
| Test 2 | 126 | 100000 | 0 | 0 | 0 / 0 | 0.0 % |
| Test 3 | 125 | 99999 | 0 | 0 | 0 / 0 | 0.0 % |
| Test 4 | 124 | 99999 | 0 | 0 | 0 / 0 | 0.0 % |
| Test 5 | 125 | 100000 | 0 | 0 | 0 / 0 | 0.0 % |
| Test 6 | 123 | 99992 | 0 | 0 | 0 / 0 | 0.0 % |
| Test 7 | 124 | 100000 | 0 | 0 | 0 / 0 | 0.0 % |
| Average | 124 | 100 % | 0 | 0 | 0 / 0 | 0.0 % |

From Table 7.22, a consistency in the results from Scenarios 1 to 4 can be observed, with the performance metrics being just slightly varying across them. This implies that the different client request counts; from 10,000 to 100,000 over 10 minutes have all been within the capability of the deployment to process. Figure 7.24 further presents the average response time and error rates for these.

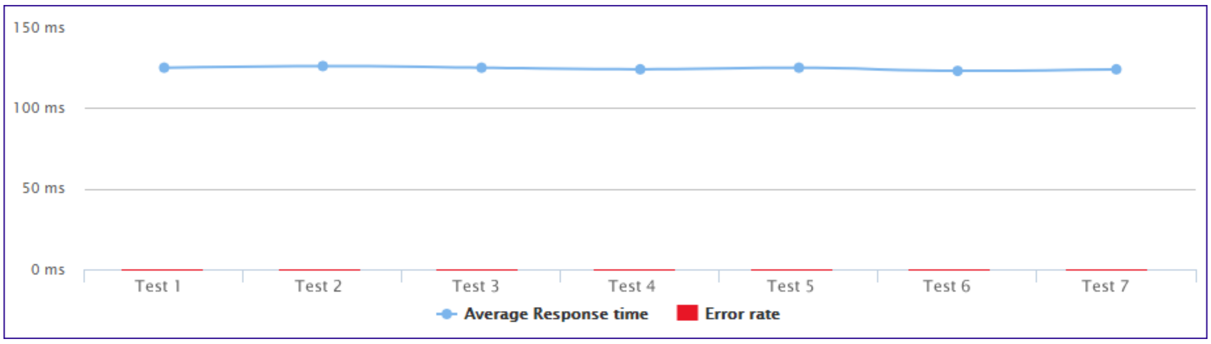


Figure 7. 24 – Average Response Time and Error Rates for Cloud-Driven Scenario 4

### 7.3.6.5 Scenario 5

With the impressive results obtained across Scenarios 1 to 4, Scenario 5 which has the same test configuration as Scenario 4 except for the reduction in processing time from 10 minutes to 5 minutes was conducted and the results obtained are presented in Table 7.23.

Table 7. 23 - Scenario 5 Results for Cloud-Driven

| | **Average Response Time** | **Successful Reponses** | **Timeout Errors** | **Network Errors** | **400/500 Errors** | **Average Error Rate** |
|---|---|---|---|---|---|---|
| Test 1 | 10637 | 9842 | 1002 | 0 | 0 / 10472 | 53.83 % |
| Test 2 | 13293 | 12049 | 1400 | 0 | 0 / 12130 | 52.89 % |
| Test 3 | 11739 | 3139 | 54 | 0 | 0 / 3723 | 54.61 % |
| Average | 11890 | 8.34 % | 819 | 0 | 0 / 8775 | 53.78 % |

The Scenario 5 results as presented in Table 7.23 can be observed to have provided a good level of stress on the deployment, with an average response time of 11,890 milliseconds, a successful response rate of 8.34 %, average 819 timeout errors, average 8,775 500 code errors and an overall error rate of 53.78 % at the point of the 50% error threshold set in the test configuration. Figure 7.25 further presents the average response time and error rates for these.

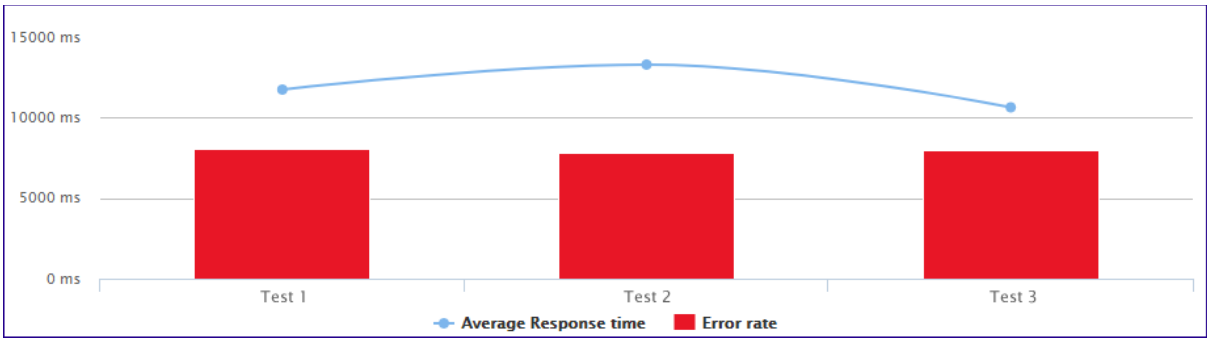


Figure 7. 25 - Average Response Time & Error Rates for Cloud-Driven Scenario 5

With the results obtained from Scenario 5 as presented in Table 7.23 and Figure 7.25, Scenario 6 which is based on sending 100,000 client requests to the API over a period of one minute was not conducted due to the stress level achieved already.

### 7.3.6.6 Comparative Analysis

In drawing comparisons between ‘Cloud-Optimised’ and ‘Cloud-Driven’, a close level of performance can be observed between them from Scenarios 1 to 3, with ‘Cloud-Driven’ providing slightly better results. However, Scenario 4 results shows a massive difference between them with ‘Cloud-Optimised’ returning an error rate of 64.71% while ‘Cloud-Driven’ had a zero-error rate. Similar huge differences can be noticed with the average response time and successful responses for both during Scenario 4. The pattern of microservices performing better than a monolith is once again observed with these two. The close level of performance observed between them across Scenarios 1 to 3 seem to be attributable to the size and scale of the ‘Cloud-Driven’ in comparison to the ‘Cloud-Optimised’. With only 3 microservices for the ‘Cloud-Driven’, it constitutes a ratio of 3:1 with the ‘Cloud-Optimised’. With a full-fledged implementation and an evolvement over time, the ratio would increase, hence, the impact of the decomposition offered by the microservices software architecture would be expected to become more prevalent and impact on the results of a comparison between both. While progression can be observed with the maturity levels from one to another, results obtained with Scenario 1 for ‘Cloud-Based Microservices’ emerged the best with an average response time of 82 milliseconds. However, with the other scenarios, ‘Cloud-Driven’ produced the best results and the progression remained linear henceforth.

Based on the overall results obtained, ‘Cloud-Driven’ has demonstrated the ability of the prototype implementation detailed in Chapter 6 to provide large-scale, automated and holistic semantic annotation to documents on the web. With more resources and its extension to implement the other microservices of CloudSea, as described in Chapter 5, it presents huge potentials towards the semantic web. Finally, it is noteworthy to reiterate that these tests did not utilise hypervisor-level virtualisation. Doing so would enable the different cloud maturity levels to be able to meet even higher levels of demand without a specific limit. However, for the basis of this experimental evaluation, automated scaling of virtual host; which is based on hypervisor-level virtualisation was not configured for two reasons; firstly because of budget constraints and secondly, to set an upper limit as a benchmark for the tests.

### 7.3.7 Experimental Evaluation Summary

The experimental evaluation conducted has been very thorough and comprehensive, with the results obtained in full support of the descriptions and solution proffered with the Holistic Semantic Annotation Cloud Maturity Model. Firstly, the results demonstrate the impact of cloud computing for deploying the CloudSea prototype as compared to deployment in a non-cloud computing environment. This is demonstrated by the results obtained with the cloud deployments in comparison with the non-cloud ones. Secondly, the evaluation supports the proposition that microservices architecture provides better software agility and capability for large-scale, data-intensive applications such as the CloudSea prototype implementation. This is demonstrated through results obtained with the microservices deployments across both cloud and non-cloud, which provided better results than their corresponding monolithic deployments.

Thirdly, the evaluation supports the facilitation of applications, especially microservices-based applications with operating system-level virtualisation, also known as containerisation and the orchestration of the containers for automated deployment, scaling, management and monitoring, as these optimise cloud computing resources more than merely leveraging hypervisor-level virtualisation; as described in Chapter 4. Fourthly, the evaluation demonstrates the need for a full application automation lifecycle that ensures continuous integration and delivery of software updates in a very dynamic environment such as the semantic web. The application automation lifecycle framework facilitates collaboration between different development teams and the continuous implementation of updates and upgrades to a system, such as would be required for providing holistic semantic annotation to documents on the web. Table 7.24 presents a summary of the experimental evaluation conducted.

Table 7. 24 – Summary Results from the Experimental Evaluation

| | | **Non-Cloud Monolithic** | **Non-Cloud Microservices** | **Cloud-Based Monolithic** | **Cloud-Based Microservices** | **Cloud-Optimised** | **Cloud-Driven** |
|---|---|---|---|---|---|---|---|
| 10,000 over 10mins for 7 iterations (70,000 requests) | Avg. Response Time | 555 | 270 | 91 | 82 | 143 | 127 |
| | Avg. Successful Responses | 99.81 % | 100 % | 99.97 % | 99.94 % | 99.97 % | 99.97 % |
| | Timeout Errors | 2 | 0 | 0 | 0 | 0 | 0 |
| | 400/500 Errors | 0 / 4 | 0 / 0 | 0 / 0 | 0 / 0 | 0 / 0 | 0 / 0 |
| | Avg. Error Rate | 0.06 % | 0.0 % | 0.0 % | 0.0 % | 0.0 % | 0.0 % |
| 20,000 over 10mins for 7 iterations (140,000 requests) | Avg. Response Time | 5,706 | 4,778 | 655 | 256 | 186 | 125 |
| | Successful Responses | 86.35 % | 90.81 % | 98.75 % | 99.82 % | 100 % | 99.997 % |
| | Timeout Errors | 180 | 126 | 215 | 26 | 0 | 0 |
| | 400/500 Errors | 0 / 2370 | 0 / 1524 | 0 / 0 | 0 / 0 | 0 / 0 | 0 / 0 |
| | Avg. Error Rate | 12.76 % | 8.25 % | 0.64 % | 0.13 % | 0.0 % | 0.0 % |
| 40,000 over 10mins | Avg. Response Time | | | | 10949 | 138 | 125 |
| | Successful Responses | | | | 65.73 % | 100 % | 100 % |
| | Timeout Errors | | | | 12473 | 0 | 0 |
| | 400/500 Errors | | | | 0 / 0 | 0 / 1 | 0 / 0 |
| | Avg. Error Rate | | | | 31.89 % | 0.0 % | 0.0 % |
| 100,000 over 10mins | Avg. Response Time | | | | | 10,084 | 124 |
| | Successful Responses | | | | | 1.68 % | 100 % |
| | Timeout Errors | | | | | 113 | 0 |
| | 400/500 Errors | | | | | 0 / 2,550 | 0 / 0 |
| | Avg. Error Rate | | | | | 64.71 % | 0.0 % |
| 100,000 over 5mins | Avg. Response Time | | | | | | 11,890 |
| | Successful Responses | | | | | | 8.34 % |
| | Timeout Errors | | | | | | 2456 |
| | 400/500 Errors | | | | | | 0 / 26325 |
| | Avg. Error Rate | | | | | | 53.78 % |

## 7.4 – Chapter Summary

This chapter focused on evaluation for the thesis in multiple ways. Firstly, a functional evaluation of CloudSea prototype was detailed by describing the approach for utilising the prototype. Secondly, a comparative evaluation of the proposed CloudSea architecture was detailed by conducting comparisons with existing systems, both from academic publications and industry tools and then a comprehensive experimental evaluation was detailed, which provided empirical investigation for the proposed Cloud Computing Maturity Model for holistic semantic annotation, including its ‘Cloud-Driven’ Maturity Level which is fundamental to the proposed CloudSea architecture. Results from the empirical investigation are in line with the arguments that led to developing the maturity model for holistic semantic annotation. The next chapter provides a summary of the entire research.

# Chapter 8: Conclusions and Recommendations

This chapter provides a summary for the entire research. This includes an overview of the research activities, contributions to the body of knowledge from the research, the research limitations and recommendations for further research in the domain.

## 8.1 Research Summary

This research was aimed at addressing the challenge of automatic semantic annotation of documents on the web at large with a set of objectives. The introductory chapter provided the fundamental building blocks for the research, in terms of background and motivation; aim and objectives, research hypothesis, the adopted research methodology and the expected contributions to the body of knowledge. With that foundation put in place, the second chapter was focused on the review of literature for the two major paradigms being investigated based on the research aim and objectives; semantic web technologies and cloud computing. The investigation focused on each one of them initially and then on both, in terms of how they leverage each other, with interesting research findings. So, alongside providing a comprehensive state-of-the-art for the two paradigms, the chapter also provided research gaps for semantic web technologies.

Based on the research findings from Chapter 2, the third chapter proposed a holistic perspective to addressing the automation challenge of semantic annotation. This was by means of defining a requirements specification for the holistic perspective. Twelve requirements were proposed with some from previous research efforts in the domain and a few others as novel requirements; lending concepts from other domains in Information technology to provide a multi-disciplinary approach for facilitating the proposed holistic perspective. Furthermore, considering the nature and mechanisms for the holistic perspective, an investigative assessment was conducted into the feasibility of cloud computing facilitating the holistic requirements and hence, perspective. The result of the assessment was a Cloud Computing Capability Model which confirmed that cloud computing could be leveraged for this purpose and detailed different cloud computing mechanisms for each of the requirements.

With the stage set for a “cloud computing” solution to the holistic semantic annotation proposed, Chapter 4 focused on further investigation into application deployment in the cloud and how that will impact on holistic semantic annotation based on its requirements. A set of determinant factors were defined and each one critically evaluated in terms of its potential role towards deploying a holistic semantic annotation solution in the cloud. The focus was to ensure that cloud computing characteristics would be fully maximised for the solution. From the critical analysis and evaluation of the set of determinant factors, several application deployment patterns for holistic semantic annotation in the cloud were defined in terms of how well they can leverage cloud computing characteristics and benefits. These produced a Cloud Computing Maturity Model for holistic semantic annotation and the optimal pattern from the model; "Cloud-Driven" was chosen as a methodological approach towards the design and development of a holistic semantic annotation solution in the cloud.

Chapter 5 focused on the design of a holistic semantic annotation solution. However, it started with a design rationale; in which the "Design Patterns" paradigm of software engineering was adopted. This produced several artefacts, including:

- Microservices-based Design Patterns for Holistic Semantic Annotation
- Pattern Language for Holistic Semantic Annotation, and
- CloudSea: A Holistic, Cloud-driven and Microservices-based Architecture for Automated Semantic Annotation of Web Documents

With a standard architecture in place for holistic semantic annotation, Chapter 6 described a prototype implementation for CloudSea. This included the functional and non-functional requirements, build and deployment on Amazon AWS public cloud. Descriptions for the development and deployment environments were also detailed. In addition, limitations for the prototype were detailed. Furthermore, based on the Cloud Computing Maturity Model developed in Chapter 4, two versions of CloudSea were developed; one using a monolithic software architectural pattern and the other based on a microservices software architectural pattern. These two were each deployed based on three different patterns – making a total of six different deployments of CloudSea.

This provided opportunity for an empirical and experimental evaluation for the Cloud Computing Maturity Model. Each of the six deployments were comprehensively evaluated based on a load test configuration to validate the model and the results obtained were positive. These can be found in Chapter 7. Furthermore, the chapter contains functional and comparative evaluation sections for CloudSea Architecture and prototype. With reference to the research hypothesis in section 1.3, the evaluations; functionality, comparative and experimental validates and confirms the hypothesis as correct; which states that "Cloud Computing can be fully leveraged as a paradigm to address the automation challenge of providing machine-understandable contextual data for semantically annotating documents on the web". This chapter provides conclusions and recommendations for further research directions in the domain.

## 8.2 Research Contributions

This research has produced some contributions to the body of knowledge. These are in domains such as semantic web, cloud computing, software engineering and computing in general. The major contributions to the body of knowledge from this research are detailed in the following sub-sections.

### 8.2.1 Semantic Web and Cloud Computing State-of-the-Art

A comprehensive critical review of existing literature for semantic web and cloud computing; as individual technologies as well as based on their integration. This included a critical review for leveraging cloud computing for the semantic web and vice versa. The review, alongside research findings from the review constitute a contribution to the body of knowledge. These provides a rich repository of knowledge for interested researchers in the area as well as provides a guide towards the gaps identified which can be taken on for research purposes.

### 8.2.2 A Holistic Semantic Annotation Requirements Specification

From the research gaps identified in the literature review, a set of requirements specification were identified and analysed for addressing automated semantic annotation holistically. A holistic perspective was proposed for the automation challenge of semantic annotation and no existing solution was observed to offer a holistic solution at the proposed level. While most of the requirements came from previous automated semantic annotation efforts from research, a few novel requirements were also proposed. These draw from concepts in software engineering and computing generally. The set of requirements together, constitute novel knowledge which can be developed based on methodologies deemed appropriate for delivering a holistic and automated semantic annotation solution for web documents.

### 8.2.3 A Cloud Capability Model for Holistic Semantic Annotation

The Cloud Computing Capability Model for Holistic Semantic Annotation developed in Chapter 3 of this thesis and the approach towards its development constitute a contribution to the body of knowledge. The model provides an assessment approach for the adoption of cloud computing in the facilitation of holistic semantic annotation. Furthermore, it demonstrates the capability of cloud computing mechanisms and how they can drive specific software requirements. While the model is specifically for holistic semantic annotation, the approach towards its development can also be utilised in the assessment of cloud computing capabilities for requirements in other domains within computing.

### 8.2.4 A Cloud Maturity Model for Holistic Semantic Annotation

The Cloud Computing Maturity Model for Holistic Semantic Annotation proposed in Chapter 4 of this thesis is another contribution to the body of knowledge. Based on a set of determinant factors identified from research, the model defines different patterns for a holistic semantic annotation solution deployment in the cloud and how well each pattern leverages cloud computing in terms of its characteristics and benefits. It provides knowledge which specifies the most appropriate pattern for a solution deployment.

Furthermore, it defines a pattern for fully maximising cloud computing benefits for a holistic semantic annotation solution, defined as a "Cloud-Driven" pattern. The development methodology based on the pattern was utilised in this research for the prototype implementation and can be utilised for same purpose or for similar solutions either in the academia or industry. In addition, while the model was developed based on the requirements proposed for holistic semantic annotation earlier in the thesis, it can be adapted for solutions in other areas of computing. In addition, the experimental evaluation data for the model provides empirical evidence for the validation of the model and its suitability for use in the selection of application deployment patterns in the cloud.

### 8.2.5 Design Patterns and Pattern Language for Cloud-Driven, Holistic Semantic Annotation

Design Patterns and their Pattern Language is a method of constructing and sharing knowledge in Software Engineering. This research further produced twelve design patterns and a pattern language for cloud-driven, holistic semantic annotation which are re-usable for the technical solutions they proffer. These stemmed from the development of technical solutions for the holistic semantic annotation requirements proposed in this thesis. Furthermore, they are designed based on microservices software architecture which demonstrates an implementation approach for developing and deploying microservices within a software architectural system.

### 8.2.6 CloudSea: A Holistic, Cloud-driven and Microservices-based Architecture for Automated Semantic Annotation of Web Documents

This research also designed and proposed a holistic, cloud-driven and microservices-based architecture for automated semantic annotation of web documents. The architecture has been designed and proposed based on a rigorous research activity and sound software engineering principles. This implies that while it is up to elite industry standards, it benefits from a thorough research background which provides a basis for its implementation as a full-fledged software. Furthermore, the overall approach to its design constitutes knowledge which can be either adopted or adapted for solutions in other areas of computing. One main output from the architecture already is the prototype implementation which delivers automated semantic annotation as a cloud service by means of an API call within web documents. A Flowchart for process flow within the architecture is also presented.

## 8.3 Research Limitations

While the research aim and objectives have been fulfilled, there are a couple of limitations recognised by the author which can be addressed in future work. They are as follows:

- The prototype is not a full implementation of the CloudSea Architecture. However, it provides the core functionality of the proposed CloudSea Architecture and constitutes a proof-of-concept and MVP (Minimum Viable Product) for the novel solution that CloudSea Architecture proposes.
- The prototype is dependent on a RESTful DBpedia API, which provides access to the DBpedia knowledge graph for fetching contextual data from the knowledge repository. However, a knowledge repository such as DBpedia would always be required for web-scale semantic annotation and there are only a few of such repositories on the web today. Hence, access to web-scale contextual data such as is available with DBpedia, Google Knowledge graph or Linked Open Data Cloud is required in this context.
- Specialist annotators were not involved in the evaluation process of the CloudSea prototype implementation.

## 8.4 Recommendations for Future Research

This research domain has been found to be exciting and one with potential impacts across diverse spheres of life as it has to do with web users making the most of information on the platform for personal, corporate and societal benefits. This also implies that there is abundant further research within the domain. However, three major areas for further research in the area are described as follows:

1. With the scale of the web and size of its users, the list of requirements necessary for a holistic solution that meets the needs of all users (billions of humans worldwide) would require further investigation and would be expected to increase significantly. Some potential determinant factors for requirements analysis might include multilingualism and industry sector. Having said that, it is opined that the emergence of more requirements or the refinement of the proposed ones would be facilitated by the actual implementation and use of the solution for delivery a continuous, consistent and dynamic semantic annotation to web documents automatically.
2. A key principle of Microservices is the decomposition of an application into bounded contexts for the provision of a specific functionality. However, the decomposition can be too coarse or too granular. Further research into an ideal level of granularity for the microservices of the CloudSea architecture is perceived to possess potentials for further leveraging the benefits of the software architectural pattern.
3. Lastly, CloudSea deployment in the cloud was on an orchestration engine platform and the concept of orchestration in computing is a well-established one. Orchestration in computing refers to a process of providing automated co-ordination and functioning of components within computer systems. Another compute technique; known as serverless computing is emerging and defines a different approach to automation techniques for computing components. It leans more towards the "Choreography" concept in computing as opposed to "Orchestration". As the concept of "Serverless Computing" matures, research into its adoption or adaptation for holistic semantic annotation would be a viable one.

# References

Adachi, T. & Fukuta, N., 2017, August. A Mapping-Enhanced Linked Data Inspection and Querying Support System using Dynamic Ontology Matching. In *Proceedings of the International Conference on Web Intelligence* (pp. 1191-1194). ACM.

Aderaldo, C.M., Mendonça, N.C., Pahl, C. & Jamshidi, P., 2017, May. Benchmark Requirements for Microservices Architecture Research. In *Proceedings of the 1st International Workshop on Establishing the Community-Wide Infrastructure for Architecture-Based Software Engineering* (pp. 8-13). IEEE Press.

Agarwal, D. & Jain, S., 2014. Efficient Optimal Algorithm of Task Scheduling in Cloud Computing Environment. *arXiv preprint arXiv:1404.2076*.

Ahmed, O.K. & Kurnaz, A.P.D.S., 2019. Data Access Layer Development for Interoperable GIS Solutions Using NHibernate Mapper.

Aken, J.E.V., 2004. Management Research based on the Paradigm of the Design Sciences: The Quest for Field-Tested and Grounded Technological Rules. *Journal of Management Studies, 41*(2), pp.219-246.

Akgun, A. & Ayvaz, S., 2018, April. An Approach for Information Discovery Using Ontology in Semantic Web Content. In *Proceedings of the 2018 International Conference on Information Science and System* (pp. 250-255). ACM.

Alam, F., Rahman, S.U., Khusro, S. & Ali, S., 2015. Towards a Semantic Web Stack Applicable for Both RDF and Topic Maps: A Survey. *University of Engineering and Technology Taxila. Technical Journal, 20*(2), p.114.

Albukhitan, S., Alnazer, A. & Helmy, T., 2018. Semantic Annotation of Arabic Web Documents using Deep Learning. *Procedia Computer Science, 130*, pp.589-596.

Alec, C., Reynaud-Delaître, C. & Safar, B., 2016, May. An Ontology-Driven Approach for Semantic Annotation of Documents with Specific Concepts. In *European Semantic Web Conference* (pp. 609-624). Springer, Cham.

Alexander, C., 1977. *A Pattern Language: Towns, Buildings, Construction*. Oxford University Press.

Alti, A., Laborie, S., & Roose, P. (2015). Cloud Semantic-Based Dynamic Multimodal Platform for Building mHealth Context-Aware Services. In *Wireless and Mobile Computing, Networking and Communications (WiMob), 2015 IEEE 11th International Conference on* (pp. 357-364). IEEE.

Amato, F., Mazzeo, A., Mazzocca, N. & Romano, S., 2012, July. Close: A Cloud SaaS for Semantic Document Composition. In *2012 Sixth International Conference on Complex, Intelligent, and Software Intensive Systems* (pp. 781-786). IEEE.

Amazon AWS (2018). *Amazon EC2 Auto Scaling*. [image] Available at: https://docs.aws.amazon.com/autoscaling/ec2/userguide/what-is-amazon-ec2-auto-scaling.html [Accessed 12 Aug. 2018].

Ambiah, N. & Lukose, D., 2012, September. Enriching Webpages with Semantic Information. In *International Conference on Dublin Core and Metadata Applications* (pp. 1-11).

Amoretti, M. & Zanichelli, F., 2018. P2P-PL: A Pattern Language to Design Efficient and Robust Peer-to-Peer Systems. *Peer-to-Peer Networking and Applications, 11*(3), pp.518-547.

Arcan, M., Dragoni, M. & Buitelaar, P., 2016, June. ESSOT: An Expert Supporting System for Ontology Translation. In *International Conference on Applications of Natural Language to Information Systems* (pp. 60-73). Springer, Cham.

Arcitura Education (2018). Mechanisms | Arcitura Patterns. [online] Patterns.arcitura.com. Available at: https://patterns.arcitura.com/cloud-computing-patterns/mechanisms [Accessed 14 Feb. 2018].

Aslam, S. & Shah, M.A., 2015, December. Load Balancing Algorithms in Cloud Computing: A Survey of Modern Techniques. In *2015 National Software Engineering Conference (NSEC)* (pp. 30-35). IEEE.

Autili, M., Inverardi, P., Spalazzese, R., Tivoli, M. & Mignosi, F., 2019. Automated Synthesis of Application-Layer Connectors from Automata-Based Specifications. *Journal of Computer and System Sciences.*

Avgeriou, P., Vogiatzis, D., Tzanavari, A. & Retalis, S., 2004. Towards a Pattern Language for Adaptive Web-based Educational Systems. *Advanced Technology for Learning Journal, 1*(4), pp.202-209.

Avi Networks (2019). *Container Orchestration.* [image] Available at: https://avinetworks.com/wp-content/uploads/2018/12/container-orchestration-diagram-1.png [Accessed 10 Mar. 2019].

Avraam, I., Bratsas, C., Antoniou, I. & Bamidis, P., 2014. DBpedia Spotlight Internationalization using the DBpedia in Greek as a Case Study – Realizing the Greek DBpedia Spotlight. *Information and Communication Technology for Education* (2 Volume Set), 58, p.69.

AWS Marketplace (2019). AWS Marketplace: SendGrid Email Delivery Service. [online] Aws.amazon.com. Available at: https://aws.amazon.com/marketplace/pp/B074CQY6KB [Accessed 16 Apr. 2019].

Bajaj, G., Agarwal, R., Singh, P., Georgantas, N. & Issarny, V., 2017. A Study of Existing Ontologies in the IoT-domain. *arXiv preprint arXiv:1707.00112*.

Bakshi, K., 2017, March. Microservices-based Software Architecture & Approaches. In *2017 IEEE Aerospace Conference* (pp. 1-8). IEEE.

Balalaie, A., Heydarnoori, A. & Jamshidi, P., 2016. Microservices Architecture Enables Devops: Migration to a Cloud-Native Architecture. *IEEE Software, 33*(3), pp.42-52.

Bangare, S.L., Borse, S., Bangare, P.S. & Nandedkar, S., 2012. Automated API Testing Approach. *International Journal of Engineering Science and Technology, 4*(2).

Baskarada, S., Nguyen, V. & Koronios, A., 2018. Architecting Microservices: Practical Opportunities and Challenges. *Journal of Computer Information Systems*, pp.1-9.

Basu, A., 2019. Semantic Web, Ontology, and Linked Data. In *Web Services: Concepts, Methodologies, Tools, and Applications* (pp. 127-148). IGI Global.

Batista-Navarro, R., Carter, J. & Ananiadou, S., 2016. Argo: Enabling the Development of Bespoke Workflows and Services for Disease Annotation. Database, 2016.

Bayoudhi, L., Sassi, N. & Jaziri, W., 2017. A Hybrid Storage Strategy to Manage the Evolution of an OWL 2 DL Domain Ontology. *Procedia Computer Science, 112*, pp.574-583.

Beedle, M., Devos, M., Sharon, Y., Schwaber, K. & Sutherland, J., 1999. SCRUM: An Extension Pattern Language for Hyper Productive Software Development. *Pattern Languages of Program Design, 4*, pp.637-651.

Belloze, K.T., Monteiro, D.I.S., Lima, T.F., Silva Jr, F.P. & Cavalcanti, M.C., 2012. An Evaluation of Annotation Tools for Biomedical Texts. In *Ontology Research in Brazil and International Workshop on Metamodels, Ontologies and Semantic Technologies* (pp. 108-119).

Ben-Yehuda, S., 1997. Pattern Language for Framework Construction. In *PLoP-97 Writer's Workshop* (pp. 97-34).

Bogner, J., Zimmermann, A. & Wagner, S., 2018. Analyzing the Relevance of SOA Patterns for Microservice-Based Systems. ZEUS, 9, pp.9-16.

Boneh, D., Di Crescenzo, G., Ostrovsky, R., & Persiano, G. (2004). Public Key Encryption with Keyword Search. In *International Conference on the Theory and Applications of Cryptographic Techniques* (pp. 506-522). Springer, Berlin, Heidelberg.

Bourgonje, P., Moreno-Schneider, J., Nehring, J., Rehm, G., Sasaki, F. & Srivastava, A., 2016, May. Towards a Platform for Curation Technologies: Enriching Text Collections with a Semantic-Web Layer. In *European Semantic Web Conference* (pp. 65-68). Springer, Cham.

Braga, R.T.V., Germano, F.S. & Masiero, P.C., 1999, August. A Pattern Language for Business Resource Management. In *Proceedings of PLOP* (Vol. 7, pp. 1-33).

Brank, J., Leban, G. & Grobelnik, M., 2018. Semantic Annotation of Documents Based on Wikipedia Concepts. Informatica, 42(1).

Bratt, S., 2008, May. Fast forward get ready for Web 3.0. In *Conferencia de World Wide Web Consortium* (pp. 4-44).

Brezillon, P., 2003. Individual and Team Contexts in a Design Process (CLNSS02). In *Proceedings of the Annual Hawaii International Conference on System Sciences* (pp. 27-27).

Buscaldi, D., Gangemi, A. & Recupero, D.R., 2018. *Semantic Web Challenges*. Springer International Publishing.

Byun, Y., Sanders, B. & Chung, K., 2002, September. A Pattern Language for Communication Protocols. In *Proceedings of the 9th Conference on Pattern Languages of Programs*, Monticello, Illinois.

Cao, D., Liu, P., Cui, W., Zhong, Y. & An, B., 2016. Cluster as a Service: A Resource Sharing Approach for Private Cloud. *Tsinghua Science and Technology, 21*(6), pp.610-619.

Carneiro Jr, C. & Schmelmer, T., 2016. Microservices From Day One: Build Robust and Scalable Software from the Start. *Apress. Berkeley, CA*.

Carnell, J., 2017. *Spring Microservices in Action*. Manning Publications Co.

Carroll, M., Kotze, P. & Van der Merwe, A., 2012. A Risk and Control Framework for Cloud Computing and Virtualisation: Enabling Technologies. *CSIR Science Scope, 6*(2), pp.62-63.

CBROnline. (2019). What is Apache? - Computer Business Review. [online] Available at: https://www.cbronline.com/what-is/what-is-apache-4909894/ [Accessed 26 Jun. 2019].

Cerny, T., Donahoo, M.J. & Pechanec, J., 2017, September. Disambiguation and Comparison of SOA, Microservices and Self-Contained Systems. In *Proceedings of the International Conference on Research in Adaptive and Convergent Systems* (pp. 228-235). ACM.

Chandra, G.S., 2016. Pattern Language for IoT applications. In *PLoP Conference*, USA.

Cheatham, M., Lambert, J. & Vardeman II, C.F., 2019. Leveraging Wikipedia for Ontology Pattern Population from Text. In *AAAI Spring Symposium: Combining Machine Learning with Knowledge Engineering.*

Chernyshov, A., Balandina, A., Kostkina, A., & Klimov, V. (2016). Intelligence Search Engine and Automatic Integration System for Web-Services and Cloud-Based Data Providers Based on Semantics. *Procedia Computer Science, 88*, 272-276.

Chiregi, M. & Navimipour, N.J., 2017. Cloud Computing and Trust Evaluation: A Systematic Literature Review of the State-of-the-art Mechanisms. *Journal of Electrical Systems and Information Technology*.

Ciccarese, P., Ocana, M. & Clark, T., 2012. DOMEO: A Web-Based Tool for Semantic Annotation of Online Documents. *Bio-Ontologies* 2011.

CNCF.io. (2018). Cloud Native Technologies Are Scaling Production Applications - Cloud Native Computing Foundation. [online] Cloud Native Computing Foundation. Available at: https://www.cncf.io/blog/2017/12/06/cloud-native-technologies-scaling-production-applications/ [Accessed 7 Feb. 2019].

Cohen, L. & Manion, L., 1994. Research Methods in Education (4th Edition) (London, Croom Helm Ltd). *Institutionalismus in der Erziehungswissenschaft*, pp.326-346.

Coronado, M., Iglesias, C.A. & Serrano, E., 2015. Modelling Rules for Automating the Evented Web by Semantic Technologies. *Expert Systems with Applications, 42*(21), pp.7979-7990.

Corradi, A., Destro, M., Foschini, L., Kotoulas, S., Lopez, V. & Montanari, R., 2016. Mobile Cloud Support for Semantic-Enriched Speech Recognition in Social Care. *IEEE Transactions on Cloud Computing*

Cortazar, G. O., Zapater, J. J. S., & Sánchez, F. G. (2012). Adding Semantics to Cloud Computing to Enhance Service Discovery and Access. In *Proceedings of the 6th Euro American Conference on Telematics and Information Systems* (pp. 231-236). ACM.

Cui, W., Zhan, H., Li, B., Wang, H. & Cao, D., 2016, March. Cluster as a Service: A Container Based Cluster Sharing Approach with Multi-User Support. In *2016 IEEE Symposium on Service-Oriented System Engineering (SOSE)* (pp. 111-118). IEEE.

D'silva, G.M., Khan, A. & Bari, S., 2017, May. Real-time Processing of IoT Events with Historic Data using Apache Kafka and Apache Spark with Dashing Framework. In *2017 2nd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT)* (pp. 1804-1809). IEEE.

Da Silva, M.A.A. & Cavalcanti, M.C., 2014. Combining Ontology Modules for Scientific Text Annotation. *Journal of Information and Data Management, 5*(3), pp.238-251.

Dammak, S.M., Jedidi, A. & Bouaziz, R., 2013, December. Automation and Evaluation of the Semantic Annotation of Web Resources. In *8th International Conference for Internet Technology and Secured Transactions (ICITST-2013)* (pp. 443-448). IEEE.

Dan, N., Hua-Ji, S., Yuan, C. & Jia-Hu, G., 2012, August. Attribute Based Access Control (ABAC)-Based Cross-Domain Access Control in Service-Oriented Architecture (SOA). In *2012 International Conference on Computer Science and Service System* (pp. 1405-1408). IEEE.

Dautov, R., Kourtesis, D., Paraskakis, I., & Stannett, M. (2013). Addressing Self-Management in Cloud Platforms: A Semantic Sensor Web Approach. In *Proceedings of the 2013 international workshop on Hot topics in cloud services* (pp. 11-18). ACM.

Davis, B., Varma, P., Handschuh, S., Dragan, L. & Cunningham, H., 2009, May. Controlled Natural Language for Semantic Annotation. In *European Semantic Web Conference* (pp. 816-820). Springer, Berlin, Heidelberg.

Daya, S., Van Duy, N., Eati, K., Ferreira, C.M., Glozic, D., Gucer, V., Gupta, M., Joshi, S., Lampkin, V., Martins, M. & Narain, S., 2016. Microservices from Theory to Practice: Creating Applications in *IBM Bluemix Using the Microservices Approach. IBM Redbooks*.

De Virgilio, R., 2017, May. Smart RDF Data Storage in Graph Databases. In *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)* (pp. 872-881). IEEE.

Delessy, N., Fernandez, E.B. & Larrondo-Petrie, M.M., 2007, March. A Pattern Language for Identity Management. In *2007 International Multi-Conference on Computing in the Global Information Technology (ICCGI'07)* (pp. 31-31). IEEE.

Dessi, N., Milia, G., Pascariello, E. & Pes, B., 2016. COWB: A Cloud-Based Framework Supporting Collaborative Knowledge Management within Biomedical Communities. *Future Generation Computer Systems, 54*, pp.399-408.

Dhingra, V. & Bhatia, K., 2012. Comparative Analysis of Ontology Ranking Algorithms. *International Journal of Information Technology and Web Engineering (IJITWE), 7*(3), pp.55-66.

Dhingra, M., Lakshmi, J. & Nandy, S.K., 2012, September. Resource Usage Monitoring in Clouds. In *2012 ACM/IEEE 13th International Conference on Grid Computing* (pp. 184-191). IEEE.

Di Francesco, P., Malavolta, I. & Lago, P., 2017, April. Research on Architecting Microservices: Trends, Focus, and Potential for Industrial Adoption. In *2017 IEEE International Conference on Software Architecture (ICSA)* (pp. 21-30). IEEE.

Di Martino, B., & Esposito, A. (2016). Semantic Techniques for Multi-Cloud Applications Portability and Interoperability. *Procedia Computer Science, 97*, 104-113.

Diaby, T. & Rad, B.B., 2017. Cloud Computing: A Review of the Concepts and Deployment Models. *International Journal of Information Technology and Computer Science, 9*(6), pp.50-58.

Dill, S., Eiron, N., Gibson, D., Gruhl, D., Guha, R., Jhingran, A., Kanungo, T., Rajagopalan, S., Tomkins, A., Tomlin, J.A. & Zien, J.Y., 2003, May. SemTag and Seeker: Bootstrapping the Semantic Web via Automated Semantic Annotation. In *Proceedings of the 12th International Conference on World Wide Web* (pp. 178-186). ACM.

Dooley, D.M., Griffiths, E.J., Gosal, G.S., Buttigieg, P.L., Hoehndorf, R., Lange, M.C., Schriml, L.M., Brinkman, F.S. and Hsiao, W.W., 2018. FoodOn: A Harmonized Food Ontology to Increase Global Food Traceability, Quality Control and Data Integration. *Nature Paper Journals Science of Food, 2*(1), p.23.

Dou, D., Wang, H. & Liu, H., 2015, February. Semantic Data Mining: A Survey of Ontology-Based Approaches. In *Proceedings of the 2015 IEEE 9th International Conference on Semantic Computing (IEEE ICSC 2015)* (pp. 244-251). IEEE.

Dragoni, N., Lanese, I., Larsen, S.T., Mazzara, M., Mustafin, R. & Safina, L., 2017, June. Microservices: How to make your Application Scale. In *International Andrei Ershov Memorial Conference on Perspectives of System Informatics* (pp. 95-104). Springer, Cham.

Drechsler, Andreas, & Alan Hevner. "A Four-Cycle Model of IS Design Science Research: Capturing the Dynamic Nature of IS Artefact Design." In *Breakthroughs and Emerging Insights from Ongoing Design Science Projects: Research-in-progress papers and poster presentations from the 11th International Conference on Design Science Research in Information Systems and Technology (DESRIST) 2016. St. John, Canada, 23-25 May.* DESRIST 2016, 2016.

Duma, M., 2011, September. RDFa Editor for Ontological Annotation. In *Proceedings of the Second Student Research Workshop associated with RANLP 2011* (pp. 54-59).

E Silva, F.J.D.S., Kon, F., Yoder, J. & Johnson, R., 2005. A Pattern Language for Adaptive Distributed Systems. In *Proceedings of the 5th Latin American Conference on Pattern Languages of Programming* (pp. 19-48).

Eder, M., 2016. Hypervisor vs. Container-based Virtualization. *Future Internet (FI) and Innovative Internet Technologies and Mobile Communications (IITM), 1.*

Edwin, N.M., 2014. Software Frameworks, Architectural and Design Patterns. *Journal of Software Engineering and Applications, 7*(08), p.670.

El-Ghobashy, A.N., Attiya, G.M. & Kelash, H.M., 2014. A Proposed Framework for Arabic Semantic Annotation Tool. *International Journal of Computing and Digital Systems, 218*(1223), pp.1-7.

Elastic.co. (2019). Elasticsearch: RESTful, Distributed Search & Analytics | Elastic. [online] Available at: https://www.elastic.co/products/elasticsearch [Accessed 13 Mar. 2019].

Elastic.co. (2019). Kibana: Explore, Visualize, Discover Data | Elastic. [online] Available at: https://www.elastic.co/products/kibana [Accessed 13 Mar. 2019].

Elastic.co. (2019). Logstash: Collect, Parse, Transform Logs | Elastic. [online] Available at: https://www.elastic.co/products/logstash [Accessed 13 Mar. 2019].

Elastic.co. (2019). Open Source Search & Analytics · Elasticsearch | Elastic. [online] Available at: https://www.elastic.co/ [Accessed 13 Mar. 2019].

Eloranta, V.P., Koskinen, J., Leppänen, M. & Reijonen, V., 2010. A Pattern Language for Distributed Machine Control Systems. Tampere University of Technology, Department of Software Systems.

Erdmann, M., Maedche, A., Schnurr, H.P. & Staab, S., 2000, August. From Manual to Semi-Automatic Semantic Annotation: About Ontology-Based Text Annotation Tools. In *Proceedings of the COLING-2000 Workshop on Semantic Annotation and Intelligent Content* (pp. 79-85). Association for Computational Linguistics.

Erl, T., Cope, R. & Naserpour, A., 2015. *Cloud Computing Design Patterns*. Prentice Hall Press.

Erl, T., Puttini, R. & Mahmood, Z., 2013. *Cloud Computing: Concepts, Technology & Architecture*. Pearson Education.

Escobar, D., Cárdenas, D., Amarillo, R., Castro, E., Garcés, K., Parra, C. & Casallas, R., 2016, October. Towards the Understanding and Evolution of Monolithic Applications as Microservices. In *2016 XLII Latin American Computing Conference (CLEI)* (pp. 1-11). IEEE.

Espinoza, R. & Melgar, A., 2015, November. An Automated Semantic Annotation Tool Supported by an Ontology in the Computer Science Domain. In *International Conference on Knowledge Engineering and Ontology Development* (pp. 133-138).

Evitts, P. & Hinchcliffe, D., 2000. *A UML Pattern Language*. Indianapolis, Ind.: Macmillan Technical.

Fang, D., Liu, X., Romdhani, I., Jamshidi, P. & Pahl, C., 2016. An Agility-Oriented and Fuzziness-Embedded Semantic Model for Collaborative Cloud Service Search, Retrieval and Recommendation. *Future Generation Computer Systems, 56*, pp.11-26.

Faria, C., Serra, I. & Girardi, R., 2014. A Domain-Independent Process for Automatic Ontology Population from Text. *Science of Computer Programming, 95*, pp.26-43.

Fazio, M., Celesti, A., Ranjan, R., Liu, C., Chen, L. & Villari, M., 2016. Open Issues in Scheduling Microservices in the Cloud. *IEEE Cloud Computing, 3*(5), pp.81-88.

Fehling, C., Leymann, F., Retter, R., Schupeck, W. & Arbitter, P., 2014. *Cloud Computing Patterns: Fundamentals to Design, Build, and Manage Cloud Applications*. Springer Science & Business Media.

Fehling, C., Leymann, F., Rütschlin, J. & Schumm, D., 2012. Pattern-Based Development and Management of Cloud Applications. *Future Internet, 4*(1), pp.110-141.

Fensel, D. ed., 2011. Foundations for the Web of Information and Services: A Review of 20 years of Semantic Web Research. *Springer Science & Business Media*.

Fernandez, E.B. & Pan, R., 2001, September. A Pattern Language for Security Models. In *Proc. of PLoP* (Vol. 1).

Fernandez, E.B. & Sorgente, T., 2005, August. A Pattern Language for Secure Operating System Architectures. In *Procs. of the 5th Latin American Conference on Pattern Languages of Programs* (pp. 16-19).

Fiorelli, M., Pazienza, M.T. & Stellato, A., 2015. A Flexible Approach to Semantic Annotation Systems for Web Content. *Intelligent Systems in Accounting, Finance and Management, 22*(1), pp.65-79.

Flahive, A., Taniar, D., Rahayu, W. & Apduhan, B.O., 2015. A Methodology for Ontology Update in the Semantic Grid Environment. *Concurrency and Computation: Practice and Experience, 27*(4), pp.782-808.

Florio, L. & Di Nitto, E., 2016, July. Gru: An Approach to Introduce Decentralised Autonomic Behaviour in Microservices Architectures. In *2016 IEEE International Conference on Autonomic Computing (ICAC)* (pp. 357-362). IEEE.

Fowler, M., 2015. Microservice Trade-offs. From: https://martinfowler. com/articles/microservice-trade-offs. html.

Fox, C., Grimm, R. & Caldeira, R., 2016. *An Introduction to Evaluation*. Sage.

Franchitti, J. (2019). Software Engineering. [online] New York University. Available at: http://www.nyu.edu/classes/jcf/g22.2440-001_sp06/slides/session8/g22_2440_001_c82.pdf [Accessed 24 Jan. 2019].

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. 2000. Design Patterns: Abstraction and Reuse of Object-Oriented Design. *ECOOP 1993: ECOOP' 93 — Object-Oriented Programming* pp 406-431

Gao, G., Liu, Y.S., Lin, P., Wang, M., Gu, M. & Yong, J.H., 2017. BIMTag: Concept-Based Automatic Semantic Annotation of Online BIM Product Resources. *Advanced Engineering Informatics, 31*, pp.48-61.

Garg, S. and Garg, S., 2019, April. Automated Cloud Infrastructure, Continuous Integration and Continuous Delivery using Docker with Robust Container Security. In *2019 IEEE Conference on Multimedia Information Processing and Retrieval (MIPR)* (pp. 467-470). IEEE.

Gawich, M., Badr, A., Hegazy, A. & Ismael, H., 2012. A Survey on Semantic Annotation Tools. *Egyptian Computer Science Journal, ECS, 36*(2).

Getting, B., 2007. Basic Definitions: Web 1.0, Web. 2.0, Web 3.0. *Practical eCommerce: Insights for Online Merchants–2007. URL: http://www. practicalecommerce. com/articles/464-Basic-Definitions-Web-1-0-Web-2-0-Web-3-0.*

Gezer, V. & Bergweiler, S., 2016. Service and Workflow Engineering based on Semantic Web Technologies. In *Tenth International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies (UBICOMM 2016), International Academy, Research, and Industry Association (IARIA). IARIA* (Vol. 10, pp. 152-157).

Ghomi, E.J., Rahmani, A.M. & Qader, N.N., 2017. Load-Balancing Algorithms in Cloud Computing: A Survey. *Journal of Network and Computer Applications, 88*, pp.50-71.

Giannopoulos, G., Bikakis, N., Dalamagas, T. & Sellis, T., 2010, May. GoNTogle: A Tool for Semantic Annotation and Search. In *Extended Semantic Web Conference* (pp. 376-380). Springer, Berlin, Heidelberg.

Gilbert, J. (2018). *Cloud Native Development Patterns and Best Practices*. Birmingham: Packt.

Giri, K., 2011. Role of Ontology in Semantic Web. *DESIDOC Journal of Library & Information Technology, 31*(2).

Gkortzis, A., Rizou, S. & Spinellis, D., 2016, December. An Empirical Analysis of Vulnerabilities in Virtualization Technologies. In *2016 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)* (pp. 533-538). IEEE.

Goodyear, P. & Retalis, S., 2010. Technology-Enhanced Learning. *Rotterdam: Sense Publishers.*

Gracia, J., Montiel-Ponsoda, E., Cimiano, P., Gómez-Pérez, A., Buitelaar, P. & McCrae, J., 2012. Challenges for the Multilingual Web of Data. *Web Semantics: Science, Services and Agents on the World Wide Web, 11*, pp.63-71.

Grafana Labs. (2019). Grafana - The Open Platform for Analytics and Monitoring. [online] Available at: https://grafana.com/ [Accessed 13 Jun. 2019].

Grcar, M. & Mladenic, D., 2009, September. Visual OntoBridge: Semi-Automatic Semantic Annotation Software. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases* (pp. 726-729). Springer, Berlin, Heidelberg.

Gregor, S. & Hevner, A.R., 2013. Positioning and Presenting Design Science Research for Maximum Impact. *MIS quarterly*, pp.337-355.

Grobe-Bolting, G., Nishioka, C. & Scherp, A., 2015, October. A Comparison of Different Strategies for Automated Semantic Document Annotation. In *Proceedings of the 8th International Conference on Knowledge Capture* (p. 8). ACM.

Gruber, T., 2007. Ontology of Folksonomy: A Mash-Up of Apples and Oranges. *International Journal on Semantic Web and Information Systems (IJSWIS), 3*(1), pp.1-11.

Guarino, N. & Musen, M.A., 2015. Ten Years of Applied Ontology. *Applied Ontology, 10*(3-4), pp.169-170.

Guerra, E.M., De Souza, J.T. & Fernandes, C.T., 2009, August. A Pattern Language for Metadata-based Frameworks. In *Proceedings of the 16th Conference on Pattern Languages of Programs* (p. 3). ACM.

Guha, R.V., Brickley, D. & Macbeth, S., 2016. Schema.org: Evolution of Structured Data on the Web. *Communications of the ACM, 59*(2), pp.44-51.

Gunelius, S. (2014). *The Data Explosion in 2014 Minute by Minute – Infographic*. [online] ACI. Available at: https://aci.info/2014/07/12/the-data-explosion-in-2014-minute-by-minute-infographic/ [Accessed 13 Aug. 2018].

Guo, D., Wang, W., Zeng, G. & Wei, Z., 2016, March. Microservices Architecture Based Cloudware Deployment Platform for Service Computing. In *2016 IEEE Symposium on Service-Oriented System Engineering (SOSE)* (pp. 358-363). IEEE.

Gutierrez, Y., Tomas, D. & Moreno, I., 2019. Developing an Ontology Schema for Enriching and Linking Digital Media Assets. *Future Generation Computer Systems*.

Guttula, C., 2012. *RADIANTWEB: A Tool Facilitating Semantic Annotation of Web Services* (Doctoral dissertation, University of Georgia).

Haimes, P., Medley, S. & Baba, T., 2016. A Pattern Language: Designing a Hazard Information Map Interface for Community-Based Users. *International Journal of Asia Digital Art and Design Association, 20*(1), pp.5-13.

Halford, S., Pope, C. & Weal, M., 2013. Digital Futures? Sociological Challenges and Opportunities in the Emergent Semantic Web. *Sociology, 47*(1), pp.173-189.

Hall, W. & Tiropanis, T., 2012. Web Evolution and Web Science. *Computer Networks, 56*(18), pp.3859-3865.

Hamza, H. & Fayad, M.E., 2002, September. A Pattern Language for Building Stable Analysis Patterns. In *Proceedings of PLoP*.

Handy, A. (2019). Docker Fully Embraces Kubernetes - The New Stack. [online] The New Stack. Available at: https://thenewstack.io/docker-fully-embraces-kubernetes/ [Accessed 26 Jun. 2019].

Hanid, M.B., 2014. *Design Science Research as an Approach to Develop Conceptual Solutions for Improving Cost Management in Construction* (Doctoral dissertation, University of Salford).

Hao, Z., Novak, E., Yi, S. & Li, Q., 2017. Challenges and Software Architecture for Fog Computing. *IEEE Internet Computing, 21*(2), pp.44-53.

Harrow, I., Balakrishnan, R., Jimenez-Ruiz, E., Jupp, S., Lomax, J., Reed, J., Romacker, M., Senger, C., Splendiani, A., Wilson, J. & Woollard, P., 2019. Ontology Mapping for Semantically Enabled Applications. *Drug Discovery Today*.

Harrer, S., Lenhard, J., Kopp, O., Ferme, V. & Pautasso, C., 2017, July. A Pattern Language for Workflow Engine Conformance and Performance Benchmarking. In *Proceedings of the 22nd European Conference on Pattern Languages of Programs* (p. 1). ACM.

He, Q., Berg, A., Li, Y., Vallejos, C.E. and Wu, R., 2010. Mapping Genes for Plant Structure, Development and Evolution: Functional Mapping meets Ontology. *Trends in Genetics, 26*(1), pp.39-46.

Hendre, A., & Joshi, K. P. (2015). A Semantic Approach to Cloud Security and Compliance. In *Cloud Computing (CLOUD), 2015 IEEE 8th International Conference on* (pp. 1081-1084). IEEE.

Hentrich, C. & Zdun, U., 2009. A Pattern Language for Process Execution and Integration Design in Service-Oriented Architectures. In *Transactions on Pattern Languages of Programming I* (pp. 136-191). Springer, Berlin, Heidelberg.

Herrera, N.P., Gomez, F.L., Bucheli, V.A. & Pabón, O.S., 2017. Semantic Annotation and Retrieval of Scientific Documents in a Big Data Environment.

Hevner, A.R., 2007. A Three-Cycle View of Design Science Research. *Scandinavian Journal of Information Systems, 19*(2), p.4.

Hevner, A. & Chatterjee, S., 2010. *Design Research in Information Systems: Theory and Practice (Vol. 22)*. Springer Science & Business Media.

Hnatkowska, B. & Kasprzyk, K., 2009, October. Integration of Application Business Logic and Business Rules with DSL and AOP. In *IFIP Central and East European Conference on Software Engineering Techniques* (pp. 30-39). Springer, Berlin, Heidelberg.

Hooi, Y.K., Hassan, M.F. & Shariff, A.M., 2015, May. Ontology Evaluation – A Criteria Selection Framework. In *2015 International Symposium on Mathematical Sciences and Computing Research (iSMSC)* (pp. 298-303). IEEE.

Hoque, S., de Brito, M.S., Willner, A., Keil, O. & Magedanz, T., 2017, July. Towards Container Orchestration in Fog Computing Infrastructures. In *2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)* (Vol. 2, pp. 294-299). IEEE.

HowToExpert (2019). [Image] Available at: http://www.howto-expert.com/wp-content/uploads/overview.png [Accessed 11 May. 2019].

Hsu, I.C. & Cheng, F.Q., 2015. SAaaS: A Cloud Computing Service Model using Semantic-Based Agent. *Expert Systems, 32*(1), pp.77-93.

Hu, J., Jia, S. & Wu, K., 2015. Semantic-Based Requirements Content Management for Cloud Software. *Mathematical Problems in Engineering, 2015*.

Hua, Y., Liu, X., & Jiang, H. (2014). Antelope: A Semantic-Aware Data Cube Scheme for Cloud Data Centre Networks. *IEEE Transactions on Computers, 63*(9), 2146-2159.

Hukerikar, S. & Engelmann, C., 2017, July. A Pattern Language for High-Performance Computing Resilience. In *Proceedings of the 22nd European Conference on Pattern Languages of Programs* (p. 12). ACM.

Husain, M., McGlothlin, J., Masud, M.M., Khan, L. & Thuraisingham, B.M., 2011. Heuristics-Based Query Processing for Large RDF Graphs using Cloud Computing. *IEEE Transactions on Knowledge and Data Engineering, 23*(9), pp.1312-1327.

Huttermann, M. (2012). *DevOps for Developers*, Apress.

IBM Cloud Education (2019). Kubernetes: A Complete Guide. [online] Ibm.com. Available at: https://www.ibm.com/cloud/learn/kubernetes [Accessed 27 June 2019].

Iivari, J., 2010. Twelve Theses on Design Science Research in Information Systems. In *Design Research in Information Systems* (pp. 43-62). Springer, Boston, MA.

Imam, F.T., 2016. Application of Ontologies in Cloud Computing: The State-Of-The-Art. *arXiv preprint arXiv:1610.02333*.

Indu, I., Anand, P.R. & Bhaskar, V., 2018. Identity and Access Management in Cloud Environment: Mechanisms and Challenges. *Engineering Science and Technology, An International Journal, 21*(4), pp.574-588.

Infoworld (2019). What is Docker? [image] Available at: https://www.infoworld.com/article/3204171/what-is-docker-docker-containers-explained.html [Accessed 21 May 2019].

Iqbal, R., Murad, M.A.A., Mustapha, A. & Sharef, N.M., 2013. An Analysis of Ontology Engineering Methodologies: A Literature Review. *Research Journal of Applied Sciences, Engineering and Technology, 6*(16), pp.2993-3000.

Iwashita, M. & Tanimoto, S., 2013, July. Consideration of Billing Management Method for Cloud Computing Services. In *2013 14th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing* (pp. 394-399). IEEE.

Jacob, P.M. & Mani, P., 2018. Software Architecture Pattern Selection Model for Internet of Things Based Systems. *IET Software, 12*(5), pp.390-396.

Jain, N. & Choudhary, S., 2016, March. Overview of Virtualisation in Cloud Computing. In *2016 Symposium on Colossal Data Analysis and Networking (CDAN)* (pp. 1-4). IEEE.

Jaramillo, D., Nguyen, D.V. & Smart, R., 2016, March. Leveraging Microservices Architecture by using Docker Technology. In *SoutheastCon 2016* (pp. 1-5). IEEE.

Jean-Mary, Y.R., Shironoshita, E.P. & Kabuka, M.R., 2009. Ontology Matching with Semantic Verification. *Journal of Web Semantics, 7*(3), pp.235-251.

Jedlitschka, A., Ciolkowski, M. & Pfahl, D., 2008. Reporting Experiments in Software Engineering. In *Guide to Advanced Empirical Software Engineering* (pp. 201-228). Springer, London.

Jiang, J., Lu, J., Zhang, G. & Long, G., 2013, May. Optimal Cloud Resource Auto-Scaling for Web Applications. In *2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing* (pp. 58-65). IEEE.

Jie, L., Jianfeng, C., Zhihong, R., Zhi, S., Hui, Y. & Rui, X., 2018, April. A Massive RDF Storage Approach based on Graph Database. In *Proceedings of the International Conference on Geoinformatics and Data Analysis* (pp. 169-173). ACM.

Johnson, J.R., Miller, A., Khan, L. & Thuraisingham, B., 2012, June. Extracting Semantic Information Structures from Free Text Law Enforcement Data. In *2012 IEEE International Conference on Intelligence and Security Informatics* (pp. 177-179). IEEE.

Jonathan, C. & Mokbel, M.F., 2017. A Demonstration of Stella: A Crowdsourcing-Based Geotagging Framework. *Proceedings of the VLDB Endowment, 10*(12), pp.1969-1972.

Joshi, M., Mittal, S., Joshi, K.P. & Finin, T., 2017, June. Semantically Rich, Oblivious Access Control using ABAC for Secure Cloud Storage. In *2017 IEEE International Conference on Edge Computing (EDGE)* (pp. 142-149). IEEE.

Jula, A., Sundararajan, E. & Othman, Z., 2014. Cloud Computing Service Composition: A Systematic Literature Review. *Expert Systems with Applications, 41*(8), pp.3809-3824.

Jung, M., Mallering, S., Dalbhanjan, P., Chapman, P. & Kassen, C., 2016. Microservices on AWS. Amazon Web Services, Inc., New York, NY, USA, Tech. Rep.

Kakadia, D., Saripalli, P. & Varma, V., 2013, July. MECCA: Mobile, Efficient Cloud Computing Workload Adoption Framework using Scheduler Customization and Workload Migration Decisions. In *Proceedings of the first international workshop on Mobile cloud computing & networking* (pp. 41-46). ACM.

Kang, H., Le, M. & Tao, S., 2016, April. Container and Microservice Driven Design for Cloud Infrastructure DevOps. In *2016 IEEE International Conference on Cloud Engineering (IC2E)* (pp. 202-211). IEEE.

Keller, W. & Coldewey, J., 1996, September. Relational Database Access Layers: A Pattern Language. In *Proceedings PLoP '96*, Allerton Park.

Kendall, E.A., Pathak, C.V., Krishna, P.M. & Suresh, C.B., 1997, September. The Layered Agent Pattern Language. In *Proceedings of the Conference on Pattern Languages of Programs*.

Khalili, A., Loizou, A. & van Harmelen, F., 2016, May. Adaptive Linked Data-driven Web Components: Building Flexible and Reusable Semantic Web Interfaces. In *European Semantic Web Conference* (pp. 677-692). Springer, Cham.

Khan, A., 2017. Key Characteristics of a Container Orchestration Platform to Enable a Modern Application. *IEEE Cloud Computing, 4*(5), pp.42-48.

Kim, P., Ng, C.K. & Lim, G., 2010. When Cloud Computing meets with Semantic Web: A New Design for E-Portfolio Systems in the Social Media Era. *British Journal of Educational Technology, 41*(6), pp.1018-1028.

Kirrane, S., Villata, S. & d'Aquin, M., 2018. Privacy, Security and Policies: A Review of Problems and Solutions with Semantic Web Technologies. *Semantic Web, 9*(2), pp.153-161.

Kiryakov, A., Popov, B., Terziev, I., Manov, D. & Ognyanoff, D., 2004. Semantic Annotation, Indexing, and Retrieval. *Web Semantics: Science, Services and Agents on the World Wide Web, 2*(1), pp.49-79.

Khurana, D., Koli, A., Khatter, K. & Singh, S., 2017. Natural Language Processing: State-of-the-art, Current Trends and Challenges. *arXiv preprint arXiv:1708.05148*.

Kodituwakku, S.R., Bertok, P. & Zhao, L., 2001. APLRAC: A Pattern Language for Designing and Implementing Role-Based Access Control. In *EuroPLoP* (Vol. 1, p. 2001).

Kollmann, T., Lomberg, C. & Peschl, A., 2016. Web 1.0, Web 2.0, and Web 3.0: The Development of E-Business. In *Encyclopedia of E-Commerce Development, Implementation, and Management* (pp. 1139-1148). IGI Global.

Kubernetes.io. (2019). Production-Grade Container Orchestration. [online] Available at: https://kubernetes.io/ [Accessed 10 Feb. 2019].

Kuechler, W. & Vaishnavi, V., 2012. A Framework for Theory Development in Design Science Research: Multiple Perspectives. *Journal of the Association for Information systems, 13*(6), p.395.

Kulesza, R., Lima, M., Araujo, C. & de Sousa, M.F., 2018, October. Evolution of Software Architectures: From Web 1.0 to Web 3.0 Systems. In *Proceedings of the 24th Brazilian Symposium on Multimedia and the Web* (pp. 11-13). ACM.

Kumar, S.K. & Harding, J.A., 2013. Ontology Mapping using Description Logic and Bridging Axioms. *Computers in Industry, 64*(1), pp.19-28.

Laclavık, M., Šeleng, M. and Babık, M., 2006. Ontea: Semi-Automatic Ontology-Based Text Annotation Method. *Tools for Acquisition, Organisation and Presenting of Information and Knowledge*, pp.49-63.

Landragin, F., Poibeau, T. & Victorri, B., 2012, May. ANALEC: A New Tool for the Dynamic Annotation of Textual Data. In *International Conference on Language Resources and Evaluation (LREC 2012)* (pp. 357-362).

Lascano, J.E., 2017. A Pattern Language for Designing Application-Level Communication Protocols and the Improvement of Computer Science Education Through Cloud Computing. Utah State University.

Le Noac'H, P., Costan, A. & Bougé, L., 2017, December. A Performance Evaluation of Apache Kafka in Support of Big Data Streaming Applications. In *2017 IEEE International Conference on Big Data (Big Data)* (pp. 4803-4806). IEEE.

Leach, K., Zhang, F. & Weimer, W., 2017, September. Scotch: Combining Software Guard Extensions and System Management Mode to Monitor Cloud Resource Usage. In *International Symposium on Research in Attacks, Intrusions, and Defenses* (pp. 403-424). Springer, Cham.

Lehtonen, S. & Pärssinen, J., 2001. A Pattern Language for Key Management. Procs. of PLoP 2001.

Leymann, F., Breitenbücher, U., Wagner, S. & Wettinger, J., 2016, April. Native Cloud Applications: Why Monolithic Virtualisation is not their Foundation. In *International Conference on Cloud Computing and Services Science* (pp. 16-40). Springer, Cham.

Li, C., Sun, H., Tang, H., & Luo, Y. (2019). Adaptive Resource Allocation based on the Billing Granularity in Edge-Cloud Architecture. *Computer Communications*.

Liaqat, M., Chang, V., Gani, A., Ab Hamid, S.H., Toseef, M., Shoaib, U. & Ali, R.L., 2017. Federated Cloud Resource Management: Review and Discussion. *Journal of Network and Computer Applications, 77*, pp.87-105.

Lins, S., Grochol, P., Schneider, S. & Sunyaev, A., 2016. Dynamic Certification of Cloud Services: Trust but Verify! *IEEE Security & Privacy, 14*(2), pp.66-71.

Liu, C.H., Chen, H.C., Jain, J.L. and Chen, J.Y., 2009, March. Semi-Automatic Annotation System for OWL-Based Semantic Search. In *2009 International Conference on Complex, Intelligent and Software Intensive Systems* (pp. 475-480). IEEE.

Liu, F., Li, P. & Deng, D., 2017. Device-Oriented Automatic Semantic Annotation in IoT. *Journal of Sensors, 2017*.

Liu, F., Tong, J., Mao, J., Bohn, R., Messina, J., Badger, L. & Leaf, D., 2011. NIST Cloud Computing Reference Architecture. *NIST Special Publication, 500*(2011), pp.1-28.

Liu, S., Chan, F.T., Yang, J. & Niu, B., 2018. Understanding the effect of Cloud Computing on Organisational Agility: An Empirical Examination. *International Journal of Information Management, 43*, pp.98-111.

Loader.io. (2019). Application Load Testing Tools for API Endpoints with loader.io. [online] Available at: https://loader.io/ [Accessed 14 Mar. 2019].

Losch, U., Rudolph, S., Vrandečić, D. & Studer, R., 2009, May. Tempus Fugit: Towards an Ontology Update Language. In *European Semantic Web Conference* (pp. 278-292). Springer, Berlin, Heidelberg.

Luczak-Rosch, M., Simperl, E., Stadtmüller, S. & Käfer, T., 2014. The Role of Ontology Engineering in Linked Data Publishing and Management: An Empirical Study. *International Journal on Semantic Web and Information Systems (IJSWIS), 10*(3), pp.74-91.

Mahemoff, M.J. & Johnston, L.J., 1998, November. Principles for a Usability-Oriented Pattern Language. In *Computer Human Interaction Conference, 1998. Proceedings. 1998 Australasian* (pp. 132-139). IEEE.

Mahemoff, M.J. & Johnston, L.J., 1999, September. The Planet Pattern Language for Software Internationalisation. In *Proceedings of the 6th Annual Conference on the Pattern Languages of Programs*, Retrieved July (Vol. 30, p. 2009).

Mainetti, L., Mighali, V. & Patrono, L., 2015. A Software Architecture enabling the Web of Things. *IEEE Internet of Things Journal, 2*(6), pp.445-454.

Makhsous, S.H., Gulenko, A., Kao, O. & Liu, F., 2016, July. High Available Deployment of Cloud-Based Virtualized Network Functions. In *2016 International Conference on High Performance Computing & Simulation (HPCS)* (pp. 468-475). IEEE.

Makki, J., 2017. OntoPRiMa: A Prototype for Automating Ontology Population. *International Journal of Web/Semantic Technology (IJWesT), 8*.

Malik, S.K., Prakash, N. & Rizvi, S.A.M., 2010. Semantic Annotation Framework for Intelligent Information Retrieval using KIM Architecture. *International Journal of Web & Semantic Technology (IJWest), 1*(4), pp.12-26.

Manjusha, R. & Ramachandran, R., 2015. Secure Authentication and Access System for Cloud Computing Auditing Services using Associated Digital Certificate. *Indian Journal of Science and Technology, 8*, p.220.

Manolescu, D.A., 1997, September. A Dataflow Pattern Language. In *Proceedings of the 4th Pattern Languages of Programming Conference*.

Manzoor, D., Ali, A., & Ahmad, A. (2014). Cloud and Web Technologies: Technical Improvements and Their Implications on E-Governance. In *International Journal of Advanced Computer Science and Applications on* (pp. 196-201).

March, S.T. & Smith, G.F., 1995. Design and Natural Science Research on Information Technology. *Decision Support Systems, 15*(4), pp.251-266.

March, S.T. & Storey, V.C., 2008. Design Science in the Information Systems Discipline: An Introduction to the Special Issue on Design Science Research. *MIS Quarterly, 32*(4), pp.725-730.

Marcinczuk, M., Kocon, J. & Broda, B., 2012. Inforex - A Web-Based Tool for Text Corpus Management and Semantic Annotation. In *International Conference on Language Resources and Evaluation* (pp. 224-230).

Marr, B. (2018). How Much Data Do We Create Every Day? The Mind-Blowing Stats Everyone Should Read. [online] Forbes.com. Available at: https://www.forbes.com/sites/bernardmarr/2018/05/21/how-much-data-do-we-create-every-day-the-mind-blowing-stats-everyone-should-read/ [Accessed 15 Jul. 2018].

Marrero, M. & Urbano, J., 2018. A Semi-Automatic and Low-Cost Method to Learn Patterns for Named Entity Recognition. *Natural Language Engineering, 24*(1), pp.39-75.

Martinez-Lopez, F.J., Anaya-Sánchez, R., Aguilar-Illescas, R. & Molinillo, S., 2016. Evolution of the Web. In *Online Brand Communities* (pp. 5-15). Springer, Cham.

Martinez-Rodriguez, J.L., Hogan, A. & Lopez-Arevalo, I., 2018. Information Extraction Meets the Semantic Web: A Survey. *Semantic Web*, (Preprint), pp.1-81.

Massingill, B.L., Mattson, T.G. & Sanders, B.A., 2001. Parallel Programming with a Pattern Language. *International Journal on Software Tools for Technology Transfer, 3*(2), pp.217-234.

Mehmood, S. & Umar, A.I., 2017. Scalable and Flexible SLA Management Approach for Cloud. *Mehran University Research Journal of Engineering & Technology, 36*(1), p.87.

Mell, P. & Grance, T., 2011. The NIST Definition of Cloud Computing. *National Institute of Standards and Technology, U.S. Department of Commerce, Special Publication* 80-145.

Menzies, T., Williams, L. & Zimmermann, T., 2016. *Perspectives on Data Science for Software Engineering*. Morgan Kaufmann.

Meszaros, G. & Brown, K., 1997, September. A Pattern Language for Workflow Systems. In *Proceedings of the 4th Pattern Languages of Programming Conference, Washington University Technical Report* (pp. 97-34).

Michel, G., Amal, Z., Francisco, A., Faezeh, E. & Ludovic, J.L., 2018. An Analysis of the Semantic Annotation Task on the Linked Data Cloud. *arXiv preprint arXiv:1811.05549*.

Mika, P. (2008). The Semantic Web in the Cloud. *IEEE Intelligent Systems*.

Milani, A.S. & Navimipour, N.J., 2016. Load Balancing Mechanisms and Techniques in the Cloud Environments: Systematic Literature Review and Future Trends. *Journal of Network and Computer Applications, 71*, pp.86-98.

Mittra, T. & Ali, M.M., 2017. Parallelized and Distributed Task-Based Ontology Matching in Clustering Environment with Semantic Verification. *CSI Transactions on ICT, 5*(3), pp.265-279.

Mohammed, B., Kiran, M., Maiyama, K.M., Kamala, M.M. & Awan, I.U., 2017. Failover Strategy for Fault Tolerance in Cloud Computing Environment. *Software: Practice and Experience, 47*(9), pp.1243-1274.

Molin, P. & Ohlsson, L., 1996, September. Points & Deviations - A Pattern Language for Fire Alarm Systems. In *Proceedings of the 3rd International Conference on Pattern Languages for Programming* (No. 6.6).

Morabito, R., 2017. Virtualization on Internet of Things Edge Devices with Container Technologies: A Performance Evaluation. *IEEE Access, 5,* pp.8835-8850.

Mulyar, N. & van der Aalst, W.M., 2005, October. Towards a Pattern Language for Coloured Petri Nets. In S*ixth Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools* (p. 39).

Munir, K. & Anjum, M.S., 2018. The Use of Ontologies for Effective Knowledge Modelling and Information Retrieval. *Applied Computing and Informatics, 14*(2), pp.116-126.

Nacer, H. & Aissani, D., 2014. Semantic Web Services: Standards, Applications, Challenges and Solutions. *Journal of Network and Computer Applications, 44*, pp.134-151.

Naeem, M.M., Memon, H.M.F., Siddique, M. & Rauf, A., 2016, July. An Overview of Virtualisation & Cloud Computing

Namasudra, S., Roy, P. & Balusamy, B., 2017, February. Cloud Computing: Fundamentals and Research Issues. In *2017 Second International Conference on Recent Trends and Challenges in Computational Models (ICRTCCM)* (pp. 7-12). IEEE.

Narula, G.S., Wason, R., Jain, V. & Baliyan, A., 2018. Ontology Mapping and Merging Aspects in Semantic Web. *International Robotics & Automation Journal, 4*(1), p.00087.

Navarrete, R. & Lujan-Mora, S., 2018. A Quantitative Analysis of the use of Microdata for Semantic Annotations on Educational Resources. *Journal of Web Engineering, 17*(1&2), pp.045-072.

Neveol, A., Doğan, R.I. & Lu, Z., 2011. Semi-Automatic Semantic Annotation of PubMed Queries: A Study on Quality, Efficiency, Satisfaction. *Journal of Biomedical Informatics, 44*(2), pp.310-318.

Newman, S., 2015. *Building Microservices: Designing Fine-Grained Systems*. "O'Reilly Media, Inc.".

Niklaus, C., Cetto, M., Freitas, A. & Handschuh, S., 2018. A Survey on Open Information Extraction. *arXiv preprint arXiv:1806.05599.*

Novak, J.H., Kasera, S.K. & Stutsman, R., 2019, January. Cloud Functions for Fast and Robust Resource Auto-Scaling. In *2019 11th International Conference on Communication Systems & Networks (COMSNETS)* (pp. 133-140). IEEE.

Nunamaker Jr, J.F., Chen, M. & Purdin, T.D., 1990. Systems Development in Information Systems Research. *Journal of Management Information Systems, 7*(3), pp.89-106.

O'reilly, T., 2007. What is Web 2.0: Design Patterns and Business Models for the Next Generation of Software. *Communications & Strategies,* (1), p.17.

Offermann, P., Levina, O., Schönherr, M. & Bub, U., 2009, May. Outline of a Design Science Research Process. In *Proceedings of the 4th International Conference on Design Science Research in Information Systems and Technology* (p. 7). ACM.

Oliveira, P. & Rocha, J., 2013, April. Semantic Annotation Tools Survey. In *2013 IEEE Symposium on Computational Intelligence and Data Mining (CIDM)* (pp. 301-307). IEEE.

Opdyke, W.F., 1990. Refactoring: An Aid in Designing Application Frameworks and Evolving Object-Oriented Systems. In *Proc. SOOPPA'90: Symposium on Object-Oriented Programming Emphasizing Practical Applications*.

Oracle (2018). Using Oracle Messaging Cloud Service. [online] Oracle Help Center. Available at: https://docs.oracle.com/en/cloud/iaas/messaging-cloud/csmes/rest-api-reference.html [Accessed 14 Mar. 2019].

Pahl, C., Brogi, A., Soldani, J. & Jamshidi, P., 2017. Cloud Container Technologies: A State-of-the-Art Review. *IEEE Transactions on Cloud Computing*.

Paladi, N., Michalas, A. & Dang, H.V., 2018, April. Towards Secure Cloud Orchestration for Multi-Cloud Deployments. In *Proceedings of the 5th Workshop on CrossCloud Infrastructures & Platforms* (p. 4). ACM.

Papadopoulos, A.V., Ali-Eldin, A., Årzén, K.E., Tordsson, J. & Elmroth, E., 2016. PEAS: A Performance Evaluation Framework for Auto-Scaling Strategies in Cloud Applications. *ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS), 1*(4), p.15.

Paris, A., Andreas, P., Symeon, R. & Manolis, S., 2003. Towards a Pattern Language for Learning Management Systems. *Journal of Educational Technology & Society, 6*(2).

Park, J., Oh, S. & Ahn, J., 2011. Ontology Selection Ranking Model for Knowledge Reuse. *Expert Systems with Applications, 38*(5), pp.5133-5144.

Pautasso, C., Ivanchikj, A. & Schreier, S., 2016, July. A Pattern Language for RESTful Conversations. In *Proceedings of the 21st European Conference on Pattern Languages of Programs* (p. 4). ACM.

Peffers, K., Tuunanen, T., Rothenberger, M.A. & Chatterjee, S., 2007. A Design Science Research Methodology for Information Systems Research. *Journal of Management Information Systems*, 24(3), pp.45-77.

Petasis, G., Karkaletsis, V., Paliouras, G., Krithara, A. & Zavitsanos, E., 2011, January. Ontology Population and Enrichment: State of the Art. In *Knowledge-driven Multimedia Information Extraction and Ontology Evolution* (pp. 134-166). Springer-Verlag.

Piao, S., Bianchi, F., Dayrell, C., D'Egidio, A. & Rayson, P., 2015. Development of the Multilingual Semantic Annotation System. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies* (pp. 1268-1274).

Pileggi, S. F., Calvo-Gallego, J., & Amor, R. (2013). Bringing Semantic Resources Together in the Cloud: From Theory to Application. In *Computational Intelligence, Modelling and Simulation (CIMSim), 2013 Fifth International Conference on* (pp. 113-118). IEEE.

Popov, B., Kiryakov, A., Kirilov, A., Manov, D., Ognyanoff, D. & Goranov, M., 2003, October. KIM – Semantic Annotation Platform. In *International Semantic Web Conference* (pp. 834-849). Springer, Berlin, Heidelberg.

Prabhu, D., 2017. Application of Web 2.0 and Web 3.0: An Overview. *LAP LAMBERT Academic Publishing.*

Prasad, M.R., Manjula, B. & Bapuji, V., 2013. A Novel Overview and Evolution of World Wide Web: Comparison from Web 1.0 to Web 3.0. *International Journal of Computer Science and Technology, 4*(1), p.352.

Prometheus.io. (2019). Prometheus - Monitoring System & Time Series Database. [online] Available at: https://prometheus.io/ [Accessed 12 Apr. 2019].

Pyarali, I., O'Ryan, C. & Schmidt, D.C., 2000. A Pattern Language for Efficient, Predictable, Scalable, and Flexible Dispatching Mechanisms for Distributed Object Computing Middleware. In *Object-Oriented Real-Time Distributed Computing, 2000. (ISORC 2000) Proceedings. Third IEEE International Symposium on* (pp. 62-69). IEEE.

Ranganathan, R., 2018. A Highly Available and Scalable Microservice Architecture for Access Management.

Rathod, N. & Surve, A., 2015, January. Test Orchestration a Framework for Continuous Integration and Continuous Deployment. In *2015 International Conference on Pervasive Computing (ICPC)* (pp. 1-5). IEEE.

Re, R., Braga, R.T. & Masiero, P.C., 2001. A Pattern Language for Online Auctions Management. In *Proceedings of PLoP* (Vol. 2001).

Reeve, L. & Han, H., 2005, March. Survey of Semantic Annotation Platforms. In *Proceedings of the 2005 ACM Symposium on Applied Computing* (pp. 1634-1638). ACM.

Rekik, M., Boukadi, K., & Ben-Abdallah, H. (2015). Cloud Description Ontology for Service Discovery and Selection. In *2015 10th International Joint Conference on Software Technologies (ICSOFT)* (pp. 1-11). IEEE.

Rezaei, R., Chiew, T.K., Lee, S.P. & Aliee, Z.S., 2014. A Semantic Interoperability Framework for Software as a Service Systems in Cloud Computing Environments. *Expert Systems with Applications, 41*(13), pp.5751-5770.

Richards, M., 2015. Software Architecture Patterns. O'Reilly Media, Incorporated.

Richards, M., 2015. Microservices vs. Service-Oriented Architecture. O'Reilly Media.

Richardson, C., 2001. A Pattern Language for J2EE Web Component Development. In *Proceedings of the 8th Conference on Pattern Languages and Programming*. ACM.

Richardson, C., 2015. Microservices Architecture (2014).

Rittinghouse, J.W. & Ransome, J.F., 2017. *Cloud Computing: Implementation, Management, and Security*. CRC Press.

Rodriguez, M.A. & Buyya, R., 2019. Container-Based Cluster Orchestration Systems: A Taxonomy and Future Directions. *Software: Practice and Experience, 49*(5), pp.698-719.

Rodriguez-Garcia, M.Á., Valencia-García, R., García-Sánchez, F. & Samper-Zapater, J.J., 2014. Ontology-Based Annotation and Retrieval of Services in the Cloud. *Knowledge-Based Systems, 56*, pp.15-25.

Rossi, G., Schwabe, D. & Garrido, A., 1996, September. Towards a Pattern Language for Hypermedia Applications. In *Proceedings of the 3rd Annual Conference on Pattern Languages of Programs* (Vol. 96).

Rudman, R. & Bruwer, R., 2016. Defining Web 3.0: Opportunities and Challenges. *The Electronic Library, 34*(1), pp.132-154.

Sabou, M., Lopez, V., Motta, E. & Uren, V., 2006. Ontology Selection: Ontology Evaluation on the Real Semantic Web.

Sachdeva, R., 2016, May. Automated Testing in DevOps. *Pacific Northwest Software Quality Conference 2016*.

Saeed, A., Ibrahim, M., Harras, K.A. & Youssef, M., 2015. Toward Dynamic Real-Time Geo-Location Databases for TV White Spaces. *IEEE Network, 29*(5), pp.76-82.

Sajja, P.S. & Akerkar, R., 2016. *Intelligent Technologies for Web Applications*. Chapman and Hall/CRC.

Salih, A.Q.M., 2013. Towards from Manual to Automatic Semantic Annotation: Based on Ontology Elements and Relationships. *International Journal of Web & Semantic Technology, 4*(2), p.21.

Samet, H., Sankaranarayanan, J., Lieberman, M.D., Adelfio, M.D., Fruin, B.C., Lotkowski, J.M., Panozzo, D., Sperling, J. & Teitler, B.E., 2014. Reading News with Maps by Exploiting Spatial Synonyms. *Communications of the ACM, 57*(10), pp.64-77.

Sanchiz, M., Chin, J., Chevalier, A., Fu, W.T., Amadieu, F. & He, J., 2017. Searching for Information on the Web: Impact of Cognitive Aging, Prior Domain Knowledge and Complexity of the Search Problems. *Information Processing & Management, 53*(1), pp.281-294.

Sangers, J., Hogenboom, F. & Frasincar, F., 2012, November. Event-Driven Ontology Updating. In *International Conference on Web Information Systems Engineering* (pp. 44-57). Springer, Berlin, Heidelberg.

Santana-Perez, I., da Silva, R. F., Rynge, M., Deelman, E., Pérez-Hernández, M. S., & Corcho, O. (2017). Reproducibility of Execution Environments in Computational Science using Semantics and Clouds. *Future Generation Computer Systems, 67*, 354-367.

Santhosh, R. & Ravichandran, T., 2013, February. Pre-Emptive Scheduling of Online Real Time Services with Task Migration for Cloud Computing. In *2013 International Conference on Pattern Recognition, Informatics and Mobile Engineering* (pp. 271-276). IEEE.

Sassi, N., Jaziri, W. & Alharbi, S., 2016. Supporting Ontology Adaptation and Versioning based on a Graph of Relevance. *Journal of Experimental & Theoretical Artificial Intelligence, 28*(6), pp.1035-1059.

Saxena, V.K. & Pushkar, S., 2016, March. Cloud Computing Challenges and Implementations. In *2016 International Conference on Electrical, Electronics, and Optimisation Techniques (ICEEOT)* (pp. 2583-2588). IEEE.

Schema.org. (2019). *Schemas - Schema.org.* [online] Available at: https://schema.org/docs/schemas.html [Accessed 14 June 2019].

Schneider, A.W. & Matthes, F., 2015, July. Evolving the EAM Pattern Language. In *Proceedings of the 20th European Conference on Pattern Languages of Programs* (p. 45). ACM.

Schummer, T., 2003, February. GAMA: A Pattern Language for Computer Supported Dynamic Collaboration. In *EuroPLoP* (pp. 53-114).

Scotland, J., 2012. Exploring the Philosophical Underpinnings of Research: Relating Ontology and Epistemology to the Methodology and Methods of the Scientific, Interpretive, and Critical Research Paradigms. *English Language Teaching, 5*(9), pp.9-16.

Senyo, P.K., Addae, E. & Boateng, R., 2018. Cloud Computing Research: A Review of Research Themes, Frameworks, Methods and Future Research Directions. *International Journal of Information Management, 38*(1), pp.128-139.

Shadija, D., Rezai, M. & Hill, R., 2017, September. Towards an Understanding of Microservices. In *2017 23rd International Conference on Automation and Computing (ICAC)* (pp. 1-6). IEEE.

Shahapure, N.H. & Jayarekha, P., 2015. Replication: A Technique for Scalability in Cloud Computing. *International Journal of Computer Applications, 122*(5).

Shaheen, J. (2017). Apache Kafka: Real Time Implementation with Kafka Architecture Review. *International Journal of Advanced Science and Technology, 109*, pp.35-42.

Shao, J., Wei, H., Wang, Q. & Mei, H., 2010, July. A Runtime Model-Based Monitoring Approach for Cloud. In *2010 IEEE 3rd International Conference on Cloud Computing* (pp. 313-320). IEEE.

Sheth, A. & Thirunarayan, K., 2012. Semantics Empowered Web 3.0: Managing Enterprise, Social, Sensor, and Cloud-Based Data and Services for Advanced Applications. *Synthesis Lectures on Data Management, 4*(6), pp.1-175.

Silva, A.R., Hayes, F., Mota, F., Torres, N. & Santos, P., 1996. A Pattern Language for the Perception, Design and Implementation of Distributed Application Partitioning. In *Workshop on Methodologies for Distributed Objects*.

Singh, P., 2019. Natural Language Processing. In *Machine Learning with PySpark* (pp. 191-218). Apress, Berkeley, CA.

Singh, S., Chana, I. & Buyya, R., 2017. STAR: SLA-Aware Autonomic Management of Cloud Resources. *IEEE Transactions on Cloud Computing*.

Sjoberg, D.I., Dyba, T. & Jorgensen, M., 2007, May. The Future of Empirical Methods in Software Engineering Research. In *2007 Future of Software Engineering* (pp. 358-378). IEEE Computer Society.

SlideWiki (2016). Ontology Engineering Phases. [Online Image] Available from - http://slidewiki.org/upload/media/images/141/2338.png. [Accessed: 14th July 2016].

Slimani, T., 2013. Semantic Annotation: The Mainstay of Semantic Web. *arXiv preprint arXiv:1312.4794.*

Slimani, T., 2015. Ontology Development: A Comparing Study on Tools, Languages and Formalisms. *Indian Journal of Science and Technology, 8*(24), pp.1-12.

Slimani, T., 2014. A Study on Ontologies and their Classification. *Recent Advances in Electrical Engineering and Educational Technologies.*

Solms, F., 2012, October. What is Software Architecture? In *Proceedings of the South African Institute for Computer Scientists and Information Technologists Conference* (pp. 363-373). ACM.

Somasundaram, T. S., Govindarajan, K., & Rao, S. M. (2012). An Architectural Framework to solve the Interoperability issue between Private Clouds using Semantic Technology. In *Recent Trends in Information Technology (ICRTIT), 2012 International Conference on* (pp. 162-167). IEEE.

Sonali, S.S.S.G.P., Bogiri, D.S.M.P.N. & Guide, S.S.S.S., 2015. Cloud Computing Security: From Single Cloud to Multi-Clouds using Digital Signature. *International Journal of Advanced Research in Computer Engineering & Technology (IJARCET), 4*(4).

Staab, S., Maedche, A. and Handschuh, S., 2001. *An Annotation Framework for the Semantic Web* (pp. 30-31). Institute of Applied Informatics and Formal Description Methods (AIFB)

Stahl, D. & Bosch, J., 2016, May. Industry Application of Continuous Integration Modelling: A Multiple-Case Study. In *2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C)* (pp. 270-279). IEEE.

Stahlin, J., Lang, S., Kajzar, F. & Zirpins, C., 2016, September. Consumer-Driven API Testing with Performance Contracts. In *European Conference on Service-Oriented and Cloud Computing* (pp. 135-143). Springer, Cham.

Stenberg, D. (2019). Introduction · Everything cURL. [online] Ec.haxx.se. Available at: https://ec.haxx.se/ [Accessed 4 Apr. 2019].

Stenetorp, P., Pyysalo, S., Topić, G., Ohta, T., Ananiadou, S. & Tsujii, J.I., 2012, April. BRAT: A Web-based Tool for NLP-assisted Text Annotation. In *Proceedings of the Demonstrations at the 13th Conference of the European Chapter of the Association for Computational Linguistics* (pp. 102-107). Association for Computational Linguistics.

Stepney, S., 2012, September. A Pattern Language for Scientific Simulations. In *Proceedings of the 2012 Workshop on Complex Systems Modelling and Simulation, Orleans, France* (pp. 77-103).

Suarez-Figueroa, M.C., Gómez-Pérez, A. & Fernández-López, M., 2012. The NeOn Methodology for Ontology Engineering. In *Ontology Engineering in a Networked World* (pp. 9-34). Springer, Berlin, Heidelberg.

Sui, B., Li, Q., Wen, J. & Yao, Y., 2018, August. Billing System Design of Cloud Services. In *2018 3rd International Conference on Control, Automation and Artificial Intelligence (CAAI 2018)*. Atlantis Press.

Tablan, V., Roberts, I., Cunningham, H. & Bontcheva, K., 2013. GATECloud.net: A Platform for Large-Scale, Open-Source Text Processing on the Cloud. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences, 371(1983)*, p.20120071.

Tate, M.A., 2018. *Web Wisdom: How to Evaluate and Create Information Quality on the Web*. CRC Press.

Taherizadeh, S. & Stankovski, V., 2018. Dynamic Multi-Level Auto-Scaling Rules for Containerised Applications. *The Computer Journal, 62*(2), pp.174-197.

Taibi, D., Lenarduzzi, V., Pahl, C. & Janes, A., 2017, May. Microservices in Agile Software Development: A Workshop-Based Study into Issues, Advantages, and Disadvantages. In *Proceedings of the XP2017 Scientific Workshops* (p. 23). ACM.

Taibi, D., Lenarduzzi, V. & Pahl, C., 2018, March. Architectural Patterns for Microservices: A Systematic Mapping Study. In *CLOSER* (pp. 221-232).

Takeda, H., Veerkamp, P., Tomiyama, T., & Yoshikawam, H. (1990). Modelling Design Processes. *AI Magazine Winter*: 37–48.

Talukdar, T., Batra, G., Vaidya, J., Atluri, V. & Sural, S., 2017, October. Efficient Bottom-Up Mining of Attribute-Based Access Control Policies. In *2017 IEEE 3rd International Conference on Collaboration and Internet Computing (CIC)* (pp. 339-348). IEEE.

Tamper, M., Leskinen, P., Ikkala, E., Oksanen, A., Mäkelä, E., Heino, E., Tuominen, J., Koho, M. & Hyvönen, E., 2017, June. AATOS – A Configurable Tool for Automatic Annotation. In *International Conference on Language, Data and Knowledge* (pp. 276-289). Springer, Cham.

Tanasseri, N. & Rai, R., 2017. Microservices with Azure: Build Highly Maintainable and Scalable Enterprise-Grade Apps.

Tang, J., Zhang, D., Yao, L. & Li, Y., 2012. Automatic Semantic Annotation Using Machine Learning. In *Machine Learning: Concepts, Methodologies, Tools and Applications* (pp. 535-578). IGI Global.

Tao, C., Song, D., Sharma, D. & Chute, C.G., 2013. Semantator: Semantic Annotator for Converting Biomedical Text to Linked Data. *Journal of Biomedical Informatics, 46*(5), pp.882-893.

Thones, J., 2015. Microservices. *IEEE Software, 32*(1), pp.116-116.

Thornhill, A., Saunders, M. & Lewis, P., 2009. *Research Methods for Business Students*. London: Pearson Education.

Tidwell, J., 1997. A Pattern Language for Human-Computer Interface Design. Available via DIALOG.

Tjoa, A.M., Andjomshoaa, A., Shayeganfar, F. & Wagner, R., 2005, August. Semantic Web Challenges and New Requirements. In *16th International Workshop on Database and Expert Systems Applications (DEXA'05)* (pp. 1160-1163). IEEE.

Tomaz, H., Lima, R., Emanoel, J. & Freitas, F., 2012, November. An Unsupervised Method for Ontology Population from the Web. In *Ibero-American Conference on Artificial Intelligence* (pp. 41-50). Springer, Berlin, Heidelberg.

Trajanov, D., Stojanov, R., Jovanovik, M., Zdraveski, V., Ristoski, P., Georgiev, M., & Filiposka, S. (2012). Semantic Sky: A Platform for Cloud Service Integration Based on Semantic Web Technologies. In *Proceedings of the 8th International Conference on Semantic Systems* (pp. 109-116). ACM.

Tulasi, R.L., Rao, M.S., Ankita, K. & Hgoudar, R., 2017. Ontology-Based Automatic Annotation: An Approach for Efficient Retrieval of Semantic Results of Web Documents. In *Proceedings of the First International Conference on Computational Intelligence and Informatics* (pp. 331-339). Springer, Singapore.

Uren, V., Cimiano, P., Iria, J., Handschuh, S., Vargas-Vera, M., Motta, E. & Ciravegna, F., 2006. Semantic Annotation for Knowledge Management: Requirements and a Survey of the State of the Art. *Web Semantics: Science, Services and Agents on the World Wide Web, 4*(1), pp.14-28.

Vaishnavi, V., Kuechler, W., & Petter, S. (Eds.) (2004/17). “Design Science Research in Information Systems” January 20, 2004 (created in 2004 and updated until 2015 by Vaishnavi, V. and Kuechler, W.); last updated (by Vaishnavi, V. and Petter, S.), December 20, 2017. URL: http://www.desrist.org/design-research-in-information-systems/.

VanHilst, M. & Notkin, D., 1996. Using Role Components in Implement Collaboration-Based Designs. *ACM SIGPLAN Notices, 31*(10), pp.359-369.

Vegetti, M., Roldán, L., Gonnet, S., Leone, H. & Henning, G., 2016. A Framework to Represent, Capture, and Trace Ontology Development Processes. *Engineering Applications of Artificial Intelligence, 56*, pp.230-249.

Veloudis, S., & Paraskakis, I. (2016). Defining an Ontological Framework for Modelling Policies in Cloud Environments. In *Cloud Computing Technology and Science (CloudCom), 2016 IEEE International Conference on* (pp. 277-284). IEEE.

Verspoor, K. & Cohen, K.B., 2013. Natural Language Processing. *Encyclopaedia of Systems Biology*, pp.1495-1498.

Verspoor, K., Kim, J.D. & Dumontier, M., 2015, December. Interoperability of Text Corpus Annotations with the Semantic Web. In *BMC proceedings* (Vol. 9, No. 5, p. A2). BioMed Central.

Villamizar, M., Garcés, O., Castro, H., Verano, M., Salamanca, L., Casallas, R. & Gil, S., 2015, September. Evaluating the Monolithic and the Microservice Architecture Pattern to Deploy Web Applications in the Cloud. In *2015 10th Computing Colombian Conference (10CCC)* (pp. 583-590). IEEE.

Vlahovic, N., 2011. Information Retrieval and Information Extraction in Web 2.0 Environment. *International Journal of Computers, 5*(1), pp.1-9.

Wang, G., Koshy, J., Subramanian, S., Paramasivam, K., Zadeh, M., Narkhede, N., Rao, J., Kreps, J. & Stein, J., 2015. Building a Replicated Logging System with Apache Kafka. *Proceedings of the VLDB Endowment, 8*(12), pp.1654-1655.

Wang, H., Wang, W., Cui, L., Sun, H., Zhao, J., Wang, Y. & Xue, Y., 2018. A Hybrid Multi-Objective Firefly Algorithm for Big Data Optimization. *Applied Soft Computing, 69*, pp.806-815.

Wang, J., Bai, X., Ma, H., Li, L. & Ji, Z., 2017, March. Cloud API Testing. In *2017 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)* (pp. 385-386). IEEE.

Wang, X., Zhang, X. & Li, M., 2015. A Survey on Semantic Sensor Web: Sensor Ontology, Mapping and Query. *International Journal of u-and e-Service, Science and Technology, 8*(10), pp.325-342.

Ward, J. S., & Barker, A. (2012). Semantic Based Data Collection for Large Scale Cloud Systems. In *Proceedings of the fifth international workshop on Data-Intensive Distributed Computing Date* (pp. 13-22). ACM.

Weiss, M., 2003, July. A Pattern Language for Motivating the use of Agents. In *International Bi-Conference Workshop on Agent-Oriented Information Systems* (pp. 142-157). Springer, Berlin, Heidelberg.

Werner, J., Westphall, C.M. & Westphall, C.B., 2017. Cloud Identity Management: A Survey on Privacy Strategies. *Computer Networks, 122*, pp.29-42.

Wilder, B., 2012. *Cloud Architecture Patterns: Using Microsoft Azure*. " O'Reilly Media, Inc.".

Wong, R.C. & Leung, C.H., 2008. Automatic Semantic Annotation of Real-World Web Images. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 30*(11), pp.1933-1944.

Wu. (2017, Dec 25). Taking the Cloud-Native Approach with Microservices. Google Inc. Retrieved from edureka.co: https://www.edureka.co/blog/what-is-microservices/

Wu, A. (2018). Running Your Modern .NET Application on Kubernetes.

Xia, Z., Zhu, Y., Sun, X., & Chen, L. (2014). Secure Semantic Expansion-Based Search Over Encrypted Cloud Data Supporting Similarity Ranking. *Journal of Cloud Computing, 3*(1), 8.

Xiangmei, Z. & Chunli, S., 2013, June. Semantic Search Technology Based on Cloud Computing Research. In *2013 Fourth International Conference on Digital Manufacturing & Automation* (pp. 1196-1198). IEEE.

Xiao, Z., Wijegunaratne, I. & Qiang, X., 2016, November. Reflections on SOA and Microservices. In *2016 4th International Conference on Enterprise Systems (ES)* (pp. 60-67). IEEE.

Yang, Y. (2015). Attribute-Based Data Retrieval with Semantic Keyword Search for E-Health Cloud. *Journal of Cloud Computing, 4*(1), 10.

Yates, A., 2007. *Information Extraction from the Web: Techniques and Applications* (Doctoral Dissertation, University of Washington).

Ye, J., Dasiopoulou, S., Stevenson, G., Meditskos, G., Kontopoulos, E., Kompatsiaris, I. & Dobson, S., 2015. Semantic Web Technologies in Pervasive Computing: A Survey and Research Roadmap. *Pervasive and Mobile Computing, 23*, pp.1-25.

Yimam, S.M., Biemann, C., de Castilho, R.E. and Gurevych, I., 2014, June. Automatic Annotation Suggestions and Custom Annotation Layers in WebAnno. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations* (pp. 91-96).

Yordanova, K. & Kruger, F., 2018. Creating and Exploring Semantic Annotation for Behaviour Analysis. *Sensors, 18*(9), p.2778.

Yosef, M.A., Hoffart, J., Bordino, I., Spaniol, M. & Weikum, G., 2011. AIDA: An Online Tool for Accurate Disambiguation of Named Entities in Text and Tables. Proceedings of the VLDB Endowment, 4(12), pp.1450-1453.

Young, T., Hazarika, D., Poria, S. & Cambria, E., 2018. Recent Trends in Deep Learning Based Natural Language Processing. *IEEE Computational Intelligence Magazine, 13*(3), pp.55-75.

Zannettou, S., Sirivianos, M., Blackburn, J. & Kourtellis, N., 2019. The Web of False Information: Rumours, Fake News, Hoaxes, Clickbait, and Various other Shenanigans. *Journal of Data and Information Quality (JDIQ), 11*(3), p.10.

Zhang, D., Yan, B.H., Feng, Z., Zhang, C. and Wang, Y.X., 2017, April. Container Oriented Job Scheduling using Linear Programming Model. In *2017 3rd International Conference on Information Management (ICIM)* (pp. 174-180). IEEE.

Zhang, M., Yuan, F. & Zhu, J., 2015, October. Integrating Semantic Knowledge into Tag-LDA Model through Cloud Model. In *2015 IEEE International Conference on Big Data (Big Data)* (pp. 2907-2909). IEEE.

Zhao, J.T., Jing, S.Y. & Jiang, L.Z., 2018, September. Management of API Gateway Based on Microservice Architecture. In *Journal of Physics: Conference Series (Vol. 1087, No. 3, p. 032032)*. IOP Publishing.

Zhao, L., Macaulay, L., Adams, J. & Verschueren, P., 2008. A Pattern Language for Designing E-business Architecture. *Journal of Systems and Software, 81*(8), pp.1272-1287.

Zhao, L., Sakr, S. & Liu, A., 2013. A Framework for Consumer-Centric SLA Management of Cloud-Hosted Databases. *IEEE Transactions on Services Computing, 8*(4), pp.534-549.

Zhu, L., Bass, L. & Champlin-Scharff, G., 2016. DevOps and its Practices. *IEEE Software, 33*(3), pp.32-34.

Zimmermann, O., 2017. Microservices Tenets. Computer Science-Research and Development, 32(3-4), pp.301-310.

Zou, L. & Ozsu, M.T., 2017. Graph-Based RDF Data Management. *Data Science and Engineering, 2*(1), pp.56-70.

# Appendix A Cloud-Based Web App Deployment Notes

This appendix details the deployment procedure for the cloud-based version of the prototype web application. The procedure detailed herein is very similar to that of the non-cloud versions; (i) the non-cloud monolithic web application and (ii) the non-cloud microservices web application.

1. Code moved to Version Control (GitHub)
2. Created Virtual Host on Amazon AWS
3. Logged into newly created virtual host
4. Enabled root login for virtual host, as follows:
    a. SSH into the machine
    b. sudo nano /etc/ssh/sshd_config
    c. sudo service ssh restart
    d. sudo passwd
5. Now able to log in using root user
6. Copied SSH public key from PC to Virtual Host /home/ubuntu/.ssh
7. Unset root password
    a. sudo passwd -l root
8. Disabled root login from password
    a. sudo nano /etc/ssh/sshd_config
    b. sudo service ssh restart
9. Installed Apache Web Server on Virtual Host, running on port 80
    a. sudo apt update
    b. sudo apt install apache2
10. Installed PHP 7.2
    a. sudo apt-get update -y
    b. sudo apt-get upgrade -y
    c. sudo apt install php libapache2-mod-php
    d. sudo systemctl restart apache2
    e. sudo apt install php-mysql php-gd
11. Installed MySQL Database on port 3306

  a. sudo apt update
  b. sudo apt install mysql-server
  c. sudo mysql_secure_installation
  d. Allowed remote login to MySQL by editing the /etc/mysql/mysql.conf.d/ mysqld.cnf
  e. MySQL
  f. GRANT ALL ON *.* to root@'%' IDENTIFIED BY '!2j*73bcXnnJ2';
  g. Application .env file is updated with MySQL database access credentials.

12. Installed Git
    a. sudo apt install git
13. Installed Composer
    a. sudo apt update
    b. sudo apt install wget php-cli php-zip unzip
    c. php -r "copy('https://getcomposer.org/installer', 'composer-setup.php');"
    d. sudo php composer-setup.php --install-dir=/usr/local/bin --filename=composer
14. Enabled mod_rewrite for Apache Web Server
    a. sudo a2enmod rewrite
    b. sudo a2enmod headers
    c. systemctl restart apache2
15. Generated SSH key for Git deployment
    a. SSH-keygen
    b. Save the public key of the server to Git repository for auto deployment
    c. Cloning the git repository at /var/www/html/cloudsea/app
    d. git clone git@bitbucket.org:tigerzs0ft/laravel.git .
16. Installed the required libraries for PHP
    a. sudo apt-get install php-mbstring
    b. sudo apt-get install php-dom
17. Installed Composer packages
    a. composer install
18. Allowed .htaccess
    a. Edit /etc/apache/apache2.conf

   b. sudo systemctl restart apache2

19. Changed home directory
    a. Edit /etc/apache2/sites-available/000-default.conf
    b. sudo systemctl restart apache2
    c. chmod -R 0755 /var/www/html/cloudsea
    d. chown -R www-data: /var/www/html/cloudsea
20. Copied database to server
21. Prepared deployment script
22. Installed Cron job on server
    a. sudo apt install cron
    b. crontab -e
    c. 30 * * * * sudo -u www-data php /var/www/html/cloudsea/app/artisan route:call --uri=/cron/annotation-fixer
    d. systemctl restart cron
23. Installed Let's encrypt for SSL (Secure Sockets Layer)
    a. sudo wget https://dl.eff.org/certbot-auto -O /usr/sbin/certbot-auto
    b. sudo chmod a+x /usr/sbin/certbot-auto
    c. sudo add-apt-repository ppa:certbot/certbot
    d. sudo apt install python-certbot-apache
    e. systemctl stop apache2
    f. sudo certbot --apache -d cloud-based.a009324a.co.uk
    g. systemctl restart apache2
    h. changed the '.env' APP URL for SSL access
    i. cd /var/www/html/cloudsea/app
    j. php artisan cache:clear
    k. php artisan config:clear
    l. Annotation API accessible via: https://cloud-based.a009324a.co.uk/annotation/annotate/guest.js

# Appendix B Web GUI for Prototype Implementation

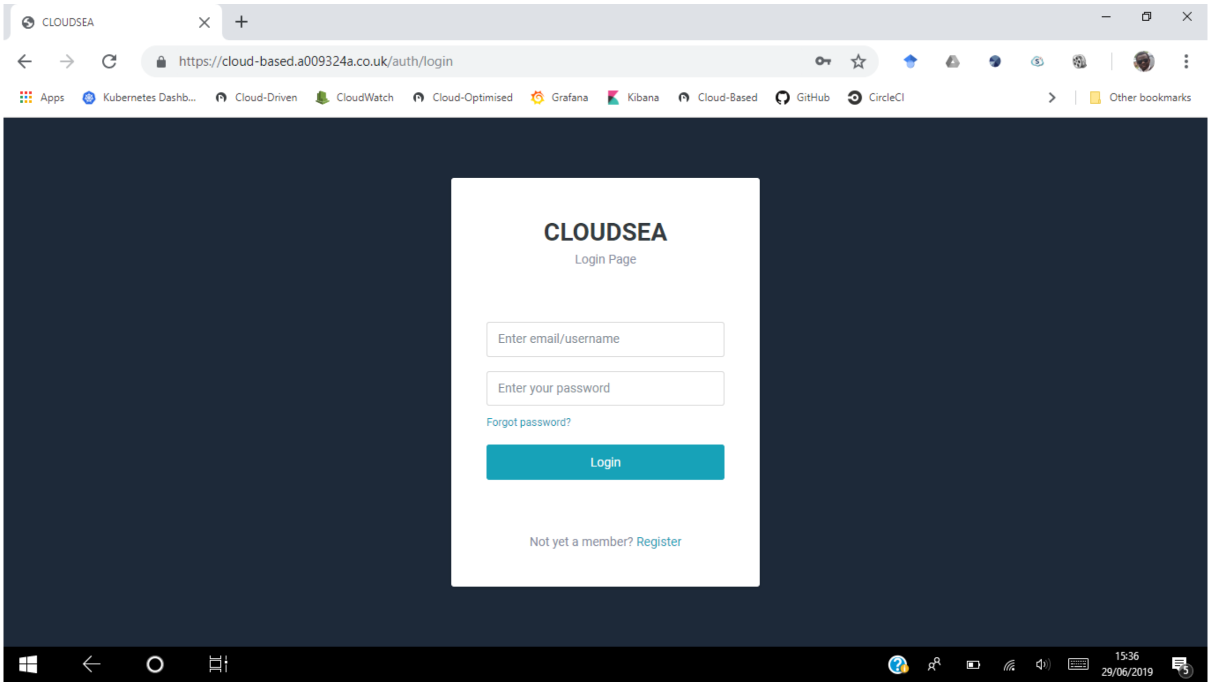


Figure B1 – The login page for CloudSea

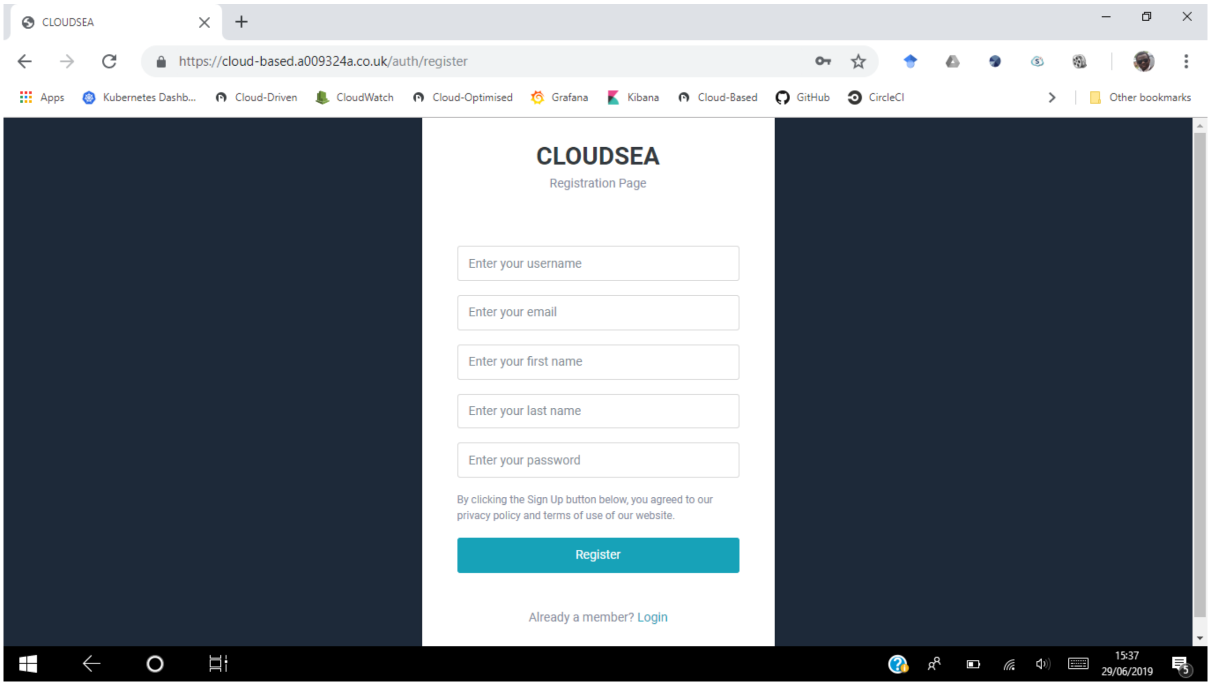
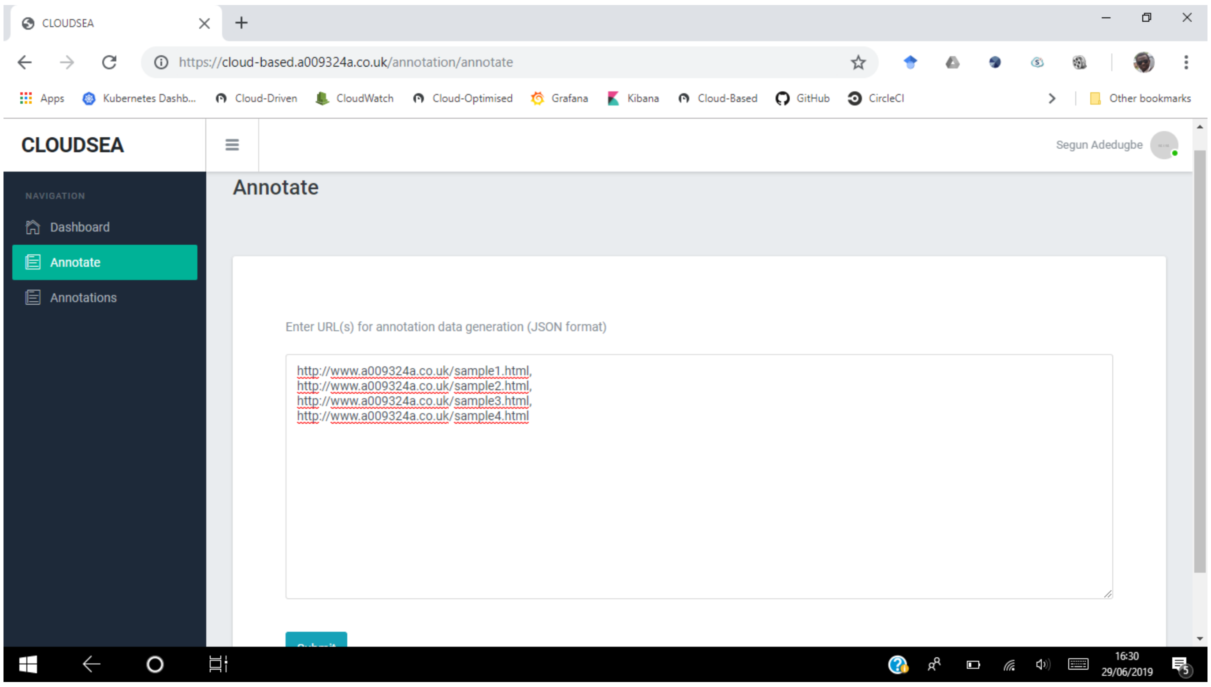


Figure B2 – The Register page for CloudSea

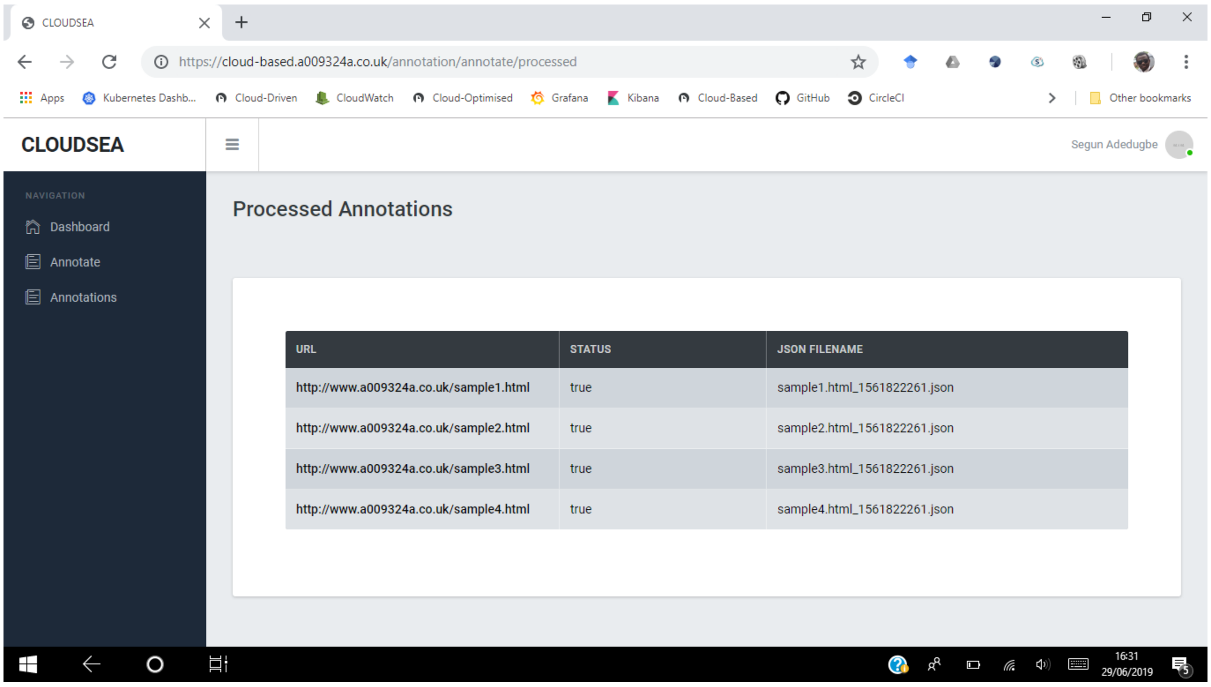

CLOUDSEA
https://cloud-based.a009324a.co.uk/annotation/annotate
Apps Kubernetes Dashb... Cloud-Driven CloudWatch Cloud-Optimised Grafana Kibana Cloud-Based GitHub CircleCI Other bookmarks
CLOUDSEA
Segun Adedugbe
NAVIGATION
Dashboard
Annotate
Annotations
16:30 29/06/2019

Annotate

Enter URL(s) for annotation data generation (JSON format)

```
http://www.a009324a.co.uk/sample1.html,
http://www.a009324a.co.uk/sample2.html,
http://www.a009324a.co.uk/sample3.html,
http://www.a009324a.co.uk/sample4.html
```

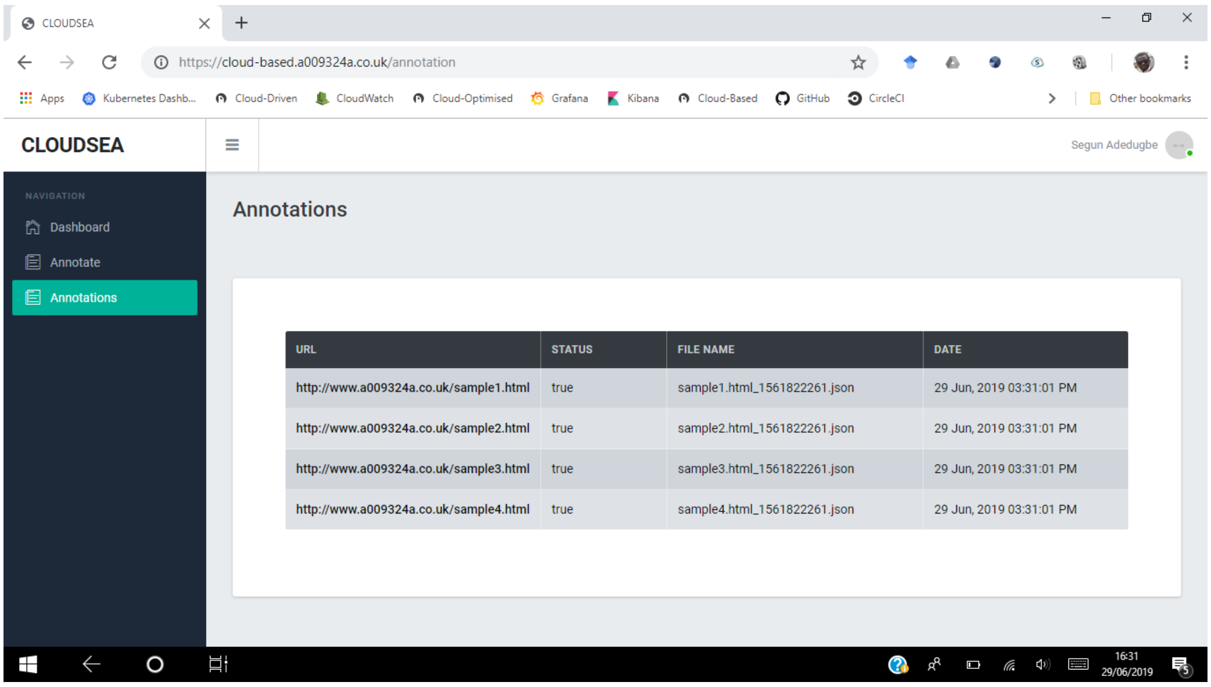
Figure B3 – CloudSea Annotate Page: Form Submission for Non-API Semantic Annotation


CLOUDSEA
https://cloud-based.a009324a.co.uk/annotation/annotate/processed
Apps Kubernetes Dashb... Cloud-Driven CloudWatch Cloud-Optimised Grafana Kibana Cloud-Based GitHub CircleCI Other bookmarks
CLOUDSEA
Segun Adedugbe
NAVIGATION
Dashboard
Annotate
Annotations
16:31 29/06/2019

Processed Annotations

| URL | STATUS | JSON FILENAME |
|---|---|---|
| http://www.a009324a.co.uk/sample1.html | true | sample1.html_1561822261.json |
| http://www.a009324a.co.uk/sample2.html | true | sample2.html_1561822261.json |
| http://www.a009324a.co.uk/sample3.html | true | sample3.html_1561822261.json |
| http://www.a009324a.co.uk/sample4.html | true | sample4.html_1561822261.json |

Figure B4 – Results of Non-API Semantic Annotation Process



Annotations

| URL | STATUS | FILE NAME | DATE |
|---|---|---|---|
| http://www.a009324a.co.uk/sample1.html | true | sample1.html_1561822261.json | 29 Jun, 2019 03:31:01 PM |
| http://www.a009324a.co.uk/sample2.html | true | sample2.html_1561822261.json | 29 Jun, 2019 03:31:01 PM |
| http://www.a009324a.co.uk/sample3.html | true | sample3.html_1561822261.json | 29 Jun, 2019 03:31:01 PM |
| http://www.a009324a.co.uk/sample4.html | true | sample4.html_1561822261.json | 29 Jun, 2019 03:31:01 PM |

Figure B5 – CloudSea Annotations Page: Displaying a list of all non-API semantic annotations by a user

# Appendix C CloudSea Annotation API Endpoint

```
function AnnotationService_Guest() {
   let annotationJquery;
   let annotationServiceEndPoint = "http://cloud-annotation-service.a009324a.co.uk";
   let annotationServiceApiEndPoint =  annotationServiceEndPoint+"/api";


   function init() {
      getScript('https://code.jquery.com/jquery-3.3.1.min.js', function () {
         annotationJquery = $.noConflict(true);
         annotate();
      });
   }


   function annotate() {
      let currentPageUrl = getCurrentPageUrl();
      annotationJquery.ajax({
         url: annotationServiceApiEndPoint+'/annotate/guest',
         method: 'POST',
         data: {
            'urls[]': currentPageUrl,
            'isOnlyTextDiv':true
         }, success: function (response) {
            if (response.hasOwnProperty('results') &&
response.results.hasOwnProperty('alreadyAssignedUrls') &&
response.results.hasOwnProperty('processedAnnotations')) {
               let alreadyAssignedUrls = response.results.alreadyAssignedUrls;
               let processedAnnotations = response.results.processedAnnotations;
               let fileName = undefined;
               let status = false;


               if (Array.isArray(alreadyAssignedUrls) && alreadyAssignedUrls.length == 1) {
                  if (alreadyAssignedUrls[0].hasOwnProperty('file_name')) {
                     fileName = alreadyAssignedUrls[0].file_name;
                     status = alreadyAssignedUrls[0].status;
```

```
          }
        } else if (Array.isArray(processedAnnotations) && processedAnnotations.length == 1) {
          if (processedAnnotations[0].hasOwnProperty('file_name')) {
            fileName = processedAnnotations[0].file_name;
            status = processedAnnotations[0].status;
          }
        }


        if(status == false){
          annotationJquery('body').prepend('' +
            '<div style="padding: 20px;background-color: #f44336;color: white;">\n' +
            '  <span style="margin-left: 15px;color: white;font-weight: bold;float: right;font-
size: 22px;line-height: 20px;cursor: pointer;transition: 0.3s;"
onclick="this.parentElement.style.display=\'none\';">&times;</span> ' +
            '  <strong>Please Note:</strong> API didn\'t return any data from the knowledge
graph to annotate the web page content semantically.' +
            '</div>' +
            '');
          return;
        }//stop the execution here if status is false


        if (fileName != undefined) {
          annotationJquery('body').prepend('<div data-
apiendpoint="'+annotationServiceEndPoint+'" id="json">' + fileName + '</div>');


            getScript('http://cloud-annotation-
service.a009324a.co.uk/annotationfiles/js/jquery.js',function () {
              getScript('http://cloud-annotation-
service.a009324a.co.uk/annotationfiles/js/jquery.ui.js',function () {
                getScript('http://cloud-annotation-
service.a009324a.co.uk/annotationfiles/js/annotationScript.js',function () {
                  getStyleSheet('http://cloud-annotation-
service.a009324a.co.uk/annotationfiles/css/jquery.ui.css');
                  getStyleSheet('http://cloud-annotation-
service.a009324a.co.uk/annotationfiles/css/annotation.css');
                })
              })
```

```
                    })
                }
            }
        }, error: function () {
        }
    });
  }
  return {
    init: init
  };
}

function getScript(src, callback) {
  var s = document.createElement('script');
  s.src = src;
  s.async = true;
  s.onreadystatechange = s.onload = function () {
    if (!callback.done && (!s.readyState || /loaded|complete/.test(s.readyState))) {
      callback.done = true;
      callback();
    }
  };
  document.querySelector('head').appendChild(s);
}
function getStyleSheet(src) {
  $('head').append('<link rel="stylesheet" type="text/css" href="'+src+'">');
}
function getCurrentPageUrl() {
  return window.location.href;
}
var annotationService_Guest = new AnnotationService_Guest();
annotationService_Guest.init();
```

# Appendix D Sample Load Test System Logs

## D.1 Sample Load Test System Log for Cloud-Driven API

```
{
 "_index": "filebeat-6.5.1-2019.05.29",
 "_type": "doc",
 "_id": "uxyqAWsBhCMhMSt_0Xen",
 "_version": 1,
 "_score": null,
 "_source": {
  "@timestamp": "2019-05-29T03:38:18.456Z",
  "stream": "stdout",
  "prospector": {
   "type": "docker"
  },
  "kubernetes": {
   "pod": {
    "name": "cloudsea-annotation-service-79664bcfcd-mxc9f"
   },
   "node": {
    "name": "ip-172-20-37-165.eu-west-1.compute.internal"
   },
   "container": {
    "name": "cloudsea-annotation-service"
   },
   "namespace": "default",
   "replicaset": {
    "name": "cloudsea-annotation-service-79664bcfcd"
   },
   "labels": {
    "cloudsea": {
     "service": "cloudsea-annotation-service"
    },
    "pod-template-hash": "3522067978"
   }
  },
  "beat": {
   "hostname": "filebeat-zmv76",
   "version": "6.5.1",
   "name": "filebeat-zmv76"
  },
  "meta": {
   "cloud": {
    "availability_zone": "eu-west-1a",
    "instance_id": "i-0e769dde061b109e6",
    "machine_type": "m4.large",
    "provider": "ec2",
    "region": "eu-west-1"
   }
  },
  "source":
"/var/lib/docker/containers/1bfb244e7a52900487ed9751c783bd26e830dbed2d2b8e56210b5f63624fb
252/1bfb244e7a52900487ed9751c783bd26e830dbed2d2b8e56210b5f63624fb252-json.log",
    "offset": 102014,
    "message": "100.96.6.113 - - [29/May/2019:03:38:11 +0000] \"GET /api/annotate/guest.js
HTTP/1.1\" 200 4792 \"-\" \"loader.io;d09a4caf5d1864c78542bdc0f050872a\"",
    "input": {
      "type": "docker"
    },
    "host": {
      "name": "filebeat-zmv76"
    }
  },
  "fields": {
    "@timestamp": [
      "2019-05-29T03:38:18.456Z"
    ]
  },
  "highlight": {
    "kubernetes.container.name": [
      "@kibana-highlighted-field@cloudsea-annotation-service@/kibana-highlighted-field@"
    ],
    "kubernetes.namespace": [
      "@kibana-highlighted-field@default@/kibana-highlighted-field@"
    ],
    "message": [
      "100.96.6.113 - - [29/May/2019:03:38:11 +0000] \"@kibana-highlighted-field@GET@/kibana-
highlighted-field@ /@kibana-highlighted-field@api@/kibana-highlighted-field@/@kibana-
highlighted-field@annotate@/kibana-highlighted-field@/@kibana-highlighted-
field@guest.js@/kibana-highlighted-field@ HTTP/1.1\" @kibana-highlighted-field@200@/kibana-
highlighted-field@ 4792 \"-\" \"loader.io;d09a4caf5d1864c78542bdc0f050872a\""
    ]
  },
  "sort": [
    1559101098456
  ]
}
```

## D.2 Sample Load Test System Log for Cloud-Optimised API

```
{
 "_index": "filebeat-6.5.1-2019.05.29",
 "_type": "doc",
 "_id": "Exf1AGsBhCMhMSt_P_KB",
 "_version": 1,
 "_score": null,
 "_source": {
  "@timestamp": "2019-05-29T00:19:59.064Z",
  "kubernetes": {
   "container": {
    "name": "cloud-optimised"
   },
   "namespace": "default",
   "replicaset": {
    "name": "cloud-optimised-54567c4b8b"
   },
   "labels": {
    "pod-template-hash": "1012370646",
    "cloudsea": {
     "service": "cloud-optimised"
    }
   },
   "pod": {
    "name": "cloud-optimised-54567c4b8b-2wpm8"
   },
   "node": {
    "name": "ip-172-20-46-27.eu-west-1.compute.internal"
   }
  },
  "host": {
   "name": "filebeat-xn5bk"
  },
  "prospector": {
   "type": "docker"
  },
  "input": {
   "type": "docker"
  },
  "beat": {
   "name": "filebeat-xn5bk",
   "hostname": "filebeat-xn5bk",
   "version": "6.5.1"
  },
  "meta": {
   "cloud": {
    "provider": "ec2",
    "instance_id": "i-072a8c0db3f571d26",
    "machine_type": "m4.large",
    "region": "eu-west-1",
    "availability_zone": "eu-west-1a"
   }
  },
    "source":
"/var/lib/docker/containers/30bf23e2553b031eca2548fff7ddec5f7cd0bf7da15fbd3c24ae7f82f04d150c
/30bf23e2553b031eca2548fff7ddec5f7cd0bf7da15fbd3c24ae7f82f04d150c-json.log",
    "offset": 86742,
    "stream": "stdout",
    "message": "100.96.6.113 - - [29/May/2019:00:19:51 +0000] \"GET /annotation/annotate/guest.js
HTTP/1.1\" 200 5399 \"-\" \"loader.io;a7dadffabed9a8454ab584b055652e3f\""
  },
  "fields": {
    "@timestamp": [
      "2019-05-29T00:19:59.064Z"
    ]
  },
  "highlight": {
    "kubernetes.container.name": [
      "@kibana-highlighted-field@cloud-optimised@/kibana-highlighted-field@"
    ],
    "kubernetes.namespace": [
      "@kibana-highlighted-field@default@/kibana-highlighted-field@"
    ]
  },
  "sort": [
    1559089199064
  ]
}
```