Aalto University
School of Science
Master's Programme in Computer, Communication and Information Sciences

Hari Shrestha

A Design Science Research Methodology for Microservice Architecture and System Research

Master's Thesis
Espoo, May 02, 2019

Supervisors: Professor Petri Vuorimaa, Aalto University
Advisors: Christopher Gerlier M.Sc. (Tech.)

Aalto University
School of Science
Master's Programme in Computer, Communication and Information Sciences

Aalto University
School of Science

ABSTRACT OF
MASTER'S THESIS

Author:	Hari Shrestha	
Title:	A Design Science Research Methodology for Microservice Architecture and System Research	
Date:	May 02, 2019	**Pages:** viii + 45
Major:	Software and Service Engineering	**Code:** SCI3043
Supervisors:	Professor Petri Vuorimaa	
Advisors:	Christopher Gerlier M.Sc. (Tech.)	

As enterprise continue their Digital journey, Monolithic architecture approach of building Digital platforms has now proven to be inefficient and obsolete. Architectural paradigms in software development are changing with the spinning of time. The paradigms of architecture, formerly considered sufficiently well architecture and even dominant over the years, are now referred to as monolithic. The demands for fresh technology approaches are continuously evolving to cope the new set of business challenges. [25]

The purpose of this thesis is to evaluate the approach with an experiment in designing a microservice system. The thesis motivates, presents, demonstrates in use, and evaluates a methodology in microservice system for conducting Design Science (DS) research. Moreover, the thesis will go through in detail description of Microservice architecture and enables us to differentiate and find the right Software Development Methodology (SDM) for the Digital platform. SDM enables the proper management of the software development processes, the project team, products and services in terms of cost effectiveness, time, and quality. [23]

The objective of this thesis is to investigate the differences in between architectural paradigms such as monolithic, cloud native and microservice and find the appropriate paradigm that satisfy the enterprises for continuing their digital business. The research of using different platforms and environment for possible improvement on the software development process that enables easy to develop, run and ship distributed application easily and anywhere will be carried out. Similarly, research also focuses on maintaining the development environment consistent, testable and maintainable and hosting the application to the cloud irrespective to underling infrastructure and operation system. The research artifacts will help the enterprises and stakeholders to take an important decision in the selection of the architectural paradigm for their digital platform in advance. The paper concludes that, Microservice architecture is one of the well-known SDM suitable for large enterprise software business application.

Keywords:	Microservices, monolithic, SDM, DS, Architectural paradigms, Digital platform, etc.
Language:	English

Acknowledgements

First of all, I would like to thank my thesis supervisor Prof. Petri Vuorimaa under his supervision and guiding me to complete the thesis work. I would appreciate his efforts in finding a new supervisor when my former supervisor ended his position.

Secondly, I would like to thank Christopher Gerlier M.Sc., my thesis advisor and colleague, for being my advisor and helping me out of the thesis. His advice and suggestions helped me to complete my thesis successfully. I would also like to appreciate his kind efforts for helping me to get the study materials and so on.

Finally, I want to thank my parent, wife and friends for inspiring and comforting me.

Espoo, May 02, 2019

Hari Shrestha

Abbreviations and Acronyms

DS	Design Science
SDM	Software Development Methodology
SOA	Software Oriented Architecture
UI	User Interface
EAR	Enterprise Application Archive
API	Application Program Interface
IDE	Integrated Development Environment
IO	Input and Output
OS	Operating System
CNCF	Cloud Native Computing Foundation
HA	High availability
HTTP	Hypertext Transfer Protocol
IP	Internet Protocol
DNS	Domain Name System
URL	Uniform Resource Locator
DSRM	Design Science Research Methodology
CQRS	Command Query Responsibility Segregation
REST	Representational State Transfer
SSO	Single Sign-on
LDAP	Lightweight Directory Access Protocol

Contents

Chapter 1

Introduction

1.1 Introduction

The research will be based on the Microservice architectural field, where a new approach to software development architecture will be explored and efforts will be made to solve the problems identified by most enterprises using the traditional software development method. The research also focuses on identifying the variations between existing architectural paradigms such as monolithic, distributed systems, Service-Oriented Architecture (SOA) and Microservices. Furthermore, the challenges of moving a large business service towards the Microservice architecture, especially when monolithic microservice application is decomposed, will be examined. Similarly, research will also focus on how these multiple services having specific business function deploying on the server can communicate with each other and what happens if one of the services is down.

In the field of software development enterprises, Microservices is a hot topic that has become increasingly popular in recent years. Applications based on Microservice architecture will be created to demonstrate the concept of Microservices architecture, understanding and knowledge. However, it depends on the problems, availability of time and progress of learning that the application level developed during the research work. At first, the investigation will begin to gain theoretical knowledge and become acquainted with the architecture of Microservices and will attempt to use its principle to understand the software development concept. In the later part, the research output will develop the prototypes of Microservices application, and there will be no limitation on selecting the programming language and framework for implementation and the coding of project work. Finally, research will always focus on the use of the framework and environment to make work faster, more efficient, easier to deploy and maintain.

1.2 Research plan

Recently, the use of microservices architecture in the field of software development industries is getting popular all over the world [23]. The microservice architecture has a good feature over the existing architecture of monolith and SOA. The developer can concentrate on coding using the microservice architecture. Microservice has simplified, facilitated, maintained and made it much quicker to develop software.

However, having multiple microservices and deploying individually on the

servers may have difficulties in communicating among different services. Similarly, the refactoring of the broader business service in several service components could require a lot of redesign and refactoring that is not suitable for the company. In addition, the use of the microservice architecture may not be fully compatible with existing software and components used. The research plan will examine these matters in detail.

Some highlight of planning the research work in order are mentioned below:

a) Getting familiar with different architectural paradigm of software development

b) Investigate how the microservices address the issues related to traditional software development approach

c) Investigate the advantages of decomposing large monolithic system to microservices

d) Investigate on improving the software development process

e) Investigate how the maintain consistent in development environment and work

f) Possibilities of using Docker and OpenShift environment

1.3 Research methods

The design science research methodology of microservice architecture will be used throughout the research period. Besides, the research will also base on reviewing the Scientifics research and articles carried out before and finally, the prototyping application of microservice architecture will be implemented and developed using some higher-level programming language. Microservices architecture involves the refactoring of large enterprise business application into multiple small services that can be run and deployed individually. Each small microservice has got its own specific business function and are independent to others.

1.4 Objectives

The long-term goal of the thesis is to develop the microservices application. The research will be focus on investigating all the possible drawbacks of using microservices architecture and try to find out the solutions to eliminate or minimized those negative aspects of the microservices. Particularly, the study has the following subobjectives:

a) To outline the positive and negative aspects of using microservices architecture

b) To differentiate the architectural design between microservice and existing architectures

c) To find the better approach of eliminating the drawbacks of using microservices

d) To provide the better knowledge of how and where to use the microservices and possibly the risk analysis

The result of this study will be valuable to the enterprise business and industries that involves in software development activities. It provides the better knowledge and understanding of how, when and where to use the microservices and eliminated the business risks that may occurs due to the selection of wrong process and finally, provides the good business value.

Chapter 2

Background

There are many driving forces behinds the evolution and the creation of new software development architecture, technologies tools, and the frameworks, in the field of Enterprise Software business world. Driving forces such as scalability, affordability, availability, maintainability, testability, security, etc. are the major factors that forces in the creation of improved version of the architecture.

2.1 Traditional Software Platform Architecture

The changes in the architectural paradigms happens with the changes on times. Traditional software platform architecture such as multi-tier architectural paradigm was one of the well-known and dominant design patterns that lasted for several years. Even now the pattern is actively used and popular in several enterprises. However, the enterprises are moving towards adopting new paradigms for their new digital platform or extending the existing platform to minimize the limitation of traditional monolithic software platform architecture.

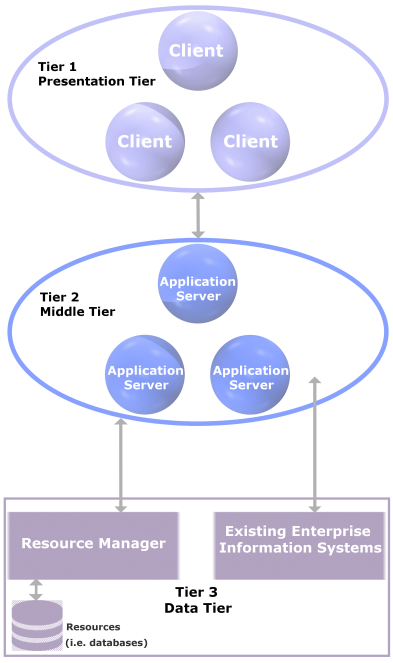

Figure 2.1: Traditional Multi-tier Monolithic Application Architecture [25]

Figure 2.1 depicts the traditional way of three-tier architectural paradigm, which is now considered as a monolithic one. Tier 1 is the Presentation tier or also called user interface layer. It includes the components of the user interface that present different clients rendering all of the web application's UI components and overall flow. Tier 2 is the Business logic layer that contains all necessary business logics for this application and is centralized in the user-interface layer separating it. Similarly, Tier 3 is the Data layer that logically resides in traditional databases and enterprise integration systems. Cross-cutting issues such as logging, surveillance, single sign-on, security, etc. are addressed in all these layers with common tools and functionality. [25]

The whole application is compiled and packed as a single collection of various modules that includes all the components from the multi-tier architecture and its tightly coupled dependencies and libraries in the form of enterprise archive such as EAR. Certainly, the single artifact makes deployment easier because only one executable file can be copied over. However, it makes the lifecycle of development a nightmare. [25] A simple and small change in any part of the multi-tier layer cause a rebuild of the entire executable. Rebuilding multi-tier application is a complex process as most of the time rebuild fails for some module or components and the reason is various and mostly unknown. As a result, the most of the developer's time will spend fixing the issues caused by the changes.

Over the period, the architecture becomes very complex, it inhibits a lot in the case of quick new developer onboarding [25]. Moreover, the features such as continuous Integration and Delivery and exposing API to external world will become ineffective and obsolete. Similarly, developing and maintaining the complex system involves several challenges. Apart from these, running such system also involves several operational challenges. The discussion about the challenges in more details are available under the following section 5.2 and 5.3.

2.2 Monolithic Application

A Monolithic application is an independent, self-contained and single tier-application that runs on single platform with single program. Monolithic application combines all supporting features such as database management, client-side user interface and server- side application into one single executable program. Monolithic application is built and based on monolithic architecture, in which whatever the feature it has or the different functional areas of the application it supports, all resides on single program using single platform [7].

Features such as exposing an API for 3rd parties to consume, supporting various customers including desktop browsers, native mobile applications, drivers for communicating with different tools and handling different types of web communication and much more are all programmed into one application [7]. Thus, the implementation code will be big and cumbersome. Due to the huge amount of code, it is highly possible that even the developing platform

such as Integrated Development Environment (IDE) and the local machine do not support or function properly. The allocated memory on the local machine might not be enough to run the program and needs to increase or more powerful machine. Moreover, it will be very difficult for new comer to understand the code and be productive quickly enough in these dynamic environments. Similarly, it will be hard to maintain the standard code quality and motivation to the developers.

2.3 Monolithic Pros and Cons

Some significant advantages and disadvantages of having all code in the single large repository aka monolithic application are mentioned below. [9]

Pros:

a) Ability to tightly match interdependent changes: Since all code is in the same repository, any changes will automatically update the interface as well as all users of the interface.

b) Simplified repository management: some common repository maintenance tasks are easier to handle than multiple repositories. Because of a single repository, for example, backup planning and writing triggers are easy to perform. [9]

c) Easy Branching and Tagging: Having single application means there's only one repository, making the branching and tagging easy.

d) Straightforward Code management: Everything requires for dealing with code and its requirements are in a same place and having it in a same place makes the work better. Some actions such as writing scripts, moving files around, searching across files, etc. benefit from a single global repository. [9]

e) Less Operational Overhead: Having a single application means there's only one application that needs to set up logging, monitoring and testing for. Deployment can be carried out with less complex. [9]

Cons:

a) Tightly Coupled: As the application evolves, the monolithic application is closely linked and entangled. As a result, services are difficult to isolate for purposes like independent scaling or code maintenance [9].

b) Harder to Understand: It is much harder to understand specially for the newcomers because the application size is usually large and complex. Besides, multiple dependencies and its side-effects add the complexity [9].

c) Security and Access control Problems: In traditional monolithic structures, restricting sections of the source securely becomes very difficult or impossible [4].

2.4 Microservices

Microservices architecture is a way to develop applications in the style of software or architecture that structure an application as a collection of loosely coupled services, where each service can be deployed independently, modularly and smallly, implementing business capabilities. [23]] Each service carries out a unique process and communicates with a well-defined, lightweight mechanic to serve a business goal. [6]

Microservice is a SOA architectural style variant designed to minimize existing limitations on architecture-based monolithic and other applications. And, improving the development of software using standard frameworks and technologies. Microservices break down the large monolithic application into various smaller services that enable or enhance the application's modularity and make it easier to understand, develop and test the application. Each small service has its own database and acts as a componentize service with specific business logic and function that can be independently deployed into the server. Decomposing the application into smaller individual componentize service enables the service to select any suitable technology and frameworks and deploy individually into the server. [23]

Microservices architecture do not restrict on the selection of technologies and the frameworks. Developers can select any emerging technology, platform and the frameworks on software development per their suite and skills. Thus, one service can be programmed using C-Sharp programming language using .Net framework, while another service can use JavaScript with node.js framework. Similarly, as each service can be independently developed, deployed and scale its respective services, parallel development is possible using microservices where each service can be fully controlled by small autonomous teams, allowing for continuous delivery and deployment.

2.4.1 Characteristics of Microservice architecture

The main characteristics of microservices are:

a) Flexibility: A service or system is flexible enough to supports all the necessary features required to remain business competitive on the dynamic business environment.

b) Modularity: Each service is independent and concentrates on some business functions which contribute towards the overall system behavior rather than the full functionality of a single service. [19]

c) Evolution: It is possible to add new features to each service and maintainable. The service should not affect to the other service even the service is down. [19]

2.4.2 Benefits of Microservices

The benefits of using Microservice architecture during software development are significantly stronger over the existing monolithic architecture. Because of its significant features and benefits, some big enterprises such as Amazon, eBay, and Netflix are using the microservices architecture for its enterprise business. Some of the key benefits discovered so far are mentioned below:

a) Scalability: Scaling can be done only to those services that needs scaling without affecting the rest of the services. Unlike large, monolithic service, smaller service can be run on smaller, less powerful hardware. [2]

b) Strong module boundaries: Each service is small and focuses on single business capability. Service boundaries become the obvious bulkhead, which is the key concept in resilience engineering. Unlike monolithic system, if any components fail, the whole system will stop. However, in microservices, if one service fails and will not affect the rest or the whole system, that service can be isolated, and the rest of the system can continue the work.

c) Independent development and deployment: Each service can be developed and deployed independently to any server.

d) Technology diversity: Unlike monolithic service, microservice facilitates the developer to select the technology, tools and frameworks based on their skills and wills. It will not limit to single programming language or frameworks.

e) Simplicity and maintainability: Microservice application is small and focus to the single business function. Since it is small, it easier for a new developer to understand the functionality of a service and maintain the bug fix.

f) Decentralized data management: Each service has its own database and effective data management can be easily carried out using microservice. [12]

g) Composability: Microservices gives the opportunities for reuse of functionality as different ways as possible for different purposes. Reusability of functionality is the key promises of distributed system and service-oriented architecture. [9]

2.4.3 Drawbacks of Microservices

a) The Complexity of a Distributed System: Each service is now an independent service that needs to carefully handle the request travelling between the modules. Complexity can be increased due to network latency on remote calls, fault tolerance, versioning, message serialization, asynchronicity, etc. [2]

b) Database management complexity: Multiple databases residing on different services and platform can created difficulties on transaction management. Retrieving data owned by multiple service can be challenging [9].

c) Difficulties on Deployment: Unlike Monolithic service, microservices having multiple services can have complexity during deployment. Each service needs to deploy individually and may be on different server.

d) Testing: Before starting testing, each dependent service must be functional. Testing can be difficult or impossible unless all the dependent service is fully functional and available [9].

e) Eventual Consistency: One key limitation existing on distribution system is to maintain strong consistency [15]. Unlike other distributed system and SOA, issues on maintaining strong consistency exists on microservices to some extent. However, microservices uses event-driven methodology to minimize the issue of consistency.

2.4.4 Adopting Microservices by Enterprises

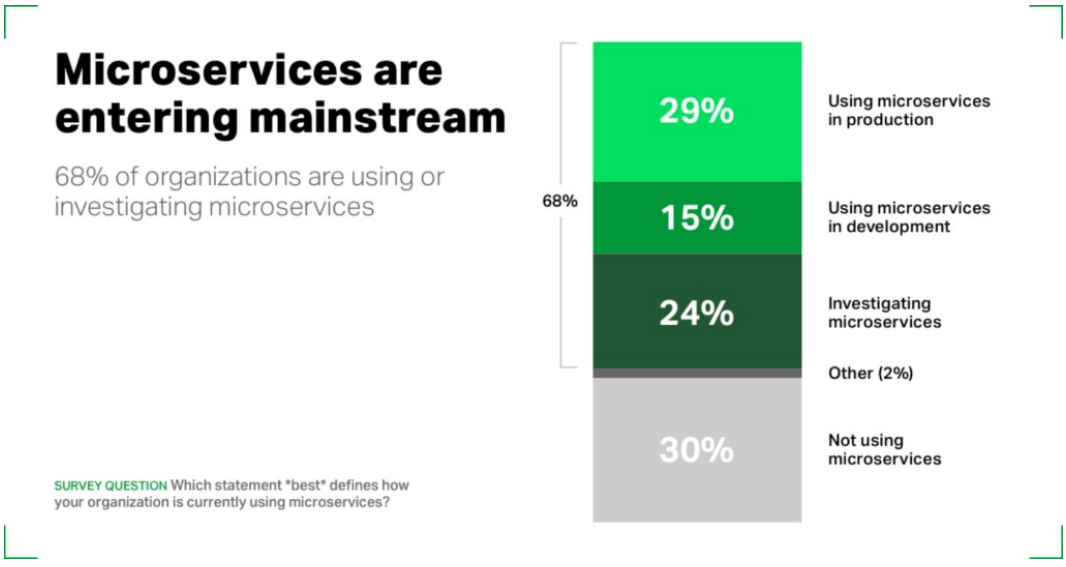

Figure 2.2: Uses of microservices by enterprises in 2015 [18]

According to the NGINX App Development Survey 2015, the percentage of organizations using or investigating microservices were 68 percent. Figure 2.2 below gives the details about the survey result. According to LeanIX Microservices Survey 2017, the number of percentages increased to 80 percent in only 2 years and most of these organization has plan to intensify the usage of microservices in coming days. [18]

Chapter 3

Development Environment

During software development process, it is vital to have consistency in the development environment and platforms used by the teams to make the development work smooth and efficient. Any differences in the development environment and platform used between the developers may create different issues related to the environment and fixing those issues will ultimately takes longer time and decreases the productivity of individual developers and team.

3.1 Docker

Docker is an operating system-level virtualization container management service, also known as "containerization." It was developed by Docker, Inc. and released in 2013 for the first time [26]. By providing an additional layer of abstraction and running on the host operating system, it automates the deployment of software applications within containers. The keywords of Docker are develop, ship and run distributed applications easily and anywhere [26]. It allows users to package an application in a standardized software development unit with all its dependencies. It also provides the strongest isolation capabilities to secure the application running in containers. As a result, the applications are portable, lightweight and secure. It is recommended to visit the official page of docker www.docker.com for more information related to the docker.

Some of the important features of the docker are mentioned below: [14]

a) Docker has the ability to reduce the development size by providing the operating system with a smaller footprint via containers.

b) With containers, teams across different units, such as development, QA and Operations, will find it easier to work seamlessly across applications.

c) Docker containers can be deployed anywhere, on any physical, virtual or cloud machine.

d) Lightweight Docker containers make them very easy to scale.

3.1.1 Docker Architecture and Its Terminology

Figure 3.1 shows the docker architecture and components. There are certain keywords and terms that are related to Dockers. Some of its terminology is mentioned below.

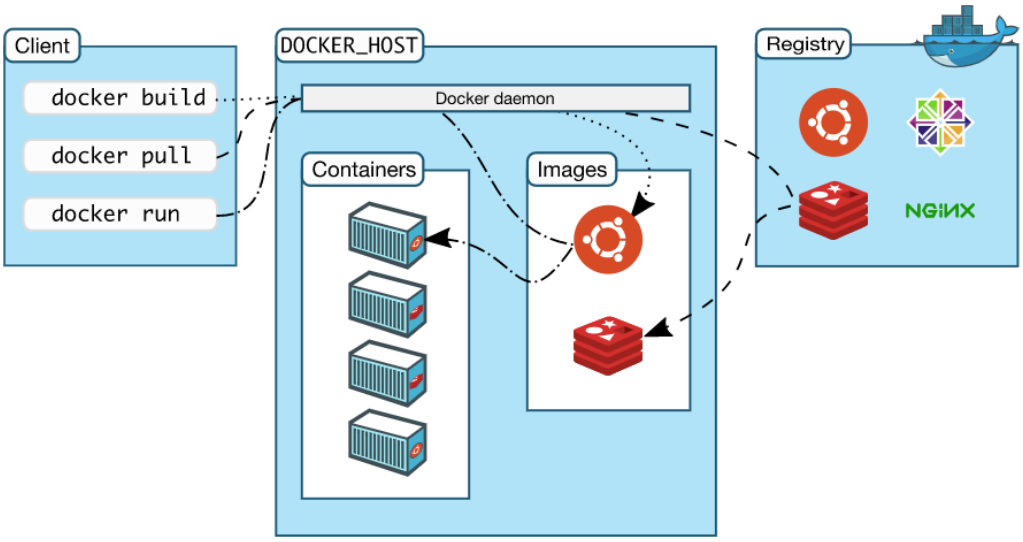

Figure 3.1: Docker Architecture [14]

a) **Images:** This is the application's blueprints that form the container base. The list of commands is written in Dockerfile. There are certain rules of writing scripts in the Dockerfile. Command such as docker pull and docker build are used to download the image from Docker Hub and create docker image.

b) **Containers:** Docker images are deployed and run in containers. It is created from images from Docker and runs the application itself.

c) **Docker Daemon:** The background service running on the host that manages Docker containers to be built, run and distributed. It is the process to which the client speaks in the operating system.

d) **Docker Client:** It is a tool command allowing the user to interact with the daemon. There are different form of clients and some client such as Kinematic, provides powerful GUI to the users through which containers can be run simply. Docker client simply calls the commands written in the Dockerfile while creating an image.

e) **Docker Hub:** A registry of Docker images. It is the directory of all Docker images that are available. If necessary, you can host your own Docker registry and use it to pull images. It is the world's largest library and community for container images.

3.1.2 Containers

Container is a standard software unit that provides a logical packaging mechanism, in which the application is packed in one package together with code

and all dependence, allowing the application to operate quickly and reliably, regardless of the target computing environment. It combines the application and its dependencies into one succinct manifest (image) that can be controlled for version, making it easier to replicate the application across cluster developers and machines. [13]

Unlike virtual machines virtualizing the hardware stack, operating system level containers virtualize, and multiple containers can run directly at the top of the OS kernel. Because multiple containers can run and share the OS kernel directly, the containers are much lighter compared to approaching virtual machines. Application running in containers can be started much faster and uses a fraction of the memory compared to running on a whole OS. Because of low overhead, containers make the underlying operating system and infrastructures more efficient to use. [13]

3.1.3 Why Containers?

Containers work best for architectures based on service. Unlike Monolithic architectures, where each application piece is intertwined – from IO to data processing to rendering – service-base architectures divide these into separate independent components. Separation and division of labor enables the services to continue to operate even if others fail, making the entire application system more reliable.

During the software development process, applications are programmed and developed using specific version of libraries, tools, dependencies and 3rd party libraries. In order to run the software application properly and fully functionally based on the designed, the supporting environments should have all those libraries and dependencies with required version. There are high chances that the supporting software environment is not identical to the development environment due to which the application fails to run, and problem arise. [16] One simple example can be Python version. The development environment uses Python 2.7 version and in the production environment or replica environment, Python 3 is running, as a result something weird will happen. In addition to the software version, there may be several reasons why the software fails to run such as network topology differences, or security policies and storage.

To solve the problems, a container consists of a complete runtime environment that satisfies the application to run without any problems due to the difference in supporting software environment. The entire runtime environment includes a downloaded and bundled application, its dependencies and libraries as well as other binaries, configuration files and even 3rd-party libraries. Differences in OS distributions and infrastructure underlying the platform and its dependency are overridden by containerizing.

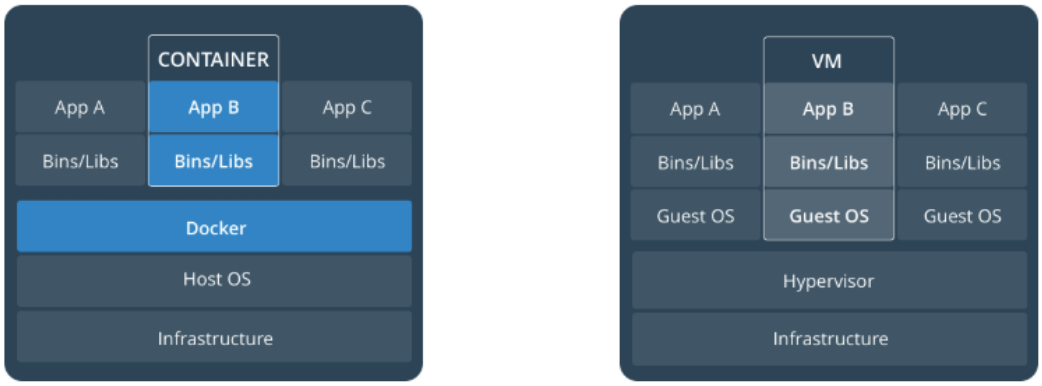

Figure 3.2: Docker container and VMs architecture [14]

3.1.4 Docker Containers Instead of VMs

Figure 3.2 shows the architectural differences between Docker container and Virtual machines (VMs). A Linux-born container shares the host machine's kernel with other containers. However, these are different in terms of windows-based containers such as Windows Hyper-V container, which does not share kernels of the host machine with other containers. [8] Linux based containers are lightweight since it runs a discreate process, which takes less memory than any executables. On the other hand, through a hypervisor, a VM runs a complete "guest" operating system with virtual access to host resources. As a result, VMs are providing the most application-needed environment with more resources.

3.1.5 Docker Images and Docker Container

Using Dockerfile, the Docker image is created. The Dockerfile contains a list of commands to carry out certain tasks. To create the image from Dockerfile, the Docker build command is used. Similarly, the command Docker run is used to run the image in the container. More information on Dockerfile and Docker commands can be found in the following paragraphs.

Dockerfile: A Dockerfile is a simple text file containing the list of commands called by the Docker client when creating an image. It is an easy way to automate the process of creating the image. The way of writing commands in the Dockerfile is identical to the equivalent Linux commands. The following are some simple examples of keywords or instructions for writing commands in the Dockerfile: ENV, RUN, ADD, COPY, FROM, LABEL, EXPOSE, STOPSIGNAL, USER, VOLUME, and WORKDIR [14]. These commands defined in the Dockerfile will download and install certain version of Node and nvm:

```
# nvm environment variables
ENV NVM_DIR /usr/local/nvm
ENV NODE_VERSION 7.7.0
```

```
# install npm and node
RUN source $NVM_DIR/nvm.sh \
&& nvm install $NODE_VERSION \
&& nvm alias default $NODE_VERSION \
&& nvm use default
```

The entire Dockerfile example can be found in Appendix 1.

Docker Build: Command such as 'docker build -t username.' is used to build the image from Dockerfile. -t refers to the tag and username refers to the tag name, which will eventually become image name once the command is executed successfully. The space and dot (.) symbols after tag name is necessary at the end of the command.

Docker Run: Command such as 'docker run -it username' is used to run the image in the container. The docker container will be automatically created once the command is executed. Some Command are given below: Find a list of all images: docker images -a Find a list of all containers: docker ps -a or docker container ls -a Remove image: docker image rm image1, image2, — Remove container: docker container rm container1 container2 –. [26]

Difficulties:

a) Windows Docker can only host Windows applications within Docker containers, and only Linux apps are supported by Docker on Linux.

b) Linux based docker container cannot access directly to the Windows Host USB devices.

c) Adding CA certificated as a part of docker image was not straight forward.

d) Windows support is being developed for orchestrators like Kubernetes and Apache Mesos.

Also, worth mentioning is that Docker is the only major container platform compatible with Windows at the moment. It underlines the fact that, for now, the Windows container ecosystem is much smaller than the Linux container world.

3.2 Kubernetes

Kubernetes has been developed by the Cloud Native Computing Foundation (CNCF) and is a container management system hosted by Google open source. It is used for management of containerized applications in various environments, including physical, virtual and cloud infrastructure. [27] It helps to create and manage application containerization and has the ability to automate cluster-wide deployment, maintenance, application scaling, and application container operations. It also has the ability to build container-centered infrastructure and helps move host-centered infrastructure to container-centered infrastructure.

A cluster of Kubernetes is made up of one or more masters and a set of nodes. High Availability (HA) masters can be configured to ensure that there is no single point of failure in the cluster. The master can be built so that it controls all the nodes and deploys the containers to all the nodes. Its main function is to control the OpenShift cluster and deployment flow using different kinds of configuration file. [27] Some of Kubernetes ' important features are listed below:

a) It contributes to environmental consistency through development and production testing

b) It allows further development, integration and implementation

c) It helps to move from host to container-centered infrastructure

d) Auto-scalable and containerzed infrastructure

e) Capality to run applications on cloud

f) Loosely coupled facilities, in which every part can function as a separate unit

g) Increased resource use density

h) Predictable infrastructure to be developed [27]

3.3 OpenShift Environment

OpenShift is an open source development platform developed by Red Hat Enterprise Linux. It provides a cloud development Platform as a Service (PaaS) and enables developing and implementing cloud-enabled services and their cloud infrastructure application. [28] Developing the cloud-enabled services and the applications is simpler and easier with the help of OpenShift environment. As a result, organizations can move their traditional application infrastructure and platform from physical and virtual to cloud media easily [3].

OpenShift environment is a hosting platform for cloud applications and applications that provides advanced features such as automating application provisioning, management and scaling. The developers can therefore concentrate on writing the business code and logic. Further, it also makes easy for developers and users to quickly build, launch, and scale container-based web apps in a cloud environment. OpenShift online provides the cloud-enabled services and the applications to be use, developed, build, launch and scale in a public cloud environment.

Figure 3.3 show the architecture of OpenShift container platform. OpenShift is a layered system that uses Kubernetes and Docker cluster to tightly bind each layer to another layer. The architecture is designed to support and

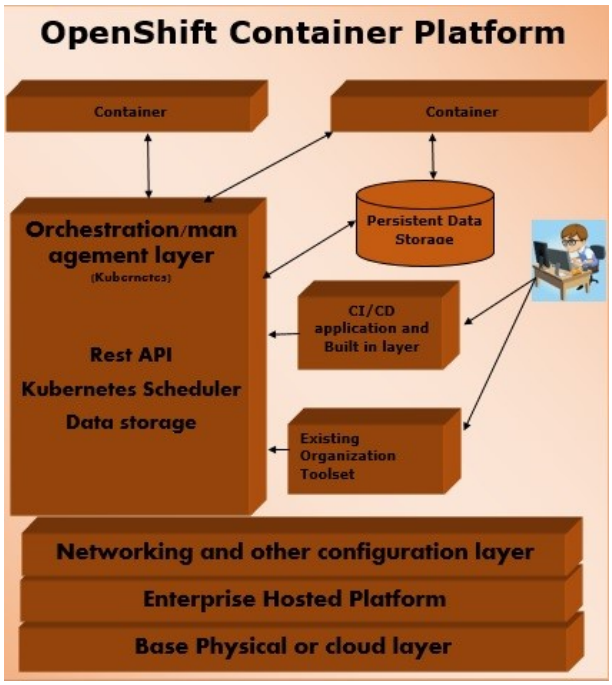

Figure 3.3: OpenShift Container Platform and its architecture [28]

manage containers from Docker hosting Kubernetes on top of all layers. The containerized infrastructure is only supported by the new version of Openshift V3. In this model, Docker helps to create lightweight Linux-based containers, and Kubernetes supports the task of multiple host container orchestration and management. [28]

3.3.1 OpenShift Platform Supports

OpenShift supports almost all kinds of applications and makes easy for developing and deploying the application on cloud platform. It supports three types of developer and user platforms, including Infrastructure as a service (IaaS), Software as a Service (SaaS), and Platform as a Service (PaaS) [28]. The short description of each platform and services are given below.

Infrastructure as a Service (IaaS) Service provider provides some predefined virtual hardware configuration for virtual machines at the hardware level. In the long run, service providers are still responsible for infrastructure management such as operating system installation and maintenance, server packages, networking, and basic system administration.

Software as a Service (SaaS) Users don't have to worry about the underlying infrastructure like IaaS with this service. It is as simple as the service plug and play that the user can use it right after logging in. Service provider provides the software with limited customizable by the users. As a result, only the minimum amount of customization allowed by the service provider can be

carried out by the user.

Platform as a Service (PaaS) Also known as PaaS is a middle layer between IaaS and SaaS with a primary goal for developers to spin the development environment with few commands. Service provider provides the development environment that satisfy all the development needs, such as web application server with a database. Users and developers need a minimum effort of few commands and much of the work is automatically done by the service provider. With OpenShift, the developer of PaaS has the freedom to design specifications for their required environment.

3.3.2 Why OpenShift Environment?

OpenShift environment provides a common platform for the business unit to develop, deploy and host cloud-based applications and container-based applications without worrying about the underlying infrastructure and operating system. In short, containers are the standalone processes independently runs within their own environment, and independent of underlying operating system and the infrastructure. The more details about the container will be mentioned in the section 6.2.1. This facilitates the applications to use, develop, and deploy easily on cloud. It also provides a self-service platform for all kinds of development and testing with managed hardware and network resources, enabling faster development and life cycle release. It enables the PaaS user and developer to have freedom to design the required environment with specification.

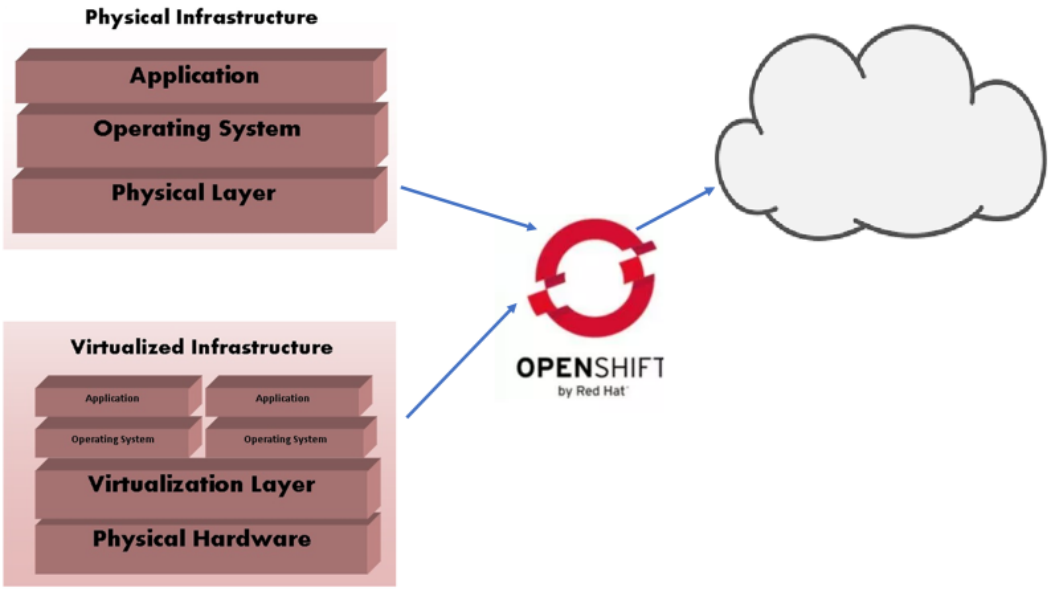

Figure 3.4: OpenShift Environment

Figure 3.4 shows the efforts of OpenShift that enables the enterprises to switch their traditional application infrastructure and platform from physical

and virtual to cloud.

3.3.3 OpenShift Environment Features

Figure 3.5 shows the list of features suported by OpenShift environment. Apart from these list of features, Openshit environment also helps in load balancing of the deployed services. [21] For example, multiple pods can be use to create and run the instance os services. If service running in one pod fails or down, the service running in rest of the pods are still functioning and make the service available for external world making the system fully functional, high availablity and fault tolerance.

- Multiple Language Support
- Multiple Database Support
- Extensible Cartridge System
- Source Code Version Management
- One-Click Deployment
- Multi Environment Support
- Standardized Developers' workflow
- Dependency and Build Management
- Automatic Application Scaling
- Responsive Web Console
- Rich Command-line Toolset
- Remote SSH Login to Applications
- Rest API Support
- Self-service on Demand Application Stack
- Built-in Database Services
- Continuous Integration and Release Management
- IDE Integration
- Remote Debugging of Applications

Figure 3.5: Openshift Environment Features [28]

3.3.4 OpenShift Basic Concept and Its Overview

Some basic terms and concepts used in OpenShift Container Platform are necessary to understand before the actual setup and deployment of the application. The following topics provide architectural high-level information on key concepts and objects that you will find when using OpenShift Container Platform. Many of these objects come from Kubernetes, which is expanded to provide a more feature-rich lifecycle development platform by OpenShift Container Platform.

Containers and images: Containers and images are OpenShift's building blocks made up of from Docker images [10]. The cluster has its own images running inside each pod on OpenShift. The field is pooled from the registry during the pod configuration and the configuration file pulls the image and deploys it in the cluster mode. Containers are created when the image of the Docker is deployed on the cluster of OpenShift. The configuration file defines the container section in which a container has several images inside it and the OpenShift Kubernetes manages all the containers running on the cluster node. [28] Example of Configuration file defining images and containers are shown in Figure 3.6 [28]:

```
apiVersion: v1
kind: pod
metadata:
   name: Tesing_for_Image_pull -----------> Name of Pod
      spec:
containers:
- name: neo4j-server ------------------------> Name of the image
image: <Name of the Docker image>----------> Image to be pulled
imagePullPolicy: Always ------------->Image pull policy
command: [“echo”, “SUCCESS”] -------------------> Massage after image pull
```

Figure 3.6: Configuration file defining the images and containers [28]

Pods and services: It allow containers to communicate between themselves and to proxy connections [10]. Pod is a container collection and its storage within an Openshift cluster node. There are generally two types of pods starting from a single pod and multi-container pod. Service can be defined as a logical set of pods that resides on top of the pod as an abstract layer. It provides a single name for IP and DNS that allows access to pods. Service helps manage the configuration of the load balancing and very easily scale the pod. [28]

Projects and users: It provide the space and means for communities to organize and manage their content together [10].

Builds and image streams: It allows us to create images that work and react to new images [10]. Build is a process, in which images are transformed

into containers. Build process works on a predefined image source code building strategy. Image stream is created after the images have been pulled. And it is looking for an update on an image's new version. [28]

Deployments: It adds expanded support to the lifecycle of software development and deployment.

Routes: It exposes the service to the world, by creating and configuring externally accessible hostname. Routes are created in Openshift using routers deployed on the cluster by the OpenShift administrator. Routers are used to bind external application ports to HTTP (80) and https (443). [28]

Templates: It allow authorized users to simultaneously create multiple objects based on custom parameters [10].

3.4 Minishift

Minishift is a tool that helps in running OpenShift environment locally by running a single-node OpenShift cluster inside a virtual machine. It starts virtual machine on the local computer and creates an OpenShift cluster inside it. As a result, it enables to try and test a fully featured OpenShift cluster on the local machine. This is very important for developers from the development point of view as it provides fully featured OpenShift environment for local application development, deployment, testing and scaling. [20]

3.4.1 Minishift architecture

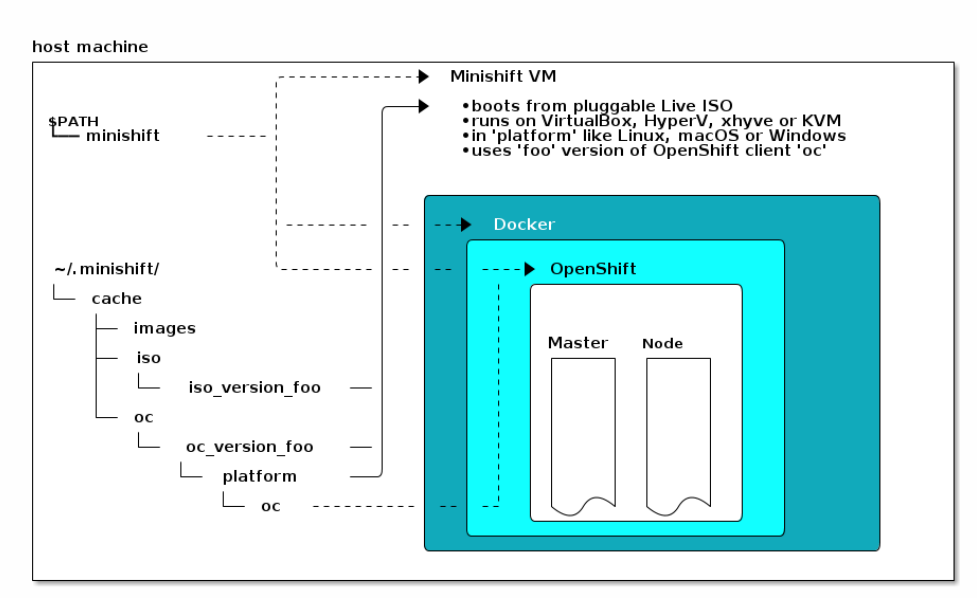

Figure 3.7: Minishift architecture [20]

Figure 3.7 describes Minishift's architecture describing the following components: the Minishift VM, the Docker daemon running on the VM, and the OpenShift cluster running on the daemon. [20] For easy execution, the Minishift binary used to start, stop, and remove the Minishift VM is placed on the PATH. A pluggable Live ISO is bootstrapped by the VM itself. Minishift commands interact with both the Docker daemon and OpenShift cluster. Once the cluster of OpenShift is up and running, the cluster is interacted with oc binary. cc is an alias for OpenShift command-line interface (cli). Binary are cached by Minishift under $MINISHIFT_HOME (per default ~/. minishift). The caching enables to speed up the provisioning of the OpenShift cluster and to minimize network traffic. [20]

3.4.2 Minishift Life-cycle

There are basically three main commands that control and manages the Minishift life-cycle [20]. The commands are minishift start, minishift stop and minishift delete. The starting command Minishift creates and configures Minishift VM and provides the Minishift VM with a local OpenShift cluster with signal node. The command also copies the oc binary to the host that allows the cluster to interact with OpenShift via the oc command line tool or the Web console. The web console access URL is provided in the start command output.

It also translates the oc binary to the host, which allows the cluster to interact via the OC commandline or the Web console with OpenShift [20]. The web console access URL is provided in the start command output. On restarting the Minishift with start command and with correct parameters, the OpenShift cluster starts and run in previous state, allowing to continue working from the last session. On the other hand, the Minishift remove command removes the OpenShift cluster and closes the Minishift VM. The delete command also deletes the Minishift VM and the cluster or state of OpenShift.

Chapter 4

Discussion

Microservices has many benefits than drawbacks. Most of the drawbacks are related to existing distributed system and SOA. However, microservices uses the different techniques to minimize those limitations. The use of Event-Driven architecture to eliminate the issues related to eventual data consistency is an example how the microservices works. However, there are several issues related to how and when to use microservice architecture. Microservice architecture cannot be efficient everywhere. The subsection 6.1 to 6.4 provides good information when to use and how to minimize the drawbacks of the microservices.

4.1 When to use the microservice architecture?

Selecting architecture is one of the most challenging and costly tasks during the software development process. Failure in right selection of architecture leads the chances of failure in software project. Different factors such as system size, scalability, security, testability, availability, computing resources, man power, supporting tools and framework, existing domain, end-users, etc. are needs to be carefully considered during the selection of architecture. If the system develop on software project is simple, small size that needs only few developers to implement and do not need the advance features such as scalability, monolithic architecture can be better architecture for software development process. Microservice architecture is suitable for large enterprise distributed business application where the features such as scalability and maintainability are very important for the business operation. [7]

4.2 How to decompose the application into services?

Decomposing the monolithic application into multiple services is quite challenging [25]. Following strategies can help on deciding how to split the monolithic application into microservices:

a) Identify and define services corresponding to the business capability.

b) Identify the domain-driven design and define services by subdomains and

c) Decompose by use case or resources and define services that are responsible for certain actions [25]. For example: Shipping service that handles all the transaction of shipping orders.

4.3 How to maintain data consistency?

Maintaining strong data consistency is always challenging on distributed system and SOA based service. The application is structured by Microservice architecture as a loosely coupled collection of services. Each service has its own database, which ensures loose connection. To maintain reliable data consistency, microservices should use an Event-Driven architecture where the updating the data between different service happened with one service publishes the event on data changes and another service consumes the events and updates accordingly. [7]

4.4 How to implement queries?

When system depends on multiple services, retrieving data owned by multiple service will be challenging. Command Query Responsibility Segregation (CQRS) methods can be used as a common solution to cope these challenges. The CQRS divides the application into two sections: the command-and query-side. The control handles request creation, updating, deletion and publishing in case of data change while the query handles queries with an up-to-date, materialized view. [7]

4.5 What is the appropriate size of the microservices?

The idea of designing microservice is to be small, autonomy, and focus to the specific business capabilities. Some degree of overhead and the fallacies of distributed computing inevitably involves in services [18]. There is always a change that the services that are decompose are not fully autonomous. Moreover, Services brings cost in terms of maintenance and performance and if services are too small then there is a risk of anti-pattern where the cost of a service outweighs its utility [18]. It is obvious that defining the boundaries between the services are challenging and crucial. Good knowledge of service design and domain business can be used defining the service boundaries. The service should be autonomous and loosely-coupled where autonomy is much more important than size. In addition, service should be defined with proper boundaries and appropriate size. A single deployable service should be no bigger than a bounded context, but no smaller than a cohesive unit [18].

4.6 What is the drawback of having small and big size of microservices?

Identifying the right size of the microservices are hard to define as it always depends on the nature of the problem domains [17]. However, based on the microservices definition, it is described in terms of single responsibility principle as it should do one thing well. The service should contain a fully functional that independently serves business logics. If the service size is small, we might need more services to build a complete system. Having too many microservices increases the complexity in handling and maintain the services.

On the other hand, if the size of microservice is too big, it will be no more microservice application instead a monolithic one. There will be no advantage of microservice architecture-based software development practice. The service should contain single business logic. Hence, the term defining the right size of microservices has some ambiguity as it is difficult to size anything as abstract as a service. [17]

Theoretically, the microservice should contain a fully functional business logic and independently capable to provide the service. The boundaries of the business logic should be limited and appropriate one. Adding too many function and feature makes the size of the service too big and hence needs to be refactored. Bigger size means bigger complexity in software development activities such as building, developing, testing, deploying, handling and maintaining the services.

4.7 What happened if there are too many microservices?

There can be several issues such as performance, optimization, synchronization, security etc. during communication among many microservices. Microservice uses standard lightweight simple (REST) API to communicate each other. If one service depends on the result of another service or the dependent service fails to execute then another service will not function properly. Besides, calling huge number of rest API sequentially reduces the performance. Increase in the number of services that interact each other to provide a complete business service will increase in the complexity to handle and maintain the microservices. As a result, the complexity of a microservices based application is directly correlated with the number of services involved and communicated.

Chapter 5

Design Science in Information Systems Research and Guidelines

Design science research is conducted to solve problems identified in the organization. It is based on a rigorous design process to solve the problem observed, to contribute to research, to evaluate designs and to communicate the results to the relevant public. [[22], p. 49] Artifacts include any objects designed for the purpose of identifying research issues, such as constructions, models, methods, instantiations or social innovations, or new features of technical, social or information resources. [[22], p. 49]

Table 1. Design-Science Research Guidelines

Guideline	Description
Guideline 1: Design as an Artifact	Design-science research must produce a viable artifact in the form of a construct, a model, a method, or an instantiation.
Guideline 2: Problem Relevance	The objective of design-science research is to develop technology-based solutions to important and relevant business problems.
Guideline 3: Design Evaluation	The utility, quality, and efficacy of a design artifact must be rigorously demonstrated via well-executed evaluation methods.
Guideline 4: Research Contributions	Effective design-science research must provide clear and verifiable contributions in the areas of the design artifact, design foundations, and/or design methodologies.
Guideline 5: Research Rigor	Design-science research relies upon the application of rigorous methods in both the construction and evaluation of the design artifact.
Guideline 6: Design as a Search Process	The search for an effective artifact requires utilizing available means to reach desired ends while satisfying laws in the problem environment.
Guideline 7: Communication of Research	Design-science research must be presented effectively both to technology-oriented as well as management-oriented audiences.

Table 5.1: Design Science Research Guideline [[1], p. 83]

DS Research is practically governed in the IS discipline by 7 guidelines describing features of well carried out research. Table 5.1 highlights the seven guidelines for research into design science, the most important being that research produces an ' artefact created to solve a problem. ' In addition, the

artifact should be relevant to the ' unresolved and significant business problem ' solution. It must be rigorously evaluated for its usefulness, quality and efficiency. In both building and assessment of the design artifact, verifiable contributions and rigor must be applied. The artifact must search for a defined problem using the existing theories and knowledge. Finally, the research needs to be effectively presented to suitable audiences. [[22], p. 49]

In this thesis, DS research objects like prototyping systems are designated and developed according to the architecture of microservices. During the software development of prototype application, the guidelines mentioned in the Table 5.1 will be followed. The development of software activities is thoroughly monitored and the findings are used for evaluating the Design Science Research Methodology (DSRM) and DS process. In chapter 5, DSRM will be discussed more in details. The results and the findings are presented to the relevence parties in order to demonstrate the concept of the architecture of the microservice. The relevent stakeholders make the correct decision to use the architectural paradigms for the microservice on the software development platform based on the findings and result artifacts.

Chapter 6

Design: Development of the Methodology

A system of principles, practices and procedures is the methodology for a particular field of knowledge. Methodological development requires the development of a DSRM process. [[22], p. 52] The DS-researchers will be researched to carry out the design of a DSRM process in a significant past research and current thinking to determine the appropriate element and the instructions. The main objectives are to develop an approach to conduct research based on the principles of DS research defined within section [8] that are commonly accepted. The main objectives are the development of a methodology. With the aid of such a methodology, researchers can produce and present high-quality, valuable, rigorous and published DS research into information systems (IS). The DS Research methodology includes three elements: the conceptual principles of the DS Research Definition, DS Research Practice Rule, and the research execution and presentation process. [[22], p. 49]

In this DS research, a consensus building approach will be used to produce the design to ensure that the DSRM-based element is well accepted. [[22], p. 52] Many IS researchers and other disciplines provided ideas for elements of the process. As shown in several representative papers and presentations and our synthesis, the process components of the DSRM process are illustrated in Table 1. The authors substantially agree on common elements. Our synthesis results in a process model consisting of six nominal activities that we describe in the following six activities graphically [[22], p. 52-56]:

1. Problem identification and motivation

2. Define the objectives for a solution

3. Design and development

4. Demonstration

5. Evaluation

6. Communication

6.1 Project Description

Building the software applications system that can be easily develop, deploy and run in OpenShift environment. The size of the software application de-

veloped should not be too big and it is very important that the system is very simple to use, develop, and maintain so that newcomer and even the non-technical person can understand the functionality and the features with less efforts. Further, the software application should be complex and if necessary, system should be split into componentize services. Each service has a certain business capability and is fully functional and exposable to the external world. The different teams with different technology stacks may develop each service independently.

The system takes the .zip file that contains the list of screenshot images taken from the Smartphone. As an output, it generates the word document reports containing all those screenshots in order. The end user has a right to select how many screenshot images the report can have in a row or page.

6.2 Design- and Development-Centered Approach

Activity 3 starts with an approach to design and development. The result would be an artifact that was not formally regarded as a solution to the explicit problem domain in which it is being used. Such an artifact could have come from another research field, it could already have been used to solve another problem, or an analog idea. [[22], p.64]

6.3 Problem identification and Motivation

Modern software development architectural approach is necessary to carry out the software development work successfully. From the requirements, the size of the software application should not be large and complex, and the complex system should split into componentize services. Beside these, the development work task should be dividable among the teams where each service can be developed based on the team expertise and their familiar technology stack and platform. In order to fulfill these requirements, it is not possible to use the old monolithic architectural way of development. Instead, microservice architecture is used. The complex system decomposed into multiple componentize services with fully functional business capability. Those services combined served the business goals. Each service is small, independent and fully functional having certain business capability; hence the development, testing, maintenance and deployment work should be easily carried out. [[22], p. 64]

6.4 Objectives of the Solution

The solution's primary goal is to develop a distributed system-based software application using Microservice. First, researchers needed to find out, how to decompose large system into multiple componentize service with business capability. Second, the appropriate size of the service needs to be identified in

advance. Also, the service needs to be loosely coupled and independent to other services so that it can be use, develop, deploy, and testing and maintaining independently. As a result, the deployment and monitoring of the service can be carried out easily and the system application keeps on running even though some service gets down. Third, why old traditional monolithic way of software development is not appropriate for the software development. It helps in differentiating the architectural design between microservice and existing architectures. Finally, it will provide the better understanding of how and where to use the microservices and possible the risk analysis. Furthermore, it helps in finding the better approach of eliminating the limitations of using microservices. [[22], p. 64]

6.5 Design and Development

The design and development process is based on a research project of the information system. The requirement is collected at very first stage and continues with the involvement of diverse set of penitential end users. The requirement documents are later used for designing the system architecture. Software was developed using a standard software development process, allowing multiple groups of people to work parallel, to provide a proof-of-concept and fully functioning client application.

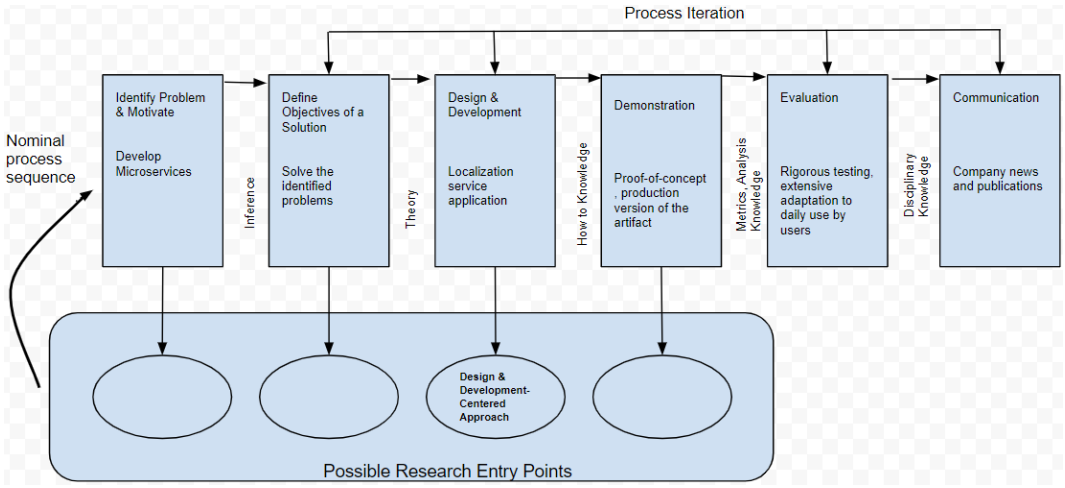

Figure 6.1: DSRM Process for the Localization Service [[22], p. 65]

Figure 6.1 shows the DSRM process for project development of localization service. The possible starting point for research into the design and development approach begins with the design and development process. Nominal sequence of processes begins with identification of problem and motivation, defining goals of solution, design and development, demonstration, evaluation and communication. The iteration of the process goes on until the design and

development work meets the needs.

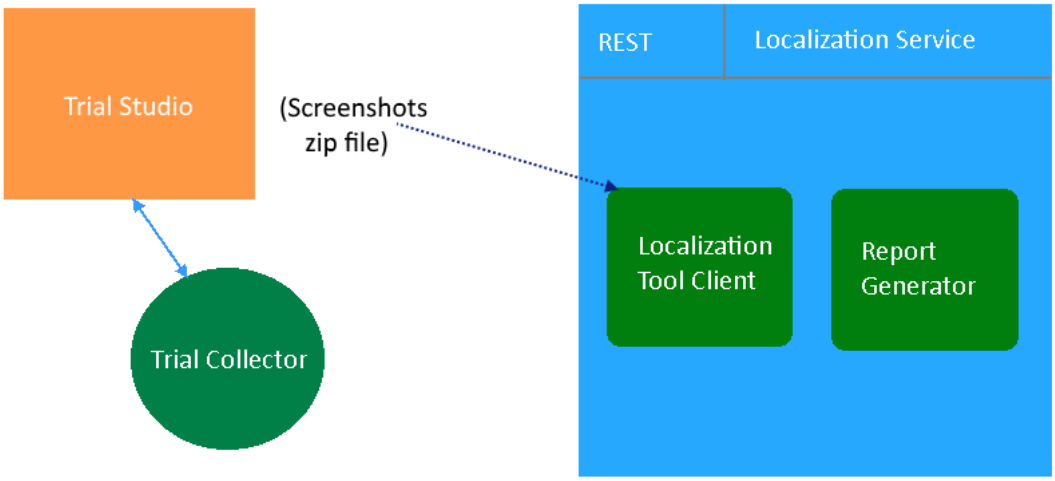

Figure 6.2: System design and architecture

Figure 6.2 provides the relations between different components. TrialStudio is a desktop application that has localization supports features that generates the screenshots from the TrialCollector. TrialCollector is an android application. The generated screenshots from TrialCollector are zipped into single portable file. The zipped file contains the screenshots taken with all the selected languages. Localization Tool Client is a microservice application that takes the screenshots zipped file and called the Report Generator microservice to generate the Word Report documents. The output of the localization service is to produce a word report document from the screenshots of selected languages. Each language produces the separate word report documents.

6.6 Demonstration

Figure 6.3 shows the deployment of microservices into the OpenShift environment. The system developed for localization service includes the two main microservice applications, i.e., Localization Tool Client and Report Generator. The application was developed using React and JavaScript. Localization tool client takes the screenshots zipped file and the file can be uploaded either manually or using drag-and-drop feature. The zipped file needs to be under certain formats to upload successfully by the localization tool client, otherwise, the error message will be shown. Once the uploading the zipped file is successful, the list of languages used for creating screenshots will be shown in the table. The word report document can be generated for each selected language. More information about the use of the OpenShift Environment microservices can be found in the section Appendix.

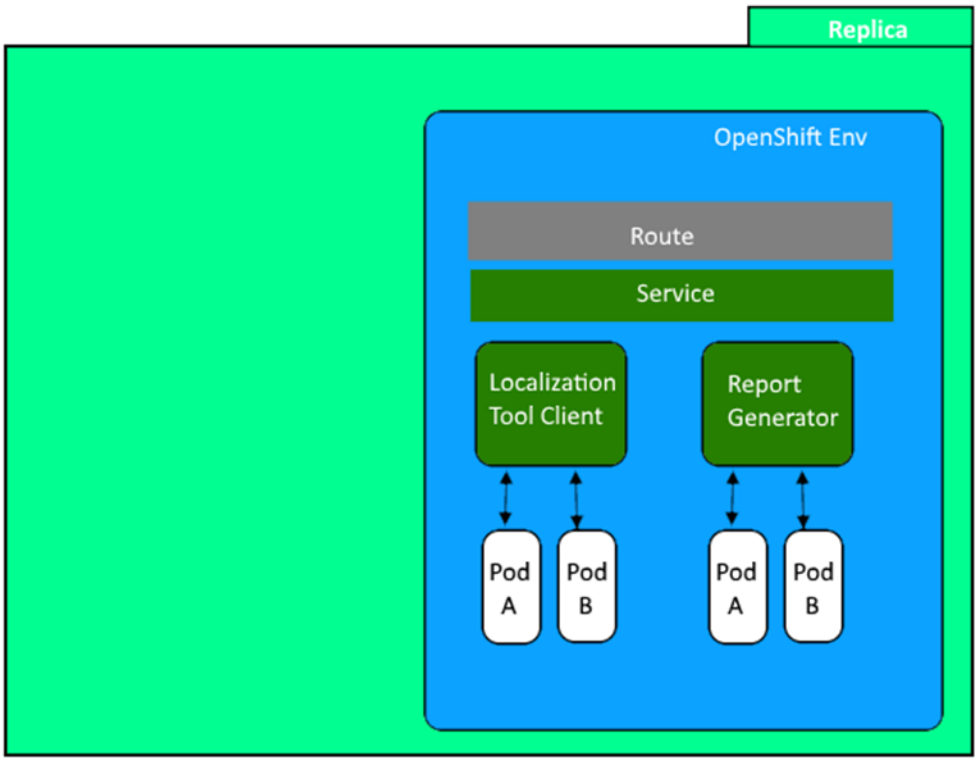

Figure 6.3: Microservices deployed in OpenShift environment

The simple prototype application services that only include two microservices are displayed in Figure 6.3. The question might arise if additional efforts are really worth using the Microservice architecture to develop this simple service. The answer is obvious: yes. It does not seem to matter how the system application is, microservice architecture in terms of simplicity, scalability and maintenance much better than a monolithic one. It takes little effort to use architecture with microservice rather than monolithic architecture. In addition, the use of microservice architecture offers several advantages since service development is independent. This will also help the teams to distribute tasks.

6.7 Evaluation

The developer, tester and Application Specialist (AS) team began a test process when the localization service application was developed. Initially, in a closed group, the system was extensively tested for debugging. The app was then shared via the intern web portal with the whole teams of the AS. Anyone who has a right to access the web portal, can use the software tool and use for testing. The tools were very efficient and helps a lot specially for the AS team

during the designing and developing of the trial protocol. More than 40 percent of their time was saved using the tool when designing and developing the trial protocols. The application tool was very user-friendly and easy to use. The efforts and contribution to the successful design of the localisation service were appreciated by all.

Enterprises have identified several issues related to the traditional approach to software development. In the current digital business platform, issues such as complexity, scalability and sustainability are identified that most businesses are trying to avoid or minimize the effects. The entire system application is programmed into a single project in monolithic application. All modules, dependencies, functions and everything from UI to business logic and data management are packaged in a single file. Complexity is obviously increasing as volume increases and the system becomes less scalable and maintainable. It has a direct effect on the onboarding of the new developer. In addition, features such as continuous integration and delivery and external world exposure of APIs will become ineffective and obsolete.

The methodology of design science research has focus on a rigorous design process to solve the observed problem, contribute to research, evaluate design and communicate the result to the relevant stakeholders. Using microservice architecture, artifacts such as prototyping software development platform has used to identify research problems and solutions. The research artifacts concludes that the use of microservice architecture in digital platform minimizes and resolves the existing monolithic application problems. The larger-volume monolithic application will decompose into several component service called microservice. Each service component is designed to be small and independent in order to solve a single task with specific business capacity. Each service can be individually developed and deployed.

Continuous development and integration can be easily accomplished with a small and independent component making the system less complex, highly scalable and maintainable. In addition, new developers can speed up their on-board learning process. However, the use of microservice architecture also has some inconvenience that may not always be a good solution for all system development. Each service component has its own directory in microservices and must be individually constructed, developed and deployed. Consequently, multiple service deployments are mandatory. It can also provide significant overhead with respect to design, interoperability of services, system resource management, and use. Overall, microservices have much more advantages than their disadvantages. Microservice architectural paradigms have solved most of the problems that companies have identified.

6.8 Communication

The approach of software development process of localization service based on microservices were highly successful. Further, the use of docker and OpenShift

environment in the software development process has helped to the developer to mainly focus on development activities and the hosting to deploy and maintain the microservice application with minimum efforts. As a result, the number of microservice based application has increases, since all the teams started following the microservices based software development practice and standards.

6.9 Contribution

The results of this study provide valid and effective measures to make an important decision on the selection of the project-based software development process. Research artifacts can be used for the assessment and evaluation of the efficiency and performance of the software development process at the organizational and project level. Research can help developers, testers, hosters and other stakeholders to understand how it contributes to software development.

Chapter 7

Software Development: Localization Service

Localization service is developed using the Microservices architecture style. The architectural style of Microservices is an approach to the development of a single application as a suite of independently deployable services, each running in their own process and communicating with lightweight mechanisms, often an HTTP resource API.These services are based on business capabilities and can be deployed independently through fully automated deployment tools. [24]

7.1 Application Stack

Technology	Pick
Language	*JavaScript, ES6*
client-side templating language (EJS) for prototyping	*EJS*
Dependency Management	*NPM*
Linting & Style	*ESLint with AirBnb configuration*
Server-side framework	*Node.js*
Web app. Framework	*Express*
Utility Library	*Lodash*
Unit testing	*Jest*
Functional testing	*Selenium based*
Test coverage	*Istanbul*
Data storage	*PostgreSQL / Mongo DB*
Front-end framework	*React*
Application lifecycle	*Redux*
Built Tool	*Webpack*
Styling	*LESS, SASS*

Figure 7.1: Application Stack

Figure 7.1 show the technologies stack used to develop the localization services application.

7.2 Localization Tool Client

Localization tool client service provides the UI that allows uploading the .zip files containing the screenshots and metadata. The .zip file is uploaded and becomes ready for generating the .doc report. For generating the report file, the localization tool client service uses the report generator service.

RESTful API	
Node Environment Variables	
GET /api/v1/nodeenv/variables/{id}	Returns a node environment variable value. If the variable is not defined an error is thrown.
Download Generated Reports	
GET /api/v1/reports/:id	The route allows to download via express the created document whether it is stored on disk or database.
Report Archives	
POST /api/v1/reportarchives	Uploads and extracts the content of a report archive. Only ZIP files are currently supported. The top-level folder inside the ZIP should be the screenshots folder generated from TrialStudio. The files are extracted to a temporary folder.
Check if report exists	
GET /api/v1/reports?id={reportName}	Checks the existence of generated report.

Figure 7.2: Localization Ttool Client Service API

Notes: React and JavaScript (ES6) are used to develop the user interface (UI) of the localization tool client service. Besides, unit testing of react component is done using Snapshot testing provided by the Jest Framework. Jest provides many features including snapshot testing, instant feedback and parallel executing and testing. At least for development environment, windows-build-tools are required to compile popular native node modules and can be installed using following commands: npm install –global windows-build-tools

7.3 Report Generator

Report generator service allows to generate the .doc report based on the screenshots available in the zip files. The generated report file can be downloaded easily by clicking the corresponding download button. JavaScript (ES6) is used on server-side implementation. Similar to the localization tool client service, this service is also bound to have those limitation and prerequisites defined

above. However, report generator service do not have client UI and hence does not use react.

RESTful API	
Reports	
POST /api/v1/reports	Creates a report in Word format

Figure 7.3: Report Generator Service API

The report generator is responsible for cleaning up the REPORTS_DATA_-FOLDER. The data held there expires after a time defined by REPORTS_-DATA_MAX_AGE_MILLISECONDS. The clean-up is executed at regular intervals defined by REPORTS_DATA_CLEANUP_INTERVAL_MILLISEC-ONDS.

7.4 Service Integration

Service communicates each other using REST API and HTTP communication protocol. Localization services such as localization tool client and report generator are deployed under OpenShift environment. The OpenShift is a container deployed in replica. OpenShift environment provides the management tools for handling the Microservices. The management tools provide features such as logging, routing, fault tolerance, security, and status of the services. [9]

7.4.1 Implementing Request/Response collaboration

Representational State Transfer (REST) and HTTP: REST is an architectural style or pattern intended for web-inspired APIs. And HTTP is one of REST api's most frequently used communication protocol. HTTP itself describes some helpful REST-style capacities .For example, in the HTTP specification, the HTTP verbs (e.g., GET, POST, PUT and DELETE) already have well-understood meanings as to how resources should be used. [23] The REST architectural style says that all resources should use the same techniques and that a set of techniques should be defined in HTTP specification. This means many different methods can be avoided to createResource or editResource. Rather, we can simply request the server to create that new resource. The notion of Hypermedia As The Engine Of Application State (HATEOAS) is another idea implemented in REST that prevents coupling between client and server. It should be the final level of a REST API. All the resulting entities that you receive with a rest call will also release their next possible actions. [23]

7.5 Security

Aspects of security such as authentication and authorization are key concepts when it comes to people and things that interact within a system. In addition, it is not efficient to log in separately for different systems when it comes to a distributed system, using a different username and password for each one. The goal is to have a single identity that can authenticate once. Single sign-on (SSO) is a property allows the user to logn in with a single ID and passport for gaining access to multiple related, yet independent, software systems and often accomplished by using the Lightweight Directory Access Protocol (LDAP) and stored LDAP databases on directory servers [11].

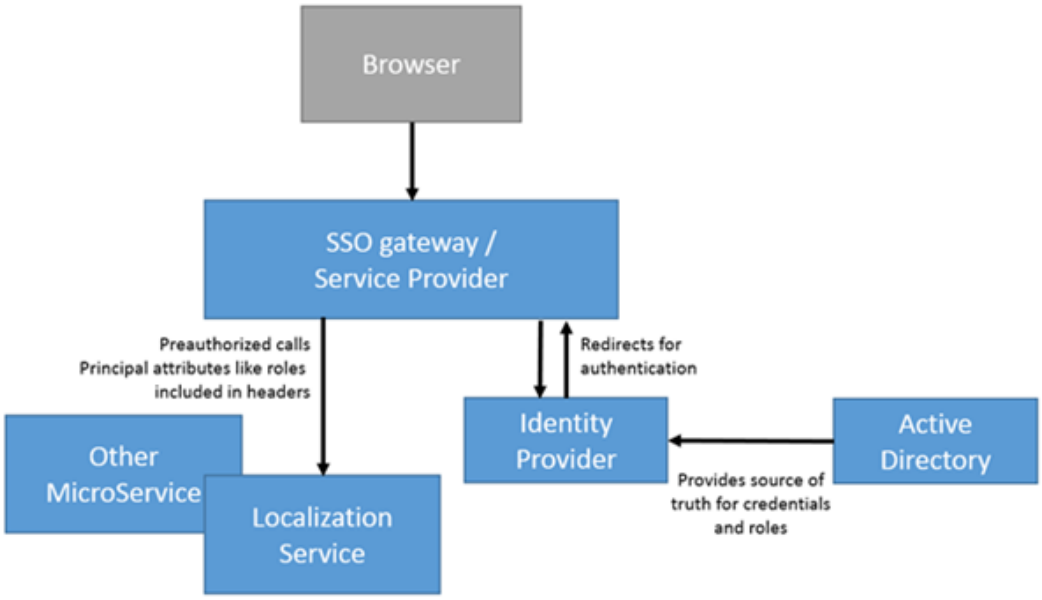

Figure 7.4: Security via Single sign-on (SSO)

When a principal attempts to access a resource (such as the localization service), the principal is directed to authenticate using a flow defined in OpenID Connect with an identity provider. In this process, the Authorization Server, also referred to as an OpenID Connect Provider (OP), returns an ID Token to the SSO portal together with the Access Token, enabling it to decide whether to give them access to the resource. [11]

7.6 Deployment

Figure 7.5 shows the OpenShift Container Platform and the localization services: ts-localization-tool-client and ts-report-generator that are deployed in OpenShift environment. Each service is running under 2 pods. However, the service can use several pods to maintain load balancing and fault tolerance

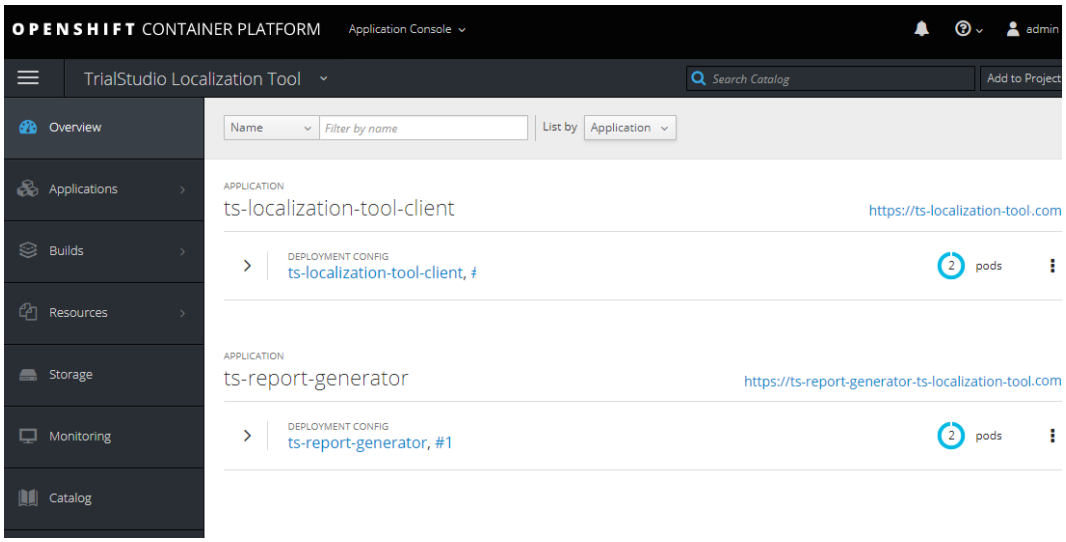

Figure 7.5: OpenShift Container Platform

Througth the platform, it is very easy to maintain the services as it allows to monitor the logs and events in real time. In addition, multiple service can be easily imported and created directly.

7.7 Discussion: Microservices and Design Science

In practice, design seience methodology was used as guidelines for the software development process. First, the enterprises problems such as complexity in development and maintenance, low scalability were identified. Since then, the objectives for a solution is define such as building the prototype application based on microservice design patterns. The use of microservices has helped distribute the task and continue the work of development in parallel so that the work of development can be carried out independently and in parallel. As a result, the product development work will be efficient, effective and with in budget time.

Similarly, from the start to the end of the growth of the localization tool service, the design science approach has helped a lot. With this DS methodology, the idea of developing and constructing prototype implementation and carrying out prototype-based inquiry and study was produced. During the software development process, it provides the guidelines and instructions to follow. As a result, the software development activities will be accomplished based on requirements and on time. The result artifact will definately solved the issues identified by the enterprises.

Chapter 8

Evaluation

The research project was successfully completed with satisfactory results according to the research plan. Most of the questions concerning architectural paradigms and software development have been answered and resolved by the research outcome. During the research work, more skills and knowledge related to the improvement of software development process were gained.

After an analysis of research, the better approach for companies to produce their products is clarified by the microservice architecture. Software development work can be done easily, effectively and efficiently with Microservice Architecture. In addition, the onboarding process for new hires will be easier and faster. Docker and Openshift environment use have helped to maintain consistency in development, testing and manufacturing. Furthermore, the deployment of microservices is much faster and easier compared to the previous one where the application on the monolithic server is so huge and contains multiple modules in a single bundle or repository. Similarly, compiling and constructing the complete monolithic application is difficult to accomplish. Usually, some module code has not been compiled or built.

However, refactoring those whole monolithic system having several modules has some drawback that has identified and experienced during the research project work. Refacturing entire system to component microservices is a huge task and the challenges can only be faced by experience and senior core developers. From the business point of view, the revalidation of system products requires enormous resources and efforts. Therefore, it does not seem feasible to refactor the entire existing system into the service component. Furthermore, having multiple microservices requires multiple repositories and each service needs to maintain and deploy individually on the server. This also brings the extra overhead to the developers.

Finally, architectural paradigms for microservices are more advantageous than monolithic ones. Larger enterprises should therefore consider using the microservice architecture to expand services further or develop new products.

Chapter 9

Conclusions

The aim of this design research project was the development and evaluation of IT objects designed to solve software development organizational problems. Bigger enterprises are always looking for better solution for the development of their products with a view to minimize the cost, time and improve the quality and maintainability of the development works. The artifacts of the research methodology in the field of design science will assist stakeholders and teams in developing efficient product development decisions and management in advance.

The design science research project was successfully completed with the design and development of two prototypes of applications based on microservice architecture. The development of prototypes and result shows that following the microservice architecture in the software development process, the development work was easy to carry out since the whole task was distributed as component among the teams and developer can fully concentrate on the coding parts. Further, the improvements and changes on the features and functionality could be done easily during iteration. The microservice application developed was easy to develop, build, deploy to the server. Similarly, the testing and maintaining tasks could be done with minimum efforts. Besides these, the uses of Docker and Openshift environment will helped a lot to maintain the consistency, effectiveness and maintainability of the product development work. The use of microservice architecture in the software development project helps a lot through the different points of view to minimize the chances of failure and produce the quality products.

Continuous development and integration can be easily accomplished with a small and independent component making the system less complex, highly scalable and maintainable. In addition, new developers can speed up their on-board learning process. However, the use of microservice architecture also has some inconvenience that may not always be a good solution for all system development. Each service component has its own directory in microservices and must be individually constructed, developed and deployed. Consequently, multiple service deployments are mandatory. It can bring with its significant overhead in terms of design, interoperability of services, management, and use of system resources. [5] The cost will be too high for the application which cannot make sufficient use of its advantages. Overall, microservices have much more advantages than their disadvantages. Microservice architectural paradigms have solved most of the problems that companies have identified. The result artifact can help to minimize the limitation of architecture by taking proper decision beforehand on estimating the design cost, time and resources.

Bibliography

[1] RH Von Alan, ST March, J Park, and S Ram. Design science in information systems research. *MIS Quarterly*, 28(1):75–105, 2004.

[2] Sascha Alpers, Christoph Becker, Andreas Oberweis, and Thomas Schuster. Microservice based tool support for business process modelling', ieee 19th international enterprise distributed object computing workshop., 2015. Enterprise Distributed Object Computing Workshop (EDOCW), 2015 IEEE 19th International. Accessed 09.05.2018.

[3] Vineet Badola. Microservices architecture: advantages and drawbacks, 2017. `http://cloudacademy.com/blog/microservicesarchitecture-challenge-advantage-drawback/`. Accessed 25.07.2018.

[4] BitKeeper. Bitkeeper, 2019. `https://www.bitkeeper.org/BK_Nested_White_Paper.pdf`. Accessed 02.10.2018.

[5] Runscope Blog. 5 reasons not to use microservices — runscope blog, 2015. `https://blog.runscope.com/posts/5-reasons-not-to-use-microservices`. Accessed 15.03.2019.

[6] Familiar Bob. Microservices, iot, and azure, 2015. ISBN 978-1-4842-1275-2 edn., online: Apress.

[7] Richardson Chris. What are microservices?, 1996. `http://microservices.io/index.html`. Accessed 22.4.2018.

[8] Google Cloud. What are containers and their benefits | google cloud, 2019. `https://cloud.google.com/containers/`. Accessed 02.10.2018.

[9] Webdesigner Depot. Monolith vs microservices: Which is the best option for you? | webdesigner depot, 2019. `https://www.webdesignerdepot.com/2018/05/monolith-vs-microservices-which-is-the-best-option-for-you/`. Accessed 05.11.2018.

[10] Docs.okd.io. Kubernetes infrastructure – infrastructure components | architecture | okd latest., 2019. `https://docs.okd.io/latest/architecture/infrastructure_components/kubernetes_infrastructure.html#architecture-infrastructure-components-kubernetes-infrastructure`. Accessed 18.10.2018.

[11] Doug Drinkwater. What is single sign-on? how sso improves security and the user experiences, 2018. https://www.csoonline.com/article/2115776/what-is-single-sign-on-how-sso-improves-security-and-the-user-experience.htmll. Accessed 16.04.2019.

[12] M. Fowler. Microservices, a definition of this new architectural term, 1996. http://www.tug.org/texlive/. Accessed 08.05.2018.

[13] Docker Inc. What is a container? | docker, 2019. https://www.docker.com/resources/what-container. Accessed 10.01.2019.

[14] Docker Inc. Docker overview | docker documentation, 2019. https://docs.docker.com/engine/docker-overview/. Accessed 05.04.2019.

[15] InfoQ. The strengths and weaknesses of microservices., 2019. https://www.infoq.com/news/2014/05/microservices. Accessed 25.06.2018.

[16] Kinnary Jangla. Accelerating development velocity using docker: Docker across microservices., 2018. 1st edition. Apress.

[17] Ben Morris. How big is a microservice?, 2015. https://www.ben-morris.com/how-big-is-a-microservice/. Accessed 30.02.2019.

[18] Neoteric. How can you refactor a monolithic application into microservices?., 2019. https://medium.com/@NeotericEU/how-can-you-refactor-a-monolithic-application-into-microservices-2eef8e32384 Accessed 20.07.2018.

[19] Saverio Giallorenzo Nicola Dragoni, Alberto Lluch Lafuente, Manuel Mazzara Fabrizio Montesi, Ruslan Mustafin, and Larisa Safina. Microservices: yesterday, today, and tomorrow, 2016. URL https://arxiv.org/abs/1606.04036.

[20] OpenShift. Basic usage - using minishift | minishift | okd latest, 2019. https://docs.okd.io/latest/minishift/using/basic-usage.html. Accessed 15.12.2018.

[21] Diego Pacheco. Building effective microservices, 2017. 1st edition. Packt Publishing.

[22] Ken Peffers, Tuure Tuunanen, Marcus A. Rothenberger, and Samir Chatterje. A design science research methodology for information systems research. *Journal of Management Information Systems / Winter 2007–8*, 24 (3):45–77, 2008.

[23] Newman Sam. Building microservices, 2015. ISBN: 9781491950340, First Edition edn., US: O'Reilly Media, Inc.

[24] Technovature Marketing Team. Why microservices, 2019. https://www.technovature.com/m/sd/microservices.html. Accessed 15.04.2019.

[25] Vamsi Talks Tech. Why legacy monolithic architectures won't work for digital platforms., 2019. http://www.vamsitalkstech.com/?p=5617. Accessed 09.05.2018.

[26] tutorialspoint.com. Docker overview, 2019. https://www.tutorialspoint.com/docker/docker_overview.htm. Accessed 22.12.2018.

[27] tutorialspoint.com. Kubernetes tutorial, 2019. https://www.tutorialspoint.com/kubernetes. Accessed 02.10.2018.

[28] tutorialspoint.com. Openshift tutorial, 2019. https://www.tutorialspoint.com/openshift/index.htm. Accessed 25.09.2018.

Appendix A

Example of DockerFile

Dockerfile

```
# set the base image to Debian
# https://hub.docker.com/_/debian/
FROM debian:latest
ENV DEBIAN_FRONTEND noninteractive
COPY . /

# replace shell with bash so we can source files
RUN rm /bin/sh && ln -s /bin/bash /bin/sh

# update the repository sources list
# and install dependencies
RUN apt-get update \
    && apt-get install -y curl \
    && apt-get -y autoclean


# nvm environment variables
ENV NVM_DIR /usr/local/nvm
ENV NODE_VERSION 8.7.0

# Python dependencies

RUN apt-get -y update && \
        apt-get -y install \
        python2.7-dev libffi-dev python-pip && \
        rm -rf /var/lib/apt/lists/* && \
        pip install -r requirements.txt

# install nvm
# https://github.com/creationix/nvm#install-script
RUN curl --silent -o- https://raw.githubusercontent.com/creationix/nvm/v0.31.2/install.sh | bash

# install node and npm
RUN source $NVM_DIR/nvm.sh \
    && nvm install $NODE_VERSION \
    && nvm alias default $NODE_VERSION \
    && nvm use default

# add node and npm to path so the commands are available
ENV NODE_PATH $NVM_DIR/v$NODE_VERSION/lib/node_modules
ENV PATH $NVM_DIR/versions/node/v$NODE_VERSION/bin:$PATH

# confirm installation
RUN python -V
RUN node -v
RUN npm -v
```

Figure A.1: Example of Dockerfile