INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




Entrega Contínua de Aplicações em Ambientes Híbridos

VÍTOR JOÃO FALLÉ CORREIA
outubro de 2023

POLITÉCNICO
DO PORTO

Instituto Superior de Engenharia do Porto

Continuous Application Delivery in Hybrid Environments

Vitor Correia

A dissertation submitted in partial fulfillment of the requirements for the degree of Master of Science, Specialisation Area of Computer Systems

Supervisor: Nuno Bettencourt, PhD

Evaluation Committee:
President:
Isabel Praça, PhD

Members:
Nuno Ferreira, PhD

Porto, October 16, 2023

Statement of Integrity

I hereby declare having conducted this academic work with integrity.

I have not plagiarised or applied any form of undue use of information or falsification of results along the process leading to its elaboration.

Therefore the work presented in this document is original and authored by me, having not previously been used for any other end.

I further declare that I have fully acknowledged the Code of Ethical Conduct of P.PORTO.

ISEP, Porto, October 16, 2023

Abstract

The use of hybrid deployment models is becoming a standard since it provides many benefits regarding on-demand scalability, high availability, and reliability. However, the management of resources (application specifications and infrastructure details) in a hybrid environment is a complex task since interfaces vary depending on the vendor. Therefore, continuous practices already established must be adjusted whenever new interfaces are to be supported.

This work aims to explore how a deployment process can be improved regarding the adoption of hybrid environments at the host organization DevScope, producing a proof of concept capable of explaining how this improvement can be achieved.

A Components-of-the-Shelf (COTS) approach is followed for developing the solution that relies on different technologies to achieve the expected results. The proof of concept consists of deploying the infrastructure that supports the new deployment process and the orchestration of the components to enable the delivery of applications to the environments. Terraform is used to define and deploy the infrastructure, and KubeVela is the technology used for managing applications.

The implementation of the solution made it clear that managing applications in heterogeneous platforms is not easy. After some experimentation and answers to a prepared questionnaire regarding the new deployment process, it was possible to conclude that the solution still has a margin for improvement and that some technologies are projects still being actively improved. Although KubeVela is not polished enough, it was possible to verify that it is easily extendable and that more scenarios for specific applications can be added when needed.

The obtained results are relevant for future studies and will contribute to a better understanding of how the delivery of multiple applications can be homogenized. It also helps to address other problems, such as oversimplification of deployments, losing important infrastructure-based details, and the associated learning curve to deploy a complex application.

Keywords: Automation, Cloud, DevOps, Kubernetes, OAM, Orchestration

Resumo

A utilização de estratégias híbridas de implantação está a crescer gradualmente, uma vez que proporciona muitos benefícios em termos de escalabilidade, alta disponibilidade e fiabilidade. No entanto, a gestão de recursos (especificações da aplicação e detalhes da infraestrutura) num ambiente híbrido é uma tarefa complexa, uma vez que as interfaces variam consoante o fornecedor, pelo que as práticas contínuas já estabelecidas devem ser ajustadas sempre que novas interfaces necessitam de ser suportadas.

Este trabalho visa explorar a forma como o processo de implementação pode ser melhorado relativamente à adoção de ambientes híbridos na DevScope.

Foi adoptada uma abordagem "Components-of-the-Shelf" (COTS) para desenvolver a solução, que se baseia em diferentes tecnologias para alcançar os resultados esperados. A prova de conceito consistiu na implementação da infraestrutura que suporta o novo processo de implementação e na preparação dos componentes para permitir a entrega de aplicações aos ambientes. O Terraform foi utilizado para definir e implementar a infraestrutura e a principal tecnologia utilizada para gerir as aplicações foi o KubeVela.

A implementação da solução tornou mais claro que a gestão de aplicações em plataformas heterogéneas não é fácil. Após alguma experimentação e respostas a um questionário preparado sobre o novo processo de implementação, foi possível concluir que a solução ainda tem margem para melhorias e que algumas tecnologias são projectos que estão a ser ainda ativamente melhorados. Embora o KubeVela não esteja suficientemente polido, foi possível verificar que é facilmente extensível e que podem ser adicionados mais cenários para aplicações específicas quando necessário.

Os resultados obtidos são relevantes para estudos futuros e contribuirão para uma melhor compreensão de como a entrega de várias aplicações pode ser homogeneizada. Também ajudarão a resolver outros problemas, como a simplificação excessiva das implementações, a perda de alguns pormenores importantes baseados na infraestrutura e a curva de aprendizagem associada à implementação de uma aplicação complexa.

Acknowledgement

Firstly, I'd like to thank my family and friends for all the support and unimaginable motivation they gave me. Also, I'd like to thank my supervisor, Nuno Bettencourt for accompanying and supporting me in my work. I'd like to thank everyone at DevScope for all the knowledge, resources, and new friendships provided. Finally, I'd like to thank Nicole for all the daily support, and motivation, and for not letting me give up on my dreams.

Contents

List of Figures

List of Tables

List of Source Code

List of Acronyms

AHP	Analytic Hierarchy Process.
AKS	Azure Kubernetes Service.
API	Application Programming Interface.
AWS	Amazon Web Services.
BPEL	Business Process Execution Language.
BPMN	Business Process Model and Notation.
CD	Continuous Deployment.
CDE	Continuous Delivery.
CI	Continuous Integration.
CI/CD	Continuous Integration and Continuous Deployment.
CLI	Command-Line Interface.
CNCF	Cloud Native Computing Foundation.
COE	Container Orchestration Engine.
COTS	Components-of-the-Shelf.
CUE	Configure Unify Execute.
DNS	Domain Name System.
DSR	Design Science Research.
FFE	Fuzzy Front End.
GCP	Google Cloud Platform.
GQM	Goal Question Metric.
GUI	Graphical User Interface.
HCL	HashiCorp Configuration Language.
IaaS	Infrastructure-as-a-Service.
IaC	Infrastructure-as-code.
JSON	JavaScript Object Notation.
K8S	Kubernetes.
MTTR	Mean Time To Restore.

NCD	New Concept Development.
NIST	National Institute of Standards and Technology.
NPD	New Product Development.
OAM	Open Application Model.
OCI	Open Container Initiative.
OCM	Open Cluster Management.
ODE	Orchestration Director Engine.
PaaS	Platform-as-a-Service.
POC	Proof of Concept.
SaaS	Software-as-a-Service.
SMR	Systematic Mapping Review.
SQL	Structured Query Language.
TOSCA	Topology and Orchestration Specification for Cloud Applications.
TTL	Time to live.
VCS	Version Control System.
VPN	Virtual Private Network.
YAML	Yet Another Markup Language.

Chapter 1

Introduction

There are more organizations taking advantage of Continuous Integration, Delivery, and Deployment practices to deploy quickly and reliably their applications and underlying infrastructure [1]. Deploying an application with a variety of existing tools is simple. The scenario starts to get complex when dozens or even hundreds of deployment processes of applications need to be managed, plus the additional dependencies that have to be provisioned to guarantee the proper functioning of the software product.

After noticing the exponential growth of the infrastructure due to the high demand for numerous applications, cloud computing emerges as a new technology and business paradigm [2]. It consists of different techniques to provide a large number of functionalities and operations. Combining continuous practices with cloud computing leverages automation regarding application deployments. With the help of cloud computing, continuous practices can be simplified since resources are controlled by external vendors, and the setup of on-premises infrastructure is reduced.

Adopting cloud computing brings benefits such as reduced costs and high flexibility, which solves the bottleneck of on-premises infrastructure, reliability, access to services over the Internet, and much more. Cloud computing is often referred to by the term cloud itself, and organizations started adopting these remote environments a while ago. Thereafter, multiple cloud vendors began to surface, such as AWS (from Amazon), Azure (from Microsoft), Google Cloud Platform (GCP), and many others that are expanding their service to serve and compete for that purpose as well.

Having many offers with competitive prices and innovative technologies, a single-vendor strategy often does not satisfy organizations' needs [3]. The scenario gets more complicated, compelling organizations to adopt a multi-cloud approach when deploying applications, taking advantage of benefits from each vendor to meet organizations' and clients' requirements. Additionally, the hybrid scenarios also behold the mix of cloud and on-premises infrastructure. Some companies opt for this approach because of some legacy applications or due to internal applications that must not be publicly available.

This research addresses the previous inconveniences, focusing on improving the application deployment processes in hybrid environments, including multi-cloud platforms and on-premises infrastructure, to reduce overhead and simplify application deployment management.

This chapter describes the identified problem, the respective goals/objectives, and research questions derived from the goals using the Goal Question Metric (GQM) approach. Also, it depicts the research methodology used throughout the project and the considered hypotheses, attending to the research questions to analyse whether the work is accomplished.

1.1 Problem

A hybrid environment (also referred to as hybrid deployments or scenarios) is a deployment environment where components are separated among different target infrastructures, whether on-premises or in the cloud.

The employment of these hybrid scenarios when deploying applications and provisioning infrastructure increases complexity and overhead when managing and maintaining these processes. More products are developed and deployed by the organization, which implies an exponential increase in concern regarding these factors. New employees must be recruited, or existing ones reskilled to manage specific deployment processes. Having specific people to manage these without the whole team knowing how to manage them is troublesome, causing high dependency on particular individuals to execute specific tasks.

Also, deploying applications seamlessly between two different platforms (*e.g.* Kubernetes and Virtual Machine) is a difficult task due to disparities in terms of parameters and specifications that must be declared for the resources to be properly created.

At DevScope, DevOps methodologies are implemented on the first day of the creation of a software project. Continuous Integration and Continuous Deployment (CI/CD) pipelines, monitoring and logging solutions, automation, and configuration management strategies, and others are used daily by developer and operation teams. Cloud technologies and infrastructure are used regularly for the deployment of applications, secret management and other scenarios.

The developer and operation teams do not have a centralized system where they can view deployed applications in different environments and check metrics and logs of applications. More specifically, the operation team works with Kubernetes clusters daily. There is no solution yet for managing artifacts across multiple clusters.

Although DevScope uses Azure more than any other cloud, the use of other clouds is inevitable due to cost management, exclusive services, and the flexibility it gives for having services for backing up the main infrastructure. The learning curve for other clouds is accentuated [3], and the management of resources in all clouds places overhead on the operation team.

1.2 Goals

Goals for this project are defined using a GQM [4] approach. It will contribute to generating research questions in this chapter and later to evaluate the solution using qualitative and quantitative metrics in Chapter 6.

The GQM paradigm focuses on the creation of a measurement plan that targets specific issues according to a set of rules for the interpretation of the measurement data [4]. The data is driven by goals. Therefore, to improve a process, it is necessary to define measurement goals based on improvement goals and transform them into activities that can be measured.

The model starts with the definition of measurement goals. These goals are then refined and converted into several questions that target major components (characteristics of the object of measurement). Therefore, these questions are refined, producing metrics that provide information to answer the previous questions.

Therefore, according to the described problem, the goal identified is the simplification of application deployment and management in hybrid environments. Then, it was transformed into a measurement goal as the GQM paradigm suggests:

G1. **Analyse** application deployment and management **for the purpose of** improving/optimising **with respect to** simplify **from the viewpoint of** the operations team **in the context of** hybrid environments.

1.3 Research Questions

After refining the improvement goal, various research questions can be derived from it.

RQ1. Can an application deployment process be simplified when it targets different platforms?

RQ2. What is the influence of extensibility upon simplification of the deployment process targeting different platforms?

RQ3. Does the degree of control regarding infrastructure details decrease when the creation of new deployments across different platforms is facilitated?

1.4 Hypothesis

According to the GQM paradigm, hypotheses are formulated as expected answers. These hypotheses stimulate thinking about the current situation of a specific process and/or product triggering a better understanding of them [4]. After the measurement process is complete, these hypotheses will be compared to the actual results to understand and analyse which reasons could cause the deviation from the defined expectations. The following hypotheses are defined:

H1. The process of deployment of an application, independently of the target platform, is faster and more reliable compared to the old one.

H2. The increase in support of different platforms or workflows in the deployment process does not affect the complexity of the deployment process.

H3. Controlling certain characteristics of the underlying infrastructure is possible through override techniques.

1.5 Research Methodology

As referred to in section 1.2, this thesis aims at expanding knowledge and exploring how to improve continuous deployment processes, more specifically, how to orchestrate applications and infrastructure across hybrid environments in a simplified manner.

The Design Science Research (DSR) method is used as a research methodology. DSR methodology has been evolving, and it consists in a rigorous process that helps in (i) research contributions (ii) evaluating designs and (iii) communicating the results to interested parties [5].

The problem and objectives have already been identified in this chapter (sections 1.1 and 1.2), therefore a problem-centred initiation approach was taken. Hypotheses were also formulated to aid in the evaluation phase, providing comparisons with actual results. After that, an analysis and design phase are presented in Chapters 3 and 4 respectively, detailing which functional requirements, concerns and possible alternatives are relevant considering the development of the solution.

Thereafter, the developed artifacts will be used to solve the stated problem, advancing to the demonstration phase where the process of implementation is detailed, as well as the technologies used.

An evaluation phase takes place after the demonstration phase, where the solution will be observed and analysed to ensure the solution's validity. In the end, it is expected to share the obtained knowledge throughout this thesis and among the team members at DevScope.

The DSR methodology was chosen among many others because of its focus on extending knowledge in a specific area by developing new artifacts and sharing them with the intent to solve identified problems, creating value or having utility for the community.

1.6 Thesis Structure

This thesis is organized into different chapters which are:

- **Chapter 1 (Introduction)** – presents the problem and identified goals of the work, also describing research questions and adopted methodology, including hypotheses.
- **Chapter 2 (State Of The Art)** – introduces basic concepts and presents related work and technologies useful for the completion of the project.
- **Chapter 3 (Analysis)** – divided into two parts: requirements engineering and value analysis. The first gathers and describes the identified functional and non-functional requirements, and the second focuses on why and how the project carries business value.
- **Chapter 4 (Design)** – presents how the solution will be evaluated and how the results will be validated.

- **Chapter 5 (Implementation)** – details the development process of the solution, including technologies or tools used.
- **Chapter 6 (Experiments and Evaluation)** – presents how the solution will be evaluated and how the results will be validated.
- **Chapter 7 (Conclusion)** – presents achievements and potential contributions. Limitations and future work are also described.

Chapter 2

State Of The Art

This chapter introduces base concepts that support the rest of the document. The related work is presented mainly linked to topics regarding resource orchestration and possible abstractions of application orchestration in hybrid scenarios. Lastly, it presents related technologies in key areas that are relevant to this project, finalizing with a summary of the whole chapter.

Research of existent work was conducted using a Systematic Mapping Review (SMR) based on conference papers and journal articles in authentic electronic databases reviewed by peers. The considered databases were ACM Digital Library, Google Scholar, and B-On. Considering the research questions in the section 1.3, the selected keywords for this research were "multi-cloud", "hybrid-cloud", "multi-platform", "continuous deployment" and synonyms considered relevant for each term. The search was also restricted to a time frame between 2013 and 2023 to get more recent results. The main search query developed was:

```
AllText:("hybrid cloud" OR "hybrid-cloud" OR "multicloud" OR "multi cloud" OR
"multi-cloud" OR "multi-platform" OR "multiplatform" OR "multi platform")
        AND AllText:("continuous deployment" OR "continuous practices" OR
"CI/CD" OR "CICD")
        AND PublicationDate:(2013-2023)
```

Results of the search query in different databases can be viewed in Table (2.1):

Table 2.1: Research query results by database (without duplicates)

Database	**Results**
B-On	94
ACM Digital Library	83
Google Scholar	117
Total	**294**

The search in the three databases resulted in 294 results, in which abstracts were read, reducing the pool results to 18. In some articles, the *snowballing* method was used to obtain more information regarding the referred topics.

Additionally, 3 articles were added to the results from a specific query that resulted after the main search presented above. The strings used were “continuum computing”, “configuration management” and “kubernetes”.

2.1 Background

This section has the purpose of giving brief insights regarding basic concepts that support related work and technologies concepts described in the following sections.

2.1.1 Container Orchestration

As the industry kept adopting containerized strategies, other problems emerged due to the massive increase of containers to manage [6]. The awareness of the applications' status itself was another concern since the monitoring, updates without downtime, scaling, and handling of unexpected errors, among others were core parts of achieving reliable production environments.

Container Orchestration Engines (COEs) were developed to address the problems previously described. Thus, these technologies were revolutionary because managing numerous containers, each one with different purposes and dependencies, was tedious and time-consuming [7]. These orchestrators also provide insights into the application that is packaged as a container, which is important to troubleshooting, where logs and metrics come in very handy.

Typically, all the operations performed regarding containers in an orchestrator target multiple nodes in a network. The container orchestration engines abstract these operations but also the computational resources used by each container [6]. There are many container orchestration engines, however, the most popular and widely used are Kubernetes, Docker Swarm, Apache Mesos, and OpenShift.

2.1.2 Cloud Computing

According to the National Institute of Standards and Technology (NIST), cloud computing is “a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources that can be rapidly provisioned and released with a minimal management effort or service provider interaction” [8].

It is an evolving paradigm that characterizes relevant comparisons of cloud services and deployment strategies. The five essential characteristics that cloud computing leverages are: (i) on-demand self-service; (ii) broad network access; (iii) resource pooling; (iv) rapid elasticity and (v) measured services.

Concerning service models, multiple have been envisioned lately, but the three core models are:

- **Infrastructure-as-a-Service (IaaS)** – the consumer has to manage computing resources, and can deploy any application or software at will.
- **Platform-as-a-Service (PaaS)** – owned applications by the consumer can be deployed on-demand independently of the kind of application if the platform supports it. Infrastructure is not managed by consumers.

- **Software-as-a-Service (SaaS)** – provides applications running on a cloud infrastructure accessible from different client interfaces. Consumers cannot manage or control the underlying infrastructure.

Aside from the service models, four deployment strategies can be adopted when using cloud computing technologies:

- **Private Cloud** – infrastructure is provisioned for exclusive use by a single organization.
- **Community Cloud** – infrastructure provisioned towards the use of the community of clients that have shared concerns.
- **Public Cloud** – the general public can provision infrastructure, and it solely exists on-premises of the cloud provider.
- **Hybrid Cloud** – strategy composed of two or more different cloud infrastructures that are independent but complement each other.

2.1.3 Continuous Practices in Software Engineering

Continuous practices in software engineering refer to an area of research and practice that provides manners for developing, deploying, and getting constant feedback from software and the customer systematically and rapidly [1]. The most used and known deployment activities are:

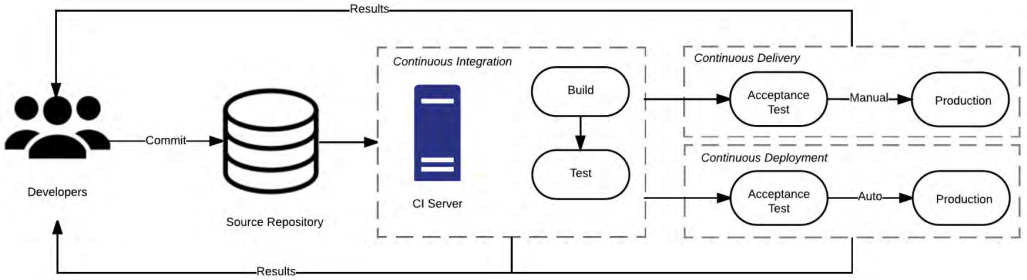


FIGURE 2.1: Relationship between Continuous Development Practices [1]

- **Continuous Integration (CI)** – development work is merged and integrated frequently. It forces teams to enter shorter release cycles, which improves software quality and team productivity. An example of this is the building and testing of the project and the production of its artifacts.
- **Continuous Delivery (CDE)** – ensures that application artifacts are always ready for entering the production state after the previous security/quality checks and tests in an automated way (the CI workflow represented in Figure 2.1). It has some benefits, such as reduced deployment risk, lower cost, and continuous feedback.

- **Continuous Deployment (CD)** – applications are automatically deployed after the CI process successfully finishes and typically target production environments. In CD every change is deployed automatically, without human intervention.

2.1.4 Infrastructure-as-code

Infrastructure-as-code (IaC) is a DevOps practice in which resources are managed and provisioned using code [9]. This practice substitutes manual configurations or low-level developed scripts that attempt to create a new environment.

IaC languages or tools can be categorized as declarative or imperative. In a declarative approach, developers describe the desired state of the environment, and the tool takes care of interpreting it and executing the proper actions to achieve the desired outcome. When an imperative or procedural model is used, developers must describe how to achieve the desired state step-by-step where the order of the steps is important. Some relevant properties of IaC that turn teams to adopting it are: (i) on-demand provisioning of computational resources; (ii) abstraction of APIs exposing them in a software-defined way; (iii) elimination of configuration drift, ensuring consistency among multiple environments; (iv) actual replacement of infrastructure when a new version is submitted; (v) rapidly and consistently recreate environments.

2.2 Related Work

This section presents related work regarding the orchestration of application deployments in hybrid environments and potential concerns. The work can be found repeatedly using the search queries and electronic databases mentioned in the introduction of this chapter.

2.2.1 Combination of Abstraction Models

Five engineers from Hanyang, Hongik and Changwon universities contributed with an article [10] that proposes a model-driven cloud application orchestration approach combining two different standard models.

Topology and Orchestration Specification for Cloud Applications (TOSCA) is an OASIS standard that pursues the portability specification of cloud applications. For promoting application interoperability and portability across different providers an application topology is defined following a platform-agnostic approach. This standard provides a type of system modeling applications' components and relationships to define the topology.

The Open Application Model (OAM) is "a runtime-agnostic modeling and a specification standard for defining cloud applications". It focuses on application definition instead of platform or infrastructure layer details, therefore it is considered application-centric. OAM was developed in a way that empowers modularity, extensibility, and portability, enabling application delivery to heterogeneous platforms.

OAM emphasizes the separation of roles' concerns when defining applications, and it is composed of two main reusable sections: the component and the trait. The component provides the application service and the operational specifications for that component, describing the runtime environment where the applications run (*e.g.* virtual machines, containers, cloud platforms). The trait is a runtime overlay that adds new operational and optional features (*e.g.* hostname specification, auto-scaling).

The solution presented in the article consists of the combination of the two models to achieve application orchestration. Using this strategy helps cover for the lack of role awareness and separation of concerns in TOSCA. OAM itself does not have a solid foundation regarding orchestration in the cloud and topology definition due to the lack of a robust meta-model. In the end, the two standards complement each other to build a model-driven, role-based cloud orchestration system.

2.2.2 Cloud Application Portability

T-Systems, Telekom, Innovation Laboratories, and the FZI Research Center for Information Technology created a team for carrying a Proof of Concept (POC) project regarding application orchestration among different platforms [11].

The goal of the project is to automate the configuration, coordination, and management of software in cloud infrastructures [11]. For this, the solution consists of four technologies: OpenStack, Chef, and Business Process Execution Language (BPEL) and Apache Orchestration Director Engine (ODE).

The project participants developed a component named TOSCA2Chef, that would create an execution environment capable of transforming a TOSCA model into a running application [11]. The component is responsible for interpreting the TOSCA file and executing the suitable commands to the Chef Knife client. This client communicates with the Chef server and the OpenStack infrastructure to orchestrate the deployment of the application.

2.2.3 TORCH

TORCH is a TOSCA-based framework for deploying and orchestrating applications on multiple cloud providers [12]. This is possible by transforming the TOSCA application model into an equivalent Business Process Model and Notation (BPMN) workflow.

It was built with extensibility in mind, diminishing coding effort by abstracting generic provisioning scenarios. Connectors are pluggable software into the TORCH framework that interpret these actions and transform them into a concrete deployment process [12]. The provisioning scenario consists of three sequential phases: (i) application specification, where the user submits the application requirements; (ii) orchestration, where the TOSCA processor transforms the requirements into a workflow plan; (iii) service binding, where the connectors transform the generic actions into a concrete deployment process.

2.2.4 Napptive

Napptive is an organization that developed a platform that allows deploying complex cloud native applications in a simplified way [13]. It abstracts low-level Kubernetes entities, consolidating multi-purpose and multi-tenant clusters. Users do not require any knowledge regarding the infrastructure and how it is provisioned since the deployment process and orchestration of applications are all performed by the platform.

The Napptive platform can achieve this level of abstraction due to KubeVela being its OAM runtime for Kubernetes deployments [13]. The use of KubeVela requires Kubernetes knowledge besides the abstraction it provides. Napptive is built on top of it, creating another layer of abstraction at the level of environments. The user can perform deployments using the web user interface or a Command-Line Interface (CLI).

Since KubeVela is extensible by design, the deployments are not bound to Kubernetes clusters. It is possible to provision and combine other technologies to achieve more complex deployments across hybrid environments. Therefore, the Napptive platform can create more complex scenarios to support their customers' needs.

2.3 Related Technologies

In this section, the technologies that are likely to be mentioned and used in coming sections are analysed and described to better understand their purpose and scope.

2.3.1 Containers and Orchestration

This subsection presents container-related technologies, namely tools for developing/shipping containers and container orchestration platforms.

Docker

As containers started to be adopted, the need to define industrial standards appeared. Docker is a project under the Linux Foundation since 2015, denominated by the Open Container Initiative (OCI) [14]. Although the OCI is just a specification, not a concrete implementation, Docker gave its container runtime, *runc*, to the OCI project to serve as a reference for future container runtimes. This means that Docker was and is always OCI-compliant [14].

Docker is an open platform for developing, shipping, and running applications. It was designed based on the virtualization strategy which adopts containers as mentioned before and it promises the fast, consistent delivery of applications, responsive deployments and scaling, and a low overhead providing running many workloads on the same infrastructure [15].

It started to act only as a container runtime, and nowadays, it consists of many components. The relevant components for this work are:

- **Registries** – where container images are stored. These can be private or public.

- **Images** – a template with instructions for creating a container. Provide additional customization and extensibility.
- **Containers** – an instance of an image. Every container instance is based on a specific image.

Kubernetes

Kubernetes (originated from a Greek word that means "helmsman, pilot") is the most widely used container orchestration technology. It was founded in 2014 by Google, based on previous attempts at the creation of internal container orchestration, namely Borg and Omega [16].

Kubernetes (K8S) uses containers as its base for running workloads, taking advantage of immutability, which is one of its key benefits. When a container image is updated, a new image is built from the beginning with no incremental changes [17]. Therefore, to update an application, the container instance needs to be replaced by new instances with the new version of the built image. This facilitates rollbacks in case of failures when updating versions.

Objects or resources are represented declaratively in Kubernetes. The Yet Another Markup Language (YAML) notation is used for this purpose. Each declarative file represents the desired state of the system. Kubernetes then ensures that the current state of the cluster matches the newly declared state.

Kubernetes is considered a self-healing system, which means it will continuously operate to maintain the current state consistent with the desired one. Beyond this feature, there are others equally important, such as service discovery, load balancing, storage orchestration, and secret and configuration management, among others.

In Figure 2.2 are represented the Kubernetes' components. When a cluster is formed, machines are divided into two types: node/worker and control plane.

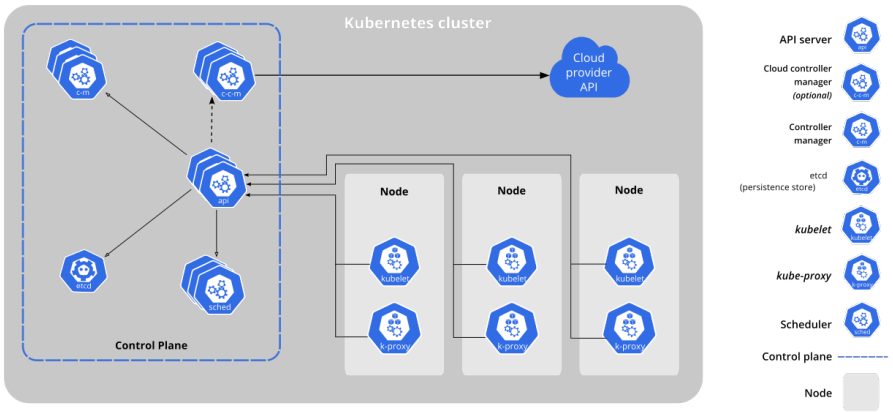


FIGURE 2.2: Components of Kubernetes [18]

The control plane is responsible for managing the global state of the cluster, ensuring worker nodes are working as they should, and reacting to certain cluster events that can occur [18]. The control plane consists of the following components:

- **Application Programming Interface (API) server** – exposes the Kubernetes API, the front end for the control plane, and is responsible for validating and configuring data for objects.
- **ETCD (Persistent Store)** – a high availability and consistent key-value store for storing all cluster data.
- **Scheduler** – watches for new unassigned Pods and depending on different factors decides which node the Pod should be running.
- **Controller Manager** – runs controller processes each one having well-separated responsibilities, to manage nodes, jobs, service accounts, etc.
- **Cloud Controller Manager** – embeds cloud-specific control logic which links the cluster into the cloud provider's API for the sake of separation of components. If the cluster is hosted on-premises there is no cloud controller manager.

The nodes often referred to as workers, consist of three main components: (i) Kubelet; (ii) Kube Proxy; (iii) Container Runtime. In general, they maintain pods running and provide the Kubernetes running environment.

As stated, Kubernetes works with objects that are defined with YAML notation. The following resources are the most used and considered the core to leverage the potential of Kubernetes:

- **Pod** – is the smallest deployable unit in a cluster. It contains a collection of application containers and volumes running in the same environment, sharing resources such as IP addresses, ports, and others. Containers in the same Pod always run in the same node [17].
- **Deployment** – manages release of new versions. It simplifies rollbacks and updates, typically referred to as rollouts. The rollout process waits for the new version of the Pods to be ready to substitute old replicas, ensuring no downtime.
- **Service** – abstraction which defines a logical set of Pods and a policy on how to access them. The set of Pods targeted by a Service is usually determined by a selector. A service can be one of four types: ClusterIP, NodePort, LoadBalancer, and ExternalName.
- **Ingress** – exposes HTTP and HTTPS routes from outside the cluster to services within the cluster. Traffic routing is controlled by rules defined on the Ingress resource. An ingress controller must be deployed for additional functionalities of more recent controllers.
- **ConfigMaps/Secrets** – a ConfigMap is an API object used to store non-confidential data in key-value pairs. On the other hand, a Secret contains sensitive data such as passwords, tokens, keys, etc. A pod can use these resources as files/folders and environment variables.

- **Namespace** – used to organize objects in the cluster. It acts like a folder, but instead of files, it holds a set of objects (pods, deployments, services, etc).

Rancher

Rancher is a multi-cluster management system that simplifies the management of multiple Kubernetes clusters anywhere and on any provider [19]. One of the most important features is the unified multi-cluster application management. It streamlines deployments across different clusters with centralized authentication, access control, and observability.

In addition to importing existing clusters into the system, Rancher also supports the provisioning of new Kubernetes clusters or even creating compute nodes, proceeding with the installation and configuration of a new cluster. Other features, such as backups, CI/CD integration, application catalogation, storage orchestration, policy enforcement, and others are included and applied to clusters that join the system.

Nomad

Nomad is a schedular and workload orchestrator that lets users deploy and manage any application type across hybrid environments [20]. The main features of Nomad are:

- **Efficient resource usage** – efficiently distributes the workloads across client nodes through a process known as bin packaging. This process is where the client is filled with items that maximize the utilization itself.
- **Self-healing** – monitor events and act according to the problem to bring up any down resources.
- **Zero downtime deployments** – it supports different update strategies to avoid downtime of applications.
- **Different workload types** – it allows the orchestration of different workload types. Nomad is extensible by design, and it is possible to add new task driver plugins for running customized workloads
- **Cross platform support** – it runs as a single binary and can be installed in different types of platforms (MacOS, Windows, Linux, etc).
- **Single unified and declarative workflow** – the workflow for deploying and maintaining applications is unified in a declarative job that specifies workload type and configuration, and other relevant attributes.

2.3.2 GitOps

GitOps is a model introduced by Weaveworks for operating cloud-native applications using the Version Control System (VCS) as a single source of truth [21]. The management of deployments is done by creating, updating, or deleting declarative files in a repository, similar to a developer-centric experience. This model leverages its use cases by adopting paradigms such as IaC and CI/CD practices, previously described in this chapter.

There are two different approaches for deployments: push-based and pull-based. The main difference between the strategies is how they ensure that the state of the infrastructure is consistent with the declaration present in the repository.

The push-based approach takes advantage of CI/CD tools (Azure DevOps pipelines, Jenkins, and others), and when changes are detected in the repository (pull request via merge), it triggers a build pipeline, compiling the software and ensuring the repository containing deployment artifacts are updated. The CI pipeline triggers a CD pipeline that will update the environment to the new desired state.

The pull-based strategy (Figure 2.3) relies on an entity often called an operator, which substitutes the CD pipeline, and this is mainly applicable in container orchestration platforms such as Kubernetes.

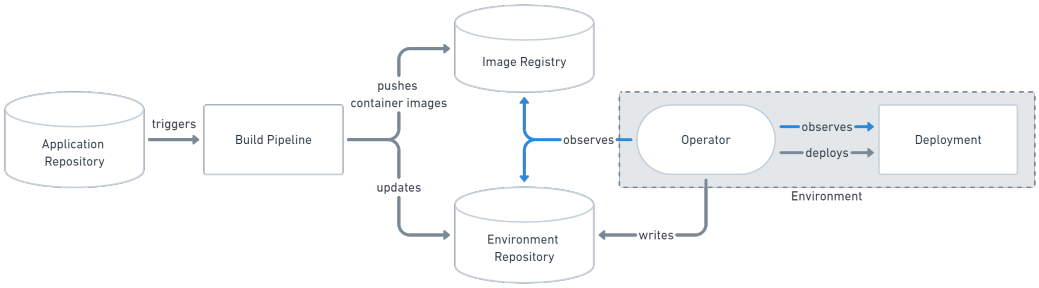


FIGURE 2.3: Pull-based deployment strategy [22]

The operator is responsible for constantly checking the repository, where the deployment artifacts reside, and changing the current state of the environment to the one described in the repository. This approach has several benefits compared to the previous one:

- Consistency between the desired and current state is always being checked by the operator, rapidly detecting deviations.
- Alerts can be configured to automatically be triggered when the state is not in a compliant state.
- Sensitive data exists only in the environment, removing the necessity to have confidential content in CD pipelines.

Flux

Flux (also known as FluxCD) is an open source, graduated project developed by the Cloud Native Computing Foundation (CNCF). It is a tool that helps synch Kubernetes clusters with the desired state, often declared in Git repositories [23]. Since it applies and maintains configurations that reside in places considered the source of truth, it follows a GitOps approach.

The deployment strategy that Flux adopts is the pull-based deployment thus, it inherits all benefits of the mentioned strategy, such as detecting deviations in configurations, alerts when such deviations occur, and more. It means an additional

entity (an operator) must exist within the Kubernetes cluster for analysing cluster state and pulling configurations whenever changes are detected.

It also integrates with different technologies, such as Prometheus and Grafana for monitoring deployments and reconciliation cycles, and Helm and Kustomize for deployment templating.

2.3.3 Infrastructure Provisioning and Management

This subsection describes infrastructure provisioning tools, such as infrastructure-as-code and management tools for Kubernetes multi-cluster orchestration.

Terraform

Terraform is an open source IaC tool that is used to define resources in declarative configuration files that facilitate the versioning, reusing, and sharing [24]. It was created and maintained by Hashicorp and developed using the Go programming language. This utility is responsible for interpreting the developed declarative files – which is the final state of the infrastructure – making the necessary API calls to one or more providers such as Amazon Web Services (AWS), Azure, GCP, and others.

Compared to other IaC tools, Terraform is a provisioning tool since it deploys infrastructure, but it does not guarantee that it will be fully configured at the end of the process. It inherits a declarative approach due to the HashiCorp Configuration Language (HCL) used for declaring resources, and it follows the principle of immutable infrastructure. This means that the state of the infrastructure must match what is declared in the actual code.

Open Cluster Management

The Open Cluster Management (OCM) is a community-driven platform that offers Kubernetes multi-cluster orchestration by being extensible and modular by design [25]. This management tool operates in a decentralized fashion – specifically in a hub-spoke architecture – where each managed cluster is responsible for its reconfiguration and policy enforcement. It achieves this by using the operator pattern in Kubernetes and the hub-spoke architecture used since managed clusters pull and reconcile needed information from the hub cluster.

The mentioned architecture, when applied with the pull-based strategy, minimizes the network requirements for registering a new cluster. Any cluster that can reach the hub cluster can join and be managed by the system. Therefore, private clusters that do not have their Kubernetes API exposed can be managed as well [26]. The hub cluster is the only one that must be accessible.

2.3.4 Application Management

This subsection presents technologies that focus on the configuration and management of complex applications in orchestration engines, such as Kubernetes.

Helm

Helm is the first application package manager that ships applications to Kubernetes [17][23]. This tool operates as other languages that create parameterized configurations. These configurations often receive a template - which hides all the extra configurations that can be automated - and a parameters file - interpreted by the template to generate the desired artifacts [17].

The packaging format used by Helm is called a chart, which is a set of files that describe related Kubernetes resources that compose an application [27]. In Helm, the concept of release is an instance of a chart associated with a configuration (input parameters). The release is tied to a specific version, and it is possible to upgrade or rollback it regarding the defined chart.

Managing applications with Helm offers a repeatable application installation, easy upgrades with custom hooks, simplified share of charts, and the possibility of rollbacks.

HyScale

HyScale is an open source application deployment tool that provides app-centric abstraction capabilities [28]. This technology offers different benefits, such as the generation of Docker files and images through package code, the generation of Kubernetes declarative files from an abstract specification, and automatically obtaining and troubleshooting application information and errors.

Although it is built towards a migration solution for applications, the abstraction layer that HyScale provides helps optimise and automate the management of application configurations and the respective deployment in multiple clusters.

This product has open source and enterprise versions. The difference is that the enterprise solution is built on top of the free version, additionally offering a PaaS service with multi-cloud management.

Crossplane

Crossplane is another CNCF community-driven project that focuses on automating application deployment on infrastructure regarding heterogeneous platforms [29]. Underlying managed services can be composed using high-level abstractions, making this technology a candidate for standardised deployment and provisioning of infrastructure across diverse cloud providers. Its backend is highly extensible, which enables building a control plane to orchestrate applications and infrastructure in a platform-agnostic way.

The community can contribute with new providers, extending the support of Crossplane. It focuses more on facilitating the creation of resources in multiple clouds but can be used to integrate with many tools such as Terraform, Ansible, Helm, ArgoCD, etc.

KubeVela

KubeVela is a cloud platform engine based on OAM specification, designed to leverage consistent application delivery on heterogeneous platforms [10]. As it uses Kubernetes as an abstraction layer, deployment artifacts are declarative, permitting the adoption of GitOps practices. The extension or refactoring of certain behaviours for components is possible due to the support of Configure Unify Execute (CUE) [30] language. Hybrid scenarios (multi-cluster, multi-cloud, etc) are supported by default, including the rollout of environments, canary, blue-green, and continuous verification strategy deployments.

The architecture of the technology is divided into several components, which are:

- **Core Controller** – core control of logic about the entire system (orchestrating deployments, garbage collecting, etc);
- **Cluster Gateway Controller** – multi-cluster access interface that ensures authentication and authorization in managed clusters;
- **Addons** – responsible for registering and managing definitions and Custom Resource Definition (CRD) controllers for extended functionalities.

OAM is being maintained by KubeVela, and the definition of an application is composed of four different parts: Component, Trait, Policy, and Workflow [31].

The component and trait sections are described in section 2.2. However, policy and workflow definitions are new to the system. The policy definition enunciates overlays of configurations, topologies, traffic rules, and more, affecting the application definition itself. The workflow section describes every step of the delivery process, which allows the trigger of alerts when a deployment occurs and other possible scenarios.

2.4 Value Analysis

Value Analysis is a process of systematic review used to compare the functionalities/characteristics of a product required by a specific customer, to meet their requirements. This has to be done keeping the lowest cost with the stated performance and reliability [32].

2.4.1 Innovation Process

The innovation process is composed of three sequential phases: the Fuzzy Front End (FFE), New Product Development (NPD), and commercialization (represented in Figure 2.4). Often chaotic, experimental, and unpredictable, the FFE stage is generally one of the greatest opportunities to improve the innovation process [33]. This is due to the dynamic nature of the process where new opportunities are identified and analysed, and from these, ideas are generated and selected.

The FFE was not developed as thoroughly as NPD regarding best practices. This underdevelopment caused a lack of common terms in the area, thus innovation process practices comparison between different companies was complex. With this, the difficulty of creating new knowledge and comparing different stages of the process

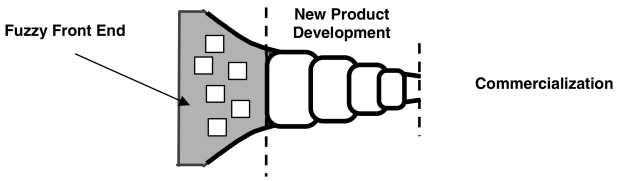


FIGURE 2.4: The Innovation Process [33]

increased, leading to the creation of a new model to standardize the insights and terminology used in the early stages of developing a new product.

The preferred model used for addressing the mentioned issues is the New Concept Development (NCD) (Figure 2.5). As described, the model was defined due to the need for a common vocabulary across companies, providing a clear definition of the market and technical requirements and an objectively defined business plan. The NCD construct is a relationship model, composed of three major parts: the engine, five activity elements around it, and influencing factors. Also, the model is represented in a circular shape highlighting the fact that new ideas are expected to flow, circulate, and iterate between and among all the five activity elements.

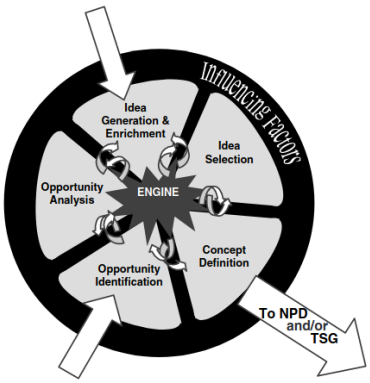


FIGURE 2.5: The New Concept Development (NCD) Model [33]

The engine is what sets the environment for successful innovation. It is composed of organization characteristics like leadership, culture, and business strategy that drive the five key elements that are controllable by the organization (opportunity identification, opportunity analysis, idea generation and enrichment, idea selection, and concept definition).

The five activity elements controllable by the organization are:

- **Opportunity Identification** – the organization identifies opportunities aligned with its interests, which normally are driven by business goals.
- **Opportunity Analysis** – the captured opportunity is analysed to confirm it is worth pursuing.

- **Idea Generation and Enrichment** – may be a formal process or not and concerns the generation and development in depth of a concrete idea.
- **Idea Selection** – from a pool of generated ideas, selecting the ones that are the most relevant for increasing business value.
- **Concept Definition** – the development of an investment case consisting of qualitative and quantitative information, often referred to as criteria, to support decisions.

The influencing factors consist of organizational capabilities, (that determine how opportunities, ideas, and concepts complement and generate each other), customer and competitor influences, the volatility of the outside world (laws, environmental regulation, etc), and the development of sciences and technology to regularly implement them into products.

2.4.2 Analytic Hierarchy Process (AHP)

The Analytic Hierarchy Process (AHP) is a general theory of measurement that helps quantify weights of decision criteria when comparing alternatives [34]. It segments the problem using a hierarchy model organized by decision levels, simplifying decisions and understanding which alternative suits a solution to solve the described problem.

This hierarchical model starts with the problem/focus and descends to criteria (or even more in-depth, subcriteria) which are considered relevant, ending at the possible alternatives where one will be selected. Alternatives may vary depending on the specified context/scope or criteria.

2.5 Summary

This chapter presents theoretical background about container orchestration, cloud computing, continuous practices and infrastructure-as-code, which supports advanced topics in the following chapters.

The SMR methodology is used to search for related work regarding the defined research questions. A total of 21 articles were considered relevant, and the presented related work describes investigations regarding the orchestration of applications in hybrid scenarios. The combination and transformation of different abstraction models (TOSCA and OAM), the creation of abstraction layers on top of other technologies or the development of a framework based on an abstraction model are some of the work found.

Finally, related technologies are described, and most are related and used in the context of the Kubernetes ecosystem. Also, the value analysis theory, including the innovation process and AHP is presented to support the next chapter's development.

Chapter 3

Analysis

In this chapter a value analysis is presented using part of the innovation process and the AHP to identify and analyse the business value of this project. Thereafter, a requirement analysis is done, using the FURPS+ method to gather functional and non-functional requirements that are relevant to develop a new continuous deployment process to orchestrate applications in hybrid environments.

3.1 Value Analysis

In this section the NCD model will be used to identify and analyse the opportunity using articles to justify such perspectives, ending with the concept definition, identifying customer segments and associated benefits and inconveniences. Having a clear and well-defined concept, the AHP method will be applied to identify which tool is more adequate according to the criteria, to meet most of the company's business requirements.

3.1.1 New Concept Development (NCD)

The innovation process briefly consists in producing an idea, thoroughly developing it, and commercialising it. As previously stated, the start of the process is often chaotic, which affects its future stages. Therefore, the NCD model aids in this early phase to gather and structure information, to define concrete concepts.

Opportunity Identification

The delivery of the product in perfect condition is a desired accomplishment for the customer but also for the entity that developed it. Nowadays, customers want new versions of products, with advanced functionalities more regularly, which creates a fast-paced development cycle. The software must be deployed quickly and seamlessly, so operations teams prepare, manage and maintain infrastructure to receive new software products.

Cloud computing surfaced not a long time ago, and many organizations adopted it, to ease the management of the infrastructure since the service provider is responsible for managing various aspects of the infrastructure and services, depending on what model they follow [35]. This implies that the end-user does not have to be concerned about the hardware, network configuration (to some extent) and other specifications

because it is being managed by a third party. Globally, the different cloud providers have been developing at fast rates [36] as shown in the Figure (3.1).

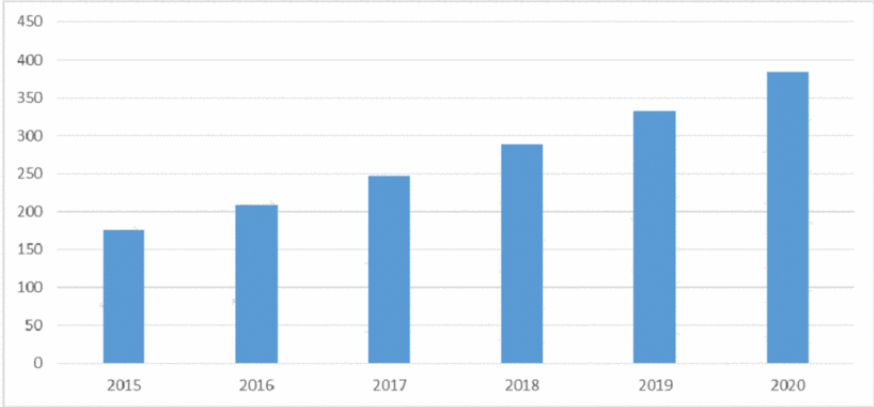


FIGURE 3.1: Global cloud technology market growth, in billions of dollars [36]

Also, compared to traditional IT infrastructure (on-premises or in-house), organizations prefer using public or private clouds over time, with highlighting on the public ones. The increase in private cloud use cases is often caused by data security policies and confidentiality (Figure 3.2).

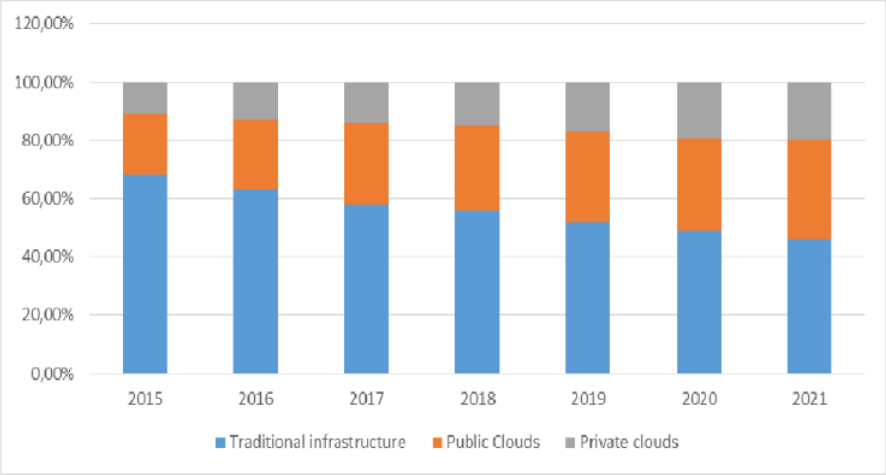


FIGURE 3.2: Cloud infrastructure growth by model [36]

However, due to organizational preferences, best practices or Disaster Recovery Plans, organizations tend to combine more than one service provider, including their on-premises infrastructure. This is because there are many benefits to adopting these hybrid models such as control, scalability, security, cost savings and business continuity [37].

An improvement of the deployment process using different technologies that are platform-agnostic could be a possible solution, focusing on interoperability standardization and federation between different providers [38].

Opportunity Analysis

As the infrastructure keeps scaling and spreading throughout platforms/environments, technology diversity increases causing difficulties in automating certain processes due to some specific challenges when adopting hybrid or multi-cloud strategies such as system/data backup and disaster recovery, auto scaling, system/data migration, resource monitoring and management and data protection [38]. In short, there are difficulties in generalizing infrastructure concepts or definitions, when having many of them to manage can lead to overwhelming tasks.

According to the information stated previously, the cloud computing market tends to increase in the next years, although many organizations certainly prefer to follow an hybrid approach due to data confidentiality, country laws regarding personal information, and other sensitive factors. Existing technology for managing applications and infrastructure can be integrated in the development of a new improved process which would, in fact, help operations teams organize and automate workflows (and sometimes, delegate tasks to software developers), while getting the most of the infrastructure regardless of its topology.

Normally, it starts with the customer service, where infrastructure can be a decisive factor in the quality of the final product [39]. Reliable and quick delivery of products are decisive criteria for customers committing to a product, including Service Level Agreements (SLA). By automating processes, IT management time and costs are greatly reduced, eliminating recurrent tasks that are time-consuming and error-prone, allowing to achieve a seamlessly and fast delivery of the product as mentioned previously. Finally, being able to effortlessly recover components or data (being it customer related or not) gives confidence when managing certain aspects of the infrastructure.

Although the benefits are huge when using these types of technologies, depending on the chosen cloud computing model, it can generate different concerns that organizations must be aware of. Vendor lock-in solutions, interoperability, data security, operational limitations, extensibility and control are the major concerns [40].

CloudFoundry is an open source PaaS that can be deployed on IaaS systems [41]. In their website are presented some known brands using their technology and exposed their user story. Samsung in one of that companies, and the latter event was the migration of Internet of Things (IoT) Developer Tools to CloudFoundry platform.

The mentioned organization believes that the integration of cloud technology is undoubtedly the latest wave to take hold in the industry. Most organizations have reached a point in time where mission-critical apps have been moved to the cloud [42]. Since they started tracking the topic of mission-critical apps (August 2016), 51% of respondents report that type of applications were running in the cloud and 45% still use legacy environments. After that, another study was issued in April

2017, and the organizations running critical applications in cloud increased by a total of 2%.

Organizations are adapting to different cloud technologies such as PaaS, containers and serverless workloads in various ways. Another research was done regarding this topic, and 41% of respondents answered that they are using PaaS, another 38% were evaluating using it and only 15% were not using technologies in that segment (Figure 3.3). Containers and Serverless technologies uses keep increasing but do not reach the community percentages as previously stated.

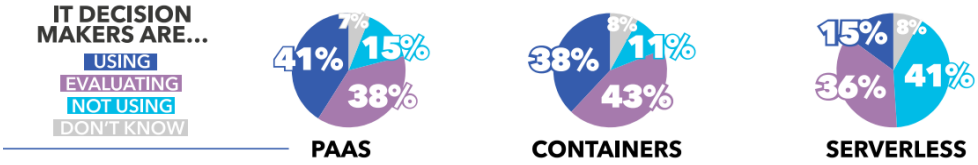


FIGURE 3.3: Research about IT Decision Makers about PaaS, Containers and Serverless [42]

The increase in use of these types of technologies at the same time, forces IT professionals in seeking tools that complement these models together. The combination of these concepts generates Multi-Platform Strategies. Almost half of the population that answered the questionnaire, exactly 48%, reported using a combination of the technologies previously described (Figure 3.4).

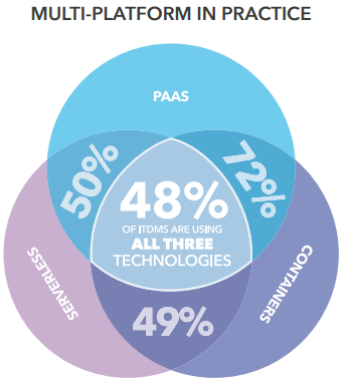


FIGURE 3.4: Market Research about use of Multi-Platform Strategies [42]

Concept Definition

After identifying and analysing the opportunity, idea generation/enrichment and selection is the next phase. However, these steps are being skipped because there is an idea that was previously selected with help of the organization (DevScope) where this project is to be developed. To determine if it is viable or not, there is a need to develop an investment case where qualitative and quantitative information [33] are gathered for sake of a decision.

Foremost, the analysis of the different customer segments' benefits and constraints regarding the possible solution is described on Table 3.1. Identifying the benefits and sacrifices of the different target segments is important, leveraging discussions regarding how the benefits can be delivered to the customers and possibly avoid constraints or difficulties.

Table 3.1: Benefits and Constraints by Customer Segment

Customer Segment	**Benefits**	**Constraints**
Software Developers	Deploy applications easily with less infrastructure or workflow details; Implemented observability out-of-the-box of deployed software applications and services.	Having to learn a specific syntax and semantics of combined specifications.
Operation Teams	Allow creation of new plugins for supporting new use cases; Simplified overview and management of the infrastructure; Increase component/template reusability.	Having to learn a specific syntax and semantics of combined specifications; Less control over specific details of the infrastructure.
Organization	Decreases time of delivering applications continuously; Less need of systems/infrastructure engineers.	High investment and low return in the first phases of the project; New employees have to be introduced to a newly developed tool.

Regarding the development or improvement of a continuous deployment process, the first phases of development of a POC, consist in gathering information about the featured topic using different sources, such as presentations, conferences, articles in electronic databases (*e.g.* ACM Digital Library, B-on, Google Scholar) and books. The use of grey literature will be avoided, however, depending on the source and on the availability of resources, it will be considered and referenced in the bibliography.

After information gathering, the focus must be switched to the POC itself. Requirements are to be gathered and described in section 3.2 that will affect how the solution is to be developed. Also, different design or model approaches can be taken, therefore an analysis must be done and after identifying the pros and cons of each one, a strategy is to be selected.

The implementation phase will consist of the development of a new improved continuous deployment process, that will consider the mentioned analysis and design processes, and also technical specifications that can affect the solution, such as programming languages, communication interface types, and supported environments, among others. The process will be tested by attending to existing scenarios in the host organization and also other employees, from different teams, will use it to get potential feedback and improve the POC.

Finally, the produced artifacts must be evaluated to ensure that they meet the objectives and requirements previously defined. In addition to the qualitative analysis,

a quantitative approach can also be taken considering the state of the product and the relevance of such metrics.

3.1.2 Analytic Hierarchy Process (AHP)

Accordingly, the first phase is the definition of a hierarchical decision tree with the levels described previously, which is represented in Figure 3.5.

Three main tools can contribute to building a new continuous deployment process, where each one, consequently, satisfies different needs of the host organization. Having said that, the problem is knowing what tool will fulfil the organization's needs.

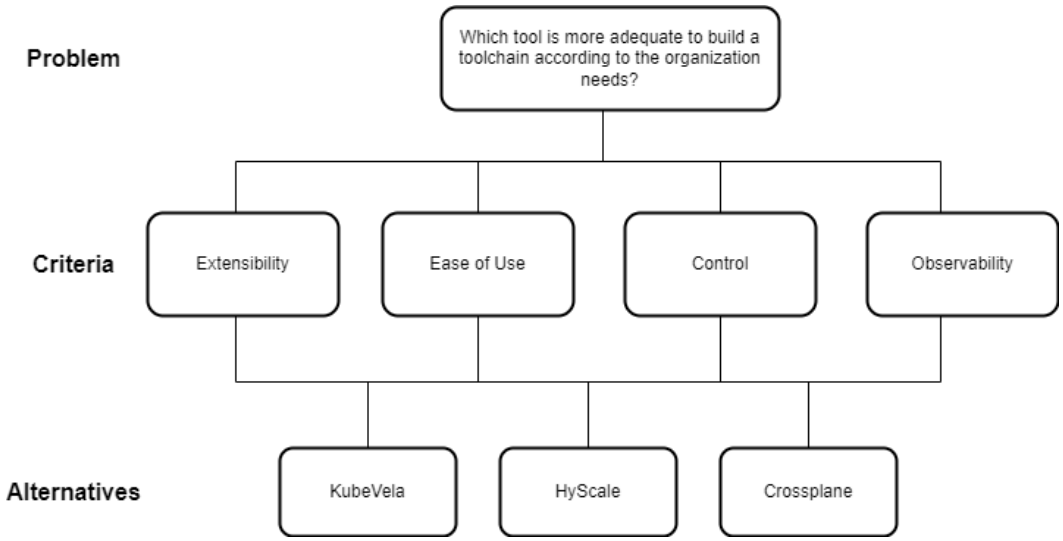


FIGURE 3.5: Hierarchical Decision Tree

The diverse criteria (second level) represent characteristics/requirements that are most relevant to constitute the product. They will be used to compare each listed alternative, and these criteria must share the same level of importance. In-depth, these are:

- **Extensibility** – easily extendable when comes to adding extra functionality or supporting new infrastructure, services, etc (*e.g.* plugins).
- **Ease of Use** – easy to use and understand without knowing infrastructure details.
- **Control** – the possibility of fine-tuning infrastructure details if needed.
- **Observability** – constantly monitor infrastructure and applications quantitative and qualitative metrics.

Finally, the options/alternatives are represented in the third level. With the help of this process, one of the listed alternatives will be highlighted as the recommended choice: (i) KubeVela, (ii) HyScale or (iii) Crossplane.

Moving on to the next stage starts with the definition of a fundamental scale to establish a priority among the different hierarchic levels. The scale that will be used throughout the process, for both criteria and alternatives, is presented on Table 3.2, as well as their definition and meaning.

Table 3.2: Fundamental Scale - Importance Levels of Comparison [34]

Rating Scale	**Definition**	**Description**
1	Equal importance	Two elements contribute equally to the objective
3	Moderate importance	Experience and judgement slightly favour one element over another
5	Strong importance	Experience and judgement strongly favour one element over another
7	Very strong or demonstrated importance	An element is favoured very strongly over another; its dominance demonstrated in practice
9	Extreme importance	The evidence favouring one element over another is one of the highest possible order or affirmation
2 4 6 or 8	Intermediate Values	When looking for a compromise condition between two definitions

Defined the rating rules for comparison between priorities, all criteria must be compared to compute the priority vector. Therefore, considering the four criteria of the hierarchical structure previously presented, the following pairwise comparison matrix was created (presented on Table 3.3):

Table 3.3: Criteria comparison in pairs

	Extensibility	**Ease of Use**	**Control**	**Observability**
Extensibility	1	1/2	1/2	4
Ease of Use	2	1	1	2
Control	2	1	1	3
Observability	1/4	1/2	1/3	1
Sum	21/4	3	17/6	10

After comparing each pair of criteria with the aid of a fundamental scale, it is possible to compute the weight for each criterion. The comparison matrix is normalized, by adding up the values of a column and dividing each cell of the same column by the result of the sum. Thereafter, the average of the values per row is computed, originating the weight for the given criteria. The obtained results, namely the priority vector, are presented on Table 3.4.

To ensure the priority vector is valid, it is mandatory to evaluate its consistency. This is done by calculating the Consistency Index (CI), which is used to compute

Table 3.4: Priority Vector (priority/weight of each criterion)

Criteria	Weight/Priority
Extensibility	0.233
Ease of Use	0.317
Control	0.342
Observability	0.108

the Consistency Ratio (CR). If the CR is less than the threshold (0.1) defined by the process, the judgements made in the comparison of the criteria are trustworthy. The obtained value for the CR was approximately 0.074, which is less than the enunciated threshold, concluding the calculated priorities are trustable.

The same process is repeated for each alternative considering the different criteria (excluding the consistency check step). In the end, after obtaining the priority vector associated with an alternative, these are appended in a matrix as columns, and it is multiplied with the estimated criteria weights (Table 3.4). These provide the alternative composite priority, shown on Table 3.5:

Table 3.5: Alternative Composite Matrix and associated Priority

	Extensibility	Ease of Use	Control	Observability	Priority
KubeVela	0.444	0.608	0.286	0.648	**0.464**
HyScale	0.111	0.120	0.140	0.122	0.125
Crossplane	0.444	0.272	0.574	0.230	0.411

Each row represents the weight of each criterion per alternative and in the final column the composite priorities. These values provide an answer to the previously defined problem. According to the results of the composite priority, one concludes that the tool to be used to help in the improvement of the process, which can manage multi-platform systems, should be KubeVela. In addition to the demonstrated matrices and vectors, further calculations were done regarding the described process which can be consulted on Appendix A.

3.2 Requirements Engineering

When developing new products or processes, it is best practice to gather functional and non-functional requirements. A good requirement analysis phase will positively impact the analysis, design, and, subsequently the implementation of the solution. It also helps organise which requirements are important to the stakeholders, what concerns they have, and constraints imposed by internal and external factors. For this, the FURPS+ approach is used. The main functional requirements are:

FR-1. Join and manage new Kubernetes clusters
The system must allow the management of new Kubernetes clusters independently of network restrictions of such clusters (publicly exposed or not).

This means that on-premises clusters which normally are not publicly exposed should take workloads from a central entity.

FR-2. Orchestrate application deployments across different platforms
The system must be able to orchestrate application deployments in the supported platforms (*e.g.* Kubernetes clusters, cloud resources, etc). One or more entities must be responsible for reading custom resource objects in Kubernetes and deploying the components attending to specified policies and workflows. It must also provide a way to create new resource definitions that will output one or more resources needed for the good operation of the application.

FR-3. Collect and store metrics/logs of applications
Whenever a new application is deployed within the system, additional configurations must be created to ensure that metrics and logs are collected and stored in long-term storage. These data must be queriable at all times. Also, the monitoring and logging of the application can be disabled at any time by changing the application specification. A centralized system must be responsible for holding metrics and logs and maintaining backups of these.

FR-4. Watch for new and reconcile existing deployment artifacts
The system must be able to read configurations from Git repositories (the source of truth) and apply them. If there is any configuration drift between the current state of the application deployments and the desired state described in the repository, such deployments must be reconciled. It must be possible to separate different parts of the declaration file - the application specification from the policies and workflows applied.

FR-5. Backup and restore deployment artifacts
Deployment artifacts, components, policies and workflow definitions must be backed up remotely. This backup must be restorable at any time and the backup policy must be configurable.

Functional requirements certainly impact the design and implementation process, but other variables influence such stages. These factors are non-functional requirements, which are extremely important since they provide a certain degree of quality measurement regarding the software to be developed. Table 3.6 describes and categorizes the non-functional attributes of the project to be developed.

Table 3.6: Non-Functional Requirements

ID	Category	Subcategory	Description
NFR-1	Usability	Learnability	Users that intend to deploy an application must be able to accomplish it without any restraint with auxiliary documentation.

NFR-2	Reliability	Predictability	The result of a deployment intent without any modifications must always be the same (assuming the deployment destinations are functional).
NFR-3	Reliability	Availability	Despite the conditions that may affect the system, it must always work stably.
NFR-4	Performance	Speed	The system must deploy the artifacts on time (depending on the scale of the project, should be around 10-20 minutes).
NFR-5	Supportability	Extensibility	Must be easy to add new functionalities to the system, considering the communication with new heterogeneous environments (new clouds, platforms, etc)
NFR-6	Supportability	Maintainability	The management of multiple declarative files related to applications and infrastructure must be straightforward, with a low number of dependencies between files.
NFR-7	+	-	The use of Kubernetes to manage and orchestrate resources is mandatory.

3.3 Summary

In this chapter, a value analysis was conducted to identify and analyse the value of the developing project. Identifying and analysing the opportunity and defining the concept were some important steps of the NCD model that helped structure and fundament target customer segments, considering their benefits and constraints.

Then, the AHP was used to decide which technology is more adequate to integrate into the process to be improved by attending to the organization's needs. KubeVela was the option with a higher priority, but Crossplane's priority came close to it, concluding these two technologies could be combined.

Finally, the functional and non-functional requirements were identified and deeply described. For this, the FURPS+ methodology was used to categorize each requirement that is relevant, as well as constraints and concerns that will limit the options in the design phase.

Chapter 4

Design

Following the steps of the DSR methodology, this chapter covers the design and development phase. Before the implementation/development phase, a design of the system artifacts according to the functional and non-functional requirements is done in this chapter. It contains the system's architecture, describing how it is structured and what components will exist within it, including communications with external systems. For this, the Unified Modelling Language (UML) notation is used, jointly with the 4+1 architecture view model for describing different perspectives of the system and the C4 model for adding variation regarding the abstraction level.

4.1 Architecture

The architecture is greatly influenced by functional and non-functional requirements. These were identified and described in 3.2, which add constraints on how the system will end up being structured. NFR-7 states that the use of Kubernetes is mandatory to manage and orchestrate resources. Figure 4.1 presents a proposal of the system's architecture in a logical view:

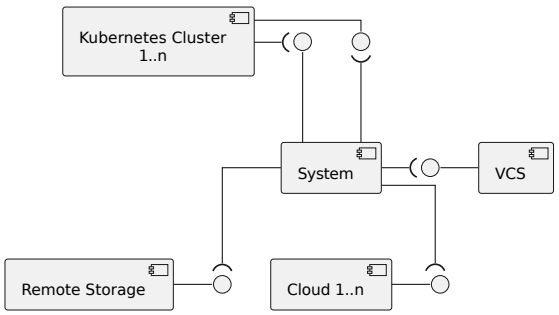


FIGURE 4.1: Logical view of the system at context level

According to the functional requirements, existing and new Kubernetes clusters must be managed by a central entity (FR-1). This entity will be another cluster that will act as a hub. Other clusters will be referred to as spokes. The spoke instances pull assigned workloads by the hub cluster. Figure 4.2 describes how the system's components will be distributed across the clusters according to their type.

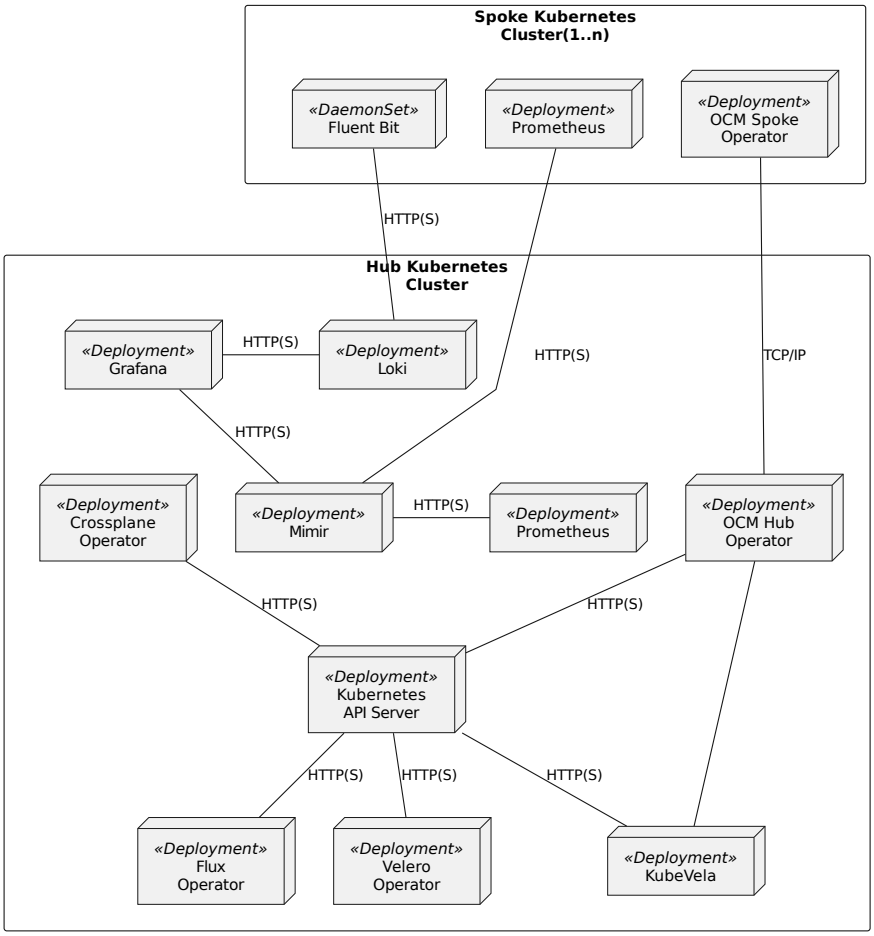


FIGURE 4.2: Deployment view of the system at container level

The operator/controller pattern [43] can be applied to extend the Kubernetes API. It also emphasizes the use of microservices, so each component can be considered a microservice having its responsibilities, following the Single Responsibility Principle (SRP) [44]. The system is entirely built using a Components-of-the-Shelf (COTS) strategy which focuses on using as many tools available to satisfy requirements.

The Kubernetes API server and the ETCD are core components of the Kubernetes system itself and they play an important role in this solution since all the deployment processes depend on native and custom Kubernetes objects interpreted by operators.

As stated, the topology of the solution will follow a hub-and-spoke strategy. The hub cluster is a coordinating entity, and the spokes are entities managed by it. This means the solution is, from a specific perspective, centralized since the hub orchestrates application deployments, stores and displays long-term metrics and backs up its state. The benefits of following such an approach are: (i) the hub ensures that spoke clusters are working towards a common goal, minimizing duplication of efforts; (ii) computing resources are allocated effectively; (iii) possibility of creating policies to load balance application traffic between spokes.

4.2 Requirements

This section presents the design for each functional requirement to better understand how the components operate together to satisfy a specific requirement.

4.2.1 Cluster Management

Regarding the cluster management requirement, KubeVela and OCM are essential components that work together to manage different clusters apart from their location and network restricions. Figure 4.3 describes how the components are related from a logical perspective.

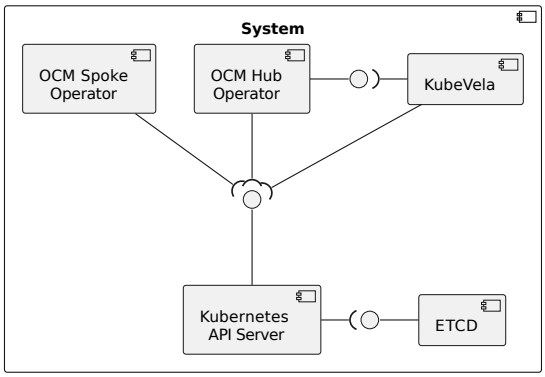


FIGURE 4.3: Cluster management design (logical view at container level)

KubeVela acts as an abstraction and communicates with the OCM hub operator to join new clusters and manage existing ones. The hub operator will be available to registrations of new clusters and these can be accepted or rejected.

The OCM spoke operator has the responsibility for pulling configurations from the hub cluster. This is done using Kubernetes objects, so almost all communication is done through the Kubernetes API Server. This component is present in all clusters and communicates with the ETCD, the Kubernetes default database that stores deployed resources.

4.2.2 Application Orchestration

KubeVela and the Crossplane operators reside in the hub cluster for orchestrating applications across platforms. Figure 4.4 presents the design regarding this functional requirement.

KubeVela is responsible for abstracting application definitions and interpreting these for deploying applications successfully. It is extensible, enabling teams to create new definitions, each having its own behaviour (NFR-5). KubeVela also offers a graphical user interface for managing deployments as an addon, contributing to the solution's usability (NFR-1). Since Kubernetes is a self-healing system and the KubeVela operator reconciles applications periodically, the system will work as expected (NFR-3).

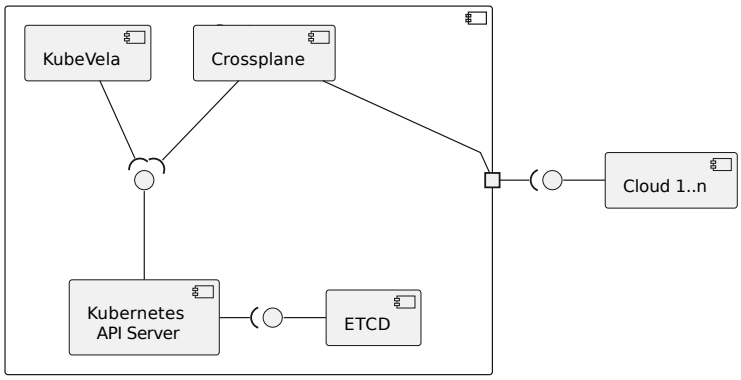


FIGURE 4.4: Application orchestration design (logical view at container level)

Combining the extensibility of KubeVela with Crossplane, it is possible to declare new cloud resources with the same level of abstraction to be deployed. This is done through templates that will abstract custom resources defined by Crossplane. KubeVela uses these templates to generate them, and Crossplane will detect new resources created in the cluster. Then, it will perform the suitable procedures to create the intended resources in the cloud.

4.2.3 Metrics and Logs Collection

One major concern about the orchestration of applications is the monitoring and logging solutions. Different technologies are used to ensure metrics and logs are all collected correctly according to the cluster they reside in. As described in section 4.1, most components will run in the hub cluster, and only collector agents will run in the spokes. Figure 4.5 describes how the system is logically structured.

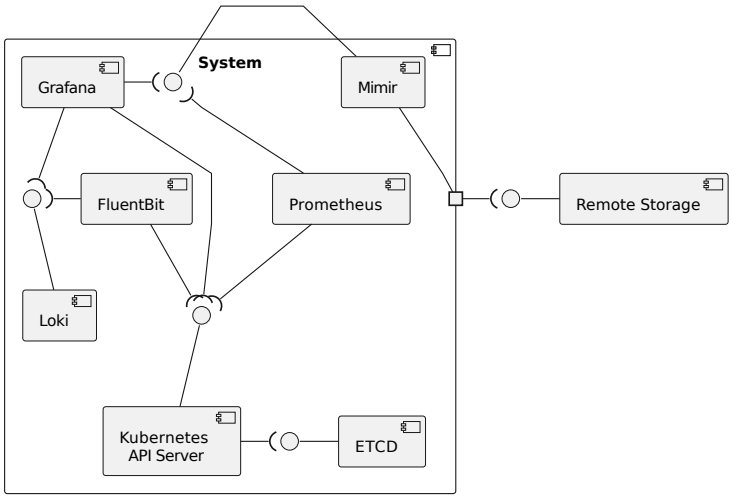


FIGURE 4.5: Metrics and logs collection design (logical view at container level)

The components that reside in the hub cluster are:

- **Mimir** – collects large volumes of metrics sent by Prometheus replicas residing in multiple clusters. It is also responsible for storing these metrics in an off-site storage solution (typically, in the cloud).
- **Loki** – stores and queries logs from applications and infrastructure.
- **Grafana** – displays dashboards and information about the applications and respective infrastructure by querying Mimir and Loki instances.

The components that reside in spoke clusters are:

- **Prometheus** – scrapes application and infrastructure metrics. Prometheus replicas are deployed across multiple clusters and send collected metrics to a long-term storage system called Mimir.
- **Fluent Bit** – collects, parses and writes logs from applications and infrastructure to Loki, the log aggregation system.

4.2.4 Artifact State Reconcilement

By adopting a GitOps approach, applications' artifacts will be stored in Git repositories. As mentioned, GitOps methodology can follow different strategies, such as pull and push-based.

A pull-based strategy is adopted for this requirement to ensure consistency between the state in the repository and the system's state (the hub and associated spokes). This ensures the result of deployment is always the same (NFR-2), also decreasing deployment time due to the constant verification of differences (NFR-4). Figure 4.6 presents the logical view regarding the communication between the components.

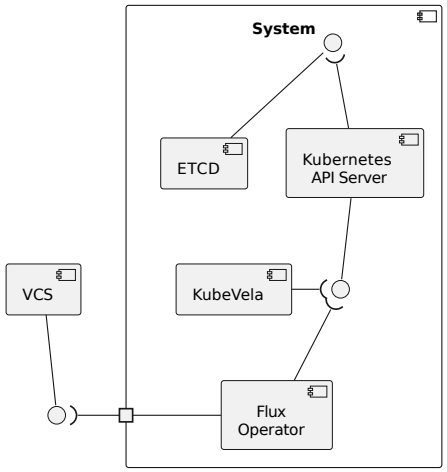


FIGURE 4.6: Artifact state reconcilement design (logical view at container level)

Flux is responsible for interpreting what deployment artifacts it must read and applying them to ensure the state of the repository is reflected on the hub cluster.

Most artifacts will be custom resources from KubeVela therefore, the declared resources will be read from KubeVela and applied as expected in the correct hub cluster. Applications' definitions are separated into several files, making the process of deployment straightforward since the commit of new artifacts in specific folders triggers new deployments into the platform (NFR-6).

4.2.5 Artifact Backup and Restore

The hub cluster is responsible for holding all the artifacts and dispatching them to spoke clusters and supported cloud vendors. Artifacts created in spoke clusters have no risk of loss since the state resides in the hub cluster.

Therefore, the creation of backup policies must be a possibility. Since the solution follows a GitOps approach, the repositories serve as one layer of backups. In case of unavailability of this first layer, the state of the hub cluster is backed up and restored at any time (NFR-3). The design of this functional requirement is demonstrated in Figure 4.7.

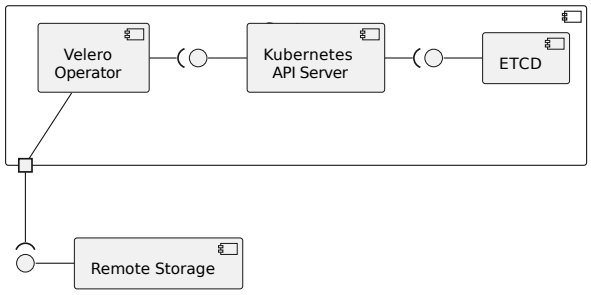


FIGURE 4.7: Artifact backup and restore design (logical view at container level)

Velero is the technology used for scheduling backups and restoring the state of the cluster. This component acts as an operator, changing its behaviour regarding custom resources configured on the system. A default schedule policy must be created to guarantee every deployed system has a restore point.

4.3 Summary

An architecture for the solution was presented using different views, namely the logical and deployment views, for understanding how the system must be structured to improve the deployment process.

According to the presented diagrams, the components will reside across multiple Kubernetes clusters. These clusters will have different roles following hub-spoke and artifact pull-based strategies. A COTS approach is taken, meaning that technologies and tools that already exist will be preferred. These components combined form a solution that will satisfy all the requirements stated in Chapter 3.

Chapter 5

Implementation

This section is part of the design and development phase of the DSR methodology. This includes the development of the solution which follows a COTS approach. The accomplishment of requirements specified in section 3.2 will be described here following the proposed architecture and design in Chapter 4. The following technologies were used for the development of the solution:

- Terraform for the provisioning of the infrastructure that will manage applications and the setup of new clusters.
- KubeVela for orchestrating application deployments across platforms, extending the functionality of these using CUE language and the OAM abstraction model. It allows the visualisation of application deployments and respective components.
- OCM Hub and Spoke operators are used for dispatching artifacts to multiple clusters without the need to expose them publicly.
- Flux for reconciling the desired state with the current state in the hub cluster.
- Crossplane for deploying resources in different clouds and platforms using Kubernetes custom resources.
- Prometheus for scraping infrastructure and application metrics.
- Mimir for collecting and aggregating metrics from multiple clusters.
- Loki for storing and querying applications' logs.
- Fluent Bit for collecting, parsing and exporting logs to Loki.
- Grafana for querying both metrics and logs from the infrastructure and applications residing in it.
- Velero for backing up the state of the hub cluster and restoring it at any time.

All of the components used in the solution are configured aside from the creation of new definitions using KubeVela, CUE language and the abstraction model OAM. The Terraform code present in this chapter was fully developed for this project, and it is used for provisioning and configuring all the components to make them work together towards the orchestration of applications.

5.1 Cluster Management

As stated, the solution follows a hub-and-spoke strategy to manage multiple clusters in a centralized manner. Terraform is used for reproducibility of the infrastructure and maintenance of the latter. The code is separated into modules to increase reusability across different instances of the infrastructure, also helping to keep them updated with the new improvements. Updating the state of the infrastructure will be reflected across all existing instances when adding more components.

Regarding the Terraform code in the repository, the organisation consisted of two main folders: "infrastructure" and "modules". The first one contains the desired state of each hub cluster, and within each hub cluster, many spoke clusters may exist. The second one contains the modules that are used to achieve reusability across the different instances of the infrastructure. The "hub" and "spoke" modules are used to set up clusters according to their role, and the others are used to create Kubernetes clusters in different clouds that will act as hubs. For now, the only one that exists is the "kubernetes-azure". By design, more modules - that provision clusters in different clouds - can be created without affecting other pieces of code (hub and spoke specifications). The folder structure described can be analysed in Figure 5.1.

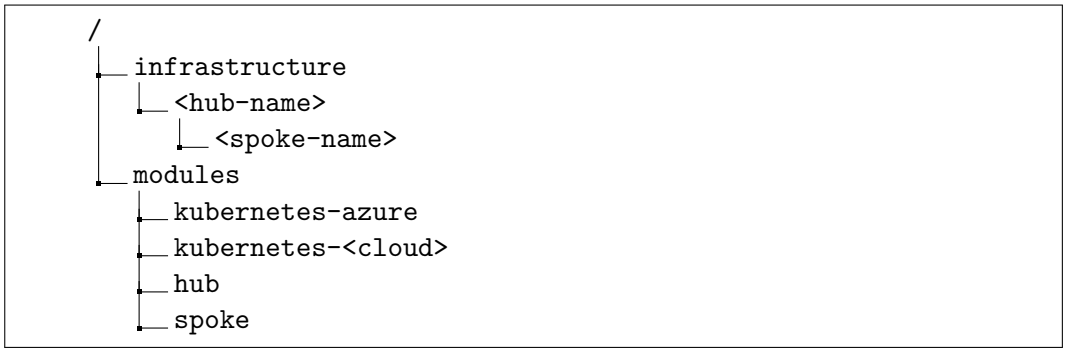


FIGURE 5.1: Infrastructure code folder structure

5.1.1 The Hub

The hub is responsible for managing the application deployments across different platforms and managing joined spoke clusters. This is all possible due to KubeVela, which provides a new way to define applications and extend capabilities in the deployment process. The OCM also operates with KubeVela since they both work to achieve multi-cluster deployments.

By default, a cluster is created in the cloud since a prerequisite of the hub-and-spoke strategy is that the hub must be accessible by any spoke. By creating new modules that contain the code to create resources in specific clouds, it is possible to provision clusters in other clouds. Terraform supports output values, making data available to other modules or on the command line. Every module, according to the used provider, must declare the following output for the subsequent modules to work:

```
1 output "kubeconfig" {
2   value     = azurerm_kubernetes_cluster.main.kube_config_raw
```

```
  sensitive = true
}
```

Listing 5.1: Kubeconfig output value for Azure Kubernetes Service (AKS)

The hub module ensures the target cluster has at least one ingress controller since access to the KubeVela dashboard and other applications inside the hub cluster is required. The installation of KubeVela is done using a helm chart. Some values must be set aside from the default ones for multi-cluster features to work as expected. The code listing 5.2 shows how to deploy the helm chart using Terraform.

```
resource "helm_release" "kubevela" {
  name             = "kubevela"
  repository       = "https://charts.kubevela.net/core"
  chart            = "vela-core"
  namespace        = "vela-system"
  create_namespace = true
  wait             = true

  set {
    name = "featureGates.preDispatchDryRun"
    value = "false"
  }

  set {
    name = "multicluster.enabled"
    value = "true"
  }
}
```

Listing 5.2: KubeVela Helm Release

KubeVela has the concept of addons. These bring functionality without the need to install and integrate them manually. Some of them were required to be installed which is the case of OCM components, Dex for OpenID authentication and the VelaUX, which is the Graphical User Interface (GUI) for KubeVela. The code listing 5.3 represents how addons can be configured.

```
module "hub" {
  ...
  addons = [
    {
      name = "velaux",
      parameters = [
        { name = "gatewayDriver", value = "nginx" },
        { name = "domain", value = local.hostname_suffix }
      ]
    },
    { name = "fluxcd", version = "2.3.4" },
    { name = "dex" },
    { name = "ocm-hub-control-plane" },
    { name = "ocm-gateway-manager-addon" },
  ]
  ...
}
```

Listing 5.3: KubeVela addons specification

For installing the addons, the KubeVela CLI is necessary. Since it is preferred to avoid the installation of additional tools where commands are executed manually, a Kubernetes job is created based on a Docker image that has the necessary software.

Therefore, a script is generated using a Terraform template created to iterate over the specified addons and install them using the CLI. The following code listing (5.4) shows the used template to generate the script, injected into the Kubernetes job.

```
#!/bin/bash
%{ for addon in addons }
vela addon enable --yes ${addon.name}%{ for param in addon.parameters } ${
    param.name}=${param.value}%{ endfor ~}%{ if addon.version != "" } --
    version ${addon.version}%{ endif ~}
%{ endfor ~}
```

Listing 5.4: Script template for installing KubeVela addons

Dex is also configured for adding authentication and authorization to VelaUX using OpenID connect. To achieve this, another template is used to create a configuration in Kubernetes. Also, an application registration is created by code in Azure to fill that template. The described code is present on Appendix B.

5.1.2 The Spokes

Spoke clusters are responsible for pulling workloads assigned to them from the hub cluster. As mentioned, this is possible due to OCM capabilities, from registration of multiple clusters to workload distribution.

The benefit of using OCM is that managed clusters can be private - their Kubernetes API is not exposed to the Internet - and can join the hub cluster if they can reach it. This also implies that the hub cluster has its API exposed. For running this Terraform code against private clusters the user needs to ensure that he has connectivity to that cluster (*e.g.* connect to Virtual Private Network (VPN)).

Since KubeVela uses OCM to manage multiple clusters, the registration of new ones must be issued to the KubeVela operator. Therefore, similar to the hub setup, the spoke will also use the KubeVela CLI through the developed Docker image to join the cluster. Once again, a Kubernetes job is created in the target cluster to perform the join operation. Both hub and spoke kubeconfig (instructions and credentials to communicate with the Kubernetes API) must be available within the job. The hub kubeconfig is obtained using the output value defined in code listing 5.1 but since the spoke infrastructure has its state, the output is not available directly. For this, the output must be read from the remote stored state of the hub infrastructure (see code listing 5.5).

```
data "terraform_remote_state" "hub" {
  backend = "azurerm"
  config = {
    tenant_id            = "<my-tenant-id>"
    subscription_id      = "<my-subscription-id>"
    resource_group_name  = "<my-resource-group>"
    storage_account_name = "<my-storage-account>"
    container_name       = "canaveral"
    key                  = "infrastructure/hub/<hub-name>/terraform.tfstate"
  }
}

module "spoke" {
    ...
    hub_kubeconfig      = data.terraform_remote_state.hub.outputs.kubeconfig
}
```

Listing 5.5: Read terraform hub remote state

An example of a complete manifest of the Kubernetes job can be viewed on Appendix B. The script template mounted in the job used for joining the hub cluster is shown in the code listing 5.6.

```
#!/bin/bash
export KUBECONFIG=/root/.kube/hub.yaml
vela cluster join /root/.kube/${spoke_cluster_name}.yaml \
    -t ocm --in-cluster-boostrap=false \
    --name ${spoke_cluster_name} \
    --yes
```

Listing 5.6: Script template for joining hub cluster

The command must be run against the hub cluster therefore, the "KUBECONFIG" environment variable is set to the hub kubeconfig. Also, a specific flag is added, demanding the use of the OCM as the multi-clustering engine.

5.2 Application Orchestration

KubeVela was chosen due to its nature of orchestrating applications. By design, it is also extensible, which means that combined with other technologies, it can satisfy a variety of scenarios. This is due to the use of CUE language, which helps KubeVela encapsulate and abstract the capabilities of Kubernetes.

In this case, KubeVela was combined with Crossplane to orchestrate applications and infrastructure independently of the platform. Crossplane takes advantage of the extensible Kubernetes API to define custom resources and make it possible to provision infrastructure in any cloud through declarative YAML files. The deployment of Crossplane is done using a helm release resource (see code listing 5.7).

```
resource "helm_release" "crossplane" {
  name = "crossplane"
  repository = "https://charts.crossplane.io/stable"
  chart = "crossplane"
  namespace = "crossplane-system"
  create_namespace = true
  wait = true

  dynamic "set" {
    for_each = var.crossplane_provider_packages

    content {
      name = "provider.packages[${count.index}]"
      value = each.value
    }
  }
}
```

Listing 5.7: Crossplane deployment through helm release

The hub cluster is responsible for holding the Crossplane operator, and therefore, responsible for provisioning cloud infrastructure. Multiple provider packages can be installed at deployment time if passed as variables to the hub module. A dynamic

block in Terraform allows it to span multiple blocks or parameters using a loop. In this case, it is used to set an array of packages to be installed.

Although providers' packages are installed, they must be configured since they require credentials and other settings to build a provider config. Creating these settings manually and replicating them across instances would lead to prone-error configurations. These provider configurations can be provisioned in a consistent and secure way, using an Azure Key Vault to store the credentials. Terraform data blocks are used to fetch these credentials as shown in code listing 5.8.

```
data "azurerm_key_vault" "main" {
  name = "devscopekeys"
  resource_group_name = data.azurerm_resource_group.main.name
}

data "azurerm_key_vault_secret" "crossplane_providers" {
  for_each = toset(var.crossplane_providers)

  name = "crossplane-${each.value}-provider"
  key_vault_id = data.azurerm_key_vault.main.id
}
```

Listing 5.8: Reading providers' credentials from Azure Key Vault

Both secrets with credentials and provider configuration are created in Kubernetes through templates. The templates folder has files named with conventional names (*e.g.* "<provider>-creds.json" and "<provider>.yaml"). This guarantees that provider names are passed as variables to the Terraform execution and will be used to fetch credentials in the key vault and create the necessary configuration for all mentioned providers. This behaviour is represented in code listing 5.9. The secrets are created first and then the providers' configurations are generated using these secrets, implicitly declaring a dependency between these resources.

```
local {
  crossplane_providers = {
    for provider, cred in kubernetes_secret.crossplane_providers :
      provider => templatefile("${path.module}/templates/crossplane/${provider
    }.yaml.tftpl", { secret_name: cred.metadata.0.name })
  }
  crossplane_providers_creds = {
    for provider, secret in data.azurerm_key_vault_secret.crossplane_providers
     :
      provider => templatefile("${path.module}/templates/crossplane/${provider
    }-creds.json.tftpl", { config = jsondecode(secret.value) })
  }
}

resource "kubernetes_secret" "crossplane_providers" {
  for_each = local.crossplane_providers_creds

  metadata {
    name = "${each.key}-proviver-creds"
    namespace = helm_release.crossplane.namespace
  }

  data = {
    "creds" = each.value
  }
}

resource "kubernetes_manifest" "crossplane_providers" {
```

```
  for_each = local.crossplane_providers
  manifest = yamldecode(each.value)
}
```

Listing 5.9: Provider configuration and credentials provisionement

The templates vary between providers, and for the current solution, only Azure configurations were developed. An example of this is shown in code listing 5.10.

```
apiVersion: azure.upbound.io/v1beta1
metadata:
  name: default
kind: ProviderConfig
spec:
  credentials:
    source: Secret
    secretRef:
      namespace: crossplane-system
      name: ${secret_name}
      key: creds
```

Listing 5.10: Azure provider configuration example

To demonstrate the extensibility of the combined technologies, a new component definition was created in the hub cluster, with the help of KubeVela, the CUE language and Crossplane. The objective is to create a new type of component among existing ones that could create a resource group, with a Structured Query Language (SQL) server and respective databases.

For creating this new definition and render it to a YAML file, the commands in code listing 5.11 are executed:

```
vela def init azure-sql-server \
    -t component \
    -o ./definitions/component/azure-sql-server.cue

vela def render ./definitions/component/azure-sql-server.cue \
    -o ./definitions/component/azure-sql-server.yaml
```

Listing 5.11: Create new component definition

The definition is then edited to match the desired outputs. This is done through templates and input parameters. The code listing 5.12 refers to the created definition.

```
import (
    "encoding/base64"
)

"azure-mssql-server": {
  attributes: workload: definition: {
    apiVersion: "sql.azure.upbound.io/v1beta1"
    kind:       "MSSQLServer"
  }
  type: "component"
}

template: {
  parameter: {
    name: string
```

```
    adminUser: string
    adminPassword: string
    location: string
    resourceGroupName: string
    resourceGroupLocation: string
    version: *"12.0" | string
    databases: *[] | [...string]
  }

  output: {
    apiVersion: "sql.azure.upbound.io/v1beta1"
    kind:       "MSSQLServer"
    metadata: name: parameter.name
    spec: {
      forProvider: {
        administratorLogin: parameter.adminUser
        administratorLoginPasswordSecretRef: {
          key: "admin-login-password"
          name: "\(parameter.name)-admin-secret"
          namespace: context.namespace
        }
        location: parameter.location
        minimumTlsVersion: "1.2"
        resourceGroupName: parameter.resourceGroupName
        version: parameter.version
      }
      writeConnectionSecretToRef: {
        name: "\(context.name)-connection-details"
        namespace: context.namespace
      }
    }
  }
}
```

Listing 5.12: Azure SQL server component definition in CUE language

A component definition must always have a primary output but support the creation of additional outputs. These are omitted, therefore to view the whole definition check Appendix B. Aside from the SQL server, a resource group and databases are created using the custom resources defined by Crossplane. The orchestration of these resources is possible due to the provider configuration that was described before. For now, there is only one default configuration, but it is possible to extend these definitions and support the injection of other configurations by their name.

Besides the component definition, a trait definition was created as well. Traits are management requirements of an artifact associated with a specific component. Trait definitions have access to the application's and component's context, including other trait outputs. Therefore, a trait definition was developed to generate a custom resource representing a new entry in the Domain Name System (DNS) provider in use. In this case, the Kubeflare operator interprets the custom resources and creates the records in Cloudflare accordingly. The code listing 5.13 describes this feature.

```
import "strings"

"cloudflare-dns": {
    type: "trait"
}

template: {
    host: context.outputs.ingress.spec.rules[0].host
```

```
    parts: strings.Split(host, ".")
    domainZone: parts[len(parts) - 2] + "." + parts[len(parts) - 1]

    parameter: {
        publicIpAddress: string
    }
    outputs: {
        dnsRecord: {
            apiVersion: "crds.kubeflare.io/v1alpha1"
            kind: "DNSRecord"
            metadata: {
                name: context.name
                labels: {
                    "app.oam.dev/cluster": "local"
                }
            }
            spec: {
                zone: domainZone
                record: {
                    type: "A"
                    name: host
                    content: parameter.publicIpAddress
                    proxied: true
                    ttl: 3600
                }
            }
        }
    }
}
```

Listing 5.13: Trait definition for creating a record in Cloudflare

The resource is created based on the output of another trait (the gateway trait) and is responsible for creating an ingress resource in the cluster. By accessing that, the name for the A record is computed instead of receiving it from a parameter. Regarding the public IP address, it has to be passed as a parameter to complete the creation of the new record targeting the correct infrastructure entry point.

5.3 Metrics and Logs Collection

Each cluster has its responsibility, depending on its role (hub or spoke), regarding the collection and storage of applications' metrics and logs. The hub clusters are responsible for storing the received data since they do not communicate with spoke clusters. The hub-and-spoke architecture combined with the pull-based strategy favours the adopted strategy for metric and log collection. The spoke clusters collect and send metrics and logs to the centralized system, in this case, to the hub cluster.

5.3.1 The Hub

A helm release is used to deploy Mimir which will be queried by Grafana and receive logs from other instances like Prometheus. The creation of the helm release is simplistic, similar to others shown previously. Since the Mimir write endpoint will be publicly exposed, it has at least to have basic authentication configured to avoid unauthorized push of metrics by unknown entities. The password varies according to the cluster group name (also known as the hub cluster name), and it is generated

within the Terraform code and stored in Azure Key Vault (code listing 5.14). Loki, the entity responsible for log collection and aggregation, follows the same approach.

```
resource "random_password" "mimir_password" {
  length = 32
  override_special = "!@#%&*()-_=+[]{}<>:?"
  keepers = {
    "cluster_group_name" = var.cluster_group_name
  }
}

resource "azurerm_key_vault_secret" "mimir_password" {
  name = "mimir-password"
  value = random_password.mimir_password.result

  key_vault_id = data.azurerm_key_vault.main.id
}
```

Listing 5.14: Generation and storing of Mimir credentials

Mimir and Loki must have their helm releases configured with the correct values to enable basic authentication. This is done using Terraform templates. The code listing 5.15 presents a part of the Mimir input configuration receiving the authentication details (username and password) and the configured hostname accessible by all spoke clusters.

```
nginx:
  replicas: 1
  resources:
    limits:
      memory: 731Mi
  basicAuth:
    enabled: true
    username: ${basic_auth.username}
    password: ${basic_auth.password}
  ingress:
    enabled: true
    ingressClassName: nginx
    hosts:
      - host: ${hostname}
        paths:
          - path: /
            pathType: ImplementationSpecific
    tls: []
```

Listing 5.15: Mimir basic authentication configuration

As stated, Mimir is a long-term storage solution for metrics. It aggregates the metrics in an external storage to have them available whenever needed. The storage configuration has to be passed to the helm release (code listing 5.16 and 5.17). The configuration depends on which cloud the hub cluster was created. For example, when creating an AKS cluster, the developed Terraform code will create a storage account generating an adequate configuration for Mimir.

```
output "storage_configuration" {
  value = {
    backend = "azure"
    parameters = {
      account_key = azurerm_storage_account.main.primary_access_key
```

```
      account_name = azurerm_storage_account.main.name
      endpoint_suffix = "blob.core.windows.net"
    }
  }
}
```

Listing 5.16: Mimir long-term storage configuration output

```
mimir:
  structuredConfig:
    common:
      storage:
        backend: ${storage.backend}
        ${storage.backend}:
          ${indent(10, yamlencode(storage.parameters))}
    blocks_storage:
      backend: ${storage.backend}
      ${storage.backend}:
        container_name: mimir-blocks
    alertmanager_storage:
      ${storage.backend}:
        container_name: mimir-alertmanager
    ruler_storage:
      ${storage.backend}:
        container_name: mimir-ruler
    limits:
      max_global_series_per_user: 1000000
```

Listing 5.17: Mimir long-term storage configuration

Grafana is the component that will allow users to view metrics and logs, apply filters to them and create or import existing dashboards to track them over time. For this, Grafana must reach the data sources where metrics and logs are collected. These data sources are handled by the Terraform code as well through templates. The output of these templates will be used to create ConfigMaps - a Kubernetes object that holds configurations typically without any sensitive data - that are injected into Grafana with a specific label indicating that must be processed by itself (see code listing 5.18).

```
local {
  mimir_datasource = templatefile("${path.module}/templates/mimir/datasource.
    yaml.tftpl", {
    query_frontend_endpoint = "${local.mimir_internal_endpoint}/prometheus"
    basic_auth = local.mimir_basic_auth
  })
  loki_datasource = templatefile("${path.module}/templates/loki/datasource.
    yaml.tftpl", {
    endpoint = "${local.loki_internal_endpoint}"
    basic_auth = local.loki_basic_auth
  })
}

resource "kubernetes_secret" "grafana_datasources" {
  metadata {
    name = "grafana-datasources"
    namespace = helm_release.kube_prometheus_stack.namespace
    labels = {
      grafana_datasource = "1"
    }
  }
```

```
20
21   data = {
22     "mimir-ds.yaml" = local.mimir_datasource
23     "loki-ds.yaml" = local.loki_datasource
24   }
25 }
```

Listing 5.18: Grafana datasources generation (for Mimir and Loki)

5.3.2 The Spokes

Helm releases are used to deploy Prometheus and Fluent Bit, each responsible for scraping metrics and collecting logs, respectively. Input values for each release are one more time done using templates which receive Mimir and Loki settings, such as the public endpoints and credentials for authentication.

Prometheus has a feature known as remote write, which sends metrics to another instance. Therefore, it is configured to send these metrics to Mimir. The configuration for the monitoring stack is described on code listing 5.19.

```
1  prometheus:
2    prometheusSpec:
3      externalLabels:
4        cluster: ${cluster_name}
5      prometheusExternalLabelName: cluster
6      replicaExternalLabelName: __replica__
7      remoteWrite:
8        - url: ${mimir_distributor_endpoint}
9          headers:
10           X-Scope-OrgID: "1"
11         basicAuth:
12           username:
13             key: username
14             name: ${mimir_auth_creds_secret_name}
15           password:
16             key: password
17             name: ${mimir_auth_creds_secret_name}
18
19 grafana:
20   enabled: false
```

Listing 5.19: Prometheus configuration

The Fluent Bit configuration is more extended since it works with inputs, parsers and outputs. By default, it will collect logs from all containers within the cluster it lives in. Depending on the size of the cluster, a massive quantity of logs will be sent to Loki, which means more infrastructure costs. Sometimes, logs from some development environments or specific non-critical instances are not required to be processed, meaning they can be ignored.

The first thing to do is to configure Fluent Bit with zero inputs and configure the outputs to point to Loki public endpoint as shown in code listing 5.20.

```
1  config:
2    inputs: ""
3    outputs: |
```

```
    [OUTPUT]
        name                    loki
        match                   *
        host                    ${loki_host}
        port                    443
        http_user               ${loki_basic_auth.username}
        http_passwd             ${loki_basic_auth.password}
        tls                     on
        tls.verify              on
        tenant_id               ${tenant_id}
        auto_kubernetes_labels  true
        labels                  job=fluentbit, cluster=${cluster_name}
```

Listing 5.20: Fluent Bit configured inputs and outputs

Although Fluent Bit has a hot reload feature that allows users to inject configurations on the fly, it does not support mounting created ConfigMaps with specific labels like Grafana. For this, an additional container was added to replicate this feature (see code listing 5.21). It watches for ConfigMaps with the label "fluent_bit_input" and the respective value "1". Then, it puts them in the "/tmp/config" folder.

```
extraContainers:
  - name: fluentbit-sc-configs
    image: kiwigrid/k8s-sidecar:1.25.0
    command: ["/bin/sh", "-c", "touch /tmp/config/dummy.conf && python -
    u /app/sidecar.py"]
    env:
      - name: METHOD
        value: WATCH
      - name: NAMESPACE
        value: ALL
      - name: LABEL
        value: fluent_bit_input
      - name: LABEL_VALUE
        value: "1"
      - name: FOLDER
        value: /tmp/config
      - name: RESOURCE
        value: configmap
      - name: REQ_METHOD
        value: POST
      - name: REQ_URL
        value: http://localhost:2020/api/v2/reload
    volumeMounts:
      - name: conf
        mountPath: /tmp/config
```

Listing 5.21: Fluent Bit configuration reloader container

The temporary folder is shared between the Fluent Bit container and the extra container. Then, in the service configuration, an include statement is present to load additional configurations (code listing 5.22, line 14) by the container that performs the creation of ConfigMaps and reload of Fluent Bit.

```
extraVolumeMounts:
  - name: conf
    mountPath: /fluent-bit/etc/conf/extra
```

```
4
5  extraVolumes:
6    - name: conf
7      emptyDir: {}
8
9  hotReload:
10   enabled: true
11
12 config:
13   service: |
14     @INCLUDE /fluent-bit/etc/conf/extra/*.conf
```

Listing 5.22: Reload mounted inputs configuration files on the fly

5.3.3 Enabling Log Collection

After the infrastructure is ready for pushing and storing logs, new Fluent Bit inputs need to be created for applications residing in a specific namespace to have their logs collected. This is done using the extensible capabilities of KubeVela and the CUE language. This time, a new policy is defined in code listing 5.23.

```
1  "container-log": {
2    type: "policy"
3  }
4
5  template: {
6    parameter: {
7      namespace: string
8      cluster: string
9    }
10   output: {
11     apiVersion: "v1"
12     kind:       "ConfigMap"
13     metadata: {
14       name: "enable-logging-config"
15       namespace: parameter.namespace
16       labels: {
17         "fluent_bit_input": "1"
18         "app.oam.dev/namespace": parameter.namespace
19         "app.oam.dev/cluster": parameter.cluster
20       }
21
22     }
23     data: "\(parameter.namespace).conf":
24         """
25         [INPUT]
26           name tail
27           path /var/log/containers/*_\(parameter.namespace)_*.log
28           multiline.parser docker, cri
29           tag kube.*
30           mem_Buf_Limit 5MB
31           skip_Long_Lines On
32         """
33   }
34 }
```

Listing 5.23: Policy definition for collecting logs of applications in a specific cluster and namespace

The inputs are generated by passing the cluster and namespace parameters. Policy definitions in KubeVela generally do not have access to other policies. Therefore,

finding the target namespace and cluster overridden by other policies is difficult. The most simplistic approach is to request the user these parameters, which makes it more descriptive since it is possible to check what namespaces are being watched, by looking at the policy logic.

Also, it is important to include the label in line 17 because the extra container described previously is watching for new configurations with that specific label and value.

5.4 Artifact State Reconcilement

As stated in functional requirements, when a declaration of an application is created in a repository, it must be created in the correct hub cluster. Then, the hub cluster will handle the deployment of the application as it should. The heeded strategy is a GitOps-based one, the pull-based approach. Following this strategy means an operator lives within the cluster and will pull configurations from repositories. Flux is used to implement this workflow and was deployed as a KubeVela addon mentioned in section 5.1.

Kustomize was used following the separation of concerns of the custom resource of KubeVela, the "Application" resource. As mentioned, the resource states the components, policies and workflow using defined policies. Operation teams are responsible for managing policy and workflow definitions, while development teams manage the components to be deployed.

The developed Terraform project sets up the repository and Kustomization resources that are applied. Although the project only supports declaring one default repository, it can handle extra Kustomizations. This is shown in the code listing 5.24 where default Kustomizations are declared and concatenated with additional ones that can be passed as parameters to the hub module.

```
1  local {
2    merged_kustomizations = concat(var.extra_kustomizations, [
3      { name = "vela-definitions", repo = "canaveral", path = "./definitions" },
4      { name = "apps", repo = "canaveral", path = "./gitops/ops/clusters/${var.
       cluster_group_name}" }
5    ])
6    kustomizations = { for k in local.merged_kustomizations : k.name => k }
7  }
8
9  resource "kubernetes_manifest" "kustomizations" {
10   for_each = local.kustomizations
11
12   manifest = {
13     apiVersion = "kustomize.toolkit.fluxcd.io/v1beta2"
14     kind = "Kustomization"
15     metadata = {
16       name = each.value.name
17       namespace = "flux-system"
18     }
19     spec = {
20       interval = "10m"
21       targetNamespace = "vela-system"
22       sourceRef = {
23         kind = "GitRepository"
24         name = each.value.repo
25       }
```

```
26        path = each.value.path
27        prune = "true"
28        timeout = "1m"
29      }
30    }
31
32    depends_on = [ kubernetes_manifest.canaveral_github_repository ]
33  }
```

Listing 5.24: Generation of Kustomization resources

These are responsible for merging the policies and workflow with the rest of the manifest. For a better understanding of how folders are structured, Figure 5.2 is presented.

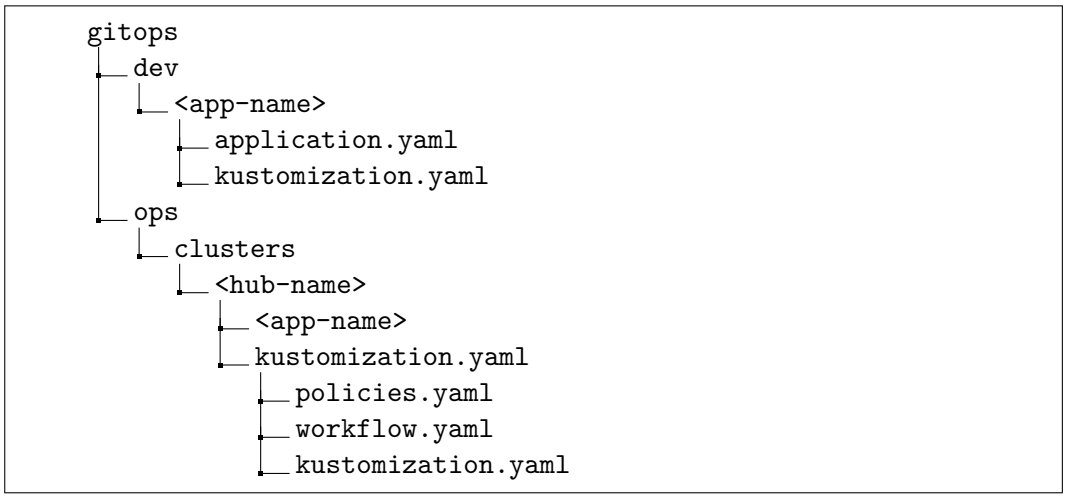


Figure 5.2: GitOps folder structure

For each application, a Kustomization file must exist for the operator to know what resources to apply. More resources can exist within each application folder. These are considered the base definition.

The "ops" folder contains a folder for each existing hub cluster. This folder structure separates resources between infrastructures, in this case, different hub clusters. A "kustomization.yaml" is present in each app (code listing 5.25) and will take their base definitions in the "dev" folder and merge the defined policies and workflow accordingly.

```
1  apiVersion: kustomize.config.k8s.io/v1beta1
2  kind: Kustomization
3  resources:
4    # can be a reference to another repository file (e.g. URL)
5    - ../../../../dev/<app-name>/
6  patchesStrategicMerge:
7    - policies.yaml
8    - workflow.yaml
```

Listing 5.25: Example of Kustomization for the Operations team

As mentioned, Terraform handles the creation of the default Kustomizations to create them already pointing to the correct folder in the repository. For private

repositories, a secret must be referenced when creating the repository custom resource. The code covers this edge case, but a secret must exist within the Azure Key Vault, which will be used across the entire project.

5.5 Artifact Backup and Restore

For backing up and restoring resources in the hub cluster, the Velero operator is used. Similarly to other components, Velero is deployed using a helm release. The module responsible for creating the hub cluster in the cloud also creates a storage solution used by Mimir. This storage is shared with Velero but in a different container present in an Azure Storage Account.

Velero also supports volume snapshotting for saving data in Kubernetes persistent volumes. For now, this feature is not used nor is contained in the scope of this project. Resources manifests are simply JavaScript Object Notation (JSON) objects which occupy minimal space, and Velero zips them to reduce the impact on costs. However, volume snapshots must be carefully handled since they can exponentially increase costs depending on which saved data.

The mentioned module creates an output value specifying the storage configuration for Velero (5.26). This is the same strategy used in Mimir, which enables dynamic configuration when using a different module to create the hub cluster. Velero uses various plugins to communicate with heterogeneous storage solutions for storing and reading its backups. Therefore, the plugin must be configured for each provider. The "backup_storage" field is a dynamic object that supports different values requested by other providers.

```
output "velero_storage_configuration" {
  value = {
    plugin = {
      repository = "velero-plugin-for-microsoft-azure"
      tag        = "v1.7.1"
    }
    provider = "azure"
    backup_storage = {
      resourceGroup               = azurerm_storage_account.main.
    resource_group_name
      storageAccount              = azurerm_storage_account.main.name
      storageAccountKeyEnvVar     = "AZURE_STORAGE_ACCOUNT_ACCESS_KEY"
      subscriptionId              = var.subscription_id
    }
    credentials = <<EOT
AZURE_STORAGE_ACCOUNT_ACCESS_KEY=${azurerm_storage_account.main.
    primary_access_key}
AZURE_CLOUD_NAME=AzurePublicCloud
AZURE_ENVIRONMENT=AzurePublicCloud
EOT
  }
}
```

Listing 5.26: Output of storage configuration for Velero (Azure specific)

To start backing up the state of the cluster, creating one or more schedule policies is needed to trigger the operator to create several backups recurrently. A default schedule is created if none are specified when provisioning the infrastructure (see

code listing 5.27). This is done to enforce backup policies for relevant resources. It backs all types of resources of all namespaces daily with a Time to live (TTL) of 24 hours, also known as the retention period.

```
variable "velero_backup_schedules" {
  type = list(object({
    name = string
    included_namespaces = list(string)
    included_resources = list(string)
    cron_expression = string
    time_to_live = string
  }))
  default = [{
    name = "default",
    included_namespaces = ["*"],
    included_resources = ["*"],
    cron_expression = "0 1 * * *",
    time_to_live = "24h0m0s",
  }]
}
```

Listing 5.27: Velero backup schedules variable definition and default value

The creation of these schedules comes from templates that generate custom resources defined by Velero itself. The input variable described is looped to create all the schedules. Code listing 5.28 represents a template of a schedule resource. Velero will read these resources and attend to these policies to create the desired backups targeting namespaces and resources defined in the policy.

```
%{ for schedule in schedules }
---
apiVersion: velero.io/v1
kind: Schedule
metadata:
  name: ${schedule.name}
  namespace: ${namespace}
spec:
  template:
    includedNamespaces:
      %{ for ns in schedule.included_namespaces ~}
      - "${ns}"
      %{ endfor }
    includedResources:
      %{ for res in schedule.included_resources ~}
      - "${res}"
      %{ endfor }
    snapshotVolumes: false
    ttl: ${schedule.time_to_live}
  schedule: ${schedule.cron_expression}
%{ endfor ~}
```

Listing 5.28: Velero backup schedules template

5.6 Summary

This chapter presented an overview of the different topics the implementation process approaches. It includes how: (i) clusters are managed; (ii) applications are

orchestrated; (iii) metrics and logs of deployed applications are collected and stored; (iv) artifacts state are continuously reconciled; (v) artifacts are backed up and restored.

The cluster management section starts by explaining the folder structure that defines the infrastructure in Terraform code. Since a hub-and-spoke strategy is used, each cluster has its role and later in the section each role's responsibilities are described. It also describes how the hub cluster is provisioned and new spoke clusters join it.

In the application orchestration section, the combination of KubeVela and Crossplane functionalities provides a way to create new component definitions that abstract the deployment of complex resources. A snippet of a component definition that outputs an Azure SQL server is presented.

Afterwards, the metrics and logs collection section presents the responsibilities of the hub and the spokes, considering the task to be performed. It describes how metrics and logs are collected and remotely written by spokes and then received, stored and queried by hub clusters.

Then, the artifact state reconciliation section describes how the state of the infrastructure and running applications within it are up to date according to a source of truth. This is done by using Kustomize and Flux operator capabilities, separating concerns such as components and infrastructure details.

Finally, the artifact backup and restore section tackles the security side. In case of a disaster aside from the sources of truth (normally Git repositories), a second backup of the resources is maintained in an off-site location, available for restoration at any time.

Chapter 6

Experiments and Evaluation

This chapter moves towards the demonstration and evaluation phases of the DSR methodology, which focuses on the experimentation and evaluation of the solution. It starts by detailing the mentioned hypotheses in Chapter 1. These help in gathering relevant indicators and related metrics, which are also connected to the functional and non-functional requirements. The methodology describes how each metric and related hypotheses are tested, referencing methods and auxiliary tools. Then, the results are described, attending to the methodology, indicators and hypotheses mentioned.

6.1 Hypothesis

Hypotheses were previously defined in section 1.4 according to the goals and questions described using the GQM paradigm. They are recalled below, with a more detailed explanation.

H1. The process of deployment of an application, independently of the target platform, is faster and more reliable compared to the old one

This hypothesis presumes deploying the same application on different platforms requires no effort if the scenario is already supported. If it is not available, the process of deployment supports the creation of custom scenarios that can be reused whenever they are needed.

The newly developed process is more robust considering the managing of applications and underlying infrastructure, thus deployments are done faster and with no downtime, ensuring the orchestration of all instances of that application.

Application, pipelines and artifacts catalogues are accessible to any team who wants to check how the components regarding a certain product are available. Logs and metrics also are available within a time frame (5-10 minutes) after the deployment phase finishes.

H2. The increase in support of different platforms or workflows in the deployment process does not affect the complexity of the deployment process

Adding new scenarios to reuse them in many applications does not make the process to be complicated and tiresome. It presumes that is easy to deploy an application across different platforms if they are supported without additional complexity when creating a deployment artifact.

H3. Controlling certain characteristics of the underlying infrastructure is possible through override techniques

Abstracting some technical settings to have readable and maintainable deployment artifacts does not erase the possibility of overriding the default behaviour. However, it may not allow a fine-grain control of all technical aspects of application and infrastructure characteristics.

6.2 Indicators

The hypotheses must be tested to understand if the actual results match the expected answers. Therefore, indicators are identified to help in analysing and evaluating the developed project. These indicators, which are presented as quality attributes, will refer to a set of metrics that originate from the research questions defined previously with the GQM methodology and the non-functional requirements specified in 3.6. The identified metrics are described below.

Usability

M1. Old process vs. new process

Reliability

M2. Time to restore service (Mean Time To Restore (MTTR)).

M3. Same inputs for application deployment, same outputs.

Performance

M4. Deployment times.

M4.1. Reconciliation time with source control.

M4.2. Deploy time in hybrid environment vs. non-hybrid environment.

Supportability

M5. Complexity and time of the creation of a new deployment scenario.

M6. Degree of control regarding low-level application and infrastructure settings.

6.3 Methodology

Relevant metrics for the evaluation of the new deployment process were identified in the previous section. Many techniques and tools can be used to collect metrics, but these depend on the nature of the metric itself (quantitative or qualitative).

Regarding the referred usability metrics (M1) and the first hypothesis (H1), practical experimentation with team members following the new deployment process must be done. Auxiliary documentation will be written and given to the participating members, and each one will issue experimentation. Feedback and time consumed to replicate a specific scenario will be gathered. It also tests the second hypothesis (H2) since supporting new scenarios should not affect performance and reliability directly.

Reliability is also related to the first hypothesis (H1). Evaluating reliability will ensure that the process is robust enough to be used in production. A low time to restore services (M2) is crucial and can be tested following a simulation. Before the simulation, the creation and setup of a Kubernetes cluster are needed, and the deployment of artifacts must be issued. The simulation is started by deleting an application and recreating it, measuring downtime. This scenario will help analyse if the same input artifacts produce the same deployment outputs (M3).

The next factor is performance, directly related to the first hypothesis (H1). It focuses on a set of quantitative metrics related to deployment times (M4) that can be divided into two more specific metrics. These are reconciliation times with source control when using a pull-based deployment strategy (M4.1) and the deployment time in a hybrid environment compared to a non-hybrid environment (M4.2).

Regarding supportability, complexity and time consumed in the creation of a new deployment scenario is measured (M5). These metrics can be computed by making use of another simulation where there is an attempt to extend the supported scenarios and migrate applications between platforms. These are also related to H1 and H2.

Finally, the degree of control of low-level details (M6) is a metric that will be measured using experimentations and directly related to the third hypothesis (H3). It will consist of modifying such details and registering successful and failed attempts.

Table 6.1 summarizes which methods and auxiliary tools will be used to test hypotheses and gather related metrics:

Table 6.1: Metrics' related methods and auxiliary tools

Metric	**Related Hypotheses**	**Method**	**Auxiliary Tools**
M1	H1, H2	Experimentation with team members	Questionnaires, Timers
M2, M3	H1	Simulation	Uptime Kuma, Grafana, Prometheus
M4, M4.1, M4.2	H1	Experimentation	Grafana, Prometheus, Azure DevOps, GitHub
M5	H1, H2	Simulation	Timers
M7	H3	Experimentation	Observation

6.4 Results

Hypotheses and evaluation methodologies were defined, and in this section, the results will help justify the hypotheses. Regarding usability, a sample application is provided for the team deploying it using the new platform. The rest of the indicators will rely on the deployment of an existing complex application, a .NET Core reference application named "eShopOnContainers" where its definition is shown on Appendix D.

6.4.1 Usability

The methodology used to measure the usability was a guided experimentation with team members. At the end of the experimentation, participants answered a questionnaire presented on Appendix C. The experimentation consisted of five main tasks:

1. Create two components: a web application and a database;
2. Add a public hostname;
3. Separate the deployment into two environments: development and production. Deploy each one in different clusters (multicluster deployment).
4. In the production environment, create a database in Azure and configure the web app to use it.
5. Enable container logging of the web application.

Population details, such as years of experience and age, are presented on Appendix C. The average time spent in experimentation was 1 hour, and the average time taken to answer the questionnaire was around 7 minutes. A total of 7 responses were gathered.

By analysing the bar charts regarding the number of answers per rating on Appendix C, it is possible to conclude that the learning curve is steep for deploying a simple application. The scenario of migrating the database to Azure has an average rating below 4. This is expected since the scenario is complex and relies upon the development of one or more policies.

Some suggestions were stated, such as formatting the application definition files and simplifying the configuration part for developers who are interested in customizing their deployments' monitoring and logging specifications. Also, it was mentioned that the development of a higher abstraction layer on top of the platform would speed up learning of the new deployment process.

The final result of the file that defines the application that is deployed is present on Appendix D.

6.4.2 Reliability

Reliability is an important factor when assessing if the platform is robust enough to withstand and adapt to scenarios of disaster, such as unhealthy computing instances, deletion of resources by mistake, and other situations.

Table 6.2 presents the collected metrics of the simulation. It consisted of deleting several deployments and checking the time taken for all the components of the application to be restored.

TABLE 6.2: MTTR metrics

Iteration	Non-hybrid environment	Hybrid environment
1	4m 50s	7m 40s
2	5m 20s	6m 23s
3	2m 45s	7m 3s
4	2m 50s	7m 10s
5	3m 26s	7m 3s

Therefore, a job was created to check the status of the application. The application exposes a health endpoint that facilitated gathering which components were unhealthy. The job was responsible for sending a request to a webhook when the application was fully restored.

The restoration of the non-hybrid environment is fast since it only relies on native resources supported by the Kubernetes local cluster (also referred to as the hub cluster). The measured times may have some slight deviations from real timestamps due to KubeVela reconciliation timeouts and the detection delay originated by Uptime Kuma.

Regarding the hybrid environment, the resources that described the SQL server and databases to be created, were eliminated several times to gather the restoration time of the application. Since these resources are in the cloud, it is only natural that it takes more time for the SQL server to be available compared to the database deployed as a container.

Finally, multiple deployments of the same application were performed to evaluate platform consistency regarding the generated artifacts. Grafana is used in this case to check created resources. Figure 6.1 presents a dashboard that interprets metrics from KubeVela.

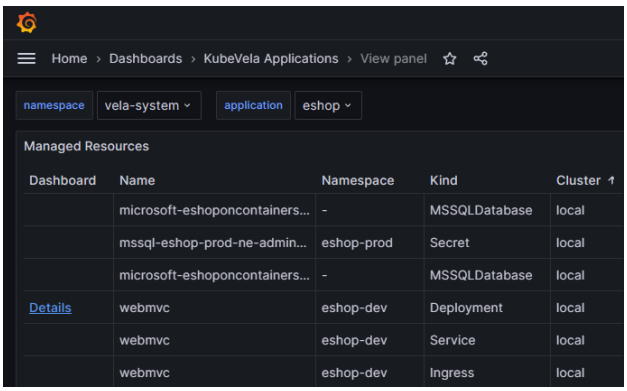


FIGURE 6.1: Grafana dashboard of KubeVela applications for checking generated resources

By checking the managed resources, the consistency can be evaluated by analysing if the same number of resources and associated specifications were present in the dashboard in different iterations.

6.4.3 Performance

As the latter indicator, the performance of the platform is also one of the most important indicators of the quality of the solution. Therefore, the collected metrics are:

1. Reconciliation time with source control (maximum value of 1 minute due to the configured reconciliation interval);
2. Deploy time of the application in a non-hybrid environment (development environment only container-based);
3. Deploy time of the application in a hybrid environment (production environment with cloud resources).

These metrics were collected using the Grafana dashboards as well. Table 6.3 describes them. The test consisted of 5 iterations of the defined workflow.

Table 6.3: Performance metrics

Iteration	**Reconciliation Time**	**Non-hybrid environment deploy time**	**Hybrid environment deploy time**	**Total time**
1	34s	17s	3m 39s	4m 30s
2	43s	17s	4m 31s	5m 31s
3	26s	16s	7m 5s	7m 47s
4	48s	15s	3m 17s	4m 20s
5	57s	13s	3m 17s	4m 27s

As mentioned, the reconciliation time is a configurable interval. Since the defined interval is 1 minute, the reconciliation time will be within that timeframe.

The deployment time of a non-hybrid environment is fast, considering that it has to deploy 19 components and wait for them to be all healthy. Since it is container-based and the deployment is done in the hub cluster, the results show only slight variations, as expected.

The deployment time of a hybrid environment consisted of provisioning an Azure SQL server instead of using a SQL server as a container. Since communication with the cloud is mandatory and Crossplane takes care of it, the time it takes to deploy varies. This can be due to throttling requests since the deployment tests were consecutively performed. The third iteration was when the deployment took more time.

For the non-hybrid deployments, the mean time spent was about 15 seconds and for hybrid deployments was approximately 4 minutes and 22 seconds. Because the non-hybrid and hybrid environments were considered as development and production

stages respectively, the deployment workflow contains both environments. The whole workflow took an average of 5 minutes and 19 seconds.

6.4.4 Supportability

Finally, the supportability of the platform is not quantifiable. The metrics associated with this indicator rely on observations of performed experimentations.

As mentioned, the platform must be extensible and support the development of new deployment scenarios. To measure the time and complexity of the process, the creation of a new component definition in KubeVela was timed.

The objective of the new component definition was to create an Azure Web App using a Docker image. This definition is present on Appendix D. The time spent developing this new definition was 24 minutes and 57 seconds. Considering the lack of documentation for advanced scenarios and the complexity of this scenario - other resources had to be created due to Azure Web App dependencies - the time spent and complexity match the expectations.

Regarding the degree of control of infrastructure settings, it is possible to conclude from the previous experimentations and simulations that controlling some settings native to Kubernetes is straightforward. An example is shown in code listing 6.1, which increases the number of replicas to two.

```
apiVersion: core.oam.dev/v1beta1
kind: Application
metadata:
  name: website
spec:
  components:
    - name: frontend
      type: webservice
      properties:
        image: nginx
      traits:
        - type: scaler
          properties:
            replicas: 2
```

Listing 6.1: Example of scaler trait on application

When extending the platform to support new scenarios or functionalities, if they provide a way to manage infrastructure details, then the templates must reflect that setting in the parameters to override defaults. For example, scaling an application attending to its logs would require KEDA. Then, a resource of the kind "ScaledObject" should be created specifying the target and the triggers (potentially a Loki trigger type).

6.5 Summary

This chapter described the hypotheses with the related indicators and methodology for capturing and validating metrics. Different techniques were used, such as simulations, experimentations with team members at the host organization and metric gathering through monitoring systems.

All of the hypotheses were verified throughout the experimentation and evaluation phase. Therefore, this proves that the developed solution gets closer to achieving the defined goal for this project.

The results end up showing satisfying values that will reflect on what should be revised. The feedback from the team members is crucial for better adoption of the POC, which leads to investing more time in developing a full-featured and robust solution.

Chapter 7

Conclusion

This thesis addresses the problem of orchestrating multiple applications in hybrid environments and describes a possible solution to mitigate it. The problem, objectives and hypothesis are described in Chapter 1, starting the first phase of the DSR methodology.

In Chapter 2, knowledge about technologies and strategies was gathered to build a solution able to respond to the problem. Also, some related work is mentioned. A value analysis was performed in Chapter 3 as well as a requirements analysis, where functional and non-functional requirements are gathered regarding the third phase of the DSR methodology.

Still, in the third phase of the DSR methodology, the architecture of the solution is presented in Chapter 4. Chapter 5 describes the implementation of the solution which is considered a POC. The solution followed a COTS approach. Combining KubeVela with other technologies, it is possible to build a platform that can orchestrate resources in the cloud, monitor and log applications, replicate the desired state of the deployments and backup/restore important artifacts anytime.

Then, in Chapter 6, an evaluation of the solution is performed considering different indicators related to hypotheses. Metrics and feedback were gathered by performing experimentations, simulations and questionnaires.

7.1 Achievements

After defining the problem, research questions are derivatives from it that break the problem into specific concerns regarding the topic of this thesis. The answers to each research question are described below.

RQ1. How can an application deployment process be simplified when it targets different platforms?
It is possible to simplify deployments across hybrid environments mainly using abstraction, templating and concrete definitions of an application. KubeVela follows the OAM principles, which help define a deployable application in any cloud native platform. It uses the CUE language to create new types of definitions that can extend the scenarios supported by KubeVela. Also, concentrating all of the orchestration components in a centralized cluster responsible for

dispatching workloads to other clusters simplifies the management of applications and their environments regarding variations and specific customizations.

RQ2. What is the influence of extensibility upon simplification of the deployment process targeting different platforms?
In the case of the current POC and after gathering different metrics through observation and experimentation, extensibility has a low impact on the simplification of the deployment process. The most complex phase when extending the platform is when new definitions are being created. The development can be simple or complex depending on the scenario to be satisfied. The provided abstraction by these definitions is the reason why the deployment process is straightforward.

RQ3. Does the degree of control regarding infrastructure details decrease when the creation of new deployments across different platforms is facilitated?
Abstraction and templating provide a way of creating new definitions to customize the applications' deployments. Simplifying the creation of new deployments across different platforms will impact the degree of control of underlying infrastructure details that support the application being deployed. This is due to the nature of templating, unless all of the parameters are specified and used in the output artifacts some specifications will be missed. Whenever a new feature is required regarding a specific necessity, the definition in question must be updated. However, extending a template and adding too many parameters will have the opposite effect.

The main goal of this work was to develop a more improved and optimised process (compared to the existing one) for deploying and orchestrating applications in hybrid environments. Although all the hypotheses were verified, the answers to the research questions help to understand that some flaws exist within the developed solution.

In the end, it was possible to rapidly orchestrate an application with different components across different platforms, having metrics and logs almost available instantly due to the extensibility and robustness of the solution.

7.2 Contributions

All of the work presented in this thesis will be compiled in an article that will explain in a more succinct way how the deployment and management of applications can be optimised and orchestrated using KubeVela.

7.3 Limitations and Future Work

Although the goal was achieved and the solution worked as intended, limitations were identified during the development and experimentation of the solution.

The first one is that the setup of clusters using the Terraform code will not ensure that the host running it has access to the target cluster. This is due to private

clusters behind VPNs, and the connection to them has to be performed manually (the user has to establish a connection).

Regarding the technology itself, KubeVela is considered an incubating project CNCF, which means it is a stable project and is used in production environments by a few users. Despite this, the documentation lacks more advanced use cases, and some problems were detected throughout the evaluation of the solution. Some of them are unexpected resources created in the wrong clusters, complexity in retrieving cloud-specific secrets from resources (*e.g.* databases) and the onboarding of new clusters that fail if certain flags are passed to the KubeVela operator.

Also, the growth of the applications in several components makes the files that describe them longer and hard to read. For complex applications that rely on many variations and a high number of components, this is potentially inconvenient.

One task identified as future work is to simplify even more the setup of infrastructure and the definition and management of applications. That would require the development of a tool that would use the Terraform code as template files and will ensure that a cluster group - one hub cluster and associated spokes - is deployed. Additionally, the creation of a new schema helps to generate complex application artifacts across cluster groups according to the type of the project, technologies used and additional metadata of the same project.

7.4 Final Remarks

All the research and the solutions that resulted from the presented work contribute to a better understanding of the complexity and limitations of orchestrating multiple applications across heterogeneous platforms.

As stated, the solution follows a COTS approach, which means the composition of the solution relies upon existing technologies. KubeVela is the technology that allows the distribution of applications among different platforms. Although it matched the organization's necessities, after a thorough evaluation, limitations were identified.

With the development of the solution, it is possible to have a starting point where applications can be deployed with almost immediately available monitoring and logging and always being reassured that backups are made daily in case of disasters, accidents and security breaches. Also, the control of clusters that host the applications without being publicly available helps reduce the attack surface.

The presented work will also build the motivation to continue studying this topic within the organization and improve the current solution with the technologies used. That does not mean that other technology stacks cannot be used, and since the solution is a POC, it is always subject to architectural, design and technology changes. It is possible to conclude that all the functional and non-functional requirements were accomplished for the POC. The POC shows that it has a margin for improvement, and other approaches can be taken for the development of a better solution.

Bibliography

[1] M. Shahin, M. Ali Babar, and L. Zhu, "Continuous integration, delivery and deployment: A systematic review on approaches, tools, challenges and practices," *IEEE Access*, vol. 5, pp. 3909–3943, 2017, Conference Name: IEEE Access, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2017.2685629.

[2] J. Surbiryala and C. Rong, "Cloud computing: History and overview," in *2019 IEEE Cloud Summit*, Aug. 2019, pp. 1–7. DOI: 10.1109/CloudSummit47114.2019.00007.

[3] V. Sharma, "Managing multi-cloud deployments on kubernetes with istio, prometheus and grafana," in *2022 8th International Conference on Advanced Computing and Communication Systems (ICACCS)*, ISSN: 2575-7288, vol. 1, Mar. 2022, pp. 525–529. DOI: 10.1109/ICACCS54159.2022.9785124.

[4] R. v. Solingen, E. Berghout, and E. W. Berghout, *The goal/question/metric method: a practical guide for quality improvement of software development*. London: The McGraw-Hill Companies, 1999, 199 pp., ISBN: 978-0-07-709553-6.

[5] K. Peffers, T. Tuunanen, M. A. Rothenberger, and S. Chatterjee, "A design science research methodology for information systems research," *Journal of Management Information Systems*, vol. 24, no. 3, pp. 45–77, Dec. 2007, ISSN: 0742-1222, 1557-928X. DOI: 10.2753/MIS0742-1222240302. [Online]. Available: https://www.tandfonline.com/doi/full/10.2753/MIS0742-1222240302 (visited on 02/04/2023).

[6] M. Moravcik and M. Kontsek, "Overview of docker container orchestration tools," in *2020 18th International Conference on Emerging eLearning Technologies and Applications (ICETA)*, Nov. 2020, pp. 475–480. DOI: 10.1109/ICETA51985.2020.9379236.

[7] A. Malviya and R. K. Dwivedi, "A comparative analysis of container orchestration tools in cloud computing," in *2022 9th International Conference on Computing for Sustainable Global Development (INDIACom)*, Mar. 2022, pp. 698–703. DOI: 10.23919/INDIACom54597.2022.9763171.

[8] P. Mell and T. Grance, "The NIST definition of cloud computing,"

[9] I. Kumara, M. Garriga, A. U. Romeu, *et al.*, "The do's and don'ts of infrastructure code: A systematic gray literature review," *Information and Software Technology*, vol. 137, p. 106 593, Sep. 2021, ISSN: 09505849. DOI: 10.1016/j.infsof.2021.106593. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0950584921000720 (visited on 02/14/2023).

[10] Y. Wang, C. Lee, S. Ren, E. Kim, and S. Chung, "Enabling role-based orchestration for cloud applications," *Applied Sciences (2076-3417)*, vol. 11, no. 14, p. 6656, Jul. 15, 2021, ISSN: 20763417. DOI: 10.3390/app11146656. [Online]. Available: https://search.ebscohost.com/login.aspx?direct=true&

AuthType=ip,shib&db=edb&AN=151562167&lang=pt-pt&site=eds-live&scope=site (visited on 02/11/2023).

[11] G. Katsaros, M. Menzel, A. Lenk, J. Rake-Revelant, R. Skipp, and J. Eberhardt, "Cloud application portability with TOSCA, chef and openstack," in *2014 IEEE International Conference on Cloud Engineering*, Boston, MA: IEEE, Mar. 2014, pp. 295–302, ISBN: 978-1-4799-3766-0. DOI: 10.1109/IC2E.2014.27. [Online]. Available: https://ieeexplore.ieee.org/document/6903486/ (visited on 10/12/2023).

[12] O. Tomarchio, D. Calcaterra, G. Di Modica, and P. Mazzaglia, "TORCH: A TOSCA-based orchestrator of multi-cloud containerised applications," *Journal of Grid Computing*, vol. 19, no. 1, p. 5, Mar. 2021, ISSN: 1570-7873, 1572-9184. DOI: 10.1007/s10723-021-09549-z. [Online]. Available: http://link.springer.com/10.1007/s10723-021-09549-z (visited on 10/12/2023).

[13] "Deploying your OAM applications in the napptive cloud-native application platform | KubeVela." (Nov. 24, 2022), [Online]. Available: https://kubevela.io/blog/2022/11/24/napptive-kubevela/ (visited on 10/12/2023).

[14] M. Terneborg, J. K. Ronnberg, and O. Schelen, "Application agnostic container migration and failover," presented at the 2021 IEEE 46th Conference on Local Computer Networks (LCN), Edmonton, AB, Canada: IEEE, Oct. 4, 2021, pp. 565–572, ISBN: 978-1-66541-886-7. DOI: 10.1109/LCN52139.2021.9525029. [Online]. Available: https://ieeexplore.ieee.org/document/9525029/ (visited on 01/17/2023).

[15] "Docker overview," Docker Documentation. (Jan. 17, 2023), [Online]. Available: https://docs.docker.com/get-started/overview/ (visited on 01/17/2023).

[16] J. Arundel and J. Domingus, "Cloud native DevOps with kubernetes," Jan. 24, 2019.

[17] Brendan Burns, Joe Beda, Kelsey Hightower, and Lachlan Evenson, *Kubernetes: Up and Running*. [S.l.]: O'Reilly Media, 2022, ISBN: 978-1-09-811020-8.

[18] "Kubernetes components," Kubernetes. Section: docs. (), [Online]. Available: https://kubernetes.io/docs/concepts/overview/components/ (visited on 02/05/2023).

[19] F. Füstös, K. Péter, N.-P. László, S.-G. Mátis, Z. Szabó, and C. Sulyok, "Managing a kubernetes cluster on raspberry pi devices," in *2022 IEEE 20th Jubilee International Symposium on Intelligent Systems and Informatics (SISY)*, ISSN: 1949-0488, Sep. 2022, pp. 133–138. DOI: 10.1109/SISY56759.2022.10036262.

[20] "Introduction to nomad | nomad | HashiCorp developer," Introduction to Nomad | Nomad | HashiCorp Developer. (), [Online]. Available: https://developer.hashicorp.com/nomad/tutorials/get-started/gs-overview (visited on 08/05/2023).

[21] F. Beetz and S. Harrer, "GitOps: The evolution of DevOps?" *IEEE Software*, vol. 39, no. 4, pp. 70–75, Jul. 2022, Conference Name: IEEE Software, ISSN: 1937-4194. DOI: 10.1109/MS.2021.3119106.

[22] "GitOps," GitOps. (), [Online]. Available: https://www.gitops.tech/ (visited on 02/12/2023).

[23] I.-C. Donca, O. P. Stan, M. Misaros, D. Gota, and L. Miclea, "Method for continuous integration and deployment using a pipeline generator for agile software projects," *Sensors*, vol. 22, no. 12, p. 4637, Jun. 20, 2022, ISSN: 1424-8220. DOI: 10.3390/s22124637. [Online]. Available: https://www.mdpi.com/1424-8220/22/12/4637 (visited on 02/13/2023).

[24] Yevgeniy Brikman, *Terraform: Up and Running*. [S.l.]: O'Reilly Media, 2022, ISBN: 978-1-09-811674-3.

[25] E. Truyen, H. Xie, and W. Joosen, "Vendor-agnostic reconfiguration of kubernetes clusters in cloud federations," *Future Internet*, vol. 15, no. 2, p. 63, Feb. 2023, Number: 2 Publisher: Multidisciplinary Digital Publishing Institute, ISSN: 1999-5903. DOI: 10.3390/fi15020063. [Online]. Available: https://www.mdpi.com/1999-5903/15/2/63 (visited on 08/05/2023).

[26] "Open cluster management." (), [Online]. Available: https://open-cluster-management.io/concepts/architecture/ (visited on 08/05/2023).

[27] S. Buchanan, J. Rangama, and N. Bellavance, "Helm charts for azure kubernetes service," in *Introducing Azure Kubernetes Service : A Practical Guide to Container Orchestration*, Berkeley, CA: Apress, 2020, pp. 151–189, ISBN: 978-1-4842-5519-3. DOI: 10.1007/978-1-4842-5519-3_8. [Online]. Available: https://doi.org/10.1007/978-1-4842-5519-3_8.

[28] "Hyscale," Hyscale. (), [Online]. Available: https://hyscale.io/ (visited on 02/14/2023).

[29] A. Alonso, H. S. Persson, and H. Kassaei, "5g architecture for hybrid and multi-cloud environments," *Ericsson Technology Review*, vol. 2022, no. 3, pp. 2–12, Mar. 2022, Conference Name: Ericsson Technology Review, ISSN: 0014-0171. DOI: 10.23919/ETR.2022.9904693.

[30] "CUE." (), [Online]. Available: https://cuelang.org/ (visited on 10/14/2023).

[31] "KubeVela." (), [Online]. Available: https://kubevela.io/ (visited on 02/14/2023).

[32] N. Rich, "Value analysis," *Value Engineering*, Jan. 2022.

[33] P. A. Koen, G. M. Ajamian, S. Boyce, *et al.*, "Effective methods, tools, and techniques," *The PDMA ToolBook for New Product Development*, 2002.

[34] R. W. Saaty. "THE ANALYTIC HIERARCHY PROCESS-WHAT IT IS AND HOW IT IS USED." ISSN: 0270-0255. (1987), [Online]. Available: https://reader.elsevier.com/reader/sd/pii/0270025587904738 (visited on 01/28/2023).

[35] I. Nowrin and F. Khanam, "Importance of cloud deployment model and security issues of software as a service (SaaS) for cloud computing," in *2019 International Conference on Applied Machine Learning (ICAML)*, May 2019, pp. 183–186. DOI: 10.1109/ICAML48257.2019.00042.

[36] A. V. Bataev, D. G. Rodionov, and D. A. Andreyeva, "Analysis of world trends in the field of cloud technology," in *2018 International Conference on Information Networking (ICOIN)*, Jan. 2018, pp. 594–598. DOI: 10.1109/ICOIN.2018.8343188.

[37] "What is hybrid infrastructure? | VMware glossary," VMware. (), [Online]. Available: https://www.vmware.com/content/vmware/vmware-published-sites/us/topics/glossary/content/hybrid-infrastructure-service (visited on 01/23/2023).

[38] D. Sitaram, S. Harwalkar, C. Sureka, *et al.*, "Orchestration based hybrid or multi clouds and interoperability standardization," in *2018 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM)*, Nov. 2018, pp. 67–71. DOI: 10.1109/CCEM.2018.00018.

[39] "IT infrastructure management: Definition, benefits and more," DevOps and Software Engineering Glossary Terms | Atatus. (Sep. 28, 2021), [Online]. Available: https://www.atatus.com/glossary/it-infrastructure-management/ (visited on 01/26/2023).

[40] "Main cloud service models: IaaS, PaaS and SaaS | stackscale." Section: Computing. (Aug. 9, 2022), [Online]. Available: https://www.stackscale.com/blog/cloud-service-models/ (visited on 01/26/2023).

[41] S. Kibe, S. Watanabe, K. Kunishima, R. Adachi, M. Yamagiwa, and M. Uehara, "PaaS on IaaS," in *2013 IEEE 27th International Conference on Advanced Information Networking and Applications (AINA)*, ISSN: 1550-445X, Mar. 2013, pp. 362–367. DOI: 10.1109/AINA.2013.73.

[42] Cloudfoundry, *Adaptation, not adoption, is the key to digital transformation*, Apr. 2019. [Online]. Available: https://www.cloudfoundry.org/wp-content/uploads/GPS-9-Adaptation-Not-Adoption-Report_FINAL.pdf.

[43] Á. Leiter, A. Hegyi, I. Kispál, P. Böõsy, N. Galambosi, and G. Z. Tar, "GitOps and kubernetes operator-based network function configuration," in *NOMS 2023-2023 IEEE/IFIP Network Operations and Management Symposium*, Miami, FL, USA: IEEE, May 8, 2023, pp. 1–5, ISBN: 978-1-66547-716-1. DOI: 10.1109/NOMS56928.2023.10154212. [Online]. Available: https://ieeexplore.ieee.org/document/10154212/ (visited on 10/14/2023).

[44] E. E. K. Senoo, E. Akansah, I. Mendonça, and M. Aritsugi, "Implementing SOLID principles for IoT arduino sensor code," in *2022 10th International Japan-Africa Conference on Electronics, Communications, and Computations (JAC-ECC)*, Alexandria, Egypt: IEEE, Dec. 19, 2022, pp. 21–26, ISBN: 978-1-66546-463-5. DOI: 10.1109/JAC-ECC56395.2022.10043950. [Online]. Available: https://ieeexplore.ieee.org/document/10043950/ (visited on 10/14/2023).

Appendix A

Analytic Hierarchy Process Steps

Table A.1: Criteria comparison normalized matrix

	Extensibility	**Ease of Use**	**Control**	**Observability**
Extensibility	0.190	0.167	0.176	0.4
Ease of Use	0.381	0.333	0.353	0.2
Control	0.381	0.333	0.353	0.3
Observability	0.048	0.167	0.118	0.1

For computing Consistency Index (CI):

$$CI = (\lambda max - n)/(n-1) = (4.20 - 4)/(4-1) = 0.067$$

, where n is the criteria number

For computing Consistency Ratio (CR):

$$CR = CI/RI = 0.067/0.90 = 0.074$$

, where RI is a tabulated value according to the criteria number

Table A.2: Alternative comparison considering *Extensibility* criteria

	KubeVela	**HyScale**	**Crossplane**
KubeVela	1	4	1
HyScale	1/4	1	1/4
Crossplane	1	4	1

Table A.3: Alternative comparison considering *Ease of Use* criteria

	KubeVela	**HyScale**	**Crossplane**
KubeVela	1	4	3
HyScale	1/4	1	1/3
Crossplane	1/3	3	1

Table A.4: Alternative comparison considering *Control* criteria

	KubeVela	**HyScale**	**Crossplane**
KubeVela	1	3	1/3
HyScale	1/3	1	1/3
Crossplane	3	3	1

Table A.5: Alternative comparison considering *Observability* criteria

	KubeVela	**HyScale**	**Crossplane**
KubeVela	1	5	3
HyScale	1/5	1	1/2
Crossplane	1/3	2	1

Table A.6: Normalized alternative comparison considering *Extensibility* criteria

	KubeVela	**HyScale**	**Crossplane**
KubeVela	0.444	0.444	0.444
HyScale	0.111	0.111	0.111
Crossplane	0.444	0.444	0.444

Table A.7: Normalized alternative comparison considering *Ease of Use* criteria

	KubeVela	**HyScale**	**Crossplane**
KubeVela	0.632	0.5	0.692
HyScale	0.158	0.125	0.0769
Crossplane	0.210	0.375	0.231

Table A.8: Normalized alternative comparison considering *Control* criteria

	KubeVela	**HyScale**	**Crossplane**
KubeVela	0.231	0.429	0.2
HyScale	0.077	0.142	0.2
Crossplane	0.692	0.429	0.6

TABLE A.9: Normalized alternative comparison considering *Observability* criteria

	KubeVela	**HyScale**	**Crossplane**
KubeVela	0.652	0.625	0.667
HyScale	0.130	0.125	0.111
Crossplane	0.217	0.25	0.222

Appendix B

Implementation Assets

```
resource "azuread_application" "main" {
  display_name = "Canaveral DevScope"
  owners       = ["5ce45fcb-c3d2-4923-bdff-197276914320"]

  web {
    redirect_uris = [
      "https://${local.hostnames.vela}/",
      "https://${local.hostnames.vela}/dex/callback",
      "https://${local.hostnames.vela}/callback",
      "https://${local.hostnames.grafana}/",
      "https://${local.hostnames.grafana}/login/azuread",
    ]
  }
}

resource "azuread_service_principal" "main" {
  application_id = azuread_application.main.application_id
  owners         = ["5ce45fcb-c3d2-4923-bdff-197276914320"]
}

resource "azuread_service_principal_password" "main" {
  service_principal_id = azuread_service_principal.main.object_id
  display_name         = "Authentication"
}

locals {
  openid_config = merge(var.openid_config, {
    client_id     = azuread_service_principal.main.application_id
    client_secret = azuread_service_principal_password.main.value
  })
  dex_openid_config = templatefile("${path.module}/templates/dex/config.json.
    tftpl", {
    config = merge(local.openid_config, { redirect_uri = "https://${local.
    hostnames.vela}/dex/callback" })
  })
}
```

Listing B.1: OpenID Connect resources and variables

```
{
    "type": "oidc",
    "oidc": {
        "issuer": "${config.issuer}",
        "clientID": "${config.client_id}",
        "clientSecret": "${config.client_secret}",
        "redirectURI": "${config.redirect_uri}",
        "insecureSkipEmailVerified": true,
        "userIDKey": "${config.user_id_key}"
    }
}
```

Listing B.2: OpenID Connect credentials template

```
resource "kubernetes_job" "join_managed_cluster" {
  metadata {
    name      = "join-managed-cluster"
    namespace = "kube-system"
  }

  wait_for_completion = true

  spec {
    template {
      metadata {}
      spec {
        container {
          name    = "clusteradm"
          image   = "devscope.azurecr.io/ocm-tools:latest"
          command = ["./scripts/join.sh"]

          volume_mount {
            name       = "hub-kubeconfig"
            mount_path = "/root/.kube/hub.yaml"
            sub_path   = "hub.yaml"
          }
          volume_mount {
            name       = "spoke-kubeconfig"
            mount_path = "/root/.kube/${var.spoke_cluster_name}.yaml"
            sub_path   = "${var.spoke_cluster_name}.yaml"
          }
          volume_mount {
            name       = "scripts"
            mount_path = "/scripts"
          }

          image_pull_policy = "Always"
        }

        image_pull_secrets {
          name = kubernetes_secret.container_registry.metadata[0].name
        }

        volume {
          name = "hub-kubeconfig"
          secret {
            secret_name = kubernetes_secret.hub_kubeconfig.metadata[0].
    name
            items {
              key  = "config"
```

```
                path = "hub.yaml"
              }
            }
          }
          volume {
            name = "spoke-kubeconfig"
            secret {
              secret_name = kubernetes_secret.spoke_kubeconfig.metadata
    [0].name
              items {
                key  = "config"
                path = "${var.spoke_cluster_name}.yaml"
              }
            }
          }
          volume {
            name = "scripts"
            config_map {
              name         = "scripts"
              default_mode = "0700"
            }
          }
        }
      }
      completions = 1
    }
}
```

Listing B.3: Kubernetes Job for joining hub cluster

```
import (
    "encoding/base64"
)

"azure-mssql-server": {
  alias: ""
  annotations: {}
  attributes: {
    workload: definition: {
      apiVersion: "sql.azure.upbound.io/v1beta1"
      kind:       "MSSQLServer"
    }
    status: {
      healthPolicy: #"""
      isHealth: (context.output.status.conditions[0].type == "Ready") && (
    context.output.status.conditions[0].status == "True")
      """#
    }
  }
  description: ""
  labels: {}
  type: "component"
}

template: {
  parameter: {
    name: string
    adminUser: string
    adminPassword: string
    location: *"NorthEurope" | string
    resourceGroupName: *"rg-\(context.name)-\(context.namespace)" | string
    resourceGroupLocation: *"NorthEurope" | string
    version: *"12.0" | string
```

```
    databases: *[] | [...string]
    allowedIpAddresses: *[] | [...string]
  }

  output: {
    apiVersion: "sql.azure.upbound.io/v1beta1"
    kind:       "MSSQLServer"
    metadata: {
      name: parameter.name
      labels: {
        "app.oam.dev/cluster": "local"
      }
    }
    spec: {
      forProvider: {
        administratorLogin: parameter.adminUser
        administratorLoginPasswordSecretRef: {
          key: "admin-login-password"
          name: "\(parameter.name)-admin-secret"
          namespace: context.namespace
        }
        location: parameter.location
        minimumTlsVersion: "1.2"
        resourceGroupName: parameter.resourceGroupName
        version: parameter.version
      }
    }
  }

  outputs: {
    resourceGroup: {
      apiVersion: "azure.upbound.io/v1beta1"
      kind:       "ResourceGroup"
      metadata: {
        name: parameter.resourceGroupName
        labels: {
          "app.oam.dev/cluster": "local"
        }
      }
      spec: forProvider: {
        location: parameter.resourceGroupLocation
        tags:
          "kubevela.app": "\(context.namespace)/\(context.appName)"
      }
    }

    secret: {
      apiVersion: "v1"
      kind:       "Secret"
      metadata: {
        name: "\(parameter.name)-admin-secret"
        labels: {
          "app.oam.dev/cluster": "local"
        }
      }
      type: "Opaque"
      data:
          "admin-login-password": base64.Encode(null, parameter.adminPassword)
    }

    connectionSecret: {
      apiVersion: "v1"
      kind:       "Secret"
      metadata: {
        name: "\(context.name)-connection-secret"
        labels: {
          "app.oam.dev/cluster": context.cluster
```

```
100         }
101       }
102       type: "Opaque"
103       data: {
104         "connectionString": base64.Encode(null, "Server=\(parameter.name).
      database.windows.net;Database=$(DATABASE);User Id=\(parameter.adminUser);
      Password=\(parameter.adminPassword);Encrypt=False;TrustServerCertificate=
      true")
105       }
106     }
107
108     for i, ip in parameter.allowedIpAddresses {
109       "\(parameter.name)-allow-rule-\(i)": {
110         apiVersion: "sql.azure.upbound.io/v1beta1"
111         kind:       "MSSQLFirewallRule"
112         metadata: {
113           name: "\(parameter.name)-allow-rule-\(i)"
114           labels: {
115             "app.oam.dev/cluster": "local"
116           }
117         }
118         spec: {
119           forProvider: {
120             startIpAddress: ip
121             endIpAddress: ip
122             serverIdRef: name: parameter.name
123           }
124         }
125       }
126     }
127
128     for dbName in parameter.databases {
129       "database-\(dbName)": {
130         apiVersion: "sql.azure.upbound.io/v1beta1"
131         kind:       "MSSQLDatabase"
132         metadata: {
133           name: dbName
134           labels: {
135             "app.oam.dev/cluster": "local"
136           }
137         }
138         spec: {
139           forProvider: {
140             serverIdRef: name: parameter.name
141           }
142         }
143       }
144     }
145   }
146 }
```

Listing B.4: Azure MSSQL Server component definition

Appendix C

Experimentation Usability Questionnaire

This appendix presents the developed questionnaire, the population that answered it and respective details, as well as the rating statistics by question.

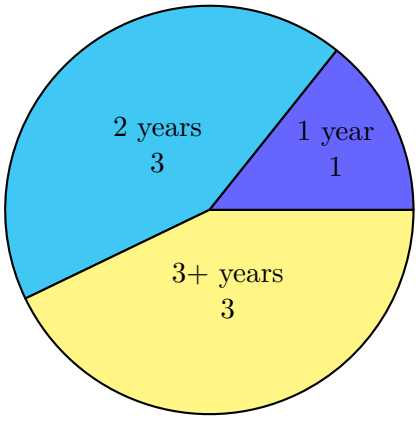


FIGURE C.1: Number of participants per years of experience

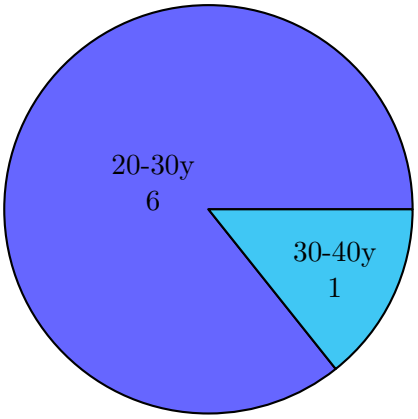


FIGURE C.2: Number of participants per age

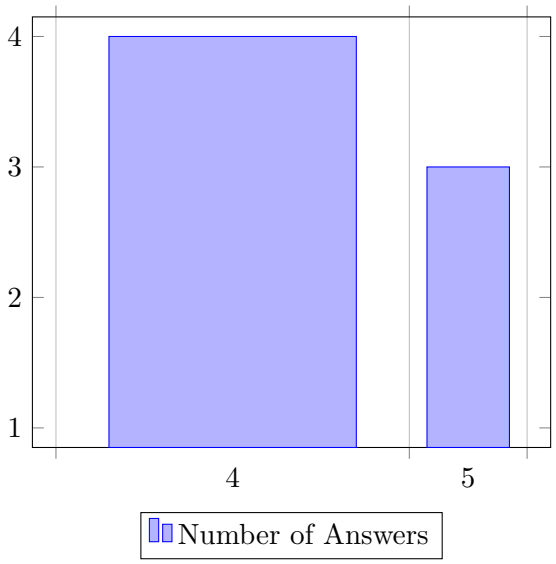


FIGURE C.3: Number of answers per rating (question 1)

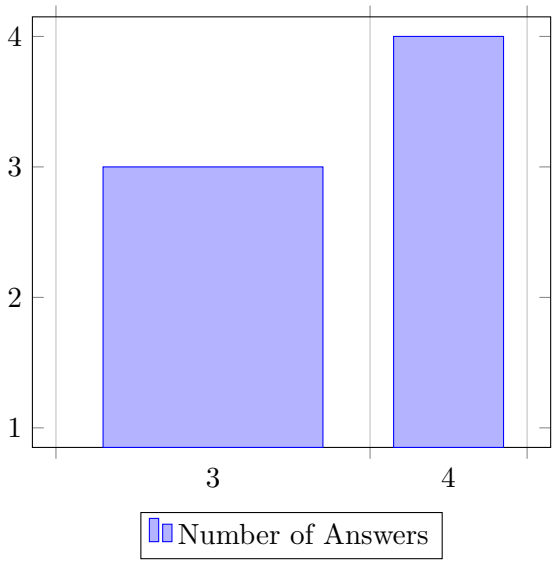


FIGURE C.4: Number of answers per rating (question 2)

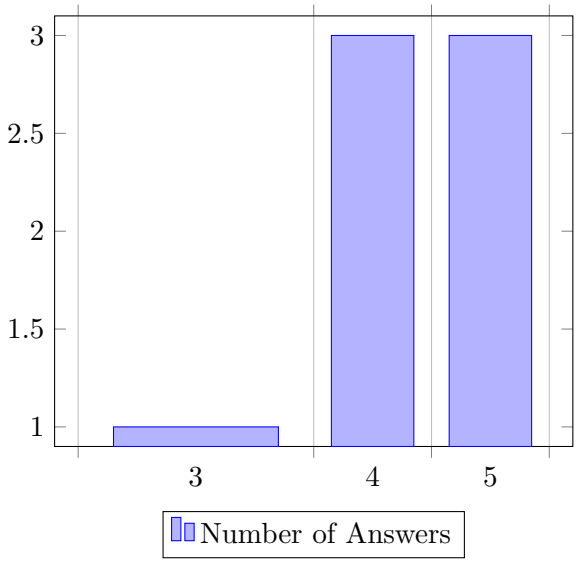


FIGURE C.5: Number of answers per rating (question 3)

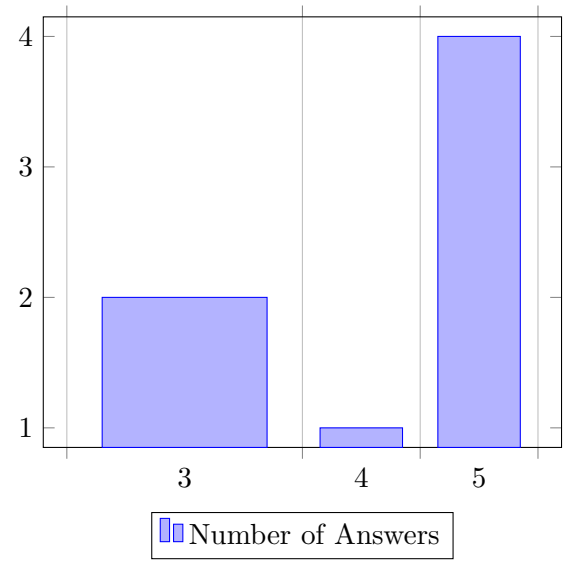


FIGURE C.6: Number of answers per rating (question 4)

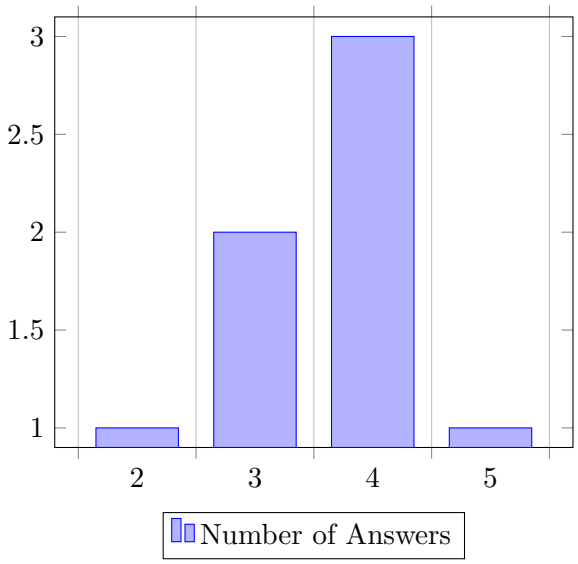


FIGURE C.7: Number of answers per rating (question 5)

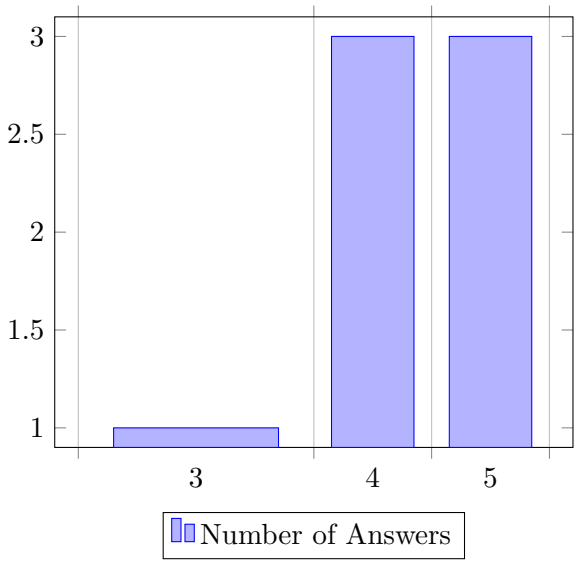


FIGURE C.8: Number of answers per rating (question 6)

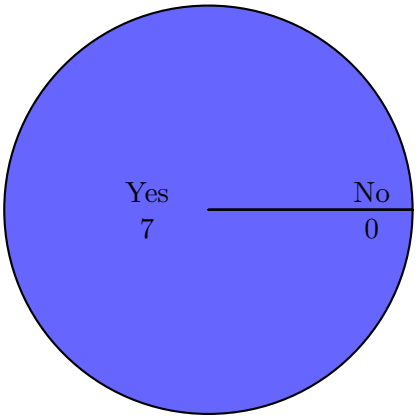
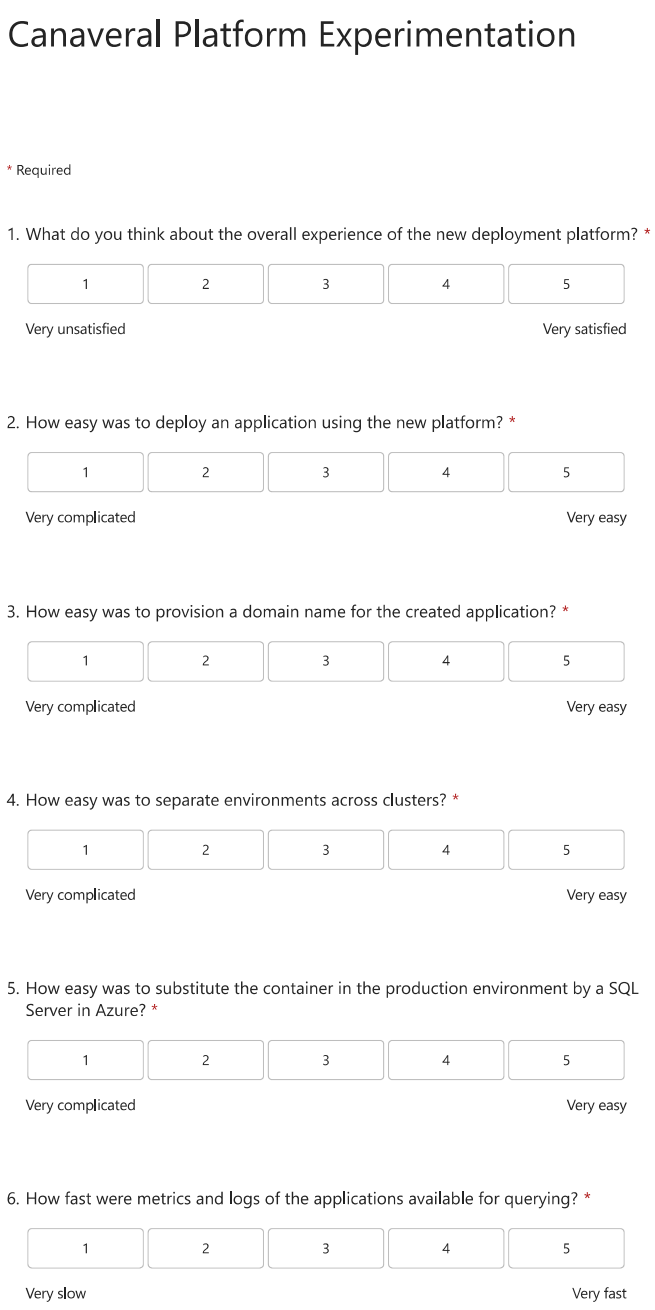


FIGURE C.9: Number of positive and negative answers (question 7)



Canaveral Platform Experimentation

* Required

1. What do you think about the overall experience of the new deployment platform? *

1 2 3 4 5

Very unsatisfied Very satisfied

2. How easy was to deploy an application using the new platform? *

1 2 3 4 5

Very complicated Very easy

3. How easy was to provision a domain name for the created application? *

1 2 3 4 5

Very complicated Very easy

4. How easy was to separate environments across clusters? *

1 2 3 4 5

Very complicated Very easy

5. How easy was to substitute the container in the production environment by a SQL Server in Azure? *

1 2 3 4 5

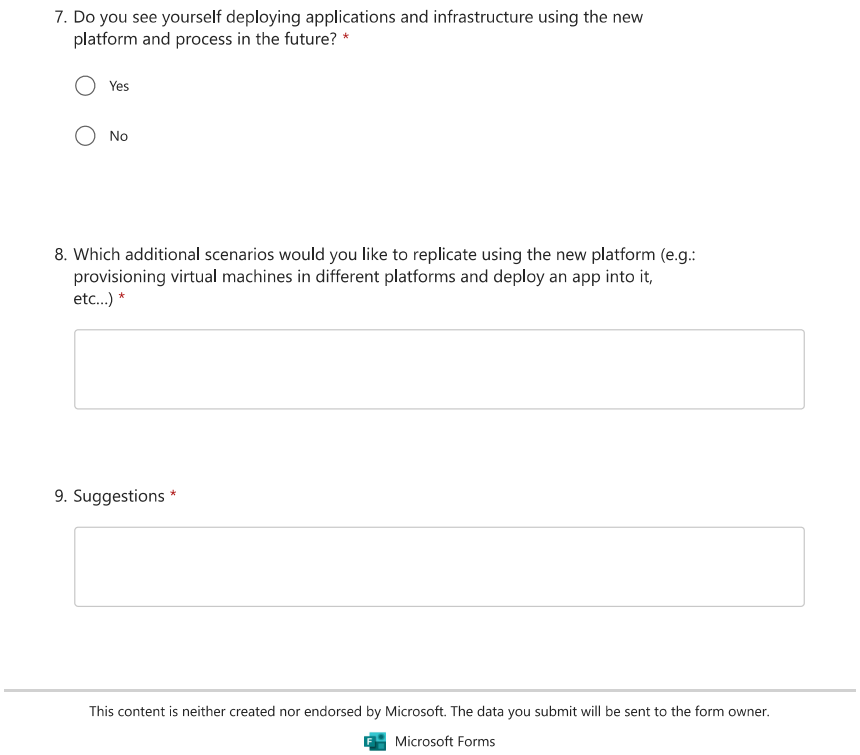
Very complicated Very easy

6. How fast were metrics and logs of the applications available for querying? *

1 2 3 4 5

Very slow Very fast

FIGURE C.10: Experimentation usability questionnaire (page 1)


7. Do you see yourself deploying applications and infrastructure using the new platform and process in the future? *

Yes

No

8. Which additional scenarios would you like to replicate using the new platform (e.g.: provisioning virtual machines in different platforms and deploy an app into it, etc...) *

9. Suggestions *

FIGURE C.11: Experimentation usability questionnaire (page 2)

Appendix D

Evaluation and Experimentation Assets

```yaml
apiVersion: core.oam.dev/v1beta1
kind: Application
metadata:
  name: contacts-app-vc
  namespace: experimentation
spec:
  components:
    - name: app
      type: webservice
      dependsOn: ["database"]
      properties:
        image: vitorjcorreia/blazor-sample-app:latest
        ports:
          - port: 80
            expose: true
        env:
          - name: DATABASE
            value: contactsdb
          - name: ConnectionStrings__DefaultConnection
            value: "Server=database;Database=$(DATABASE);User Id=sa;
    Password=<REDACTED>;Encrypt=False;TrustServerCertificate=true"
      traits:
        - type: gateway
          properties:
            domain: contacts-app-vc.devscope.com
            http:
              "/": 80
        - type: cloudflare-dns
          properties:
            publicIpAddress: "<REDACTED>"
    - name: database
      type: webservice
      properties:
        image: mcr.microsoft.com/mssql/server:2019-latest
        ports:
          - port: 1433
            expose: true
        env:
          - name: SA_PASSWORD
            value: <REDACTED>
          - name: ACCEPT_EULA
```

```
            value: "Y"
  policies:
    - name: target-dev
      type: topology
      properties:
        namespace: vc-dev
        clusters: ["local"]
    - name: target-prod
      type: topology
      properties:
        namespace: vc-prod
        clusters: ["bonfim"]
    - name: create-database-cloud
      type: override
      properties:
        components:
          - name: database
            type: azure-mssql-server
            properties:
              name: mssql-vc-prod-ne
              adminUser: <REDACTED>
              adminPassword: <REDACTED>
              databases: ["contactsdb"]
              allowedIpAddresses: ["<REDACTED>"]
    - name: override-app-traits
      type: override
      properties:
        components:
          - name: app
            properties:
            traits:
              - type: gateway
                properties:
                  domain: contacts-app-vc-bonfim.devscope.net
                  http:
                    "/": 80
              - type: env
                properties:
                  unset: ["ConnectionStrings__DefaultConnection"]
              - type: storage
                properties:
                  secret:
                    - name: database-connection-secret
                      mountToEnvs:
                        - envName: ConnectionStrings__DefaultConnection
                          secretKey: connectionString
    - name: enable-logging-prod
      type: container-log
      properties:
        namespace: vc-prod
        cluster: bonfim
  workflow:
    steps:
      - name: deploy2dev
        type: deploy
        properties:
          policies: ["target-dev"]
      - name: manual-approval
```

```
99          type: suspend
100       - name: deploy2prod
101         type: deploy
102         properties:
103           policies: ["create-database-cloud", "override-app-traits", "
    enable-logging-prod", "target-prod"]
```

Listing D.1: Final stage of the application for usability evaluation

```
1  apiVersion: core.oam.dev/v1beta1
2  kind: Application
3  metadata:
4    name: eshop
5  spec:
6    components:
7      - name: seq
8        type: webservice
9        properties:
10         image: datalust/seq:latest
11         ports:
12           - port: 80
13             expose: true
14         env:
15           - name: ACCEPT_EULA
16             value: "Y"
17     - name: sqldata
18       type: webservice
19       properties:
20         image: mcr.microsoft.com/mssql/server:2019-latest
21         ports:
22           - port: 1433
23             expose: true
24         env:
25           - name: SA_PASSWORD
26             value: <REDACTED>
27           - name: ACCEPT_EULA
28             value: "Y"
29     - name: nosqldata
30       type: webservice
31       properties:
32         image: mongo:5.0.19
33         ports:
34           - port: 27017
35             expose: true
36     - name: basketdata
37       type: webservice
38       properties:
39         image: redis:alpine
40         ports:
41           - port: 6379
42             expose: true
43     - name: rabbitmq
44       type: webservice
45       properties:
46         image: rabbitmq:3-management-alpine
47         cmd: ["/bin/bash"]
48         args:
49           - -c
```

```
50          - |
51            #!/bin/bash
52            rabbitmq-server &
53
54            rabbitmq-diagnostics --quiet check_running
55            HEALTH=$?
56            while [ ! "$HEALTH" -eq 0 ]
57            do
58              rabbitmq-diagnostics --quiet check_running
59              HEALTH=$?
60              echo $HEALTH
61              sleep 1
62            done
63
64            rabbitmqctl add_user admin <REDACTED>
65            rabbitmqctl set_user_tags admin administrator
66            rabbitmqctl set_permissions -p / admin ".*" ".*" ".*"
67
68            tail -f /dev/null
69        ports:
70          - port: 5672
71            expose: true
72          - port: 15672
73            expose: true
74    - name: identity-api
75      type: webservice
76      dependsOn: ["sqldata"]
77      properties:
78        image: eshop/identity.api:linux-dev
79        ports:
80          - port: 80
81            expose: true
82        env:
83          - name: ASPNETCORE_ENVIRONMENT
84            value: Development
85          - name: ASPNETCORE_URLS
86            value: http://0.0.0.0:80
87          - name: SpaClient
88            value: https://eshop-dev-spa.devscope.com
89          - name: XamarinCallback
90            value: https://eshop-dev-identity-api.devscope.com/
    xamarincallback
91          - name: MvcClient
92            value: https://eshop-dev-mvc.devscope.com
93          - name: BasketApiClient
94            value: https://eshop-dev-basket-api.devscope.com
95          - name: OrderingApiClient
96            value: https://eshop-dev-ordering-api.devscope.com
97          - name: MobileShoppingAggClient
98            value: https://eshop-dev-mobile-shopping-agg.devscope.com
99          - name: WebShoppingAggClient
100           value: https://eshop-dev-web-shopping-agg.devscope.com
101         - name: WebhooksApiClient
102           value: https://eshop-dev-webhooks-api.devscope.com
103         - name: WebhooksWebClient
104           value: https://eshop-dev-webhooks-client.devscope.com
105         - name: UseCustomizationData
106           value: "True"
```

```
107             - name: OrchestratorType
108               value: K8S
109         traits:
110           - type: gateway
111             properties:
112               domain: eshop-dev-identity-api.devscope.com
113               http:
114                 "/": 80
115       - name: basket-api
116         type: webservice
117         dependsOn: ["basketdata", "identity-api", "rabbitmq"]
118         properties:
119           image: eshop/basket.api:linux-dev
120           ports:
121             - port: 80
122               expose: true
123             - port: 81
124               expose: true
125           env:
126             - name: ASPNETCORE_ENVIRONMENT
127               value: Development
128             - name: Kestrel__Endpoints__HTTP__Url
129               value: http://0.0.0.0:80
130             - name: Kestrel__Endpoints__GRPC__Url
131               value: http://0.0.0.0:81
132             - name: Kestrel__Endpoints__GRPC__Protocols
133               value: Http2
134             - name: IdentityUrl
135               value: http://identity-api
136             - name: Identity__Url
137               value: http://identity-api
138             - name: Identity__ExternalUrl
139               value: http://identity-api
140             - name: ConnectionStrings__Redis
141               value: basketdata
142             - name: ConnectionStrings__EventBus
143               value: rabbitmq
144             - name: EventBus__UserName
145               value: admin
146             - name: EventBus__Password
147               value: <REDACTED>
148             - name: OrchestratorType
149               value: K8S
150             - name: PATH_BASE
151               value: /basket-api
152         traits:
153           - type: gateway
154             properties:
155               domain: eshop-dev-basket-api.devscope.com
156               http:
157                 "/": 80
158       - name: catalog-api
159         type: webservice
160         dependsOn: ["sqldata", "rabbitmq"]
161         properties:
162           image: eshop/catalog.api:linux-dev
163           ports:
164             - port: 80
```

```
            expose: true
          - port: 81
            expose: true
        env:
          - name: ASPNETCORE_ENVIRONMENT
            value: Development
          - name: Kestrel__Endpoints__HTTP__Url
            value: http://0.0.0.0:80
          - name: Kestrel__Endpoints__GRPC__Url
            value: http://0.0.0.0:81
          - name: Kestrel__Endpoints__GRPC__Protocols
            value: Http2
          - name: ConnectionStrings__EventBus
            value: rabbitmq
          - name: PicBaseUrl
            value: http://webshoppingagg/c/api/v1/catalog/items/[0]/pic/
          - name: EventBus__UserName
            value: admin
          - name: EventBus__Password
            value: <REDACTED>
          - name: UseCustomizationData
            value: "True"
          - name: AzureStorageEnabled
            value: "False"
          - name: PATH_BASE
            value: /catalog-api
          - name: OrchestratorType
            value: K8S
    - name: ordering-api
      type: webservice
      dependsOn: ["sqldata", "rabbitmq"]
      properties:
        image: eshop/ordering.api:linux-dev
        ports:
          - port: 80
            expose: true
          - port: 81
            expose: true
        env:
          - name: ASPNETCORE_ENVIRONMENT
            value: Development
          - name: Kestrel__Endpoints__HTTP__Url
            value: http://0.0.0.0:80
          - name: Kestrel__Endpoints__GRPC__Url
            value: http://0.0.0.0:81
          - name: Kestrel__Endpoints__GRPC__Protocols
            value: Http2
          - name: Identity__Url
            value: http://identity-api
          - name: Identity__ExternalUrl
            value: http://identity-api
          - name: ConnectionStrings__EventBus
            value: rabbitmq
          - name: EventBus__UserName
            value: admin
          - name: EventBus__Password
            value: <REDACTED>
          - name: UseCustomizationData
```

```
            value: "True"
          - name: AzureStorageEnabled
            value: "False"
          - name: CheckUpdateTime
            value: "30000"
          - name: PATH_BASE
            value: /ordering-api
          - name: UseLoadTest
            value: "False"
          - name: GRPC_PORT
            value: "81"
          - name: PORT
            value: "80"
          - name: OrchestratorType
            value: K8S
      traits:
        - type: gateway
          properties:
            domain: eshop-dev-ordering-api.devscope.com
            http:
              "/": 80
    - name: ordering-backgroundtasks
      type: webservice
      dependsOn: ["sqldata", "rabbitmq"]
      properties:
        image: eshop/ordering.backgroundtasks:linux-dev
        ports:
          - port: 80
            expose: true
        env:
          - name: ASPNETCORE_ENVIRONMENT
            value: Development
          - name: ASPNETCORE_URLS
            value: http://0.0.0.0:80
          - name: ConnectionStrings__EventBus
            value: rabbitmq
          - name: EventBus__UserName
            value: admin
          - name: EventBus__Password
            value: <REDACTED>
          - name: UseCustomizationData
            value: "True"
          - name: AzureStorageEnabled
            value: "False"
          - name: CheckUpdateTime
            value: "30000"
          - name: UseLoadTest
            value: "False"
          - name: GracePeriodTime
            value: "1"
          - name: OrchestratorType
            value: K8S
    - name: payment-api
      type: webservice
      dependsOn: ["rabbitmq"]
      properties:
        image: eshop/payment.api:linux-dev
        ports:
```

```
281           - port: 80
282             expose: true
283         env:
284           - name: ASPNETCORE_ENVIRONMENT
285             value: Development
286           - name: ASPNETCORE_URLS
287             value: http://0.0.0.0:80
288           - name: ConnectionStrings__EventBus
289             value: rabbitmq
290           - name: EventBus__UserName
291             value: admin
292           - name: EventBus__Password
293             value: <REDACTED>
294           - name: AzureServiceBusEnabled
295             value: "False"
296           - name: OrchestratorType
297             value: K8S
298     - name: webhooks-api
299       type: webservice
300       dependsOn: ["sqldata"]
301       properties:
302         image: eshop/webhooks.api:linux-dev
303         ports:
304           - port: 80
305             expose: true
306         env:
307           - name: ASPNETCORE_ENVIRONMENT
308             value: Development
309           - name: ASPNETCORE_URLS
310             value: http://0.0.0.0:80
311           - name: ConnectionStrings__EventBus
312             value: rabbitmq
313           - name: EventBus__UserName
314             value: admin
315           - name: EventBus__Password
316             value: <REDACTED>
317           - name: Identity__Url
318             value: http://identity-api
319           - name: Identity__ExternalUrl
320             value: http://identity-api
321       traits:
322         - type: gateway
323           properties:
324             domain: eshop-dev-webhooks-api.devscope.com
325             http:
326               "/": 80
327     - name: mobileshoppingagg
328       dependsOn: ["nosqldata", "sqldata", "identity-api", "rabbitmq", "
    ordering-api", "catalog-api", "basket-api"]
329       type: webservice
330       properties:
331         image: eshop/mobileshoppingagg:linux-dev
332         ports:
333           - port: 80
334             expose: true
335         env:
336           - name: ASPNETCORE_ENVIRONMENT
337             value: Development
```

```
338             - name: urls__basket
339               value: http://basket-api
340             - name: urls__catalog
341               value: http://catalog-api
342             - name: urls__orders
343               value: http://ordering-api
344             - name: urls__identity
345               value: http://identity-api
346             - name: urls__grpcBasket
347               value: http://basket-api:81
348             - name: urls__grpcCatalog
349               value: http://catalog-api:81
350             - name: urls__grpcOrdering
351               value: http://ordering-api:81
352             - name: CatalogUrlHC
353               value: http://catalog-api/hc
354             - name: OrderingUrlHC
355               value: http://ordering-api/hc
356             - name: IdentityUrlHC
357               value: http://identity-api/hc
358             - name: BasketUrlHC
359               value: http://basket-api/hc
360             - name: PaymentUrlHC
361               value: http://payment-api/hc
362             - name: Identity__Url
363               value: http://identity-api
364             - name: Identity__ExternalUrl
365               value: http://identity-api
366         traits:
367           - type: gateway
368             properties:
369               domain: eshop-dev-mobile-shopping-agg.devscope.com
370               http:
371                 "/": 80
372       - name: webshoppingagg
373         type: webservice
374         dependsOn: ["nosqldata", "sqldata", "identity-api", "rabbitmq", "
      ordering-api", "catalog-api", "basket-api"]
375         properties:
376           image: eshop/webshoppingagg:linux-dev
377           ports:
378             - port: 80
379               expose: true
380           env:
381             - name: ASPNETCORE_ENVIRONMENT
382               value: Development
383             - name: urls__basket
384               value: http://basket-api
385             - name: urls__catalog
386               value: http://catalog-api
387             - name: urls__orders
388               value: http://ordering-api
389             - name: urls__identity
390               value: http://identity-api
391             - name: urls__grpcBasket
392               value: http://basket-api:81
393             - name: urls__grpcCatalog
394               value: http://catalog-api:81
```

```
395             - name: urls__grpcOrdering
396               value: http://ordering-api:81
397             - name:
    ReverseProxy__Clusters__basket__Destinations__destination0__Address
398               value: http://basket-api
399             - name:
    ReverseProxy__Clusters__catalog__Destinations__destination0__Address
400               value: http://catalog-api
401             - name:
    ReverseProxy__Clusters__orders__Destinations__destination0__Address
402               value: http://ordering-api
403             - name:
    ReverseProxy__Clusters__signalr__Destinations__destination0__Address
404               value: http://ordering-signalrhub
405             - name: CatalogUrlHC
406               value: http://catalog-api/hc
407             - name: OrderingUrlHC
408               value: http://ordering-api/hc
409             - name: IdentityUrlHC
410               value: http://identity-api/hc
411             - name: BasketUrlHC
412               value: http://basket-api/hc
413             - name: PaymentUrlHC
414               value: http://payment-api/hc
415             - name: Identity__Url
416               value: http://identity-api
417             - name: Identity__ExternalUrl
418               value: http://identity-api
419         traits:
420           - type: gateway
421             properties:
422               domain: eshop-dev-web-shopping-agg.devscope.com
423               http:
424                 "/": 80
425       - name: ordering-signalrhub
426         type: webservice
427         dependsOn: ["nosqldata", "sqldata", "identity-api", "rabbitmq", "
    ordering-api", "catalog-api", "basket-api"]
428         properties:
429           image: eshop/ordering.signalrhub:linux-dev
430           ports:
431             - port: 80
432               expose: true
433           env:
434             - name: ASPNETCORE_ENVIRONMENT
435               value: Development
436             - name: ASPNETCORE_URLS
437               value: http://0.0.0.0:80
438             - name: ConnectionStrings__EventBus
439               value: rabbitmq
440             - name: EventBus__UserName
441               value: admin
442             - name: EventBus__Password
443               value: <REDACTED>
444             - name: AzureServiceBusEnabled
445               value: "False"
446             - name: Identity__Url
447               value: http://identity-api
```

```
448             - name: OrchestratorType
449               value: K8S
450       - name: webstatus
451         type: webservice
452         properties:
453           image: eshop/webstatus:linux-dev
454           ports:
455             - port: 80
456               expose: true
457           env:
458             - name: ASPNETCORE_ENVIRONMENT
459               value: Production
460             - name: ASPNETCORE_URLS
461               value: http://0.0.0.0:80
462             - name: HealthChecksUI__HealthChecks__0__Name
463               value: WebMVC HTTP Check
464             - name: HealthChecksUI__HealthChecks__0__Uri
465               value: http://webmvc/hc
466             - name: HealthChecksUI__HealthChecks__1__Name
467               value: WebSPA HTTP Check
468             - name: HealthChecksUI__HealthChecks__1__Uri
469               value: http://webspa/hc
470             - name: HealthChecksUI__HealthChecks__2__Name
471               value: Web Shopping Aggregator GW HTTP Check
472             - name: HealthChecksUI__HealthChecks__2__Uri
473               value: http://webshoppingagg/hc
474             - name: HealthChecksUI__HealthChecks__3__Name
475               value: Mobile Shopping Aggregator HTTP Check
476             - name: HealthChecksUI__HealthChecks__3__Uri
477               value: http://mobileshoppingagg/hc
478             - name: HealthChecksUI__HealthChecks__4__Name
479               value: Ordering HTTP Check
480             - name: HealthChecksUI__HealthChecks__4__Uri
481               value: http://ordering-api/hc
482             - name: HealthChecksUI__HealthChecks__5__Name
483               value: Basket HTTP Check
484             - name: HealthChecksUI__HealthChecks__5__Uri
485               value: http://basket-api/hc
486             - name: HealthChecksUI__HealthChecks__6__Name
487               value: Catalog HTTP Check
488             - name: HealthChecksUI__HealthChecks__6__Uri
489               value: http://catalog-api/hc
490             - name: HealthChecksUI__HealthChecks__7__Name
491               value: Identity HTTP Check
492             - name: HealthChecksUI__HealthChecks__7__Uri
493               value: http://identity-api/hc
494             - name: HealthChecksUI__HealthChecks__8__Name
495               value: Payments HTTP Check
496             - name: HealthChecksUI__HealthChecks__8__Uri
497               value: http://payment-api/hc
498             - name: HealthChecksUI__HealthChecks__9__Name
499               value: Ordering SignalRHub HTTP Check
500             - name: HealthChecksUI__HealthChecks__9__Uri
501               value: http://ordering-signalrhub/hc
502             - name: HealthChecksUI__HealthChecks__10__Name
503               value: Ordering HTTP Background Check
504             - name: HealthChecksUI__HealthChecks__10__Uri
505               value: http://ordering-backgroundtasks/hc
```

```
506             - name: OrchestratorType
507               value: K8S
508       - name: webspa
509         type: webservice
510         properties:
511           image: eshop/webspa:linux-dev
512           ports:
513             - port: 80
514               expose: true
515           env:
516             - name: ASPNETCORE_ENVIRONMENT
517               value: Production
518             - name: ASPNETCORE_URLS
519               value: http://0.0.0.0:80
520             - name: IdentityUrl
521               value: https://eshop-dev-identity-api.devscope.com
522             - name: PurchaseUrl
523               value: https://eshop-dev-web-shopping-agg.devscope.com
524             - name: IdentityUrlHC
525               value: http://identity-api/hc
526             - name: SignalrHubUrl
527               value: https://eshop-dev-web-shopping-agg.devscope.com
528             - name: UseCustomizationData
529               value: "True"
530             - name: OrchestratorType
531               value: K8S
532         traits:
533           - type: gateway
534             properties:
535               domain: eshop-dev-spa.devscope.com
536               http:
537                 "/": 80
538       - name: webmvc
539         type: webservice
540         dependsOn: ["webshoppingagg"]
541         properties:
542           image: eshop/webmvc:linux-dev
543           ports:
544             - port: 80
545               expose: true
546           env:
547             - name: ASPNETCORE_ENVIRONMENT
548               value: Development
549             - name: ASPNETCORE_URLS
550               value: http://0.0.0.0:80
551             - name: PurchaseUrl
552               value: https://eshop-dev-web-shopping-agg.devscope.com
553             - name: IdentityUrl
554               value: https://eshop-dev-identity-api.devscope.com
555             - name: SignalrHubUrl
556               value: https://eshop-dev-web-shopping-agg.devscope.com
557             - name: IdentityUrlHC
558               value: http://identity-api/hc
559             - name: UseCustomizationData
560               value: "True"
561             - name: UseLoadTest
562               value: "False"
563             - name: OrchestratorType
```

```
564             value: K8S
565       traits:
566       - type: gateway
567         properties:
568           domain: eshop-dev-mvc.devscope.com
569           http:
570             "/": 80
571     - name: webhooks-client
572       type: webservice
573       dependsOn: ["webhooks-api"]
574       properties:
575         image: eshop/webhooks.client:linux-dev
576         ports:
577           - port: 80
578             expose: true
579         env:
580           - name: ASPNETCORE_URLS
581             value: http://0.0.0.0:80
582           - name: Token
583             value: 6168DB8D-DC58-4094-AF24-483278923590
584           - name: IdentityUrl
585             value: http://identity-api
586           - name: CallBackUrl
587             value: http://0.0.0.0
588           - name: WebhooksUrl
589             value: http://webhooks-api
590           - name: SelfUrl
591             value: http://webhooks-client/
592       traits:
593         - type: gateway
594           properties:
595             domain: eshop-dev-webhooks-client.devscope.com
596             http:
597               "/": 80
598   policies:
599     - name: target-dev
600       type: topology
601       properties:
602         clusters: ["local"]
603         namespace: eshop-dev
604     - name: target-local-prod
605       type: topology
606       properties:
607         clusters: ["local"]
608         namespace: eshop-prod
609     - name: target-prod
610       type: topology
611       properties:
612         clusters: ["bonfim"]
613         namespace: eshop-prod
614     - name: change-database-type
615       type: override
616       properties:
617         selector: ["sqldata"]
618         components:
619           - name: sqldata
620             type: azure-mssql-server
621             properties:
```

```
622                 name: mssql-eshop-prod-ne
623                 adminUser: azureadmin
624                 adminPassword: <REDACTED>
625                 location: NorthEurope
626                 resourceGroupName: rg-eshop-prod-ne
627                 resourceGroupLocation: NorthEurope
628                 databases: ["microsoft-eshoponcontainers-services-
    catalogdb"]
629       - name: disable-reload-config-files
630         type: override
631         properties:
632           components:
633             - type: webservice
634               traits:
635               - type: env
636                 properties:
637                   env:
638                     DOTNET_HOSTBUILDER__RELOADCONFIGONCHANGE: "false"
639       - name: add-db-connection-string
640         type: override
641         properties:
642           components:
643             - name: identity-api|catalog-api|ordering-api|webhooks-api
644               traits:
645               - type: storage
646                 properties:
647                   secret:
648                     - name: sqldata-connection-details
649                       mountToEnvs:
650                         - envName: MSSQL_SERVER
651                           secretKey: endpoint
652                         - envName: MSSQL_USER
653                           secretKey: username
654                         - envName: MSSQL_PASSWORD
655                           secretKey: password
656                       data:
657                         endpoint: c3FsZGF0YQ==
658                         username: c2E=
659                         password: <REDACTED>
660             - name: identity-api
661               traits:
662                 - type: env
663                   properties:
664                     env:
665                       ConnectionStrings__IdentityDb: "Server=sqldata;
    Database=Microsoft.eShopOnContainers.Service.IdentityDb;User Id=sa;
    Password=<REDACTED>;Encrypt=False;TrustServerCertificate=true"
666             - name: catalog-api
667               traits:
668                 - type: env
669                   properties:
670                     env:
671                       ConnectionStrings__CatalogDb: "Server=sqldata;
    Database=Microsoft.eShopOnContainers.Service.CatalogDb;User Id=sa;
    Password=<REDACTED>;Encrypt=False;TrustServerCertificate=true"
672             - name: ordering-api
673               traits:
674                 - type: env
```

```
675                 properties:
676                   env:
677                     ConnectionStrings__OrderingDb: "Server=sqldata;
    Database=Microsoft.eShopOnContainers.Service.OrderingDb;User Id=sa;
    Password=<REDACTED>;Encrypt=False;TrustServerCertificate=true"
678           - name: webhooks-api
679             traits:
680               - type: env
681                 properties:
682                   env:
683                     ConnectionStrings__WebhooksDb: "Server=sqldata;
    Database=Microsoft.eShopOnContainers.Service.WebhooksDb;User Id=sa;
    Password=<REDACTED>;Encrypt=False;TrustServerCertificate=true"
684       - name: change-db-connection-string
685         type: override
686         properties:
687           components:
688             - name: identity-api|catalog-api|ordering-api|webhooks-api
689               traits:
690               - type: storage
691                 properties:
692                   secret:
693                     - name: sqldata-connection-details
694                       mountToEnvs:
695                         - envName: MSSQL_SERVER
696                           secretKey: endpoint
697                         - envName: MSSQL_USER
698                           secretKey: username
699                         - envName: MSSQL_PASSWORD
700                           secretKey: password
701           - name: identity-api
702             traits:
703               - type: env
704                 properties:
705                   env:
706                     ConnectionStrings__IdentityDb: "Server=$(
    MSSQL_SERVER);Database=microsoft-eshoponcontainers-services-
    identitydb;User Id=$(MSSQL_USER);Password=$(MSSQL_PASSWORD);Encrypt=
    True;TrustServerCertificate=true"
707           - name: catalog-api
708             traits:
709               - type: env
710                 properties:
711                   env:
712                     ConnectionStrings__CatalogDb: "Server=$(MSSQL_SERVER
    );Database=microsoft-eshoponcontainers-services-catalogdb;User Id=$(
    MSSQL_USER);Password=$(MSSQL_PASSWORD);Encrypt=True;
    TrustServerCertificate=true"
713           - name: ordering-api
714             traits:
715               - type: env
716                 properties:
717                   env:
718                     ConnectionStrings__OrderingDb: "Server=$(
    MSSQL_SERVER);Database=microsoft-eshoponcontainers-services-
    orderingdb;User Id=$(MSSQL_USER);Password=$(MSSQL_PASSWORD);Encrypt=
    True;TrustServerCertificate=true"
719           - name: webhooks-api
```

```
720             traits:
721               - type: env
722                 properties:
723                   env:
724                     ConnectionStrings__WebhooksDb: "Server=$(
    MSSQL_SERVER);Database=microsoft-eshoponcontainers-services-
    webhooksdb;User Id=$(MSSQL_USER);Password=$(MSSQL_PASSWORD);Encrypt=
    True;TrustServerCertificate=true"
725     - name: add-gateways-dev
726       type: override
727       properties:
728         components:
729           - name: webstatus
730             type: webservice
731             traits:
732               - type: gateway
733                 properties:
734                   domain: eshop-dev-webstatus.devscope.com
735                   http:
736                     "/": 80
737               - type: cloudflare-dns
738                 properties:
739                   publicIpAddress: "<REDACTED>"
740                   namespace: kubeflare-system
741     - name: add-gateways-prod
742       type: override
743       properties:
744         components:
745           - name: webstatus
746             type: webservice
747             traits:
748               - type: gateway
749                 properties:
750                   domain: eshop-prod-webstatus.devscope.com
751                   http:
752                     "/": 80
753               - type: cloudflare-dns
754                 properties:
755                   publicIpAddress: "<REDACTED>"
756                   namespace: kubeflare-system
757     - name: deploy-ha
758       type: override
759       properties:
760         components:
761           - type: webservice
762             traits:
763               - type: scaler
764                 properties:
765                   replicas: 2
766     - name: enable-logging-prod
767       type: container-log
768       properties:
769         namespace: eshop-prod
770         cluster: bonfim
771   workflow:
772     steps:
773       - name: deploy-to-dev
774         type: deploy
```

```
        properties:
          policies: ["add-gateways-dev", "add-db-connection-string", "target-dev"]
      - name: manual-approval
        type: suspend
      - name: deploy-database
        type: deploy
        properties:
          policies: ["change-database-type", "target-local-prod"]
      - name: manual-approval2
        type: suspend
      - name: read-db-secret
        type: read-object
        properties:
          apiVersion: v1
          kind: Secret
          name: sqldata-connection-details
          namespace: eshop-prod
          cluster: local
        outputs:
          - name: endpoint
            valueFrom: output.value.data["endpoint"]
          - name: username
            valueFrom: output.value.data["username"]
          - name: password
            valueFrom: output.value.data["password"]
      - name: apply-secret
        type: apply-object
        inputs:
          - from: endpoint
            parameterKey: value.data.endpoint
          - from: username
            parameterKey: value.data.username
          - from: password
            parameterKey: value.data.password
        properties:
          cluster: bonfim
          value:
            apiVersion: v1
            kind: Secret
            metadata:
              name: sqldata-connection-details
              namespace: eshop-prod
            data: {}
      - name: deploy-to-prod
        type: deploy
        properties:
          policies: ["add-gateways-prod", "change-db-connection-string", "enable-logging-prod", "disable-reload-config-files", "deploy-ha", "target-prod"]
```

Listing D.2: Application used for performance evaluation of the solution

```
import "strings"

"azure-linux-webapp": {
  alias: ""
```

```
  annotations: {}
  attributes: {
    workload: definition: {
      apiVersion: "web.azure.upbound.io/v1beta1"
      kind:       "LinuxWebApp"
    }
    status: {
      healthPolicy: #"""
      isHealth: (context.output.status.conditions[0].type == "Ready") &&
     (context.output.status.conditions[0].status == "True")
      """#
    }
  }
  description: ""
  labels: {}
  type: "component"
}

template: {
  parameter: {
    name: string
    location: *"NorthEurope" | string
    image: string
    appServicePlanName: *"" | string
    resourceGroupName: string
    resourceGroupLocation: *"NorthEurope" | string
  }

  appServicePlanName: "asp-\(context.appName)"

  output: {
    apiVersion: "web.azure.upbound.io/v1beta1"
    kind:       "LinuxWebApp"
    metadata: name: paramater.name
    spec: {
      forProvider: {
        resourceGroupName: parameter.resourceGroupName
        if parameter.appServicePlanName != _|_ {
          servicePlanIdRef: parameter.appServicePlanName
        }
        if parameter.appServicePlanName == _|_ {
          servicePlanIdRef: appServicePlanName
        }
        applicationStack: {
          dockerImage: strings.Split(parameter.image, ":")[0]
          dockerImageTag: strings.Split(parameter.image, ":")[1]
        }
      }
    }
  }

  outputs: {
    resourceGroup: {
      apiVersion: "azure.upbound.io/v1beta1"
      kind:       "ResourceGroup"
      metadata: name: parameter.resourceGroupName
      spec: forProvider: {
        location: parameter.resourceGroupLocation
```

```
        tags:
          "kubevela.app": "\(context.namespace)/\(context.appName)"
      }
    }

    if parameter.appServicePlanName != _|_ {
      servicePlan: {
        apiVersion: "web.azure.upbound.io/v1beta1"
        kind:       "AppServicePlan"
        metadata: name: appServicePlanName
        spec: forProvider: {
          kind: "Linux"
          resourceGroupName: parameter.resourceGroupName
          location: parameter.resourceGroupLocation
          sku: {
            capacity: 0
            size: "Y1"
            tier: "Dynamic"
          }
        }
      }
    }
  }
}
```

Listing D.3: Azure Web Application component definition