POLITECNICO DI TORINO

Master's Degree Course

In Computer Engineering

Master's Degree Thesis

Dynamic Application Placement in a Kubernetes Multi-cluster Environment

Supervisor:
prof. Fulvio Risso

Candidate:
Lorenzo Marino

Anno Accademico 2020/2021

Table of contents

ACKNOWLEDGEMENTS

I would like to thank my supervisor Professor Fulvio Risso and Davis Quirico for helping me to achieve this great goal during a difficult year for the whole world.

I would like to thank my family. My parents and my sister always supported me. It was a difficult journey. Probably without them I would have been lost. Specially, I would like to thank my grandparents. More than others were waiting for this achievement. I can tell them now, I did it.

Finally, I would like to thank my friends. New friends have come, others have gone but I thank everyone because they brought me here.

ABSTRACT

During the past years the way of deploying applications is changed. We started from deploy them on traditional servers, then move toward virtualized environments to ending in a more recent technology such as the containers.

In particular, microservices architecture suits very well in the containers. As they grown in popularity it came up a new necessity: a way to manage all of them. In fact, the enterprises were facing not just one or two containers but dozens or hundreds. The solution is Kubernetes.

Kubernetes is a platform for deploying and managing containerized applications, a container orchestrator. One of the main advantages of Kubernetes is that it offers a system that can be installed on different environments but the applications will run on any of them in the same way. For this reason, it is common for the enterprises run Kubernetes both on cloud or on-premise based on particular requirements.

This thesis aims to study and resolve a specific design challenge of an enterprise that adopt a multi-cluster configuration for Kubernetes: the dynamic application placement among different clusters. Two solutions are proposed and discussed, one based on KubeFed and one exploiting the basic mechanisms of Kubernetes.

1 Introduction

This introduction explains why Kubernetes is became a standard platform for the enterprises to develop their applications. In fact, with the evolution of developing applications, new challenges arise for the developers.

1.1 Distributed applications: microservices

Microservice architecture is a different approach in developing applications opposite to monolithic architecture. The monolithic approach has some limits: the application is deployed as a single big entity and this implies some difficulties during the developing and future updates. With microservices the application is split in its basic functions and each one of them is developed independently as single service. This service can communicate between them and produce the final result. Development, future updates or scaling the application are easier with microservices because the teams can focus only on the service of interest instead of the whole.

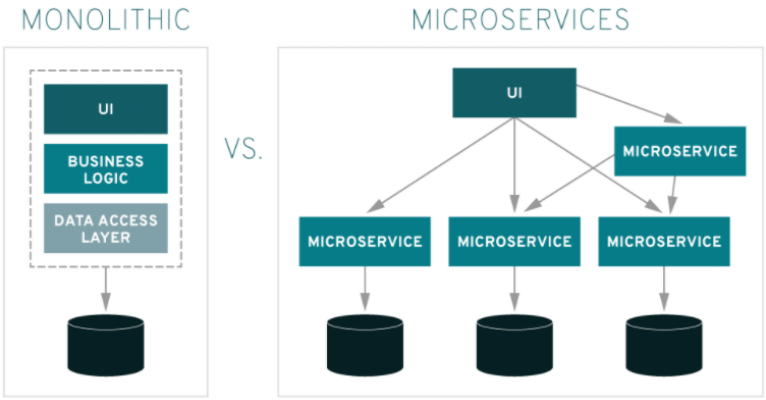


Figure 1: Monolithic vs Microservices architecture [1]

Changing the architecture also changes the deployment. If at first the applications were deployed on physical server then on virtual machine to reduce the costs, now the enterprises are moving towards the containers. Containers guarantee isolation similar to virtual machine, but they are lighter because the operating system is shared among applications. In the end a container is portable on different environment without any modification.

1.2 Why Kubernetes

The enterprises were facing hundreds or thousands of containers because they were developing containerized application. A new challenge arose, how to manage all of them. In 2014, Google presented to the world Kubernetes, an open-source platform for managing containerized workloads and services as a result of years of experience with a proprietary system called Borg. [2] [3]

Kubernetes provides some useful utilities. After the deployment of the application, the system is in charge of constant monitoring and self-healing the containers: in order to preserve the declared state of the application, if a container fails this is replaced with a new one. Scaling containers is very easy, it is a matter of change the number of desired replicas and this can also be automated to proper scale based on the amount of resource the service need. Kubernetes abstracts the hardware; it is decoupled from the underlying infrastructure so it can be installed on different platform like bare-metal or cloud, but the environment will be always consistent. Thank to that, now developing and production are more closer because they will face the same environment: an application can be tested locally or on a restricted area of the cluster(refers to namespace) and then moved to production on the desired platform without big difficulties.

In the end, being an open-source project means that Kubernetes is largely extensible and pluggable. All the native resources can be used to produce a specific final product or make use of one already developed by the community.

1.3 Same Kubernetes, different platforms

As mentioned in previous section, one of the main advantages of Kubernetes is that provides the same environment independently of the underlying platform. A Kubernetes cluster can be built on virtual machines, directly on machines or even

on some Raspberry PI [4]. This allows the enterprises to choose different configurations between on premise and cloud (or both).

1.3.1 On-premise

Organizations may decide to build a cluster locally in their data-center. This can happen due to some constraints or preferences. The downside of this setup it is that the cluster must be built from zero. It means that installing Kubernetes is not enough, but further action should be taken. A full operating cluster needs solutions for networking, storage, load balancing, monitoring, authentication. A team must install all these and more solutions before to use the cluster, an operation that requires some knowledges and it is not immediate. On-premise solutions are less scalable than cloud one because it requires to buy new hardware or install new virtual machine.

1.3.2 Cloud

All the installing process discussed before is not applied if organizations rely on cloud solutions. Nowadays many cloud providers provide their solutions of Kubernetes to the users. For example: Google GKE, Amazon EKS, Azure AKS and many more. After a subscription and few configurations, the cluster is already up. Because is a cloud solution an organization can decide among different type of machine configurations based on the necessities. After the purchase, the cluster is ready to use without further setups. Usually the Kubernetes version proposed it is not the latest or without alpha feature to provide a more stable environment. Cloud solution are very scalable because in a few seconds it possible to bring up or down a node.

1.3.3 Multi/Hybrid Clusters

Kubernetes provides an environment independent from the underlying platform and the same application will work on the same way no matter where it will be. Considering that, many enterprises choose a hybrid or multi cluster configuration. That is mean utilize more than one cluster and of different

types. For example, a cluster on premises and one or more on cloud even of different cloud providers. These choices depend on the necessities of an enterprise and lead to a new challenge: how to manage multiple cluster?

2 Multi-cluster architecture

Kubernetes is an independent system, and this gives to the enterprises different ways of building their own cluster or also multiple clusters. The strategy of managing multiple clusters as a unique entity is very common because leads to different benefits and it is not very hard to achieve. In facts there are lots of possibilities to build a cluster, no matter if the final location will be on premises data center, on cloud or on personal computer.

2.1 Why to choose multi-cluster?

There are several reasons why a company decides to build its business on multiple cluster:

- When a company has both development and production teams it can help the process to reach the final product. Production environment is quite sensible because no one wants a service that can go down after an update or as soon the new product is deployed. So, it should be pleasant to have a test environment more similar to the final one. With Kubernetes is possible. Test clusters can also be built locally with solutions like Minikube [5] or Kind [6] which gives the opportunity to build a minimal cluster on a local machine without adding extra cost for hardware or cloud resource consumption. In case are required more advanced clusters with more resources, which a personal computer can't offer, they can be built on premises or on some cloud provider to have in the end different clusters, some dedicated to the development and others to the production environment.
- Some businesses, to guarantee good performance in term of latency, need to be deployed across different regions. After to have built up different clusters in different regions the same application can be deployed in all of them and users will face similar experience independently of the location. The replicas work independently while a load balancer is in charge to redirect the requests of the clients to the closest cluster.

- To guarantee high availability of the services it can be used a similar approach of the previous one. Some providers offers the possibility to replicate the cluster in multiple regions so in case of some failures, like hardware failures or maintenance of the cloud facility, the workload is migrated to the backup cluster [7]. The clusters redundancy can be achieved by running clusters on different zones, different regions or even on different cloud providers. Again, a load balancing solution is needed to redirect the traffic in case of some cluster failure.
- Many enterprises already have a local data center that can be used for running a Kubernetes cluster to not waste a previous investment. A data center has limited resources and it is not convenient to add more of them only when needed. In this case, when the enterprise needs more resources just for a limited time, instead of upgrading the hardware it is more convenient start up a cluster on a cloud provider to satisfy the request in this short period. When the overload falls down, the instance of the cloud cluster can be shut down to reduce the cost. In facts cloud provider's solutions are not so cheap. This technique is called cloud bursting and let the companies to reduce the costs because extra resources are added only when needed without paying for it all the time not used.
- Every enterprise must follow some regulations or policies about their own data. Maybe for some enterprises is forbidden to have some applications with their sensitive information on cloud because it is a place not directly controlled by them. They are forced to have a local cluster and deploy these applications on premise data center. Anyway, if the enterprise does not want to renounce the cloud approach, it will manage multiple clusters in different environments and based on the workload, the applications can be executed on premise or on cloud. For some applications is possible that the workload changes in time so whenever in the workload change the type of data processed, the application originally running in the cloud can be transferred to the on-premise data center.
- To avoid the dependence by a single cloud provider an organization may runs different clusters on different cloud providers. Different providers can

offer different features because a provider has its own version of Kubernetes and functionalities. An enterprise can test a provider and then moving to another one without making any change to the application because the Kubernetes environment is independent from the platform. The cost for the cloud providers is different, so whenever a solution becomes more convenient than others, the applications can be migrated. Or in case of some cloud provider unavailability, having clusters on other providers increase the availability of the applications because can be migrated to the working clusters.

2.2 How to manage multi-cluster?

Reasons to adopt a multi cluster architecture have been exposed. The next step is to choose some solutions to operate on all the clusters.

The first choice is a method for managing the deployment of applications across the clusters. Firstly, let us remark this concept: instead of having a single big cluster with nodes across the regions or in different environments, the chosen strategy is to own different clusters. In this way, what the actual technologies are trying to do is to handle all the clusters as a whole entity while in practice they are split. I used the term technologies in plural form because there is not one unique solution to achieve the objective. Some solutions have in common the concept of "federation" as the entity composed by multiple parts, the clusters, that must be managed. Kubernetes itself is developing its own solution called KubeFed in order to provide a standard solution. This approach will be discussed in more details later because it is the one adopted in the thesis's project. During the waiting for a standard solution, some companies has developed their own by extending the functionalities of Kubernetes:

- For example, the online travel agency Booking.com has open-sourced its project Shipper [8]. Shipper realizes advanced rollout strategies in a multi-cluster environment.
- Google offers its proprietary solution called Anthos [9]. It offers a solution to avoid a vendor lock-in with a platform where deploy and run workloads across different providers.

Different approaches but the common objective of deploying applications across different clusters no matter if on bare metal or on different cloud providers.

Another choice can be how to implement the connectivity between the clusters. Some applications require a connection between the cluster to guarantee a distributed service across different regions in order to distribute the workflow or the database information for example. It can be adopted a service mesh solution like Istio [10] or for a layer 3 connectivity between cluster, Submariner [11] is an optimal solution.

3 Problem statement

After an introduction, in the previous chapters, on the evolution of the architecture of the recent applications and how the world of Kubernetes can offer a solid platform where to deploy them, I will expose the issue addressed by my thesis.

3.1 A real case study

During my thesis I had the opportunity to collaborate with an international company whose core business is civil and military aeronautics systems. Their applications are heavily based on microservices so in the last years, since Kubernetes became the new standard for deploying this kind of applications, they decide to integrate their already existing system with Kubernetes clusters with the future objective to migrate a big part of the production services there. After an initial period of employment, a new necessity came up for the company. To better understand this necessity, the clusters architecture should be explained.

3.2 Clusters architecture

The company chose a cluster architecture already described in the previous chapters known as multi cluster. In particular their configuration can be described as hybrid multi cluster because it groups both a cluster on-premise and more than one on cloud. The clusters are not an exact replica of each other, but everyone hosts their own applications. The kind of cloud provider or on-premise solution is not relevant for the purpose of the thesis because it does not affect the company's necessity and the relative solution, thanks to the fact of Kubernetes being independent from the underlying platform.

3.3 A design challenge

The adoption of this clusters design brought a new challenge to the developer and production teams. For some particular situations, their desire was the automation of moving a particular application from a cluster to another one. This feature is not present in Kubernetes out of the box for two reasons:

- Kubernetes is born with the idea of managing only one cluster. In particular, a cluster typically is composed by nodes and on the nodes are deployed the applications. In this way, with a basic installation of Kubernetes every cluster is a single entity by its own and the communication with another cluster is not immediate.
- The feature it can be considered a particular case. Being Kubernetes a relatively young project it is reasonable that this kind of function is not supported by default.

The situations in which this feature can be useful are different.

3.3.1 Data classification

Many companies have some policies or constraints about sensible data. Having this kind of data in a public environment like in the cloud of external provider may be forbidden. If an application does not process all the time sensible data and it is projected to running the most of the time on a public cloud, when these data enter in the workload, the application can be automatically moved on-premise in a private cluster.

3.3.2 Cost of cloud providers

Subscription and use of a cloud provider can have significant cost if the use is intensive. In fact, the price changes based on the number of nodes and configuration of the machine picked. Usually the price is hourly so to reduce cost some application can be moved on premises or on cloud with a more convenient price.

3.3.3 Latency of the applications

Based on the type of business, some applications can require a limited response time. Not all companies have the fortune to be located near the cloud provider data center. For example, at the time of this writing, Google Cloud does not have data center located in Italy but they are working on it. The better

solution is to automatically move these applications when this constraint is required.

Other situations can exist, and this depends on the type of business. Not all the enterprises have the same necessity but this particular feature, automate the moving of an application, can be useful for many.

3.4 The challenge to overcome

During the meetings with the company, with which I collaborated, I noticed that they were more concerned about one of the previous challenges. In fact, they are very restricted on the kind of data that can reside on public clouds. Some type of sensible data must reside within the perimeter of the facility. The only possible place for these data is their local data center where is installed a Kubernetes cluster on-premise. These data restriction is not always applied: the restriction is only on a part of all the data processed by the application. It means that the workflow of some applications does not change during the time but what changes is the kind of data processed. If the application resides on a public cloud, whenever sensible data have to be processed, the application must be moved on-premise. Their future project is to overcome this situation.

3.5 AS-IS

Currently the mechanics described before are all made by hand without automation. Among their services, in particular one is in charge to indicate what kind of data, if classified or not, every application is processing. When the service notifies a change in the data classification, the production team sets up and perform the moving. From this point, I will refer to this service as “classification service”. The “classification service” is aware of the applications deployed in all the clusters.

3.6 TO-BE

The project is to automate the process described in the AS-IS section. To not bring heavy changes to the “classification service” it will be developed a new application

and these two will communicate between them in order to decide which application has to be moved and where to move it. In the end, the "classification service" will handle monitoring and notification of the data classification while the new developed application will handle the practical moving.

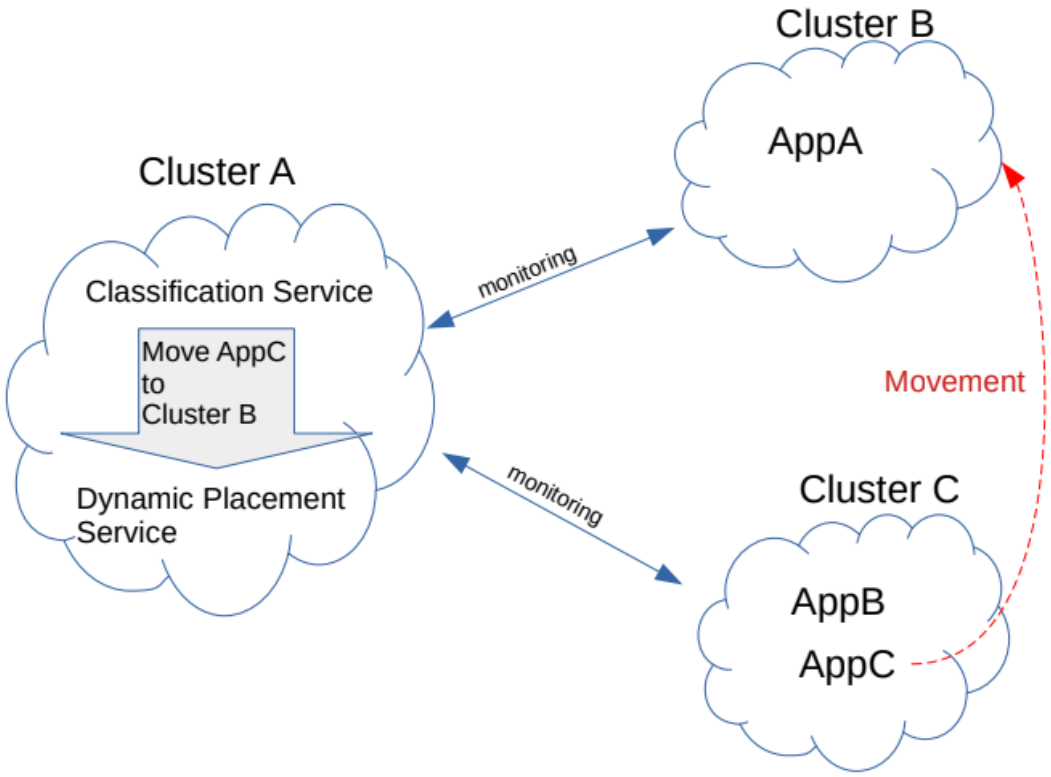


Figure 2 Summary of the challenge

After this illustration of the needs for the company to implement this function, I will expose my proposed solutions. Two solutions will be proposed and analysed. The solutions have to be intended as a prototype, a starting point for a future development. In fact, this is a feature that the company should like to bring in production. Being the production a very sensible environment, the feature should be fully tested and exploited before making it operative.

4 Technologies involved

Before exposing my proposed solutions, I would like to introduce the technologies involved in this project. This project is a combination of four main technologies. They are the followings:

- Kubernetes itself because is the environment of the study.
- Docker, a container runtime.
- Client-go, the official client to interact with Kubernetes and let the developers to build custom applications.
- KubeFed, a young project to coordinate and managing a federation of multiple Kubernetes cluster.

4.1 Kubernetes

Kubernetes [2] is an open-source system to orchestrate and manage containers, originally developed by Google in 2014 as the result of year of experience on a proprietary system called Borg. It provides mechanisms for deployment, maintenance, and scaling of applications. Actually, it is maintained by the Cloud Native Computing Foundation (CNCF). Kubernetes offers a perfect environment to deploy containerized applications and guarantees an elevated resilience for the apps. The need of a system like Kubernetes arose when applications, in particular distributed applications such as microservices, were growing more and more in the number of containers involved with the relative managing becoming too difficult. Notice that a company have to face not a dozen of containers but maybe hundreds or thousands. What Kubernetes can offer is:

- Deploying and scaling application across different host declaring a desired state to maintain.
- Service discovery and load balancing among the containers. A virtual network is built to connect all the containers across the cluster.
- Easy and controlled way to perform rollout or rollback of an application.

- Continue monitoring and self-healing of the containers based on health checks to maintain the desired state. If a container dies for some reason, a new one will be created to replace it.
- High level of security for sensible password or data and authentication/authorization for users and clients to perform actions on the cluster.

A Kubernetes cluster is composed by nodes. Nodes are categorized in "master" and "worker" node. The master node hosts the Kubernetes Control Plane to manage all the cluster. The worker node hosts the Pods in which runs the application workload.

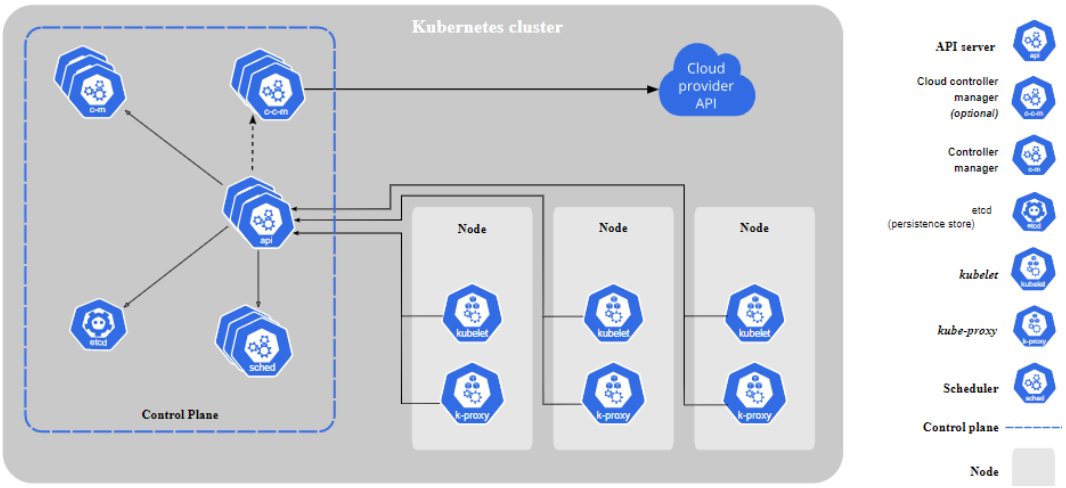


Figure 3A Kubernetes cluster view [12]

4.1.1 Control Plane

The Control Plane on the master node hosts vital components for the correct functioning of the cluster. For this reason, it is a good practice replicate the master node to have a more resilient system. The components are:

- Kubernetes API Server: the implementation is called kube-apiserver and expose the Kubernetes API. Kubernetes API is a REST API that lets users or components to query and manipulate the state of the objects in the cluster. It can be access by kubectl (the Kubernetes CLI), by client libraries or directly by REST requests. The REST operation permitted by an entity are defined in the corresponding role.

- Scheduler: kube-scheduler decides on which node deploy a new pod. The scheduling depends on resource requirements, pod or node constraints, affinity.

- Controller Manager: kube-control-manager runs different controllers. Each of this controller is responsible for a determinate object. For example, there are: node controller, replication controller, endpoints controller, service account & token controller.

- Cloud Controller Manager: it is optional because it is a controller manager specific for a cloud provider. It integrates the cloud provider API with the cluster

- Etcd: it is a reliable key value data store that stores all the cluster configuration. It is a sensible part and it is a good practice have a backup of it.

4.1.2 Node

On every node there are components that guarantee the correct functionality of pods:

- Kubelet: it manages the pods and the container inside the guarantee the pod specification defined in the PodSpecs.

- Kube-proxy: it is a network proxy the implements network functionalities on the nodes like service discovery or load balancing.

- Container runtime: it is the software for running the containers. For example, Docker.

4.1.3 Kubernetes Objects

The Kubernetes system is composed by entities called Kubernetes objects. Every object contains a desired state and a current state. The system continuously monitors and acts to keep these two states consistent. The desired state of an object is defined by a .yaml file that is send to the API

server through kubectl or can be sent directly to the API server with a client library for example. There are many objects, the follows are the most relevant for my solutions:

- Pod: a pod is the smallest deployable unit in Kubernetes. It groups containers and volumes belonging to the same workflow. A pod can run one or more containers. Usually it is not created directly but by other objects like Deployments or Jobs. Every pod has its own IP address and domain name while the containers inside share the same network namespace. The pod assures that an application is running in a unique environment and is not spread among the nodes.

- Deployment: the deployment is the abstraction of an application. In a deployment is defined a desired state of the application. It will own the pods and the ReplicaSet that creates the pods. It provides a simple way to update the application because through the deployment you can change number of replicas, roll out a new version of the software and eventually roll back if there are some errors.

- Service: a service exposes the applications as a network service. When a service is defined, all the pods of an application are linked to the service. This is possible thanks to a label selector. To the service is assigned a virtual IP, called Cluster IP, and a mapped port to access the application on the network. If multiple pods are defined for the single application, the traffic is load balanced between the pods. Every pod can discover any services in the cluster through DNS service discovery. DNS service is provided by a component such as CoreDNS.

- ConfigMap: a configMap is an object to store some configurations to be consumed by the pods. It can be passed to the pod as environment variable, command line argument or as file in a volume. The data in the configMap are stored as key-value pairs. This object lets to not hard-code the configuration in the application code so in case of some

configuration changes, the application must not be recompiled but just the configMap should be edited with the new configurations. The name must be a valid DNS name.

- Secret: a secret is an object similar to a configMap but for sensitive configurations with additional features for sensitive data like passwords, tokens or keys. The content of a secret is encoded in Base64. A secret can be used as environment variable or as a file mounted in a volume. The name must be a valid DNS name

- Volume: a volume is a directory mounted into a container of the pod. The volume contains some data and can be composed by different types of data. For example, a volume can contain the content of a configMap or of a secret. It can be used to claim a persistent storage so the data inside the container will live outside when the container will be shut down.

4.2 Docker

Docker is an open-source solution for packaging and deploying containerized applications. An application needs several dependencies in order to run correctly. In the container, the application is packaged with its environment formed by libraries and part of the files of the filesystem. This means that the application has everything it needs to run, and it is not necessary anymore to install a dedicated full operating system. This leads the container being lighter than the virtual machines affirming a new standard to deploy applications. A Docker container is based on a Linux container, it runs on a host with Docker installed and will be isolated from the host and other processes running on it.

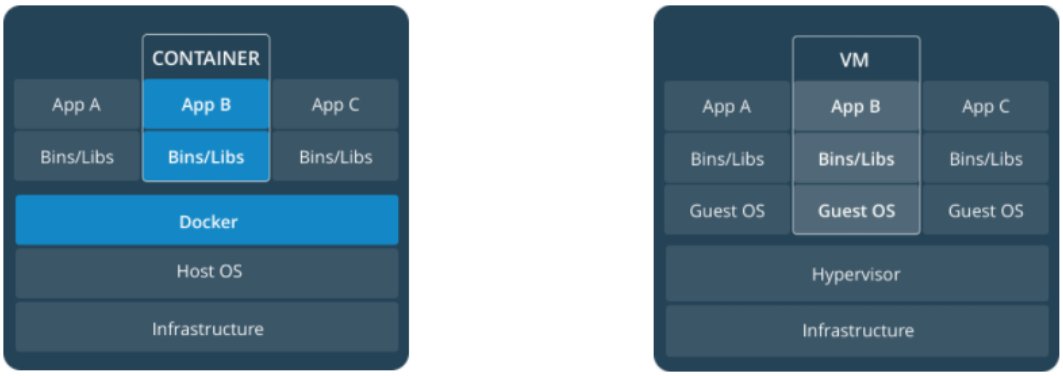


Figure 4Difference between Docker container and VMs [13]

The main components of Docker are:

- Docker Engine: it is a client-server application composed by three main part. A server called docker-deamon, a REST API used to talk with the docker-deamon, a client that represent the docker CLI. Through the cli and the API, the docker-deamon is controlled to create and manage the Docker objects.
- Image: it is a template for the application containing environment files and metadata needed for running the application. The image is a set of layers providing certain information. If changes are made, only the layer involved is update. There are basic images where you can start from to build your own image.
- Container: as described before it is a Linux container created from an image. It can be managed by the Docker API or the cli.
- Docker Registry: it is a repository for storing Docker images. Images are uploaded and downloaded with the cli commands push and pull. A repository can be public or private. Docker Hub is a public repository hosted by Docker and by default is looked for an image to be pulled or pushed.

Kubernetes needs a container runtime installed on every node in order to run the pods. Docker is a supported choice. Inside the pod specification it is defined the container image to download from a container registry.

4.3 Client-go

Client-go [14] is the official client library written in Go for interacting with the Kubernetes API and perform action on the cluster. It supports all official types of Kubernetes and also user defined types (Custom Resource Definition). The library is available on GitHub and the version is parallel to that of Kubernetes, so you need to check what version to use based on your cluster. When you build a client there are two types of authentications to perform in order to use the Kubernetes API:

- Out-of-cluster configuration: this method needs a kubeconfig file with the context information of the cluster to initialize the client. It is the method used by kubectl cli. The application built with this authentication method can run out of the cluster but still operating on it.
- In-cluster configuration: this method is for applications that run in a pod inside the cluster. When the pod is running, the kubelet creates a service account token at the path /var/run/secrets/kubernetes.io/serviceaccount that can be used for the authentication. To get the permission of operating on the cluster if RBAC is enabled, a cluster role must be created also.

After a configuration is obtained, a client set can be built to interact with the API.

All Kubernetes resources are represented as a struct in Go. All the object has some fields in common:

- TypeMeta: it describes the type of the object (kind) and the API schema version.
- ObjectMeta: it contains different metadata information of a resource like name, namespace, resource version (different from API version) and many more.
- Spec: it is the desired state of an object defined during the creation of the object
- Status: it is the actual state of the object compiled by Kubernetes controller

4.4 Kubernetes Cluster Federation

Kubernetes Cluster Federation [15] it is a SIG Multicluster project. It is also known as KubeFed or Federation V2 because it is a new version of the previous federation project. The objective is to introduce the concept of federation in a Kubernetes environment composed by multiple clusters and how to coordinate the deployment of applications on the desired cluster in the federation. KubeFed is based on Custom Resource Definition (CRD) that redefines all Kubernetes resource as federated resources. At the moment it is in alpha release with the aim to move toward beta in the future.

4.4.1 Architecture

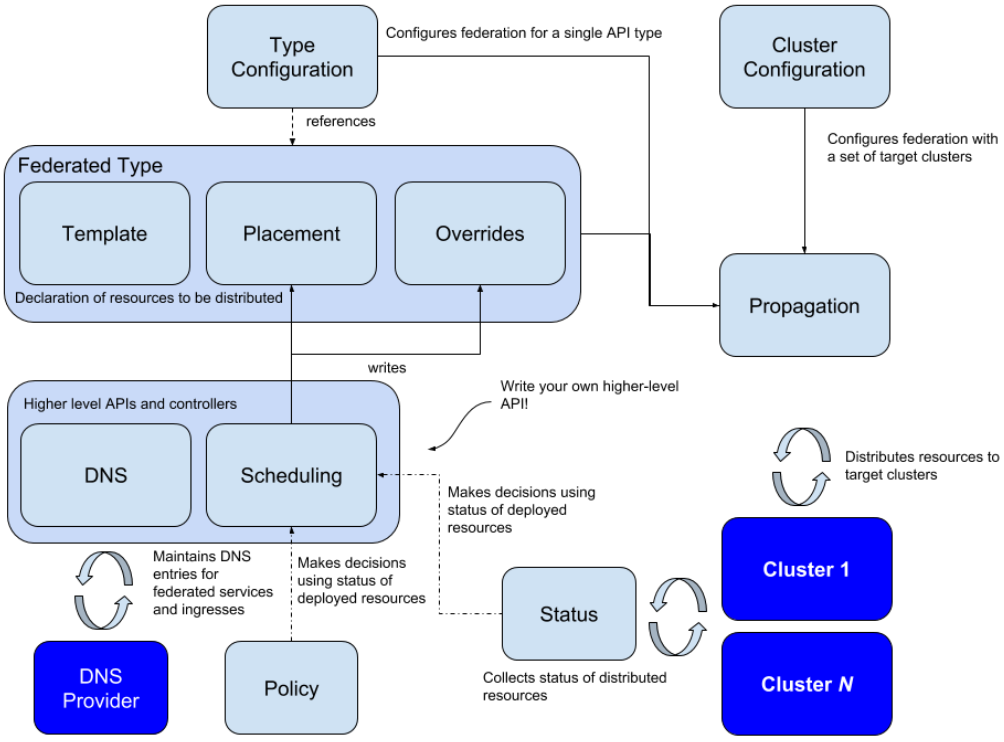


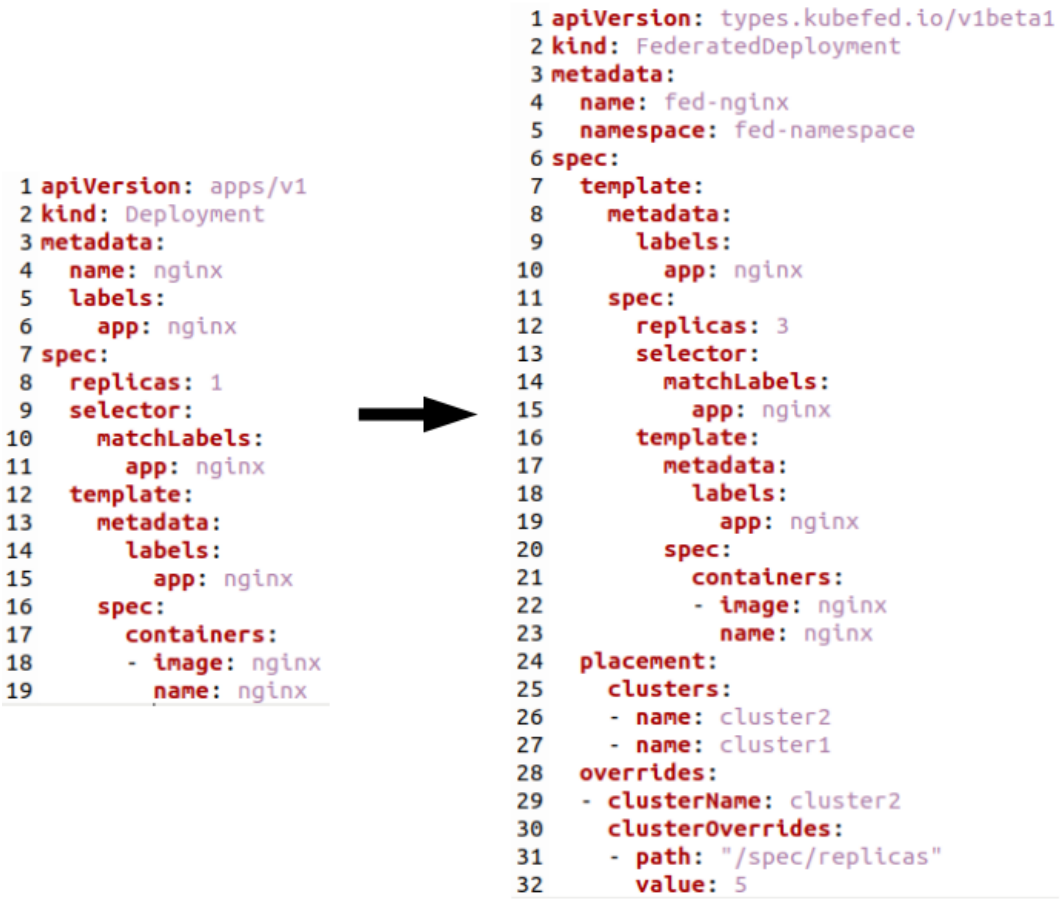
Figure 5 KubeFed architecture [15]

The picture shows the KubeFed control plane. This control plane is installed only on one cluster that will become the “Host cluster”. Two configurations are stored:

- Cluster configuration: it contains sets of configurations to target the clusters. When a cluster is registered to the federation it becomes a “Member cluster”. With the cluster configuration, the federated resources can be deployed on the desired clusters.
- Type configuration: it stores the Kubernetes API types (objects) that should be federated. The Federated Type is a CRD of a Kubernetes resource type (for example, FederatedNamespace is the CRD for Namespace) formed by three significant parts:
    - Template: it is a wrapper of the federated type containing the original spec definition.
    - Placement: it specifies in which federated cluster a federated resource should be deployed
    - Overrides: in this section can be defined some per-cluster parameter variations. For example, in a federatedDeployment specify a different number of replicas for a certain cluster.

A controller, called “PushReconciler”, is in charge to push the changes of the federated resources to all member clusters in order to have the desired state in all the clusters. It continually collects and re-sync the status of federated resources across the federation to keep a consistent desired state defined on the host cluster. For example, if a federated deployment is deleted on a member cluster, this will be recreated after a while by the PushReconciler.

The following example shows the relationship between a Deployment and a FederatedDeployment:



```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - image: nginx
        name: nginx
```

```yaml
apiVersion: types.kubefed.io/v1beta1
kind: FederatedDeployment
metadata:
  name: fed-nginx
  namespace: fed-namespace
spec:
  template:
    metadata:
      labels:
        app: nginx
    spec:
      replicas: 3
      selector:
        matchLabels:
          app: nginx
      template:
        metadata:
          labels:
            app: nginx
        spec:
          containers:
          - image: nginx
            name: nginx
  placement:
    clusters:
    - name: cluster2
    - name: cluster1
  overrides:
  - clusterName: cluster2
    clusterOverrides:
    - path: "/spec/replicas"
      value: 5
```

Figure 6 Differences from Deployment and FederatedDeployment

The metadata and spec sections of the deployment are put inside the template section of the federated resource. The placement lists the clusters where to deploy the resource. The overrides section changes the number of replicas on the cluster “cluster2”

4.4.2 Installation and use

As anticipated before, the KubeFed control plane must be installed on the Host cluster. Then the member clusters can be joined with the KubeFedctl cli. Now the federation is created. Every federated resource deployed will be also deployed on other clusters based on the Placement definition. Actually, the

control plane can be installed from Kubernetes v1.13 and is fully tested on these environments: kind, Minikube, Google GKE and IBM Cloud Private.

5 Proposed solutions

My desire is to propose two different solutions and make a comparison between them. Both the solutions aim to build an application in charge to moving other applications but with a different setup. One solution exploits the concepts of KubeFed, the other one the basic Kubernetes environment. The containers created for the applications are Docker containers written in Go using client-go, the official library to create a Kubernetes client in Go. Before to describe the solutions, I will show my testing setup and describe an application that offers a service in common with the two solutions.

5.1 Clusters setup

The challenge is to handle multiple Kubernetes clusters. I recreate this setup with:

- Minikube: I started two instances of Minikube locally by creating two different profiles. Every profile creates a dedicated VM with the opportunity to run two cluster independently. The result is two clusters with one node for each one, on my local pc.
- Google GKE: GKE is the Kubernetes service available on Google Cloud Platform. I created one cluster with two nodes.

As result, I have three different Kubernetes clusters that can be controlled by my local machine:

Figure 7 Different contexts in the kubeconfig file

The clusters “cluster1” and “cluster2” are the Minikube one, “google” that on Google Cloud Platform. The last one is not part of the thesis project.

5.2 Application classification

As discussed in the problem statement section, there can be different reasons why to move an application among the clusters. The main requirement for the company is moving based on the type of data processed by their services. They have a service in charge to list the type of data processed by every other application in all the clusters in real time. I will refer to this service as “classification service”. This service can list two different type of data: classified or not classified. The idea behind my solutions is to use this classification service as input for the service responsible of moving the applications. In fact, a constraint for the company is to must have and process classified data inside the perimeter of the company’s building. In other words, this kind of data must reside in their local data center, in their local Kubernetes cluster. For obvious reasons classified data cannot be located on other external cloud because the cloud is physically located outside the building.

The operating scenario is the following: the classification service registers all the running applications in the clusters and lists the type of data handled by them. Whenever it is reported a data of type “classified”, that application must be moved on the on-premise cluster. So, the classification service sends a notify to the application in charge to re-place the target.

The development of this classification service is not the main objective of my thesis, so I did not develop a full operational version of it. Instead what I did is to write an application that emulates in a generic way the operation of the classification service.

5.2.1 Business logic

The classification service must send to the dynamic placement application the information required to perform the replacement. This information depends on the parameters required by the client-go methods (REST methods) to operate on the Kubernetes resources and on the current/future cluster location.

- For the REST methods: we will go through the details of all the methods in the next sections where I will describe the dynamic application. For now to understand the mechanism we can just focus

on a simple GET request to obtain a resource from the cluster: the parameters needed for the request are the name of the resource and the namespace where the resource is located.

- For the cluster location: it depends on the type of data processed. This service at every notification alternates the type of data processed: if the data is set to “C”(classified) the target application will be moved to on-premise, if is set to “NC” (not classified) it will be moved to the cloud.

In the end, the information about the target application to send to the dynamic application are: current cluster location (source), future cluster location (destination), namespace and name (application). The parameters are loaded from the configuration files and stored in the following structures:

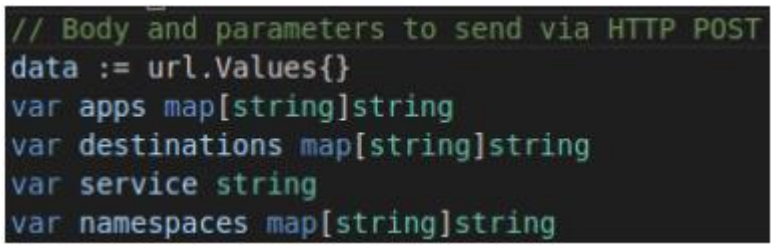


Figure 8 Parameters to send via HTTP POST

All these parameters are embedded in the variable “data” that will be the body of a HTTP POST form directed to the dynamic application. The source location is the last destination value related to the app (taken from “destinations” map) while the future location is the new opposite value. In the simulation this operation is looped every 3 minutes.

```
// Simulation
for {
    waitingTime := time.Duration(3 * time.Minute)
    for k, v := range apps {
        data.Set("namespace", namespaces[k])
        data.Set("application", k)
        data.Set("source", destinations[v])
        if v == "C" {
            apps[k] = "NC"
            data.Set("destination", destinations["NC"])
        } else {
            apps[k] = "C"
            data.Set("destination", destinations["C"])
        }
        resp, err := http.PostForm(service, data)
        if err != nil {
            panic(err)
        }

        body, err := ioutil.ReadAll(resp.Body)
        if err != nil {
            panic(err)
        }
        fmt.Println("sended:\n", string(body))
        time.Sleep(5 * time.Second)
    }
```

Figure 9 Core logic of the classification service simulation

5.2.2 Initial configurations

The classification service requires some initial configurations. These configurations provide the initial location of the applications by the relations between C/NC and on-premise/cloud, the namespace of the applications and the name of the dynamic placement service in order to send the http form. Instead of hard coding these configurations inside the application, a Kubernetes configuration resource has been used: a ConfigMap. The key-value pairs in the “data” section of a configMap can be consumed by a pod as files mapped in a mounted volume in the pod. The key will be used as file name, the value as file content. The configMap can be generated with a kubectl command:

kubectl create configmap appsclassification-config --from-file=config.

The parameter "config" is the name of the folder that contains all the configuration files. The configMap created can be checked with:

kubectl describe configmaps appsclassification-config:

Figure 10 Description of the created ConfigMap

The pod specification inside the deployment definition must be filled with a volumeMounts and volumes definition to mapping the configMap to the pod:

```
spec:
    containers:
    - name: appsclassification
      image: lorenzomarino94/appsclassification
      volumeMounts:
        - mountPath: /go/src/app/config
          name: config-volume
    volumes:
      - name: config-volume
        configMap:
         name: appsclassification-config
```

Figure 11 Volume mount for the ConfigMap inside pod definition

The volume is mounted on the start of the pod. Changes to the configMap after the pod is started have no effect. To make the updates have effect, the pod should be restarted with:

`kubectl rollout restart deployment deploymentName`.

5.2.3 Service discovery

The communication between the classification service and the dynamic placement application is possible due to the service discovery performed in Kubernetes. To expose an application running in a pod to other applications (or pods) within the cluster or to the external, it can be used a Service. A service creates a stable entry point for the target pods assigning a virtual IP called clusterIP. Then the pods can access the service through a DNS look up. Every pod is configured with a DNS server that can resolve the domain name of a service. The domain name has a specific format: for example, the domain name of a service called my-service in the namespace my-ns has the following format [16]: my-service.my-ns. In this way a pod can resolve the desired clusterIP for reach another pod.

In the configuration file service.conf it will be present the address with the domain name of the service associated to the dynamic placement application to let the communication with the classification service: for example, http://dynamic-moving-custom.default . The address in stored in the variable "service" and can be used in the function that performs the http POST and the DNS service will resolve it:

Figure 12 HTTP POST with information about the target app

This is how my version of the classification service works and triggers the dynamic placement application. In order to have a simple and quick

communication with the two apps I decided to run both apps in the same cluster.

5.3 KubeFed solution

To achieve a dynamic placement of the applications I decided to take advantage of KubeFed because it is an open-source project strictly related to Kubernetes, an extension of its functionalities. KubeFed out of the box provides a single point of management for all the clusters in the federation. This solves lots of problems like the monitoring and synchronization of the resources. The managed clusters must be joined to the host cluster where is present the KubeFed control plan and is exposed the KubeFed API. After the joining, whenever a federated resource is deployed on the host cluster, it will be deployed on the clusters defined in the "placement" section.

5.3.1 Placement section

A KubeFed resource definition introduces three new section: template, placement and override. The placement section lists the clusters in which the resource will be deployed. There are two ways to determinate where a federated resource will be propagated, the fields clusters or clusterSelector [17] under the placement section:

- Clusters: it is simply a list (array of elements in JSON) with the name of the federated clusters where the resource should be propagated. More are the cluster to list more verbose became the section. If the field is an empty list, the resource will be propagated to all member clusters.
- clusterSelector: this field offers an easier way to target a subset of the federated clusters. The clusterSelector is a label selector used by KubeFed to identify a set of clusters. It is similar to a nodeSelector but applied to entire clusters. If the field "clusters" is defined, the content of clusterSelector will be ignored. To be active, only the

clusterSelector should be defined. In this way the federated resource will be propagated only on clusters with the specific label.

My dynamic placement application takes advantage of this functionality to decide where to move the target application. When the classification service reports a change of the data classification, this app automatically modifies the clusterSelector section in order to move the target to right cluster. As soon as the change is applied, the PushReconciler takes care of re-sync the status of the federation an re-deploy the target application in the desired cluster (or clusters).

5.3.2 Federation configuration

Before using the dynamic placement application, the federation must be created. For the purpose of the thesis I skip the details about the installation of the KubeFed control plane. What is important to understand about, is that the cluster where is installed the control plane becomes the "host" cluster. After the installation, the clusters can be joined, a process called cluster registration [18]. The join is performed by the tool "kubefedctl", the KubeFed cli utility. The parameters for the join are: the cluster name to be joined, host cluster name, host and joined cluster context. Here, an example of the command:

`kubefedctl join cluster1 --cluster-context cluster1 --host-cluster-context cluster1.`

As you can notice, also the host cluster can be joined to be the destination of future resources. The contexts are taken by a single config file, the same used by kubectl. So before performing the join, a config file with all the desired contexts must be created.

For my setup I choose one of the Minikube clusters(cluster1) to be the host. After the installation of the control plane I joined all the clusters, including the host. The result is shown in the next figure:

```
lorenzo@lorenzo-MS-7C02:~$ kubectl -n kube-federation-system get kubefedclusters.core.kubefed.io --show-labels
NAME       AGE   READY   LABELS
cluster1   13d   True    location=onpremise
cluster2   13d   True    location=cloud
google     18h   True    location=cloud
lorenzo@lorenzo-MS-7C02:~$
```

Figure 13 Federation clusters

A READY status True means that the cluster is correctly joined with the host. I call the attention to the last section of the table, LABELS. This shows all the labels applied to the clusters and that will be used by the clusterSelector. In fact, to complete the configuration after the joins, I added a label to every cluster. In Kubernetes, a label is a key/value pair to organize a subset of object. The goal is to attach a label to the cluster based on its location. The key is indeed "location", while the value can be:

- Onpremise: it stands for a cluster that resides on-premise, on the local datacenter of the company.
- Cloud: it stands for a cluster that resides on some public cloud provider; in my case it was on Google cloud.

Notice that for testing purposes I labelled cluster2, a Minikube cluster, as cloud even if it is physically on my local pc. The AGE section, indeed, shows a different age between "cluster2" and "google" because I started the tests only on Minikube clusters. The clusterSelector section of a federated resource will be like:

```
spec:
  placement:
    clusterSelector:
      matchLabels:
        location: cloud
```

The use of labels by my application will be described in detail in the next section where I show the business logic of the application.

5.3.3 Business logic

The application, deployed in the host cluster, runs waiting for a target application to move. In both my solutions, a target application is meant as

composed by two Kubernetes objects: a deployment and a service. These two objects will be moved to another cluster. The deployment handles the pods where the application logic is running meaning that if the deployment is moved to another cluster, the whole application logic is moved away. I also considered a service attached to the application because many applications expose some kinds of service inside or outside the cluster. As result, when a notify from the classification service arrives, the related deployment and service object will be transferred to the future destination. The following sections describe the main parts of the developed KubeFed solution.

5.3.3.1 Notification system

This application and the classification service communicate with HTTP requests. To be reachable, a service must be created to expose the pod of this application. In this way, the classification service can reach the dynamic application thanks to the service discovery mechanism present in Kubernetes.

```
apiVersion: v1
kind: Service
metadata:
    name: dynamic-moving
    labels:
      app: dynamic-moving
spec:
  type: NodePort
  selector:
    app: dynamic-moving
  ports:
  - port: 80
    targetPort: 8080
```

Figure 14 Service of the dynamic placement application

In particular, the classification service sends an HTTP POST form with all the parameters needed for the dynamic placement. These parameters are: namespace of the target, name of the target and the future destination. These are grouped in a struct for easy access:

```
type Values struct {
    namespace string
    application string
    destination string
}
```

Figure 15 Struct to store the parameters received

```
if req.Method == "POST" {
    v.namespace = req.FormValue("namespace")
    v.application = req.FormValue("application")
    v.destination = req.FormValue("destination")
```

Figure 16 Parameters taken from the HTTP POST

The current position of the target is not needed because all the changes are done on the federated definition present in the host cluster, so always in the same place. The namespace of the target is a federatedNamespace: in this way, federated resources are kept organized in dedicated namespaces.

5.3.3.2 Possible destinations

The destination depends on the type of data processed by the target app. The classification service can notify two kind of destination: onpremise for classified data or cloud for not classified one. The current location of a federated resource can be changed by modifying the clusterSelector field in order to match the desired label of a particular cluster. If the destination is "onpremise" the resource will be moved to "cluster1", if "cloud" will be moved to "cluster2" or "google". The clusterSelector is changed with a HTTP method PATCH on the KubeFed API. The PUT method is not necessary because just a field must be changed so it is not required the full object. The content of the patch is in json format:

```
{"spec":{"placement":{"clusterSelector":{"matchLabels":
{"location":"cloud"}}}}}
```

The possible patch values are fixed and stored in a map. With the received destination, the correct patch can be chosen:

```
destination = map[string][]byte{
    "all":     []byte(`{"spec":{"placement":{"clusterSelector":{"matchLabels":null}}}}`),
    "cloud":   []byte(`{"spec":{"placement":{"clusterSelector":{"matchLabels":{"location":"cloud"}}}}}`),
    "premise": []byte(`{"spec":{"placement":{"clusterSelector":{"matchLabels":{"location":"onpremise"}}}}}`),
}
```

Figure 17 Map with the possible patches

5.3.3.3 Interact with the Kubernetes API server

KubeFed resources are custom resources. To interact with the API server, I used a dynamic client provided by client-go. Firstly, it requires the cluster config to be authorized and authenticated by the API and have access to the resources. The federated resources are on the host cluster so this app must run in a pod in the host cluster. For pod running in the cluster, the credentials of the pod can be used for the authentication. The credentials can be get with the method rest.InClusterConfig(). Now the dynamic client can be created but requires an extra configuration respect to a typed client offered by client-go that will be used in the next custom solution. The dynamic client uses a generic type unstructured.Unstructured to recreate the objects from the API server. It needs the schema of the resource that must be handled so I provide the schema for both the federated deployment and service:

```
fedDepGVR = schema.GroupVersionResource{
    Group:    "types.kubefed.io",
    Version:  "v1beta1",
    Resource: "federateddeployments",
}
fedSvcGVR = schema.GroupVersionResource{
    Group:    "types.kubefed.io",
    Version:  "v1beta1",
    Resource: "federatedservices",
}
```

Figure 18 Schema of the resources for the dynamic client

The group, version and resource can be retrieved from a **`kubectl get`** on the relative resource.

The app is ready to perform the dynamic placement. Whenever a notify arrives, it builds the client and then applies the patch, one to the deployment and one to the service. The federated resources are updated, and the target application is automatically moved from the previous location to the new one:

```
newDestination := destination[v.destination]

_, err = dynClient.Resource(fedDepGVR).Namespace(namespace).Patch(context.TODO(), v.application,
            types.MergePatchType, newDestination, metav1.PatchOptions{})
_, err = dynClient.Resource(fedSvcGVR).Namespace(namespace).Patch(context.TODO(), v.application,
            types.MergePatchType, newDestination, metav1.PatchOptions{})
```

Figure 19 Patch methods on the resources

The dynamic application (as every other developed application) requires the authorizations to perform actions on the resources. By default, only view permissions are granted. Other permissions can be grant with the creation of a clusterRoleBinding: a clusterRoleBinding binds a clusterRole, that has specific rules of operation, to a particular subject: by default, the subject of a developed app is the service account “default”. Being in a testing environment, I create a clusterRole with permission of all verbs(actions) on all the federated resources (apiGroup=types.kubefed.io).

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: my-kubefed-role
rules:
- apiGroups: ["types.kubefed.io"]
  resources: ["*"]
  verbs: ["*"]

---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: default-my-kubefed-role
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: my-kubefed-role
subjects:
  - kind: ServiceAccount
    name: default
    namespace: default
```

Figure 20 ClusterRole and ClusterRoleBinding declaration

In a production environment the verbs should be limited only to the minimum required.

To verify the result of the dynamic placement, it can be checked the status section of the federated resources (updated with the new cluster location) or perform a **`kubectl get`** with both the contexts of the old and new cluster:

```
kubectl -n my-federation get all --context=OldClusterContext

kubectl -n my-federation get all --context=NewClusterContext
```

5.4 Custom solution

During the research for a possible solution to the challenge showed to me by the company I decided to implement another possible solution to the problem. The reasons that lead me to this decision are different. I will list these reasons and other considerations in the last main chapter “Conclusions”.

I refer to this current solution as custom because it is based on the implementation of an application that performs the actions required without extending the functionality of Kubernetes or without installing additional feature but it just exploits the basic functions of Kubernetes. There is not a federation that organize and manage all the clusters. The clusters and data processed by the applications are always monitored by the classification service as the KubeFed solution, indeed the notification system is very similar. What really changes is the way how the target application is dynamically moved. No labels or cluster selectors are involved but the resources of the target application are taken from the current cluster, modified and then pushed to the new location.

5.4.1 Business logic

Also in this solution an application is represented by a deployment and a service to expose the pod. The current dynamic placement application requires several configuration files to let the interaction with the different clusters. The following sections describe the main parts of the developed solution.

5.4.1.1 Notifications system

The interaction between the dynamic placement application and the classification service is the same as the KubeFed version. The only difference is the need for a fourth parameter, the current location of the target application. This parameter is required because there is not a unique representation of the resource among all the clusters, but the target must be taken directly from the initial location and moved to the final location. The structure to store the received parameters is the following:

```
type Values struct {
    namespace string
    application string
    source      string
    destination string
}
```

Figure 21 Structure to store the content of HTTP POST

To be reachable by the classification service, the dynamic application must be exposed creating a service resource:

```
apiVersion: v1
kind: Service
metadata:
    name: dynamic-moving-custom
    labels:
      app: dynamic-moving-custom
spec:
  type: NodePort
  selector:
    app: dynamic-moving-custom
  ports:
  - port: 80
    targetPort: 8080
```

Figure 22 Content of the service .yaml file

The name of the service must be set in the configuration file "service.conf" of the classification service. In this way the classification service can reach the dynamic application using the service discovery feature of Kubernetes.

5.4.1.2 Interaction with different Kubernetes API servers

In the KubeFed solution the application interacts only with the KubeFed API because the control plane stores a unique representation of the resources distributed among the clusters. Without a federation, every resource must be handled independently. It means that if the target application resides in a generic cluster A and has to be moved to a generic cluster B, the dynamic application must interact first with cluster A API server and then with cluster B API server. To interact with the API servers, you need a client and for this

solution it is used the typed client set provided by client-go. The difference with the dynamic client set used in the previous solution is that instead of using generic data structure to represent the resources, typed clients use specific Golang types for every kind. The types to be represented must be known at compile time. Typed clients are simpler to use but less flexible because they strictly depend on the version and types used.

To build the client set you need a configuration with the information and credentials to be authenticated and authorized by the API server. These information are the same used by kubectl to interact with the cluster and are stored in a so called kubeconfig file. To permit the communication, the dynamic application needs the kubeconfig files of all the clusters where the target apps can be moved. The kubeconfig file of a cluster is passed to the method **`clientcmd.BuildConfigFromFlags`** to build the configuration. Then with the configuration can be created the client set with the method **`kubernetes.NewForConfig`**. I create a function to perform these steps every time you need a client set:

```
// Wrapper function to build a client from a kubeconfig file
func buildClientSet(configFile string) (*kubernetes.Clientset, error) {
    // kubeconfig files are under ./config
    kubeconfig := filepath.Join("config", configFile)
    config, err := clientcmd.BuildConfigFromFlags("", kubeconfig)
    if err != nil {
        fmt.Println(err.Error())
        return nil, nil
    }
    fmt.Println("Cluster master: ", config.Host)
    return kubernetes.NewForConfig(config)
}
```

Figure 23 Wrapper function to build a clientset

The function **`buildClientSet(configFile string)`** receives as parameter the name of the target cluster and this name is used to build the full name of the kubeconfig file in order to open it and get the configuration data to build the client set.

All the kubeconfig files and possible locations are passed to the application through objects provided by Kubernets to pass configuration parameters to

the pods. The objects used are a Secret and a ConfigMap. The kubeconfig files contains sensible data like certificates, keys and cluster servers addresses. To store these configurations, I used a Secret. Then the secret is passed to the pod as a volume so the application can access to the kubeconfig files.

```
lorenzo@lorenzo-MS-7C02:~$ kubectl --context=cluster1 describe secrets dynamic-custom-secret
Name:         dynamic-custom-secret
Namespace:    default
Labels:       <none>
Annotations:  <none>

Type:  Opaque

Data
====
config-gke:       1998 bytes
gsa-key.json:     2316 bytes
config-cluster1:  5481 bytes
config-cluster2:  5481 bytes
lorenzo@lorenzo-MS-7C02:~$
```

Figure 24 Secret to store the kubeconfig files

The gsa-key.json contains information to be authenticated on GKE cluster. More details are discussed on chapter 5.4.4.

The possible locations instead, indicated by the cluster name, are written down in a file .json and this file is used to create a configMap.

```
lorenzo@lorenzo-MS-7C02:~$ kubectl --context=cluster1 describe configmaps dynamic-moving-custom-config
Name:         dynamic-moving-custom-config
Namespace:    default
Labels:       <none>
Annotations:  <none>

Data
====
clusters.json:
----
{
    "cloud":    ["gke", "cluster2"],
  "premise":  ["cluster1"]
}
Events:  <none>
lorenzo@lorenzo-MS-7C02:~$
```

Figure 25 ConfigMap to store the possible locations

Then both the secret containing the kubeconfig files and the configMap are mounted in a volume inside the container of the pod. To mount the volume into the pod I used a particular volume instead of a volume of type configMap or secret directly. The volume type used is called projected. This type of volume is used to map different volume type sources under the same

directory. Not all the volume types can be used but configMaps and secrets can.

```
16      spec:
17        containers:
18        - name: dynamic-moving-custom
19          image: lorenzomarino94/moving-apps-custom
20          ports:
21          - containerPort: 8080
22          volumeMounts:
23            - name: config
24              mountPath: /go/src/app/config
25          env:
26            - name: GOOGLE_APPLICATION_CREDENTIALS
27              value: "/go/src/app/config/gsa-key.json"
28        volumes:
29          - name: config
30            projected:
31              sources:
32              - secret:
33                  name: dynamic-custom-secret
34              - configMap:
35                  name: dynamic-moving-custom-config
36
```

Figure 26 Spec section of the pod definition with the volume mount

With the volumes mounted, the files can be used by the application. The name of the kubeconfig files must be in the following format: config-clusterName. In this way the correct file can be easily picked by the application through the map **`clusters map[string][]string`**:

```
{
    "cloud":    ["gke", "cluster2"],
    "premise":  ["cluster1"]
}
```

Figure 27 Configuration file .json with the clusters location

```
clusters map[string][]string
```

Figure 28 Map to store the file .json

The above structure lists the possible locations. When the dynamic application receives the notify by the classification service, the source and destination parameters (cloud or premise) are used as key to access the clusters map and get name of the source/destination clusters. In my setup, the key “premise” has the value “cluster1” and the key “cloud” have the values “cluster2” and “gke”. The correct value is then passed to the function **`buildClientSet(configFile string)`** to get the kubeconfig file and build the client set. This operation is performed before moving the target app to remove it from the current location, and after to move the target to the new location:

```
//Source clusters: remove the application from the clusters
fmt.Println("Removing resources from clusters...")
for _, s := range clusters[v.source] {
    sourceClient, err := buildClientSet("config-" + s)
```

Figure 29 Clientset to interact with the source clusters

```
//Destination clusters: move the application to new clusters
fmt.Println("Moving resources to destination clusters...")
for _, s := range clusters[v.destination] {
    destClient, err := buildClientSet("config-" + s)
```

Figure 30 Clientset to interact with the destination clusters

5.4.2 Moving the resources

After getting the access to the cluster API is possible to move the application. In this solution will be moved the deployment and the service associated to the target application. the process is split in different steps:

- **Get**: A resource is taken from the current location and stored in an empty object of the same type to obtain all the resource’s configuration. The parameters received by the classification service, namespace and name, are used to get the target resource:

```
deployment := &appsv1.Deployment{}
service := &corev1.Service{}
```

Figure 31 Objects to store the resources

```
//get deployment and service from source cluster
deployment, err = sourceClient.AppsV1().Deployments(v.namespace).Get(context.TODO(),
                  v.application, metav1.GetOptions{})
service, err = sourceClient.CoreV1().Services(v.namespace).Get(context.TODO(),
                  v.application, metav1.GetOptions{})
```

Figure 32 GET method on the resources

- **Delete**: with the resource saved is possible to delete it from the cluster because it will be moved:

```
//delete deployment and service from source
deletePolicy := metav1.DeletePropagationForeground
err = sourceClient.AppsV1().Deployments(v.namespace).Delete(context.TODO(), deployment.Name,
          metav1.DeleteOptions{PropagationPolicy: &deletePolicy})
err = sourceClient.CoreV1().Services(v.namespace).Delete(context.TODO(), service.Name,
          metav1.DeleteOptions{PropagationPolicy: &deletePolicy})
```

Figure 33 DELETE method on the resources

- **Modify**: this step is the most crucial because without some modifications, the stored resource cannot be moved to a different cluster. When a resource is created on the cluster, some unique fields are created. These field are strictly related to the cluster and to the time of creation. Before moving, the application deletes some fields from the ObjectMeta section and the entire Status section. The ObjectMeta contains all the metadata of the resource like name, namespace; the Status contains the result of the desired state of the resource (specified in the Spec section). From ObjectMeta are removed the following fields:
    - ResourceVersion: it is a string to identify the version of an object. The value of the string comes from the etcd and it is incremented every time the object is modified.

- SelfLink: it is a URL that represent a resource
- UID: it is a unique string generated by Kubernetes to identify an object in the cluster

In addition, for the service object must be deleted some fields present in the Spec section:

- ClusterIP: it is a cluster's internal IP address and it is automatically assigned during the creation of the service if not statically specified in the service's definition. By removing it we assure that will not be collisions with already existing services in the cluster or incompatibility with the CIDR range inside the cluster. In fact, deploying a Service with an invalid ClusterIP returns a 422 HTTP Status from the API server. The CIDR range depends on the CNI provider installed in the cluster. Anyway, if all the clusters have the same CIDR range, it is possible not modify the service ClusterIP if it has to be statically provided but it must be assured its uniqueness into the cluster.
- NodePort: it is not referred to the service type but is a consequence of it. When defining a service of type NodePort, the service is exposed on each node's IP at a static port called nodePort. This happens also for the type LoadBalancer because it is an extension of the type NodePort. If not statically specified, the control plane allocates a value for the nodePort in the range 30000-32767 by default. After some tests I found out an issue with the nodePort value, so I decide to remove the value before the placement. This issue is described in detail in the next chapter 5.4.3. The nodePort value is of type integer so to remove it, the nodePort is set to 0, the zero value in Golang for numeric types. This reset is performed in a for loop because the ports information are into an array of type SerivcePort so

in case are defined multiple port element for the service, all the nodePort values are reset to 0.

```
//Modify resources before moving
deployment.ResourceVersion = ""
deployment.SelfLink = ""
deployment.UID = ""
deployment.Status = appsv1.DeploymentStatus{}

service.Spec.ClusterIP = ""
for k := range service.Spec.Ports {
    service.Spec.Ports[k].NodePort = 0
}
service.ResourceVersion = ""
service.SelfLink = ""
service.UID = ""
service.Status = corev1.ServiceStatus{}
```

Figure 34 Fields modified before moving

- **Moving**: Finally, after the previous modifications, it is possible to move the application to the future cluster. The destination is acquired by the classification service and it is used to build the client set to interact with the future cluster API. The previous modified objects will be used in the method Create to recreate the target application in the new cluster:

```
movedDep, err := destClient.AppsV1().Deployments(v.namespace).Create(context.TODO(), deployment,
                    metav1.CreateOptions{})
movedSvc, err := destClient.CoreV1().Services(v.namespace).Create(context.TODO(), service,
                    metav1.CreateOptions{})
```

Figure 35 CREATE method on the new cluster

With the described process is possible to perform a dynamic placement of the applications under particular conditions. The correct placement can be checked using the **`kubectl get`** command on all the objects moved with the context of the involved clusters.

5.4.3 NodePort port issue report

During some tests of my solution I found an issue during the placement of a service on the GKE cluster. The problem occurs when the service is placed in the GKE cluster, moved to another cluster and then moved back to GKE cluster. In this last step the service is not placed because the API server returns the following error:

```
Service "nginx" is invalid: spec.ports[0].nodePort: Invalid value: 31254: provided port is already allocated
```

Figure 36 Error during service port allocation

Originally, I did not remove the nodePort value associated to the service, so the service had the same value on every cluster where it was moved in. When the service is brought back to GKE cluster, this error occurs even if the resource is not present in the cluster because previously deleted. The deletion of the previous version can be confirmed by a kubectl get service. This problem, in my setup, occurs only on GKE cluster when performing the placement of the same service in a short period of time since its deletion. This issue is being reported by several Kubernetes users in the past years but never officially fixed. The issue seems to be related to a missing synchronization between the local map of ports allocated and the map present in the etcd [19]. The deletion of the service correctly deletes the resource from the etcd (confirmed by the get on the API server) but the information about the port allocated is not immediately updated. In fact, after repeating the deployment of the service for a period of time (in my case it was about 8-10 minutes), the deployment is successful because the two maps have been synchronized.

The port allocator, during the checking if the port is already allocated, firstly checks the local map (in local memory) instead of the etcd. So, if the two are not updated, the allocation fails. The bug should be resolved in future releases of Kubernetes by checking first the etcd. I noticed this bug by myself because in my simulations the moving is performed every 3 minutes, not enough for the GKE cluster to update the information. Discussing with the company, we confirm that moving services in this short time and with a fixed nodePort

value is a very particular use case and highly depends on the type of the application. For example, they should move back the applications to the original location after a period of about an hour so the synchronization problem should not occur. Furthermore, currently they do not use services with static nodePort values so resetting the value and let Kubernetes to assign a new one is not an issue for their environment.

5.4.4 Service account for GKE cluster

The access to the GKE cluster requires an extra configuration. When you build a cluster on the Google Cloud Platform is possible to control it from the local command line with **kubectl** by getting the credentials from the cloud. Once the gcloud SDK is installed locally, with the command:

```
gcloud container clusters get-credentials clusterName
 --zone clusterZone --project projectName
```

a kubeconfig file with the credentials to access the cluster is created. Kubectl will use this kubeconfig file to interact with the API server. The obstacle in my solution is that this kubeconfig cannot be used by my application because part of the authentication process is handled by the local gcloud tool and in the cluster where the dynamic placement application is running, gcloud SDK is not present. The solution is to create a IAM policy [20]. The IAM defines which member has access (based on a role) to a resource. In this case, the member will be a service account related to my application, with the role of Kubernetes engine developer. Then get a key for the service account just created:

```
gcloud iam service-accounts create dynamic-moving

gcloud projects add-iam-policy-binding tesi-282514
 --member=serviceAccount:dynamic-moving@tesi-
282514.iam.gserviceaccount.com
 --role=roles/container.developer

gcloud iam service-accounts keys create gsa-key.json --
iam-account=dynamic-moving@tesi-
282514.iam.gserviceaccount.com
```

The final step is to build a kubeconfig file for the GKE cluster, where the user is the service account. This service account will use the generated key for the authentication on the cluster. This key can be passed to the pod of the application by mounting a volume with the content of key inside and then an environment variable of the container must be set with the path to reach the file containing the key. The gsa-key.json, that contains the key, has been stored in a secret, alongside the kubeconfig files, because contains sensible data.

```
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: [redacted]
    server: https://[redacted]
  name: gke_tesi-282514_europe-west1-b_thesis-cluster
contexts:
- context:
    cluster: gke_tesi-282514_europe-west1-b_thesis-cluster
    user: dynamic-moving-gsa
  name: gke_tesi-282514_europe-west1-b_thesis-cluster-dynamic
current-context: gke_tesi-282514_europe-west1-b_thesis-cluster-dynamic
kind: Config
preferences: {}
users:
- name: dynamic-moving-gsa
  user:
    auth-provider:
      name: gcp
```

Figure 37 Kubeconfig file of the Google GKE cluster

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: dynamic-moving-custom
  labels:
    app: dynamic-moving-custom
spec:
  selector:
    matchLabels:
      app: dynamic-moving-custom
  replicas: 1
  template:
    metadata:
      labels:
        app: dynamic-moving-custom
    spec:
      containers:
      - name: dynamic-moving-custom
        image: lorenzomarino94/moving-apps-custom
        ports:
        - containerPort: 8080
        volumeMounts:
          - name: config
            mountPath: /go/src/app/config
        env:
          - name: GOOGLE_APPLICATION_CREDENTIALS
            value: "/go/src/app/config/gsa-key.json"
      volumes:
        - name: config
          projected:
            sources:
            - secret:
                name: dynamic-custom-secret
            - configMap:
                name: dynamic-moving-custom-config
```

Figure 38 Deployment declaration with environment variable for the authentication

With this setup the client will be authenticated through the service account and will perform the actions on the resources.

6 Validation

In the previous chapter I described the proposed solutions to resolve the real case challenge presented by the company. In this section I will walk through a simulation for testing both the solutions and demonstrate the working concepts behind.

6.1 Simulation setup

The simulation environment aims to emulate a real production scenario with the adoption of a multi-cluster setup. The multi-cluster setup has been described in the 5.1 Clusters setup section. To quickly remind, the clusters involved are three, two deployed with Minikube (kubectl contexts **cluster1** and **cluster2**) and one on Google Cloud Platform (kubectl context **google**). To reproduce a real possible multi-cluster architecture, I have split the clusters in two locations: on-premise and on cloud. Cluster1 represents an on-premise cluster while cluster2 (logically) and google two cloud clusters. All the three clusters must be intended as under the same organization control. I created two different namespaces on every cluster where deploy the applications involved in the test. The KubeFed solution use the "my-federation" namespace while the Custom solution use the namespace "test-dynamic". The application classification service and the dynamic placement services are deployed only on cluster1 in the default namespace instead, to underline the fact that they do not have to be necessarily in the same cluster and namespace of the target applications.

In the simulation two applications will be dynamically moved between the clusters. These applications are: nginx, a basic Nginx web server [21] and hostname, a minimal web server developed by myself. The last responds to an HTTP GET with the name of the pod and node where is currently running. In this way I can have a practical demonstration on where the application is located.

By querying the three different Kubernetes API of all the clusters and the hostname application I can check the effects of the dynamic placement performed by the developed solutions.

The simulation is repeated for both the solutions with different initial configurations to match the required parameters for the apps involved.

6.2 Testing the KubeFed solution

In this solution, the two target applications must be composed by federated resources. In facts the KubeFed solution exploits the mechanism of KubeFed by patching the value of the clusterSelector. The initial state presents the two target applications deployed on the cloud (cluster2 and google). The following are the representations of the federated resources:

```yaml
apiVersion: types.kubefed.io/v1beta1
kind: FederatedDeployment
metadata:
  name: hostname
  namespace: my-federation
spec:
  template:
    metadata:
      labels:
        app: hostname
    spec:
     replicas: 3
     selector:
        matchLabels:
            app: hostname
     template:
        metadata:
            labels:
              app: hostname
        spec:
            containers:
            - name: hostname
              image: lorenzomarino94/hostnamepod
              env:
                  - name: NAMESPACE
                    value: "my-federation"
              ports:
              - containerPort: 80
  placement:
    clusterSelector:
     matchLabels:
      location: cloud
---
apiVersion: types.kubefed.io/v1beta1
kind: FederatedService
metadata:
  name: hostname
  namespace: my-federation
spec:
  placement:
    clusterSelector:
     matchLabels:
      location: cloud
  template:
    spec:
      ports:
      - port: 80
        targetPort: 80
      selector:
        app: hostname
      type: LoadBalancer
```

```yaml
apiVersion: types.kubefed.io/v1beta1
kind: FederatedDeployment
metadata:
  name: nginx
  namespace: my-federation
spec:
  template:
    metadata:
      labels:
        app: nginx
    spec:
      replicas: 1
      selector:
        matchLabels:
          app: nginx
      template:
        metadata:
          labels:
            app: nginx
        spec:
          containers:
          - image: nginx
            name: nginx
  placement:
    clusterSelector:
     matchLabels:
      location: cloud
---
apiVersion: types.kubefed.io/v1beta1
kind: FederatedService
metadata:
  name: nginx
  namespace: my-federation
spec:
  placement:
    clusterSelector:
     matchLabels:
      location: cloud
  template:
    spec:
      ports:
      - port: 80
      selector:
        app: nginx
      type: LoadBalancer
```

Figure 39 Yaml description of the federated resources

The clusters in the federation have been previously labelled with location=onpremise or location=cloud based on the relative location (chapter 5.3.2 as reference). With the current setup, clusterSelector matches the label

“location=cloud” so the applications will be deployed on the cloud clusters. This can be checked by querying the API servers:

```
lorenzo@lorenzo-MS-7C02:~$ kubectl --context=cluster1 -n my-federation get all
No resources found in my-federation namespace.
lorenzo@lorenzo-MS-7C02:~$ kubectl --context=cluster2 -n my-federation get all
NAME                            READY   STATUS    RESTARTS   AGE
pod/hostname-6b6ffcc67d-4cnlb   1/1     Running   0          10m
pod/hostname-6b6ffcc67d-ds85p   1/1     Running   0          10m
pod/hostname-6b6ffcc67d-fbmq8   1/1     Running   0          10m
pod/nginx-f89759699-cwjxt       1/1     Running   0          2m31s

NAME               TYPE           CLUSTER-IP       EXTERNAL-IP    PORT(S)
service/hostname   LoadBalancer   10.100.194.154   172.168.20.0   80:32245/TCP
service/nginx      LoadBalancer   10.99.114.59     172.168.20.2   80:31249/TCP

NAME                       READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/hostname   3/3     3            3           10m
deployment.apps/nginx      1/1     1            1           2m31s

NAME                                  DESIRED   CURRENT   READY   AGE
replicaset.apps/hostname-6b6ffcc67d   3         3         3       10m
replicaset.apps/nginx-f89759699       1         1         1       2m31s
lorenzo@lorenzo-MS-7C02:~$ kubectl --context=google -n my-federation get all
NAME                            READY   STATUS    RESTARTS   AGE
pod/hostname-75944cf5d4-9jl95   1/1     Running   0          10m
pod/hostname-75944cf5d4-cvhwf   1/1     Running   0          10m
pod/hostname-75944cf5d4-qgg6x   1/1     Running   0          10m
pod/nginx-554b9c67f9-fmcsf      1/1     Running   0          2m35s

NAME               TYPE           CLUSTER-IP    EXTERNAL-IP      PORT(S)
service/hostname   LoadBalancer   10.0.2.207    35.195.207.131   80:30950/TCP
service/nginx      LoadBalancer   10.0.0.251    34.77.245.38     80:31810/TCP

NAME                       READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/hostname   3/3     3            3           10m
deployment.apps/nginx      1/1     1            1           2m36s

NAME                                  DESIRED   CURRENT   READY   AGE
replicaset.apps/hostname-75944cf5d4   3         3         3       10m
replicaset.apps/nginx-554b9c67f9      1         1         1       2m36s
lorenzo@lorenzo-MS-7C02:~$
```

Figure 40 Queries to the different Kubernetes API server

Another way to track the current position of the applications is to check the Status of the related federated resources. As example, the next picture shows the description of the federated deployment hostname (part of the response from the API is intentionally omitted because not useful for the purpose):

Figure 41 Description of the federated deployment hostname

The Status section lists the current clusters where the resource is currently deployed. The Events sections lists the actions of the creation on both the clusters. The same kind of information can be retrieved for the relative federated service and for the nginx application. These sections are managed by the KubeFed control plane.

To test the hostname application, we can perform a curl to hit the related service:

```
lorenzo@lorenzo-MS-7C02:~$ curl http://192.168.99.101:32245
200 OK
You've hit hostname-6b6ffcc67d-ds85p on node: cluster2
lorenzo@lorenzo-MS-7C02:~$ curl http://35.195.207.131
200 OK
You've hit hostname-75944cf5d4-qqg6x on node: gke-thesis-cluster-default-pool-01c436e3-4tqg
lorenzo@lorenzo-MS-7C02:~$
```

Figure 42 Command curl to perform a GET to the hostaname services

The IP addresses are very different because of the type of the cluster. The first is a private IP address assigned to the local Minikube VM. The address can be retrieved by: **`kubectl --context=cluster2 get nodes -o wide`**. It will be listed the nodes of the cluster but Minikube create a single node cluster so there is only one IP to take. Then the service can be access at the address nodeIP:servicePort.

```
lorenzo@lorenzo-MS-7C02:~$ kubectl --context=cluster2 get nodes -o wide
NAME       STATUS   ROLES    AGE    VERSION   INTERNAL-IP      EXTERNAL
cluster2   Ready    master   125d   v1.18.3   192.168.99.101   <none>
```

Figure 43 How to get the IP of a node

Instead, for the service on Google Cloud, because the service is of type LoadBalancer, it is assigned a public IP address from the cloud provider's load

balancer and the service is directly exposed on that address. In my Minikube clusters is present a local load balancer but it is not completely configured so the services are accessed by the IP address of the node and the port of the service. The service type LoadBalancer is an extension of the type NodePort. The type NodePort let to access the service through the node IP and the reserved port. That is why even if the services on cluster1 and cluster2 are created as LoadBalancer it still works the access through the node IP.

The next step is to configure correctly the configMaps for the application classification and for the dynamic moving application and deploy them on the cluster1. First deploy the dynamic moving application and then the app classification. The order is important because the simulation starts as soon as the application classification is deployed: in facts it has the information to trigger the re-placement.

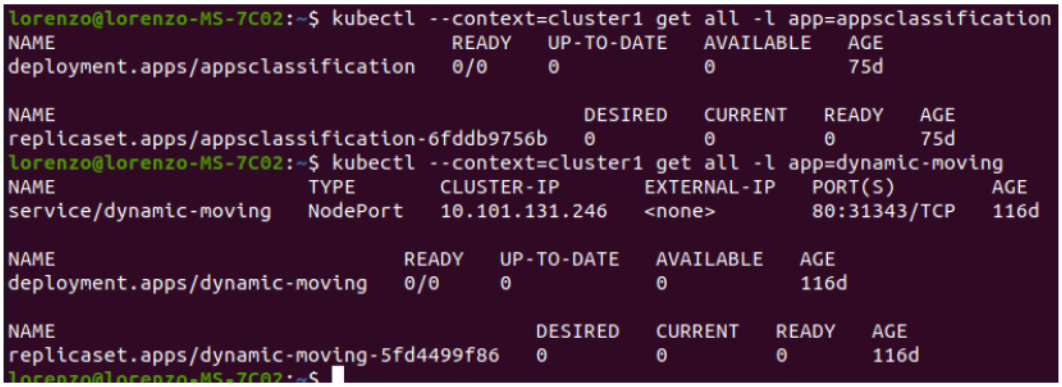


Figure 44 Dynamic and Classification application deployed on cluster1

The appClassification simulates a changing in the type of data processed by the applications and triggers the moving. The PATCH on the clusterSelector is applied and the applications hostname and nginx, one by one, are transferred to the on-premise cluster. Both the relative deployments and services are moved. If we check again the resources on the clusters, we can notice this change:

```
lorenzo@lorenzo-MS-7C02:~$ kubectl --context=cluster1 -n my-federation get all
NAME                                 READY   STATUS    RESTARTS   AGE
pod/hostname-6b6ffcc67d-kl8hd        1/1     Running   0          62s
pod/hostname-6b6ffcc67d-nh8bd        1/1     Running   0          62s
pod/hostname-6b6ffcc67d-zxh2q        1/1     Running   0          62s
pod/nginx-f89759699-s7z48            1/1     Running   0          67s

NAME               TYPE           CLUSTER-IP       EXTERNAL-IP    PORT(S)        AGE
service/hostname   LoadBalancer   10.101.145.246   192.168.20.3   80:31249/TCP   62s
service/nginx      LoadBalancer   10.100.43.121    192.168.20.1   80:31502/TCP   67s

NAME                         READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/hostname     3/3     3            3           62s
deployment.apps/nginx        1/1     1            1           67s

NAME                                    DESIRED   CURRENT   READY   AGE
replicaset.apps/hostname-6b6ffcc67d     3         3         3       62s
replicaset.apps/nginx-f89759699         1         1         1       67s
lorenzo@lorenzo-MS-7C02:~$ kubectl --context=cluster2 -n my-federation get all
No resources found in my-federation namespace.
lorenzo@lorenzo-MS-7C02:~$ kubectl --context=google -n my-federation get all
No resources found in my-federation namespace.
lorenzo@lorenzo-MS-7C02:~$
```

Figure 45 Queries on the API servers after the dynamic re-placement

The curl now can be performed only to the service on cluster1:

```
lorenzo@lorenzo-MS-7C02:~$ curl http://192.168.99.102:31249
200 OK
You've hit hostname-6b6ffcc67d-nh8bd on node: cluster1
lorenzo@lorenzo-MS-7C02:~$
```

Figure 46 Command curl after the dymanic re-placement

The simulation continues to move the applications among the clusters. In facts in a real scenario it must be possible bring back the applications when new conditions let to deploy the applications on the cloud again. The events inside the description of a federated resource confirm this correct behaviour:

```
Events:
  Type    Reason           Age                From                            Message
  ----    ------           ----               ----                            -------
  Normal  DeleteInCluster  5m21s              federateddeployment-controller  Deleting Deployment "my-federation/hostname" in cluster "google"
  Normal  CreateInCluster  5m21s              federateddeployment-controller  Creating Deployment "my-federation/hostname" in cluster "cluster1"
  Normal  DeleteInCluster  5m21s              federateddeployment-controller  Deleting Deployment "my-federation/hostname" in cluster "cluster2"
  Normal  CreateInCluster  2m11s (x2 over 41m)  federateddeployment-controller  Creating Deployment "my-federation/hostname" in cluster "google"
  Normal  CreateInCluster  2m11s (x2 over 41m)  federateddeployment-controller  Creating Deployment "my-federation/hostname" in cluster "cluster2"
  Normal  DeleteInCluster  2m11s              federateddeployment-controller  Deleting Deployment "my-federation/hostname" in cluster "cluster1"
lorenzo@lorenzo-MS-7C02:~$
```

Figure 47Events on the federated deployment hostname

In the simulation the applications are moved every 3 minutes. Obviously in a real scenario the moving depends on the occurring of the specific conditions.

To repeat the simulation with the custom solution, the dynamic moving and classification application can be scaled to 0 pods with the following commands: **`kubectl --context=cluster1 scale deployment`** `deploymentName`

`--replicas=0`. In this way there is no need to redeploy the two applications, but it let to just stop the running simulation.

6.3 Testing the custom solution

The solution exploits the mechanism of a basic installation of Kubernets, it does not require particular resources. The target applications are deployed on the cloud clusters (cluster2 and google). The object involved are simple deployment and service, no more federated:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
    name: hostname
    namespace: test-dynamic
    labels:
        app: hostname
spec:
    replicas: 3
    selector:
        matchLabels:
            app: hostname
    template:
        metadata:
            name: hostname
            labels:
                app: hostname
        spec:
            containers:
            - name: hostname
              image: lorenzomarino94/hostnamepod
              env:
                  - name: NAMESPACE
                    value: "test-dynamic"
              ports:
              - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
    name: hostname
    namespace: test-dynamic
    labels:
        app: hostname
spec:
    type: LoadBalancer
    selector:
        app: hostname
    ports:
    - port: 80
      targetPort: 80
```

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
  namespace: test-dynamic
  labels:
    app: nginx
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 2
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
        - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
    name: nginx
    namespace: test-dynamic
    labels:
      app: nginx
spec:
  type: LoadBalancer
  selector:
    app: nginx
  ports:
  - port: 80
    targetPort: 80
```

Figure 48Yaml descripion of the two target applications

We can notice that part of the content of the resource description is the same under the template section of the previous federated resource. In facts the goal of Kubernetes Federation is to federate the Kubernetes resources.

The target applications are deployed in the namespace “test-dynamic”. This process is not fundamental for the working of the solution, but it just helps to organize the workload:

```
lorenzo@lorenzo-MS-7C02:~/Documents/kubernetes/yamls$ kubectl --context=cluster1 -n test-dynamic get all
No resources found in test-dynamic namespace.
lorenzo@lorenzo-MS-7C02:~/Documents/kubernetes/yamls$ kubectl --context=cluster2 -n test-dynamic get all
NAME                            READY   STATUS    RESTARTS   AGE
pod/hostname-5bc96b4885-55l7h   1/1     Running   0          62m
pod/hostname-5bc96b4885-chtv9   1/1     Running   0          62m
pod/hostname-5bc96b4885-t4qnc   1/1     Running   0          62m
pod/nginx-d46f5678b-8c4lx       1/1     Running   0          9m22s
pod/nginx-d46f5678b-rj687       1/1     Running   0          9m22s

NAME               TYPE           CLUSTER-IP      EXTERNAL-IP    PORT(S)        AGE
service/hostname   LoadBalancer   10.107.209.40   172.168.20.5   80:31190/TCP   9m22s
service/nginx      LoadBalancer   10.96.159.218   172.168.20.4   80:32059/TCP   9m22s

NAME                       READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/hostname   3/3     3            3           62m
deployment.apps/nginx      2/2     2            2           9m22s

NAME                                  DESIRED   CURRENT   READY   AGE
replicaset.apps/hostname-5bc96b4885   3         3         3       62m
replicaset.apps/nginx-d46f5678b       2         2         2       9m22s
lorenzo@lorenzo-MS-7C02:~/Documents/kubernetes/yamls$ kubectl --context=google -n test-dynamic get all
NAME                            READY   STATUS    RESTARTS   AGE
pod/hostname-5bc96b4885-lg54r   1/1     Running   0          9m14s
pod/hostname-5bc96b4885-lwm7l   1/1     Running   0          9m14s
pod/hostname-5bc96b4885-qskwm   1/1     Running   0          9m14s
pod/nginx-d46f5678b-7s9z8       1/1     Running   0          9m15s
pod/nginx-d46f5678b-wdt9h       1/1     Running   0          9m15s

NAME               TYPE           CLUSTER-IP     EXTERNAL-IP    PORT(S)        AGE
service/hostname   LoadBalancer   10.0.9.20      34.76.81.160   80:32724/TCP   9m14s
service/nginx      LoadBalancer   10.0.8.22      34.78.30.119   80:30346/TCP   9m15s

NAME                       READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/hostname   3/3     3            3           9m14s
deployment.apps/nginx      2/2     2            2           9m15s

NAME                                  DESIRED   CURRENT   READY   AGE
replicaset.apps/hostname-5bc96b4885   3         3         3       9m14s
replicaset.apps/nginx-d46f5678b       2         2         2       9m15s
lorenzo@lorenzo-MS-7C02:~/Documents/kubernetes/yamls$
```

Figure 49 Resources present in the "test-dynamic" namespace

The applications are correctly deployed, and the services can be reached:

```
lorenzo@lorenzo-MS-7C02:~/Documents/kubernetes/yamls$ curl http://192.168.99.101:31190
200 OK
You've hit hostname-5bc96b4885-t4qnc on node: cluster2
lorenzo@lorenzo-MS-7C02:~/Documents/kubernetes/yamls$ curl http://34.76.81.160
200 OK
You've hit hostname-5bc96b4885-lwm7l on node: gke-thesis-cluster-default-pool-01c436e3-ttb6
lorenzo@lorenzo-MS-7C02:~/Documents/kubernetes/yamls$
```

Figure 50 HTTP Get on the two web servers

Before starting the simulation, it must be changed a configuration parameter for the app classification. I used two different names for the service of the relative dynamic solutions. The configMap of the app classification previously created, must be edited with the new name of the service to reach.

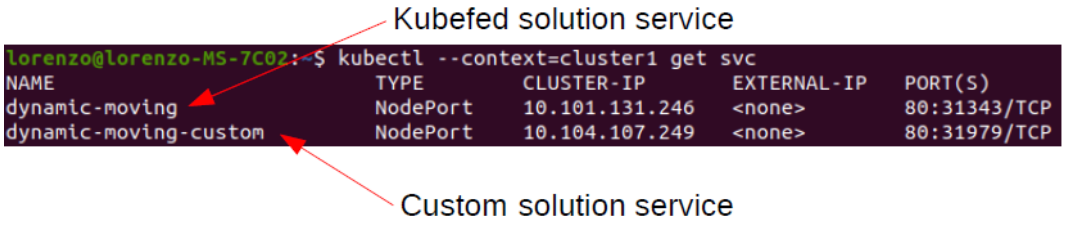


Figure 51Name of the dynamic services

The figure shows the name of the services of the dynamic applications.

When the appClassification deployment will be rescale to 1 replica, the updated configurations will be uploaded. Firstly, deploy the custom dynamic solution and then to start the simulation, just rescale the appClassification deployment. As soon as the relative pod is created, the simulation starts, and the target applications are moved to the on-premise cluster. To check the correct working, we query the three Kubernetes API servers and will see that the nginx and hostname applications are moved to cluster1. In addiction an Http Get is performed to the hostname service:

```
lorenzo@lorenzo-MS-7C02:~/Documents/kubernetes/yamls$ kubectl --context=cluster1 -n test-dynamic get all
NAME                          READY   STATUS    RESTARTS   AGE
pod/hostname-5bc96b4885-lq6sz   1/1     Running   0          57s
pod/hostname-5bc96b4885-rxtgr   1/1     Running   0          57s
pod/hostname-5bc96b4885-x6dfx   1/1     Running   0          57s
pod/nginx-d46f5678b-gsfz6       1/1     Running   0          63s
pod/nginx-d46f5678b-x7bmq       1/1     Running   0          63s

NAME               TYPE           CLUSTER-IP       EXTERNAL-IP    PORT(S)        AGE
service/hostname   LoadBalancer   10.108.246.187   192.168.20.2   80:32198/TCP   57s
service/nginx      LoadBalancer   10.97.93.56      192.168.20.1   80:32707/TCP   63s

NAME                         READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/hostname   3/3     3            3           57s
deployment.apps/nginx      2/2     2            2           63s

NAME                                  DESIRED   CURRENT   READY   AGE
replicaset.apps/hostname-5bc96b4885   3         3         3       57s
replicaset.apps/nginx-d46f5678b       2         2         2       63s
lorenzo@lorenzo-MS-7C02:~/Documents/kubernetes/yamls$ kubectl --context=cluster2 -n test-dynamic get all
No resources found in test-dynamic namespace.
lorenzo@lorenzo-MS-7C02:~/Documents/kubernetes/yamls$ kubectl --context=google -n test-dynamic get all
No resources found in test-dynamic namespace.
```

Figure 52 Applications moved to on-premise cluster1

```
lorenzo@lorenzo-MS-7C02:~/Documents/kubernetes/yamls$ curl http://192.168.99.102:32198
200 OK
You've hit hostname-5bc96b4885-x6dfx on node: cluster1
lorenzo@lorenzo-MS-7C02:~/Documents/kubernetes/yamls$
```

Figure 53 Http Get to the hostname service

After 3 minutes the appClassification notifies a change in the type of data processed by both the applications and they will be moved back to the cloud clusters. Check again the resources present on the clusters:

```
lorenzo@lorenzo-MS-7C02:~/Documents/kubernetes/yamls$ kubectl --context=cluster1 -n test-dynamic get all
No resources found in test-dynamic namespace.
lorenzo@lorenzo-MS-7C02:~/Documents/kubernetes/yamls$ kubectl --context=cluster2 -n test-dynamic get all
NAME                            READY   STATUS    RESTARTS   AGE
pod/hostname-5bc96b4885-958vk   1/1     Running   0          69s
pod/hostname-5bc96b4885-pb4rm   1/1     Running   0          69s
pod/hostname-5bc96b4885-pwddx   1/1     Running   0          69s
pod/nginx-d46f5678b-mkdrj       1/1     Running   0          74s
pod/nginx-d46f5678b-mtc4n       1/1     Running   0          74s

NAME               TYPE           CLUSTER-IP      EXTERNAL-IP    PORT(S)        AGE
service/hostname   LoadBalancer   10.98.226.145   172.168.20.5   80:32130/TCP   69s
service/nginx      LoadBalancer   10.98.143.9     172.168.20.4   80:30113/TCP   74s

NAME                       READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/hostname   3/3     3            3           69s
deployment.apps/nginx      2/2     2            2           74s

NAME                                  DESIRED   CURRENT   READY   AGE
replicaset.apps/hostname-5bc96b4885   3         3         3       69s
replicaset.apps/nginx-d46f5678b       2         2         2       74s
lorenzo@lorenzo-MS-7C02:~/Documents/kubernetes/yamls$ kubectl --context=google -n test-dynamic get all
NAME                            READY   STATUS    RESTARTS   AGE
pod/hostname-5bc96b4885-8494h   1/1     Running   0          73s
pod/hostname-5bc96b4885-qjhw9   1/1     Running   0          73s
pod/hostname-5bc96b4885-twsv6   1/1     Running   0          73s
pod/nginx-d46f5678b-95gkh       1/1     Running   0          79s
pod/nginx-d46f5678b-rvqjp       1/1     Running   0          79s

NAME               TYPE           CLUSTER-IP     EXTERNAL-IP    PORT(S)        AGE
service/hostname   LoadBalancer   10.0.2.2       34.76.81.160   80:32451/TCP   73s
service/nginx      LoadBalancer   10.0.15.122    34.78.30.119   80:31234/TCP   79s

NAME                       READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/hostname   3/3     3            3           73s
deployment.apps/nginx      2/2     2            2           79s

NAME                                  DESIRED   CURRENT   READY   AGE
replicaset.apps/hostname-5bc96b4885   3         3         3       73s
replicaset.apps/nginx-d46f5678b       2         2         2       79s
```

Figure 54 Applications moved back to the cloud clusters

```
lorenzo@lorenzo-MS-7C02:~/Documents/kubernetes/yamls$ curl http://192.168.99.101:32130
200 OK
You've hit hostname-5bc96b4885-958vk on node: cluster2
lorenzo@lorenzo-MS-7C02:~/Documents/kubernetes/yamls$ curl http://34.76.81.160
200 OK
You've hit hostname-5bc96b4885-qjhw9 on node: gke-thesis-cluster-default-pool-01c436e3-ttb6
lorenzo@lorenzo-MS-7C02:~/Documents/kubernetes/yamls$
```

Figure 55 Reaching the hostname services on cloud clusters

The simulation continues until the appClassification is stopped.

6.4 Results of the tests

With the previous simulations I was able to test the two solutions. The expected results have been achieved and it was found a mechanism to automate the placement of applications on different location under certain conditions. The process can be adapted to any kind of condition, in facts the dynamic moving applications can be driven by any kind of notification system and not just by the change of the classification of data processed.

I did not find significant differences in the performance. In both cases the removing and placement of the applications on a new cluster were performed in similar times. The big difference is the underlying environment. The KubeFed control plane offers controllers that simplify the managing and monitoring of the resources. The information retrieved in the description of a federated object about the status and the events are very useful to easily monitoring the state of the resource. For a similar feature in the custom solution, it must be developed extra controllers.

7 Conclusions

To achieve the objective of the company I developed two solutions based on open-source resources. The final result is very similar, but the two approaches differ on the surrounding environment. We consider these solutions has a starting point for a future and more complete development.

7.1 Why two solutions

Two approaches have been analysed because discussing with the company, the KubeFed solution it is a valid option but difficult to adopt nowadays. In facts the Kubernetes Cluster Federation is currently in alpha release and this represent a problem for the company. Notice that the current project is also called Federation V2 because an initial project for a Kubernetes federation was abandoned to starting a new one with different concepts. The hope is that the actual project can become production ready in a while but, without a certain future, if the company has to choose between the two proposed solutions, they will opt for the custom one. I do not think the KubeFed solution has to be totally discarded but just keep in mind and ready to use in case the project becomes production ready. In facts KubeFed can offer some management features out of the box that with a custom solution have to be built from zero. The dynamic placement exploits the main function of KubeFed that is synchronize a desired state in all the clusters: by changing just a label (the ClusterSelector) the resource will be replaced in the correct location. In addiction the control plane offers a mechanism to easily keep track of the current position of a resource by looking at the status section in the object description. At the moment, Kubefed offers other few features, like a service discovery across the federation, and we do not know if other additional features will be developed. The risk of adopting this solution is that if in the future will be added many new features not related with the problem of dynamic moving the applications, in the end the clusters will run a control plane with not required or not adopted functionalities.

Writing a custom solution means develop only the required functionalities. It represents for the company a more stable starting point for future developments

because it is strictly related to the basic mechanisms of Kubernetes and can be easily integrated with desired features. It needs to implement a way to interact with the cluster's API servers, a feature not required for the KubeFed solution because after the joining of the clusters in the federation, KubeFed provides a single point of interaction. The moving process results different because the KubeFed solution exploits a function of the KubeFed control plane not present in the basic Kubernetes environment.

7.2 Future improvements

Both my solutions are in an early stage of development, so some functionalities have been simplified for the sake of testing the main process of dynamic placing the applications. In this section I will introduce some future improvements that can be applied to the solutions to continue their development.

7.2.1 CustomResourceDefinitions

To obtain a deeper integration with the Kubernetes environment is possible to introduce custom resources in our project. A custom resource [22] is an object that extends the Kubernetes API, a new resource type. The new resource is described by a CustomResourceDefiniton (CRD), a Kubernetes resource that allows to define a new custom resource with a specific name and schema. After a custom resource is defined it can be created and managed like any other Kubernetes resource in the cluster. At this point a custom resource represents only a set of structured data and to be very useful usually it is combined with custom controllers. A controller watches on the related custom resource and reacts on the events on the resource.

For example, in my custom solution can be integrated a custom resource to define the clusters configuration instead to store it in configuration objects. The result is a more flexible and integrated way to provide the configuration to the application. When my application needs the configuration of a cluster, that can be obtained with a GET of the related custom resource from the API server. Another custom resource can be used in substitution of the HTTP

POST with the information of the target application sent by the classification service. In facts, the classification service sends a set of information that can be easily packed in a resource. Instead of collecting the data in a structure an pass this data with a HTTP POST, the service could create a new custom resource, for example called “Target”, and the logic of the two solutions moved inside a controller for that kind of custom resource. The controller watches for events on that resource so when a new resource is added to the cluster, via the CREATE operation on the API performed by the classification service, it can run the process to dynamically move the target application in a new location.

```yaml
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  name: targets.dynamic.moving.com
spec:
  group: dynamic.moving.com
  versions:
    - name: v1
      served: true
      storage: true
      schema:
        openAPIV3Schema:
          type: object
          properties:
            spec:
              type: object
              properties:
                namespaceTarget:
                  type: string
                application:
                  type: string
                sourceCluster:
                  type: string
                destinationCluster:
                  type: string
  scope: Namespaced
  names:
    plural: targets
    singular: target
    kind: Target
    shortNames:
    - tg
```

Figure 56 Example of a custom resource definition for the target application

The Custom Resource Definition above contains the definition of the custom resource Target. The properties section contains the parameters exchanged between the dynamic moving and classification application. With the use of this new custom resource, the classification service will create a Target resource like in the following figure:

```
apiVersion: "dynamic.moving.com/v1"
kind: Target
metadata:
  name: targetinfo
spec:
  namespaceTarget: test-dynamic
  application: hostname
  sourceCluster: cluster2
  destinationCluster: cluster1
```

Figure 57 Example of the Target custom resource

The resource, called “targetinfo”, is now present in the cluster and can be obtained by the API server.

```
lorenzo@lorenzo-MS-7C02:~$ kubectl get targets
NAME         AGE
targetinfo   3m15s
```

Figure 58 Get the custom resource from the API server

7.2.2 Improve security

During the developing of my solutions I did not focus too much on the security of the applications. This was intentionally done because the main goal was to exploit the basics of Kubernetes and KubeFed to achieve a dynamic placement on particulars requirements. These applications can have potentially the authorization to manipulate every kind of resource because if they need to move a whole application, they need to move all the component of that. It is better to create dedicated service accounts because if it is not specified inside the pod spec section, Kubernetes assigns to the pod a default service account. If you grant some authorization’s policies with high privilege to the default service account, all the pods associated to the default account

will have high privilege. Therefore, create separate service accounts and role bindings is a good practice to limit the privilege to not authorized pods.

In addition, all the keys and certificates used for the cluster authentication are passed as a Secret [23]. Thinking about a more improved version close to the production, these sensitive data must be handled correctly. Kubernetes offers this kind of resource to store sensitive data. This prevents to hard code these data inside the application but to pass them from the outside by mounting the secret as volume in the pod or through environment variable of the pod. Anyways secrets by default are just encoded in base64 by Kubernetes so they can be easily read and decoded by anyone. For this reason, when using this type of resource in a production environment you should consider applying additional security practices. Some best practices are:

- Limit the access to the secret resources with authorization policies. Only the developed client/application must access the relative secrets. The methods get, list, and watch should be limited only to privileged entities.
- Enable encryption for the secrets so they are no longer just encoded in base64. In facts base64 encoding is way different from encryption and can be considered as storing the data as plain-text.
- Make sure that the developed application handles the secrets in a secure way, by not exposing to the outside in some logging or with untrusted party.

Security is a very important aspect in an application especially for companies that process particulars sensitive data. There are external tools that can be integrated with Kubernetes to implement different security features.

7.3 Final considerations

During my master's degree thesis, I had the opportunity to collaborate with a company on a real-case scenario. The company, like many others, has been trying for the last years to migrate their service on a new technology. This recent

technology is Kubernetes, an open-source container orchestrator born to satisfy the requests of the market. In fact, the widely adoption of microservices application required a new way to manage all the containers involved. Kubernetes offers a starting environment that can be extended and customized based on personal necessities. In this case the company has the necessity to dynamic place the running applications on different location from the current one. Even if the reasons of this behaviour can differ from company to company, it is a solution that affect many. In my thesis, the interested reason is the kind of data processed by the services, where the company has the constraint to handle a determined type of data, considered by them as classified, only on a cluster running on-premise.

My solutions required an initial deep study on the functionalities of Kubernetes and on the ways to interact with different clusters. The research led me to the concept of federation implemented by KubeFed and I found in it a valid technology to build a solution around. In particular, I found very useful the feature of the KubeFed control plane of continuously monitoring and re-synching the state of the federated resources. The status of a federated resource simplifies the localization of that resource by other services in case they need. The limit of the KubeFed solution is that this technology is currently in alpha release and the company is not intended to bring in production a technology in early release. For this reason, my final proposal solution to the company was the custom solution. The objective was to interact with a service already present in their environment, which I referred to as classification service, and trigger another service in charge to place the target application in a new location.

The solutions are to be intended as a prototype, a starting point of developing a production solution. The final goal of the company is to build a technology that automize a process that is currently performed by the production team manually every time a service processes classified data. With my thesis I exploited the basics to implement an interaction with multiple clusters and in the final consideration section I pointed out the important aspect of security. This aspect needs to be widely analysed before bringing the solution in production because the configuration

parameters potentially expose crucial data like keys and certificates that allow to take the control of the cluster if not managed correctly.

The objective of the thesis is considered achieved. After an initial analysis of a real use case, a solution has been proposed to the company for a good starting point to continue the developing of a final production solution.

8 Bibliografia

[1] Red Hat, «Cosa sono i microservizi?,» [Online]. Available: https://www.redhat.com/it/topics/microservices/what-are-microservices.

[2] Kubernetes, «What is Kubernetes?,» [Online]. Available: https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/.

[3] Kubernetes, «Borg: The Predecessor to Kubernetes,» 23 April 2015. [Online]. Available: https://kubernetes.io/blog/2015/04/borg-predecessor-to-kubernetes/.

[4] J. B. K. H. Brendan Burns, «APPENDIX A: Building a Raspberry Pi Kubernetes Cluster,» in *Kubernetes: Up and Running, 2nd Edition*, O'Reilly Media, Inc., 2019.

[5] Kubernetes, «minikube,» [Online]. Available: https://minikube.sigs.k8s.io/docs/.

[6] Kubernetes, «kind,» [Online]. Available: https://kind.sigs.k8s.io/.

[7] Kubernetes, «Running in multiple zones,» [Online]. Available: https://kubernetes.io/docs/setup/best-practices/multiple-zones/.

[8] Booking.com, «Shipper,» [Online]. Available: https://github.com/bookingcom/shipper.

[9] Google, «Anthos,» [Online]. Available: https://cloud.google.com/anthos.

[10] Istio, «What is Istio?,» [Online]. Available: https://istio.io/latest/docs/concepts/what-is-istio/.

[11] Submariner, «SUBMARINER,» [Online]. Available: https://submariner.io/.

[12] Kubernetes, «Kubernetes Components,» [Online]. Available: https://kubernetes.io/docs/concepts/overview/components/.

[13] Docker, «Orientation and setup,» [Online]. Available: https://docs.docker.com/get-started/#containers-and-virtual-machines.

[14] Kubernetes, «client-go,» [Online]. Available: https://github.com/kubernetes/client-go.

[15] kubernetes-sigs, «kubefed,» [Online]. Available: https://github.com/kubernetes-sigs/kubefed.

[16] Kubernetes, «Service, Discovering services,» [Online]. Available: https://kubernetes.io/docs/concepts/services-networking/service/#discovering-services.

[17] kubernetes-sigs, «Using Cluster Selector,» [Online]. Available: https://github.com/kubernetes-sigs/kubefed/blob/master/docs/userguide.md#using-cluster-selector.

[18] kubernetes-sigs, «Cluster registration,» [Online]. Available: https://github.com/kubernetes-sigs/kubefed/blob/master/docs/cluster-registration.md.

[19] K. aojea, «portAllocator sync local data before allocate,» [Online]. Available: https://github.com/kubernetes/kubernetes/pull/89937.

[20] Google, «Authenticating to the Kubernetes API server,» [Online]. Available: https://cloud.google.com/kubernetes-engine/docs/how-to/api-server-authentication#environments-without-gcloud.

[21] I. Nginx, «Nginx,» [Online]. Available: https://hub.docker.com/_/nginx.

[22] Kubernetes, «Custom Resources,» [Online]. Available: https://kubernetes.io/docs/concepts/extend-kubernetes/api-extension/custom-resources/.

[23] Kubernetes, «Secrets,» [Online]. Available: https://kubernetes.io/docs/concepts/configuration/secret/.