MÁSTER UNIVERSITARIO EN

INGENIERÍA DE TELECOMUNICACIÓN

TRABAJO FIN DE MÁSTER

THE IMPACT OF ON-PREMISES PLATFORM AS A SERVICE AND ITS POTENTIAL IMPACTS TO INFORMATION TECHNOLOGY INSTRUCTION

Estudiante	*Rada Dafonte, Unai*
Director/a	*Hajek, Jeremy*
Departamento	*Information Technology and Management*
Curso académico	*2020-2021*

Bilbao, 3 de septiembre de 2021

Resumen

Hace unos años, el sector profesional se introdujo en el término "cloud" mediante la implementación de tecnologías basadas en el procesamiento y almacenamiento de datos en la nube que permiten una mayor disponibilidad y eficiencia en la utilización de recursos físicos. Sin embargo, esta tendencia no ha sido la misma en el sector educativo, el cual puede verse igualmente beneficiado por la adopción de dichas tecnologías para una gestión más eficiente de la infraestructura y una mejora en la calidad de la educación ofertada.

Con el fin de demostrar estos beneficios, se ha desarrollado una plataforma como servicio (PaaS) basada en OpenShift, de RedHat. La elaboración de este trabajo cuenta con el proceso de instalación del "cluster" de servidores en un entorno virtual y la descripción de los beneficios obtenidos tras realizar varias pruebas sobre la plataforma. Además, se ha incluido un plan para mejorar el proyecto en el futuro, con una explicación detallada de cómo replicarlo en un entorno físico.

Abstract

A few years ago, the professional sector was introduced to the term "cloud" by implementing technologies based on the processing and storage of data in the cloud that allow greater availability and efficiency in the use of physical resources. However, this trend has not been the same in the education sector, which can also benefit from the adoption of these technologies for a more efficient management of the infrastructure and an improvement in the quality of the education offered.

In order to demonstrate these benefits, a platform as a service (PaaS) based on OpenShift, from RedHat, has been developed. The preparation of this work includes the process of installing the "cluster" of servers in a virtual environment and the description of the benefits obtained after performing several tests on the platform. In addition, a plan to improve the project in the future has been included, with a detailed explanation of how to replicate it in a physical environment.

Laburpena

Duela urte batzuk, sektore profesionala "Cloud" terminoan sartu zen, datuak hodeian prozesatzean eta biltegiratzean oinarritutako teknologiak inplementatuz. Teknologia hauek, baliabide fisikoen erabilgarritasunean eskuragarritasun eta eraginkortasun hobea ahalbidetzen dute. Hala ere, joera hori ez da berdina izan hezkuntza-sektorean; izan ere, teknologia horiek azpiegitura modu eraginkorragoan kudeatzeko eta eskainitako hezkuntzaren kalitatea hobetzeko erabil daitezke.

Onura horiek frogatzeko, RedHaten OpenShift oinarritutako zerbitzu (PaaS) plataforma bat garatu da. Lan hau egiterakoan, zerbitzarien "klusterra" ingurune birtual batean instalatzeko prozesua eta plataformari buruzko hainbat proba egin ondoren lortutako onurak deskribatu dira. Gainera, etorkizunean proiektua hobetzeko plan bat elaboratu da, ingurune fisiko batean lana erreplikatzeko azalpen zehatz-mehatz batekin.

Table of Contents

Table of Figures

Table of Figures (Tables)

ACRONYMS

CRC	CodeReady Containers
IIT	Illinois Institute of Technology
ITM	Information Technology & Management
OS	Operative System
RHEL	RedHat Enterprise Linux
GUI	Graphical User Interface
FCOS	Fedora Core OS
AWS	Amazon Web Services
GCP	Google Cloud Platform
VE	Virtual Environment
KVM	Kernel Virtual Machine
CPU	Core Processing Unit
RAM	Random Access Memory
GB	Gigabyte
DNS	Domain Name System
DHCP	Dynamic Host Configuration Protocol
OCP	OpenShift Container Platform
XRDP	Microsoft Remote Desktop Protocol
VPN	Virtual Private Network
ISO	Optical Disc Image
LAN	Local Area Network
MAC	Media Access Control
AWS	Amazon Web Services
EKS	Elastic Kubernetes Service
AKS	Azure Kubernetes Service
CSR	Certificate Signing Requests
NFS	Network File System
CLI	Command Line Interface
DB	Database
URL	Uniform Resource Locator
API	Application Programming Interface

SRE	Site Reliability Engineering
EFK	Elasticsearch, Fluentd and Kibana
ELK	Elasticsearch, Logstash and Kibana
RBAC	Role-based Access Control

1. Introduction

In recent years, application deployment has started to show a drastic evolution in the enterprise sector of software development. Numerous companies all around the world have initiated a transition to cloud-based infrastructures in their datacenters and have taken advantage of the benefits of the term “cloud”.

Alongside the enterprise sector transitioning to cloud environments to deploy their applications and provide their services, many other kinds of organizations like high-level educational organizations have started to study the inclusion of on-premises and off-premises cloud infrastructures in an effort to improve their students’ skills in that contemporary technology, making it easier to transition to the enterprise, and to provide a better infrastructure for research teams to deploy their applications and centralize the management of those services.

Since nowadays the cloud has a great importance in Information Technology instruction, deploying a cloud-based platform for the use of the university can help the organization transition from its dated infrastructure to a modern environment, improving the educational level offered. Furthermore, at Illinois Institute of Technology, deploying an application for a department can take plenty of time since each project needs to include the physical server in the budget, acquire the machine, install the corresponding OS and libraries and install the application without any reliability or scaling possibility.

All things considered, this project intends to research the best solution to that matter by creating an on-premises OpenShift cluster and studying the capabilities of a Kubernetes Platform as a Service and its impacts in the Information Technology instruction. The main goal is to prove that the IIT can benefit from updating their infrastructure to deploy a OCP counting on the support of one of the biggest companies providing cloud services such as RedHat.

In this document, the different steps followed in the development of the project are presented. First, the context section goes over the basic definitions to understand the technical aspects of the cloud environments and explains the different considerations that were made to decide between different solutions. It also describes why the proposed solution may help develop future projects and how it can be improved in the future. After setting the goals of the project, detailing the possible benefits of it, and studying the different alternatives to the chosen platform, a detailed description of the methodology of the solution is described. The solution contains the process followed step by step with proper explanations as to why each step is necessary to achieve the final goal of deploying the cluster. Finally, the conclusions section provides personal thoughts of the researcher about the project and describes possible future improvements to further develop after the completion of this current stage of the project.

2. Context

In the application development sector, the monolithic approach is still the most common application architecture pattern in production. A monolithic system is a server-side system based on a single application. This means all the different services or tasks that form an application are entirely bundled in a single pack. This approach has its benefits when developing and deploying applications in certain conditions. However, there are some challenges or problems this type of systems present:

- They are highly dependent systems. There are shared libraries within the architecture and if the developers make a change, they have to take into account what other components rely on those libraries. They are easy to break over time
- They are language and framework dependent. If the application is coded in a certain language, all the additional components need to be coded in that same language. This limits the development by the decisions made in the past.
- Growth or scaling up the application. Adding components to the

application makes it harder over time to understand every little intricacy about the application as it grows. It is difficult to stabilize the deployments of the applications after adding components. Scaling the application is difficult too because it takes a lot of time to replicate the entire bundle pack of the components of the application.

The most popular alternative to these monolithic systems is the concept of microservices. The microservices approach is based on decoupling every different component of the application, separating every functionality in different files, classes and functions. This is a service-oriented architecture, where the application splits into different layers, and each of those layers has a different business goal and task boundaries. A microservice is an application architecture that takes every application function and put them in their own service, that run into a container. The communication among the different layers or containers is done through APIs. Some of the advantages of this architecture are the following:

- Because the services are independent, it gives the developers flexibility to choose different languages and frameworks for each of the services without affecting the others.
- Each service gets developed independently, making it easy to iterate at will and improve each service at its own pace.
- There is less risk in changing the services, because the changes are minimal in functionality and only an independent service fails without affecting the entire application.
- The components are scalable independently.

In the microservice architecture, the concept of container becomes significantly important. In the monolithic architecture, when deploying an application, it is common to create a VM with a specific OS and installed libraries just for that application. In the case of containers, they differentiate by the way the virtualization happens. Containers are based on Operating System virtualization, whereas VMs are based on Hardware level virtualization. This means containers can achieve a process level isolation and have their own process system instead of the machine isolation VMs provide. Containerization can help consume less resources from the physical server they are running in because they are much more lightweight than actual VMs where each of those VMs has its own OS and libraries for which the hypervisor needs to provide resources.

Containers solve some big issues with VMs, like shipping software from A to B being reliable and automatic and running exactly in the same way in both A and B. Since applications have gotten more complicated, they have a lot of configurations that need to be transferred from one machine to another expecting the same results in every different environment they run in. However, there are some cases where VMs can still be the best option to deploy an application so both possibilities should be studied before starting the development.

When dealing with multiple containers that need to communicate between them to form the final application, container orchestration comes into play. Kubernetes is the most popular container orchestration platform. It is cloud native and its main purpose is to make cloud computing universal and sustainable. It is open source and is installed into multiple hosts to create a cluster of servers where containers are scheduled into each of the hosts based on the available resources and can be easily found and connected by other microservices via service discovery mechanisms and internet protocols. Kubernetes has numerous features that make it the perfect platform for running a microservices based architecture:

- Automatically run containers (in pods, as will be explained later) in any host, and resource scheduling system, running containers in hosts with the required set of resources or other requirements
- Self-healing (restarts crashed containers, reschedules pods to healthy hosts, etc.)
- Service discovery so that each microservice can find and communicate with one another
- Horizontal scaling and load balancing between microservices instances

- Automatic rollouts and rollbacks
- Storage access, and dynamic storage provisioner
- Stateful microservices support
- Configuration management and encrypted config files
- Logs viewer
- Batch execution and jobs manager
- Roll-based Access Control (RBAC) permissions for every action

To take advantage of all these features, it is important to know the inner workings of Kubernetes. However, in most organizations, developers only focus on developing the applications and improving their functionality, and all the knowledge required to use Kubernetes can be an additional burden for them. Deploying, monitoring and managing the applications is usually an operator's job. To make the Kubernetes platform more accessible to the developers making it easier for them to use the container orchestration platform, there are some companies that provide their own service on top of the Kubernetes platform as PaaS. One of the most popular ones and the one reviewed in this project is OpenShift, from RedHat. This platform is built on top of Kubernetes and adds some features and functionalities to provide all the benefits from Kubernetes and more to the developers without making it difficult to learn.

Some of the additions or differences between Kubernetes and OpenShift are the following:

- **Deployment.**

Deploying an application in Kubernetes can be a little time consuming. Having the code in GitHub, pulling that code to the developer's local machine and spin up a container. Once the container is running, they have to decide, where do they host it (the registry). One of the options is DockerHub. Then they have to decide their CI/CD story. This is where gets complicated because there are so many options to deploy their applications.

In OpenShift all developers have to do is create an app and a project. OpenShift does the heavy lifting on the backline. It creates the pipelines, all the automation they need to do things like development, test and production for their apps. It makes it a lot easier than with only using Kubernetes. However, there's a lot more flexibility on the Kubernetes side of things. This flexibility may not always be necessary. In this case, OpenShift is going to be tested out for Information Technology instruction and it can help students get familiar with the Kubernetes environment and main features without having to worry about every little detail.

- **Management**

In Kubernetes, there's usually the need to install additional dashboards (ELK stack, Grafana, Istio…). On the openshift side of things, there's an opinionated way of doing this. It has a great web console that builds on the Kubernetes APIs and comes with a lot of great capabilities for SREs and operation teams to really manage their workloads. For the dashboards, they suggest an EFK stack, Ansible playbooks… Managing applications is a little bit easier than in Kubernetes with the caveat that some of the flexibility is lost.

- **Node configuration / Day to day operations**

The Kubernetes way of adding new VMs into the cluster can be time consuming. Things like setting up self-registration, different cloud automation, creating new VMs, etc require scripts to be developed. On the other hand, OpenShift makes it easier by providing Ansible playbooks and installers to bring new VMs into the cluster. It also has ways to handle autoscaling in response to load.

- **Security**

In this section, RedHat tries to fill the gaps where the open-source community hasn't done so. Since they work with real enterprise customers, they realize that by creating best security practices from scratch, they really are able to tackle some of the issues that some of the customers need to solve be able to use Kubernetes. Since projects are usually developed by teams where each of the developers needs to have different permissions. In the beginning, Kubernetes didn't have the capability to solve that problem (Nowadays, there are some tools like RBAC or IAM Authentication that allow this). These tools need to be set up and added to the cluster, which can be time consuming. In OpenShift, these services come setup by default when creating a project and the administrator only needs to worry about adding the users. The tradeoff is that in OpenShift there are restricted permissions where the containers don't run as root, which can make images not run as expected.

After considering all the differences between plain Kubernetes and OpenShift, an educational based organization like the IIT can be benefited from having a cluster available for students and research teams to get used to the platform in a cloud-based infrastructure. These benefits are reviewed in the Benefits section. This document reviews the installation and deployment of an OpenShift cluster with OKD, the open-source version of OpenShift.

3. Goals

In this section, the main goals of the project are presented and the scope of the project is stablished. These goals are clear from the beginning and achievable within the scope of the project. The need for this project has been defined in the Introduction section. The goals presented in this section are related with the need of the project and help the researchers involved make their own conclusions when the project development has come to an end.

- Study the benefits of an on-premises and an off-premises environment from the point of view of an educational organization and their effect in Information Technology instruction.
- Analyze the best PaaS options in the market considering the needs of the organization and provide the reasons why one of them can be the best solution.
- Use the available resources of the Information Technology department to create an on-premises infrastructure to deploy a cluster and test its

functionalities and tools and develop a solution to scale up the environment for use in the entire organization.

- Learn the basic concepts of Kubernetes and transfer that knowledge to the OpenShift Container Platform by testing the deployment of simple applications and configuring different services.

4. Benefits

Throughout this section, the benefits provided by the execution of the project are presented from an economic, social and professional point of view.

4.1. Economic impact

As an educational based organization, the budget for new research projects is limited to the budget of the department carrying the project. Furthermore, these projects don't usually bring any revenue. All the departments that need to run applications for their projects need to spend some of the budget in a new physical server to install the application in it and manage it by themselves. This spent could be used for other stuff like hiring other researchers or buying more equipment to work on the project. The OCP can solve this issue by centralizing the management of the applications of multiple departments in a unique cluster and allowing each department to focus their spends on other needs for their projects.

4.2. Professional impact

From a professional point of view, allowing the researchers or students not worry about the servers and management of the machines to deploy their applications lets them focus on improving those applications and spend less time in deploying them. Also, those applications being deployed can help other projects progress faster and thanks to the benefits of having a cloud-based platform, they can be easily scaled up, monitored and highly available, avoiding waiting periods of time of contacting the IT department to solve the issues in the physical machines.

4.3. Social impact

Finally, the project can also have a big social impact. Most of the research projects that are developed in universities are focused on solving social issues. This project can help speed up the development of those projects by helping the researchers focus on other parts of their projects and spend less time in deploying the applications and test them as they keep improving the functionality of their applications. In the case of students, it lets them get close to the deployment process of applications in a platform designed for enterprises, which allows them get an education closer to what they will find in the organizations they join when they finish their studies.

5. Analysis of alternatives

The cloud-based environments have become more and more popular in recent years at an enterprise level. This has made companies that provide cloud services design some of their products specifically to meet production level requirements. At the moment, there are numerous options that create Kubernetes infrastructures and provide that Platform as a Service so that the final user doesn't need to worry about managing the infrastructure. In this section, some of the most popular alternatives to the solution chosen for this project are reviewed, focusing on the main differences between them and explaining why the OpenShift Container Platform was chosen as the best option.

5.1. Amazon Elastic Kubernetes Service (Amazon EKS)

Amazon EKS gives the user flexibility to start, run, and scale Kubernetes applications in the AWS cloud or on-premises. This service helps provide highly-available and secure clusters allowing the automation of tasks such as patching, node provisioning and updates. It focuses on standardizing operations across all the environments. Some of its benefits are:

- Improvement of availability and observability. EKS runs the Kubernetes control plane across multiple AWS Availability Zones, automatically detects and replaces unhealthy control plane nodes, and provides on-demand, zero downtime upgrades and patching.
- Provisioning and scaling of resources efficiently. Compute capacity doesn´t need to be provisioned separately to scale the Kubernetes applications. Compatibility with other Amazon Web Services such as AWS Fargate.
- Secure the Kubernetes environment. It automatically applied the latest security patches to the control plane of the cluster.

However, Amazon EKS does have some differences with OpenShift that make it not as good of an option as OpenShift for this project. OpenShift gives more flexibility about where to run the Kubernetes cluster. Even though both require some Kubernetes knowledge, OpenShift WebApp interface makes it easier to get up to speed and start to understand K8 terminology. In this case, since it is being considered to be deployed in an education-based organization such as a university campus, this can be a sufficient reason to choose OpenShift over EKS. Furthermore, OpenShift has the support of a big company like RedHat. This support is part of the reason why OpenShift is more expensive, but it ensures the organization that the cluster will be able to meet all the requirements thanks to the monitoring from RedHat.

5.2. Microsoft Azure Kubernetes Service (AKS)

Azure Kubernetes Service (AKS) simplifies deploying a managed Kubernetes cluster in Azure by offloading the operational overhead to Azure. As a hosted Kubernetes service, Azure handles critical tasks, like health monitoring and maintenance. Since Kubernetes masters are managed by Azure, the user only manages and maintains the agent nodes. Thus, the user only pays for the agent nodes within your clusters, not for the masters. This means the user only pays for the compute power. Going through the official documentation, the main benefits of this service are similar to the ones EKS provides. However, Microsoft Azure doesn't offer as many availability zones as Amazon. Even though this would not be much of an issue for this project, it also shares the same differences with OpenShift as EKS, which make OpenShift a better option too.

5.3. VMware Tanzu

VMware Tanzu is the VMware solution to PaaS Kubernetes environments. It is built for enterprises that deploy and manage applications at scale. Tanzu is a set of products. VMware's Kubernetes offering is called VMware Tanzu Kubernetes Grid. It has a lot of similarities with OpenShift, however, in case the cluster needs to be deployed in the data center, it is tied to the VMware-based private cloud. The management of the cluster is done through a different product called Tanzu Mission Control. The management console is not as intuitive for developers as the OpenShift one, but it does provide support from experts.

For this project, OpenShift has been chosen as the best option due to the following three main characteristics:

- It provides support from a big company like RedHat
- It has a powerful and intuitive web console to monitor and manage the cluster, as well as to deploy applications and scale them very easily. It separates the developer actions from the administrator actions allowing separate logins for each of those users.
- It has an open-source version called OKD, which is the one that is going to be installed for this project, that lets the user try the service for free and has almost all functionalities from the OCP.

6. Description of the solution

After considering the benefits of the project, studying the different alternatives, and establishing the functional details of the desired result, the next step is to carry out the solution of the project. In this section, the process of building an OpenShift cluster is going to be covered, explaining the different steps that led to the final result. In order to make the process as clear as possible, justifications of the most important decisions will be provided.

6.1. Code Ready Containers

Red Hat CodeReady Containers, also called CRC, brings a minimal OpenShift 4 cluster to a local computer [1]. The goal of this cluster is to let the developers test their applications with a minimal Openshift environment. Therefore, it is intended to run locally in the developers' desktops. There are a few differences between CodeReady Containers and a production OpenShift installation:

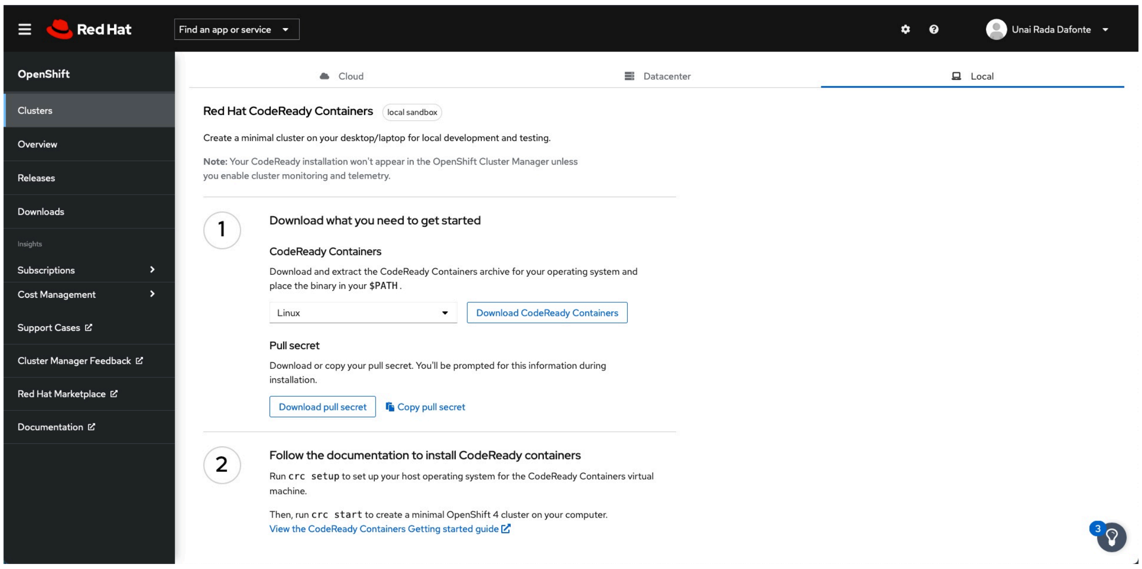
- The CodeReady Containers cluster is ephemeral and is not intended for production use.
- The cluster is formed by a single node which behaves as both a master and a worker node.
- It disables some operators by default such as *machine-config* and *monitoring*.
- The two nodes run in a single virtual machine.
- The default domain is ***.crc.testing**.

For this project, the CRC cluster will be installed to get the first contact with the Openshift web-console. After reviewing the minimum system requirements from the RedHat documentation [2] it has been decided to install the CRC cluster in one of the machines of the ITM departments' lab in the IIT campus that meets the hardware requirements. As for the Operating System requirements, RedHat states that it can be installed in any OS (Windows, macOS and various Linux distributions). Since Windows and macOS aren't free open-source OS, it has been chosen to install CRC in one of the Linux distributions. Moreover, CentOS is the community-developed version of RHEL, so it has more compatibility with CRC than Ubuntu or Debian. These last two aren't officially supported and require manual set up of the host machine, as stated in the official documentation.

First of all, CRC requires the *libvirt* and *NetworkManager* packages to run on Linux, so before starting the installation of CRC, it is necessary to install and enable these required Linux packages. In this case, for the CentOS distribution, a few other useful packages are installed and the *libvirtd* service is enabled.

```
$ sudo yum -y install qemu-kvm libvirt virt-install bridge-utils
NetworkManager
$ sudo systemctl enable --now libvirtd
```

The next step is to download the latest release of CodeReady Containers for the corresponding platform from the following url: https://cloud.redhat.com/openshift/create/local.



Red Hat CodeReady Containers local sandbox

Create a minimal cluster on your desktop/laptop for local development and testing.

Note: Your CodeReady installation won't appear in the OpenShift Cluster Manager unless you enable cluster monitoring and telemetry.

1 Download what you need to get started

CodeReady Containers

Download and extract the CodeReady Containers archive for your operating system and place the binary in your $PATH .

Linux | Download CodeReady Containers

Pull secret

Download or copy your pull secret. You'll be prompted for this information during installation.

Download pull secret | Copy pull secret

2 Follow the documentation to install CodeReady containers

Run `crc setup` to set up your host operating system for the CodeReady Containers virtual machine.

Then, run `crc start` to create a minimal OpenShift 4 cluster on your computer.
View the CodeReady Containers Getting started guide

Figure 1: RedHat CodeReady Containers installer and pull secret downloadable web page

Since CRC is managed by RedHat, it is necessary to have a RedHat account. The pull secret is a key tied to each account that will be required during installation. After extracting the binary and placing it in the $PATH, the *crc* cli is available in the system. Finally, to set up the operating system for the CRC virtual machine and starting it, there are only two commands left to run:

```
$ crc setup
$ crc start
```

The *crc setup* command performs operations to set up the environment of the host machine for the CodeReady Containers machine. The *crc start* command starts the CodeReady Containers virtual machine and Openshift cluster. When the installation is completed, there are two ways to access the cluster: through the OpenShift web console or through the client executable (*oc*). The login information is printed in the output of the *crc start* command. There are two users available at first: kubeadmin and developer. Openshift is designed so as developer functionalities are separated from operator or administrator functionalities. The developer user is meant to create projects or OpenShift applications and for application development. On the other hand, the kubeadmin user takes care of administrative tasks such as creating new users, setting roles, and so on.

As it was mentioned earlier, Openshift has a cli through the *oc* command. The steps to enable it on the host machine are specified in the official guide by RedHat. However, this is a similar way of using plain Kubernetes. The easiest way of using OpenShift and take advantage of all the tools and benefits the platform provides, is to access the cluster through the web console. It provides a well-defined GUI that lets the user interact with the cluster in a very visual way, either if it's a developer or an administrator.

The web GUI looks the same as the one implemented for OpenShift and has almost all the functionalities available, as mentioned earlier. Since the cluster is only composed by a single virtual machine, the applications deployed in it can only be accessed locally. However, due to its simple installation, it is a good alternative way of trying some of OpenShift's features without having to deploy a whole OpenShift cluster. The next section goes over the deployment of a full OpenShift cluster with OKD, which is the open-source version of OpenShift.

6.2. OKD

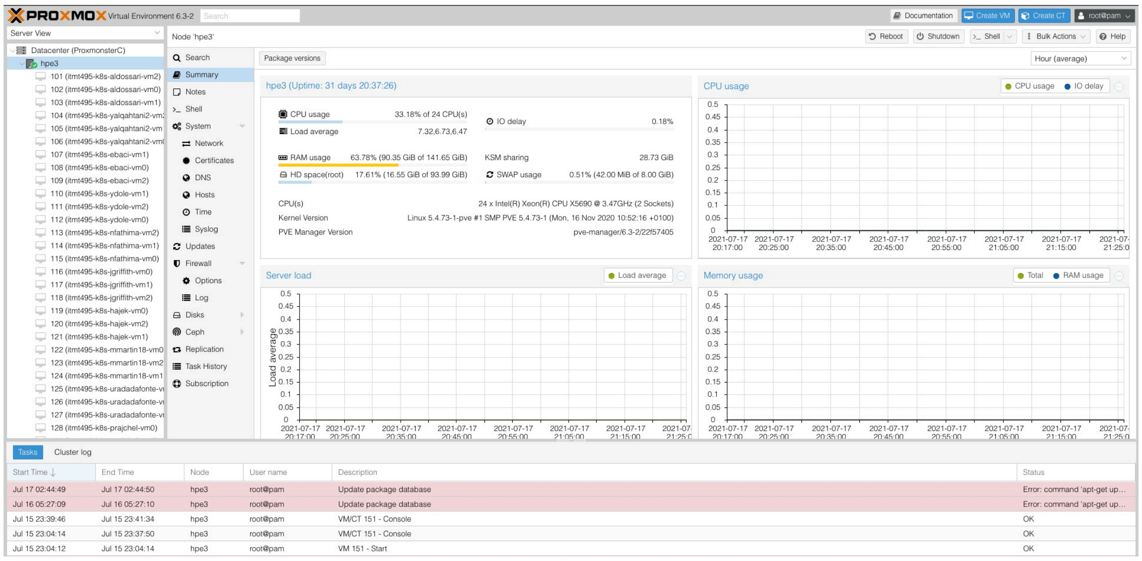
As stated in the official webpage [3], OKD is a distribution of Kubernetes optimized for continuous application development and multi-tenant deployment. OKD adds developer and operations-centric tools on top of Kubernetes to enable rapid application development, easy deployment and scaling, and long-term lifecycle maintenance for small and large teams. OKD is a sibling Kubernetes distribution to Red Hat OpenShift. OKD embeds Kubernetes and extends it with security and other integrated concepts. OKD is also referred to as Origin in github and in the documentation.

Starting with the release of OpenShift 4, the default operating system is Red Hat CoreOS, which provides an immutable infrastructure and automated updates. Fedora CoreOS, like OKD, is the upstream version of Red Hat CoreOS. Therefore, to experience OpenShift in the ITM lab, the open-source combination of FCOS and OKD are being used. The OKD installation can be done in multiple environments: public cloud infrastructures like AWS, Azure, GCP, in a type one hypervisor such us vSphere, Proxmox or VirtualBox, or even in bare metal. Before starting the installation, it is necessary to choose one of the environments where the cluster is going to be deployed and follow the required instructions from the official documentation. In the "Selecting a cluster installation method and preparing it for users" section in the documentation there are numerous questions that help decide what is the best option for the user's specific requirements. In this case, it has been decided to operate the cluster from within the ITM lab. However, since some of the network services that the machines in the lab use and the cluster installation needs, such as DHCP server or DNS server, aren't located inside the lab and it is not possible to configure them as needed, the best option is to do a bare-metal installation of the cluster with user-provisioned infrastructure in a virtual environment. Furthermore, to keep the open-source feature of the cluster installation and because of the available resources to develop this project, the cluster is going to be installed in a virtual environment with Proxmox hypervisor.

Proxmox VE, as described in the official documentation of the software [4], "*is a complete, open-source server management platform for enterprise virtualization. It tightly integrates the KVM hypervisor and Linux Containers (LXC), software-defined storage and networking functionality, on a single platform. With the integrated web-based user interface, it allows to manage VMs and containers, high availability for clusters, or the integrated disaster recovery tools with ease.*

The enterprise-class features and a 100% software-based focus make Proxmox VE the perfect choice to virtualize an IT infrastructure, optimize existing resources, and increase efficiencies with minimal expense. It allows to easily virtualize even the most demanding of Linux and Windows application workloads, and dynamically scale computing and storage as the environment needs grow, ensuring that the data center adjusts for future growth."

Proxmox VE was previously installed in one of the hosts in the lab. The IP address of this host is 192.168.172.77. The port to access the Proxmox VE GUI is 8006. The version of the installed software is 6.3-2. The host is a server with 24 CPUs and 144 GB of RAM. As it will be proven later, this is enough for the minimum physical requirements to install the OKD cluster.



hpe3 (Uptime: 31 days 20:37:26)

CPU usage	33.18% of 24 CPU(s)
Load average	7.32,6.73,6.47
IO delay	0.18%
RAM usage	63.78% (90.35 GiB of 141.65 GiB)
KSM sharing	28.73 GiB
HD space(root)	17.61% (16.55 GiB of 93.99 GiB)
SWAP usage	0.51% (42.00 MiB of 8.00 GiB)
CPU(s)	24 x Intel(R) Xeon(R) CPU X5690 @ 3.47GHz (2 Sockets)
Kernel Version	Linux 5.4.73-1-pve #1 SMP PVE 5.4.73-1 (Mon, 16 Nov 2020 10:52:16 +0100)
PVE Manager Version	pve-manager/6.3-2/22f57405

Tasks Cluster log

Start Time ↓	End Time	Node	User name	Description	Status
Jul 17 02:44:49	Jul 17 02:44:50	hpe3	root@pam	Update package database	Error: command 'apt-get up...
Jul 16 05:27:09	Jul 16 05:27:10	hpe3	root@pam	Update package database	Error: command 'apt-get up...
Jul 15 23:39:46	Jul 15 23:41:34	hpe3	root@pam	VM/CT 151 - Console	OK
Jul 15 23:04:14	Jul 15 23:37:50	hpe3	root@pam	VM/CT 151 - Console	OK
Jul 15 23:04:12	Jul 15 23:04:14	hpe3	root@pam	VM 151 - Start	OK

Figure 2: Proxmox VE Graphical User Interface

As it is stated in the official documentation, to be able to deploy an OKD cluster, the following hosts are required:

- **One temporary bootstrap machine.** The cluster requires the bootstrap machine to deploy the OKD cluster on the three control plane machines. It can be removed after the installation is completed.
- **Three control plane machines.** The control plane machines run the Kubernetes and OKD services that form the control plane.
- **Two compute machines.** Compute machines are also known as worker machines, and they run the workloads requested by OKD users.

To keep the installation simpler and due to this project being experimental and not expecting too many workloads in the cluster, the proposed cluster will be formed by the minimum number of nodes: three master nodes or control plane machines and two worker nodes or compute machines. The physical requirements for each of the nodes are also defined in the official documentation.

Machine	Operating System	vCPU	Virtual RAM	Storage
Bootstrap	FCOS	4	16 GB	120 GB
Control Plane	FCOS	4	16 GB	120 GB
Compute	FCOS	2	8 GB	120 GB

Table 1: Minimum hardware requirements for the OCP cluster

Since the infrastructure where the cluster is deployed is user-provisioned, there are also other types of requirements that have to be considered before starting the installation process. To understand the need of that previous configuration, it is necessary to first understand the installation process OKD follows.

Installation Process

When installing an OpenShift Container Platform cluster, the installation program has to be downloaded from the appropriate infrastructure provider page on the Red Hat OpenShift Cluster Manager site. In this case, since OKD is open-source, this program is downloaded from a GitHub repository [5]. Before running the installation program, in the user provisioned infrastructure, the user needs to provide all of the cluster infrastructure and resources, including the bootstrap machine, networking, load balancing, storage, and individual cluster machines.

There are three sets of files used during installation: an installation configuration file that is named "*install-config.yaml*", Kubernetes manifests, and Ignition config files for the machine types that will form the cluster. The installation configuration file is transformed into Kubernetes manifests, which are then wrapped into Ignition config files. The installation program uses three Ignition config files to create the cluster (one for the bootstrap node, one for the control plane nodes and another one for the compute nodes).

The installation program is used to generate the required assets to provision the cluster infrastructure, create the cluster infrastructure, and then deploy the cluster to the infrastructure provided by the user. There are various cluster resources that the user must manage and maintain:

- The underlying infrastructure for the control plane and compute machines that make up the cluster.
- Load balancers.
- Cluster networking, including the DNS records and required subnets.
- Storage for the cluster infrastructure and applications.

Since each machine in the cluster requires information about the cluster when it is provisioned, OCP uses a temporary bootstrap machine during the initial configuration to provide the required information to the permanent control plane nodes. The bootstrap node boots by using an Ignition config file that describes how to create the cluster. The bootstrap machine creates the control plane machines that make up the control plane of the cluster. The control plane machines then create the compute machines, as it is shown in the following figure.

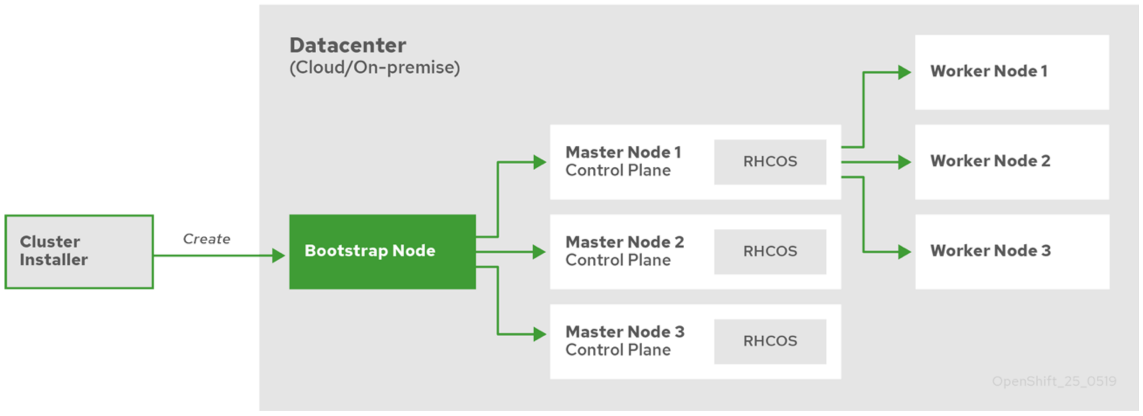


Figure 3: Installation process overview

When the worker nodes have been initialized, it means the cluster is deployed, so the bootstrap machine is no longer useful. The bootstrapping process is composed by the following steps:

1. The bootstrap machine boots and starts hosting the remote resources required for the control plane machines to boot.
2. The bootstrap machine starts a single-node etcd cluster and a temporary Kubernetes control plane.
3. The control plane machines fetch the remote resources from the bootstrap machine and finish booting.
4. The temporary control plane schedules the production control plane to the production control plane machines.
5. The Cluster Version Operator (CVO) comes online and installs the etcd Operator. This operator scales up etcd on all control plane nodes.
6. The temporary control plane shuts down and passes control to the production control plane.
7. The bootstrap machine injects OpenShift Container platform components into the production control plane.
8. The installation program shuts down the bootstrap machine.
9. The control plane sets up the worker nodes.
10. The control plane installs additional services in the form of a set of Operators.

In this case, since the user provisioned installation process is being described, some of the bootstrapping steps have to be made by the user. In step one, it is the user the one that has to provide the remote resources for the control plane so they can fetch them in step three. In step eight, it is the user the one in charge of shutting the bootstrap machine down.

The result of the bootstrapping process is a fully running OCP cluster. The cluster then downloads and configures remaining components needed for the day-to-day operations, including the creation of worker machines in supported environments.

As it has been explained, the user takes an important part in the user-provisioned installation process. Thus, some of the services the installation program needs have to be set up by the user. These requirements are detailed in the official documentation.

Network requirements

Network configuration has to be configured before starting the installation process. This is because all the FCOS machines require networking to be configured in *initramfs* during boot to fetch their Ignition config files.

During the initial boot, the machines require an IP address configuration that is set either through a DHCP server or statically by providing the required boot options. In this case, since a virtual environment is being user, there's going to be a VM specifically created to act as a DHCP server and provide each node with its corresponding IP address. It is recommended to use a DHCP server for long-term management of the cluster machines. Also, the DHCP server has to provide persistent IP addresses, DNS server information, and hostnames to the cluster machines. After a network connection is established, the machines download their Ignition config files from an HTTP server. This has to be provided by the user. In this case, there's also going to be a VM in the network of the cluster that is going to provide these files to the nodes from the cluster in a HTTP server. The Ignition config files are then used to set the exact state of each machine. The Machine Config Operator completes more changes to the machines, such as the application of new certificates or keys, after installation.

As mentioned before, a DNS server is also needed for the cluster installation. One of the main reasons why the cluster is being installed in a virtual network, isolated from the ITM lab network, is because the DNS server the machines from the lab use isn't available for this project, so it is not possible to add the configuration of the OCP cluster to it. The Kubernetes API server must be able to resolve the node names of the cluster machines. If the API servers and worker nodes are in different zones, a default DNS search zone can be configured to allow the API server to resolve the node names. Another supported approach is to always refer to hosts by their fully-qualified domain names in both the node objects and all DNS requests. This last approach is the one that is going to be followed in this document.

Setting up the environment

As it has been explained during the current section of this document, the installation is user provisioned. It is the user's responsibility to provision all the required services so the installation program can install the OCP cluster. First of all, in the Proxmox VE, the following Virtual Machines have to be created. The hardware specifications of the machines that form the cluster are the minimum hardware requirements, which have been already detailed in a previous table. In the case of the okd4-services and the okd4-pfsense machines, since they are not part of the cluster, they don't have any requirements. In the case of the okd4-services machine, it has been assigned 4 GB of RAM, 4 vCPUs and 100 GB of storage. In the case of the okd4-pfsense machine, since it is only going to host the DHCP server, it has only been assigned 1 GB of RAM, 1 vCPU and 8 GB of storage. The Operative Systems of each of the machines is displayed in the following table too. The machines of the cluster need to be FCOS, as specified previously. The DHCP server uses FreeBSD, which is an operating system used to power modern servers, desktops, and embedded platforms. Lastly, okd4-services will host CentOS 8, since it is compatible with its upstream source, RHEL and it is a robust open-source OS.

Machine	Type	OS	IP Address
okd4-bootstrap	Bootstrap	FCOS	10.0.0.200
okd4-control-plane-1	Master	FCOS	10.0.0.201
okd4-control-plane-2	Master	FCOS	10.0.0.202
okd4-control-plane-3	Master	FCOS	10.0.0.203
okd4-compute-1	Worker	FCOS	10.0.0.204
okd4-compute-2	Worker	FCOS	10.0.0.205
okd4-services	DNS/LB/Web/NFS	Centos 8	10.0.0.210
okd4-pfsense	Router/DHCP	FreeBSD	10.0.0.1

Table 2: Information about the implementation of the virtual machines

As of the IP addresses, in Proxmox VE, the virtual network created to host the components of the cluster is the 10.0.0.0/22 network. In the following illustration, there is a diagram of the most important IP addresses for this project in the ITM lab.

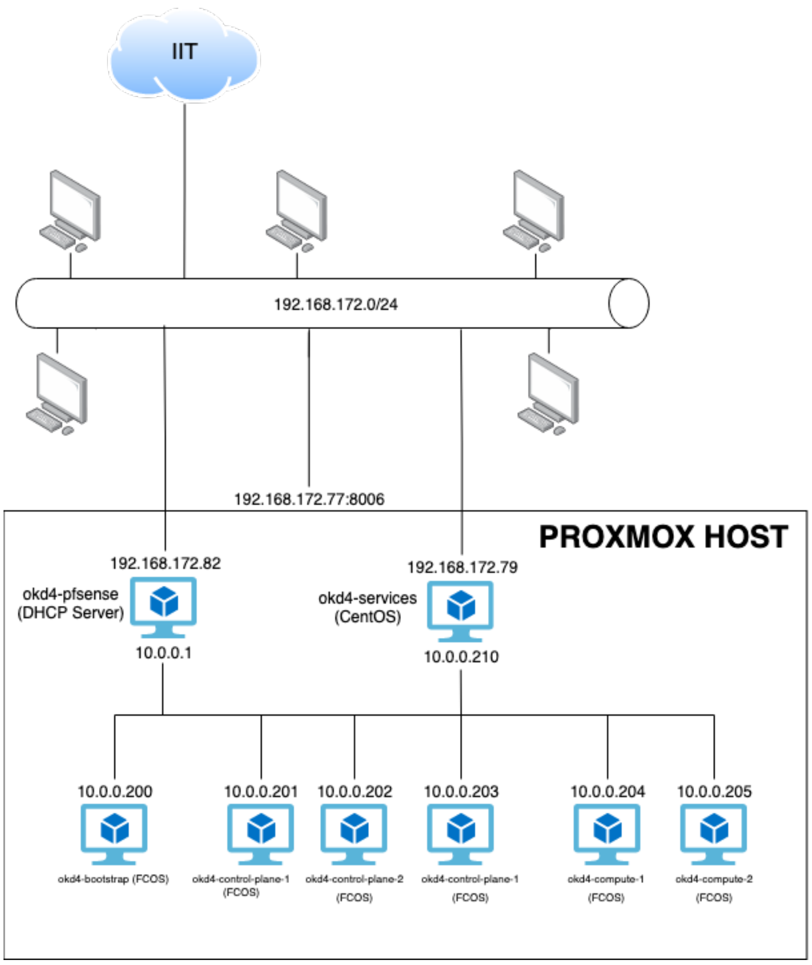


Figure 4: Network diagram of the environment

As it shows in the figure, the cluster is in a virtual network inside the Proxmox host and its only connection to the ITM lab and consequently, to the internet, is through the machines okd4-pfsense and okd4-services, which have two network interfaces. One of them in the virtual network and the other one in the network from the lab. The okd4-pfsense machine acts as a DHCP server and as a router, so the packets in and out of the virtual network go through this VM. Furthermore, as it is detailed in the documentation, it is good practice to assign the nodes' hostnames through the DHCP server. This can bypass any manual DNS record name configuration errors in environments that have a DNS split-horizon implementation.

Another important part of the network requirements are the ports each machine needs to have opened. These ports are shown in the following tables, as it is detailed in the official documentation of OKD.

Protocol	Port	Description
ICMP	N/A	Network reachability tests
TCP	1936	Metrics
	9000 - 9999	Host level services, including the node exporter on ports 9100-9101 and the Cluster Version Operator on port 9099.
	10250 - 10259	The default ports that Kubernetes reserves
	10256	openshift-sdn
UDP	4789	VXLAN and Geneve
	6081	VXLAN and Geneve
	9000 - 9999	Host level services, including the node exporter on ports 9100 - 9101
TCP/UDP	30000 - 32767	Kubernetes node port

Table 3: Ports used for all-machine to all-machine communications

Protocol	Port	Description
TCP	6443	Kubernetes API

Table 4: Ports used for all-machine to control plane communications

Protocol	Port	Description
TCP	2379 - 2380	etcd server and peer ports

Table 5: Ports used for control plane machine to control plane machine communications

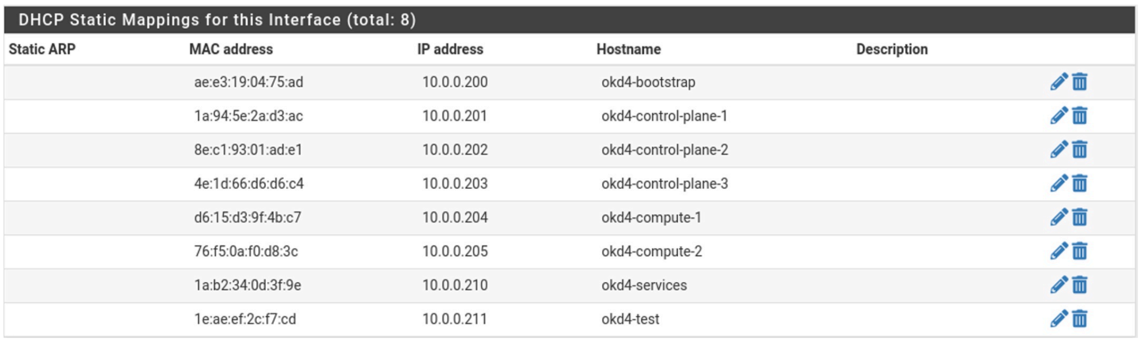
The **okd4-services** is the first one to be created and there are several services and tools that need to be installed that are crucial for the installation of the cluster. After creating the VM in Proxmox VE, it is necessary to go through the CentOS 8 installation. During this process, the “Server with GUI” is selected. It is important to have a GUI because later on this machine can access the OKD console from the browser. Another thing to take into account is that the IP address inside the virtual network has to be assigned statically, because the DHCP server isn’t configured yet. In this case, as it is shown in the network diagram, it is 10.0.0.210. The other IP address, however, is assigned by the DHCP server of the IIT that assigns IP addresses to the machines inside the ITM lab.

DHCP server

After completing the CentOS installation, XRDP is installed and enabled, to be able to access the GUI of the OS from a remote laptop connected to the IIT VPN. Before starting the setup of all the services this machine is going to host, the okd4-pfsense VM is going to be created and the DHCP server is configured. To do that, a VM is created in Proxmox with the pfsense ISO and, after assigning the 10.0.0.1 IP address statically from the console during installation, the GUI of the pfsense OS is accessible from the okd4-services VM’s browser through the following url: http://10.0.0.1. The next step is to login with the default credentials “admin” as username and “pfsense” as the password. Pfsense provides a wizard to configure the OS and assign a hostname, a domain and a primary DNS server. In this case, the following values have been assigned:

- Hostname: okd4-pfsense
- Domain: okd.local
- Primary DNS server: 10.0.0.210

To be able to configure the DHCP server and set up the DHCP reservations, all the virtual machines that form the cluster need to be created and Proxmox and a list is compiled with all the MAC addresses of the OKD nodes by viewing the hardware configuration of the VMs. In the pfsense GUI, each of the nodes of the cluster is added to the DHCP Static Mapping for the LAN interface by providing a hostname of each of the nodes and configuring the 10.0.0.210 as their DNS server. This way, when the nodes are initialized to start the installation of the cluster, they retrieve the hostname, their IP address and the DNS server's IP address from the DHCP server. All these elements are necessary to complete the installation successfully.



DHCP Static Mappings for this Interface (total: 8)

Static ARP	MAC address	IP address	Hostname	Description
	ae:e3:19:04:75:ad	10.0.0.200	okd4-bootstrap	
	1a:94:5e:2a:d3:ac	10.0.0.201	okd4-control-plane-1	
	8e:c1:93:01:ad:e1	10.0.0.202	okd4-control-plane-2	
	4e:1d:66:d6:d6:c4	10.0.0.203	okd4-control-plane-3	
	d6:15:d3:9f:4b:c7	10.0.0.204	okd4-compute-1	
	76:f5:0a:f0:d8:3c	10.0.0.205	okd4-compute-2	
	1a:b2:34:0d:3f:9e	10.0.0.210	okd4-services	
	1e:ae:ef:2c:f7:cd	10.0.0.211	okd4-test	

Figure 5: DHCP reservations in the DHCP server

Okd4-services environment setup

The okd4-services VM provides DNS, NFS exports, a web server and load balancing to the cluster environment.

DNS Server

In a OKD deployment, the following components need to have DNS name resolution:

- The Kubernetes API
- The OKD application wildcard
- The bootstrap, control plane, and compute machines

Reverse DNS resolution is also required for the Kubernetes API, the bootstrap machine, the control plane machines, and the compute machines. DNS A/AAAA or CNAME records are used for name resolution and PTR records are used for reverse name resolution. The reverse records are important because Fedora CoreOS (FCOS) uses the reverse records to set the hostnames for all the nodes, unless the hostnames are provided by DHCP. Additionally, the reverse records are used to generate the certificate signing requests (CSR) that OKD needs to operate.

In the following table, the required DNS records for a user-provisioned OKD cluster are specified. In this case, the ‹cluster_name› is “lab” and the ‹base_domain› is “okd.local”. Both variables are configured in the “install-config.yaml” file, from which the manifests are created for the installation of the cluster.

Component	Record	Description
Kubernetes API	api.‹cluster_name›.‹base_domain›	A DNS A/AAAA or CNAME record, and a DNS PTR record, to identify the API load balancer. These records must be resolvable by both clients external to the cluster and from all the nodes within the cluster.
	api-int.‹cluster_name›. ‹base_domain›	A DNS A/AAAA or CNAME record, and a DNS PTR record, to internally identify the API load balancer. These records must be resolvable from all the nodes within the cluster.
Routes	*.apps.‹cluster_name›.‹base_domain›	A wildcard DNS A/AAAA or CNAME

		record that refers to the application ingress load balancer. The application ingress load balancer targets the machines that run the Ingress Controller pods. The Ingress Controller pods run on the compute machines by default. These records must be resolvable by both clients external to the cluster and from all the nodes within the cluster.
Bootstrap machine	bootstrap.<cluster_name>.<base_domain>	A DNS A/AAAA or CNAME record, and a DNS PTR record, to identify the bootstrap machine. These records must be resolvable by the nodes within the cluster.
Control plane machines	<master><n>.<cluster_name>.<base_domain>	DNS A/AAAA or CNAME records and DNS PTR records to identify each machine for the master

		nodes. These records must be resolvable by the nodes within the cluster.
Compute machines	‹worker›‹n›.‹cluster_name›.‹base_domain›	DNS A/AAAA or CNAME records and DNS PTR records to identify each machine for the worker nodes. These records must be resolvable by the nodes within the cluster.

Table 6: Required DNS records

To create a DNS server in CentOS 8, the packets bind and bind-utils need to be installed. After that, the “named.conf” file is added to the path /etc/named.conf and the file named.conf.local is added to the path /etc/named/. Also, the “db.10.0.0” and the “db.okd.local” files need to be added to the /etc/named/zones path. These files are created based on the requirements from the official documentation. In this document, these files are included in the Appendix A section. Apart from that, it is also important to configure the localhost address in the okd4-services as the DNS server. The DNS can be tested out with the “dig” command from the okd4-services VM.

```
[urada@okd4-services ~]$ dig -x 10.0.0.210

; <<>> DiG 9.11.26-RedHat-9.11.26-4.el8_4 <<>> -x 10.0.0.210
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 23660
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 3, AUTHORITY: 1, ADDITIONAL:
2

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1232
; COOKIE: db67a129019b5879aea94dac60f7b8a6bd9de6e99e110249 (good)
;; QUESTION SECTION:
;210.0.0.10.in-addr.arpa.     IN     PTR
```

```
;; ANSWER SECTION:
210.0.0.10.in-addr.arpa. 604800    IN    PTR   okd4-
services.okd.local.
210.0.0.10.in-addr.arpa. 604800    IN    PTR   api.lab.okd.local.
210.0.0.10.in-addr.arpa. 604800    IN    PTR   api-int.lab.okd.local.

;; AUTHORITY SECTION:
0.0.10.in-addr.arpa.    604800     IN    NS    okd4-
services.okd.local.

;; ADDITIONAL SECTION:
okd4-services.okd.local. 604800    IN    A     10.0.0.210

;; Query time: 0 msec
;; SERVER: 127.0.0.1#53(127.0.0.1)
;; WHEN: Wed Jul 21 01:03:18 CDT 2021
;; MSG SIZE  rcvd: 191
```

Load Balancing

There are some load balancing requirements that the user needs to provide before installing OKD.

- API load balancer. Provides a common endpoint for users to interact with and configure the platform. The following conditions have to be configured:
  - Layer 4 load balancing only.
  - A stateless load balancing algorithm. Session persistence is not required for the API load balancer to function properly.

Port	Back-end machines (pool members)	Internal	External	Description
6443	Bootstrap and control plane. The user removes the bootstrap machine from the load balancer after the bootstrap machine initializes the cluster control plane. The user must configure the */readyz* endpoint for the API server health check probe.	X	X	Kubernetes API server
22623	Bootstrap and control plane. The user removes the bootstrap machine from	X		Machine config

	the load balancer after the bootstrap machine initializes the cluster control plane.			server

Table 7: API load balancer ports

- Application ingress load balancer. Provides an ingress point for application traffic flowing in from outside the cluster. The following conditions have to be configured:
  - Layer 4 load balancing only
  - A connection-based or session-based persistence is recommended, based on the options available and types of applications that will be hosted on the platform.

Port	Back-end machines (pool members)	Internal	External	Description
443	The machines that run the Ingress Controller pods, compute, or worker, by default.	X	X	HTTPS traffic
80	The machines that run the Ingress Controller, pods, compute, or worker, by default.	X	X	HTTP traffic

Table 8: Application ingress load balancer ports

To provide the load balancing to the cluster installed in this project, HAProxy is going to be used. HAProxy is a free and open-source proxy and load balancing server software. It provides high availability at the network and application layers. This service is installed in the okd4-services machine with the corresponding packet manager. The configuration of this service is done through a configuration file in */etc/haproxy/haproxy.cfg*. The official documentation provides a sample of this document. The configuration file used for this project is shown in the Appendix A section of this document. After adding the configuration file to the corresponding path, the service needs to be enabled and started in CentOS. To finish the configuration of the load balancer, the previously shown ports have to be open. This can be done by adding them to the firewall configuration.

Web Server

The next service that needs to be configured by the user in the okd4-services VM is the web server. The web server provides the Ignition files to the nodes in the booting process. Therefore, the web server needs to be in the same network as the nodes from the cluster. In this case, the network is 10.0.0.0/22 and the IP address of the web server is the same as the IP address of the okd4-services VM.

To configure the http server in a CentOS 8 machine, the service *httpd* is installed and, in this case, it has been configured to listen on port 8080 by changing it from port 80 in the "httpd.conf" file. After enabling, starting the service, and opening the port 8080 in the firewall configuration, the web server is available. It can be tested with the "curl" command. When running the command "*$ curl localhost:8080*" from the okd4-services machine, the metadata of the web page should be retrieved.

Setup of openshift-installer and oc client

The openshift installer is used to generate the ignition files based on the install configuration described in the file "install-config.yaml". Since OKD is open-source, the installation program is downloaded from the GitHub public repository named okd [5]. For this project, the version installed is the latest 4.7. The openshift client and the installer are downloaded and installed in the okd4-services machine. After extracting the OKD version of the oc client and the openshift install, the binaries "kubectl", "oc" and "openshift-install" are copied into /usr/local/bin to install them.

The installation configuration file includes information about the cluster. The name of the cluster, the domain and the internal IP addresses are specified in it. It also needs a public key previously generated with SSH to authenticate SSH access to the nodes later on. In the official documentation, an example of the installation config file is provided. For this project, a pull-secret has also been detailed in this file, which links the installation to a RedHat user account. This file can be found in the APPENDIX A section of this document.

To start the installation, a directory is created in the okd4-services machine where all the necessary files for the installation are going to be stored. The "install-config.yaml" filed is copied into that directory and with the "openshift-install" command, the Kubernetes manifests for the cluster are created. Since okd can be deployed without any compute nodes, there's an option in the manifests that allows pods to be scheduled on the control plane nodes. In this case, this cluster is going to have compute nodes, so that option has been changed to "false" so that pods are only scheduled on the compute nodes. After the manifests are created, the Ignition config files are created in the same directory. These files are then copied into a directory inside the web server created before in the path /var/www/html/okd4. This way, to access those files, the nodes can do that from the "10.0.0.210:8080/okd4" path.

Start the nodes

Since the FCOS ISO has been uploaded to the Proxmox VE and the nodes have been provided that datastore ISO file when the VMs have been created, the nodes can be powered on. During boot of each of the nodes, it needs to be interrupted so that the kernel boot options can be modified to add the corresponding ignition config file's location and specify the creation of the /dev/sda partition.

This process is repeated with each of the nodes in the cluster. In the case of the bootstrap, it downloads the bootstrap.ign ignition config file from the web server in okd4-services, in the case of the control-plane nodes, they download the master.ign config file and finally, in the case of the compute nodes, they download the worker.ign config file. To be able to do that, during boot, the DHCP server provides them with the specified static IP address in the pfsense DHCP server. After some time, the installation process is completed and the cluster installation is successful. However, there are still some things that need to be configured from the cluster to have a fully functional OCP cluster ready to deploy applications.

CSRs approval

Since the cluster has limited access to machine management when the user provides the infrastructure, a mechanism for approving cluster certificate signing requests (CSRs) has to be provisioned after installation. The *kube-controller-manager* only approves the kubelet client CSRs. The *machine-approver* cannot guarantee the validity of a serving certificate that is requested by using kubelet credentials because it cannot confirm that the correct machine issued the request. The user must determine and implement a method of verifying the validity of the kubelet serving certificate requests and approving them. In this project, this is done with the *jq* package from the okd4-services machine. The following command can be run while the cluster operators are installed:

```
oc get csr -ojson | jq -r '.items[] | select(.status == {} ) |
.metadata.name' | xargs oc adm certificate approve
```

After a while, the cluster operators get installed, including the console, so the cluster is finally accessible from the web browser. By accessing the following URL, the console shows up asking for login information: https://console-openshift-console.apps.lab.okd.local/

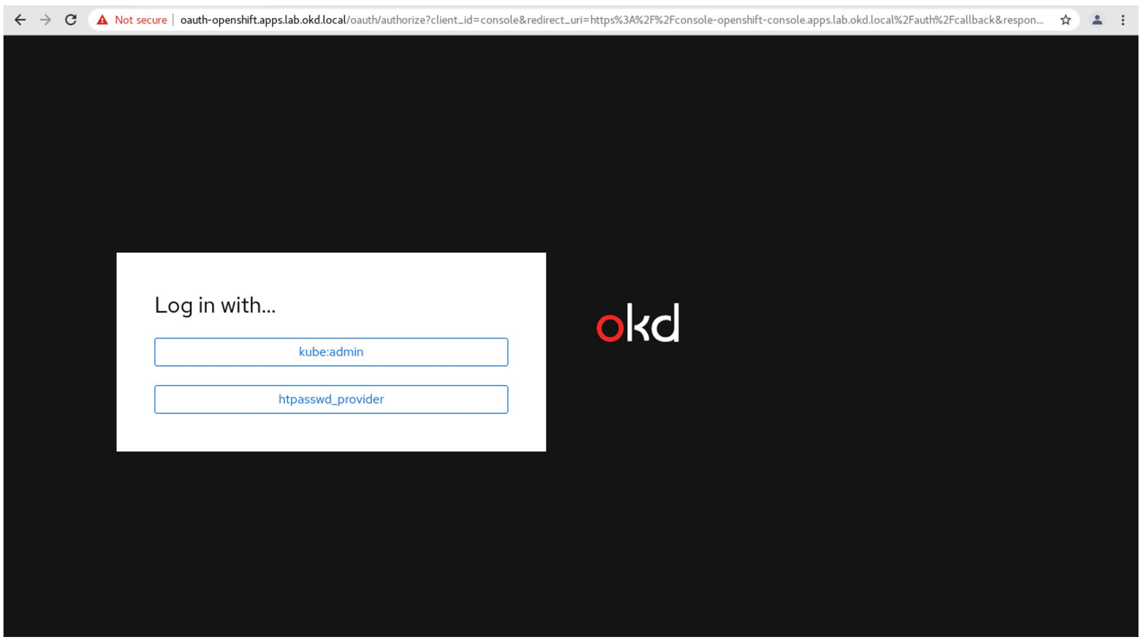

Not secure
oauth-openshift.apps.lab.okd.local/oauth/authorize?client_id=console&redirect_uri=https%3A%2F%2Fconsole-openshift-console.apps.lab.okd.local%2Fauth%2Fcallback&respon...
Log in with...
kube:admin
htpasswd_provider
okd

Figure 6: OKD login page

At first, the only available user is the *kube: admin* user. The password for this user is stored in the following path "install_dir/auth/kubeadmin-password". However, more users can be added with HTPasswd Setup, which will be explained hereunder.

HTPasswd Setup

By using the htpasswd tool in linux, it is possible to create a local user that then gets added to the *openshift-config* project by creating a secret. After creating the secret from the *users.htpasswd* file created with *htpasswd*, the identity provider is added too from the file *htpasswd_provider.yaml*. The commands to setup a htpasswd user are the following:

```
$ htpasswd -c -B -b users.htpasswd testuser testpassword
$ oc create secret generic htpass-secret --from-
file=htpasswd=users.htpasswd -n openshift-config
$ oc apply -f htpasswd_provider.yaml
```

After that, in the console the htpasswd_provider login option is selected and the "testuser" and "testpassword" credentials are entered. At that moment, this user doesn't have any administrator permissions so it is not able to see the Administrator page in the console. To give a user cluster-admin access, it can be easily done through the *oc* cli:

```
$ oc adm policy add-cluster-role-to-user cluster-admin testuser
```

This way, the users with admin access can be limited and the rest of the users can only access the developer settings from the cluster, increasing the security level of the cluster.

Persistent Storage

To be able to complete the cluster installation, a private registry needs to be configured. To do that, this section goes through the setup of a NFS server in the okd4-services machine, creating an empty directory that will be the storage location for the images created in the cluster.

To create a NFS server, the *nfs-utils* package is installed in the okd4-services machine and the *nfs-server* and the *rpcbind* services are enabled and started. After that, a directory is created in /var/nfsshare/. In this case, the file is called "registry", being the complete path /var/nfsshare/registry. After that, the folder permissions are configured. In this case, it has been given full permission.

Finally, the following line is added in /etc/exports file, indicating the address of the network from which the nfs server is accessible:

```
[urada@okd4-services okd4_files]$ cat /etc/exports
/var/nfsshare
10.0.0.0/16(rw,sync,no_root_squash,no_all_squash,no_wdelay)
```

The NFS server is finally ready after enabling the *mountd*, *rpc-bind* and *nfs* services in the firewall configuration and restarting the *nfs-server* service.

Registry Configuration

In the end of the cluster, the Image Registry Operator is not initially available for platforms that do not provide default storage. After installation, the user must configure a private registry to use storage so that the Registry Operator is made available.

In the official documentation, the instructions are shown for configuring a persistent volume, which is required for production clusters. In this case, an empty directory must be configured as the storage location. To create the persistent volume on the NFS share, the "registry_py.yaml" file is used, which is shown in the APPENDIX A section at the end of the document. This file contains the configuration for the persistent volume. In this case, it is a volume of 100 GB, which can be increased in the future. This is accomplished through the *oc* cli.

After creating the persistent volume, the image-registry operator needs to be edited to indicate that the state of the volume is "Managed". This way, the persistent storage status should automatically change to "Bound". At the beginning, the export size of the /var/nfsshare/registry should be zero. However, as images are built in the cluster, these are stored in this directory, adding the size of the images to the export size.

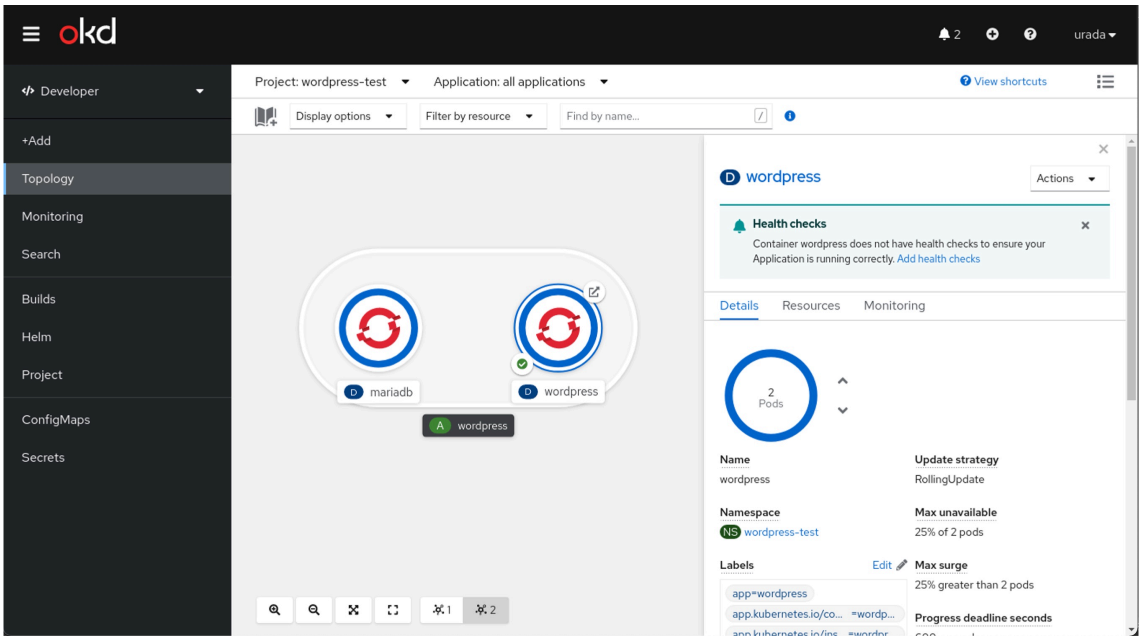
Deploying the first application

At this point, the cluster is running and fully operating. In this document, the creation of a simple WordPress application through the console is covered, by creating a WordPress container and a MariaDB container that define the complete application. This shows how to take advantage of microservices in OpenShift and how easy it is for the developer to focus on the development of the application instead of the management of the deployed containers.

After creating a project, which is similar to namespaces in Kubernetes, the two apps need to be created, one for the WordPress frontend and another one for the MariaDB database. To create the WordPress application, the centos php73 container is downloaded from GitHub by the cluster. The "$ oc expose svc/wordpress" command exposes the application to outside of the cluster by creating a Kubernetes service of type ClusterIP. OpenShift also creates a "route", which provides a full domain name for the IP address and port where the service is exposed. This way, the developer grabs doesn't need to worry about that and can just access the application through the URL provided by OpenShift. The MariaDB app is created from the centos7 MariaDB image with some environment variables.

```
$ oc new-app centos/php-73-
centos7~https://github.com/WordPress/WordPress.git
$ oc expose svc/wordpress
$ oc new-app centos/mariadb-103-centos7 --name mariadb --env
MYSQL_DATABASE=wordpress --env MYSQL_USER=wordpress --env
MYSQL_PASSWORD=wordpress
```

As a result, in the following figures, the OKD console is shown with both containers created and added to an application group inside the "wordpress-test" project. When opening the wordpress container from the external link created by OpenShift for the first time and going through the configuration of wordpress, the "Hello World" page is shown and the application is finally deployed and running.



Project: wordpress-test
Application: all applications

wordpress

Health checks
Container wordpress does not have health checks to ensure your Application is running correctly. Add health checks

Name
wordpress

Update strategy
RollingUpdate

Namespace
NS wordpress-test

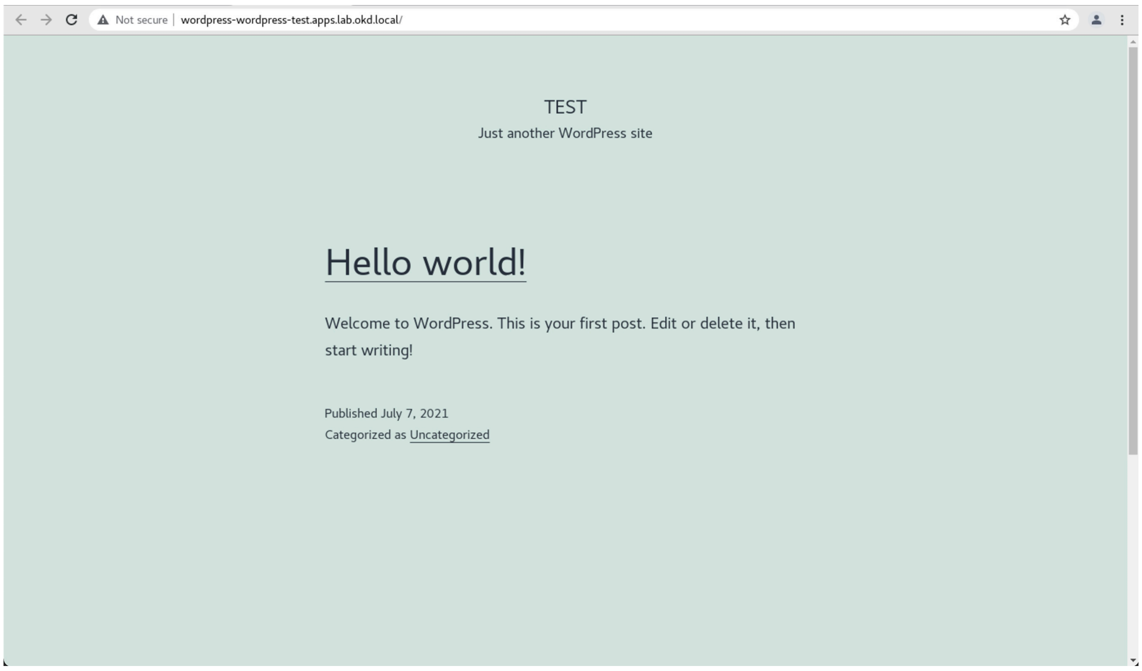
Max unavailable
25% of 2 pods

Labels Edit
app=wordpress
app.kubernetes.io/co... =wordp...

Max surge
25% greater than 2 pods

Progress deadline seconds

Figure 7: "wordpress-test" project topology tab in the OKD console


Not secure | wordpress-wordpress-test.apps.lab.okd.local/

TEST
Just another WordPress site

Hello world!

Welcome to WordPress. This is your first post. Edit or delete it, then start writing!

Published July 7, 2021
Categorized as Uncategorized

Figure 8: "wordpress" frontend container externally access from the browser

7. Conclusions

This project can take an important part in the start of a technological revolution within the IIT. The project shows clear advantages of implementing an OpenShift cluster for educational purposes opening the possibility of creating a more reliable and stronger cluster with physical servers available from the entire campus or even from VPN access. We get the following conclusions after the development of the project:

- Implementing an on-premises Kubernetes environment has proven to be useful for the IIT while being able to manage the environment based on the priorities of the organization, adding more features or scaling up the infrastructure without involving third-party companies.
- After analyzing the different PaaS solution, OpenShift proves to be the best alternative for an education-based organization thanks to its easy-to-learn web console that provides an easier learning curve so that students can get familiar with Kubernetes and its different tools and features while working on the projects from different classes of the ITM department.
- The cluster allows the deployment of applications taking advantage of some of the benefits of a Kubernetes platform. This document also goes over the installation of the cluster, providing all the necessary information to replicate the cluster in a different environment like in physical servers
- Developing this project allowed me to get comfortable with managing Linux Machines in a virtual environment, as well as getting familiar with Kubernetes and the web console of OpenShift. I got to see how easy and user friendly it is to deploy applications in an OpenShift Container Platform compared to plain Kubernetes.

This project didn't test one of the main benefits of OpenShift, which is the enterprise-level support RedHat offers to organizations that want to implement an OpenShift cluster. This is because we used the open-source version of OpenShift to minimize the expenses of the project. However, in the future improvements proposed in the next section, this feature could take an important role in a campus wide cluster.

7.1. Future development

This project marks the start of the development of a much bigger project to provide Kubernetes services to an organization like the IIT. In this document, the development of a virtual Kubernetes environment with OpenShift is described. However, there are multiple steps to follow next to achieve the main goal of creating a campus wide OpenShift cluster.

- Create a cluster with the physical servers in the ITM lab and test its accessibility from within the lab by deploying applications and exposing them to outside the cluster.
- Create a cluster with physical servers provided by the IIT, having access to the DNS and DHCP servers of the campus to configure them so that the cluster is accessible campus-wide. After deploying the cluster, configure the user management to make it accessible for professors and students with different permissions based on the role of the users and use the cluster for educational purposes in some of the classes of the IIT.
- Provide control of the cluster over the VPN of the university, so that students can access the cluster from outside the campus.

8. References

[1] Chapter 1. Introducing Red Hat CodeReady Containers Red Hat CodeReady Containers 1.28. (n.d.). Red Hat Customer Portal. Retrieved July 2, 2021, from https://access.redhat.com/documentation/en-us/red_hat_codeready_containers/1.28/html/getting_started_guide/introducing-codeready-containers_gsg

[2] Chapter 2. Installation Red Hat CodeReady Containers 1.28. (n.d.). Red Hat Customer Portal. Retrieved July 2, 2021, from https://access.redhat.com/documentation/en-

us/red_hat_codeready_containers/1.28/html/getting_started_guide/installation_gsg

[3] OKD - The Community Distribution of Kubernetes that powers Red Hat OpenShift. (n.d.). OKD. Retrieved July 10, 2021, from https://www.okd.io/

[4] Proxmox VE - Virtualization Management Platform. (n.d.). ProxMox VE. Retrieved July 5, 2021, from https://proxmox.com/en/proxmox-ve

[5] GitHub - openshift/okd: The self-managing, auto-upgrading, Kubernetes distribution for everyone. (n.d.). GitHub. Retrieved June 20, 2021, from https://github.com/openshift/okd/

[6] Installing a cluster on any platform - Installing on any platform | Installing | OKD 4. (n.d.). OpenShift. Retrieved July 12, 2021, from https://docs.okd.io/latest/installing/installing_platform_agnostic/installing-platform-agnostic.html#prerequisites

[7] Welcome | About | OKD 4. (n.d.). OpenShift. Retrieved June 2, 2021, from https://docs.okd.io/latest/welcome/index.html

[8] Hidalgo García, J. J. H. G., Hajek, J. H., & Illinois Institute of Technology. (2019, July). Analysis of Microservices software architecture on Kubernetes: monolithic migration, design choices, best practices and applications. Jaime Joseba Hidalgo García.

[9] Robinson, C. (2020, July 29). Guide: Installing an OKD 4.5 Cluster - ITNEXT. Medium. https://itnext.io/guide-installing-an-okd-4-5-cluster-508a2631cbee

9. Appendices

9.1. APPENDIX A: OKD INSTALLATION AND OTHER NETWORK SERVICES CONFIGURATION SCRIPTS

OPENSHIFT INSTALL CONFIGURATION FILE: install-config.yaml

```
apiVersion: v1
baseDomain: okd.local
metadata:
  name: lab

compute:
- hyperthreading: Enabled
  name: worker
  replicas: 0

controlPlane:
  hyperthreading: Enabled
  name: master
  replicas: 3

networking:
  clusterNetwork:
  - cidr: 10.128.0.0/14
    hostPrefix: 23
  networkType: OpenShiftSDN
  serviceNetwork:
  - 172.30.0.0/16

platform:
  none: {}

fips: false

pullSecret:
'{"auths":{"cloud.openshift.com":{"auth":"b3BlbnNoaWZ0LXJlbGVhc2UtZGV2
K29jbV9hY2Nlc3NfZDllOWQzMThiN2M3NGYyZGE4MDViZjRjYWVkY2Q1ZTg6R0VYTkFNSE
tOUzFaOVVCN0JXSUdKQk1OM0ZKMkNIMVBWN05GMlRZQkc0NzdWMUZINTRKM1Y4TlRJWktI
R0ZSQQ==","email":"uradadafonte@hawk.iit.edu"},"quay.io":{"auth":"b3Bl
bnNoaWZ0LXJlbGVhc2UtZGV2K29jbV9hY2Nlc3NfZDllOWQzMThiN2M3NGYyZGE4MDViZj
RjYWVkY2Q1ZTg6R0VYTkFNSEtOUzFaOVVCN0JXSUdKQk1OM0ZKMkNIMVBWN05GMlRZQkc0
NzdWMUZINTRKM1Y4TlRJWktIR0ZSQQ==","email":"uradadafonte@hawk.iit.edu"}
,"registry.connect.redhat.com":{"auth":"fHVoYy1wb29sLTRjMzZmOGMxLThiYT
ctNDc0Ni1iODEyLThmYmFhYzZhYTkyYzpleUpoYkdjaU9pSlNVelV4TWlKOS5leUp6ZFdJ
aU9pSTBPVFpoWVROallqTXlObVUwT1RRMVlURTFZbVl4WXpBNE9ESTJPREF5WkNKOS5uSV
laUHh0Z05GbE9hNDdFSWxZeGd5QVFaSXB1M2llWlpJMWE0dHJWaU5feVJhNjJHYWUwQTJJ
aUVQZEFWUjJwLXdhRFQ2VE5IdWhqOThYeFAxbTh5cE1WaWl4SzNsa1JBcGtCSGN1NzV0Rn
VpdFIwRnE5cHU4bk1Qdk11N1R6NVBvbTVod0gzajFJSmVlUTM2NUxxM2x6M0kxNGVfSGZz
UER4ajlNLUdDZ1dycWVQM2VRejZtdUE1cGZGZGpodlU0S1dkbWswOFo2M0JvSDg0aG1XUX
k1YzlqOVhadGhSXzBnSVdlRFNjc2ZkU0FQSFVtMnVvVHNrcVFDZVdkRy1GVjVWb29sWUpf
TUJRN0JFTHBkSXBwbWdjTk92VkRfUmxUTy1lN2N1M3JNcDZmS1FMcG9USkFhVGlUVmRXYW
VieExvYWNCUnFxb2tqekRkaW1CR2RDWENjVW1sTGxzTnZrYnRBMmFkUUF0Yl9qM3BVTjBf
SHI3ckpLMWxFQWJtX2FhbTlDdlg2WWg5QzVIRElUMHRZdmdmcUh6VmY5T1VUZ1piVFFySU
YwZFdwc0VveWZTWDdIZTJzaF9pZ01LZzVOaTlHcC0tYUc3WGpJT2xtaElxN0RfU3VVSE5v
cm9RV3gwWmtpWEZhUWVuaF8xTXA5UFhkTVRTNThlc3VWODlQTkpySWZURHU0dmJzeFZmUn
RxS0t1bkF6V0pHR3VLSjNuaDNYSldZbE9ISl9VY2h2XzlvVTAtbTZ4bHh3WDdsY1llZWpQ
MUlscjFsTWlHSjBKNlJ1TVEzZWpFOUhIMm1vNTlyUEJuS3NaUU9ROVo0SGZMcm95Z2s3QX
JrN3hDaWZpSk83U0J2cU5QclFJZjVFRFhZMzRyUWFKY0lwMDVfV2ozX0hIa21HdldPTQ==
","email":"uradadafonte@hawk.iit.edu"},"registry.redhat.io":{"auth":"f
HVoYy1wb29sLTRjMzZmOGMxLThiYTctNDc0Ni1iODEyLThmYmFhYzZhYTkyYzpleUpoYkd
jaU9pSlNVelV4TWlKOS5leUp6ZFdJaU9pSTBPVFpoWVROallqTXlObVUwT1RRMVlURTFZb
Vl4WXpBNE9ESTJPREF5WkNKOS5uSVlaUHh0Z05GbE9hNDdFSWxZeGd5QVFaSXB1M2llWlp
JMWE0dHJWaU5feVJhNjJHYWUwQTJJaUVQZEFWUjJwLXdhRFQ2VE5IdWhqOThYeFAxbTh5c
E1WaWl4SzNsa1JBcGtCSGN1NzV0RnVpdFIwRnE5cHU4bk1Qdk11N1R6NVBvbTVod0gzajF
JSmVlUTM2NUxxM2x6M0kxNGVfSGZzUER4ajlNLUdDZ1dycWVQM2VRejZtdUE1cGZGZGpod
lU0S1dkbWswOFo2M0JvSDg0aG1XUXk1YzlqOVhadGhSXzBnSVdlRFNjc2ZkU0FQSFVtMnV
vVHNrcVFDZVdkRy1GVjVWb29sWUpfTUJRN0JFTHBkSXBwbWdjTk92VkRfUmxUTy1lN2N1M
3JNcDZmS1FMcG9USkFhVGlUVmRXYWVieExvYWNCUnFxb2tqekRkaW1CR2RDWENjVW1sTGx
zTnZrYnRBMmFkUUF0Yl9qM3BVTjBfSHI3ckpLMWxFQWJtX2FhbTlDdlg2WWg5QzVIRElUM
HRZdmdmcUh6VmY5T1VUZ1piVFFySUYwZFdwc0VveWZTWDdIZTJzaF9pZ01LZzVOaTlHcC0
tYUc3WGpJT2xtaElxN0RfU3VVSE5vcm9RV3gwWmtpWEZhUWVuaF8xTXA5UFhkTVRTNThlc
3VWODlQTkpySWZURHU0dmJzeFZmUnRxS0t1bkF6V0pHR3VLSjNuaDNYSldZbE9ISl9VY2h
2XzlvVTAtbTZ4bHh3WDdsY1llZWpQMUlscjFsTWlHSjBKNlJ1TVEzZWpFOUhIMm1vNTlyU
EJuS3NaUU9ROVo0SGZMcm95Z2s3QXJrN3hDaWZpSk83U0J2cU5QclFJZjVFRFhZMzRyUWF
KY0lwMDVfV2ozX0hIa21HdldPTQ==","email":"uradadafonte@hawk.iit.edu"}}}
'
sshKey: 'ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABgQCvv2fK/+Qjsf0u8XOTDBiq2WMKgZCg/sQ0tRDsnD
DKEQIyMSp33rQTo1G3iUXmZ6bqhDiCcFaTlq+ezSQDMPO//ORet74cDGlLCy5Giyqx4q6a
pNKfPU4oHLgWRkeS0m3Dn1HmB5TpkyJMzCTWoLZOOYdS5FkCy5eOoOjOp6z2lnzXudieJJ
XgJn3nX8Usf0BrN8h5Tas7vxAcvxin0Y+S8ztJCSaq4w7pXlJStyDVikQxzTMX/JSFIUkQ
BonrNt2Wp0kIZHpe1GPFGNzVYNIcr2phQ7VZDsrAQUYTxAOSLw2kz3AOOQ2Ows9GVjgsVr
zhh0oTwsdP02oO0vXDsNUzvJAYwfsLpL2ZCxNmyOLtc/aCj0eQuCXQazhyuAhO+TyOanj/
6CAjGDFvzT5iAYgOOYOGArwoE6dY3XJOKecGJPhgfT3+CW77ME40Q26H7wMTpqMfbsRdiZ
/cg+qnPkeWjwRFA9YB0qTOakvPgu86v7T6DvBnwd2ncSE6/J1UzU8= urada@okd4-
services'
```

DNS CONFIGURATION IN OKD4-SERVICES: named.conf

```
//
// named.conf
```

```
//
// Provided by Red Hat bind package to configure the ISC BIND named(8)
DNS
// server as a caching only nameserver (as a localhost DNS resolver
only).
//
// See /usr/share/doc/bind*/sample/ for example named configuration
files.
//
// See the BIND Administrator's Reference Manual (ARM) for details
about the
// configuration located in /usr/share/doc/bind-{version}/Bv9ARM.html

options {
      listen-on port 53 { 127.0.0.1; 10.0.0.210; };
#     listen-on-v6 port 53 { ::1; };
      directory   "/var/named";
      dump-file   "/var/named/data/cache_dump.db";
      statistics-file "/var/named/data/named_stats.txt";
      memstatistics-file "/var/named/data/named_mem_stats.txt";
      recursing-file  "/var/named/data/named.recursing";
      secroots-file   "/var/named/data/named.secroots";
      allow-query     { localhost; 10.0.0.0/16; };

      /*
       - If you are building an AUTHORITATIVE DNS server, do NOT
enable recursion.
       - If you are building a RECURSIVE (caching) DNS server, you
need to enable
         recursion.
       - If your recursive DNS server has a public IP address, you
MUST enable access
         control to limit queries to your legitimate users. Failing to
do so will
         cause your server to become part of large scale DNS
amplification
         attacks. Implementing BCP38 within your network would greatly
         reduce such attack surface
      */
      recursion yes;

      forwarders {
                10.131.110.44;
                10.131.110.45;
            8.8.8.8;
            8.8.4.4;
        };

      dnssec-enable yes;
      dnssec-validation no;

      /* Path to ISC DLV key */
      bindkeys-file "/etc/named.root.key";

      managed-keys-directory "/var/named/dynamic";

      pid-file "/run/named/named.pid";
      session-keyfile "/run/named/session.key";
};

logging {
        channel default_debug {
                file "data/named.run";
                severity dynamic;
        };
};

zone "." IN {
      type hint;
      file "named.ca";
};

include "/etc/named.rfc1912.zones";
include "/etc/named.root.key";
include "/etc/named/named.conf.local";
```

DNS CONFIGURATION IN OKD4-SERVICES: named.conf.local

```
zone "okd.local" {
    type master;
    file "/etc/named/zones/db.okd.local"; # zone file path
};

zone "0.0.10.in-addr.arpa" {
    type master;
    file "/etc/named/zones/db.10.0.0";  # 10.0.0.0/16 subnet
};
```

DNS ZONES CONFIGURATION IN OKD4-SERVICES: db.10.0.0

```
$TTL    604800
@       IN      SOA     okd4-services.okd.local. admin.okd.local. (
                  6     ; Serial
             604800     ; Refresh
```

```
                86400     ; Retry
              2419200     ; Expire
               604800     ; Negative Cache TTL
)

; name servers - NS records
    IN      NS      okd4-services.okd.local.

; name servers - PTR records
210    IN    PTR    okd4-services.okd.local.

; OpenShift Container Platform Cluster - PTR records
200    IN    PTR    okd4-bootstrap.lab.okd.local.
201    IN    PTR    okd4-control-plane-1.lab.okd.local.
202    IN    PTR    okd4-control-plane-2.lab.okd.local.
203    IN    PTR    okd4-control-plane-3.lab.okd.local.
204    IN    PTR    okd4-compute-1.lab.okd.local.
205    IN    PTR    okd4-compute-2.lab.okd.local.
210    IN    PTR    api.lab.okd.local.
210    IN    PTR    api-int.lab.okd.local.
```

DNS ZONES CONFIGURATION IN OKD4-SERVICES: db.okd.local

```
$TTL    604800
@       IN      SOA     okd4-services.okd.local. admin.okd.local. (
                    1     ; Serial
               604800     ; Refresh
                86400     ; Retry
              2419200     ; Expire
               604800     ; Negative Cache TTL
)

; name servers - NS records
    IN      NS      okd4-services

; name servers - A records
okd4-services.okd.local.          IN      A       10.0.0.210

; OpenShift Container Platform Cluster - A records
okd4-bootstrap.lab.okd.local.        IN      A      10.0.0.200
okd4-control-plane-1.lab.okd.local.        IN      A      10.0.0.201
okd4-control-plane-2.lab.okd.local.         IN      A      10.0.0.202
okd4-control-plane-3.lab.okd.local.         IN      A      10.0.0.203
okd4-compute-1.lab.okd.local.        IN      A      10.0.0.204
okd4-compute-2.lab.okd.local.        IN      A      10.0.0.205

; OpenShift internal cluster IPs - A records
```

```
api.lab.okd.local.    IN    A    10.0.0.210
api-int.lab.okd.local.    IN    A    10.0.0.210
*.apps.lab.okd.local.    IN    A    10.0.0.210
etcd-0.lab.okd.local.    IN    A     10.0.0.201
etcd-1.lab.okd.local.    IN    A     10.0.0.202
etcd-2.lab.okd.local.    IN    A    10.0.0.203
console-openshift-console.apps.lab.okd.local.     IN     A
10.0.0.210
oauth-openshift.apps.lab.okd.local.     IN     A     10.0.0.210

; OpenShift internal cluster IPs - SRV records
_etcd-server-ssl._tcp.lab.okd.local.    86400     IN    SRV     0
10    2380    etcd-0.lab
_etcd-server-ssl._tcp.lab.okd.local.    86400     IN    SRV     0
10    2380    etcd-1.lab
_etcd-server-ssl._tcp.lab.okd.local.    86400     IN    SRV     0
10    2380    etcd-2.lab
```

LOAD BALANCING CONFIGURATION IN OKD4-SERVICES: haproxy.cfg

```
# Global settings
#---------------------------------------------------------------------
global
    maxconn     20000
    log         /dev/log local0 info
    chroot      /var/lib/haproxy
    pidfile     /var/run/haproxy.pid
    user        haproxy
    group       haproxy
    daemon

    # turn on stats unix socket
    stats socket /var/lib/haproxy/stats

#---------------------------------------------------------------------
# common defaults that all the 'listen' and 'backend' sections will
# use if not designated in their block
#---------------------------------------------------------------------
defaults
    mode                    http
    log                     global
    option                  httplog
    option                  dontlognull
    option http-server-close
    option forwardfor       except 127.0.0.0/8
    option                  redispatch
    retries                 3
    timeout http-request    10s
    timeout queue           1m
    timeout connect         10s
    timeout client          300s
    timeout server          300s
    timeout http-keep-alive 10s
    timeout check           10s
    maxconn                 20000

listen stats
    bind :9000
    mode http
    stats enable
    stats uri /

frontend okd4_k8s_api_fe
    bind :6443
    default_backend okd4_k8s_api_be
    mode tcp
    option tcplog

backend okd4_k8s_api_be
    balance source
    mode tcp
    server      okd4-bootstrap 10.0.0.200:6443 check
    server      okd4-control-plane-1 10.0.0.201:6443 check
    server      okd4-control-plane-2 10.0.0.202:6443 check
    server      okd4-control-plane-3 10.0.0.203:6443 check

frontend okd4_machine_config_server_fe
    bind :22623
    default_backend okd4_machine_config_server_be
    mode tcp
    option tcplog

backend okd4_machine_config_server_be
    balance source
    mode tcp
    server      okd4-bootstrap 10.0.0.200:22623 check
    server      okd4-control-plane-1 10.0.0.201:22623 check
    server      okd4-control-plane-2 10.0.0.202:22623 check
    server      okd4-control-plane-3 10.0.0.203:22623 check

frontend okd4_http_ingress_traffic_fe
    bind :80
    default_backend okd4_http_ingress_traffic_be
    mode tcp
    option tcplog
```

```
backend okd4_http_ingress_traffic_be
    balance source
    mode tcp
    server      okd4-compute-1 10.0.0.204:80 check
    server      okd4-compute-2 10.0.0.205:80 check

frontend okd4_https_ingress_traffic_fe
    bind *:443
    default_backend okd4_https_ingress_traffic_be
    mode tcp
    option tcplog

backend okd4_https_ingress_traffic_be
    balance source
    mode tcp
    server      okd4-compute-1 10.0.0.204:443 check
    server      okd4-compute-2 10.0.0.205:443 check

frontend okd4_etcd
    bind :2379
    default_backend okd4_etcd
    mode tcp
    option tcplog

backend okd4_etcd
    balance source
    mode tcp
    server      okd4-bootstrap 10.0.0.200:2379 check
    server      okd4-control-plane-1 10.0.0.201:2379 check
    server      okd4-control-plane-2 10.0.0.202:2379 check
    server      okd4-control-plane-3 10.0.0.203:2379 check


frontend okd4_console
    bind :8443
    default_backend okd4_console
    mode tcp
    option tcplog

backend okd4_console
    balance source
    mode tcp
    server      okd4-control-plane-1 10.0.0.201:8443 check
```

REGISTRY CONFIGURATION: registry_pv.yaml

```
apiVersion: v1
```

```
kind: PersistentVolume
metadata:
  name: registry-pv
spec:
  capacity:
    storage: 100Gi
  accessModes:
    - ReadWriteMany
  persistentVolumeReclaimPolicy: Retain
  nfs:
    path: /var/nfsshare/registry
    server: 10.0.0.210
```

HTPASSWD PROVIDER: htpasswd_provider.yaml

```
apiVersion: config.openshift.io/v1
kind: OAuth
metadata:
  name: cluster
spec:
  identityProviders:
  - name: htpasswd_provider
    mappingMethod: claim
    type: HTPasswd
    htpasswd:
      fileData:
        name: htpass-secret
```

HTPASSWD USERS: users.htpasswd

```
urada:$2y$05$SZpNVzXHzYVKFv3lwd5UvusyNB11UM219aSR9KqWK8miUUVzjv1Y.
```