CENG365 Distributed Systems

Assignment-I

Ayben GÜLNAR-191180041

DECEMBER 2022

Contents

1.INTRODUCTION 3
2.WHAT IS CONTAİNERİZATİON? 4
3.WHAT IS DOCKER ORCHESTRATİON? 4
4.WHAT IS KUBERNETES? 5
 4.1 Going back in time 5
 4.2 Why you need Kubernetes and what it can do? 5
 4.3 What Kubernetes is Not ? 6
 4.4 Kubernetes Cluster Architecture 7
 4.5 Kubernetes Components 10
 4.6 Kubernetes Workloads 11
 4.7 Kubernetes Configuration 11
 4.8 Advantages and Disadvantages 11
5. KUBERNETES VS SWARM COMPARİSON 12
6.MOST COMMON KUBERNETES COMMANDS 13
7. CONCLUSİON 19
8.KAYNAKÇA 20

Figures

Figure 4.4.1 7
Figure 4.4.2 8
Figure 4.4.3 9

1.INTRODUCTION

Kubernetes is an open-source system for automating the deployment, scaling, and management of containerized applications. It groups containers that make up an application into logical units for easy management and discovery. Kubernetes is often used with Docker, a technology that allows applications to be packaged into containers that can be run on any compatible system.

Kubernetes is a powerful tool for deploying and managing containerized applications at scale. It provides a number of features that make it easier to deploy, scale, and manage applications in a clustered environment, including:

Deployment and scaling: Kubernetes can automatically deploy and scale your application based on specified criteria, such as CPU or memory usage. This makes it easier to ensure that your application has the resources it needs to handle a varying workload.

Service discovery and load balancing: Kubernetes can automatically expose your application's services and distribute incoming traffic across multiple replicas of a service. This makes it easier to build resilient, highly available applications.

Configuration management: Kubernetes provides a unified way to manage the configuration of your application, including environment variables, secrets, and other settings. This makes it easier to manage your application's dependencies and ensure that it is running with the correct configuration.

Storage orchestration: Kubernetes can automatically mount and manage storage volumes for your application, making it easier to manage the persistent data that your application relies on.

Kubernetes is a system for managing and deploying containerized applications. It allows for the use of declarative configuration and automation, and has a large ecosystem of services, support, and tools. The name Kubernetes comes from Greek and means "helmsman" or "pilot". The project was open-sourced by Google in 2014, and is based on their experience running production workloads at scale combined with best practices from the community [1].

In this research, I will first discuss what Kubernetes is, where it is used, and why we need it. Then, we will discuss commonly used commands.

2.WHAT IS CONTAİNERİZATİON?

Containerization is the process of packaging software code with just the necessary operating system libraries and dependencies so it can be run in a lightweight, portable container. This makes it easier and faster for developers to create and deploy applications, and also makes them more secure. In traditional methods, transferring code from one computing environment to another often causes bugs and errors. Containerization solves this problem by bundling the code with the necessary configuration files and dependencies, so it can run on any platform without issues. This makes the resulting container portable and able to run on any infrastructure.

The use of containerization has been around for decades, but the popularity of the open source Docker Engine has led to a widespread adoption of this technology. Many organizations are using containerization to create new applications and modernize existing ones for the cloud. Containers are often referred to as "lightweight" because they share the machine's operating system kernel and do not require an individual operating system for each application. This makes them smaller and faster to start up, allowing more containers to run on the same compute capacity as a single virtual machine. Containerization also allows applications to be "written once and run anywhere," which speeds up development and prevents cloud vendor lock-in. This portability also offers benefits such as fault isolation, ease of management, and simplified security [2].

3.WHAT IS DOCKER ORCHESTRATİON?

Docker orchestration refers to the practice of managing large numbers of Docker containers, including tasks such as provisioning containers, scaling up and down, and managing networking and load balancing. There are several options for Docker orchestration, including Kubernetes, Docker Swarm, and cloud container platforms like Amazon Elastic Container Service (EKS) and Azure Container Instances (ACI). Kubernetes is the most powerful and popular option, but it has a steep learning curve. Docker Swarm and other native Docker tools are easier to use but less powerful. Cloud container platforms provide basic orchestration capabilities and are simpler to use than Kubernetes [3].

Container orchestration is a set of practices and technologies for managing large numbers of containers in a production environment. Containerized applications, especially those that use microservices, can require managing hundreds or thousands of containers. This can be complex and time-consuming to manage manually, but container orchestration makes it easier to automate much of the work. This is particularly useful for DevOps teams, which value speed and agility. Container orchestration allows them to operate more efficiently and manage the operational complexity of large-scale containerized systems.

4.WHAT IS KUBERNETES?

4.1 Going back in time

Traditional development era: In the early days of computing, applications were run on physical servers. This led to problems with resource allocation as it was difficult to define boundaries for the resources that each application could use. To solve this problem, organizations would run each application on a separate physical server. However, this was not scalable and resulted in underutilized resources and high maintenance costs.

Virtualized deployment era: Virtualization was introduced as a solution to the problems with running applications on physical servers. It allows multiple virtual machines (VMs) to run on a single physical server, providing isolation between applications and allowing for better resource utilization. Virtualization also allows for improved scalability, as applications can be easily added or updated, and reduces hardware costs. Each VM is a full machine running its own operating system on top of virtualized hardware.

Container Deployment era: Containers have become popular in recent years due to their many benefits over virtual machines (VMs). They are lightweight and portable across different operating systems and clouds, making it easier to create and deploy applications. Containers also provide a higher level of abstraction, allowing for application-centric management and improved resource utilization. Some other benefits of using containers include increased agility in application creation and deployment, improved continuous development and integration, and separation of concerns between development and operations teams. They also provide improved observability, environmental consistency, and portability. Additionally, containers support the use of microservices, allowing for greater flexibility and resource isolation [1].

4.2 Why you need Kubernetes and what it can do?

Kubernetes (k8s) is a de-facto standard project in the container orchestration field that is developed by Google as open-source, where we can run our containerized applications.

Service Discovery & Load Balancing: Kubernetes provides DNS definitions for containers running inside so that we can access them. At the same time, it also takes responsibility for load balancing in order to distribute incoming load evenly among the running applications.

Dynamic Storage Management: It allows us to dynamically mount storage for applications that need to persist data. This storage can be local storage on the node we are working on or storage that we can use with a cloud provider.

Automated Rollout & Rollback: You can specify how many instances you want your application to run. This way, when you make a new deployment, Kubernetes automates the process for you.

Healthcheck: Kubernetes provides us with the ability to define healthcheck rules for your applications, so that applications that do not comply with those rules will not work or will be restarted in case of errors during runtime.

Secret & Configuration Management: You can store your application configurations and sensitive information required for your application on Kubernetes [4].

4.3 What Kubernetes is Not ?

Kubernetes is not a traditional PaaS system that provides all services in one package. Instead, it operates at the container level and offers some features commonly found in PaaS systems, such as deployment, scaling, and load balancing. However, these features are not mandatory and can be replaced with other solutions. Kubernetes provides the necessary components for creating a platform for developers, but allows users to choose and customize the services they need.

Kubernetes does not impose limitations on the types of applications it can support. It is designed to handle a wide range of workloads, including stateless, stateful, and data-processing workloads. As long as an application can run in a container, it should work well with Kubernetes.

Kubernetes does not handle the deployment of source code or the building of applications. The CI/CD workflow is determined by the organization's culture, preferences, and technical requirements.

Kubernetes does not provide application-level services such as middleware, data-processing frameworks, databases, caches, or cluster storage systems. These components can be run on Kubernetes or accessed by applications through portable mechanisms such as the Open Service Broker.

Kubernetes does not dictate the solutions for logging, monitoring, or alerting. It provides some integrations as examples and mechanisms to collect and export metrics.

Kubernetes does not mandate the use of a specific configuration language or system. It offers a declarative API that can be used with any declarative specification.

Kubernetes does not provide comprehensive machine configuration, maintenance, management, or self-healing systems.

Furthermore, Kubernetes is not just an orchestration system. It eliminates the need for orchestration by providing a set of independent, composable control processes that continuously drive the current state towards the desired state. It does not matter how you get from one point to another, and centralized control is not required. This makes Kubernetes easier to use and more powerful, robust, resilient, and extensible [1].

4.4 Kubernetes Cluster Architecture

Kubernetes Architecture has the following main components:

- Master nodes
- Worker/Slave nodes
- Distributed key-value store(etcd)

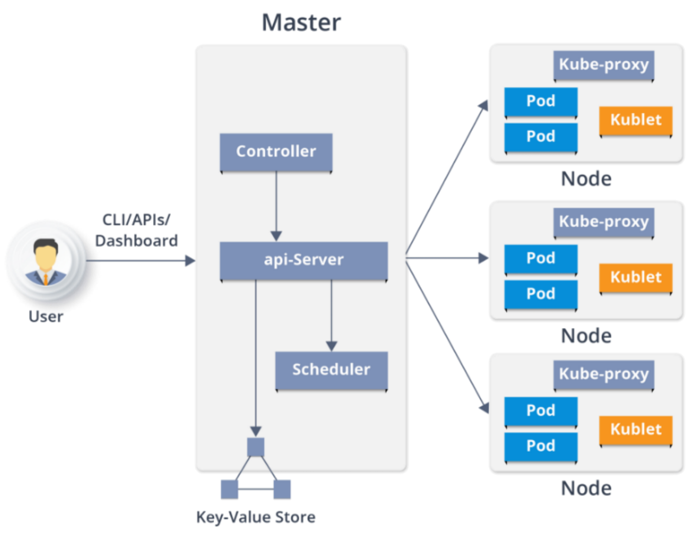

Figure 4.4.1

Master Node:

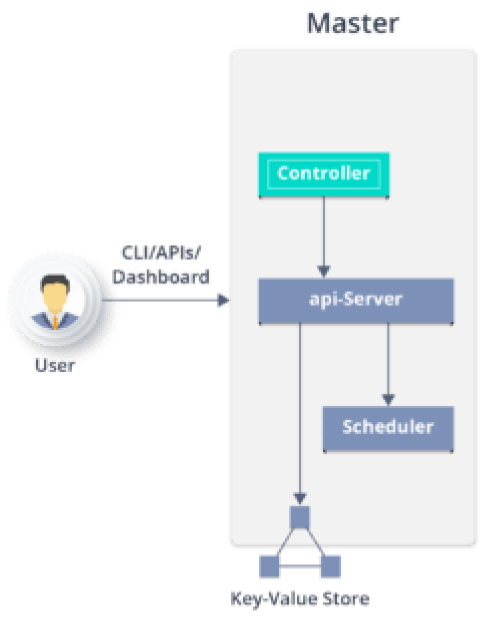

Figure 4.4.2

The Kubernetes master node is the entry point for all administrative tasks related to managing the Kubernetes cluster. There can be multiple master nodes in the cluster to ensure fault tolerance. Having multiple master nodes enables the system to operate in High Availability mode, where one of them serves as the main node for performing tasks.

The Kubernetes master node uses etcd to manage the cluster state, and all master nodes connect to etcd.As you can see in the diagram above it consists of 4 components:

The Kubernetes master node has several components that perform different tasks. The API server is the main component that handles all administrative tasks through the API. REST commands are sent to the API server, which validates and processes the requests. The resulting state of the cluster is stored in a distributed key-value store.

The scheduler is responsible for scheduling tasks to slave nodes. It stores information about the resource usage of each slave node and schedules work in the form of Pods and Services. The scheduler takes into account factors such as the quality of service requirements, data locality, affinity, and anti-affinity before scheduling a task.

The controller manager, also known as controllers, is a daemon that regulates the Kubernetes cluster. It manages non-terminating control loops and performs lifecycle functions such as namespace creation and lifecycle, event garbage collection, terminated-pod garbage collection, cascading deletion garbage collection, and node garbage collection. A controller watches the desired state of the objects it manages and checks their current state through the API server. If the current state does not match the desired state, the controller takes corrective steps to ensure that the current state is the same as the desired state. Etcd is a distributed key-value store that stores the cluster state. It can be part of the Kubernetes Master or configured externally. etcd is written in Go and, in addition to storing the cluster state (using the Raft Consensus Algorithm), it also stores configuration details such as subnets, ConfigMaps, and Secrets. Raft is a consensus algorithm that defines three roles (Leader, Follower, and Candidate) and achieves consensus through an elected leader.

Worker Node (formerly minions)

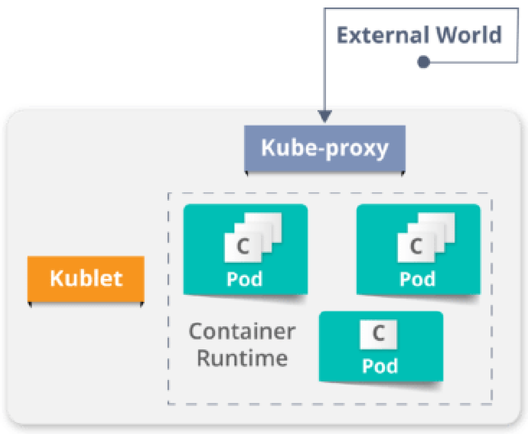

Figure 4.4.3

A Kubernetes worker node is a physical server or virtual machine that runs applications using Pods, which are controlled by the master node. Pods are scheduled on physical servers (worker/slave nodes), and nodes are accessed from the external world to access the applications.

A container runtime is required on the worker node to run and manage the lifecycle of a container. Docker is often referred to as a container runtime, but it is actually a platform that uses containers as a container runtime.

The Kubelet is an agent that communicates with the master node and runs on nodes or worker nodes. It receives Pod specifications through the API server and ensures that the containers associated with the Pod are running and healthy.

The Kube-proxy runs on each node to handle individual host sub-netting and make services available to external parties. It acts as a network proxy and load balancer for a service on a single worker node and manages the network routing for TCP and UDP packets. It listens to the API server for the creation and deletion of Service endpoint and sets up the routes accordingly. A Kubernetes Pod is a group of one or more containers that are treated as a single logical unit. Pods run on nodes and share the same IP address and storage. Pods do not have to run on the same machine, as containers can span multiple machines. A single node can run multiple pods. Pods are used to logically group containers that belong together and share resources [5].

4.5 Kubernetes Components

When you deploy Kubernetes, you get a cluster. A Kubernetes cluster is a group of worker machines called nodes that run containerized applications. Every cluster has at least one worker node.

The worker nodes host the Pods that make up the application workload. The control plane manages the worker nodes and the Pods in the cluster. In production environments, the control plane typically runs on multiple computers, and a cluster typically has multiple nodes to provide fault tolerance and high availability.

This document describes the different components that are required to have a complete and functional Kubernetes cluster [6].

The control plane components of a Kubernetes cluster make global decisions about the cluster, such as scheduling, and detect and respond to cluster events, such as starting new pods. Control plane components can run on any machine in the cluster, but for simplicity, they are typically all started on the same machine. This machine does not run user containers.

The API server is a component of the control plane that exposes the Kubernetes API. It is the frontend for the control plane. The main implementation of the Kubernetes API server is kube-apiserver, which is designed to scale horizontally by deploying more instances. Traffic can be balanced between multiple instances of kube-apiserver.

etcd is a consistent and highly-available key-value store that is used as the backing store for all cluster data. If the Kubernetes cluster uses etcd, make sure to have a backup plan for this data.

The kube-scheduler watches for newly created pods that have no assigned node and selects a node for them to run on. The scheduling decision takes into account factors such as resource requirements, hardware/software/policy constraints, affinity and anti-affinity specifications, data locality, and deadlines.

The kube-controller-manager runs controller processes. Each controller is a separate process, but they are all compiled into a single binary and run in a single process to reduce complexity. Some types of controllers include the node controller, which responds when nodes go down; the job controller, which creates pods to run one-off tasks represented by job objects; the endpoint slice controller, which populates endpoint slice objects; and the service account controller, which creates default service accounts for new namespaces.

4.6 Kubernetes Workloads

Kubernetes allows you to run applications by creating and managing a group of containers, called a workload. These workloads are managed by controllers, which ensure that the desired number and type of containers are running at all times. Kubernetes provides several built-in workload resources, such as Deployments and StatefulSets, as well as the ability to use third-party resources. Pods, which consist of one or more containers, are used to run the workloads [7].

4.7 Kubernetes Configuration

When creating configurations for your cluster, it's best to use the latest stable version of the API. You should also store your configuration files in a version control system so that you can easily roll back any changes if necessary. Additionally, it's recommended to use YAML instead of JSON for writing your configuration files, as YAML is generally considered to be more user-friendly. It's also a good idea to group related objects into a single file, as this can make it easier to manage your configuration. Furthermore, you can use the kubectl command to apply a directory of configuration files. To avoid unnecessary errors, it's best to avoid specifying default values in your configuration, and instead keep your configuration as simple and minimal as possible. Finally, you can add object descriptions in annotations to allow for better introspection of your configuration [8].

4.8 Advantages and Disadvantages

There are several benefits to using Kubernetes, including its self-healing abilities and the ability to manage and monitor multiple containers simultaneously. Kubernetes also has minimal performance overhead, and supports a wide range of applications. Additionally, it is an extensible platform that can be customized to meet the needs of different users. Overall, Kubernetes is a powerful tool for managing and deploying applications in a cluster environment.

While Kubernetes offers many benefits, it can also be complex and overwhelming for some users, especially those working on local development projects or simple applications. The constant innovation and updates to Kubernetes can also make it difficult for new users to navigate, and the learning curve can be steep. Additionally, debugging and troubleshooting issues with Kubernetes can require extensive training and experience. As a result, it may take time, effort, and resources for teams to fully take advantage of the benefits of Kubernetes [9].

5. KUBERNETES VS SWARM COMPARİSON

Before comparing, let's talk about swarm briefly:

Docker Swarm is an open-source container orchestration platform that enables users to manage and deploy clusters of Docker engines. A Swarm cluster consists of manager nodes, which are responsible for orchestrating and managing the cluster, and worker nodes, which execute tasks as directed by the manager nodes. Swarm mode is Docker's native support for orchestrating Swarm clusters, and is an important part of the Docker platform.

Kubernetes and Docker Swarm are both platforms for managing containers and scaling application deployment. However, there are some key differences between the two. Kubernetes is known for its efficiency and ability to handle complex configurations, making it well-suited for high-demand applications. In contrast, Docker Swarm is designed to be easy to use, making it a good option for simple, quickly-deployed applications. Overall, the choice between Kubernetes and Docker Swarm will depend on the specific needs and requirements of your project.

There are several differences between Kubernetes and Docker Swarm, including their installation and setup, scalability, load balancing, and high availability. Kubernetes can be more complex to install and configure, but offers all-in-one scaling based on traffic. In contrast, Docker Swarm is easier to install and offers quick scaling of groups on demand. Additionally, Docker Swarm has automatic load balancing, while Kubernetes does not, although an external load balancer can be easily integrated. Both Kubernetes and Docker Swarm offer high levels of availability, but Kubernetes is self-healing and offers intelligent scheduling and service replication.

The choice between Kubernetes and Docker Swarm ultimately depends on the specific needs and requirements of your organization. Kubernetes is widely adopted and supported by major cloud providers, and offers a high degree of customization and flexibility. However, it can be more complex to use and requires experienced teams to manage it effectively. In contrast, Docker Swarm is easier to use and can be a good choice for smaller workloads. It is also familiar to many users and is deployed with the Docker Engine, making it easy to get started with [10].

6.MOST COMMON KUBERNETES COMMANDS

Kubectl autocomplete

BASH

```
source <(kubectl completion bash) # set up autocomplete in bash into the
current shell, bash-completion package should be installed first.
echo "source <(kubectl completion bash)" >> ~/.bashrc # add autocomplete
permanently to your bash shell.
```

Set which Kubernetes cluster kubectl communicates with and modifies configuration information.

```
kubectl config view # Show Merged kubeconfig settings.

# use multiple kubeconfig files at the same time and view merged config
KUBECONFIG=~/.kube/config:~/.kube/kubconfig2

kubectl config view

# get the password for the e2e user
kubectl config view -o jsonpath='{.users[?(@.name == "e2e")].user.password}'

kubectl config view -o jsonpath='{.users[].name}'    # display the first user
kubectl config view -o jsonpath='{.users[*].name}'   # get a list of users
kubectl config get-contexts                          # display list of
contexts
kubectl config current-context                       # display the current-
context
kubectl config use-context my-cluster-name           # set the default context
to my-cluster-name

kubectl config set-cluster my-cluster-name           # set a cluster entry in
the kubeconfig

# configure the URL to a proxy server to use for requests made by this client
in the kubeconfig
kubectl config set-cluster my-cluster-name --proxy-url=my-proxy-url

# add a new user to your kubeconf that supports basic auth
kubectl config set-credentials kubeuser/foo.kubernetes.com --username=kubeuser
--password=kubepassword

# permanently save the namespace for all subsequent kubectl commands in that
context.
kubectl config set-context --current --namespace=ggckad-s2

# set a context utilizing a specific username and namespace.
kubectl config set-context gce --user=cluster-admin --namespace=foo \
  && kubectl config use-context gce

kubectl config unset users.foo                       # delete user foo
```

```
# short alias to set/show context/namespace (only works for bash and bash-
compatible shells, current context to be set before using kn to set namespace)
alias kx='f() { [ "$1" ] && kubectl config use-context $1 || kubectl config
current-context ; } ; f'
alias kn='f() { [ "$1" ] && kubectl config set-context --current --namespace
$1 || kubectl config view --minify | grep namespace | cut -d" " -f6 ; } ; f'
```

Kubernetes manifests can be defined in YAML or JSON. The file extension .yaml, .yml, and .json can be used.

```
kubectl apply -f ./my-manifest.yaml            # create resource(s)
kubectl apply -f ./my1.yaml -f ./my2.yaml      # create from multiple files
kubectl apply -f ./dir                         # create resource(s) in all
manifest files in dir
kubectl apply -f https://git.io/vPieo          # create resource(s) from url
kubectl create deployment nginx --image=nginx  # start a single instance of
nginx

# create a Job which prints "Hello World"
kubectl create job hello --image=busybox:1.28 -- echo "Hello World"

# create a CronJob that prints "Hello World" every minute
kubectl create cronjob hello --image=busybox:1.28   --schedule="*/1 * * * *" -
- echo "Hello World"

kubectl explain pods                           # get the documentation for pod
manifests

# Create multiple YAML objects from stdin
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: Pod
metadata:
  name: busybox-sleep
spec:
  containers:
  - name: busybox
    image: busybox:1.28
    args:
    - sleep
    - "1000000"
---
apiVersion: v1
kind: Pod
metadata:
  name: busybox-sleep-less
spec:
  containers:
  - name: busybox
    image: busybox:1.28
    args:
    - sleep
    - "1000"
EOF
```

```
# Create a secret with several keys
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: Secret
metadata:
  name: mysecret
type: Opaque
data:
  password: $(echo -n "s33msi4" | base64 -w0)
  username: $(echo -n "jane" | base64 -w0)
EOF
```

Viewing and finding resources

```
# Get commands with basic output
kubectl get services                          # List all services in the
namespace
kubectl get pods --all-namespaces             # List all pods in all
namespaces
kubectl get pods -o wide                      # List all pods in the current
namespace, with more details
kubectl get deployment my-dep                 # List a particular deployment
kubectl get pods                              # List all pods in the namespace
kubectl get pod my-pod -o yaml                # Get a pod's YAML

# Describe commands with verbose output
kubectl describe nodes my-node
kubectl describe pods my-pod

# List Services Sorted by Name
kubectl get services --sort-by=.metadata.name

# List pods Sorted by Restart Count
kubectl get pods --sort-by='.status.containerStatuses[0].restartCount'

# List PersistentVolumes sorted by capacity
kubectl get pv --sort-by=.spec.capacity.storage

# Get the version label of all pods with label app=cassandra
kubectl get pods --selector=app=cassandra -o \
  jsonpath='{.items[*].metadata.labels.version}'

# Retrieve the value of a key with dots, e.g. 'ca.crt'
kubectl get configmap myconfig \
  -o jsonpath='{.data.ca\.crt}'

# Retrieve a base64 encoded value with dashes instead of underscores.
kubectl get secret my-secret --template='{{index .data "key-name-with-
dashes"}}'

# Get all worker nodes (use a selector to exclude results that have a label
# named 'node-role.kubernetes.io/control-plane')
kubectl get node --selector='!node-role.kubernetes.io/control-plane'

# Get all running pods in the namespace
```

```
kubectl get pods --field-selector=status.phase=Running

# Get ExternalIPs of all nodes
kubectl get nodes -o
jsonpath='{.items[*].status.addresses[?(@.type=="ExternalIP")].address}'

# List Names of Pods that belong to Particular RC
# "jq" command useful for transformations that are too complex for jsonpath,
it can be found at https://stedolan.github.io/jq/
sel=${$(kubectl get rc my-rc --output=json | jq -j '.spec.selector |
to_entries | .[] | "\(.key)=\(.value),"')%?}
echo $(kubectl get pods --selector=$sel --
output=jsonpath={.items..metadata.name})

# Show labels for all pods (or any other Kubernetes object that supports
labelling)
kubectl get pods --show-labels

# Check which nodes are ready
JSONPATH='{range .items[*]}{@.metadata.name}:{range
@.status.conditions[*]}{@.type}={@.status};{end}{end}' \
 && kubectl get nodes -o jsonpath="$JSONPATH" | grep "Ready=True"

# Output decoded secrets without external tools
kubectl get secret my-secret -o go-template='{{range $k,$v := .data}}{{"###
"}}{{$k}}{{"\n"}}{{$v|base64decode}}{{"\n\n"}}{{end}}'

# List all Secrets currently in use by a pod
kubectl get pods -o json | jq
'.items[].spec.containers[].env[]?.valueFrom.secretKeyRef.name' | grep -v null
| sort | uniq

# List all containerIDs of initContainer of all pods
# Helpful when cleaning up stopped containers, while avoiding removal of
initContainers.
kubectl get pods --all-namespaces -o jsonpath='{range
.items[*].status.initContainerStatuses[*]}{.containerID}{"\n"}{end}' | cut -d/
-f3

# List Events sorted by timestamp
kubectl get events --sort-by=.metadata.creationTimestamp

# List all warning events
kubectl events --types=Warning

# Compares the current state of the cluster against the state that the cluster
would be in if the manifest was applied.
kubectl diff -f ./my-manifest.yaml

# Produce a period-delimited tree of all keys returned for nodes
# Helpful when locating a key within a complex nested JSON structure
kubectl get nodes -o json | jq -c 'paths|join(".")'

# Produce a period-delimited tree of all keys returned for pods, etc
kubectl get pods -o json | jq -c 'paths|join(".")'
```

```
# Produce ENV for all pods, assuming you have a default container for the
pods, default namespace and the `env` command is supported.
# Helpful when running any supported command across all pods, not just `env`
for pod in $(kubectl get po --output=jsonpath={.items..metadata.name}); do
echo $pod && kubectl exec -it $pod -- env; done

# Get a deployment's status subresource
kubectl get deployment nginx-deployment --subresource=status
```

Updating resources

```
kubectl set image deployment/frontend www=image:v2               # Rolling
update "www" containers of "frontend" deployment, updating the image
kubectl rollout history deployment/frontend                      # Check the
history of deployments including the revision
kubectl rollout undo deployment/frontend                         # Rollback to
the previous deployment
kubectl rollout undo deployment/frontend --to-revision=2         # Rollback to
a specific revision
kubectl rollout status -w deployment/frontend                    # Watch
rolling update status of "frontend" deployment until completion
kubectl rollout restart deployment/frontend                      # Rolling
restart of the "frontend" deployment


cat pod.json | kubectl replace -f -                              # Replace a
pod based on the JSON passed into stdin

# Force replace, delete and then re-create the resource. Will cause a service
outage.
kubectl replace --force -f ./pod.json

# Create a service for a replicated nginx, which serves on port 80 and
connects to the containers on port 8000
kubectl expose rc nginx --port=80 --target-port=8000

# Update a single-container pod's image version (tag) to v4
kubectl get pod mypod -o yaml | sed 's/\(image: myimage\):.*$/\1:v4/' |
kubectl replace -f -

kubectl label pods my-pod new-label=awesome                      # Add a Label
kubectl label pods my-pod new-label-                             # Remove a
label
kubectl annotate pods my-pod icon-url=http://goo.gl/XXBTWq       # Add an
annotation
kubectl autoscale deployment foo --min=2 --max=10                # Auto scale
a deployment "foo"
```

Deleting resources

```
kubectl delete -f ./pod.json                                      # Delete a
pod using the type and name specified in pod.json
kubectl delete pod unwanted --now                                 # Delete a
pod with no grace period
```

```
kubectl delete pod,service baz foo                               # Delete
pods and services with same names "baz" and "foo"
kubectl delete pods,services -l name=myLabel                     # Delete
pods and services with label name=myLabel
kubectl -n my-ns delete pod,svc --all                            # Delete all
pods and services in namespace my-ns,
# Delete all pods matching the awk pattern1 or pattern2
kubectl get pods  -n mynamespace --no-headers=true | awk
'/pattern1|pattern2/{print $1}' | xargs  kubectl delete -n mynamespace pod
```

Interacting with Deployments and Services

```
kubectl logs deploy/my-deployment                        # dump Pod logs for
a Deployment (single-container case)
kubectl logs deploy/my-deployment -c my-container        # dump Pod logs for
a Deployment (multi-container case)

kubectl port-forward svc/my-service 5000                 # listen on local
port 5000 and forward to port 5000 on Service backend
kubectl port-forward svc/my-service 5000:my-service-port  # listen on local
port 5000 and forward to Service target port with name <my-service-port>

kubectl port-forward deploy/my-deployment 5000:6000      # listen on local
port 5000 and forward to port 6000 on a Pod created by <my-deployment>
kubectl exec deploy/my-deployment -- ls                  # run command in
first Pod and first container in Deployment (single- or multi-container cases)
```

Interacting with Nodes and cluster

```

kubectl cordon my-node                                               # Mark
my-node as unschedulable
kubectl drain my-node                                                # Drain
my-node in preparation for maintenance
kubectl uncordon my-node                                             # Mark
my-node as schedulable
kubectl top node my-node                                             # Show
metrics for a given node
kubectl cluster-info                                                 #
Display addresses of the master and services
kubectl cluster-info dump                                            # Dump
current cluster state to stdout
kubectl cluster-info dump --output-directory=/path/to/cluster-state   # Dump
current cluster state to /path/to/cluster-state

# View existing taints on which exist on current nodes.
kubectl get nodes -o='custom-
columns=NodeName:.metadata.name,TaintKey:.spec.taints[*].key,TaintValue:.spec.
taints[*].value,TaintEffect:.spec.taints[*].effect'

# If a taint with that key and effect already exists, its value is replaced as
specified.
kubectl taint nodes foo dedicated=special-user:NoSchedule [11]
```

7. CONCLUSİON

In conclusion, Kubernetes is a powerful platform for managing and deploying applications in a cluster environment. Its self-healing abilities, ability to support a wide range of applications, and extensible design make it a valuable tool for many organizations. However, it can also be complex and challenging to learn, making it important for teams to carefully consider their needs and requirements before choosing to use Kubernetes. Overall, Kubernetes offers many benefits, but may not be the right fit for every project.

8.KAYNAKÇA

[1]Kubernetes. "Concepts Overview." Kubernetes, https://kubernetes.io/docs/concepts/overview/. Accessed December 14, 2022.

[2]IBM. "Learn about Containerization." IBM Cloud Learn, https://www.ibm.com/cloud/learn/containerization. Accessed December 14, 2022.

[3]Aqua Security. "Docker Container Orchestration." Cloud Native Academy, https://www.aquasec.com/cloud-native-academy/docker-container/docker-orchestration. Accessed December 14, 2022.

[4]mstrYoda. "Kubernetes Nedir? (What is Kubernetes?)" GitHub, https://github.com/mstrYoda/kubernetes-kitap/blob/master/docs/kubernetes-nedir.md. Accessed December 14, 2022.

[5]Edureka. "Kubernetes Architecture." Medium, https://medium.com/edureka/kubernetes-architecture-c43531593ca5. Accessed December 14, 2022.

[6]Kubernetes. "Components Overview." Kubernetes, https://kubernetes.io/docs/concepts/overview/components/. Accessed December 14, 2022.

[7]Kubernetes. "Workloads Overview." Kubernetes, https://kubernetes.io/docs/concepts/workloads/. Accessed December 14, 2022.

[8]Kubernetes. "Configuration Overview." Kubernetes, https://kubernetes.io/docs/concepts/configuration/overview/. Accessed December 14, 2022.

[9]HiTechNectar. "Pros and Cons of Kubernetes." HiTechNectar, https://www.hitechnectar.com/blogs/pros-cons-kubernetes/. Accessed December 14, 2022.

[12]IBM. "Docker Swarm vs. Kubernetes: A Comparison." IBM Cloud Blog, https://www.ibm.com/cloud/blog/docker-swarm-vs-kubernetes-a-comparison. Accessed December 14, 2022.

[11]Kubernetes. “Commands.” Kubernetes, https://kubernetes.io/docs/reference/kubectl/cheatsheet/