**Kubernetes Application Developer**

**(CKAD)**

**Some Introduction of Kubernetes and a Brief recap**

**Official definition of Kubernetes**

An open-source system for automating deployment, scaling and management of containerized applications

- open source container orchestration (management) tool

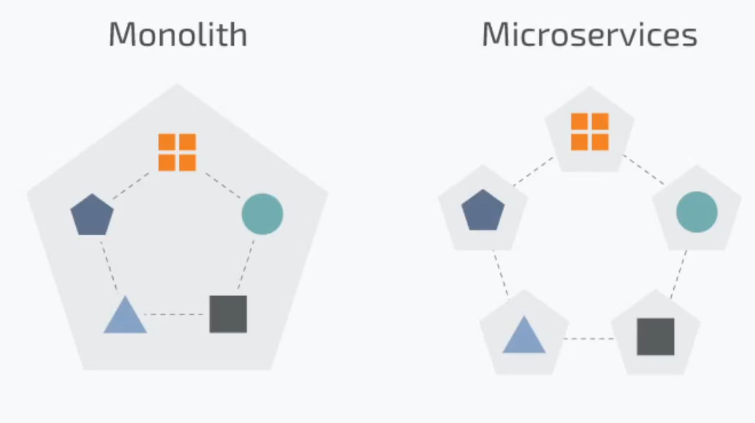
- Originally developed by google

- Also known as K8s

- Developed on Golang

- Helps you manage containerized applications

**What Problems does Kubernetes solve?**

- Trend from monolithic to micro services

- Increase usage of containers

- Demand for a proper way of

managing hundreds of containers

**Features offered by kubernetes**

- high availability or no downtime

- Scalability or high performance

- Disaster Recovery – Backup and restore

**Why k8s rise so fast?**

- Easy to configure and manage

- Saves a lot of manual work

- less prone to errors

- highly scalable

- can define your own resources

- extensible

**Some famous Tools build on top of Kubernetes**

- Istio Service mesh

- Rancher (Network tool)

- Cert manager

- Robusta (for observability of cluster)

- KubeCost (breakdown cluster cost)

**Kubernetes Certifications**

1. Certified Kubernetes Application Developer (CKAD)

2. Certified Kubernetes Administrator (CKA)

3. Certified Kubernetes Security Specialist (CKS)

**Pre-requisites**

- Docker

- Containerization

- Linux

- Vms

- KCNA (not necessary but recommended atleast read my notes)

**Course Curriculum**

 **Application Design and Build 20%**

Define, build and modify container images  
Choose and use the right workload resource (Deployment, DaemonSet, CronJob, etc.)  
Understand multi-container Pod design patterns (e.g. sidecar, init and others)  
Utilize persistent and ephemeral volumes

 **Application Deployment 20%**Understand Deployments and how to perform rolling updates  
Use the Helm package manager to deploy existing packages  
Kustomize

 **Application Observability and Maintenance 15%**Implement probes and health checks  
Use built-in CLI tools to monitor Kubernetes applications  
Utilize container logs  
Debugging in Kubernetes

 **Application Environment, Configuration and Security 25%**Understand authentication, authorization and admission control  
Understand requests, limits, quotas  
Understand ConfigMaps  
Define resource requirements  
Create & consume Secrets  
Understand ServiceAccounts  
Understand Application Security (SecurityContexts, Capabilities, etc.)

 **Services and Networking 20%**Provide and troubleshoot access to applications via services  
Use Ingress rules to expose applications

**Note:-**

I will not discuss the basic concepts of k8s from basics as I have already covered them in the KCNA fundamentals course, those who want to have them click on the following link

https://drive.google.com/file/d/1Q1WASBkprlkzNtWDdS-BRqdEKj7oCuq4/view

**Tools used to make a Kubernetes Environment**

- Kind (easy to setup, run on docker container)

- MiniKube (single node cluster)

- Kubeadm (**most used,** multi-node cluster setup can make production grid)

- Kubespray (powerful but very complex to setup so we’ll not use it)

**Cluster configuration using Kind tool**

**Installing kubectl tool in linux**

### Install kubectl binary with curl on Linux

### Download the latest release with the command:

curl -LO "https://dl.k8s.io/release/**$(**curl -L -s https://dl.k8s.io/release/stable.txt**)**/bin/linux/amd64/kubectl"

**Validate the binary (optional)**

**curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl.sha256"**

**Validate the kubectl binary against the checksum file:**

echo "**$(**cat kubectl.sha256**)** kubectl" | sha256sum --check

If valid, the output is:

kubectl: OK

**Install kubectl**

sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl

**Test to ensure the version you installed is up-to-date:**

kubectl version --client

Or use this for detailed view of version:

kubectl version --client --output=yaml

### **Installing kind tool**

### Installing From Release Binaries

Pre-built binaries are available on our [releases page](https://github.com/kubernetes-sigs/kind/releases).

To install, download the binary for your platform from “Assets”, then rename it to kind (or perhaps kind.exe on Windows) and place this into your $PATH at your preferred binary installation directory.

**On Linux:**

# For AMD64 / x86\_64 [ $(uname -m) = x86\_64 ] && curl -Lo ./kind https://kind.sigs.k8s.io/dl/v0.27.0/kind-linux-amd64 # For ARM64 [ $(uname -m) = aarch64 ] && curl -Lo ./kind https://kind.sigs.k8s.io/dl/v0.27.0/kind-linux-arm64 chmod +x ./kind sudo mv ./kind /usr/local/bin/kind

|  |
| --- |

|  |
| --- |
| # For AMD64 / x86\_64  [ $(uname -m) = x86\_64 ] && curl -Lo ./kind https://kind.sigs.k8s.io/dl/v0.27.0/kind-linux-amd64  # For ARM64  [ $(uname -m) = aarch64 ] && curl -Lo ./kind https://kind.sigs.k8s.io/dl/v0.27.0/kind-linux-arm64  chmod +x ./kind  sudo mv ./kind /usr/local/bin/kind |

**Getting start with the kind tool**

run the following commands as I will tell

#kind create cluster => a cluster will be create by a docker image will be pulled

# kind get clusters => to verify cluster being created

# kubectl get all -A => to see all the resources being made while the cluster was made

# kind delete cluster --name kind ==> the cluster that was created will be removed

# kind create cluster --name alnafi-cluster

# kubectl get nodes -o wide ==> for gaining more information

**Making a cluster via yml file**

cluster-config.yml

kind: Cluster

apiVersion: kind.x-k8s.io/v1alpha4

name: saim-talha

nodes:

- role: control-plane

- role: worker

image: kindest/node:v1.32.2 # to add container images for nodes

- role: worker

image: kindest/node:v1.32.2

run this command for execution

# kind create cluster --config cluster-config.yml

### **Cluster configuration using minikube**

**Installing and setting up minikube**

Note:- virtual box should be installed as we’re gonna use vagrant

run the following commands

# curl -LO <https://github.com/kubernetes/minikube/releases/> [latest/download](https://github.com/kubernetes/minikube/releases/latest/download)/minikube-linux-amd64

# sudo install minikube-linux-amd64 /usr/local/bin/minikube && rm minikube-linux-amd64

To verify run this

minikube version

Generating a clusters

# minikube start

Minikube dashboard

#minikube dashboard ==> a dashboard will be opened on browser

Minikube cluster deleting

minikube delete –all

selecting a specific driver

minikube start - -driver virtualbox

Pausing (stopping ) a clusters

minikube pause

Unpausing (starting) a clusters

minikube unpause

Getting pods in a specific node

kubectl get pods -n kube-system

**Some freewebsites where could practice k8s**

playwithk8s

killercoda

**What is a pods?**Pods are **the smallest deployable units of computing that you can create and manage in Kubernetes**. A Pod (as in a pod of whales or pea pod) is a group of one or more containers, with shared storage and network resources, and a specification for how to run the containers.

**Getting hands-on single container pods**

first make a cluster using the following code

#config.yml

kind: Cluster

apiVersion: kind.x-k8s.io/v1alpha4

name: my-cluster

nodes:

- role: control-plane

- role: worker

- role: worker

for verification run this

# kind get clusters

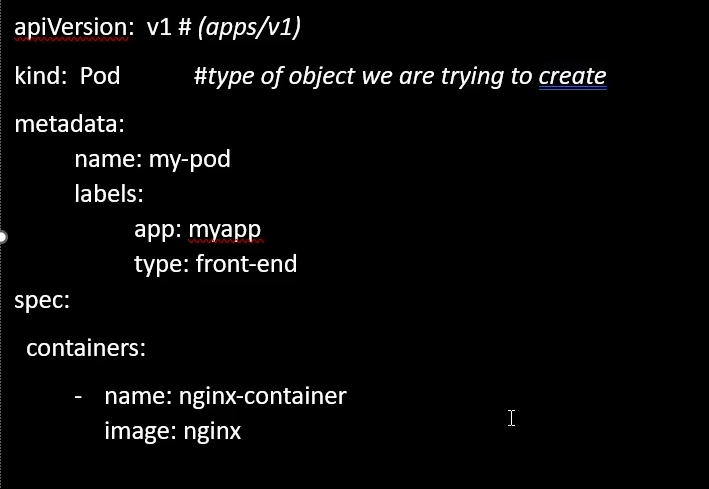
my-cluster

Checking how much pods are running

# kubectl get pods

o/P:- No resources found in default namespace.

**Four** basic structural things to mention in a k8s object configuration file



apiVersion: #what type api you want to use

kind: #what object you have to make Pods/deployments/ /*replicaset/*conifgmap

metadata: #to specify name, label

spec: #Most important part defines what resources it would have

Sample file (mypod.yml)

apiVersion: v1

kind: Pod

metadata:

name: my-pod

labels:

app: frontend

version: development

spec:

containers:

- name: alpha

image: nginx

**Command so this file could run**

#kubectl apply -f mypod.yml

**For gathering additional information about the pod**

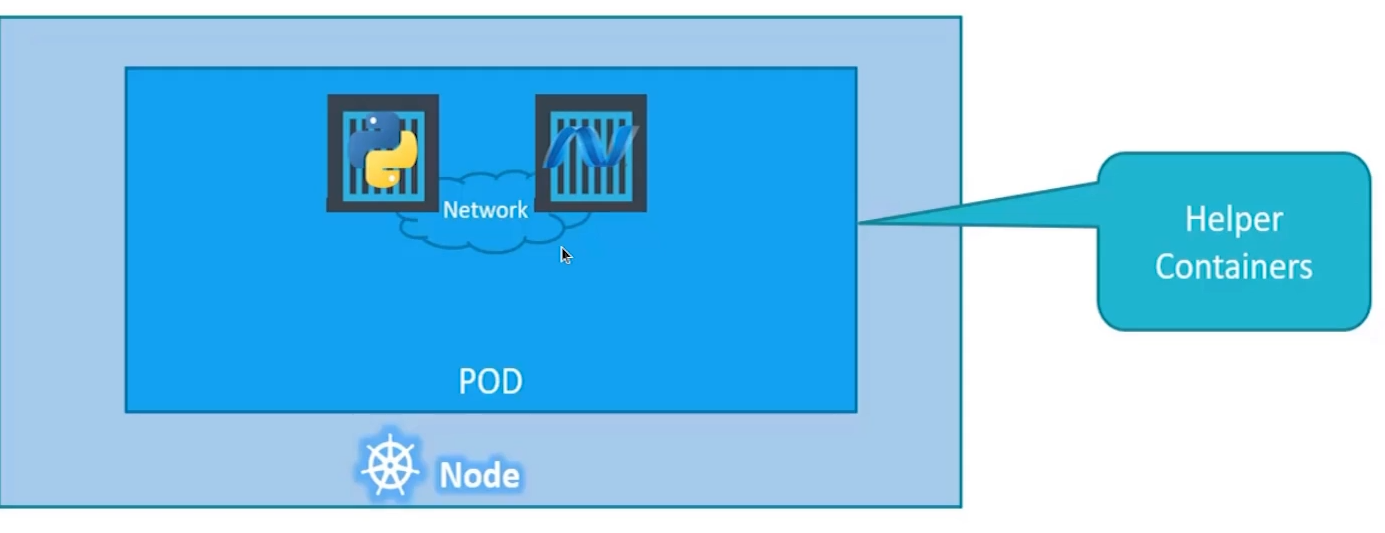
#kubectl describe pod my-pod

**Run pod directly without make a yml file**

# kubectl run mypod –image nginx

**For deleting pods**

# kubectl delete pod <pod-name>

**Multi-container pods (**very rarely used**)**

**Note:-**

- it is forbidden to use same image for both containers in a single pod

- Most of the time we use single container in a single pod

- Both container will have same fate means if one container goes down all the pod goes down

**Design Patterns**

**1. Side Car**

- Performs a task that will access the main container (logs for main container)

**2. Adapter**

**-** proxies network traffic from the main container (shifting portt 80 to 81 )

**3. Ambassador**

- Transforms the output of the container in some way

**Hands-on with sidecar**

let’s first make a configuration file

# sidecar-practice-pod.yml

apiVersion: v1

kind: Pod

metadata:

name: sidecar-practice-pod

spec:

containers:

- name: main-container

image: busybox:stable

command: ['sh', '-c', 'echo "i am learning kubernertes" > /output/data.txt; while true; do sleep 5; done']

volumeMounts:

- name: shared-volume

mountPath: /output

- name: sidecar-container

image: busybox:stable

command: ['sh', '-c', 'while true; do cat /input-dir/data.txt; sleep 5; done']

volumeMounts:

- name: shared-volume

mountPath: /input-dir

volumes:

- name: shared-volume

emptyDir: {}

**How to get logs**

kubectl logs sidecar-practice-pod -c sidecar-container

**Output:-**

i am learning kubernertes

i am learning kubernertes

i am learning kubernertes

i am learning kubernertes

**What is init container in kubernetes**?  
An "init container" is a special type of container that runs before the main application containers within a pod, allowing for initialization tasks like setting up configuration files, downloading dependencies, or preparing data before the primary application container starts running in a pod.

**Why Use init container**

* Dependency Initialization – Ensure required services (e.g., databases) are up before starting the main container.
* Configuration Setup – Fetch configuration files or secrets before the main container runs.
* Data Preprocessing – Download datasets, extract files, or set up necessary directories.
* Security & Privilege Separation – Perform tasks needing higher privileges separately from the main application.
* Delay Execution **–** Pause the main container until external conditions (like network readiness) are met.

**Note:-**

init container is not a multi-container pod

**Hands-on with init container**

first make a file yml

apiVersion: v1

kind: Pod

metadata:

name: init-container-pod

spec:

containers:

- name: main-container

image: nginx:stable

**initContainers: # main keyword to make init container**

- name: init-container

image: busybox:stable

command: ['sh', '-c' ,'sleep 60']

**Running this command to see the magic of init containers**

# kubectl apply -f init-pods.yml

pod/init-container-pod created

# kubectl get po -w

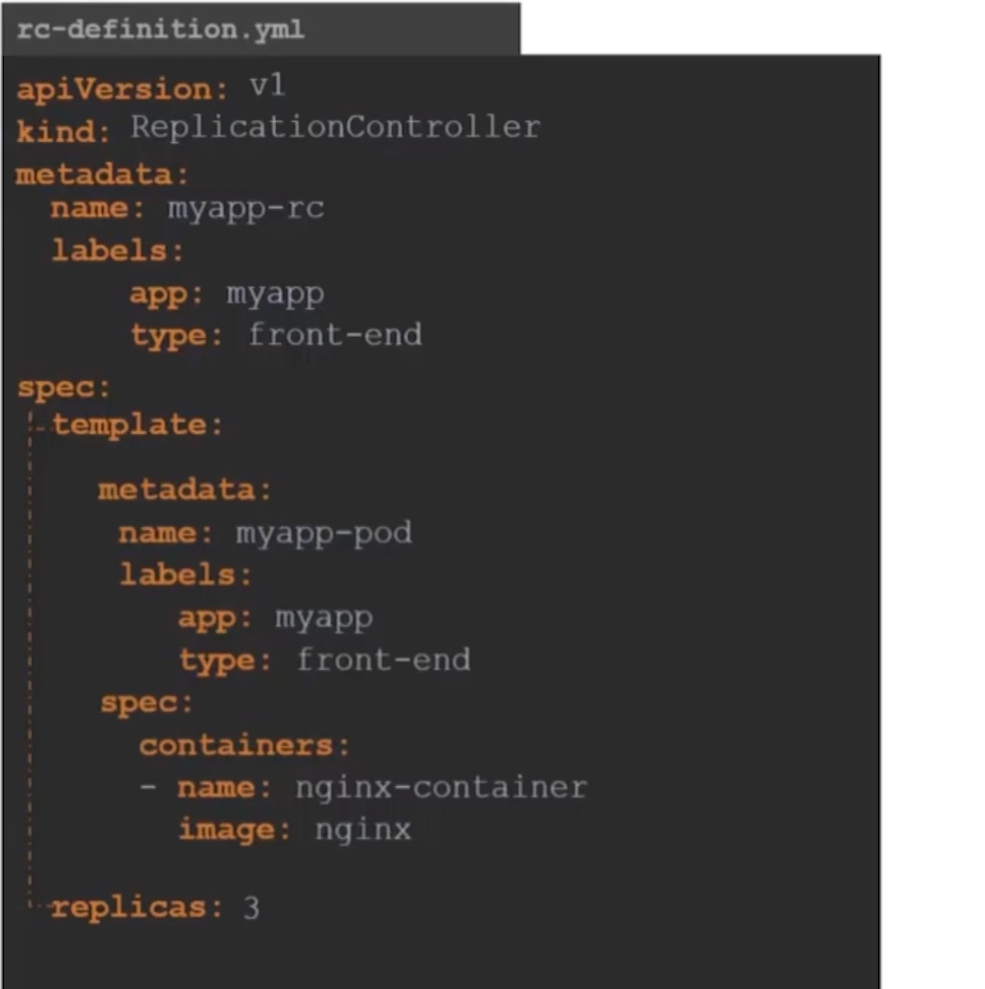
**Replication Controller**

A Replication Controller (RC) in Kubernetes is a resource that ensures a specified number of pod replicas are running at any given time. If a pod crashes or is deleted, the Replication Controller automatically creates a new pod to maintain the desired state.

### **Why Use a Replication Controller?**

1. Ensures Availability: If a pod fails or is accidentally deleted, the Replication Controller replaces it automatically.
2. Scalability: You can easily scale applications up or down by increasing or decreasing the number of replicas.
3. Load Balancing: It distributes traffic across multiple pod replicas, improving performance and reliability.
4. Self-Healing: If a node fails, Kubernetes schedules the pods on another available node to maintain application uptime.
5. Rolling Updates: It helps in safely updating applications by incrementally

**Replication Controller (tempelate)**

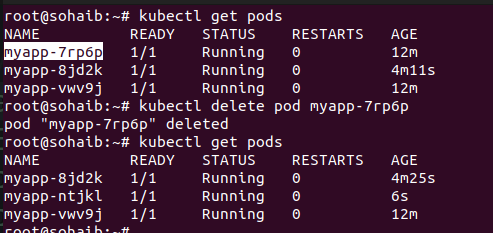


**Hands-on with replication controller**

|  |  |
| --- | --- |
| # cat rc-configuration.yml  apiVersion: v1  kind: ReplicationController  metadata:  name: myapp  labels:  app: frontend  type: development  spec:  selector:  app: frontend | template:  metadata:  name: my-pod  labels:  app: frontend  spec:  containers:  - name: alpha  image: nginx  replicas: 3  # kubectl apply -f rc-configuration.yml  # kubectl get po,replication controller |

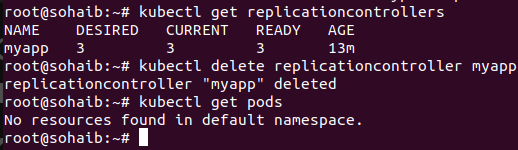
**what we learn from this hands-on?**

If we delete a pod or it’s failed for some reason then replication controller will automatically deploy a pod itself let’s see how



**Note:-**

if we delete a replication-controller so all pods in it will be deleted



**Replica – Sets (**more powerful, used, features then replication controller**)**

**What is replica set?**

A ReplicaSet (RS) ensures a specified number of pod replicas are running, replacing failed pods automatically. It is the successor to the Replication Controller with improved features.

**Difference between**

- Label Selector: ReplicaSet supports set-based selectors, allowing more flexible pod selection, whereas Replication Controller uses only equality-based selectors.

- Efficiency: ReplicaSet is more powerful and integrates with Deployments, making it easier to manage rolling updates.

### **Why Use ReplicaSet?**

ReplicaSet is preferred as it works with Deployments, enabling better scalability, updates, and management of workloads.

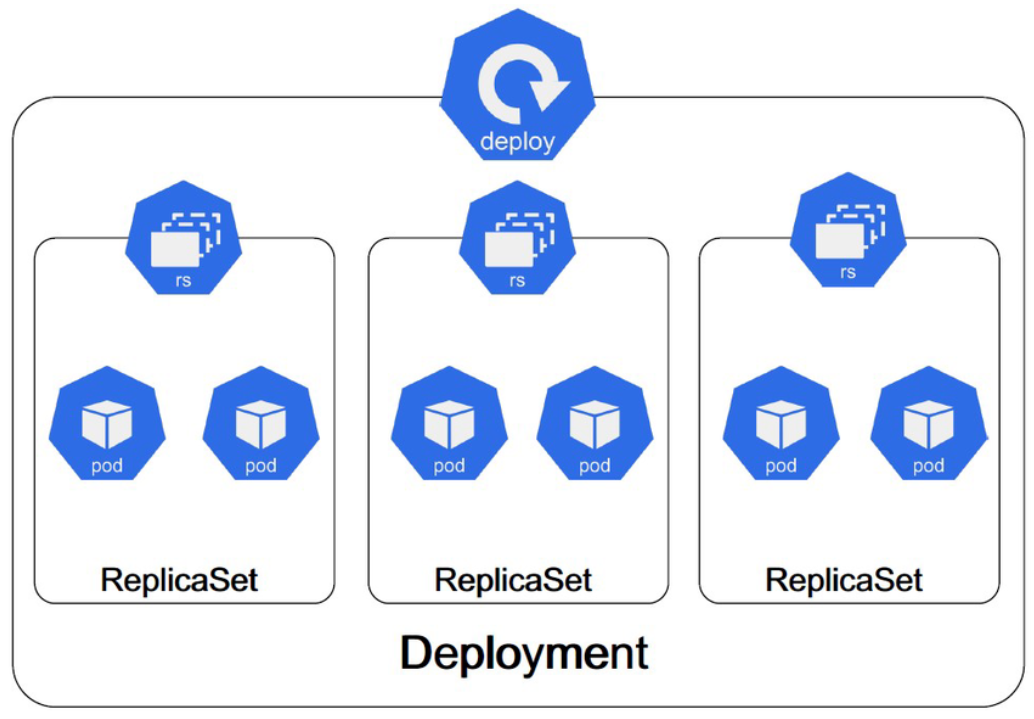
**Deployments In Kubernetes**

**what is deployment in k8s?**

A deployment in k8s defined a desired state for a set of replica pods. Kubernetes constantly works to maintain that desireed state by creating, deleting and replacing those pods

**Note:-**

a replica set doesn’t deploy pods itself instead it create replica sets first and then with help of replica pods are managed as shown in the hierachy following:



**Working hands-on with deployments**

# cat deployment.yml

apiVersion: apps/v1

kind: Deployment

metadata:

name: simple-deployment

spec:

replicas: 3

selector:

matchLabels:

app: frontend

template:

metadata:

name: simple-pod

labels:

app: frontend

spec:

containers:

- name: nginx

image: nginx:stable

ports:

- containerPort: 80

**Note:-**

if we apply this file and delete a pod it will be re-created automatically as it is under-provisioned of replicaset and if replica-set is deleted it will be re-created as it is under-provisioned of deployments

**For scaling number of replicas**

# kubectl scale deploy simple-deployment --replicas 4

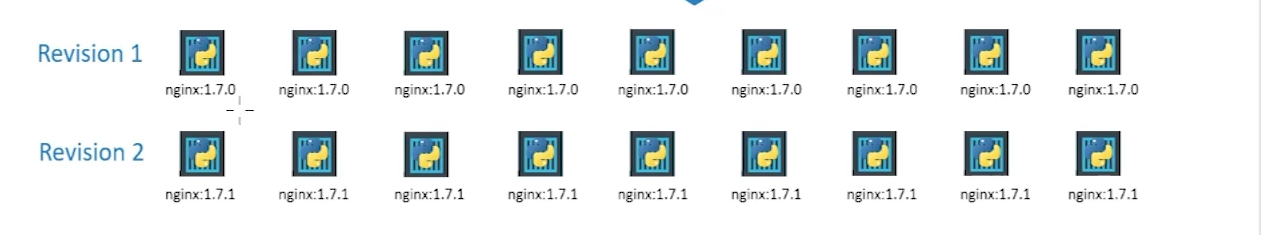
#kubectl edit deploy simple-deployment ==> make change in the api-server file which is an internal file

**One more thing to note:**

"A Kubernetes Deployment creates and manages Pods through ReplicaSets. It does not adopt existing Pods, even if they have the same labels. Instead, it creates new Pods based on the spec.template defined in the Deployment.

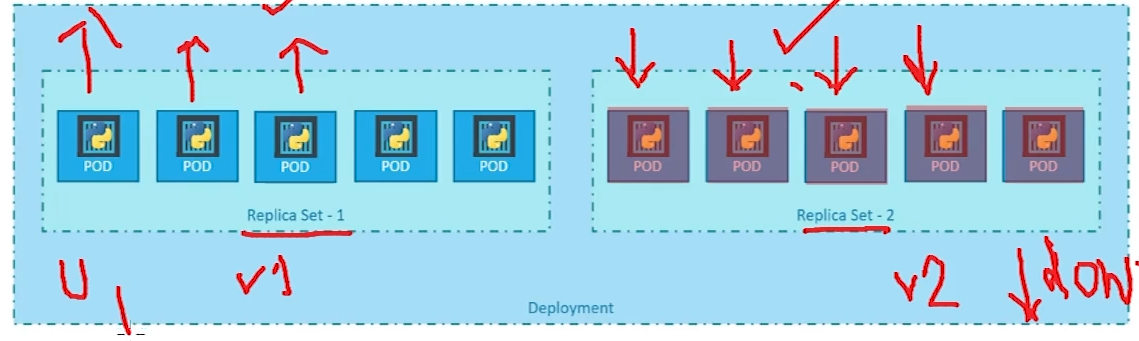
### **Deployments -- Updates and Rollbacks**

* Rollouts: The process of deploying a new version of an application using a Deployment. Kubernetes ensures a smooth transition with zero downtime.



* Updates: Modifying an existing Deployment (e.g., changing the container image). Kubernetes gradually replaces old pods with new ones.



* Rollbacks: Reverting to a previous Deployment version if an update fails. Kubernetes keeps track of previous revisions and allows restoring a stable state using kubectl rollout undo.

**Some commands to remember for rollbacks**

* kubectl rollout status deploy myapp-deployment – Checks the rollout progress of the deployment.
* kubectl get po / kubectl get pods – Lists all running pods.
* kubectl rollout history deploy myapp-deployment – Shows the rollout history of the deployment.
* kubectl delete deploy myapp-deployment – Deletes the deployment.
* kubectl get po,rs,deployment / kubectl get rs,po,deploy – Lists pods, replica sets, and deployments.
* kubectl apply -f rolling-deploy.yml --record – Deploys resources while recording changes for history tracking.
* kubectl describe deploy myapp-deployment – Provides detailed information about the deployment.
* kubectl edit deploy myapp-deployment – Opens the deployment definition for editing.
* kubectl edit deploy myapp-deployment --record – Edits the deployment while recording changes.
* kubectl set image deploy myapp-deployment nginx=nginx:1.22-alpine --record – Updates the deployment's container image and records the change.
* kubectl set image deploy myapp-deployment nginx-container=nginx:1.22-alpine-doesn't-exist --record – Attempts to update to a non-existent image (used for testing rollback scenarios).
* kubectl rollout undo deploy myapp-deployment – Rolls back the deployment to the previous stable version.
* kubectl rollout undo deploy myapp-deployment --to-revision=1 – Rolls back the deployment to a specific revision.

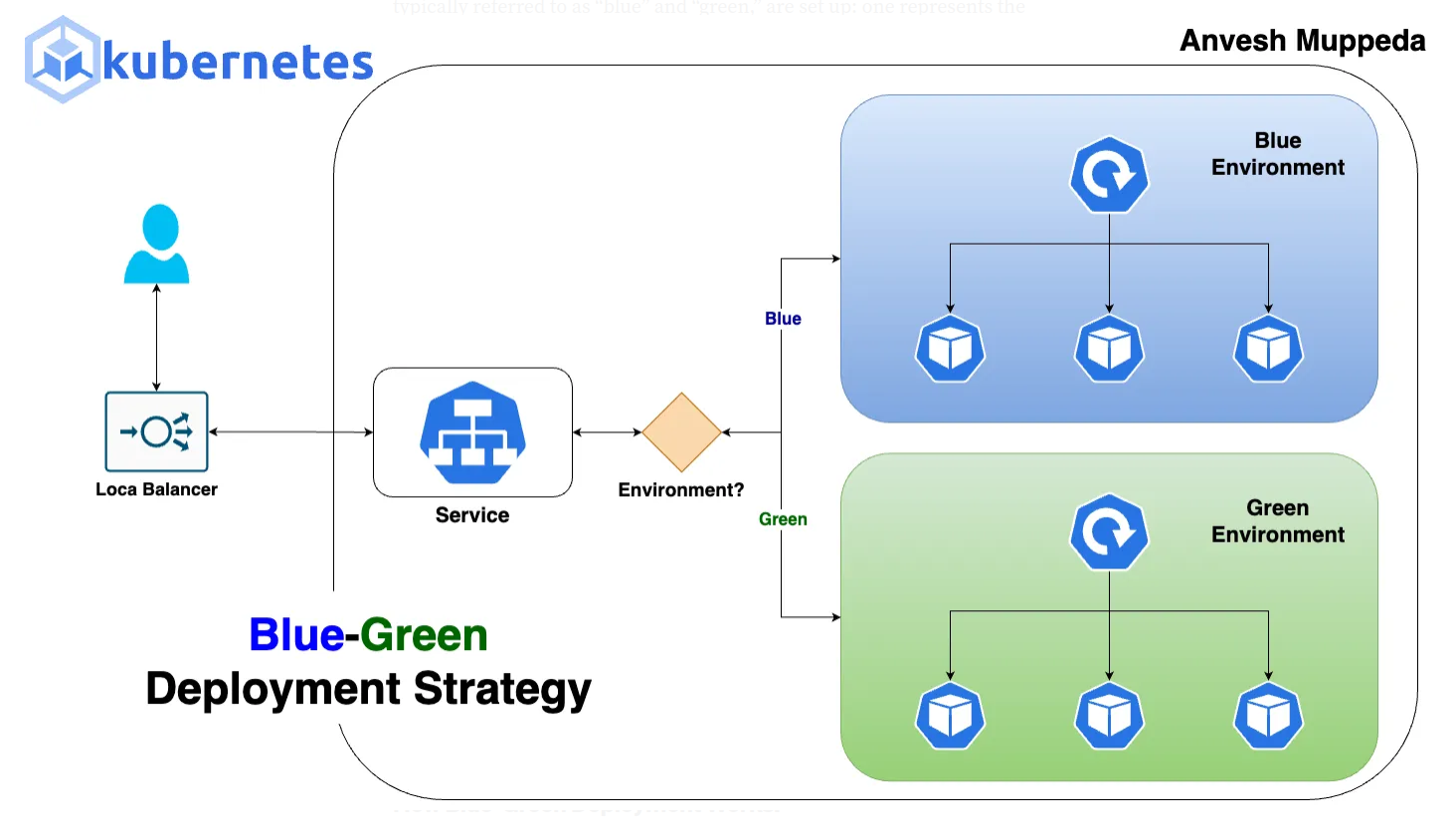
**Deployment strategies**

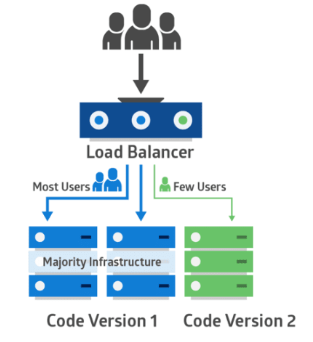
**What is a deployment strategy**

A deployment strategy is a method of rolling out new code that is used to archive some benefit such as increasing reliability and minimizing the risk

**Blue green deployment**

Blue-green deployment is a popular deployment strategy in Kubernetes that runs two versions of your app side-by-side, with traffic directed to the old release until you promote the new one



**Canary deployment**

A "canary deployment" in Kubernetes is , allowing developers to test the new version in a live environment with minimal risk and monitor its performance before exposing it to the wider user base; essentially, it's a controlled way to gradually roll out software updates by directing a small percentage of traffic to the new version while keeping the majority on the stable one.

### **Blue-Green Deployment Strategy Hands-On**

**#1 First let’s make a blue deployment**

# cat blue-deployment.yml

apiVersion: apps/v1

kind: Deployment

metadata:

name: blue-deployment

spec:

replicas: 1

selector:

matchLabels:

app: bluegreen-test

color: blue

template:

metadata:

labels:

**app: bluegreen-test**

**color: blue #** play very important role

spec:

containers:

- name: nginx

image: linuxacademycontent/ckad-nginx:blue

ports:

- containerPort: 80

**#2 make a service so it can redirect the traffic to the blue deployment first**

# cat blue-green-svc.yml

apiVersion: v1

kind: Service

metadata:

name: blue-green-svc

spec:

selector:

**app: bluegreen-test**

**color: blue**

ports:

- protocol: TCP

port: 80

targetPort: 80

**#3 checking that blue deployment is running correctly**

- take ssh into the worker node

run the following curl commands and output **should be as following:**

# curl <ip of the service>

o/p:-

I’m Blue

**#4 making green deployment**

# cat green-deployment.yml

apiVersion: apps/v1

kind: Deployment

metadata:

name: green-deployment

spec:

replicas: 1

selector:

matchLabels:

app: bluegreen-test

color: green

template:

metadata:

labels:

app: bluegreen-test

color: green

spec:

containers:

- name: nginx

image: linuxacademycontent/ckad-nginx:green

ports:

- containerPort: 80

**#5 checking that green deployment is running correctly**

- take ssh into the worker node

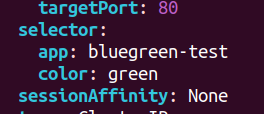
run the following curl commands and output **should be as following:**

# curl <ip of the green-pod>

o/p:-

I’m Green

**#6 Finally redirecting traffic into the green deployment**

#kubectl edit svc blue-green-svc

# just change the selector from blue to green as shown

**Task**

Just like blue green-blue-deployment try to test the canary deployment following things will be needed

#main environment image is linuxacademycontent/ckad-nginx:1.0.0 will have 3 replicas

#new version image is linuxacademycontent/ckad-nginx:canary will have 1 replicas

# a service which will be a load balancer with a selector of the labels defined

**Note:-**

Answer of the task could be found in my-github repo

### **Namespaces**

Namespaces in Kubernetes are used for multi-tenancy and resource organization within a cluster. They help in logically isolating resources and managing them efficiently. Below are the key concepts related to Namespaces according to the Certified Kubernetes Application Developer (CKAD) curriculum:

### **1. Basics of Namespaces**

* A Namespace is a virtual cluster within a physical Kubernetes cluster.
* Useful for environment separation (e.g., dev, staging, production).
* They allow multiple teams or projects to share the same Kubernetes cluster without conflicts.

### **2. Default Namespaces**

Kubernetes comes with three pre-defined namespaces:

1. default → Where all resources are placed if no namespace is specified.
2. kube-system → Contains Kubernetes system components (e.g., kube-dns).
3. kube-public → Used for publicly accessible resources.

### **3. Creating and Using Namespaces**

* kubectl create namespace my-namespace
* Making with a yml file

# cat namespaces.yml

apiVersion: v1

kind: Namespace

metadata:

name: alpha-namespace

### **4. Switching and Viewing Namespaces**

* List all namespaces

#kubectl get namespaces

* View resources in a specific namespace

#kubectl get pods -n my-namespace

* Set a default namespace for kubectl (current session)

#kubectl config set-context $(kubectl config current-context) --namespace=kube-system

* Creating resource In specific namespaces

# kubectl apply -f pod.yml -n **default**

* Check the current namespace

#kubectl config get-contexts

**5. Setting resource quotas for**

# cat resource-quota.yml

|  |  |
| --- | --- |
| # cat resource-quota.yml  apiVersion: v1  kind: ResourceQuota  metadata:  name: compute-quota  namespace: alpha-namespace  spec:  hard:  pods: "2"  requests.cpu: "2"  requests.memory: 2Gi  limits.cpu: "3"  limits.memory: 3Gi | cat pod.yml  apiVersion: v1  kind: Pod  metadata:  name: saim-pdo  namespace: alpha-namespace  spec:  containers:  - name: nginx  image: nginx  resources:  requests:  memory: "1Gi"  cpu: "1"  limits:  memory: "1.5Gi"  cpu: "1" |
|  |  |

# Kubernetes Jobs

## What is a Job in Kubernetes?

A Job in Kubernetes is a workload resource that ensures a specified number of pods run to completion. Unlike Deployments or ReplicaSets, which manage continuously running applications, Jobs are used for batch processing or one-time execution tasks.

### Key Characteristics of Jobs:

* Runs tasks to completion.
* Can retry failed pods.
* Suitable for data processing, backups, batch scripts, and maintenance tasks.
* Can be scheduled to run once or multiple times.

## Approaches to Create Jobs in Kubernetes

### 1. **Single-Execution Job**

A single-execution Job runs a pod once and completes after the task finishes.

#### Example:

apiVersion: batch/v1

kind: Job

metadata:

name: simple-job

spec:

template:

spec:

containers:

- name: hello

image: busybox

command: ["echo", "Hello, Kubernetes!"]

restartPolicy: Never

### 2. **Job with Multiple Parallel Pods**

A parallel Job runs multiple pods in parallel to complete the task faster.

#### Example:

apiVersion: batch/v1

kind: Job

metadata:

name: parallel-job

spec:

completions: 5 # Ensures 5 pods complete execution

parallelism: 2 # Runs 2 pods concurrently

template:

spec:

containers:

- name: worker

image: busybox

command: ["echo", "Processing..."]

restartPolicy: Never

### 3. **Job with a Backoff Limit (Retry Mechanism)**

A backoff limit defines how many times Kubernetes retries a failed pod before marking the Job as failed.

#### Example:

apiVersion: batch/v1

kind: Job

metadata:

name: retry-job

spec:

backoffLimit: 3 # Retries the job 3 times if it fails

template:

spec:

containers:

- name: error-prone-task

image: busybox

command: ["/bin/sh", "-c", "exit 1"] # Forces failure

restartPolicy: Never

### 4. **CronJob (Scheduled Job)**

A CronJob runs Jobs on a schedule, similar to cron jobs in Linux.

#### Example:

apiVersion: batch/v1

kind: CronJob

metadata:

name: scheduled-job

spec:

schedule: "\*/5 \* \* \* \*" # Runs every 5 minutes

jobTemplate:

spec:

template:

spec:

containers:

- name: scheduled-task

image: busybox

command: ["echo", "This job runs every 5 minutes"]

restartPolicy: Never

## Conclusion

Kubernetes Jobs are essential for batch processing and scheduled tasks. Understanding different Job configurations is crucial for the CKAD (Certified Kubernetes Application Developer) exam. The key points to remember:

* Single-execution Jobs run once and complete.
* Parallel Jobs execute multiple pods simultaneously.
* Retry mechanisms (backoffLimit) ensure resilience.
* CronJobs allow scheduling periodic tasks.

Mastering these concepts will help you efficiently manage workloads in Kubernetes.

**Command and arguments in a Pod**

In Kubernetes, command line and arguments allow you to specify how a container should run inside a Pod. These are defined in the Pod’s YAML file using the command and args fields under spec.containers.

* command: Overrides the default entrypoint of the container.
* args: Passes arguments to the command.

Example 1: Using 'command' to define the entrypoint

apiVersion: v1

kind: Pod

metadata:

name: sleep-pod

spec:

containers:

- name: sleep-container

image: busybox

command: ["sleep", "3600"]

Example 2: Using 'args' to provide arguments

apiVersion: v1

kind: Pod

metadata:

name: sleep-pod

spec:

containers:

- name: sleep-container

image: busybox

command: ["sleep"]

args: ["3600"]

### Environment Variables and ConfigMaps in Kubernetes (CKAD Perspective)

In Kubernetes, Environment Variables and ConfigMaps are essential for managing application configuration in a decoupled and scalable way.

#### **Environment Variables**

* Used to pass configuration data directly into containerized applications.
* Defined in Pod specifications under env or sourced from ConfigMaps/Secrets using envFrom.
* Example:

apiVersion: v1

kind: Pod

metadata:

name: env-demo

spec:

containers:

- name: demo-container

image: nginx

env:

- name: APP\_MODE

value: "production"

#### **ConfigMaps**

* Used to store non-sensitive configuration data in key-value pairs.
* Can be consumed as environment variables, command-line arguments, or mounted as files.
* Helps in keeping configurations separate from containerized applications, improving maintainability.
* Example:

apiVersion: v1

kind: ConfigMap

metadata:

name: app-config

data:

APP\_MODE: "production"

Pod consuming ConfigMap as environment variables:

Key Differences:

* Environment variables are directly set in the Pod spec, while ConfigMaps externalize configuration.
* ConfigMaps support larger sets of configurations and can be updated dynamically without modifying Pod definitions.
* Environment variables are immutable per Pod restart, whereas ConfigMaps can be updated without redeploying the application.

CKAD Relevance: Understanding how to manage configuration using Environment Variables and ConfigMaps is crucial for maintaining portable, scalable, and configurable Kubernetes applications.

### Kubernetes Secrets

#### **What are Kubernetes Secrets?**

In Kubernetes, Secrets are used to store and manage sensitive data like passwords, API keys, and certificates. Unlike ConfigMaps, which store non-sensitive data, Secrets help keep sensitive information secure and prevent it from being exposed in plaintext inside Pod definitions.

#### **Why Use Secrets?**

* Security: Prevents exposing sensitive information directly in Pod YAML files.
* Decoupling: Separates secret data from application configurations.
* Better Management: Can be updated without modifying the application pods.

### **Example Lab Using Secrets**

#### **Step 1: Creating a Secret**

We will create a Secret to store the MySQL database credentials instead of hardcoding them in the Pod YAML file.

Create a file named db-secret.yaml:

apiVersion: v1

kind: Secret

metadata:

name: db-secret

namespace: default

type: Opaque

data:

MYSQL\_ROOT\_PASSWORD: cGFzc3dvcmQxMjM= # Base64 encoded password123

Note:

* The password password123 is encoded in Base64 (echo -n 'password123' | base64).
* Kubernetes stores Secrets in an encoded format, but they are not encrypted by default.

Apply the secret:

kubectl apply -f db-secret.yaml

#### **Step 2: MySQL Pod Without Secrets (Bad Practice)**

The following mysql-pod.yaml defines a MySQL Pod without using Secrets, which is insecure:

apiVersion: v1

kind: Pod

metadata:

labels:

name: mysql

name: mysql

namespace: default

spec:

containers:

- name: mysql

image: mysql:5.6

env:

- name: MYSQL\_ROOT\_PASSWORD

value: password123 # Hardcoded Password (Not Secure)

ports:

- containerPort: 3306

protocol: TCP

Problem: If someone accesses this file, they can see the database password.

#### **Step 3: Web Application Pod Using Secrets (Best Practice)**

The web-pod.yaml file shows how to use Secrets properly:

apiVersion: v1

kind: Pod

metadata:

labels:

name: webapp-pod

name: webapp-pod

namespace: default

spec:

containers:

- image: kodekloud/simple-webapp-mysql

name: webapp

envFrom:

- secretRef:

name: db-secret # Using the Secret

Here, instead of storing the MySQL password in the Pod YAML, we use the db-secret to inject environment variables.

Apply the Pod:

kubectl apply -f web-pod.yaml

#### **Step 4: Exposing Web Application with a Service**

A Service is needed to expose the web application:

apiVersion: v1

kind: Service

metadata:

name: webapp-service

namespace: default

spec:

ports:

- nodePort: 30080

port: 8080

protocol: TCP

targetPort: 8080

selector:

name: webapp-pod

type: NodePort

Apply the service:

kubectl apply -f web-pod-svc.yml

### **Final Steps: Testing the Setup**

1. Check if the Secret is created:

kubectl get secrets

1. Verify the Pods:

kubectl get pods

1. Access the Web Application:  
   Open the browser and go to:

http://<Node-IP>:30080

Replace <Node-IP> with the IP of your Kubernetes node.

### **Key Takeaways**

* Never hardcode sensitive information inside Pod definitions.
* Use Secrets to securely store and inject sensitive data into containers.
* Base64 encoding is not encryption; additional security measures (like Kubernetes Encryption at Rest) should be used.

This lab demonstrates both a bad and a good practice when handling sensitive data, helping students understand why Secrets are essential in Kubernetes!

Security Context

A Security Context in Kubernetes defines privilege and access control settings for a Pod or Container. It includes settings like user/group IDs, capabilities, SELinux, AppArmor, seccomp profiles, and privilege escalation.

Example:

apiVersion: v1

kind: Pod

metadata:

name: secure-pod

spec:

securityContext:

runAsUser: 1000

runAsGroup: 3000

fsGroup: 2000

containers:

- name: secure-container

image: nginx

securityContext:

allowPrivilegeEscalation: false

### **Resource Requests and Limits in Kubernetes**

In Kubernetes, resource requests and limits define how much CPU and memory a container is guaranteed and the maximum it can use.

* Requests: The minimum amount of CPU/memory a container is guaranteed. The scheduler uses requests to decide where to place the pod.
* Limits: The maximum CPU/memory a container can use. If a container exceeds its limit, Kubernetes takes action.

### **Behavior When Limits are Exceeded:**

* CPU Limit Exceeded: The container is throttled (its CPU usage is reduced but it continues running).
* Memory Limit Exceeded: The container is terminated with an OOM (Out of Memory) error and may restart if a restart policy is set

Example:

apiVersion: v1

kind: Pod

metadata:

name: my-pod

spec:

containers:

- name: my-container

image: nginx

resources:

requests:

cpu: "250m" # 250 millicores (0.25 vCPU)

memory: "64Mi" # 64 Megabytes

limits:

cpu: "500m" # 500 millicores (0.5 vCPU)

memory: "128Mi" # 128 Megabytes

### **Key Differences:**

| Aspect | Requests (Guaranteed) | Limits (Maximum) |
| --- | --- | --- |
| Purpose | Minimum resources needed | Max resources allowed |
| Scheduler Role | Used to schedule the pod | Not considered in scheduling |
| CPU Exceeding | Can use more if available | Throttled |
| Memory Exceeding | Can use more if available | OOMKill (terminated) |

### **Taints and Tolerations in Kubernetes**

Taints and tolerations help control which nodes can run which pods in Kubernetes.

* Taints are like “No Entry” signs on a node. They tell pods not to run on that node unless they have permission.
* Tolerations are like special passes that let certain pods ignore the "No Entry" sign and run on that node.

#### **Example:**

* A node has a taint: node-role=database:NoSchedule (meaning only database-related pods can run here).
* A database pod has a matching toleration, allowing it to run on that node. Other pods without tolerations will not be scheduled there.

#### **Analogy:**

Imagine a VIP lounge at an airport (the node). There is a “VIP Only” sign (the taint). Only people with a VIP pass (toleration) can enter. Regular passengers cannot.

**Note:-**

It is not necessary that if a pod is tolerated with a node so it will deploy on that specific pod it could also deploy on other nodes that are not tainted so to solve this problem we will use node selector and node affinity coming text so stay tuned!

Taints and Tolerations Lab Output

### 1): Get Node Name

#### Command:

kubectl get nodes

#### Expected Output:

NAME STATUS ROLES AGE VERSION

node1 Ready master 10d v1.25.0

node2 Ready worker 10d v1.25.0

### 2): Apply a Taint

#### Command:

kubectl taint nodes node2 key=database:NoSchedule

#### Expected Output:

node/node2 tainted

### 3): Deploy a Pod with a Toleration

#### Command:

kubectl apply -f pod-toleration.yaml

#### Expected Output:

pod/db-pod created

#### Verify Pod is Scheduled on Tainted Node:

kubectl get pods -o wide

NAME READY STATUS NODE AGE

db-pod 1/1 Running node2 10s

### 4): Deploy a Pod Without a Toleration (Fails to Schedule)

#### Command:

kubectl apply -f pod-no-toleration.yaml

#### Expected Output:

pod/normal-pod created

#### Check Pod Status:

kubectl get pods -o wide

NAME READY STATUS NODE AGE

normal-pod 0/1 Pending 10s

### 5): Remove the Taint (Cleanup)

#### Command:

kubectl taint nodes node2 key=database:NoSchedule-

#### Expected Output:

node/node2 untainted

**Node Selectors**

**What**: A way to assign pods to specific nodes based on labels.

**Why**: Ensures that pods run only on nodes with matching labels.

**Drawback:** doesn’t support advanced filtering so we use node

affinity

 **How:** Pods specify a nodeSelector that matches the labels on a

node.

**Labs**

first label a pod with specifc label like size=large using following command

# kubectl label node my-cluster-control-plane mem=large

Then make a pod definition and specify the nodeSelector field as following:-

spec:

containers:

- name: nginx-container

image: nginx

nodeSelector:

mem: Large



**Node Affinity**

**Definition:** node affinity is a k8s features that allows you to

specify preference for scheduling pods onto particular nodes in the

cluster.

**Purpose:** its used to influence pod placement based on node

characteristics or constraints.

**Types of Node Affinity:**

Type 1

*RequiredDuringSchedulingIgnoredDuringExecution*

**(Hard Requirement)**

Pods will only be scheduled on nodes that math the specified affinity rules.

Type 2

*PreferredDuringSchedulingIgnoredDuringExecution*

**(Soft Requirement)**

Pods prefer to be scheduled on nodesthat match the specified affinity rules, but its not a strict requirement.

**What would be the best practice?**

To use node affinity and taints and tolerance together for better scheduling of pods to nodes

**Probes in Kubernetes**

In Kubernetes, probes are used to check the health and readiness of a container. They help determine whether a container should receive traffic or be restarted if it's unhealthy.

### Types of Probes:

1. Liveness Probe – Checks if the container is still running. If it fails, Kubernetes restarts the container.
2. Readiness Probe – Checks if the container is ready to accept traffic (means that application is ready or not). If it fails, traffic is not sent to the container until it recovers.
3. **StartUp Probe –** Not much used so will not discuss

### Probe Mechanisms:

* HTTP Probes (httpGet) – Sends an HTTP request to a specified path and port inside the container. If the response status is between 200-399, the probe is successful.
* TCP Probes (tcpSocket) – Tries to establish a TCP connection to a specified port. If the connection is successful, the probe passes.
* Command Execution (exec) – Runs a specified command inside the container. If the exit code is 0, the probe is considered successful.

These probes ensure applications run smoothly in Kubernetes by enabling self-healing and controlled traffic flow.

Container Logging in Kubernetes

### **Overview**

Container logging in Kubernetes involves collecting, accessing, and troubleshooting application logs generated by containers running in Pods. Logs are essential for debugging and monitoring application behavior.

### **Key Concepts**

* Pod Logs: Kubernetes captures logs written to stdout and stderr.
* Ephemeral Nature: Logs are lost if a Pod is deleted unless stored externally.
* Kubelet Role: The Kubelet manages logs for running containers.
* Centralized Logging: External tools (e.g., Fluentd, Elasticsearch, Loki) can aggregate logs.

### **Commands to View Logs**

1. View logs of a single-container Pod:

kubectl logs <pod-name>

1. View logs of a multi-container Pod:

kubectl logs <pod-name> -c <container-name>

1. Stream live logs:

kubectl logs -f <pod-name>

1. View logs from a previous failed container instance:

kubectl logs --previous <pod-name>

1. View logs of all containers in a Pod:

kubectl logs <pod-name> --all-containers

### **Best Practices**

* Use sidecar logging patterns for better log management.
* Store logs externally to prevent data loss.
* Integrate logging solutions for centralized log collection and analysis.

Monitoring Using Metrics Server in Kubernetes

### Overview

Metrics Server is a lightweight, in-cluster resource monitoring tool that collects CPU and memory usage of Pods and Nodes in Kubernetes. It enables features like Horizontal Pod Autoscaler (HPA) and kubectl top commands.

### **Key Concepts**

* Metrics API: Metrics Server provides resource usage data through the Kubernetes Metrics API.
* No Persistent Storage: It does not store historical data, only recent metrics.
* Required for Auto-Scaling: HPA relies on Metrics Server to scale Pods based on CPU or memory usage.

### **Enabling Metrics Server**

#### **On Minikube:**

# minikube addons enable metrics-server

#### **On Kind (Kubernetes in Docker):**

# kubectl apply -f https://github.com/kubernetes-sigs/metrics-server/releases/latest/download/components.yaml

After installation, ensure it's working with:

#kubectl get deployment metrics-server -n kube-system

### **Commands to View Metrics**

1. View node resource usage:

# kubectl top nodes

1. View pod resource usage:

# kubectl top pods

1. View pod resource usage in a namespace:

# kubectl top pods -n <namespace>

### **Troubleshooting**

* If kubectl top returns errors, check if Metrics Server is running:

kubectl get apiservices | grep metrics

* Restart Metrics Server if necessary:

kubectl rollout restart deployment metrics-server -n kube-system

### **Best Practices**

* Use Metrics Server with HPA for auto-scaling based on resource usage.
* Monitor node and pod utilization regularly to optimize cluster performance.
* Combine Metrics Server with external monitoring tools like Prometheus for advanced insights. (will discuss in CKA)

**Services**

In Kubernetes, a Service is a method for exposing a network application that is running as one or more [Pods](https://kubernetes.io/docs/concepts/workloads/pods/) in your cluster.

There will be three types of services mainly we gonna discuss

**1. NodePort**

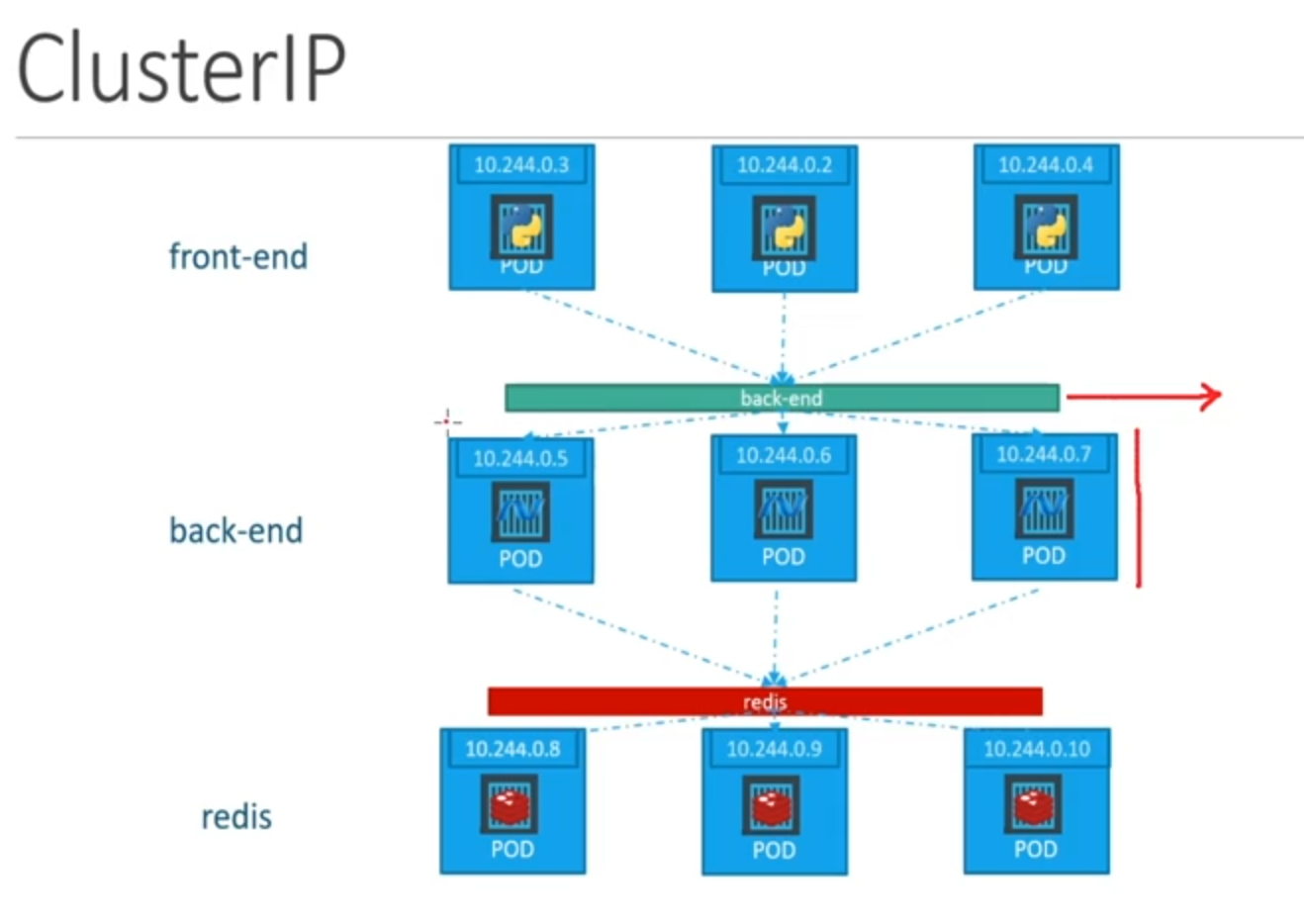
Exposes the Service on each Node's IP at a static port (the NodePort). To make the node port available to the public, Kubernetes sets up a cluster IP address, the same as if you had requested a Service of type: ClusterIP.

**2. Load balancer**

Exposes the Service externally using an external load balancer. Kubernetes does not directly offer a load balancing component; you must provide one, or you can integrate your Kubernetes cluster with a cloud provider.

**3. ClusterIP**

Exposes the Service on a cluster-internal IP. Choosing this value makes the Service only reachable from within the cluster. This is the default that is used if you don't explicitly specify a type for a Service.

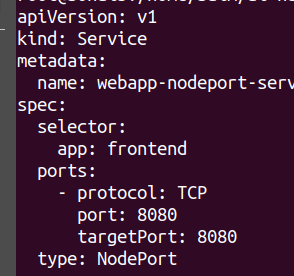


**Hands-On on Cluster-IP service and Nodeport services**

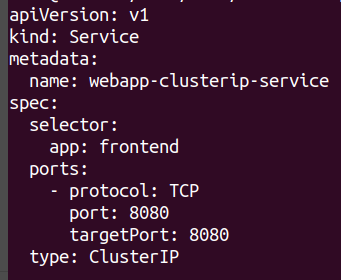
step#1 first make a deployment which will be a webapp

|  |  |
| --- | --- |
| apiVersion: apps/v1  kind: Deployment  metadata:  name: frontend  labels:  app: frontend | spec:  replicas: 3  selector:  matchLabels:  app: frontend  template:  metadata:  labels:  app: frontend  spec:  containers:  - name: webapp  image: kodekloud/simple-webapp:red  ports:  - containerPort: 8080 |

step#2 make a node-port service to check node-port is working

 Now apply this file and paste the node ip: and the nodeport on the browser and you will see webapp accessible on browser publicly **Note:-** selector should be same as the labels of pods otherwise it wouldn’t work can verify by checking the endpoints by describing the service

step#3 make a Cluset-IP service to Cluster-IP is working

 Now apply this file and paste the node ip and the target port by taking ssh into the node and see it is internally accessible example command:-

curl <node-ip>:<target-port>

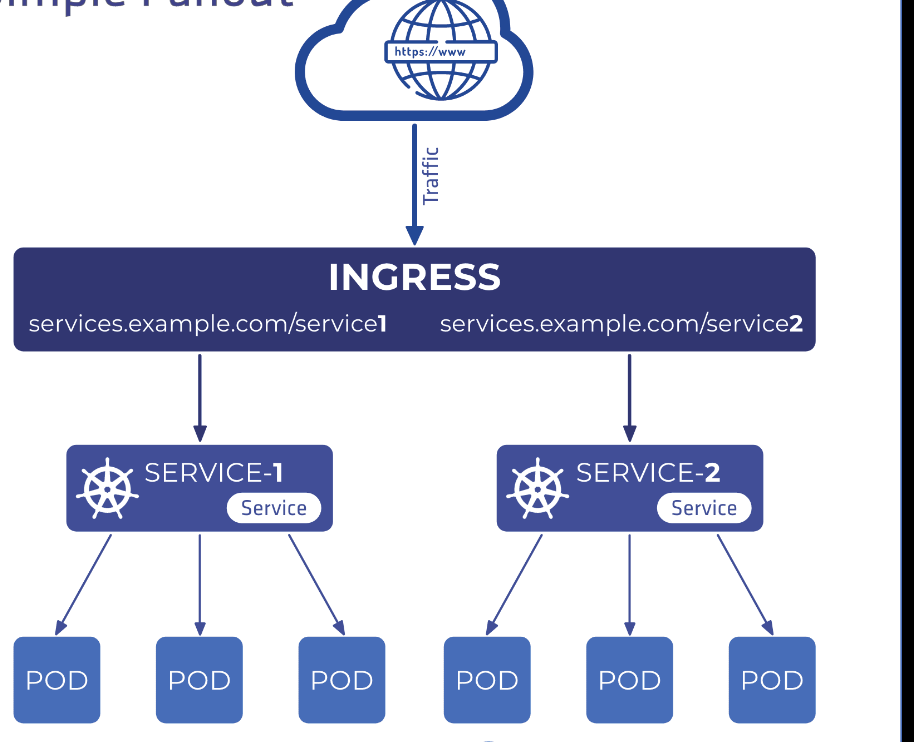
**Ingress Networking**

### What is Ingress Networking?

Ingress is a way to manage external access to services in a Kubernetes cluster. Instead of exposing each service with a separate LoadBalancer or NodePort, Ingress routes traffic through a single entry point using an Ingress Controller.

### What Does Ingress Do?

* Routes external traffic to the correct service inside the cluster.
* Provides a single access point instead of multiple exposed services.
* Enables advanced features like SSL/TLS termination, authentication, and load balancing.



### What Problem Does It Solve?

* Reduces the need for multiple LoadBalancers (saving cost).
* Provides cleaner and more manageable service exposure.
* Allows path-based and host-based routing (e.g., example.com/api → Service A, example.com/web → Service B).

## Lab to Practice

### Step 1: Install an Ingress Controller

If using Kind, first enable an Ingress Controller:

# kind-config.yaml

kind: Cluster

apiVersion: kind.x-k8s.io/v1alpha4

nodes:

- role: control-plane

extraPortMappings:

- containerPort: 80

hostPort: 80

- containerPort: 443

hostPort: 443

Apply the config:

kind create cluster --config kind-config.yaml

kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/main/deploy/static/provider/kind/deploy.yaml

### Step 2: Deploy a Test App

Create a simple web service:

# app.yaml

apiVersion: apps/v1

kind: Deployment

metadata:

name: web-app

spec:

replicas: 2

selector:

matchLabels:

app: web

template:

metadata:

labels:

app: web

spec:

containers:

- name: nginx

image: nginx

ports:

- containerPort: 80

---

apiVersion: v1

kind: Service

metadata:

name: web-service

spec:

selector:

app: web

ports:

- protocol: TCP

port: 80

targetPort: 80

Apply the YAML:

kubectl apply -f app.yaml

### Step 3: Create an Ingress Resource

Define how traffic should be routed:

# ingress.yaml

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

name: web-ingress

annotations:

nginx.ingress.kubernetes.io/rewrite-target: /

spec:

rules:

- host: myapp.local

http:

paths:

- path: /

pathType: Prefix

backend:

service:

name: web-service

port:

number: 80

Apply the Ingress rule:

kubectl apply -f ingress.yaml

### Step 4: Test the Ingress

Edit /etc/hosts (Linux/macOS) or C:\Windows\System32\drivers\etc\hosts (Windows) and add:

127.0.0.1 myapp.local

Now, open a browser and visit:  
👉 [**http://myapp.local**](http://myapp.local/)

You should see the Nginx welcome page, proving that Ingress is working!

## What are Network Policies?

Network Policies in Kubernetes control how pods communicate with each other and with external networks. They help enforce security by restricting or allowing traffic based on defined rules.

### What Do Network Policies Do?

* Allow or deny traffic between pods based on labels.
* Control ingress (incoming) and egress (outgoing) traffic.
* Improve security by preventing unauthorized access between services.

### What Problem Do They Solve?

* By default, all pods in Kubernetes can communicate with each other.
* Network Policies restrict access, ensuring that only authorized traffic flows between services.
* Helps follow security best practices (zero-trust networking).

## Lab to Practice

### Step 1: Create Required Pods and Services

Create payroll-pod.yaml:

apiVersion: v1

kind: Pod

metadata:

labels:

name: payroll

name: payroll

namespace: default

spec:

containers:

- env:

- name: APP\_NAME

value: payroll-application

- name: BG\_COLOR

value: blue

image: kodekloud/webapp-conntest

name: payroll

imagePullPolicy: Always

ports:

- containerPort: 8080

protocol: TCP

Create payroll-svc.yaml:

apiVersion: v1

kind: Service

metadata:

name: payroll-service

namespace: default

spec:

selector:

name: payroll

ports:

- protocol: TCP

port: 8080

targetPort: 8080

Apply the configurations:

kubectl apply -f payroll-pod.yaml

kubectl apply -f payroll-svc.yaml

### Step 2: Apply a Network Policy to Restrict Access

Create network-policy.yaml:

apiVersion: networking.k8s.io/v1

kind: NetworkPolicy

metadata:

name: payroll-policy

namespace: default

spec:

podSelector:

matchLabels:

name: payroll

policyTypes:

- Ingress

ingress:

- from:

- podSelector:

matchLabels:

name: internal

ports:

- port: 8080

protocol: TCP

Apply the Network Policy:

kubectl apply -f network-policy.yaml

### Step 3: Verify Network Policy

Try to access the payroll-service from another pod without the internal label:

kubectl run test --rm -it --image=busybox -- /bin/sh

wget --spider --timeout=2 payroll-service:8080

It should fail.

Now, create a pod labeled internal:

apiVersion: v1

kind: Pod

metadata:

labels:

name: internal

name: internal

namespace: default

spec:

containers:

- image: busybox

name: internal

command: ["sleep", "3600"]

Apply it:

kubectl apply -f internal-pod.yaml

Now, exec into this pod and retry the request:

kubectl exec -it internal -- wget --spider --timeout=2 payroll-service:8080

It should succeed, proving the Network Policy works!

Note:-

This last two topics is pretty much confusing, but very important so to understand this topic you could contact me on linkedin or via my phone number:+92 337 4005052