**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

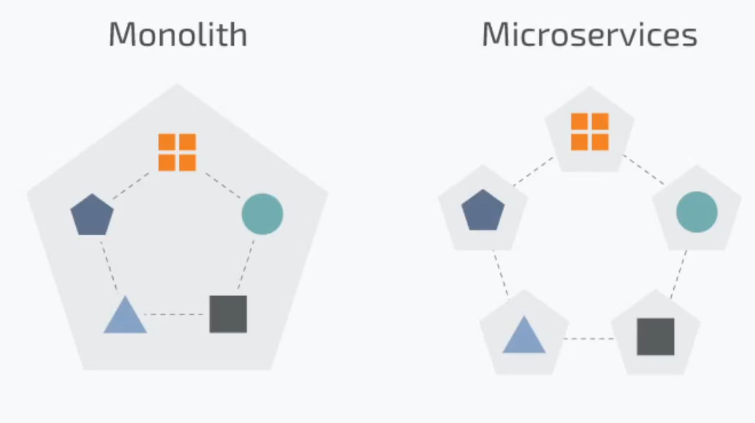
- Originally developed by google

- 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

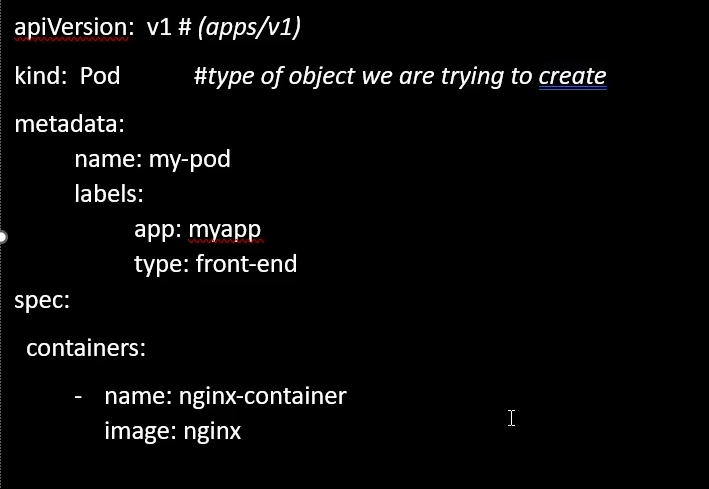
my-cluster

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**

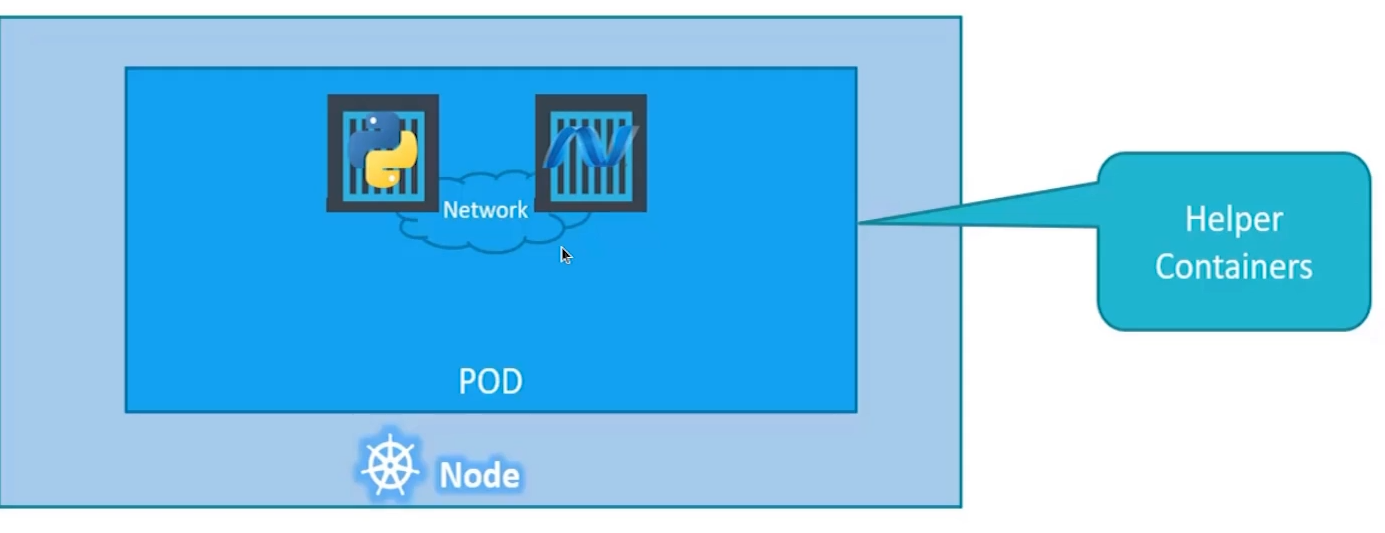
#kubectl describe pod my-pod

**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

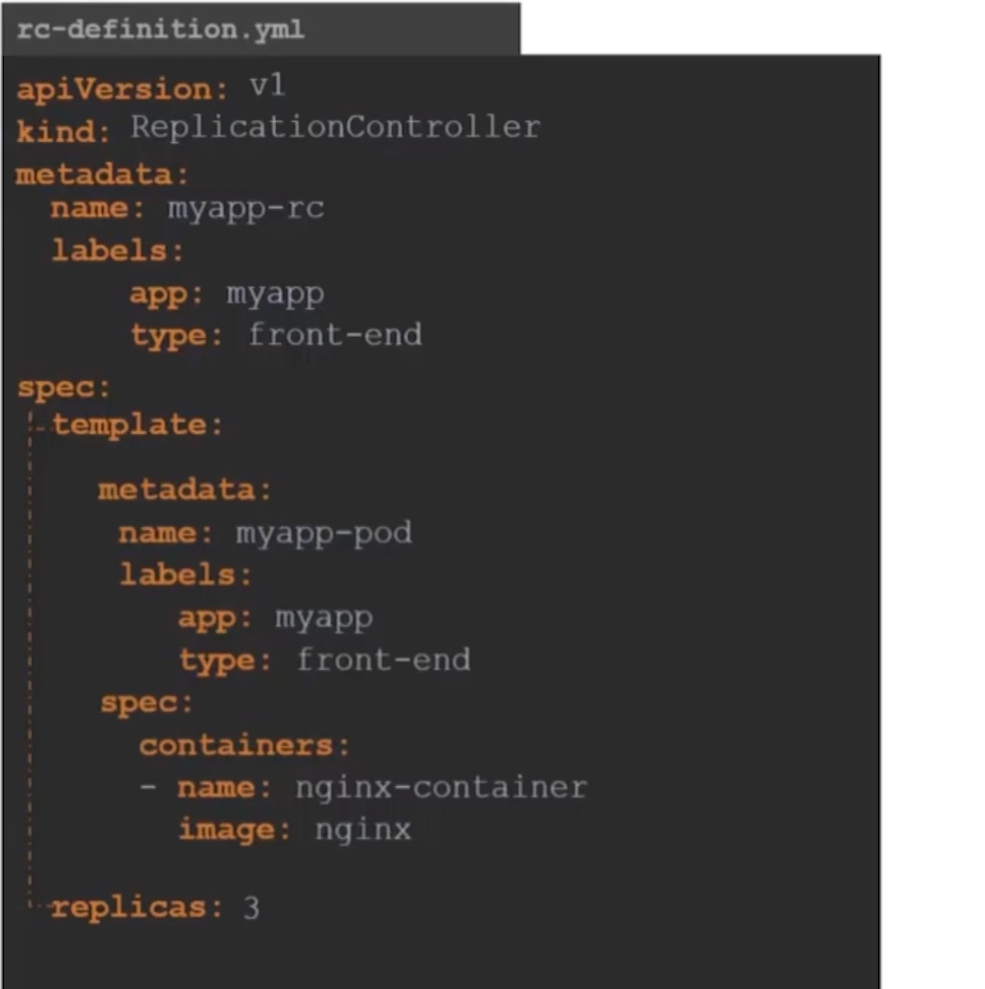
**Replication Controller**

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

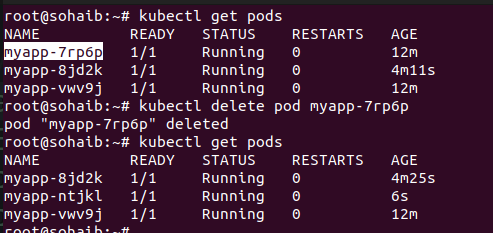
replicas: 3

# kubectl apply -f rc-configuration.yml

# kubectl get po,replicationcontroller

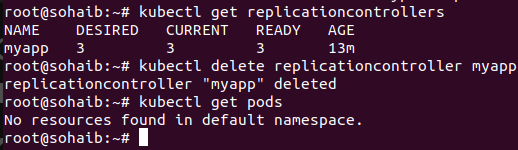
**what we learn from this hands-on?**

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.

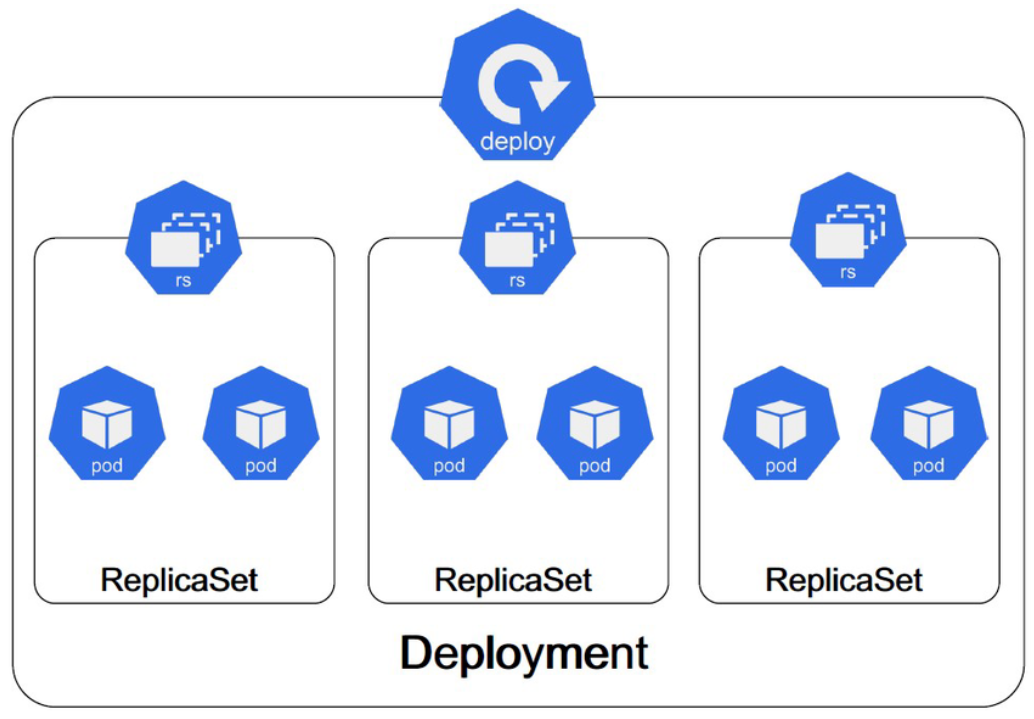
**Deployments In Kubernetes**

**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

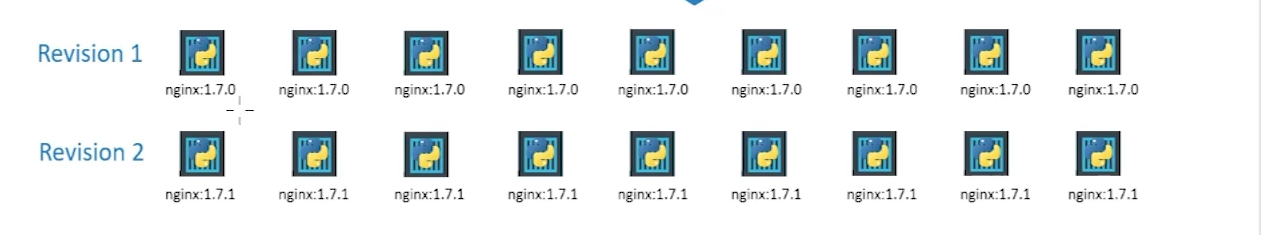
#kubectl edit deploy simple-deployment ==> make change in the api-server file which is an internal file

**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.

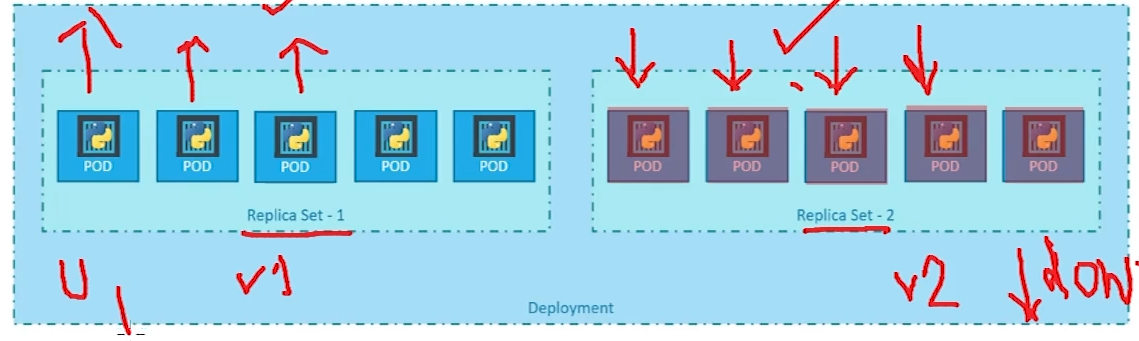
### **Deployments -- Updates and Rollbacks**

* 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.

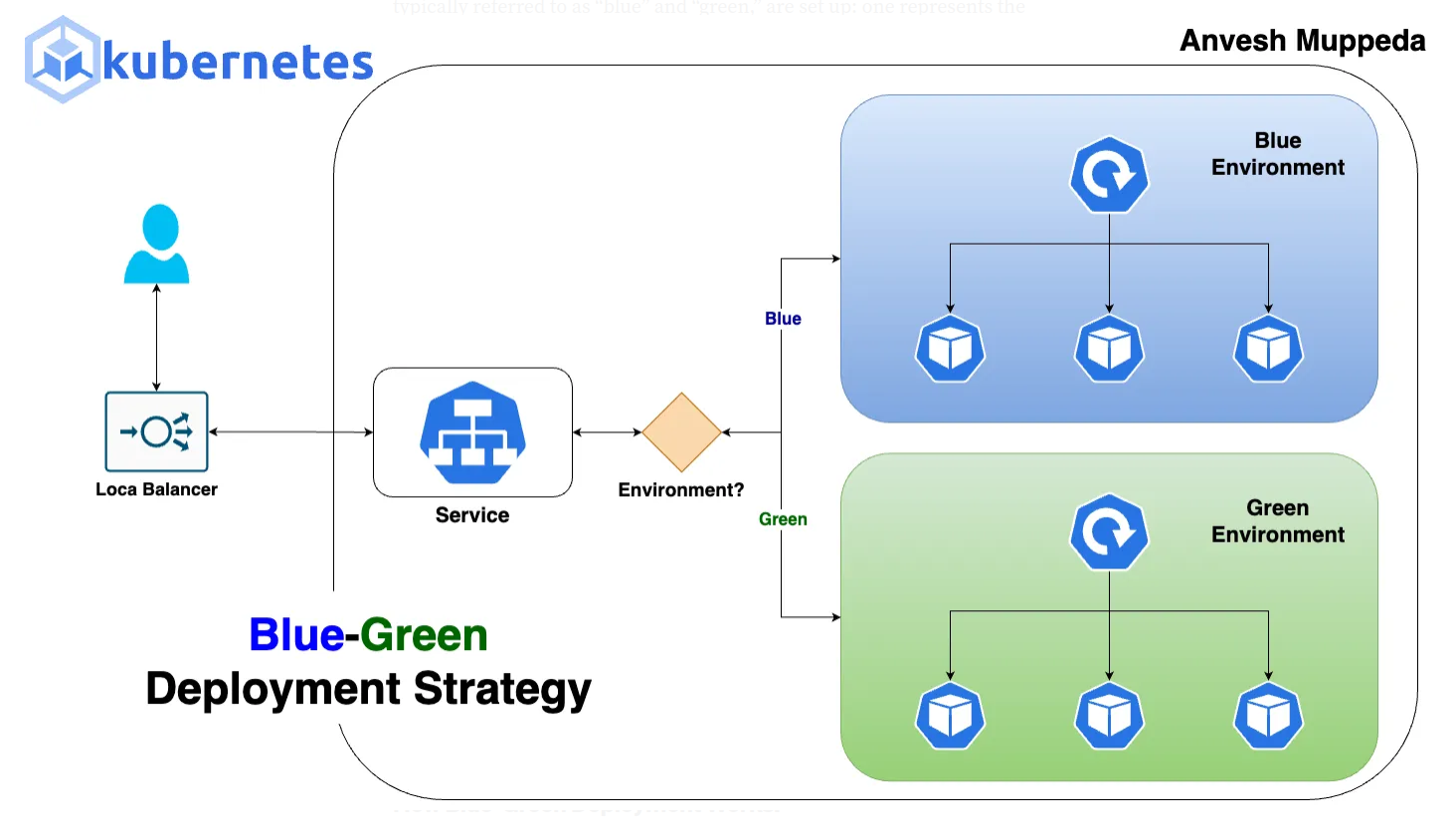
**Deployment strategies**

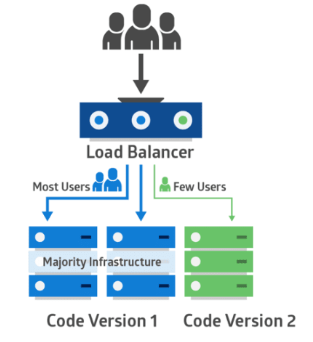
**What is a deployment strategy**

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

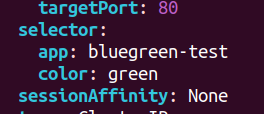
run the following curl commands and output **should be as following:**

# 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"]