KUBERNETES or K8’s –

Kubernetes is an open-source container orchestration system for automating software deployment, scaling, and management.

DOCKER and CONTAINER –

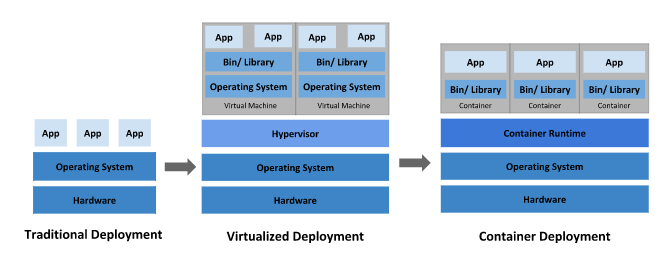
Docker is an open source containerization platform. It enables developers to package applications into *CONTAINERS*—standardized executable components combining application source code with the operating system (OS) libraries and dependencies required to run that code in any environment.

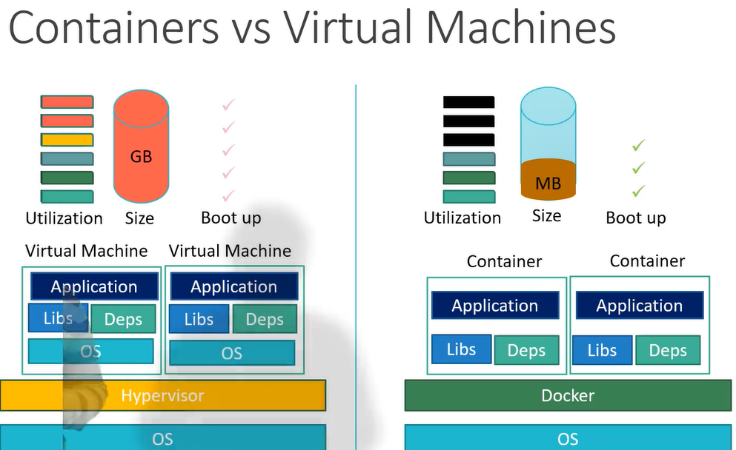
Q. Where can containers be stored ?

It can be stored either in the private repositories or it can be hosted in public repositories (Docker HUB)

**Docker command**: docker run: Creates a container.  
docker run –d - - name=redis redis (-d means detached and creates a container named redis, it is the memory db to be installed).  
docker run –d - - name=db postgres:9.4 (creates a container named db and installs image of postgres sql db of version 9.4).  
docker ps –a : Shows the total number of docker container created inside the docker host.  
docker stop <container name>: This will stop the container and will show status as exited.  
docker start <container name>: This will start the container   
docker rm <container name>: Removes a specific container.  
docker container prune: This will remove/delete all the containers.

Create a docker admin user: useradd dockeradmin  
Set password for docker admin: passwd dockeradmin





Cluster – Set of nodes (More than 1 node) grouped together. If 1 node fails, the other nodes will be up and will keep running the application. Having multiple nodes also helps in sharing the load as well. Multiple nodes together form a cluster

Master – Master is just another node with kubernetes installed inside in it and is configured as master. Information about all the cluster is stored inside the master and is responsible for managing the cluser. The master watches over the node in the cluster and is responsible for actual orchestration of containers on the worker nodes.

**Components of Kubernetes Architecture** –

1.Master node contains the following components - </>kube-apiserver, scheduler, controller, etcd, kubectl

2. Worker node contains – </>Kubelets,Kube-proxy, Containter runtime, Docker container, PODS

**Kubernetes package components** – **API server** (front-end)/**etcd** (key-value store <contains all metadata and config data>)/**kubelet/container runtime (**responsible for running application in containers)**/controller** (controls the pod lifecycle)**/scheduler** (responsible for distributing work or containers across multiple nodes)

**KUBELET**: The kubelet is a service agent that monitors the pod life cycle that ensures a given set of pods are always running as they should. It runs on each node and enables communication between master and slave nodes

**HEAPSTER**: It does cluster wide monitoring

**KUBECTL**: Command line tool for Kubernetes

**Kubectl commands** –

Kubectl run: Deploy an application on the cluster  
kubectl cluster-info: View information about the cluster   
kubectl get all: Provides information about all the cluster/pod/container and services created  
kubectl get nodes: To list all of the nodes part of the cluster  
kubectl create deployment hello-minikube  
kubectl get pods: Lists out all the pod details  
Kubectl describe pod <podname> : Provide the details of the pod engine  
kubectl get pods –o wide: Provide the details of the pod on which node is it running with IP address   
kubectl delete pods <podname> : Deletes the specific pod   
kubectl delete pods - -all: Delete all the pods inside the container  
kubectl run <pod name> - - image=<image name> : Creates a pod in the container of the image specified  
kubectl create –f <file name>.yml: Creates a pod in the container  
kubectl rollout restart <pod name>: This will restart the pod without taking it down

Example to create <nginx> image in the container: kubectl run nginx - - image=nginx

**NOTE**: We cannot deploy containers directly to kubernetes, which is why we need PODS, which are deployed 1st in the kubernetes cluster.

**KUBERNETES NAMESPACES** –

Namespaces in kubernetes provides a mechanism for isolating groups of resources within a single cluster.

Command: kubectl get namespace

NAME STATUS AGE

default Active 1d

kube-node-lease Active 1d

kube-public Active 1d

kube-system Active 1d

kubectl create namespace <name>: Creates a new namespace inside the cluster.

**Command to create a cluster**:

**eksctl create cluster \**

**--name *my-cluster* \**

**--version *1.22* \**

**--without-nodegroup**

Deployment of an application process: Deploy pods>Enable connectivity>Provide external access

**KUBERNETES POD** –

Pods are **the smallest, most basic deployable objects in Kubernetes**. A Pod represents a single instance of a running process in your cluster. Pods contain one or more containers, such as Docker containers. When a Pod runs multiple containers, the containers are managed as a single entity and share the Pod's resources.

A pod is a single instance of an application, smallest deployable object in Kubernetes.

Pods in a Kubernetes cluster are used in two main ways:

* **Pods that run a single container**. The "one-container-per-Pod" model is the most common Kubernetes use case; in this case, you can think of a Pod as a wrapper around a single container; Kubernetes manages Pods rather than managing the containers directly.
* **Pods that run multiple containers that need to work together**. A Pod can encapsulate an application composed of multiple co-located containers that are tightly coupled and need to share resources. These co-located containers form a single cohesive unit of service—for example, one container serving data stored in a shared volume to the public, while a separate *sidecar* container refreshes or updates those files. The Pod wraps these containers, storage resources, and an ephemeral network identity together as a single unit.

NOTE: When load increases on any application, we need to scale up additional instances of the web application

To scale up any instance, you need to deploy a new pod having containers of your application instances for balancing the load and likewise deleting the pod after the load decreases

Example of a pod running a single container image of nginx –

**apiVersion**: v1

**kind**: Pod

**metadata**:

**name**: nginx

**spec**:

**containers**: (Container here is an Array/list)

- **name**: nginx (Name and image are properties of container)

**image**: nginx:1.14.2

**ports**:

- **containerPort**: 80

Command for creating the pod: kubectl create -f <file name.yml>

**Properties inside the yaml file must contain** –

* apiVersion/kind/metadata (list)/spec (dictionary)

apiVersion:

kind:

metadata:

spec:

**Example – Create a kubernetes pod defination file using below values:**

Name: postgres

Labels: tier => db-tier

Container name: postgres

Image: postgres

**Solution** –

apiVersion: v1

kind: Pod

metadata:

name: postgres

labels:

tier: db-tier

spec:

containers:

- name: postgres

image: postgres

**Replication Controllers** –

A Replication Controller **ensures that a specified number of pod replicas are running at any one time**.

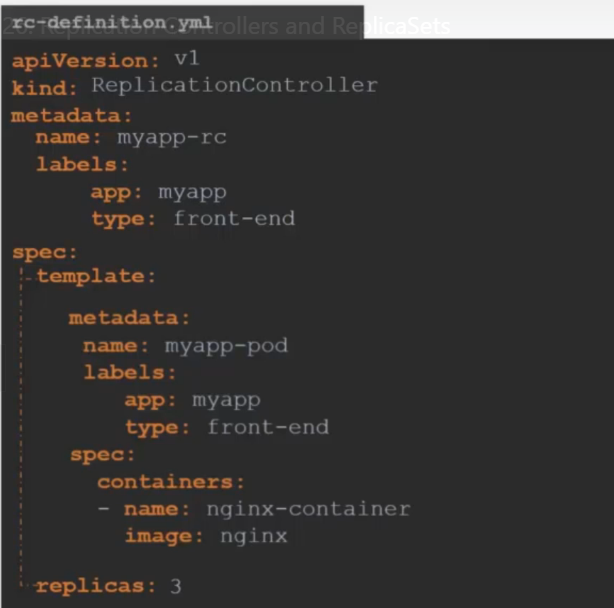
A Replica Set's purpose is to maintain a stable set of replica Pods running at any given time. As such, it is often used to guarantee the availability of a specified number of identical Pods.

If there are too many pods, the Replication Controller terminates the extra pods. If there are too few, the Replication Controller starts more pods. Unlike manually created pods, the pods maintained by a Replication Controller are automatically replaced if they fail, are deleted, or are terminated

Replication controller yaml definition –

Under spec properties, we need to define the pod template that we need to replicate. The template must begin from metadata properties of the pod we want to replicate.

Also define the no.of replicas needed: Below is mentioned as 3



RC Commands:

Command to create the replica : kubectl create –f <rc-definaion.yml>  
Command to view RC details created: kubectl get replicationcontroller

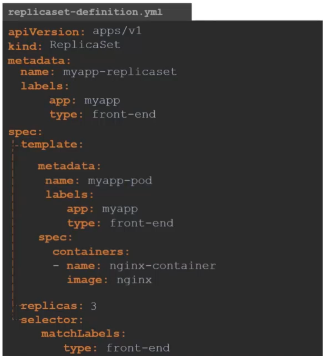
REPLICA SET –

A Replica Set's purpose is to maintain a stable set of replica Pods running at any given time. As such, it is often used to guarantee the availability of a specified number of identical Pods.

A ReplicaSet is **a process that runs multiple instances of a Pod and keeps the specified number of Pods constant**. Its purpose is to maintain the specified number of Pod instances running in a cluster at any given time to prevent users from losing access to their application when a Pod fails or is inaccessible.

The goal of the replicaset is to monitor the pods and if any of them were to fail, it will deploy new one’s. It is infact a process that monitors the pods overall.

The basic difference between replication controller and replica set is that, replica set will have a “selector” defination included in the syntax

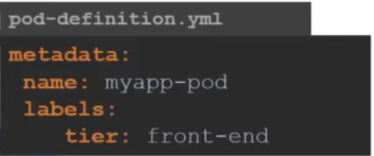
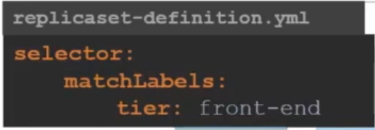
  
“Selector” definition is included at the end of replicas

RS Commands –

1. Create a replica set: kubectl –f <replicaset-defination.yml>  
2. To view the replica set created: kubectl get replicaset  
3. Command to edit/scale the replicas: kubectl scale replicaset - - replicas=X <file name.yml> (x number)   
4. Delete a replica set: kubectl delete replicaset <file name.yml> (Also deletes all underlying pods)  
5. Command to replace the replicaset: kubectl replace –f <file name.yml>  
6. Command to edit a replicaset: kubectl edit replicaset <file name>

Labels and Selectors –

Selector definition is defined for the replicaset to define which type of pod exactly to monitor.  
The type of pod is already has been defined in the label property.

Ex:   
 

Here the tier of the pod is defined as: front-end  
Hence the selector definition will have a “match label” with “type” defined as: front-end (Whatever app is defined in the pod template)

NOTE: “**SPEC**” section inside a replicaset /deployment yaml file will have 3 fields: replicas, template and selector (Parent-child property)

Sample replicaset definition file –

**apiVersion**: apps/v1

**kind**: ReplicaSet

**metadata**:

name: frontend

labels:

app: mywebsite

tier: frontend

**spec**:

**replicas**: 4

**template**: (POD Template is defined in this section)

metadata:

name: myapp-pod

labels:

app: myapp

spec:

containers:

- name: nginx

image: nginx

**selector**:

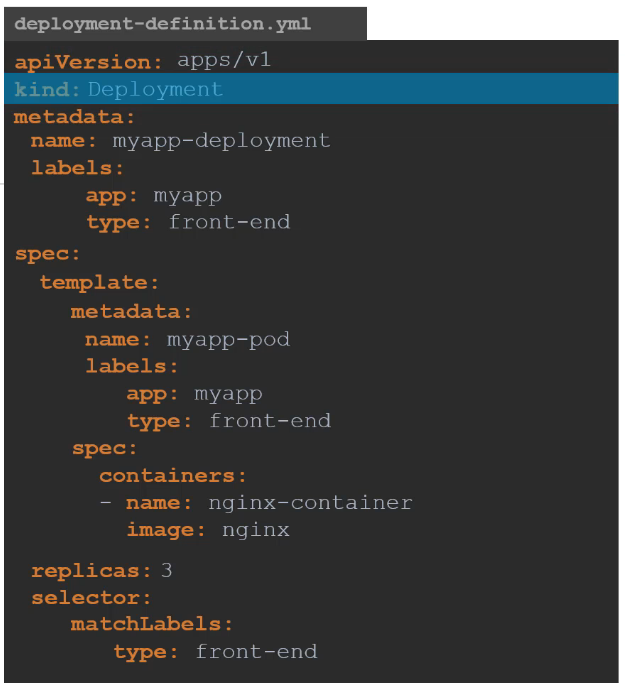
matchLabels:

app: myapp

Replicas – 4 (It will create 4 stable pods)

DEPLOYMENTS –

Deployment definition in yaml format:



**Deployment commands** –

Kubectl create –f <file name.yaml>: Create a deployment

Or

Kubectl apply –f <file name.yaml>: Creates the deployment

Example: Create a new deployment of file “**xyz.yaml**” located at /root/

Answer: kubectl create –f/root/**xyz.yaml**

kubectl get deployments: To view the deployments   
kubectl get all: To view all the deployments, replica sets and pods created  
kubectl describe deployment <deployment file>: Provided information of the deployment

Create an explicit deployment file of Nginx image using 3 replicas:

Kubectl create deployment - - <name-deployment> - -image=nginx - -replicas=3

**Sample deployment YAML File** –

**1.apiVersion**: apps/v1

**2.kind**: Deployment

**3.metadata**:

name: myapp-deployment

labels:

app: myapp

tier: frontend

**4.spec**:

**replicas**: 4

**template**: (POD template Definition)

metadata:

name: myapp-pod

labels:

app: myapp

spec: (Container template definition)

containers:

- name: nginx

image: nginx

**selector**:

matchLabels:

app: myapp

**UPDATES & ROLLBACKS** –

Rollout command: kubectl rollout status <deployment file name>  
Rollout history: kubectl rollout history <deployment file name>  
rolling back a deployment: kubectl rollout undo <deployment file name>

Delete a deployment: kubectl delete deployment <file name>

Editing the image version inside the deployment: kubectl set image deployment <file name> nginx= nginx:version

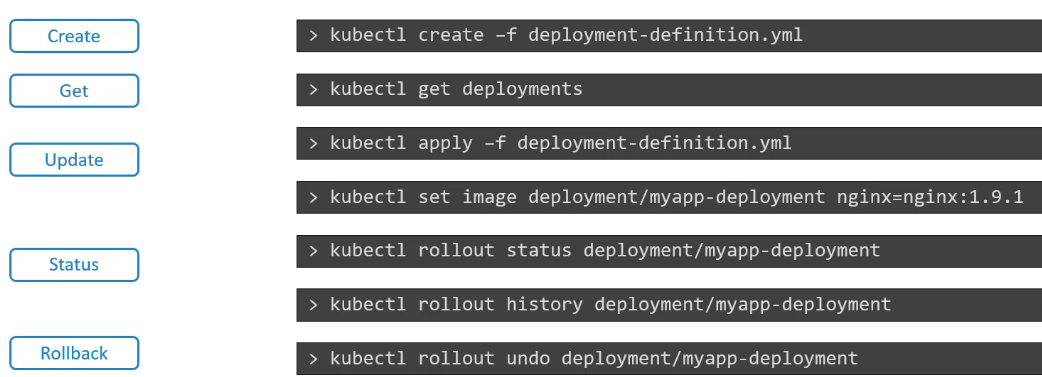
OR

Edit any deployment file: Kubectl edit deployment <file name>

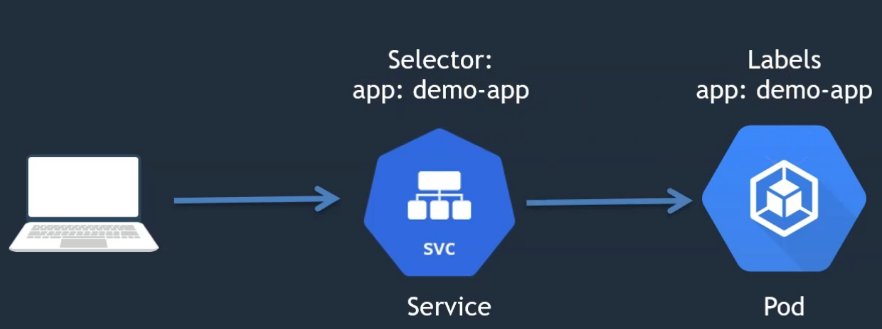
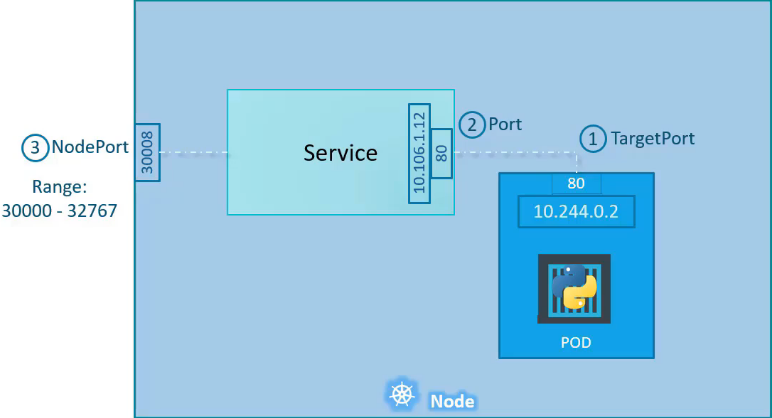
Create HPA (Horizontal pod autoscaler):

kubectl create autoscale deployment <file name> - -cpu-percent=50 - -min=1 - -max=10

**Commands**

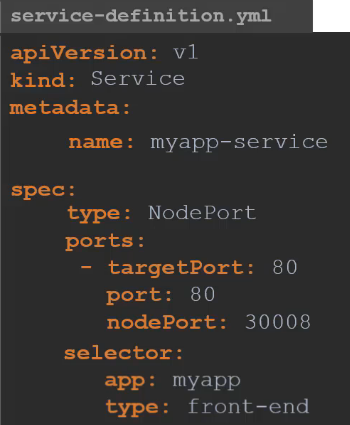


SERVICE – Node Port: (Enabling communication/connectivity)



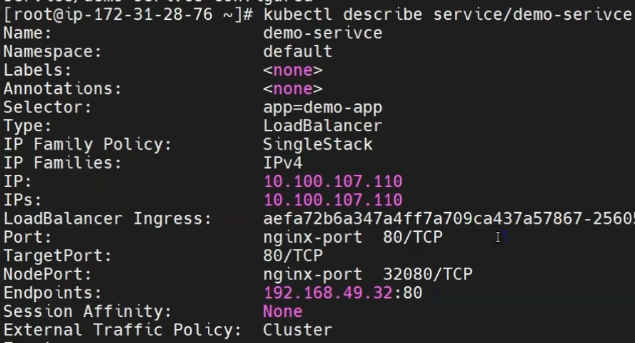
Whenever there is an external request, to establish a connection between a service and a pod, the service file should have a selector label and the pod should have the same label app name.

Since there would be many pods inside the containers, services will not be able to establish connectivity with a particular pod, unless the selector is specified same.



Services commands –

Command to create service definition file: kubectl create –f <file name.yaml>.  
Command to view all the service files: kubectl get services  
Command to access the web browser/service: curl <http://ip address (node)>   
Command to view the URL of the service file created: minikube service <file name> - - url  
Command to view the details of the service file/ Endpoints: kubectl describe service



**Service definition yaml file** –

**apiVersion**: v1

**kind**: Service

**metadata**:

name: frontend

labels:

app: myapp (Pod name and selector name should match to establish connectivity)

**spec**:

type: NodePort

ports: (Ports is an array/list)

- port: 80 (Port and targetPort will be the properties defined)

targetPort: 80

selector:

app: myapp(Selector name and app pod name should match to establish connectivity)

**Example 2** –

Use the below information to create a backend service definition file:

* Service name: image-processing
* Labels: app=> myapp
* Type: ClusterIP
* Port on the service: 80
* Port exposed by image processing container: 8080

**Solution-**

**apiVersion**: v1

**kind**: Service

**metadata**:

name: image-processing

labels:

app: myapp

**spec**:

type: ClusterIP

ports:

- port: 80

targetPort: 8080

selector:

tier: backend

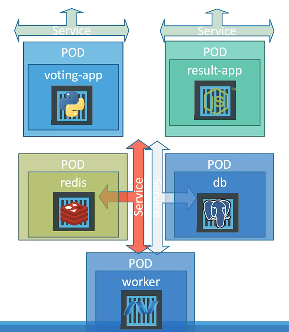
Q. **Create a service definition for nginx deployment which serves on port 80 and connects to container on 8080**

Kubectl expose deployment <deployment-file> nginx - - port=80 - - target-port=8080 (Deployment was exposed as service on port 8080)

Kubectl get services to view the service file created

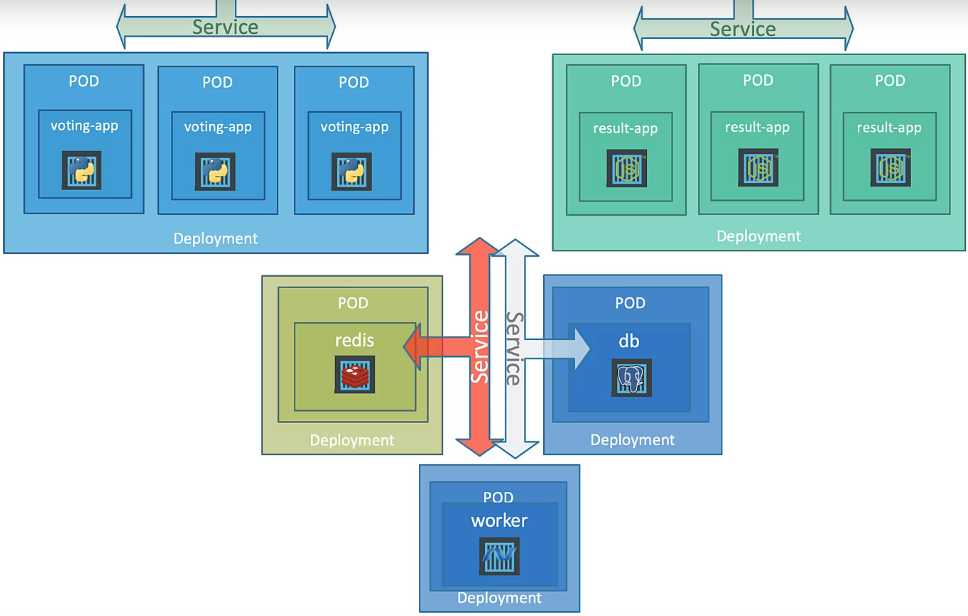
What are the types of services in Kubernetes: ClusterIP (Default service can only be seen internally not exposed, cannot be reached externally), NodePort (Exposed to a target port), LoadBalancer (It provide the service by taking the LB provided by cloud service (aws,gcp,azure), External Name

**Deploying Micro-services application on Kubernetes** –

-

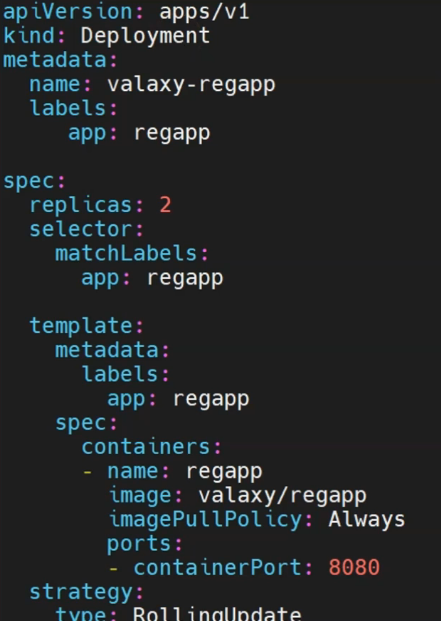
Create Pods (Containing images of web app instances) > Create their services (Enabling connectivity between pods) > Create a worker node> Provide external access (Node-Port)

It is always advised to use a deployment version definition file instead of pods/service file for deploying an application as it enables scaling your pods as per our requirements. (Deployment file contains replicasets). Hence, if any of our pod goes down, other pods will be active and running. Also it contains provision of rolling back, updating versions.



Kubernetes on Google Cloud: <https://cloud.google.com/kubernetes-engine/docs/>

Sample Deployment definition file –



1st section defines the Deployment name (valaxy-regapp) and deployment labels (regapp)

2nd section provides the required replica set (i.e. 2) it is going to create matching the pod defined in the file

3rd section (template) provides the definition of the pod created, and creates a pod in the backend. This section is divided into pod and container definition

4th section (strategy) defines the updating strategy (Here **RollingUpdate** means there is no downtime while deploying)

Strategy:  
 type:RollingUpdate  
 maxSurge: 1  
 maxUnavailable: 1

CRON JOBS –

A cron job is a Linux command used for scheduling tasks to be executed sometime in the future. Cron Jobs allow you to automate specific commands or scripts on your server to complete repetitive tasks automatically.

Q. How to stop a kubernetes job that should finish in 40seconds exceeds the time limit?

activeDeadlineSeconds: 200

By defining this flag, the job will be terminated once it reaches the defined threshold.

**FAQs**

1. **How to test a manifest without executing it?**

A: kubectl create –f <file name>.yml - - dry-run

2. **How do you initiate a rollback of an application?**

A: kubectl rollout undo <deployment file name>.yml

3. **How do you package kubernetes applications?**

A: Helm is a package manager which allows users to package, configure and deploy application and services to the kubernetes cluster.  
helm init # when executed this command, client is going to create a deployment in the cluster and that deployment will install the tiller, the server side of helm.

Helm search redis : Searches for a specific application  
Helm install redis: Install the particular application  
helm ls: Lists out all the applications

4. **Node Affinity vs Pod Affinity**

A: Node affinity is a set of rules used by the scheduler to assign the pods to a particular node using Node Affinity**.**

Pod affinity/anti-affinity allows you to constrain which nodes your pod is eligible to be scheduled on based on the labels on other pods.

5. **Ingress vs Egress in Kubernetes**

A.  An Ingress is an object that allows access to your Kubernetes services from outside the Kubernetes cluster. Ingress is incoming traffic to the pod, and Egress is outgoing traffic from the pod.

6. **Stateful set vs Stateless set in Kubernetes**

A. The key difference between stateful and stateless applications is that stateless applications don't “store” data. On the other hand, stateful applications require backing storage