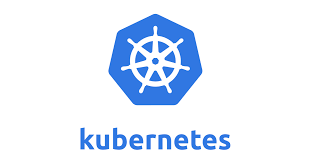
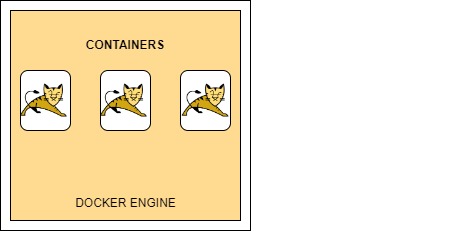
**KUBERNETES**

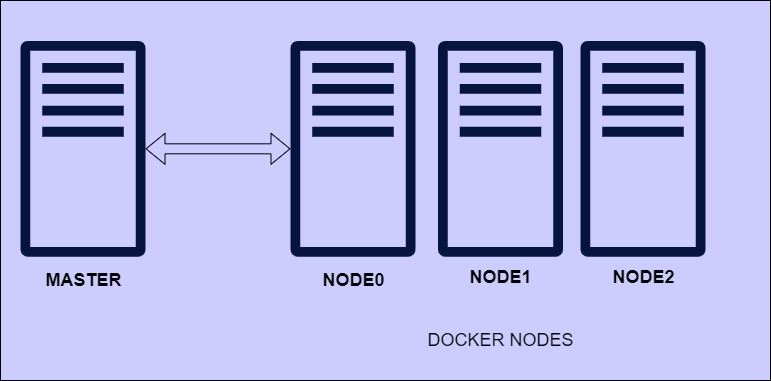
[Kubernetes](https://kubernetes.io/docs/concepts/overview/), also known as K8s, is an open-source system for automating deployment, scaling, and management of containerized applications.

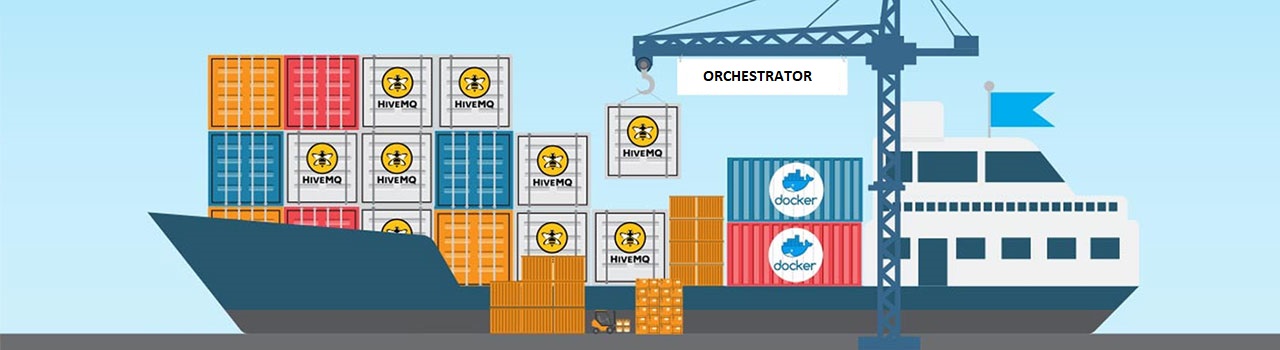
Also known as most famous container orchestration tool.

**Docker Engine**



**CLUSTERING**



**Container Orchestration**

**Orchestration Tools**

* **Docker Swarm**

**Self-Managed / Unmanaged Kubernetes**

* **Kubernetes**
* **OpenShift**
* **Rancher**
* **Mesos**

**Managed Kubernetes**

* **AWS EKS & ECS**
* **Azure AKS**
* **Google Container Engine (GKE)**

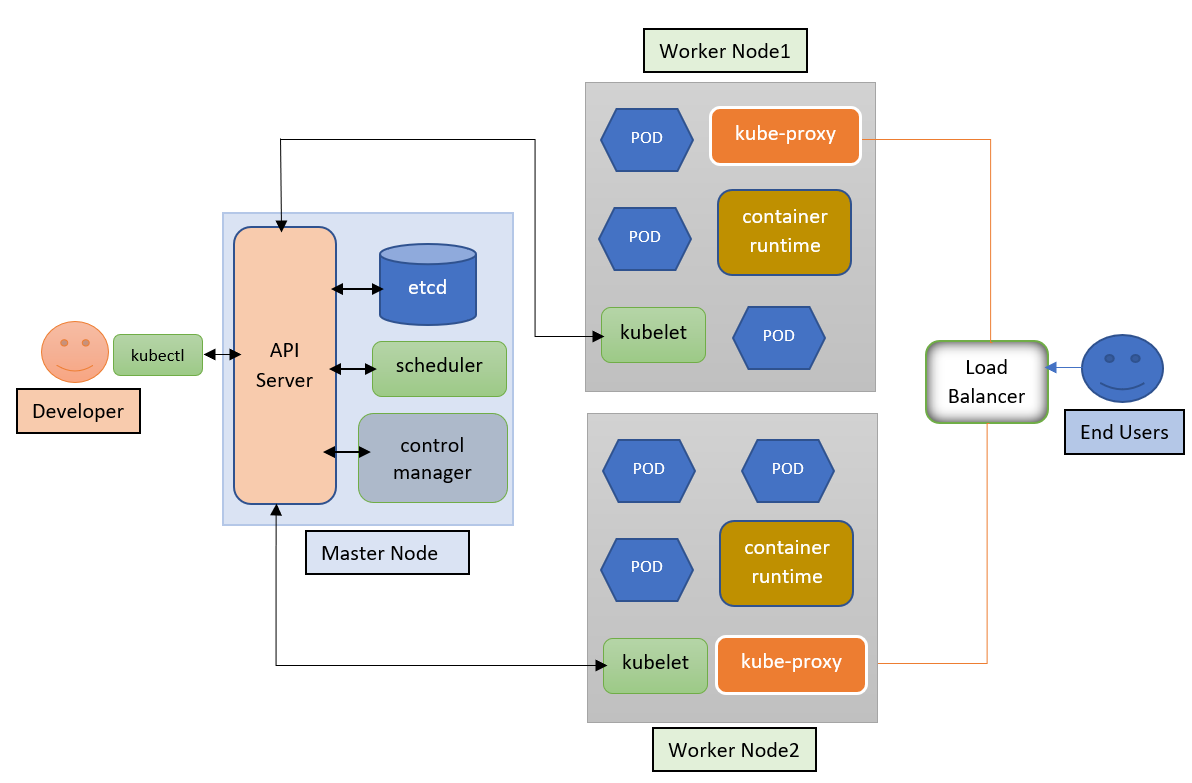
**Kubernetes History**

* Created by **Google** to manage their containers AKA Borg
* **Mid-2014:** Google introduced Kubernetes as an open-source version of Borg
* **July 21-2015:** **Kubernetes v1.0** gets released. Along with the release, google partnered with the Linux Foundation to form the **Cloud Native Computing Foundation (CNCF)**.
* **2016:** Kubernetes Goes Mainstream!
  + Kops, Minikube, Kubeadm etc
  + September 29: **Pokeman GO! Kubernetes Case Study Released**
* **2017:** Enterprise Adoption
  + Google and IBM announce Istio
  + GitHub runs on Kubernetes
  + **Oracle joined** the Cloud Native Computing Foundation

**Kubernetes Provides**

* Service discovery and load balancing
* Storage orchestration
* Automated rollouts and rollbacks
* Automatic bin packing
* Self-healing
* Secret and configuration management

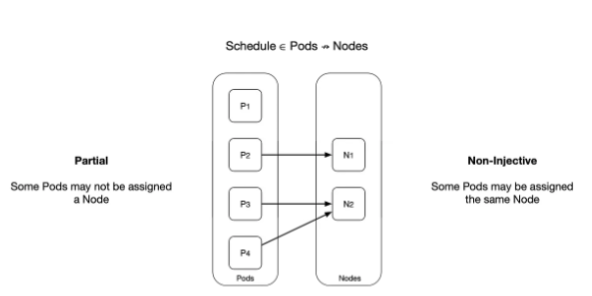
**Kubernetes Architecture**

****

**Master Node / Control Plane Components**

* **Kube API Server:** Main Hero! Handles all the requests and enables communication across stack services. It is the front end of Kubernetes control plane.

Admins connect to it using **Kubectl CLI**

* **Kube Scheduler:** It watches newly created pods that have no node assigned, selects a node from them to run on
  + Factors taken into account for scheduling decisions include
    - Individual and collective resource requirements
    - Hardware/software/policy constraints
    - Affinity and anti-affinity specifications
* **ETCD Server:** It stores all the information. Consistent and highly-available key value store used as Kubernetes backing store for all the cluster data.

Kube API stores retrieves information from it.

Should be backed up regularly

* **Controller Manager:** It manages various **controllers** in Kubernetes.

1. **Node Controller**

It is responsible for onboarding new nodes to the cluster handling situations where nodes become unavailable or get destroyed to keep our application running.

If it stops receiving signals from a node, the node is marked unreachable but it waits for 40 seconds before marking it unreachable. After the node is marked unreachable it waits for 5 minutes to come back up if it doesn’t, it removes the [POD’s](https://kubernetes.io/docs/concepts/workloads/pods/) assigned to that node and provisions them on the healthy nodes if [POD](https://kubernetes.io/docs/concepts/workloads/pods/)s are part of the replica set.

1. **Replica Controller**

It is responsible for monitoring the status of replica sets and ensuring that the desired number of [PODs](https://kubernetes.io/docs/concepts/workloads/pods/) are available at all times within the set. If a POD dies it will create another POD. It makes sure that desired number of PODs or at least one POD is in running state. It has the capability to bring up or down the specified no of PODs.

1. **Endpoints Controller**

Populates the Endpoints object (that is, joins services & pods)

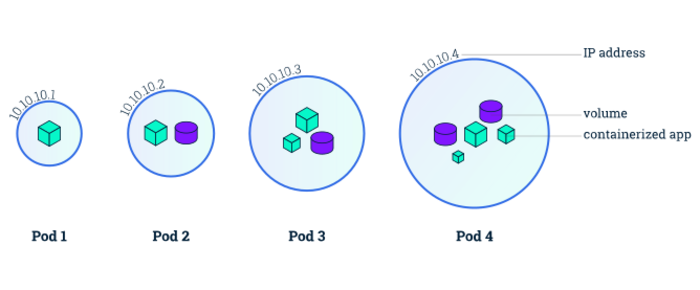
1. **Service Account & Token Controllers:**

Create default accounts and API access tokens for new namespace

**Worker Node Components**

* **Kubelet:** An agent that runs on each node in the cluster. It makes sure that containers are running in a pod.
* **Kubeproxy:** Network proxy that runs on each node in your cluster.
  + Network rule
    - Rules allow network communication to your pods inside or outside of your cluster.
* **Container Runtime:** Kubernetes supports several container runtime
  + Docker
  + Containerd
  + Cri-o, rktlet
  + Kubernetes CRI (Container Runtime Interface)

**What is POD in Kubernetes**

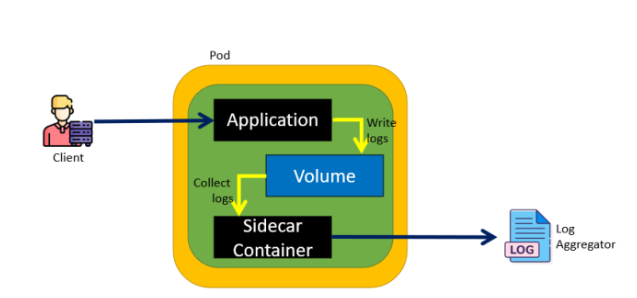


**Use of Multi Container POD:**

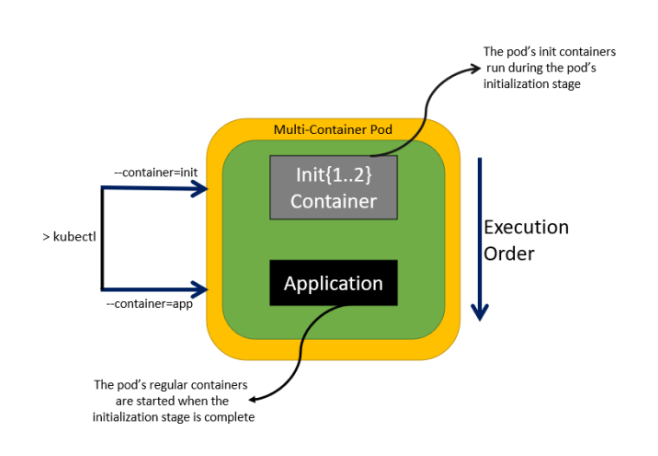
It depends on requirement.

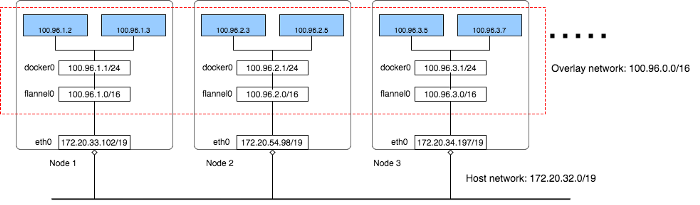
Example 1: Side Car Container along with application container

A sidecar is just a container that runs on the same pod as the application container. It shares the same volume and network as the application container, it can “help” or enhance how the application operates.



Example 2: Init Container is short lived container; it will start does some command execution and then it will be dead. Once it dead the main/application container will start.



**POD Networking**

**Kubernetes Setup**

* **Hard way:** Manual setup
* **Minikube:**
  + One node Kubernetes Cluster on your laptop
* **Kubeadm:**
  + Multi node Kubernetes Cluster
  + Can be treated on any Platforms VM’s, EC2, Physical machines etc.
* **Kops:**
  + Multi node Kubernetes Cluster on AWS
* **Kind:**
  + Supports multi–Node Cluster on your laptop using docker containers
* **GKE:** 
  + Supports multi-node cluster on Google cloud

**Setup with Minikube**

**Pre-requisites:**

1. Chocolatey

<https://chocolatey.org/install>

1. Oracle virtual box

<https://www.virtualbox.org/wiki/Downloads>

1. Install Docker Desktop & open it from start menu

<https://docs.docker.com/desktop/install/windows-install/>

**Steps**

* Open PowerShell as Administrator
* Install with Chocolaty

**$ choco install minikube Kubernetes-cli -y**

* Open Powershell again

**$ minikube start**

**Verify it by running below command**

$ kubectl get nodes

$ cat .kube/config

<https://minikube.sigs.k8s.io/docs/start/>

<https://minikube.sigs.k8s.io/docs/handbook/deploying/>

Example Deployment:

$ kubectl create deployment hello-minikube1 --image=kicbase/echo-server:1.0

$ kubectl expose deployment hello-minikube1 --type=NodePort --port=8080

$ minikube service hello-minikube1 –url

Cleanup the Minikube setup

$ minikube stop

$ minikube delete

**Setup Kubernetes with Kops**

**Pre-requisites**

1. Domain for Kubernetes DNS records
   * E.g., sscademy-k8s.com
2. Create a Linux VM and setup
   * Kops, kubectl, ssh keys, awscli
3. Login to AWS account and setup
   * S3 bucket, IAM User for AWSCli, Route53 Hosted Zones.
4. Login to Domain Registrar (E.g. Hostinger.com, GoDaddy.com)
   * Create NS records for subdomain pointing to Route 53 hosted zone NS servers

**Let’s start the setup**

1. **Login to AWS cloud**
   * Launch an EC2 instance
     + Give name as - Kops
     + Select Ubuntu 20.04 Image
     + Select Server type – t2.micro
     + Create a security group
       - Name – kops-SG
       - Allow port 22 from anywhere
     + Create a security key
     + Click on Launch instance
2. **S3 Bucket**
   * Create a s3 bucket
   * sscademy-kops-state
   * Click on Create
3. **Create an IAM user**
   * Give username as – kopsadmin
   * Select Programmatic access
   * Next
   * Attach policy – AdministratorAccess
   * Create user
   * Download th .csv credential file
4. Create Hosted Zone (sub domain)
   * Click on Create Hosted Zone
   * Type sub domain name as – k8s.sscademy.in
   * Select Public hosted zone
   * Click on Create hosted zone
5. Add the NS server records in domain registrar
   * Login to Hostinger.com/godaddy.com
   * Add 4 NS records for your domain
   * Select record type – Nameserver
   * Host – k8s
   * Points to – copy & paste the NS server
   * Likewise create 4 records
6. Login to EC2 instance which we have created
   * Generate the ssh keys
     1. $ ssh-keygen
   * Install AWS cli
     1. $ sudo apt update && sudo apt install awscli -y
     2. $ aws configure
        1. Copy & paste the access key and secret access key
        2. Enter the region – us-east-1
        3. Output format – json
   * Install kubectl

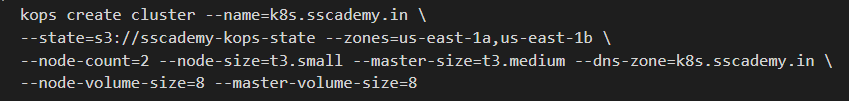
<https://kubernetes.io/docs/tasks/tools/install-kubectl-linux/>

* + Install kops

<https://kubernetes.io/docs/setup/production-environment/tools/kops/>

* + Select Linux OS
  + Follow the instructions to install kops
  + Verify it using command – kops –help
  + Verify the domain
    1. $ nslookup -type=ns k8s.ssdevops.in

1. Now let’s create the cluster. For that follow the below command





After this we have to wait for 15min, then we can verify it.

$ kops validate cluster –state=s3://sscademy-kops-state

If everything is good, then it will say that your cluster is ready

$ cat ~/.kube/config

$ kubectl get nodes

$ kubectl get pods

**To Delete the Kops cluster run below command**



**Kubernetes Objects**

* POD
* Service
* Replica Set
* Deployment
* Config Map
* Secret
* Volumes

**Kubeconfig File**

Use kubeconfig files to organize information about

1. Clusters
2. Users
3. Namespaces
4. Authentication Mechanisms

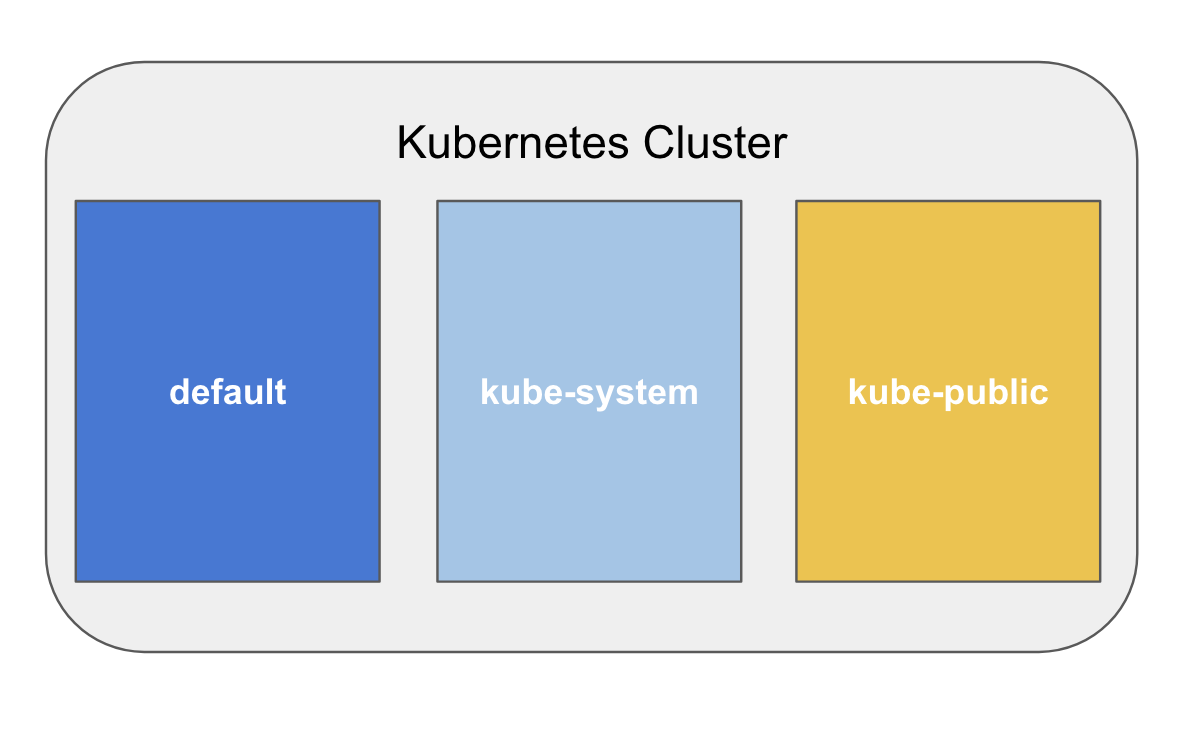
You will find the kube **config** file under the directory called **.kube**. Most of the time it will be in user home directory.

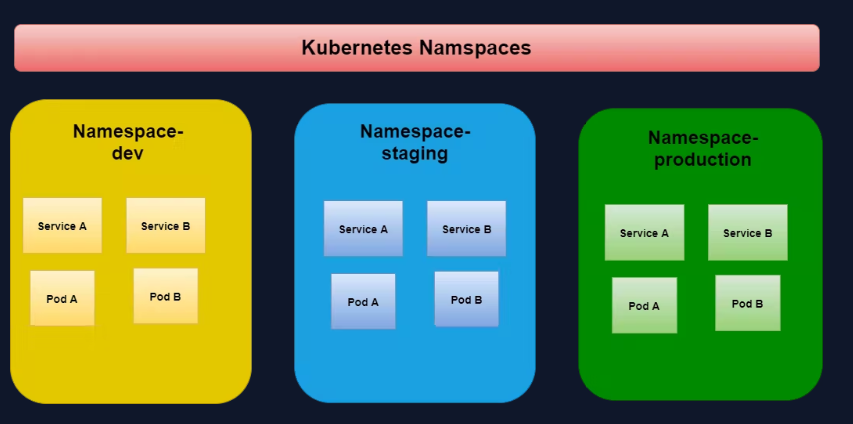
$ cat .kube/config

<https://kubernetes.io/docs/concepts/configuration/organize-cluster-access-kubeconfig/>

**Namespaces**

Group the resources using Namespaces





<https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/>

$ kubectl get ns

$ kubectl get all

$ kubectl get all –all-namespaces

$ kubectl get svc -n kube-system

$ kubectl create ns devops

$ kubectl run nginx –image=nginx -n devops

$ kubectl apply -f pod1.yaml

$ kubectl delete ns devops

**PODS**

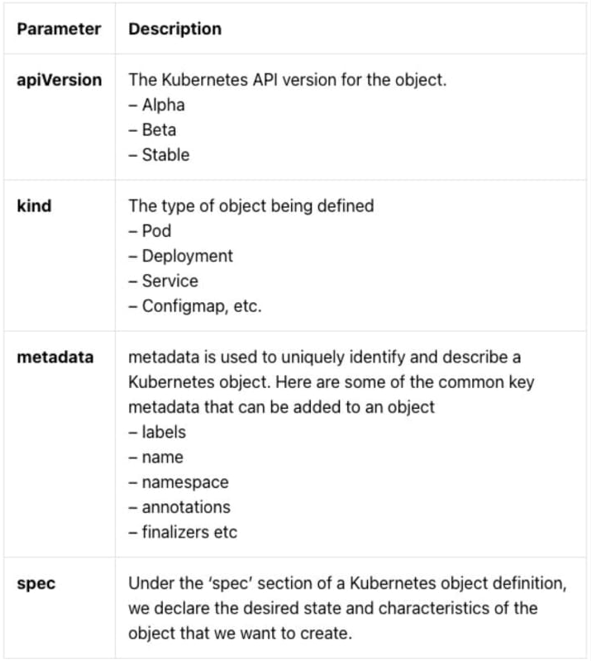
Run your apps isolated

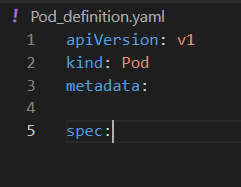
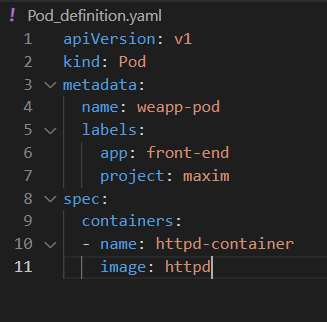
A Pod is the basic execution unit of a Kubernetes application – the smallest and simplest unit in the Kubernetes object model that you create or deploy.

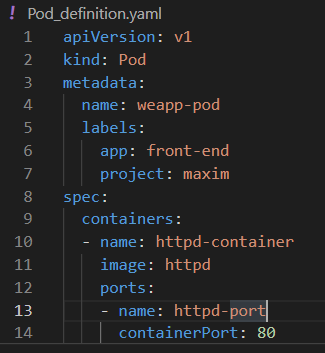
A Pod represents process running on your Cluster.

* **Pods that run a single container**
  + The “one-container-per-Pod” model is the most common Kubernetes use case
  + Pod as a wrapper around a single container
  + Kubernetes manages the Pods rather than the container directly
* **Multi Container Pod**
  + Tightly coupled and need to share the resources
  + One Main container and other as a sidecar or init container
  + Each Pod is mean to run a single instance of a Given application
  + Should use multiple Pods to use horizontally.

**POD Definition YAML**







**Create POD and get POD Information**

$ kubectl apply -f pod1.yaml

$ kubectl get pod

$ kubectl describe pod webapp-pod

$ kubectl get pod webapp-pod -o yaml

$ kubectl get pod webapp-pod -o yaml > webpod-definition.yaml

$ kubectl edit pod webapp-pod

$ kubectl get pods -o wide

$ kubectl logs weapp-pod -n dev

**SERVICES**

Connect with or To your POD

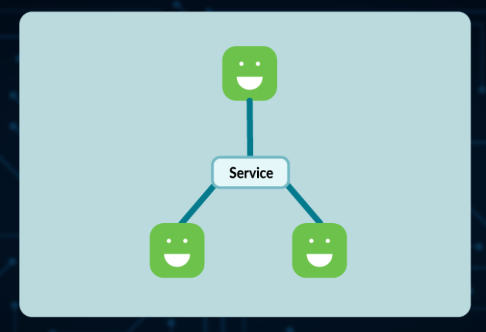
In Kubernetes, a Service is a method for exposing a network application that is running as one or more [Pods](https://kubernetes.io/docs/concepts/workloads/pods/) in your cluster.

**Types of Services in Kubernetes**

1. ClusterIP (default)
2. NodePort
3. LoadBalancer

**ClusterIP**

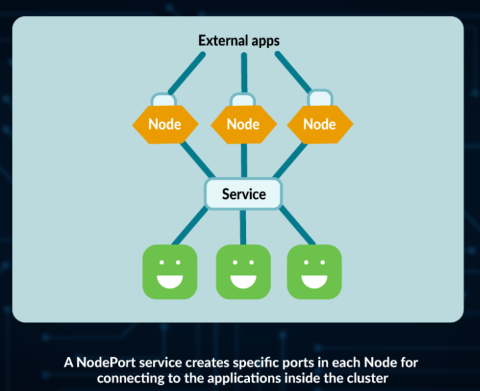
A ClusterIP service is the default Kubernetes service. It gives you a service inside your cluster that other apps inside your cluster can access. There is no external access.



**NodePort**

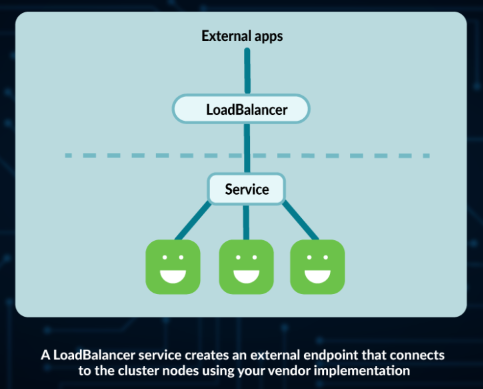
A NodePort differs from the ClusterIP in the sense that it exposes a port in each Node

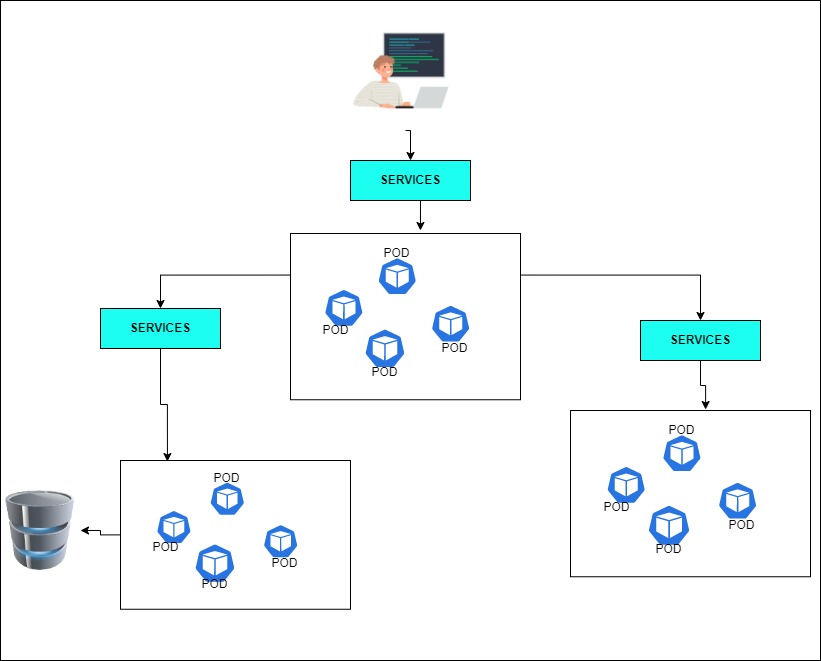
NodePort range : 30000 - 32767



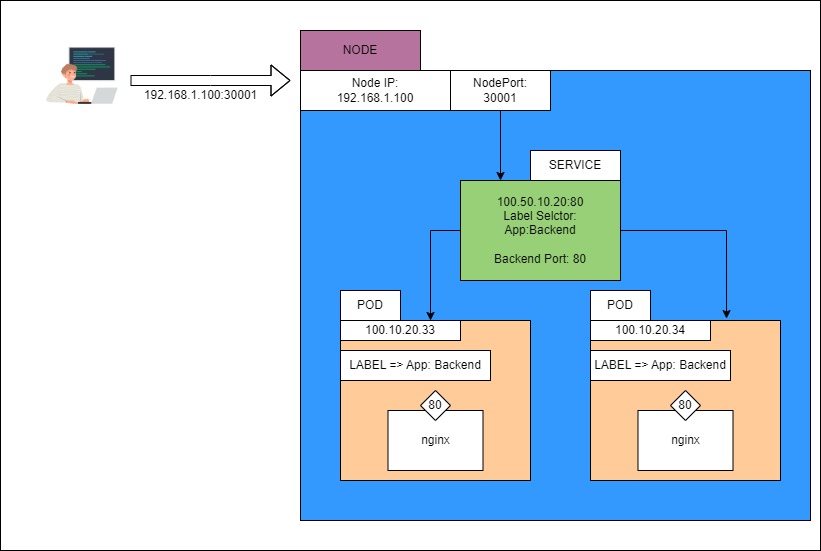
**LoadBalancer**

A LoadBalancer service is the standard way to expose a service to the internet. On GKE, this will spin up a [Network Load Balancer](https://cloud.google.com/compute/docs/load-balancing/network/) that will give you a single IP address that will forward all traffic to your service.





**NodePort**

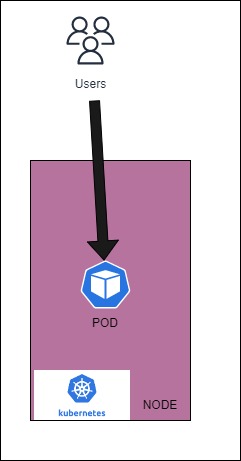
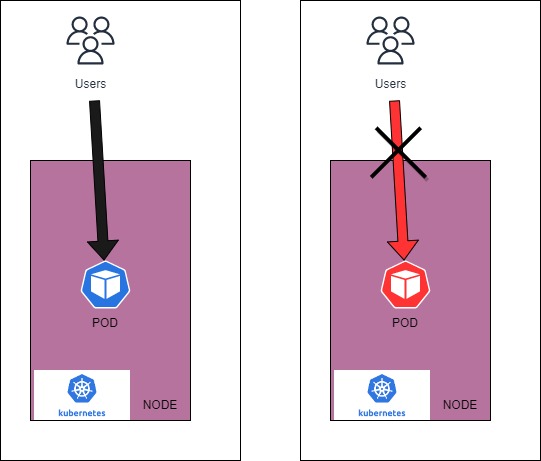
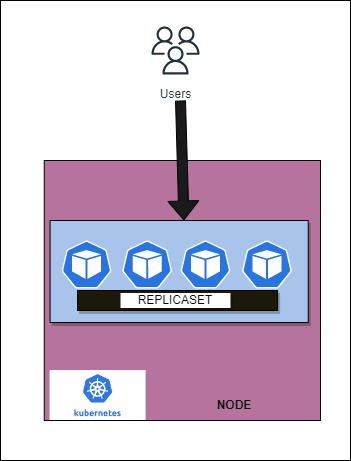


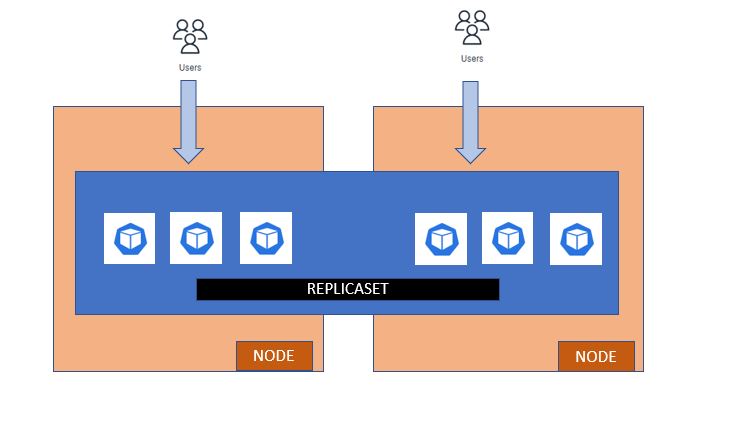
**ReplicaSet**

Maintain Pod Replica

A ReplicaSet'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

<https://kubernetes.io/docs/concepts/workloads/controllers/replicaset/>



$ vi replicaset.yaml

Copy & Paste example replicaset yaml file from official website

<https://kubernetes.io/docs/concepts/workloads/controllers/replicaset/>

$ kubectl scale –replicas=1 rs/frontend

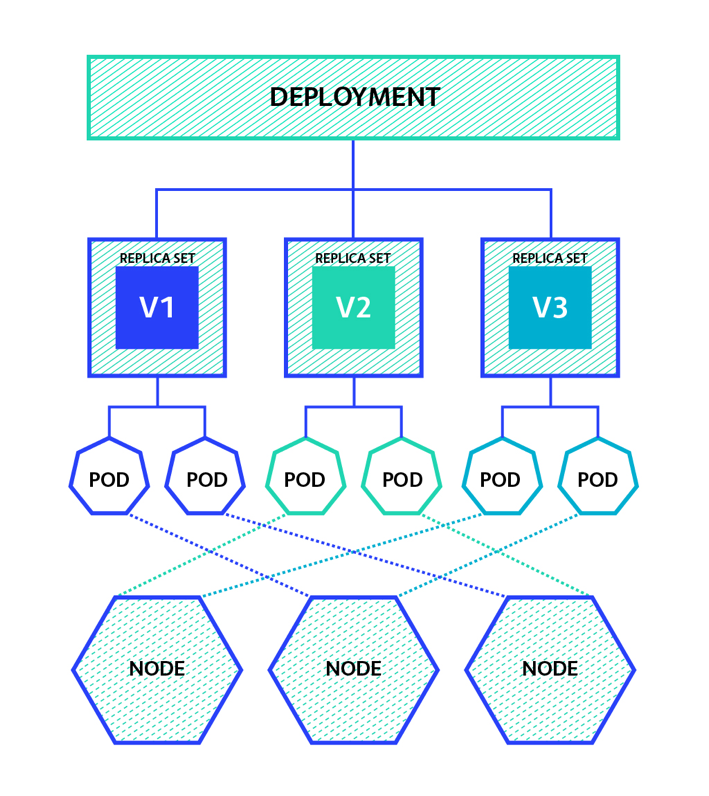
$ kubectl get pods

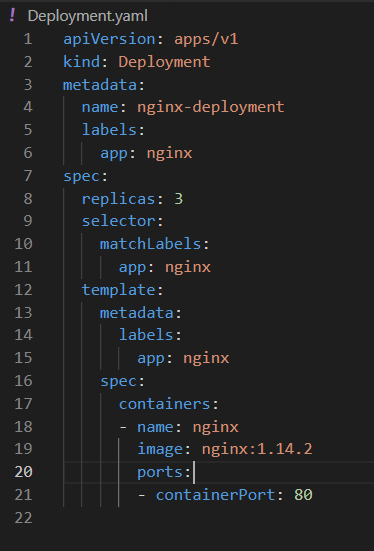
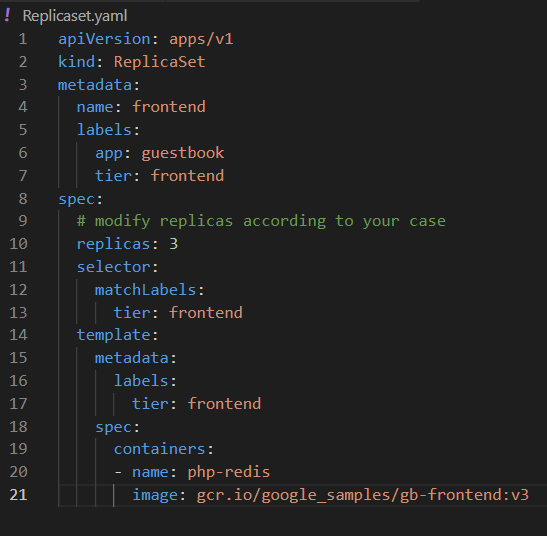
$ kubectl edit rs frontend

**Deployment**

Upgrade, Rollback, Changes Gracefully

* A Deployment controller provides declarative updates for Pods and ReplicaSets
* Define desired state in a Deployment, and the Deployment controller changes the actual state to desired state at a controlled rate.
* Deployment creates ReplicaSet to manage number of PODS



Deployment vs ReplicaSet

<https://kubernetes.io/docs/concepts/workloads/controllers/deployment/>

**For Upgrading the image version**

$ kubectl set image deployment.v1.apps/nginx-deployment nginx=nginx:1.16.1

$ kubectl rollout status deployment/nginx-deployment

$ kubectl get rs

**Going back to the previous version or revisions**

$ kubectl rollout undo deployment/nginx-deployment

**List out the versions or revisions**

$ kubectl rollout history deployment/nginx-deployment

**Scaling the Deployment**

$ kubectl scale deployment/nginx-deployment --replicas=10

**COMMAND & ARGUMENTS**

Pass commands & arguments to container running inside a Pod.

<https://kubernetes.io/docs/tasks/inject-data-application/define-command-argument-container/>

Refer commands.yaml file from the repo to understand how to use commands in Pod definition file.

$ kubectl apply -f commands.yaml

$ kubectl get pods

$ kubectl logs command-demo

**VOLUMES**

<https://kubernetes.io/docs/concepts/storage/volumes/>

Refer volume.yaml Pod definition file to understand the hostPath type volume in Kubernetes.

**Config Map**

*Environment Variables*

Refer env\_vars.yaml file to understand how environment variables works in Kubernetes

*ConfigMaps*

Refer configMap.yaml file

$ kubectl apply -f configMap.yaml

To inject the config map into a Pod check example as “Pod\_with\_configmap.yaml” file

$ kubectl apply -f Pod\_with\_configmap.yaml

<https://kubernetes.io/docs/concepts/configuration/configmap/>

**SECRETS**

<https://kubernetes.io/docs/concepts/configuration/secret/>

Secrets will encode the value using base64 method

Example:

Encode: $ echo -n “secretpassword” | base64

Decode: $ echo ‘r34dckop==’ | base64 –decode

You can refer secrets.yaml definition file. But before applying it you have to encode the data as below

$ echo -n “mypassword” | base64

It will generate the encoded data as - **bXlwYXNzd29yZA==**

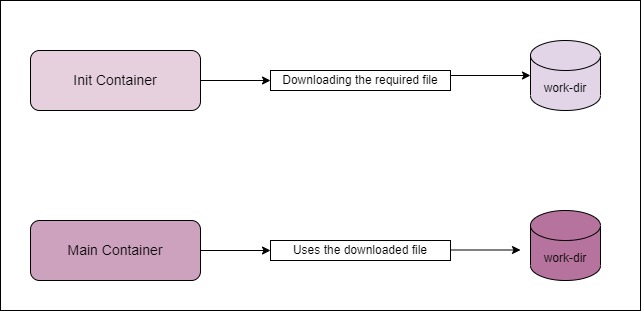
vi secrets.yaml

Image Pull Secret

<https://kubernetes.io/docs/tasks/configure-pod-container/pull-image-private-registry/>

**Init Containers & Side Car Containers**

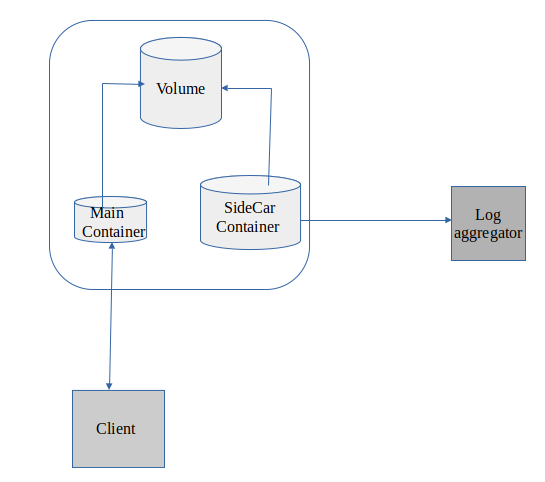
Init containers run before applications containers run in a pod, and sidecar containers run alongside application containers in a pod



<https://kubernetes.io/docs/tasks/configure-pod-container/configure-pod-initialization/#create-a-pod-that-has-an-init-container>

Side Car Container

<https://www.containiq.com/post/kubernetes-sidecar-container>



**Kubernetes Kubectl & cheetsheet**

<https://kubernetes.io/docs/reference/kubectl/cheatsheet/>

kubectl config view

kubectl create deployment nginx --image=nginx –dry-run=client -o yaml > nginx.yaml

**Taints and Tolerations**

https://kubernetes.io/docs/concepts/scheduling-eviction/taint-and-toleration/

**Limits**

<https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/>

**CronJob**

<https://kubernetes.io/docs/concepts/workloads/controllers/cron-jobs/>

**Lens**