## **Container Orchestration**

Container orchestration is all about managing the life cycles of containers, especially in large, dynamic environments.

Container orchestration can be used to perform lot of tasks, some of them includes:

* Provisioning and deployment of containers
* Scaling up or removing containers to spread application load evenly
* Movement of containers from one host to another if there is a shortage of resources.
* Load balancing of service discovery between containers
* Health monitoring of containers and hosts

## **Monolithic Architecture**

Monolith means composed all in one piece. The **Monolithic** application describes a single-tiered **software** application in which different components combined into a single program from a single platform.

Packaged and deployed as single unit.

Components

* Presentation Layer

Front End [User Interface] Responsible for handling http request. Build with Web technologies like HTML, JAVASCRIPT, CSS etc.., It communicates other layer usng API calls.

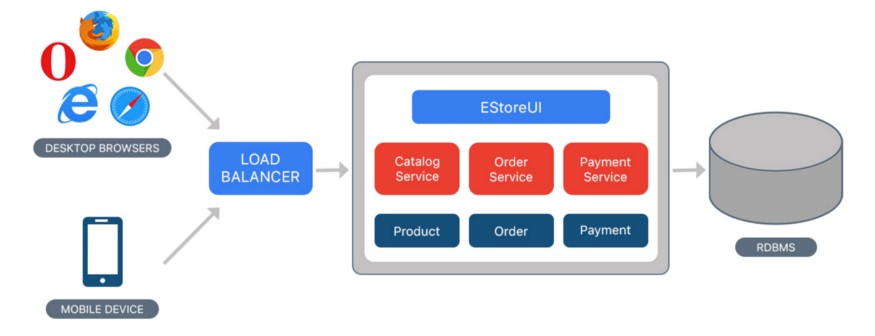
* Application Layer

Business logic often written in Java, .Net, C-sharp, Python

* Data Layer

Where all the data get stored and fetched using API calls. MYSQL, ORACLE, REDDIS, MONGODB

**Example**



* Consider an example of Ecommerce application, that authorizes customer, takes an order, check products inventory, authorize payment and ships ordered products. This application consists of several components including e-Store User interface for customers (Store web view) along with some backend services to check products inventory, authorize and charge payments and shipping orders.
* Despite having different components/modules/services, the application is built and deployed as one Application for all platforms (i.e. desktop, mobile and tablet) using RDBMS as a data source.

**Benefits**

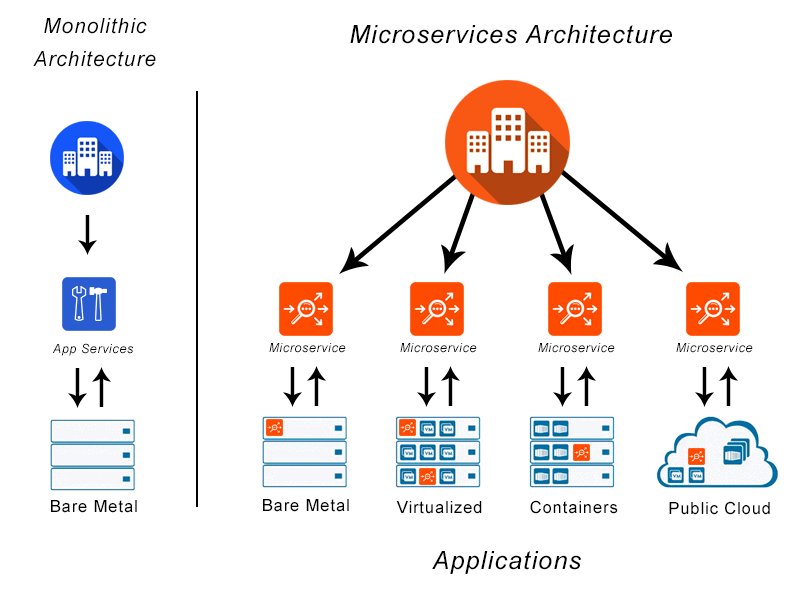
* Single unit of deployment
* Simple to develop — At the beginning of a project it is much easier to go with Monolithic Architecture.
* Simple to test. For example, you can implement end-to-end testing by simply launching the application and testing the UI with Selenium.
* Simple to deploy. You have to copy the packaged application to a server.
* Simple to scale horizontally by running multiple copies behind a load balancer.

**Drawbacks**

* A large code base can be significantly harder to understand.
* Technology dependent on initail decision
* All components impemetned using single development stack, Java or python.
* Frequent deployment are not practical
* Bug in any module (e.g. memory leak) can potentially bring down the entire process. Moreover, since all instances of the application are identical, that bug impact the availability of the entire application
* You must redeploy the entire application on each update.
* Need to scale entire Application stack

## **What is Microservices ??**

* Here in Microservices, The Application is composed of multiple services, which can be developed, deployed and maintained independently. Each of these services is responsible for discrete task and can communicate with other services through simple APIs to solve a large complex business problem.



## **When to use Microservices ?**

Ultimately, any size company can benefit from the use of a microservices architecture if they have applications that need frequent updates, experience dynamic traffic patterns, or require near real-time communication.

## **Microsservice Benefits**

* As the constituent services are small, they can be built by one or more small teams from the beginning separated by service boundaries which make it easier to scale up the development effort if need be.
* Once developed, these services can also be deployed independently of each other and hence its easy to identify hot services and scale them independent of whole application. Microservices also offer improved fault isolation whereby in the case of an error in one service the whole application doesn’t necessarily stop functioning. When the error is fixed, it can be deployed only for the respective service instead of redeploying an entire application.
* Another advantage which a microservices architecture brings to the table is making it easier to choose the technology stack (programming languages, databases, etc.) which is best suited for the required functionality (service) instead of being required to take a more standardized, one-size-fits-all approach.

Flexible Scaling Easy Deployment Technology Freedom Resuable code

Resilience

## **Disadvanages**

* First, communication between services can be complex. An application can include dozens or even hundreds of different services, and they all need to communicate securely.
* Second, debugging becomes more challenging with microservices. With an application consisting of multiple microservices and with each microservice having its own set of logs, tracing the source of the problem can be difficult.
* Up-front costs may be higher with microservices.
* For microservices architecture to work for your organization, you need sufficient hosting infrastructure with security and maintenance support, and you need skilled development teams who understand and manage all the services.

## **How are Microservies are deployed ?**

Deployment of microservices requires the following:

• Ability to scale simultaneously among many applications, even when each service has different amounts of traffic

• Quickly building microservices which are independently deployable from others

• Failure in one microservice must not affect any of the other services

Docker is a standard way to deploy microservices using the following steps:

• Package the microservice as a container image

• Deploy each service instance as a container

• Scaling is done based on changing the number of container instances

* Using Kubernetes with an orchestration system like Docker in deployment allows for management of a cluster of containers as a single system. It also lets enterprises run containers across multiple hosts while providing service discovery and replication control. Large scale deployments often rely on Kubernetes.

## **What is Kubernetes ??**

* Kubernetes is a portable, extensible, open-source platform for managing containerized workloads and services, that facilitates both declarative configuration and automation.
* It’s a container orchestration tool

## **Why you need Kubernetes and what it can do**

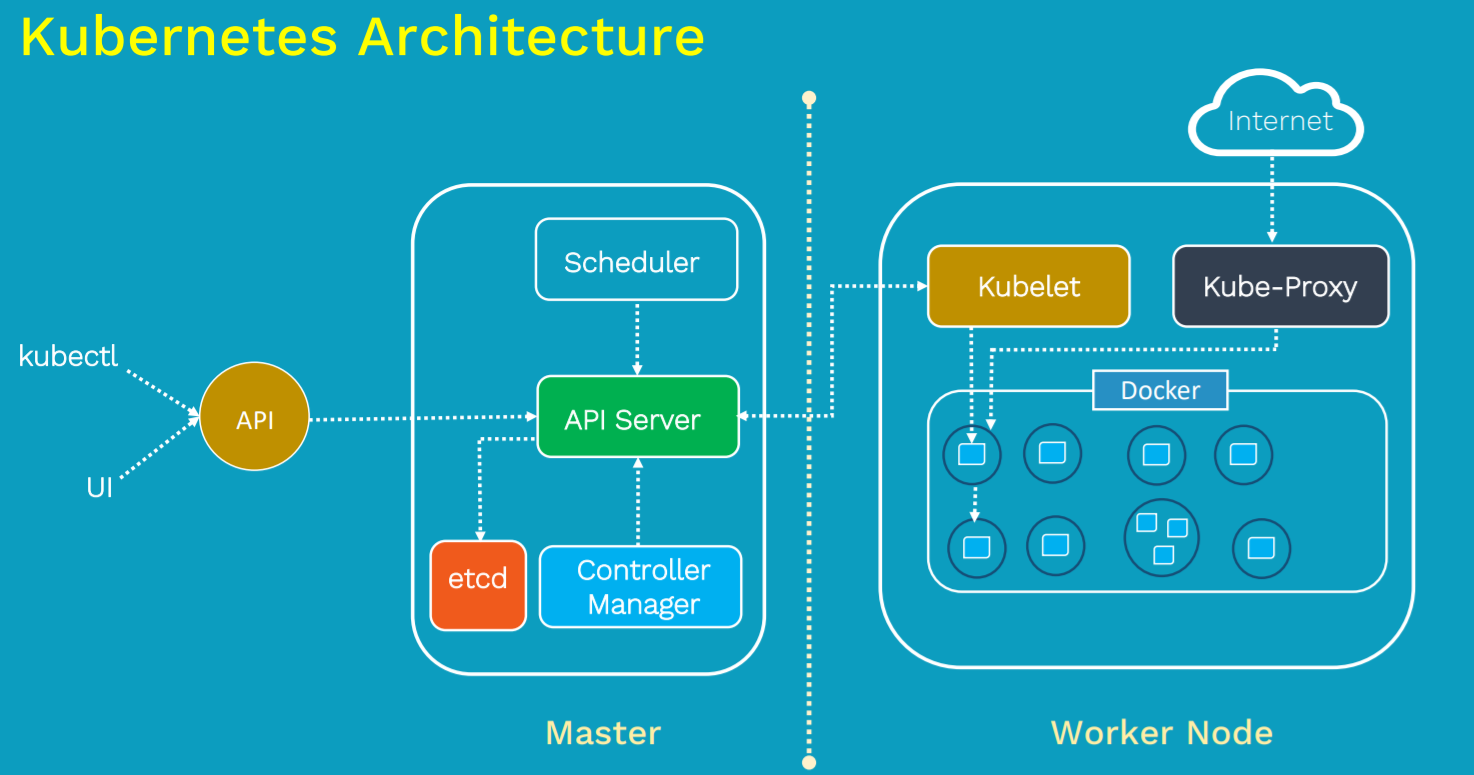
Containers are a good way to bundle and run your applications. In a production environment, you need to manage the containers that run the applications and ensure that there is no downtime. For example, if a container goes down, another container needs to start. Wouldn't it be easier if this behavior was handled by a system?

That's how Kubernetes comes to the rescue! Kubernetes provides you with a framework to run distributed systems resiliently. It takes care of scaling and failover for your application, provides deployment patterns, and more. For example, Kubernetes can easily manage a canary deployment for your system.

**Kubernetes provides you with:**

* Service discovery and load balancing Kubernetes can expose a container using the DNS name or using their own IP address. If traffic to a container is high, Kubernetes is able to load balance and distribute the network traffic so that the deployment is stable.
* Storage orchestration Kubernetes allows you to automatically mount a storage system of your choice, such as local storages, public cloud providers, and more.
* Automated rollouts and rollbacks You can describe the desired state for your deployed containers using Kubernetes, and it can change the actual state to the desired state at a controlled rate. For example, you can automate Kubernetes to create new containers for your deployment, remove existing containers and adopt all their resources to the new container.
* Automatic bin packing You provide Kubernetes with a cluster of nodes that it can use to run containerized tasks. You tell Kubernetes how much CPU and memory (RAM) each container needs. Kubernetes can fit containers onto your nodes to make the best use of your resources.
* Self-healing Kubernetes restarts containers that fail, replaces containers, kills containers that don't respond to your user-defined health check, and doesn't advertise them to clients until they are ready to serve.
* Secret and configuration management Kubernetes lets you store and manage sensitive information, such as passwords, OAuth tokens, and SSH keys. You can deploy and update secrets and application configuration without rebuilding your container images, and without exposing secrets in your stack configuration.

## **Kubernetes Architecture**

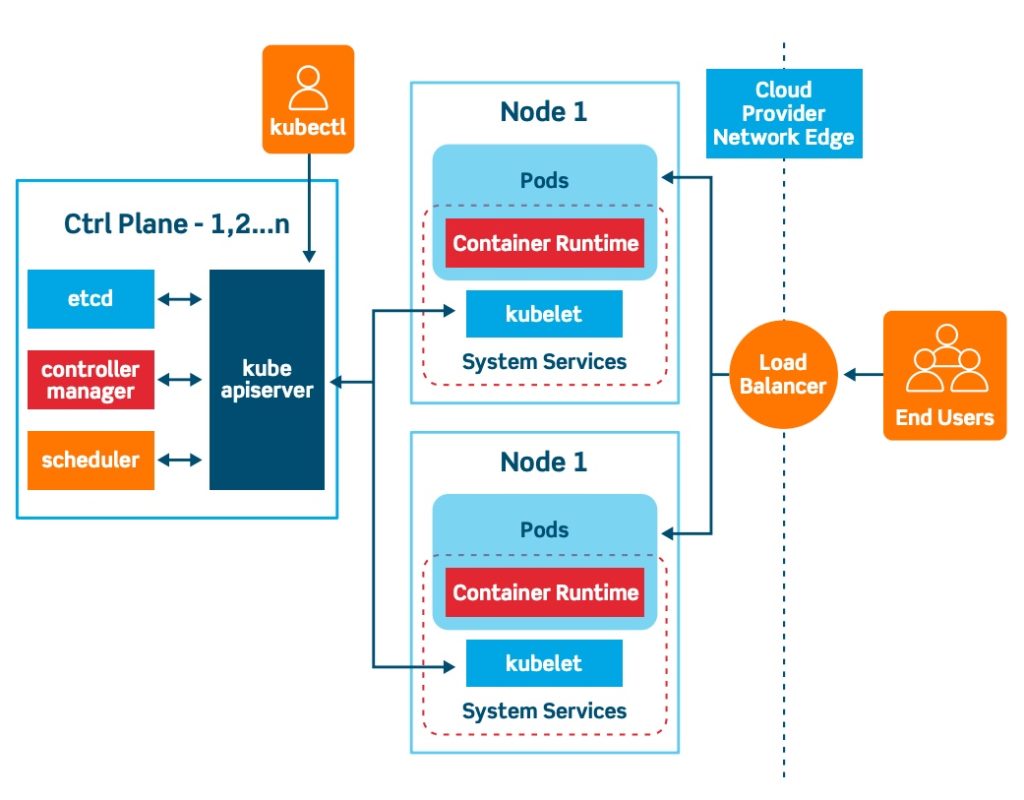


Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated



|  |  |
| --- | --- |
| **Control Panel | Master** | **Worker Node** |
| Kube-Api Server | Kubelet |
| Scheduler | Kube-proxy |
| Controller Manager | Pod |
| ETCD | Containers |

## **Kubernetes Installation**

**Installation Methods**

* Use Managed Kubernetes Service AWS | GCP | IBM
* Use MiniKube
* Install & Configure Kubernetes Manually [Hard Way]

**Play with K8s**

<https://labs.play-with-k8s.com/>



It’s a Kubernetes Playground. It provides configured environment.

## **Overview of Kubectl**

* The Kubernetes command-line tool, kubectl, allows you to run commands against Kubernetes clusters. You can use kubectl to deploy applications, inspect and manage cluster resources, and view logs.
* For configuration, kubectl looks for a file named config in the $HOME/.kube directory. You can specify other kubeconfig files by setting the KUBECONFIG environment variable or by setting the --kubeconfig flag.
* To connect to the Kubernetes Master there are two important data which kubectl needs:
* Kubectl refer to kubeconfig file

**Kubeconfig file**

* + DNS/IP of the cluster
  + Authentication Credentials

**KubeCtl Windows Installation**

Create New folder & Download kubectl binaries.  
  
<https://dl.k8s.io/release/v1.22.0/bin/windows/amd64/kubectl.exe>

Also Place the config file to the same folder.

Open CMD && go to the folder

>kubectl --kubeconfig "k8s-1-21-3-do-0-blr1-1631600530750-kubeconfig.yaml" get nodes

**Make Kubectl global env**

* For Simplicity we can make **kubectl** binaries as global environment.
* Create Folder called Binaries in C drive and place kubectl.exe file to it.
* Now go to user variable and add this path to PATH variable
* Now you can able to execute kubectl command from anywhere.

**Place config file in C:\Users\Venkatesh\.kube**

* Copy k8s-1-21-3-do-0-blr1-1631600530750-kubeconfig.yaml to C:\Users\Venkatesh\.kube
* Rename it as config and remove the .yaml extension

Graphical user interface, text, application

Description automatically generated

Now you test it using CMD or bash

**$ kubectl get nodes**

**Kubectl Linux Installation**

* $ mkdir K8s && cd K8s
* $ curl -LO [https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl](https://dl.k8s.io/release/$(curl%20-L%20-s%20https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl)
* chmod +x kubectl
* cp kubectl /usr/local/bin
* mkdir -p /home/venkat/.kube && cd /home/venkat/.kube
* vim config && paste the content
* Now you can test the kubectl command.
* $ kubectl get nodes
* $ kubectl get nodes -o wide

## **MiniKube**

Tool that makes it easy to run Kubernetes locally.

In case to setup all components of Kubernetes its time consuming, You don’t wnt to burn system resources by creating multiple VMs

Minikube comes in to rescue. It bundles all K8s components in single-node inside a VM.

Virtual Box >>> Minikibe >>> Kubecctl

Pre-Requesite

* 2 CPUs or more
* 2GB of free memory
* 20GB of free disk space
* Internet connection
* Container or virtual machine manager, such as: Docker, Hyperkit, Hyper-V, KVM, Parallels, Podman, VirtualBox, or VMWare.

Installation in Linux

$ nproc #Check no. of CPU info

$ setenforce 0

$ egrep --color ‘vmx|svm’ /proc/cpuinfo #check if Virtual machine is installed  
or Install Docker

$ sudo yum install docker

**MiniKube Installation**

* $ curl -LO <https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64>  
    
  $ chmod +x minikube
* $ sudo install minikube-linux-amd64 /usr/local/bin/minikube
* $ minikube start –vm-drive=none

## **ETCD**

It’s a distributed reliable key-value store that is simple secure & fast.  
  
**ETC Distributed**  
Traditional Database  
  
Table

Description automatically generated

**What is Key Value Store ??**

You put a key && value , it saves in the database. You get the key which returns the value.  
You cannot have duplicate keys.  
It is not replacement to tradional database, it is used to store and retrieve the small data such as configuration data which requires fast read and write data.

Graphical user interface, table

Description automatically generated

**Install ETCD**

Text

Description automatically generated

**ETCD Operation**

When you RUN etcd , its starts a service listens on port 2379 by default, then you can attach anu client to ETCD service to store and retrieve the information.

Default client ofr ETCD is etcdctl

Graphical user interface, text

Description automatically generated

**ETCD in Kubernetes**

ETCD datastore stores information regarding the clusters such as

Nodes | PODS | Configs | Secrets | Accounts | Roles | Bindings | Others

Every Information you see when you run the kubectl get command is from the ETCD server. Every change you make to your cluster such as adding additional nodes, deploying pods or replica sets are updated in the ETCD server.

Depending on how you setup your cluster, ETCD is deployed differently.

## **Pods**

**What is POD ??**

* 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. A Pod's contents are always co-located and co-scheduled, and run in a shared context. A Pod models an application-specific "logical host": it contains one or more application containers which are relatively tightly coupled.
* It’s a Run time Environment using which we deploy our application.
* Here in POD containers are encapsulated inside the POD.
* A POD in Kubernetes represents a group of one or more application containers, and some shared resources [Volume] for those containers.

A POD is always runs on a node.

A node is a worker machine in Kubernetes

Each node is managed by the Master

A node can have Mutiple PODS

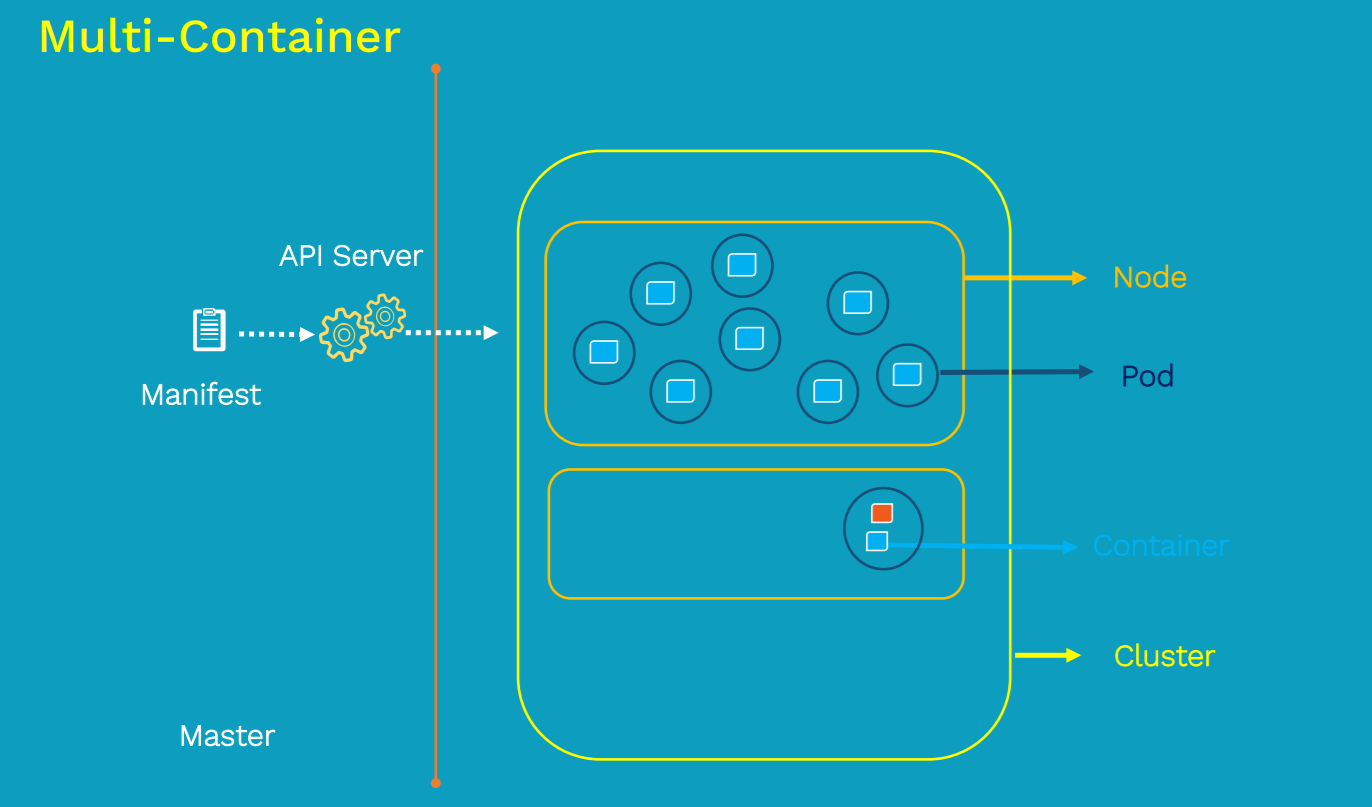
Graphical user interface, application

Description automatically generated

**POD Deployment**  
  
Diagram

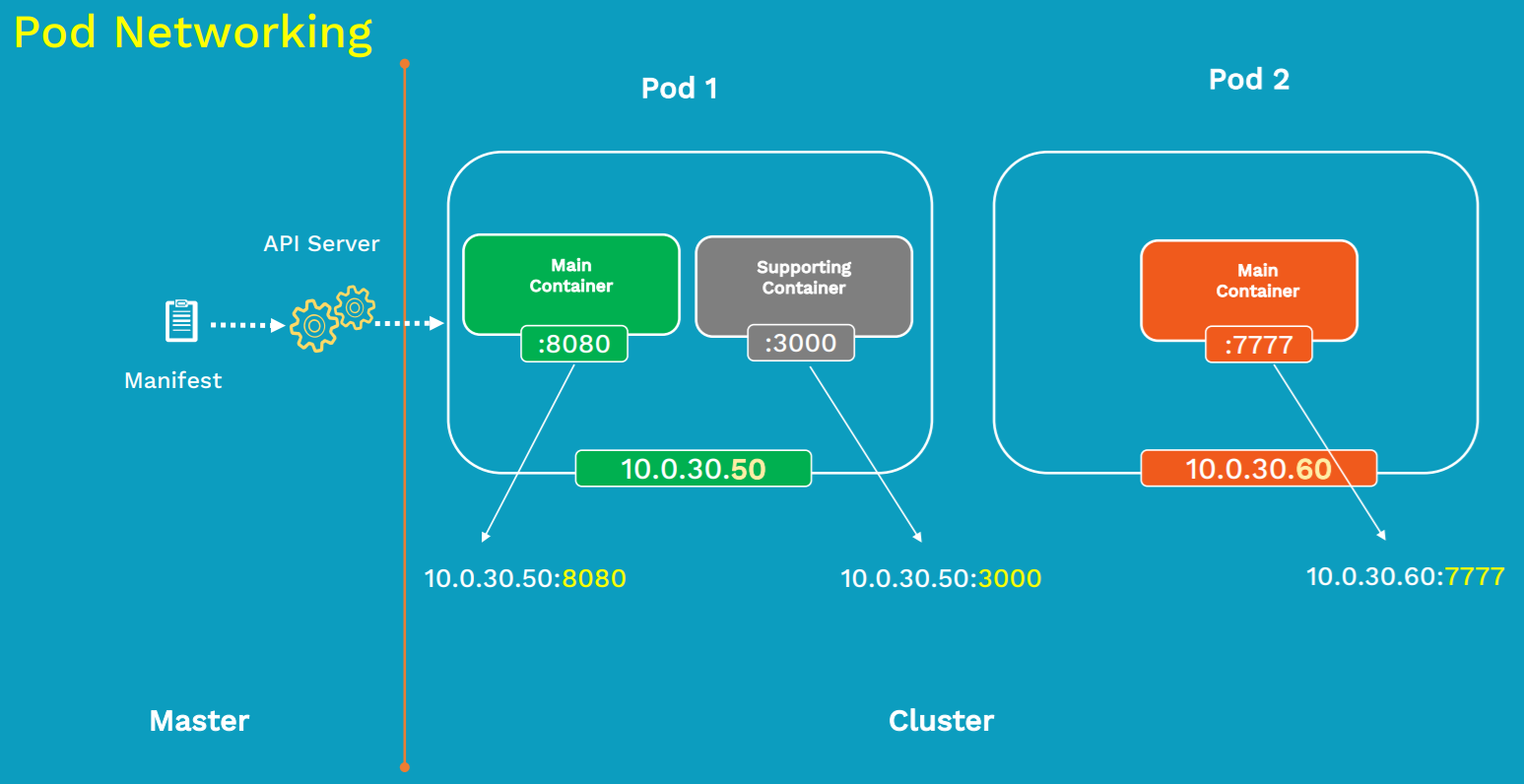
Description automatically generated

* To Deploy the POD generally write POD manifest file which consist of containers image that we are able to deploy and submit to the API server on the master node.
* API server and Scheduler components of the master node decides and deploy these pods into appropriate worker nodes.
* Generally out aim is to deploy our app in the form of containers that are distributed across the set of worker nodes and set s the Kubernetes cluster.
* However Kubernetes does not deploy containers directly instead containers are encapsulated into PODS.
* To scale the app , we create new POD with same app instance. If recourse NODE1 is not sufficient then we can spin up new VM and deploy our POD.
* We do not involve containers directlty.

**Multi Container POD**  
  


Sometimes we come across a scenario where you have a helper container that might be doing some kind of supporting task for me in the back, such as processing a user into data,or processing file uploaded bya user or etc. And you want thic container to live along side of s container.In that case we can have both these container into same POD, so that when new container is created helper container also gets created and it does when app container dies.

**POD Networking**



Every Node inside the Kubernetes cluster has its unique IP address, which is called node IP address. But in Kubernetes there is additionally one more IP address which is called POD IP address.

So once we deploy the pod onto the worker node inside the cluster, it will get its own IP address. Every POD has its unique IP address.

Containers withing same POD shares the same IP address, but it also shares the access to the Volumes, CMS group limits but different port.

**Inter POD communication**

A picture containing diagram

Description automatically generated

How PODs gets communicated with each other ?? >>> Define POD networking

All the POD IP address are fully routable on the POD Network inside the cluster. No PORT mapping required.

**Intera POD communication**

Diagram

Description automatically generated

Containers inside the POD will talk each other using shared local host interface.

**POD Life Cycle**

Diagram, schematic

Description automatically generated

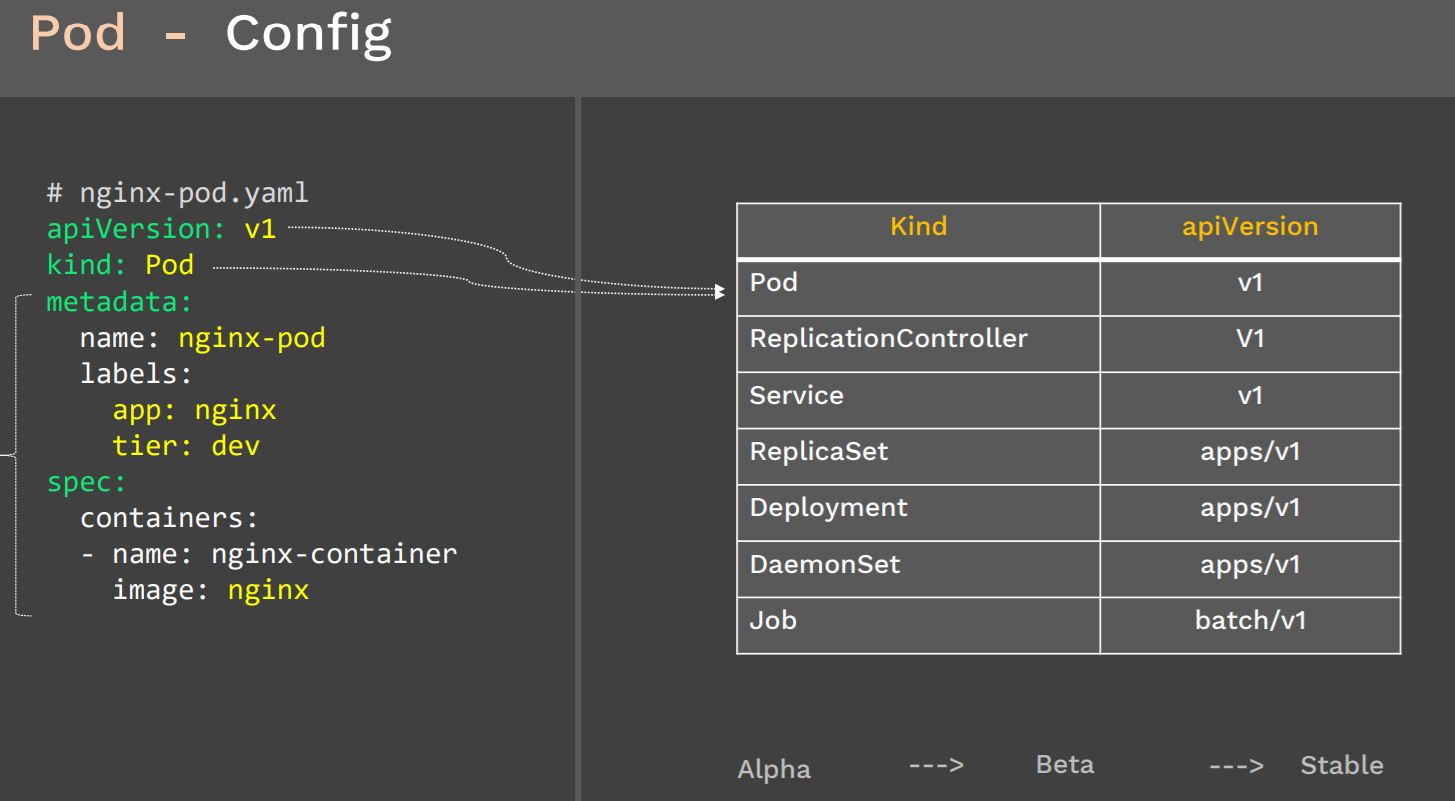
First you define POD configuration inside a Manifest file [Yaml or Json], Then you submit the Manifest file to the API server on the master node, it will be scheduled on the worker node inside the cluster using scheduler .

Once it is scheduled it goes to Pending state, during this state node will download all the Images and start the containers. It stays pending until all containers are up and running.

Once it done it goes into running state , here in running state main purpose of the POD is achived. It gets shutdown and state changes to succeeded.

If some reason containers didn’t get startet in PENDING state then POD moves to fail state. When POD dies , it dies you cant bring that back. Need to replace by creating new POD.

**POD Manifest File**



Most of the Kubernetes Objects consist of four Top Level fields, Even in POD manifest files it contains same 4 Top level fields.

apiVersion kind metadata spec

**apiVersion**

* It define the version number in which the Kubernetes objects belongs to ,
* If a API version is V1 then the Kubernetes object is part of 1st stable release of kubernetes API. It container many of the core functions like POD, ReplicationController, service.
* Apps/v1 includes the functionality related to running apps on Kubernetes like Deployment, Rolling updates, Replicaser.
* Batch/v1 containes objects related to batch processing and jobs like tasks.

**Kind**

Defines the kind of object you are creating

**metadata**

It has 2 fields Name of the objects we creating && labels is optionals.

Labels comes in handy when comes to filtering , Assume you have 1000 of PODS running and now you want to filter that related to nginx.

**spec**

Define container configuration.

**POD Creation and Display**  
  
A picture containing graphical user interface

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated with medium confidence

**Practical**

# docker run –name mywebserver nginx

# kubectl run mywebserver –image=nginx  
  
Pod Object creation using configuration File



$ kubectl apply -f pod.yaml

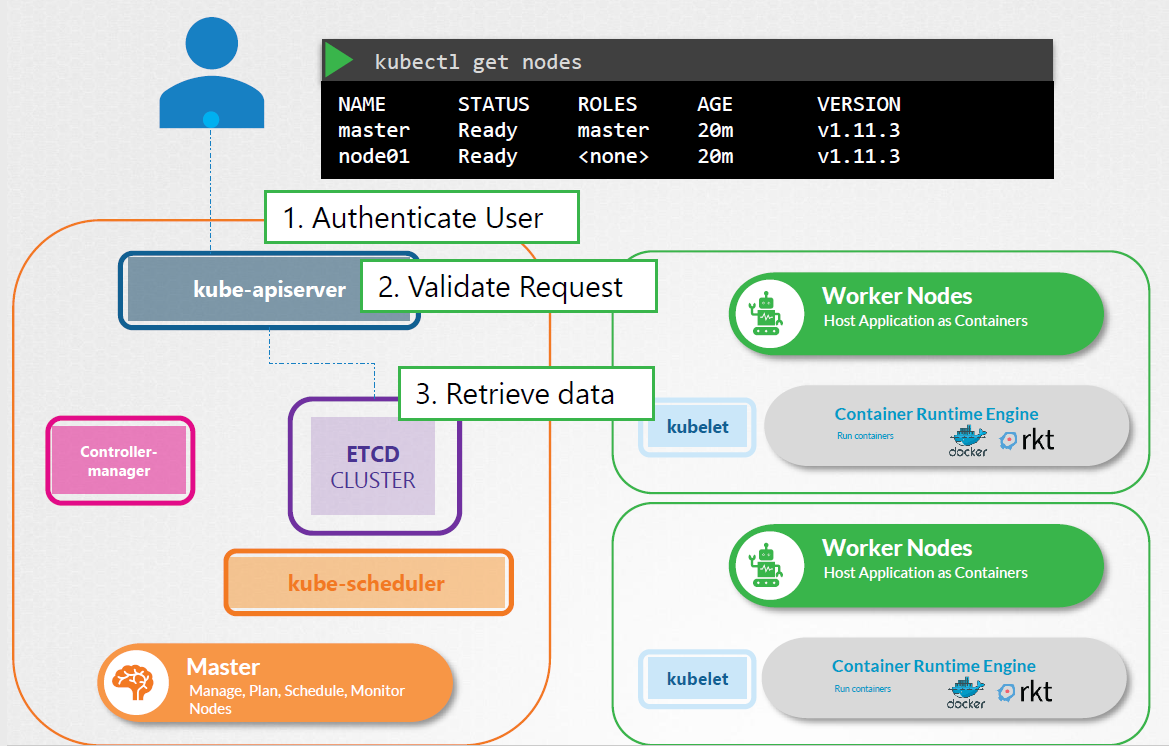
$ kibectl get pods

$ kubectl delete -f pod.yaml  
  
Creates Manifest Files  
kubectl run nginx --image=nginx --port=80 --dry-run=client -o yaml

## **Kube-API server**

Kube API is the primary management component in Kubernetes. When you run a kubectl command the kubectl utility is infact reaching to the kube-api server. The kube-api server first authenticates the request and validates it, then retrieves the data from the ETCD cluster and responds back to with the request information.

**Kube-API server act as Gateway server.**



Instead Kubectl you can also invoke the API-Server directltry using APIs by seding post request.

Graphical user interface, application

Description automatically generated

**Workflow**

When you create POD,

Request is first authenticated && validated.

API server creates POD object without assigning it to a node, updates the information in the ETCD server and updates the user that POD is created.

The scheduler continoulsy monitor the API server and realizes the new pod is created with no POD assigned then it Identifies the right node to place the new POD on and communicate to the API server.

API server then updates the nformation in the ETCD cluster . Then pass it to the kubelet in approviare worker node.

Now Kubelet created the POD on the node and instructs the container runtime engine to deploy the application image.

Once done the kubelet updates the status back to the API server and the API server then updates the data back to ETCD server.

Everytime change is requested similar pattern is followed.

KUBE-API SERVER responsible for

Authenticate User | Validate request | Retrieve data |

Update ETCD | Scheduler | Kublet updates

WorkFlow 2  
Diagram

Description automatically generated

* Kubectl writes to the API server (kubectl run mywebserver --image=nginx)
* API server will authenticate and authorize. Upon validation, it will write it to etcd.
* Upon write to etcd, API Server will invoke the scheduler.
* 4.    Scheduler decides which node the pod should run and return data to API server. API will                                                             in-turn write it back to etcd.
* API Server will invoke the kubelet in the node decided by the scheduler.
* Kubelet communicates to the docker daemon via Docker socket to create the container.
* Kubelet will update the status of the POD back to the API server.
* API Server will write the status details back to etcd.

## **KuberNetes API Primitives**

Diagram

Description automatically generated

$ kubectl proxy –port 8080  
  
 <https://127.0.0.1.8080>  
 <https://127.0.0.1.8080/api/v1>

It allows you to connect with API server in local host.

## **Kube Controller Manager**

* It manages Varius controller in Kubernetes. A controller is a department in the master node that have a own set of responsibilities. Controller is a process that continuously monitors the state of various components within the system and works towards bring the whole system to the desired functioning state.
* For example the node controller is reposnible for monitoring the status of the nodes and taking necessary actions to the application running. It does through the Kube-API server.
* **NODE Controller** checks the status of the node every 5 seconds. That way the node controller can monitor the health of the nodes if it stops receiving heartbeat from a node, then the node is marked as unreachable but it waits for 40 seconds before marking it unreachable.
* After a node is marked unreachable it gives it five min to come back up , if it doesn’t it removes the POD and provisioned them on the healthy ones.If the PODs are part of a replica set.
* **Replcation Controller** : It is responsible for monitoring the status replica sets and ensuring that the desired number of PODS are available at all times within the set. If POD dies it creates another one.
* Similar there are many controlles available, all are packaged into single process called kube controller Manager.

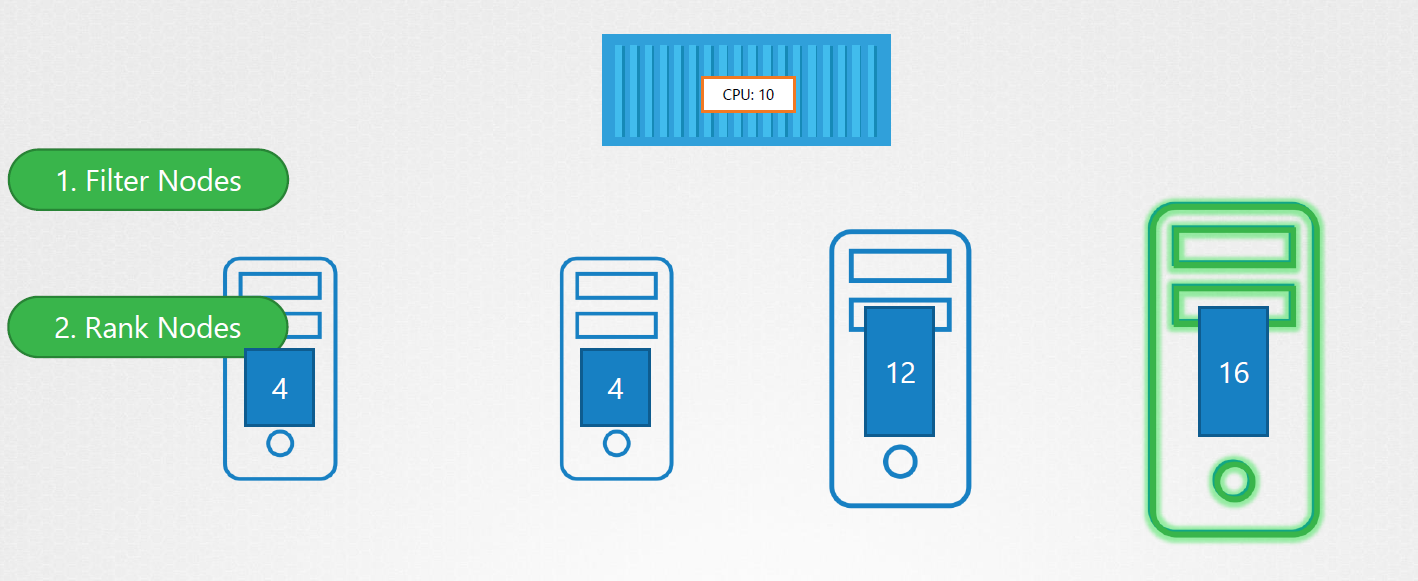


Diagram

Description automatically generated

## **Kube Scheduler**

* Scheduler is only responsible for deciding which pod goes on which node. It doesn’t actually place the nodes, that’s the job of the kubelet.
* The scheduler decides which nodes the pods are placed on depending on certain criteria. You may have PODs with dif resource requirements. You can have nodes in the cluster dedicated to certain applications.
* How it will assign ??
* The scheduler looks for each POD and tries to find best node for it. For example, below one it has set of CPU and memory requirements. The scheduler goes in two phases to identify the best PODs.
* First phase it filter ou the nodes that do not fit the profile for this POD. Here the nodes that do not have suffient CPU and memory resources requested by the pod.
* Second phase : Now schedules rank the the nodes to identiy the best fit for thr POD. It uses priority function to assign a scrore to the nodes on a scale of 0 to 10. Here in this scheduler calculates the amount of resources that free on the node after placing the pod on them.
* It can ble customized and you can create won scheduler



## **Kubelet**

The kubelet in the Kubernetes worker node, registers the node with the Kubernetes cluster.

When it receives instructions to load a container or a POD on the node, it requests the container run time engine which might be Docker , to pull the required the image and run an instance.

The kubelet then continues to monitor the state of the POD and containers in it and report to the Kube-Api server on timely basis.

You must always manually install the kubelet on your worker node.

## **Kube Proxy**

Within kubenetes cluster every POD can rech every other POD using POD networking solution. A POD network is an internal virtual network that spans across all the nodes in the cluster to which all the POD connects to . Through this network are able to communicate with each other.

There are many solutions available for deploying such a network

Example.

In this case I have a webapplciation deployed on the first node and a database application deployed on seocnad node. The web app can reach the database, simply by using the IP of the database POD. But there is no guarantee that the IP of the database part eill always remain the same. A better way for th web application to access the database is using a service. So we create a service to expose the database application across the cluster.

Now the web application can now access the database using the name of the service db.

The service also gets an IP address assigned to it, whenever a POD tries to reach the service using its IP or name it forwards the traffic to the back end pod. In this case the database.

But what is this service & how does it get an IP ? Does the service join the same POD network ? No the service cannot join the pod network because the service is not actual thing. It is not a container like pod so it doesnt have any interfaces or an actively listening process. It is a virtual component that only lives in the cabinet as memory. But then we also said that the service should be accessible across the cluster from any node . That’s where kube-proxy comes in .

Kube-proxy is a process that runs on each node in the Kubernetes cluster. Its job is to look for new services and every time a new service is created it creates the appropriate rules on each node to forward traffic to those services to the back end pods.

One way it does this is using IPTABLES rules. In this case it creates an IP table on each node in the cluster to forward traffic heading to the IP of the service which is 10.96.0.12 to the IP of the actual pod which is 10.32.0.15. So that how kube-proxy configure the service.

## **Replication Controller [Old] | Replica Set [New]**

A ReplicaSet purpose is to maintain a stable set of replica Pods running at any given time.

Desired State -  The state of pods which is desired.

Current State - The actual state of pods that are running.

Chart, diagram

Description automatically generated

**Creating Our First ReplicaSets**

Replication controller helps to run multiple Instance of Single POD in a Kubernetes cluster. This provides high availability.  
 **High Availability**  
Even if you have single POD replication controller helps by automatically bring up new POD when existing one is failed. It ensures that specified no. of POD running always at all time.

**Load Balancing && Scaling**  
Another reason we need replication controller is to create multiple pods to share the load across them.

When no. of users increase we deploy additional pod to balance the load across the two pods. If the demand further increases and if we could deploy additional parts across the other nodes in the cluster.

Here Replication controller spans across multiple nodes in the cluster, It helps us balance the load across multiple pods on different nodes as well as scale our application when the demand increases.

Replication controller is the older tech that is being replaced by replicas set up. Replica set is the new recommended way to set up.

## **Deployments**

ReplicaSets works well in basic functionality like managing pods, scaling pods and similar.

Deployments provide replication functionality with the help of ReplicaSets, along with various additional capability like rolling out of changes, rollback changes if required.

Diagram

Description automatically generated

Create Deployment

$ kubectl apply -f deployment.yaml

$ kubectl get deployments

$ kubectl get replicaset or # kubectl get rs

$ kubectl get pods

Benefits:

Rollout  
We can easily roll out new updates to our application using deployments.

Deployments will perform an update in a rollout manner to ensure that your app is not down.

Update the latest version of images in the yaml file. And deploy the changes using $ kubectl apply -f deployment.yaml.

It creates new set of replica set and once up and running it deletes the old replica sets one by one.  
  
By default, it ensures that at least 25% of the desired number of Pods are up (25% max unavailable).

Deployments keep the history of revision which had been made.  
  
We can change it by changing rollingout strategy.

Diagram

Description automatically generated

Roll Back Deployment Changes  
  
Sometimes, you may want to rollback a Deployment; for example, when the Deployment is not stable, such as crash looping.  
  
Deployment ensures that only a certain number of Pods are down while they are being updated.  
  
$ kubectl rollout history deployment.v1/apps/kplabs-deployment

It shows all revision so you can able to roll back to specific version.

$ kubectl rollout undo deployment.v1/apps/kplabs-deployment –to-revision=1

$ kubectl get rs

$ kubectl describe deployment kplabs-deployment

Diagram

Description automatically generated

Important Pointer of Deployment

* You should know on how to set a new image to deployment as part of rolling updates.
* You should know importance of –record instruction
* You should know how to rollback a deployment
* You should be able to scale the deployment

$ kubectl set image deployment <Name of Deployment> ContianerName: New Image Name:version –record

## **NameSpace**

## **Services**

Kubernetes Service can act as an abstraction which can provide a single IP address and DNS through which pods can be accessed.

Endpoints track the IP address of the objects that service can send traffic to.

This layer of abstrcction allows us to perform lot of operations like load balancing, scaling of pods and others.

Diagram

Description automatically generated

There are several types of Services which are available

* NodePort
* ClusterIP
* LoadBalancer
* ExternalName

**Create Service and EndPoints**

Create Service objects with service port details && Target port details. Internally it has Enpoinds configuration where we need to mention exact IP address of the target.

Create Endpoints with service Name && Define target IP address. [Manual]

You can use selector and pick the pod label so that IP address taken automatically.

**ClusterIP**

Whenever the service type is ClusterIP, an internal cluster IP address is assigned to the service.

Since an internal cluster IP is assigned, it can only be reachable from within the cluster.

This is a default ServiceType.

**Port vs TargetPort**

Diagram, schematic

Description automatically generated

**NodePort**

From the name, we can identify that it has to do with opening a port on the nodes.

If the service type is NodePort, then Kubernetes will allocate a port (default: 30000-32767) on every worker node.

Each node will proxy that port into your service.

Diagram

Description automatically generated

**Type: LoadBalancer**

Challenges with NodePort

We know that NodePort ServiceType will assign a node in all the worker node which can forward the traffic to the underlying service.

Challenge in NodePort:  We need to access it via IP/DNS:Port

Example:  google.com:31514

**LoadBalancer Service Type** will automatically deploy an external load balancer.

This load balancer takes care of routing requests to the underlying service.

Diagram

Description automatically generated

The overall implementation of LoadBalancer depends on your Cloud Provider.

If you plan to use it in bare-metal, then you will have to provide your own load balancer implementation.

## **Overview of Ingress**

Whenever making use of the LoadBalancer Service Type, out of the box, you can make use of a single website. But if it Mutiple application running then you might need multiple Loadbalancer. Here Ingress will help .  
Diagram

Description automatically generated

With ingress, we can set up multiple rules through which traffic can be routed to an appropriate service depending on the URL of the website that is requested.

A screenshot of a computer

Description automatically generated with low confidence

Kubernetes Ingress is a collection of routing rules which governs how external users access the services running within the Kubernetes cluster.

Ingress can provides various features which includes:

* Load Balancing
* SSL Termination
* Named-based virtual hosting

**Ingress Resources and Ingress Controllers**

There are two sub-components when we discuss about Ingress:

* Ingress Resource
* Ingress Controllers

Ingress Resource contains set of routing rules based on which traffic is routed to a service.

Ingress Controller takes care of the Layer 7 proxy to implement ingress rules.

You must have an ingress controller to satisfy an Ingress. Only creating an Ingress resource has no effect

Diagram

Description automatically generated

## **Helm**

Helm is one of the package manager for Kubernetes.

Kubernetes application can contain lot of lot of objects like:

 Deployments, Secrets, LoadBalancers, Volumes, services, ingress  and others.

Shape

Description automatically generated

Diagram

Description automatically generated

**Helm Configuration**

There are two major components to configure:

* Helm Client
* Tiller (Server Side component of Helm)

Diagram

Description automatically generated

## **Name Based Virtual Hosting**

Name-based virtual hosts support routing HTTP traffic to multiple host names at the same IP address.

The requests are routed based on the HTTP host-header.

Diagram

Description automatically generated

## **DaemonSets**

Lets say that we have 3 nodes and we want to run a single ccopy of pod in every node.  
Example : Anitvirus deployment in each node.

Daemonset help to achive this, It ensure that pod is deployed in each node, even if you create new node it ensures the pod is deployed.

## **ConfigMaps**

## **Kuberrnetes Object**

Kubernetes Objects is basically a record of intent that you pass on to the Kubernetes cluster

Once you create the object, the Kubernetes system will constantly work to ensure that object exists.

Kubernetes objects are

* Pod
* ReplicationController
* Service
* ReplicaSet
* Deployment
* DaemonSet
* Job

There are various ways in which we can configure a Kubernetes Object.

* The first approach is through the kubectl commands.
* The second approach is through a configuration file written in YAML

Text

Description automatically generated

$ kubectl apply -f pod.yaml

**Benefits of Configuraiton File**

* + Integrates well with change review process
  + Provides the source of record on what is live within the Kubernetes Cluster.
  + Restart to troubleshoot changes with versio control.

## **CMD VS ENTRYPOINT in K8s**

Table

Description automatically generated

## **EXPOSE Ports in K8s**

kubectl run nginx-port --image=nginx --port=80

## **Labels && Selectors**

Without Labels

Graphical user interface, application, Teams

Description automatically generated

With Labels

Graphical user interface, application, Teams

Description automatically generated

**Selectors**

Selectors allows us to filter objects based on Labels.

Graphical user interface, application, Teams

Description automatically generated

**Practicle Examples Using Kubectl**

$ kubectl run nginx1 –image=nginx

$ kubectl run nginx2 –image=nginx

$ kubectl get pods

$ kubectl label pods <podName> env=dev

$ kubectl label pods <podName> env=prod

$ kubectl get pods –show-labels

$ kubectl get pods -l env=dev #Display all the pods which has labelled dev

$ kubectl get pods -l env!=prod

**Practicle Examples Using Kubectl YAML file**

apiVersion: v1

kind: Pod

metadata:

name: nginxwebserver

labels:

env: dev

$ kubectl apply -f labels.yaml

## **NodeSelector**

nodeSelector allows us to add a constraint about running a pod in a specific worker node

Step 1: Add a Label to the Node  
  
$ kubectl get nodes

$ kubectl describe node NodeName

By default it has few labels, Nodes come pre-populated with the standard set of labels. These includes

* Kubernetes.io/hostname
* Kubernetes.io/os
* Kubernetes.io/arch
* Failure-domain.beta.kubernetes.io/zone
* Beta.kubernetes.io/instance-type

We can also assign custom label to the node

$ kubectl label node NodeName disk=hdd

$ kubectl label node NodeName disk=ssd

Rectangle

Description automatically generated with medium confidence

Step2 : Create a NodeSelctor configuration to run pods only on nodes which has label of disk=ssd

nodeSelector:

disk: ssd

$ kubectl apply -f nodeselector.yaml

## **Node Affinity**

For multiple reasons, there can be a need to run a pod on a specific worker node.There can be multiple reasons, node hardware being the common one.

Node affinity is a set of rules used by the scheduler to determine where a pod can be placed

In Kubernetes terms, it is referred as nodeSelector, and nodeAffinity/podAffinity fields under PodSpec.

In Kubernetes, we can achieve nodeAffinity with the help of:

* nodeSelector Will be depricated
* nodeAffinity (more flexibility)

Node affinity is conceptually similar to nodeSelector – it allows you to constrain which nodes your pod is eligible to be scheduled on, based on labels on the node.  
  
Graphical user interface, text, application

Description automatically generated

Soft Preference basically means that even if the Contraint is not fulfilled, still your scheduler can go ahead and place your pod to that specific node.

Hard Preference >> your constraint must match in order to place the pod to specific node.

## **POD Affinity**

First Question:   Where should I be running this pod?

With Node Affinity, the question became:  Should I be running my pod in this node?

The considerations are still about the node. No outside information is considered apart from node.

Graphical user interface

Description automatically generated with medium confidence

Step 1: Pod Selector

The first thing to figure out: “What other POD are we referring to”

In this case, we are referring to other POD as AppA which has a label of app:frontend

Answer: I want BackEnd Pod to be running in the same place as AppA Pod.

Step 2: Topology

Topology refers to “what does the same place mean”?

It can mean the same place if we look at the zone or region level (same AZ / same region)

It can mean a different place if we look at the host level.

Graphical user interface

Description automatically generated

Step 3: Yes/No

Should I run my pod in the same place as the other POD?  (Yes or No)

Yes:  Pod Affinity

No:   Pod Anti-Affinity

## **Resource Request & Limits**

If you schedule a large application in a node which has limited resource, then it will soon lead to OOM or others and will lead to downtime.

Diagram

Description automatically generated

Requests and Limits are two ways in which we can control the amount of resource that can be assigned to a pod (resource like CPU and Memory)

**Requests**:  Guaranteed to get. >>> Primary Consideration

**Limits**:       Makes sure that the container does not take node resources above a specific value.

Kubernetes Scheduler decides the ideal node to run the pod depending on the requests and limits.

If your POD requires 8GB of RAM, however, there are no nodes within your cluster which has 8GB RAM, then your pod will never get scheduled.

Chart, bar chart, waterfall chart

Description automatically generated

## **Static Pods**

* Scheduling Pods without Scheduler.
* You can directly inform the kubelet that it needs to run a specific pod.
* There are multiple ways in which you can tell kubelet to run a pod.
* Pod created directly without schedulers are also referred as Static Pods.

## **Taits && Tolerations**

**Taints** are used to repel the pods from a specific node.  
Whenever tait is applied to the Node, By default POD get blocked to that specific node

Diagram

Description automatically generated

In order to enter the taint worker node, you need a special pass.

This pass is called toleration.

Toleration

Diagram

Description automatically generated

To create Taint Node  
$ kubectl taint nodes nodeName key=value:noschedule

Now it block all the new pod to the node called nodeName  
  
To Delete Taint Node

$ kubectl taint nodes nodeName key=value:noschedule-

**Tolerations**

Tolerations:

* Key: “key”

Operator: “Exists”

Effect: “NoSchedule”

Components of Taints && Tolerations  
  
A taint allows a node to refuse pod to be scheduled unless that pod has matching toleration.

We can apply toleration to a pod within the PodSpec

Text

Description automatically generated with medium confidence



Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

## **Security**

**Authentication**

* Kubernetes Clusters have two categories of users:
* Normal Users.
* Service Accounts
* Kubernetes does not have objects which represent normal user accounts.
* Kubernetes does not manage the user accounts natively.

**Authenticate Strategies**

* There are multiple ways in which we can authenticate. Some of these include:
  + Usernames / Passwords.
  + Client Certificates
  + Bearer Tokens

**Asymmetric Key Encryption**