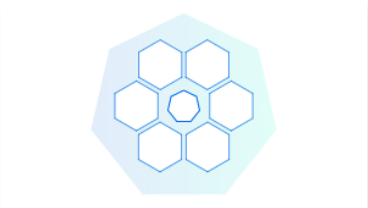
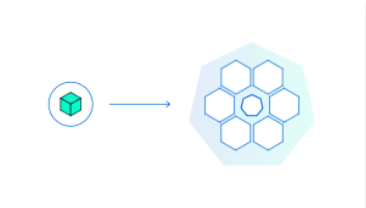
What is Kubernetes?

Kubernetes Basics Modules

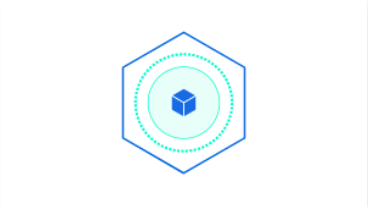


1. Create a Kubernetes cluster

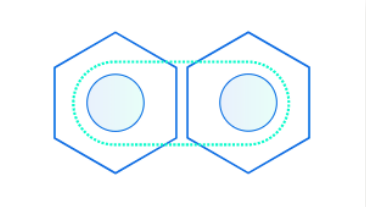


2. Deploy an app

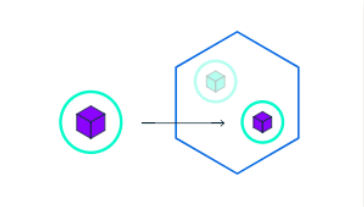
3. Explore your app



4. Expose your app publicly

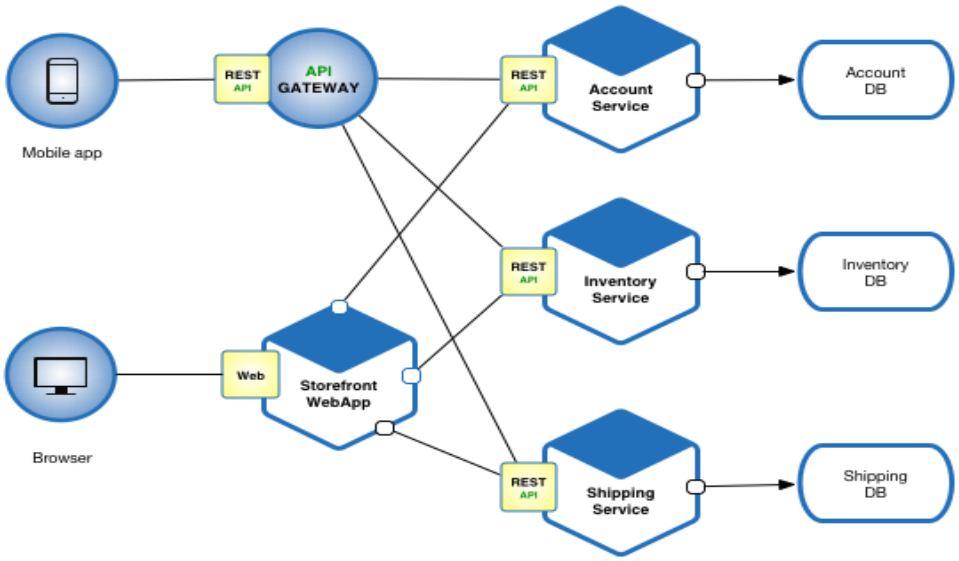


5. Scale up your app



6. Update your app

What's a micro-service application?



Kubernetes setup

Kubernetes can be installed using multiple ways.

- Minikube – for MacOS and Windows…. Works good for testing purpose that spins up and VM and create all Kubernetes components in it.
- Search the page for **minikube** latest release on page https://github.com/kubernetes/minikube/releases
- Google Container Engine.. this is integrated with google cloud and has Kubernetes available as a service. (SAAS)
- On AWS, also available as a service (SAAS) as 'Kubernetes operations (kops)'. We can install all Kube componenets using 'kops' on AWS EC2 instances.
- Manual install method, usually used while working with local datacentre or cloud. This uses the 'kubeadm'. With using commands like 'Kubeadm init' and 'kubeadm join –token <token>' we can create a kubernetes cluster and join nodes to the cluster.

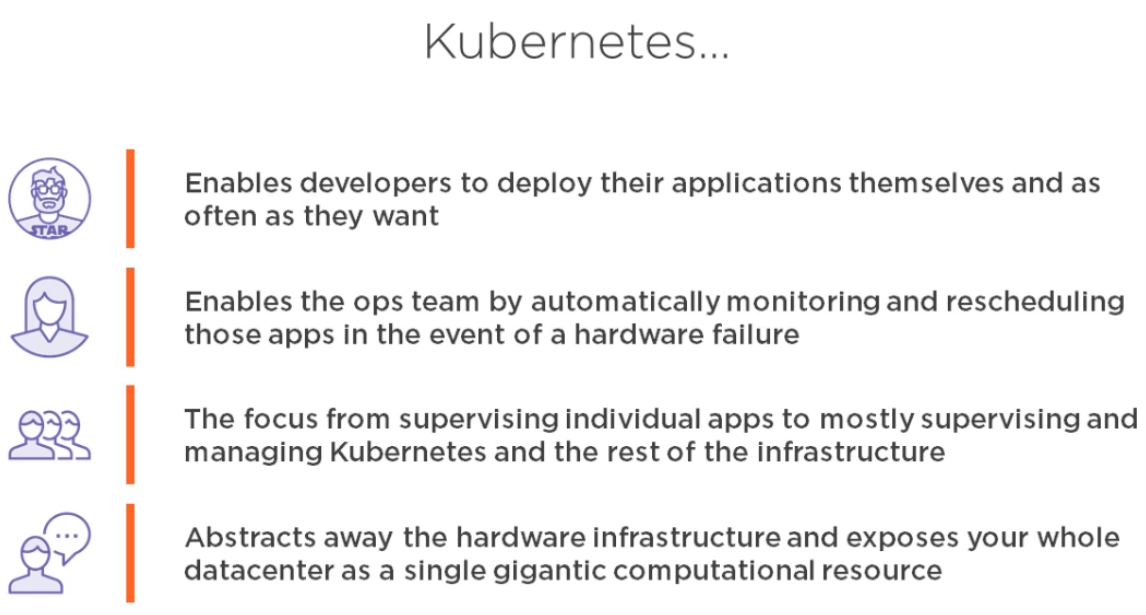
We will look at K8S using the last method of manual installs on cloud environment.

Spin up three instances on google cloud with one installed as Master and other two as nodes.

Runs only on Linux.

Each machine to be installed with below apps.

1) Docker or Rocket : Container runtime
2) Kubelet: kubernetes node agent (running on each node)
3) Kubeadm: Tool to build the cluster
4) Kubectl: Kubernetes client
5) CNI: Support for CNI networking (Container network interface)



Kubernetes Architecture:

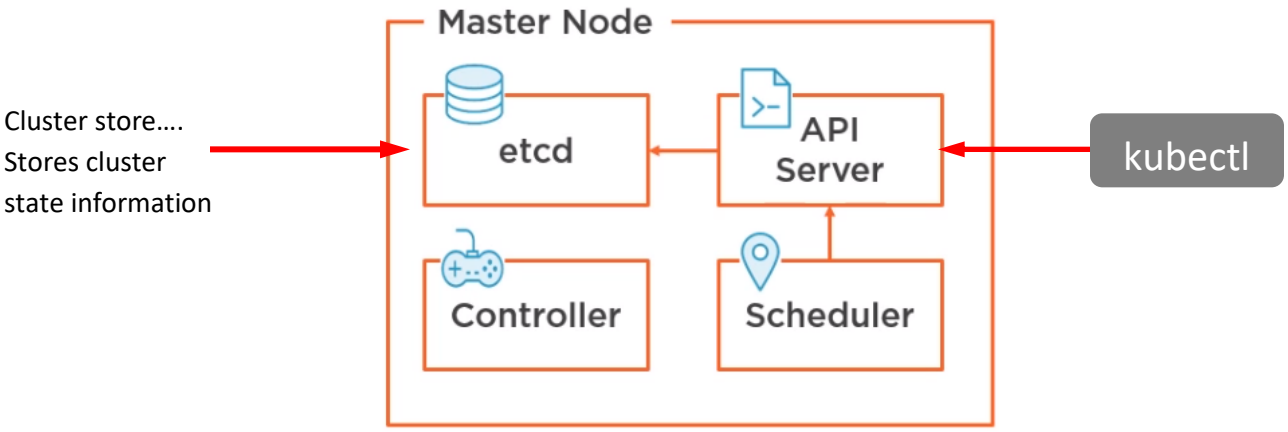
Master and Node:

K8S Master:

A single K8S Master or multi Master environment is definitely a possibility.

The Master has components like, API server, Proxy server, Container controller, cluster store etc.

Nodes are the VMs on which we will install three major components as the '**kubelet', 'Container runtime' and the 'kube-proxy'.**



Functionalities of every control plane component.

API Server	etcd	Scheduler	Controller Manager
Central	Persists State	Watches API Server	Controller Loops
Simple	API Objects	Schedules Pods	Lifecycle functions and desired state
RESTful	Key-value	Resources	Watch and update the API Server
Updates etcd		Respects contraints	ReplicaSet

Windows nodes in Kubernetes

To enable the orchestration of Windows containers in Kubernetes, include Windows nodes in your existing Linux cluster. Scheduling Windows containers in Pods on Kubernetes is similar to scheduling Linux-based containers.

In order to run Windows containers, your Kubernetes cluster must include multiple operating systems. While you can only run the control plane on Linux, you can deploy worker nodes running either Windows or Linux depending on your workload needs.

Windows nodes are supported provided that the operating system is Windows Server 2019.

RESTful API Verbs:

GET	Get the data for a specified resource(s)
POST	Create a resource
DELETE	Delete a resource
PUT	Create or update entire existing resource
PATCH	Modify the specified fields of a resource

The kubectl commands helps to interact with the API server by using above mentioned methods, like get, post etc.

Special API requests

LOG	Retrieve logs from a container in a Pod
EXEC	Exec a command in a container get the output
WATCH	Change notifications on a resource with streaming output

API Resource Location (API Paths)

Core API (Legacy)

```
http://apiserver:port/api/$VERSION/$RESOURCE_TYPE
http://apiserver:port/api/$VERSION/namespaces/$NAMESPACE/$RESOURCE_TYPE/$RESOURCE_NAME
```

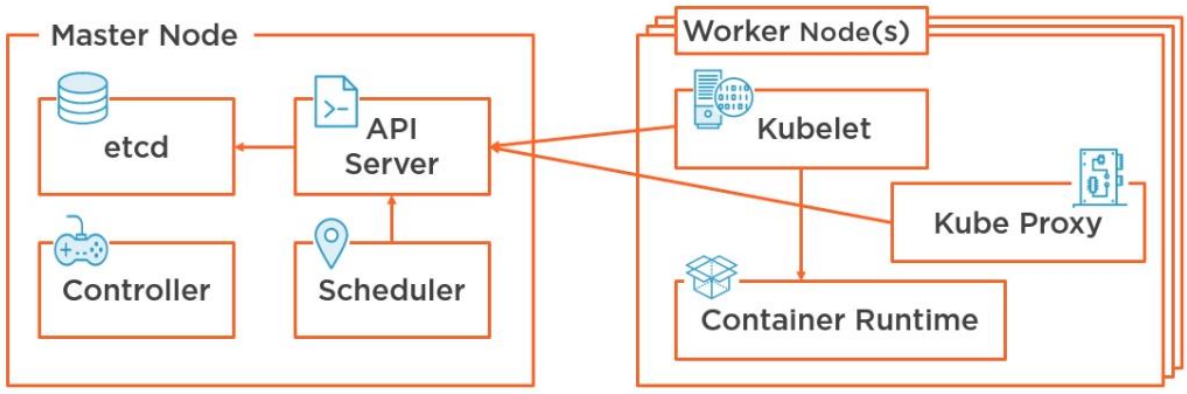
API Groups

```
http://apiserver:port/apis/$GROUPNAME/$VERSION/$RESOURCE_TYPE
http://apiserver:port/apis/$GROUPNAME/$VERSION/namespaces/$NAMESPACE/$RESOURCE_TYPE/$RESOURCE_NAME
```

Response code from API server

Success (2xx)	Client Errors (4xx)	Server Errors (5xx)
200 - OK	401 - Unauthorized	500 - Internal Server Error
201 - Created	403 - Access Denied	
202 - Accepted	404 - Not Found	

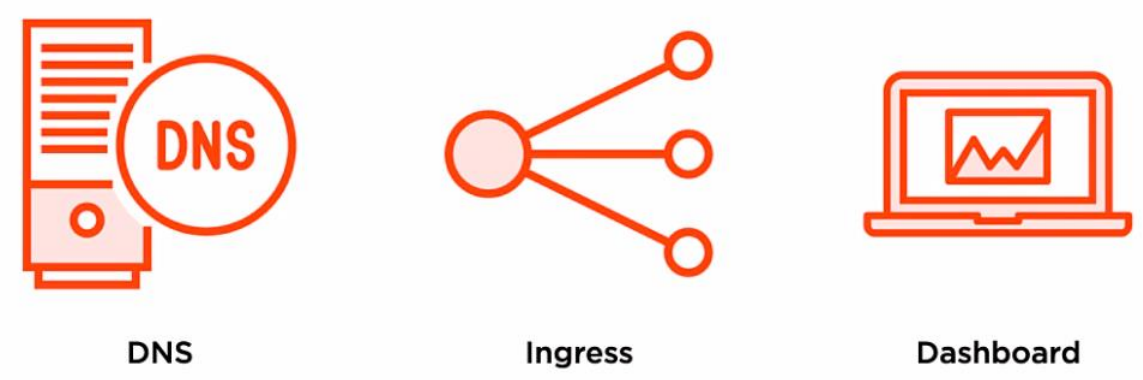
Cluster structure and components:



Components running on all nodes (including Control plane) and their responsibilities.

Kubelet	kube-proxy	Container Runtime
Monitors API Server for changes	iptables	Downloads images & runs containers
Responsible for Pod Lifecycle	Implements Services	Container Runtime Interface (CRI)
Reports Node & Pod state	Routing traffic to Pods	containerd
Pod probes	Load Balancing	Many others...

Add-on PODs



Kubelet is the main Kubernetes agent on the node, in fact we can say, the 'kubelet' is the node.

- It's the main kubernetes agent
- Registers node with cluster
- And watches the 'apiserver' on the master for work assignments.
- Instantiate PODs
- In case of any error, reports back to master.
- Exposes port 10255 to inspect the status of PODs
- Any stopped POD, kubelet on the node does not take any action to restart or recreate it, but simply reports back to master. Thus talks to the container runtime (Docker / rkt) for container management.

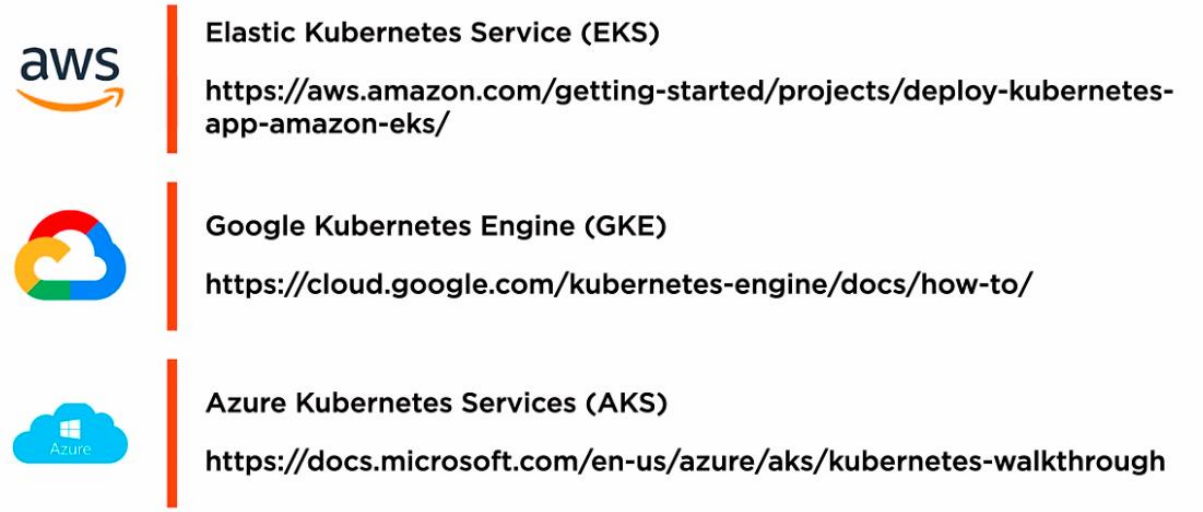
Container Runtime on the node is responsible for container management on the node,

- Pulling images
- Starting stopping container.
- Thus pluggable and uses 'Docker' or 'CoreOS rkt'.

Kube-proxy on the node is responsible for networking within PODs on the node.

- POD IP addresses
    - All container inside a POD share a single IP
- Load balancing across all PODs in a service inside a node.

Working with Kubernetes cloud service:



Declarative Model and Desired State:

In Kubernetes we do not specify how to create a cluster and containers inside a POD but we just define what we want to achieve. So that becomes our 'Desired State Configuration' of the Container and related services in it. This is stated using a YAML or JSON as the kubernetes manifest files.

So if we want to have 10 PODS running always with a particular version of a file deployed to it, this is defined in the manifest and K8S will always make sure to achieve this state.

Working with Pods

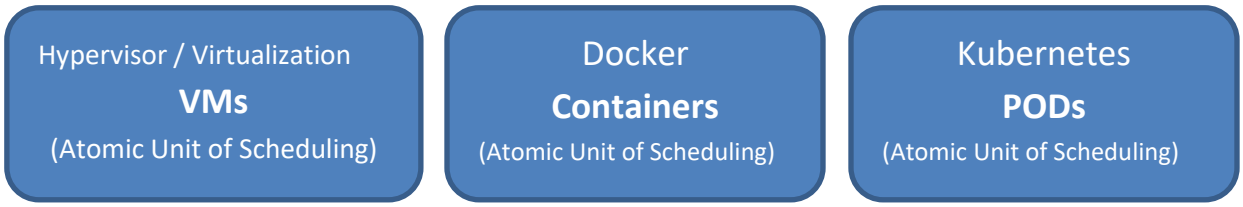
PODs theory:

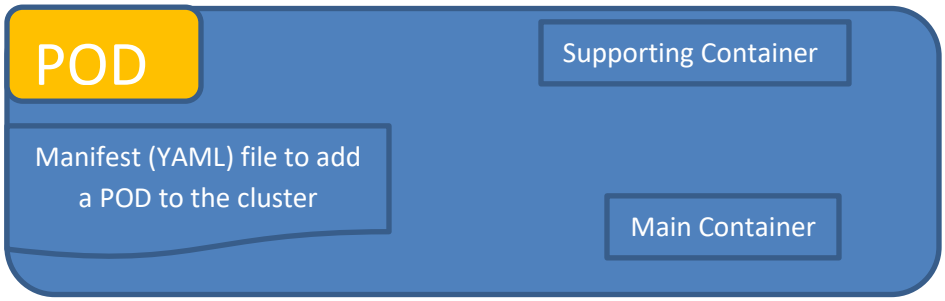
1. POD is the smallest atomic unit of scheduling in Kubernetes.
2. One POD can contain multiple containers in it with same or multiple services running in it.
3. But one POD can only be running on one node and can never be split on multiple nodes.
4. PODs are declarative via Manifest files.

A simple comparison:

POD can be termed as a simple ringed fenced environment with Network stack, Kernel namespaces, volume mounts and containers in it.

If two containers need to share same volume inside a POD, those can reside inside a single POD as well.



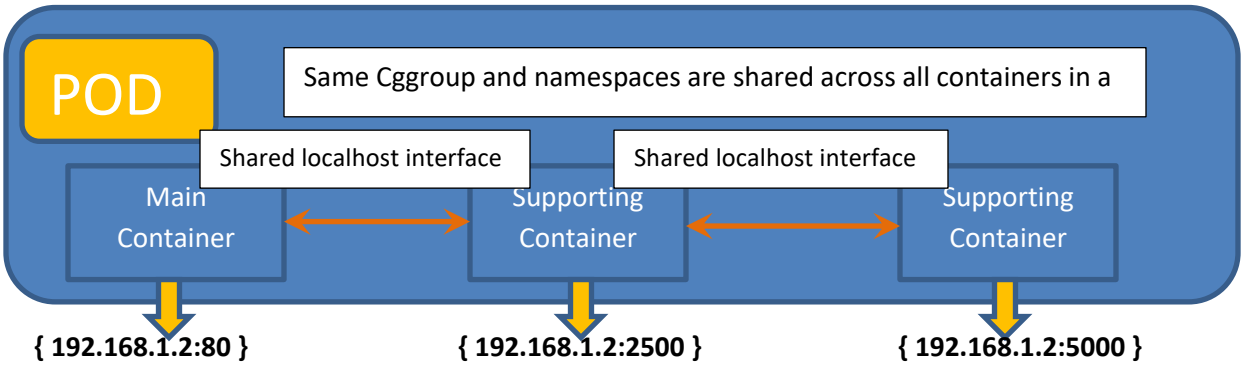



```
[root@my-node1 ~] $ mkdir /etc/kubelet.d/
[root@my-node1 ~] $ cat <<EOF >/etc/kubelet.d/static-web.yaml
apiVersion: v1
kind: Pod
metadata:
  name: static-web
  labels:
    role: myrole
spec:
  containers:
    - name: web
      image: nginx
      ports:
        - name: web
          containerPort: 80
          protocol: TCP
EOF
```

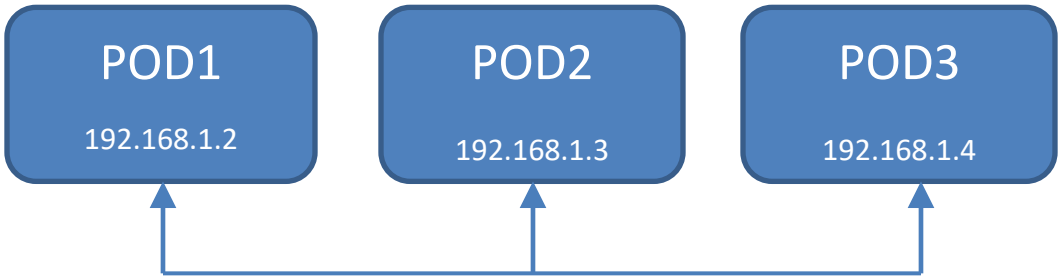
This file is then fed to the API server and the scheduler deploys this to a Kube node.

In a POD, even if there are multiple containers, each POD has only one IPAdddress. So the containers inside each POD is accessed over different port like, as shown below,

POD Networking: Intra-POD communication

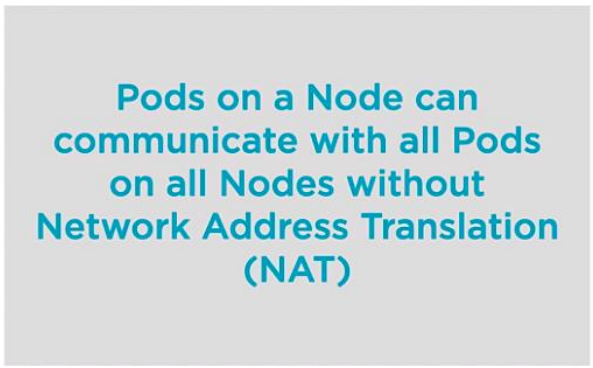


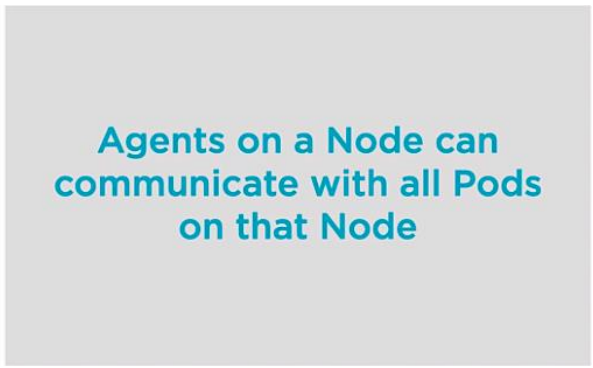
Inter POD communication: POD network



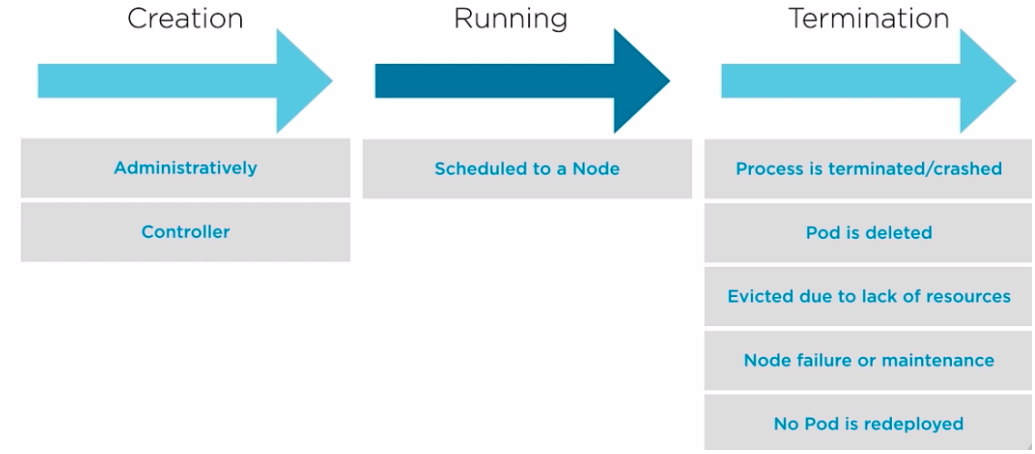
Like Container in Docker, PODs are mortal units in Kubernetes. We never get into the mode of repairing a POD.

Kubernetes Networking Fundamentals:

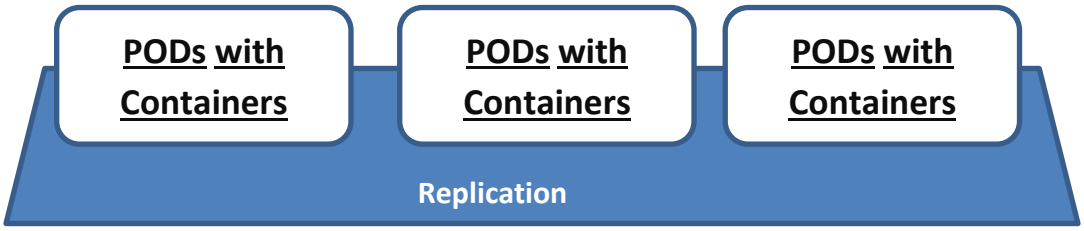




How does a POD progress in its lifecycle?



Replication Controller:

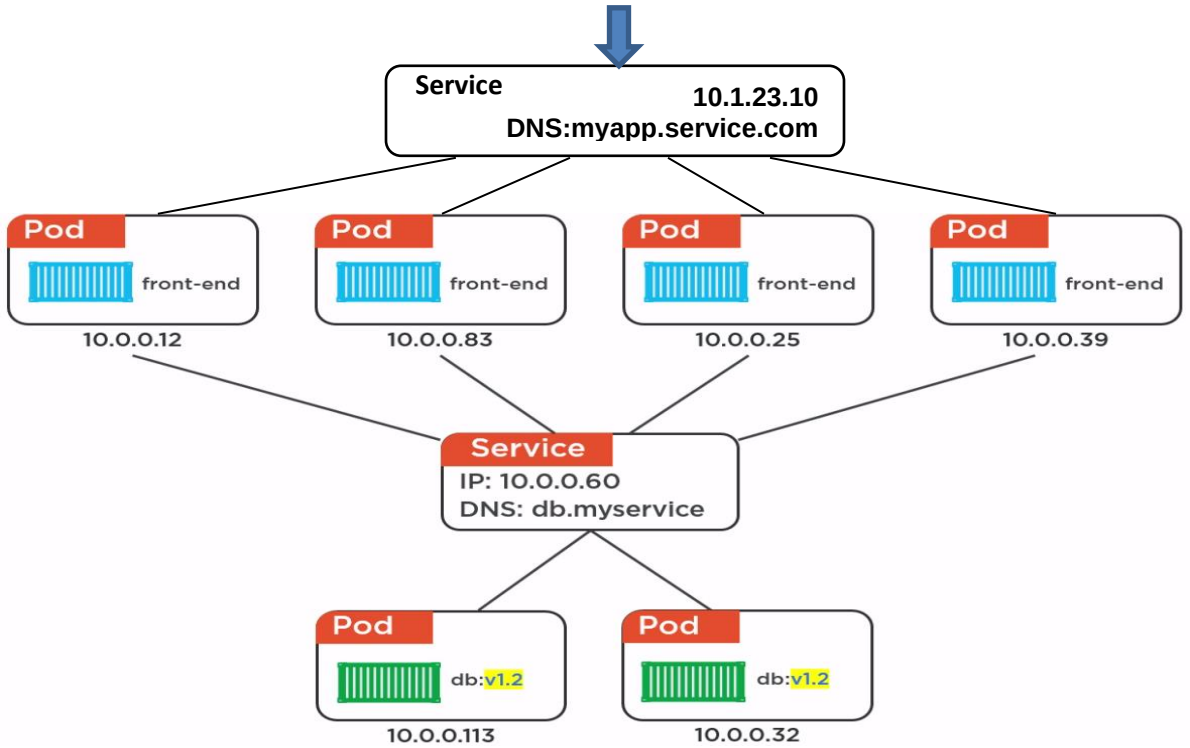


Kubernetes Services:

Theory:

'Service' is a REST object in the Kubernetes API. Services ares defined in YAML files.

With IP addresses flushed in case a new node is brought or a POD is recreated a new IPaddress is allocated, in such case we use the service in between the front-end pods of an application to talk to the back-end pods in the application. Here the Service component is important to manage the request routing to the backend Pods. In a way it is the load balancing for the backend Pods.



The way a Pod belongs to a service is done via assigning labels.
Service types:

1) ClusterIP: Stable Internet cluster IP
2) NodePort: exposes the app outside of the cluster by adding a cluster-wide port on top of the ClusterIP.
3) LoadBalancer: Integrates NodePort with loud based load balancers.

-
- Services provide reliable network endpoint
    - IP address
    - DNS Name
    - Port
- Exposes Pods to the outside world.
    - NodePort provide cluster-wide port

Kubernetes Deployments:

In Kubernetes, how deployments are managed?... it is done by defining a manifest written in JSON format and sent to the 'apiserver'.

Deployments are REST objects defined in JSON / YAML format and help in below challenges..

1) Deployment Manifests are self-documenting
2) Specify once and deploy multiple times.

3) Easy versioning
4) Simple rolling updates and rollbacks

```yaml
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: hello-deploy
spec:
  replicas: 5
  minReadySeconds: 2
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 1
      maxSurge: 1
  template:
    metadata:
      labels:
        app: hello-world-updated
    spec:
      containers:
      - name: hello-pod
        image: ganeshhp/maven-petclinic-project:latest
        ports:
        - containerPort: 8080
```

JSON / YAML files are deployed using API server on master. Uses replica sets to define number of PODS to be deployed.

Installing and getting started with Master and Nodes on Ubuntu:

Where to Install?

- Cloud
    - IAAS – Virtual Machines
    - PAAS – Kubernetes Managed service on cloud (AKS, EKS)
- On-Premises
    - Bare Metal
    - Virtual Machines

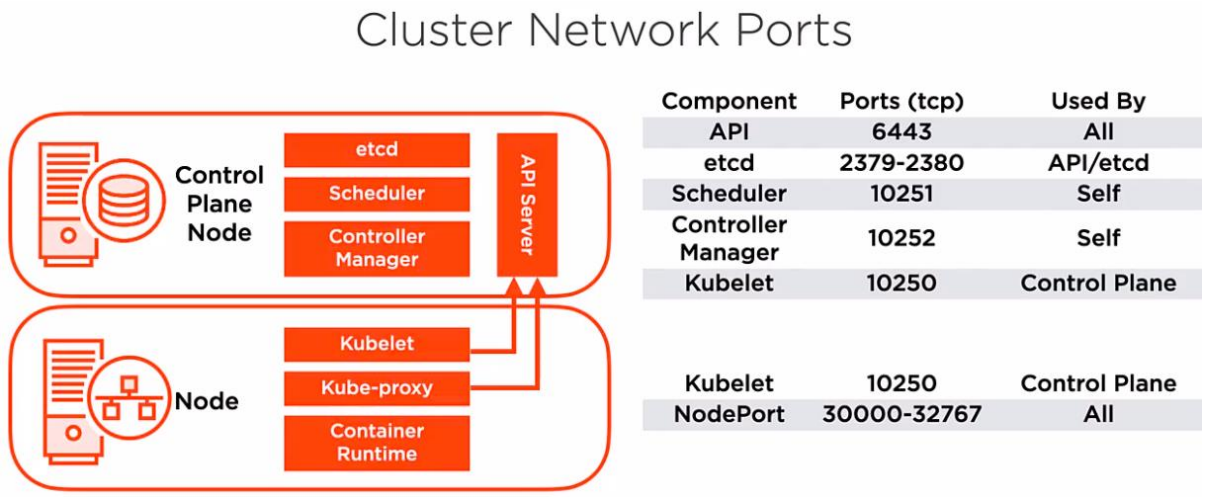
Which one do you choose?

We will first install and configure cluster on local virtual machines, in a way choosing Infra as a Service (IAAS)

Installation requirements:

System Requirements	Container Runtime	Networking
Linux - Ubuntu/RHEL	Container Runtime Interface (CRI)	Connectivity between all Nodes
2 CPUs	containerd	Unique hostname
2GB RAM	Docker (Deprecated 1.20)	Unique MAC address
Swap Disabled	CRI-O	

Setting up Network Port



Component	Ports (tcp)	Used By
API	6443	All
etcd	2379-2380	API/etcd
Scheduler	10251	Self
Controller Manager	10252	Self
Kubelet	10250	Control Plane
Kubelet	10250	Control Plane
NodePort	30000-32767	All

Installing Kubernetes standalone cluster

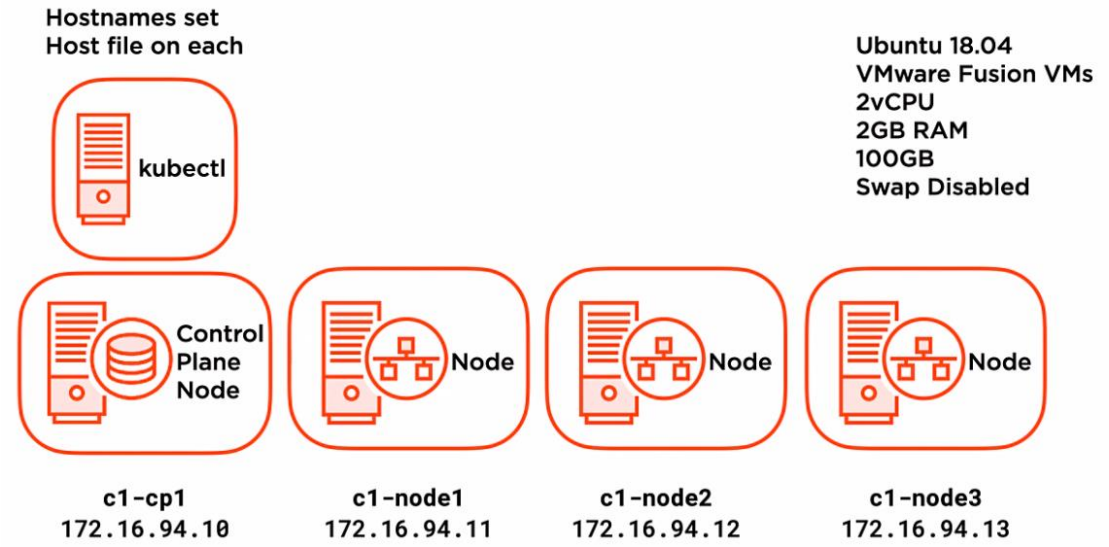
Required Packages:

- Container runtime- containerd / Docker
- kubelet – client service running on all nodes
- kubeadm – cluster administrative utility.
- Kubectl – command line utility to communicate with cluster.

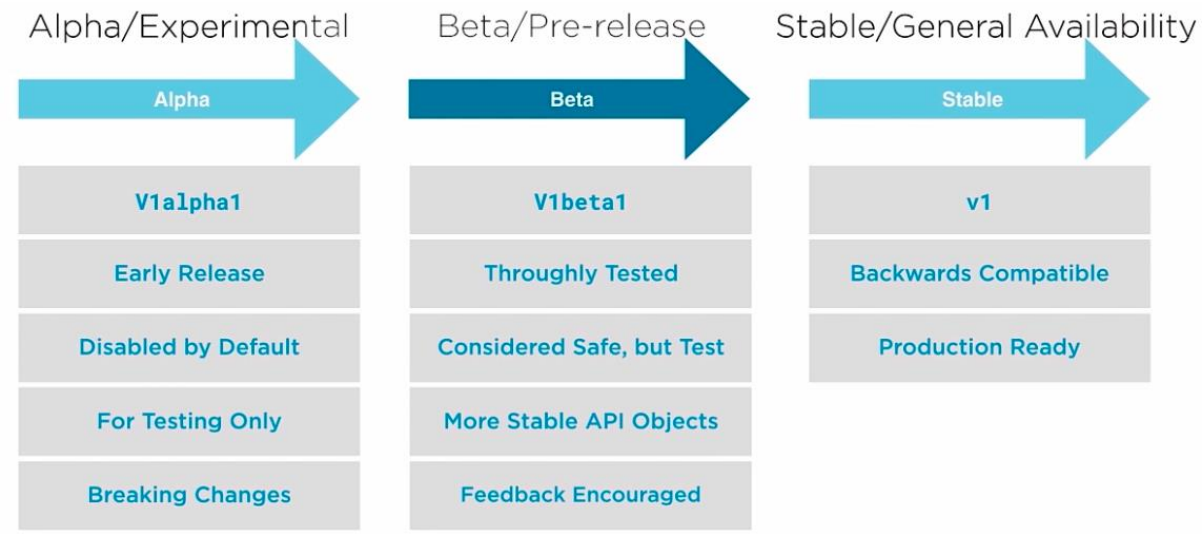
Sequence of commands:

```
sudo apt-get install -y containerd

curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -

cat <<EOF >/etc/apt/sources.list.d/kubernetes.list
deb https://apt.kubernetes.io/ kubernetes-xenial main
EOF

apt-get update
apt-get install -y kubelet kubeadm kubectl
apt-mark hold kubelet kubeadm kubectl containerd
```

Sample cluster configuration:



API Versioning:



Starting cluster installation:

```
$ swapoff –a
```

To implement swapoff, open file at `/etc/fstab` and comment out 2nd line in the file that reads about swap memory.

On the **Ubuntu VM** that's going to be the Kubernetes Master, follow below commands, to start.

```
$ apt-get update && apt-get install -y apt-transport-https
$ curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | apt-key add -
$ cat <<EOF >/etc/apt/sources.list.d/kubernetes.list
      deb http://apt.kubernetes.io/ kubernetes-xenial main
   EOF
$ apt-get update
$ apt-get install docker.io kubeadm kubectl kubelet kubernetes-cni
```

For Centos / RHEL:

Install docker-ce using below steps first,

```
$ sudo yum install -y yum-utils device-mapper-persistent-data lvm2
$ sudo yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo
$ sudo yum install docker-ce
```

```
$ sudo systemctl start docker

$ sudo systemctl enable docker

$ cat <<EOF > /etc/yum.repos.d/kubernetes.repo
   [kubernetes]
   name=Kubernetes
   baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-el7-x86_64
   enabled=1
   gpgcheck=1
   repo_gpgcheck=1
   gpgkey=https://packages.cloud.google.com/yum/doc/yum-key.gpg
   https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg
   EOF
$ yum makecache fast

$ yum update

$ yum install -y kubelet kubeadm kubectl –y

$ source <(kubectl completion bash)

$ kubectl completion bash > /etc/bash_completion.d/kubectl
```

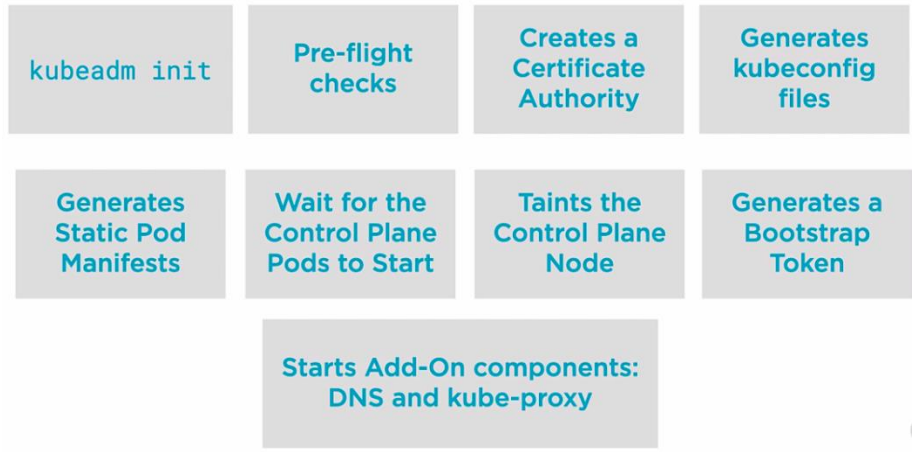
This will install apps as per below latest version available at the time of installation.

- docker.io (1.13.1-0ubuntu1~16.04.2).
- kubeadm (1.9.0-00).
- kubectl (1.9.0-00).
- kubelet (1.9.0-00).
- kubernetes-cni (0.6.0-00).

Follow above steps on all VMs irrespective of their status as ‘Nodes’ or ‘Master’.

Bootstrapping Cluster with kubeadm

Below are the steps in the process of cluster bootstrapping.



```
$ kubeadm config images pull
```

```
[root@kube-master plusforum_in]# kubeadm config images pull
[config/images] Pulled k8s.gcr.io/kube-apiserver:v1.20.4
[config/images] Pulled k8s.gcr.io/kube-controller-manager:v1.20.4
[config/images] Pulled k8s.gcr.io/kube-scheduler:v1.20.4
[config/images] Pulled k8s.gcr.io/kube-proxy:v1.20.4
[config/images] Pulled k8s.gcr.io/pause:3.2
[config/images] Pulled k8s.gcr.io/etcd:3.4.13-0
[config/images] Pulled k8s.gcr.io/coredns:1.7.0
```

First get the kubeadm Config file. If we are required to modify this file we can do it here.

```
$ kubeadm config print init-defaults | tee ClusterConfiguration.yaml
```

To create a kube cluster Run,

```
$ kubeadm init
```

Or,

```
$ kubeadm init --apiserver-advertise-address 192.168.1.8 --ignore-preflight-errors=all
```

IP address in above command is the ip address of Kubemaster server that you want to advertise.

This will download and initiate kuberadm container and other apps,

****

While running the Kubeadm join command on the nodes, in case of nodes created using vagrant, the IP address published are not for corrected NIC. To resolve this problem set the correct IP Address for the Master, by using the command,

```
kubeadm init --apiserver-advertise-address=192.168.33.135
```

****

Below message is displayed on kubemaster

```
“You should now deploy a pod network to the cluster.
Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at:
  https://kubernetes.io/docs/concepts/cluster-administration/addons/
”
```

…. You will be prompted to follow below steps.

```
$ mkdir -p $HOME/.kube

$ sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

$ sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

But you won't be able to see the nodes ready and to make those ready we have to initiate all other nodes in the cluster.

We need to install networking specific add-ons and available options are listed at,

https://kubernetes.io/docs/concepts/cluster-administration/addons/

We use the command to start network router service (POD) using below command,

```
$ kubectl apply --filename https://git.io/weave-kube-1.6
```

Or,

```
$ kubectl apply -f "https://cloud.weave.works/k8s/net?k8s-version=$(kubectl version | base64 | tr -d '\n')"
```

Now on the VMs that are going to be the K8S nodes, run below command

```
$ kubeadm join --token fc3846.48223acdabb2ae31 <master-ipaddress>:6443 --discovery-token-ca-cert-hash sha256:a905fc15814645e8a1cb3a18234feea5206263db820889d0b773cc1cd7751a0e
```

If the Kubeadm join command is not available, we can create a new token with below command,

```
$ sudo kubeadm token create --print-join-command
```

Or, below can retrieve existing token

```
$ kubeadm join --discovery-token-unsafe-skip-ca-verification --token=`kubeadm token list` 172.17.0.92:6443.
```

Also we can list all available and valid tokens using command,

```
$ kubeadm token list
```