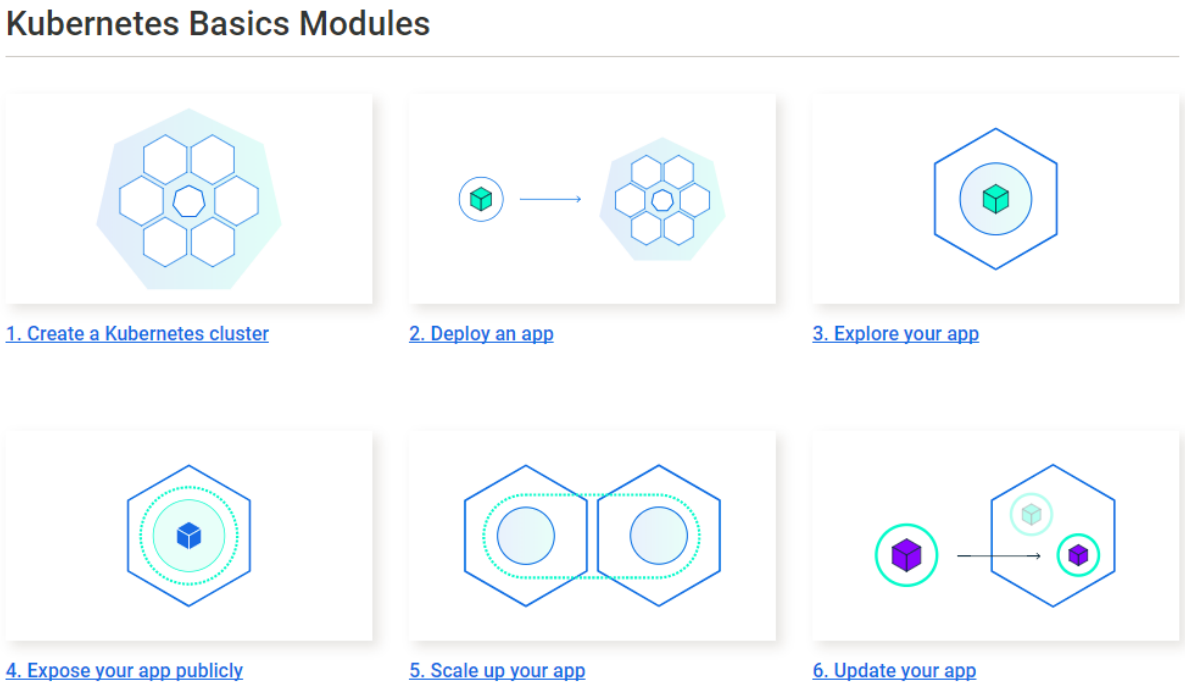
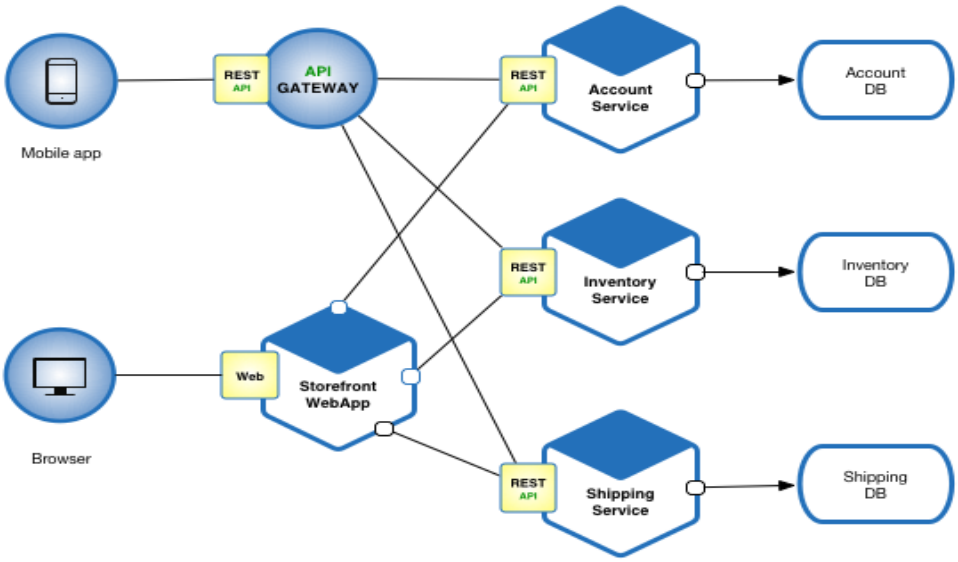
What is Kubernetes?



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...

Enables developers to deploy their applications themselves and as often as they want

Enables the ops team by automatically monitoring and rescheduling those apps in the event of a hardware failure

The focus from supervising individual apps to mostly supervising and managing Kubernetes and the rest of the infrastructure

Abstracts away the hardware infrastructure and exposes your whole datacenter as a single gigantic computational resource

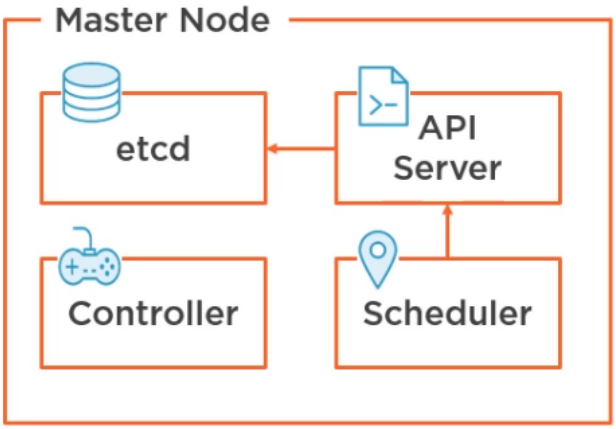
Master and Node:

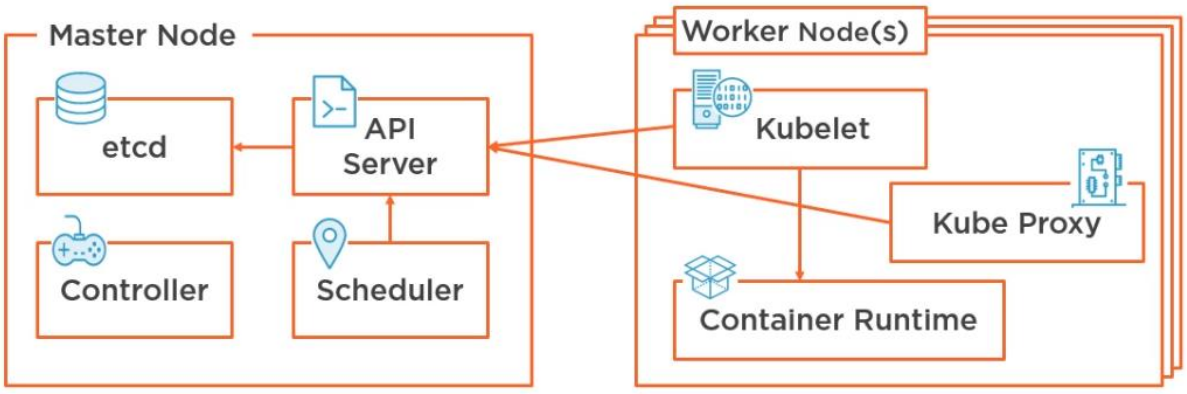
K8S Master:

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'.**





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.

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.

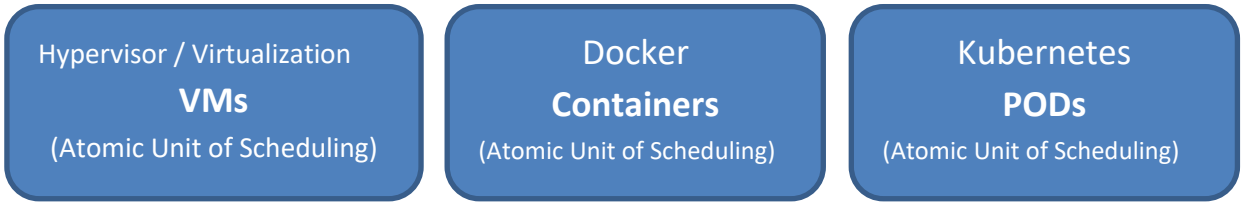
So it we want to have 10 PODS running always with a particular version of a file deployed to it, this is defined in the manifest and K8S will always make sure to achieve this state.

Working with Pods

PODs theory:

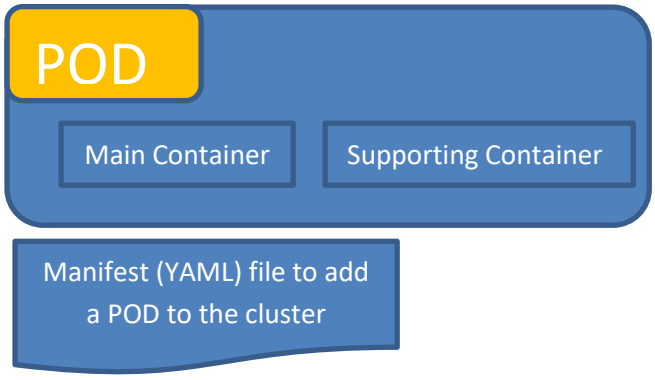
1. POD is the smallest atomic unit of scheduling in Kubernetes.
2. One POD can contain multiple containers in it with same or multiple services running in it.
3. But one POD can only be running on one node and can never be split on multiple nodes.
4. PODs are declarative via Manifest files.

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.



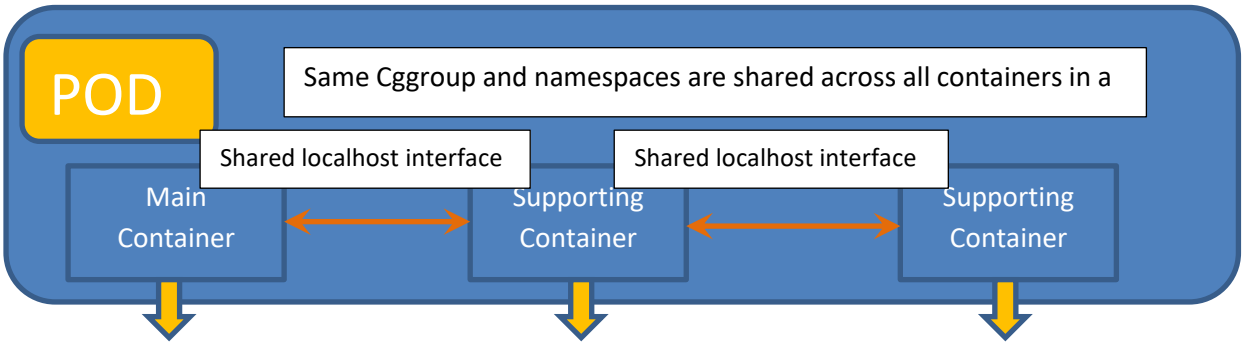

```
[root@my-node1 ~] $ mkdir /etc/kubelet.d/
[root@my-node1 ~] $ cat <<EOF >/etc/kubelet.d/static-web.yaml
apiVersion: v1
kind: Pod
metadata:
  name: static-web
  labels:
    role: myrole
spec:
```

```
  containers:
    - name: web
      image: nginx
      ports:
        - name: web
          containerPort: 80
          protocol: TCP
EOF
```

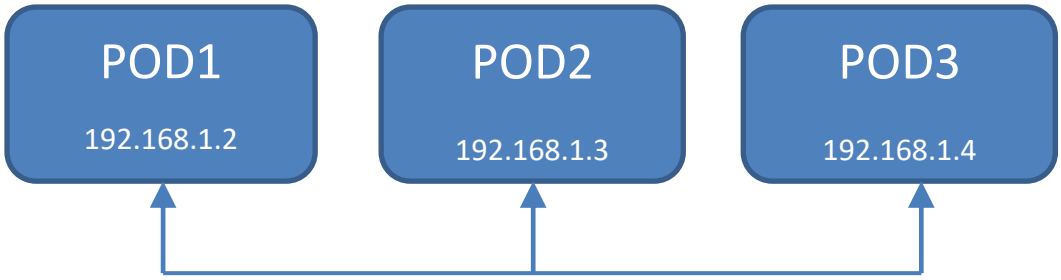
This file is then fed to the API server and the scheduler deploys this to a Kube node.

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

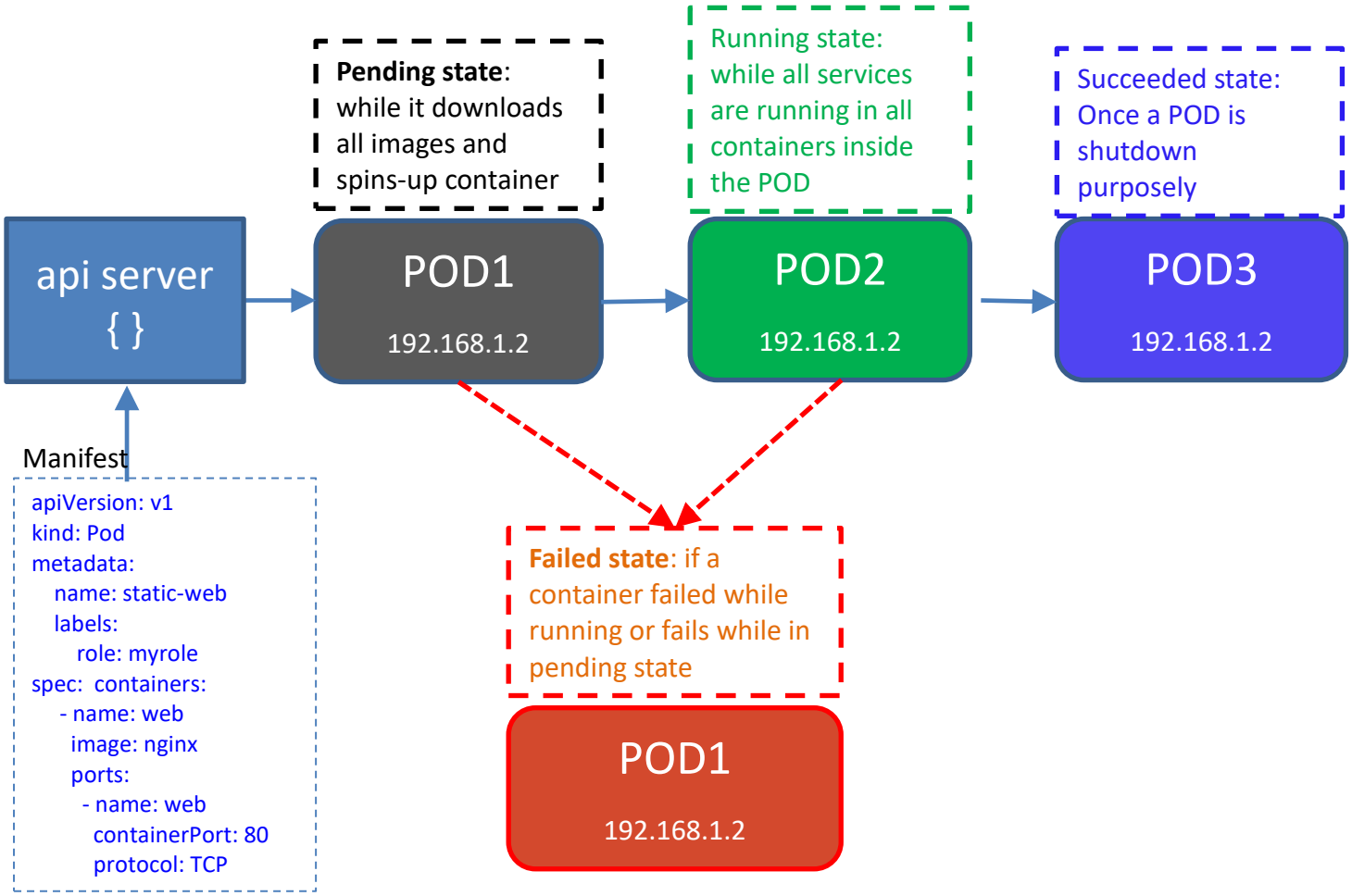


Inter POD communication: POD network

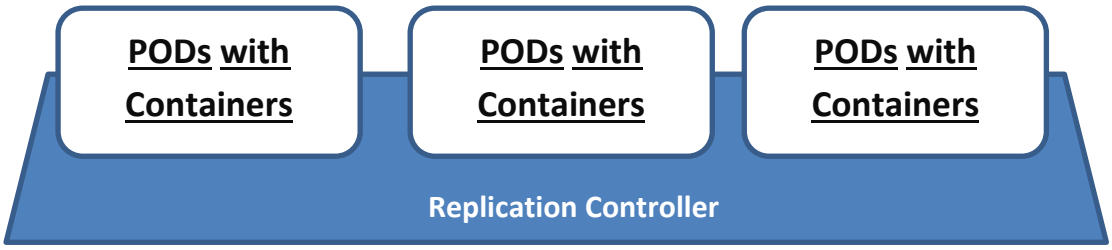


Like Container in Docker, PODs are mortal units in Kubernetes. We never get into the mode of repairing a POD.

PODs : Lifecycle



Replication Controller:

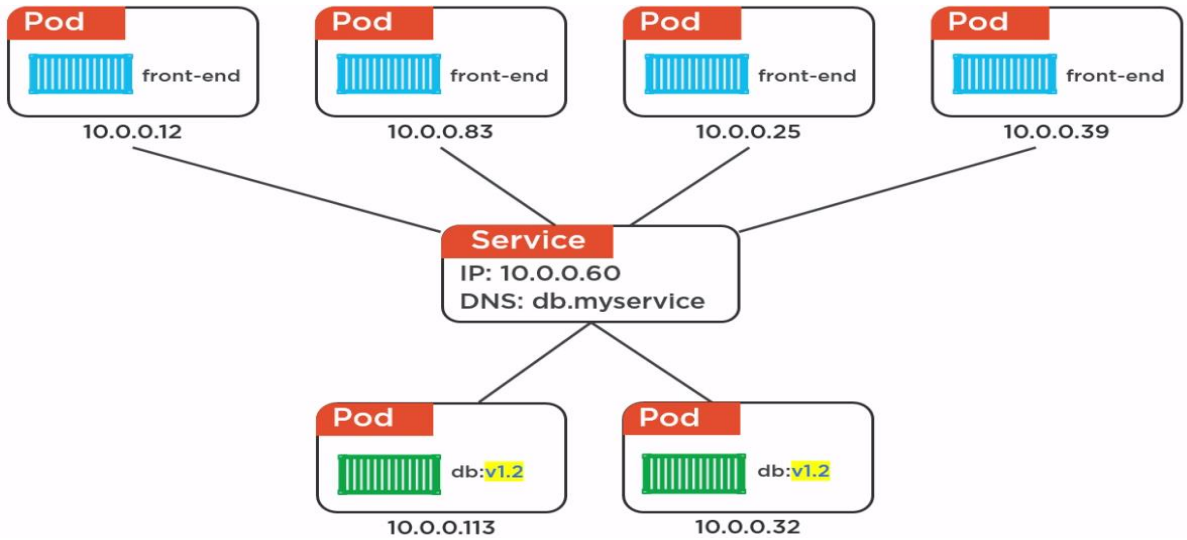


Kubernetes Services:

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:

On the 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:

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
$ kubeadm config images pull
```

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’.

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 on the node that we want to add..

```
$ kubeadm join --token fc3846.48223acdabb2ae31 <master-
ipaddress>:6443 --discovery-token-ca-cert-hash
sha256:a905fc15814645e8a1cb3a18234feea5206263db820889d0b773cc1cd7751
a0e
```

If the Kubeadm join command is not available, we can create anew token it 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
```

Once the pods are joined we can check the same on the Kubernetes master

```
root@kube-master:#kubectl get nodes

NAME          STATUS    ROLES     AGE       VERSION
kube-master   Ready     master    21m       v1.9.0
kubenode-1    Ready     <none>    2m        v1.9.0
kubenode-2    Ready     <none>    39s       v1.9.0
```

Here we can see the additional nodes which have hostname as ‘kubenode-1’ and ‘kubenode-2’.

Now that we are done with our Master and Node setup let’s start with getting the PODs ready,

Starting with PODs practical:

The POD is defined using a manifest file.

Here is sample manifest file.. pod.yml

```yaml
apiVersion: v1
kind: Pod
metadata:
    name: static-web
    labels:
      role: myrole
      zone: prod
      version: v1
spec:
    containers:
    - name: web
      image: nginx
      ports:
      - name: web
        containerPort: 80
        protocol: TCP
```

Now the manifest is fed to the api-server using below command,

```
root@kube-master:# kubectl create -f pod.yml

pod "static-web" created
```

Check the status of POD using command,

```
root@kube-master:# kubectl get pods

NAME          READY     STATUS    RESTARTS   AGE
static-web    1/1       Running   0          14s
```

To check the POD status let's run the describe command.

```
root@kube-master:# kubectl describe pods

Name:         static-web
Namespace:    default
Node:         kubenode-2/10.142.0.4
Start Time:   Tue, 02 Jan 2018 08:18:51 +0000
Labels:       role=myrole
              version=v1
              zone=prod
Annotations:  <none>
Status:       Running
IP:           10.36.0.1
Containers:
  web:
    Container                                                                ID:
docker://8cabde7b6892ba3db454f0d73622d05659698f0ed484364c5d990b22e970fe73
    Image:          nginx
    Image       ID:                                                          docker-
pullable://nginx@sha256:cf8d5726fc897486a4f628d3b93483e3f391a76ea4897de0500
ef1f9abcd69a1
    Port:           80/TCP
    State:          Running
      Started:      Tue, 02 Jan 2018 08:18:59 +0000
    Ready:          True
```

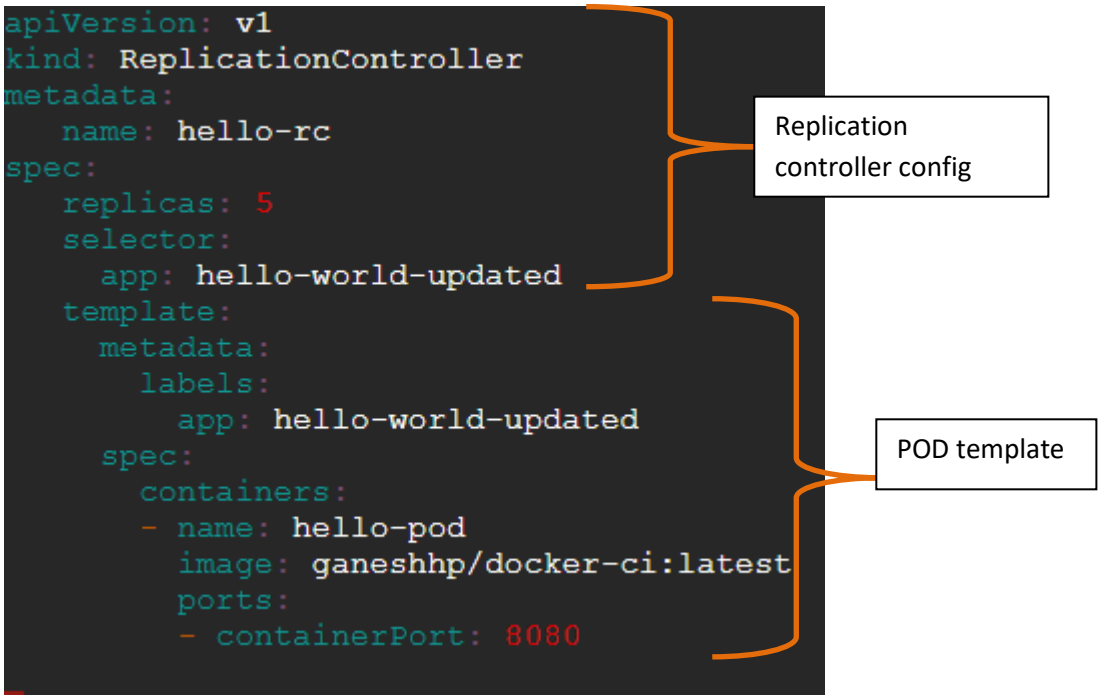
```
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount   from   default-token-
qtvzc (ro)
Conditions:
  Type           Status
  Initialized    True
  Ready          True
  PodScheduled   True
Volumes:
  default-token-qtvzc:
    Type:        Secret (a volume populated by a Secret)
    SecretName:  default-token-qtvzc
    Optional:    false
QoS Class:       BestEffort
Node-Selectors:  <none>
Tolerations:     node.kubernetes.io/not-ready:NoExecute for 300s
                 node.kubernetes.io/unreachable:NoExecute for 300s
Events:
  Type    Reason                 Age   From                 Message
  ----    ------                 ----  ----                 -------
  Normal  Scheduled              1m    default-scheduler    Successfully
assigned static-web to kubenode-2
  Normal      SuccessfulMountVolume      1m           kubelet,    kubenode-2
MountVolume.SetUp succeeded for volume "default-token-qtvzc"
  Normal  Pulling                1m    kubelet, kubenode-2  pulling image
"nginx"
  Normal  Pulled                 1m    kubelet, kubenode-2  Successfully
pulled image "nginx"
  Normal  Created                1m    kubelet, kubenode-2  Created
container
  Normal  Started                1m    kubelet, kubenode-2  Started
container
```

To delete a POD, we run the command,

```
$ kubectl delete pods <pod-name>
```

```
root@kube-master:/home/ghpalnitkar/Kube-project# kubectl delete pods static-web
pod "static-web" deleted
```

Now let's look at Replication Controller, which is the right ways of applying Desired State Configuration.



So here the replication controller manifest is the actual 'Desired State Configuration' (DSC), where we state how many PODs we want to create and also supply the POD configuration that's what we have seen in earlier POD manifest.

Now try running the file one more time again using the same command,

```
$ kubectl create -f rc.yml

$ kubectl get pods

NAME                          READY     STATUS    RESTARTS   AGE
static-web-replica-bh8rk      1/1       Running   0          23s
static-web-replica-m9ck4      1/1       Running   0          23s
static-web-replica-pkd8h      1/1       Running   0          23s
static-web-replica-qwflh      1/1       Running   0          23s
static-web-replica-t4sjs      1/1       Running   0          23s

$ Kubectl describe pods static-web-replica-bh8rk

Name:           static-web-replica-bh8rk
Namespace:      default
Node:           kubenode-1/10.142.0.3
Start Time:     Tue, 02 Jan 2018 09:06:02 +0000
Labels:         app=static-web
Annotations:    <none>
Status:         Running
IP:             10.44.0.1
Controlled By:  ReplicationController/static-web-replica
Containers:
  web:
    Container                                                                ID:
docker://781f3009341dbc5df4beb5b54b815d39063dc2a16737cf2b7e6d71a1b06b3c27
```

```
    Image:          nginx
    Image       ID:                                                    docker-
pullable://nginx@sha256:cf8d5726fc897486a4f628d3b93483e3f391a76ea4897de0500
ef1f9abcd69a1
    Port:           80/TCP
    State:          Running
      Started:      Tue, 02 Jan 2018 09:06:11 +0000
    Ready:          True
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount   from   default-token-
qtvzc (ro)
Conditions:
  Type           Status
  Initialized    True
  Ready          True
  PodScheduled   True
Volumes:
  default-token-qtvzc:
    Type:        Secret (a volume populated by a Secret)
    SecretName:  default-token-qtvzc
    Optional:    false
QoS Class:       BestEffort
Node-Selectors:  <none>
Tolerations:     node.kubernetes.io/not-ready:NoExecute for 300s
                 node.kubernetes.io/unreachable:NoExecute for 300s
Events:
  Type    Reason                 Age   From                 Message
  ----    ------                 ----  ----                 -------
  Normal  Scheduled              1m    default-scheduler    Successfully
assigned static-web-replica-bh8rk to kubenode-1
  Normal      SuccessfulMountVolume      1m           kubelet,    kubenode-1
MountVolume.SetUp succeeded for volume "default-token-qtvzc"
  Normal  Pulling                1m    kubelet, kubenode-1  pulling image
"nginx"
  Normal  Pulled                 1m    kubelet, kubenode-1  Successfully
pulled image "nginx"
  Normal  Created                59s   kubelet, kubenode-1  Created
container
  Normal  Started                59s   kubelet, kubenode-1  Started
container
```

Now if we make any changes to the rc.yml file we can use command, to apply the changes to the cluster.

```
$ kubectl apply –f rc.yml
```

Here check the age of container the earlier containers will keep on running and additional gets added.

```
root@kube-master:/home/ghpalnitkar/Kube-project# kubectl apply -f rc.yml
Warning: kubectl apply should be used on resource created by either kubectl create --save-config or kubectl apply
replicationcontroller "static-web-replica" configured
root@kube-master:/home/ghpalnitkar/Kube-project# kubectl get pods
NAME                     READY     STATUS    RESTARTS   AGE
static-web-replica-2nxqp   1/1       Running   0          27s
static-web-replica-bh8rk   1/1       Running   0          21m
static-web-replica-jx7pk   1/1       Running   0          27s
static-web-replica-ksgsg   1/1       Running   0          27s
static-web-replica-m9ck4   1/1       Running   0          21m
static-web-replica-pkd8h   1/1       Running   0          21m
static-web-replica-qwflh   1/1       Running   0          21m
static-web-replica-t4sjs   1/1       Running   0          21m
static-web-replica-v5nzr   1/1       Running   0          27s
static-web-replica-x6dx7   1/1       Running   0          27s
root@kube-master:/home/ghpalnitkar/Kube-project#
```

Similarly if we reduce the number of container, kubernetes will make sure to destroy surplus containers and keep the one running as mentioned in the rc.yml.

```
root@kube-master:/home/ghpalnitkar/Kube-project# kubectl apply -f rc.yml
replicationcontroller "static-web-replica" configured
root@kube-master:/home/ghpalnitkar/Kube-project# kubectl get pods
NAME                     READY     STATUS    RESTARTS   AGE
static-web-replica-bh8rk   1/1       Running   0          25m
static-web-replica-pkd8h   1/1       Running   0          25m
static-web-replica-qwflh   1/1       Running   0          25m
static-web-replica-t4sjs   1/1       Running   0          25m
```

K8S Services:

We can create a service object by firing a command as shown below.

```
$ kubectl expose rc static-web-replica --name=web-service --target-port=80 --type=NodePort
```

Here we are creating a service type object ‘web-service’ to and exposing the replication cluster port 80 of each pod to the service on port 80.

The type of the service that is used is NodePort service.

But running the service using command line is not always the option. We use a YAML file to define the service and tag the service to a replication controller which in turn runs PODs.

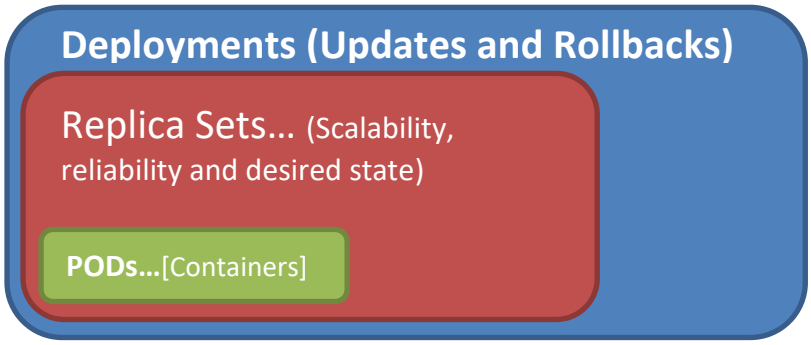
Below is such YAML file for Service.

```yaml
apiVersion: v1
kind: Service
metadata:
  name: hello-service
  labels:
    app: hello-world-updated
spec:
  type: NodePort
  ports:
  - port: 8080
    nodePort: 30000
    protocol: TCP
  selector:
    app: hello-world-updated
```

```
$ kubectl describe service
$ kubectl describe ep <same-as-service-name>
```

Kubernetes Deployment:

Deployments manage Replica sets and Replica sets manage PODs.



Replication controllers have been replaced by Replica sets in deployment object.

```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
```

In order to allow PODs to be created on kubernetes master, use below command,

```
$ kubectl taint nodes --all node-role.kubernetes.io/master-
```

Horizontal POD Autoscaler (HPA):

To AutoScale existing deployment object in cluster.

```
$ kubectl autoscale deployment.apps/wordpress-deploy --min=2 --max=10

$ kubectl autoscale deployment.apps/wordpress-deploy --max=10 --cpu-percent=80

$ kubectl autoscale rc.apache=rc/hello-apache --max=10 --cpu-percent=80
```

Few handy commands…

1) Using Command line to create PODS and other Kubernetes objects…

```
$ kubectl run hello-world --replicas=5 --labels="run=load-balancer-example" --image=gcr.io/google-samples/node-hello:1.0 --port=8080
```

2) Setting up environment variables in Container running inside cluster…

```
apiVersion: v1
kind: Pod
metadata:
  name: print-greeting
spec:
  containers:
  - name: env-print-demo
    image: bash
    env:
    - name: GREETING
      value: "Warm greetings to"
    - name: HONORIFIC
      value: "The Most Honorable"
    - name: NAME
      value: "Kubernetes"
    command: ["echo"]
    args: ["$(GREETING) $(HONORIFIC) $(NAME)"]
```

3) Creating service object from command line to expose existing deployment.

```
$ kubectl expose deployment hello-world --type=LoadBalancer --
name=my-service
```

4) Get information of PODs / namespaces running on a specific node in Kubernetes cluster, use below command.

```
$ kubectl get pods --all-namespaces --field-selector
spec.nodeName=k8s2 -o wide
```

5) In case if a nodes is to be taken out of the kubernetes cluster, use below steps,

```
$ kubectl cordon k8s1
```

Once the Node is disabled for scheduling, the PODs are needed to be shifted from the cordoned node.

```
$ kubectl drain k8s1 --delete-local-data --ignore-daemonsets
```

Noe after the nodes os corrected and ready to join the cluster we can 'uncordon' the node and run the deployment manifest file again, so that all PODs are rescheduled on the added Node.

```
$ kubectl uncordon k8s1
```

```
$ kubectl apply –f deployment.yml
```

To get detailed information on running object, use –o wide switch, such as,

```
$ kubectl get all –o wide
```

Incase of error faced while initing Kubeadm on VM running on VMware or VirtualBox:

- kubeadm reset
- echo 'Environment="KUBELET_EXTRA_ARGS=--fail-swap-on=false"' >> /etc/systemd/system/kubelet.service.d/10-kubeadm.conf
- systemctl daemon reload
- systemctl daemon-reload
- systemctl restart kubelet
- swapoff –a
- kubeadm init

if required run the below command with ignoring error.

- kubeadm init --ignore-preflight-errors=all

For many steps here you will want to see what a `Pod` running in the cluster sees.

The simplest way to do this is to run an interactive busybox `Pod`:

```
$ kubectl run -it --rm --restart=Never busybox --image=busybox sh
```

If you don't see a command prompt, try pressing enter.

```
/ #
```

If you already have a running `Pod` that you prefer to use, you can run a command in it using:

```
$ kubectl exec <POD-NAME> -c <CONTAINER-NAME> -- <COMMAND>
```

Create object ‘secret’.

```
$  kubectl create secret generic mysql-pass --from-literal=password=my_password
```

‘Secrete’ being an object can be explored using describe command.