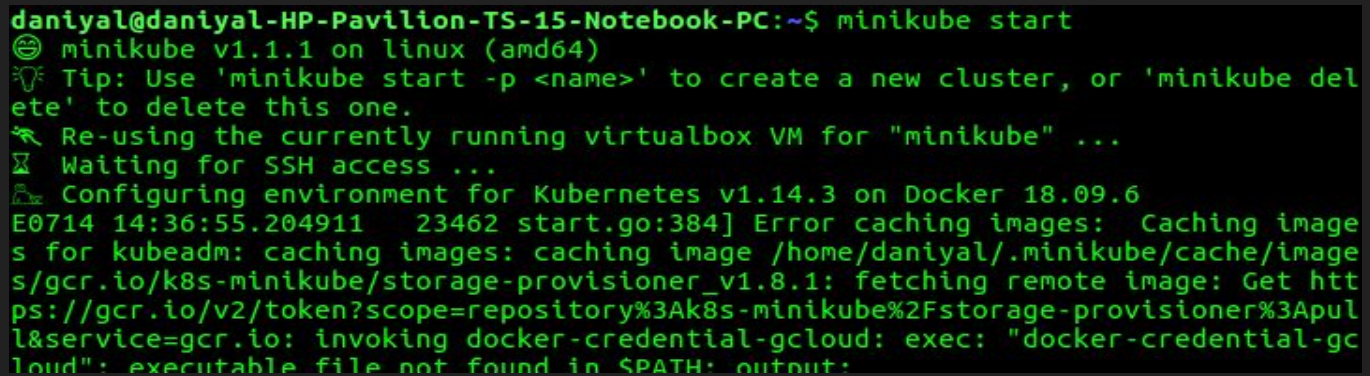
KUBERNETES

MINIKUBE

- **Minikube** is a tool that makes it easy to run **Kubernetes** locally.
- **Minikube** runs a single-node **Kubernetes** cluster on your laptop.
- Kubernetes architecture consist of Master node and worker nodes
- Minikube is a tool that makes it easy to run Kubernetes locally
- Minikube runs a single-node Kubernetes cluster on your laptop to use kubernetes for practice or development
- To start Kubernetes cluster using minikube.



MINIKUBE COMMANDS

To check the minikube status

```
daniyal@daniyal-HP-Pavilion-TS-15-Notebook-PC:~$ minikube status
host: Running
kubelet: Running
apiserver: Running
kubectl: Correctly Configured: pointing to minikube-vm at 192.168.99.105
daniyal@daniyal-HP-Pavilion-TS-15-Notebook-PC:~$
```

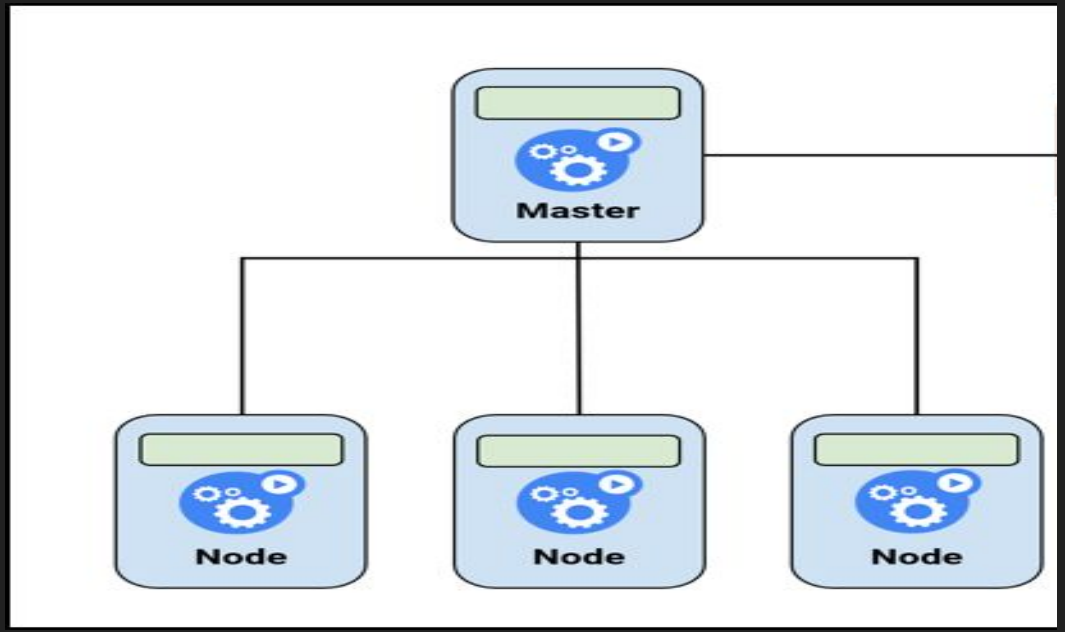
To check the cluster info like addresses of the kubernetes master and services

```
daniyal@daniyal-HP-Pavilion-TS-15-Notebook-PC:~$ kubectl cluster-info
Kubernetes master is running at https://192.168.99.105:8443
KubeDNS is running at https://192.168.99.105:8443/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.
daniyal@daniyal-HP-Pavilion-TS-15-Notebook-PC:~$
```

NODES

- A **node** is a worker machine in **Kubernetes**.
- A **node** may be a VM or physical machine.



NODES

Get the list of the nodes

```
daniyal@daniyal-HP-Pavilion-TS-15-Notebook-PC:~$ kubectl get nodes
NAME       STATUS   ROLES    AGE   VERSION
minikube   Ready    master   16d   v1.14.3
daniyal@daniyal-HP-Pavilion-TS-15-Notebook-PC:~$ kubectl get node
NAME       STATUS   ROLES    AGE   VERSION
minikube   Ready    master   16d   v1.14.3
daniyal@daniyal-HP-Pavilion-TS-15-Notebook-PC:~$ kubectl get no
NAME       STATUS   ROLES    AGE   VERSION
minikube   Ready    master   16d   v1.14.3
daniyal@daniyal-HP-Pavilion-TS-15-Notebook-PC:~$
```

Get the detail of all nodes

```
daniyal@daniyal-HP-Pavilion-TS-15-Notebook-PC:~$ kubectl describe no
Name:               minikube
Roles:              master
Labels:             beta.kubernetes.io/arch=amd64
                    beta.kubernetes.io/os=linux
                    kubernetes.io/arch=amd64
                    kubernetes.io/hostname=minikube
                    kubernetes.io/os=linux
                    node-role.kubernetes.io/master=
```

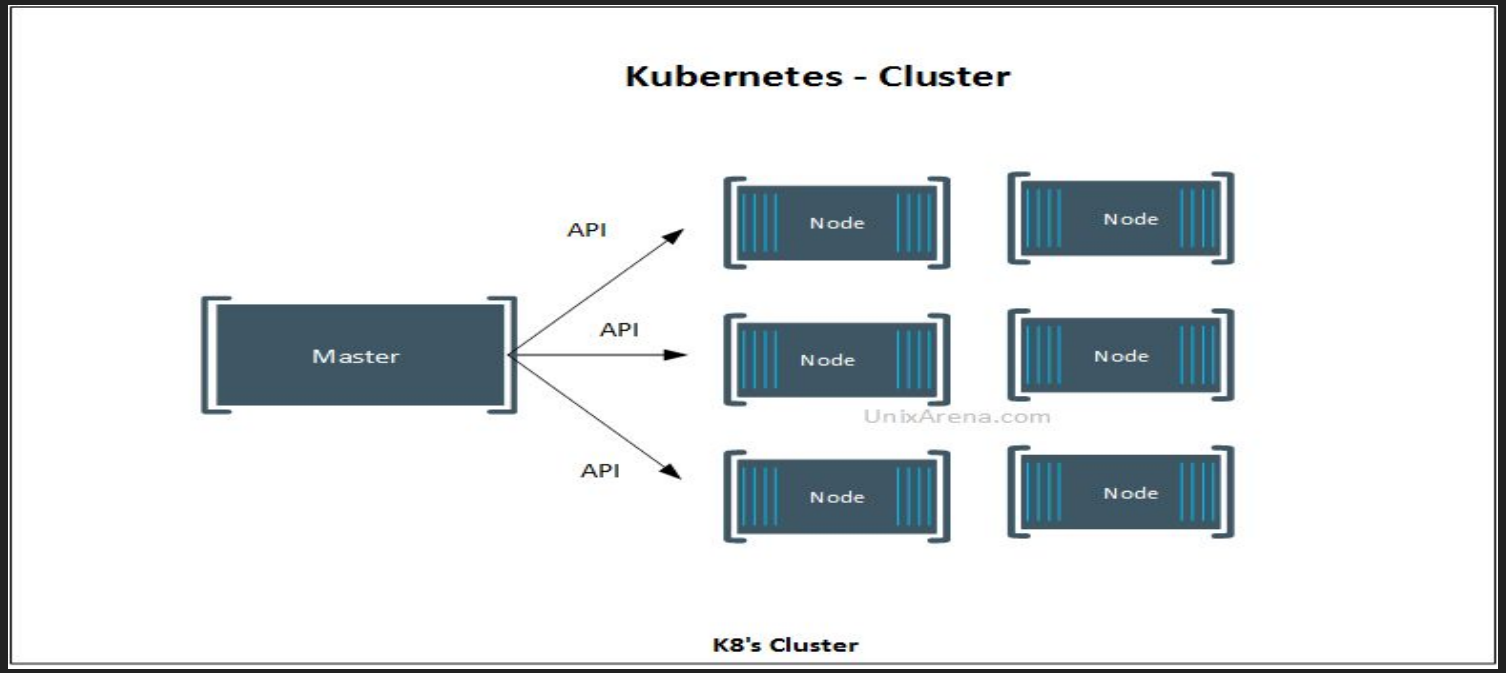
ALIAS

We can use alias command to make the long command shorter.

```
daniyal@daniyal-HP-Pavilion-TS-15-Notebook-PC:~$ alias kgn="kubectl get nodes"
daniyal@daniyal-HP-Pavilion-TS-15-Notebook-PC:~$ kgn
NAME       STATUS   ROLES    AGE   VERSION
minikube   Ready    master   16d   v1.14.3
daniyal@daniyal-HP-Pavilion-TS-15-Notebook-PC:~$ alias kgp="kubectl get pod"
daniyal@daniyal-HP-Pavilion-TS-15-Notebook-PC:~$ kgp
NAME     READY   STATUS    RESTARTS   AGE
my-pod   1/1     Running   1          6d16h
daniyal@daniyal-HP-Pavilion-TS-15-Notebook-PC:~$
```

CLUSTER

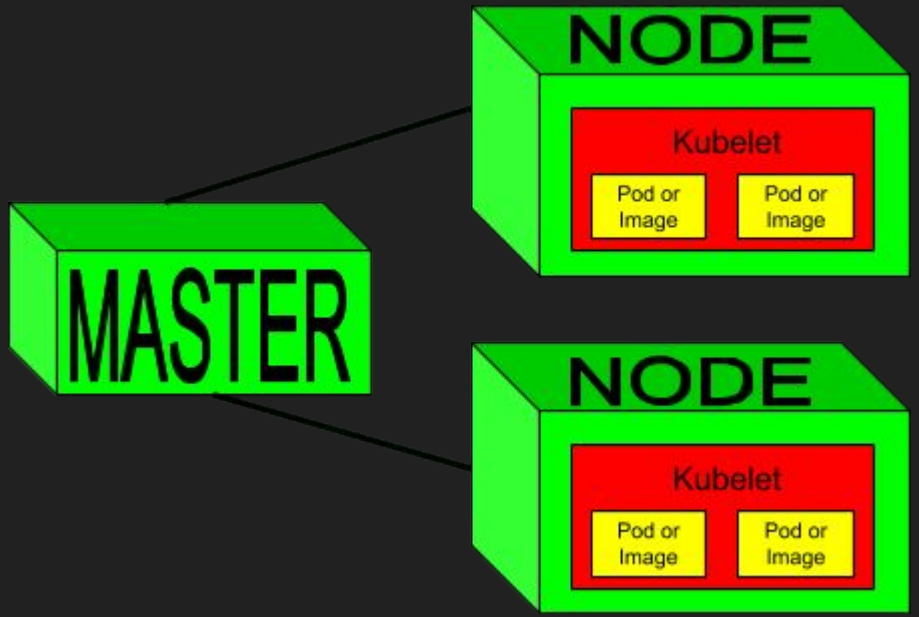
- Cluster is a set of nodes group together.
- A **CLUSTER** consists of at least one master and multiple worker machines called nodes.



MASTER

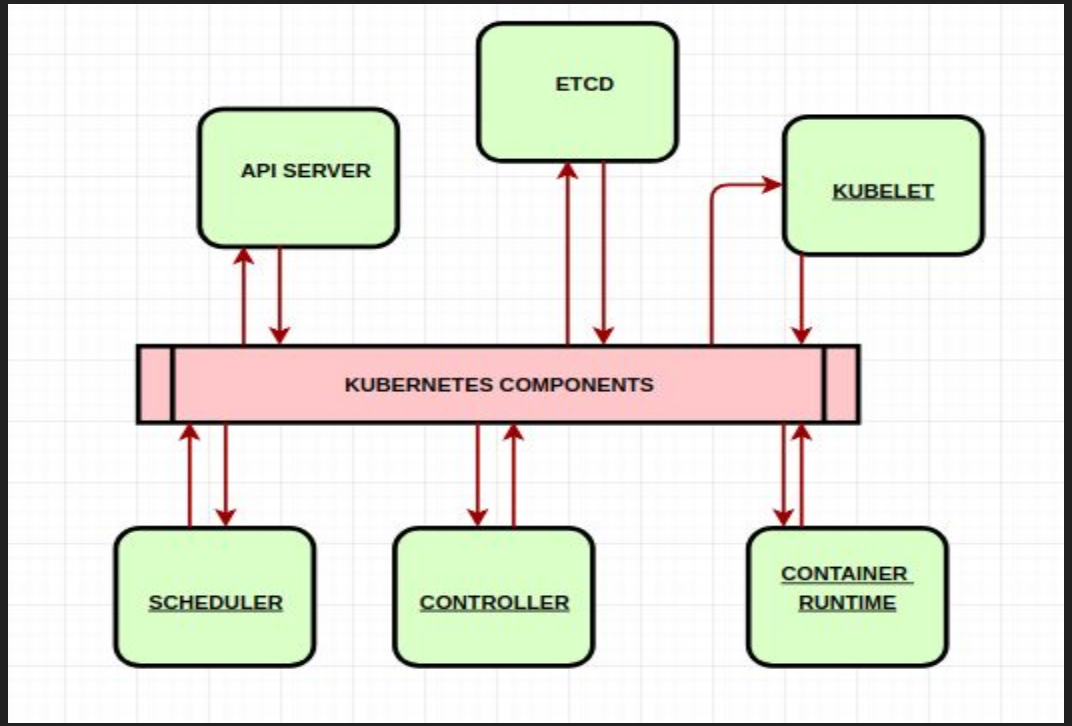
The **Kubernetes master** is responsible for maintaining the desired state for your **cluster**.

The master components manage the state of the cluster. This includes accepting client requests (describing the desired state), scheduling containers and running control loops to drive the actual cluster state towards the desired state.



COMPONENTS

When install kubernetes on a system the following components install with it.



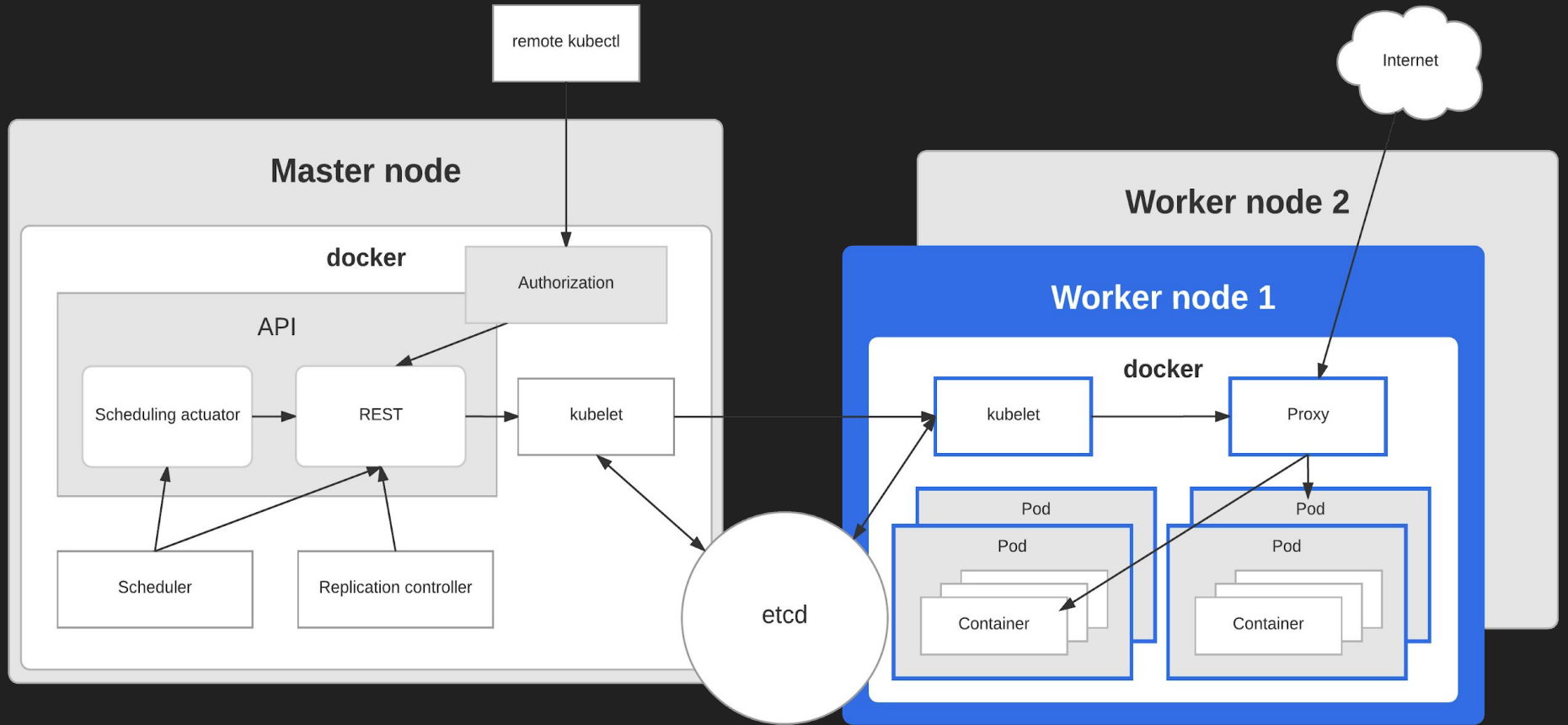
COMPONENTS

- API Server
  - The API server acts as the frontend for kubernetes the users, management devices, command line interfaces all talk to the API Server to interact with the kubernetes cluster.
- ETCD
  - **Kubernetes** uses **etcd** to store all its data. Its configuration data, its state, and its metadata.
  - **Kubernetes** is a distributed system, so it needs a distributed data store like **etcd**.
  - **Etcd** lets any of the nodes in the **Kubernetes** cluster read and write data.
- SCHEDULER
  - The scheduler is responsible for distributing work or containers across multiple nodes. It looks for newly created containers and assigns them to nodes.
- CONTROLLER
  - The controllers are the brain behind orchestration. They are responsible for noticing and responding when nodes, containers and end-points goes down.
  - The controllers make decisions to bring up new containers in such cases.

COMPONENTS

- CONTAINER-RUNTIME
  - The container runtime is the underline software that is use to run containers, in our case it happens to be docker but there are other options as well.
- KUBELET
  - The kubelet is the agent that runs on each node in the cluster.
  - The agent is responsible for making sure that the containers are running on the nodes as expected.

Master vs Worker Nodes



PODs

Kubernetes aim is to deploy our application in the form of containers on a set of machines that are configured as a worker nodes in a cluster.

Kubernetes does not deploy containers directly on the worker nodes.

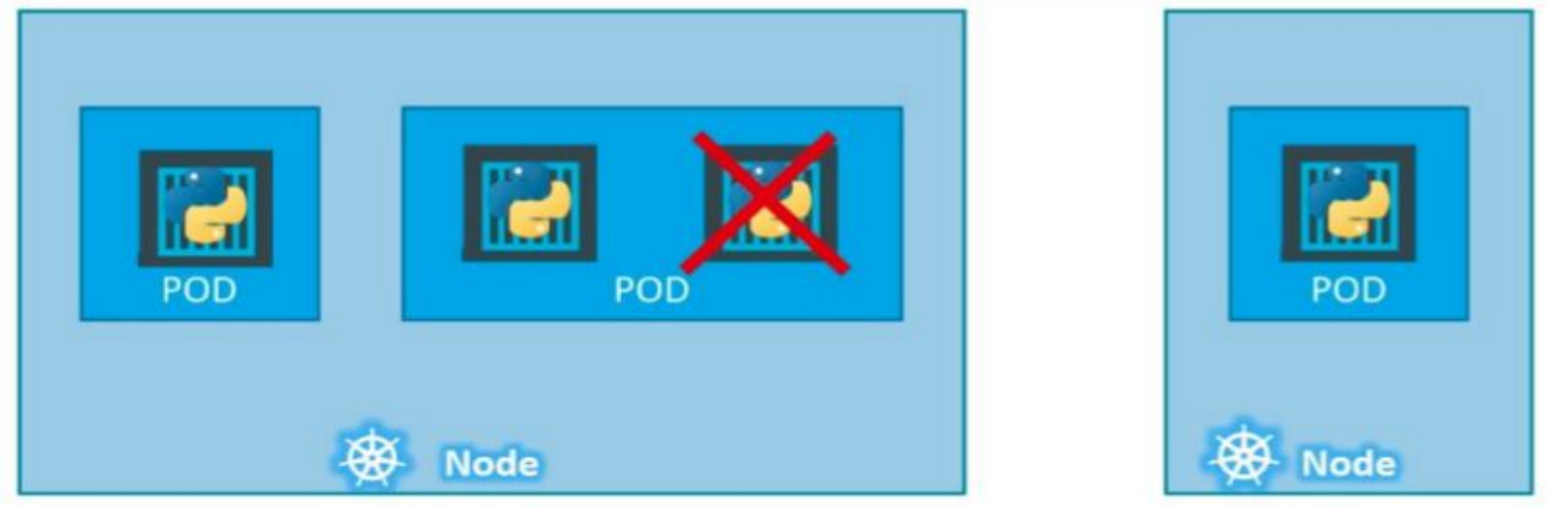
The containers are encapsulated into a Kubernetes object known as POD’s.

A POD is a single instance of an application.

A POD is the smallest object that can create in kubernetes.

POD

- A **Kubernetes pod** is a group of containers that are deployed together on the same host.
- If you frequently deploy single containers, you can generally replace the word "**pod**" with "**container**" and accurately understand the concept.
- To scale up your application, you add more pods. You delete pods if you scale down.



KUBECTL

- The primary command line utility for interacting with a k8s cluster is kubectl.
- Kubectl (Kube control tool) is use to deploy and manage applications on kubernetes cluster.
- kubectl controls the Kubernetes cluster manager.
- Kubectl is a command line interface for running commands against Kubernetes clusters.

Deploy A POD Using **KUBECTL**

- What does kubectl run command?
  - It Deploys a Docker container by creating a POD.
  - It first creates a POD automatically and deploys an instance of the nginx Docker image.
- Where does it get the application image from?
  - You need to specify the image name using the --image parameter.
  - The application image in this case the nginx image is downloaded from the docker hub repository.

```
daniyal@daniyal-HP-Pavilion-TS-15-Notebook-PC:~$ kubectl run nginx --image nginx
```

POD

- A pod of containers allows you to run closely related processes together
- Kubernetes provide these containers with (almost) the same environment as if they were all running in a single container, while keeping them somewhat isolated
- “Somewhat isolated”, this is because you want containers inside each group to share certain resources, although not all, so that they’re not fully isolated
- For example if your main application’s container write logs in some file and you want another application to use some part of that log file and make some report.

POD IPs

- To get the POD IPs.

```
daniyal@daniyal-HP-Pavilion-TS-15-Notebook-PC:~$ kubectl describe pod | grep IP
IP:           172.17.0.5
IP:             172.17.0.7
daniyal@daniyal-HP-Pavilion-TS-15-Notebook-PC:~$
```

YAML FILE

It contains 4 mandatory top level specification:

a. apiVersion

b. Kind

c. metadata

d. spec

YAML

pod-definition.yml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: myapp-pod
  labels:
    app: myapp
    type: front-end
spec:
  containers:
    - name: nginx-container
      image: nginx
```

Kind	Version
POD	v1
Service	v1
ReplicaSet	apps/v1
Deployment	apps/v1

YAML

```
admin-pc@adminpc-desktop:/media/admin-pc/9C80A0D480A0B5E4/Kubernetes/Pods$ sudo kubectl get pods
No resources found.
admin-pc@adminpc-desktop:/media/admin-pc/9C80A0D480A0B5E4/Kubernetes/Pods$ sudo kubectl create -f pod-def.yml
pod/myapp-pod1 created
admin-pc@adminpc-desktop:/media/admin-pc/9C80A0D480A0B5E4/Kubernetes/Pods$ sudo kubectl get pods
NAME         READY   STATUS              RESTARTS   AGE
myapp-pod1   0/1     ContainerCreating   0          3s
admin-pc@adminpc-desktop:/media/admin-pc/9C80A0D480A0B5E4/Kubernetes/Pods$ sudo kubectl delete pod myapp-pod1
pod "myapp-pod1" deleted
admin-pc@adminpc-desktop:/media/admin-pc/9C80A0D480A0B5E4/Kubernetes/Pods$ sudo kubectl get pods
No resources found.
admin-pc@adminpc-desktop:/media/admin-pc/9C80A0D480A0B5E4/Kubernetes/Pods$
```

KUBERNETES Commands

1. Get Pods
    a. Kubectl get pods
2. Delete Pods
    a. Kubectl delete pod <pod name>

KUBERNETES

- If you are given a pod definition file, edit that file and use it to create a new pod.
- If you are not given a pod definition file , you may extract the definition to a file using.
- the below command: kubectl get pod <pod-name> -o yaml > pod-definition.yaml
- Then edit the file to make the necessary changes, delete and re-create the pod.
- Use the kubectl edit pod <pod-name> command to edit pod properties.

REPLICA

Kubectl replace -f filename

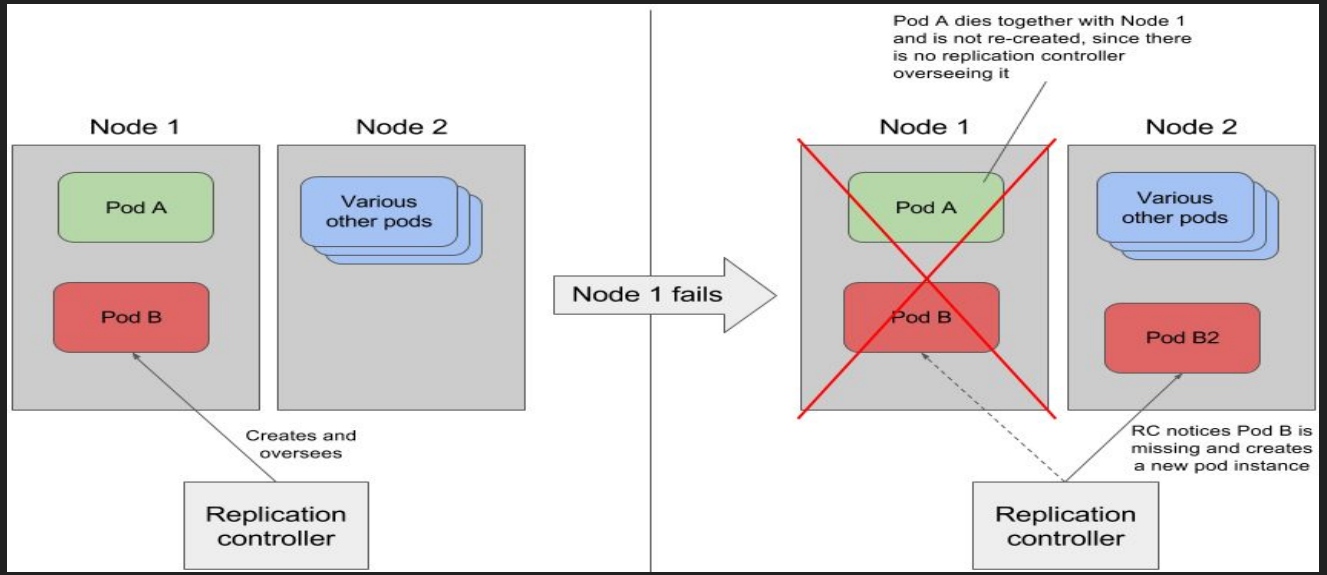
Kubectl scale --replicas=6 -f filename

Kubectl scale --replicas=6 replicaset <name>

Kubectl get po --show-labels

REPLICATION CONTROLLER

- Replication controller is responsible to up new pod if previous one fails/crash to give high availability.
- It also help us to create replica set of existing pods for scaling up.
- Replication controller is older technology which is replaced by Replica Set



REPLICATION CONTROLLER

rc-definition.yml

```yaml
apiVersion: v1
kind: ReplicationController
metadata:
  name: myapp-rc
  labels:
      app: myapp
      type: front-end
spec:
  template:
    metadata:
      name: myapp-pod
      labels:
        app: myapp
        type: front-end
    spec:
      containers:
      - name: nginx-container
        image: nginx

  replicas: 3
```

REPLICATION CONTROLLER

```
> kubectl create -f rc-definition.yml
replicationcontroller "myapp-rc" created
```

```
> kubectl get replicationcontroller
NAME       DESIRED   CURRENT   READY     AGE
myapp-rc   3         3         3         19s
```

```
> kubectl get pods
NAME              READY     STATUS    RESTARTS   AGE
myapp-rc-4lvk9    1/1       Running   0          20s
myapp-rc-mc2mf    1/1       Running   0          20s
myapp-rc-px9pz    1/1       Running   0          20s
```

REPLICA-SET

replicaset-definition.yml

```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: myapp-replicaset
  labels:
      app: myapp
      type: front-end
spec:
  template:
    metadata:
      name: myapp-pod
      labels:
        app: myapp
        type: front-end
    spec:
      containers:
      - name: nginx-container
        image: nginx

  replicas: 3
  selector:
    matchLabels:
      type: front-end
```

REPLICA-SET

```
> kubectl create -f replicaset-definition.yml
replicaset "myapp-replicaset" deleted
```

```
> kubectl get replicaset
NAME              DESIRED   CURRENT   READY     AGE
myapp-replicaset  3         3         3         19s
```

```
> kubectl get pods
NAME                    READY     STATUS    RESTARTS   AGE
myapp-replicaset-9dd19  1/1       Running   0          45s
myapp-replicaset-9jtpx  1/1       Running   0          45s
myapp-replicaset-hq84m  1/1       Running   0          45s
```

REPLICATION VS REPLICA SET:

- Replica set need apiVersion as apps/v1 while replication controller needs simple v1 as apiVersion
- Replica set needs additional selector property to select pods by their labels.
- The major difference is that the rolling-update command works with **Replication**Controllers, but won't work with a **Replica** Set. This is because **Replica** Sets are meant to be used as the backend for Deployments. Again, the pods that were created are deleted when we delete the **Replica** Set.

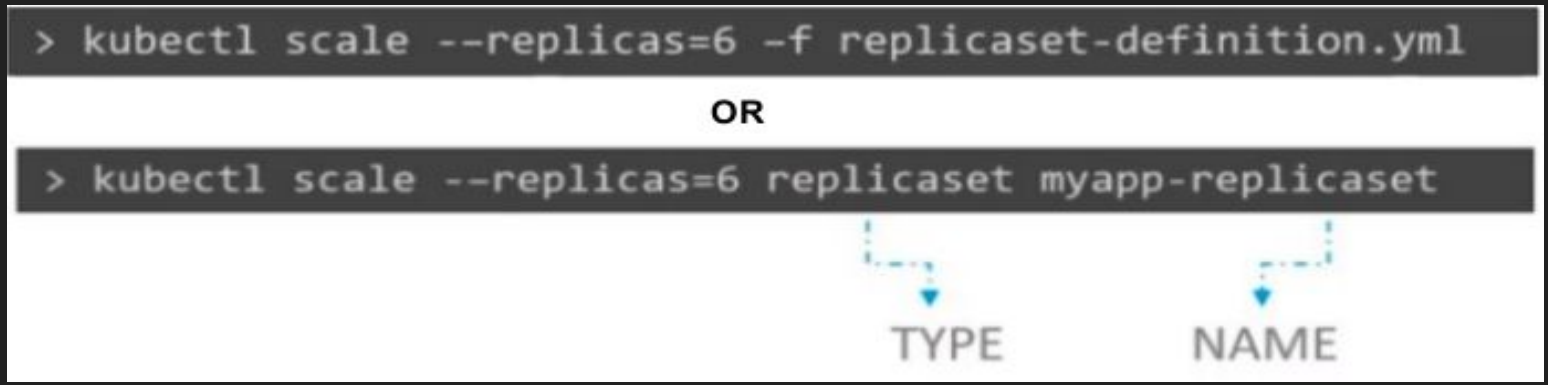
SCALING UP PODS

SCALING UP PODS:

1. You can either update number of pods in replicas of a replica set and run command:

    Kubectl replace -f replica-def.yml.

2. You can also update replicas from command line but that will not update the definition file.



DELETING REPLICA SET

This command will delete all underlying pods too.

```
> kubectl delete replicaset myapp-replicaset
```

THANKS