# Kubernetes

- What is Kubernetes?
- Advantages of Kubernetes
- Kubernetes Architecture
- Kubernetes Components
- Kubernetes  Cluster Setup
- Kubernetes Objects
- Demo

# What is Kubernetes?

- Kubernetes is an orchestration engine and open-source platform for managing containerized applications.
- Responsibilities include container deployment, scaling & descaling of containers & container load balancing.
- Actually, Kubernetes is not a replacement for Docker, But Kubernetes can be considered as a replacement for Docker Swarm, Kubernetes is significantly more complex than Swarm, and requires more work to deploy.
- Born in **Google** ,written in Go/Golang. Donated to CNCF(Cloud native computing foundation) in 2014.
- Kubernetes v1.0 was released on **July 21, 2015.**
- Current stable release v1.14.0.

# Kubernetes Features


The features of Kubernetes, are as follows:

- **Automated Scheduling:** Kubernetes provides advanced scheduler to launch container on cluster nodes based on their resource requirements and other constraints, while not sacrificing availability.

- **Self Healing Capabilities:** Kubernetes allows to replaces and reschedules containers when nodes die. It also kills containers that don't respond to user-defined health check and doesn't advertise them to clients until they are ready to serve.

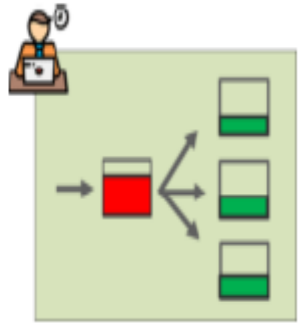
- **Automated rollouts & rollback:** Kubernetes rolls out changes to the application or its configuration while monitoring application health to ensure it doesn't kill all your instances at the same time. If something goes wrong, with Kubernetes you can rollback the change.

- **Horizontal Scaling & Load Balancing:** Kubernetes can scale up and scale down the application as per the requirements with a simple command, using a UI, or automatically based on CPU usage.

# Kubernetes Features

## 5. Service Discovery & Load balancing

With Kubernetes, there is no need to worry about networking and communication because Kubernetes will automatically assign IP addresses to containers and a single DNS name for a set of containers, that can load-balance traffic inside the cluster.

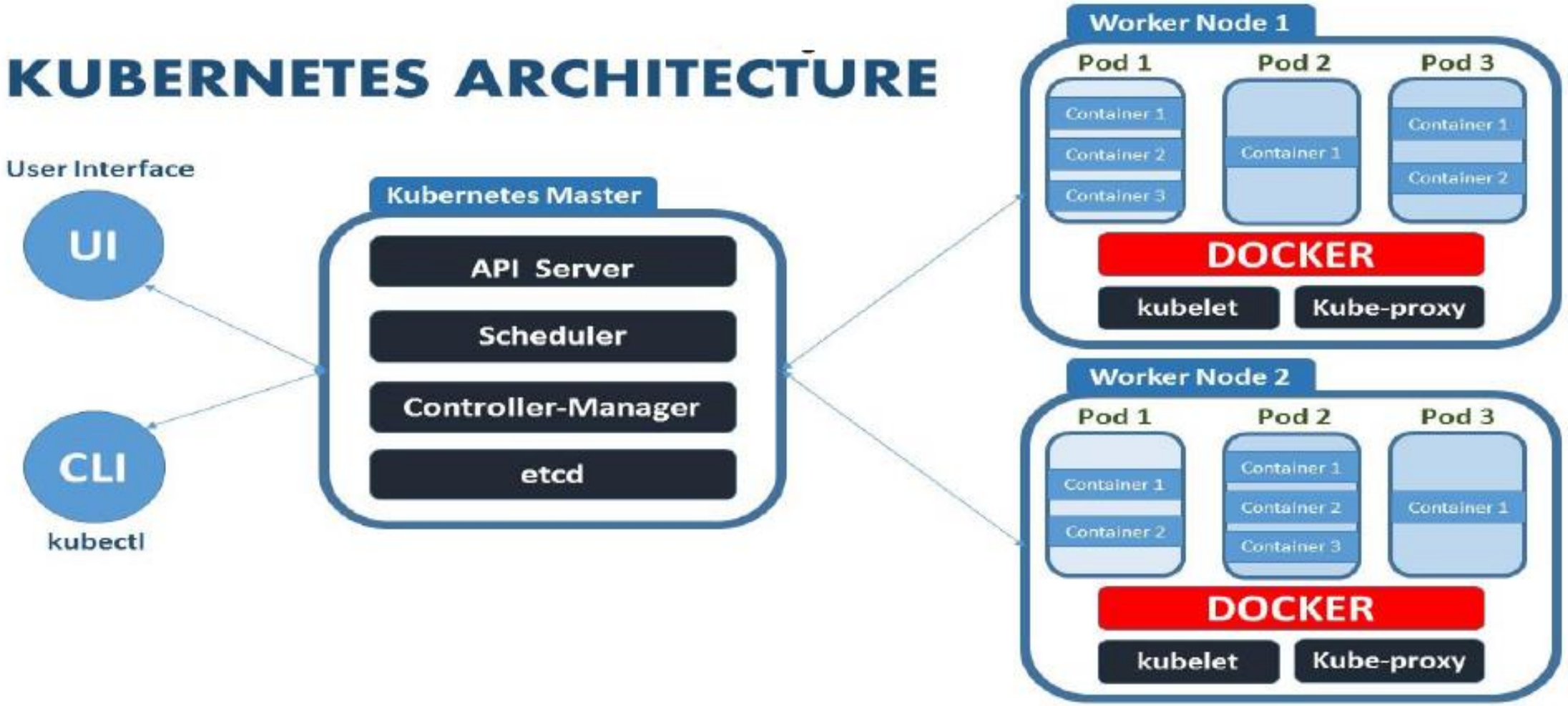


## 6. Storage Orchestration

With Kubernetes, you can mount the storage system of your choice. You can either opt for local storage, or choose a public cloud provider such as GCP or AWS, or perhaps use a shared network storage system such as NFS, iSCSI, etc.

# Kubernetes Architecture

Kubernetes implements a cluster computing background, everything works from inside a ***Kubernetes Cluster***. This cluster is hosted by one node acting as the 'master' of the cluster, and other nodes as 'nodes' which do the actual 'containerization'. Below is a diagram showing the same.



# Kubernetes Components

**Web UI (Dashboard)**

Dashboard is a web-based Kubernetes user interface. You can use Dashboard to deploy containerized applications to a Kubernetes cluster, troubleshoot your containerized application, and manage the cluster itself along with its available resources.

**Kubectl**

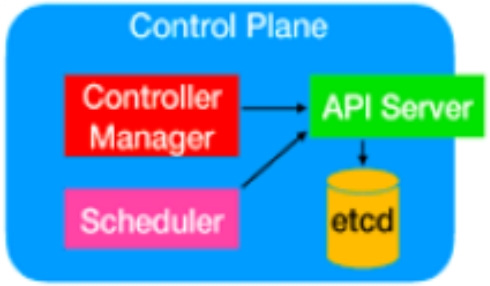
Kubectl is a command line configuration tool (CLI) for Kubernetes used to interact with master node of kubernetes. Kubectl has a config file called kubeconfig, this file has the information about server and authentication information to access the API Server.

## Master Node

The master node is responsible for the management of Kubernetes cluster. It is mainly the entry point for all administrative tasks. It handles the orchestration of the worker nodes. There can be more than one master node in the cluster to check for fault tolerance.

### Master Components

It has below components that take care of communication, scheduling and controllers.




**API Server:**

Kube API Server interacts with API, Its a frontend of the kubernetes control plane. Communication center for developers, sysadmin and other Kubernetes components

**Scheduler**

Scheduler watches the pods and assigns the pods to run on specific hosts.

# Kubernetes Components

**Controller Manager**:

Controller manager runs the controllers in background which runs different tasks in Kubernetes cluster. Performs cluster-level functions (replication, keeping track of worker nodes, handling nodes failures...).

Some of the controllers are,

1. Node controller - Its responsible for noticing and responding when nodes go down.
2. Replication controllers - It maintains the number of pods. It controls how many identical copies of a pod should be running somewhere on the cluster.
3. Endpoint controllers joins services and pods together.
4. Replicaset controllers ensure number of replication of pods running at all time.
5. Deployment controller provides declarative updates for pods and replicasets.
6. Daemonsets controller ensure all nodes run a copy of specific pods.
7. Jobs controller is the supervisor process for pods carrying out batch jobs

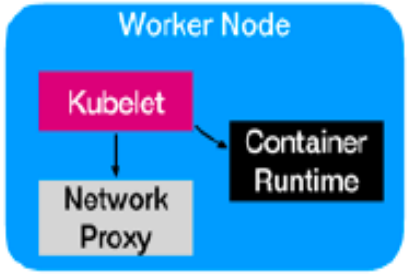
**etcd**

etcd is a simple distribute key value store. kubernetes uses etcd as its database to store all cluster datas. some of the data stored in etcd is job scheduling information, pods, state information and etc.

# Kubernetes Components

## Worker Nodes

- Worker nodes are the nodes where the application actually running in kubernetes cluster, it is also know as minion. These each worker nodes are controlled by the master node using kubelet process.
- Container Platform must be running on each worker nodes and it works together with kubelet to run the containers, This is why we use Docker engine and takes care of managing images and containers.
- We can also use other container platforms like CoreOS, Rocket.




## Node Components

### Kubelet

- Kubelet is the primary node agent runs on each nodes and reads the container manifests which ensures that containers are running and healthy.
- It makes sure that containers are running in a pod. The kubelet doesn't manage containers which were not created by Kubernetes.

### Kube-proxy

- kube-proxy enables the Kubernetes service abstraction by maintaining network rules on the host and performing connection forwarding.
- It helps us to have network proxy and load balancer for the services in a single worker node. Worker nodes can be exposed to internet via kubeproxy.

### Container Runtime

- Each node must have a container runtime, such as Docker, rkt, or another container runtime to process instructions from the master server to run containers.

# Installation

Different ways to install Kubernetes

Play-with-k8s

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

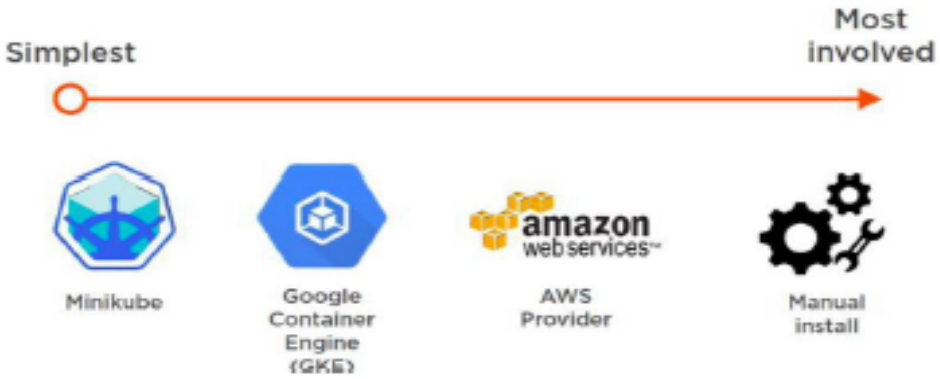
Minikube

kubeadm

Google Kubernetes Engine(GKE)

Amazon EKS

Azure Kubernetes Service (AKS)



# Install Kubernetes with kubeadm

**In Master Node & Worker/Slave Nodes:**

```
sudo hostnamectl set-hostname master-node
sudo hostnamectl set-hostname node1
sudo hostnamectl set-hostname node2
sudo apt update
sudo apt install docker.io -y
sudo systemctl enable docker
curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add
```

If Curl not installed

```
sudo apt install curl
sudo apt-add-repository "deb http://apt.kubernetes.io/ kubernetes-xenial main"
sudo apt install kubeadm -y
sudo swapoff -a
```

**In Master Node:**

```
sudo kubeadm init --pod-network-cidr=10.244.0.0/16
 mkdir -p $HOME/.kube
 sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
 sudo chown $(id -u):$(id -g) $HOME/.kube/config
sudo kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml
```

**Add Nodes to cluster**

```
kubeadm join 172.31.30.92:6443 --token sp444y.zi3g85e2zegnb489 \
   --discovery-token-ca-cert-hash sha256:d455990a683c9cd527e4b17f4bbf478be10b1ca2b84beab92e37eff0da7351fe
kubeadm token create --print-join-command
```

# Kubernetes Objects

- Kubernetes Objects are persistent entities in the Kubernetes system. Kubernetes uses these entities to represent the state of your cluster.
- A Kubernetes object is a "record of intent"–once you create the object, the Kubernetes system will constantly work to ensure that object exists.
- To work with Kubernetes objects–whether to create, modify, or delete them–you'll need to use the Kubernetes API. When you use the kubectl command-line interface, for example, the CLI makes the necessary Kubernetes API calls for you.

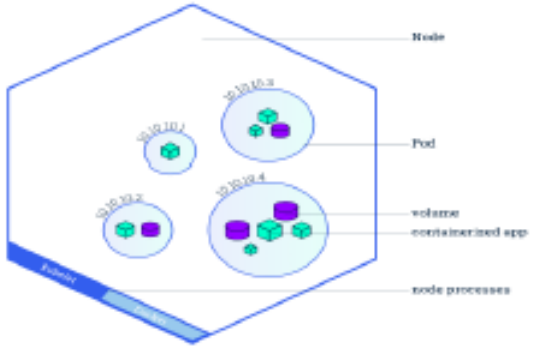
The basic Kubernetes objects include:

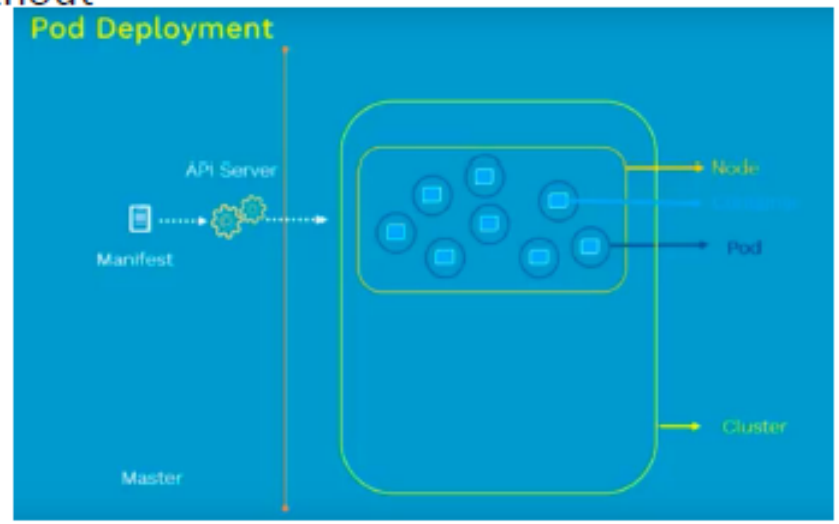
- Pod
- Replication Controller
- ReplicaSet
- DaemonSet
- Deployment
- Service
- Volume
- Job

# Kubernetes Objects

## POD


- A Pod always runs on a **Node.**
- A pod is the smallest building block or basic unit of scheduling in Kubernetes.
- In a Kubernetes cluster, a pod represents a running process.
- Inside a pod, you can have one or more containers. Those containers all share a unique network IP, storage, network and any other specification applied to the pod.
- Pods abstract network and storage away from the underlying container.
- This lets you move containers around the cluster more easily.
- Each Pod has its unique IP Address within the cluster.
- Any data saved inside the Pod will disappear without a persistent storage.




```
apiVersion: v1
kind: Pod
metadata:
  name: nginx-pod
  labels:
    app: nginx
spec:
  containers:
    - name: first-container
      image: nginx
      ports:
      - containerPort: 80
```

**Command**

kubectl apply –f pod.yml

# Kubernetes Objects

**Pod Lifecycle**

- Make a Pod reuqest to API server using a local pod defination file
- The API server saves the info for the pod in ETCD
- The scheduler finds the unscheduled pod and shedules it to node.
- Kubelet running on the node, sees the pod sheduled and fires up docker.
- Docker runs the container
- The entire lifecycle state of the pod is stored in ETCD.

**Pod Concepts**

- Pod is ephemeral(lasting for a very short time) and won't be rescheduled to a new node once it dies.
- You should not directly use Pod for deployment, Kubernetes have controllers like Replica Sets, Deployment, Deamon sets to keep pod alive.

**Kubernetes Labels and Selectors**

**Labels**

- When One thing in k8s needs to find another things in k8s, it uses labels.
- Labels are key/value pairs attached to Object
- You can make your own and apply it.
- it's like tag things in kubernetes

For e.g.

```
labels:
  app: nginx
  role: web
  env: dev
```

**Selectors**

- Selectors use the label key to find a collection of objects matched with same value
- It's like Filter, Conditions and query to your labels

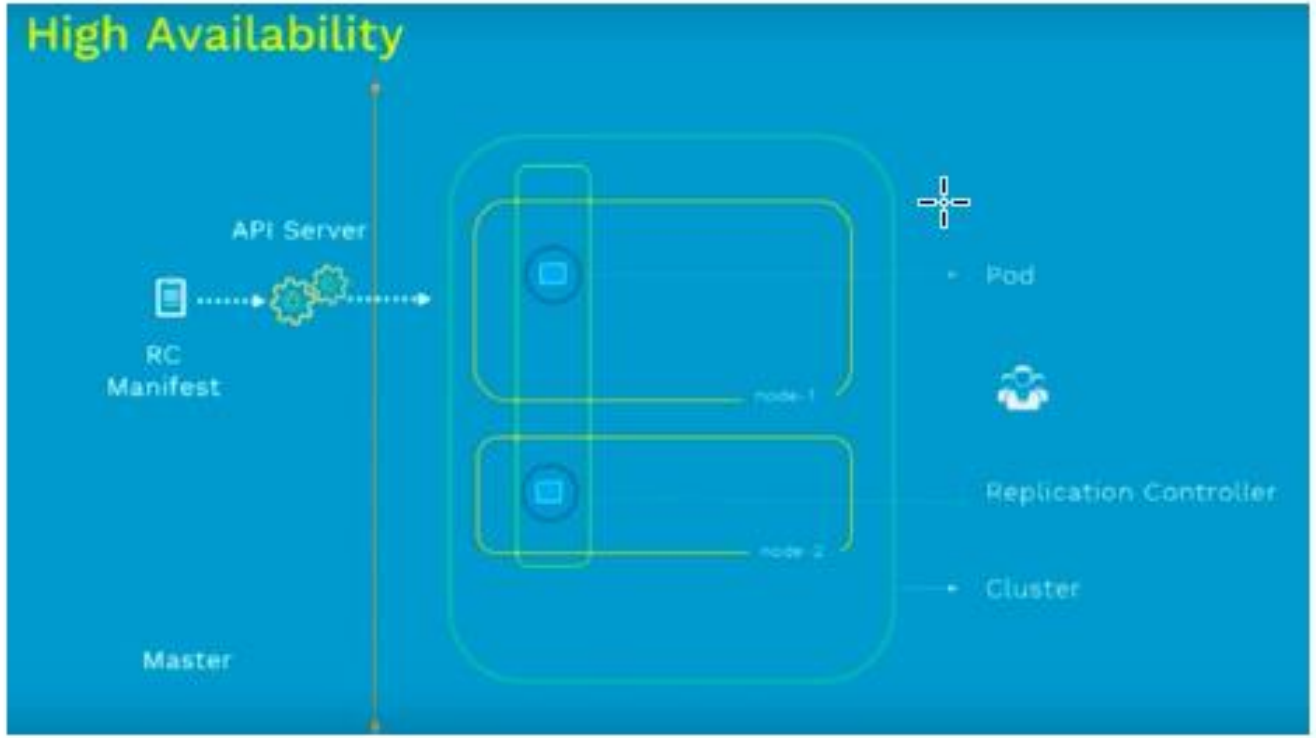
For e.g.

```
selectors:
  env = dev
  app != db
  release in (1.3,1.4)
```

- Labels and Selectors are used in many places like Services, Deployment and we will see now in Replicasets

# Replication Controller

- Replication Controller is one of the key features of Kubernetes, which is responsible for managing the pod lifecycle. It is responsible for making sure that the specified number of pod replicas are running at any point of time.
- A Replication Controller is a structure that enables you to easily create multiple pods, then make sure that that number of pods always exists. If a pod does crash, the Replication Controller replaces it.
- Replication Controllers and PODS are associated with labels.
- Creating "RC" with count of 1 ensure that a POD is always available.




```
apiVersion: v1
kind: ReplicationController
metadata:
  name: nginx-rc
spec:
  replicas: 2
  selector:
    app: nginx
  template:
    metadata:
      name: nginx
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
        - containerPort: 80
```

**Command**

kubectl apply –f rc.yml

# ReplicaSet

- ReplicaSet is the next-generation Replication Controller.
- The only difference between a *ReplicaSet* and a *Replication Controller* right now is the selector support.
- ReplicaSet supports the new set-based selector requirements as described in the labels user guide whereas a Replication Controller only supports equality-based selector requirements.

```
kubectl apply –f rs.yml
kubectl scale rs nginx-replicaset --replicas 3
```

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
 name: nginx-replicaset
spec:
 replicas: 2
 selector:
  matchLabels:
   app: nginx-rs-pod
  matchExpressions:
  - key: env
   operator: In
   values:
   - dev
 template:
  metadata:
   labels:
    app: nginx-rs-pod
    env: dev
  spec:
   containers:
   - name: nginx
    image: nginx
    ports:
    - containerPort: 80
```

# ReplicaSet

- ReplicaSet is the next-generation Replication Controller.
- The only difference between a *ReplicaSet* and a *Replication Controller* right now is the selector support.
- ReplicaSet supports the new set-based selector requirements as described in the labels user guide whereas a Replication Controller only supports equality-based selector requirements.

```
kubectl apply –f rs.yml
kubectl scale rs nginx-replicaset --replicas 3
```

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: nginx-replicaset
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx-rs-pod
    matchExpressions:
    - key: env
      operator: In
      values:
      - dev
  template:
    metadata:
      labels:
        app: nginx-rs-pod
        env: dev
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
        - containerPort: 80
```

# DaemonSet

A *DaemonSet* make sure that all or some kubernetes Nodes run a copy of a Pod.

When a new node is added to the cluster, a Pod is added to it to match the rest of the nodes and when a node is removed from the cluster, the Pod is garbage collected.

Deleting a DaemonSet will clean up the Pods it created.

```
# Daemon Set
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: nginx-ds
spec:
  selector:
    matchLabels:
      app: nginx-pod
  template:
    metadata:
      name: nginx-pod
      labels:
        app: nginx-pod
  spec:
    nodeSelector:
      name: node1
    containers:
      - name: webserver
        image: nginx
        ports:
        - containerPort: 80
```

**Command**

kubectl apply –f daemonset.yml

# Kubernetes Objects

**Deployment**

In Kubernetes, Deployment is the recommended way to deploy Pod or RS, simply because of the advance features it comes with.

Below are some of the key benefits.

- Deploy a RS.
- Updates pods (PodTemplateSpec).
- Rollback to older Deployment versions.
- Scale Deployment up or down.
- Pause and resume the Deployment.
- Use the status of the Deployment to determine state of replicas.
- Clean up older RS that you don't need anymore.

kubectl scale deployment ***[DEPLOYMENT_NAME]*** --replicas ***[NUMBER_OF_REPLICAS]***

kubectl rollout status deployment nginx

kubectl get deployment

kubectl set image deployment nginx nginx=nginx:latest --record

*kubectl get replicaset*

kubectl rollout history deployment niginx

kubectl rollout history deployment niginx **--revision**=1

kubectl rollout undo deployment nginx --to-revision=1

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
spec:
  replicas: 3
  strategy:
    type: Recreate
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
```

# Kubernetes Objects

***NodePort***–Exposes the service on each Node's IP at a static port. A ClusterIPservice, to which the NodePortservice will route, is automatically created. You'll be able to contact the NodePortservice, from outside the cluster, by using "<*NodeIP*>:<*NodePort*>".

apiVersion: v1

kind: Service

metadata:

name: my-nodeport-service

spec:

selector:

app: nginx-rs-pod

type: NodePort

ports:

-name: http

port: 80

targetPort: 80

nodePort: 30036

protocol: TCP