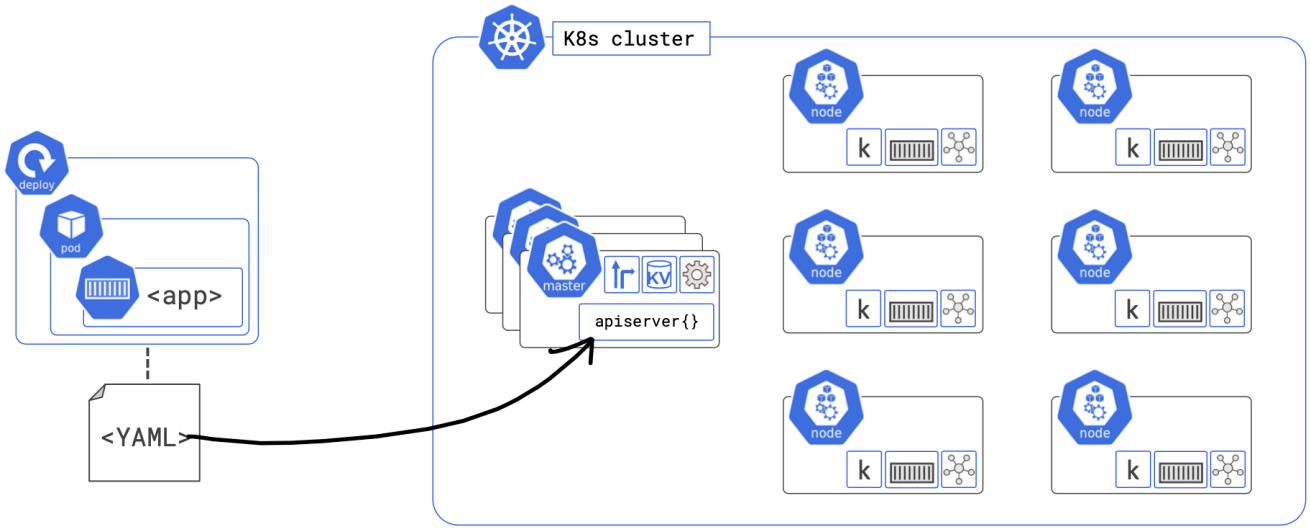
Kubernetes Introduction

Kubernetes: https://github.com/kubernetes/kubernetes

also known as K8s, is an open source system for managing containerized applications across multiple hosts. It provides basic mechanisms for the deployment, maintenance, and scaling of applications.

Containers and microservices bring a whole new set of management challenges Kubernetes to the rescue!

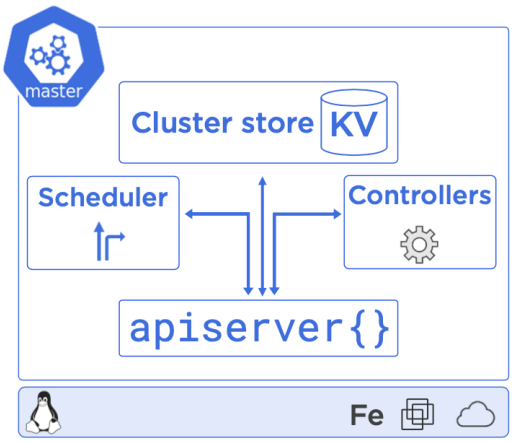
Kubernetes: Big picture overview

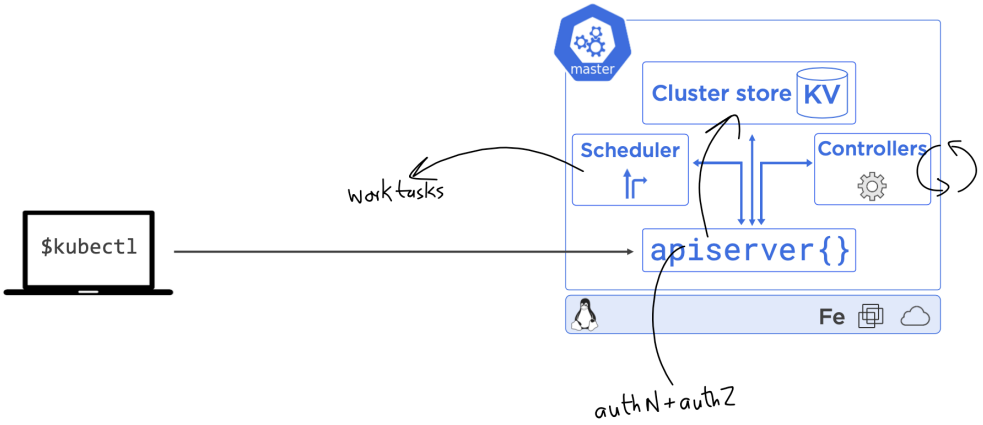


k8s cluster:

1. Node: A VM or a bare metal machine
2. masters: also known as head nodes or the control plane
3. worker node: where the workload run. The control plane manages the worker nodes.

Control plane node:





1. kube-apiserver:
    a. Front-end to the control plane
    b. Exposes Restful API
    c. Consumes JSON/YAML
2. Cluster Store:
    a. Persists cluster state and config
    b. Based on ectd
    c. KV db
3. Scheduler:
    a. Watches API server for new tasks
    b. Assign work to cluster nodes
4. controllers:
    a. watch loops
    b. Reconciles observed state with desired state
    c. Controllers of Node/ deployment.Endpoints ...

Worker node

1. kubelet
    a. Main kubernetes agent
    b. Registers node with cluster
    c. Watches API server for work tasks
    d. Executes Pods
    e. Reports back to masters
2. Container runtime
    a. Pluggable: Container runtime interface (CRI)
    b. Can be docker
3. Kube-proxy
    a. Networking component
    b. Pod IP address

1. Kubectl: command interface for API-server, kubectl knows how to talk to the api server sitting as the front-end of control node via client-server communication
2. kubeadm: setup and manage the cluster. You don't need it anymore once you are done with the cluster setup for the node
3. kubelet: a deamon process on worker node(control plane node tells worker node what to do via this)

Cloud providers built up for orchestration based upon kubernetes:

1. AWS(Amazon Elastic kubernetes Service)
2. Azure(AKS)
3. Google (Google kubernetes Engine)

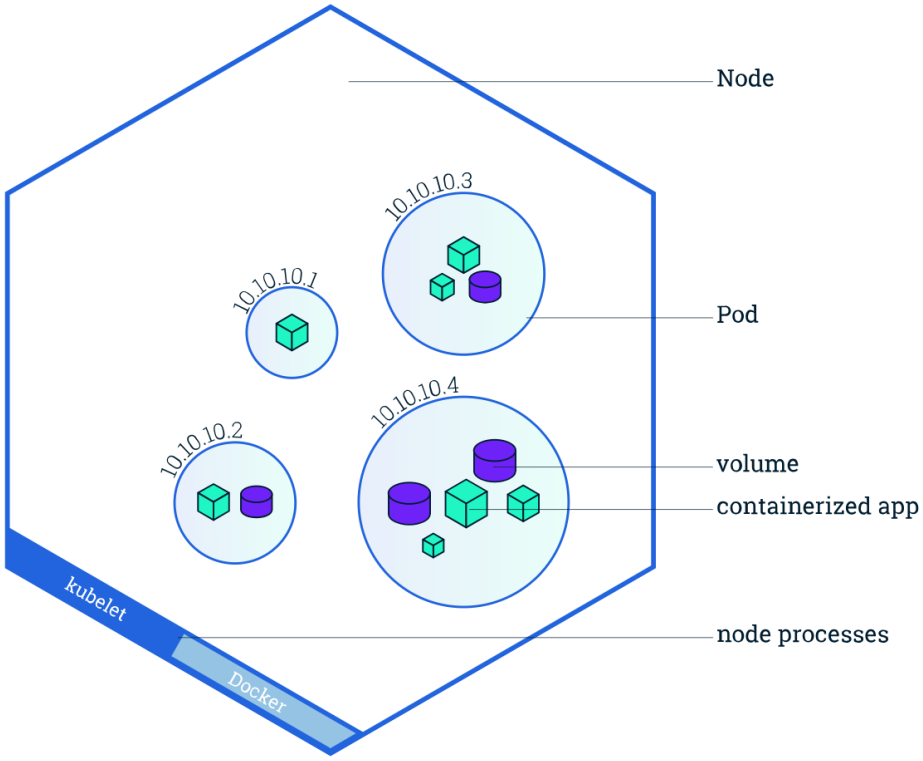
Example of two node, one is control-plane, the left one is worker node. Use kubeadm to set up control-plane, and join the worker node to the cluster

```
root@ubuntu180402:~# kubectl get nodes
NAME           STATUS   ROLES           AGE     VERSION
kube-slave     Ready    <none>          3d12h   v1.27.3
ubuntu180402   Ready    control-plane   3d12h   v1.27.2

kubectl describe node kube-slave


root@ubuntu180402:~# kubectl get nodes -o wide
NAME           STATUS   ROLES           AGE     VERSION   INTERNAL-IP    EXTERNAL-IP   OS-IMAGE
KERNEL-VERSION       CONTAINER-RUNTIME
kube-slave     Ready    <none>          3d12h   v1.27.3   10.108.60.82   <none>        Ubuntu 18.04.6 LTS
4.15.0-212-generic   containerd://1.6.21
ubuntu180402   Ready    control-plane   3d12h   v1.27.2   10.108.4.152   <none>        Ubuntu 18.04.6 LTS
4.15.0-212-generic   containerd://1.6.12
```

Kubernetes: Pods

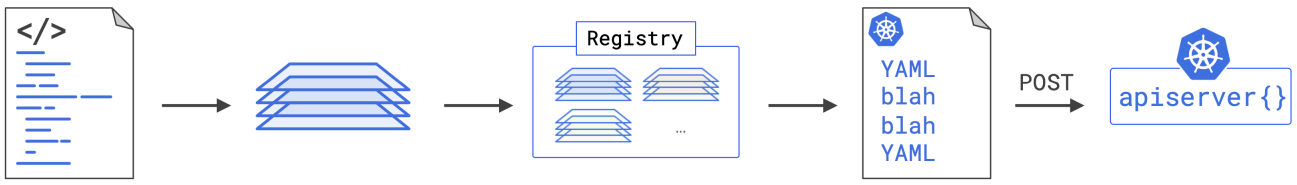


1. atomic scheduling unit
2. A pod is built in a node, a node can have many pods
3. A pod is a group of containers
4. All these containers share the network
5. Containers in the same pod are guaranteed to be scheduled on the same pods
6. All containers in the pod must be healthy in order to have the pod healthy. Otherwise k8s will consider that pod unhealthy and try to take action

```
# To create a pod
kubectl run nginx --image nginx
kubectl get pods
kubectl describe pod nginx

kubectl exec -it <pod-name> -- /bin/bash
# Inside the container
echo "Hello" > /usr/share/nginx/html/index.html
# Outside the container
curl <pod-ip> # should see Hello instead.
```

Kubernetes: Yaml



Declarative mode: Describe what you want (desired state) in a manifest file


```
apiVersion: v1
kind: Pod
metadata:
  name: nginx-via-yaml
spec:
  containers: #network is being shared (in the same namespace but processes are isolated)
  - name: nginx
    image: nginx:alpine
  - name: curl
    image: curlimages/curl
    stdin: true # keep stdin
    tty: true  # keep terminal (interact with shell)
    command: ["/bin/sh"]
```

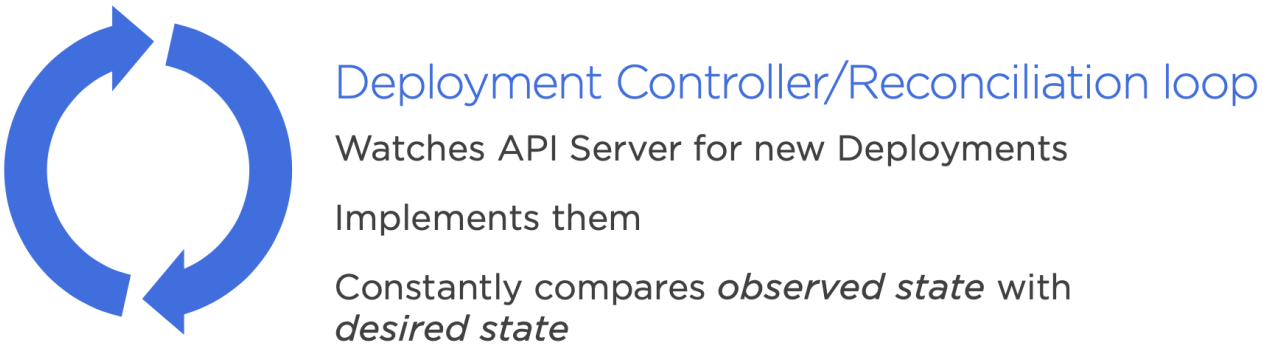
```
kubectl get pod xx -o yaml
kubectl apply -f my-first-pod.yaml
```

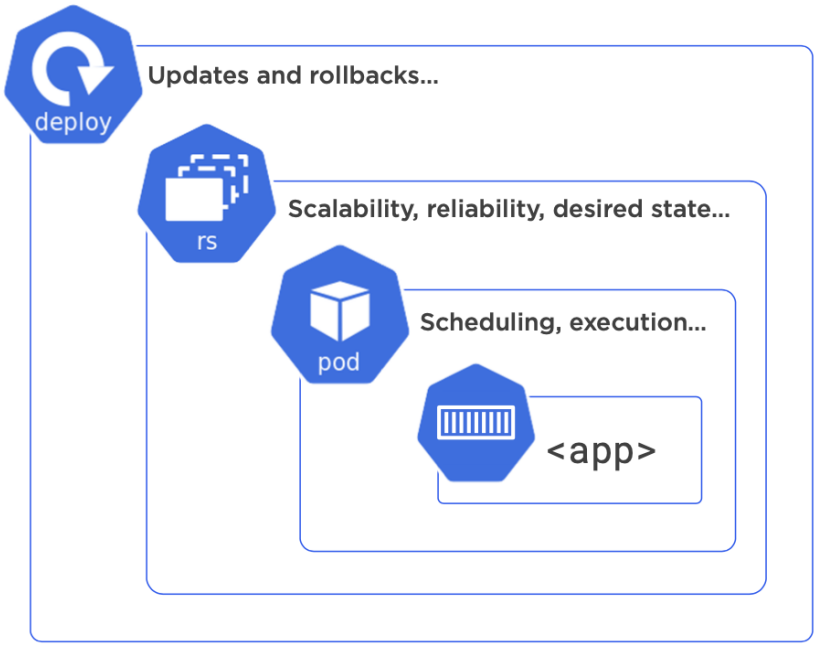
Kubernetes: Deployment

Deployments provide reliability and scalability to our application. Deployment makes sure that desired number of pods, which is specified declaratively in the deployment file are always up and running. If a pod fails to run, deployment will remove that pod and replace it with a new one.

self-healing and scaling

Rolling updates and rollbacks





Default deployment strategies in K8S is Rolling update:

- We can avoid down time since old version of application is still receiving traffic.
- Only a few pod is updated at a time.

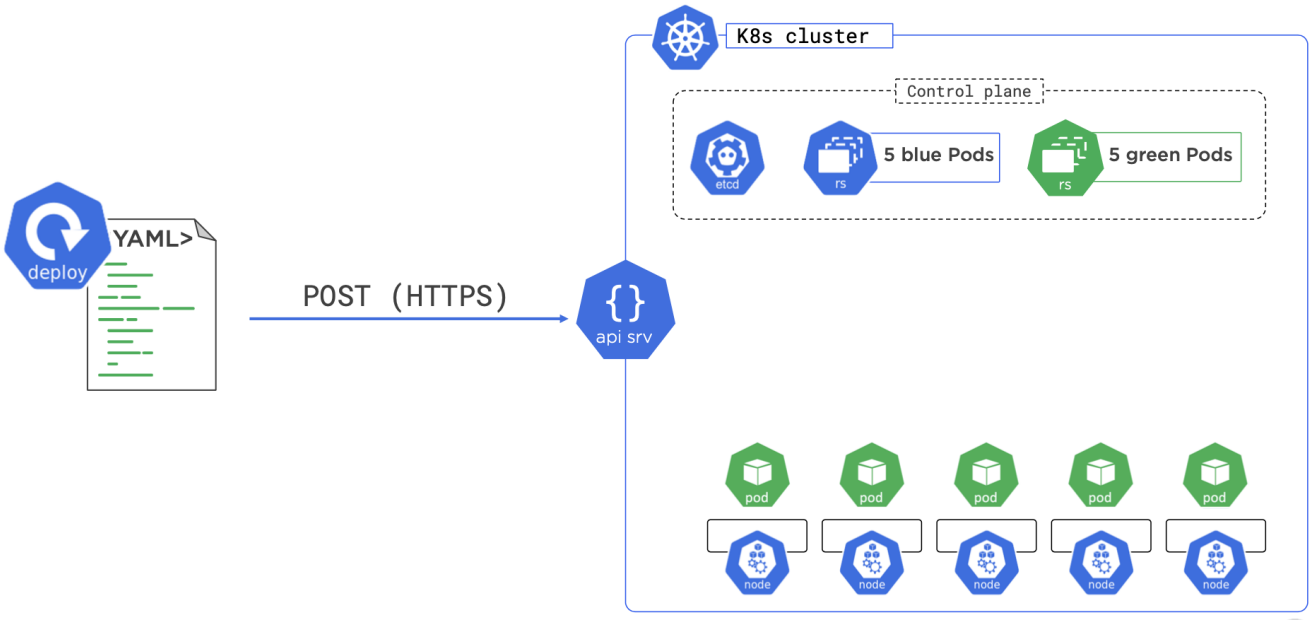
When you do an update, a new replica set is created. But the old replica set is kept for roll back purposes.

For a new deployment:

1. Kubectl tells the API Server in control node that you want to update the deployment with the given spec file
2. API Server saves the updated spec to etcd (db to store cluster data).
3. Deployment Controller reads the spec and see that you want three replicas, so it go ahead and create a replicaset object and set the replicas field to 3
4. Replication Controller is monitoring this, it sees that the current state does not match the desired state in replicaset object (We want 3 but there is 0 pod), it issues a command to create 3 pods. It won't create the pods itself
5. These pods haven't been assigned with any nodes. The scheduler sees this and says let me assign the node for you guys.
6. Then the scheduler interacts with the kubelet daemon on each node, telling them how many pods they need to create and the kubelet daemon on each node will actually create those pods

If the deployment already exists:

1. Deployment controller sees the image changes, it creates a new replicaset object (let's call it r-new), it then set the replicas field in r-new to be 1. Let's call the old replicaset object we mentioned above r-old
2. Replication controller detects this, so it issues a command to create 1 stable pod replica
3. Deployment controller sees that r-old is 3 and r-new is 1, both are stable. It sets the replica field in r-old to be 2 (reduce by 1) and r-new to be 2 (increase by 1)
4. Replication controller detects this, and go ahead issues command to reach the desired state
5. The above process repeats until r-old goes to 0 and r-new becomes 3




```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx # name of the resources you created
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx-app
  template:
    metadata:
      labels:
        app: nginx-app
    spec:
      containers:
        - name: nginx
          image: nginx:1.18
```

```
kubectl apply -f my-first-deployment.yaml
kubectl get deploy
kubectl get rs
kubectl get pods

kubelet describe deploy # Observe the "describe" output of deploy and rs and notice what happens when you
change the image and apply
You can also check the rollout status of your deployment: kubectl rollout status --help
kubectl rollout status deploy/nginx
# Change the image to nginx:1.18 and apply kubectl apply -f my-first-deployment.yaml
# Immediately run the following command to see the sequence of rollout events kubectl rollout status deploy
/nginx

# The revisions
kubectl rollout history deploy/nginx
# A new RS (replicaset) is created for every new rollout, old ones are retained in case you want to rollback
kubectl get rs
# You can rollback using this
kubectl rollout undo --help
kubectl rollout undo deploy/nginx --to-revision=1
kubectl rollout status deploy/nginx
kubectl get rs
# A new revision is created even when we roll back
kubectl rollout history deploy/nginx
# What configurations you can tweak
# Observe, strategy, limit, progress deadline etc
kubectl get deploy nginx -o yaml # this output the complete spec kubernetes reads to create the deployment,
including the spec you don't put in the yaml file (kubernetes will use default value). You can add/tweak the
corresponding fields to control granularity on how you want kubernetes to rollout your deployments (could
certainly let k8 use approach 1 and approach 2 if needed)
```

Kubernetes: Service (To be cond)