

Application definition

Kubernetes: An application can be deployed in Kubernetes utilizing a combination of services (or microservices), deployments, and pods.

Docker Swarm: The applications can be deployed as micro-services or services in a swarm cluster in Docker Swarm. YAML(YAML Ain’t Markup Language) files can be utilized to identify multi-container. Moreover, Docker compose can install the application.

Networking

Kubernetes: The networking model is a flat network, allowing all pods to interact with one another. The network policies specify how the pods interact with each other. The flat network is implemented typically as an overlay. The model needs two CIDRs: one for the services and the other from which pods acquire an IP address.

Docker Swarm: The Node joining a swarm cluster generates an overlay network for services that span every host in the docker swarm and a host-only docker bridge network for containers. The users have a choice to encrypt container data traffic while creating of an overlay network by on their own in docker swarm.

Scalability

Kubernetes: For distributed systems, Kubernetes is more of an all-in-one framework. It is a complex system because it provides strong guarantees about the cluster state and a unified set of APIs. This slows down container scaling and deployment.

Docker Swarm: Docker Swarm, when compared to Kubernetes, can deploy container much faster and this allows faster reaction times to scale on demand.

High Availability

Kubernetes: All the pods in kubernetes are distributed among nodes and this offers high availability by tolerating the failure of application. Load balancing services in kubernetes detect unhealthy pods and get rid of them. So, this supports high availability.

Docker Swarm: As the services can be replicated in Swarm nodes, Docker Swarm also offers high availability. The Swarm manager nodes in Docker Swarm are responsible for the entire cluster and handle the worker nodes’ resources.

Container Setup

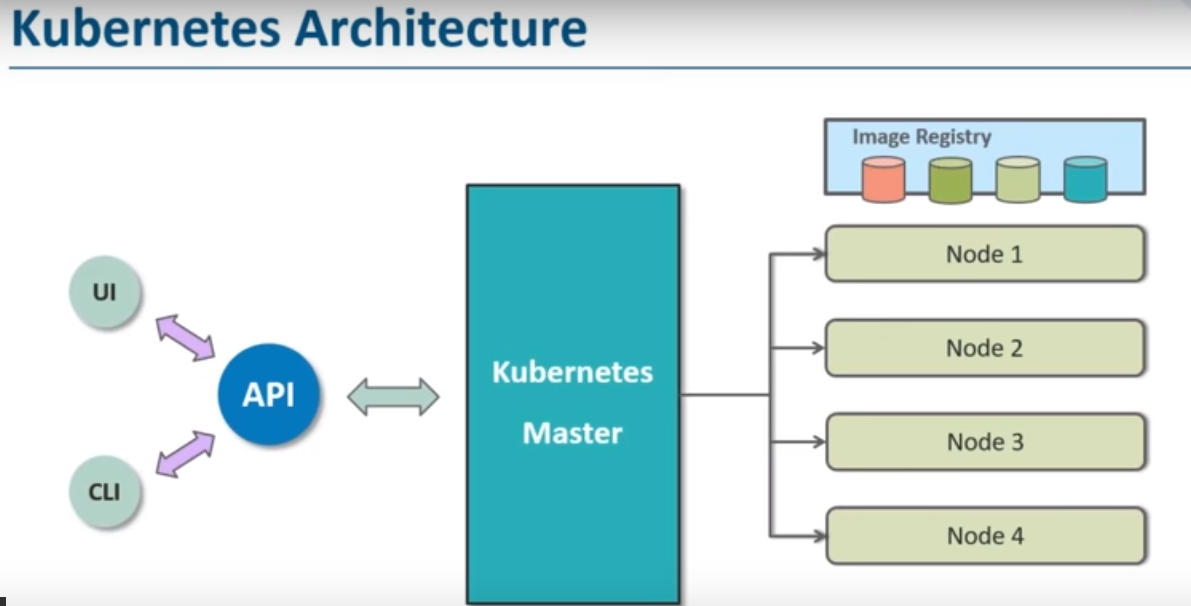
Kubernetes: Kubernetes utilizes its own YAML, API, and client definitions and each of these differ from that of standard docker equivalents. That is to say, you cannot utilize Docker Compose nor Docker CLI to define containers. While switching platforms, YAML definitions and commands need to be rewritten.

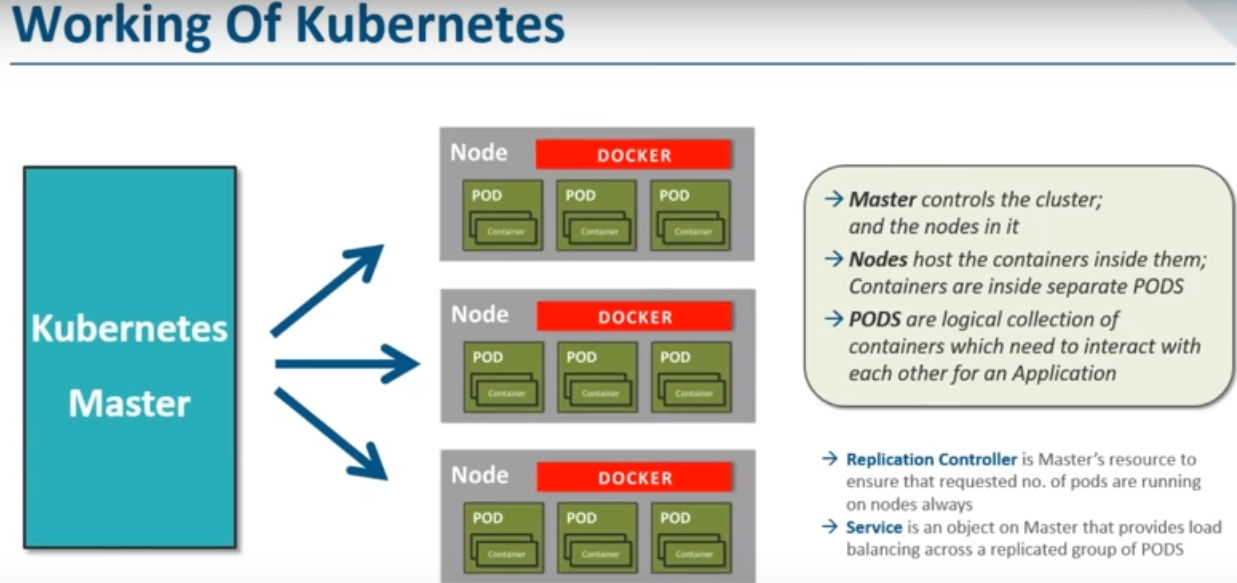
Docker Swarm: The Docker Swarm API doesn’t entirely encompass all of Docker’s commands but offers much of the familiar functionality from Docker. It supports most of the tools that run with Docker. Nevertheless, if Docker API is deficient of a particular operation, there doesn’t exist an easy way around it utilizing Swarm.

Load Balancing

Kubernetes: Pods are exposed via service, which can be utilized as a load balancer within the cluster. Generally, an ingress is utilized for load balancing.

Docker Swarm: Swarm mode consists of a DNS element that can be utilized for distributing incoming requests to a service name. Services can be assigned automatically or can run on ports specified by the user."





**Installation of Minikube on EC2**

### 1. Run a public EC2 Server with the following setup

|  |  |
| --- | --- |
| **AMI** | Ubuntu Server 18.04 LTS (HVM), SSD Volume Type |
| **Instance Type** | t2.micro |
| **Storage** | 8 GB (gp2),2vcpus |
| **Tags** | – Key: Name – Value: Minikube |
| **Security Group** | Name: Minikube Security Group – SSH, 0.0.0.0/0 Later we will be editing this. |
| **Key Pair** | Create your own keypair. You will need this to SSH to your EC2 Instance |

2. SSH into your created EC2 Instance using your keypair.

3. Install kubectl

#curl -LO https://storage.googleapis.com/kubernetes-release/release/$(curl -s https://storage.googleapis.com/kubernetes-release/release/stable.txt)/bin/linux/amd64/kubectl

chmod +x ./kubectl

#sudo mv ./kubectl /usr/local/bin/kubectl

4. Install Docker

#sudo apt-get update && \

sudo apt-get install docker.io –y

5. Install Minikube

#curl -Lo minikube <https://storage.googleapis.com/minikube/releases/latest/minikube-linux->amd64 && chmod +x minikube && sudo mv minikube /usr/local/bin/

6. Check Minikube Version

#minikube version

## Running Minikube on EC2 Ubuntu

## 1.Start Minikube

## #minikube start --vm-driver=none

2. Check the status of Minikube

#minikube status

**Running first Pod on cluster**

#kubectl run hello-pod --image=gcr.io/google\_containers/echoserver:1.4 --port=8080

#kubectl get pods

#kubectl describe pod hello-pod

### Expose the container ports

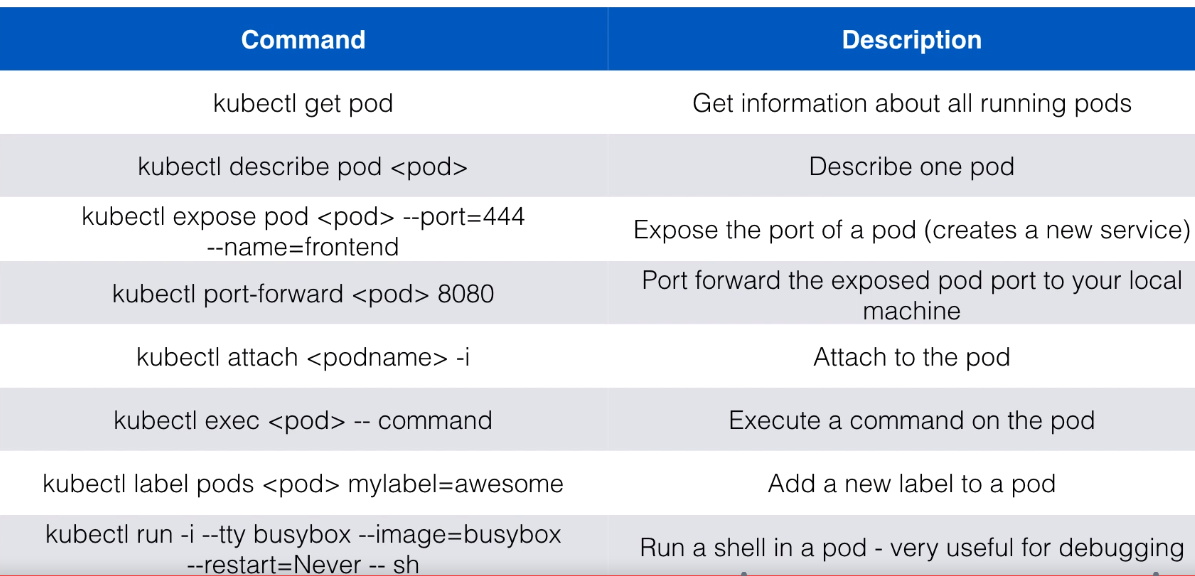
#kubectl expose pod hello-minikube --type=NodePort

### Find where port 8080 in container exposed in EC2 Instance port

#kubectl get services

**Access the our container via the EC2**





**Running First-app**

#vi hello.yml

apiVersion: v1

kind: Pod

metadata:

name: shell-demo

spec:

volumes:

- name: shared-data

emptyDir: {}

containers:

- name: nginx

image: nginx

volumeMounts:

- name: shared-data

mountPath: /usr/share/nginx/html

hostNetwork: **true**

dnsPolicy: Default

#vi helloworld.yml

apiVersion: v1

kind: Pod

metadata:

name: nodehelloworld.example.com

labels:

app: helloworld

spec:

containers:

- name: k8s-demo

image: **Krishna@capitalinfo**/k8s-demo

ports:

- name: nodejs-port

containerPort: 3000

#kubectl create –f path-to-file.yml

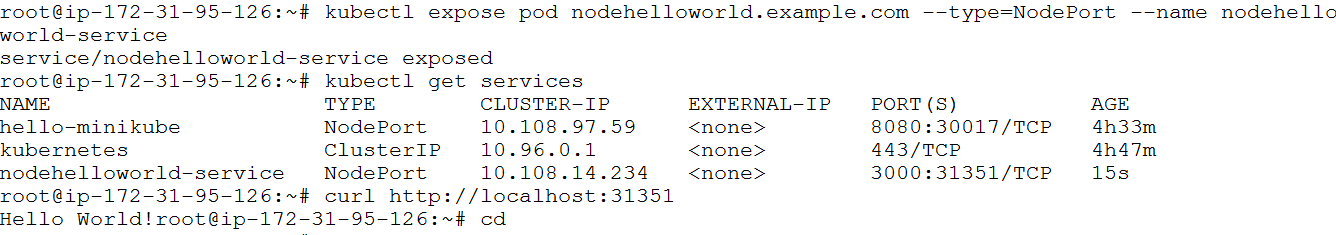
#kubectl get pods

#kubectl describe pods nodehelloworld.example.com

# kubectl expose pod nodehelloworld.example.com --type=NodePort --port =444 –name=nodehelloworld-service

#kubetctl get services

#curl http://localhost:31351

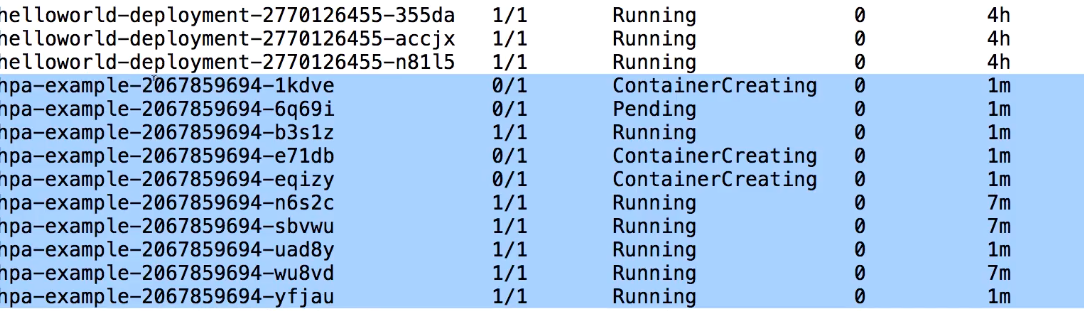


**Login into pod**

# kubectl exec -i --tty shell-demo -- /bin/bash

Run below command in shell after logging into pod

#while true; do wget -q -O- http://hpa-example.default.svc.cluster.local:31001; done



**Launch multiple containers in single pod**

apiVersion: v1

kind: Pod

metadata:

name: shell-demo6

labels:

app: shelldemolabel6

spec:

volumes:

- name: shared-data

emptyDir: {}

containers:

- name: tomcat

image: tomcat

ports:

- name: nodejs-port8

containerPort: 3004

volumeMounts:

- name: shared-data

mountPath: /usr/share/nginx/html

- name: database

image: mongo

ports:

- name: dbport

containerPort: 27018

hostNetwork: true

dnsPolicy: Default

**Port, Target port, Nodeport:**

* Port exposes the Kubernetes service on the specified port within the cluster. Other pods within the cluster can communicate with this server on the specified port.
* TargetPort is the port on which the service will send requests to, that your pod will be listening on. Your application in the container will need to be listening on this port also.
* NodePort exposes a service externally to the cluster by means of the target nodes IP address and the NodePort. NodePort is the default setting if the port field is not specified.

#vi createservice.yml

apiVersion: v1

kind: Service

metadata:

name: hello-world

spec:

type: NodePort

selector:

app: hello-world

ports:

- protocol: TCP

port: 8080

targetPort: 80

nodePort: 30036

#vi createpod.yml

apiVersion: v1

kind: Pod

metadata:

name: nginx

labels:

app: hello-world

spec:

containers:

- name: nginx

image: nginx

ports:

- containerPort: 80

Test(curl) pod:

$ kubectl run -i --tty ubuntu --image=ubuntu --restart=Never -- sh

Then.. $ curl hello-world:8080

**Load Balanacer:**

**apiVersion**: v1

**kind**: Service

**metadata**:

**name**: example-service

**spec**:

**selector**:

**app**: example

**ports**:

- **port**: 8765

**targetPort**: 9376

**type**: LoadBalancer

OR

kubectl expose rc example --port=8765 --target-port=9376 **\**

--name=example-service --type=LoadBalancer

**Nodeselector**

Decides what pod to run on what node.

Steps:

1.Add a label to node

#kubectl label nodes <node-name> <label-name>

Ex: #kubectl label nodes node-name env=dev

2. add nodeselector in the manifest

apiVersion: v1

kind: Pod

metadata:

name: nginx

labels:

app: hello-world

spec:

containers:

- name: nginx

image: nginx

ports:

- containerPort: 80

nodeSelector:

env: dev

3.remove node from label

#kubectl label nodes <node-name> <label-name>-

Ex: # kubectl label nodes node-name env-

(OR)

**..we can also schedule a pod to a specific node using nodename**

apiVersion: v1

kind: Pod

metadata:

name: nginx-pod-node-test

labels:

app: nginx-label-test

spec:

nodeName: ip-172-31-6-57

containers:

- name: nginx-cont

image: nginx

ports:

- containerPort: 80

apiVersion: v1

kind: Pod

metadata:

name: nginx

labels:

app: hello-world

spec:

nodeName: Node-Name

containers:

- name: nginx

image: nginx

ports:

- containerPort: 80

**Replication controller:**

apiVersion: v1

kind: ReplicationController

metadata:

name: tomcat-replicationcontroller

spec:

replicas: 3

template:

metadata:

name: tomcat-replicationcontroller

labels:

app: App

component: neo4j

spec:

containers:

- name: tomcat

image: tomcat

ports:

- name: nodejs-port

containerPort: 3020

**Deployment**

**apiVersion**: apps/v1

**kind**: Deployment

**metadata**:

**name**: nginx-deployment

**labels**:

**app**: nginx

**spec**:

**replicas**: 3

**selector**:

**matchLabels**:

**app**: nginx

**template**:

**metadata**:

**labels**:

**app**: nginx

**spec**:

**containers**:

- **name**: nginx

**image**: nginx:1.14.2

**ports**:

- **containerPort**: 80

kubectl apply -f

kubectl get deployments

kubectl rollout status deployment/nginx-deployment

kubectl get deployments

kubectl set image deployment/nginx-deployment nginx=nginx:1.16.1 --record

kubectl edit deployment.v1.apps/nginx-deployment

kubectl rollout status deployment/nginx-deployment

Horizontal Pod Autoscale..HPA

**apiVersion**: apps/v1

**kind**: Deployment

**metadata**:

**name**: php-apache

**spec**:

**selector**:

**matchLabels**:

**run**: php-apache

**replicas**: 1

**template**:

**metadata**:

**labels**:

**run**: php-apache

**spec**:

**containers**:

- **name**: php-apache

**image**: k8s.gcr.io/hpa-example

**ports**:

- **containerPort**: 80

**resources**:

**limits**:

**cpu**: 500m

**requests**:

**cpu**: 200m

**---**

**apiVersion**: v1

**kind**: Service

**metadata**:

**name**: php-apache

**labels**:

**run**: php-apache

**spec**:

**ports**:

- **port**: 80

**selector**:

**run**: php-apache

kubectl autoscale deployment php-apache --cpu-percent=50 --min=1 --max=10

kubectl get hpa

kubectl run -i --tty load-generator --rm --image=busybox --restart=Never -- /bin/sh -c "while sleep 0.01; do wget -q -O- http://php-apache; done"