Installing Kubernetes Using Kubeadm

===========================================

#

Topic: Setting up Control-plane/Master node

#############################################

1)

# Install container runtime - Docker

Follow this source.

Source:

https://docs.docker.com/install/linux/docker-ce/ubuntu/

2)

# Install Kubeadm

Follow this source or below commands.

Source:

https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/install-kubeadm/

Commands:

- sudo apt-get update

- sudo apt-get install -y apt-transport-https curl

- Add repository keys to download tools:

curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -

- Add repository:

cat <<EOF | sudo tee /etc/apt/sources.list.d/kubernetes.list

deb https://apt.kubernetes.io/ kubernetes-xenial main

EOF

- sudo apt-get update

- sudo apt-get install -y kubelet kubeadm kubectl

- sudo apt-mark hold kubelet kubeadm kubectl

#3) Initialize a new Kubernetes cluster

$ sudo kubeadm init

NOTE: Save this output as it's required to add worker nodes.

4)

# As per the instruction from 'kubeadm init' command output, To make kubectl work for your non-root user, run these commands.

mkdir -p $HOME/.kube

sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

sudo chown $(id -u):$(id -g) $HOME/.kube/config

#5) Verify if cluster is initialized succussfuly

$ kubectl get nodes

O/P:

NAME STATUS ROLES AGE VERSION

node1 NotReady master 2m43s v1.12.1

#9) Run the following kubectl command to find the reason why the cluster STATUS is showing as NotReady.

- This command shows all Pods in all namespaces - this includes system Pods in the system (kube-system) namespace.

- As we can see, none of the coredns Pods are running

- This is preventing the cluster from entering the Ready state, and is happening because we haven’t created the Pod network yet.

O/P:

$ kubectl get pods --all-namespaces

NAMESPACE NAME READY STATUS RESTARTS AGE

kube-system coredns-...vt 0/1 ContainerCreating 0 8m33s

kube-system coredns-...xw 0/1 ContainerCreating 0 8m33s

kube-system etcd... 1/1 Running 0 7m46s

kube-system kube-api... 1/1 Running 0 7m36s

#7) Create Pod Network. You must install a pod network add-on so that your pods can communicate with each other. (As per kubeadm init output)

Source: https://www.weave.works/docs/net/latest/kubernetes/kube-addon/#install

$ kubectl apply -f "https://cloud.weave.works/k8s/net?k8s-version=$(kubectl version | base64 | tr -d '\n')"

#8) Check if the status of Master is changed from 'NotReady' to 'Ready'

$ kubectl get nodes

NAME STATUS ROLES AGE VERSION

node1 Ready master 3m51s v1.12.1

GREAT - the cluster is ready and all dns system pods are now working. Cluster is ready now.

Now that the cluster is up and running, it’s time to add some worker-nodes.

#

Topic: Worker Node Setup & Joining to the cluster:

#############################################

1

# Create a worker node machine in GCP / AWS cloud platform.

2

# Install kubeadm

https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/install-kubeadm/

3

# Install container runtime

https://docs.docker.com/install/linux/docker-ce/ubuntu/

4

# To bootstrap a Kubernetes worker node and join it to the cluster run below command from $kubeadm init output.

Note: bootstrap: It automatically installs kubectl and kubelet.

Note: Below Bootstrap token will be different for your Control-plane/Master node. Use the one which you have copied at step:3.

kubeadm join 10.128.0.18:6443 --token 9ril81.t4k4sqh1ionqv1om \

--discovery-token-ca-cert-hash sha256:de57d9e08877db501a8b503db3ee91596f8f5657878c3087bc0343ece7df3eb2

# Verify node Join (Run below in Control-plane node)

$ kubectl get nodes

NAME STATUS ROLES AGE VERSION

control-plane Ready master 26m v1.16.3

worker-node1 Ready <none> 3m18s v1.16.3

$ kubectl get nodes -o wide

--> this will display IP, OS, Kernel and more details about all Nodes

#

Project-1 [Nginx]

#############################################

Deploying/Creating a pod

#############################################

#

1.) Create Pod manifest file

$ mkdir nginx

$ vim pod.yaml

pod.yaml

=========

apiVersion: v1

kind: Pod

metadata:

name: nginx-pod

labels:

env: prod

version: v1.2.3

spec:

containers:

- name: nginx-container

image: nginx

ports:

- containerPort: 80

pod.yaml - Manifest file description:

----------------------

- Straight away we can see four top-level resources.

• .apiVersion

• .kind

• .metadata

• .spec

--> .apiVersion:

- Tells API Server about what version of Yaml is used to create the object (Pod object in this case)

- Pods are currently in v1 API group

--> .kind:

- Tells us the kind of object being deployed. In this case we are creating POD object.

- It tells control plane what type of object is being defined.

--> .metadata:

- this section again has two sub-sections i.e name & labels

- You can name the Pod using "name" key.

- Using labels, we can identify a particular pod.

--> .spec:

- This is where we specify details about the containers that will run in the Pod.

- In this section we specify container name, image, ports ..etc.

#

2.) Creating a Pod

- Check if all Nodes are ready before creating a Pod

$ kubectl get nodes

- This POSTs the manifest file to API server and deploy/create a Pod from it

$ kubectl apply -f pod.yml

Note: Your Pod has been scheduled to a healthy node in the cluster and

is being monitored by the local kubelet process on the node.

# Introspecting Running Pods

- Get IP and worker node of the Pod

$ kubectl get pod -o wide

- Launch nginx server application running in the Pod from Controle-plane node

$ curl http://10.44.0.1:80

$ curl http://POD-IP:Server-Port

- You can also login into the Pod container to get more information.

$ kubectl exec -it nginx-pod /bin/bash

Note: Let's add some code and launch our nginx application

- $ echo "Gamut Gurus Technologies" > /usr/share/nginx/html/index.html

- Launch nginx application

$ curl http://10.44.0.1:80

- Login into a specific container in case you have multi container Pod

using --container or -c option.

$ kubectl exec -it nginx-pod --container nginx-container /bin/bash

#

3.) Deleting a Pod

$ kubectl get pods

$ kubectl delete pods nginx-pod

$ kubectl delete -f pod.yml

NOTE:

kubelet takes the PodSpec and is responsible for pulling all images and starting all containers in the Pod.

What Next?

- If a Pod fails, it is not automatically rescheduled. Because of this, we usually deploy

them via higher-level object such as Deployments.

- This adds things like "scalability" (scale-up/down), "self-healing", "rolling updates" and "roll backs" and makes Kubernetes so powerful.

Misc. CMDs:

- Get full copy of the Pod manifest from cluster store. desired state is (.spec) and oberved state will be under (.status)

$ kubectl get pod -o yaml

- Check if Pod is created

$ kubectl get pods

$ kubectl get pods --watch (monitor the status continuously)

- Another great Kubernetes introspection command. Provides Pods(object's) lifecycle events.

$ kubectl describe pod nginx-pod

--wk-end-2PM-batch--

#

Project-2 [Nginx]

#############################################

Creating Deployments & Services

#############################################

- Pods don’t self-heal, they don’t scale, and they don’t allow for easy updates

- Deployments do all things like

- "auto scale" (scale-up/down)

- "self-heal"

- "rolling updates"

- "roll backs"

- That's why we almost always deploy Pods via 'Deployments"

#

Creating Deployments

--------------------

# List all nodes in K8s cluster

$ kubectl get nodes

# List all pods in K8s cluster

$ kubectl get pods

# Create the deployment

$ kubectl create -f deploy-nginx.yml

vim deploy-nginx.yml

-------

apiVersion: apps/v1

kind: Deployment

metadata:

name: flipkart-prod-deploy

labels:

app: flipkart-prod

spec:

replicas: 6

selector:

matchLabels:

app: flipkart-pod-template

template:

metadata:

labels:

app: flipkart-pod-template

spec:

containers:

- name: nginx-container

image: nginx

ports:

- containerPort: 80

# Creating deployment

$ kubectl create -f deploy-nginx.yml

# Check pod creations

$ kubectl get pods --watch

# Login to pods and verify nginx application

$ kubectl get pods -o wide

$ kubectl exec -it nginx-deploy-5f654bcccd-27xtg /bin/bash

# launch application from individual Pod

$ curl http://10.44.0.1:80

$ curl http://pod\_ip:80

# Testing Self-healing capability

----------------------

If you delete some Pods, Kubernetes can automatically re-create the same for us to make sure given no. of Pods are always running.

- Delete the Pods

$ kubectl delete pods POD\_NAME1 POD\_NAME2

- Check if the Pods are re-created

$ kubects get pods

Creating service to expose the application to outside world and setting up load balancer

===============================================

$ vim service-nginx.yml

-----

apiVersion: v1

kind: Service

metadata:

name: flipkart-service

labels:

app: flipkart-pod-service

spec:

selector:

app: flipkart-pod-template

type: NodePort

ports:

- nodePort: 31000

port: 80

targetPort: 80

# Create the service

$ kubectl create -f service-nginx.yml

# Enable networking

Click on Navigation menu(three lines on top left) --> Go to VPC Network --> Firewal rules --> select on one existing rule --> edit --> Source IP ranges

--> 0.0.0.0/0 --> In 'Specified protocols and ports', write this range "0-65535"

# Access the application from browser using worker-node port

http://34.93.139.52:31000/

http://WorkerNodeIP:Node-port/

Project-3 [GamutKart]

==========================

# Creating deployment for GamutKart

$ vim deploy-gamutkart.yml

apiVersion: apps/v1

kind: Deployment

metadata:

name: gamutkart-deploy

labels:

app: gamutkart-app

spec:

replicas: 8

selector:

matchLabels:

app: gamutkart-app

template:

metadata:

labels:

app: gamutkart-app

spec:

containers:

- name: gamutkart-container

image: nageshvkn/gamutkart-img

ports:

- containerPort: 8080

command: ["/bin/sh"]

args: ["-c", "/root/apache-tomcat-8.5.38/bin/startup.sh; while true; do sleep 1; done;"]

# Execute deployment

$ kubectl create -f deploy-gamutkart.yml

# Creating service for GamutKart

$ vim service-gamutkart.yml

apiVersion: v1

kind: Service

metadata:

name: gamutkart-service

labels:

app: gamutkart-app

spec:

selector:

app: gamutkart-app

type: NodePort

ports:

- nodePort: 31000

port: 8080

targetPort: 8080

# Creating the service

$ kubectl create -f service-gamutkart.yml

#

# Enable networking

TODO:

Go to VPC Network --> Firewal --> select on one existing rule --> edit --> Source IP ranges

--> 0.0.0.0/0 --> In "Specified protocols and ports", write this range "0-65535"

Note:

Kubernates Port Range: 30,000 - 32,767

#

3.) Deleting a Pod

$ kubectl get pods

$ kubectl delete pods nginx-pod

$ kubectl delete -f pod.yml

# Misc:

4.) Get all nodes IPs in Kubernetes cluster

$ kubectl get nodes -o wide

#

List Deployments & Service

$ kubectl get deployment

$ kubectl get svc (Or service)

#

5.) Deleting Deployment & Service

$ kubectl delete -f deploy-gamutkart.yaml(deployment yaml file name)

$ kubectl delete -f service-gamutkart.yml( service yaml file name)

$ kubectl delete deployment <deployment-name>

$ kubectl delete service <service-name>

#Scaleup Pods

$ kubectl scale deployments/gamutkart-deploy --replicas=2

8:30 PM daily

--

- Creatng a POD on a particular WN

- Autoscale example

- Assign different names to different PODs manually

- Create different replicas for different PODs

- select a deployment as part ofa serice

- How to list all the API versions w.r.t Object implementation.

-