Kubernetes

"Kubernetes is an open source system for automating deployments, scaling and management of containerised applications."

Let's learn how to setup kubernetes cluster on AWS using KOPS (kubernetes operations).

NOTE: Before proceeding, we assume that you have a basic understanding about Kubernetes and AWS.

#Prerequisites:

- Ubuntu instance
- AWS-cli setup
- S3 bucket

#Video Tutorial:

https://youtu.be/IImQrJWbaDo

#Install kubectl

On your ubuntu instance, make sure AWS CLI is setup and also the kops binary. We shall also need kubectl (Kubernetes CLI)

- macOS: ***brew install kubernetes-cli***

- Linux:

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

chmod +x ./kubectl


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

#Install kops on ubuntu instance:

```
wget
https://github.com/kubernetes/kops/releases/download/1.6.1/kops-li
nux-amd64
chmod +x kops-linux-amd64

sudo mv kops-linux-amd64 /usr/local/bin/kops
```

#Create Route53 domain for the cluster(on aws)

kubernetes uses DNS for discovery inside the cluster so that you can reach out kubernetes API server from clients.
create a hosted zone on Route53, say, k8s.appychip.vpc. The API server endpoint will then be **api.k8s.appychip.vpc**

#Create a S3 bucket

Now, create a S3 bucket which will keep the configuration for the cluster:

```
$ aws s3 mb s3://clusters.k8s.appychip.vpc
```

#Expose environment variable:

```
$ export KOPS_STATE_STORE=s3://clusters.k8s.appychip.vpc
```

#Create Kubernetes Cluster

Now we're ready to create a cluster. You can reuse existing VPC (kops will create a new subnet in this VPC) by providing the vpc-id option:

```
$ kops create cluster --cloud=aws --zones=us-east-1d
--name=useast1.k8s.appychip.vpc --dns-zone=appychip.vpc --dns
private
```

NOTE: Make sure you have ssh keys already generated otherwise it will throw an error.

#To actually create cluster run:

```
kops update cluster useast1.k8s.appychip.vpc --yes
```

This will do all the required stuff of creating the VPC, subnets, autoscaling-groups, nodes etc. which you can observe in the output. If you want to review what all things going to happen when this command would be run then run the above command without --yes option. Without --yes option, it will print the action it is going to perform without actually doing it.

You can then edit the cluster settings with one of these commands:

- List clusters with: kops get cluster
- Edit this cluster with: kops edit cluster useast1.k8s.appychip.vpc
- Edit your node instance group: kops edit ig --name=useast1.k8s.appychip.vpc nodes
- Edit your master instance group: kops edit ig --name=useast1.k8s.appychip.vpc master-us-east-1d

Then wait, it takes quite some time for the instances to boot and the DNS entries to be added in the zone. Once everything is up you should be able to get the kubernetes nodes:

```
$ kubectl get nodes

NAME                                        STATUS AGE VERSION
ip-172-20-33-144.ec2.internal Ready  4m  v1.6.2
ip-172-20-39-78.ec2.internal  Ready  1m  v1.6.2
ip-172-20-45-174.ec2.internal Ready  2m  v1.6.2
```

To enable the Kubernetes UI you need to install the UI service:

```
$ kubectl create -f
https://rawgit.com/kubernetes/dashboard/master/src/deploy/kubernetes-dashboard.yaml
```

Then you can use the kubectl proxy to access the UI from your machine:

```
$ kubectl proxy --port=8080 &
```

The UI should now be available at http://localhost:8080

#Deploying Nginx Container

To test our new Kubernetes cluster, we could deploy a simple service made up of some nginx containers:

#Create an nginx deployment:

```
$ kubectl run sample-nginx --image=nginx --replicas=2
--port=80
$ kubectl get pods

NAME                    READY     STATUS    RESTARTS   AGE
sample-nginx-379829228-xb9y3   1/1       Running   0          10s
sample-nginx-379829228-yhd25   1/1       Running   0          10s

$ kubectl get deployments

NAME       DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
sample-nginx   2         2         2            2           29s
```

Expose the deployment as service. This will create an ELB in front of those 2 containers and allow us to publicly access them:

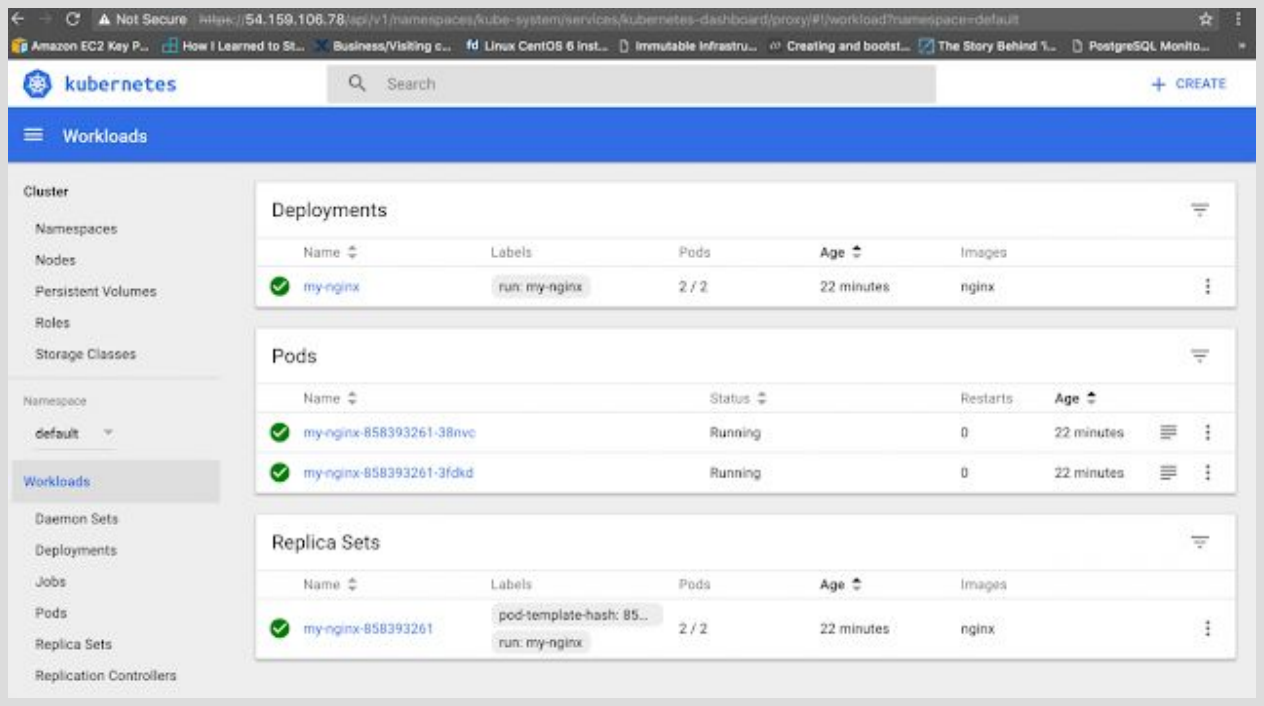
```
$ kubectl expose deployment sample-nginx --port=80
--type=LoadBalancer

$ kubectl get services -o wide
```

```
NAME          CLUSTER-IP      EXTERNAL-IP
PORT(S)   AGE       SELECTOR
kubernetes    100.64.0.1      <none>
443/TCP   25m       <none>
sample-nginx      100.70.129.69
adca6650a60e611e7a66612ae64874d4-175711331.us-east-1.elb.am
azonaws.com/   80/TCP    19m       run=sample-nginx
```

There is an ELB running on http://adca6650a60e611e7a66612ae64874d4-175711331.us-east-1.elb.amazonaws.com with our nginx containers behind it:

You can also view the UI by accessing master node. Hit master node's IP/Domain in browser, it will ask for credentials. Run command ***kubectl config view*** to see the credentials.



#How to shutdown a kops Kubernetes cluster on AWS

$ kops get ig nodes

$ kops get ig

$ kops edit ig master-us-west-1c
Edit the minSize and maxSize to {0}

- Update your cluster with the changes

$ kops update cluster --yes
$ kops rolling-update cluster

#To restart the cluster instances :
Edit the minSize and maxSize to {1/2}

#To get and validate all the clusters :

```
$kops get clusters
$kops validate cluster
```