1) Introduction to AWS EKS (Small Overview)
2) Set up EKS Clusters Using eksctl
3) Deploy MEAN Stack Web App using kubectl
4) Autoscaling: Horizontal Pod autoscaling and Cluster Nodes Autoscaling
5) Kubernetes Cluster Monitoring using Prometheus and Grafana

A practical guide to AWS EKS

By Sandip Das

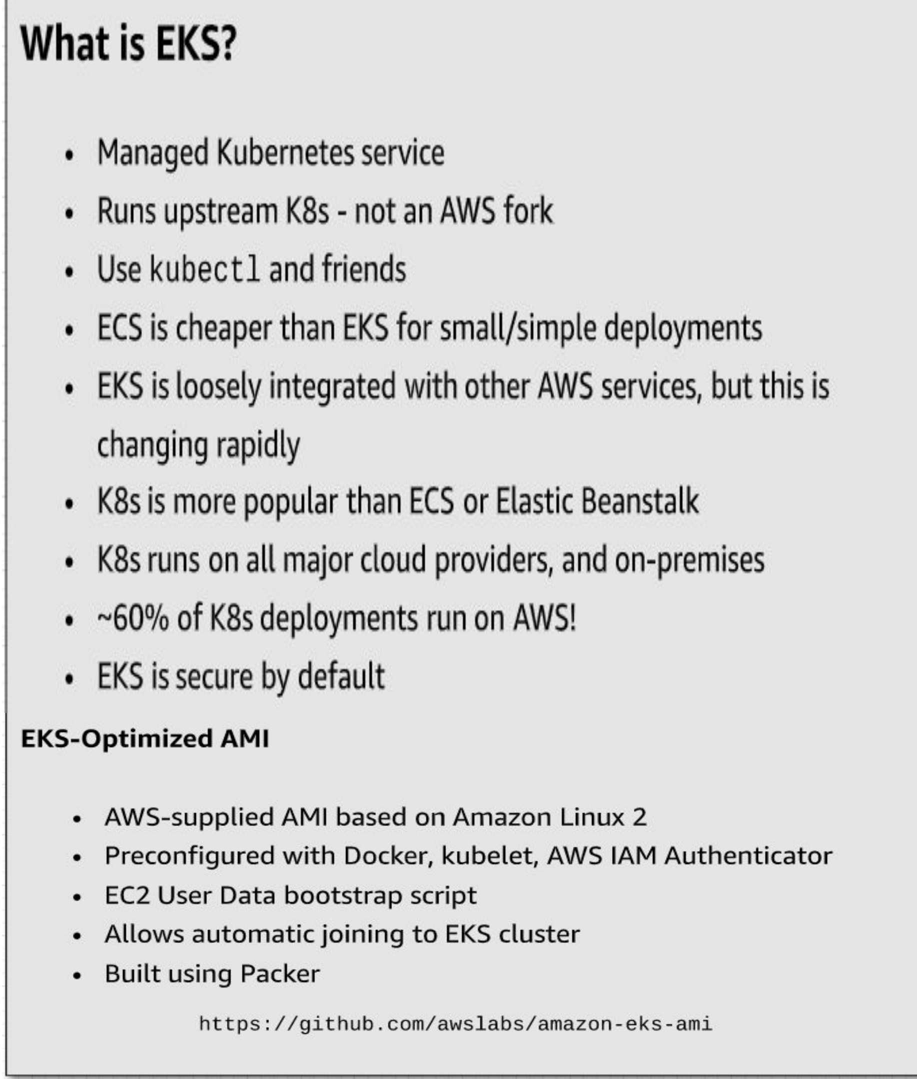
Contents

AWS EKS

Amazon Elastic Kubernetes Service (Amazon EKS) makes it easy to deploy, manage, and scale containerized applications using Kubernetes on AWS.

Amazon EKS runs the Kubernetes management infrastructure for you across multiple AWS availability zones to eliminate a single point of failure. Amazon EKS is certified Kubernetes conformant so you can use existing tooling and plugins from partners and the Kubernetes community. Applications running on any standard Kubernetes environment are fully compatible and can be easily migrated to Amazon EKS.

Amazon EKS is generally available for all AWS customers.



What is EKS?

- Managed Kubernetes service
- Runs upstream K8s - not an AWS fork
- Use `kubectl` and friends
- ECS is cheaper than EKS for small/simple deployments
- EKS is loosely integrated with other AWS services, but this is changing rapidly
- K8s is more popular than ECS or Elastic Beanstalk
- K8s runs on all major cloud providers, and on-premises
- ~60% of K8s deployments run on AWS!
- EKS is secure by default

EKS-Optimized AMI

- AWS-supplied AMI based on Amazon Linux 2
- Preconfigured with Docker, kubelet, AWS IAM Authenticator
- EC2 User Data bootstrap script
- Allows automatic joining to EKS cluster
- Built using Packer

https://github.com/awslabs/amazon-eks-ami

Benefits

Let’s discuss in short what features it provides

- ➔ No Control plane to manage
- ➔ Secure by default
- ➔ Comformant and Compatible
- ➔ Optimized for cost
- ➔ Build with the community

Cost

Let's discuss about the cost

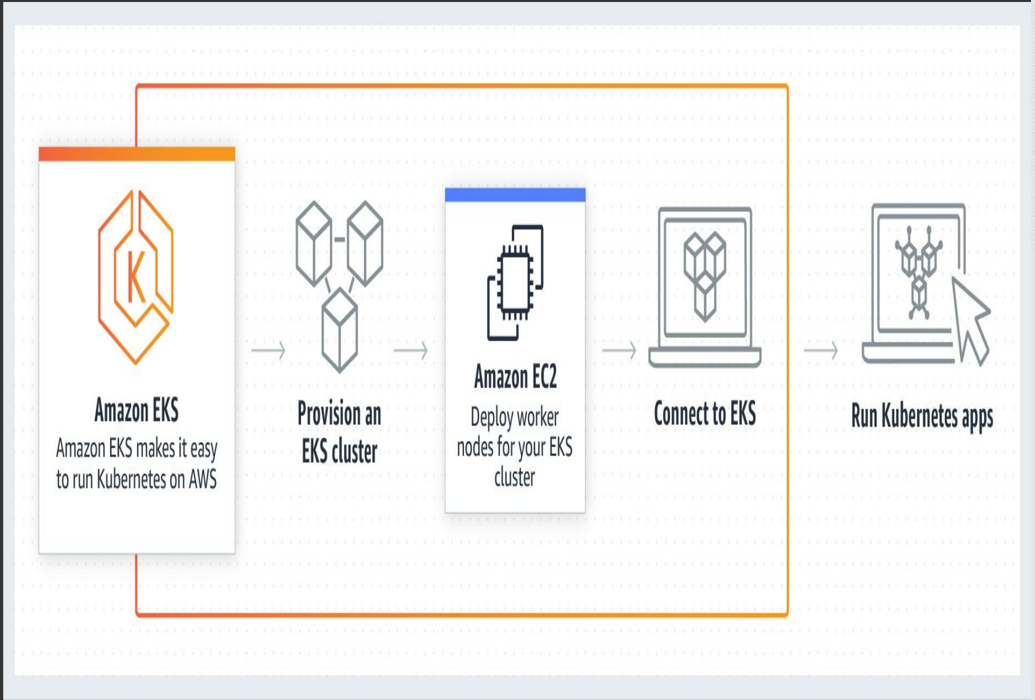
- ➔ **Each EKS Cluster Cost $0.20/hour i.e. $144/month**

  You can use a single Amazon EKS cluster to run multiple applications by taking advantage of Kubernetes namespaces and IAM security policies.

- ➔ **EC2 Instances & EBS volumes used in AS Worker Nodes**

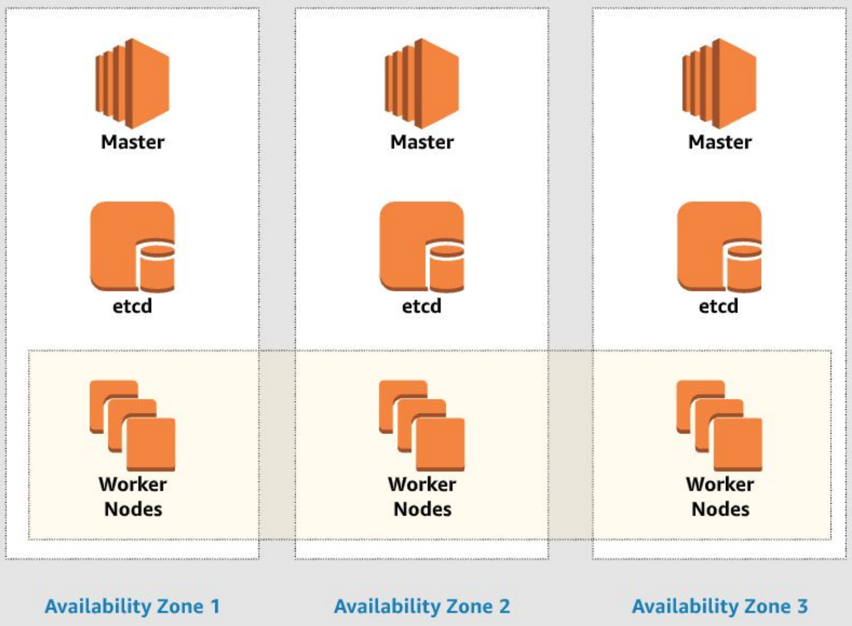
  You pay for AWS resources , you create to run your Kubernetes worker nodes. You only pay for what you use, as you use it; there are no minimum fees and no upfront commitments.

EKS Architecture



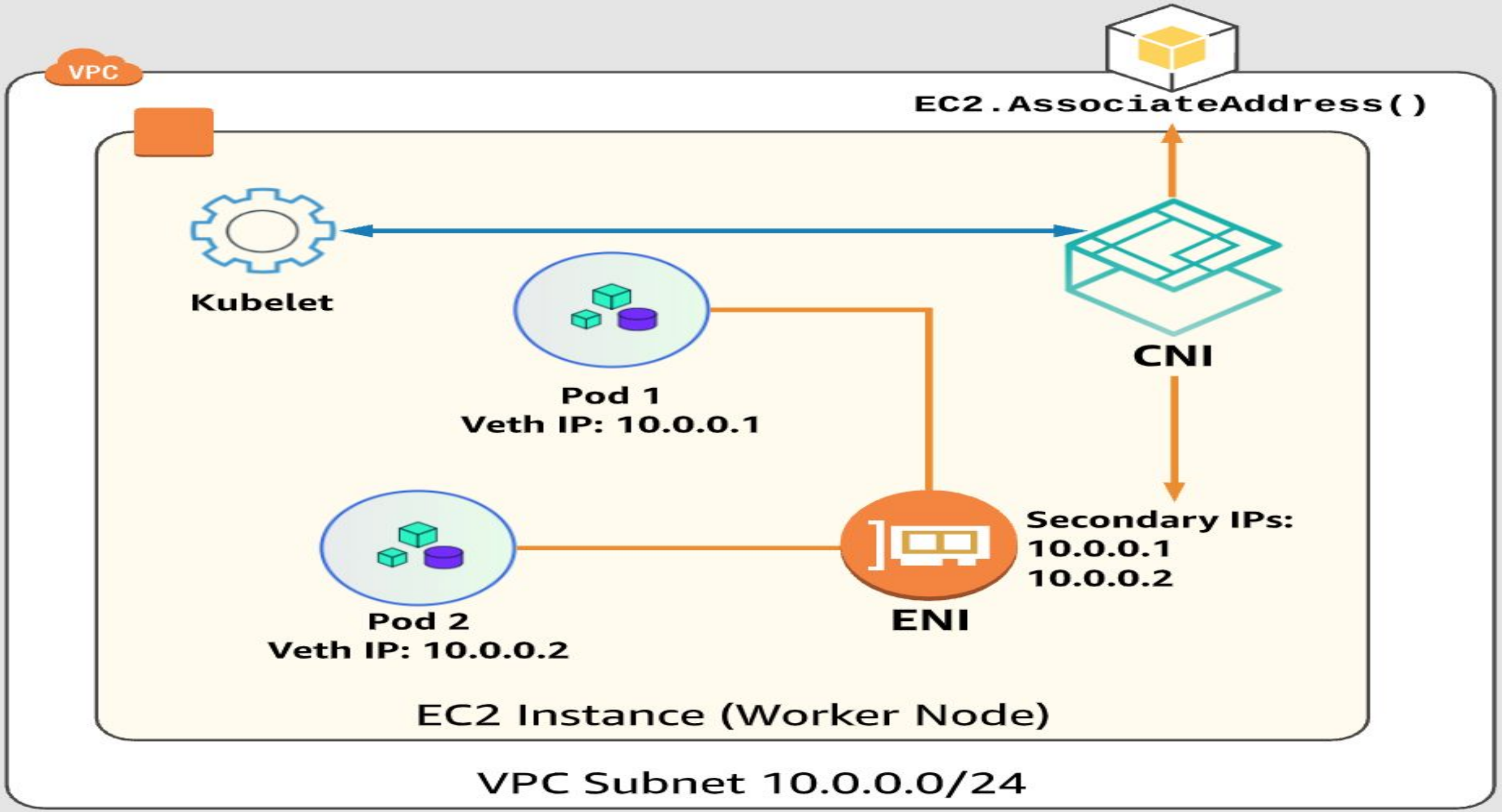
Amazon EKS
Amazon EKS makes it easy to run Kubernetes on AWS
Provision an EKS cluster
Amazon EC2
Deploy worker nodes for your EKS cluster
Connect to EKS
Run Kubernetes apps

Managed Control Plane



- EKS provieds K8s master nodes, API servers, etcd layer
- 3 master and 3 etcd nodes by default
- Backups, etcd snapshots, autoscaling included
- You provision and manage the EC2 worker nodes
- Unlike kops, you don't roll your own master (https://github.com/kubernetes/kops)
- Masters and etcd are Multi-AZ
- EKS scales master nodes for you

Kubernetes and VPC Networking



Companies Using AWS EKS

GoDaddy
Pearson
FICO™
intuit.
verizon✓

Honeywell
THE POWER OF CONNECTED
logicworks
RetailMeNot
LogMeIn®
CONDÉ NAST
INTERCULTURE GROUP | TAIWAN
mercari

Verisk
EBSCO
skyscanner
axway
CyberAgent.
trainline
Wonderfully Predictable

Let’s Start an AWS EKS Cluster and Deploy A MEAN Stack Application

For best security perspective, I will suggest utilizing Amazon Linux v2 micro instance with ec2 role and not using personal computer system for any kind of cluster operation or kubernetes related operations to avoid unwanted issues related to security ,project management or confusion / conflict with other projects . You should use personal computer for development purpose only.

I will be using Amazon Linux 2 EC instance with admin IAM role attached to it for entire demo. Don’t forget to assign administrator IAM role to the instance or if you are doing set-up in personal machine then make sure to configure aws cli locally.

Project git: https://github.com/cncfkol/mean_eks_demo

All Kubernetes related config files are in “k8s” folder

Install Essential Tools

We will need below tools to be installed

- ➔ **Install aws cli**
  https://docs.aws.amazon.com/cli/latest/userguide/cli-chap-install.html
- ➔ **Install kubectl**
  https://docs.aws.amazon.com/eks/latest/userguide/install-kubectl.html
- ➔ **Install aws-iam-authenticator**
  https://docs.aws.amazon.com/eks/latest/userguide/install-aws-iam-authenticator.html
- ➔ **Install eksctl**

  https://docs.aws.amazon.com/eks/latest/userguide/getting-started-eksctl.html or
  https://eksctl.io/introduction/installation/

Create AWS EKS Cluster

Create Cluster:

```
eksctl create cluster -f cluster.yaml
```

```
Then check for the message:
 EKS cluster "<cluster name>" in "<aws region>" region is ready
```

```
Check that kubectl client get auto set properly or not by:
cat /home/ec2-user/.kube/config
```

If want to Delete Cluster anytime:

```
eksctl delete cluster -f cluster.yaml
```

Sample Cluster yml file:

https://gist.github.com/sd031/e72eb9f454340a7c844da31b97716e0a

Installing: Dashboard, Heapster, InfluxDb

To Install Kubernetes Dashboard:

Check: https://github.com/kubernetes/dashboard

```
Below are the commands:
```

```
kubectl apply -f
https://raw.githubusercontent.com/kubernetes/dashboard/v1.10.1/src/deploy/recommended/kubernetes-dashboard.yaml
```

Installing Heapster and InfluxDB

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes/heapster/master/deploy/kube-config/influxdb/heapster.yaml
```

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes/heapster/master/deploy/kube-config/influxdb/influxdb.yaml
```

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes/heapster/master/deploy/kube-config/rbac/heapster-rbac.yaml
```

`There is a problem currently, and for fix check:`

https://github.com/kubernetes/dashboard/wiki/Accessing-Dashboard---1.7.X-and-above

https://github.com/kubernetes/dashboard/issues/916 `Just use https to fix it`

Create an administrative account and cluster role binding to access the dashboard

```
kubectl apply -f eks-admin-service-account.yaml
```

`File Source:` https://gist.github.com/sd031/37de261794f8fe00d1e277027a862039

```
kubectl apply -f eks-admin-cluster-role-binding.yaml
```

`File Source:` https://gist.github.com/sd031/46673ac1d04554504393a95d405863b2

Start proxy

```
kubectl proxy --address 0.0.0.0 --accept-hosts '.*' &
```

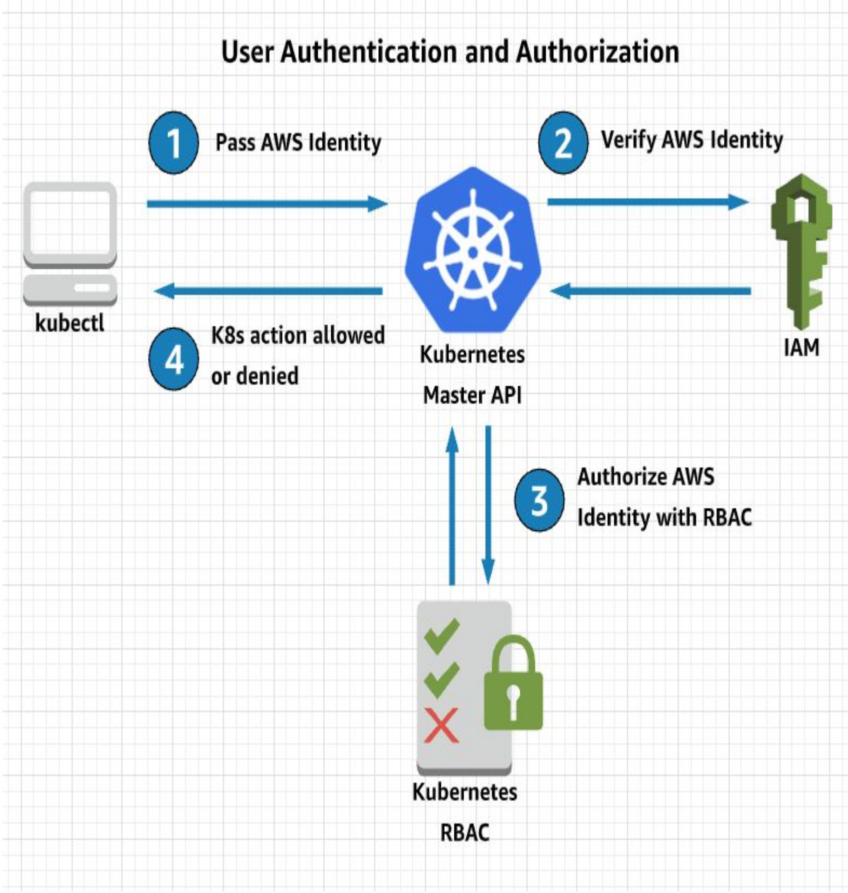
Get a token

```
aws-iam-authenticator -i <cluster_name> token
E.g: aws-iam-authenticator -i mean-eks-demo token or
aws-iam-authenticator token -i mean-eks-demo --token-only
```

Log in

http://ip:8001/api/v1/namespaces/kube-system/services/https:kubernetes-dashboard:/proxy/#!/login

```
If any issue check all pods running properly or not: kubectl get pods -o wide --all-namespaces
```


User Authentication and Authorization
1 Pass AWS Identity
2 Verify AWS Identity
kubectl
Kubernetes Master API
IAM
4 K8s action allowed or denied
3 Authorize AWS Identity with RBAC
Kubernetes RBAC

Build the docker image using Dockerfile

Run Below command from Project Directory to build the image

Before that install docker: https://docs.aws.amazon.com/AmazonECS/latest/developerguide/docker-basics.html

E.g.

```
sudo yum install docker
```

Sample MEAN Stack APP GIT: https://github.com/linnovate/mean

Docker File is already available in the git itself , I have modified a bit:

https://gist.github.com/sd031/a54bd5f575279bf88d7b4be0ea34f38a

```
docker image build -t <image_name>:tag . (this dot refer to current directory)
```

e.g.

```
docker image build -t mean_demo:v1 .
```

Test image running fine or not:

```
docker container run  --name mean_demo -p 4040:4040 mean_demo:v1
```

Publishing Image to AWS ECR

It’s just like Docker Hub, where we can push and Pull image and we can use it with AWS ECR same way. To get the docker login and auto execute the login the command is:

```
$(aws ecr get-login --no-include-email --region region-name)
```

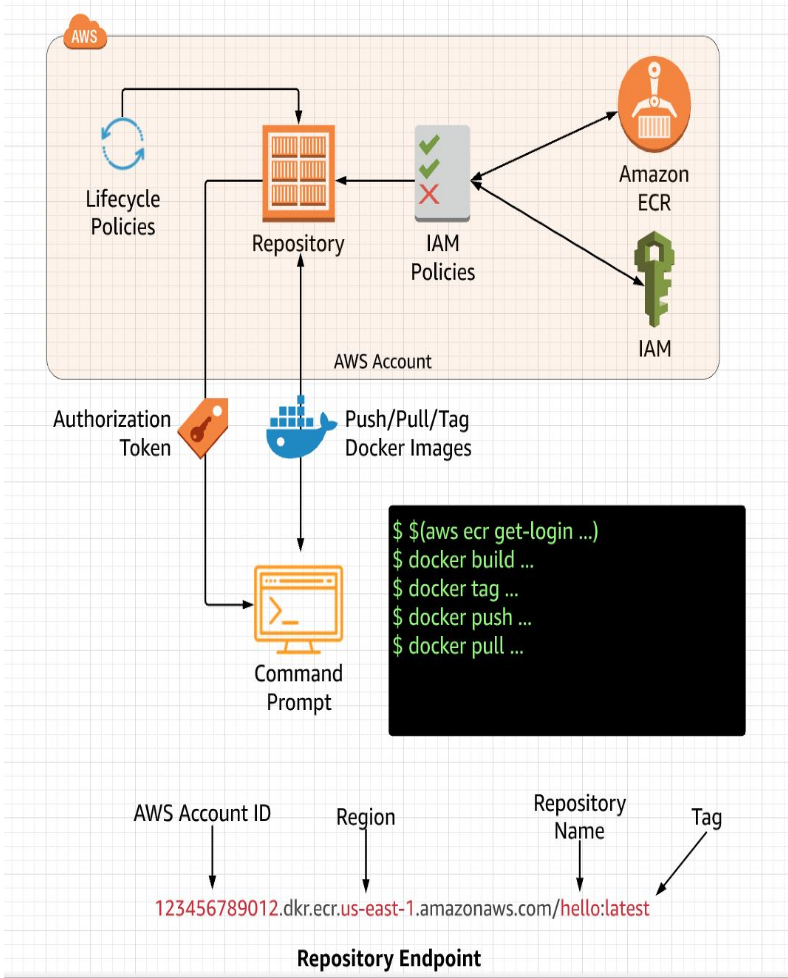
```
E.g $(aws ecr get-login --no-include-email --region us-west-2)
```

After this create a ECR repo from AWS Management console, it will give a url like Showing in right side diagram, after that tag The image: (All command are already given in AWS ECR UI, just use the same)

```
docker tag mean_demo:v1
123456789012.dkr.ecr.us-west-2.amazonaws.com/mean_demo:v1
docker push
123456789012.dkr.ecr.us-west-2.amazonaws.com/mean_demo:v1
```

Make sure you use the right tags

ECR Components



```
$ $(aws ecr get-login ...)
$ docker build ...
$ docker tag ...
$ docker push ...
$ docker pull ...
```

Finally Let's deploy the Application

```
kubectl apply -f deployment.yaml
```

deployment.yaml File sample-url:

https://gist.github.com/sd031/d28f1bd0ca1b0aec7eb109804f966c6

```
kubectl apply -f kubernetes/service.yaml
```

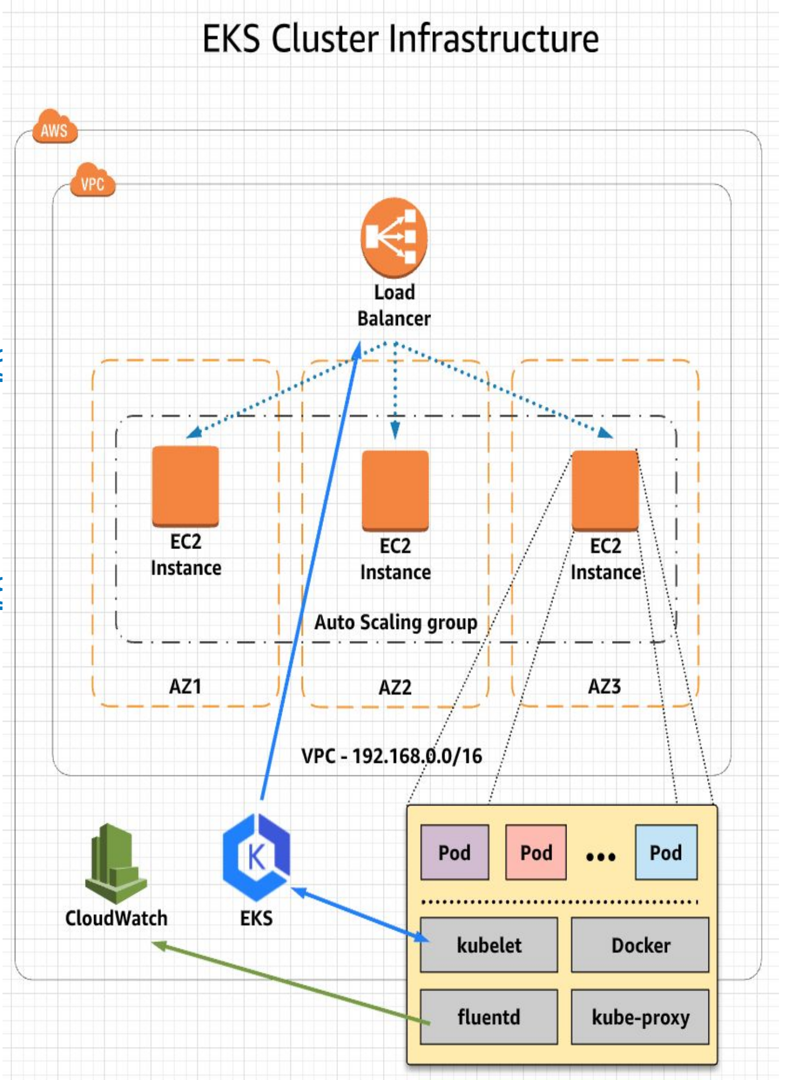
service.yaml File Sample:

https://gist.github.com/sd031/e5cd4a3d80813e779a7bb20b6cc3a8

If any issue check logs:

```
kubectl get pods

kubectl describe pod pod_name_here
```



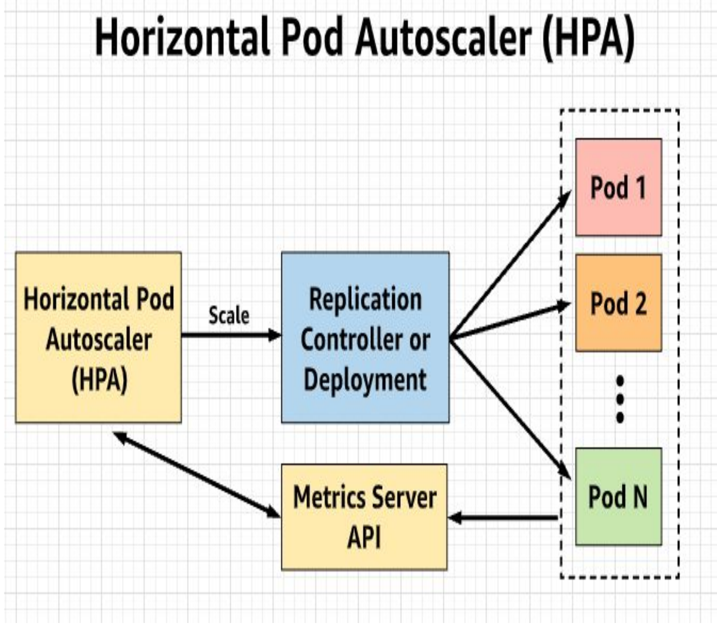
Auto Scaling: Horizontal Pod Autoscaler (HPA)

```
Install Helm

curl https://raw.githubusercontent.com/kubernetes/helm/master/scripts/get > get_helm.sh
chmod +x get_helm.sh
./get_helm.sh
`kubectl apply -f tiller-rbac.yaml`
Tiller-rbac.yaml file:
```

https://gist.github.com/sd031/6d0207d5920b00cc475475174eed36a0

~~~
### Install Helm using Tiller Service Account
`helm init --service-account tiller`
### Install Metrics Server
```bash
helm install stable/metrics-server --name metrics-server --version 2.0.4 --namespace metrics`
kubectl get apiservice v1beta1.metrics.k8s.io -o yaml
### Autoscale the Deployment
`kubectl autoscale deployment mean-demo-deployment --cpu-percent=50 --min=2 --max=10`
Check status: kubectl get hpa or kubectl get hpa -w
To run load test: Deploy Apache/PHP
`kubectl run php-apache --image=k8s.gcr.io/hpa-example --requests=cpu=200m --expose --port=80`
`kubectl autoscale deployment php-apache --cpu-percent=50 --min=1 --max=10`
kubectl run -i --tty load-generator --image=busybox /bin/sh
while true; do wget -q -O - http://php-apache; done
~~~


Horizontal Pod Autoscaler (HPA)
Horizontal Pod Autoscaler (HPA)
Scale
Replication Controller or Deployment
Pod 1
Pod 2
Pod N
Metrics Server API

Auto Scaling: Cluster Autoscaler (CA)

Find Auto Scaling group using the AWS Management Console, noting its name.

Edit the ASG's min/max size to 2 and 8 nodes, respectively.

Edit `cluster_autoscaler.yaml`, replacing `<AUTOSCALING GROUP NAME>` with the ASG name you found in the console.

Cluster_autoscaler.yaml file: https://gist.github.com/sd031/3f0f4c89559e0c4bf026d44db4855ad3

Optionally change the `AWS_REGION` to something other than `us-east-1` if you are working in a different region.

We need to configure an inline policy and add it to the EC2 instance profile of the worker nodes

Policy json: https://gist.github.com/sd031/8d21542f81b85d551462a86d84b6b926

Deploy the autoscaler:

`kubectl apply -f cluster_autoscaler.yaml`

Watch the logs:

`kubectl logs -f deployment/cluster-autoscaler -n kube-system`

Scale out test:

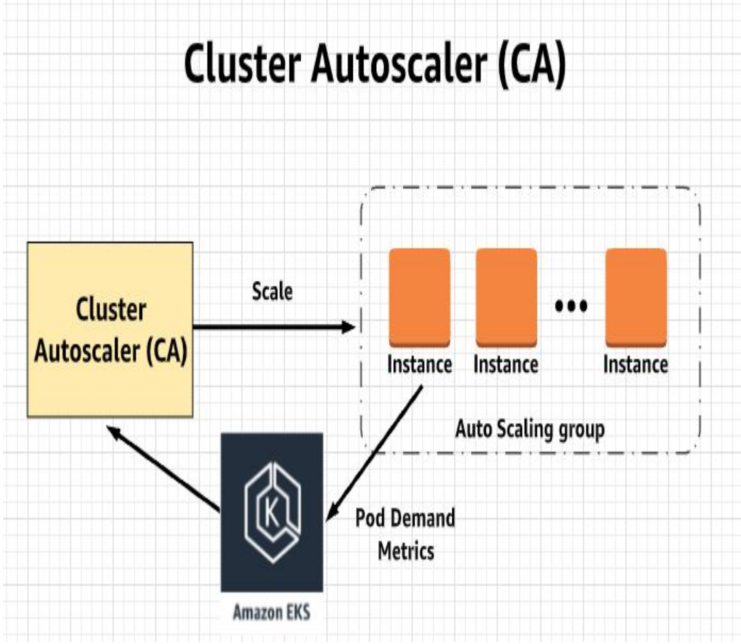
kubectl apply -f nginx.yaml

Nginx.yaml file:

https://gist.github.com/sd031/95b966269a4d99ce0cd45b79df3195a3

kubectl get deployment/nginx-scaleout

kubectl scale --replicas=10 deployment/nginx-scaleout



Monitoring an EKS Cluster

Configure Storage Class

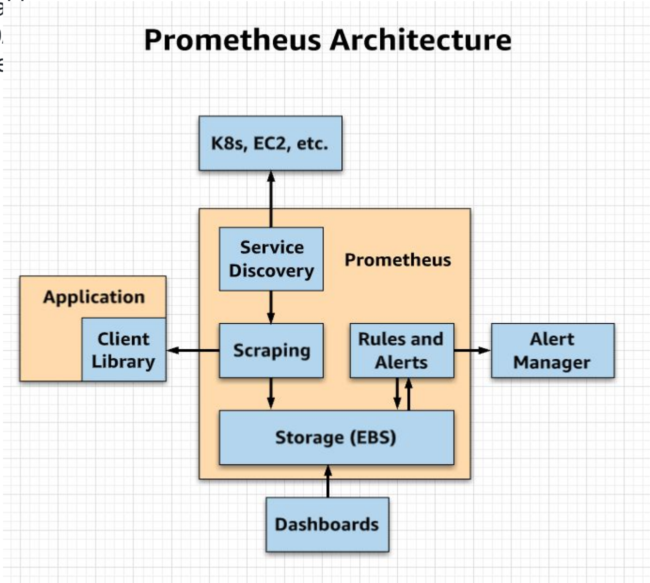
kubectl create -f prometheus-storageclass.yaml

prometheus-storageclass.yaml File Link: https://gist.github.com/sd031/e094d23fe0e817bec82901b9a93dc5b1

Deploy Prometheus
helm install -f prometheus-values.yaml stable/prometheus --name prometheus --namespace prometheus
Prometheus-values.yaml file: https://gist.github.com/sd031/1cd40bb5a5b14c39e87242845a92d1f3

kubectl get all -n prometheus
You should see all the Prometheus pods, services, daemonsets, deployments, and replica sets are all ready and availa
You can access Prometheus server URL by going to any one of your Worker node IP address and specify port `:30900
Remember to open port 30900 in your Worker nodes Security Group. In the web UI, you can see all the targets and me

Install Grafana
kubectl create namespace grafana
Execute Script:
https://gist.github.com/sd031/2c54da7bb4476045a4ae7d3fda90e707
Run the following command to check if Grafana is deployed properly:
kubectl get all -n grafana
Get Load Balancer url:
export ELB=$(kubectl get svc -n grafana grafana -o jsonpath='{.status.loadBalancer.ingress[0].hostname}')
echo "http://$ELB"
When logging in, use the username admin and get the password hash by running the following:
kubectl get secret --namespace grafana grafana -o jsonpath="{.data.admin-password}" | base64 --decode ; echo



Final Grafana Dashboard Set-up

Create Dashboards:

Login into Grafana dashboard using credentials supplied during configuration

You will notice that 'Install Grafana' & 'create your first data source' are already completed. We will import community created dashboard for this tutorial

Click '+' button on left panel and select 'Import'

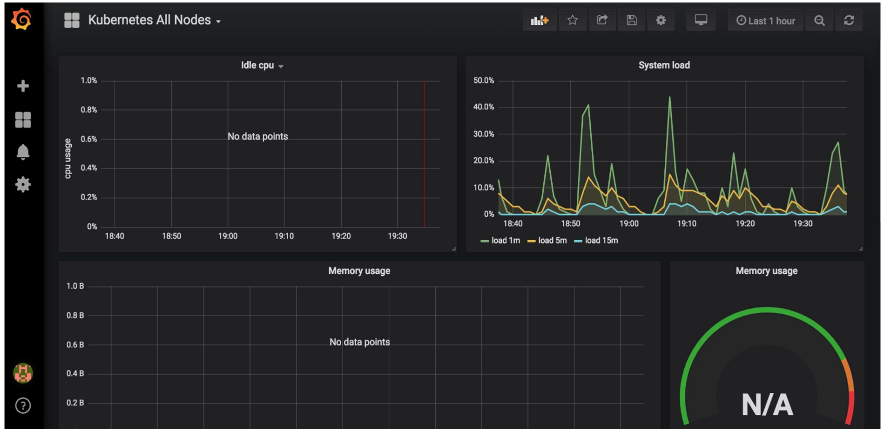
Enter 3131 dashboard id under Grafana.com Dashboard & click 'Load'.

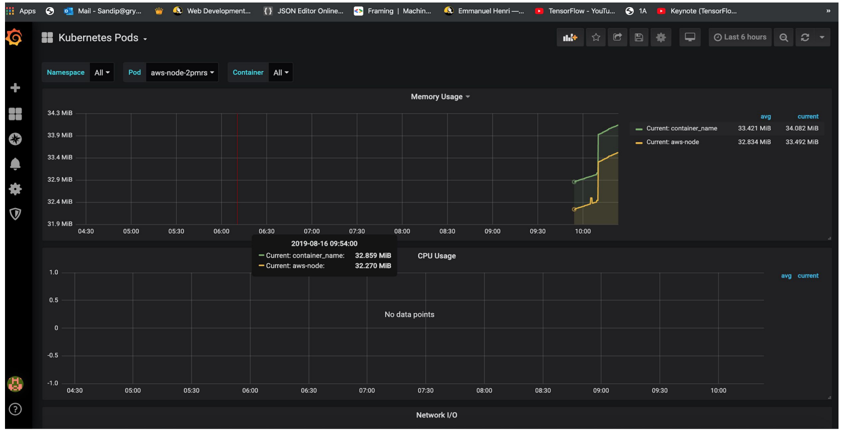
Leave the defaults, select 'Prometheus' as the endpoint under prometheus data sources drop down, click 'Import'.

This will show monitoring dashboard for all cluster nodes

For creating dashboard to monitor all pods, repeat same process as above and enter 3146 for dashboard id, I personally like: 7249, 3091, 1860

You can find more dashboards and plugins in: https://grafana.com/grafana/dashboards/ - but as long term goal, learn how to do it by yourself




Kubernetes Pods
Last 6 hours
Namespace
All
Pod
aws-node-2pmrs
Container
All
Memory Usage
avg
current
Current: container_name 33.421 MiB 34.082 MiB
Current: aws-node 32.834 MiB 33.492 MiB
2019-08-16 09:54:00
Current: container_name: 32.859 MiB
Current: aws-node: 32.270 MiB
CPU Usage
No data points
Network I/O

Clean up everything

Since we have used eksctl, it’s a lot easier,

Before doing anything, remove the inline policy from cluster instance role, just got to instance worker node, then check details, you will find the role link: IAM role e.g. eksctl-mean-eks-demo-nodegroup-ng…..

Click on that, and you will see, under permission the policy you had added, e.g. CA, just delete that.
Then remove autoscaling, deployment , services, prometheus , grafana etc etc e.g.
kubectl delete -f cluster_autoscaler.yaml
kubectl delete -f nginx.yaml
Delete the horizontal pod autoscaler and load test:
kubectl delete hpa,svc php-apache
kubectl delete deployment php-apache load-generator
kubectl delete -f deplyment.yaml
helm delete prometheus
helm del --purge prometheus
helm delete grafana
helm del --purge grafana

Now finally delete the cluster:

Check how many are running:

eksctl get clusters

Delete cluster:

```
eksctl delete cluster -f cluster.yaml
```

Behind the scene the cloud formation stack will get deleted and accordingly resources will be deleted as well, must do it if you are doing in development or test as a temporary deployment otherwise it will cost you a lot

There are few more things you need to know

This demo is just the start points and there is a lot more out there, the more you use it , the more experience you will gather, so I will highly suggest try by yourself and deploy your own AWS EKS Cluster.

After trying the basic app deployments , the next thing you might be interested to learn are:

1) Using Spot instances with Kubernetes and save 90% of cost
2) Types of Deployments available in Kubernetes Deployment and how to configure it
3) Types of Services available in Kubernetes Service and how to configure it.
4) CI/CD with Kubernetes
5) Logging with Cloud Trail
6) Logging to Cloudwatch with Fluentd
7) Complex Authentication , Roles etc

There are lot of resources available online you can learn (check the linked resources) or maybe you want another meet-up to dig deeper into any of these, Let Me or Chirag know the same.

Good luck!

I hope you'll use this knowledge and build awesome solutions.

If any issue contact me in Linkedin: https://www.linkedin.com/in/sandip-das-developer/