# Serverless Kubernetes Pods Using Amazon EKS and AWS Fargate



Prepared by

Raghav Kumar (Infosys)

Dheeraj Rajaram (Infosys)

Saket Suman (Infosys)

Reviewed by

Sk Khurshed Jahangir

## Table of Contents:

* Introduction
* Getting Started
* How to setup an EKS cluster on Faregate
* Prerequisites
* Why Fargate
* Setup a Python/Java web application
* Steps to Push an Image into Amazon ECR with Docker
* Setting up Full Cluster
* Step01: To create a new EKS cluster with Fargate
* Step02: creating the fargate-Profile
* Step03: Creating a cluster role, ClusterRoleBinding and Service Account for Alb-ingress-controller
* Step04: Application Load Balancer (ALB) Ingress Controller Deployment Manifest
* Step05: Creating a Namespace
* Step06: To deploy Matrix-Server
* Step07: Deployment config files for creating pod of running container from docker-hub
* Step08: Creating a service type of cluster IP for exposing the deployment
* Step09: Creating an ingress-resources for redirecting user traffic to our backend services
* Step10: creating a hpa for autoscaling our deploymet based on CPU-percentage utilization
* Step11: To delete the Cluster
* All the RUN-Commands
* Conclusion

Important Link

## Introduction:

- Amazon Elastic Kubernetes Service (EKS) is a managed service that makes it easy for running Kubernetes on AWS.

- AWS Fargate provides on-demand, right-sized compute capacity for containers that run as Kubernetes pods as part of an Amazon EKS cluster.

- Using Fargate, Kubernetes pods run with just the compute capacity they request, and each pod runs in its own VM-isolated environment without sharing resources with other pods.

- If you are using AWS Fargate, pricing is calculated based on the vCPU and memory resources used from the time you start to download your container image until the Amazon EKS pod terminates, rounded up to the nearest second.

- For Cost, please visit: https://aws.amazon.com/fargate/pricing/

## Getting Started:

### AWS Fargate:

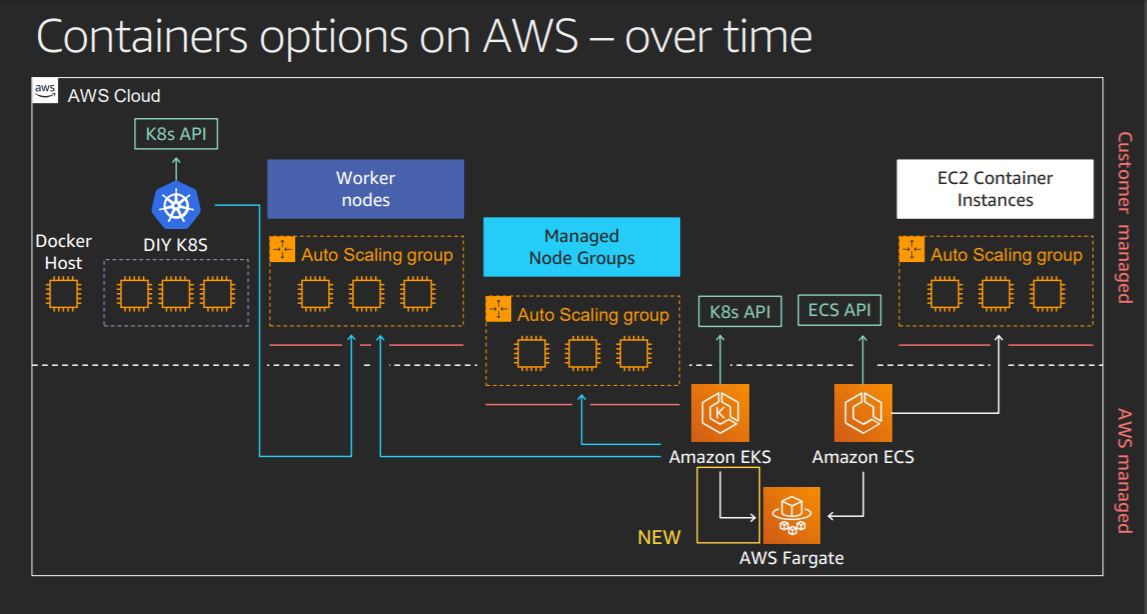
AWS Fargate lets developers build and deploy containerized applications without having to manage any underlying infrastructure. It reduces a lot of the friction involved with being able to deploy applications.

### Elastic Kubernetes Service (EKS):

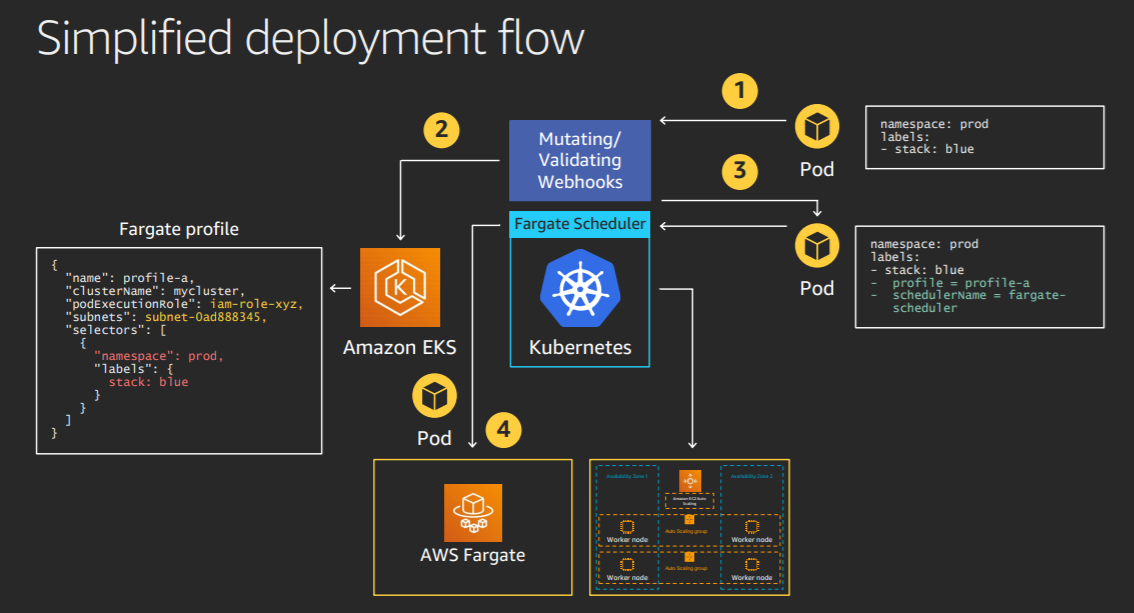
EKS is Amazon’s fully managed Kubernetes Service that lets people run Kubernetes on AWS without having to manage the Kubernetes control plane. It reduces the operational overhead that comes with having to manage and run the control plane.

EKS support for Fargate is available in US East (Virginia), US East (Ohio), Europe (Ireland), and Asia Pacific (Tokyo) regions.

## How to setup an EKS cluster on Faregate:



## 

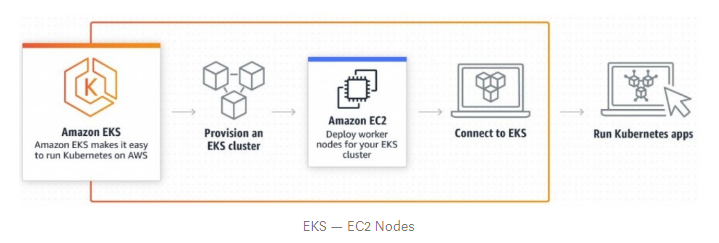


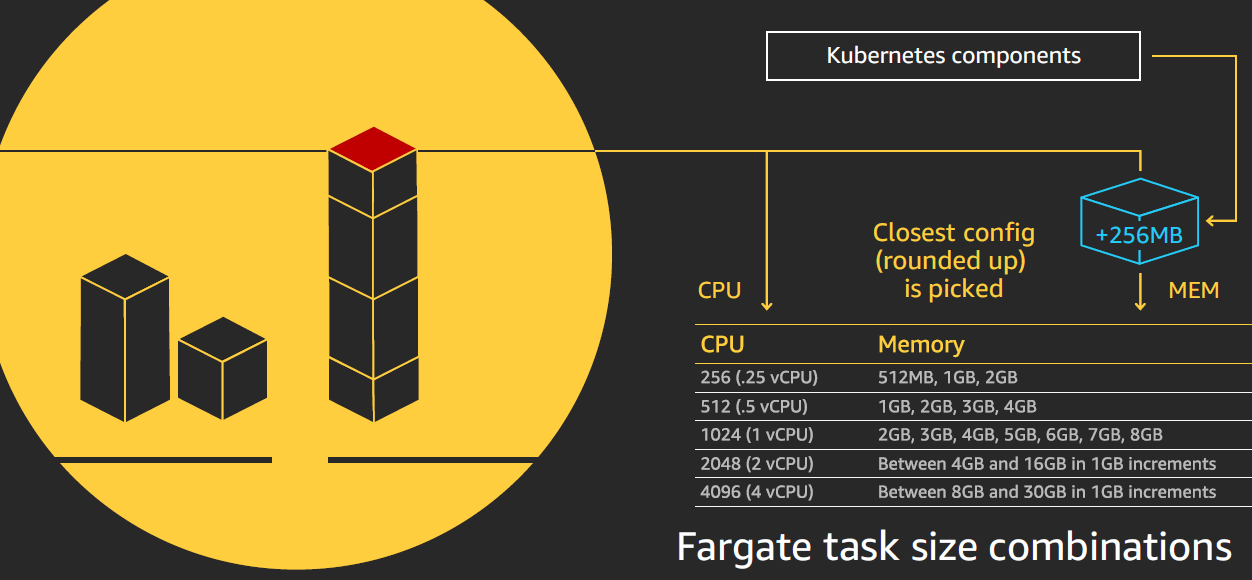
## Prerequisites:

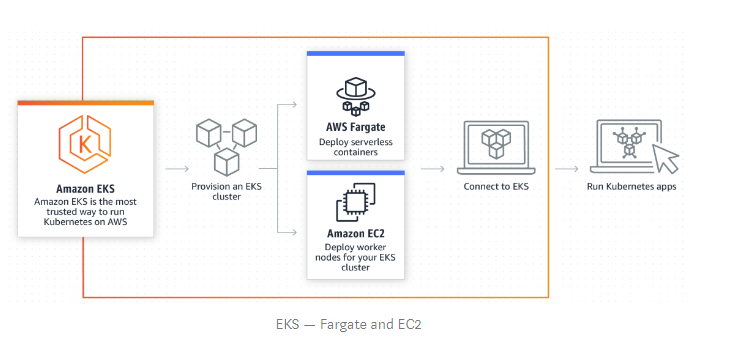
The following are the tools needed:

* [eksctl](https://eksctl.io/introduction/installation/): Official CLI to create a new EKS cluster(eksctl version 0.11 or higher required).
* [kubectl](https://kubernetes.io/docs/tasks/tools/install-kubectl/): CLI to interact with the Kubernetes API server
* [AWS CLI](https://docs.aws.amazon.com/cli/latest/userguide/cli-chap-install.html) + [Docker](https://docs.docker.com/v17.09/engine/installation/): We will use Docker and the AWS CLI to build and push a Docker image for our application.

## Why Fargate:







|  |  |  |
| --- | --- | --- |
|  | EKS | EKS Fargate |
| User Perspective | User launches EKS control plane and adds Amazon EC2 to the EKS cluster.  The required compute capacity to run Kubernetes based workloads is provided by EC2 | User launches EKS control plane and configures a fargate-profile and may also add EC2 to the cluster. Networking on Fargate use the same subnets configured with AWS CNI for EKS, where each Fargate pod is allocated with an ENI from the specified set of subnets. |
| Cost | Charge of $ 0.20/cluster/hr for managing the master node and worker nodes are charged at the regular EC2 instance costs | Only pay for the amount of vCPU and memory resources that pod needs to run. |
| Creating a cluster | eksctl create cluster \   --name *my-cluster* \   --without-nodegroup | eksctl create cluster \   --name *my-cluster* \   --fargate   -- alb-ingress-access  User needs to mention –fargate option and create an ingress controller |
| Fargate Profile | NA | Fargate profiles allow users to declare which pods to run on Fargate. This enables users to have workloads running on both Nodes (EC2 instances) and Fargate. At least one Fargate profile should be specified to determine pods that should use Fargate when they are launched. Fargate profiles are immutable, a new updated profile should be created to replace an existing profile followed by the deletion of the original profile. |
| Pod Execution Role | Kubernetes clusters managed by Amazon EKS make calls to other AWS services on your behalf to manage the resources that you use with the service. Before you can create Amazon EKS clusters, you must create an IAM role | When cluster creates pods on AWS Fargate infrastructure, the pod needs to make calls to AWS APIs on our behalf. The Amazon EKS pod execution role provides the IAM permissions to do this.  When you create a Fargate profile, you must specify a pod execution role to use with your pods. |
| Limitations |  | * There is a maximum of 4 vCPU and 30Gb memory per pod. * Currently there is no support for stateful workloads that require persistent volumes or file systems. * You cannot run Daemonsets, Privileged pods, or pods that use HostNetwork or HostPort. * The only load balancer you can use is an application load balancer |

## 

## Setup a Python/Java web application:

For this tutorial, we will setup a simple Python web application which can be found at [https:// git@github.com:starsaket/AWS\_Fargate.git](https://github.com/abhishekray07/fargate-web-app).

This web-app returns Hello, world! whenever a request is made.

#### Clone the repo:

First, we will clone the sample repo and use that as the application we want to deploy to the EKS cluster. This repository also contains the Kubernetes manifests required for our application.

git clone <https://git@github.com:starsaket/AWS_Fargate.git>

## Steps to Push an Image into Amazon ECR with Docker: -

### Amazon elastic container registry (ECR) is a managed AWS Docker registry service. In this topic, we will use the Docker CLI to push a CentOS image into Amazon ECR.

### 1. Install Docker desktop for Windows and AWS CLI

### Verify and confirm that each version has been installed properly:

### docker (dash dash) version

### aws (dash dash) version

### 2. Authentication to AWS

### Open Power Shell interface with administration privileges and enter the following commands:

### aws configure

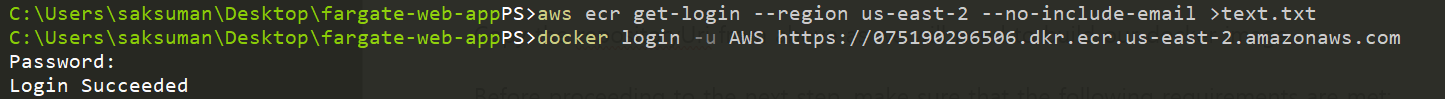
### 

### 3. Log in to AWS elastic container registry

### Use the get-login command to log in to AWS elastic container registry and save it to a text file (see below):

### aws ecr get-login (dash dash)region eu-west-3 --no-include-email > text.txt

### 4. Authenticate Docker to AWS elastic container registry:



### 5. Download/create the CentOS/Application image

### Use the pull command to download the CentOs image:

### docker pull centos:6.6

### Docker\_Pull\_image

### 6. Create a repository

### 

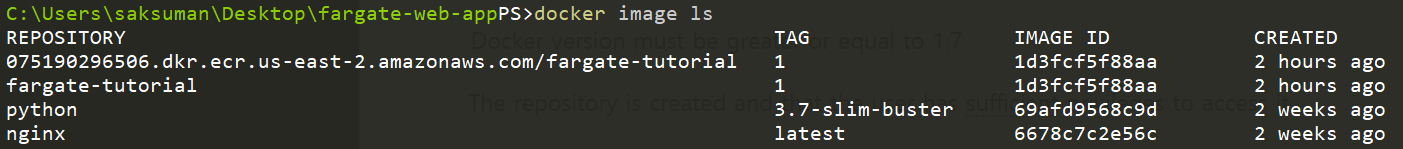
### Copy the repositoryUri from above as we will need to build our docker image.

### Before proceeding to the next step, make sure that the following requirements are met:

### Docker version must be greater or equal to 1.7

### The repository is created and that the user has sufficient privileges to access it

### 7. List the images stored into Docker and tag them

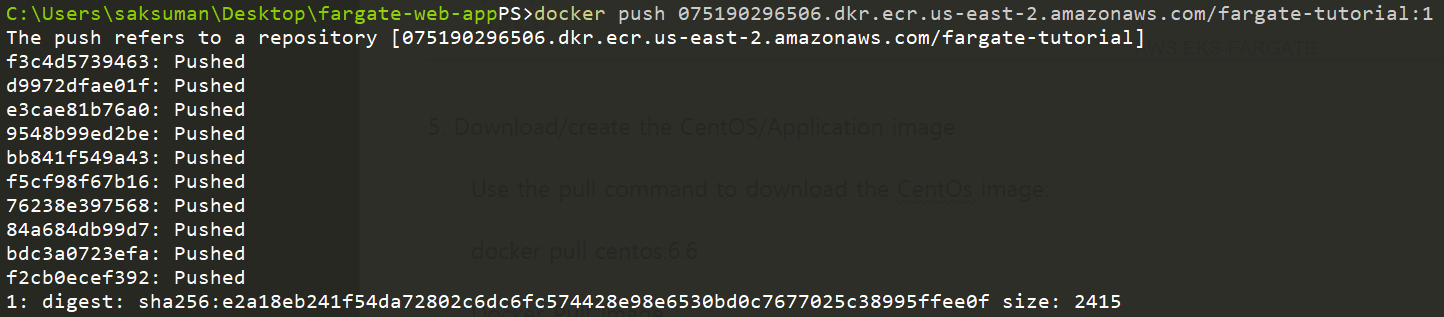


### Docker images

### docker build -t fargate-tutorial:1 .

### docker tag fargate-tutorial:1 945568844480.dkr.ecr.us-east-2.amazonaws.com/fargate-tutorial:1

### 8. Push the Application-image into Amazon ECR



## Setting up Full Cluster:

### Step1: To create a new EKS cluster with Fargate

The following command creates a new EKS cluster named aws-fargate-poc1 with Fargate enabled.

# creating an EKS cluster using eksctl-command line tool with existing VPC, Subnet and EKS Service Role

---

    apiVersion: eksctl.io/v1alpha5

    kind: ClusterConfig

    metadata:

      name: aws-fargate-poc1

      region: us-east-1

    vpc:

      id: "vpc-0c5fe1cf4e9832c08"  # (optional, must match VPC ID used for each subnet below)

      subnets:

          # must provide 'private' and/or 'public' subnets by availibility zone as shown

        private:

          us-east-1f:

            id: "subnet-014336c77037389d4"

          us-east-1b:

            id: "subnet-0675333d25444ff21"

        public:

          us-east-1f:

            id: "subnet-0dc5ce34a7d6df33a"

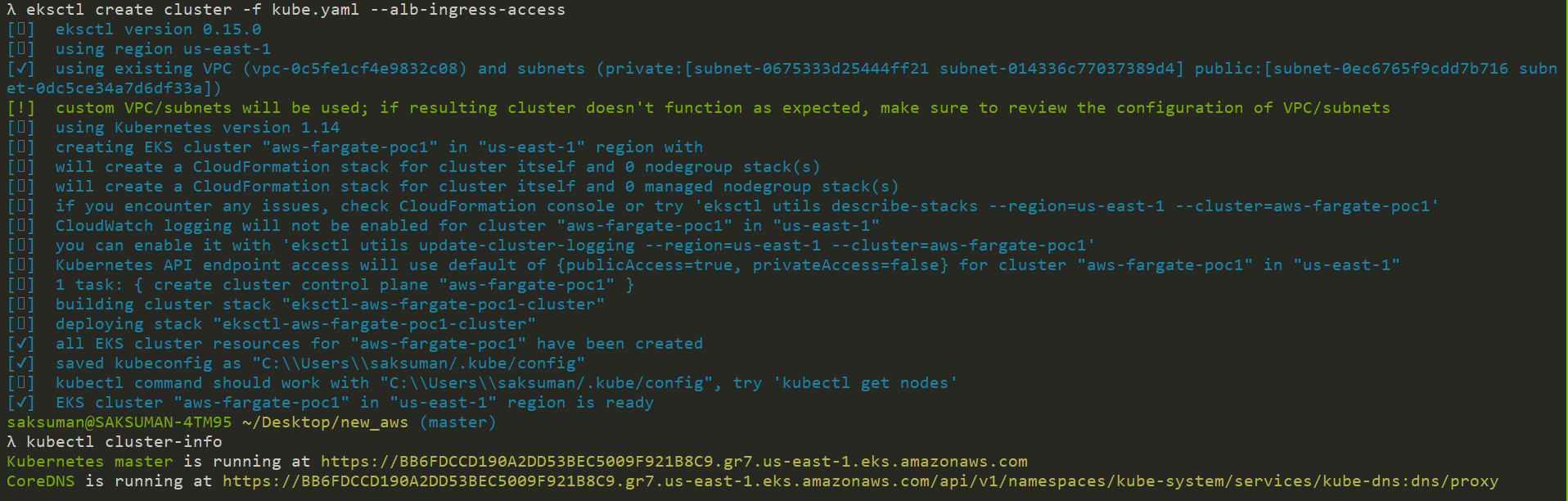
          us-east-1b:

            id: "subnet-0ec6765f9cdd7b716"

    iam:

      serviceRoleARN: "arn:aws:iam::616223086019:role/eksctl-aws-fargate-cluster-ServiceRole-Z760EP6JZN8X"

The Output Shown after execution: -

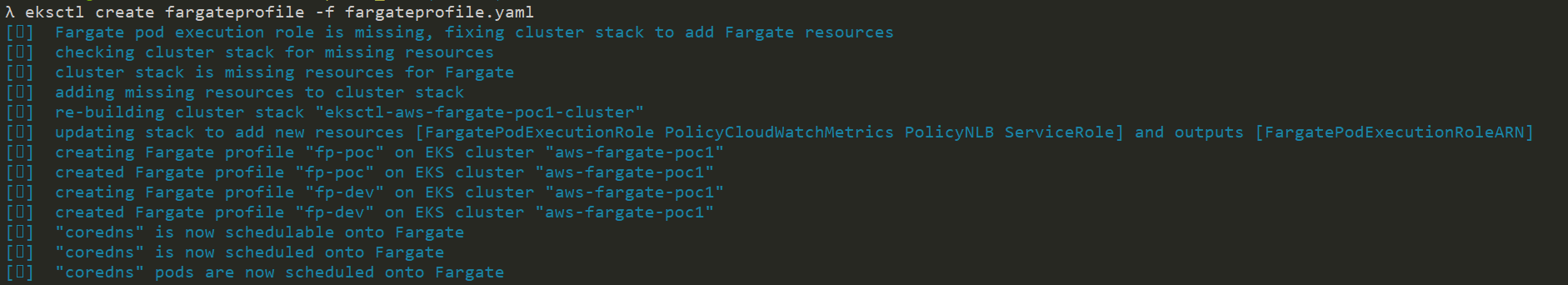


A deeper look into the command we ran above:

* name: Name of the cluster we want to create
* region: Only allowed regions for EKS on Fargate currently are: US East (N. Virginia), US East (Ohio), Europe (Ireland), and Asia Pacific (Tokyo)
* fargate: This creates a [Fargate profile](https://docs.aws.amazon.com/eks/latest/userguide/fargate-profile.html), which is used to run Kubernetes pods as Fargate tasks. This fargate profile has access to the default and kube-system namespaces
* ALB ingress access: EKS on Fargate only supports the Application Load Balancer (ALB) so this flag helps setup some of the scaffolding required to setup an ALB to work with EKS.

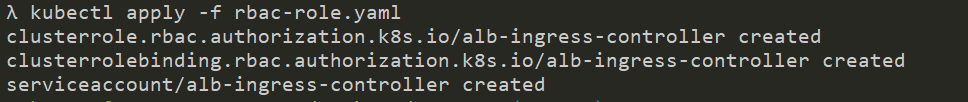
**Step2: creating the fargate-Profile**

* . # creating an EKS Fargate-profile with EKS cluster,Private-subnet and podExecutionRole
* apiVersion: eksctl.io/v1alpha5
* kind: ClusterConfig
* metadata:
* name: aws-fargate-poc1
* region: us-east-1
* fargateProfiles:
* - name: fp-poc
* podExecutionRoleARN: "arn:aws:iam::616223086019:role/eksctl-aws-fargate-cluster-FargatePodExecutionRole-I94J9F0JSOTJ"
* selectors:
* # All workloads in the "default" Kubernetes namespace will be
* # scheduled onto Fargate:
* - namespace: default
* # All workloads in the "kube-system" Kubernetes namespace will be
* # scheduled onto Fargate:
* - namespace: kube-system
* - name: fp-dev
* podExecutionRoleARN: "arn:aws:iam::616223086019:role/eksctl-aws-fargate-cluster-FargatePodExecutionRole-I94J9F0JSOTJ"
* selectors:
* # All workloads in the "dev" Kubernetes namespace matching the following
* # label selectors will be scheduled onto Fargate:
* - namespace: python-web
* labels:
* env: dev
* app: python



**Step 3: Creating a cluster role, ClusterRoleBinding and Service Account for Alb-ingress-controller**

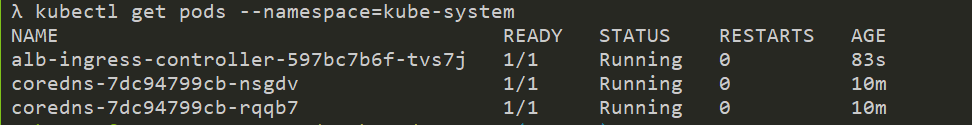
* ---
* apiVersion: rbac.authorization.k8s.io/v1
* kind: ClusterRole
* metadata:
* labels:
* app.kubernetes.io/name: alb-ingress-controller
* name: alb-ingress-controller
* rules:
* - apiGroups:
* - ""
* - extensions
* resources:
* - configmaps
* - endpoints
* - events
* - ingresses
* - ingresses/status
* - services
* - pods/status
* verbs:
* - create
* - get
* - list
* - update
* - watch
* - patch
* - apiGroups:
* - ""
* - extensions
* resources:
* - nodes
* - pods
* - secrets
* - services
* - namespaces
* verbs:
* - get
* - list
* - watch
* ---
* apiVersion: rbac.authorization.k8s.io/v1
* kind: ClusterRoleBinding
* metadata:
* labels:
* app.kubernetes.io/name: alb-ingress-controller
* name: alb-ingress-controller
* roleRef:
* apiGroup: rbac.authorization.k8s.io
* kind: ClusterRole
* name: alb-ingress-controller
* subjects:
* - kind: ServiceAccount
* name: alb-ingress-controller
* namespace: kube-system
* ---
* apiVersion: v1
* kind: ServiceAccount
* metadata:
* labels:
* app.kubernetes.io/name: alb-ingress-controller
* name: alb-ingress-controller
* namespace: kube-system



**Step 4: Application Load Balancer (ALB) Ingress Controller Deployment Manifest.**

* # This manifest details sensible defaults for deploying an ALB Ingress Controller.
* # GitHub: https://github.com/kubernetes-sigs/aws-alb-ingress-controller
* apiVersion: apps/v1
* kind: Deployment
* metadata:
* labels:
* app.kubernetes.io/name: alb-ingress-controller
* name: alb-ingress-controller
* # Namespace the ALB Ingress Controller should run in. Does not impact which
* # namespaces it's able to resolve ingress resource for. For limiting ingress
* # namespace scope, see --watch-namespace.
* namespace: kube-system
* spec:
* selector:
* matchLabels:
* app.kubernetes.io/name: alb-ingress-controller
* template:
* metadata:
* labels:
* app.kubernetes.io/name: alb-ingress-controller
* spec:
* containers:
* - name: alb-ingress-controller
* args:
* # Limit the namespace where this ALB Ingress Controller deployment will
* # resolve ingress resources. If left commented, all namespaces are used.
* # - --watch-namespace=your-k8s-namespace
* # Setting the ingress-class flag below ensures that only ingress resources with the
* # annotation kubernetes.io/ingress.class: "alb" are respected by the controller. You may
* # choose any class you'd like for this controller to respect.
* - --ingress-class=alb
* # REQUIRED
* # Name of your cluster. Used when naming resources created
* # by the ALB Ingress Controller, providing distinction between
* # clusters.
* - --cluster-name=Enter your EKS cluster name
* # AWS VPC ID this ingress controller will use to create AWS resources.
* # If unspecified, it will be discovered from ec2metadata.
* - --aws-vpc-id=Enter your VPC ID
* # AWS region this ingress controller will operate in.
* # If unspecified, it will be discovered from ec2metadata.
* # List of regions: http://docs.aws.amazon.com/general/latest/gr/rande.html#vpc\_region
* - --aws-region=Specify the reagion
* # Enables logging on all outbound requests sent to the AWS API.
* # If logging is desired, set to true.
* # - --aws-api-debug
* # Maximum number of times to retry the aws calls.
* # defaults to 10.
* # - --aws-max-retries=10
* env:
* # AWS key id for authenticating with the AWS API.
* # This is only here for examples. It's recommended you instead use
* # a project like kube2iam for granting access.
* - name: AWS\_ACCESS\_KEY\_ID
* value: ---------AWS\_ACCESS\_KEY\_ID------------
* # AWS key secret for authenticating with the AWS API.
* # This is only here for examples. It's recommended you instead use
* # a project like kube2iam for granting access.
* - name: AWS\_SECRET\_ACCESS\_KEY
* value: ---enter AWS\_SECRET\_ACCESS\_KEY--------
* # Repository location of the ALB Ingress Controller.
* image: docker.io/amazon/aws-alb-ingress-controller:v1.1.6
* serviceAccountName: alb-ingress-controller





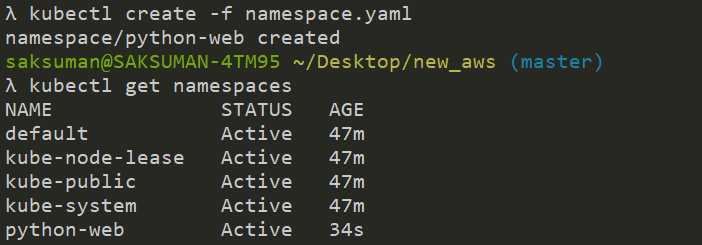
## Deploy Application to EKS Cluster:

There are 4 manifest files under the Kubernetes folder in the repository linked above. These files are:

* Namespace: Creates a new [Namespace](https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/) for our application
* Deployment: Creates a [Deployment](https://kubernetes.io/docs/concepts/workloads/controllers/deployment/) object for our application
* Service: Creates a [Service](https://kubernetes.io/docs/concepts/services-networking/service/) object for our application
* Ingress: Setup [ingress](https://kubernetes.io/docs/concepts/services-networking/ingress/) for the application so that it is accessible externally

**Step 5: Creating a Namespace:**

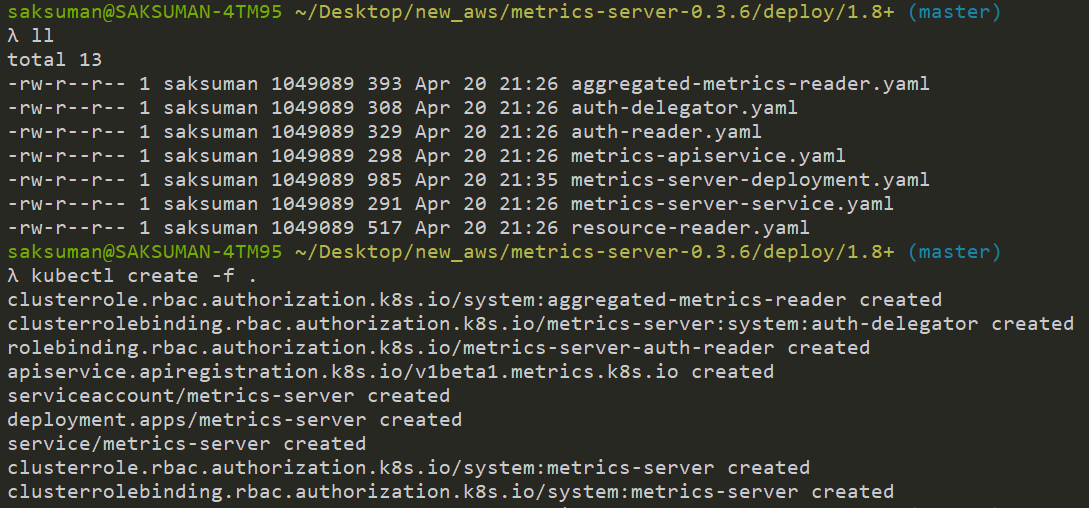
* # Creating a Namespace
* apiVersion: v1
* kind: Namespace
* metadata:
* name: python-web

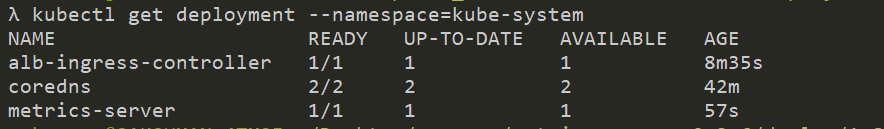


**Step 6: To deploy Matrix-Server**

For Matrix-deployment clone the following repo:

<https://github.com/kubernetes-sigs/metrics-server/releases/tag/v0.3.6>





**Step 7: Deployment config files for creating pod of running container from docker-hub**

apiVersion: apps/v1

kind: Deployment

metadata:

  name: python-web

  namespace: python-web

  labels:

    app: python-web

spec:

  replicas: 1

  selector:

    matchLabels:

      project: s3

  strategy: {}

  template:

    metadata:

      labels:

        project: s3

        env: dev

        app: python

    spec:

      containers:

      - name: python-web

        image: 1202680/awspoc:latest

        ports:

          - containerPort: 5000

        resources:

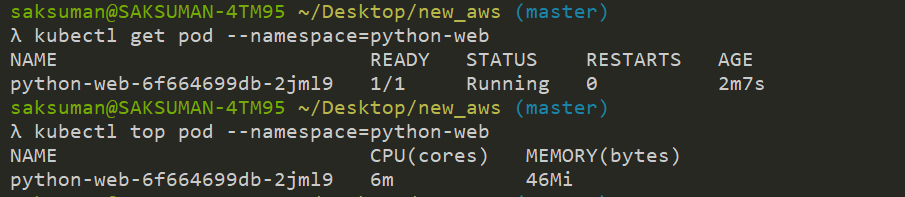
            limits:

              cpu: "200m"

            requests:

              cpu: "200m"





**Step 8: Creating a service type of cluster IP for exposing the deployment**

# Creating a service type of clusterIP for exposing the deployment

kind: Service

apiVersion: v1

metadata:

  name: python-web

  namespace: python-web

spec:

  selector:

    project: s3

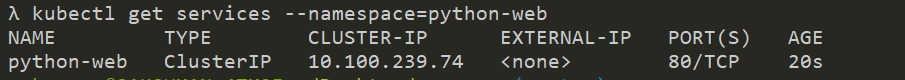
  ports:

  - protocol: TCP

    port: 80

    targetPort: 5000





**Step 9: Creating an ingress-resources for redirecting user traffic to our backend services**

# Creating an ingress-resources for redirecting user traffic to our backend services

apiVersion: extensions/v1beta1

kind: Ingress

metadata:

  name: python-web

  namespace: python-web

  annotations:

    kubernetes.io/ingress.class: alb

    alb.ingress.kubernetes.io/target-type: ip

    alb.ingress.kubernetes.io/scheme: internet-facing

spec:

  rules:

    - http:

        paths:

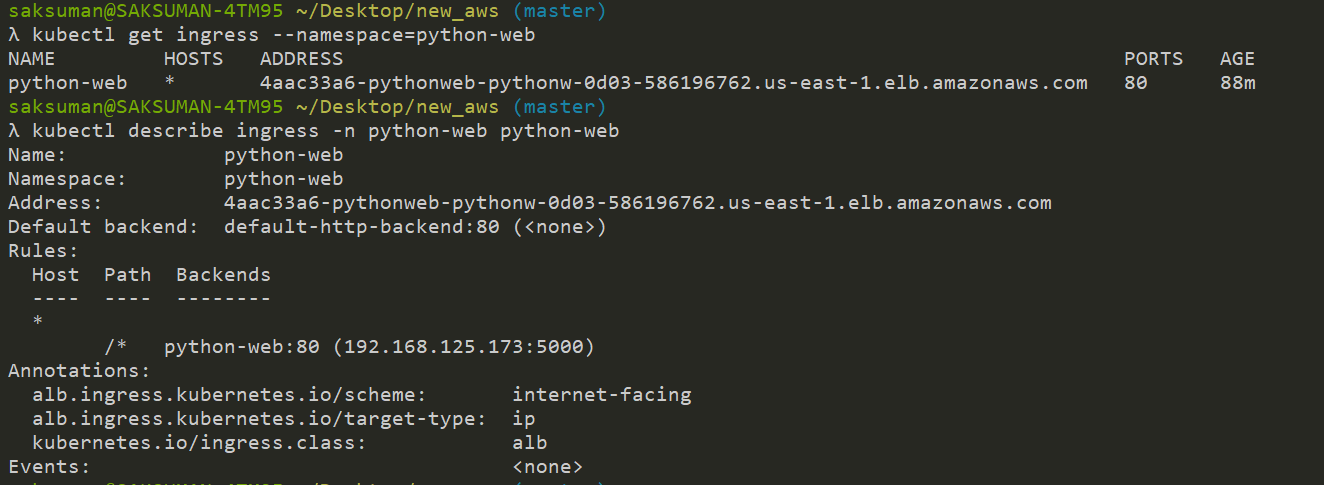
          - path: /\*

            backend:

              serviceName: python-web

              servicePort: 80





**Step 10: creating a hpa for autoscaling our deploymet based on CPU-percentage utilization**

# creating a hpa for autoscaling our deploymet based on CPU-percentage utilization

apiVersion: autoscaling/v1

kind: HorizontalPodAutoscaler

metadata:

  name: python-web

  namespace: python-web

spec:

  maxReplicas: 5

  minReplicas: 1

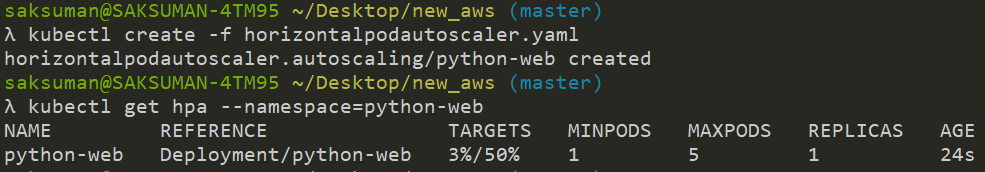
  scaleTargetRef:

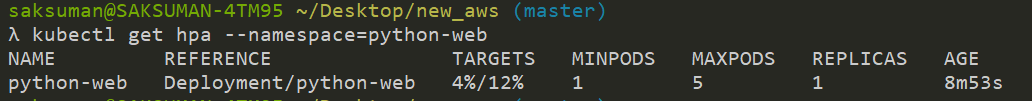
    apiVersion: apps/v1

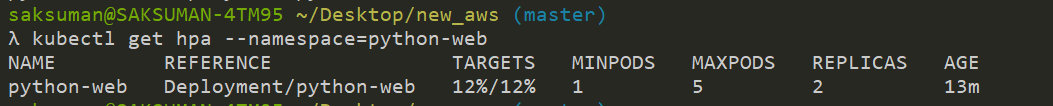
    kind: Deployment

    name: python-web

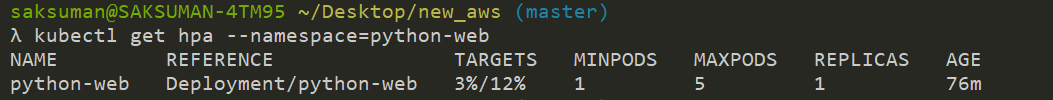
  targetCPUUtilizationPercentage: 12





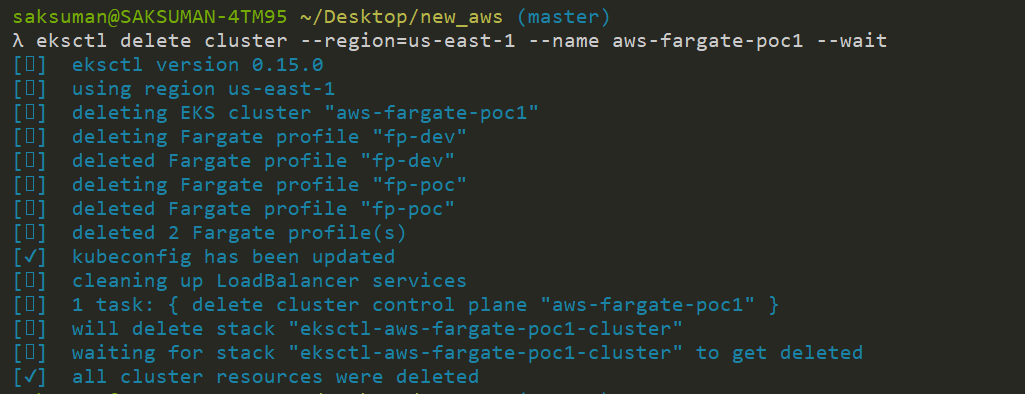


Here: Once the target reach 12% automatically new Replica started.



Here: Once the target reach 3% automatically new Replica get decreased.

**Step 11: To delete the Cluster**



**============================================================================================================================================================All the RUN-Commands: -**

* eksctl create cluster -f kube.yaml
* kubectl cluster-info
* kubectl get svc
* eksctl create fargateprofile -f fargateprofile.yaml
* eksctl get fargateprofile --cluster <Cluster\_name>
* kubectl get pods --namespace=kube-system
* kubectl create -f rbac-role.yaml
* kubectl create -f alb-ingress-controller.yaml
* kubectl describe deployment alb-ingress-controller --namespace=kube-system
* kubectl get pods --namespace=kube-system
* kubectl create -f .(for Metrics-Server)
* kubectl get pods --namespace=kube-system
* kubectl top pod --namespace= kube-system
* kubectl describe deployment metrics-server --namespace=kube-system
* kubectl create -f namespace.yaml
* kubectl create -f deployment.yaml
* kubectl describe deployment python-web --namespace=python-web
* kubectl get pods --namespace=python-web -o wide
* kubectl exec -it podname /bin/sh --namespace=namespacename
* kubectl create -f service.yaml
* kubectl get service --namespace=python-web
* kubectl create -f ingress-resources.yaml
* kubectl get ingress --namespace=python-web
* kubectl describe ing python-web -n python-web
* kubectl create -f horizontalpodautoscaler.yaml
* kubectl get hpa
* kubectl top pod --namespace=python-web
* kubectl get hpa --namespace=python-web -w
* kubectl describe hpa python-web --namespace=python-web
* kubectl describe deployment python-web --namespace=python-web
* kubectl get pods --namespace=python-web
* eksctl delete fargateprofile --cluster <Cluster\_name>--name<fargateprofile\_name> --wait
* eksctl delete cluster --region=us-east-1 --name <Cluster\_name> --wait

## Conclusion:

* With the addition of EKS on Fargate, Amazon has made it simpler to run Kubernetes applications.
* EKS and Fargate make it straightforward to run Kubernetes-based applications on AWS by removing the need to provision and manage infrastructure for pods.

## Reference: -

* https://eksctl.io/
* https://eksctl.io/usage/fargate/
* https://docs.aws.amazon.com/eks/latest/userguide/fargate-getting-started.html
* https://aws.amazon.com/blogs/aws/amazon-eks-on-aws-fargate-now-generally-available/
* https://github.com/weaveworks/eksctl/tree/master/examples
* https://docs.aws.amazon.com/eks/latest/userguide/metrics-server.html
* https://github.com/kubernetes-sigs/metrics-server/releases/tag/v0.3.6
* https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/
* https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale-walkthrough/
* https://kubernetes.io/docs/concepts/services-networking/ingress/
* https://kubernetes.io/docs/concepts/services-networking/ingress-controllers/
* https://docs.aws.amazon.com/eks/latest/userguide/alb-ingress.html
* https://www.youtube.com/watch?v=uxuyPru3\_Lc&t=989s
* https://www.youtube.com/watch?v=KS5MzK4EDg8&t=310s
* https://kubernetes.io/docs/concepts/services-networking/ingress-controllers/
* https://docs.aws.amazon.com/cli/latest/reference/ecr/get-login.html
* https://blog.dbi-services.com/how-to-push-an-image-into-amazon-ecr-with-docker/
* https://aws.amazon.com/blogs/aws/amazon-eks-on-aws-fargate-now-generally-available/
* <https://docs.aws.amazon.com/eks/latest/userguide/eks-ug.pdf>
* https://www.youtube.com/watch?v=m-3tMXmWWQws