AWS EKS Application Deployment Architecture/ Implementation

Kiran Sahoo, Jaswanth Kumar Jonnalagadda
Amazon Web Services

July 2019

© 2019 Amazon Web Services, Inc. or its affiliates. All rights reserved. This work may not be reproduced or redistributed, in whole or in part, without
prior written permission from Amazon Web Services, Inc. Commercial copying, lending, or selling is prohibited.
All trademarks are the property of their owner

Table of Contents

Overview: ***4***

End to End Implementation Architecture: ***4***

Amazon EKS: ***5***

Build & Deployment: ***5***

Pre-requisites: **5**

Launch Cloud9: 5

IAM Role for Workspace: 6

Install Kubernetes Tools: 7

Update IAM for Workspace: 8

Create SSH Key: 9

Download eksctl Binary: 9

Build EKS Cluster: **10**

Cluster Creation: 10

Application Deployment: 11

Security: ***11***

Managing Cluster Authentication: **12**

IAM Authentication: **12**

IAM Role for Cluster Interaction: **13**

Create IAM Role: 13

Add IAM Role to aws-auth configmap: 14

Application Deployment Automation: ***15***

Routing & SSL: ***16***

Domain Setup: **16**

Domain Registration: **16**

Certificate Authorization: **20**

Cloud Front Component: **22**

Lambda @Edge: **29**

Create IAM Role: 29

Create lambda @edge function 31

Logging(EFK Stack): ***34***

Fluentd: **34**

ElasticSearch: **34**

Kibana: **34**

Configure IAM Policy: 34

ElasticSearch Cluster: 35

Deploy FluentD: 35

CloudWatch Logs Subscription: 36

EKS Cluster Monitoring: *40*
Prerequisites: **40**
Deploy Prometheus: **41**
Deploy Grafana: **42**
Autoscaling: *43*
Prerequisites: **43**
Deploy the Metric Sever: **43**
Deploying App with Horizontal Auto Scaling: 45
Configure Cluster Autoscaler: **46**
Download Cluster Autoscaler script: 46
Configure Autoscaling Group: 46
Create IAM Policy: 47
Deploy Cluster Autoscaler: 48
Web Application Firewall(WAF): *49*
Performance & Sizing: *50*
Conclusion: *50*

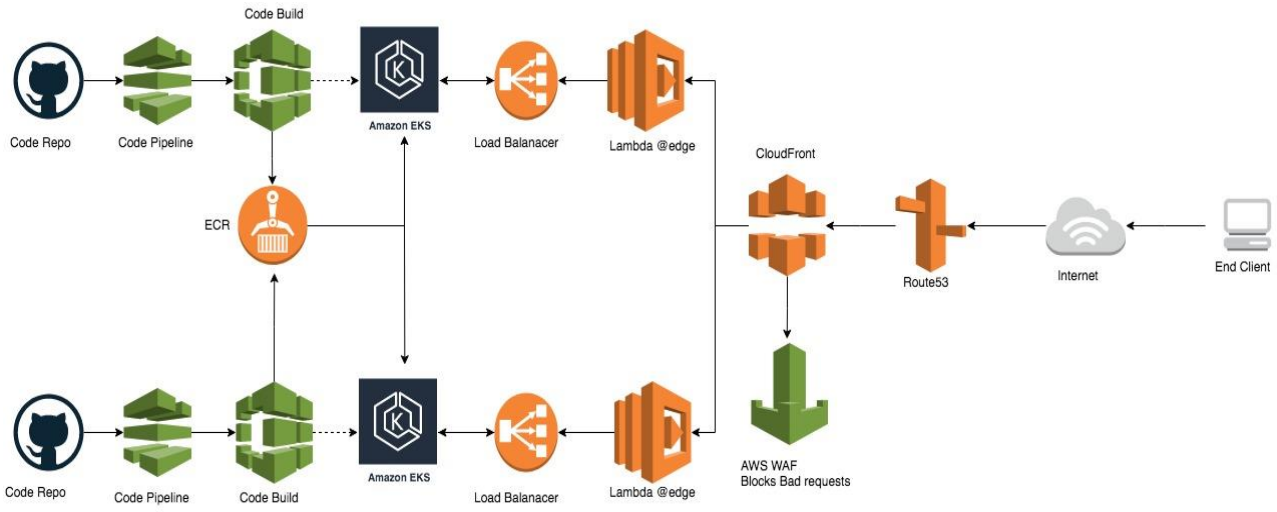
Overview:

This document provides a walkthrough of the App deployment architecture, implementation and automation using the following services

- Amazon Elastic Container Service for Kubernetes(Amazon EKS)
- AWS CloudFront
- AWS Lambda @Edge
- Amazon Route 53
- Amazon Certificate Manager(ACM)
- AWS Code Pipeline
- AWS CloudFormation
- AWS Cloud9
- EFK Stack
- Prometheus/Grafana
- AWS WAF

End to End Implementation Architecture:

As part of initial phase - Client decided to deploy one application per EKS cluster to achieve faster migration, easy isolation and reduction in debugging and deployment complexity.



Amazon EKS:

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

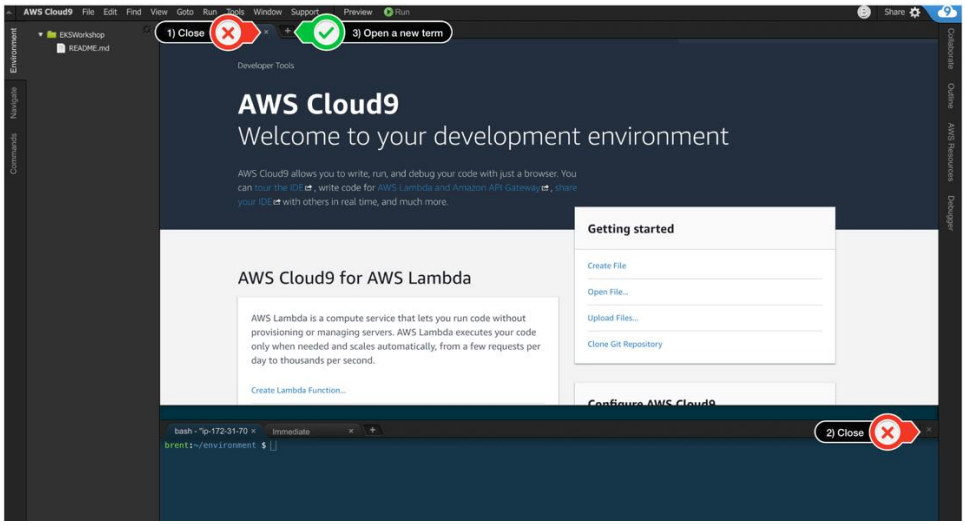
Amazon EKS runs the Kubernetes management infrastructure across multiple AWS availability zones to eliminate a single point of failure. Amazon EKS is certified Kubernetes conformant so that one can use existing tooling and plugins from partners and the Kubernetes community. Applications running on any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

Build & Deployment:

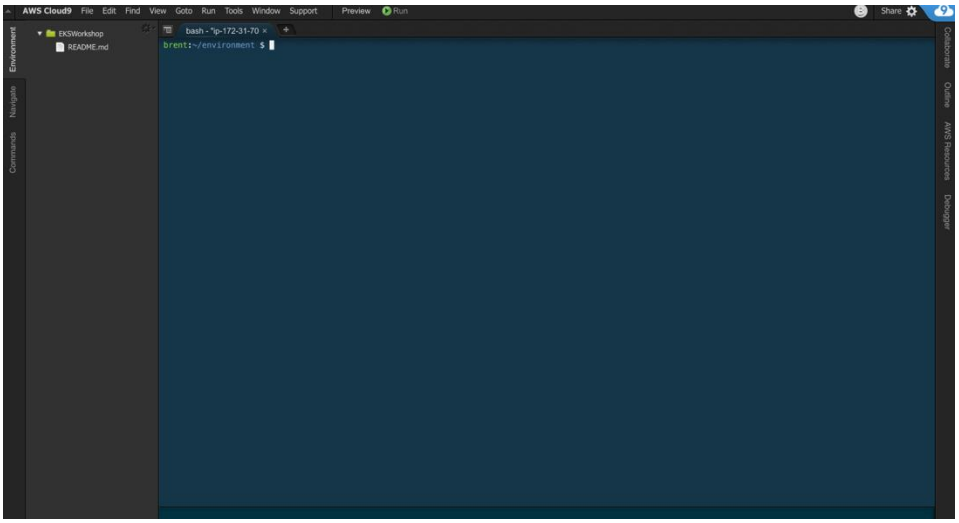
Pre-requisites:

Launch Cloud9:

- Login to AWS Console
- Name it **awapps_cloud9**, and take all other defaults
- When it comes up, customize the environment by closing the **welcome tab** and **lower work area**, and opening a new **terminal** tab in the main work area:

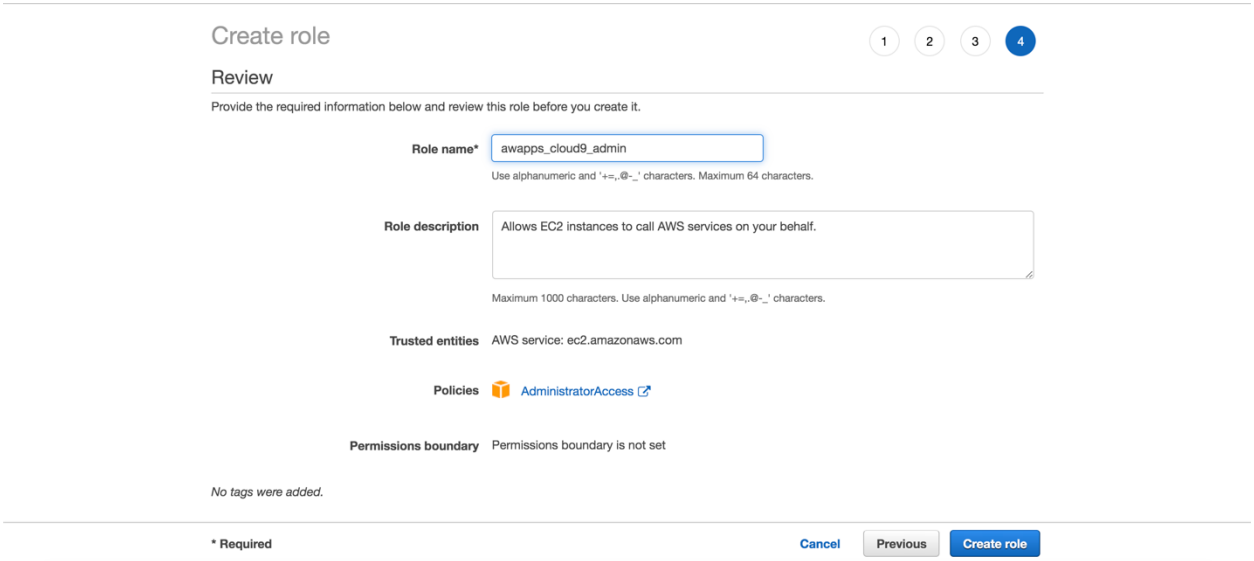


- Workspace should now look like this:

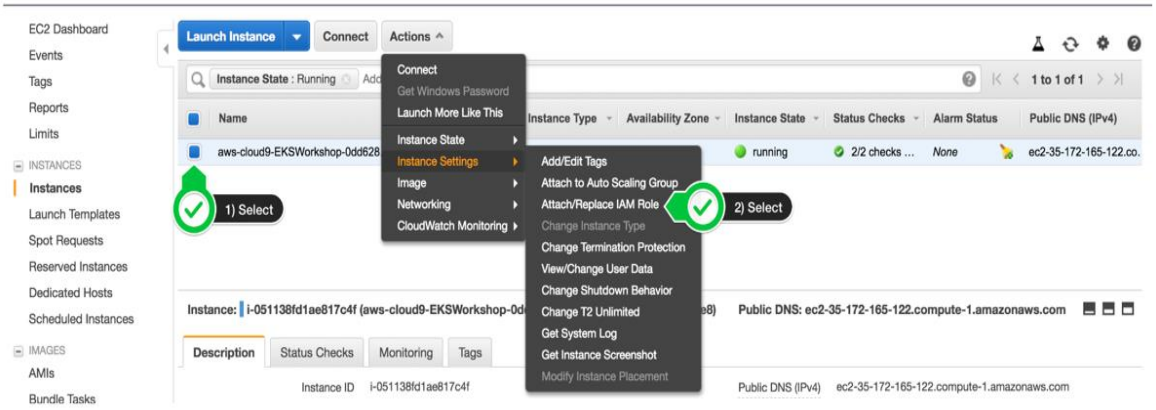


IAM Role for Workspace:

- Follow [this deep link to create an IAM role with Administrator access.]
- Confirm that **AWS service** and **EC2** are selected, then click **Next** to view permissions
- Confirm that **Administrator Access** is checked, then click **Next** to review
- Enter **awsapps_cloud9_admin** for the Name, and select **Create Role**



- Follow [this deep link to find your Cloud9 EC2 instance]
- Select the instance, then choose **Actions / Instance Settings / Attach/Replace IAM Role**



- Choose **awsapps_cloud9_admin** from the **IAM Role** drop down, and select **Apply**

Install Kubernetes Tools:

Create the default ~/.kube directory for storing kubectl configuration

- mkdir -p ~/.kube

install kubectl

- sudo curl --silent --location -o /usr/local/bin/kubectl https://amazon-eks.s3-us-west-2.amazonaws.com/1.12.7/2019-03-27/bin/linux/amd64/kubectl
- sudo chmod +x /usr/local/bin/kubectl

Install AWS IAM Authenticator

- go get -u -v github.com/kubernetes-sigs/aws-iam-authenticator/cmd/aws-iam-authenticator
- sudo mv ~/go/bin/aws-iam-authenticator /usr/local/bin/aws-iam-authenticator

Install JQ and envsubst

- sudo yum -y install jq gettext

Verify the binaries are in the path and executable

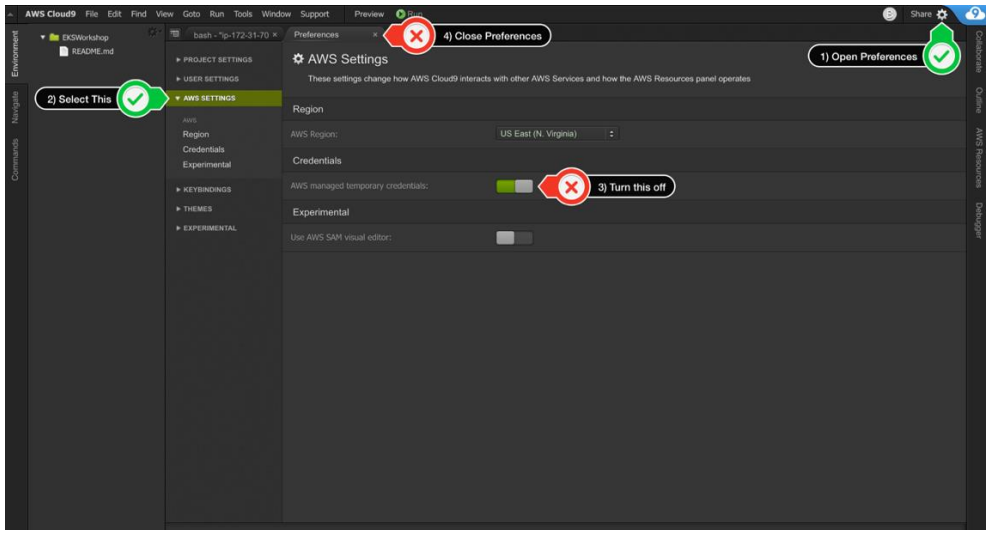
- for command in kubectl aws-iam-authenticator jq envsubst
  do
  which $command &>/dev/null && echo "$command in path" || echo "$command NOT FOUND"
  done

Note: To update the kubectl to a specific version follow below steps

- curl -LO https://storage.googleapis.com/kubernetes-release/release/<**specific-kubectl-version**>/bin/darwin/amd64/kubectl
- Replace <specific-kubectk-version> with the version required eg: 1.11.5
- chmod +x ./kubectl
- sudo mv ./kubectl $(which kubectl)

Update IAM for Workspace:

- Return to workspace and click the sprocket, or launch a new tab to open the Preferences tab
- Select **AWS SETTINGS**
- Turn off **AWS managed temporary credentials**
- Close the Preferences tab



- Removing any credentials that are already in place.
  - rm -vf ${HOME}/.aws/credentials
- Configure aws cli with the current region as default
  - export ACCOUNT_ID=$(aws sts get-caller-identity --output text --query Account)
  - export AWS_REGION=$(curl -s 169.254.169.254/latest/dynamic/instance-identity/document | jq -r '.region')
  - echo "export ACCOUNT_ID=${ACCOUNT_ID}" >> ~/.bash_profile
  - echo "export AWS_REGION=${AWS_REGION}" >> ~/.bash_profile
  - aws configure set default.region ${AWS_REGION}
  - aws configure get default.region
- Validate IAM Role
  - aws sts get-caller-identity

  Valid response:

```
{
        "Account": "123456789012",
        "UserId": "AROA1SAMPLEAWSIAMROLE:i-01234567890abcdef",
        "Arn": "arn:aws:sts::123456789012:assumed-role/awapps_cloud9_admin /i-
        01234567890abcdef"
}
or
{
        "Account": "123456789012",
        "UserId": "AROA1SAMPLEAWSIAMROLE:i-01234567890abcdef",
        "Arn": "arn:aws:sts::123456789012:assumed-role/TeamRole/i-01234567890abcdef
}
```

Create SSH Key:

- Generate SSH Key in Cloud9
  - ssh-keygen
- Update the public key to Ec2 Region
  - aws ec2 import-key-pair --key-name "awapps" --public-key-material file://~/.ssh/id_rsa.pub

Download eksctl Binary:

- Download eksctl
  - curl --silent --location "https://github.com/weaveworks/eksctl/releases/download/latest_release/eksctl_$(uname -s)_amd64.tar.gz" | tar xz -C /tmp

  - sudo mv -v /tmp/eksctl /usr/local/bin
- Verify
  - eksctl version

Build EKS Cluster:

Cluster Creation:

eksctl is a powerful CLI tool for creating clusters on EKS - Amazon's new managed Kubernetes service for EC2. It is written in Go, and uses CloudFormation. One can create a cluster in minutes with just one command – **eksctl create cluster.**

To deploy a custom configured cluster with eksctl write a custom cluster creation yaml file and apply it while creating the cluster.

Sample eksClusterCreation.yaml file to build a cluster inside a VPC, deploying the worker nodes in both private and public subnets.

```yaml
apiVersion: eksctl.io/v1alpha4
kind: ClusterConfig

metadata:
  name: majoraccounts1
  region: us-east-1

vpc:
  id: "vpc-01b579cdbb7bf5d94"  # (optional, must match VPC ID used for each subnet below)
  cidr: "10.3.0.0/16"       # (optional, must match CIDR used by the given VPC)
  subnets:
    # must provide 'private' and/or 'public' subnets by availibility zone as shown
    public:
      us-east-1a:
        id: "subnet-024224a58b66cd09f"
        cidr: "10.3.20.0/24" # (optional, must match CIDR used by the given subnet)

      us-east-1b:
        id: "subnet-0255646f670cb107e"
        cidr: "10.3.21.0/24"  # (optional, must match CIDR used by the given subnet)

    private:
      us-east-1a:
        id: "subnet-0c2f7abf9e68baa91"
        cidr: "10.3.10.0/24" # (optional, must match CIDR used by the given subnet)

      us-east-1b:
        id: "subnet-0c8ac3a16b0849fa1"
        cidr: "10.3.11.0/24"  # (optional, must match CIDR used by the given subnet)

iam:
  serviceRoleARN: "arn:aws:iam::682651395775:role/eksServiceRole"


nodeGroups:
  - name: majoraccounts1-public-workloads
    instanceType: m5.xlarge
    labels:
      nodegroup-type: frontend-workloads
    desiredCapacity: 2
  - name: majoraccounts1-private-workloads
    instanceType: m5.large
    desiredCapacity: 2
    labels:
      nodegroup-type: backend-workload
    privateNetworking: true
```

Meta-data for the cluster like name and region to be deployed on

VPC, cidr block and the subnets in which the cluster nodes to be deployed on are mentioned here

Label node groups to separate front end and backend workloads. Also mention the type of ec2 instances depending on the work load

Deploy the cluster with the command "**eksctl create cluster -f eksClusterCreation.yml**"

Application Deployment:

Once the cluster is up and active, docker image of the application can be deployed on to the nodes of the application. A load balancer also needs to be deployed to access the application running inside the pods.
A sample deployment file

```yaml
apiVersion: v1
kind: Service
metadata:
  name: majoraccounts1
  annotations:
    service.beta.kubernetes.io/aws-load-balancer-ssl-cert: "arn:aws:acm:us-east-1:682651395775:certificate/ebeac7f9-8167-4680-9f72-0a23a67eec99"
    service.beta.kubernetes.io/aws-load-balancer-backend-protocol: "http"
    service.beta.kubernetes.io/aws-load-balancer-ssl-ports: "443"
    service.beta.kubernetes.io/aws-load-balancer-connection-draining-enabled: "true"
    service.beta.kubernetes.io/aws-load-balancer-connection-draining-timeout: "60"
    service.beta.kubernetes.io/aws-load-balancer-additional-resource-tags: "environment=dev,app=app"
spec:
  selector:
    app: app
  type: LoadBalancer
  ports:
  - name: http
    port: 80
    targetPort: 8080
    protocol: TCP
  - name: https
    port: 443
    protocol: TCP
---
apiVersion: majoraccounts1/v1
kind: Deployment
metadata:
  name: majoraccounts1
  labels:
    app: majoraccounts1
  namespace: default
spec:
  replicas: 2
  selector:
    matchLabels:
      app: majoraccounts1
  strategy:
    rollingUpdate:
      maxSurge: 25%
      maxUnavailable: 25%
    type: RollingUpdate
  template:
    metadata:
      labels:
        app: app
    spec:
      containers:
        - image: CONTAINER_IMAGE
          imagePullPolicy: Always
          name: app
          ports:
            - containerPort: 8080
```

This part will deploy a load balancer for the application with ssl-cert i.e., https

This part of the script will deploy the docker image of the application over nodes with Rolling update enabled

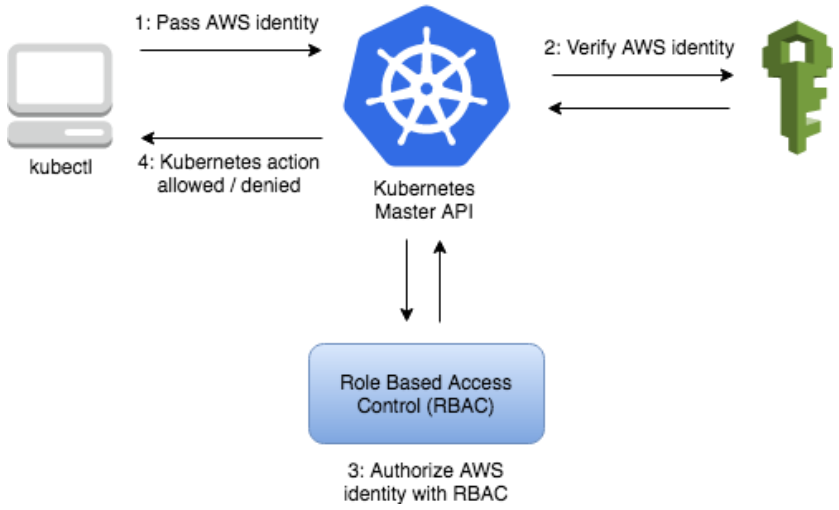
This will deploy a container image from ECR and run the application inside the worker nodes. This deployment file will also deploy a load balancer to access the application running inside the cluster.

"kubectl apply -f deployment.yml" command will deploy the application and load balancer

Security:

Managing Cluster Authentication:

Amazon EKS uses IAM to provide authentication to your Kubernetes cluster (through the **aws eks get-token**command, available in version 1.16.156 or greater of the AWS CLI, or the AWS IAM Authenticator for Kubernetes), but it still relies on native Kubernetes Role Based Access Control (RBAC) for authorization. This means that IAM is only used for authentication of valid IAM entities. All permissions for interacting with your Amazon EKS cluster's Kubernetes API is managed through the native Kubernetes RBAC system.



IAM Authentication:

- Download aws-iam-authenticator binary from Amazon S3

  curl -o aws-iam-authenticator https://amazon-eks.s3-us-west-2.amazonaws.com/1.12.7/2019-03-27/bin/linux/amd64/aws-iam-authenticator

- Apply execute permission to the binary

  chmod +x ./aws-iam-authenticator

- Copy the binary to a folder in your $PATH. We recommend creating a$HOME/bin/aws-iam-authenticator and ensuring that$HOME/bin comes first in your $PATH.

  mkdir -p $HOME/bin && cp ./aws-iam-authenticator $HOME/bin/aws-iam-authenticator && export PATH=$HOME/bin:$PATH

- Add $HOME/bin to your PATH environment variable.

  echo 'export PATH=$HOME/bin:$PATH' >> ~/.bashrc

- Test aws-iam-authenticator

  aws-iam-authenticator help

IAM Role for Cluster Interaction:

As a part of best practice create an IAM Role capable of interacting with EKS cluster. Same role can also be used in code build stage for interacting with EKS cluster via kubectl

Create IAM Role:

Sample script for creating an IAM role for interacting with EKS cluster

```
#!/usr/bin/env bash

ACCOUNT_ID=$(aws sts get-caller-identity --query Account --output text)

TRUST="{ \"Version\": \"2012-10-17\", \"Statement\": [ { \"Effect\": \"Allow\", \"Principal\": { \"AWS\": \"arn:aws:iam::$ACCOUNT_ID:root\" }, \"Action\": \"sts:AssumeRole\" } ] }"

TEMP='/tmp/iam-role-policy'

echo '{ "Version": "2012-10-17", "Statement": [ { "Effect": "Allow", "Action": "eks:Describe*", "Resource": "*" } ] }' > $TEMP

ROLE='EKSKubectlRole'

POLICY='eks-describe'

aws iam create-role --role-name $ROLE --assume-role-policy-document "$TRUST" --output text --query 'Role.Arn'

aws iam get-role --role-name $ROLE --output table

aws iam put-role-policy --role-name $ROLE --policy-name $POLICY --policy-document file://$TEMP

aws iam get-role-policy --role-name $ROLE --policy-name $POLICY --output table
```

Add IAM Role to aws-auth configmap:

Use the below sample script to add the above created role to configmap

```
#!/usr/bin/env bash

ACCOUNT_ID=$(aws sts get-caller-identity --query Account --output text)

ROLE="- rolearn: arn:aws:iam::$ACCOUNT_ID:role/EKSDeepDiveCodeBuildKubectlRole\n
username: build\n     groups:\n       - system:masters"

kubectl get -n kube-system configmap/aws-auth -o yaml | awk "/mapRoles: \|/{print;print
\"$ROLE\";next}1" > /tmp/aws-auth-patch.yml

kubectl patch configmap/aws-auth -n kube-system --patch "$(cat /tmp/aws-auth-patch.yml)"
```

Use the AWS CLI **update-kubeconfig** command to create or update your kubeconfig for your cluster.

- By default, the resulting configuration file is created at the default kubeconfig path (.kube/config) in your home directory or merged with an existing kubeconfig at that location. You can specify another path with the --kubeconfig option.
- You can specify an IAM role ARN with the --role-arn option to use for authentication when you issue **kubectl** commands. Otherwise, the IAM entity in your default AWS CLI or SDK credential chain is used. You can view your default AWS CLI or SDK identity by running the **aws sts get-caller-identity** command.
- For more information, see the help page with the **aws eks update-kubeconfig help** command or see update-kubeconfig in the *AWS CLI Command Reference*.

  aws eks --region region update-kubeconfig --name cluster_name --role-arn "Role_arn_created_above"

  Note: Replace region and cluster_name with respective values

To Test your configuration, use the below command

kubectl get svc

Sample Output looks like this

Output:

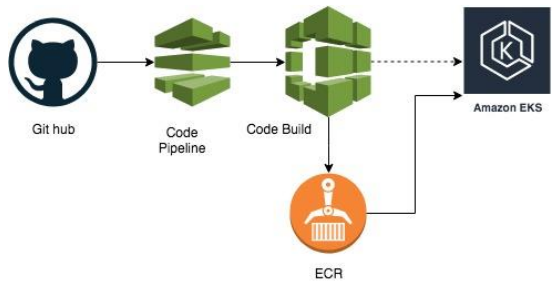
```
NAME              TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
svc/kubernetes    ClusterIP   10.100.0.1   <none>        443/TCP   1m
```

Application Deployment Automation:

Code Pipeline:

AWS Code Pipeline is a fully managed continuous delivery service that helps you automate your release pipelines for fast and reliable application and infrastructure updates. Code Pipeline automates the build, test, and deploy phases of your release process every time there is a code change, based on the release model you define. This enables you to rapidly and reliably deliver features and updates

Architecture:



To move the application from code repository to EKS cluster, implement AWS code pipeline integrated with Code Build and ECR. One can deploy a complete code pipeline with a cloud formation template.

NOTE: For the template please replace the placeholders with appropriate parameter values

To build the docker image from the code repository and push the built image to ECR repository to deploy the same on the deployed EKS cluster we integrate code build with **builspec.yml** file

```yaml
---
version: 0.2
phases:
  install:
    commands:
      - curl -sS -o aws-iam-authenticator https://amazon-eks.s3-us-west-2.amazonaws.com/1.10.3/2018-07-26/bin/linux/amd64/aws-iam-authenticator
      - curl -sS -o kubectl https://amazon-eks.s3-us-west-2.amazonaws.com/1.10.3/2018-07-26/bin/linux/amd64/kubectl
      - chmod +x ./kubectl ./aws-iam-authenticator
      - export PATH=$PWD/:$PATH
      - apt-get update && apt-get -y install jq python3-pip python3-dev && pip3 install --upgrade awscli
      - echo installing maven...
      - apt-get update -y
      - apt-get install software-properties-common
      - add-apt-repository ppa:openjdk-r/ppa
      - apt-get update
      - apt-get install -y openjdk-8-jdk
      - apt-get install -y maven
  pre_build:
      commands:
        - TAG="$REPOSITORY_NAME.$REPOSITORY_BRANCH.$ENVIRONMENT_NAME.$(date +%Y-%m-%d.%H.%M.%S).$(echo $CODEBUILD_RESOLVED_SOURCE_VERSION | head -c 8)"
        - sed -i 's@CONTAINER_IMAGE@'"$REPOSITORY_URI:$TAG"'@' appdeploy.yml
        - $(aws ecr get-login --no-include-email)
        - export KUBECONFIG=$HOME/.kube/config
  build:
    commands:
      - echo Entering build phase...
      - echo Build started on `date`
      - mvn clean package
      - docker image build --tag $REPOSITORY_URI:$TAG .

  post_build:
    commands:
      - echo Entering post_build phase...
      - echo Build completed on `date`i
      - docker push $REPOSITORY_URI:$TAG
      - CREDENTIALS=$(aws sts assume-role --role-arn $EKS_KUBECTL_ROLE_ARN --role-session-name codebuild-kubectl --duration-seconds 900)
      - export AWS_ACCESS_KEY_ID="$(echo ${CREDENTIALS} | jq -r '.Credentials.AccessKeyId')"
      - export AWS_SECRET_ACCESS_KEY="$(echo ${CREDENTIALS} | jq -r '.Credentials.SecretAccessKey')"
      - export AWS_SESSION_TOKEN="$(echo ${CREDENTIALS} | jq -r '.Credentials.SessionToken')"
      - export AWS_EXPIRATION=$(echo ${CREDENTIALS} | jq -r '.Credentials.Expiration')
      - aws eks update-kubeconfig --name $EKS_CLUSTER_NAME
      - kubectl apply -f appdeploy.yml
      - printf '[{"name":"appdeploy","imageUri":"%s"}]' $REPOSITORY_URI:$TAG > build.json
artifacts:
  files:
    - 'target/SampleServlet.war'
    - '**/*'
```

Part of script to download all the dependencies and command build the application

Commands to be executed before building the image

Commands to build the docker image form the app build

Command after build, pushing the image to ECR, deploying the app updates on to the cluster

NOTE: "EKS_KUBECTL_ROLE_ARN" in the post build phase need to be exported as a part of environment variables

Routing & SSL:

Domain Setup:

A new domain can be setup in two ways.

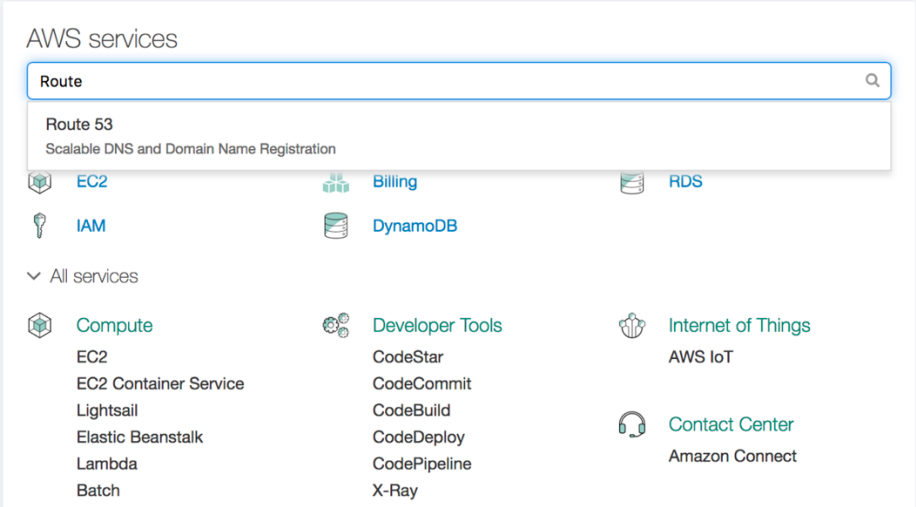
1. New Domain by Route 53
2. Third Party Domain vendor (eg: GoDaddy.com)

Domain Registration:

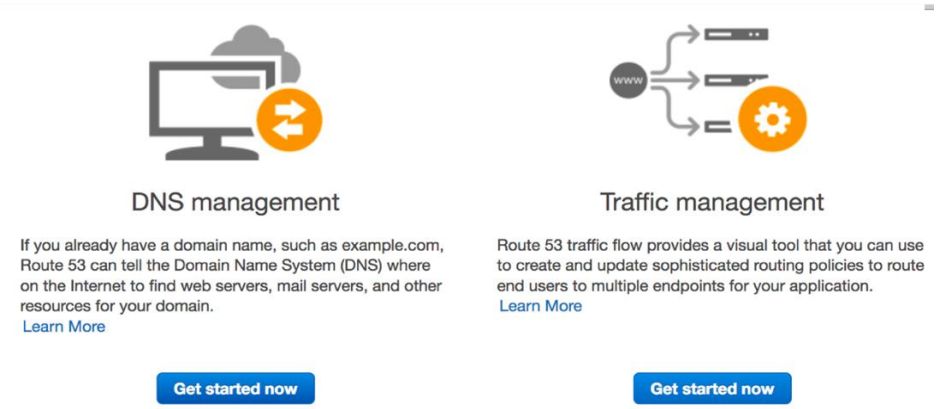
Domains created with Route 53 are automatically available with Route 53 as DNS server. Third party domain names need to be registered with Route 53 to utilize certificates issued by Amazon certificate manager.

Domains registered with GoDaddy can enable Route53 as DNS server with following steps:

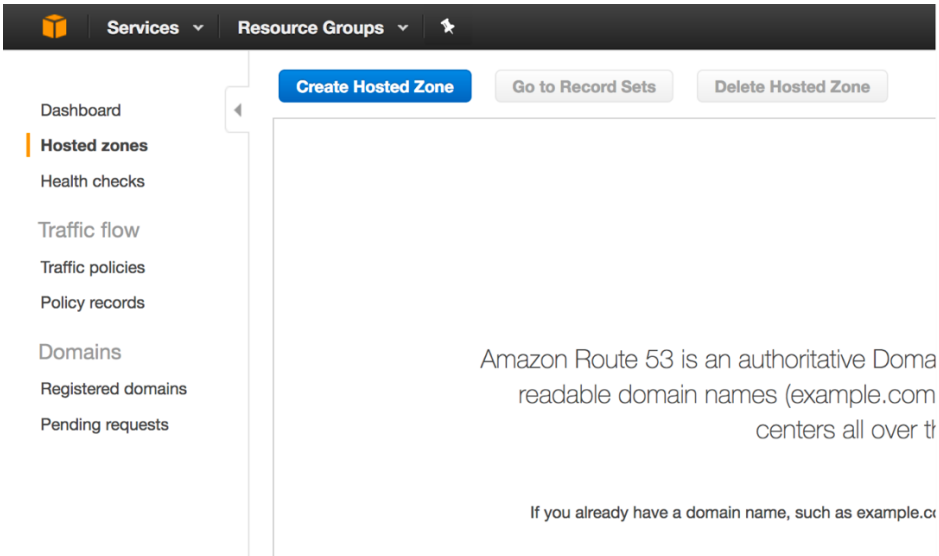
1. Search for Route 53 from the home page



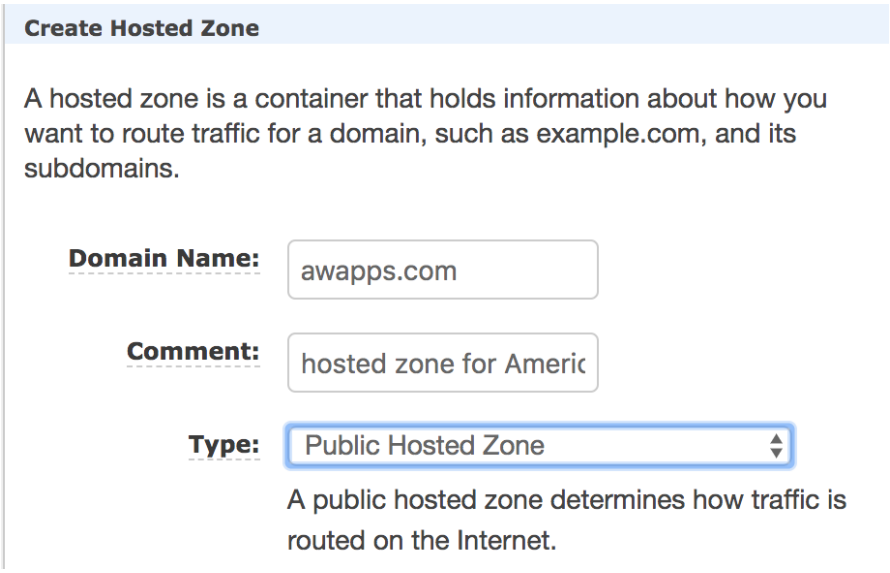
2. Click the Get Started Now under the DNS Management section.



3. Click the **Create Hosted Zone** Button.

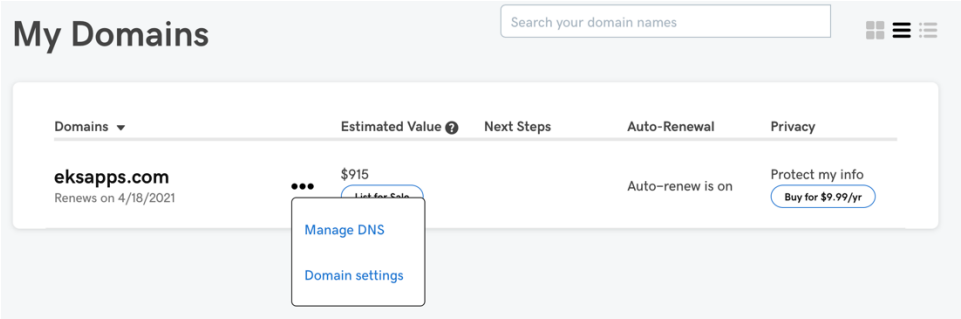


4. A form will show up in the right panel. Enter your domain name (skip the `www`), and an optional description. Keep the `Public Hosted Zone` selected. Click **Create**.

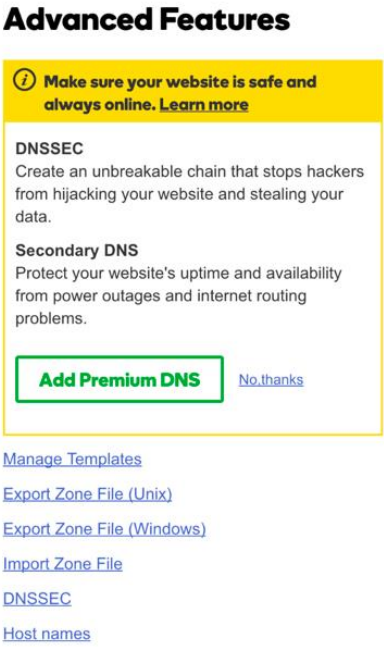


5. Once done, one can see the zone details, with only an `NS` record set and a `SOA` record set. `NS` has a list of name servers, and `SOA` is the `Start of Authority` record, which at a minimum contains the primary name server host name. Notice this is usually the first name server address from the `NS` record set.

6. Next, add our record sets from GoDaddy to this hosted zone. To do this, login to GoDaddy, click on **Manage DNS** under the **Domain** being moved.

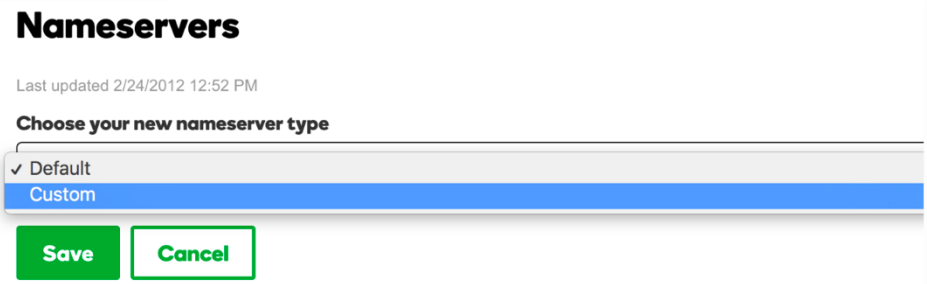


7. Scroll all the way down and notice a section called Advanced Features. Click the link that says Export Zone File (Unix).

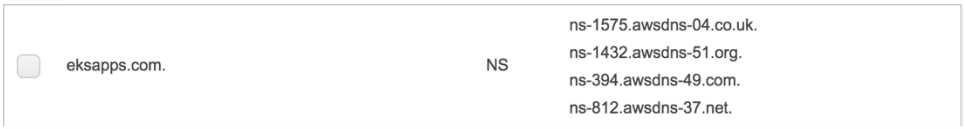


8. Notice a file `your_domain_name.txt` that has been downloaded. Open that file in any text editor of your choice.
9. Now come back to the Route 53 page with the Hosted Zone, and click Import Zone File at the top. A form will open up on the right. Copy and paste the contents of the `your_domain_name.txt` file in this section, and click Import button.
10. Click refresh at the top right or just refresh your browser and you'll notice a much bigger list of record sets in the hosted domain. These have been imported from the GoDaddy Zone File.
11. Now, one just need to point the Name Servers listed in GoDaddy for our Domain to Amazon's. To do this, go to GoDaddy, and just above Advanced Features, look for a section called Nameservers.
12. Click change

13. Select Custom



14. Go back to **Route 53**, and Notice the **NS** type record in the Route53 record set list.



15. Copy paste the name servers from the **NS** Type recordset one by one in the GoDaddy custom list. One need to click **Add Nameserver** twice to add 2 more rows for a total of 4 Amazon Nameservers.

By following all the steps, one can enable Route 53 as DNS server for the Domain created by third party vendor.

Certificate Authorization:

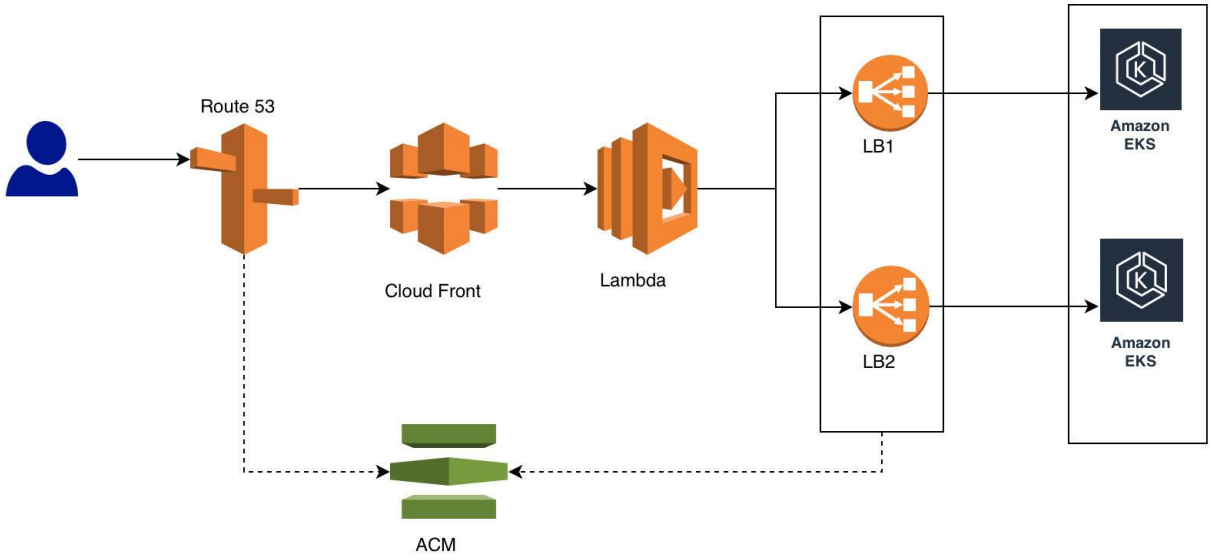
AWS Certificate Manager is a service that lets you easily provision, manage, and deploy public and private Secure Sockets Layer/Transport Layer Security (SSL/TLS) certificates for use with AWS services and your internal connected resources. SSL/TLS certificates are used to secure network communications and establish the identity of websites over the Internet as well as resources on private networks.

1. Sign into the AWS Management Console and open the ACM console at https://console.aws.amazon.com/acm/home.
2. On the **Request a certificate** page, type your domain name. One can use a fully qualified domain name (FQDN) such as **www.awapps.com** or a bare or apex domain name such as **awapps.com**. also, can use an asterisk (*****) as a wildcard in the leftmost position to protect several site names in the same domain.
   For example, ***.awapps.com** protects **pa.awapps.com**, and **ny.awapps.com**. The wildcard name will appear in the **Subject** field and the **Subject Alternative Name** extension of the ACM certificate.
3. To add more domain names to the ACM certificate, choose **Add more names** and type another domain name in the text box that opens. This is useful for protecting both a bare or apex domain (like **awapps.com**) and its subdomains (***.awapps.com**).
4. After valid domain names are entered, choose **Review and Request** or choose **Cancel** to quit.

5. Before ACM issues a certificate, it validates that you own or control the domain names in your certificate request. One can use either email validation or DNS validation.
6. Choose **DNS validation**.
7. Choose **Review and request**. Verify that the domain name and validation method are correct.
8. Choose **Confirm and request**.
9. On the **Validation** page, expand your domain information or choose **Export DNS configuration to a file**. Expand your domain information, ACM displays the name and value of the CNAME record you must add to your DNS database to validate that you control the domain.
10. Sign in to the AWS Management Console and open the Route 53 console at https://console.aws.amazon.com/route53/.
11. In the navigation pane, choose **Hosted zones**.
12. On the **Hosted Zones** page, choose the row for the hosted zone that contains the records that need to be edited
13. Choose the row for the record that need to be edited.
14. Choose **Save Record Set**

By following all the steps, certificate is validated successfully.

After the successful validation we can use the certificate while deploying the load balancer to enable **https**

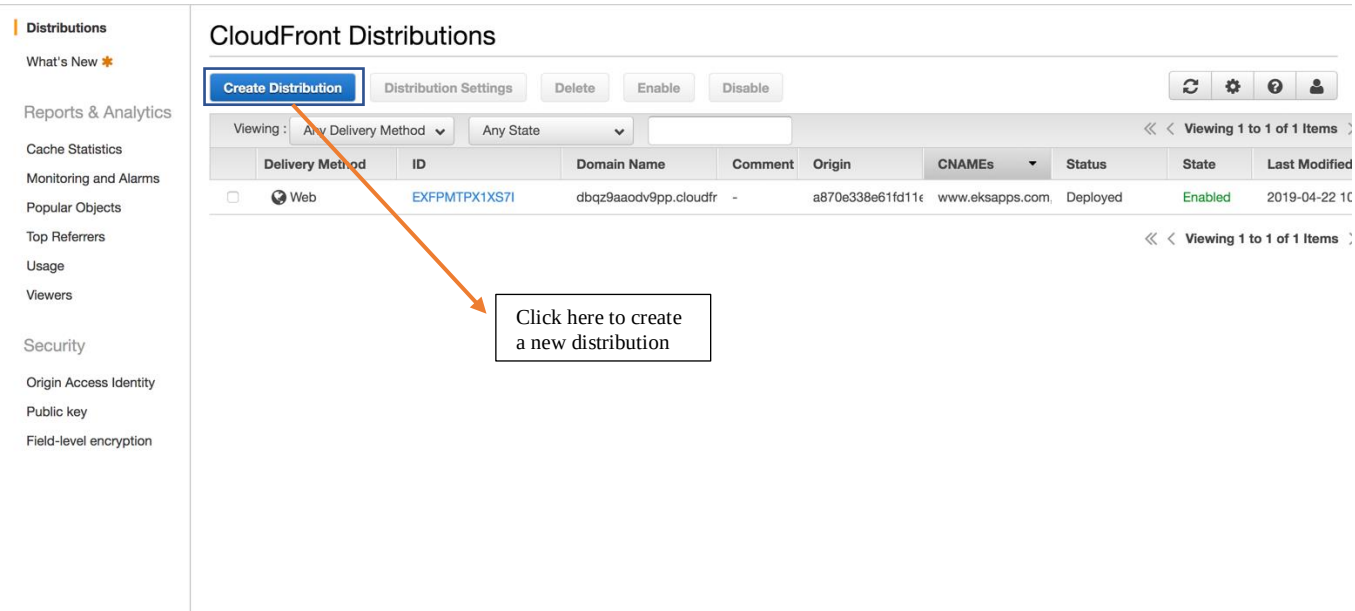


Cloud Front Component:

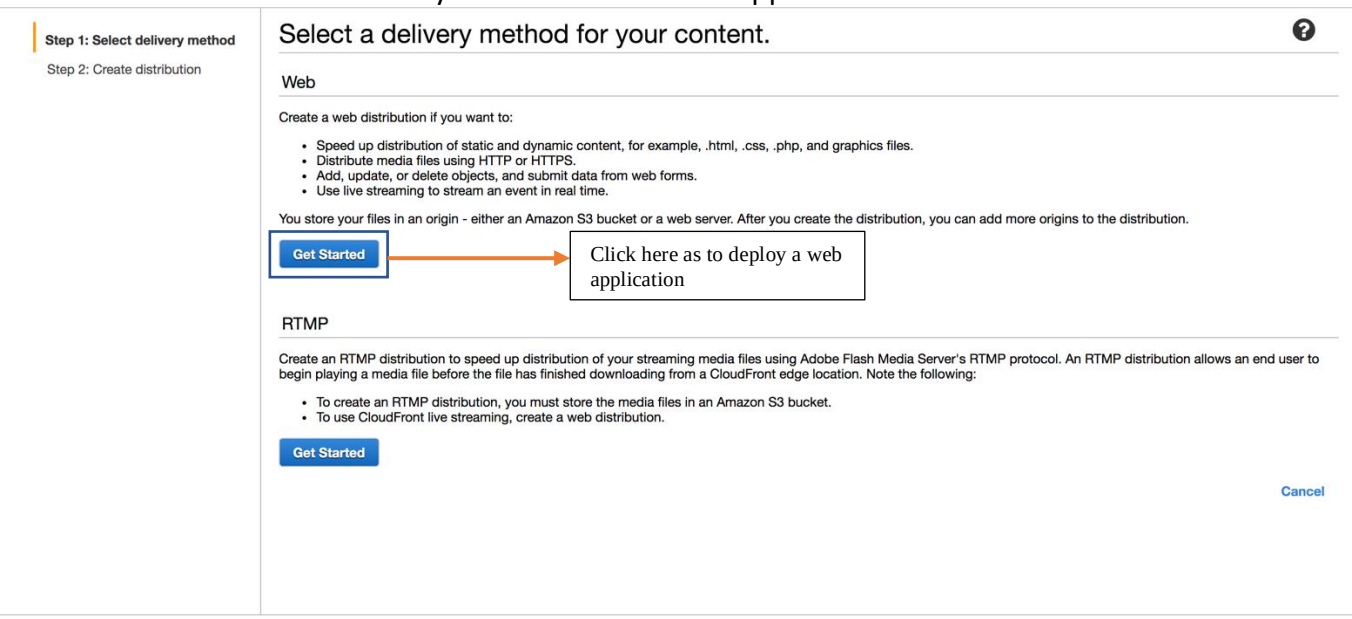
Amazon CloudFront service is being utilized to achieve path-based routing. With the help of Amazon CloudFront, a load balancer can be attached to path to achieve path-based routing.

Follow the below steps to attach an origin(load-balancer) to a path using amazon CloudFront.

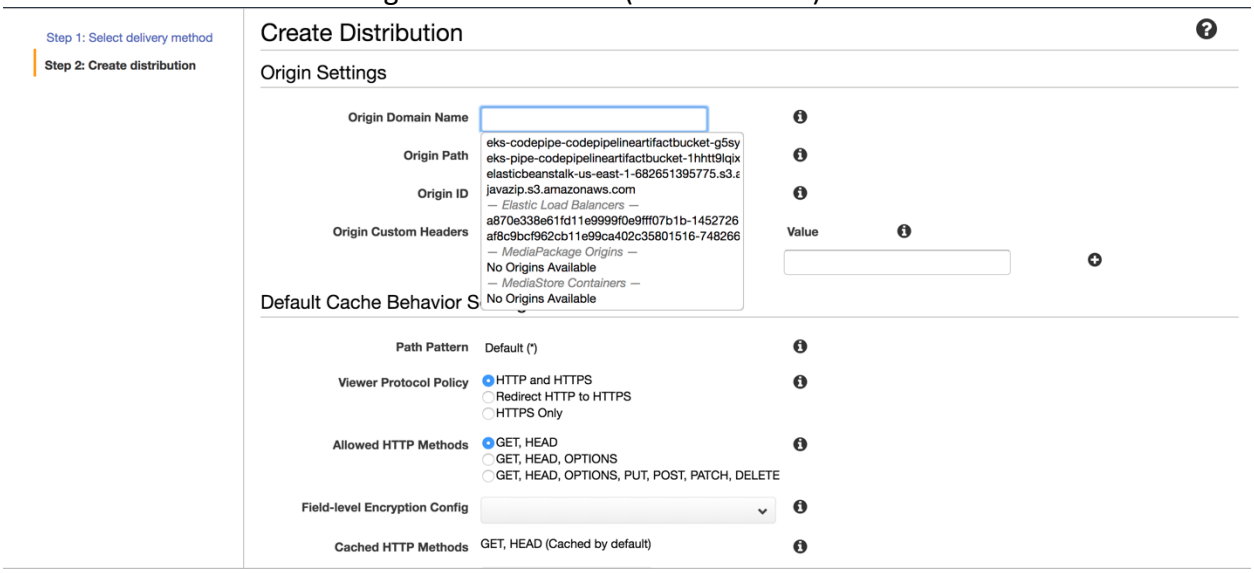
1. Login to AWS Console and open CloudFront, click on create distribution.



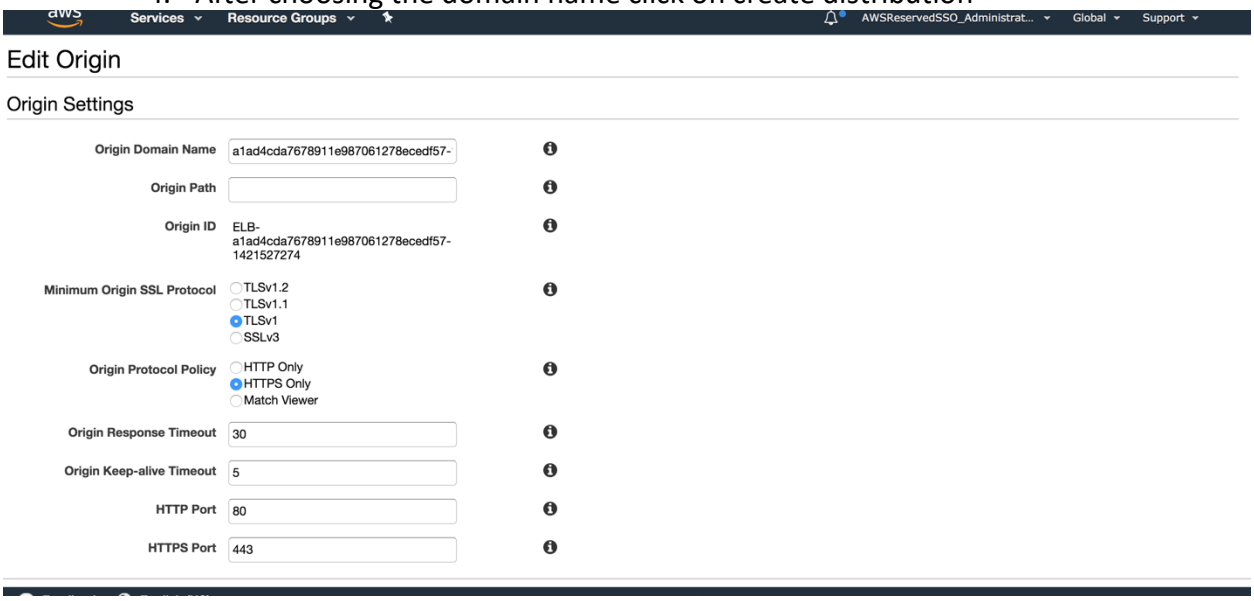
2. Select Web delivery method to deliver an application over web



3. Choose the origin Domain Name (load-balancer)



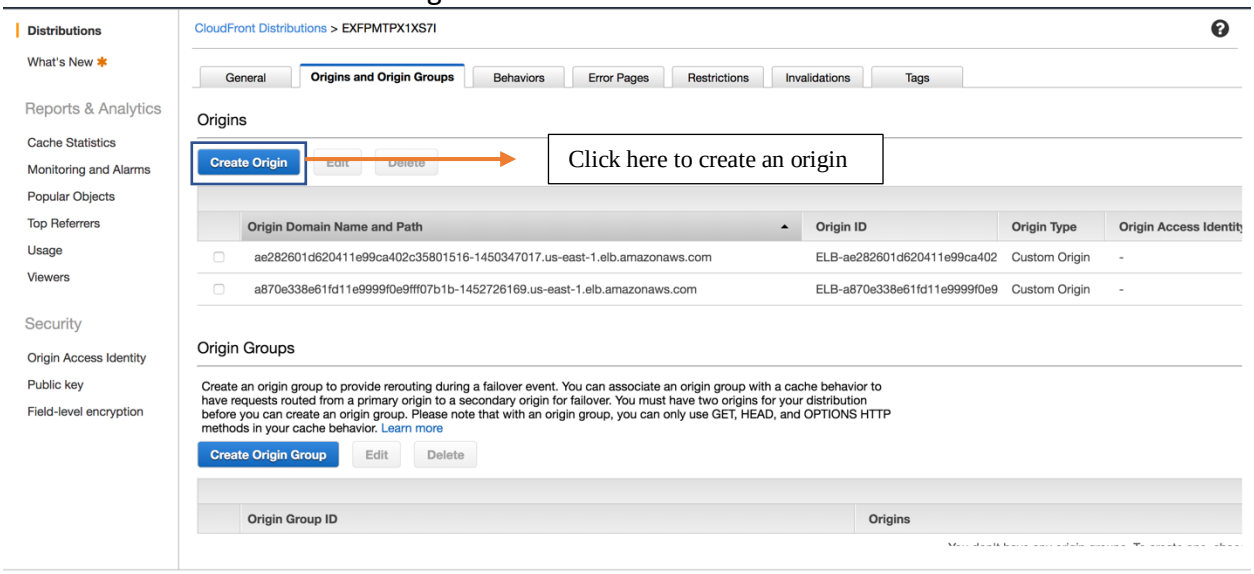
4. After choosing the domain name click on create distribution



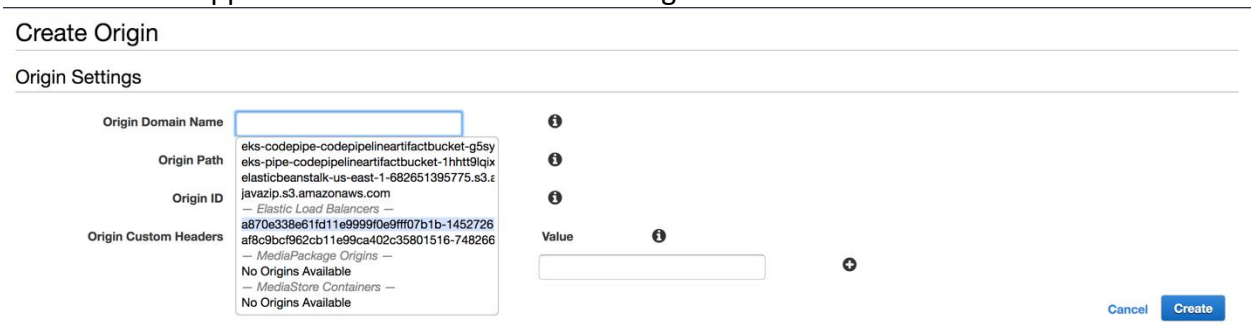
Leave the default values and click on “Create Distribution” at the bottom of page.’
It will take 20 to 25 minutes to create the distribution.

Once the distribution is created click on the distribution ID and navigate to “Origins and Origin Group” page.

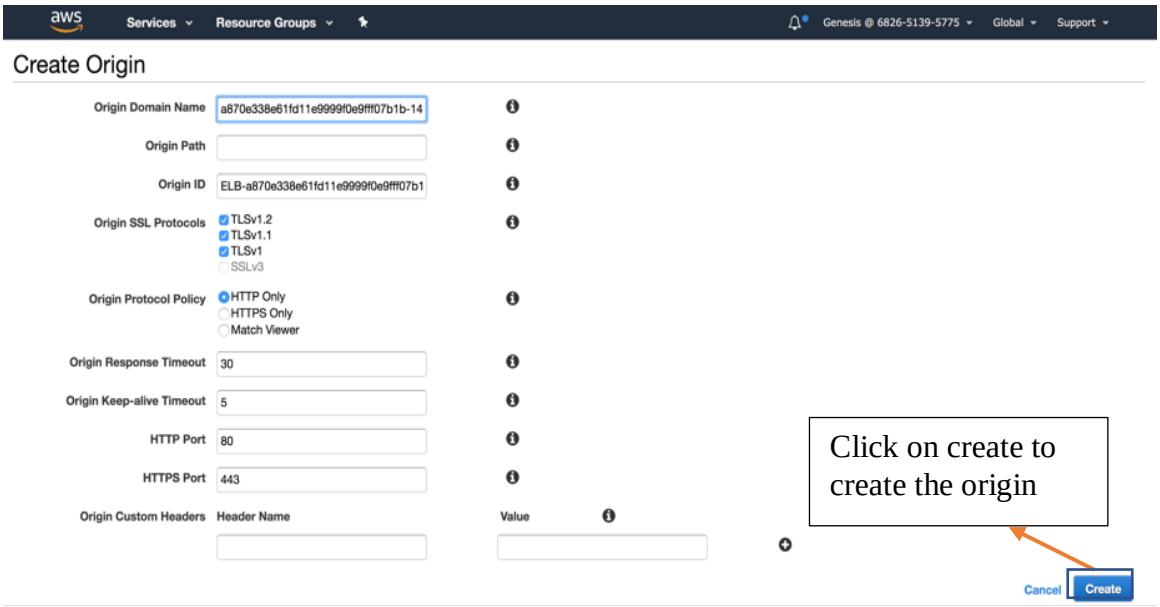
5. Click on create origin



6. Tap on the "Origin Domain Name" text box and select the load balancer for application 1and leave the remaining fields to default values.

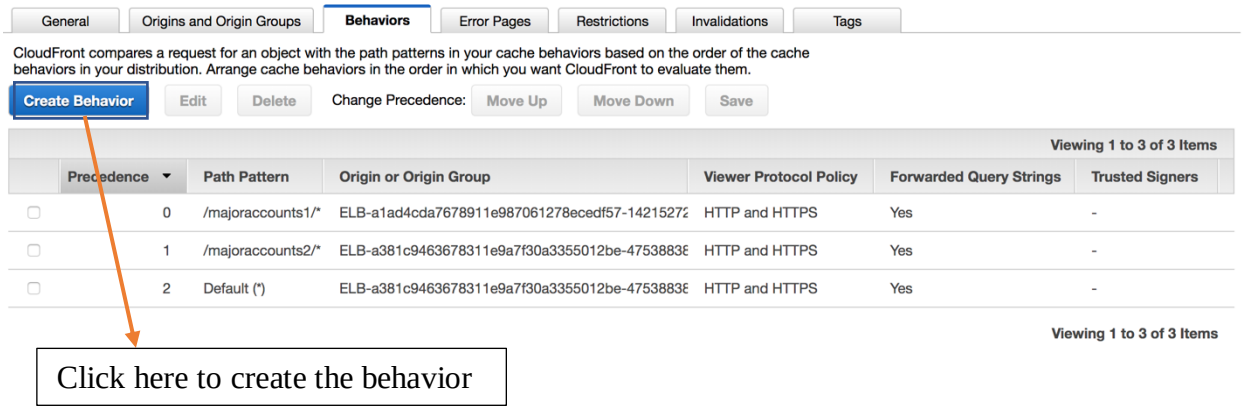


7. Now navigate to Behaviors page to create the behavior for the origin

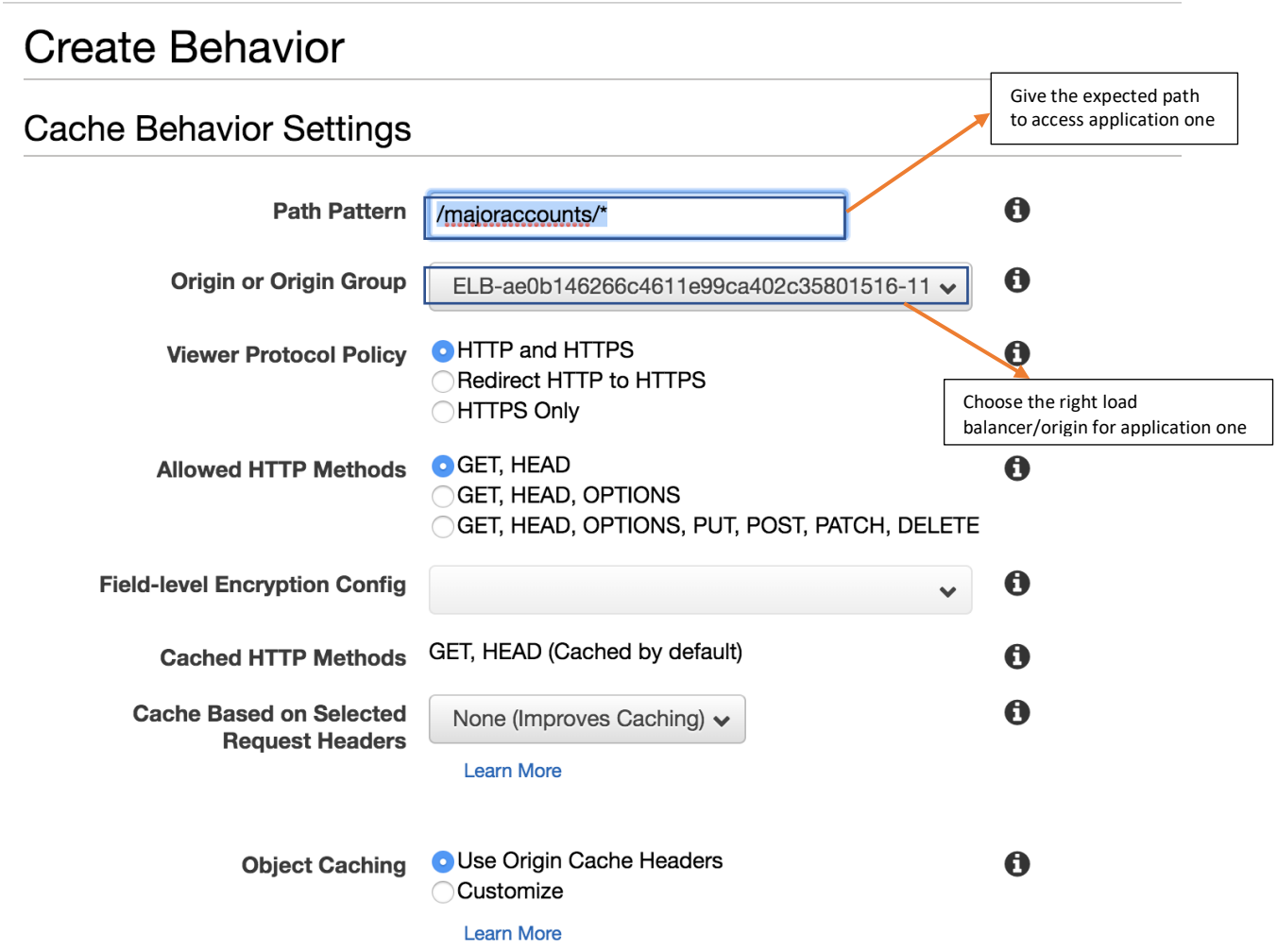


After the origin is created we need to add a behavior to the origin

8. Click on create behavior to attach a path to the origin (load-balancer)



9. Add a path pattern and choose the origin(load-balancer) to be associated to the path.



*Allow all HTTP methods

NOTE: Always add '/*' after the path i.e., /majoraccounts1 is always wrong, use the path as /majoraccounts1/*. With this practice the context path can be retained

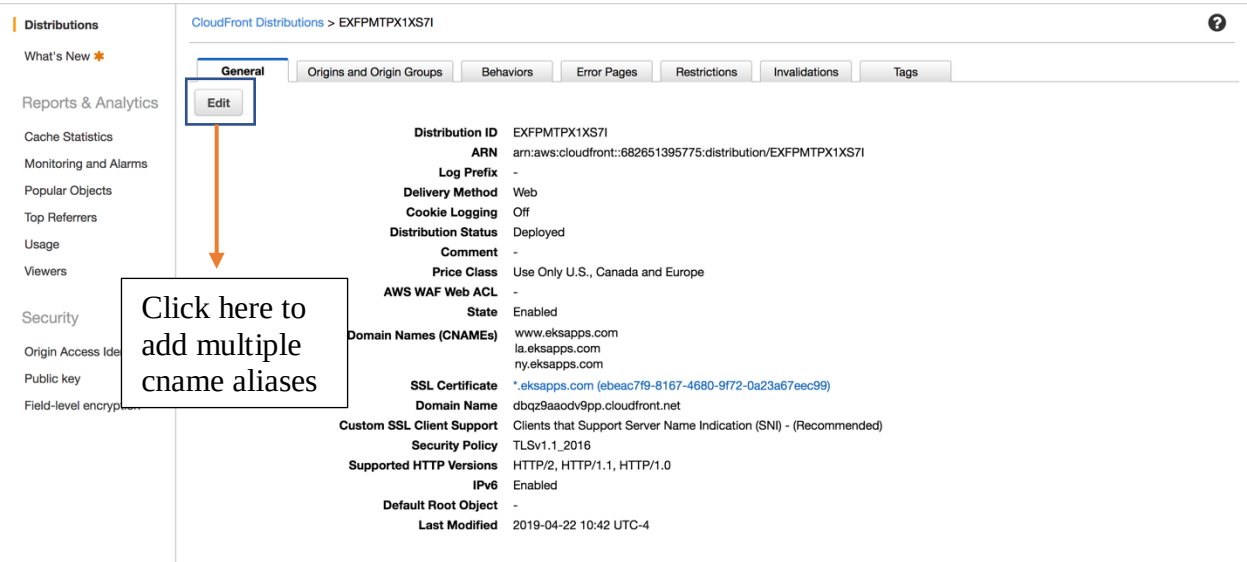
Leave all other fields to be default and click on create.

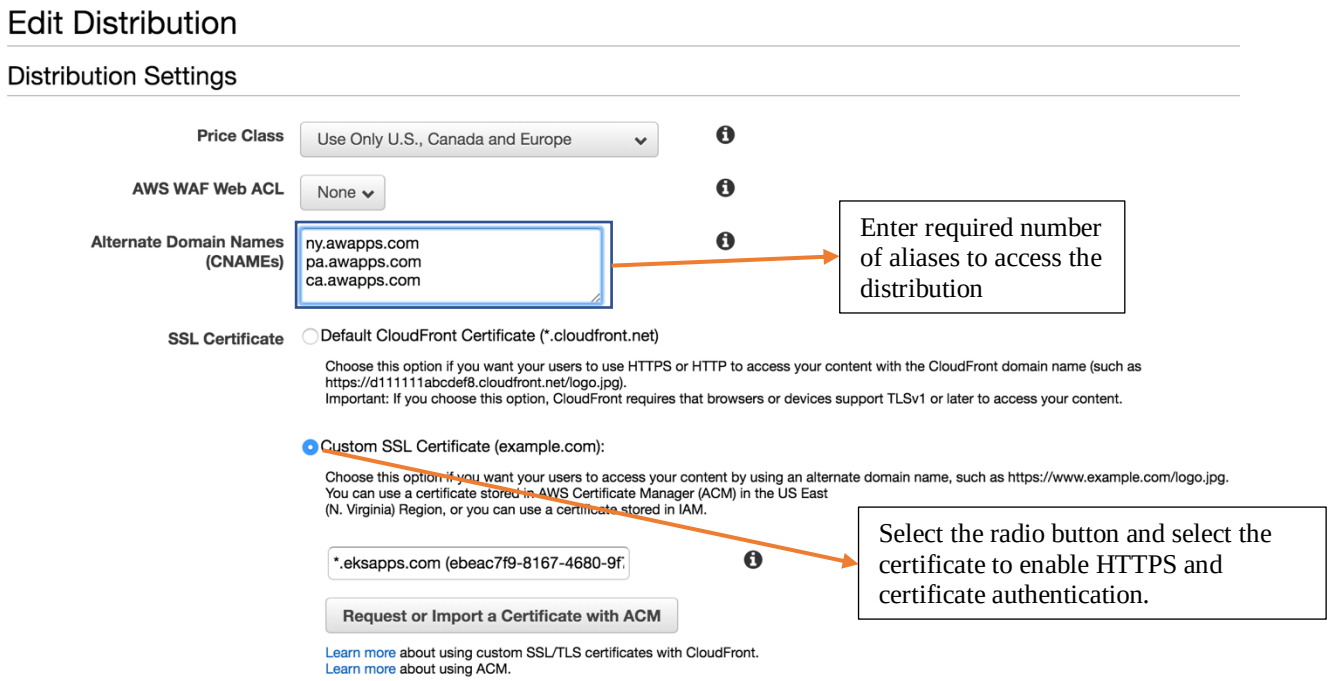
Wait for 15 to 20 min for the cloud front distribution to be redeployed.

Repeat the same steps from 5 to 9 for as many applications as required i.e., create one origin for each load balancer and one behavior for every origin.
For application 2 we will create an origin with respective load balancer and create a behavior for the origin created for second application.

After creating all the origins and respective behaviors we need to create multiple cname aliases

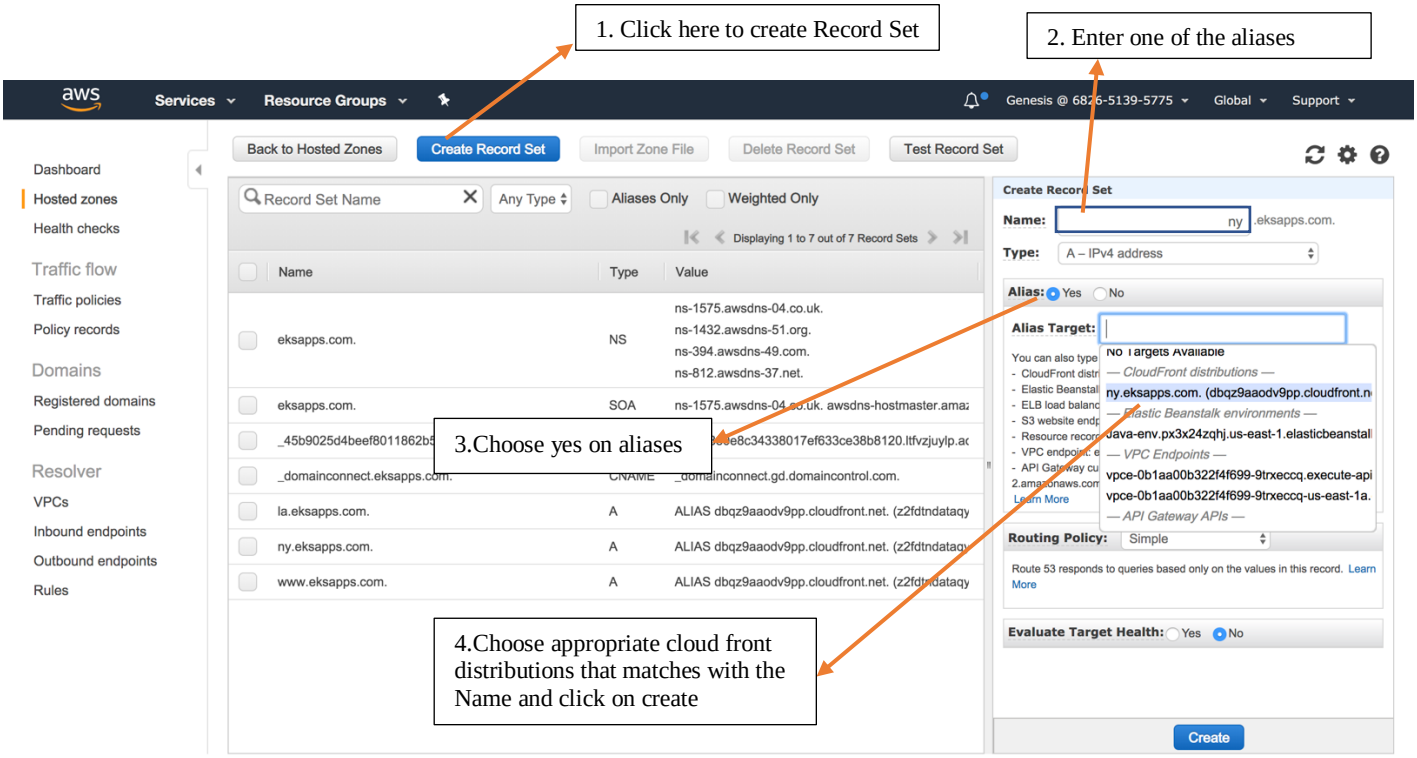
Navigate to General tab and click of Edit button to add multiple cname aliases.





Once done click on “Yes, Edit” and wait for 15 – 20 min for re-deployment of the distribution.

After success re-deployment we need to register the aliases with Route 53 as below.
Navigate to Route 53 and choose the appropriate Hosted Zone. Then we need to create Record Set’s for all the aliases as shown below.



After choosing all the options click on create button.
Repeat the same steps for all the domains and create as many records as required.

Wait for 15 minutes before accessing the application through the URL and Route 53 need to register the new records with the DNS servers.

NOTE:

1. While requesting for the certificate from AWS Certificate Manager, request the certificate for domain as ***.awapps.com** rather than **www.awapps.com.** This will let us to use the same certificate for all the sub-domains like pa.example.com, ca.example.com, etc.,

2. Name of the record in Route 53 should exactly match with the cname alias in Cloud Front. If not, we will not be able to see the cloud front distributions in the drop down.

Lambda @Edge:

Lambda@Edge is a feature of Amazon CloudFront that lets run code closer to users of your application, which improves performance and reduces latency.

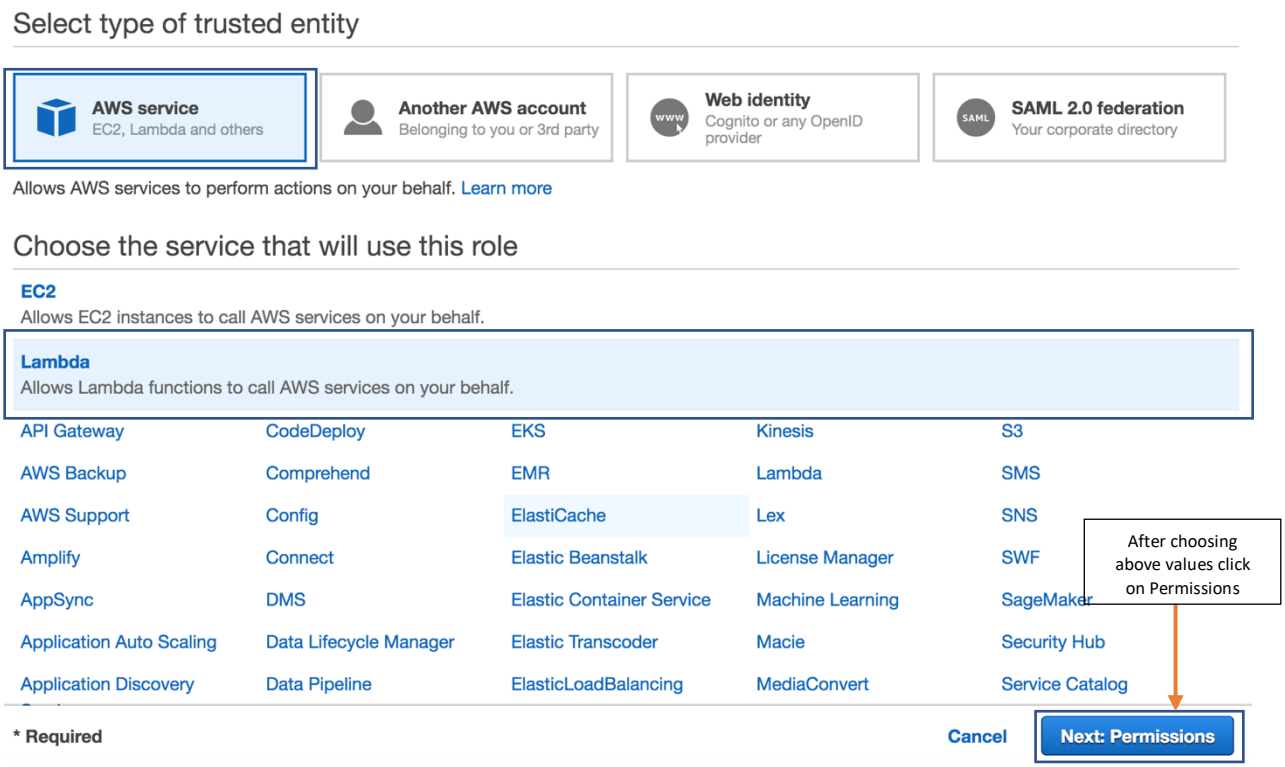
AM Water applications require URL Re-writing before the request is forwarded to the load balancer of the eks-cluster running the application.

To achieve the URL rewriting at the edge, lambda @edge had been used.

Following the below steps URL re-writing can be achieved.

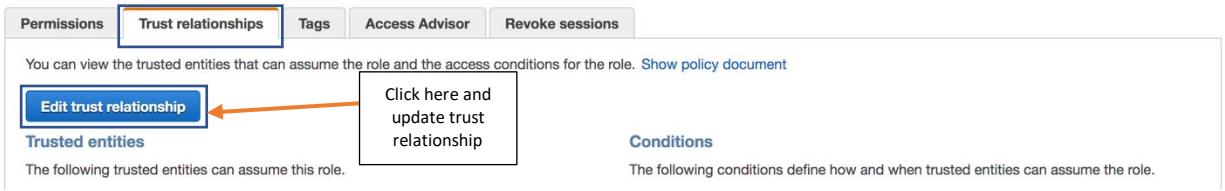
Create IAM Role:

1. Login to AWS IAM Console
2. Choose **Roles** from left pane, click on **Create Role**
3. Choose AWS Services and Lambda from the services list that will use the role



4. Attach Following polices and any other policies that are required.
    - AWS Lambda Full Access
    - AWS Lambda Basic Execution Role
5. Add Necessary Tags

6. Give the Role a name {{Eg: AWS_Lamabda_URL_Rewrite}} and add a description for the role
7. Click on create Role.
8. From the IAM Console choose Roles and select the role created (AWS_Lamabda_URL_Rewrite)
9. Choose **Trust Relationships** and click on **Edit trust relationship**

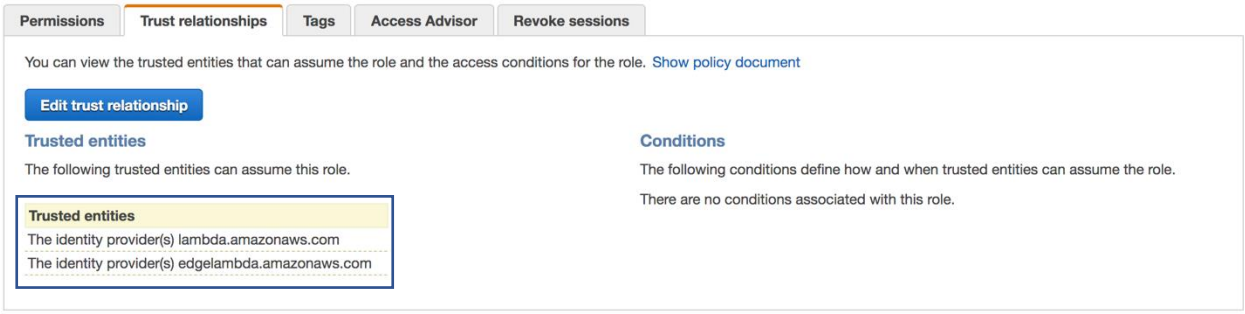


10. Add the sample script to update trust relationship

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Principal": {
    "Service": "lambda.amazonaws.com"
   },
   "Action": "sts:AssumeRole"
  },
  {
   "Effect": "Allow",
   "Principal": {
    "Service": "edgelambda.amazonaws.com"
   },
   "Action": "sts:AssumeRole"
  }
 ]
}
```

And click on **Update Trust Policy**

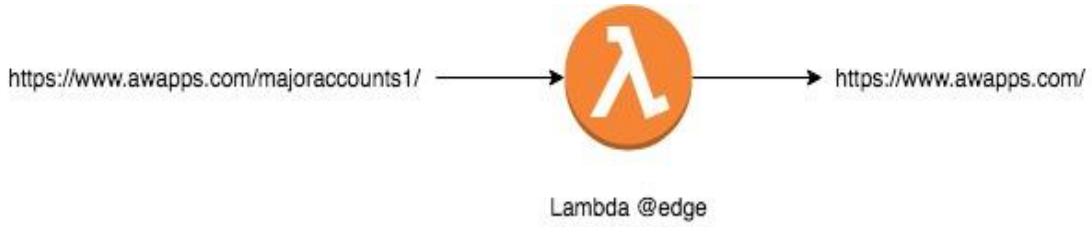
Then we can see the below entities added as trusted entities.



NOTE: This role needs to be used while creating the lambda function that need to be deployed as @edge

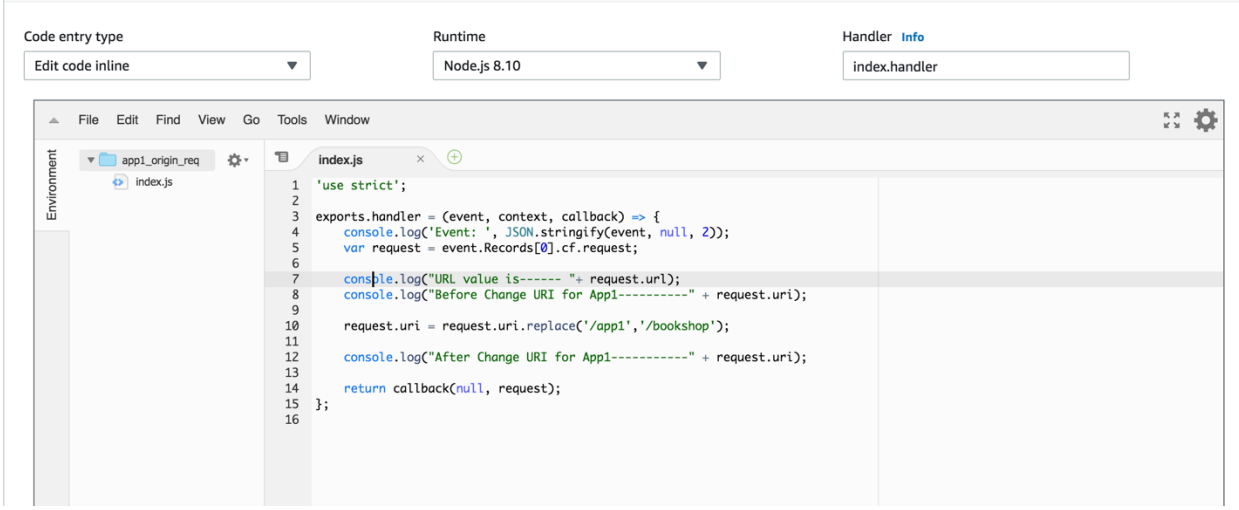
Create lambda @edge function

1. Login to the AWS Lambda Console
2. Click on **create function**
3. Choose **Author from scratch**
4. Runtime as **Node.js 10.x**
5. Expand permission and choose the role created in previous step
6. click on **Create Function.**
7. Following code snippet reads the URL, and send the re-written url to load balancer.

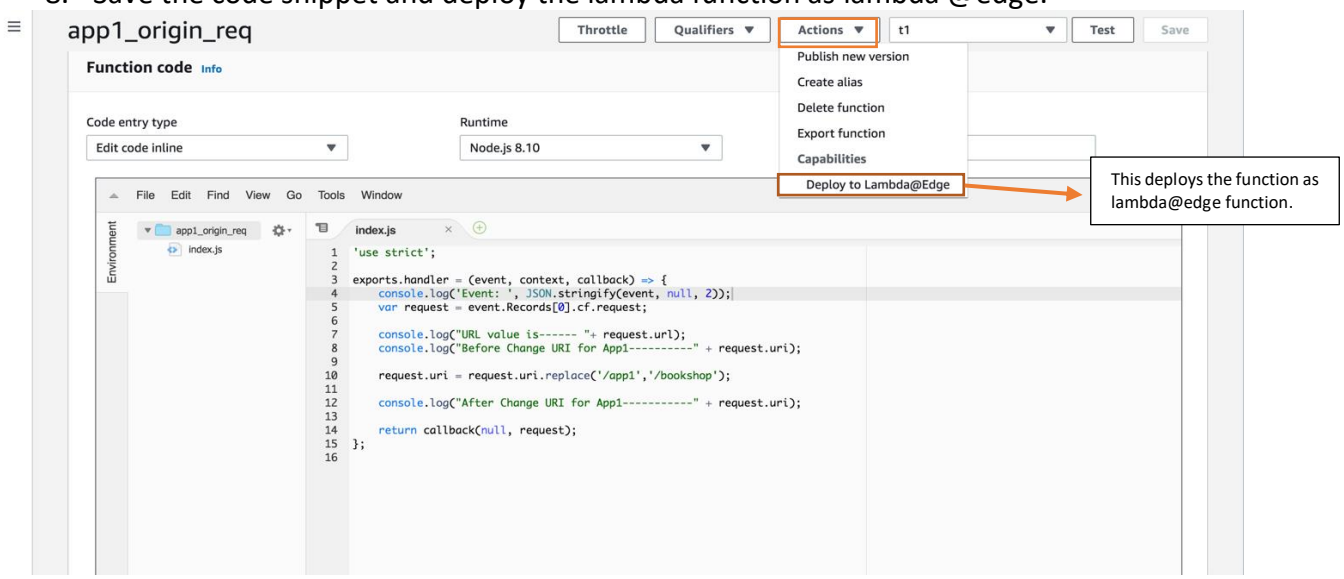


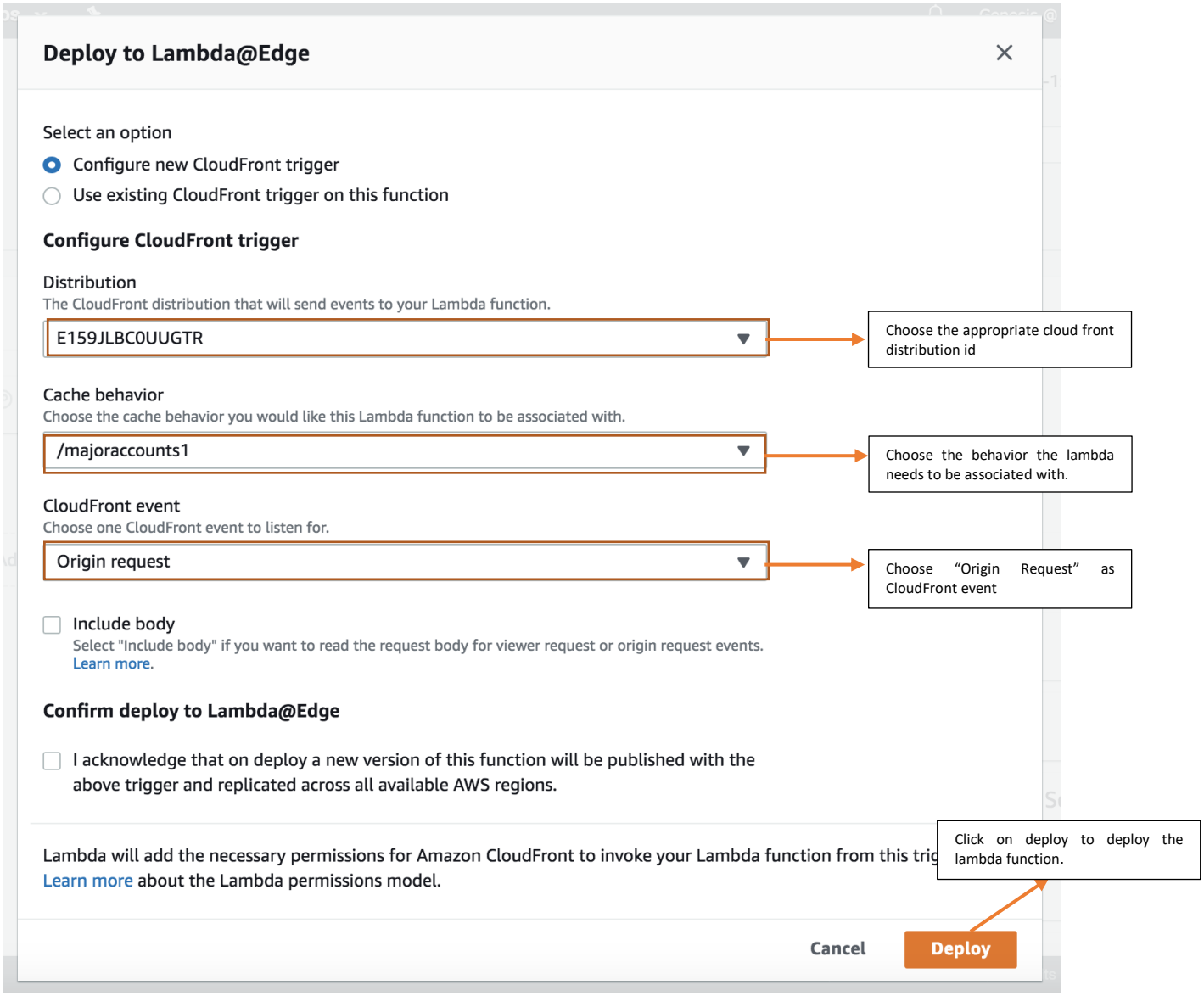

```
'use strict';

exports.handler = (event, context, callback) => {
   console.log('Event: ', JSON.stringify(event, null, 2));
   var request = event.Records[0].cf.request;

   console.log("URL value is------ "+ request.url);
   console.log("Before Change URI for App1----------" + request.uri);

   request.uri = request.uri.replace('/majoraccounts1,'' );

   console.log("After Change URI for App1----------" + request.uri);

   return callback(null, request);
};
```

This snippet will append bookshop to app1 i.e., app1/bookshop/



8. Save the code snippet and deploy the lambda function as lambda @edge.





9. This will re-deploy the CloudFront distribution with lambda@edge association.

Logging(EFK Stack):

Fluentd:

Fluentd allows you to unify data collection and consumption for a better use and understanding of data.

ElasticSearch:

Elasticsearch is a distributed, RESTful search and analytics engine capable of solving a growing number of use cases.

Kibana:

Kibana lets you visualize your Elasticsearch data and navigate the Elastic Stack. Kibana gives the freedom to select the way you give shape to your data.

Configure IAM Policy:

The fluentd log daemon will collect logs and forward to CloudWatch Logs. This will require the nodes to have permissions to send logs and create log groups and log streams. This can be accomplished with an IAM user, IAM role

- Need to ensure the Role Name for the workers use the same set in the environment

  "**test -n "$ROLE_NAME" && echo ROLE_NAME is "$ROLE_NAME" || echo ROLE_NAME is not set**"

  NOTE: If the result is "**ROLE_NAME is not set**" then one need to export the Role_name with the below command

  "INSTANCE_PROFILE_NAME=$(aws iam list-instance-profiles | jq -r '.InstanceProfiles[].InstanceProfileName' | grep nodegroup)
  INSTANCE_PROFILE_ARN=$(aws iam get-instance-profile --instance-profile-name $INSTANCE_PROFILE_NAME | jq -r '.InstanceProfile.Arn')
  ROLE_NAME=$(aws iam get-instance-profile --instance-profile-name $INSTANCE_PROFILE_NAME | jq -r '.InstanceProfile.Roles[] | .RoleName')
  echo "export ROLE_NAME=${ROLE_NAME}" >> ~/.bash_profile
  echo "export INSTANCE_PROFILE_ARN=${INSTANCE_PROFILE_ARN}" >> ~/.bash_profile"

  This will export the role to be used across the eks cluster

- IAM Policy to export logs to elastic search

  cat <<EoF > awapps-eks-logs.json

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "logs:DescribeLogGroups",
        "logs:DescribeLogStreams",
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents"
      ],
      "Resource": "*",
      "Effect": "Allow"
    }
  ]
}
EoF
```

- Attach the policy to the EKS Cluster

  "aws iam put-role-policy --role-name $ROLE_NAME --policy-name Logs-Policy-For-Worker --policy-document file://~/environment/iam_policy/awapps-eks-logs.json"

- Verify if the policy is attached to the EKS cluster

  "aws iam get-role-policy --role-name $ROLE_NAME --policy-name Logs-Policy-For-Worker"

ElasticSearch Cluster:

- AM Water had been setup to use the elastic search cluster in the shared account.

Deploy FluentD:

- Download fluentd

  " wget https://eksworkshop.com/logging/deploy.files/fluentd.yml "

- Fluentd will be deployed as a DaemonSet, i.e. one pod per worker node
- Open the downloaded fluentd.yml file and update the Region and Cluster_Name at lines 197 & 199

```
196            - name: REGION
197              value: aws-region
198            - name: CLUSTER_NAME
199              value: eks_cluster_name
```

- Deploy the fluentd daemon on the eks cluster with the below command

  "**kubectl apply -f fluentd.yml**"

- Check the status of the running pods with the command

  "**kubectl get pods -w --namespace=kube-system**"

- Wait till all the pods are up and running. Once the pods are up and running we can see the logs in CloudWatch

CloudWatch Logs Subscription:

- CloudWatch Logs can be delivered to other services such as Amazon Elasticsearch for custom processing. This can be achieved by subscribing to a real-time feed of log events. A subscription filter defines the filter pattern to use for filtering which log events gets delivered to Elasticsearch, as well as information about where to send matching log events to.
- Lambda IAM Execution Role

  Following command will create an IAM Role which is required to push the logs from CloudWatch to elastic search cluster

```
cat <<EoF >  awapps_eks_logs_lambda.json
{
  "Version": "2012-10-17",
  "Statement": [
  {
    "Effect": "Allow",
    "Principal": {
      "Service": "lambda.amazonaws.com"
    },
  "Action": "sts:AssumeRole"
  }
 ]
}
EoF
```

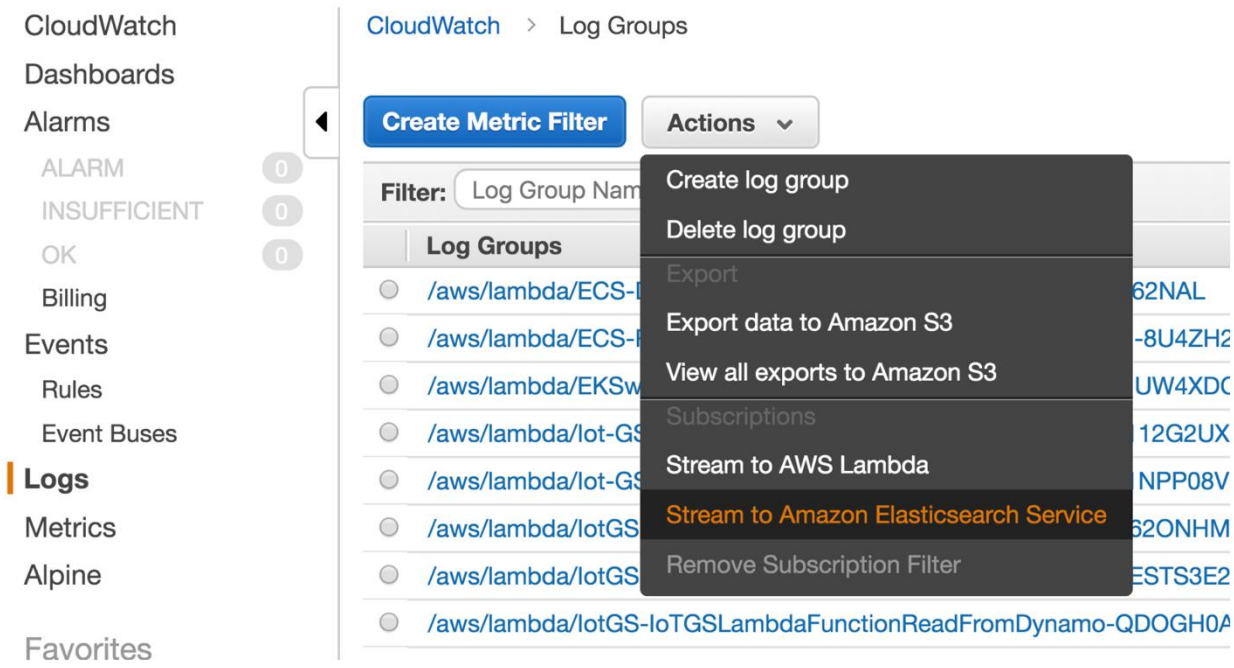
aws iam create-role --role-name lambda_basic_execution --assume-role-policy-document file:awapps_eks_logs_lambda

- Attach the role policy with the below command

  "aws iam attach-role-policy --role-name lambda_basic_execution --policy-arn arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole"

- Login to the CloudWatch Logs console

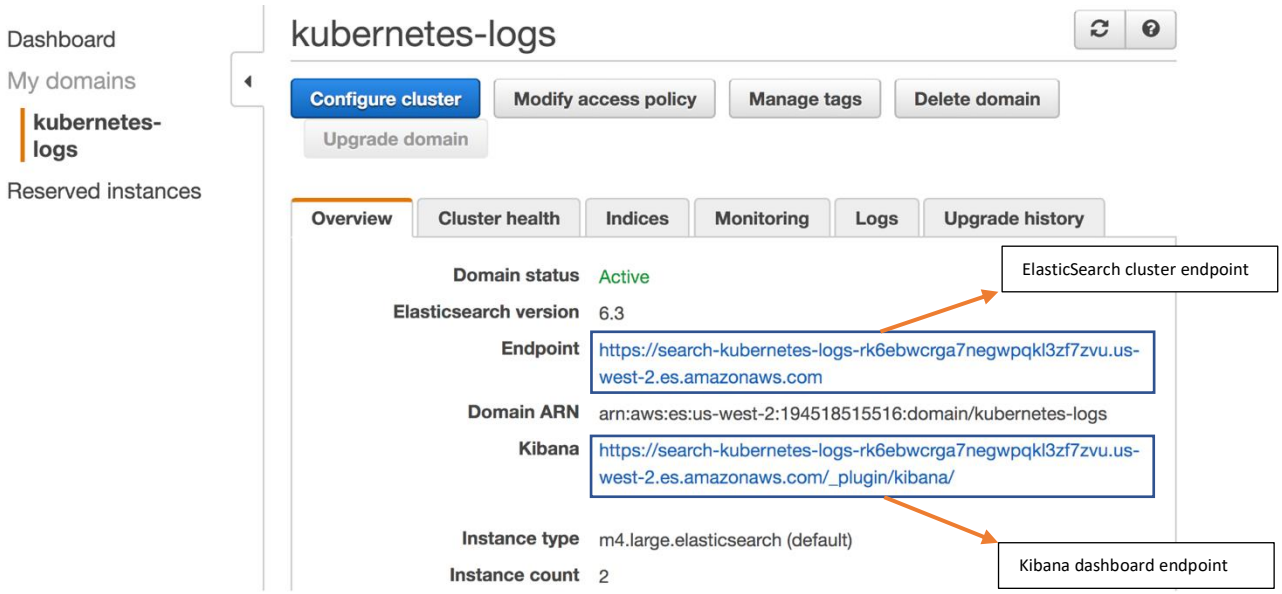
  Select the log group similar to **/eks/ekscluster-eksctl/containers** click on **Actions** and select Stream to **Amazon Elastic Search Service**



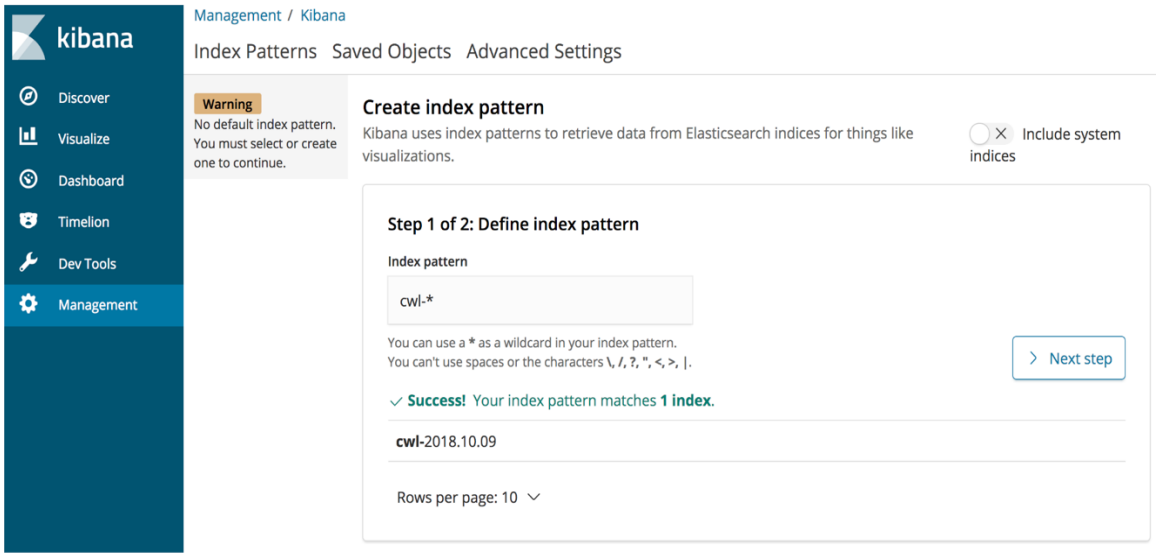
- Select the cluster and IAM role and click next
- Select **Common Log Format** and click **Next**
- Review the configuration and click on **Start Streaming**
- CloudWatch page is refreshed to show that the filter was successfully created

Configure Kibana:

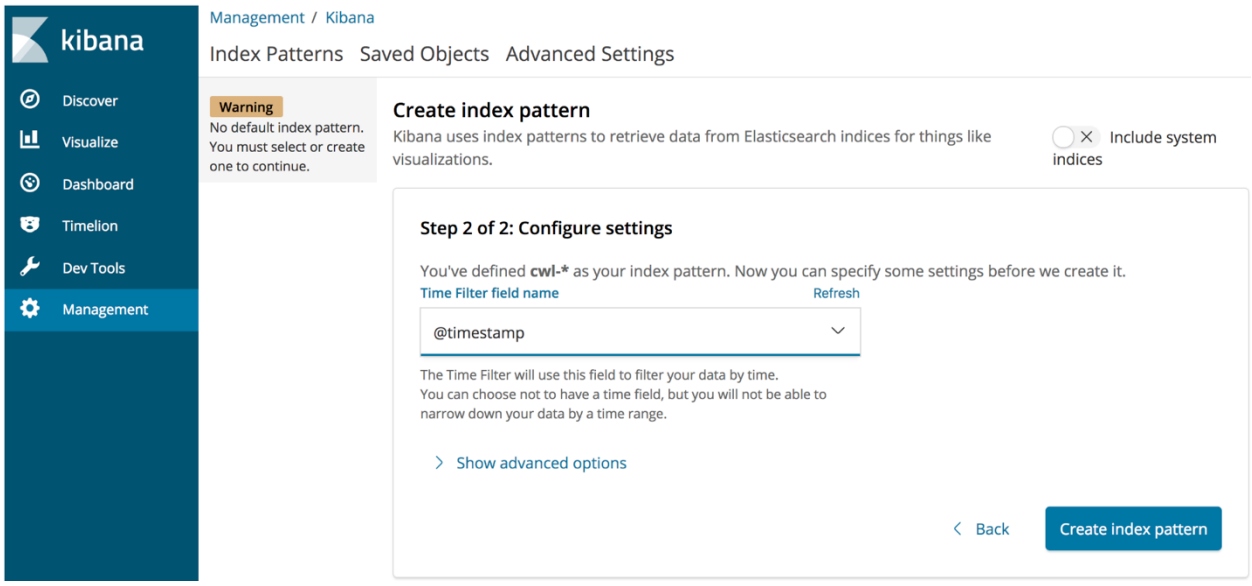
- Login to the AWS ElasticSearch console and choose kubernetes-logs
- Open the Kibana dashboard from the link. After a few minutes, records will begin to be indexed by ElasticSearch. You'll need to configure an index pattern in Kibana.

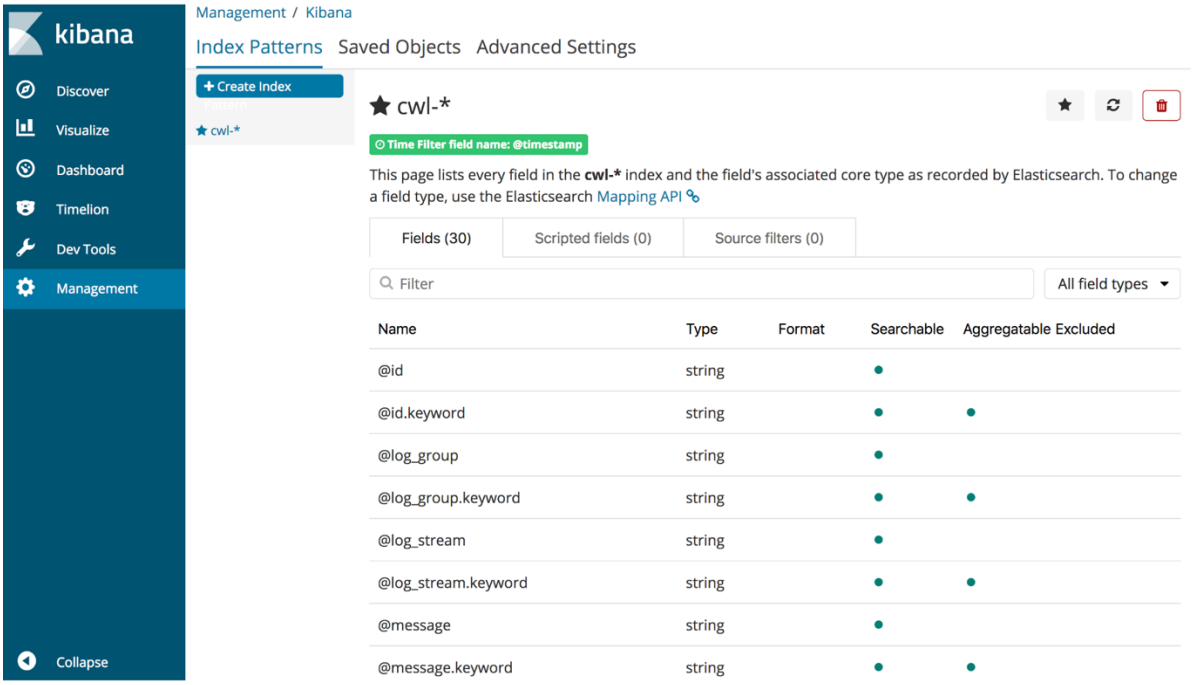


- Click on index pattern as **cwl-*** and click **Next**

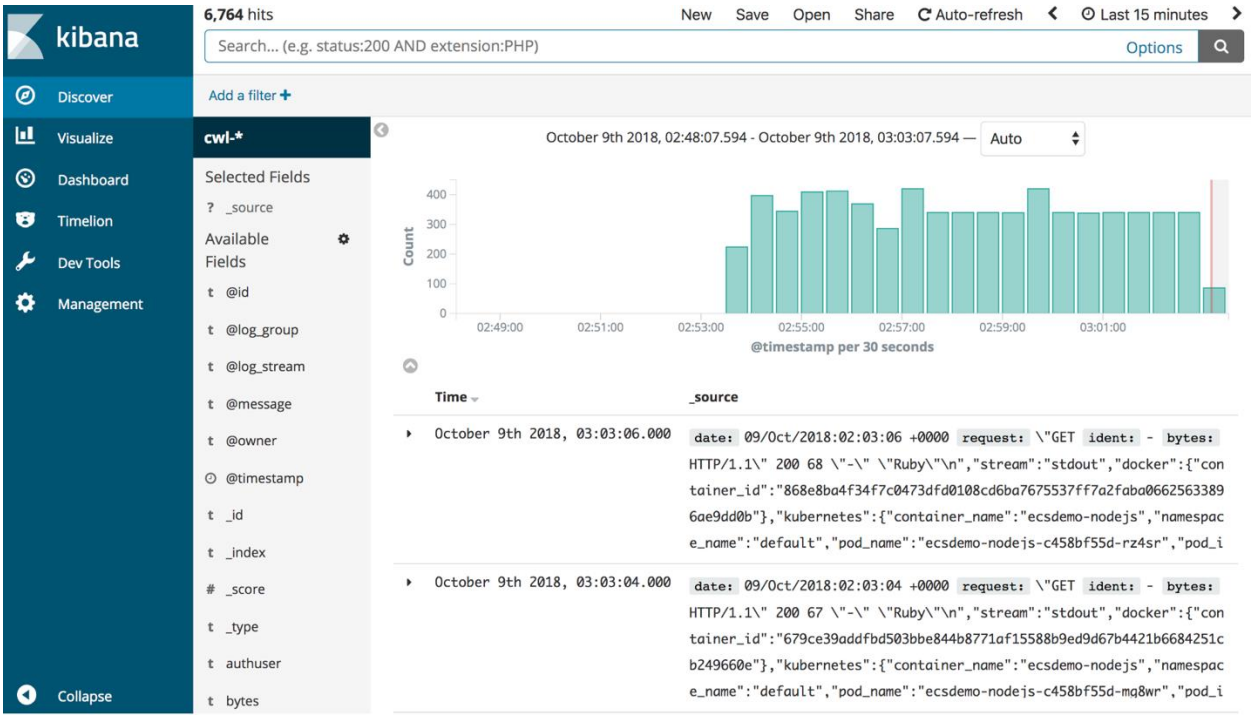


- Select **@timestamp** from dropdown list and select **Create index pattern**





Click on **Discover** and explore the logs



EKS Cluster Monitoring:

Prerequisites:

- Install helm

  Helm helps you manage Kubernetes applications — Helm Charts help you define, install, and upgrade even the most complex Kubernetes application.

  Install helm with the following steps
  - cd ~/environment
  - curl https://raw.githubusercontent.com/kubernetes/helm/master/scripts/get > get_helm.sh
  - chmod +x get_helm.sh
  - ./get_helm.sh

NOTE: Do not run 'helm init' yet

- Configure helm with RBAC

  cat <<EoF > ~/environment/rbac.yaml
  ---
  apiVersion: v1
  kind: ServiceAccount

```
metadata:
  name: tiller
  namespace: kube-system
---
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRoleBinding
metadata:
  name: tiller
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
  - kind: ServiceAccount
    name: tiller
    namespace: kube-system
EoF
```

- Now apply the configuration with the below command

  kubectl apply -f ~/environment/rbac.yaml
- Now install tiller using helm tooling

  helm init --service-account tiller

Deploy Prometheus:

Prometheus collects metrics from monitored targets by scraping metrics HTTP endpoints on these targets. Since Prometheus also exposes data in the same manner about itself, it can also scrape and monitor its own health.

- Deploy Prometheus with the below commands
  - **kubectl create namespace Prometheus**
  - **helm install stable/prometheus **
    **--name prometheus **
    **--namespace prometheus **
    **--set alertmanager.persistentVolume.storageClass="io1" **
    --set server.persistentVolume.storageClass="io1"

    NOTE: Replace "gp2" with "io1" for better performance and production deployments

- Copy the Prometheus end point to clipboard
- To check if the Prometheus components are deployed use the following command

- **kubectl get all -n Prometheus**
- Access the application by port forwarding from 8080 to 9090 with the below command
  - **kubectl port-forward -n prometheus deploy/prometheus-server 8080:9090**
- In cloud9 click on Tools -> Preview -> Preview Running Applications. Scroll to the end and append '/targets' to the URL
- A web UI will be displayed to show all the targets and metrics that are being monitored

Deploy Grafana:

Grafana is an open source visualization tool that can be used on top of a variety of different data stores

- Deploy Grafana with the below commands
  - **kubectl create namespace Grafana**
  - **helm install stable/grafana **
    **--name grafana **
    **--namespace grafana **
    **--set persistence.storageClassName="io1" **
    --set adminPassword="********" **
    **--set datasources."datasources\.yaml".apiVersion=1 **
    **--set datasources."datasources\.yaml".datasources[0].name=Prometheus **
    **--set datasources."datasources\.yaml".datasources[0].type=prometheus **
    **--set datasources."datasources\.yaml".datasources[0].url=http://prometheus-server.prometheus.svc.cluster.local **
    **--set datasources."datasources\.yaml".datasources[0].access=proxy **
    **--set datasources."datasources\.yaml".datasources[0].isDefault=true **
    --set service.type=LoadBalancer
- Successful deployment can be checked with below command
  - **kubectl get all -n Grafana**
- Copy the Grafana URL and export the same with the below command
  - **export ELB=$(kubectl get svc -n grafana grafana -o jsonpath='{.status.loadBalancer.ingress[0].hostname}')**
  - **echo http://$ELB**
- Login to the application with username as admin and get the hash password with below command
  - **kubectl get secret --namespace grafana grafana -o jsonpath="{.data.admin-password}" | base64 --decode ; echo**

NOTE: For the password as a part of best practices and future enhancements use SSO with Grafana

Autoscaling:

The Horizontal Pod Autoscaler(HPA) and Kubernetes Metrics Server are now supported by Amazon Elastic Container Service for Kubernetes (EKS). This makes it easy to scale your Kubernetes workloads managed by Amazon EKS in response to custom metrics.

HPA is a Kubernetes component that automatically scales your service based on CPU utilization or other metrics that you define through the Kubernetes Metrics Server.

Prerequisites:

To have metrics read by the metric server to auto scale one need to update the resource policies of application deployment component of the deployment.yaml file as shown below

```
resources:
  limits:
    cpu: 1000m
    memory: 512Mi
  requests:
    cpu: 1000m
    memory: 512Mi
```

This will enable the metric server to apply scaling policies to the pods.

Deploy the Metric Sever:

- Metrics Server is a cluster-wide aggregator of resource usage data. These metrics will drive the scaling behavior of the deployments

  Will deploy the metrics server with below steps

- Download the metrics server from github with the below url
  - https://github.com/kubernetes-incubator/metrics-server/releases/latest

- Navigate to your downloads folder and extract the source code archive.
  - Extract the download the file with the below command

    tar -xzf metrics-server-<*version*>.tar.gz

- Apply all of the YAML manifests in the metrics-server-0.3.1/deploy/1.8+ directory (substituting your release version).

  kubectl apply -f metrics-server-<*version*>/deploy/1.8+/

- Verify that the metrics-server deployment is running the desired number of pods with the following command

  kubectl get deployment metrics-server -n kube-system

```
NAME              DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
metrics-server    1         1         1            1           56m
```

- To test the deployed metrics server status

  kubectl get apiservice v1beta1.metrics.k8s.io -o yaml

  If all is good we can see the result like this

  status:
  conditions:
  - lastTransitionTime: 2018-10-15T15:13:13Z
  message: all checks passed
  reason: Passed
  status: "True"
  type: Available

- To view the logs of metric server
  - **kubectl -n metrics get pods**

```
(global-venv) ➜ ~ kubectl -n metrics get pods
NAME                              READY     STATUS    RESTARTS   AGE
metrics-server-5f64dbfb9d-j8jmg   1/1       Running   0          8d
```

  This shows the pods that are running under metrics server.

- To view detailed event logs for the pod we can do with the below command
  - **kubectl logs -n metrics metrics-server-5f64dbfb9d-j8jmg**

  This command will display the detailed event logs for the pod gathering metric logs.

Deploying App with Horizontal Auto Scaling:

Deploy.yaml file that will be used in the code pipeline to deploy the application on to EKS cluster need to be updated with the following sample snippet to enable HPA for the pods.

```yaml
---
apiVersion: autoscaling/v2beta1
kind: HorizontalPodAutoscaler
metadata:
  name: majoraccounts-frontend
  namespace: default
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: majoraccounts-frontend
  minReplicas: 2
  maxReplicas: 10
  metrics:
  - type: Resource
    resource:
      name: cpu
      targetAverageUtilization: 50
```

Here we are associating the hpa to majoraccounts-frontend node cluster

Here we are stating that 2 replicas are min and we can scale up to 10 replicas and a new replica will be creates if the CPU consumption is over 50%

Redeploy the application with the updates yaml file, this time we can see the autoscaler being deployed along with the application and load balancer.

NOTE:

- If the metrics server cannot read metrics from the pods then we need to update the **deploy/1.8+/metrics-server-deployment.yaml** file as below
- Edit the **deploy/1.8+/metrics-server-deployment.yaml** file, to override the default command by adding a command section that didn't exist before

command:
- /metrics-server
- --kubelet-preferred-address-types=InternalIP

```yaml
containers:
- name: metrics-server
  image: k8s.gcr.io/metrics-server-amd64:v0.3.3
  command:
    - /metrics-server
    - --kubelet-preferred-address-types=InternalIP
```

Delete the existing deployment and re-deploy the application with the updated yaml file.

Configure Cluster Autoscaler:

Cluster Autoscaler automatically adjusts the number of nodes in a Kubernetes cluster when there are insufficient capacity errors to launch new pods, and also decreases the number of nodes when they are underutilized.

Download Cluster Autoscaler script:

In cloud9 execute below commands:

mkdir ~/environment/cluster-autoscaler

cd ~/environment/cluster-autoscaler

wget https://eksworkshop.com/scaling/deploy_ca.files/cluster_autoscaler.yml

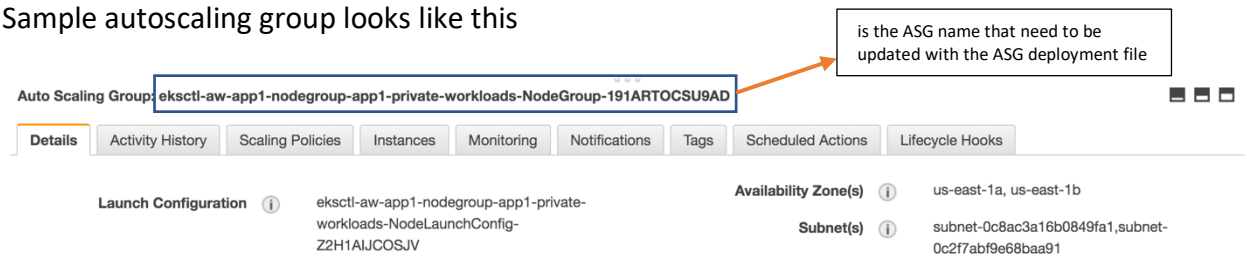
Configure Autoscaling Group:

While we are creating the EKS cluster with CI/CD pipeline AWS will create Auto Scaling Groups(ASG) for the respective node clusters we need to pick the name of the ASG and update the ASG yaml file to associate the recommendation to scale the cluster.

To view the autoscaling groups

- login to AWS console and navigate to Ec2 instances control panel.
- Choose Auto Scaling Groups from the left pane.

  Sample autoscaling group looks like this



- Open the cluster_creation.yaml file and update the command component with min and max nodes and the ASG group name.

```
command:
  - ./cluster-autoscaler
  - --v=4
  - --stderrthreshold=info
  - --cloud-provider=aws
  - --skip-nodes-with-local-storage=false
  - --nodes=2:10:eksctl-aw-app1-nodegroup-app1-private-workloads-NodeGroup-191ARTOCSU9AD
```

Here minimum nodes to run the application are 2 and max nodes are 10 and associating the autoscaling script with the ASG group.

Create IAM Policy:

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

- Create the following policy to be attached to the EC2 worker nodes

```
mkdir ~/environment/asg_policy
cat <<EoF > ~/environment/asg_policy/k8s-asg-policy.json
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "autoscaling:DescribeAutoScalingGroups",
    "autoscaling:DescribeAutoScalingInstances",
    "autoscaling:SetDesiredCapacity",
    "autoscaling:TerminateInstanceInAutoScalingGroup"
   ],
   "Resource": "*"
  }
 ]
}
EoF
```

- Apply the policy with the below command

  aws iam put-role-policy --role-name $ROLE_NAME --policy-name ASG-Policy-For-Worker --policy-document file://~/environment/asg_policy/k8s-asg-policy.json

- Validate the policy attached to the role with the below command

aws iam get-role-policy --role-name $ROLE_NAME --policy-name ASG-Policy-For-Worker

Deploy Cluster Autoscaler:

Deploy the cluster autoscaling with the below command

kubectl apply -f ~/environment/cluster-autoscaler/cluster_autoscaler.yml

This will deploy an auto scaling group, to check the logs use the below command

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

NOTE:

If the $ROLE_NAME is throwing an error please follow the below steps

Export the ROLE_NAME variable with below commands

```
INSTANCE_PROFILE_NAME=$(aws iam list-instance-profiles | jq -r
'.InstanceProfiles[].InstanceProfileName' | grep nodegroup)

INSTANCE_PROFILE_ARN=$(aws iam get-instance-profile --instance-profile-name
$INSTANCE_PROFILE_NAME | jq -r '.InstanceProfile.Arn')
ROLE_NAME=$(aws iam get-instance-profile --instance-profile-name $INSTANCE_PROFILE_NAME | jq -r
'.InstanceProfile.Roles[] | .RoleName')

echo "export ROLE_NAME=${ROLE_NAME}" >> ~/.bash_profile

echo "export INSTANCE_PROFILE_ARN=${INSTANCE_PROFILE_ARN}" >> ~/.bash_profile
```

Web Application Firewall(WAF):

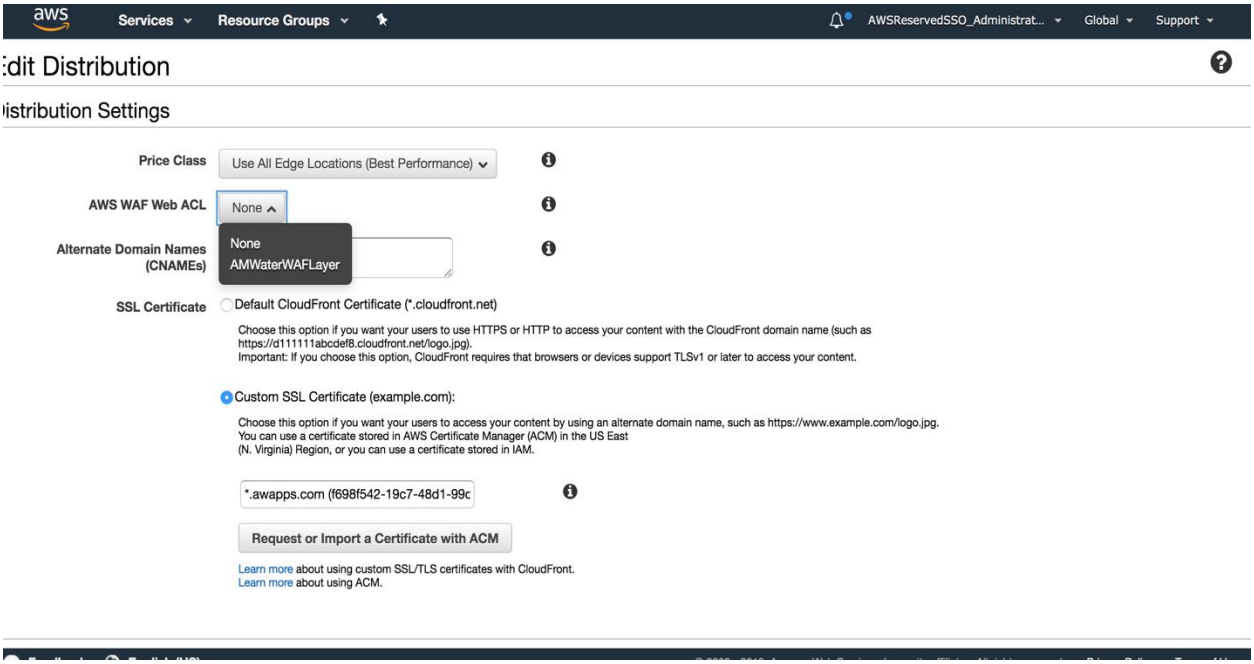
AWS WAF is a web application firewall that lets you monitor the HTTP and HTTPS requests that are forwarded to an Amazon API Gateway API, Amazon CloudFront or an Application Load Balancer.

Following cloud formation template had been used to build the WAF

https://s3.amazonaws.com/solutions-reference/aws-waf-security-automations/latest/aws-waf-security-automations.template

Using the above s3 url build the WAF with all the necessary parameters.

Associate the WAF with CloudFront as shown below



Performance & Sizing:

Performance, sizing , configuration and optimization considerations are yet to be executed as the team is currently is working on building/refactoring an end to end application. The team should perform load testing/QA and optimize/size/configure cluster accordingly before moving the app to UAT/Prod.

Cluster Creation Automation:

There are many ways how we can deploy an EKS cluster Eg: AWS console, CloudFormation, etc.,. Here the customer wanted to have a cluster created with boto3 and have the autoscaling enabled instead of using the console and retrieve the autoscaling group and modifying it. With this **script** we can achieve the complete EKS cluster created with autoscaling enabled.

Conclusion:

Putting all together, architecture has been implemented using the following technologies and services

- **Application deployment platform**: One EKS cluster per application
- **Automation:** CloudFormation, Code Pipeline, eksctl
- **DNS Services**: Route 53
- **Content Deliver Network**: Cloud Front
- **Routing**: Cloud Front
- **Load Balancing**: ELB/Classic
- **Certificate Manager**: ACM
- **Logging**: Elastic Search, FluentD, Kibana
- **Monitoring**: Prometheus, Grafana
- **Web Application Firewall**: AWS WAF