

# Securing AWS EKS Kubernetes Ingress gateway resource with Trusted TLS certificate using Cert-manager-Atlas Issuer



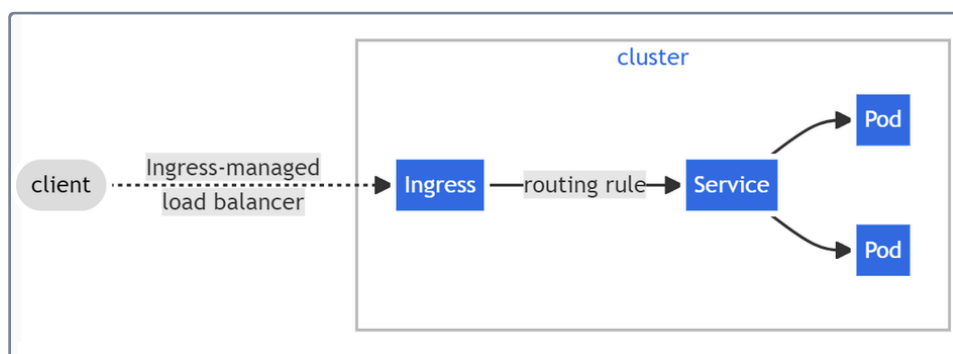
## Pre-requisites:

1. AWS Account
2. Nginx Ingress
3. One Valid Domain Name
4. Kops and Kubectl
5. Helm Package Manager
6. Cert-manager & its CRD's
7. Cert-manager-Atlas Plugin

## What is Ingress?

[Ingress](#) exposes HTTP and HTTPS routes from outside the cluster to [services](#) within the cluster. Traffic routing is controlled by rules defined on the Ingress resource.

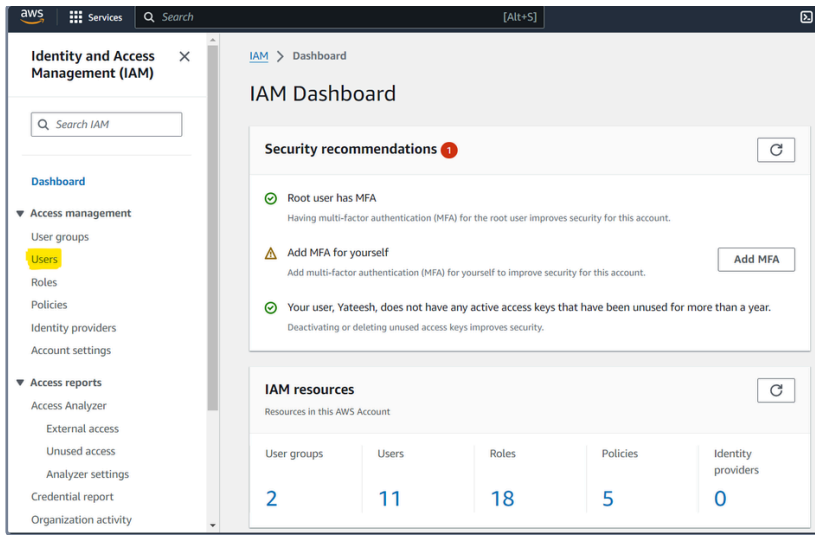
Here is a simple example where an Ingress sends all its traffic to one Service:



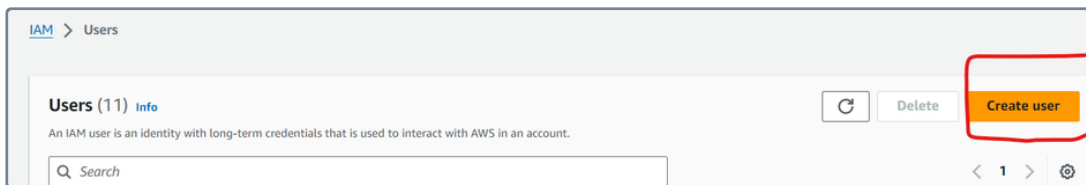
All the below mentioned steps has been executed over Linux (amd64)

Here are the following steps for your to secure your Ingress with GlobalSign's Trusted TLS certificate using Cert-manager-Atlas Plugin

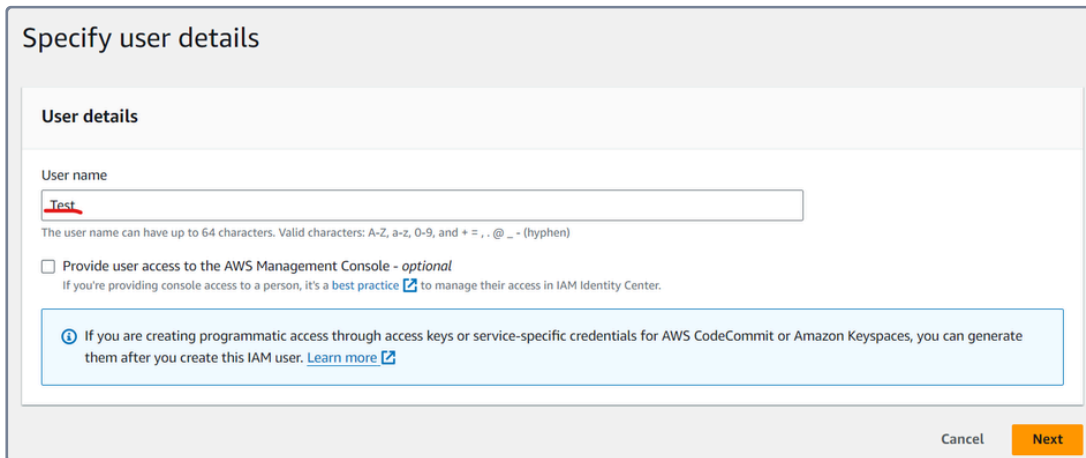
1. **AWS Ubuntu EC2 Instance - Use the AWS documentation for creating Ubuntu Instance**
  2. **Create A User in IAM**
    - a. Go to IAM Console and select Users
-



b. Click on create user



c. Give a name to the user



d. Fill the checkbox of "Provide user Access to the AWS Management Console"

- i. Select "I want to create an IAM user "
- ii. In Console Password, choose "Autogenerate Password" or "Custom Password" based on your choice.
- iii. Click "Next" in the bottom right corner.

**User details**

User name

Test

The user name can have up to 64 characters. Valid characters: A-Z, a-z, 0-9, and + = , . @ \_ - (hyphen)

**Provide user access to the AWS Management Console - optional**  
 If you're providing console access to a person, it's a [best practice](#) to manage their access in IAM Identity Center.

**Are you providing console access to a person?**

User type

Specify a user in Identity Center - Recommended  
 We recommend that you use Identity Center to provide console access to a person. With Identity Center, you can centrally manage user access to their AWS accounts and cloud applications.

I want to create an IAM user  
 We recommend that you create IAM users only if you need to enable programmatic access through access keys, service-specific credentials for AWS CodeCommit or Amazon Keyspaces, or a backup credential for emergency account access.

e. In Set Permissions, choose

- i. Select "Add User to a group" in case if you already have defined policies for a particular user group, otherwise choose "Attach Policies directly".

**Set permissions**

Add user to an existing group or create a new one. Using groups is a best-practice way to manage user's permissions by job functions. [Learn more](#)

**Permissions options**

**Add user to group**  
 Add user to an existing group, or create a new group. We recommend using groups to manage user permissions by job function.

**Copy permissions**  
 Copy all group memberships, attached managed policies, and inline policies from an existing user.

**Attach policies directly**  
 Attach a managed policy directly to a user. As a best practice, we recommend attaching policies to a group instead. Then, add the user to the appropriate group.

ii. In Permission Policies, provide the following permissions to the user(Note:- You can provide permissions based on your own requirements as this is just for the example purposes.)

1. AdministratorAccess
2. AmazonEC2FullAccess
3. AmazonEKSClusterPolicy
4. AmazonEKSServicePolicy
5. AmazonEventBridgeFullAccess
6. AmazonRoute53FullAccess
7. AmazonVPCFullAccess
8. AWSCloudFormationFullAccess
9. IAMFullAccess

10. Click "Next" in the bottom-right corner

**Permissions policies (1181)** [Refresh](#) [Create policy](#)

Choose one or more policies to attach to your new user.

Filter by Type: All types 1 match

Search: VPCFull

<input type="checkbox"/>	Policy name	Type	Attached entities
<input type="checkbox"/>	AmazonVPCFullAccess	AWS managed	2

iii. Review your User Permissions and Policies

Review your choices. After you create the user, you can view and download the autogenerated password, if enabled.

### User details

User name Test	Console password type Autogenerated	Require password reset Yes
-------------------	--	-------------------------------

### Permissions summary

< 1 >

Name	Type	Used as
<a href="#">IAMUserChangePassword</a>	AWS managed	Permissions policy

### Tags - optional

Tags are key-value pairs you can add to AWS resources to help identify, organize, or search for resources. Choose any tags you want to associate with this user.

No tags associated with the resource.

[Add new tag](#)

You can add up to 50 more tags.

Cancel Previous **Create user**

iv. Select "Create User" in the bottom-right corner and your user will be created.

v. Retrieve Login URL and Password

### Retrieve password

You can view and download the user's password below or email users instructions for signing in to the AWS Management Console. This is the only time you can view and download this password.

#### Console sign-in details

[Email sign-in instructions](#)

Console sign-in URL  
<https://800548176231.signin.aws.amazon.com/console>

User name  
 Test

Console password  
 \*\*\*\*\* [Show](#)

### 3. Provide programmatic Access to your user

- a. Go to IAM and then users again
- b. Select your created user
- c. Select "Security Credentials"

IAM > Users > Test

### Test

[Delete](#)

#### Summary

ARN <a href="#">arnaws:iam::800548176231:user/Test</a>	Console access <a href="#">Enabled without MFA</a>	Access key 1 <a href="#">Create access key</a>
Created February 16, 2024, 17:53 (UTC+05:30)	Last console sign-in <a href="#">Never</a>	

Permissions | Groups | Tags | **Security credentials** | Access Advisor

d. Go to "Access Keys" in your Security Credentials and Choose "Create Access Key"

### Access keys (0)

[Create access key](#)

Use access keys to send programmatic calls to AWS from the AWS CLI, AWS Tools for PowerShell, AWS SDKs, or direct AWS API calls. You can have a maximum of two access keys (active or inactive) at a time. [Learn more](#)

No access keys. As a best practice, avoid using long-term credentials like access keys. Instead, use tools which provide short term credentials. [Learn more](#)

[Create access key](#)

e. Go to the Use Case and select "AWS CLI"

Use case

**Command Line Interface (CLI)**  
You plan to use this access key to enable the AWS CLI to access your AWS account.

**Local code**  
You plan to use this access key to enable application code in a local development environment to access your AWS account.

**Application running on an AWS compute service**  
You plan to use this access key to enable application code running on an AWS compute service like Amazon EC2, Amazon ECS, or AWS Lambda to access your AWS account.

**Third-party service**  
You plan to use this access key to enable access for a third-party application or service that monitors or manages your AWS resources.

**Application running outside AWS**  
You plan to use this access key to authenticate workloads running in your data center or other infrastructure outside of AWS that needs to access your AWS resources.

**Other**  
Your use case is not listed here.

f. Click Next and then "Create Access Key"

g. You will get your Programmatic Access Keys here

Retrieve access keys Info

**Access key**  
If you lose or forget your secret access key, you cannot retrieve it. Instead, create a new access key and make the old key inactive.

Access key: AKIASUZDDVFTZHLRYLOT

Secret access key: \*\*\*\*\* [Show](#)

**Access key best practices**

- Never store your access key in plain text, in a code repository, or in code.
- Disable or delete access key when no longer needed.
- Enable least-privilege permissions.
- Rotate access keys regularly.

For more details about managing access keys, see the [best practices for managing AWS access keys](#).

[Download .csv file](#) [Done](#)

4. Connect to your AWS ec2 instance which you have created in the Step-1 ([Ref.](#))

5. Once you are logged in to your instance, then Install the following tools

a. **Install Unzip**

```
1 $sudo apt install
```

b. **Configure AWS CLI**

```
1 $curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
2 $unzip awscliv2.zip
3 $sudo ./aws/install
```

6. Now, Configure AWS CLI with the following commands and the programmatic access keys created in Step -3(g)

```
1 $aws configure
2 #enter the Access key ID and Secret access key.
3 #Provide the region details i.e., us-east-1 or any other
4 #Give output format as "json".
```

```
5 #Generate public and private keys
6 $ssh-keygen
```

## 7. Install Helm

```
1 $curl -fsSL -o get_helm.sh https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3
2 $chmod 700 get_helm.sh
3 $./get_helm.sh
```

## 8. Install kubectl and eksctl (tools to manage and interact with the kubernetes cluster)

### a. Installing latest version of kubectl

```
1 $curl -LO"https://dl.k8s.io/release/$(curl -L -shttps://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubec
2 #make the downloaded file executable
3 $chmod +x kubectl
4 #Move the executable to the /usr/local/bin
5 $sudo mv kubectl /usr/local/bin
```

### b. Installing the latest version of eksctl:

```
1 #for ARM systems, set ARCH to: arm64, armv6 or armv7
2 $ARCH=amd64
3 $PLATFORM=$(uname -s)_$ARCH
4 $curl -sLO "https://github.com/eksctl-io/eksctl/releases/latest/download/eksctl_${PLATFORM}.tar.gz"
5 #(Optional) Verify checksum
6 $curl -sL "https://github.com/eksctl-io/eksctl/releases/latest/download/eksctl_checksums.txt" | grep $PLATF
7 $tar -xzf eksctl_${PLATFORM}.tar.gz -C /tmp && rm eksctl_${PLATFORM}.tar.gz
8 $sudo mv /tmp/eksctl /usr/local/bin
```

## 9. Creating the cluster with 3 worker node and 1 master node with the below command:

```
1 $eksctl create cluster --name test-cluster --version 1.29 --region eu-west-1 --nodegroup-name linux-nodes --no
```

```
ubuntu@ip-10.0.0.10:~$ eksctl create cluster --name test-cluster --version 1.29 --region eu-west-1 --nodegroup-name linux-nodes --node-type m4.
large --nodes 3
2024-02-20 10:06:35 [0] eksctl version 0.171.0
2024-02-20 10:06:35 [0] using region eu-west-1
2024-02-20 10:06:35 [0] setting availability zones to [eu-west-1b eu-west-1c eu-west-1a]
2024-02-20 10:06:35 [0] subnets for eu-west-1b - public:192.168.0.0/19 private:192.168.96.0/19
2024-02-20 10:06:35 [0] subnets for eu-west-1c - public:192.168.32.0/19 private:192.168.128.0/19
2024-02-20 10:06:35 [0] subnets for eu-west-1a - public:192.168.64.0/19 private:192.168.160.0/19
2024-02-20 10:06:35 [0] nodegroup "linux-nodes" will use "" [AmazonLinux2/1.29]
2024-02-20 10:06:35 [0] using Kubernetes version 1.29
2024-02-20 10:06:35 [0] creating EKS cluster "test-cluster" in "eu-west-1" region with managed nodes
2024-02-20 10:06:35 [0] will create 2 separate CloudFormation stacks for cluster itself and the initial managed nodegroup
2024-02-20 10:06:35 [0] if you encounter any issues, check CloudFormation console or try 'eksctl utils describe-stacks --region=eu-west-1 --clust
er=test-cluster'
2024-02-20 10:06:35 [0] Kubernetes API endpoint access will use default of {publicAccess=true, privateAccess=false} for cluster "test-cluster" in
"eu-west-1"
2024-02-20 10:06:35 [0] CloudWatch logging will not be enabled for cluster "test-cluster" in "eu-west-1"
2024-02-20 10:06:35 [0] you can enable it with 'eksctl utils update-cluster-logging --enable-types={SPECIFY-YOUR-LOG-TYPES-HERE (e.g. all)} --reg
ion=eu-west-1 --cluster=test-cluster'
2024-02-20 10:06:35 [0]
2 sequential tasks: { create cluster control plane "test-cluster",
  2 sequential sub-tasks: {
    wait for control plane to become ready,
    create managed nodegroup "linux-nodes",
  }
}
2024-02-20 10:06:35 [0] building cluster stack "eksctl-test-cluster-cluster"
2024-02-20 10:06:36 [0] deploying stack "eksctl-test-cluster-cluster"
2024-02-20 10:07:06 [0] waiting for cloudFormation stack "eksctl-test-cluster-cluster"
```

It will take around 10 to 15 mins for cluster to be ready to use. After the said time you can check the status of the cluster by running the below command:

```
1 $eksctl get cluster
```

```
ubuntu@ip-10-100-96-178:~$ eksctl get cluster
NAME          REGION    EKSTL CREATED
test-cluster  eu-west-1 True
```

When the cluster is ready with 3 node machines running in eu-west-1 region and 1 master running in eu-west-1 as per the availability zones.

#### 10. Install cert-manager and its CRD's

a. Add and update the Jetstack Helm repository

```
1 $helm repo add jetstack https://charts.jetstack.io --force-update
```

b. Install the CRD's( Custom Resource Definition) of Certmanager using the following command

```
1 $kubectl apply -f https://github.com/cert-manager/cert-manager/releases/download/v1.13.3/cert-manager.crds
```

c. Install the Latest cert-manager using helm

```
1 $helm install cert-manager jetstack/cert-manager --namespace cert-manager --create-namespace --version v1.13.3
```

d. Now, Install the GlobalSign's Certmanager-Atlas CRD. Once it is installed, then it is ready to handle Atlas Certificate requests.

```
1 $kubectl apply -f https://github.com/globalsign/atlas-cert-manager/releases/download/v0.0.1/install.yaml
```

#### 11. Label the cert-manager namespace to disable resource validation

```
1 $kubectl label namespace cert-manager certmanager.k8s.io/disable-validation=true
```

#### 12. Now, Install Nginx-ingress-controller in namespace cert-manager

```
1 $helm upgrade --install ingress-nginx ingress-nginx --repo https://kubernetes.github.io/ingress-nginx --namespace cert-manager
2 $kubectl get svc -n cert-manager
```

#### 13. Create A record in your Route 53 to the Hosted Zone for the below created Load Balancer IP(Here the cluster IP is 10.100.96.178)

---

```

ubuntu@ip-:~$ kubectl get svc -n cert-manager
NAME                                TYPE                CLUSTER-IP      EXTERNAL-IP      PORT(S)          POR
cert-manager                        ClusterIP           10.100.241.61   <none>           940
cert-manager-webhook                ClusterIP           10.100.246.181 <none>           443
ingress-nginx-controller            LoadBalancer       10.100.96.178   a99d49e4a1a62409689a287279f59e79-2014922034.eu-west-1.elb.amazonaws.com 80:
ingress-nginx-controller-admission ClusterIP           10.100.219.192 <none>           443

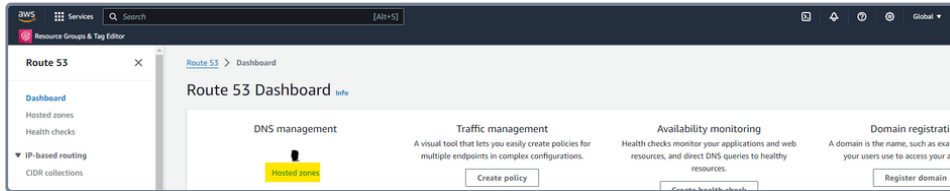
```

14. As soon as the ingress-nginx-controller get the EXTERNAL-IP value with extension \*.eu-west-1.elb.amazonaws.com, Add this value as A record into hosted zone. It would be in the sync within 60sec.

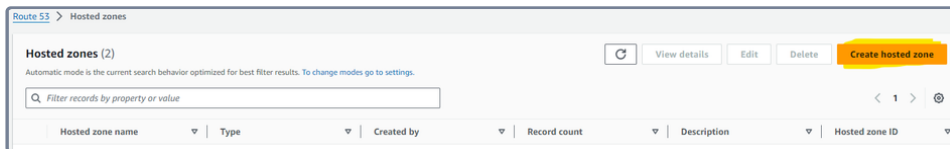
Note:- Before creating the hosted zone kindly make sure you have the valid domain.

15. Creating A record over AWS Route53

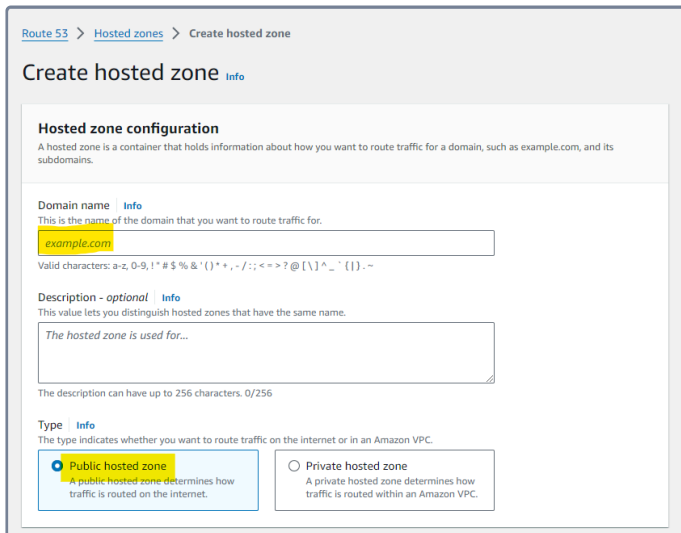
a. Go to <https://us-east-1.console.aws.amazon.com/route53/v2/home?region=eu-west-1#Dashboard> and click on "Hosted zones".



b. Click on "Create hosted zone"



c. Enter the name followed by your actual domain name and make sure the "Public hosted zone" should be selected:

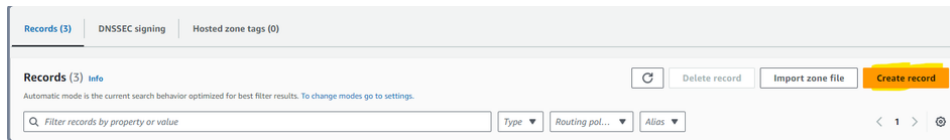


d. After creating the hosted zone, you would get some NS record along with SOA record. Now add the NS records into your domain registrar

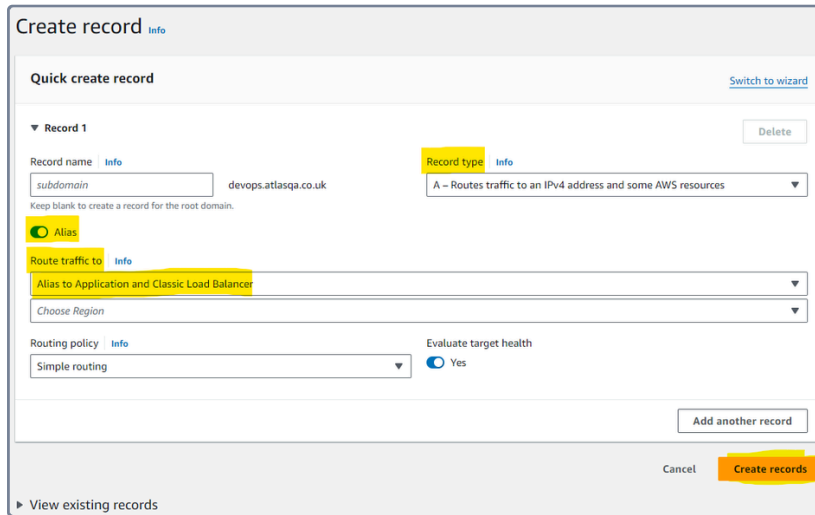
e. After adding the NS into domain registrar your hosted zone is now ready to accept traffic, Now you can create the A record into the hosted zone:

i. Into your hosted zone, Click on "Create record"





- ii. On the next screen make sure that Record Type is "A" and "Alias" are selected. Also make sure that "Route traffic to" "Alias to Application and Classic Load Balancer" is selected. After selecting the required fields click on "Create records":



- iii. The record would be created and it would take around 60sec to get in the sync.

16. Create GlobalSign Issuer to issue a TLS certificate for your Ingress using the following steps:-

- a. Create a secret to store the GlobalSign's ATLAS account api\_key, secrets along with mTLS and private key(You can get these API credentials from GlobalSign's Team)

```
1 $kubectl create secret generic issuer-credentials --from-literal=apikey=$API_KEY --from-literal=apisecret=$
```

b. Create an Issuer of GlobalSign.

```
1 cat <<EOF | kubectl apply -f -
2 apiVersion: hvca.globalsign.com/v1alpha1
3 kind: Issuer
4 metadata:
5   name: gs-issuer
6   namespace: cert-manager
7 spec:
8   authSecretName: "issuer-credentials"
9   url: "https://emea.api.hvca.globalsign.com:8443/v2"
10 EOF
```

c. Create Certificate Resource with the following Configuration

```
1 cat <<EOF | kubectl apply -f -
2 apiVersion: cert-manager.io/v1
3 kind: Certificate
```

```

4 metadata:
5   name: pki.atlasqa.co.uk
6   namespace: cert-manager
7 spec:
8   # Secret names are always required.
9   secretName: www.atlasqa.co.uk
10
11   duration: 2160h # 90d
12   renewBefore: 360h # 15d
13   subject:
14     # organizations:
15     #   - jetstack
16     # The use of the common name field has been deprecated since 2000 and is
17     # discouraged from being used.
18     commonName: pki.atlasqa.co.uk
19     isCA: false
20   privateKey:
21     algorithm: RSA
22     encoding: PKCS1
23     size: 2048
24   usages:
25     - server auth
26     #- client auth
27   # At least one of a DNS Name, URI, or IP address is required.
28   # dnsNames:
29   #   -
30   #www.atlasqa.co.uk
31   # Issuer references are always required.
32   issuerRef:
33     name: gs-issuer
34     # We can reference ClusterIssuers by changing the kind here.
35     # The default value is Issuer (i.e. a locally namespaced Issuer)
36     kind: Issuer
37     # This is optional since cert-manager will default to this value however
38     # if you are using an external issuer, change this to that issuer group.
39     group: hvca.globalsign.com
40 EOF

```

d. At times the certificate object can take couple of seconds to become READY.

```

ubuntu@ip-10-0-1-10:~$ kubectl apply -f object-cert.yml
certificate.cert-manager.io/pki.atlasqa.co.uk created
ubuntu@ip-10-0-1-10:~$ kubectl get certificate -n cert-manager
NAME                READY    SECRET                AGE
pki.atlasqa.co.uk  True    www.atlasqa.co.uk    4s

```

17. Securing nginx ingress resource by the below configuration:

```

1 cat <<EOF | kubectl apply -f -
2 apiVersion: networking.k8s.io/v1
3 kind: Ingress
4 metadata:
5   name: nginx
6   namespace: cert-manager
7 annotations:

```

```

8     cert-manager.io/issuer: GS-issuer
9     kubernetes.io/ingress.class: nginx
10 spec:
11   tls:
12     - hosts:
13       - pki.atlasqa.co.uk
14       secretName: www.atlasqa.co.uk
15   rules:
16     - host: pki.atlasqa.co.uk
17     http:
18       paths:
19         - path: /
20         pathType: Prefix
21       backend:
22         service:
23           name: example-service
24           port:
25             number: 80
26 EOF

```

18. The ingress resource that has been created could take up to 1min to get the load balancer URL as ADDRESS.

```

ubuntu@ip-10-0-1-10:~$ kubectl apply -f ingress.yml
Warning: annotation "kubernetes.io/ingress.class" is deprecated, please use 'spec.ingressClassName' instead
ingress.networking.k8s.io/nginx-ingress created
ubuntu@ip-10-0-1-10:~$ kubectl get ingress -n cert-manager
NAME          CLASS  HOSTS                                ADDRESS                                                                 PORTS  AGE
nginx-ingress <none>  devops.atlasqa.co.uk               a215bae5711314b3cb9a9bd1556950fb-1850122743.us-east-2.elb.amazonaws.com 80, 443 87s

```

19. Attaching the screen shot:

