Automated SSL Termination in EKS with Cert Manager(Venafi-Cloud)

Jaswanth Kumar Jonnalagadda

Amazon Web Services

October 2019

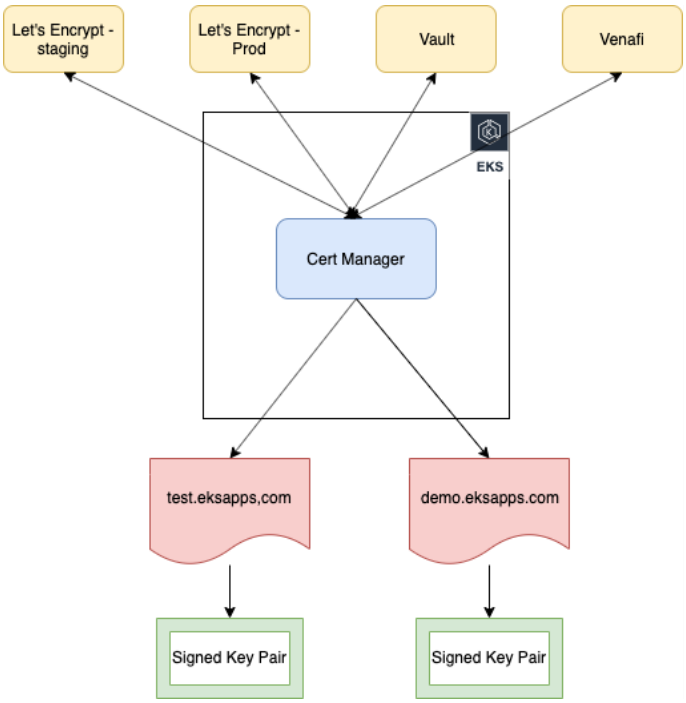
Table of Contents

Kubernetes Cert-Manager:

cert-manager is a native Kubernetes certificate management controller.

cert-manager can help with issuing certificates from a variety of sources, such as Let's Encrypt, HashiCorp Vault, Venafi, a simple signing keypair, or self-signed. cert-manager will ensure certificates are valid and up to date, and attempt to renew certificates at a configured time before expiry.

Architecture:



Pre-Requisite:

Following are the expected perquisites for the cert manager to issue certificate dynamically.

1. Fully functional EKS cluster.
2. Domain of your own.

Installing Cert-Manager:

Follow the below steps to install cert-manager in the EKS cluster.

1. Create a separate namespace in the EKS cluster for cert-manager

⇒ **kubectl create namespace cert-manager**

```
cat <<EOF > sample-resources.yaml
apiVersion: v1
kind: Namespace
metadata:
  name: cert-manager-check
---
apiVersion: cert-manager.io/v1alpha2
kind: Issuer
metadata:
  name: test-selfsigned
  namespace: cert-manager-test
spec:
  selfSigned: {}
---
apiVersion: cert-manager.io/v1alpha2
kind: Certificate
metadata:
  name: selfsigned-cert
  namespace: cert-manager-test
spec:
  commonName: example.com
  secretName: selfsigned-cert-tls
  issuerRef:
    name: test-selfsigned
EOF
```

2. Install cert manager in the namespace with the below command.
   ⇒ **kubectl apply -f https://github.com/jetstack/cert-manager/releases/download/v0.11.0/cert-manager.yaml --validate=false**

** Reason for the –validate=false tag is to overcome the way kubectl performs resource validation on kubernetes version less that 1.15

3. Verify Installation with the below command
   ⇒ **kubectl get pods --namespace cert-manager**
   ⇒ If everything went well we will see three pods in running state as shown
   ⇒ To view the custom resources that are deployed along with cert-manager, run the following command: **kubectl get crd --all-namespaces**

Verifying Functionality:

1. Build a sample self-signed certificate issuer in the cluster with the following script

2. Execute the script with the command
   ⇒ **Kubectl apply -f sample-resources.yaml**

3. Wait for few seconds for the cert-manager to process the certificate request.
   ⇒ Execute the command to describe the generated certificate
   ⇒ **Kubectl describe cert -n cert-manager-test**

   You should see a similar output

With this we can confirm that the cert manager is installed without any errors.

Clean up test resources with the command: **kubectl delete -f sample-resources.yaml**

Configuring Issuer:

Before you can begin issuing certificates, you must configure at least one Issuer or ClusterIssuer resource in your cluster.

These represent a certificate authority from which signed x509 certificates can be obtained, such as Let's Encrypt, or your own signing key pair stored in a Kubernetes Secret resource. They are referenced by Certificate resources in order to request certificates from them.

An Issuer is scoped to a single namespace, and can only fulfill Certificate resources within its own namespace. This is useful in a multi-tenant environment where multiple teams or independent parties operate within a single cluster.

On the other hand, a ClusterIssuer is a cluster wide version of an Issuer. It is able to be referenced by Certificate resources in any namespace.

In this article we will use Venafi cloud to explain issuer for a namespace scope

Configuring with Venafi Cloud Issuer:

1. Deploy an ingress-nginx using an ELB to expose the service.
   Run the following commands to deploy the ingress controller.
   ⇒ kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/master/deploy/static/mandatory.yaml
   ⇒ kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/master/deploy/static/provider/aws/service-nlb.yaml

** It will take few minutes for the ingress controller to be up.

2. Verify the deployed service with the command: **kubectl get service -n ingress-nginx**
   Sample Output:

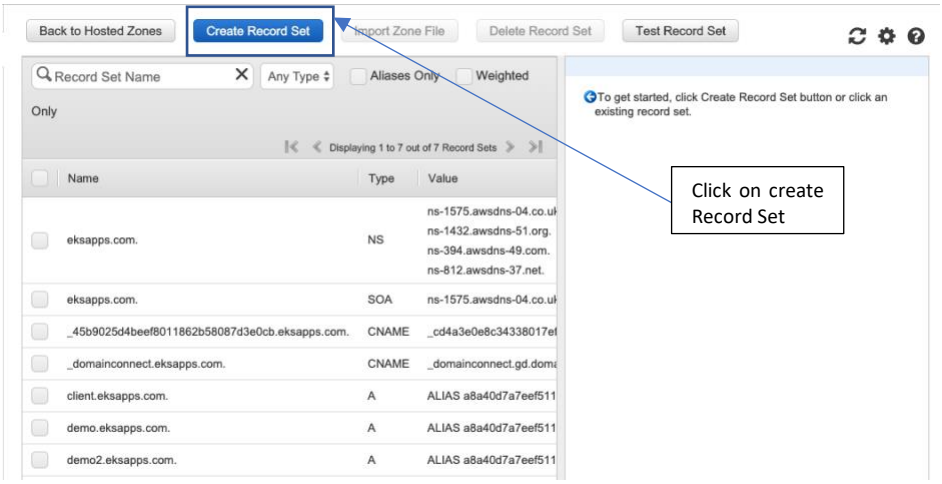
```
Genesis:~/environment $ kubectl get svc -n ingress-nginx
NAME            TYPE           CLUSTER-IP      EXTERNAL-IP                                                                PORT(S)                      AGE
ingress-nginx   LoadBalancer   172.20.247.63   a8a40d7a7eef511e9a9320e7de11db72-b1e2865cb8933f90.elb.us-east-1.amazonaws.com   80:30084/TCP,443:31856/TCP   3d12h
```

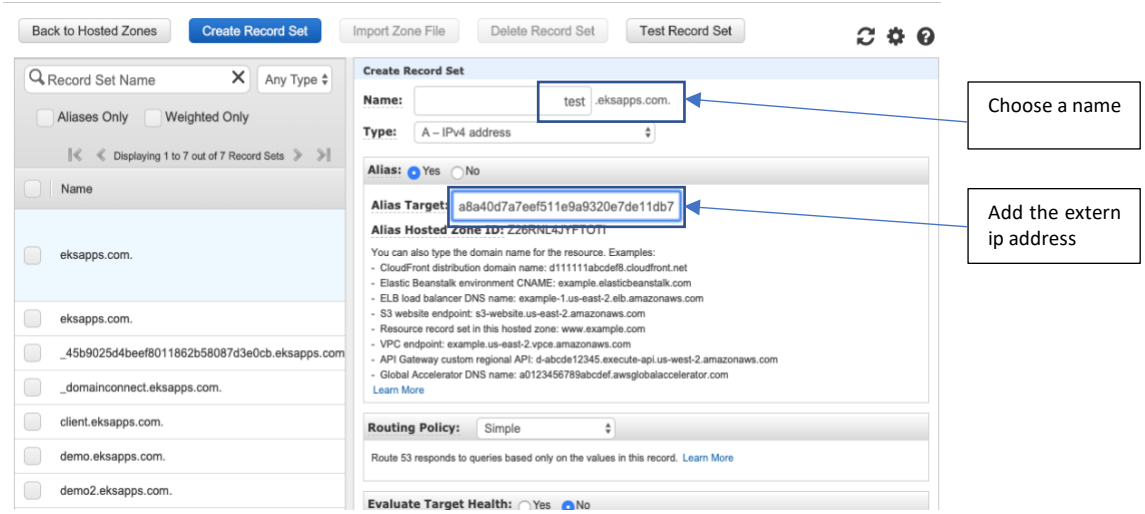
** If the external-ip is not available, please wait for few minutes for the address to be issued.

3. Once the external ip is issued, then verify if the traffic is being routed to the ingress-nginx
   Command: **curl http:// a8a40d7a7eef511e9a9320e7de11db72-b1e2865cb8933f90.elb.us-east-1.amazonaws.com**
   Sample Output:

```
Genesis:~/environment $ curl http://a8a40d7a7eef511e9a9320e7de11db72-b1e2865cb8933f90.elb.us-east-1.amazonaws.com
<html>
<head><title>404 Not Found</title></head>
<body>
<center><h1>404 Not Found</h1></center>
<hr><center>openresty/1.15.8.2</center>
</body>
</html>
```

4. Now that our NLB has been provisioned, we should point our application's DNS records at the NLBs address. In the DNS provider's console set an A record to pointing to your NLB external ip.





⇒ Click on Create button.

⇒ This will create a new entry in the DNS record set.

⇒ This will resemble following

test.eksapps.com A http://a8a40d7a7eef511e9a9320e7de11db72-b1e2865cb8933f90.elb.us-east-1.amazonaws.com

5. Create a namespace demo
   ⇒ Command: kubectl create namespace demo
6. Deploy a sample application in the demo namespace with the below deployment script

```
---
apiVersion: v1
kind: Service
metadata:
  name: hello-kubernetes
  namespace: demo
spec:
  type: ClusterIP
  ports:
  - port: 80
    targetPort: 8080
  selector:
    app: hello-kubernetes
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hello-kubernetes
  namespace: demo
spec:
  replicas: 2
  selector:
    matchLabels:
      app: hello-kubernetes
  template:
    metadata:
      labels:
        app: hello-kubernetes
    spec:
      containers:
      - name: hello-kubernetes
        image: '682651395775.dkr.ecr.us-east-1.amazonaws.com/java_app_one:latest'
        resources:
          requests:
            cpu: 100m
            memory: 100Mi
        ports:
        - containerPort: 8080
```

Command: **kubectl apply -f demo-application.yml -n demo**

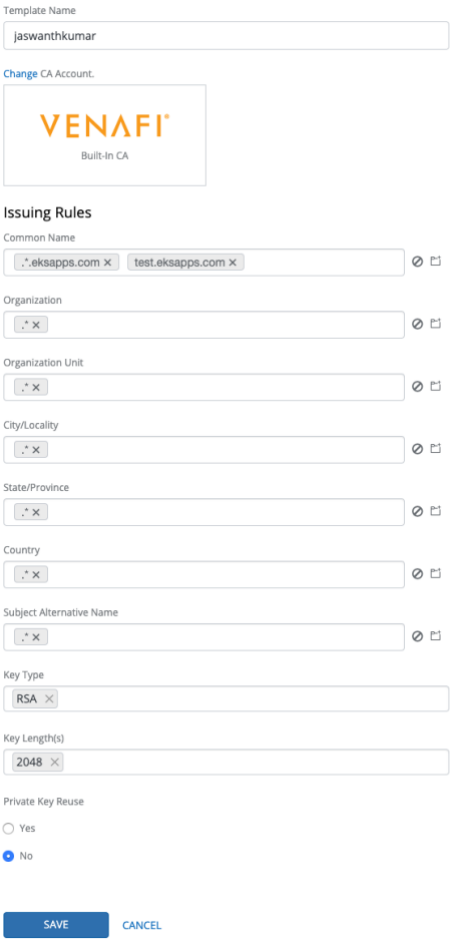
7. Verify the application deployment with the below command
   ⇒ **Kubectl get po,svc -n demo**

Sample output:

```
Genesis:~/environment $ kubectl get po,svc -n demo
NAME                          READY   STATUS    RESTARTS   AGE
pod/appd-6d45d68d8-2tzvx      1/1     Running   0          3d
pod/appd-6d45d68d8-47ws9      1/1     Running   0          3d
pod/appd-6d45d68d8-vjk7f      1/1     Running   0          3d
pod/appd2-7fccff49bb-6ns45    1/1     Running   0          2d1h
pod/appd2-7fccff49bb-7l6g8    1/1     Running   0          2d1h
pod/appd2-7fccff49bb-b5ksq    1/1     Running   0          2d1h
pod/client-7694bdf5b9-25mpf   1/1     Running   0          25h
pod/client-7694bdf5b9-44vzx   0/1     Pending   0          25h
pod/client-7694bdf5b9-7gn6c   1/1     Running   0          25h
pod/client-7694bdf5b9-wt8g2   0/1     Pending   0          25h

NAME              TYPE       CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
service/appd      NodePort   172.20.85.94    <none>        80:32061/TCP   3d
service/appd2     NodePort   172.20.27.235   <none>        80:31261/TCP   2d1h
service/client    NodePort   172.20.14.93    <none>        80:30269/TCP   25h
```

8. Create Venafi Cloud account
   - ⇒ Visit the url: https://ui.venafi.cloud/enroll
   - ⇒ Create an account with corporate email and login to dashboard
   - ⇒ Choose Configuration from the left plane
     - i. Click on issuing Templates
     - ii. Create a new temaplate
     - iii. Sample filled template as shown
     - iv. Choose built in CA for ease of use.
     - v. Common Name must include your domain name.



vi. Click on save.

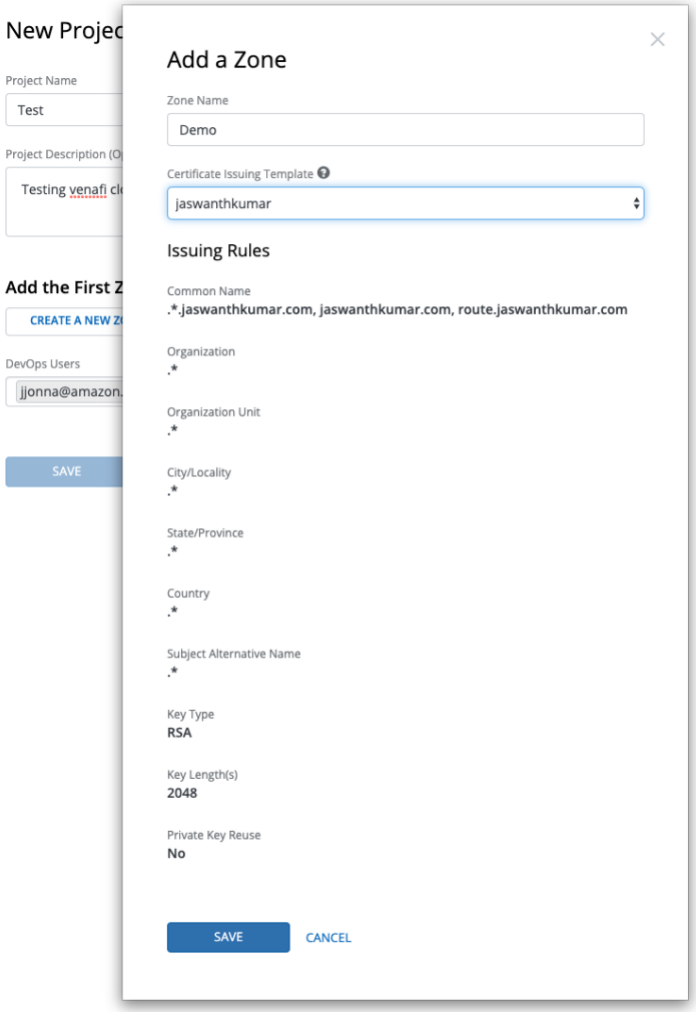
⇒ Click on Projects and choose “Create New Project”

i. Enter Project Name, Description

ii. Add a Zone

1. Choose a name for the zone
2. Choose the the previously created template

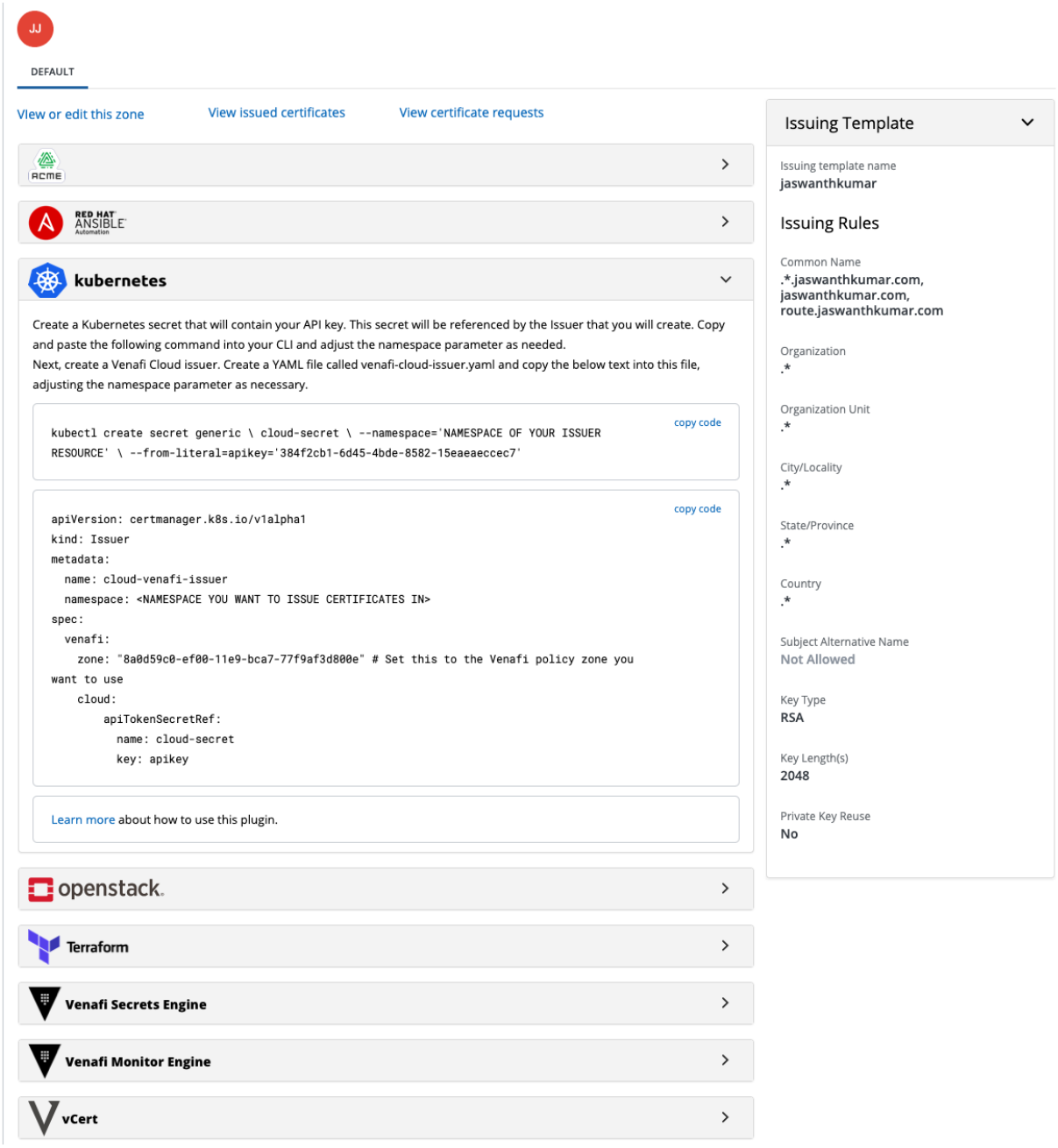
Sample zone:



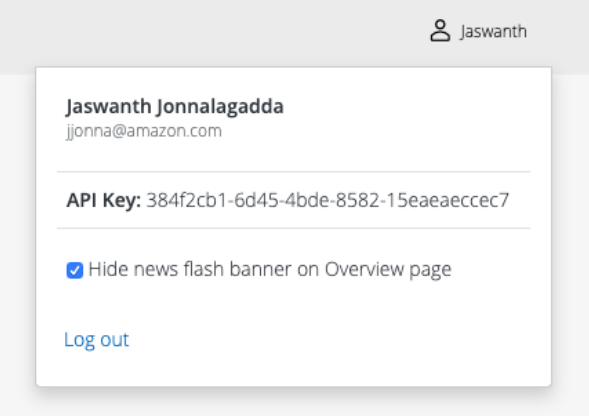
3. Click on save
4. And choose a user for DevOps user
5. Create the project.

9. In the projects plane choose the project created

⇒ Choose kubernetes to view the configuration for K8



10. With this we had set up a cloud certificate issuer(Venafi Cloud)
11. Once registered, you should fetch your API key by clicking your name in the top right of the control panel interface.



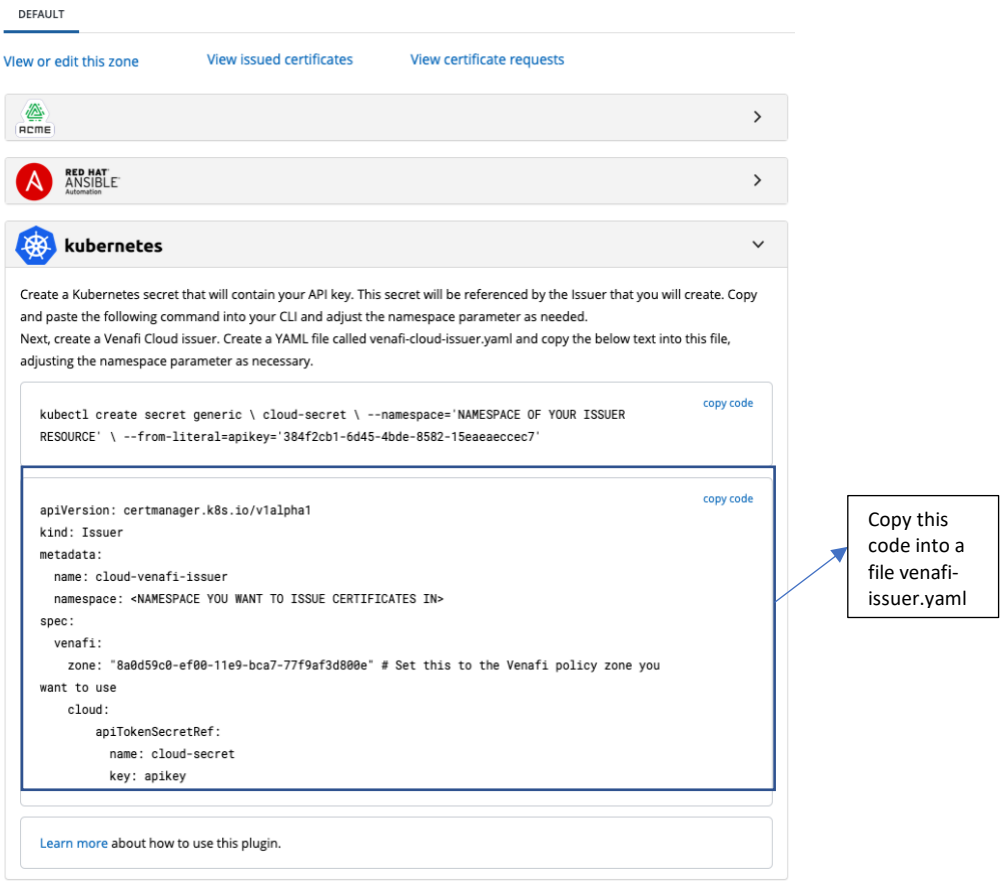
12. In order for cert-manager to be able to authenticate with your Venafi Cloud account and set up an Issuer resource, you'll need to create a Kubernetes Secret containing your API key

    ⇒ kubectl create secret generic \
    venafi-cloud-secret \
    --namespace=demo \
    --from-literal=apikey=**<API_KEY>**

    Replace **<API_KEY> with API key from venafi cloud account.

13. Create a Venafi certificate issuer with the script below

    ⇒ Copy the configuration code from the project and zone you want to use.



14. Replace the secret in the file with the secret generated above.
15. Run the script to deploy the certificate issuer with the command
    ⇒ **Kubectl apply -f venafi-issuer.yaml -n demo**
16. Verify the issuer installation with the command
    ⇒ **Kubectl describe issuer cloud-venafi-issuer -n demo**

```
Genesis:~/environment $ kubectl describe issuer cloud-venafi-issuer -n demo
Name:         cloud-venafi-issuer
Namespace:    demo
Labels:       <none>
Annotations:  kubectl.kubernetes.io/last-applied-configuration:
                {"apiVersion":"cert-manager.io/v1alpha2","kind":"Issuer","metadata":{"annotations":{},"name":"cloud-venafi-issuer","namespace":"demo"},"sp...
API Version:  cert-manager.io/v1alpha2
Kind:         Issuer
Metadata:
  Creation Timestamp:  2019-10-15T04:04:26Z
  Generation:          1
  Resource Version:    58135
  Self Link:           /apis/cert-manager.io/v1alpha2/namespaces/demo/issuers/cloud-venafi-issuer
  UID:                 e1258305-ef00-11e9-8dc9-02e77e9a27d8
Spec:
  Venafi:
    Cloud:
      API Token Secret Ref:
        Key:   apikey
        Name:  venafi-cloud-secret
      URL:
    Zone:      8a0d59c0-ef00-11e9-bca7-77f9af3d800e
Status:
  Conditions:
    Last Transition Time:  2019-10-15T04:04:26Z
    Message:               Venafi issuer started
    Reason:                Venafi issuer started
    Status:                True
    Type:                  Ready
Events:                    <none>
```

This shows that the issuer had successfully validated itself with the Venafi Cloud service.

17. As the issuer is configured correct, we can now issue a certificate
    ⇒ Create a yaml script that can issue a certificate for the domain 'test.eksapps.com'
    ** Replace test.eksapps.com with a domain you own.
    ⇒ Save the yaml into a file named eksapps-com-tls.yaml

```
apiVersion: cert-manager.io/v1alpha2
kind: Certificate
metadata:
  name: testeksapp-com-tls
  namespace: demo
spec:
  secretName: testeksapp-com-tls
  dnsNames:
  - test.eksapps.com
  issuerRef:
    name: venafi-issuer
    Kind: Issuer
```

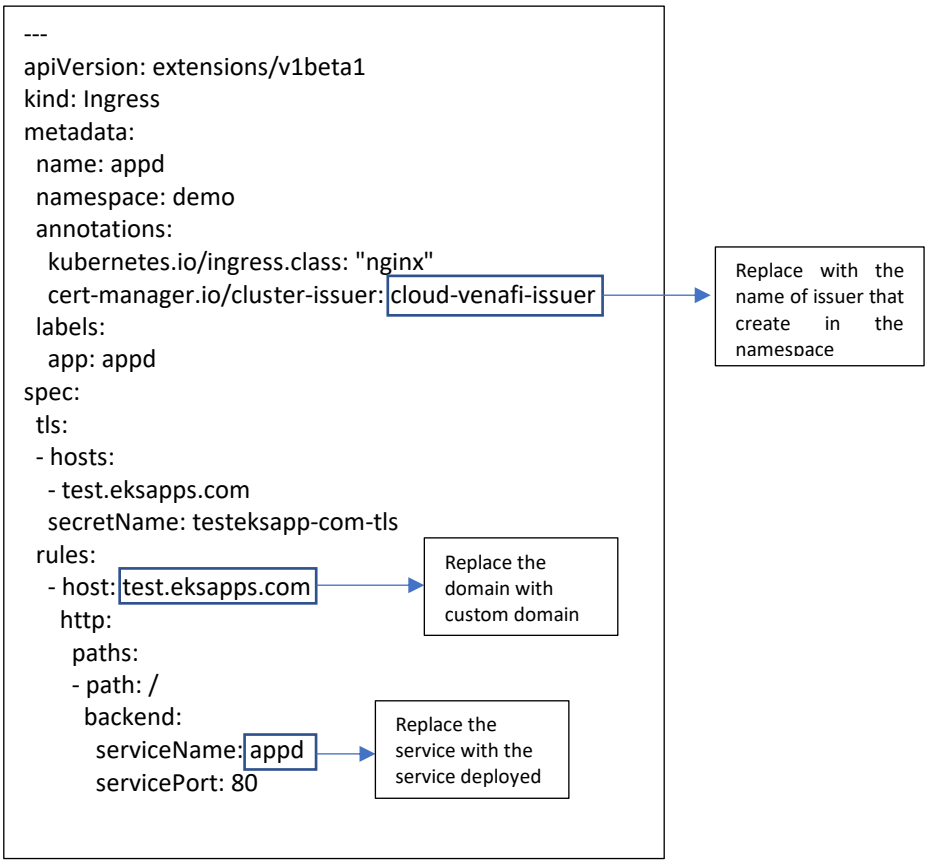
    ⇒ Run the script with the command
    Kubectl apply -f eksapps-com-tls.yaml -n demo

```
Status:
  Conditions:
    Last Transition Time:  2019-10-18T18:21:49Z
    Message:               Certificate is up to date and has not expired
    Reason:                Ready
    Status:                True
    Type:                  Ready
  Not After:               2019-10-25T18:21:47Z
Events:
  Type    Reason        Age    From          Message
  ----    ------        ----   ----          -------
  Normal  GeneratedKey  3m21s  cert-manager  Generated a new private key
  Normal  Requested     3m21s  cert-manager  Created new CertificateRequest resource "jaswanthkumar-com-2596748224"
  Normal  Issued        3m19s  cert-manager  Certificate issued successfully
```

This shows that cert manager had issued a certificate and it is ready to be consumed

18. Now we can expose the application with the kubernetes ingress resource

⇒ Create a file named application-ingress.yaml



With this we had completely automated the certificate issuing inside EKS cluster.

Now we can login to the browser and reach the website over https

https://test.eksapps.com