Setting up NGINX Ingress Controller with EKS and Deploying a Spring Boot Application using Domain over HTTPS.

- **Create an IAM Role ⇒**
  - **Open IAM**
  - **Click on Roles**
  - **Create Role**
  - **Trusted Entity Type ⇒ AWS Service**
  - **Service or Use Case ⇒ EC2**
  - **Attach policies ⇒ For this we need AmazonEKSClusterPolicy, AmazonEKSServicePolicy**
  - **Give the Role a name and then, we have successfully created a Iam role which we can attach it to EC2 instance.**

- **Pre-requisites**
  - **Start an EC2 instance with the following properties.**
  - **AMI ⇒ Amazon-linux-2**
  - **Node Type ⇒ t2.medium**
  - **Key Pair ⇒ your-key-value-pair**
  - **Under Network Settings, look for Security group rules**
  - **Add security group rules for 80, 443, 8080 (Spring Application Port)**
  - **Under Configure Storage instead of 8 make it 25 and then create the Instance.**
  - **Once the instance is ready and Instance state is green click on actions ⇒ security ⇒ modify Iam Role ⇒ attach the EKS role we created.**
  - **ssh and connect to the instance using the following command**
    ssh -i <your-pem-file-location> ex2-user@<your-public-ip>

- **Now we create a shell script for the dependencies needed to start a EKS cluster**
  - **Eksctl**
  - **Kubectl**
  - **Helm**
  - **Java**
  - **Awscli2**
  - **Docker**

vi package.sh

```
# eksctl installation

# for ARM systems, set ARCH to: `arm64`, `armv6` or `armv7`

ARCH=amd64

PLATFORM=$(uname -s)_$ARCH

curl -sLO
"https://github.com/eksctl-io/eksctl/releases/latest/download/eksctl_$PLATFORM.tar.
gz"

# (Optional) Verify checksum

curl -sL
"https://github.com/eksctl-io/eksctl/releases/latest/download/eksctl_checksums.txt" |
grep $PLATFORM | sha256sum --check

tar -xzf eksctl_$PLATFORM.tar.gz -C /tmp && rm eksctl_$PLATFORM.tar.gz

sudo mv /tmp/eksctl /usr/local/bin

# kubectl installation

curl -LO "https://dl.k8s.io/release/$(curl -L -s
https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"

curl -LO "https://dl.k8s.io/release/$(curl -L -s
https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl.sha256"

echo "$(cat kubectl.sha256)  kubectl" | sha256sum --check

sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl

kubectl version --client

#awscli2 installation
```

```
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"

unzip awscliv2.zip

sudo ./aws/install --bin-dir /usr/local/bin --install-dir /usr/local/aws-cli --update

aws --version
curl -fsSL -o get_helm.sh
https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3
chmod 700 get_helm.sh
./get_helm.sh

# Docker installation

sudo yum install docker -y

sudo systemctl start docker

# java installation

wget https://download.oracle.com/java/17/latest/jdk-17_linux-x64_bin.rpm

sudo rpm -ivh jdk-17_linux-x64_bin.rpm -y

java --version
```

Make the shell script executable

```
chmod +x package.sh

./package.sh
```

- **Create a eksctl cluster**
  - **eksctl create cluster --name demo --region us-east-1 --nodegroup-name demong --node-type t2.medium --managed --nodes 3**

```
[ec2-user@ip-172-31-94-61 ~]$ eksctl create cluster --name demo --region us-east-1 --nodegroup-name demong --node-type t2.medium --managed
--nodes 3 --nodes-min 1 --nodes-max 4
2024-10-01 23:18:34 [ℹ]  eksctl version 0.191.0
2024-10-01 23:18:34 [ℹ]  using region us-east-1
2024-10-01 23:18:34 [ℹ]  setting availability zones to [us-east-1c us-east-1d]
2024-10-01 23:18:34 [ℹ]  subnets for us-east-1c - public:192.168.0.0/19 private:192.168.64.0/19
2024-10-01 23:18:34 [ℹ]  subnets for us-east-1d - public:192.168.32.0/19 private:192.168.96.0/19
2024-10-01 23:18:34 [ℹ]  nodegroup "demong" will use "" [AmazonLinux2/1.30]
2024-10-01 23:18:34 [ℹ]  using Kubernetes version 1.30
2024-10-01 23:18:34 [ℹ]  creating EKS cluster "demo" in "us-east-1" region with managed nodes
2024-10-01 23:18:34 [ℹ]  will create 2 separate CloudFormation stacks for cluster itself and the initial managed nodegroup
2024-10-01 23:18:34 [ℹ]  if you encounter any issues, check CloudFormation console or try 'eksctl utils describe-stacks --region=us-east-1
--cluster=demo'
2024-10-01 23:18:34 [ℹ]  Kubernetes API endpoint access will use default of {publicAccess=true, privateAccess=false} for cluster "demo" in
"us-east-1"
2024-10-01 23:18:34 [ℹ]  CloudWatch logging will not be enabled for cluster "demo" in "us-east-1"
2024-10-01 23:18:34 [ℹ]  you can enable it with 'eksctl utils update-cluster-logging --enable-types={SPECIFY-YOUR-LOG-TYPES-HERE (e.g. all)
} --region=us-east-1 --cluster=demo'
2024-10-01 23:18:34 [ℹ]  default addons vpc-cni, kube-proxy, coredns were not specified, will install them as EKS addons
2024-10-01 23:18:34 [ℹ]
2 sequential tasks: { create cluster control plane "demo",
    2 sequential sub-tasks: {
        2 sequential sub-tasks: {
            1 task: { create addons },
            wait for control plane to become ready,
        },
        create managed nodegroup "demong",
    }
}
```

- **This will trigger in creating a cluster and takes about 15 to 20 min, Once that is done**
- **Connect to the cluster we just created**
- **eksctl get cluster –name demo –region us-east-1**

```
[ec2-user@ip-172-31-94-61 ~]$ eksctl get cluster --name demo --region us-east-1
NAME    VERSION STATUS  CREATED              VPC                   SUBNETS                                                                 S
ECURITYGROUPS          PROVIDER
demo    1.30    ACTIVE  2024-10-01T23:18:57Z   vpc-046f955ee9b3b9a1b   subnet-0051fbfd4b601c1fb,subnet-062bf4205f67956a1,subnet-0b6af8d033
f041005,subnet-0f095125624a017bf       sg-01f0c4be13e4f2e00    EKS
```

- **Check the nodes ```kubectl get nodes```**

```
[ec2-user@ip-172-31-94-61 ~]$ kubectl get nodes
NAME                           STATUS   ROLES    AGE     VERSION
ip-192-168-15-56.ec2.internal   Ready    <none>   2m35s   v1.30.4-eks-a737599
ip-192-168-48-120.ec2.internal  Ready    <none>   2m36s   v1.30.4-eks-a737599
ip-192-168-63-206.ec2.internal  Ready    <none>   2m38s   v1.30.4-eks-a737599
```

- **Creating a tls certificate for out domain**

  ```openssl req -x509 -newkey rsa:2048 -sha256 -nodes -keyout tls.key -out tls.crt -subj "/CN=spring.randomthat.com" -days 365```

```
[ec2-user@ip-172-31-94-61 ~]$ openssl req -x509 -newkey rsa:2048 -sha256 -nodes -keyout tls.key -out tls.crt -subj "/CN=spring.randomthat.c
om" -days 365
Generating a 2048 bit RSA private key
.....................................................................+++
..+++
writing new private key to 'tls.key'
-----
```

- **Creating a Secret using kubectl**

  ```kubectl create secret tls spring-randomthat-com-tls --cert=tls.crt --key=tls.key
  secret/spring-randomthat-com-tls created
  ```
  Check the secret ```kubectl get secret```

```
[ec2-user@ip-172-31-94-61 ~]$ kubectl create secret tls spring-randomthat-com-tls --cert=tls.crt --key=tls.key
secret/spring-randomthat-com-tls created
[ec2-user@ip-172-31-94-61 ~]$ kubectl get secret
NAME                        TYPE                DATA   AGE
spring-randomthat-com-tls   kubernetes.io/tls   2      5s
```

- **Download Ingress Controller**

  kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/controller-v1.1.0/deploy/static/provider/aws/deploy.yaml

```
[ec2-user@ip-172-31-94-61 ~]$ kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/controller-v1.1.0/deploy/static/p
rovider/aws/deploy.yaml
namespace/ingress-nginx created
serviceaccount/ingress-nginx created
configmap/ingress-nginx-controller created
clusterrole.rbac.authorization.k8s.io/ingress-nginx created
clusterrolebinding.rbac.authorization.k8s.io/ingress-nginx created
role.rbac.authorization.k8s.io/ingress-nginx created
rolebinding.rbac.authorization.k8s.io/ingress-nginx created
service/ingress-nginx-controller-admission created
service/ingress-nginx-controller created
deployment.apps/ingress-nginx-controller created
ingressclass.networking.k8s.io/nginx created
validatingwebhookconfiguration.admissionregistration.k8s.io/ingress-nginx-admission created
serviceaccount/ingress-nginx-admission created
clusterrole.rbac.authorization.k8s.io/ingress-nginx-admission created
clusterrolebinding.rbac.authorization.k8s.io/ingress-nginx-admission created
role.rbac.authorization.k8s.io/ingress-nginx-admission created
rolebinding.rbac.authorization.k8s.io/ingress-nginx-admission created
job.batch/ingress-nginx-admission-create created
job.batch/ingress-nginx-admission-patch created
```

⇒ This will create a namespace by the name ingress-nginx

- **Check the Ingress Pods**
  ```kubectl get pods -n ingress-nginx```

```
[ec2-user@ip-172-31-94-61 ~]$ kubectl get pods -n ingress-nginx
NAME                                        READY   STATUS      RESTARTS   AGE
ingress-nginx-admission-create-rrb94        0/1     Completed   0          41s
ingress-nginx-admission-patch-hk4rh         0/1     Completed   1          41s
ingress-nginx-controller-7454c5f7fb-xc6wc   1/1     Running     0          41s
```

⇒ **Now copy the following yaml files which have the Deployment, Service and Ingress files.**

YAML Files: Deploy-service.yml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
 name: spring-deploy
spec:
 replicas: 2
 selector:
   matchLabels:
     app: spring
 template:
   metadata:
     labels:
       app: spring
   spec:
     containers:
       - name: spring-container
         image: saikrishna682/spring-maven:v1
         ports:
           - containerPort: 8080
---
apiVersion: v1
kind: Service
metadata:
 name: spring-svc
spec:
 selector:
   app: spring
 type: LoadBalancer
```

```yaml
 ports:
   - port: 80
     targetPort: 8080
     protocol: TCP
```

YAML Files: Ingress.yml

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
 name: spring-ingress
 annotations:
   kubernetes.io/ingress.class: nginx
   nginx.ingress.kubernetes.io/ssl-redirect: "true"
spec:
 tls:
   - secretName: spring-randomthat-com-tls
     hosts:
       - "spring.randomthat.com"
 rules:
   - host: spring.randomthat.com
     http:
       paths:
         - path: /course-svc/getAllDevopsTools
           pathType: Prefix
           backend:
             service:
               name: spring-svc
               port:
                 number: 80

         - path: /course-svc/getAllAwsServices
           pathType: Prefix
           backend:
```

```
            service:
              name: spring-svc
              port:
                number: 80
```

- **Now Deploy just the Deploy-service.yml and check if we are able to get the application up and running.**

  ⇒ Kubectl apply -f Deploy-service.yml

  ⇒ Kubectl get pods

  - **Once the pods are up and running**

  ⇒ Kubectl get svc

```
[ec2-user@ip-172-31-94-61 ~]$ kubectl get svc
NAME         TYPE           CLUSTER-IP      EXTERNAL-IP                                                          PORT(S)        AGE
kubernetes   ClusterIP      10.100.0.1      <none>                                                               443/TCP        36m
spring-svc   LoadBalancer   10.100.209.111  a16c53cb33fd647a9990ccd4e803fa22-833014691.us-east-1.elb.amazonaws.com  80:32394/TCP   22m
```

You should be able to get the external ip since the type we used in the service is a LoadBalancer

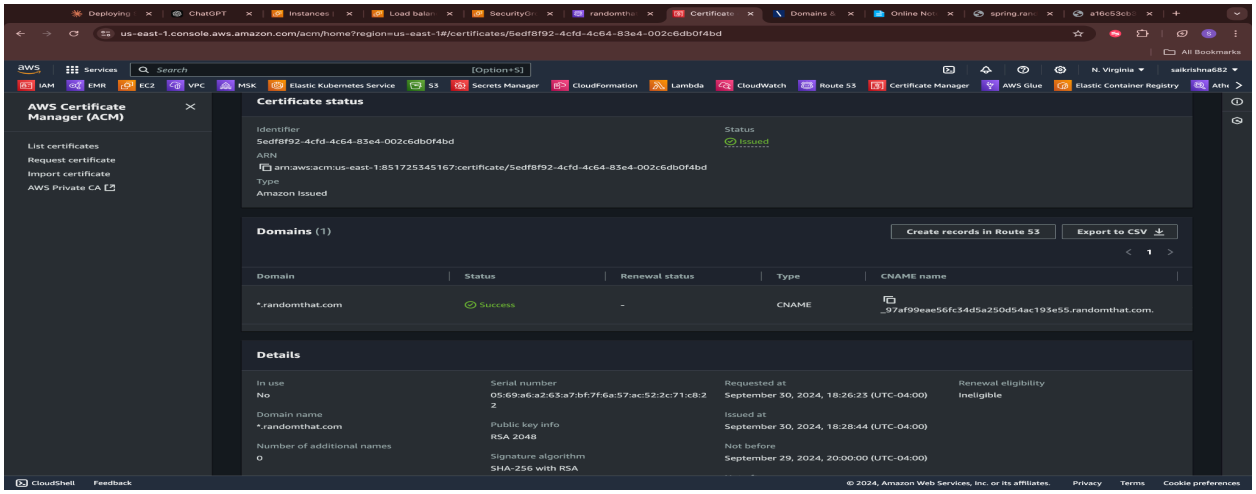
http://<your-external-ip>/course-svc/getAllDevopsTools

Part one is successfully done by achieving by getting the service up and running with load balancer.

Now will have to make use of ingress yaml files so that we can redirect requests to multiple paths and as well as add the Domain instead of load balancer link.

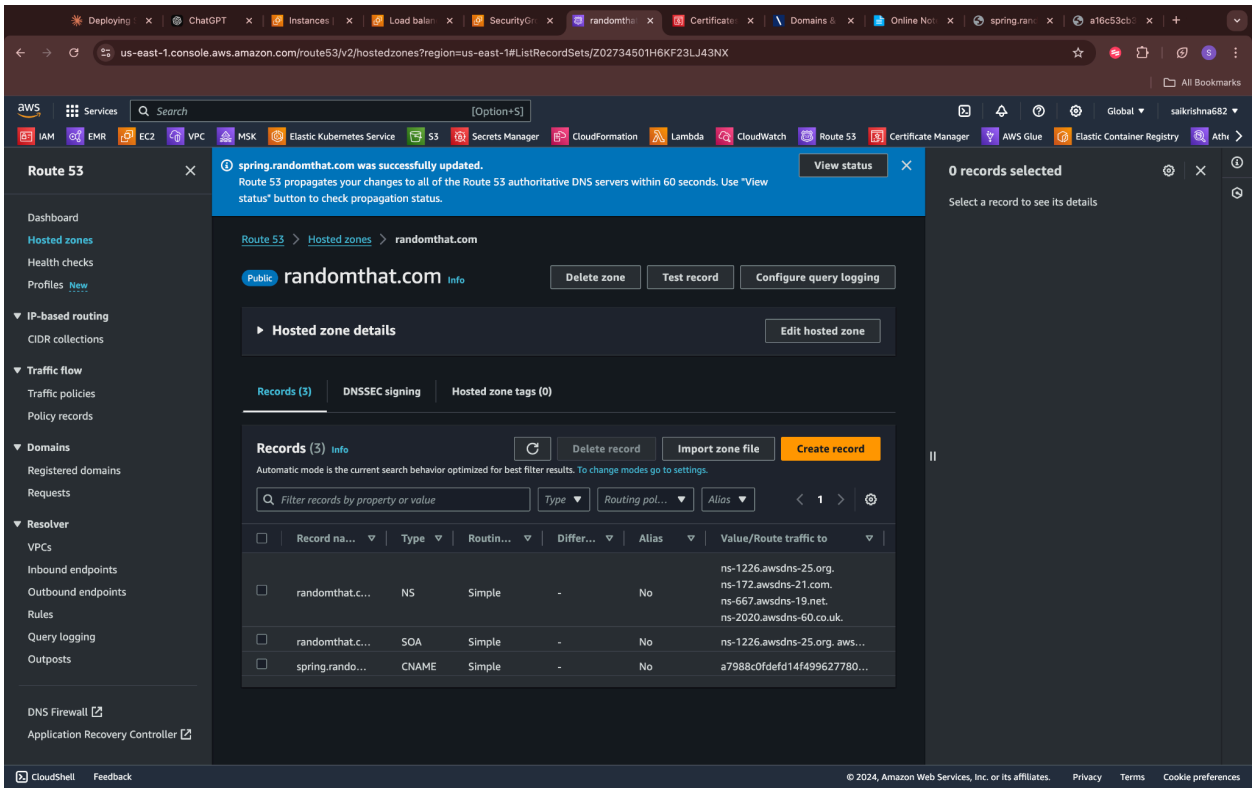
Request a public certificate using AWS ACM.

- **Now hoping you have your own domain, my domain is randomthat.com**
- **Search for Certificate Manager on AWS console.**



Make sure you match the CNAME and CNAME VALUE in your domain DNS so that it gets approved.

Now once the certificate is approved we go to Route 52 and create a Hosted Zone



Create a hosted zone

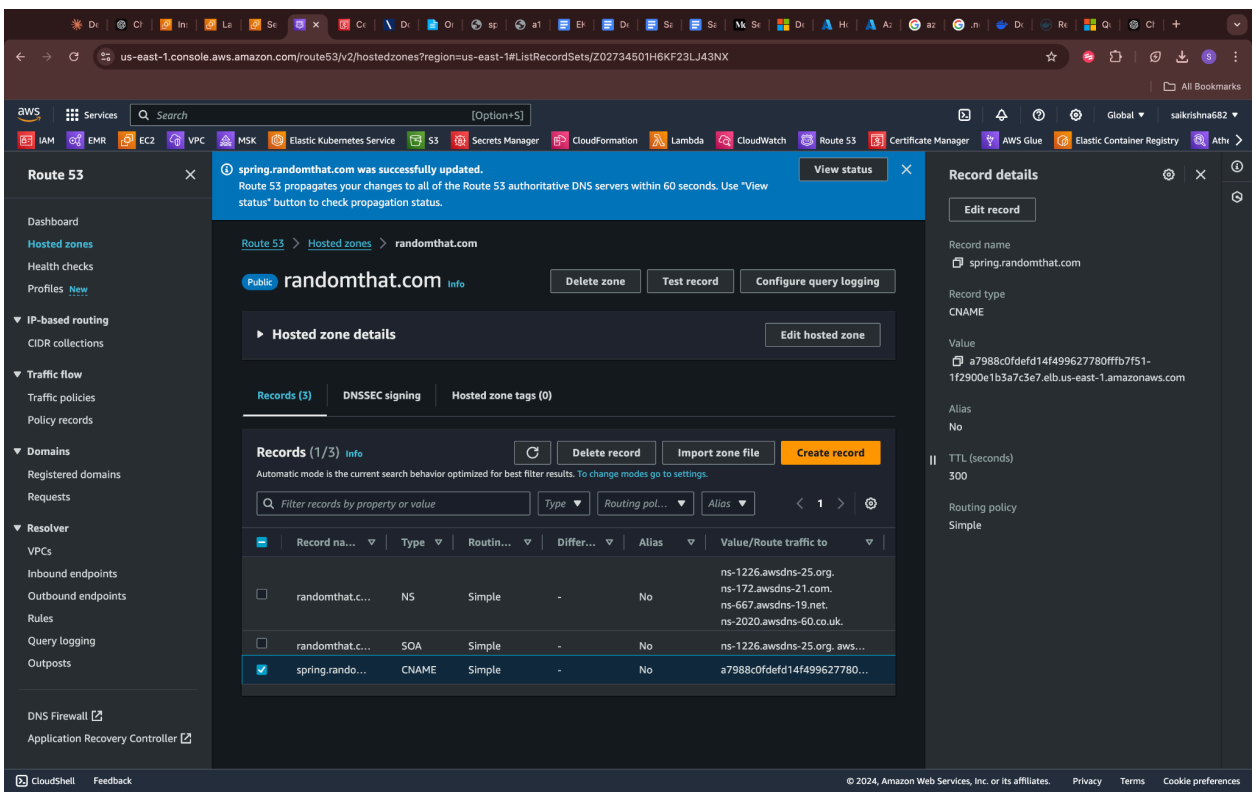
- **Domain is ⇒ randomthat.com**

- **And then create a record where the name is spring.**
- **Type ⇒ CNAME**
- **Value ⇒ Service LoadBalancer URL.**
- **Make sure the same name and value match within your Domain DNS.**
- **So now you should be able to access the same application using this following link ⇒ http://<your-domain-name>/course-svc/getAllDevopsTools**
- **I.e ⇒ http://spring.randomthat.com/course-svc/getAllDevopsTools**

    - **Now deploy the ingress controller yaml file ie⇒ kubectl -f ingress.yml**

```
[ec2-user@ip-172-31-94-61 ~]$ kubectl get svc -n ingress-nginx
NAME                                 TYPE           CLUSTER-IP      EXTERNAL-IP
        PORT(S)                      AGE
ingress-nginx-controller             LoadBalancer   10.100.28.29    a7988c0fdefd14f499627780fffb7f51-1f2900e1b3a7c3e7.elb.us-east-1.amazona
ws.com   80:30179/TCP,443:30692/TCP   34m
ingress-nginx-controller-admission   ClusterIP      10.100.67.230   <none>
        443/TCP                      34m
```

Now Replace the ingress-nginx-controller External-IP with the CVALUE under Route53 spring.randomthat.com record as well as edit the DNS for spring with the new CVALUE.



If everything is perfect then we should be able to get the application at

https://<your-domain-name>/course-svc/getAllDevopsTools

https://spring.randomthat.com/course-svc/getAllDevopsTools

- **To delete a cluster**

  eksctl delete cluster --name <cluster-name> --region <your-region>

  eksctl delete cluster --name demo --region us-east-1