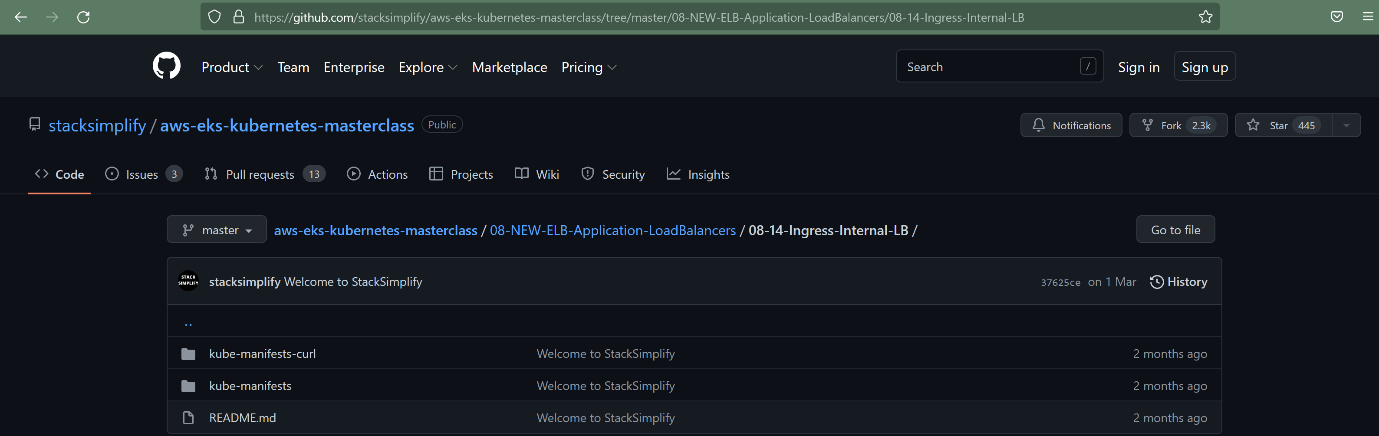
**2. Create Internal ALB using Ingress and Test and Clean-Up**

--- **Reference** - <https://github.com/stacksimplify/aws-eks-kubernetes-masterclass/tree/master/08-NEW-ELB-Application-LoadBalancers/08-14-Ingress-Internal-LB>



--- we are creating curl pod manifests in separate folder.

**Introduction**

--- Create Internal Application Load Balancer using Ingress

--- To test the Internal LB, use the curl-pod

--- Deploy curl-pod

--- Connect to curl-pod and test Internal LB from curl-pod

**Update Ingress Scheme annotation to Internal**

--- **File Name: 04-ALB-Ingress-Internal-LB.yml**

    # Creates Internal Application Load Balancer

    alb.ingress.kubernetes.io/scheme: internal

--- **File Name: 04-ALB-Ingress-Internal-LB.yml**

# Annotations Reference: https://kubernetes-sigs.github.io/aws-load-balancer-controller/latest/guide/ingress/annotations/

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

  name: ingress-internal-lb-demo

  annotations:

    # Load Balancer Name

    alb.ingress.kubernetes.io/load-balancer-name: ingress-internal-lb

    # Ingress Core Settings

    #kubernetes.io/ingress.class: "alb" (OLD INGRESS CLASS NOTATION - STILL WORKS BUT RECOMMENDED TO USE IngressClass Resource)

    # Creates External Application Load Balancer

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

    # Creates Internal Application Load Balancer

    alb.ingress.kubernetes.io/scheme: internal

    # Health Check Settings

    alb.ingress.kubernetes.io/healthcheck-protocol: HTTP

    alb.ingress.kubernetes.io/healthcheck-port: traffic-port

    #Important Note:  Need to add health check path annotations in service level if we are planning to use multiple targets in a load balancer

    alb.ingress.kubernetes.io/healthcheck-interval-seconds: '15'

    alb.ingress.kubernetes.io/healthcheck-timeout-seconds: '5'

    alb.ingress.kubernetes.io/success-codes: '200'

    alb.ingress.kubernetes.io/healthy-threshold-count: '2'

    alb.ingress.kubernetes.io/unhealthy-threshold-count: '2'

spec:

  ingressClassName: my-aws-ingress-class   # Ingress Class

  defaultBackend:

    service:

      name: app3-nginx-nodeport-service

      port:

        number: 80

  rules:

    - http:

        paths:

          - path: /app1

            pathType: Prefix

            backend:

              service:

                name: app1-nginx-nodeport-service

                port:

                  number: 80

          - path: /app2

            pathType: Prefix

            backend:

              service:

                name: app2-nginx-nodeport-service

                port:

                  number: 80

# Important Note-1: In path based routing order is very important, if we are going to use  "/\*", try to use it at the end of all rules.

# 1. If  "spec.ingressClassName: my-aws-ingress-class" not specified, will reference default ingress class on this kubernetes cluster

# 2. Default Ingress class is nothing but for which ingress class we have the annotation `ingressclass.kubernetes.io/is-default-class: "true"`

--- **01-Nginx-App1-Deployment-and-NodePortService.yml**

apiVersion: apps/v1

kind: Deployment

metadata:

  name: app1-nginx-deployment

  labels:

    app: app1-nginx

spec:

  replicas: 1

  selector:

    matchLabels:

      app: app1-nginx

  template:

    metadata:

      labels:

        app: app1-nginx

    spec:

      containers:

        - name: app1-nginx

          image: stacksimplify/kube-nginxapp1:1.0.0

          ports:

            - containerPort: 80

---

apiVersion: v1

kind: Service

metadata:

  name: app1-nginx-nodeport-service

  labels:

    app: app1-nginx

  annotations:

#Important Note:  Need to add health check path annotations in service level if we are planning to use multiple targets in a load balancer

    alb.ingress.kubernetes.io/healthcheck-path: /app1/index.html

spec:

  type: NodePort

  selector:

    app: app1-nginx

  ports:

    - port: 80

      targetPort: 80

--- **02-Nginx-App2-Deployment-and-NodePortService.yml**

apiVersion: apps/v1

kind: Deployment

metadata:

  name: app2-nginx-deployment

  labels:

    app: app2-nginx

spec:

  replicas: 1

  selector:

    matchLabels:

      app: app2-nginx

  template:

    metadata:

      labels:

        app: app2-nginx

    spec:

      containers:

        - name: app2-nginx

          image: stacksimplify/kube-nginxapp2:1.0.0

          ports:

            - containerPort: 80

---

apiVersion: v1

kind: Service

metadata:

  name: app2-nginx-nodeport-service

  labels:

    app: app2-nginx

  annotations:

#Important Note:  Need to add health check path annotations in service level if we are planning to use multiple targets in a load balancer

    alb.ingress.kubernetes.io/healthcheck-path: /app2/index.html

spec:

  type: NodePort

  selector:

    app: app2-nginx

  ports:

    - port: 80

      targetPort: 80

--- **03-Nginx-App3-Deployment-and-NodePortService.yml**

apiVersion: apps/v1

kind: Deployment

metadata:

  name: app3-nginx-deployment

  labels:

    app: app3-nginx

spec:

  replicas: 1

  selector:

    matchLabels:

      app: app3-nginx

  template:

    metadata:

      labels:

        app: app3-nginx

    spec:

      containers:

        - name: app2-nginx

          image: stacksimplify/kubenginx:1.0.0

          ports:

            - containerPort: 80

---

apiVersion: v1

kind: Service

metadata:

  name: app3-nginx-nodeport-service

  labels:

    app: app3-nginx

  annotations:

#Important Note:  Need to add health check path annotations in service level if we are planning to use multiple targets in a load balancer

    alb.ingress.kubernetes.io/healthcheck-path: /index.html

spec:

  type: NodePort

  selector:

    app: app3-nginx

  ports:

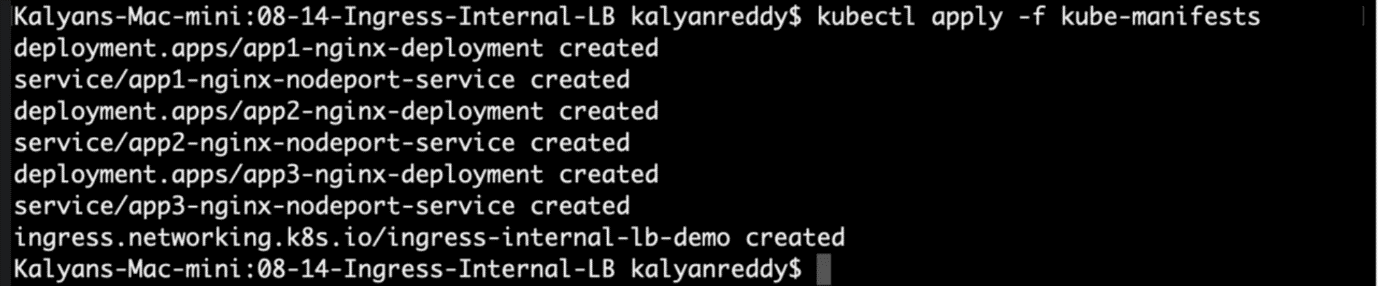
    - port: 80

      targetPort: 80

**Deploy all Application Kubernetes Manifests and Verify**

**# Deploy kube-manifests**

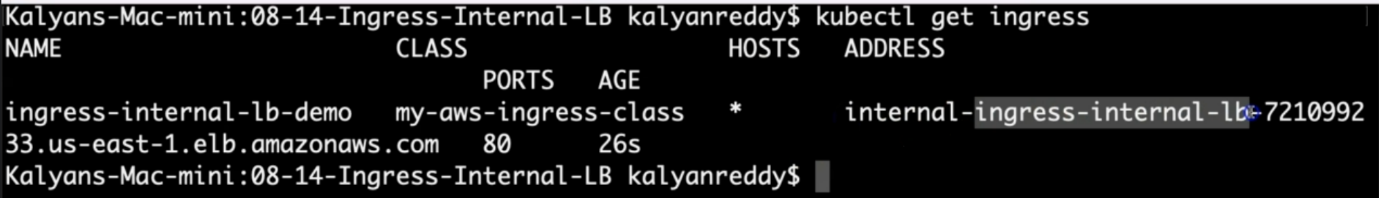
--- **kubectl apply -f kube-manifests/**



--- **note** - it will create ingress internal load balancer along with above mentioned deployments and nodeport services.

**# Verify Ingress Resource**

--- **kubectl get ingress**

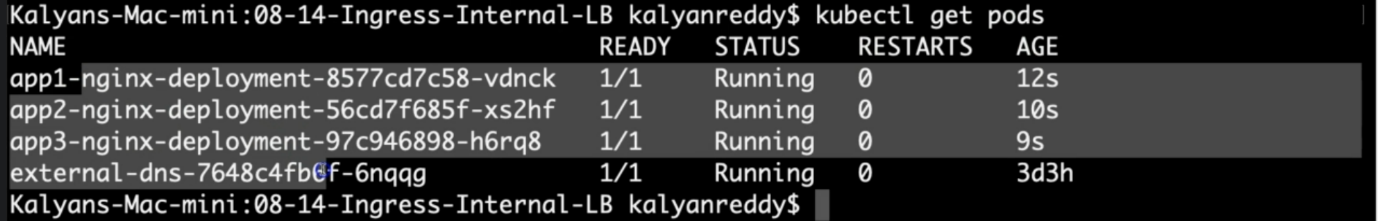


--- **note** – ingress internal dns name under address. The ingress ensures that the dns name is appended with **internal-**. You can also check the ingress service file. This way it is telling us that it had created the internal load balancer.

**# Verify Apps**

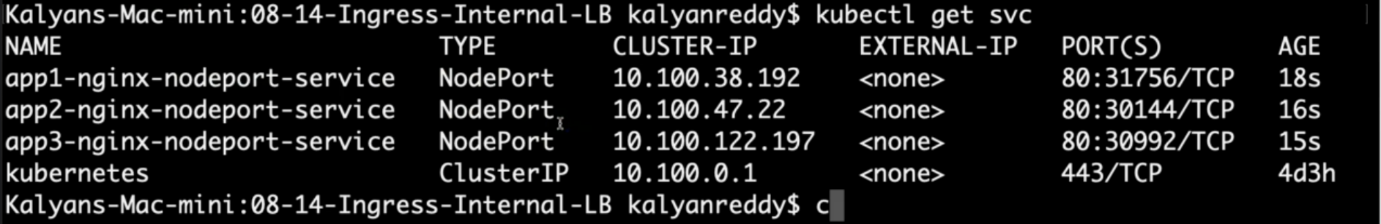
--- **kubectl get deploy**

--- **kubectl get pods**



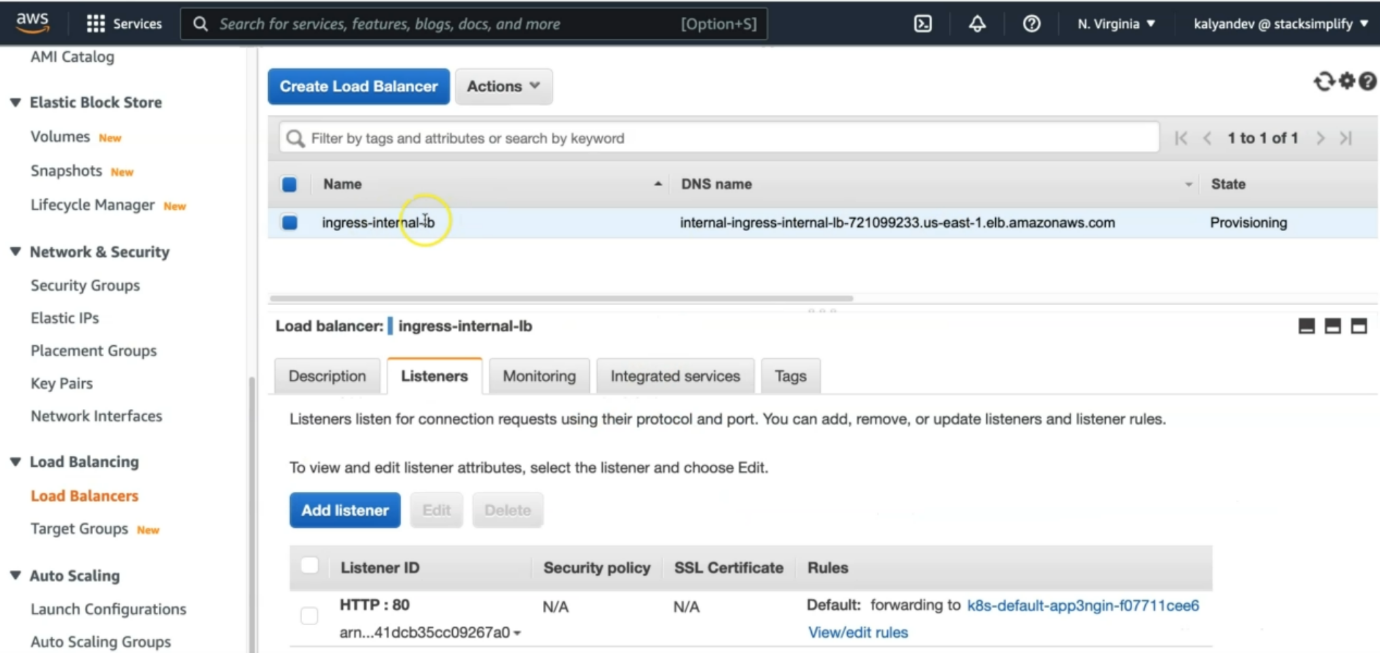
**# Verify NodePort Services**

--- **kubectl get svc**



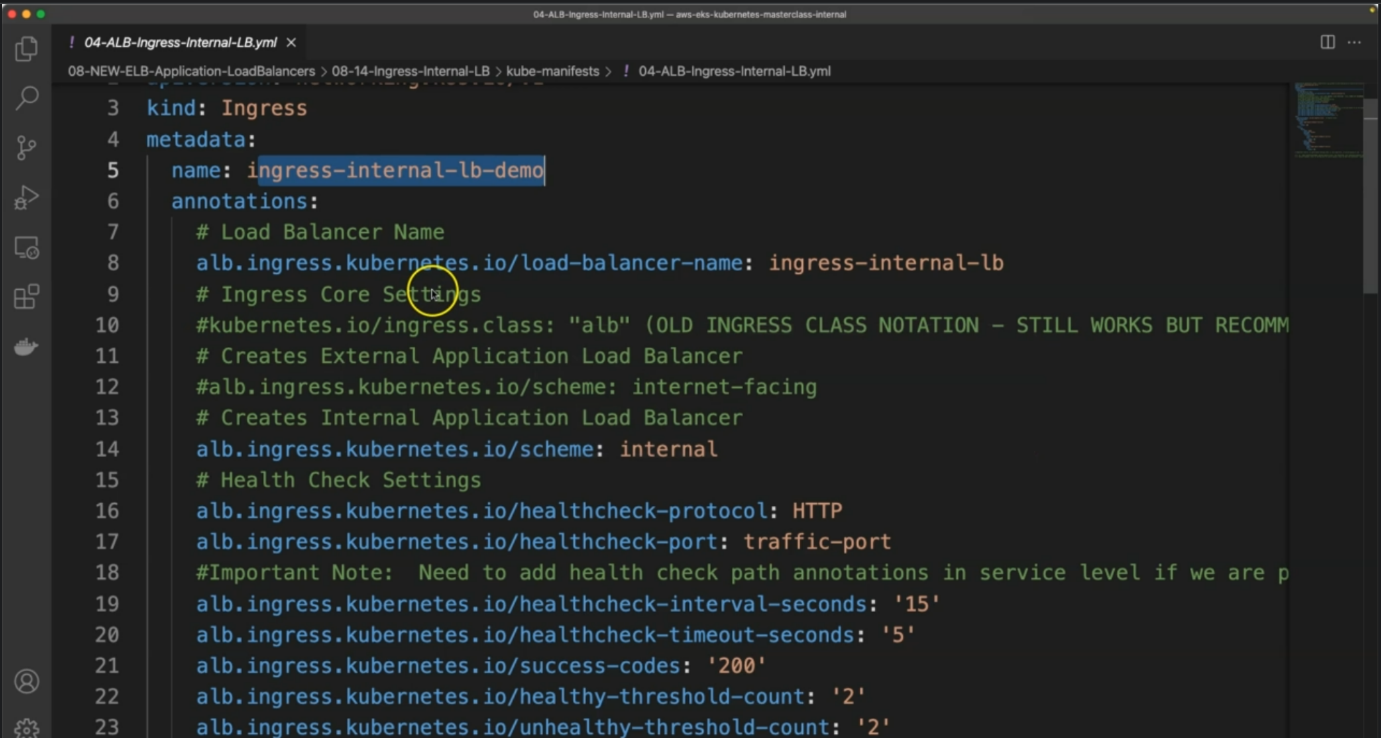
**Verify Load Balancer & Target Groups**

--- Load Balancer - Listeners (Verify both 80 & 443)

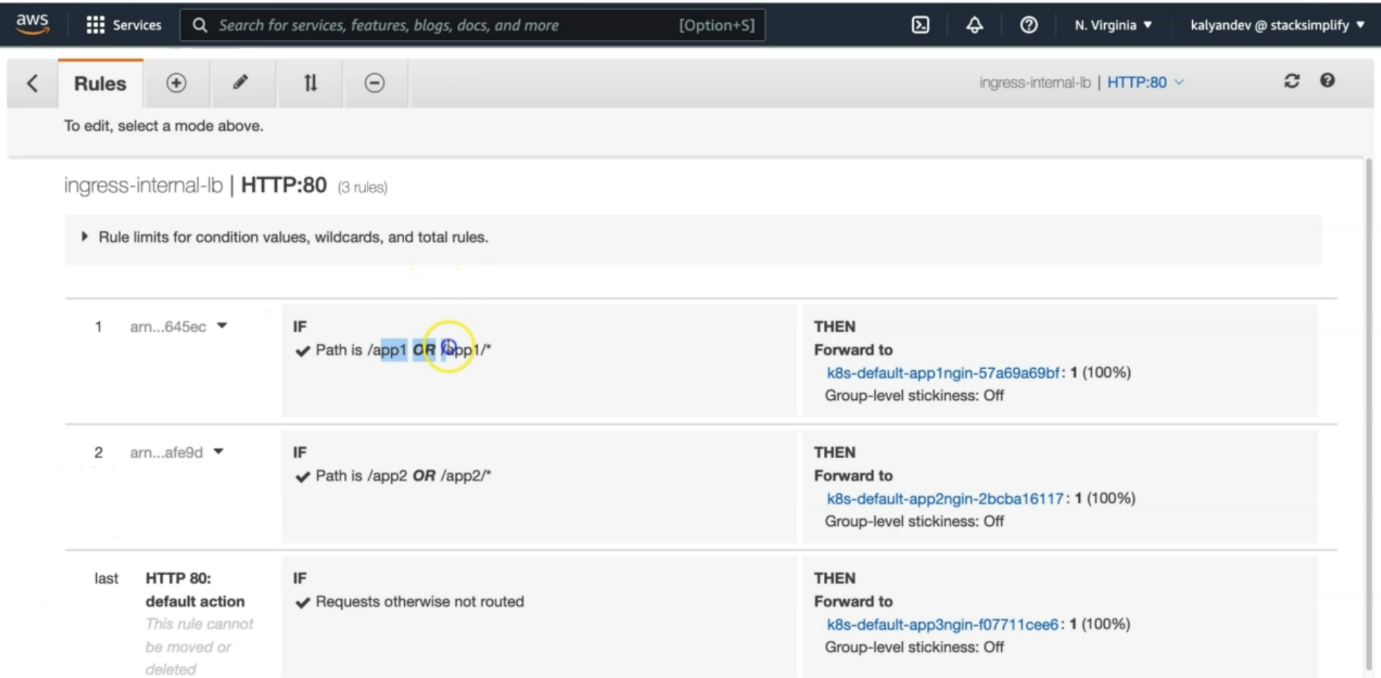


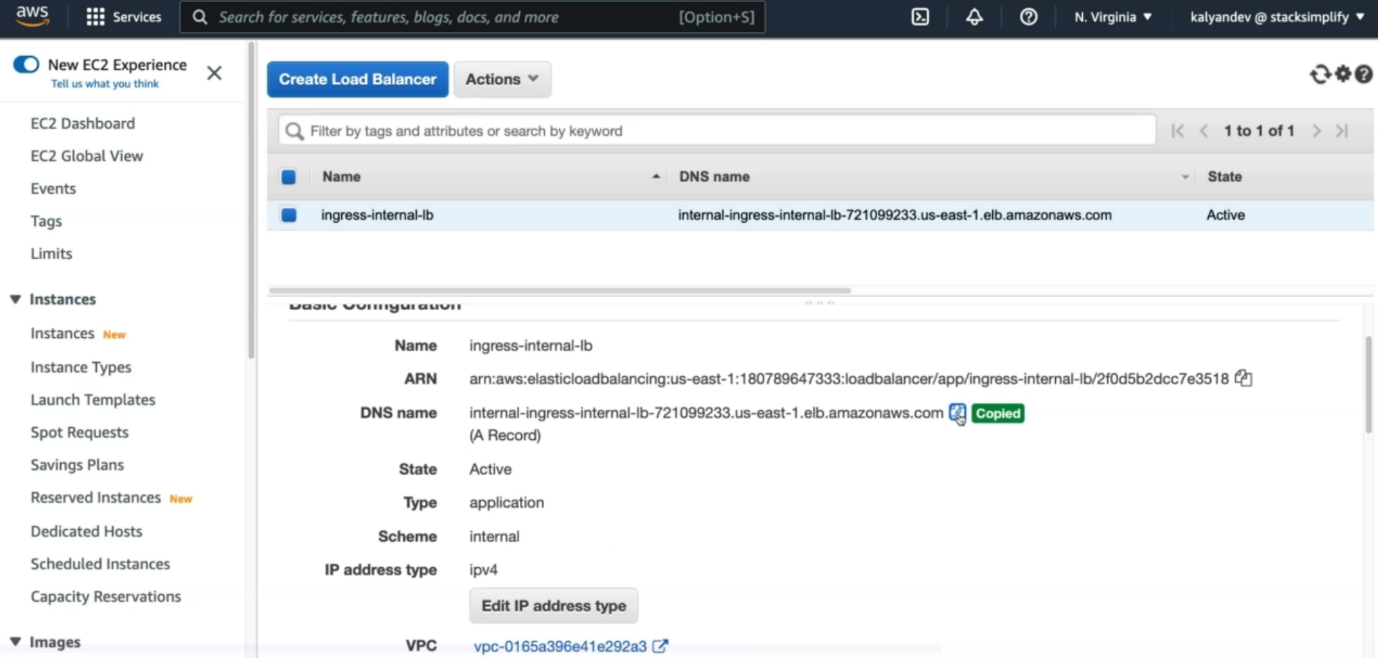
--- there is no HTTPS:443 in the ingress service file, that is why, we are not able to see the port HTTPS:443.

--- here, we are able see only one port, that is HTTP:80.

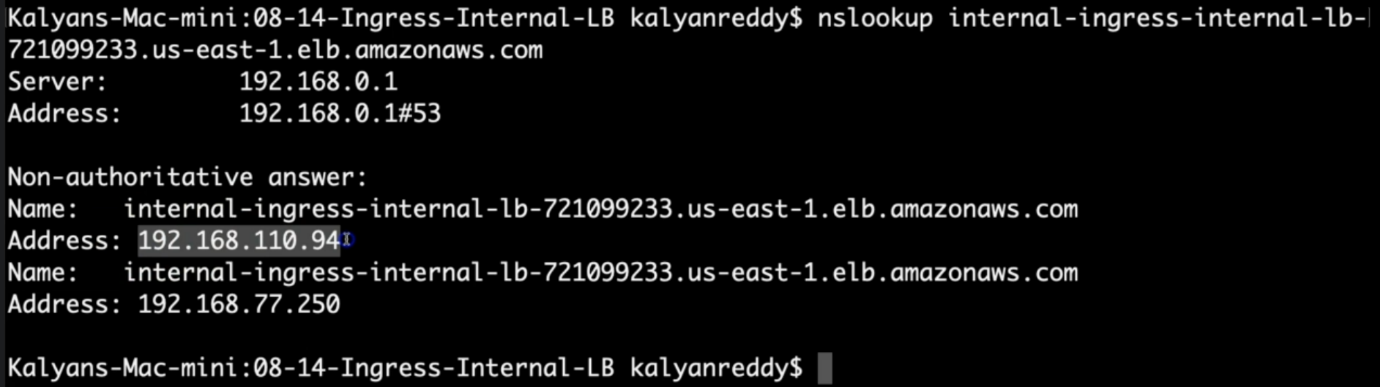


--- click on edit/view rules.





--- this is load balancer dns name and copy the dns name and do a nslookup for that.



--- Load Balancer - Rules (Verify both 80 & 443 listeners)

--- Target Groups - Group Details (Verify Health check path)

--- Target Groups - Targets (Verify all 3 targets are healthy)

**How to test this Internal Load Balancer?**

--- We are going to deploy a curl-pod in EKS Cluster

--- We connect to that curl-pod in EKS Cluster and test using curl commands for our sample applications load balanced using this Internal Application Load Balancer

**curl-pod Kubernetes Manifest**

--- **File Name: kube-manifests-curl/01-curl-pod.yml**

apiVersion: v1

kind: Pod

metadata:

  name: curl-pod

spec:

  containers:

  - name: curl

    image: curlimages/curl

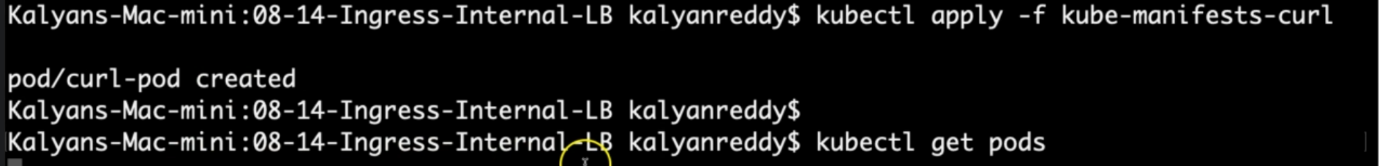
    command: [ "sleep", "600" ]

--- it will download the curl pod and run it.

**Deploy curl-pod and Verify Internal LB**

**# Deploy curl-pod**

--- **kubectl apply -f kube-manifests-curl**



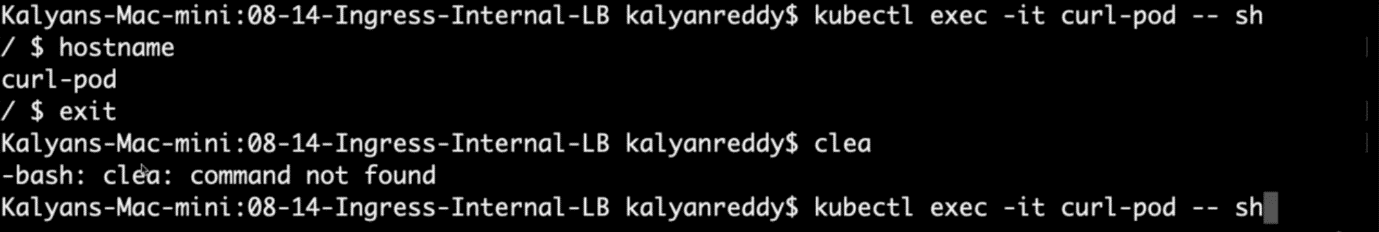
**# Get pods**

--- **kubectl get pods**



**# Will open up a terminal session into the container**

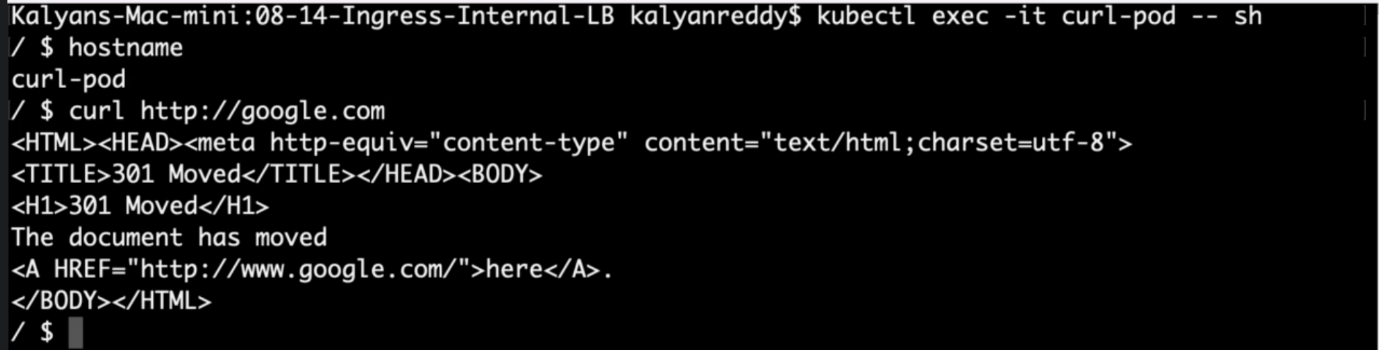
--- **kubectl exec -it curl-pod – sh**



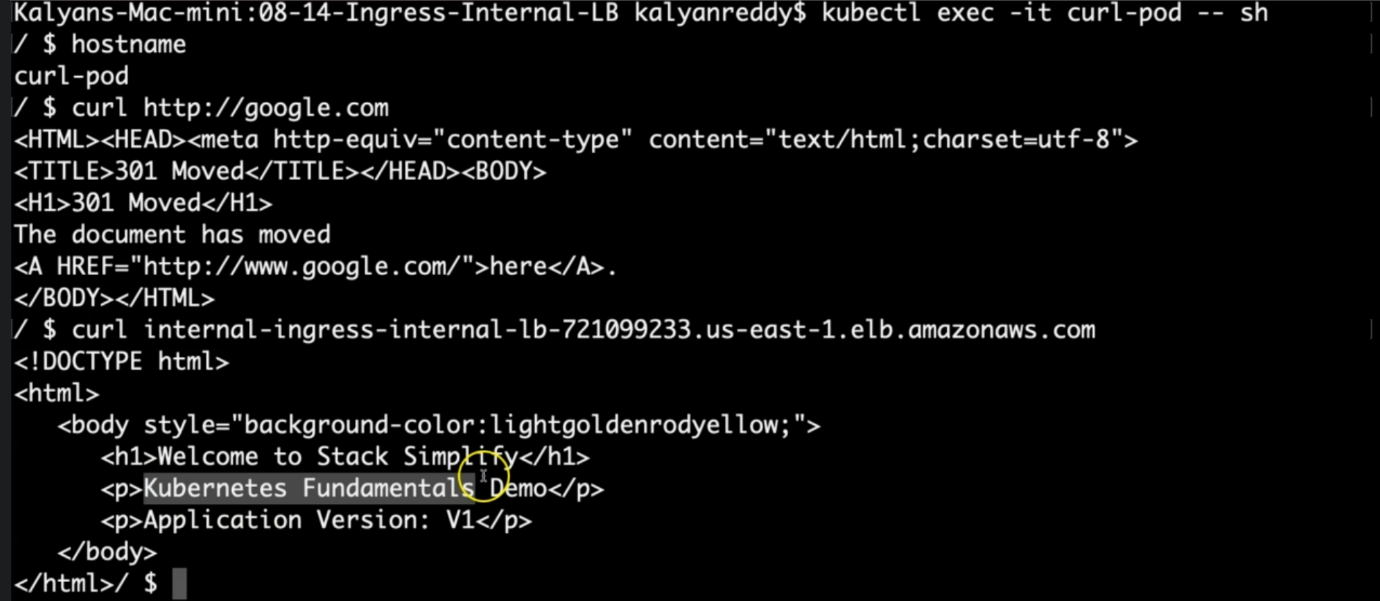
--- **note** - so outbound is allowed for us via Nate gate way. So, if you inside of curl pod and access google.com then it should accessible.

**# We can now curl external addresses or internal services:**

--- **curl** [**http://google.com/**](http://google.com/)



--- **curl <INTERNAL-INGRESS-LB-DNS>**

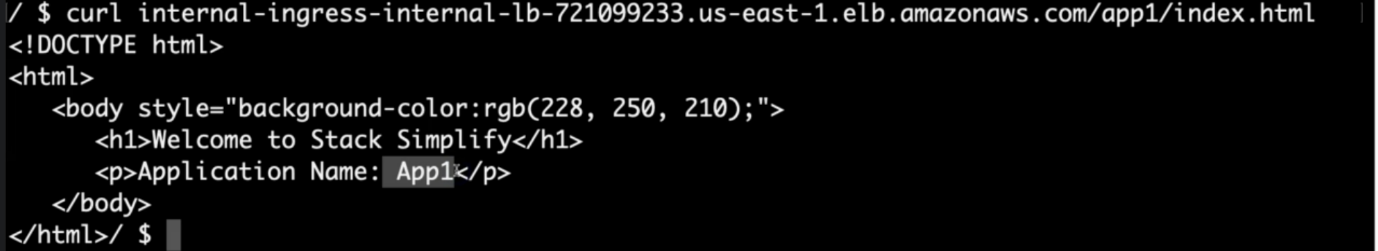


**# Default Backend Curl Test**

--- **curl internal-ingress-internal-lb-1839544354.us-east-1.elb.amazonaws.com**

**# App1 Curl Test**

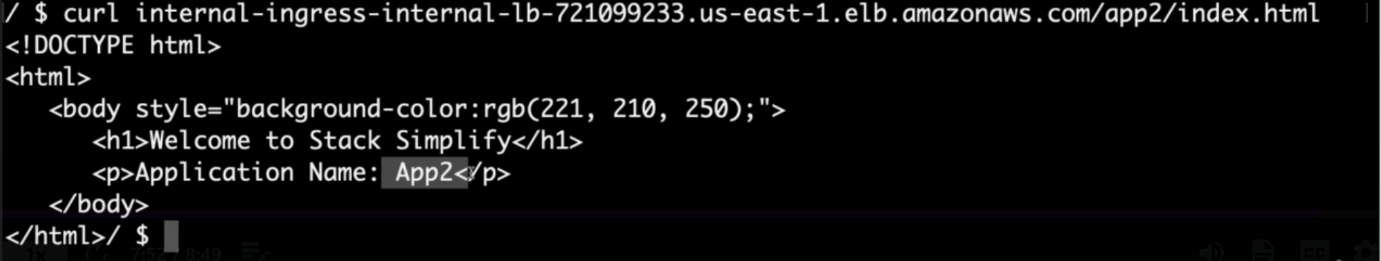
--- **curl internal-ingress-internal-lb-1839544354.us-east-1.elb.amazonaws.com/app1/index.html**



--- it should give me the app1 nginx.

**# App2 Curl Test**

--- **curl internal-ingress-internal-lb-1839544354.us-east-1.elb.amazonaws.com/app2/index.html**



--- it should give me the app2 nginx.

**# App3 Curl Test**

--- **curl internal-ingress-internal-lb-1839544354.us-east-1.elb.amazonaws.com**

**Clean Up**

**# Delete Manifests**

--- **kubectl delete -f kube-manifests/**

--- **kubectl delete -f kube-manifests-curl/**