**2. Implement Ingress Target Type IP Mode**

--- Reference - <https://github.com/stacksimplify/aws-eks-kubernetes-masterclass/tree/master/08-NEW-ELB-Application-LoadBalancers/08-13-Ingress-TargetType-IP>

**Introduction**

--- **alb.ingress.kubernetes.io/target-type** specifies how to route traffic to pods.

--- You can choose between **instance** and **ip**

--- **Instance Mode**: instance mode will route traffic to all ec2 instances within cluster on NodePort opened for your service.

--- **IP Mode**: ip mode is required for sticky sessions to work with Application Load Balancers.

**Ingress Manifest - Add target-type**

--- **File Name: 04-ALB-Ingress-target-type-ip.yml**

    # Target Type: IP

    alb.ingress.kubernetes.io/target-type: ip

# 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-target-type-ip-demo

  annotations:

    # Load Balancer Name

    alb.ingress.kubernetes.io/load-balancer-name: target-type-ip-ingress

    # Ingress Core Settings

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

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

    # 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'

    ## SSL Settings

    alb.ingress.kubernetes.io/listen-ports: '[{"HTTPS":443}, {"HTTP":80}]'

    #alb.ingress.kubernetes.io/certificate-arn: arn:aws:acm:us-east-1:180789647333:certificate/d86de939-8ffd-410f-adce-0ce1f5be6e0d

    #alb.ingress.kubernetes.io/ssl-policy: ELBSecurityPolicy-TLS-1-1-2017-01 #Optional (Picks default if not used)

    # SSL Redirect Setting

    alb.ingress.kubernetes.io/ssl-redirect: '443'

    # External DNS - For creating a Record Set in Route53

    external-dns.alpha.kubernetes.io/hostname: target-type-ip-501.stacksimplify.com

    # Target Type: IP

    alb.ingress.kubernetes.io/target-type: ip

spec:

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

  defaultBackend:

    service:

      name: app3-nginx-clusterip-service

      port:

        number: 80

  tls:

  - hosts:

    - "\*.stacksimplify.com"

  rules:

    - http:

        paths:

          - path: /app1

            pathType: Prefix

            backend:

              service:

                name: app1-nginx-clusterip-service

                port:

                  number: 80

    - http:

        paths:

          - path: /app2

            pathType: Prefix

            backend:

              service:

                name: app2-nginx-clusterip-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-ClusterIPService.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-clusterip-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: ClusterIP

  selector:

    app: app1-nginx

  ports:

    - port: 80

      targetPort: 80

--- **02-Nginx-App2-Deployment-and-ClusterIPService.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-clusterip-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: ClusterIP

  selector:

    app: app2-nginx

  ports:

    - port: 80

      targetPort: 80

--- **03-Nginx-App3-Deployment-and-ClusterIPService.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-clusterip-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: ClusterIP

  selector:

    app: app3-nginx

  ports:

    - port: 80

      targetPort: 80

--- **note** - **spec: type: ClusterIP** – you can give NodePort service here but I want to show that the request will directly comes to the pod from load balancer. Even though you opened the nodeport then it will no use. ClusterIP for internal purpose only, for ex application to connect backend db.

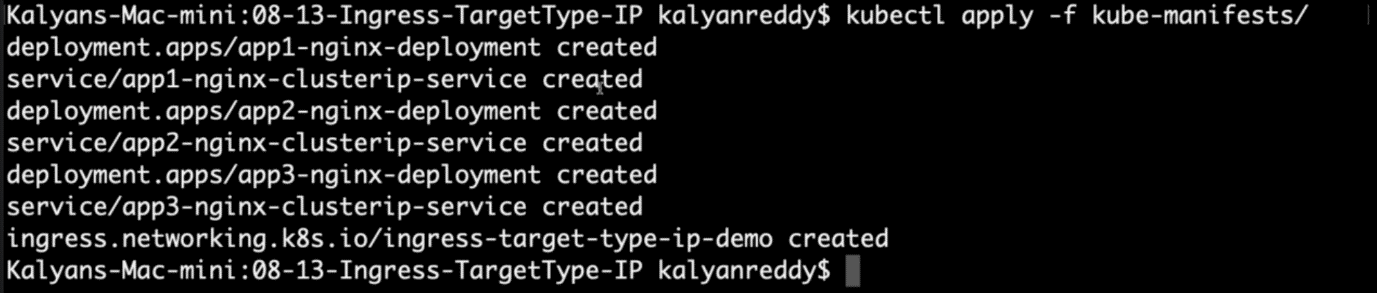
--- **if the request is directly coming to the pods from load balancer, then why we need service…?**

You need service to have the ingress created target group and register the targets inside of that. So, you need to create service type of nodeport or clusterip needs to be created. So that ingress can create target groups in your application load balancer.

**Deploy all Application Kubernetes Manifests and Verify**

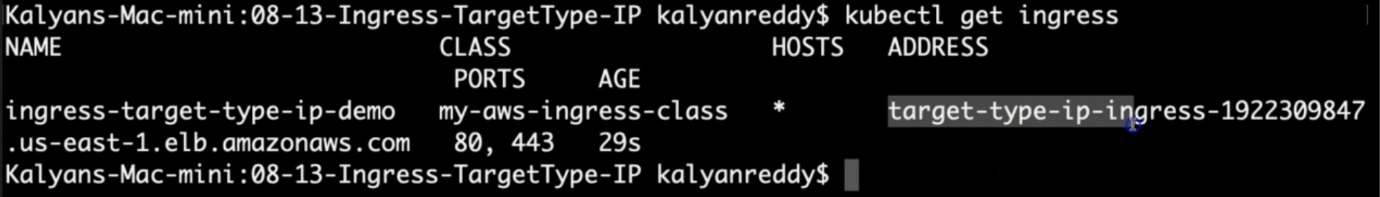
**# Deploy kube-manifests**

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



**# Verify Ingress Resource**

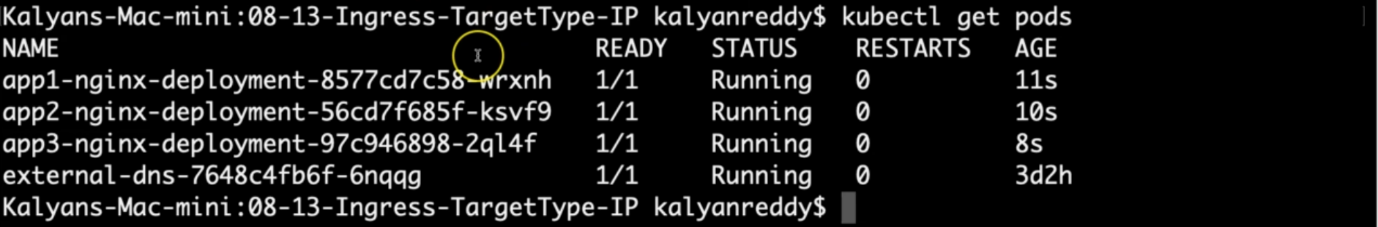
--- **kubectl get ingress**



**# Verify Apps**

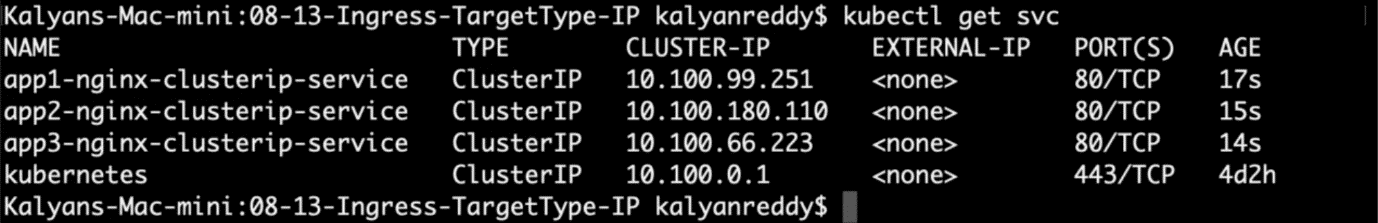
--- **kubectl get deploy**

--- **kubectl get pods**



**# Verify NodePort Services**

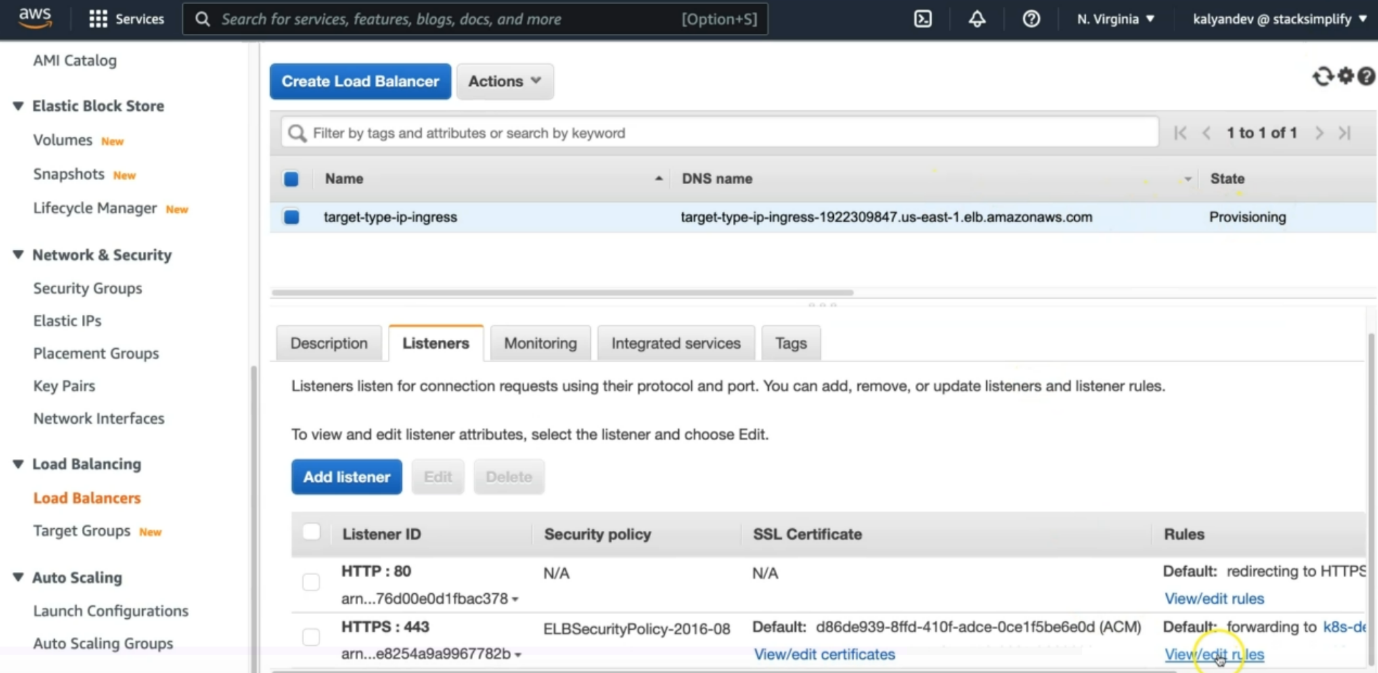
--- **kubectl get svc**



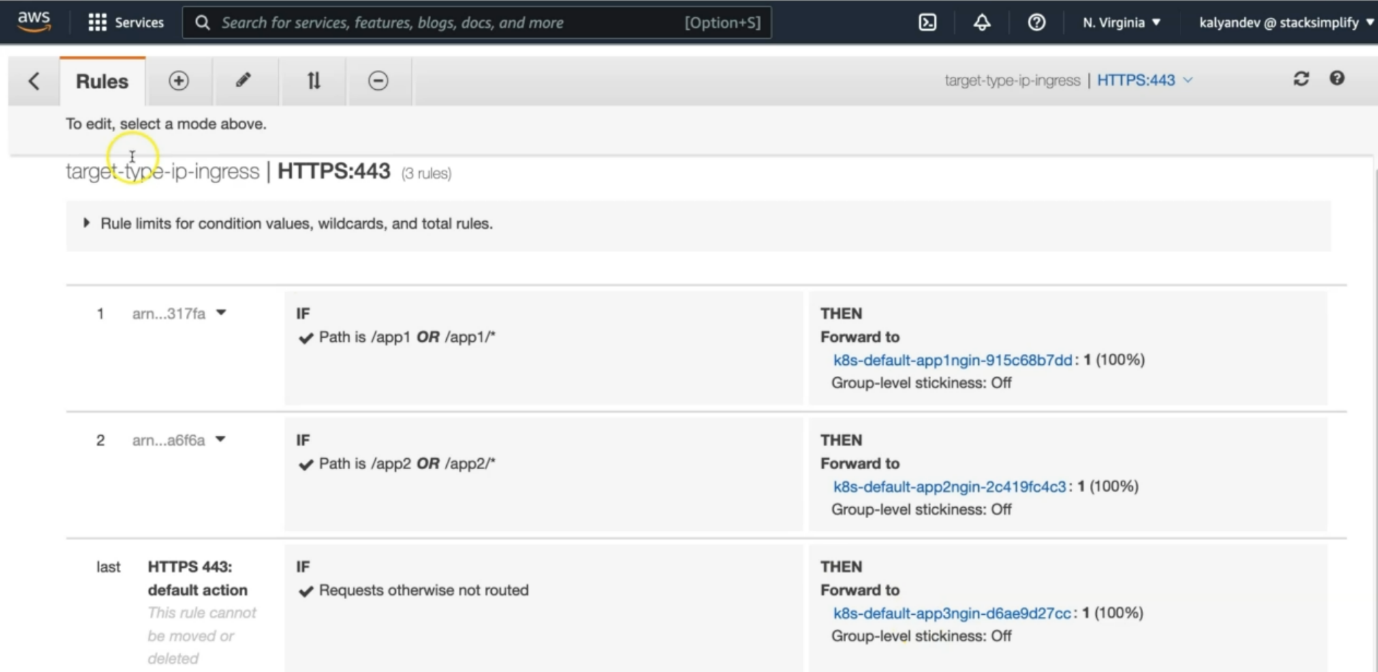
--- **note** – just remembers the cluster Ip’s here.

**Verify Load Balancer & Target Groups**

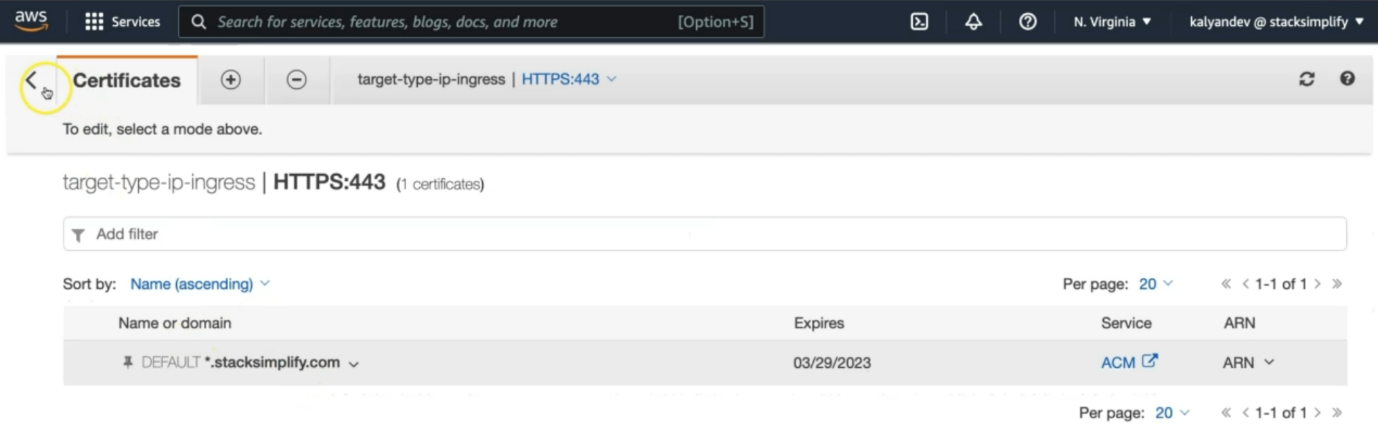
--- Load Balancer - Listeners (Verify both 80 & 443)



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



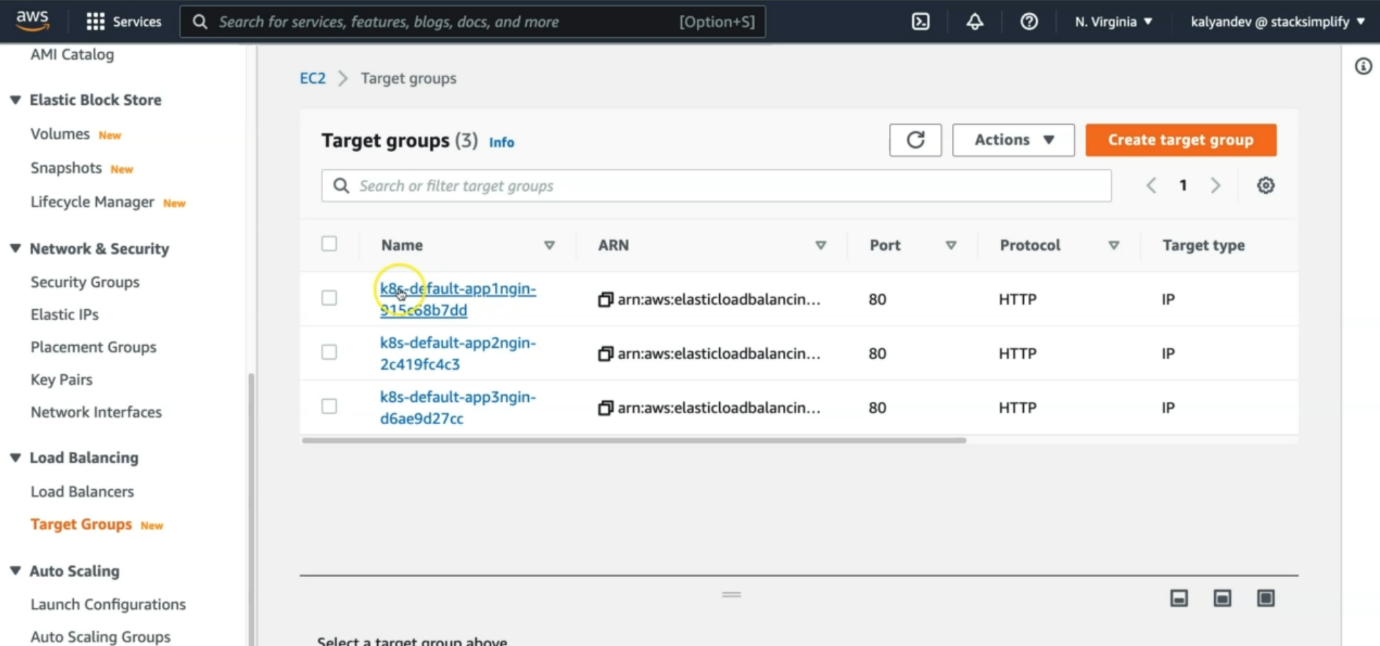
--- verify certificate.



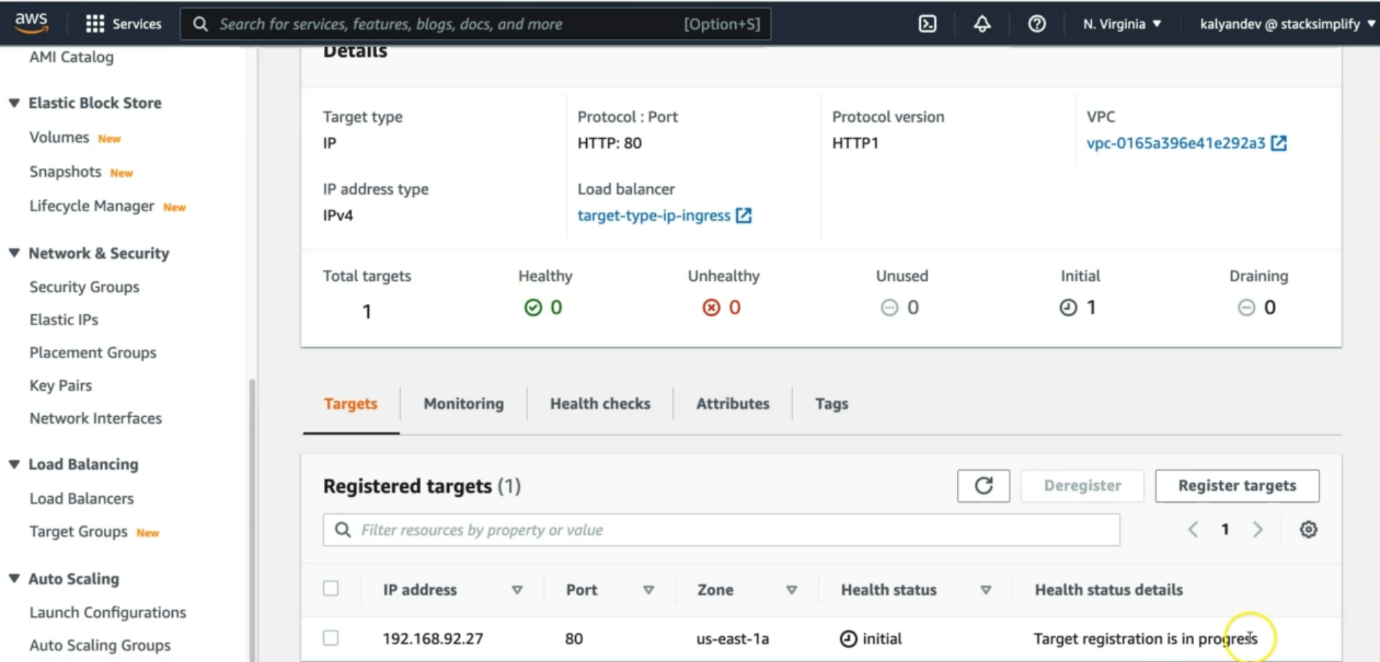
--- **note** - the \*.stacksimplify.com certificate is automatically added to the load balancer.

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

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

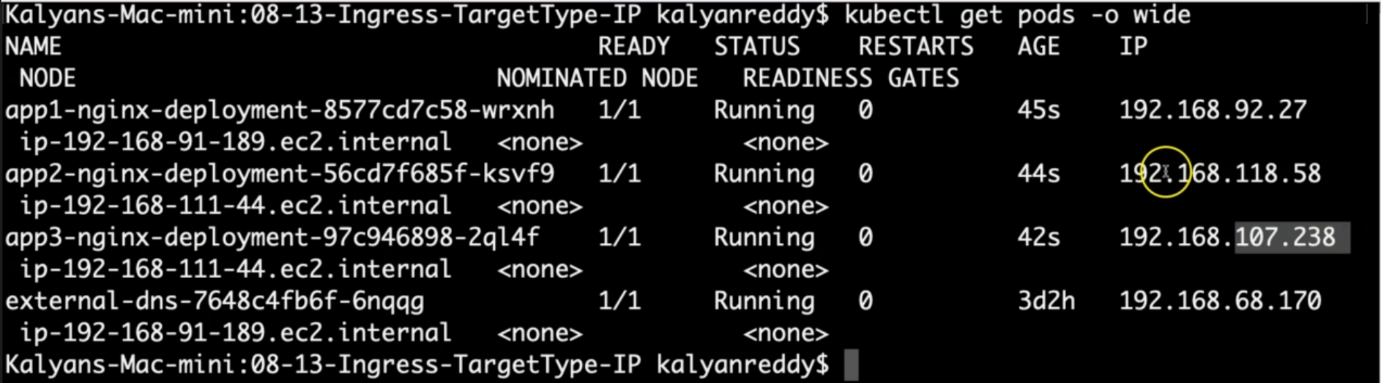


--- click on the 1st target group.



--- heath checks are good and the pod is presented here.

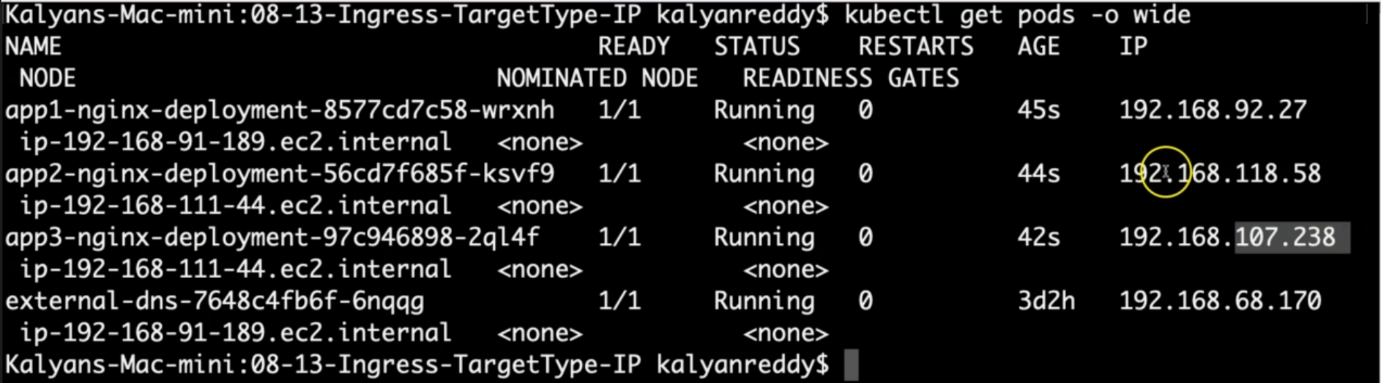
--- **kubectl get pods -o wide**



--- PRIMARILY VERIFY - TARGET GROUPS which contain the POD IPs instead of WORKER NODE IP with NODE PORTS

**# List Pods and their IPs**

--- **kubectl get pods -o wide**



--- Verify External DNS Log

**# Verify External DNS logs**

--- **kubectl logs -f $(kubectl get po | egrep -o 'external-dns[A-Za-z0-9-]+')**

**Verify Route53**

--- Go to Services -> Route53

--- You should see Record Sets added for

target-type-ip-501.stacksimplify.com

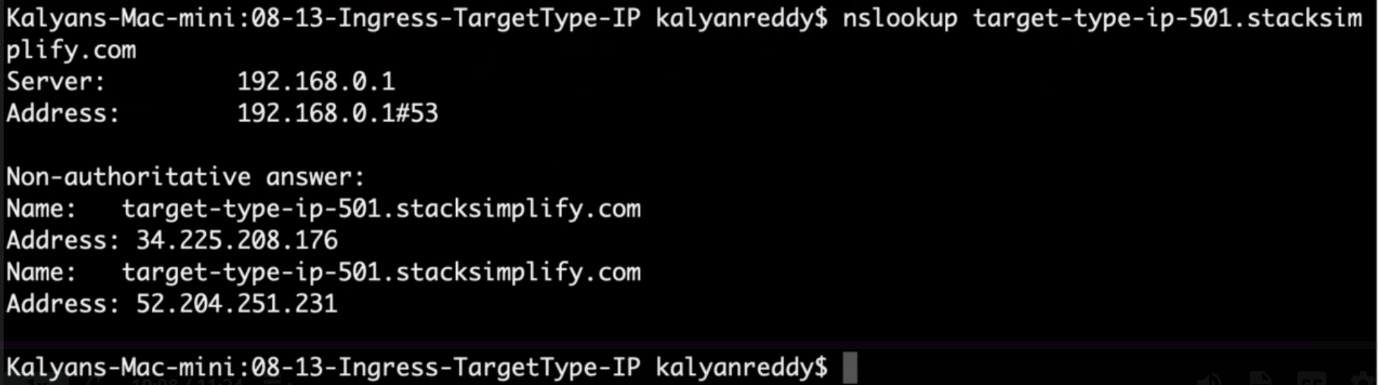
**Access Application using newly registered DNS Name**

--- Perform nslookup tests before accessing Application

--- Test if our new DNS entries registered and resolving to an IP Address

**# nslookup commands**

--- **nslookup target-type-ip-501.stacksimplify.com**



**Access Application using DNS domain**

**# Access App1**

---- **http://target-type-ip-501.stacksimplify.com /app1/index.html**

**# Access App2**

--- **http://target-type-ip-501.stacksimplify.com /app2/index.html**

**# Access Default App (App3)**

--- **http://target-type-ip-501.stacksimplify.com**

**Clean Up**

**# Delete Manifests**

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

**## Verify Route53 Record Set to ensure our DNS records got deleted**

--- Go to Route53 -> Hosted Zones -> Records

--- The below records should be deleted automatically

1. target-type-ip-501.stacksimplify.com