Kubernetes

*(@theshivanshvasu)*

**1. Introduction to Kubernetes**

**What is Kubernetes?**

An open-source container orchestration platform.

Developed by Google, now maintained by CNCF (Cloud Native Computing Foundation).

**Why Use Kubernetes?**

*Automates deployment, scaling, and management of containerized applications.*

Ensures high availability, scalability, and flexibility.

**2. Kubernetes Architecture**

**Master Node Components**

*API Server*: Exposes the Kubernetes API.

etcd: Key-value store for cluster data.

Controller Manager: Manages controllers that regulate the state of the cluster.

Scheduler: Assigns workloads to nodes.

**Worker Node Components**

Kubelet: Ensures containers are running.

Kube-proxy: Manages network routing for services.

Container Runtime: Runs the containers (e.g., Docker, containerd).

**Pods**

Smallest deployable units in Kubernetes.

Encapsulate one or more containers.

**3. Setting Up a Kubernetes Cluster**

Local Setup with Minikube

Install Minikube and kubectl.

**Start a Minikube cluster:**

minikube start

Cloud-based Setup

Use managed Kubernetes services like GKE (Google Kubernetes Engine), EKS (Amazon Elastic Kubernetes Service), or AKS (Azure Kubernetes Service).

**4. Kubernetes Objects**

**Deployments**

Manages stateless applications.

| Example: yaml Copy code apiVersion: apps/v1 kind: Deployment metadata:  name: nginx-deployment spec:  replicas: 3  selector:  matchLabels:  app: nginx  template:  metadata:  labels:  app: nginx  spec:  containers:  - name: nginx  image: nginx:1.14.2  ports:  - containerPort: 80 Services |
| --- |

Exposes a set of Pods as a network service.

Types: ClusterIP, NodePort, LoadBalancer.

| Example: apiVersion: v1 kind: Service metadata:  name: nginx-service spec:  selector:  app: nginx  ports:  - protocol: TCP  port: 80  targetPort: 80  type: ClusterIP |
| --- |

**ConfigMaps and Secrets**

ConfigMap: Stores configuration data as key-value pairs.

Secret: Stores sensitive data, such as passwords.

**Persistent Volumes (PVs) and Persistent Volume Claims (PVCs)**

PV: Provisioned storage in the cluster.

PVC: Request for storage by a user.

**5. Deploying Applications**

**Step-by-Step Deployment**

Create a Deployment YAML file.

Apply the configuration:

| kubectl apply -f deployment.yaml |
| --- |

Expose the Deployment via a Service:

| kubectl expose deployment nginx-deployment --type=NodePort --port=80 |
| --- |

**6. Managing Applications**

Scaling

Scale the number of replicas:

| kubectl scale deployment nginx-deployment --replicas=5 |
| --- |

Rolling Updates

Update an application without downtime:

| kubectl set image deployment/nginx-deployment nginx=nginx:1.16.0 |
| --- |

Monitoring and Logging

Use kubectl logs to view Pod logs.

Use kubectl top to monitor resource usage.

**7. Advanced Topics**

**Helm**

Package manager for Kubernetes.

Use Helm charts to define, install, and upgrade complex Kubernetes applications.

**Network Policies**

Define rules for Pod communication.

RBAC (Role-Based Access Control)

Manage permissions within the cluster.

**RESOURCES**

Kubernetes is an open-source container orchestration system for automating software deployment, scaling, and management. Google originally designed Kubernetes, but the Cloud Native Computing Foundation now maintains the project.

Resources:

* <https://www.youtube.com/watch?v=X48VuDVv0do>
* <https://www.youtube.com/watch?v=s_o8dwzRlu4>
* <https://www.youtube.com/watch?v=yznvWW_L7AA>
* <https://www.youtube.com/watch?v=YzaYqxW0wGs&list=PL34sAs7_26wNBRWM6BDhnonoA5FMERax0>
* <https://www.youtube.com/watch?v=l_lWfipUimk&list=PLhW3qG5bs-L8EU_Oocu6RkNPpYpaamtXX>
* <https://www.youtube.com/watch?v=umXEmn3cMWY>
* <https://www.youtube.com/watch?v=azuwXALfyRg&t=2s>
* <https://www.youtube.com/watch?v=VnvRFRk_51k&list=PLy7NrYWoggjziYQIDorlXjTvvwweTYoNC>

Links:

* <https://kubernetes.io/docs/home/>
* <https://www.tutorialspoint.com/kubernetes/index.htm>
* <https://medium.com/free-code-camp/learn-kubernetes-in-under-3-hours-a-detailed-guide-to-orchestrating-containers-114ff420e882>

Courses:

* <https://kube.academy/>
* <https://killercoda.com/>
* <https://kodekloud.com/learning-path-kubernetes/>

Getting Started:

* [Kubernetes installation tools](https://kubernetes.io/docs/tasks/tools/)
* [Kubernetes installation with Kind](https://kind.sigs.k8s.io/docs/user/quick-start/)
* [Kubernetes Installation with minikube](https://minikube.sigs.k8s.io/docs/start/)
* [Kubectl cheat sheet](https://kubernetes.io/docs/reference/kubectl/cheatsheet/)
* [Kubernetes Core concepts](https://www.cncf.io/blog/2019/05/10/kubernetes-core-concepts/)

Certifications:

* <https://www.cncf.io/certification/ckad/>
* <https://www.cncf.io/certification/cka/>
* [https://training.linuxfoundation.org/certification/certified-kubernetes-security-specialist](https://training.linuxfoundation.org/certification/certified-kubernetes-security-specialist/)

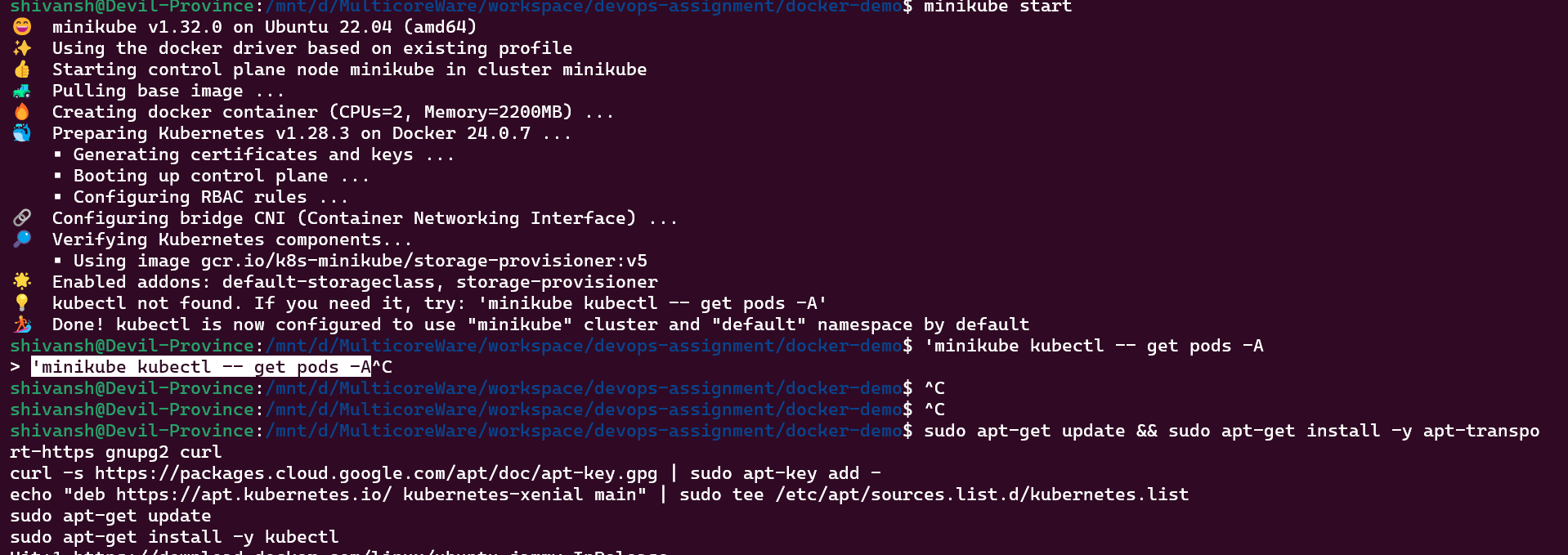
**Kubernetes**:

* Create a local kubernetes cluster with minikube or kind
* Install kubectl in your machine to access the kubernetes cluster
* Create a nginx deployment with 4 replicas using kubectl
* Scale the above replicas to 10 and access the nginx via a load balancer or ingress ( for ingress you can use nginx ingress controller and loadbancer use metallb )
* Delete the above deployment and create the above deployment using kubernetes manifest files
* Append a config map to the above deployment to output hello-world from browser
* Create a secret and mount it to the deployment
* What is the difference between statefulsets vs deployments vs daemonsets
* Create a sidecar and attach to nginx deployment
* Use a init container to make changes to the index.html file for the deployment to output “hello from <your name>”
* Create 2 different nginx deployments in different namespaces and curl each other to get the outputs.
* Create a HPA for nginx to automatically scale for huge loads ( simulate the load using open source tools like locust / j meter / k6 )
* Create resource quota for namespaces
* Limit resources for nginx deployment to use max of 1CPU and 1GB of ram
* Create a network policy to only allow nginx1 to talk to nginx2 which are 2 different deployments and block communication from any other pods ( you need to install a CNI for this, you can use cilium, flannel or calico for this )
* Create a multi node kubernetes cluster ( if you have spare laptop attach it as another node ) if not skip
* Pull a private image from dockerhub by passing the image pull secrets
* Configure affinity and anti-affinity to deployments
* What is Qos in kubernetes
* Install helm and create a basic helm chart
* Install a nginx helm chart
* Learn the templating engine of helm chart and what the .tpl file does in a chart
* Chart dependencies - i.e install nginx1 chart while nginx2 chart is being installed
* Override existing nginx helm chart with values.yaml file

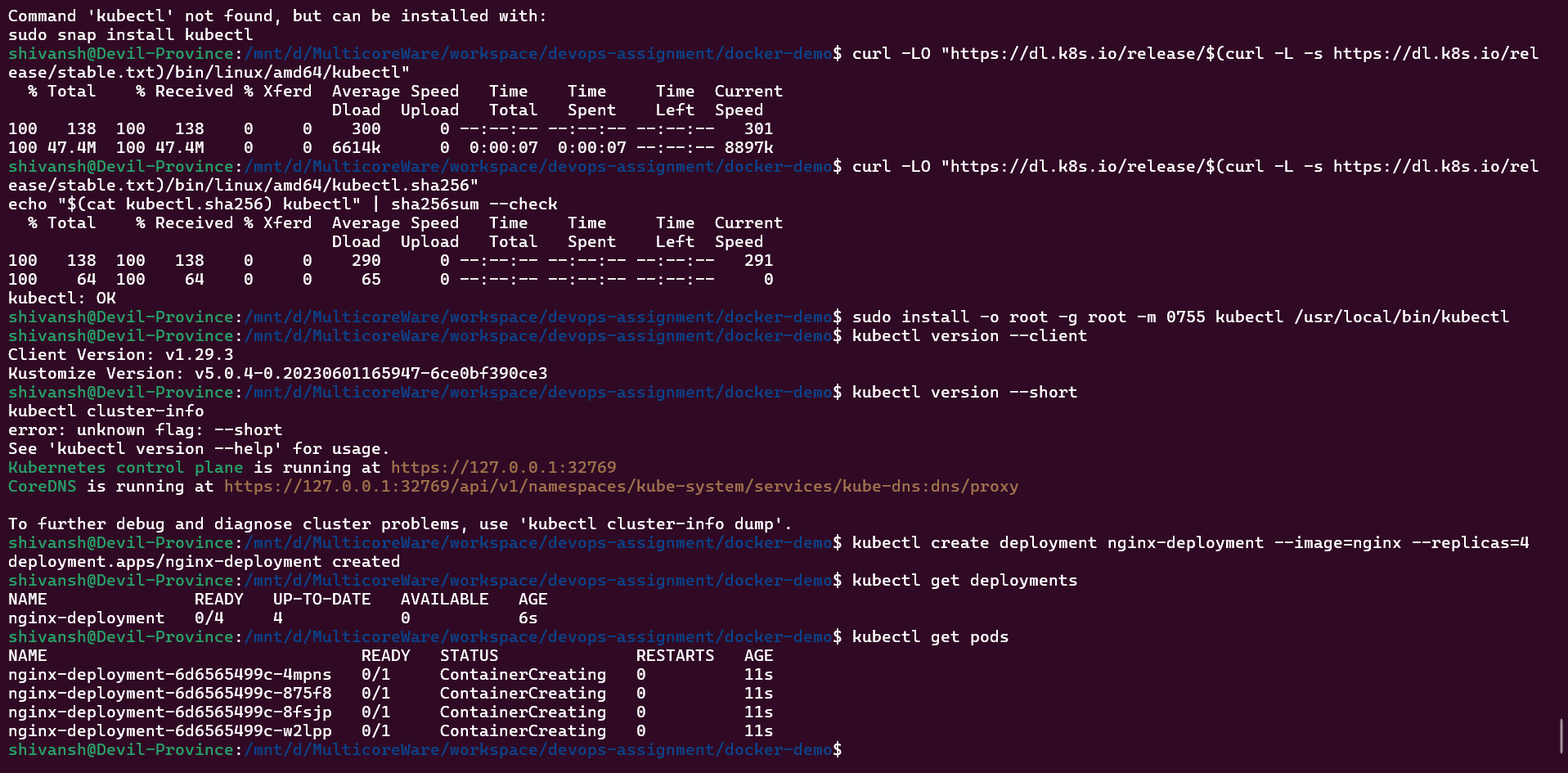
**SOLUTION:**

* Create a local kubernetes cluster with minikube or kind

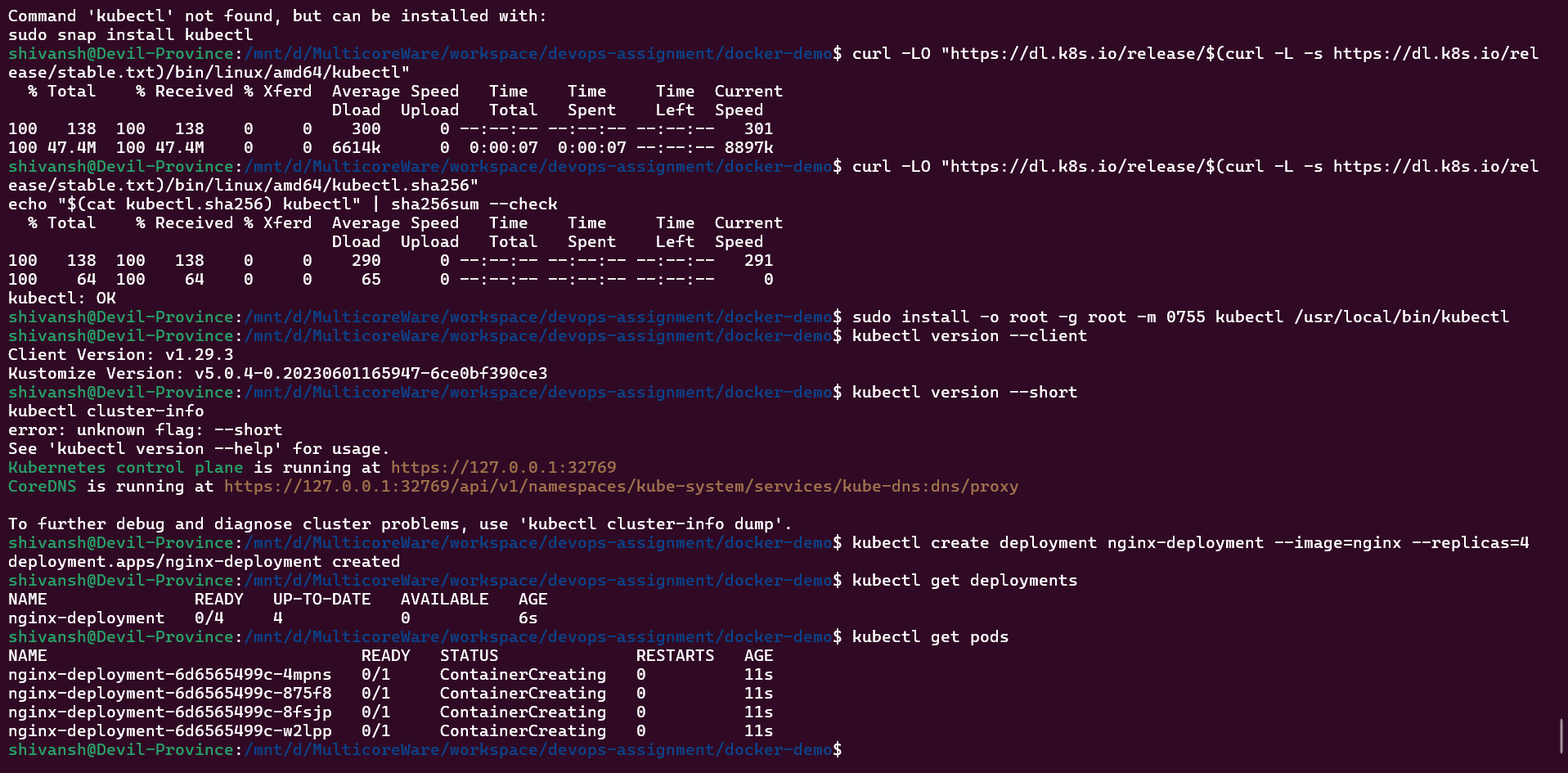
| # Install Minikube curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64 sudo install minikube-linux-amd64 /usr/local/bin/minikube minikube start --driver=none  # Install Kubectl curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl" sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl kubectl version --client  # Create Nginx Deployment kubectl create deployment nginx --image=nginx --replicas=4 kubectl get deployments kubectl get pods  # Expose Nginx Deployment kubectl expose deployment nginx --port=80 --type=NodePort kubectl get services minikube service nginx --url |
| --- |



* Install kubectl in your machine to access the kubernetes cluster



* Create a nginx deployment with 4 replicas using kubectl

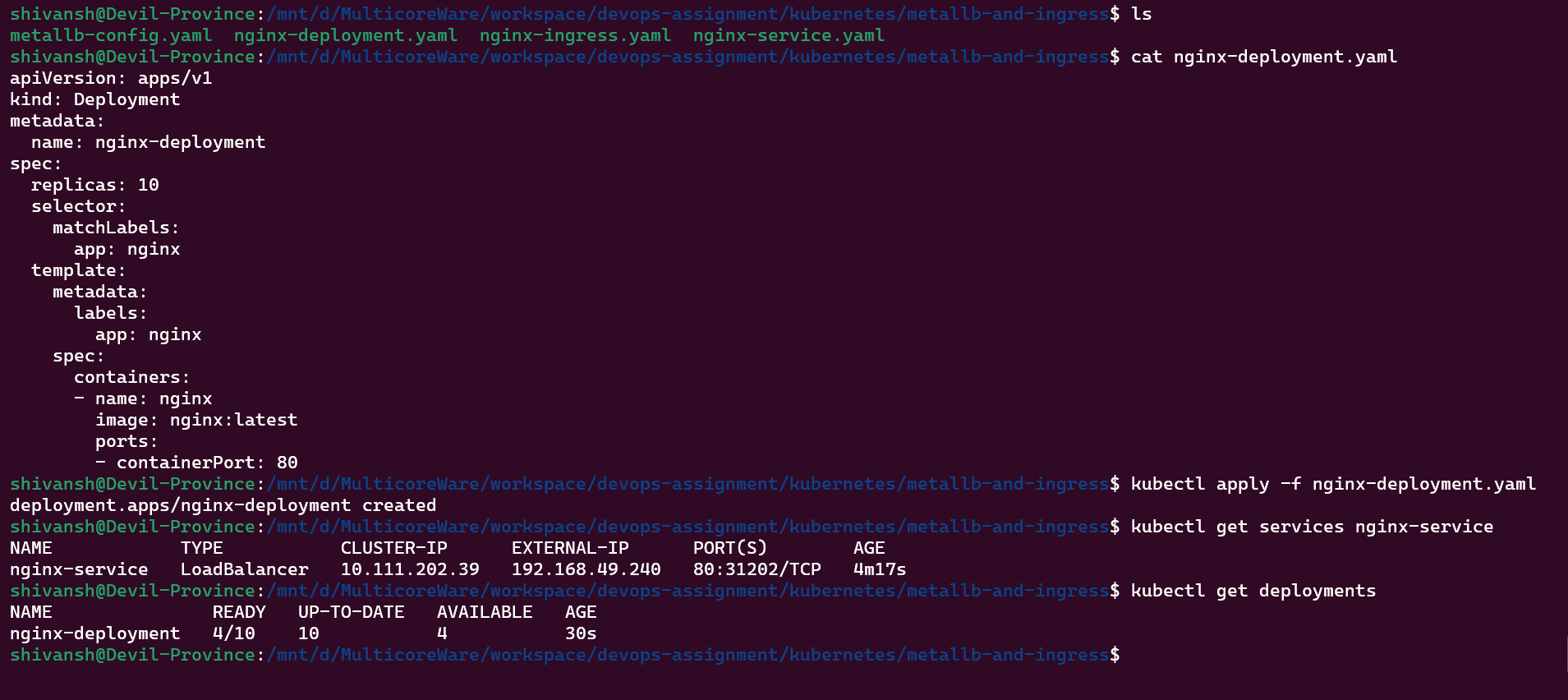


* Scale the above replicas to 10 and access the nginx via a load balancer or ingress ( for ingress you can use nginx ingress controller and loadbancer use metallb )

| # Scale Nginx Deployment to 10 Replicas kubectl scale --replicas=10 deployment/nginx-deployment  # Install Nginx Ingress Controller using Helm # Add the Helm repository for Ingress Nginx helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx # Update Helm repositories helm repo update # Install Ingress Nginx using Helm helm install nginx-ingress ingress-nginx/ingress-nginx  # Install MetalLB (Load Balancer) # Apply MetalLB namespace manifest kubectl apply -f https://raw.githubusercontent.com/metallb/metallb/v0.10.3/manifests/namespace.yaml # Apply MetalLB manifest kubectl apply -f https://raw.githubusercontent.com/metallb/metallb/v0.10.3/manifests/metallb.yaml  # Create and apply MetalLB configuration file cat <<EOF > metallb-config.yaml apiVersion: v1 kind: ConfigMap metadata:  namespace: metallb-system  name: config data:  config: |  address-pools:  - name: default  protocol: layer2  addresses:  - 192.168.49.240-192.168.49.250 EOF # Apply MetalLB configuration kubectl apply -f metallb-config.yaml  # Create a LoadBalancer Service for Nginx deployment cat <<EOF > nginx-service.yaml apiVersion: v1 kind: Service metadata:  name: nginx-service spec:  selector:  app: nginx  ports:  - protocol: TCP  port: 80  targetPort: 80  type: LoadBalancer EOF # Apply the LoadBalancer Service YAML file kubectl apply -f nginx-service.yaml  # Access Nginx via Load Balancer # Get the external IP assigned by MetalLB kubectl get services nginx-service  # Configure Ingress cat <<EOF > nginx-ingress.yaml apiVersion: networking.k8s.io/v1 kind: Ingress metadata:  name: nginx-ingress  annotations:  nginx.ingress.kubernetes.io/rewrite-target: / spec:  rules:  - http:  paths:  - path: /  pathType: Prefix  backend:  service:  name: nginx-service  port:  number: 80 EOF # Apply the Ingress YAML file kubectl apply -f nginx-ingress.yaml |
| --- |

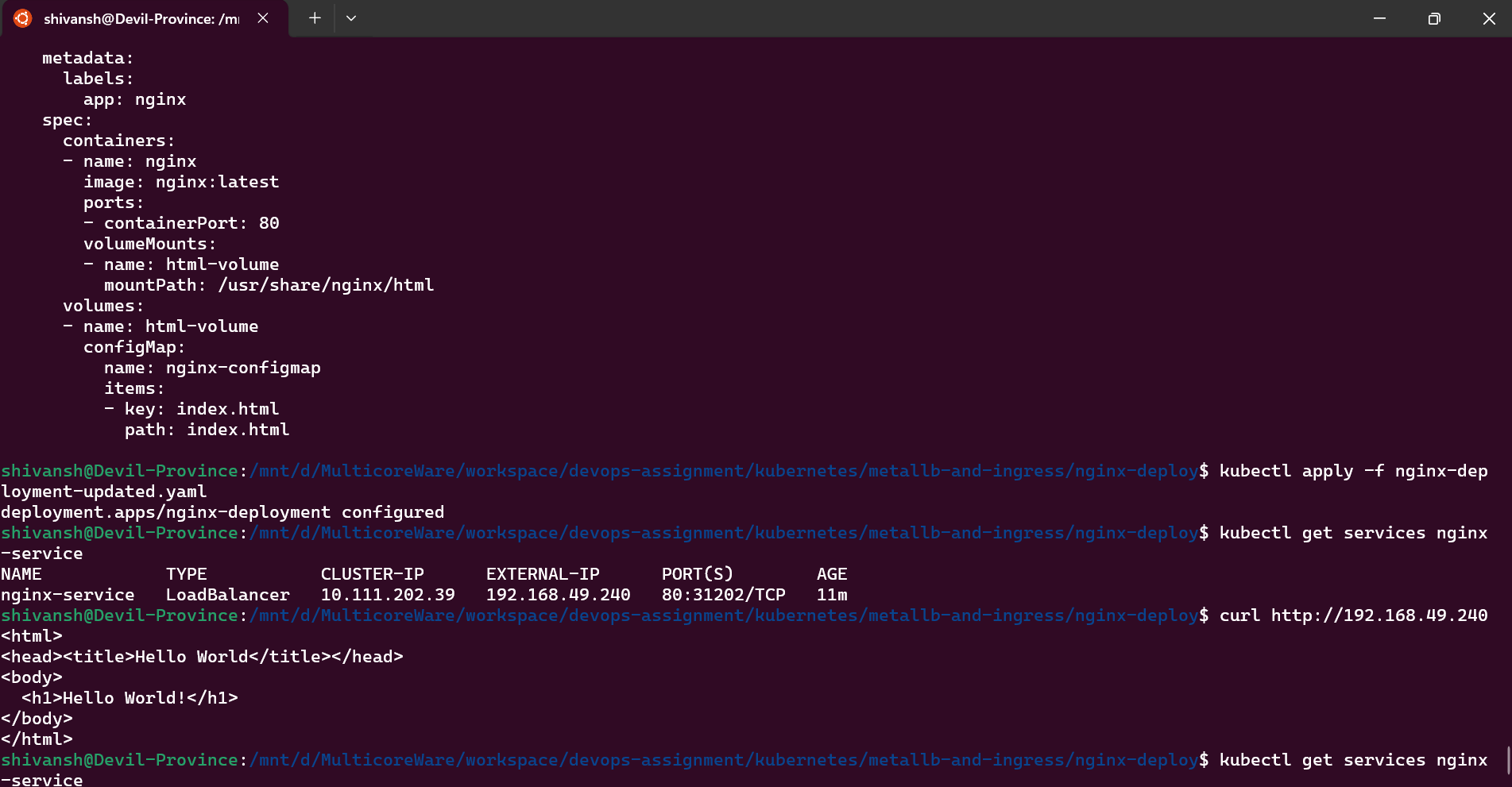


* Delete the above deployment and create the above deployment using kubernetes manifest files

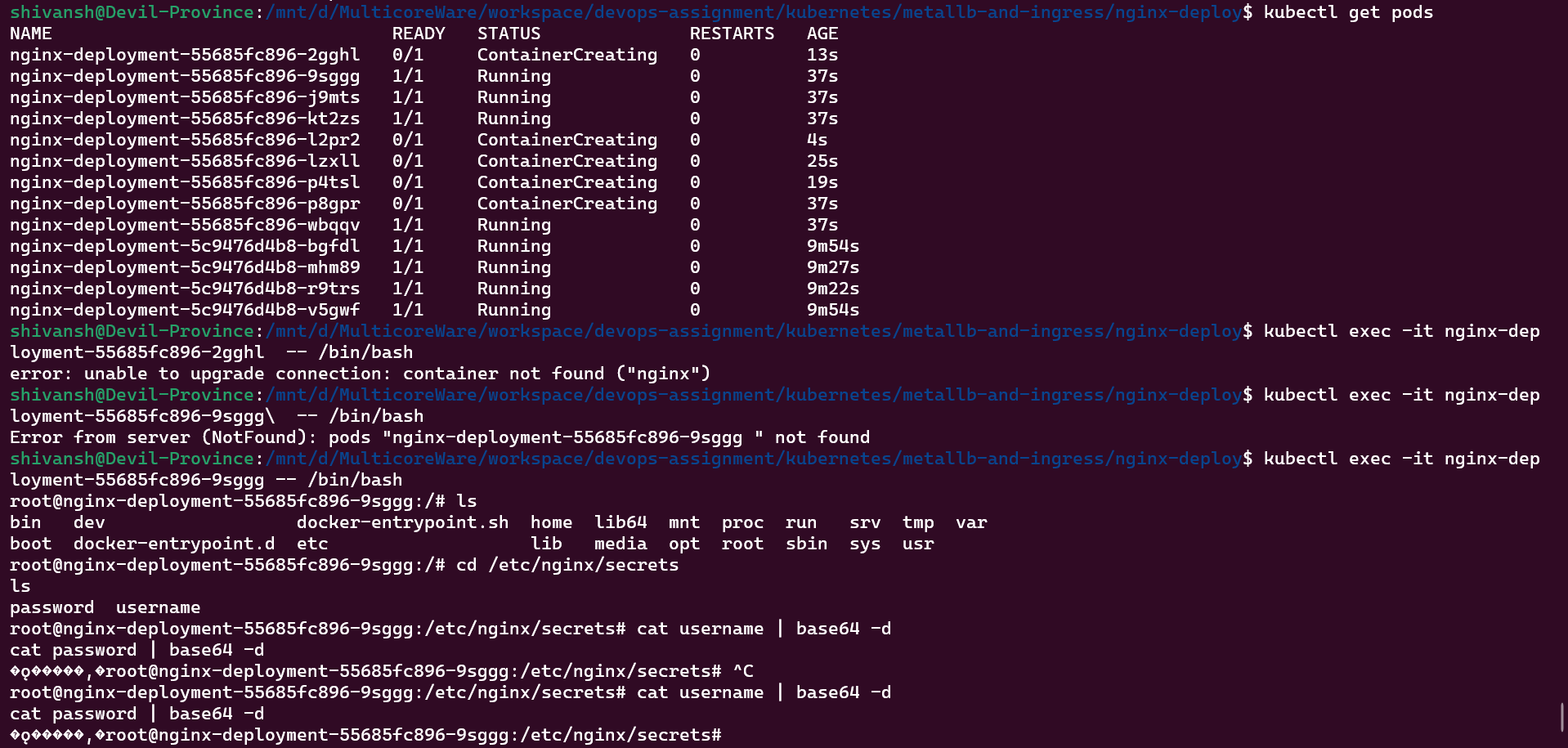


* Append a config map to the above deployment to output hello-world from browser

| # Create the ConfigMap YAML file cat <<EOF > nginx-configmap.yaml apiVersion: v1 kind: ConfigMap metadata:  name: nginx-configmap data:  index.html: |  <html>  <head><title>Hello World</title></head>  <body>  <h1>Hello World!</h1>  </body>  </html> EOF  # Apply the ConfigMap to the Kubernetes cluster kubectl apply -f nginx-configmap.yaml  # Create the updated Nginx deployment YAML file cat <<EOF > nginx-deployment-updated.yaml apiVersion: apps/v1 kind: Deployment metadata:  name: nginx-deployment spec:  replicas: 10  selector:  matchLabels:  app: nginx  template:  metadata:  labels:  app: nginx  spec:  containers:  - name: nginx  image: nginx:latest  ports:  - containerPort: 80  volumeMounts:  - name: html-volume  mountPath: /usr/share/nginx/html  volumes:  - name: html-volume  configMap:  name: nginx-configmap  items:  - key: index.html  path: index.html EOF  # Apply the updated deployment manifest to use the ConfigMap kubectl apply -f nginx-deployment-updated.yaml  # Get the external IP of the Nginx service EXTERNAL\_IP=$(kubectl get services nginx-service -o=jsonpath='{.status.loadBalancer.ingress[0].ip}')  # Print the external IP for reference echo "External IP of Nginx Service: $EXTERNAL\_IP"  # Access Nginx using the external IP in your browser echo "Open your browser and visit: http://$EXTERNAL\_IP" |
| --- |



* Create a secret and mount it to the deployment

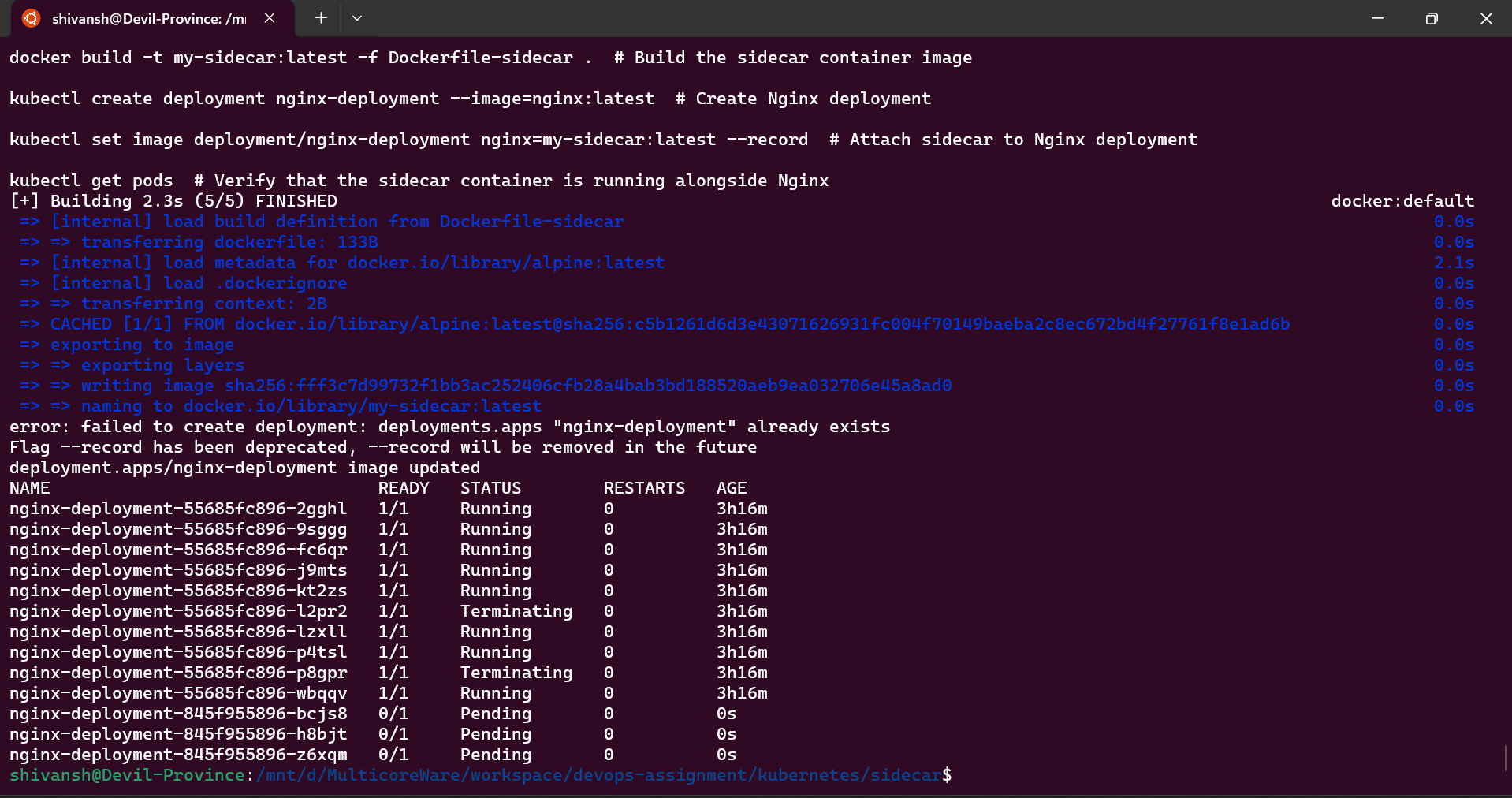


* What is the difference between statefulsets vs deployments vs daemonsets

| —StatefulSets: Ideal for applications requiring stable, unique identities and ordering, such as databases, due to persistent identifiers and sequential pod startup. –Deployments: Best suited for stateless applications that can scale horizontally, offering features like rolling updates, automatic scaling, and ease of management. –DaemonSets: Used for deploying system-level agents or tools on every node, ensuring that specific services or functionalities are available cluster-wide on each node. |
| --- |

* Create a sidecar and attach to nginx deployment

| cat <<EOF > Dockerfile-sidecar # Create Dockerfile for sidecar container FROM alpine:latest CMD ["sh", "-c", "echo 'This is a sidecar container' && sleep 3600"] EOF  docker build -t my-sidecar:latest -f Dockerfile-sidecar . # Build the sidecar container image  kubectl create deployment nginx-deployment --image=nginx:latest # Create Nginx deployment  kubectl set image deployment/nginx-deployment nginx=my-sidecar:latest --record # Attach sidecar to Nginx deployment  kubectl get pods # Verify that the sidecar container is running alongside Nginx |
| --- |



* Use a init container to make changes to the index.html file for the deployment to output “hello from <your name>”

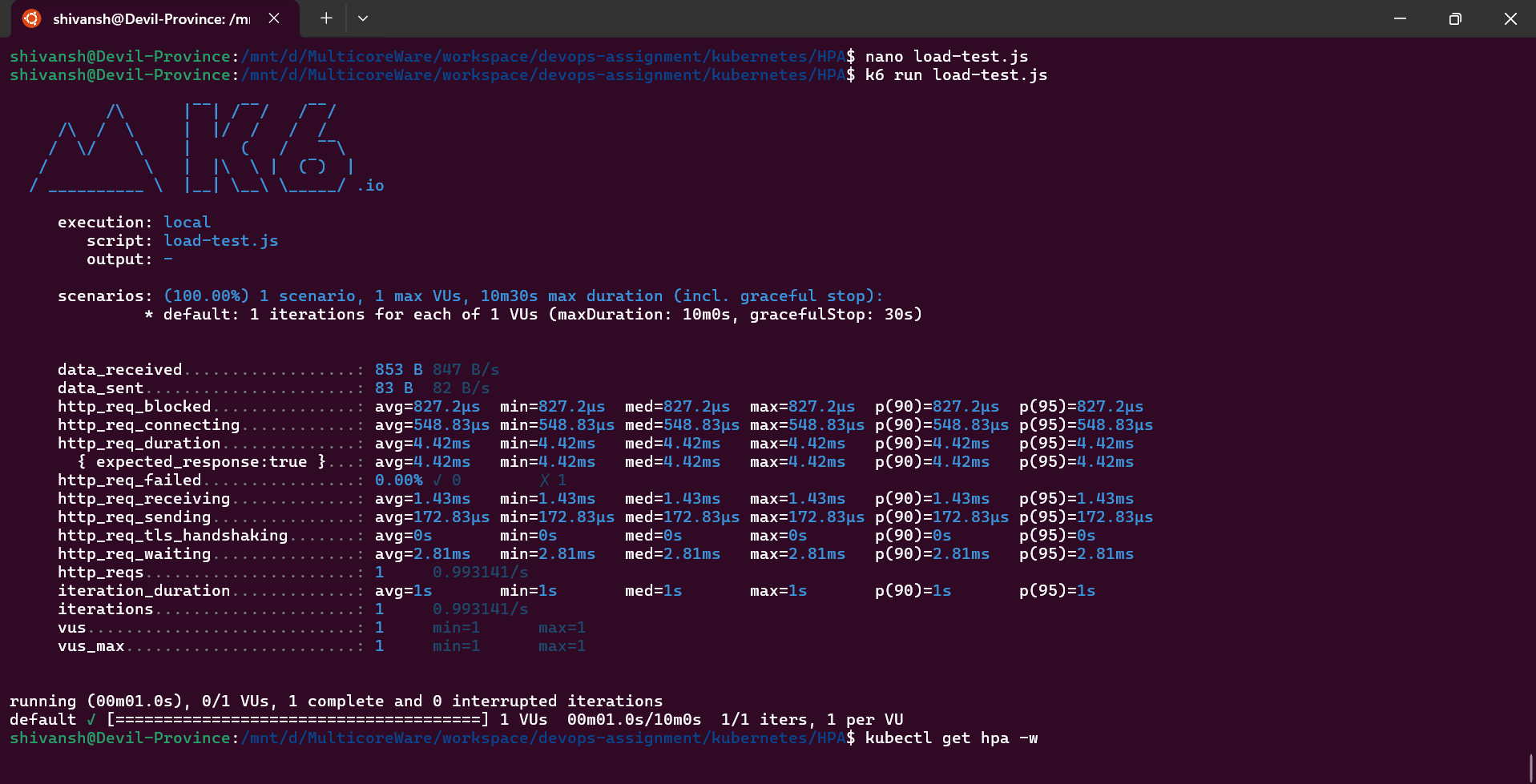
| apiVersion: apps/v1 kind: Deployment metadata:  name: nginx-deployment spec:  replicas: 3  selector:  matchLabels:  app: nginx  template:  metadata:  labels:  app: nginx  spec:  containers:  - name: nginx  image: nginx:latest  ports:  - containerPort: 80  initContainers: # Add init container section  - name: init-container  image: alpine:latest # Use Alpine Linux image for simplicity  command: ["sh", "-c", "/path/to/init-container-script.sh"]  volumeMounts:  - name: html-volume  mountPath: /usr/share/nginx/html  volumes:  - name: html-volume  emptyDir: {} # Use emptyDir volume for simplicity |
| --- |

* Create 2 different nginx deployments in different namespaces and curl each other to get the outputs.

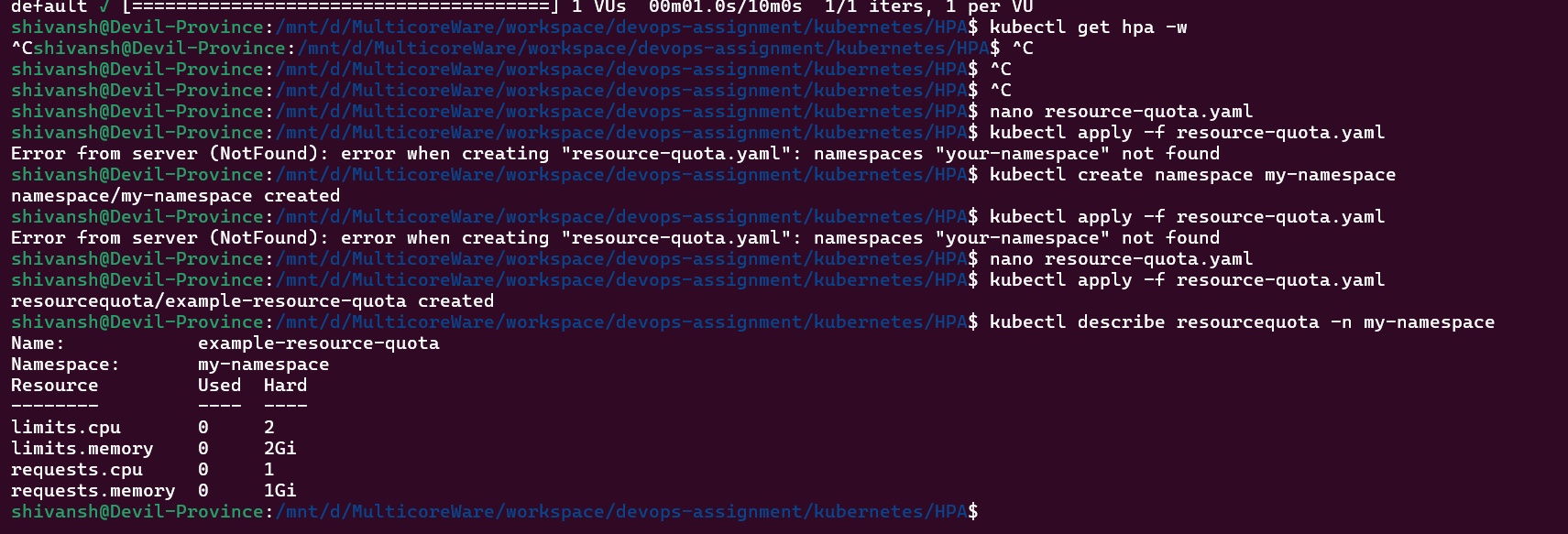
| # Create namespaces kubectl create namespace namespace1 kubectl create namespace namespace2  # Create Nginx deployment YAML files for each namespace cat <<EOF > nginx-deployment-ns1.yaml apiVersion: apps/v1 kind: Deployment metadata:  name: nginx-deployment-ns1  namespace: namespace1 spec:  replicas: 3  selector:  matchLabels:  app: nginx  template:  metadata:  labels:  app: nginx  spec:  containers:  - name: nginx  image: nginx:latest  ports:  - containerPort: 80 EOF  cat <<EOF > nginx-deployment-ns2.yaml apiVersion: apps/v1 kind: Deployment metadata:  name: nginx-deployment-ns2  namespace: namespace2 spec:  replicas: 3  selector:  matchLabels:  app: nginx  template:  metadata:  labels:  app: nginx  spec:  containers:  - name: nginx  image: nginx:latest  ports:  - containerPort: 80 EOF  # Apply deployment YAML files to create deployments kubectl apply -f nginx-deployment-ns1.yaml kubectl apply -f nginx-deployment-ns2.yaml  # Run curl from a pod in namespace1 to fetch output from Nginx in namespace2 kubectl run curl-ns1 --image=alpine --namespace=namespace1 --restart=Never --rm -it -- sh -c "apk add --no-cache curl && curl nginx-deployment-ns2.namespace2.svc.cluster.local"  # Run curl from a pod in namespace2 to fetch output from Nginx in namespace1 kubectl run curl-ns2 --image=alpine --namespace=namespace2 --restart=Never --rm -it -- sh -c "apk add --no-cache curl && curl nginx-deployment-ns1.namespace1.svc.cluster.local" |
| --- |

* Create a HPA for nginx to automatically scale for huge loads ( simulate the load using open source tools like locust / j meter / k6 )

| # Create Nginx Deployment YAML file cat <<EOF > nginx-deployment.yaml apiVersion: apps/v1 kind: Deployment metadata:  name: nginx-deployment spec:  replicas: 1 # Start with a single replica  selector:  matchLabels:  app: nginx  template:  metadata:  labels:  app: nginx  spec:  containers:  - name: nginx  image: nginx:latest  ports:  - containerPort: 80 EOF  # Apply Nginx Deployment kubectl apply -f nginx-deployment.yaml  # Create HPA YAML file cat <<EOF > nginx-hpa.yaml apiVersion: autoscaling/v2beta2 kind: HorizontalPodAutoscaler metadata:  name: nginx-hpa spec:  scaleTargetRef:  apiVersion: apps/v1  kind: Deployment  name: nginx-deployment  minReplicas: 1  maxReplicas: 5 # Adjust according to your requirements  metrics:  - type: Resource  resource:  name: cpu  targetAverageUtilization: 50 # Set the target CPU utilization percentage EOF  # Apply HPA kubectl apply -f nginx-hpa.yaml  # Create k6 load test script cat <<EOF > load-test.js import http from 'k6/http'; import { sleep } from 'k6';  export default function () {  http.get('http://<nginx-service-ip>:<nginx-service-port>');  sleep(1); } EOF  # Replace <nginx-service-ip> and <nginx-service-port> with actual values in load-test.js  # Run k6 load test # Install k6 if not already installed: https://k6.io/docs/getting-started/installation/ k6 run load-test.js  # Monitor HPA and Scaling kubectl get hpa -w |
| --- |

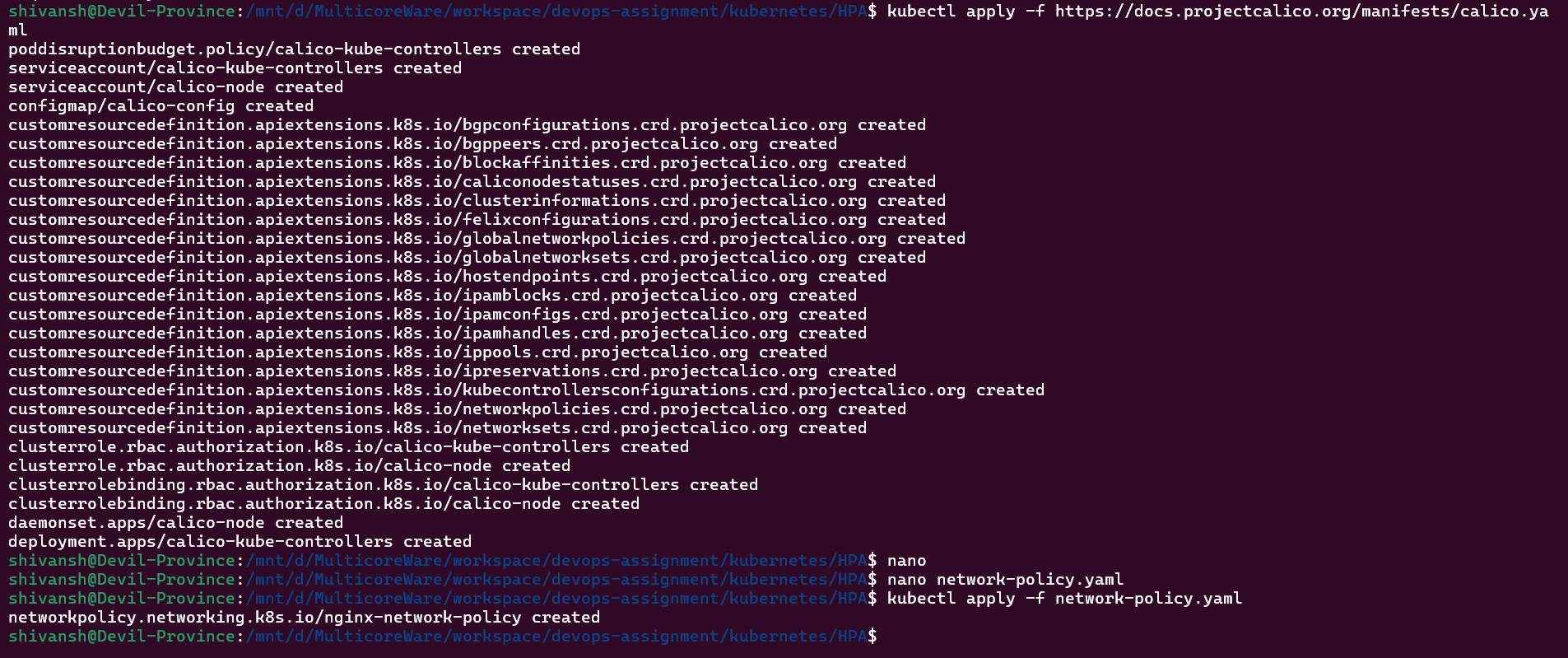


* Create resource quota for namespaces



* Limit resources for nginx deployment to use max of 1CPU and 1GB of ram

| # Create a ResourceQuota YAML file named resource-quota.yaml cat <<EOF > resource-quota.yaml apiVersion: v1 kind: ResourceQuota metadata:  name: nginx-resource-quota # Name for the ResourceQuota  namespace: my-namespace  spec:  hard:  requests.cpu: "1" # Maximum CPU request allowed  requests.memory: "1Gi" # Maximum memory request allowed  limits.cpu: "1" # Maximum CPU limit allowed  limits.memory: "1Gi" # Maximum memory limit allowed EOF  # Apply the ResourceQuota in Kubernetes kubectl apply -f resource-quota.yaml |
| --- |

* Create a network policy to only allow nginx1 to talk to nginx2 which are 2 different deployments and block communication from any other pods ( you need to install a CNI for this, you can use cilium, flannel or calico for this )
* 
* **Create a multi node kubernetes cluster ( if you have spare laptop attach it as another node ) if not skip**
  + Prepare Nodes:
    - Ensure spare laptops or machines meet Kubernetes requirements.
  + Install Docker on each node.
  + Install Kubernetes Tools
    - Install kubeadm, kubelet, and kubectl on all nodes.
  + Initialize Master Node:
    - Choose one node as the master and run kubeadm init to set up control plane.
  + Join Worker Nodes:
    - On other nodes, run kubeadm join to connect them to the master.
  + Set Up Networking:
    - Choose a CNI plugin (e.g., Calico, Flannel) and configure it on the cluster.
  + Verify Cluster Setup:
    - Use kubectl get nodes to ensure all nodes, including master and workers, are visible.
  + Test Functionality:
    - Deploy sample applications (e.g., Nginx, WordPress) to test cluster functionality.
* Pull a private image from dockerhub by passing the image pull secrets

| # Create Docker Hub Secret with your credentials kubectl create secret docker-registry dockerhub-secret \  --docker-username=<username> \  --docker-password=<password> \  --docker-server=https://index.docker.io/v1/ && \  # Apply Pod manifest that uses the created secret for image pulling kubectl apply -f pod-manifest.yaml |
| --- |

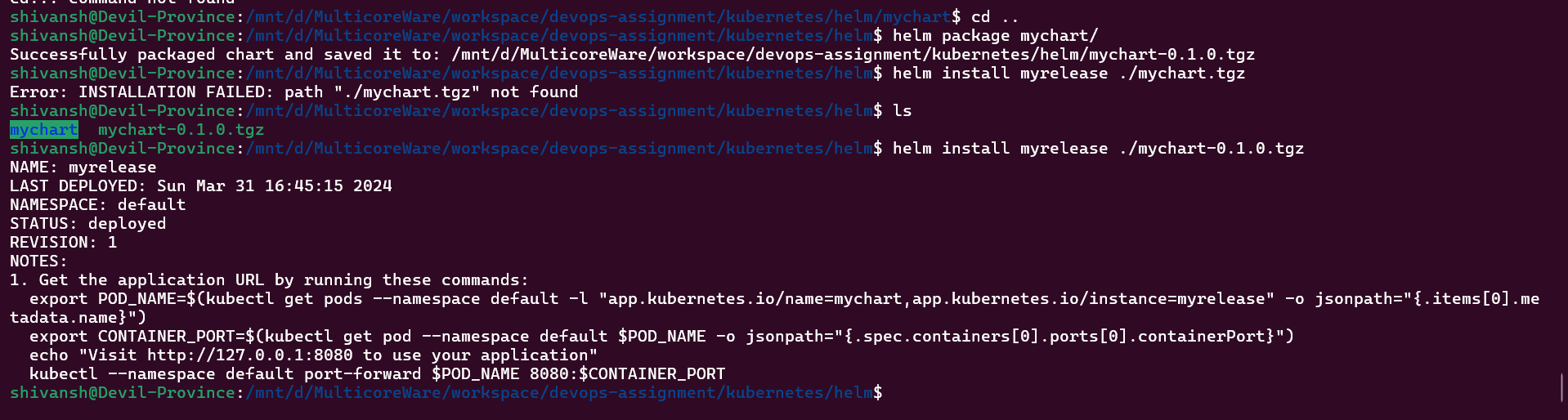
* Configure affinity and anti-affinity to deployments
  + Affinity:

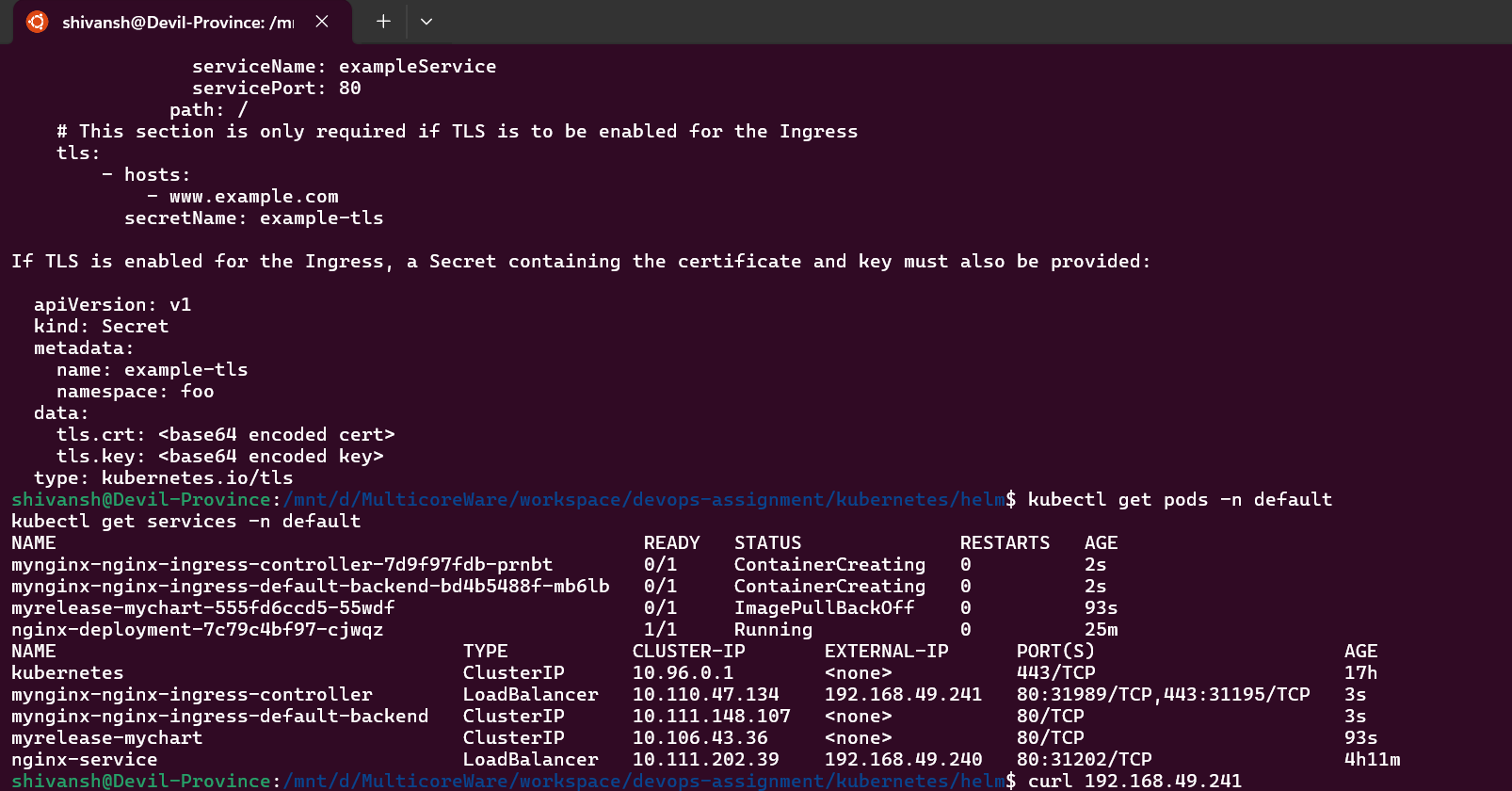
| apiVersion: apps/v1 kind: Deployment metadata:  name: example-deployment spec:  replicas: 3  selector:  matchLabels:  app: example-app  template:  metadata:  labels:  app: example-app  spec:  affinity:  podAffinity:  requiredDuringSchedulingIgnoredDuringExecution:  - labelSelector:  matchExpressions:  - key: "app"  operator: In  values:  - example-app  topologyKey: "kubernetes.io/hostname" |
| --- |

Anti-Affinity:

| apiVersion: apps/v1 kind: Deployment metadata:  name: example-deployment spec:  replicas: 3  selector:  matchLabels:  app: example-app  template:  metadata:  labels:  app: example-app  spec:  affinity:  podAntiAffinity:  requiredDuringSchedulingIgnoredDuringExecution:  - labelSelector:  matchExpressions:  - key: "app"  operator: In  values:  - example-app  topologyKey: "kubernetes.io/hostname" |
| --- |

* What is Qos in kubernetes
  + In Kubernetes, QoS (Quality of Service) classes categorize pods based on resource guarantees: Guaranteed (high priority), Burstable (medium priority), and BestEffort (low priority), ensuring efficient resource allocation and workload prioritization.
* Install helm and create a basic helm chart



* Install a nginx helm chart
* 
* Learn the templating engine of helm chart and what the .tpl file does in a chart

Helm uses Go templating engine to dynamically generate Kubernetes manifest files in Helm charts. .tpl files within a Helm chart's templates directory contain Kubernetes manifest definitions with embedded Go templating syntax, allowing for parameterization and reusability of configurations during chart installation.

* Chart dependencies - i.e install nginx1 chart while nginx2 chart is being installed

| # Create or modify the requirements.yaml file in your Helm chart directory with the specified dependencies echo "dependencies:  - name: nginx1  version: \"1.0.0\"  repository: \"https://example.com/charts/nginx1\"" > requirements.yaml  # Replace "https://example.com/charts/nginx1" with the actual repository URL of the nginx1 chart  # Install your Helm chart and its dependencies using the helm install command helm install my-nginx-chart ./my-nginx-chart  # Helm will automatically fetch and install the specified dependencies along with your main chart # Helm will resolve the dependencies specified in requirements.yaml and install nginx1 along with your main chart my-nginx-chart |
| --- |

* Override existing nginx helm chart with values.yaml file

| cat <<EOF > values.yaml && helm install my-nginx ./nginx-chart --values values.yaml # values.yaml content nginx:  replicaCount: 2  image:  repository: nginx  tag: "1.19.10"  service:  type: LoadBalancer EOF |
| --- |