#### **Project details are available from page 13**

#### **1. ConfigMap**

**ConfigMap** is a Kubernetes object that lets you store configuration data in key-value pairs. It is used to manage non-sensitive configuration information separately from the application code.

**Creating a ConfigMap**

You can create a ConfigMap from a literal value or from a file. Here’s an example of creating a ConfigMap from literal values:

apiVersion: v1

kind: ConfigMap

metadata:

name: web-config

data:

DATABASE\_URL: "jdbc:mysql://db-server:3306/mydatabase"

APP\_ENV: "production"

**Using ConfigMap in a Pod**

To use the ConfigMap in a Pod, you need to reference it in your Pod specification. Here’s how you can inject ConfigMap values as environment variables:

apiVersion: v1

kind: Pod

metadata:

name: web-app

spec:

containers:

- name: web-container

image: my-web-app:latest

env:

- name: DATABASE\_URL

valueFrom:

configMapKeyRef:

name: web-config

key: DATABASE\_URL

- name: APP\_ENV

valueFrom:

configMapKeyRef:

name: web-config

key: APP\_ENV

**Example Use Case: Mounting ConfigMap as a File**

Sometimes, an application may require configuration files. You can mount a ConfigMap as a file inside a container.

apiVersion: v1

kind: ConfigMap

metadata:

name: config-files

data:

config.yaml: |

database:

url: "jdbc:mysql://db-server:3306/mydatabase"

environment: "production"

**Mount the ConfigMap as a volume in the Pod:**

apiVersion: v1

kind: Pod

metadata:

name: web-app

spec:

containers:

- name: web-container

image: my-web-app:latest

volumeMounts:

- name: config-volume

mountPath: /etc/config

volumes:

- name: config-volume

configMap:

name: config-files

The configuration file config.yaml will be available at /etc/config/config.yaml inside the container.

#### **2. Secrets**

**Secrets** is a Kubernetes object designed to hold sensitive data such as passwords, OAuth tokens, and SSH keys. Secrets ensure that sensitive information is stored securely.

**Creating a Secret**

You can create a Secret from literal values or from files. Here’s an example of creating a Secret from literal values:

apiVersion: v1

kind: Secret

metadata:

name: db-credentials

type: Opaque

data:

username: dXNlcm5hbWU= # base64 encoded 'username'

password: cGFzc3dvcmQ= # base64 encoded 'password'

**Using Secrets in a Pod**

To use the Secret in a Pod, reference it in your Pod specification and inject it as environment variables:

apiVersion: v1

kind: Pod

metadata:

name: web-app

spec:

containers:

- name: web-container

image: my-web-app:latest

env:

- name: DB\_USERNAME

valueFrom:

secretKeyRef:

name: db-credentials

key: username

- name: DB\_PASSWORD

valueFrom:

secretKeyRef:

name: db-credentials

key: password

**Example Use Case: Mounting Secrets as Files**

For applications that require secrets as files, you can mount the Secret as a volume inside a container.

apiVersion: v1

kind: Secret

metadata:

name: ssh-keys

type: Opaque

data:

ssh-privatekey: <base64-encoded-private-key>

ssh-publickey: <base64-encoded-public-key>

Mount the Secret as a volume in the Pod:

apiVersion: v1

kind: Pod

metadata:

name: web-app

spec:

containers:

- name: web-container

image: my-web-app:latest

volumeMounts:

- name: ssh-volume

mountPath: /etc/ssh

readOnly: true

volumes:

- name: ssh-volume

secret:

secretName: ssh-keys

The SSH keys will be available at /etc/ssh inside the container.

#### **3. Environment Variables**

Environment variables are a way to pass configuration settings to applications running inside containers. They can be defined directly in the Pod specification or sourced from ConfigMaps and Secrets.

**Example Use Case: Passing Configuration to a Container**

Environment variables can be used to pass various configurations like application mode, API endpoints, and feature flags to the container.

**Defining Environment Variables in Pod Specification**

apiVersion: v1

kind: Pod

metadata:

name: web-app

spec:

containers:

- name: web-container

image: my-web-app:latest

env:

- name: APP\_MODE

value: "production"

- name: API\_ENDPOINT

value: "https://api.example.com"

**Example Use Case: Using Environment Variables from ConfigMaps and Secrets**

Combining ConfigMaps and Secrets with environment variables provides a flexible and secure way to manage configurations.

**Using ConfigMap and Secret Environment Variables Together**

apiVersion: v1

kind: Pod

metadata:

name: web-app

spec:

containers:

- name: web-container

image: my-web-app:latest

env:

- name: DATABASE\_URL

valueFrom:

configMapKeyRef:

name: web-config

key: DATABASE\_URL

- name: APP\_ENV

valueFrom:

configMapKeyRef:

name: web-config

key: APP\_ENV

- name: DB\_USERNAME

valueFrom:

secretKeyRef:

name: db-credentials

key: username

- name: DB\_PASSWORD

valueFrom:

secretKeyRef:

name: db-credentials

key: password

**Autoscaling in Kubernetes**

### **1. Horizontal Pod Autoscaler (HPA)**

**1.1. Define a Deployment**

apiVersion: apps/v1

kind: Deployment

metadata:

name: web-app

spec:

replicas: 2

selector:

matchLabels:

app: web-app

template:

metadata:

labels:

app: web-app

spec:

containers:

- name: web-container

image: my-web-app:latest

ports:

- containerPort: 80

resources:

requests:

cpu: "500m"

limits:

cpu: "1"

**1.2. Apply the Deployment**

kubectl apply -f deployment.yaml

**1.3. Create a Service**

apiVersion: v1

kind: Service

metadata:

name: web-service

spec:

selector:

app: web-app

ports:

- protocol: TCP

port: 80

targetPort: 80

type: LoadBalancer

**1.4. Apply the Service**

kubectl apply -f service.yaml

**1.5. Create an HPA**

Define an HPA to scale the number of pods based on CPU utilization:

apiVersion: autoscaling/v2beta2

kind: HorizontalPodAutoscaler

metadata:

name: web-app-hpa

spec:

scaleTargetRef:

apiVersion: apps/v1

kind: Deployment

name: web-app

minReplicas: 2

maxReplicas: 10

metrics:

- type: Resource

resource:

name: cpu

target:

type: Utilization

averageUtilization: 50

**1.6. Apply the HPA**

kubectl apply -f hpa.yaml

### **Vertical Pod Autoscaler (VPA)**

**2.1. Define a Deployment**

Create a Deployment for the batch job:

apiVersion: apps/v1

kind: Deployment

metadata:

name: batch-job

spec:

replicas: 1

selector:

matchLabels:

app: batch-job

template:

metadata:

labels:

app: batch-job

spec:

containers:

- name: batch-container

image: my-batch-job:latest

resources:

requests:

cpu: "500m"

memory: "1Gi"

limits:

cpu: "1"

memory: "2Gi"

**2.2. Apply the Deployment**

kubectl apply -f deployment.yaml

**2.3. Create a VPA**

Define a VPA to manage the resource requests and limits for the Pod:

apiVersion: verticalpodautoscaler.k8s.io/v1

kind: VerticalPodAutoscaler

metadata:

name: batch-job-vpa

spec:

targetRef:

apiVersion: apps/v1

kind: Deployment

name: batch-job

updatePolicy:

updateMode: Auto

**2.4. Apply the VPA**

kubectl apply -f vpa.yaml

**Linux Scripts**

### **Viewing Processes (ps, top)**

#### **Use Cases:**

1. **System Monitoring**:
   * **Example**: An administrator needs to check the status of all running processes to ensure that critical applications are running smoothly.

**Commands**:  
  
ps aux # Displays detailed information about all running processes

top # Interactive view of system processes, updates in real time

1. **Troubleshooting Performance Issues**:
   * **Example**: A developer notices that the server is slow and needs to find out which processes are consuming the most CPU or memory.

**Commands**:  
  
top # Look for processes consuming high CPU or memory

ps -eo pid,comm,%cpu,%mem --sort=-%cpu | head # Display top 10 processes by CPU usage

1. **Identifying Zombie Processes**:
   * **Example**: The system administrator is dealing with processes that are stuck in the “zombie” state.

**Commands**:  
  
ps aux | grep 'Z' # Finds processes in a zombie state

#### **Examples:**

**Example 1**:  
  
ps aux | grep nginx

* Finds processes related to the nginx web server.

**Example 2**:  
  
top -u username

* Displays processes owned by a specific user.

### **Managing Processes (kill, nice)**

#### **Use Cases:**

1. **Stopping Unresponsive Applications**:
   * **Example**: A user needs to stop a process that has become unresponsive or is consuming excessive resources.

**Commands**:  
  
kill -9 12345 # Forcefully terminates the process with PID 12345

1. **Adjusting Process Priority**:
   * **Example**: A system administrator wants to lower the priority of a process to ensure it does not hog resources.

**Commands**:  
  
nice -n 10 command # Start a process with a lower priority

renice +10 -p 12345 # Change the priority of an existing process with PID 12345

1. **Gracefully Stopping Services**:
   * **Example**: An admin needs to restart a service to apply configuration changes.

**Commands**:  
  
kill -HUP 12345 # Sends a SIGHUP signal to the process to reload configuration

#### **Examples:**

**Example 1**:  
  
killall -9 firefox

* Kills all processes named firefox.

**Example 2**:  
  
nice -n -10 ./heavy\_script.sh

* Runs heavy\_script.sh with a higher priority.

### **Configure SSH**

### **Shell Scripts**

#### **Writing Basic Shell Scripts**

##### **Use Cases:**

1. **Automating Routine Tasks**:
   * **Example**: A sysadmin wants to automate the backup of log files.

**Commands**:  
  
#!/bin/bash

cp /var/log/syslog /backup/syslog-$(date +%F).log

1. **System Maintenance**:
   * **Example**: A developer creates a script to clean up temporary files.

**Commands**:  
  
#!/bin/bash

rm -rf /tmp/\*

1. **Batch Processing**:
   * **Example**: A data analyst needs to process multiple files in a directory.

**Commands**:  
  
#!/bin/bash

process\_file() {

local file="$1"

echo "Processing $file"

# Add more commands to process the file here

}

for file in /data/\*.csv; do

process\_file "$file"

done

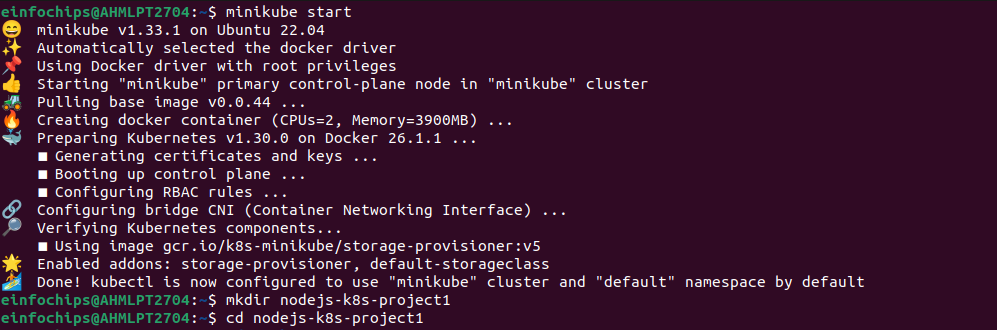
**Project 01**

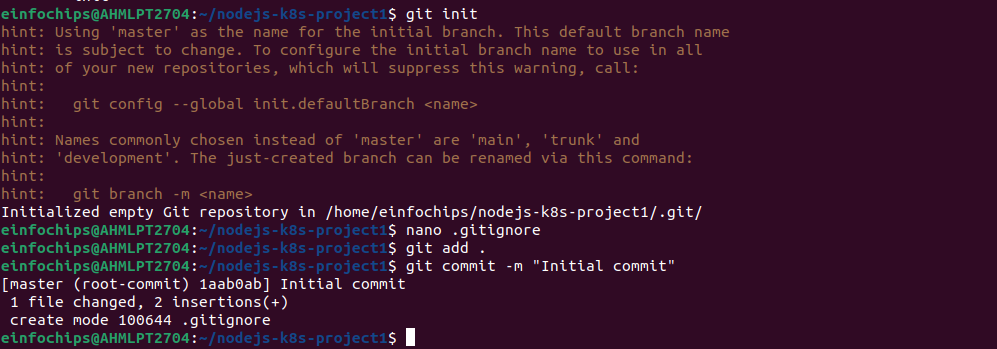
In this project, you will develop a simple Node.js application, deploy it on a local Kubernetes cluster using Minikube, and configure various Kubernetes features. The project includes Git version control practices, creating and managing branches, and performing rebases. Additionally, you will work with ConfigMaps, Secrets, environment variables, and set up vertical and horizontal pod autoscaling.

## **Project 01**

## **Project Steps**

### **1. Setup Minikube and Git Repository**





### **2. Develop a Node.js Application**

#### 

### **3. Create Dockerfile and Docker Compose**

#### 

### **4. Build and Push Docker Image**

#### 

### 

### **5. Create Kubernetes Configurations**

#### 

### **6. Implement Autoscaling**

#### **6.1 Create Horizontal Pod Autoscaler**

**Create kubernetes/hpa.yaml**:



#### **6.2 Create Vertical Pod Autoscaler**

**Create kubernetes/vpa.yaml**:  
  
apiVersion: autoscaling.k8s.io/v1beta2

kind: VerticalPodAutoscaler

metadata:

name: nodejs-app-vpa

spec:

targetRef:

apiVersion: apps/v1

kind: Deployment

name: nodejs-app-deployment

updatePolicy:

updateMode: "Auto"

**Apply the VPA**:  
  
kubectl apply -f kubernetes/vpa.yaml

### **7. Test the Deployment**

#### 

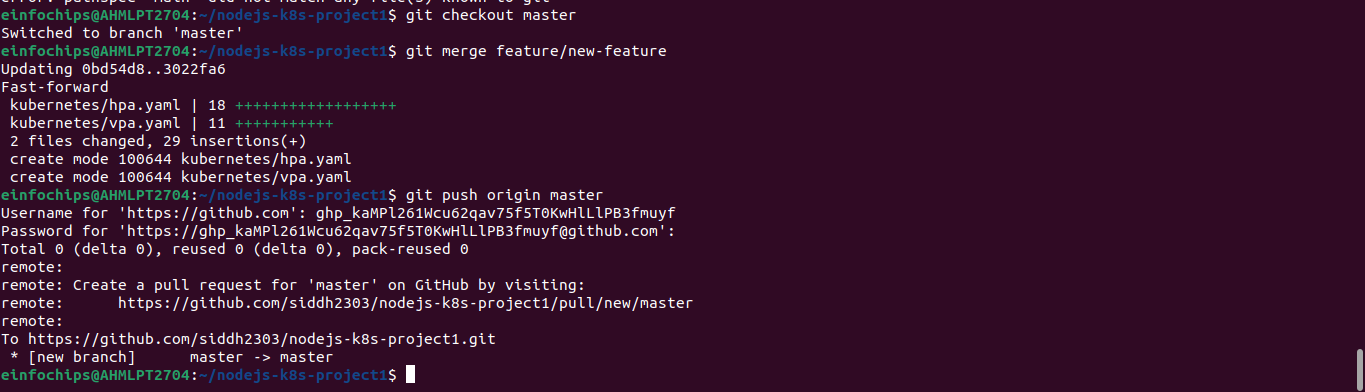
### **8. Git Version Control**

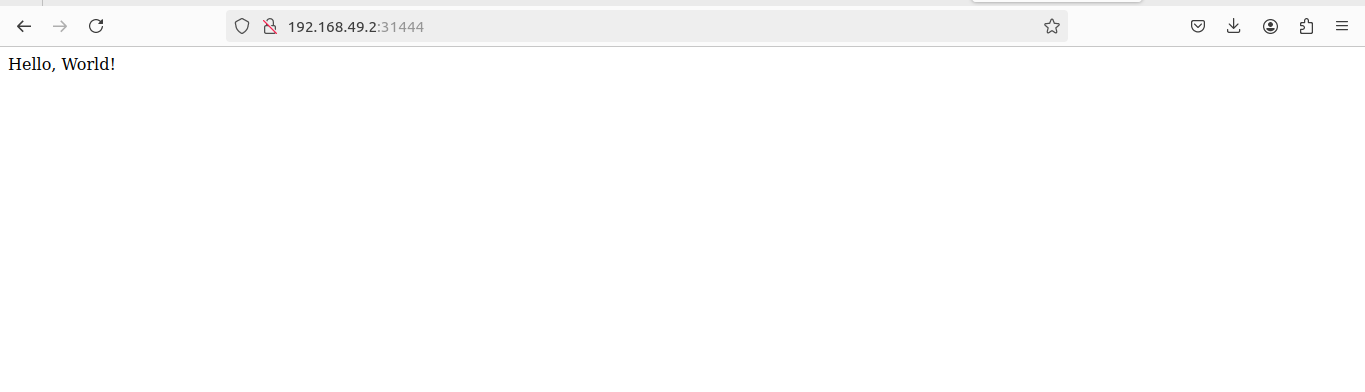
#### 

### 

### 

### **9. Final Commit and Cleanup**



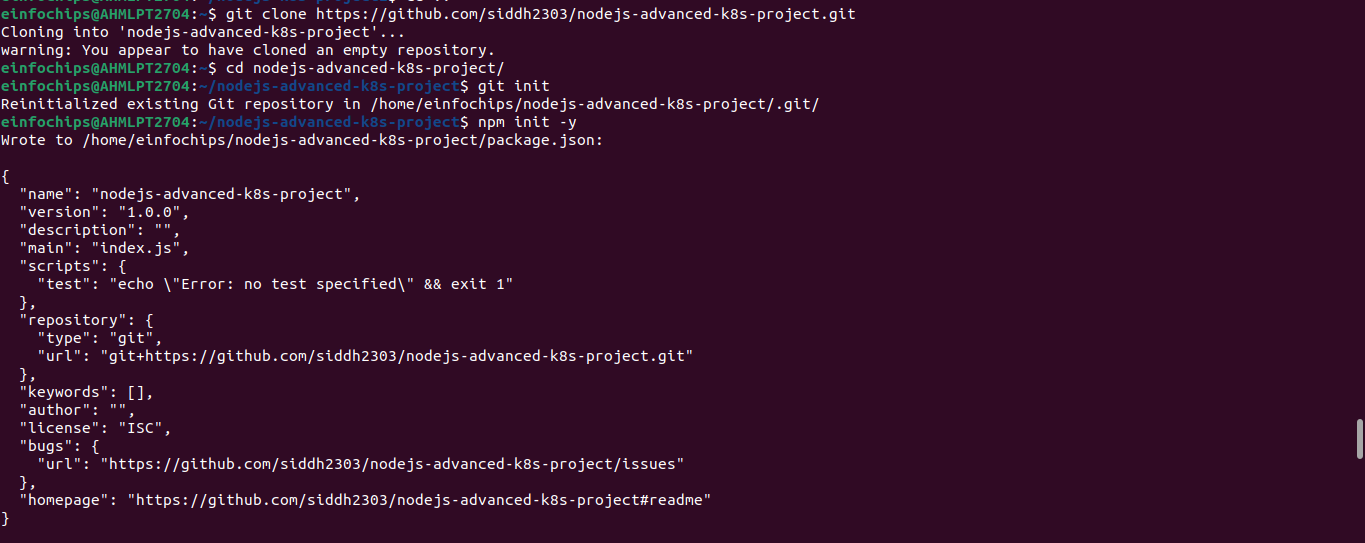


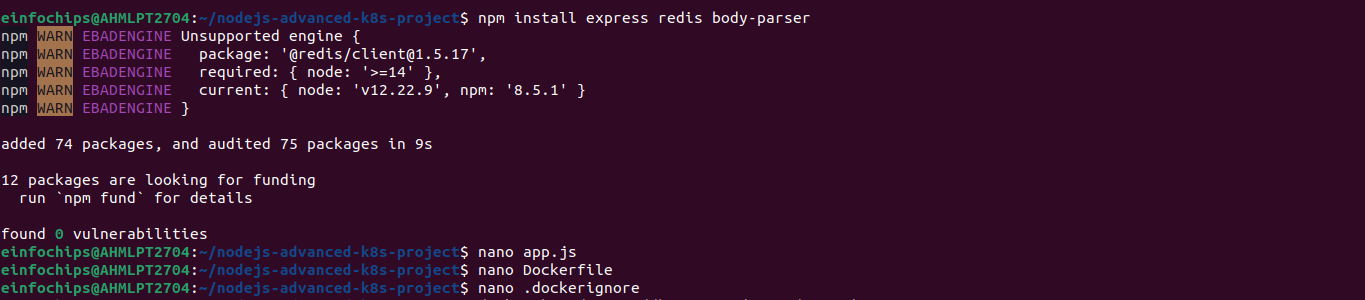
**Project 02**

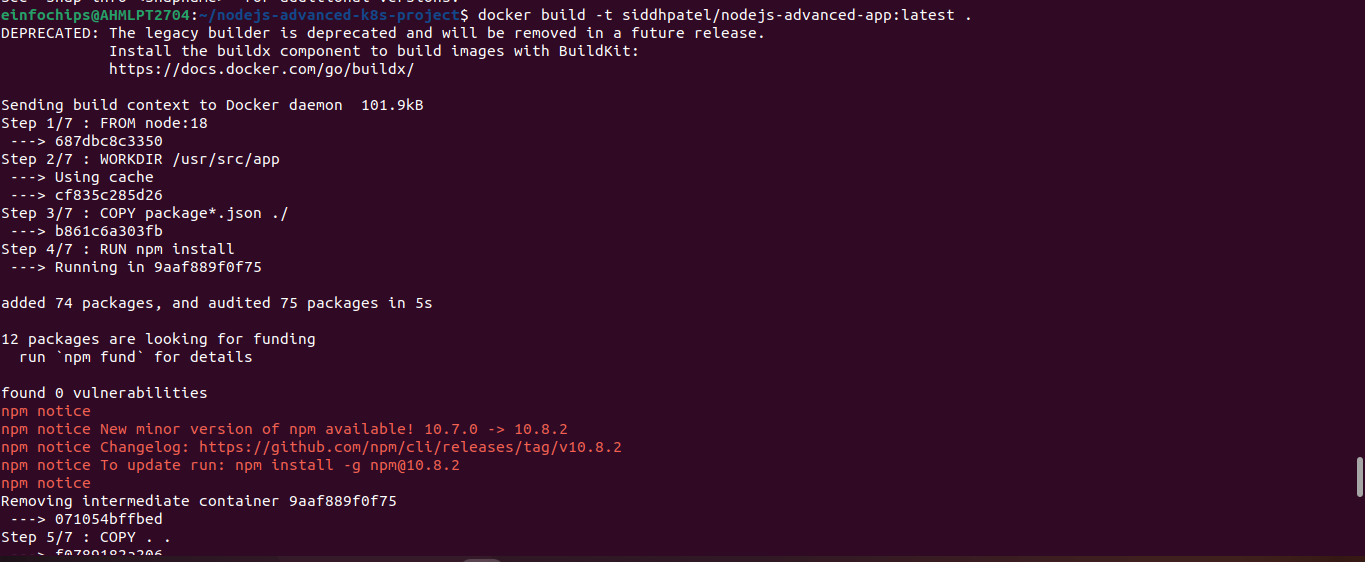
Deploy a Node.js application to Kubernetes with advanced usage of ConfigMaps and Secrets. Implement Horizontal Pod Autoscaler (HPA) with both scale-up and scale-down policies. The project will include a multi-environment configuration strategy, integrating a Redis cache, and monitoring application metrics.

## **Project Setup**

### **1.1 Initialize a Git Repository**

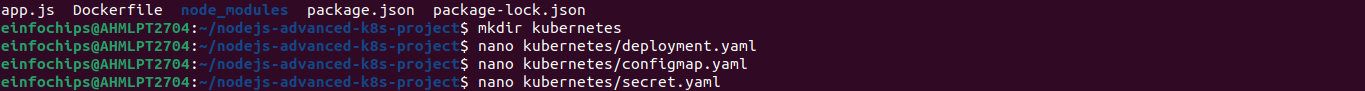


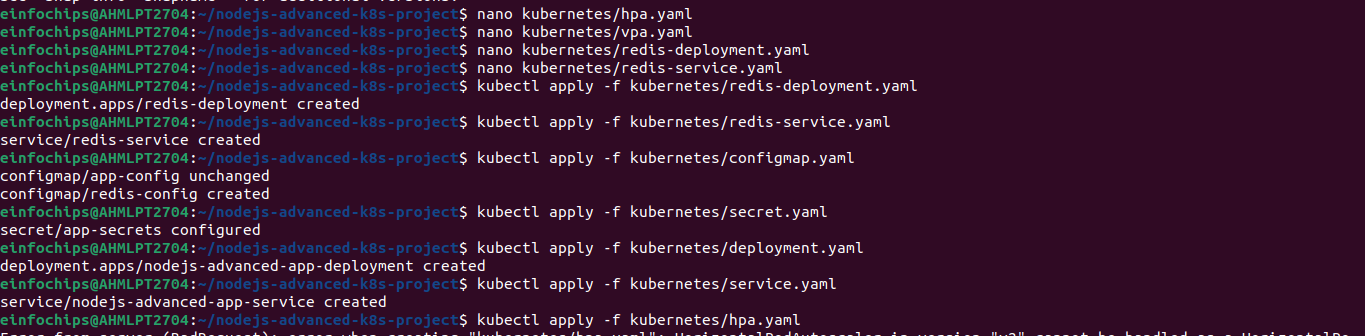




****

**2. Advanced Kubernetes Configuration**





### **2.9 Verify Deployments and Services**

Check the status of your deployments and services:

kubectl get all

Access the application via Minikube:

minikube service nodejs-advanced-app-service --url

### **2.10 Testing Scaling**

Simulate load on the application to test the HPA:

kubectl run -i --tty --rm load-generator --image=busybox --restart=Never -- /bin/sh

# Inside the pod, run the following command to generate load

while true; do wget -q -O- http://nodejs-advanced-app-service; done

### **2.11 Validate Autoscaling Behavior**

Observe the HPA behavior:

kubectl get hpa

Watch the scaling events and verify that the application scales up and down based on the policies you configured.

## **3. Project Wrap-Up**

### **3.1 Review and Clean Up**

After completing the project, review the configurations and clean up the Minikube environment if needed:

minikube delete