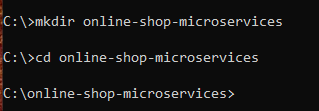
**MICROSERVICE E-COMMERCE WEBSITE K8S USING MINIKUBE.**

Create dir on cmd terminal called “**online-shop-microservices**” and navigate to that dir.



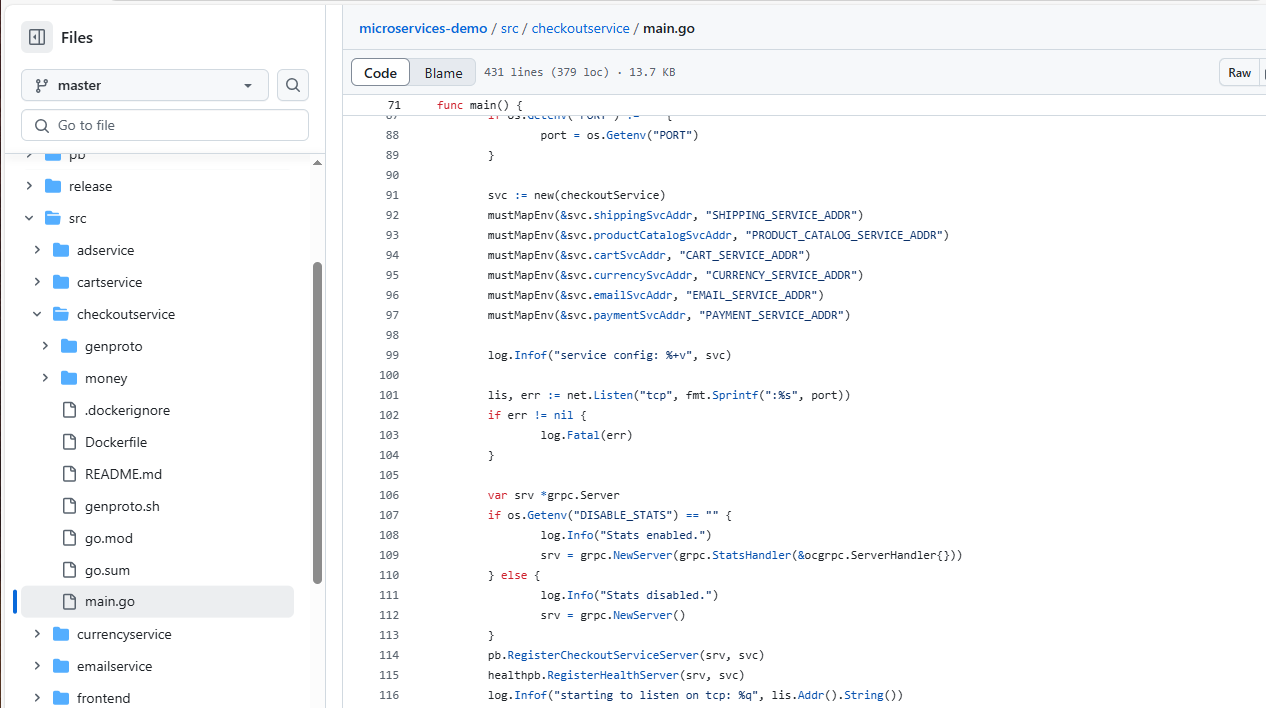
Create a YAML file where all microservice deployment and service configuration would be defined.

Open project in VISUAL STUDIO CODE by typing

The **checkoutservice** needs to know the;

Endpoint: **service Name + service port** for each microservice it connects to.

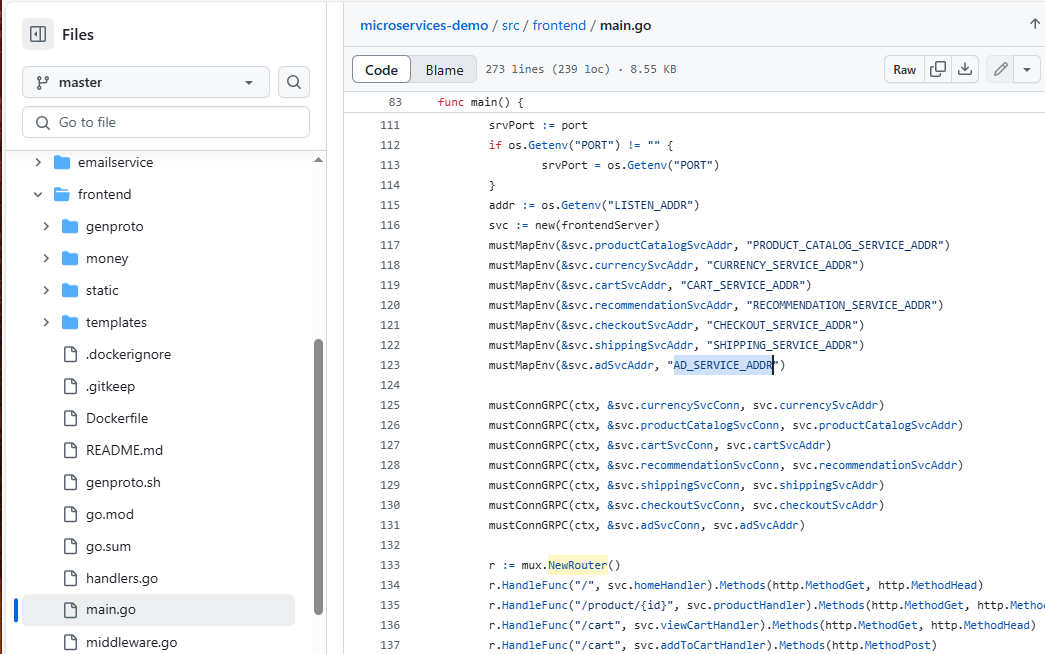
List of **environmental** **variables** that the **checkoutservice** talks to



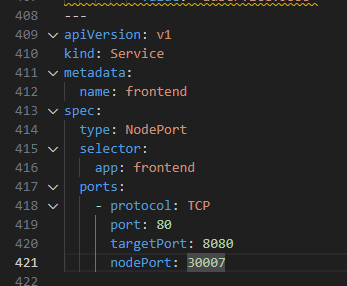
The **frontend** service needs to know the;

Endpoint: **service Name + service port** for each microservice it connects to.

List of **environmental** **variables** that the **frontend** talks to

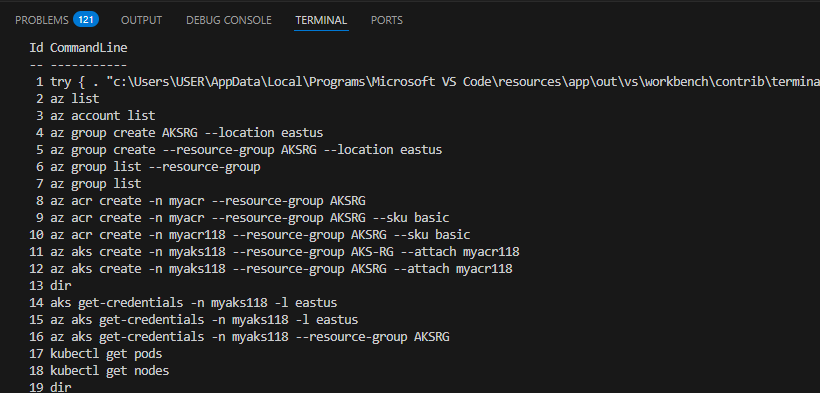


Since the **frontend** service needs to be accessible from the browser, we create a NodePort or LoadBalancer Ip for it.



**Deploy Microservice into K8s cluster**

Use the AZ CLI to create a RG, create ACR registry and then AKS cluster.

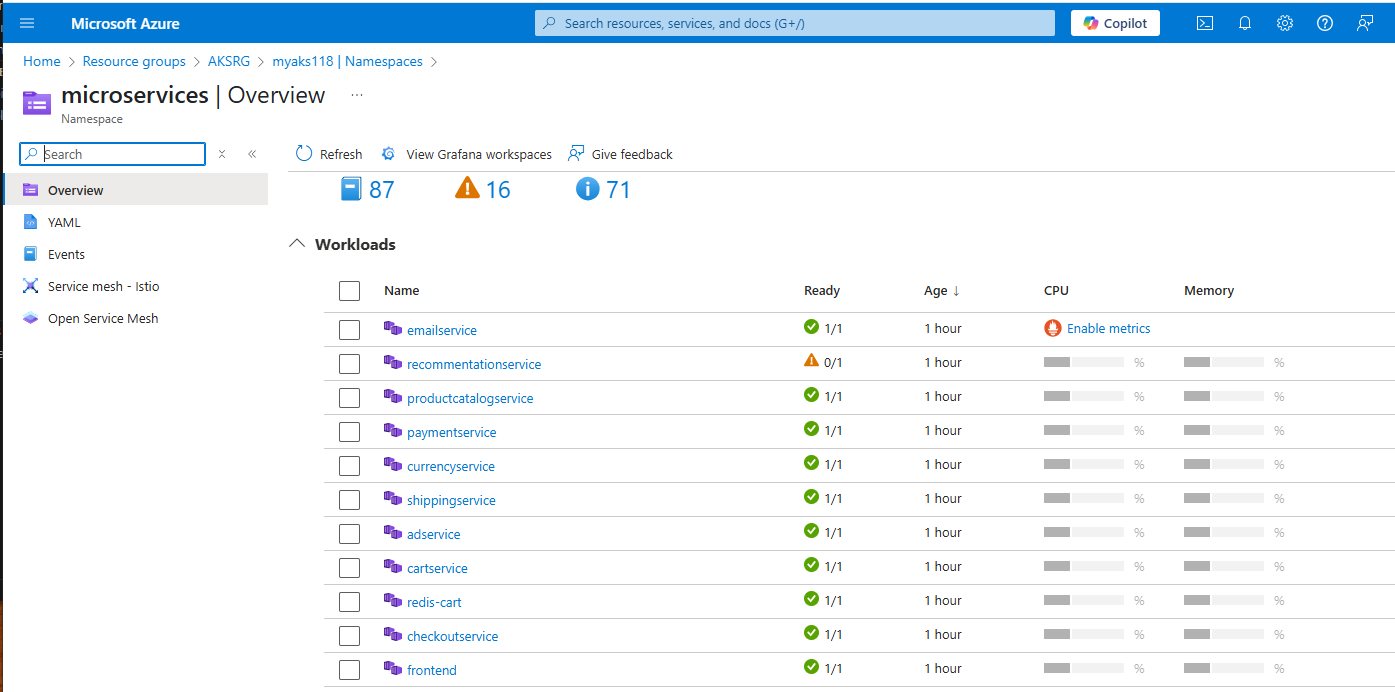


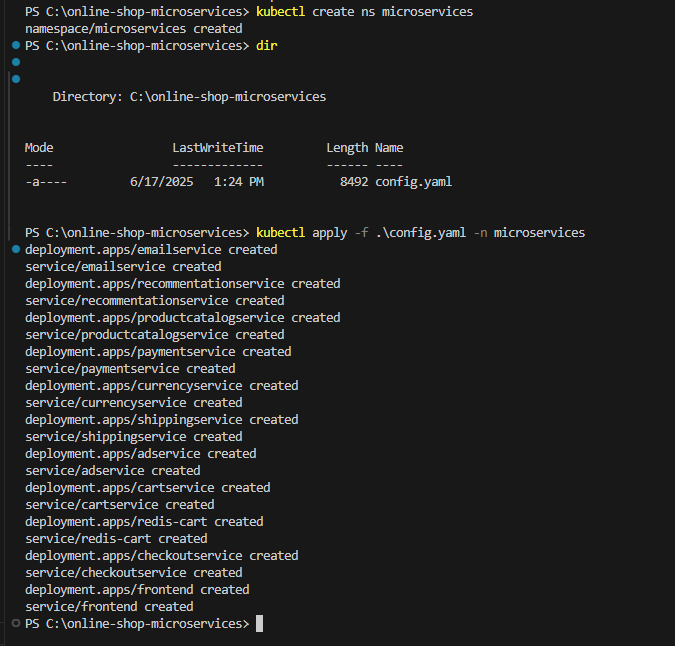
(history of commands. Some are wrong but the error free ones lye within).

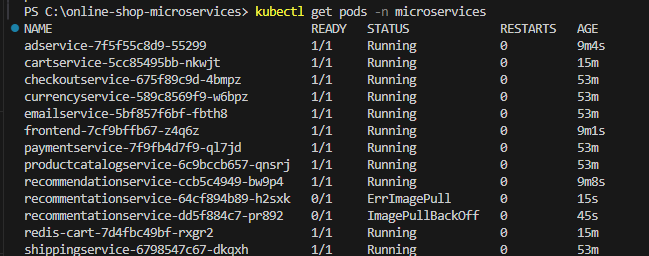
Create a namespace.

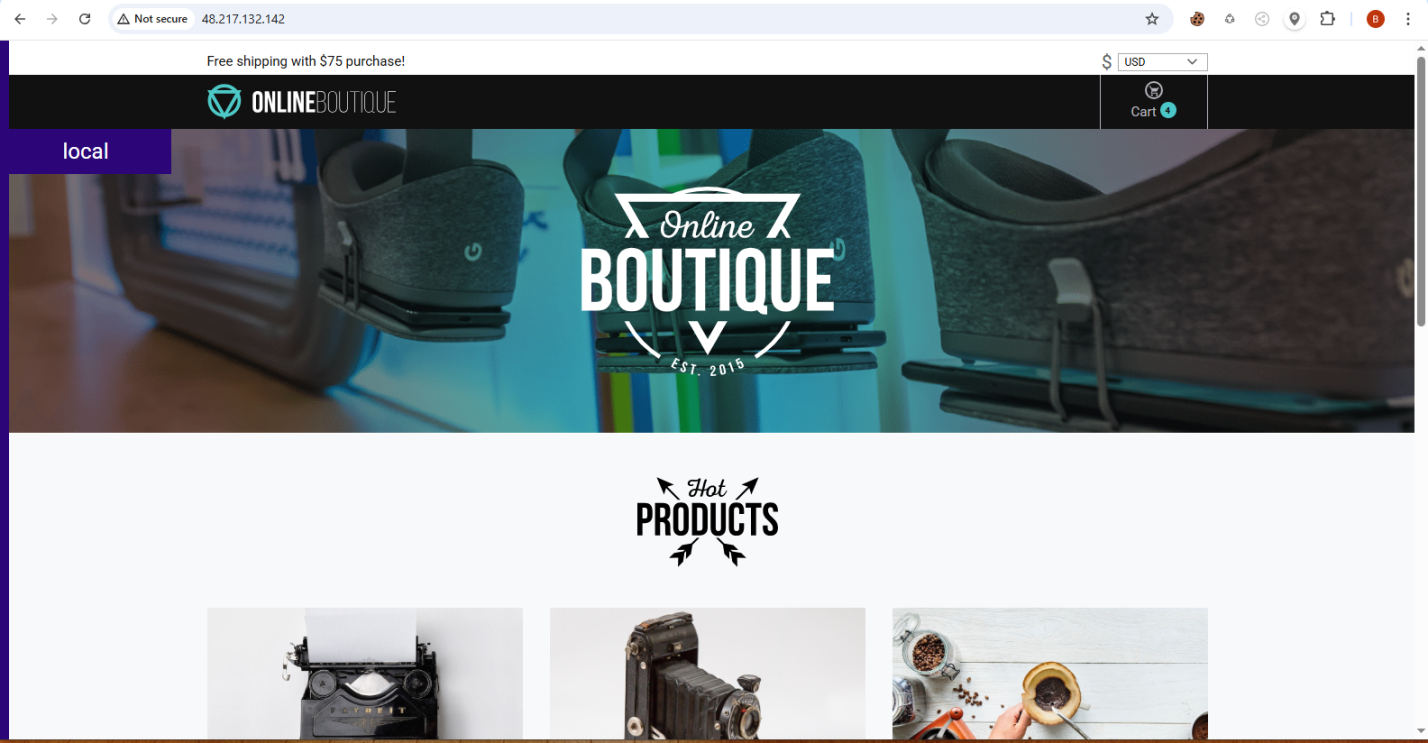


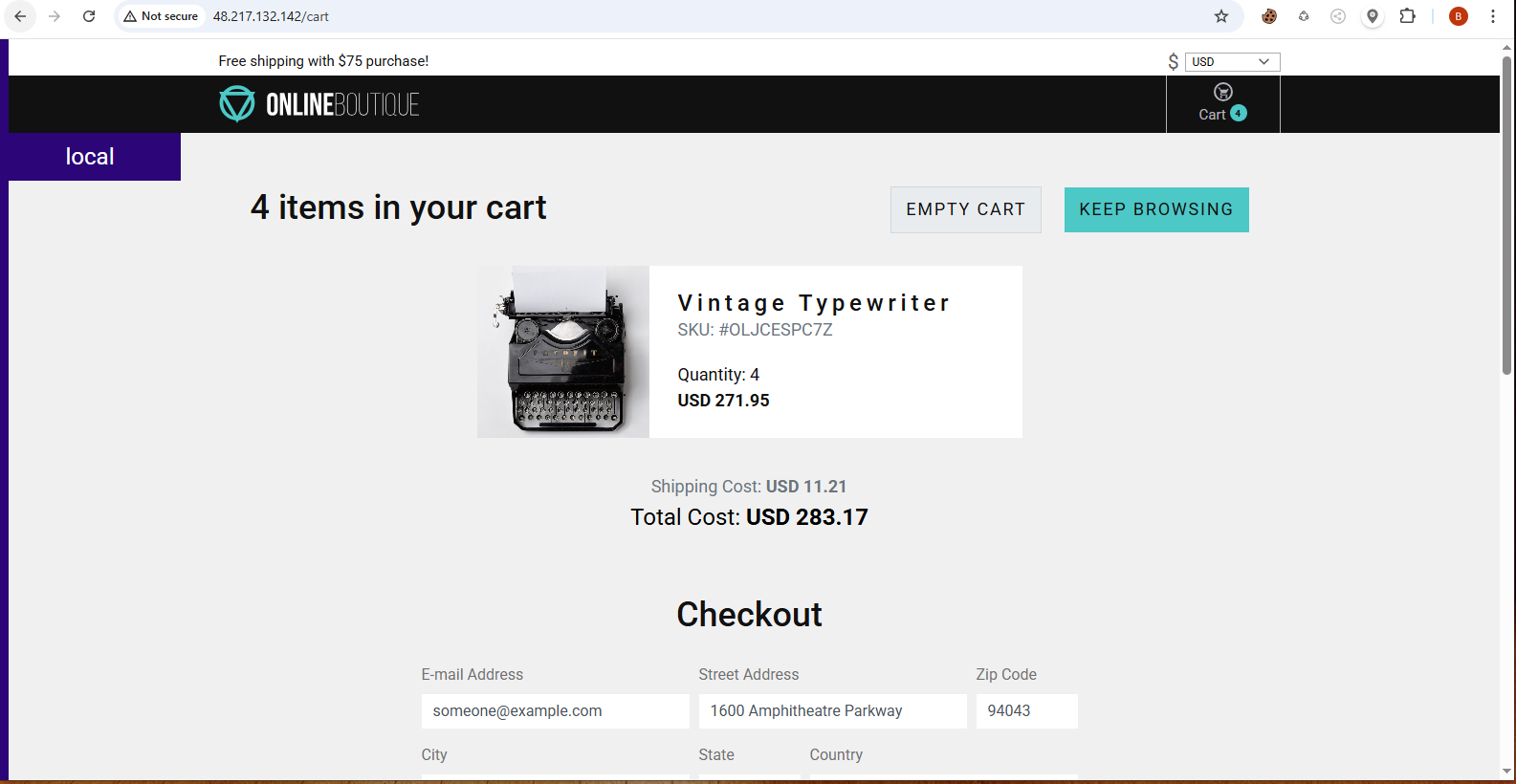
Apply command to execute the config.yaml file which would create the (deployment and service ) for all 11 microservices.

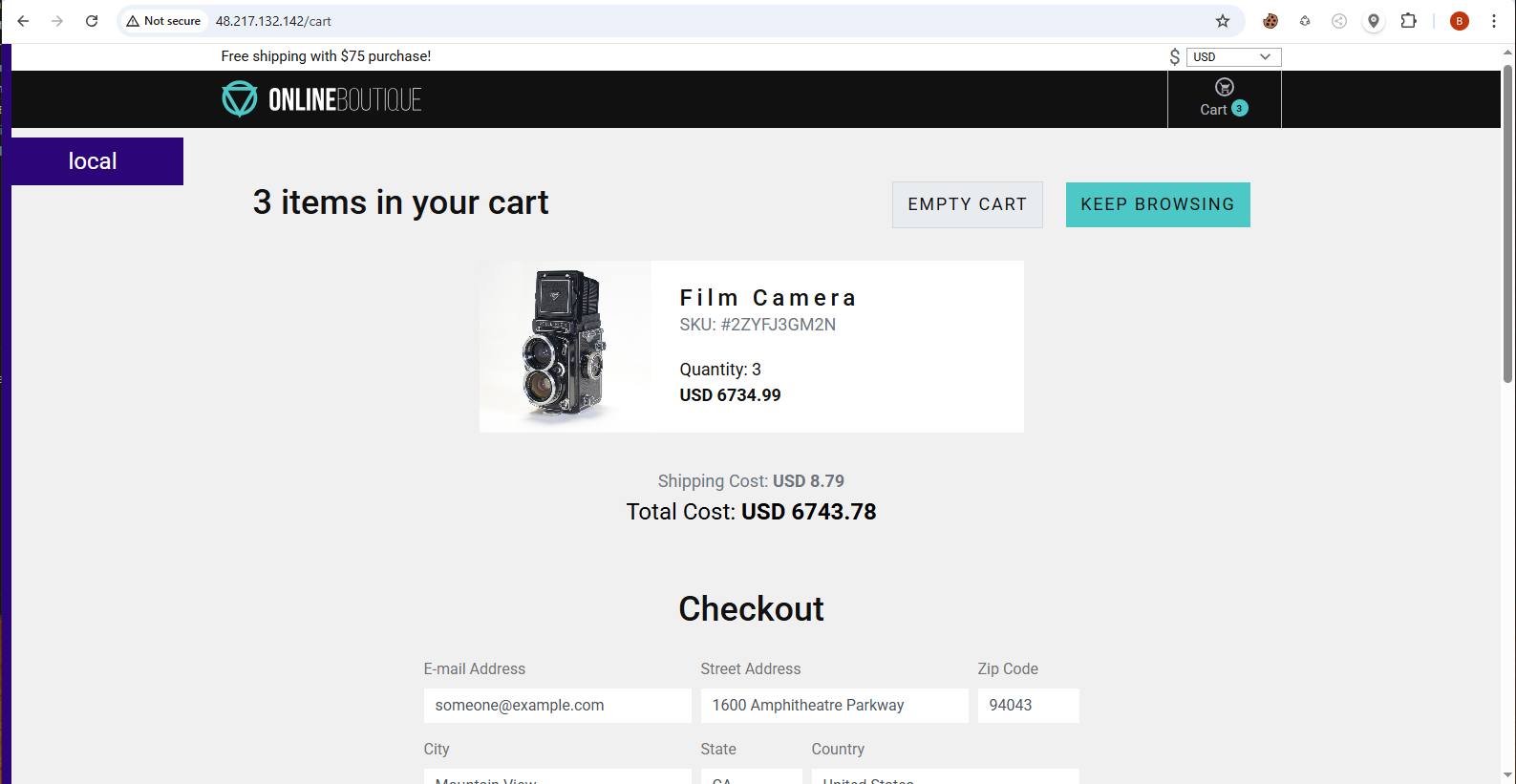




List of services and pods

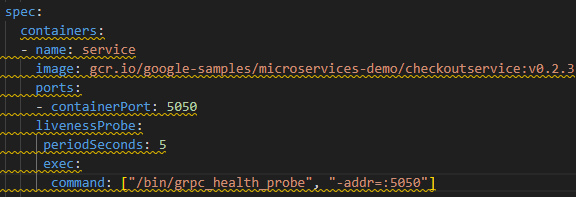






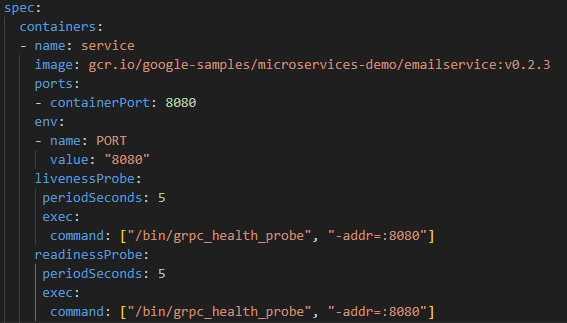
**PRODUCTION AND SECURITY BEST PRACTICES**

1. Pinned (tag) version for each container image.
2. Liveness probe for each container: it is actually a script or liveness program that pings the application endpoint every 05-10 seconds to see if the application responds/is running

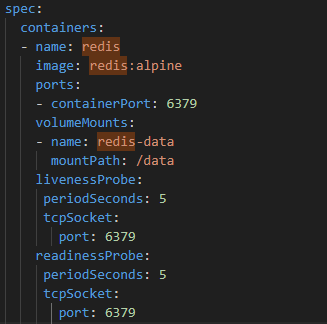


1. Readiness probe for each container: it lets the application know that kubernetes is ready to receive traffic. Without **Readiness Probe,**  K8S assumes the app is ready to receive traffic as the container starts. READINESS PROBE config is similar to that of LIVELINESS PROBE. Both check applications availability.

|  |  |
| --- | --- |
| **READINESS PROBE** | During application startup |
| **LIVELINESS PROBE** | While the application is already running |

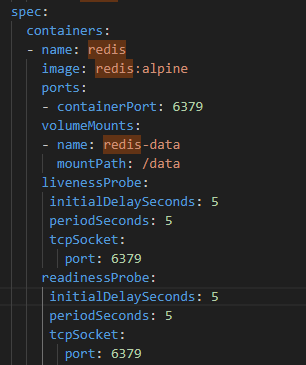


Liveliness and readiness probe for REDIS DB



The kubelet would try to reach the port which the application listens on (6379 above). If it fails, then the application is considered as unhealthy and not running.

If the application usually takes longer to start, you can add a property to delay for a specified period of time before the READINESS PROBE & LIVELINESS PROBE start working. (**initlalDelaySeconds**)



Application health can also be checked on HTTP endpoints.

**Best practice 3: Resource request for each container**

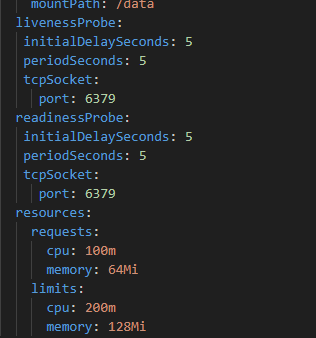
-keep CPU request at “1” or below. CPU resources are defined in **millicores.** Memory resources are defined in **bytes.**

**m = “millicore”**

**Mi= “mebibyte”**

**1000m = 1 CPU (or 1 full vCPU/core)**

**Best practice 4: Resource limits for each container**



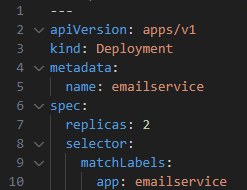
Note: For the **limits,** if you put values larger than your biggest node, your pod will never be scheduled!

**Best practice 5: DON’T expose a NodePort**

NodePort is less secured as it exposed nodeports to your application. It is more secured to use a **LoadBalancer** service type or **ingress** which is going to provide a single point of entry into your cluster.

**Best practice 6: More than 1 Replica for Deployment**

It’s advisable to always have more than 1 replica because assuming you have only 1 pod running, if that pod crashed, your application would become inaccessible/downtime until that pod gets restarted/self-healing. Having 2 or more active replicas increases the availability of your application.



**Best practice 7: More than 1 worker Node in your cluster**

**Why it’s important?**

* **Single Point of Failure**  with just 1 Node
* **You n**eed to **replicate all nodes as well**
* each replica should run on a different node.

**Reasons for server unavailability**

* + server crashes
  + server reboots, because of some update
  + server maintenance
  + server broken

**Best practice 8: Use Labels for all resources**

Labels are **Key Value pairs**

**Best practice 9: Use Namespaces** to isolate resources

**Best practice: Security best practices to follow**

* **En**sure that images are free of vulnerabilities. Use tools or 3rd party tools to scan your images especially those that are pulled from public repositories like **dockerhub, google.cr.io** etc. Image scans can also be automated in pipelines.
* Make sure no container in your cluster has **ROOT ACCESS** capabilities. With root access, they have access to **host-level resources.** If a container is hacked, much more damage can be done. So, **configure containers to use unprivileged users.**
* Update K8S version **Node by Node. Y**ou would have to do this one node at a time to avoid application downtime

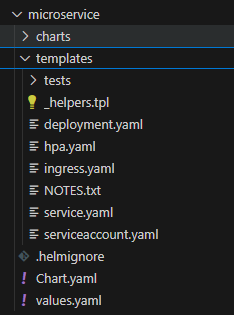
**855 845 7434**

**888 467 2080**

**HELM CHARTS AND MICROSERVICES**

Create a helm chart called **microservice**

|  |
| --- |
| **$ helm create microservice** |

****

**CONFIGURE THE DEPLOYMENT AND SERVICE TEMPLATE**

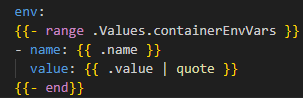
|  |
| --- |
| apiVersion: apps/v1  kind: Deployment  metadata:    name: {{ .Values.appName }}  spec:    replicas: {{ .Values.appReplicas }}    selector:      matchLabels:        app: {{ .Values.appName }}    template:      metadata:        labels:          app: {{ .Values.appName }}      spec:        containers:        - name: {{ .Values.appName }}          image: "{{ .Values.appImage }}:{{ .Values.appVersion }}"          ports:          - containerPort: {{ .Values.containerPort }} |

**SERVICE**

|  |
| --- |
| apiVersion: v1  kind: Service  metadata:    name: {{ .Values.appName }}  spec:    type: {{ .Values.serviceType }}    selector:      app: {{ .Values.appName }}    ports:      - protocol: TCP        port: {{ .Values.servicePort }}        targetPort: {{ .Values.containerPort }} |

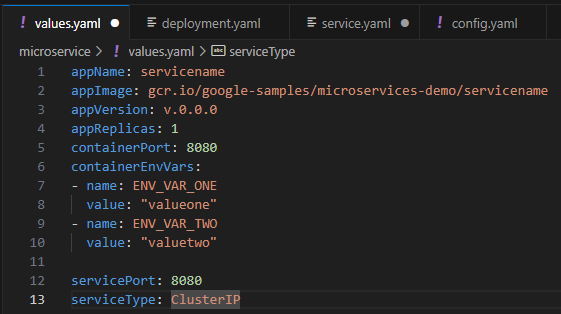
**DYNAMIC ENVIRONMENTAL VARIABLES**

Declare it in the form of a loop such that it would be able to iterate as many times as possible to take any given environmental variable that is being provided by the user.



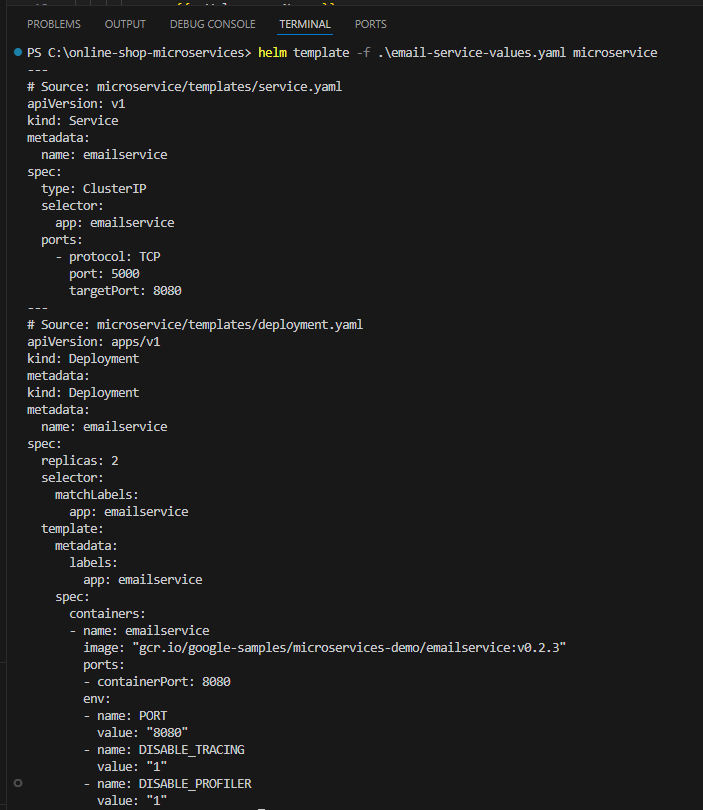
**SET THE VARIABLE VALUES**

Template for **values.yaml;** we now have to create custom values.yaml file for each microservice.



In order to check if we are deploying valid kubernetes manifest, we can actually test that locally using helm command

|  |
| --- |
| $ **helm template –f <custom-values.yaml> <chartname>** |



We can use the **--set command** to change the values of variables on the command line. E.g.

|  |
| --- |
| $ **helm template –f email-service-values.yaml --set appReplicas=3 microservice** |

From the command above, **appReplicas (variable)** has been set to 3.

**“helm lint” command**

* It examines a chart for possible issues.
* **ERROR**: issues that will cause the chart to fail installation
* **WARNING**: issues that break with convention or recommendations

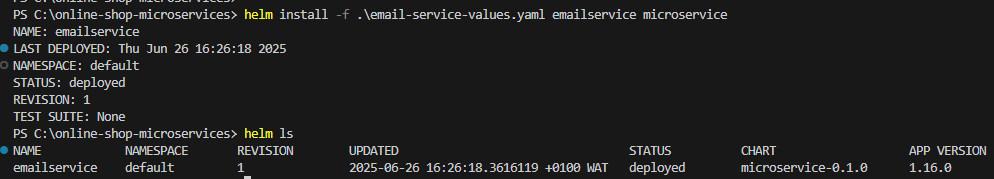
**e**.g.

|  |
| --- |
| $ **helm lint –f <yaml-file> <chart name>** |
| Example: **helm lint -f email-service-values.yaml microservice** |

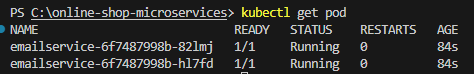
**DEPLOY THE EMAIL SERVICE**

The **email-service-values.yaml** file can be deployed using the **helm install** command. Its syntax goes thus;

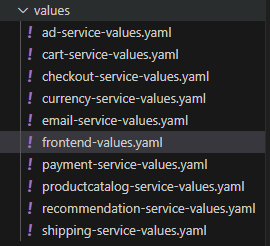
|  |
| --- |
| $ **helm install -f myValues.yaml <release-name> <chart-name>** |



2 pods are now running



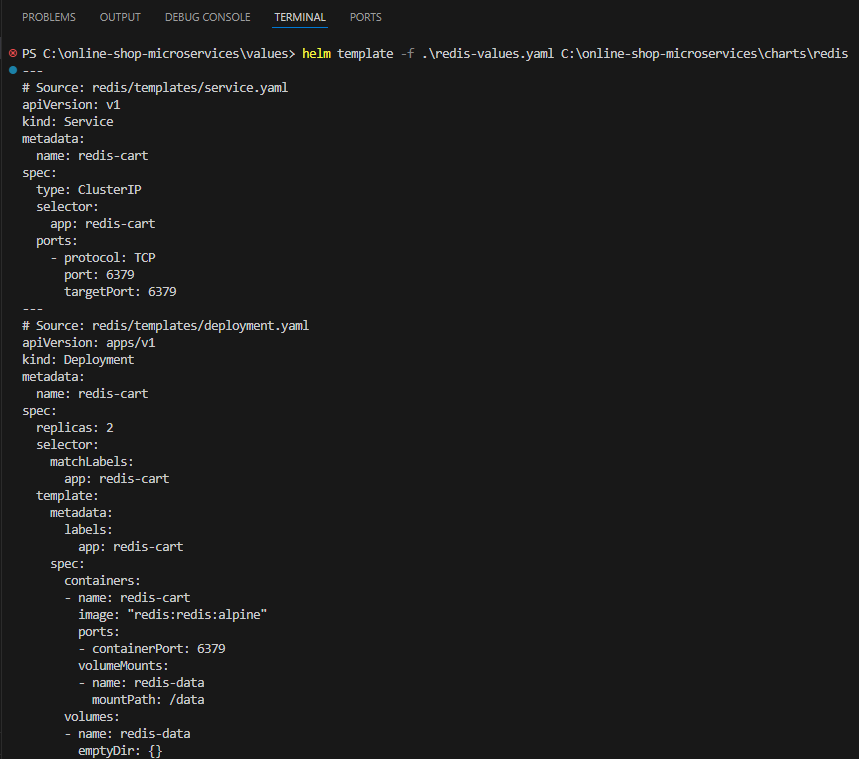
**CREATE VALUES FILE FOR ALL MICROSERVICES**



**CREATE REDIS HELM CHART**

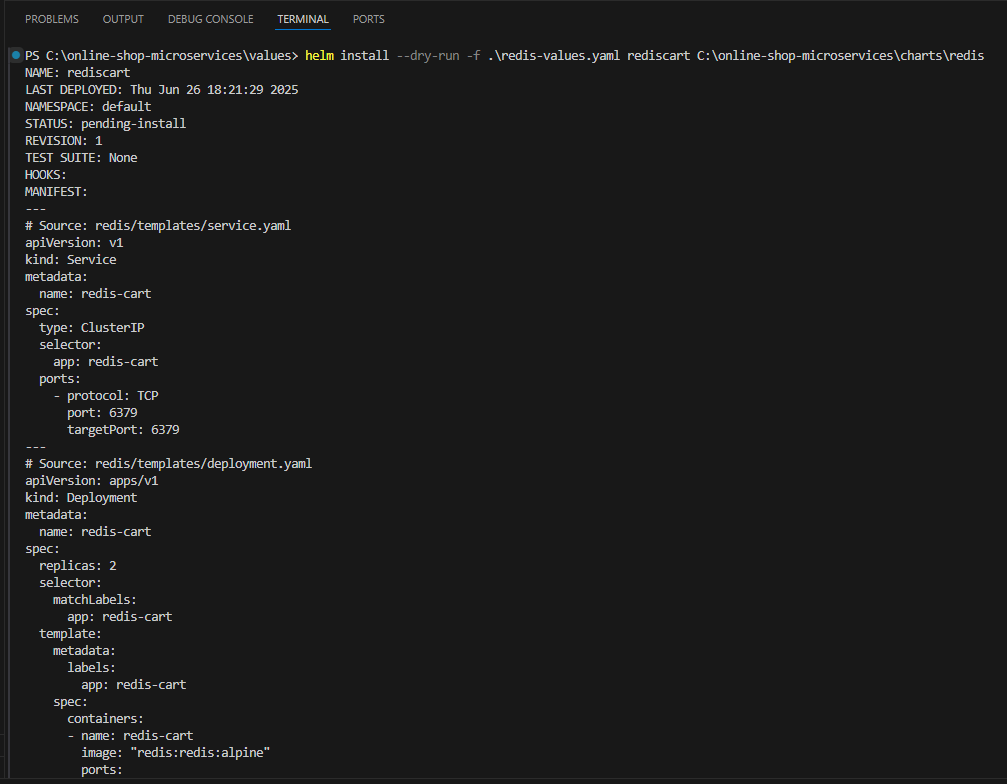
|  |
| --- |
| $ **helm create redis** |

Create its service and deployment yaml files and change the entries into a format that can accept variables



Another method of previewing that our manifest is valid, is by using

|  |
| --- |
| $ **helm install --dry-run** |
| The difference between **--dry-run & template** is that, --dry-run sends files to the K8S cluster, while template only validates it locally |
| $ **helm install --dry-run –f <your .yaml file> <release-name> <chart-name>** |
| EXAMPLE:  $ **helm install --dry-run -f .\redis-values.yaml rediscart C:\online-shop-microservices\charts\redis** |



The chart isn’t going to execute. It would just give us a plan of what is going to happen.

**DEPLOY THE MICROSERVICE WITH HELM INSTALL**

Since you have 11 microservices to install. Doing that 11 times is time consuming. So, write a shell script to automate that process.

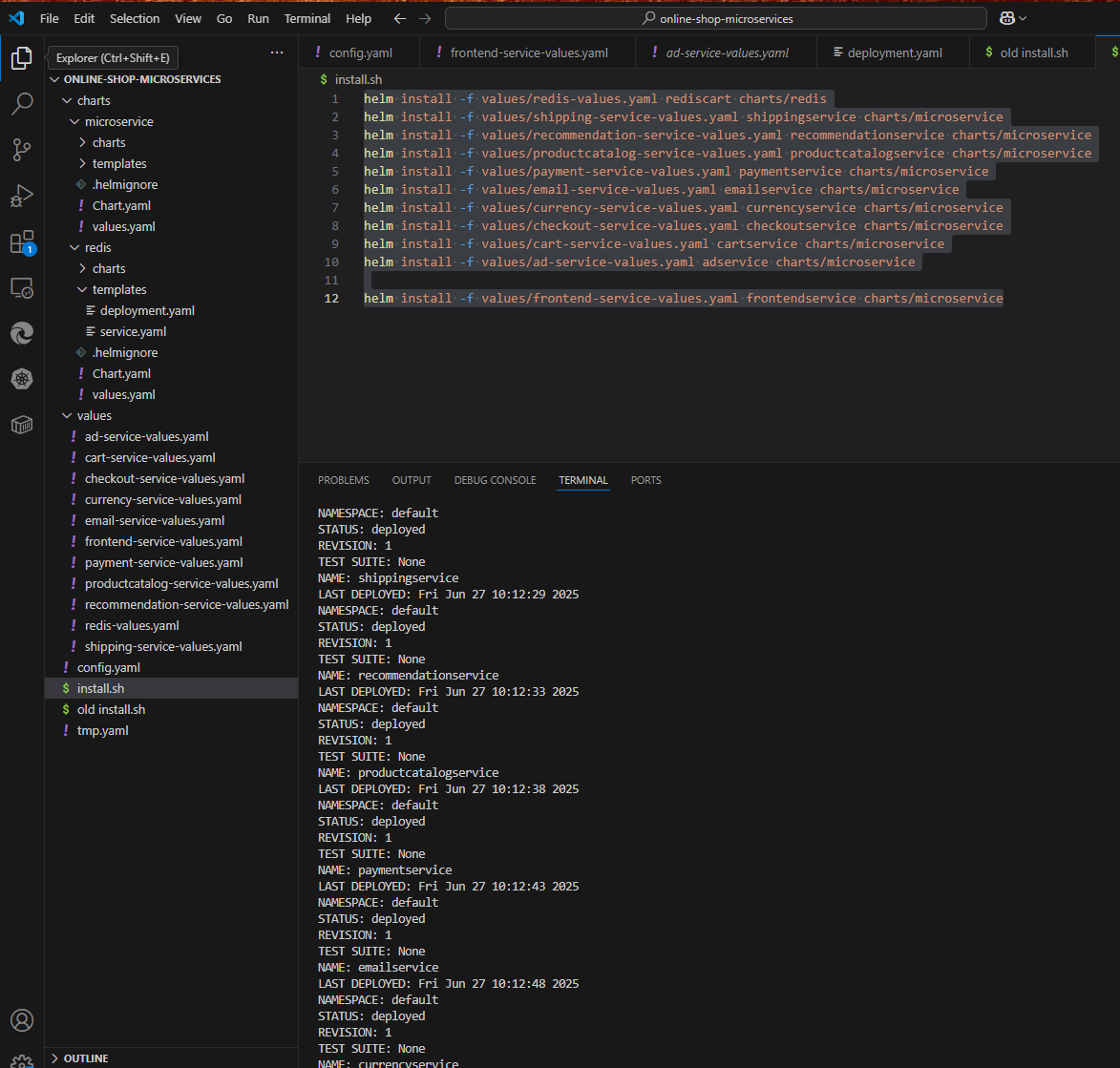
Create a file called **install.sh** & type the following

|  |
| --- |
| helm install -f values/redis-values.yaml rediscart charts/redis  helm install -f values/shipping-service-values.yaml shippingservice charts/microservice  helm install -f values/recommendation-service-values.yaml recommendationservice charts/microservice  helm install -f values/productcatalog-service-values.yaml productcatalogservice charts/microservice  helm install -f values/payment-service-values.yaml paymentservice charts/microservice  helm install -f values/email-service-values.yaml emailservice charts/microservice  helm install -f values/currency-service-values.yaml currencyservice charts/microservice  helm install -f values/checkout-service-values.yaml checkoutservice charts/microservice  helm install -f values/cart-service-values.yaml cartservice charts/microservice  helm install -f values/ad-service-values.yaml adservice charts/microservice  helm install -f values/frontend-service-values.yaml frontendservice charts/microservice |

-Change your VS studio terminal to **bash**

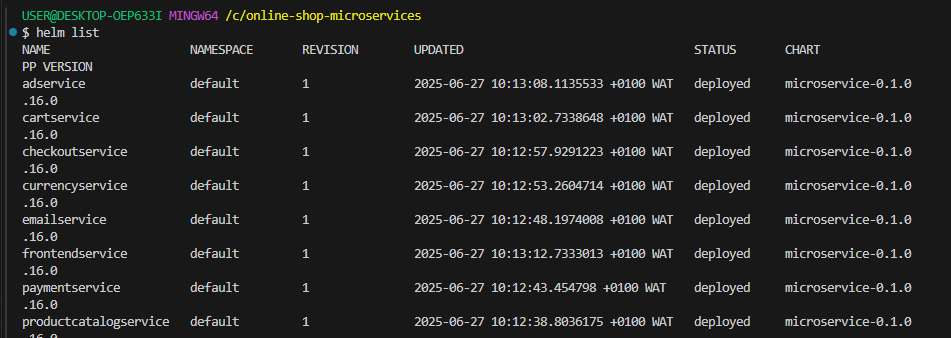
-Run the script in that directory:

|  |
| --- |
| **./install.sh** |



In order to see all the releases, type

|  |
| --- |
| $ **helm list** |

****

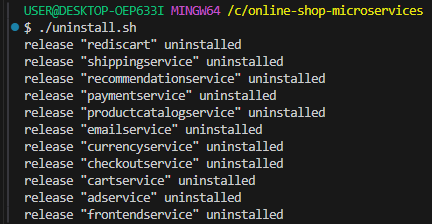
All can’t be deleted at once and redeployment cannot be created since it would conflict the existing release names. To delete all these releases at once, we use a **Helmfile**

**Helmfile.yaml** is a declarative way of deploying helm charts of an entire K8S cluster. Specifications can also be changed depending on the application or type of environment.

**UNINSTALLING HELM CHARTS**

Create a script to uninstall all microservices. Create a file called **uninstall.sh**

|  |
| --- |
| helm uninstall rediscart  helm uninstall shippingservice  helm uninstall recommendationservice  helm uninstall paymentservice  helm uninstall productcatalogservice  helm uninstall emailservice  helm uninstall currencyservice  helm uninstall checkoutservice  helm uninstall cartservice  helm uninstall adservice  helm uninstall frontendservice |



Verify that the releases have been deleted

|  |
| --- |
| $ **helm ls** |

**CREATE A HELM FILE**

Create a YAML file called **helm.yaml**

|  |
| --- |
| releases:    - name: redis-cart      chart: charts/redis      values:        - values/redis-values.yaml    - name: shippingservice      chart: charts/microservice      values:        - values/shipping-service-values.yaml    - name: recommendationservice      chart: charts/microservice      values:        - values/recommendation-service-values.yaml    - name: productcatalogservice      chart: charts/microservice      values:        - values/productcatalog-service-values.yaml    - name: paymentservice      chart: charts/microservice      values:        - values/payment-service-values.yaml    - name: emailservice      chart: charts/microservicev      values:        - values/email-service-values.yaml    - name: currencyservice      chart: charts/microservice      values:        - values/currency-service-values.yaml    - name: checkoutservice      chart: charts/microservice      values:        - values/checkout-service-values.yaml    - name: cartservice      chart: charts/microservice      values:        - values/cart-service-values.yaml    - name: adservice      chart: charts/microservice      values:        - values/ad-service-values.yaml    - name: frontendservice      chart: charts/microservice      values:        - values/frontend-service-values.yaml |

In this **helmfile.yaml**, we can actually override the value of variables in the values files with new custom values e.g.

|  |
| --- |
| releases:    - name: redis-cart      chart: charts/redis      values:        - values/redis-values.yaml        - appReplicas: "1"        - volumeName: "redis-cart-data" |

Here, I am able to override the values of **appReplicas & volumeName** in my defauld values.yaml file.

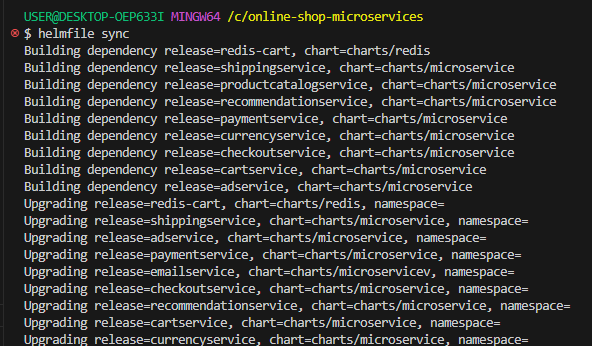
**INSTALLING HELM FILE**

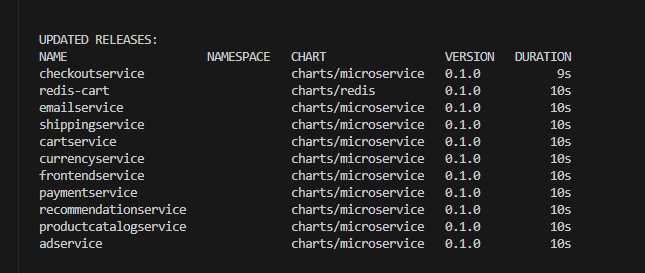
Since I have windows, I install **helmfile tool** using the windows package manager called **chocolatey (choco).**

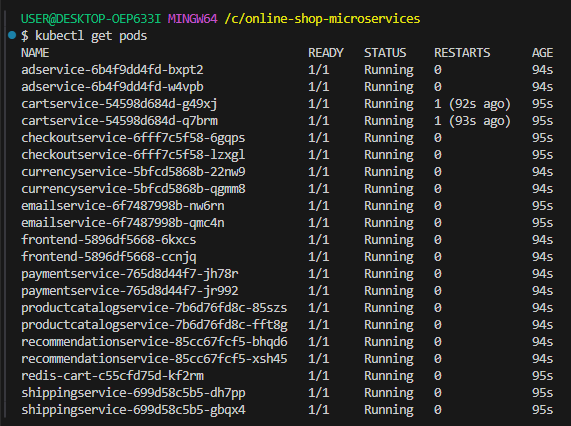
|  |
| --- |
| $ **choco install kubernetes-helmfile**  $ **helmfile --version** |

**DEPLOY HELM CHART –using helmfile**

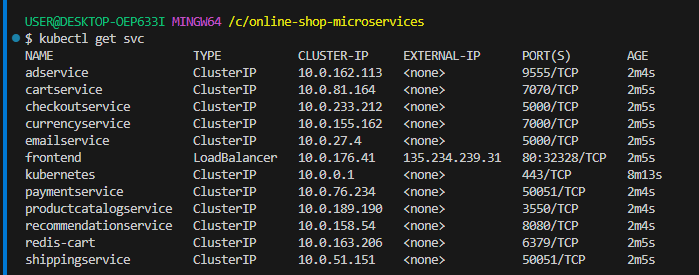
Firstly, run **helmfile sync** command. It ensures that the cluster matches the desired state described in the helmfile, installs or updrades all helm releases.

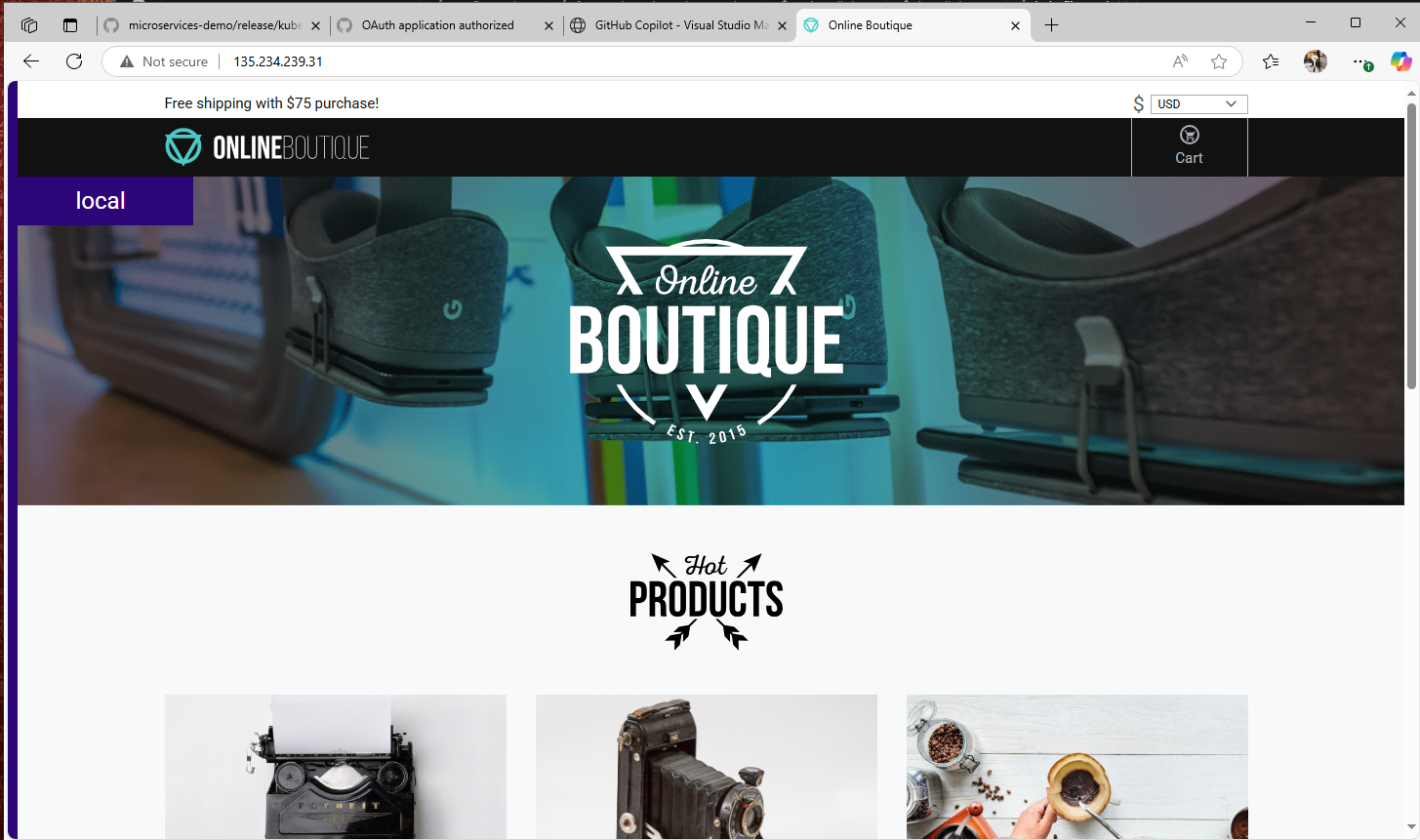






Check the available service and access the userinterface of the application from the **frontend external ip : 135.234.239.31**





**UNINSTALL ALL RELEASES –with a single command**

|  |
| --- |
| $ **helmfile destroy** |

