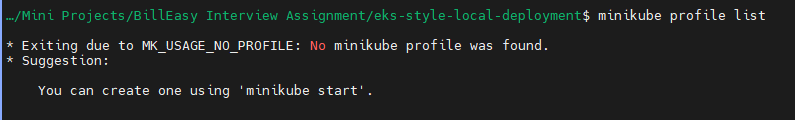
0. Installation and Prerequisites:

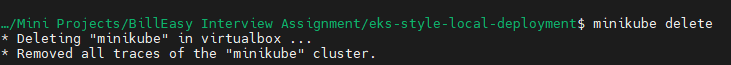
Check for existing minikube cluser:

* minikube profile list



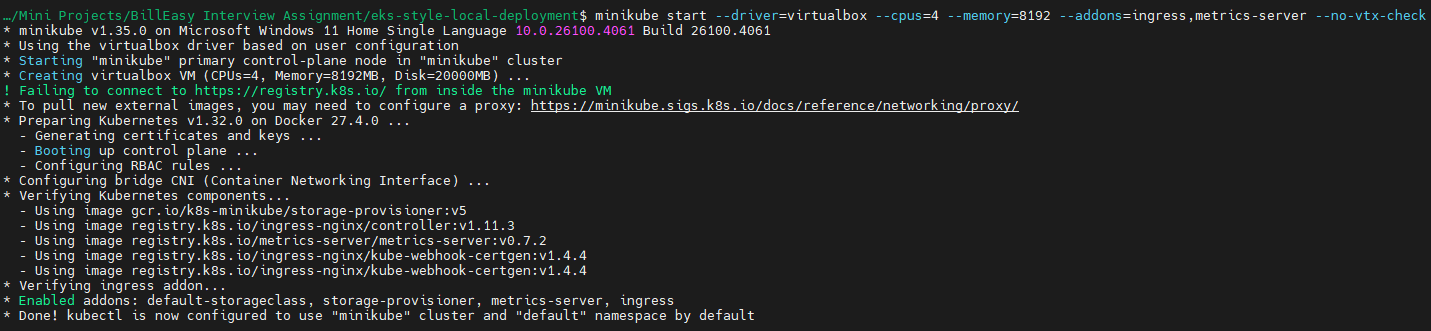
If exists, delete it

* minikube delete



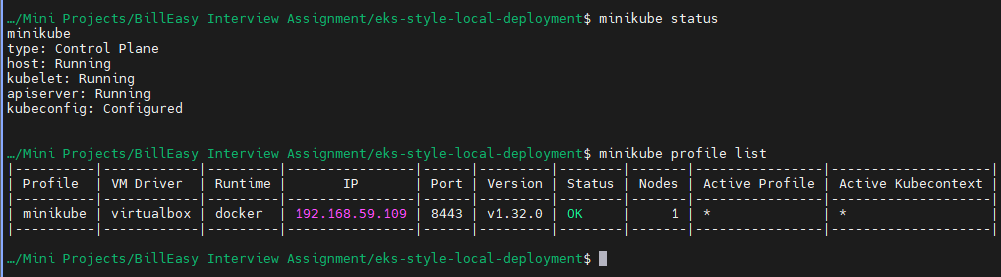
Start your minikube cluster:

* minikube start --driver=virtualbox --cpus=4 --memory=8192 --addons=ingress,metrics-server --no-vtx-check



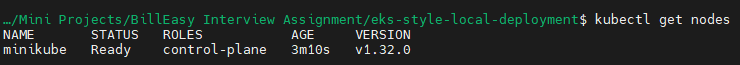
Check the status of minikube cluster:

* minikube status
* minikube profile list

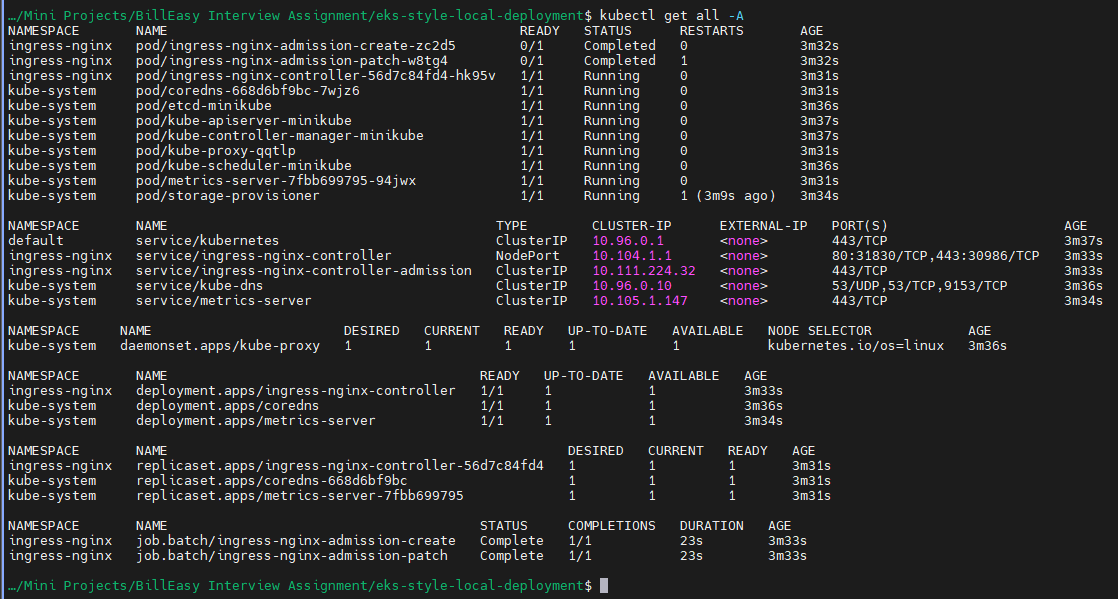


Ensure that your system is ready:

* kubectl get nodes

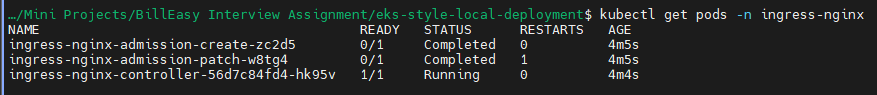


* kubectl get all -A



Verify your ingress:

* kubectl get pods -n ingress-nginx

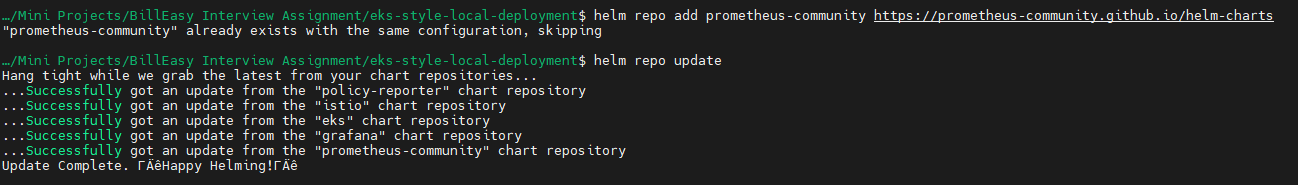


Now we setup Prom and Grafana in a new namespace monitoring by referring: <https://medium.com/@joudwawad/comprehensive-beginners-guide-to-kube-prometheus-in-kubernetes-monitoring-alerts-integration-4ade4fa8fa8c> :

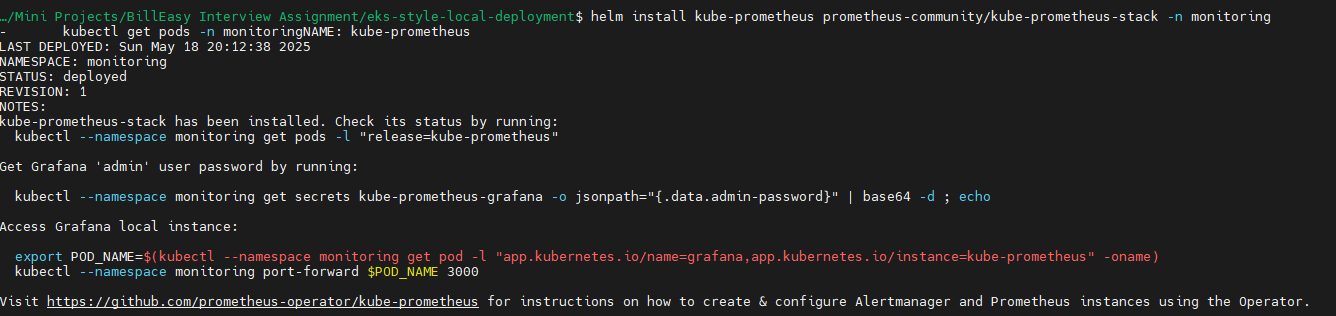
* kubectl create namespace monitoring



* helm repo add prometheus-community <https://prometheus-community.github.io/helm-charts>
* helm repo update

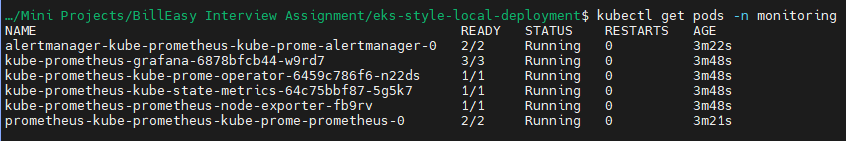


* helm install kube-prometheus prometheus-community/kube-prometheus-stack -n monitoring



Wait for the pods to be in running stage:

* kubectl get pods -n monitoring

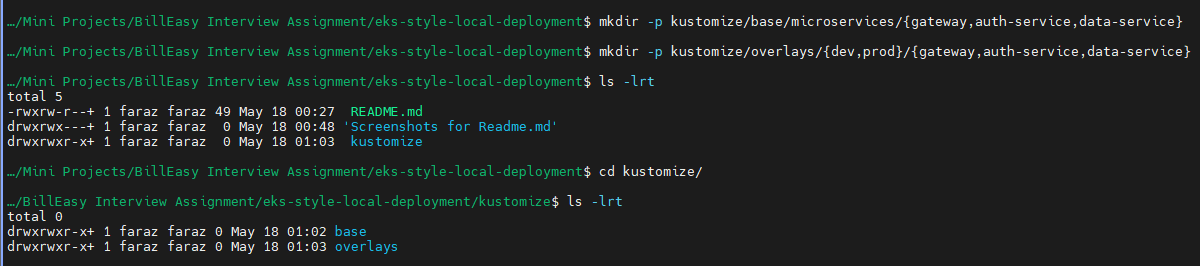


1. Microservice Stack

Refer: <https://subbaramireddyk.medium.com/kustomize-kubernetes-native-configuration-management-f51630d29ac0>

Creating kustomize project structure:

* mkdir -p kustomize/base/microservices/{gateway,auth-service,data-service}
* mkdir -p kustomize/overlays/{dev,prod}/{gateway,auth-service,data-service}

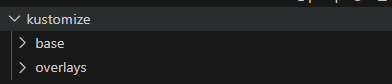


Create kustomization.yaml files:

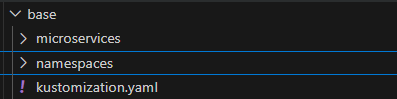


Create rest of the project structure

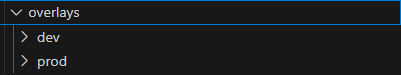
Final project structure:



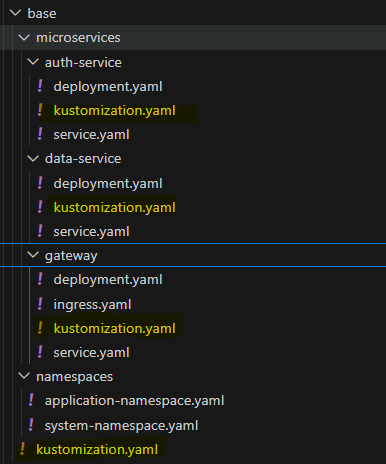
Base:



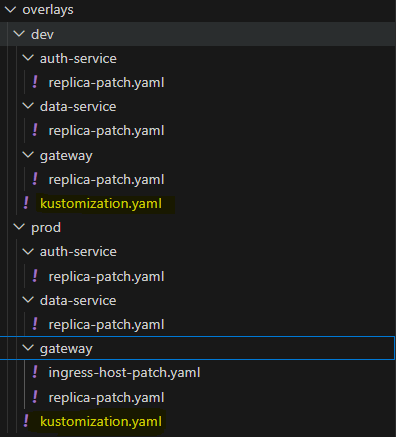
Overlays:



Base project folders:



Overlays project folders:

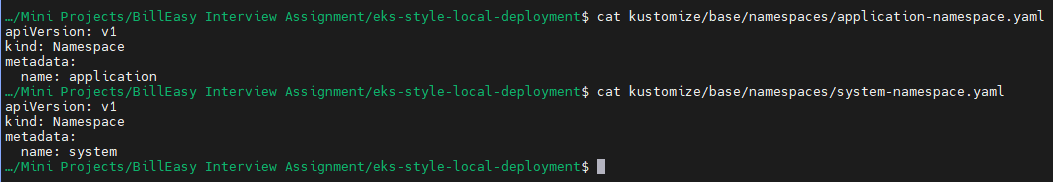


Create namespaces application and system:

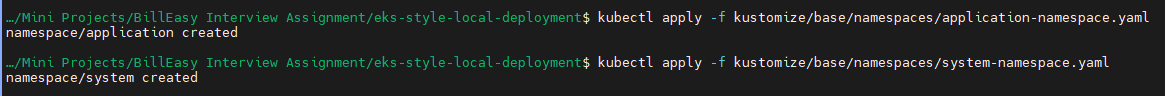
Check the existing namespaces:



Create yaml to create namespaces:



Apply to create namespace:



Verify your namespace creation:



\*\* Assuming that gateway will belong to the application namespace since it needs to be publicly accessible via Ingress, while auth-service and data-service should reside in the system namespace to keep them internal-only.

Now we will deploy the following services:

|  |  |  |
| --- | --- | --- |
| **Service** | **Purpose** | **Docker Image** |
| gateway | API gateway, exposed via ingress | nginxdemos/hello |
| auth-service | Auth logic, logs headers | kennethreitz/httpbin |
| data-service | Mock business logic | hashicorp/http-echo |

Create yaml manifests of gateway, auth-service and data-service with the mentioned requirements

and apply it

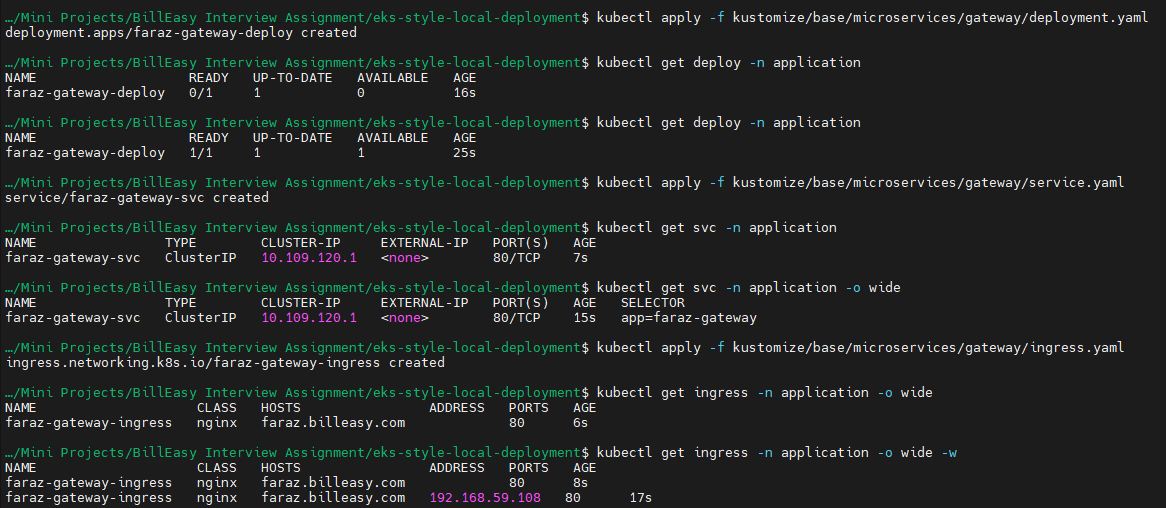
Requirements:

* Containerized deployment using Helm or Kustomize
* Liveness/readiness probes
* Proper resource requests/limits
* gateway should be publicly accessible via ingress
* auth-service and data-service should be internal-only
* Use separate namespaces for system and application

Applying and manually testing if every manifest created is working as expected:

Manually Testing Gateway manifests:

* + kubectl apply -f kustomize/base/microservices/gateway/deployment.yaml
  + kubectl get deploy -n application
  + kubectl apply -f kustomize/base/microservices/gateway/service.yaml
  + kubectl get svc -n application -o wide
  + kubectl apply -f kustomize/base/microservices/gateway/ingress.yaml
  + kubectl get ingress -n application -o wide

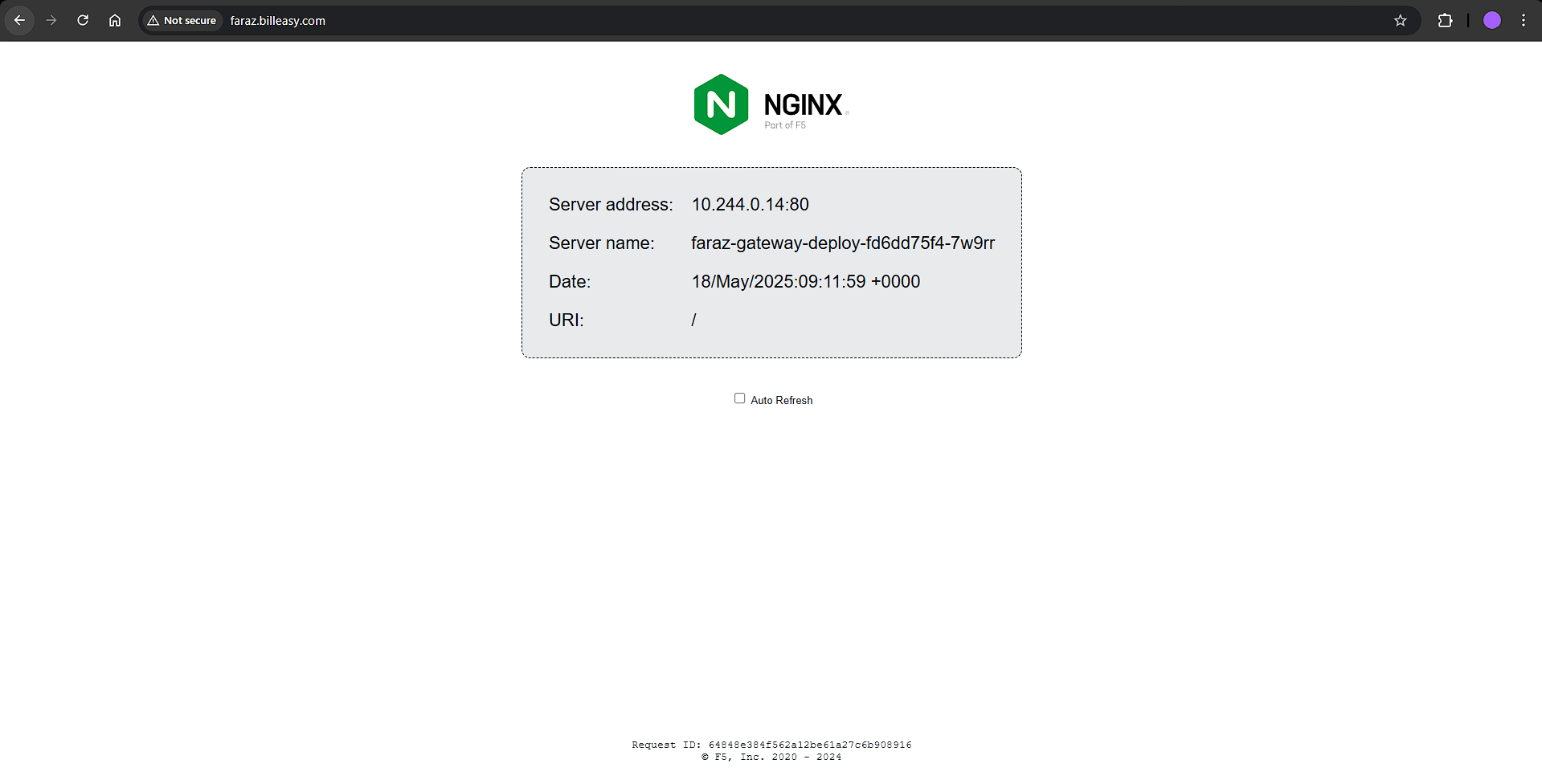


Also add the ingress ip to /etc/hosts

* sudo vim /etc/hosts

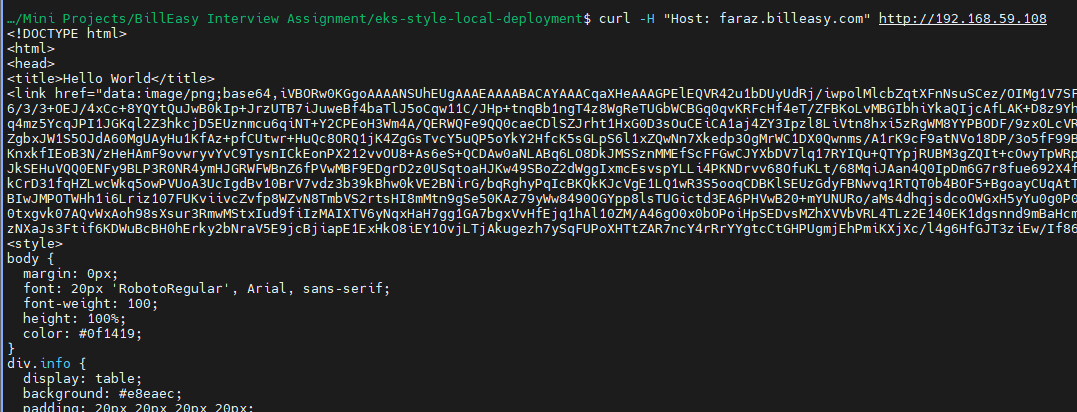


You must now be able to access the application from your browser:



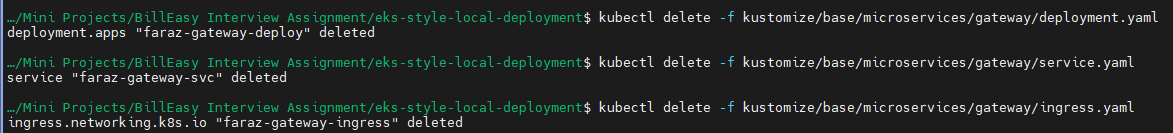
You can also test it using curl:

* curl -H "Host: faraz.billeasy.com" http://192.168.59.108



After testing, delete the created services:

* kubectl delete -f kustomize/base/microservices/gateway/deployment.yaml
* kubectl delete -f kustomize/base/microservices/gateway/service.yaml
* kubectl delete -f kustomize/base/microservices/gateway/ingress.yaml

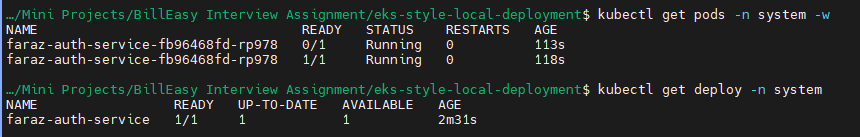


Testing auth-service:

Apply the deployment:

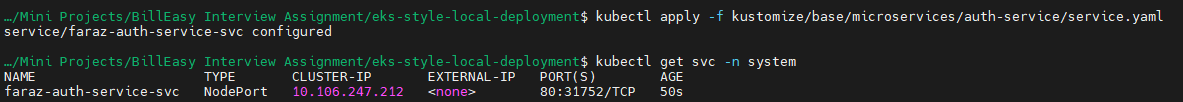
* kubectl apply -f kustomize/base/microservices/auth-service/deployment.yaml
* kubectl get pods -n system
* kubectl get deploy -n system





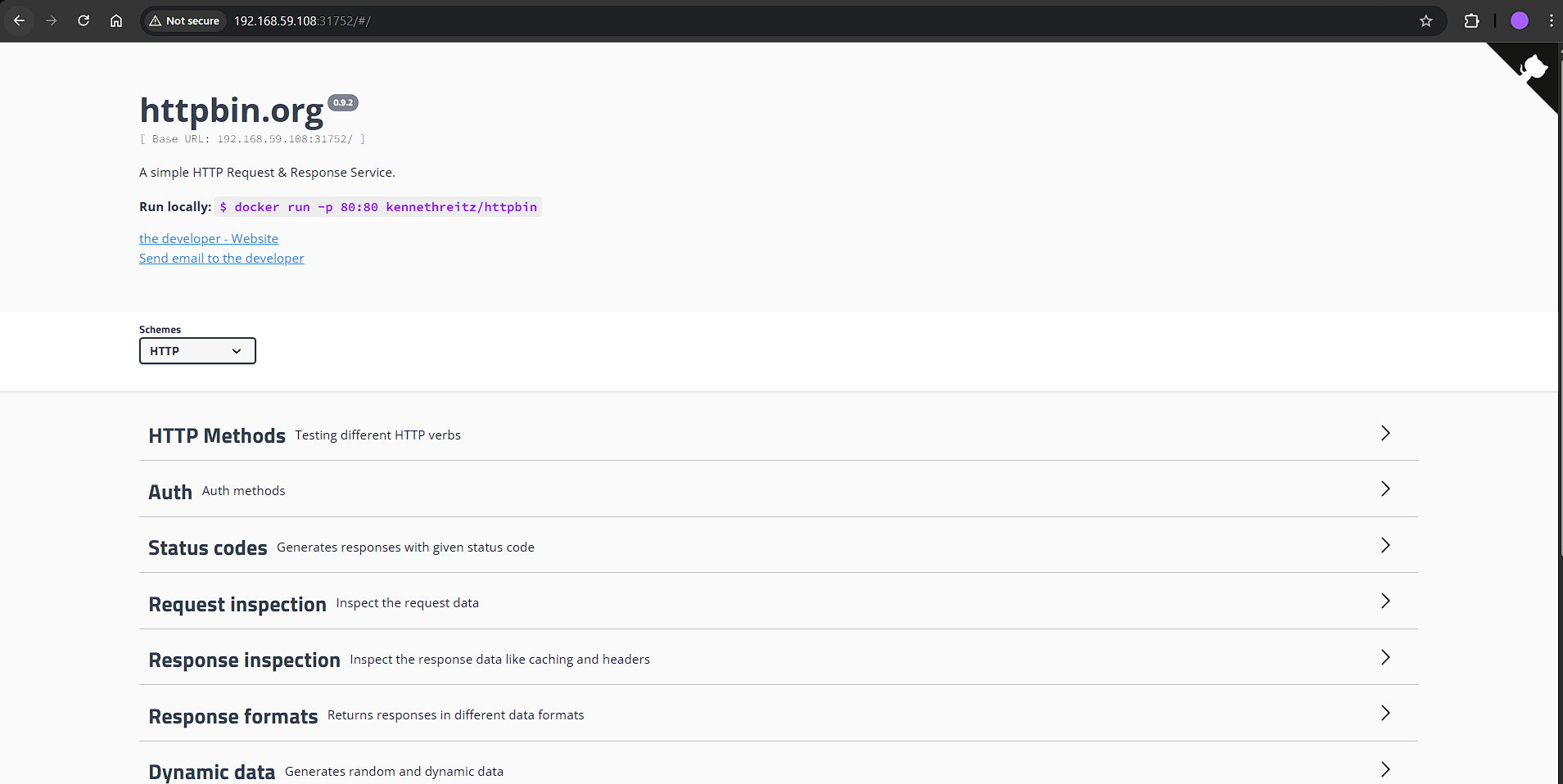
For testing we will apply the service in NodePort mode:

* kubectl apply -f kustomize/base/microservices/auth-service/service.yaml
* kubectl get svc -n system



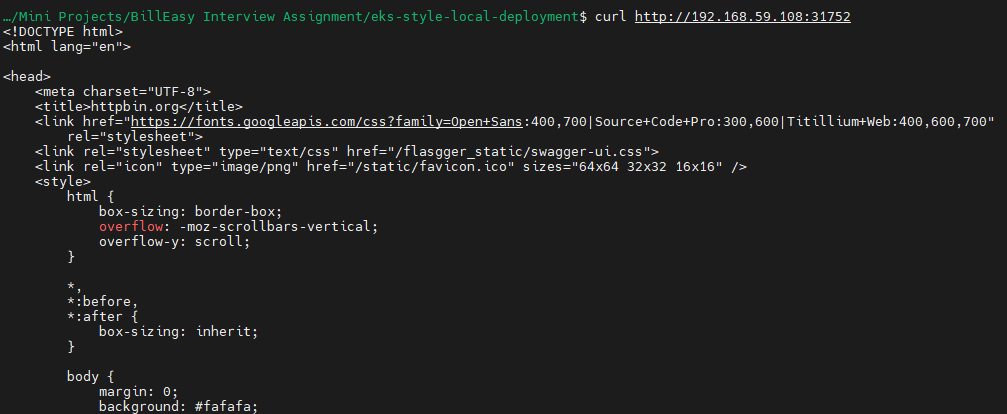
By getting the minikube IP and the Node port:

Verify if it is working:



You can also use curl:

* curl http://192.168.59.108:31752



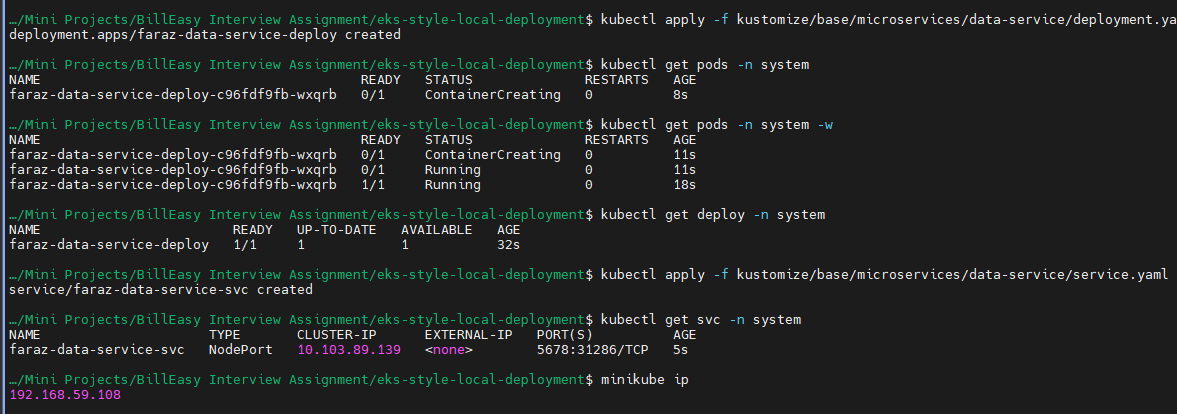
After testing deleting the deployment and service:

* kubectl delete -f kustomize/base/microservices/auth-service/service.yaml
* kubectl delete -f kustomize/base/microservices/auth-service/deployment.yaml



Similarly test data-service:

* kubectl apply -f kustomize/base/microservices/data-service/deployment.yaml
* kubectl get pods -n system
* kubectl get deploy -n system
* kubectl apply -f kustomize/base/microservices/data-service/service.yaml
* kubectl get svc -n system

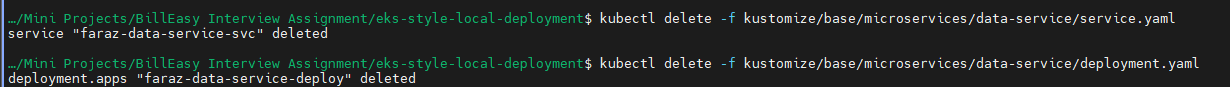


Open it in your browser:



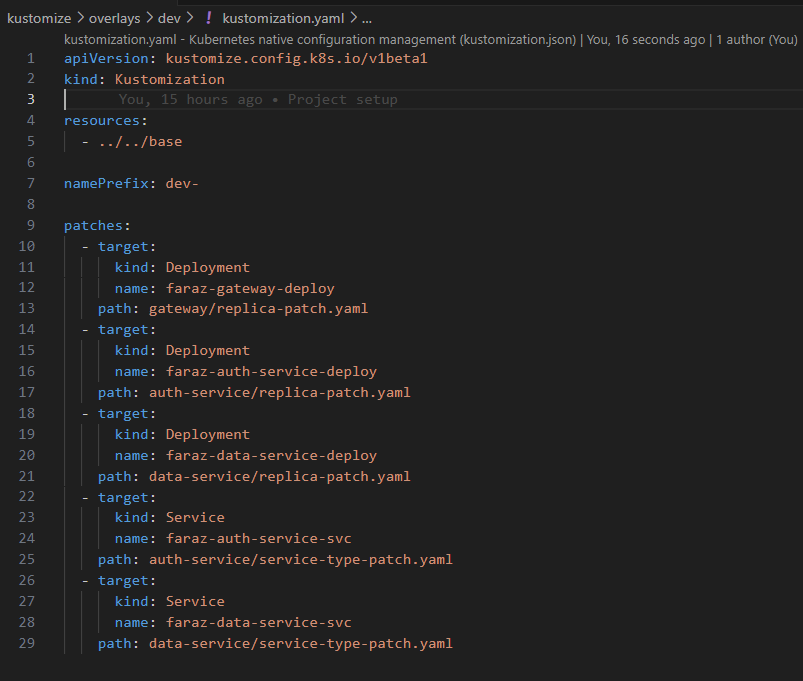
Delete the service and deployment:

* kubectl delete -f kustomize/base/microservices/data-service/service.yaml
* kubectl delete -f kustomize/base/microservices/data-service/deployment.yaml



Now we will use kustomize to test:

Create your kustomization.yaml and create your patches:

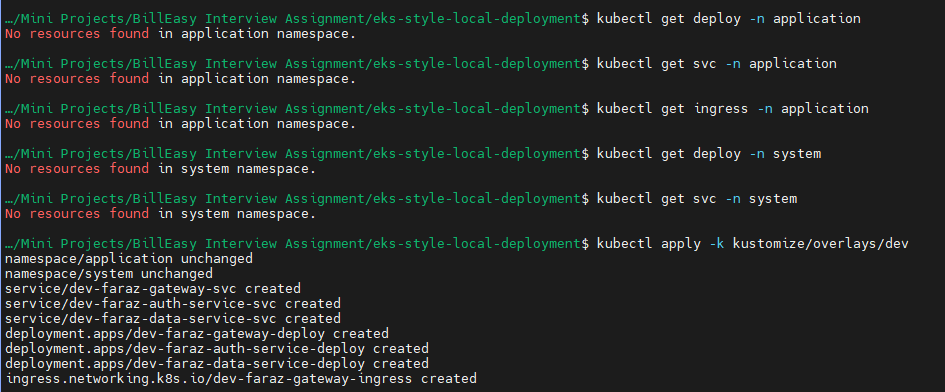


Verify your kustomize using:

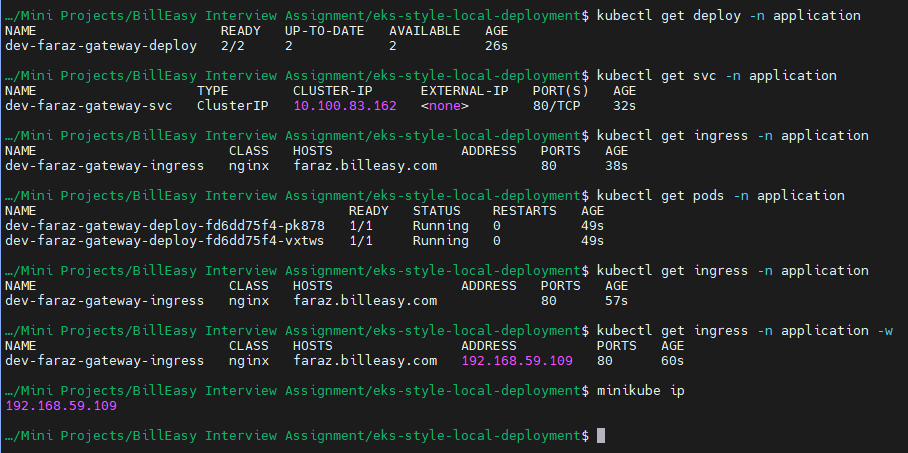
* kubectl kustomize kustomize/overlays/dev/
* kubectl kustomize kustomize/overlays/prod/

Now apply the kustomization:

* kubectl apply -k kustomize/overlays/dev

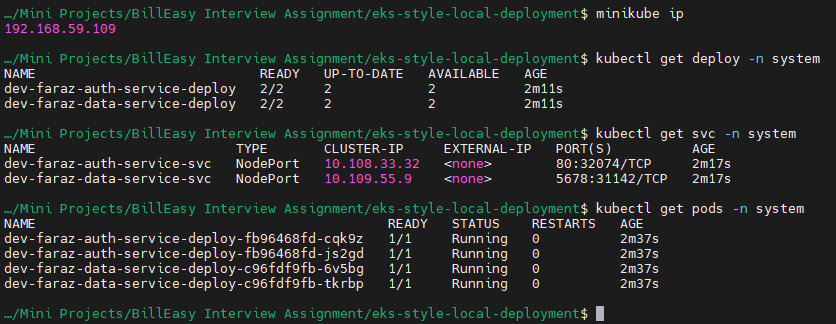


Verify your service, deployment and ingress(only for gateway):

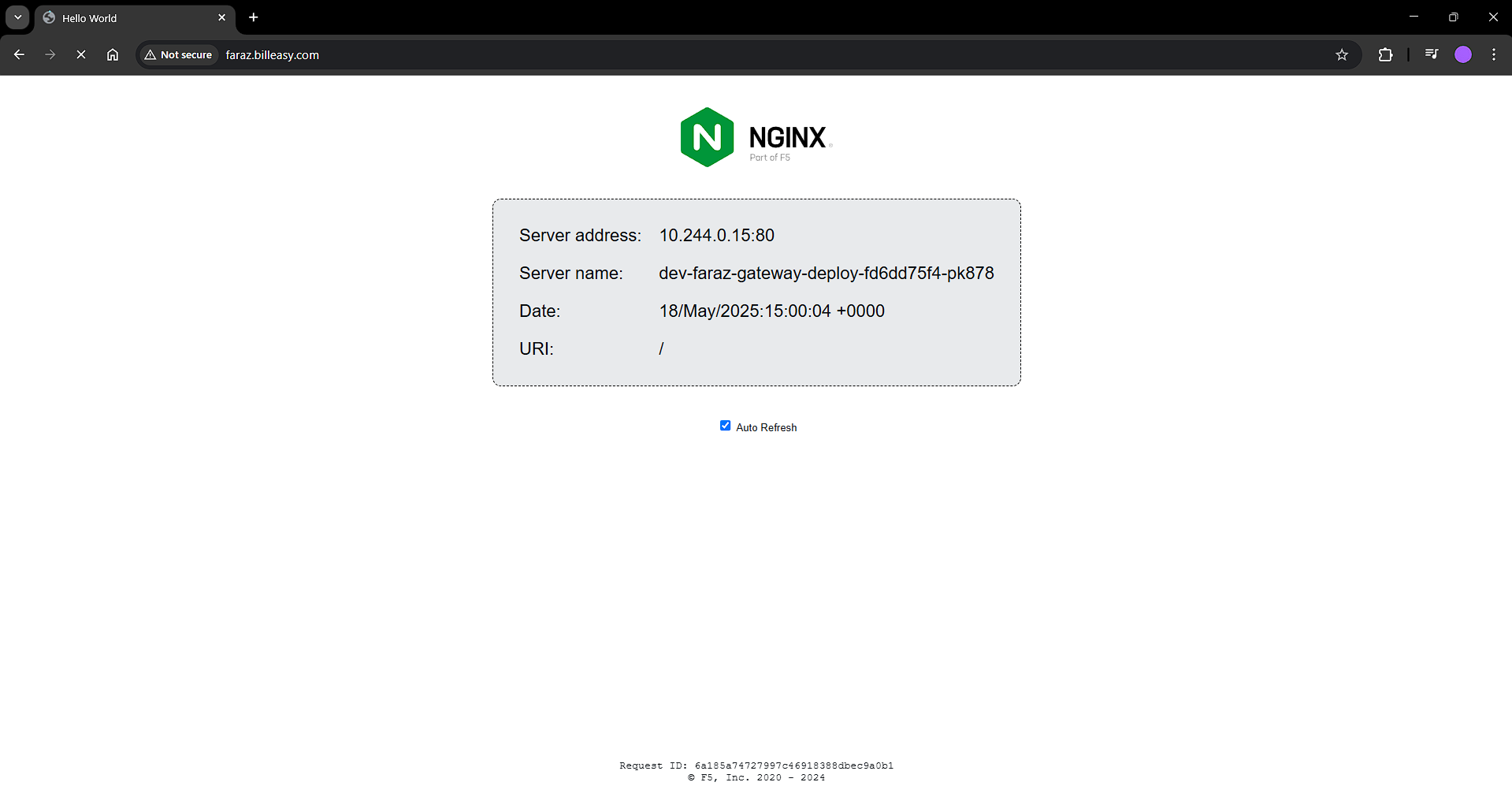


Ensure that you’re mentioning the IP in /etc/hosts

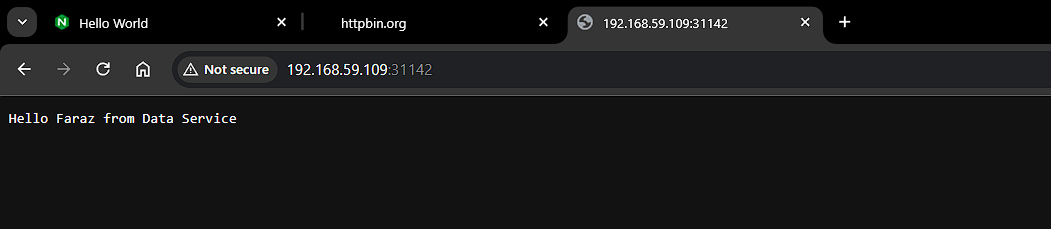
* sudo vim /etc/hosts
* 



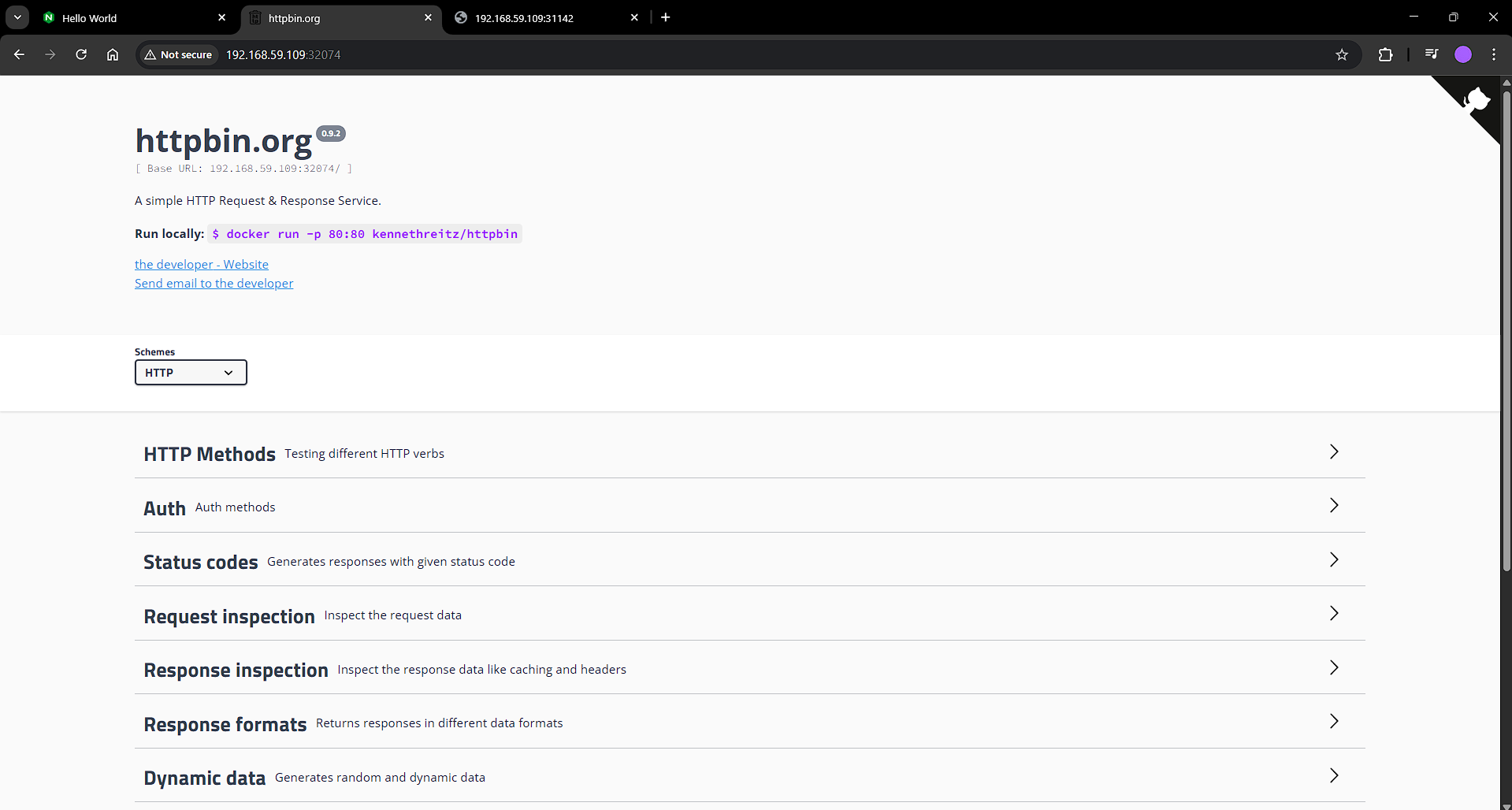
Verify your gateway ingress, svc, deploy on browser:



Verify your data service:

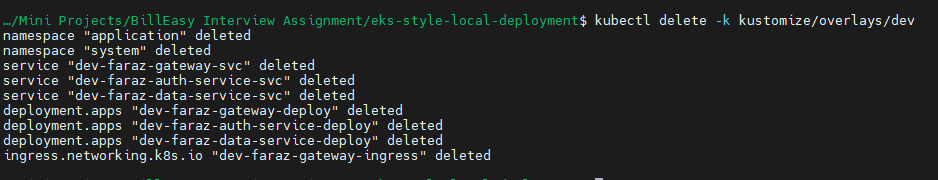


Verify your auth-service:



Deleting the resources:

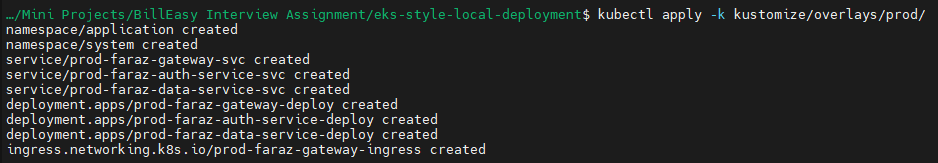
* kubectl delete -k kustomize/overlays/dev



Now similarly, we will verify in prod:

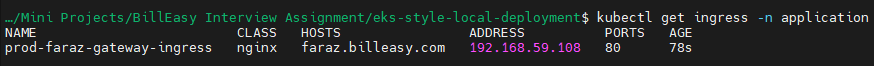
Apply the kustomization:

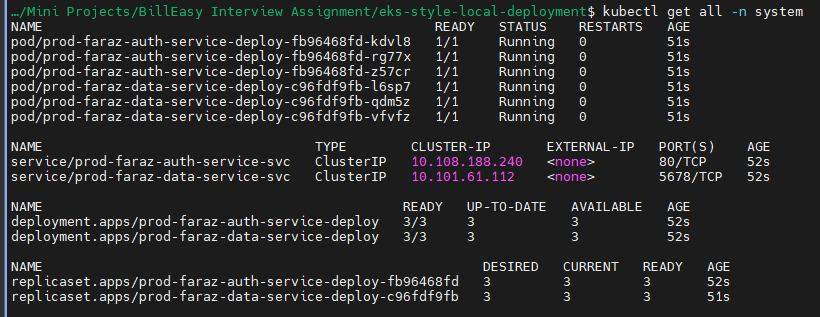
* kubectl apply -k kustomize/overlays/prod/



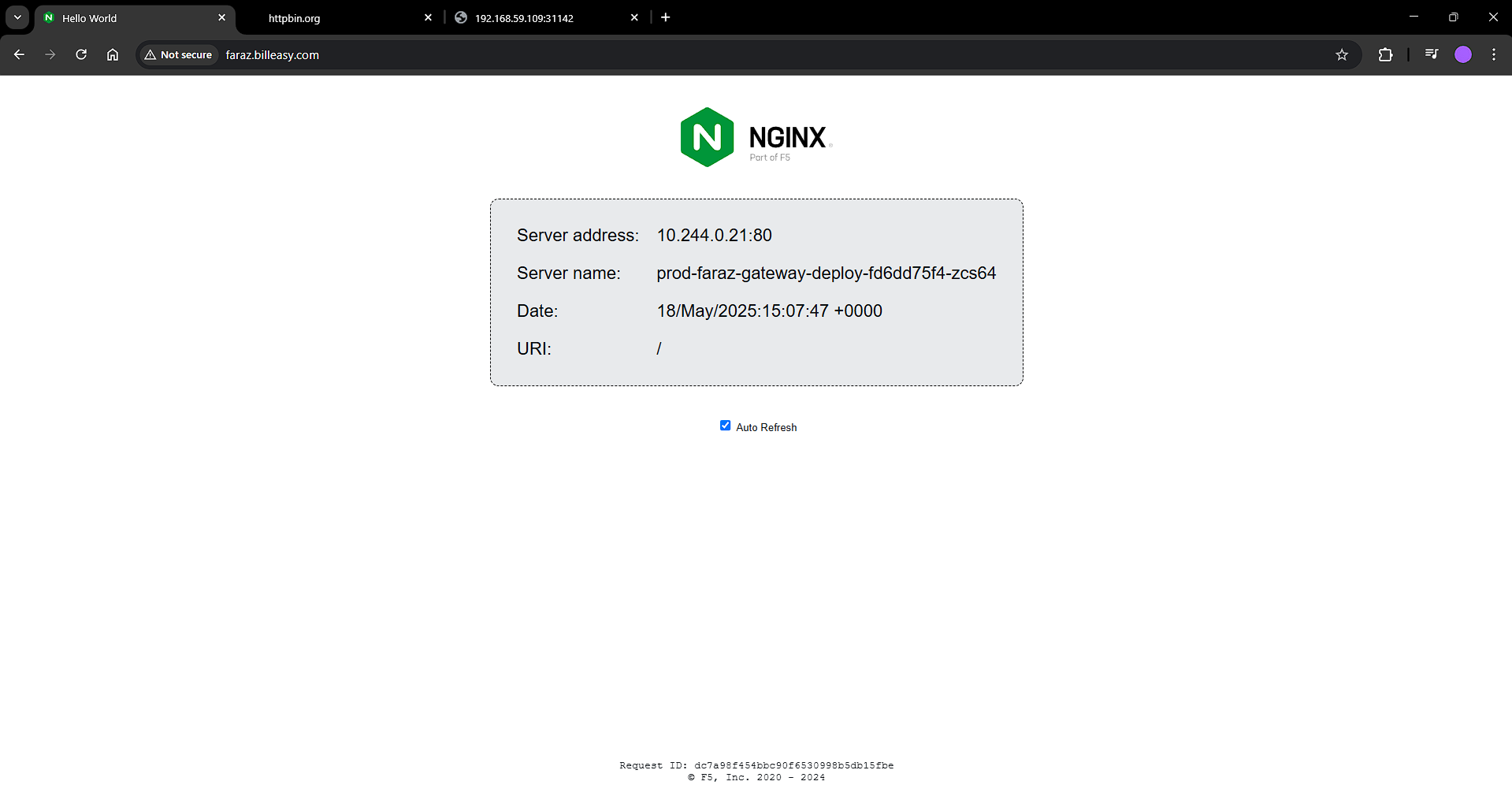
Verify your ingress, svc, deploy, pods:





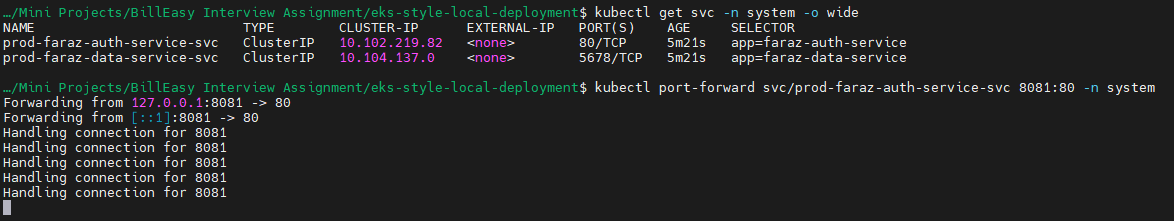


Visible in browser as ingress interacting with service with clusterIP mode and to prod-faraz-gateway-deploy deployment:



Using port forwarding to verify auth-service and data-service:

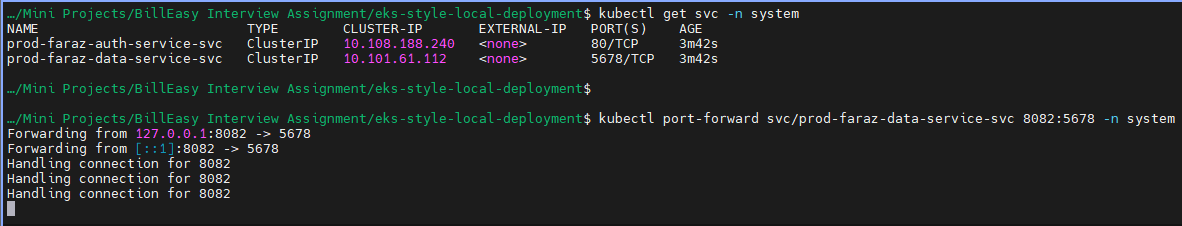
* kubectl port-forward svc/prod-faraz-auth-service-svc 8081:80 -n system



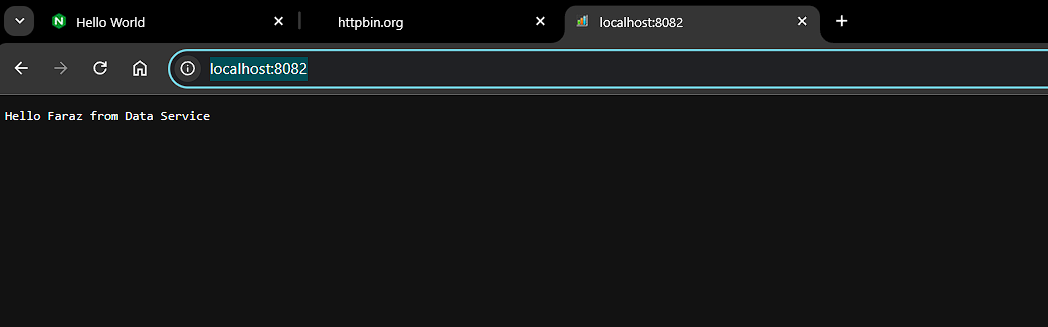
Open: http://localhost:8081/



* kubectl port-forward svc/prod-faraz-data-service-svc 8082:5678 -n system



Open: http://localhost:8082/



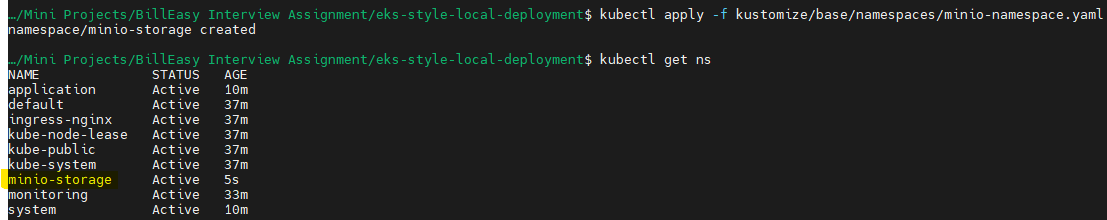
Task 2: Simulate IAM with MinIO

Create a namespace:

* kubectl create namespace minio-storage

or:

* kubectl apply -f kustomize/base/namespaces/minio-namespace.yaml

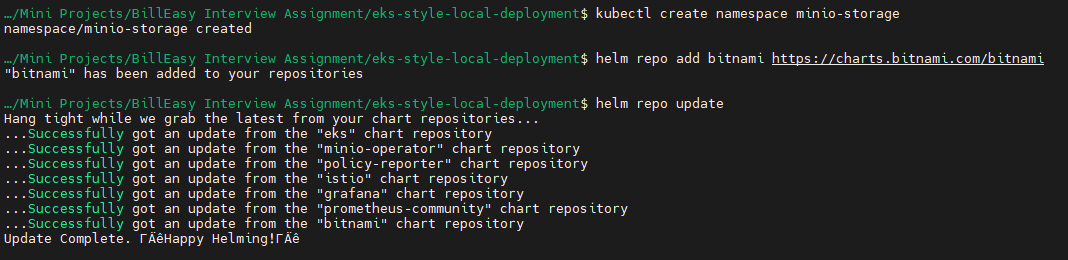


Add the chart:

* helm repo add bitnami <https://charts.bitnami.com/bitnami>

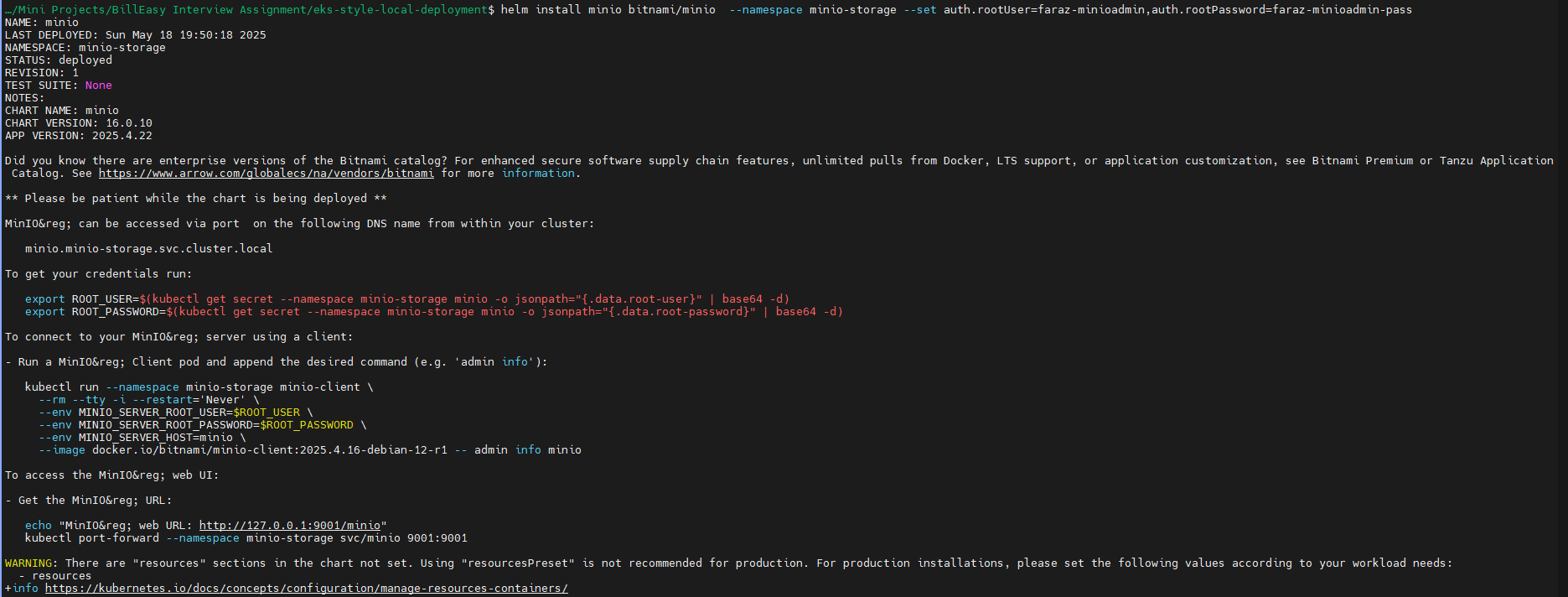
Update the repo:

* helm repo update

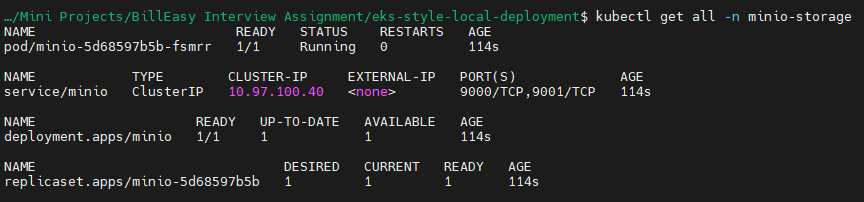


Install:

* helm install minio bitnami/minio --namespace minio-storage --set auth.rootUser=faraz-minioadmin,auth.rootPassword=faraz-minioadmin-pass

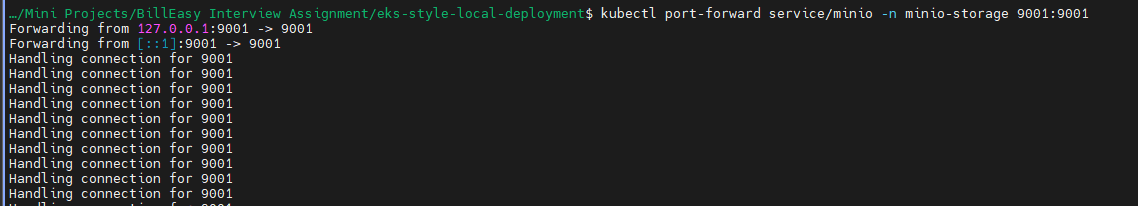


Wait for the pod to be created:



Now since the service is in cluster IP, we can use port forwarding:

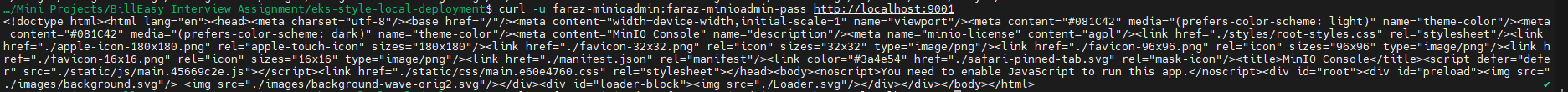
* For UI: kubectl port-forward service/minio -n minio-storage 9001:9001
* For CLI: kubectl port-forward service/minio -n minio-storage 9000:9000



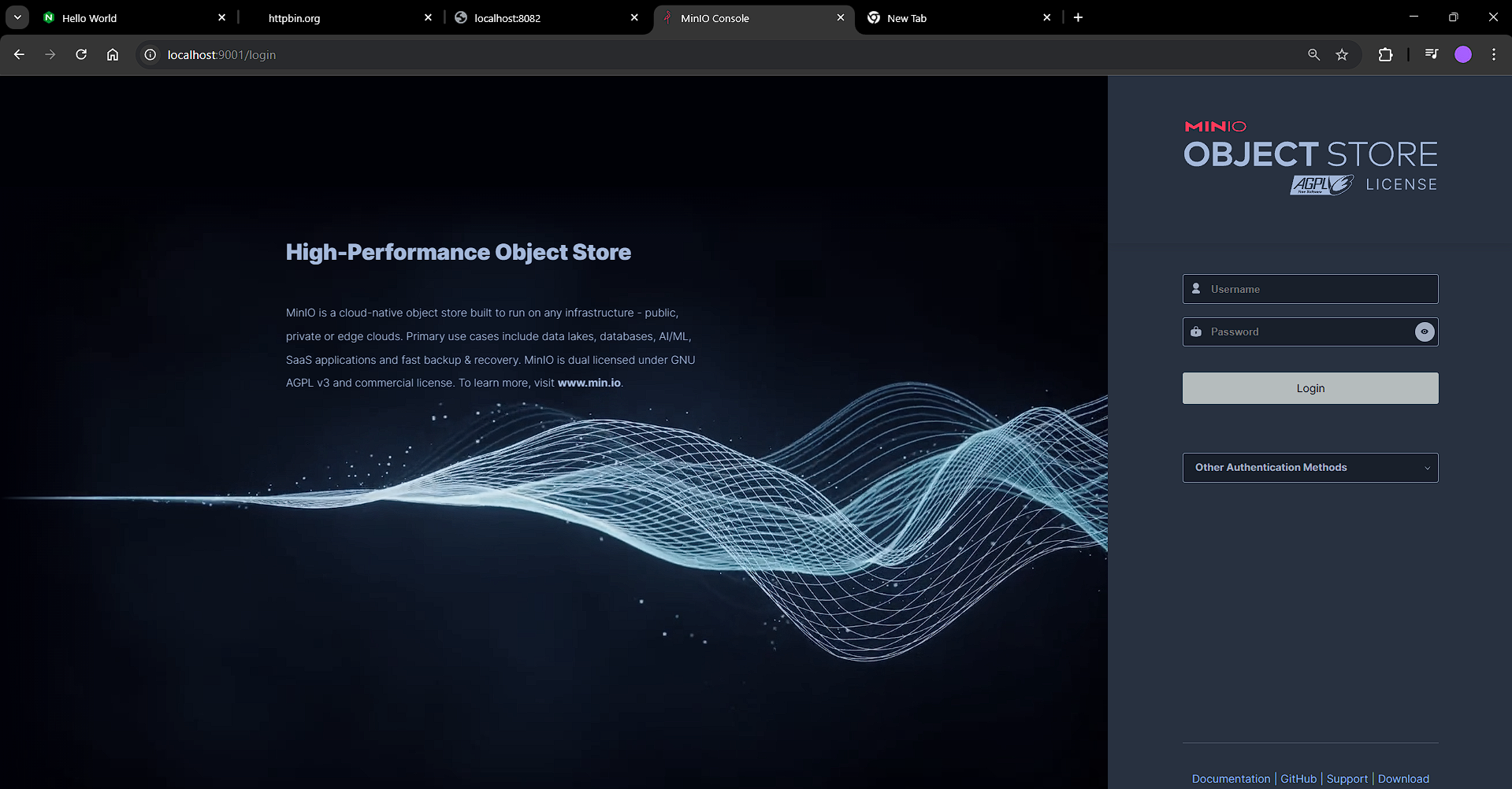
Open in browser: <http://localhost:9001/>

Or you can also curl:

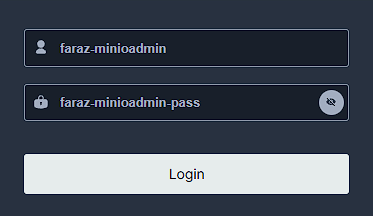
* curl -u faraz-minioadmin:faraz-minioadmin-pass http://localhost:9001



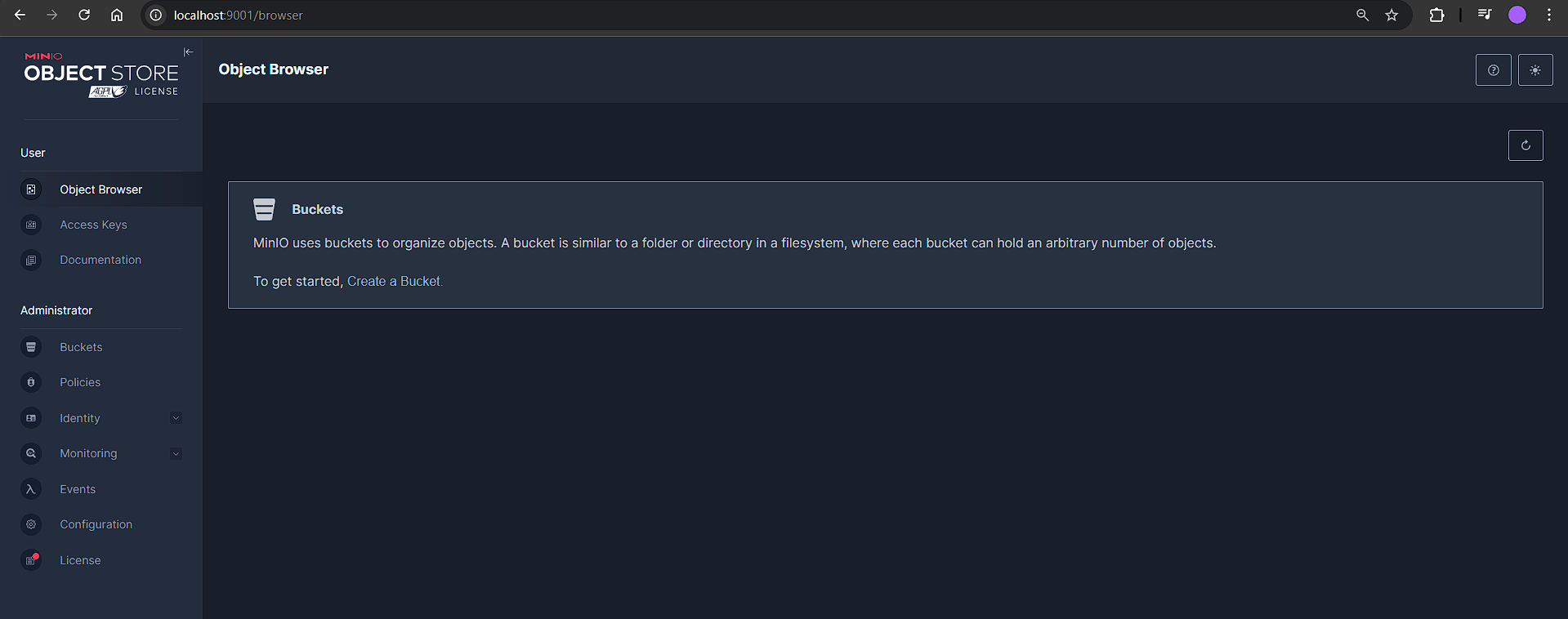
You will get this:



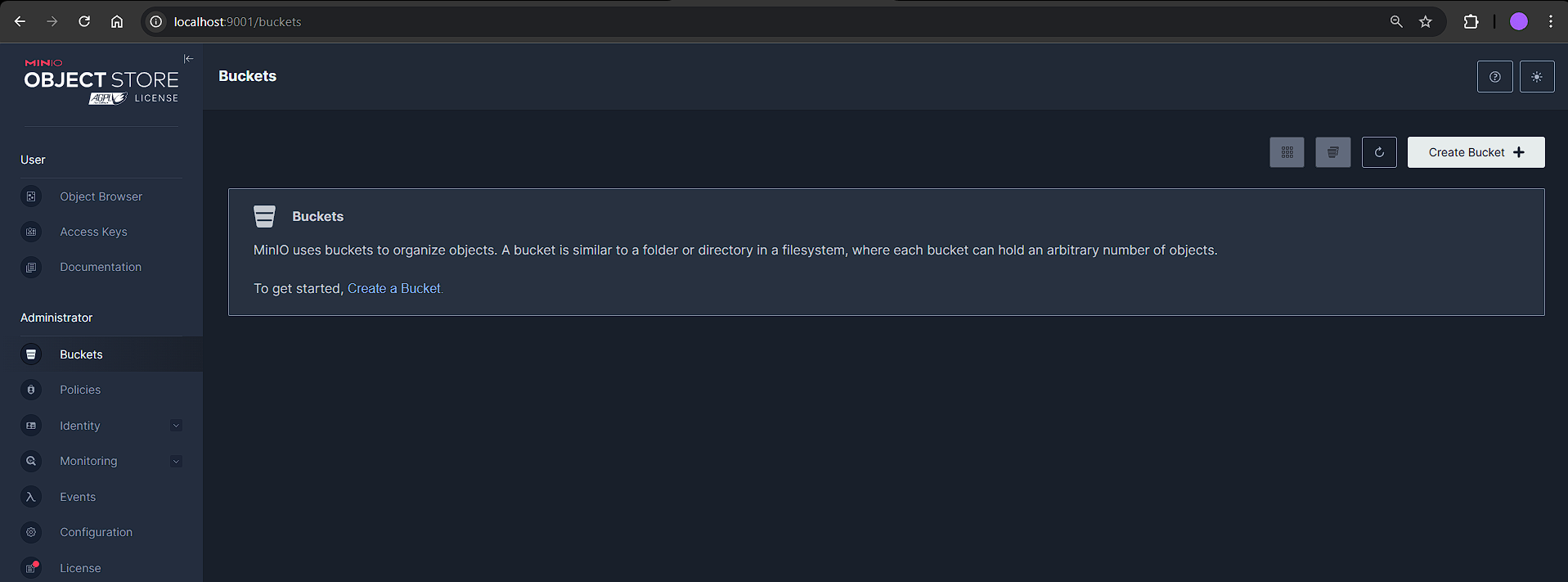
Type in the credentials:



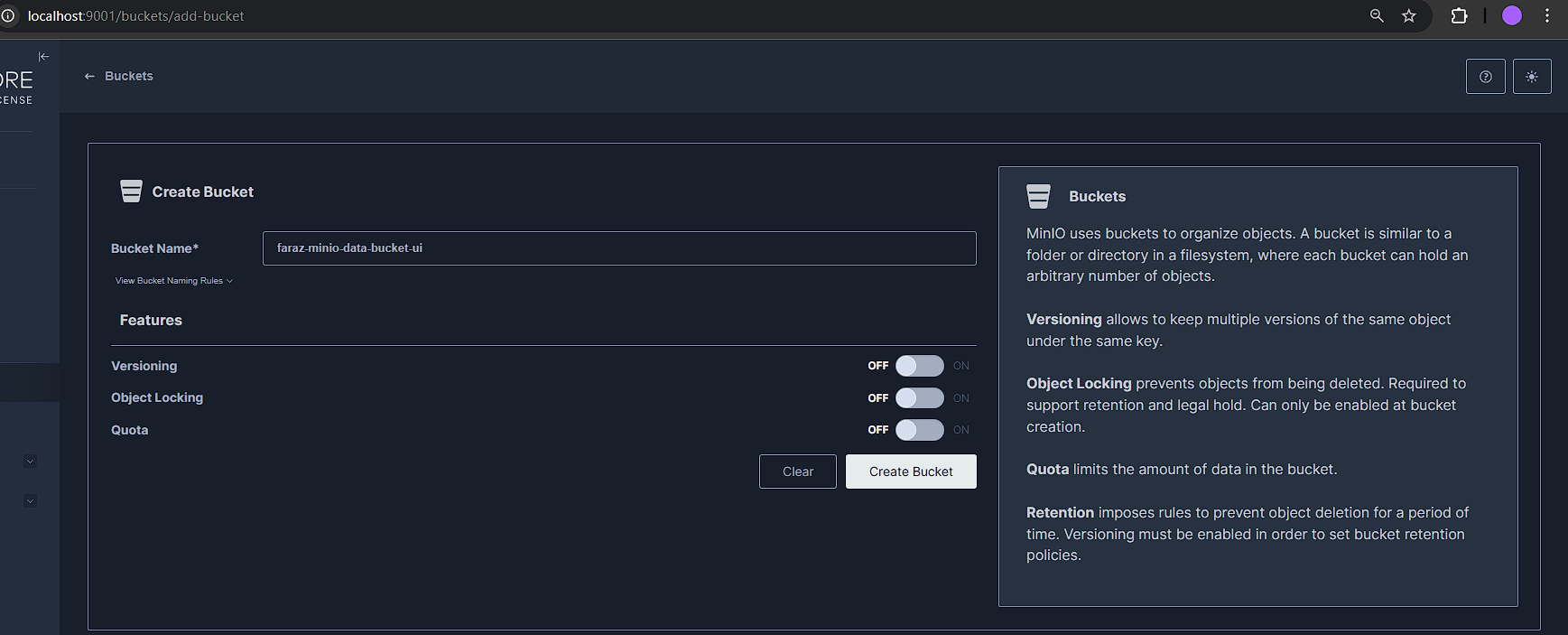
You will be directed to the /browser screen:

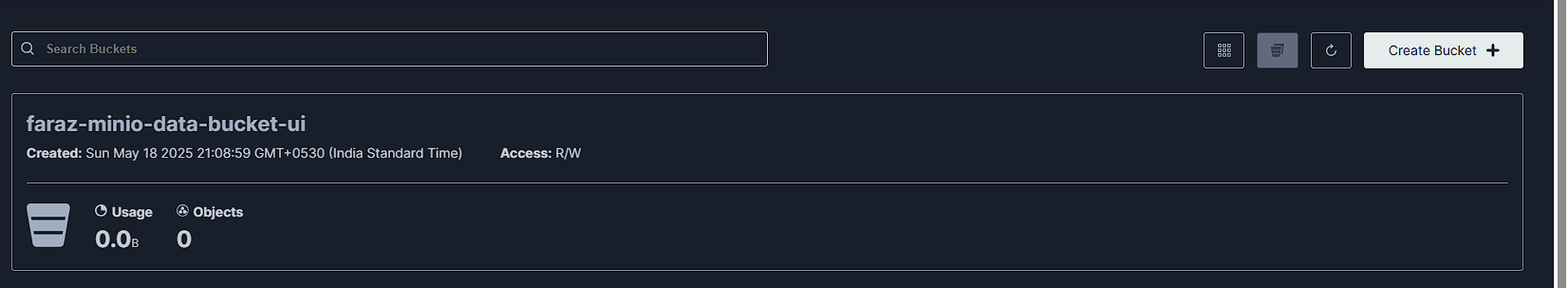


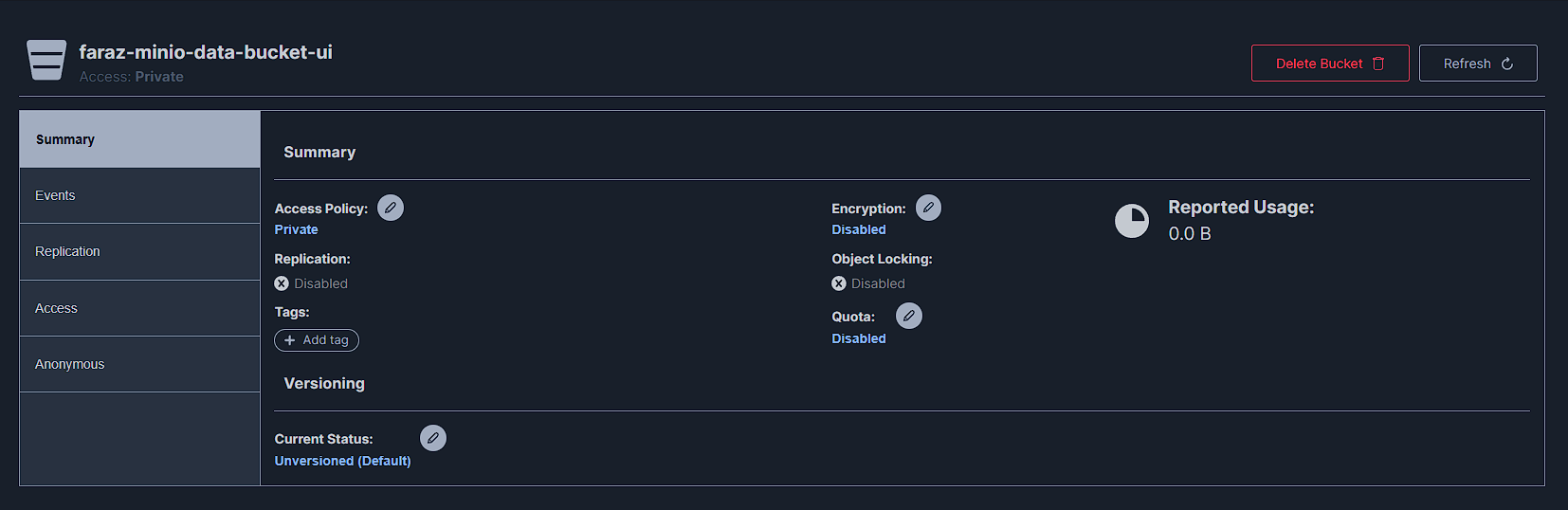
Now create a bucket, by clicking on create a bucket:



Create bucket with name “faraz-minio-data-bucket-ui”:

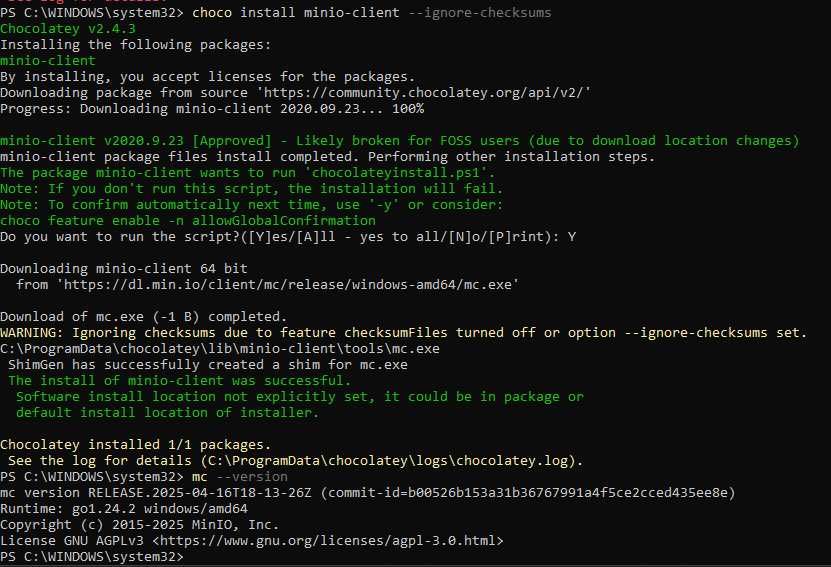






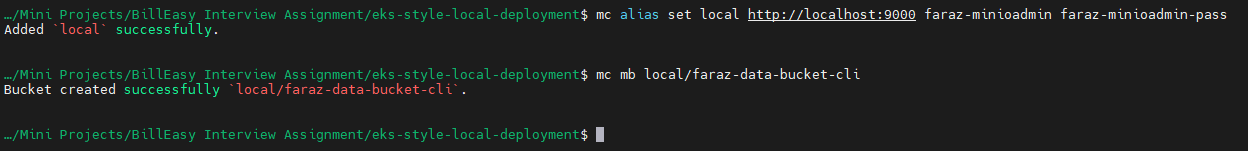
You can also create buckets by installing minio-client:

* Run as admin:
  + choco install minio-client --ignore-checksums

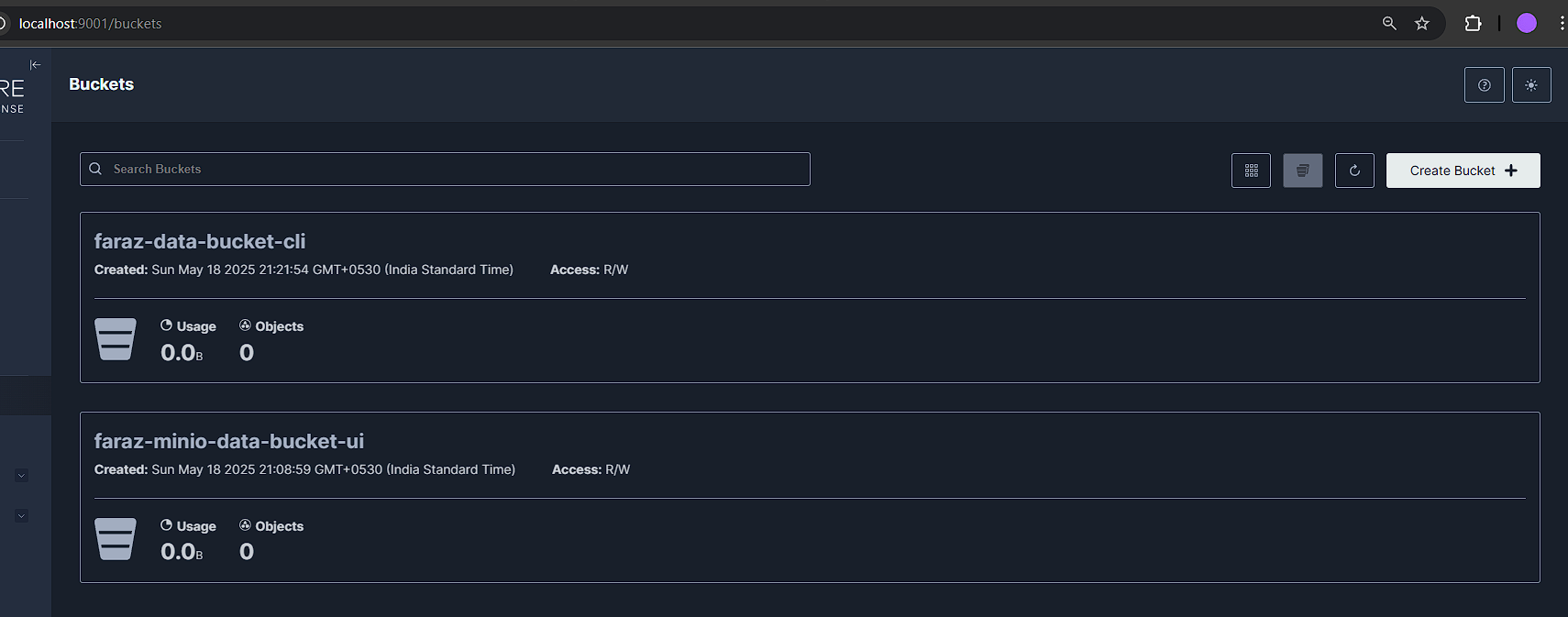


Now to create a bucket from cli:

* kubectl port-forward service/minio -n minio-storage 9000:9000
* mc alias set local http://localhost:9000 faraz-minioadmin faraz-minioadmin-pass
* mc mb local/faraz-data-bucket-cli

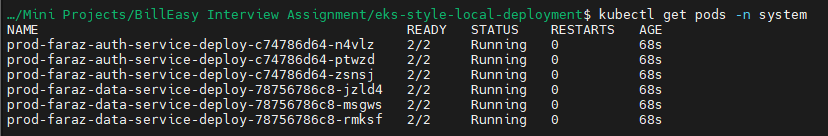


Now you will be able to see a new bucket created from cli on UI:

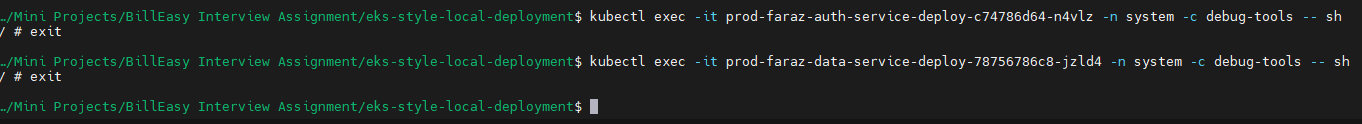


Adding Sidecars to the auth and data service pods deploy so that we can login and verify our connection:

Delete the prod overlay and reapply, you will now notice that there are 2 containers running inside the pod:

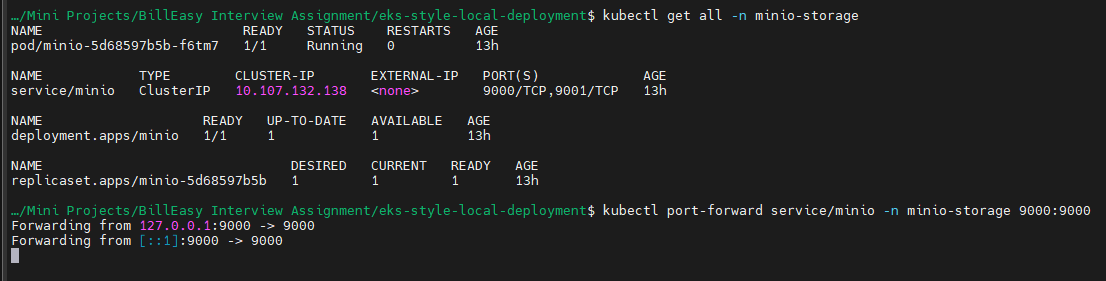


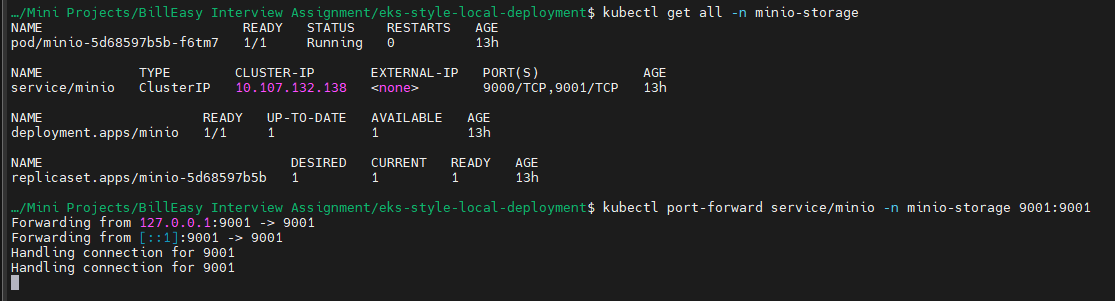
You will now be able to exec and login to the pods:



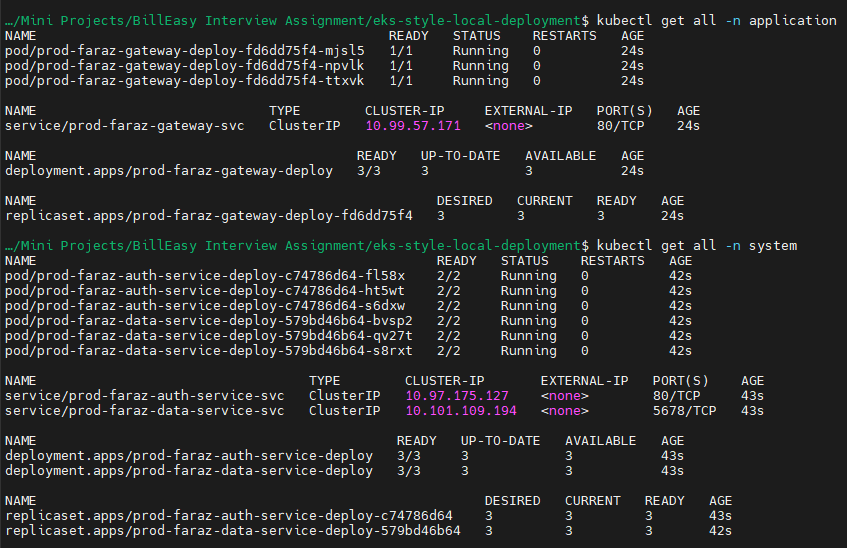
We need this as we will curl to the minio bucket

Ensure that you are port forwarding:

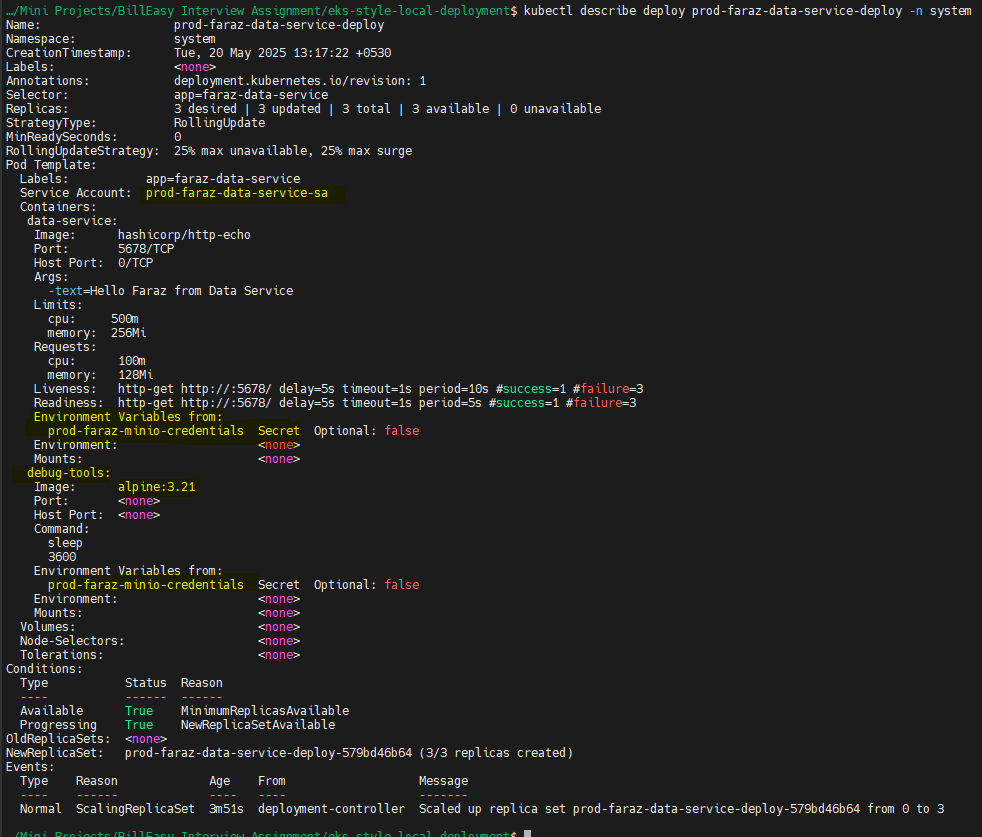




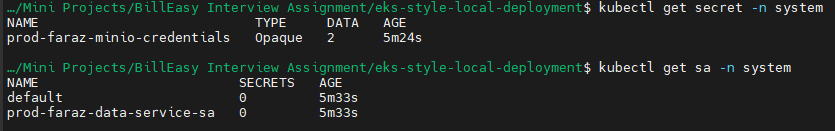
Ensuring that all the resources are up and running:



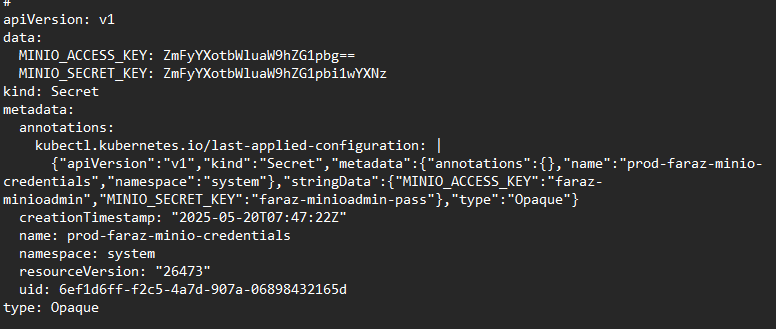
Now if you describe your data-service, you may see that environment variables from :



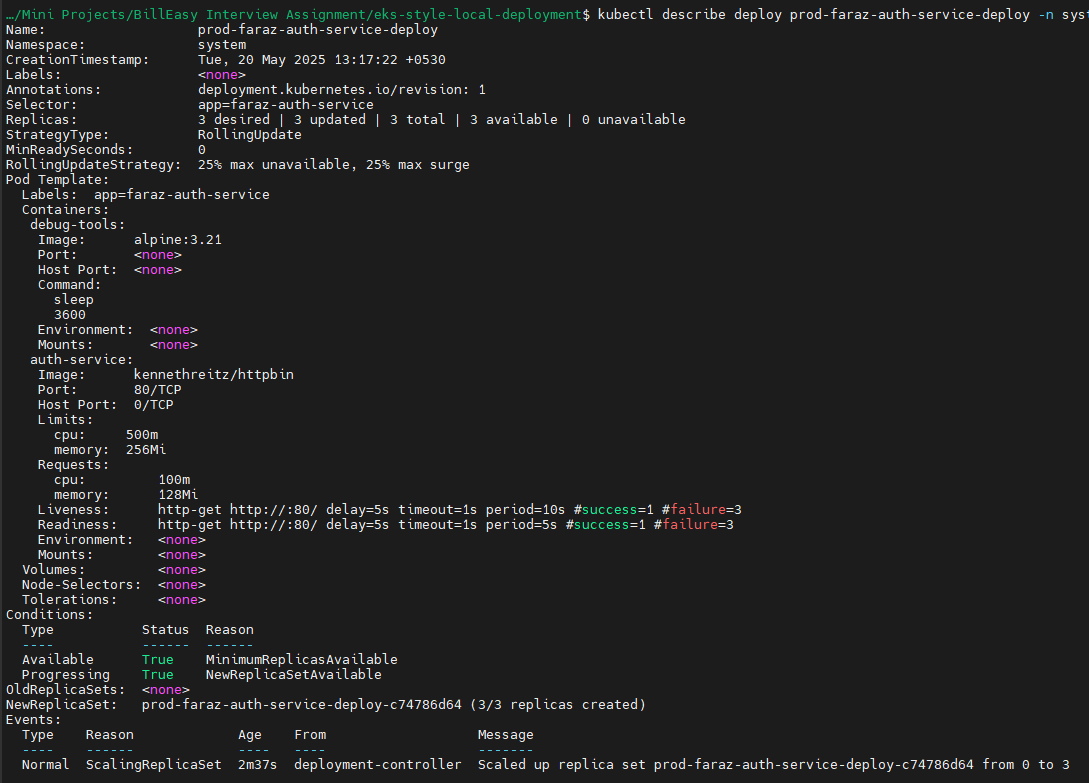
Service account and secret:



Your secret is stored:

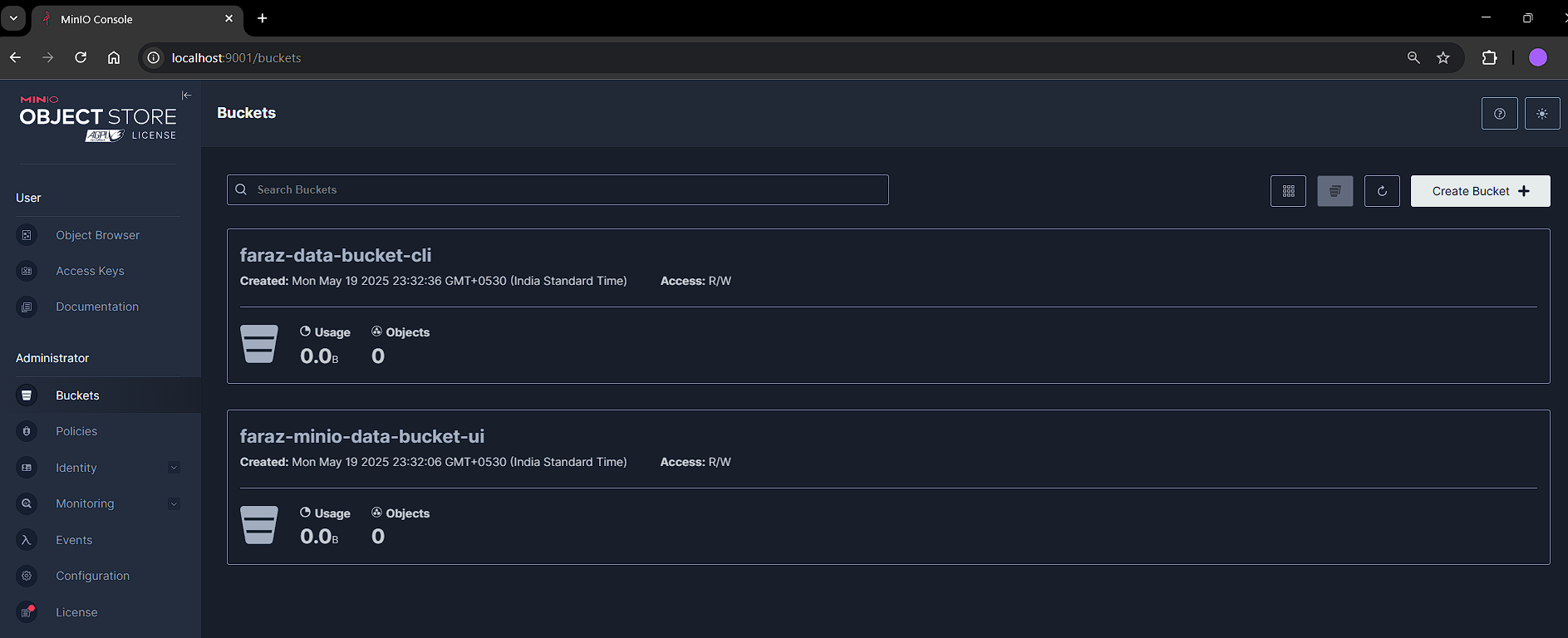


Now if you describe your auth-service, you may not find any environment variables or service account linked:



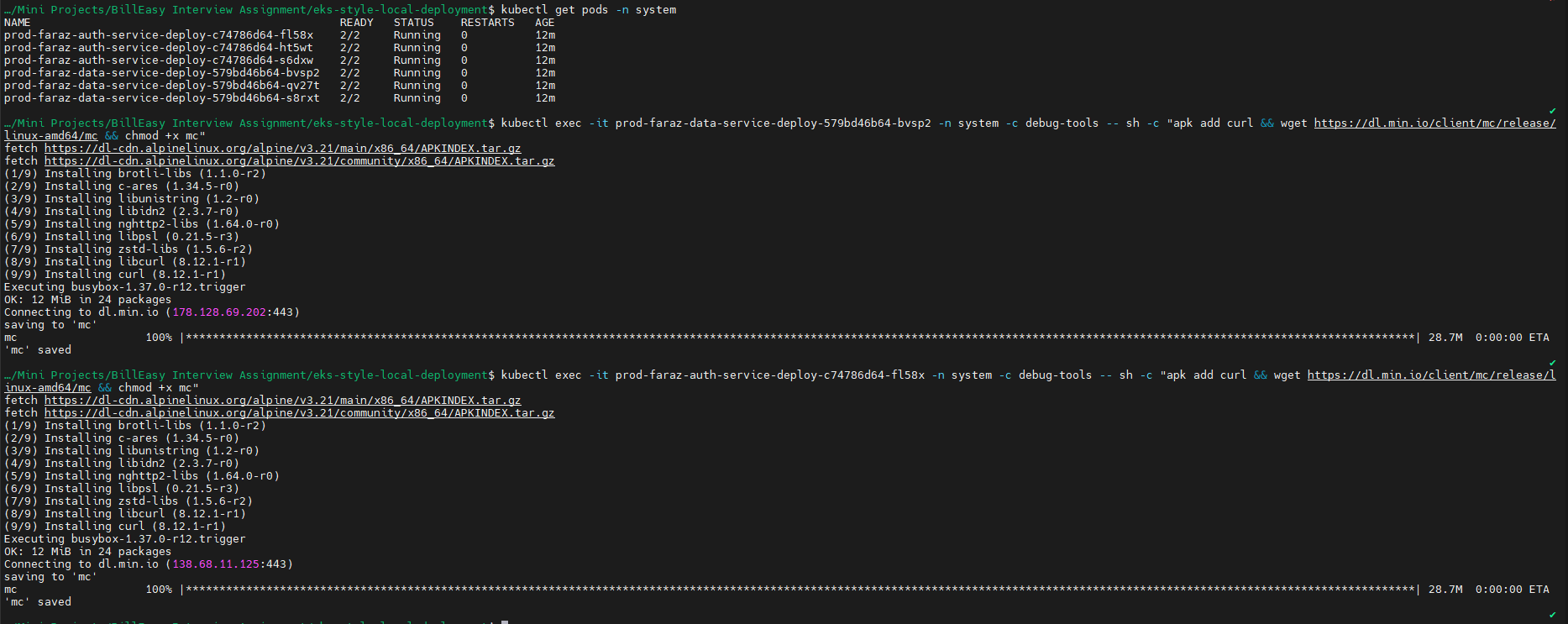
Now let’s test if my data-service has the ability to talk to my minio buckets by listing the buckets:

Currently we have the following buckets:



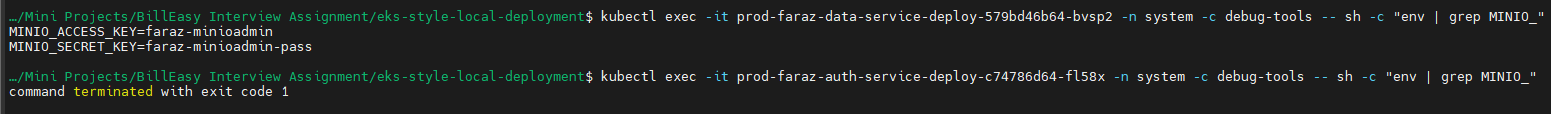
Now lets setup curl, and minio-client so that we have the ability to list the buckets:

* kubectl exec -it prod-faraz-**data-service**-deploy-579bd46b64-bvsp2 -n system -c debug-tools -- sh -c "apk add curl && wget https://dl.min.io/client/mc/release/linux-amd64/mc && chmod +x mc"
* kubectl exec -it prod-faraz-**auth-service-**deploy-c74786d64-fl58x -n system -c debug-tools -- sh -c "apk add curl && wget https://dl.min.io/client/mc/release/linux-amd64/mc && chmod +x mc"



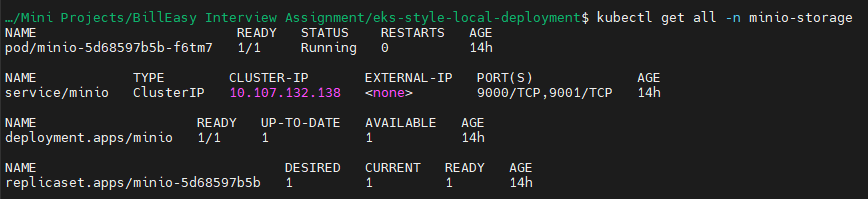
Now try to get the environment variables on from this pods:

* kubectl exec -it prod-faraz-**data-service**-deploy-579bd46b64-bvsp2 -n system -c debug-tools -- sh -c "env | grep MINIO\_"
* kubectl exec -it prod-faraz-**auth-service**-deploy-c74786d64-fl58x -n system -c debug-tools -- sh -c "env | grep MINIO\_"



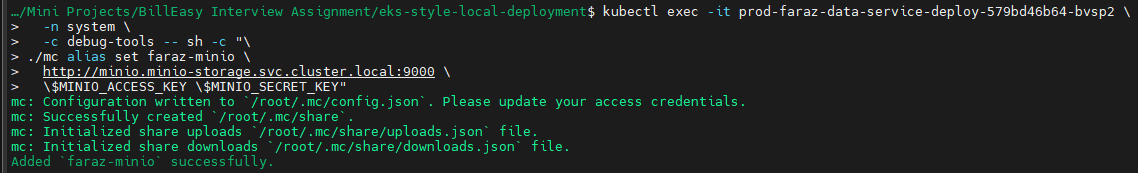
Here you will notice that data-service has the access\_key and secret\_key which we set using secrets and just allowed service account to access this secret. Hence, data-service returned the env secret while auth-service did not return anything.

Now let’s try listing the buckets in minio-storage namespace from data-service in system namespace:



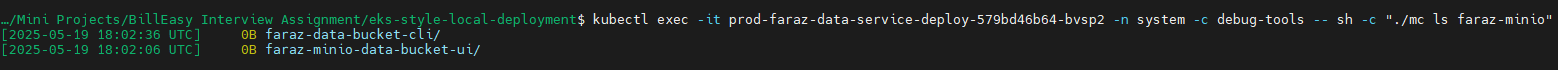
First we need to set alias:

* kubectl exec -it prod-faraz-data-service-deploy-579bd46b64-bvsp2 -n system -c debug-tools -- sh -c "./mc alias set faraz-minio http://minio.minio-storage.svc.cluster.local:9000 \$MINIO\_ACCESS\_KEY \$MINIO\_SECRET\_KEY"



Listing the buckets:

* kubectl exec -it prod-faraz-data-service-deploy-579bd46b64-bvsp2 -n system -c debug-tools -- sh -c "./mc ls faraz-minio"

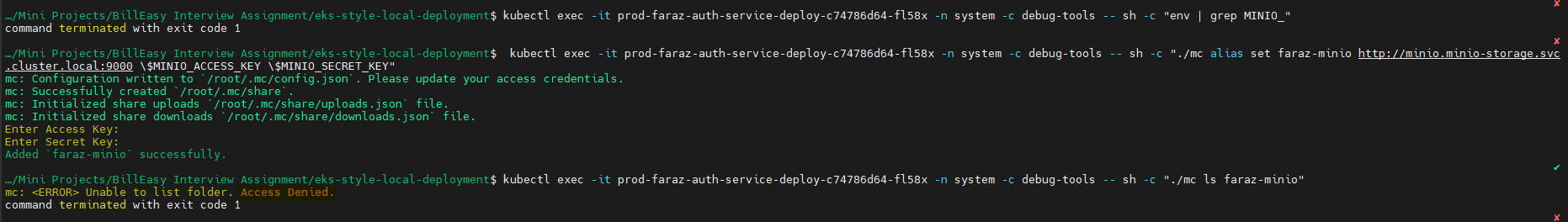


You can also create a test bucket:

* kubectl exec -it prod-faraz-data-service-deploy-579bd46b64-bvsp2 -n system -c debug-tools -- sh -c "./mc mb faraz-minio/testbucket"



Let’s try to do the same using auth-service:



Here as you can see that the access is denied.

Let’s say somehow I got access to my credentials:

I will be able to access and list my buckets on minio:

