Smart Cow - React Flask NGINX Minikube_Kubernetes Deployment Guide

Task #3

https://github.com/ryancomia/sc-exercise/tree/feature/kubernetes

Before I begin the final task (making it work with mini-kube/kubernetes) I wanted to start with creating a workflow which would automate the build of images and deploy them into container registry.

Found here:
https://github.com/ryancomia/sc-exercise/blob/feature/kubernetes/.github/workflows/build.yml

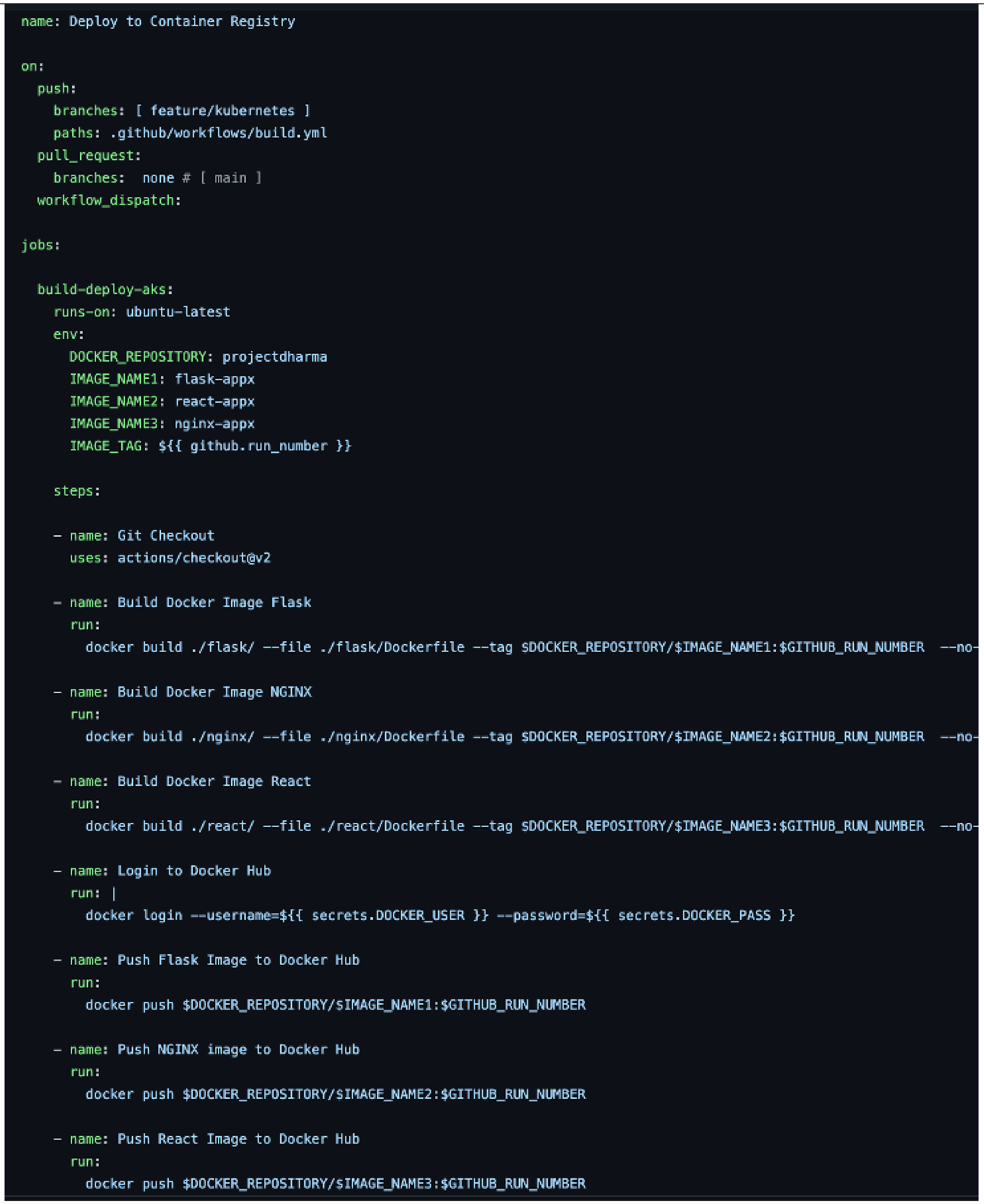
For this I created a workflow using github action which does the following:

- Build the images from the source code
- Logins to Docker HUB
-Upload to Repo

Note: Initially I was planning to integrate SEC into the step with
Tools such as Trivy, Snyk etc. However, with the time constraints and such step will take more time for me to upload the images to CR. I decided to skip it.

Shown are the steps that the workflow will run

I created a branch called feature/Kubernetes
The workflow is trigerred upon a push/pull request to the branch.



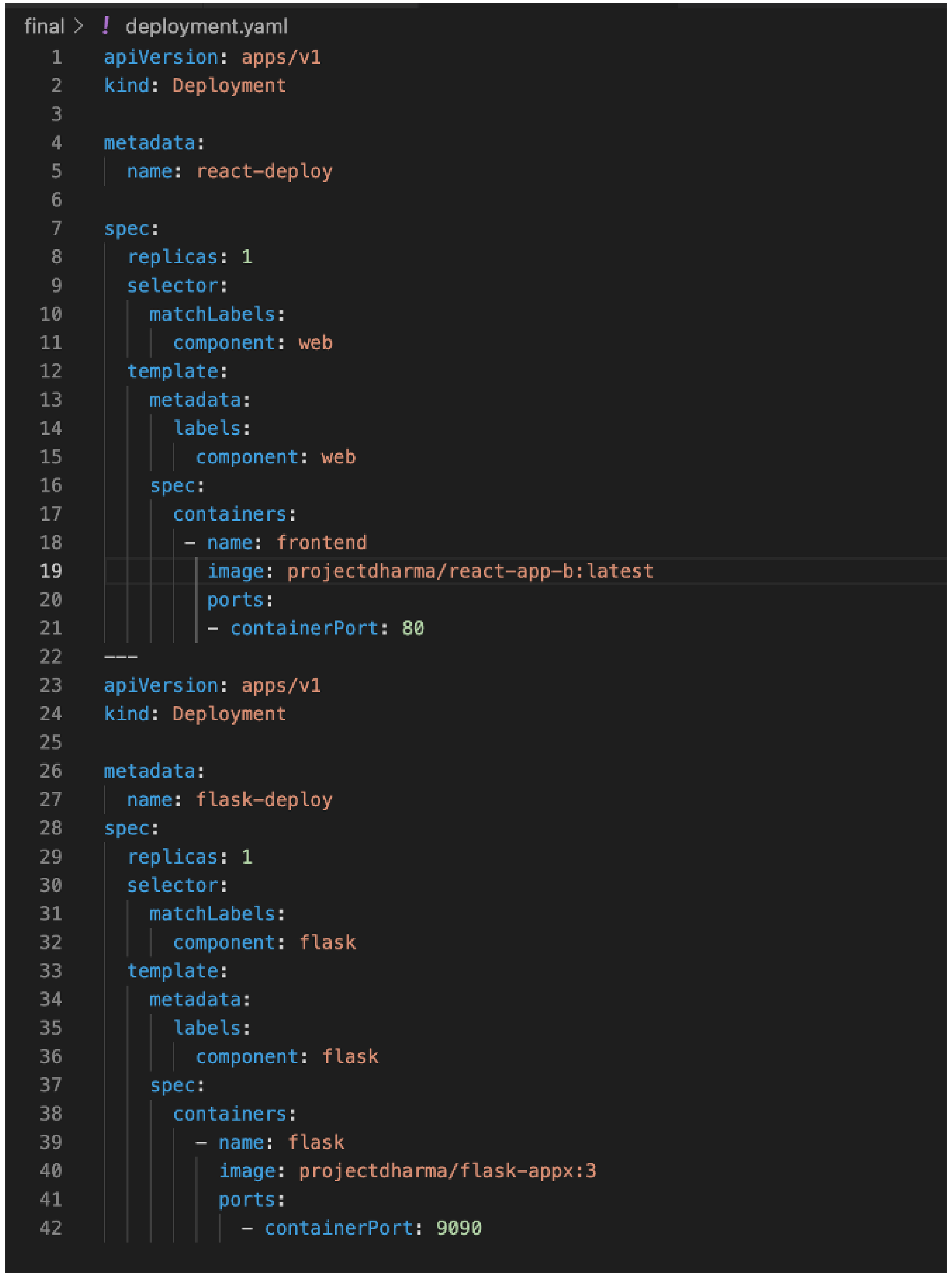
```
name: Deploy to Container Registry

on:
  push:
    branches: [ feature/kubernetes ]
    paths: .github/workflows/build.yml
  pull_request:
    branches:  none # [ main ]
  workflow_dispatch:

jobs:

  build-deploy-aks:
    runs-on: ubuntu-latest
    env:
      DOCKER_REPOSITORY: projectdharma
      IMAGE_NAME1: flask-appx
      IMAGE_NAME2: react-appx
      IMAGE_NAME3: nginx-appx
      IMAGE_TAG: ${{ github.run_number }}

    steps:

    - name: Git Checkout
      uses: actions/checkout@v2

    - name: Build Docker Image Flask
      run:
        docker build ./flask/ --file ./flask/Dockerfile --tag $DOCKER_REPOSITORY/$IMAGE_NAME1:$GITHUB_RUN_NUMBER  --no-

    - name: Build Docker Image NGINX
      run:
        docker build ./nginx/ --file ./nginx/Dockerfile --tag $DOCKER_REPOSITORY/$IMAGE_NAME2:$GITHUB_RUN_NUMBER  --no-

    - name: Build Docker Image React
      run:
        docker build ./react/ --file ./react/Dockerfile --tag $DOCKER_REPOSITORY/$IMAGE_NAME3:$GITHUB_RUN_NUMBER  --no-

    - name: Login to Docker Hub
      run: |
        docker login --username=${{ secrets.DOCKER_USER }} --password=${{ secrets.DOCKER_PASS }}

    - name: Push Flask Image to Docker Hub
      run:
        docker push $DOCKER_REPOSITORY/$IMAGE_NAME1:$GITHUB_RUN_NUMBER

    - name: Push NGINX image to Docker Hub
      run:
        docker push $DOCKER_REPOSITORY/$IMAGE_NAME2:$GITHUB_RUN_NUMBER

    - name: Push React Image to Docker Hub
      run:
        docker push $DOCKER_REPOSITORY/$IMAGE_NAME3:$GITHUB_RUN_NUMBER
```

Deployment.yaml

In here I created a basic deployment file
Which specifies the react and flask service.

And for this task I omitted the use of nginx reverse proxy on task 1 and 2 (I will explain later why)

I just defined the basic parameters that would provide identity for the container and its characteristics such as port ingress



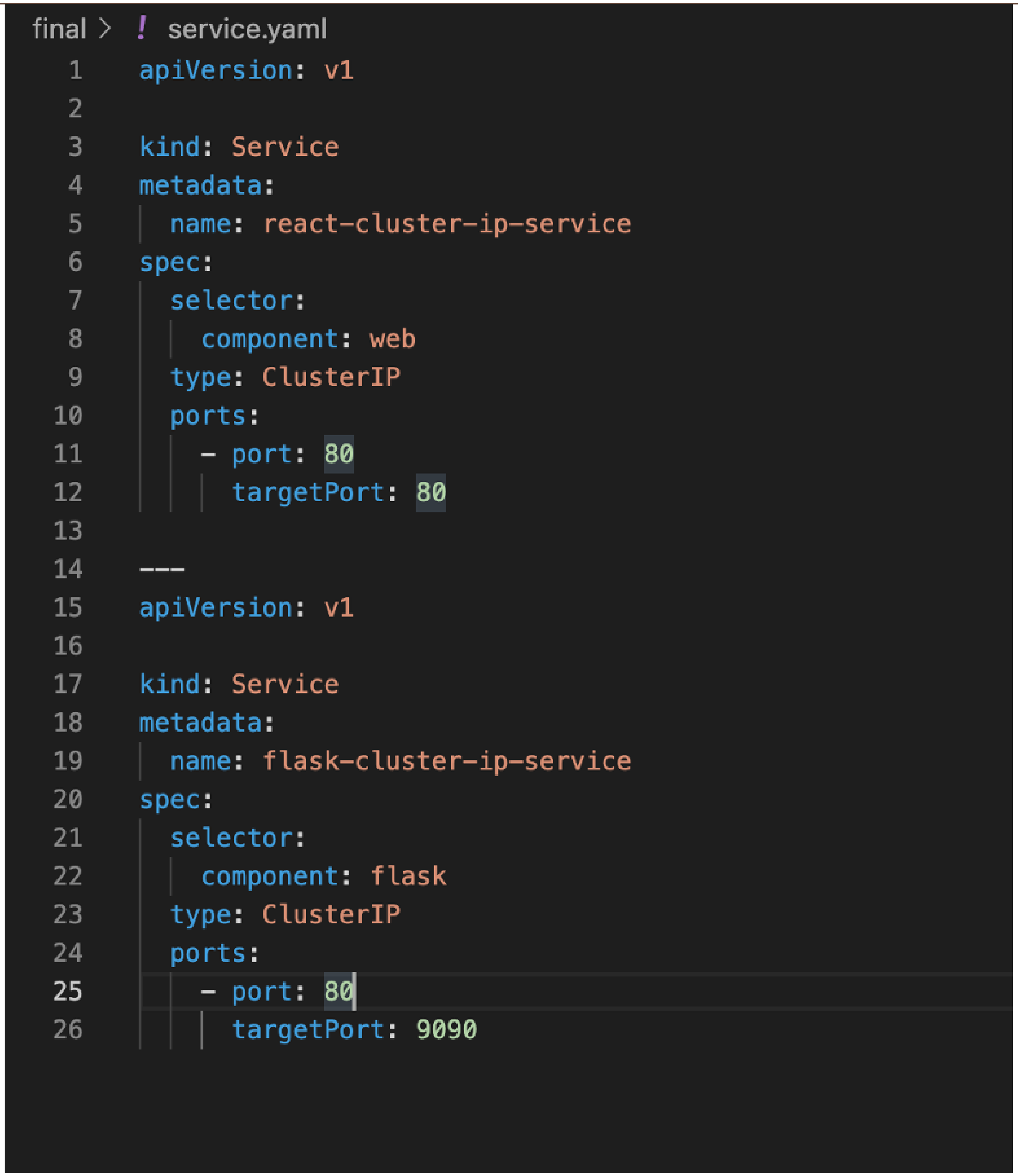
final > ! deployment.yaml

```yaml
apiVersion: apps/v1
kind: Deployment

metadata:
  name: react-deploy

spec:
  replicas: 1
  selector:
    matchLabels:
      component: web
  template:
    metadata:
      labels:
        component: web
    spec:
      containers:
       - name: frontend
         image: projectdharma/react-app-b:latest
         ports:
         - containerPort: 80
---
apiVersion: apps/v1
kind: Deployment

metadata:
  name: flask-deploy
spec:
  replicas: 1
  selector:
    matchLabels:
      component: flask
  template:
    metadata:
      labels:
        component: flask
    spec:
      containers:
        - name: flask
          image: projectdharma/flask-appx:3
          ports:
           - containerPort: 9090
```

Service.yaml

This section we can configure the networking and DNS properties of the container.
Using a ClusterIP type as it would use the Virtual IP inside the Kubernetes cluster

Exposing the from and to ports


final > ! service.yaml

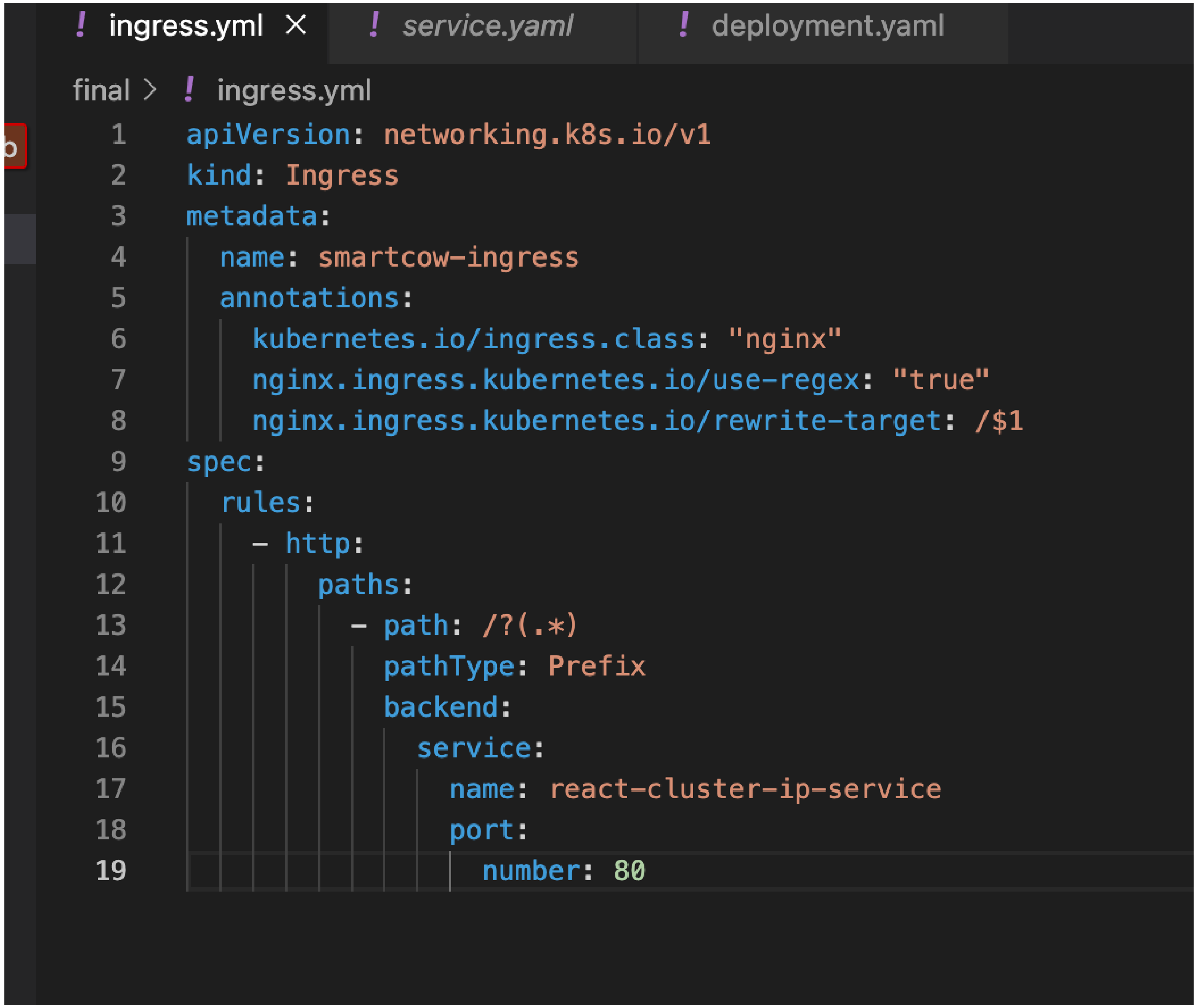
```yaml
apiVersion: v1

kind: Service
metadata:
  name: react-cluster-ip-service
spec:
  selector:
    component: web
  type: ClusterIP
  ports:
    - port: 80
      targetPort: 80

---
apiVersion: v1

kind: Service
metadata:
  name: flask-cluster-ip-service
spec:
  selector:
    component: flask
  type: ClusterIP
  ports:
    - port: 80
      targetPort: 9090
```

Ingress.yaml

Since Kubernetes resides on a virtual environment, it follows that it has a different network isolated from the rest. So the way we communicate from external to Kubernetes is by using Ingress service.

This is similar to what a reverse proxy like nginx would do in context. (Well infact it is nginx)

So in here we specify the kind which Ingress Followed by the rules and root path.

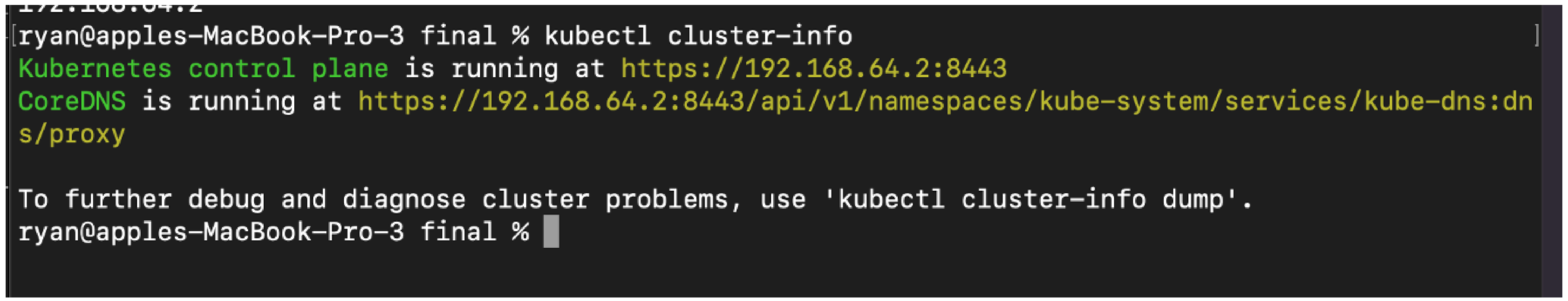


```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: smartcow-ingress
  annotations:
    kubernetes.io/ingress.class: "nginx"
    nginx.ingress.kubernetes.io/use-regex: "true"
    nginx.ingress.kubernetes.io/rewrite-target: /$1
spec:
  rules:
    - http:
        paths:
          - path: /?(.*)
            pathType: Prefix
            backend:
              service:
                name: react-cluster-ip-service
                port:
                  number: 80
```

Unlike on task1/2
In here I just specified an entry for my react container, which runs nginx in the background.
I let the react in nginx do the routing

The reason why I omitted the reverse proxy container

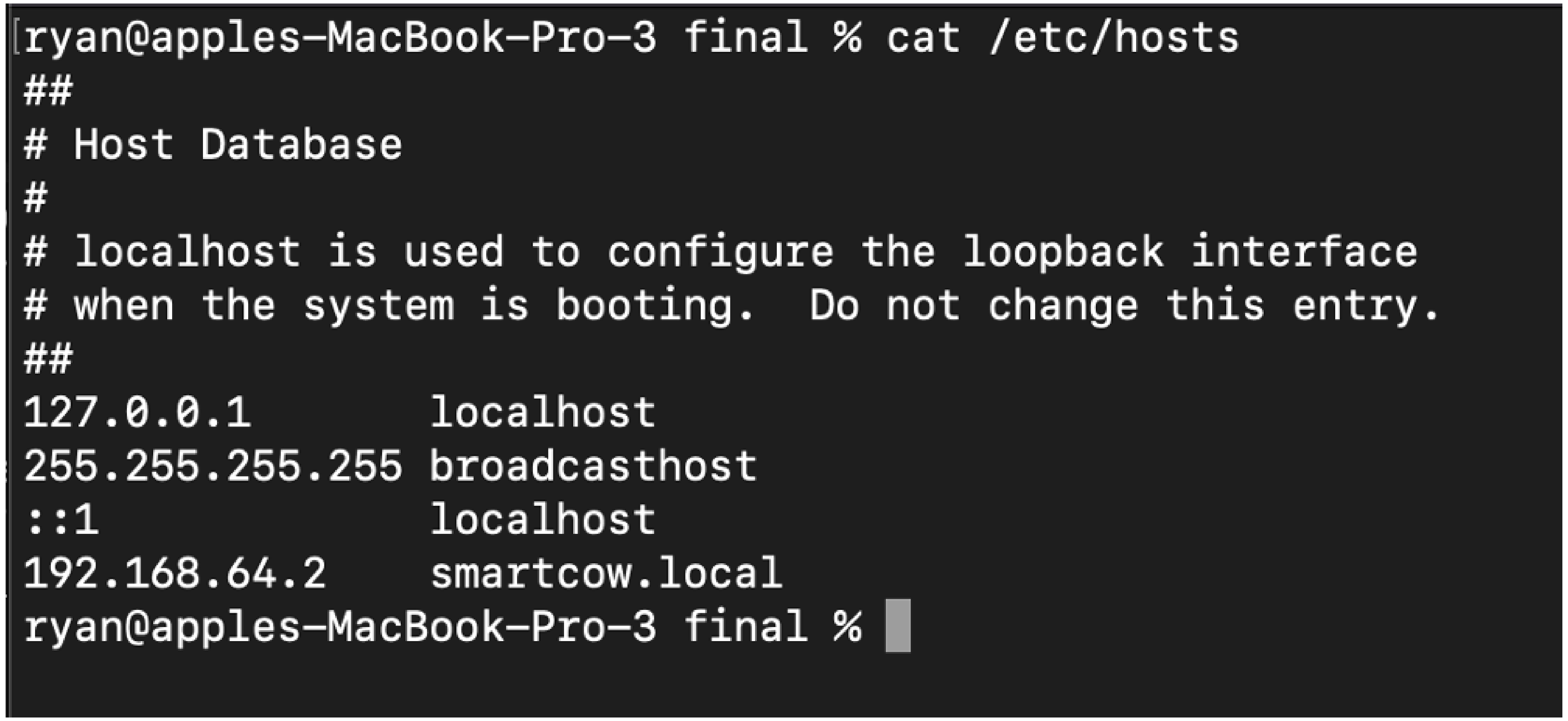
Adding the minikube ingress plugin I was able now to route the traffic from and to my local



```
ryan@apples-MacBook-Pro-3 final % kubectl cluster-info
Kubernetes control plane is running at https://192.168.64.2:8443
CoreDNS is running at https://192.168.64.2:8443/api/v1/namespaces/kube-system/services/kube-dns:dn
s/proxy

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.
ryan@apples-MacBook-Pro-3 final %
```

Similar to task 2, I had to find a way to route the traffic back to flask. So what I did is to create a stub record into my local host file

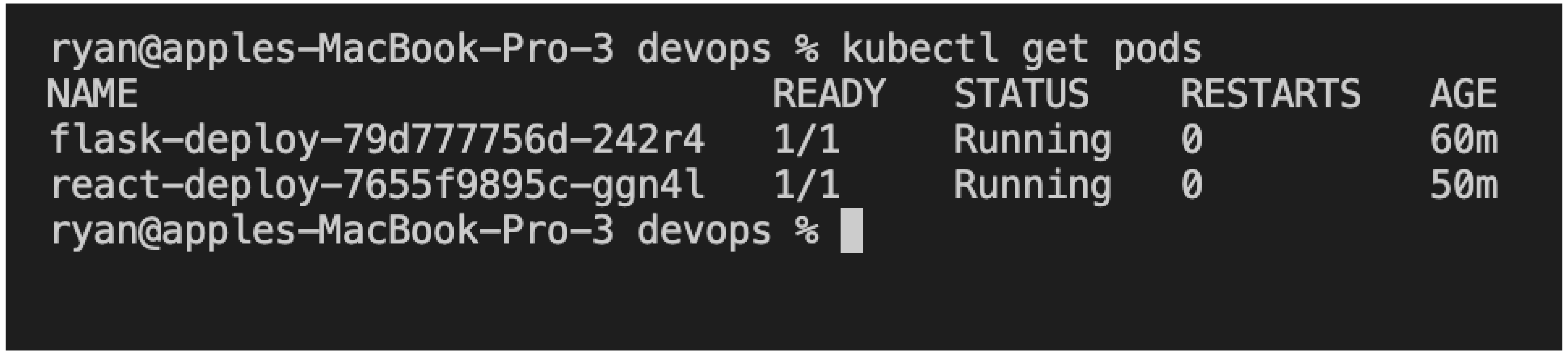
The minikube router IP will be CNAME'd to smartcow.local/



```
ryan@apples-MacBook-Pro-3 final % cat /etc/hosts
##
# Host Database
#
# localhost is used to configure the loopback interface
# when the system is booting.  Do not change this entry.
##
127.0.0.1       localhost
255.255.255.255 broadcasthost
::1             localhost
192.168.64.2    smartcow.local
ryan@apples-MacBook-Pro-3 final %
```

Finally running
Kubectl apply -f .

Spins up 2 container app



```
ryan@apples-MacBook-Pro-3 devops % kubectl get pods
NAME                            READY   STATUS    RESTARTS   AGE
flask-deploy-79d777756d-242r4   1/1     Running   0          60m
react-deploy-7655f9895c-ggn4l   1/1     Running   0          50m
ryan@apples-MacBook-Pro-3 devops %
```

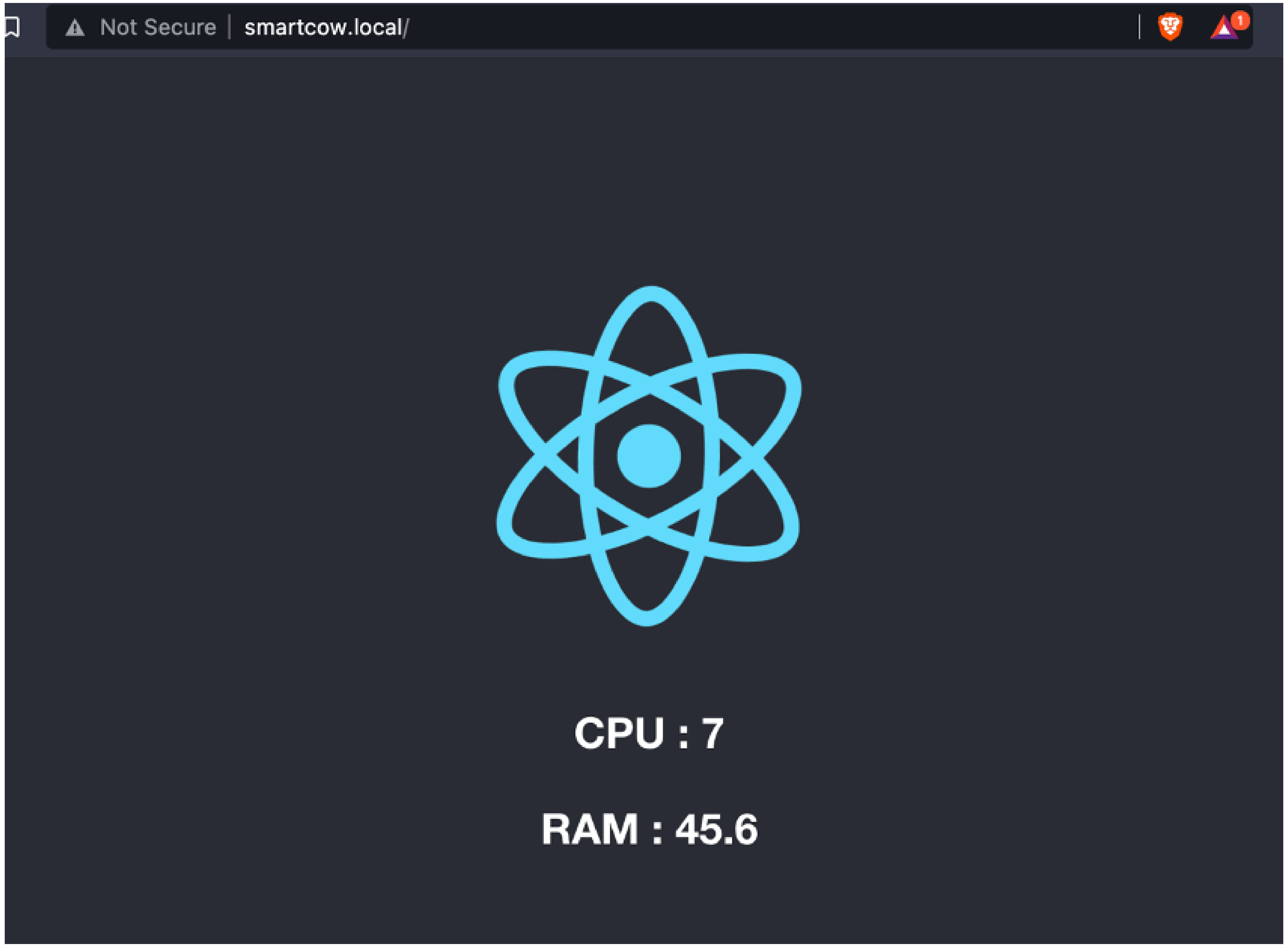
Since we have a dns record locally.

I can use the url:

http://smartcow.local

To view the app on my browser

Note: the app.js uses the same url to fetch the data from flask



CPU : 7

RAM : 45.6