**Kubernetics:**

**Container Orchestration system (K8S). If lot of docker containers to manage, and a container crashes, to restart the container.** So we need orchestration system. Will create manifests (dockerFile), no intervention required by administrator. Load balancing, Kubernetes: for automating deployment, scaling, management of containerized applications. Originated from google, now managed independently of Google. Kubernetes is far richer than Docker swarm, can deploy these manifests to diff cloud providers as it can configure your infrastructure. All load balancers, hard disks, instances are all automatically created.

Docker swarm comes integrated with docker engine, so need to do nothing to install it. Built-in. Very simple compared to Kubernetes. Requires manual set-up on cloud platform.

Google running 2 billion containers in a single week.

**Install Minikube for local machine:**

<https://kubernetes.io/docs/tasks/tools/install-minikube/>

To check if virtualization is supported on Windows 8 and above, run the following command on your Windows terminal or command prompt.

systeminfo

If you see the following output, virtualization is supported on Windows.

Hyper-V Requirements: VM Monitor Mode Extensions: Yes

Virtualization Enabled In Firmware: Yes

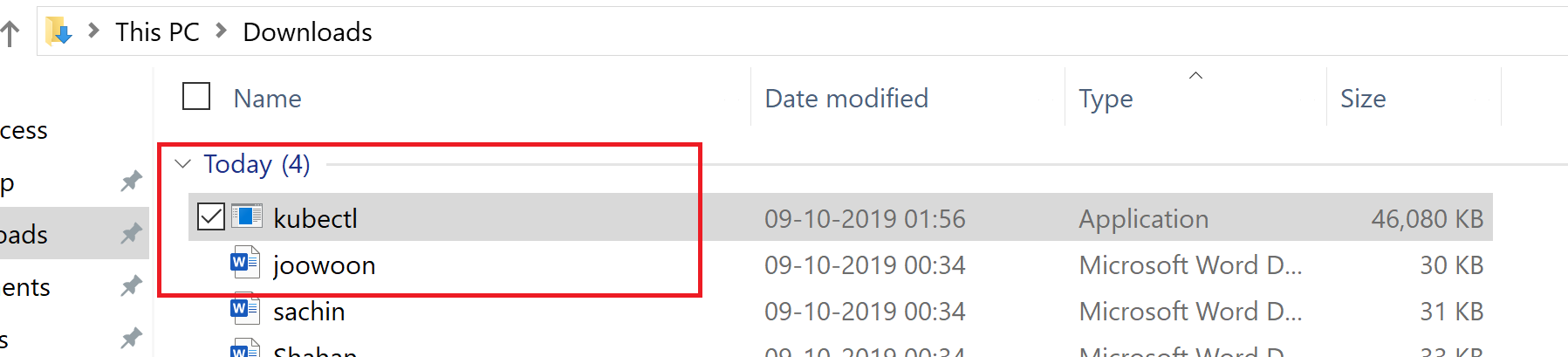
Second Level Address Translation: Yes

Data Execution Prevention Available: Yes

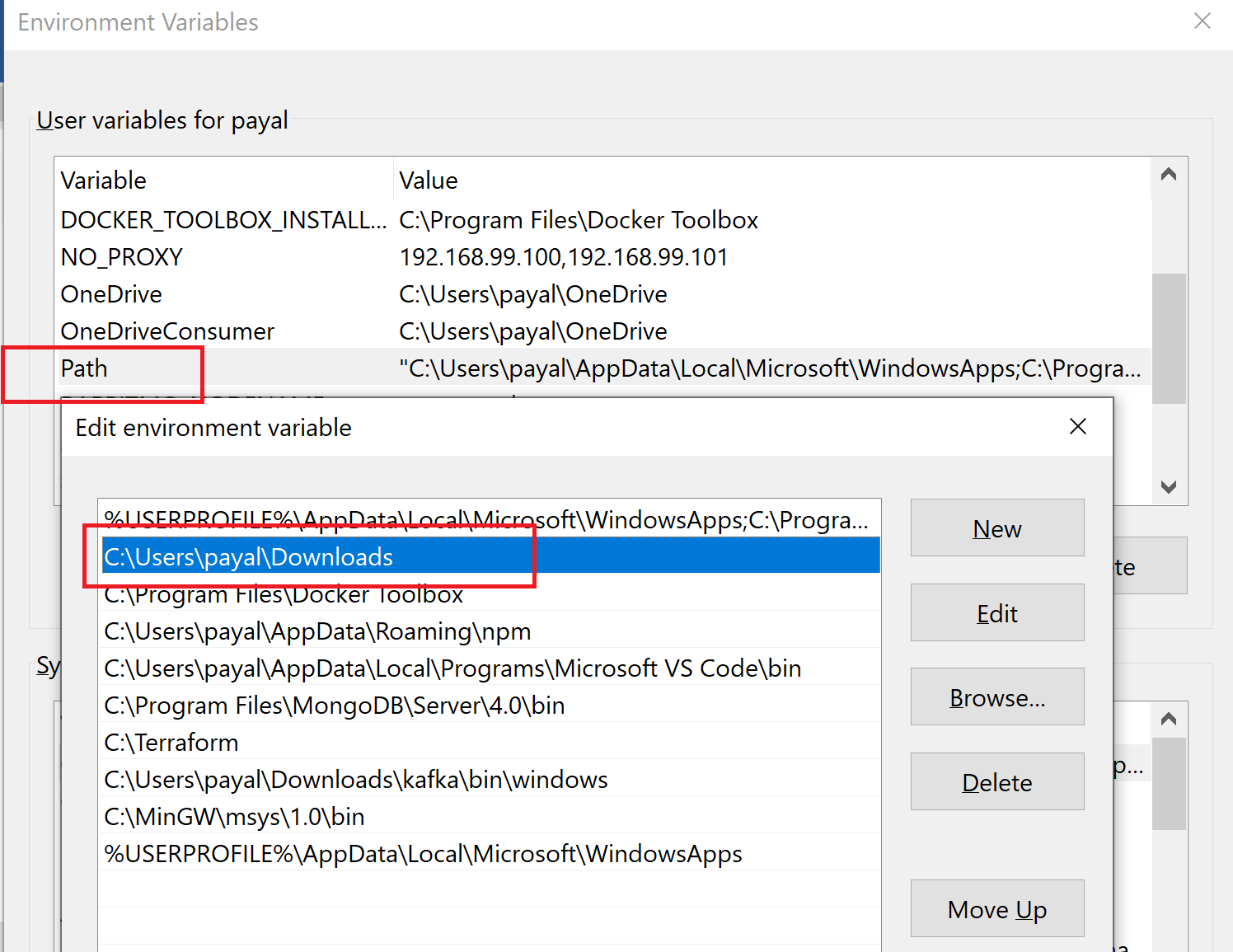
If you see the following output, your system already has a Hypervisor installed and you can skip the next step.

Hyper-V Requirements: A hypervisor has been detected. Features required for Hyper-V will not be displayed.

1. **Download kubectl file :**

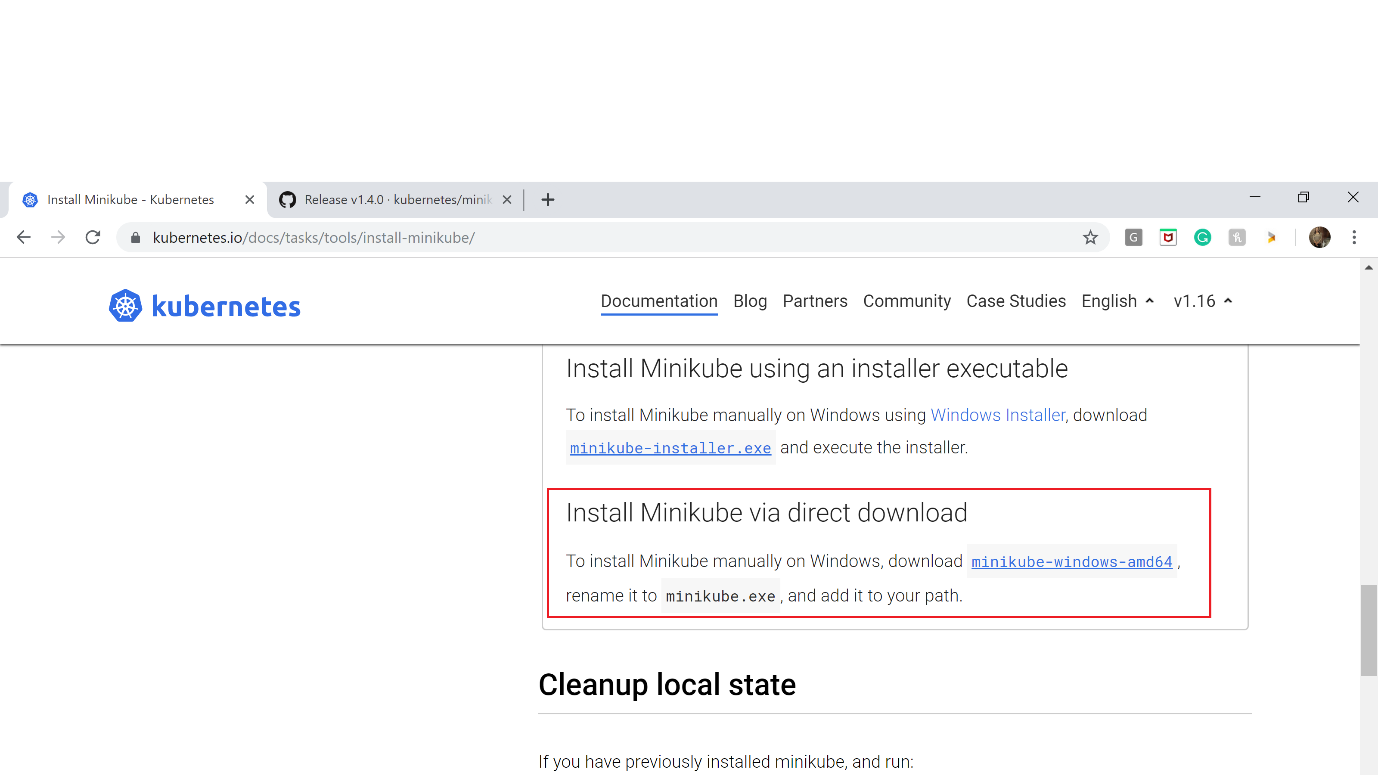
****

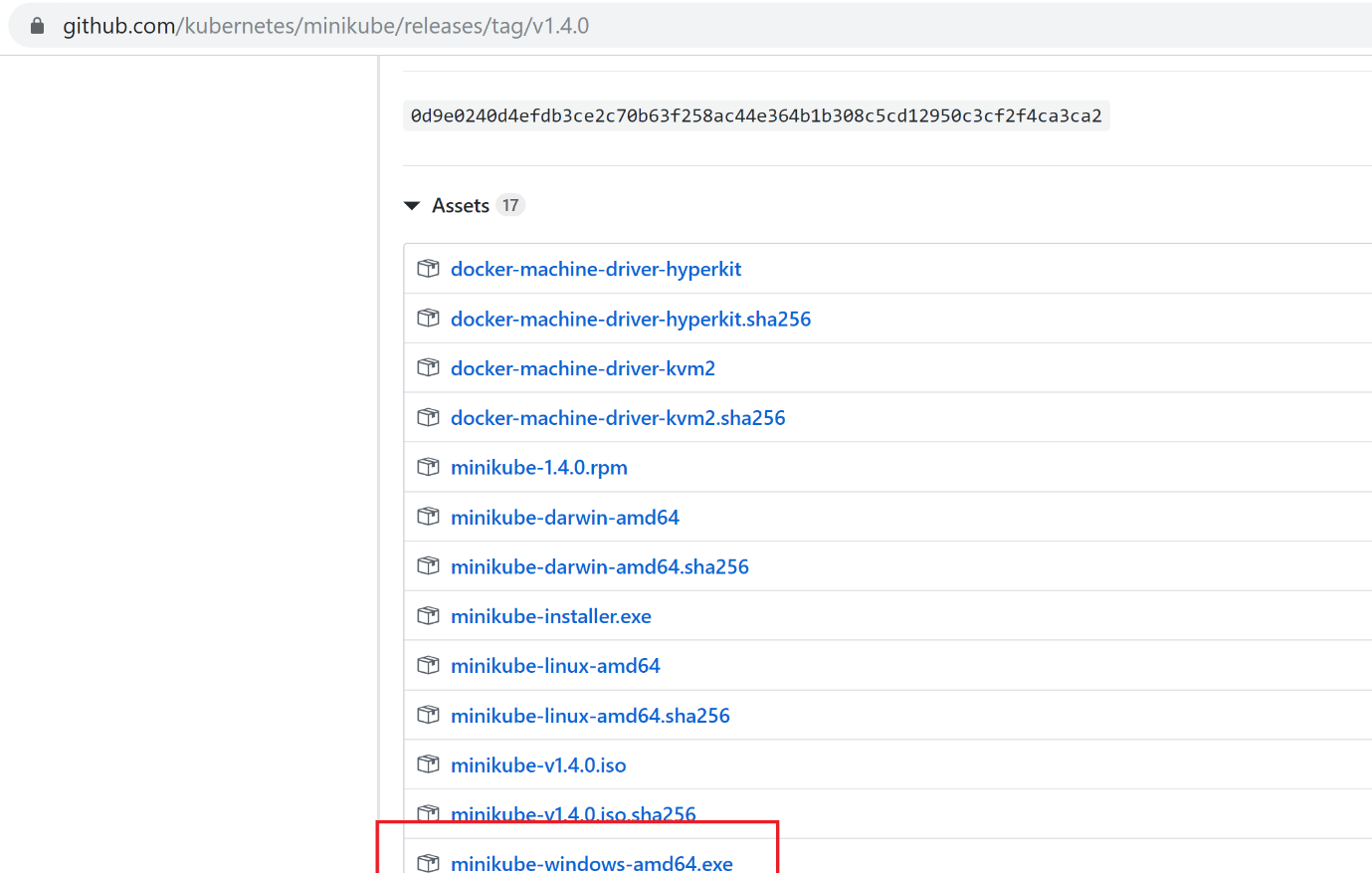
Set the Path variable to include the folder where you have downloaded this file, in our case it I downloads folder.



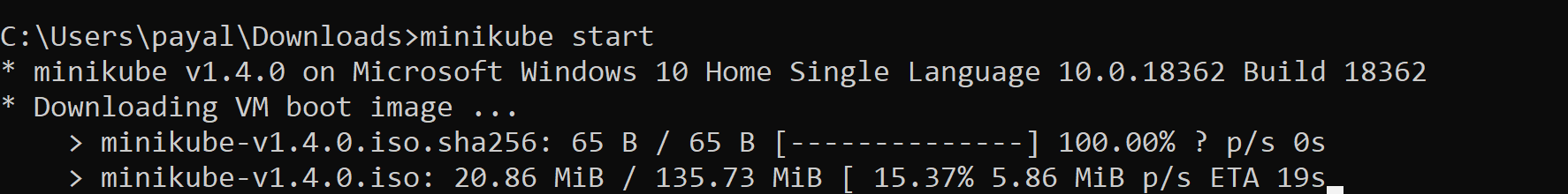
1. **Install Minikube:**

<https://kubernetes.io/docs/tasks/tools/install-minikube/>

****

****Rename it to minikube.exe.

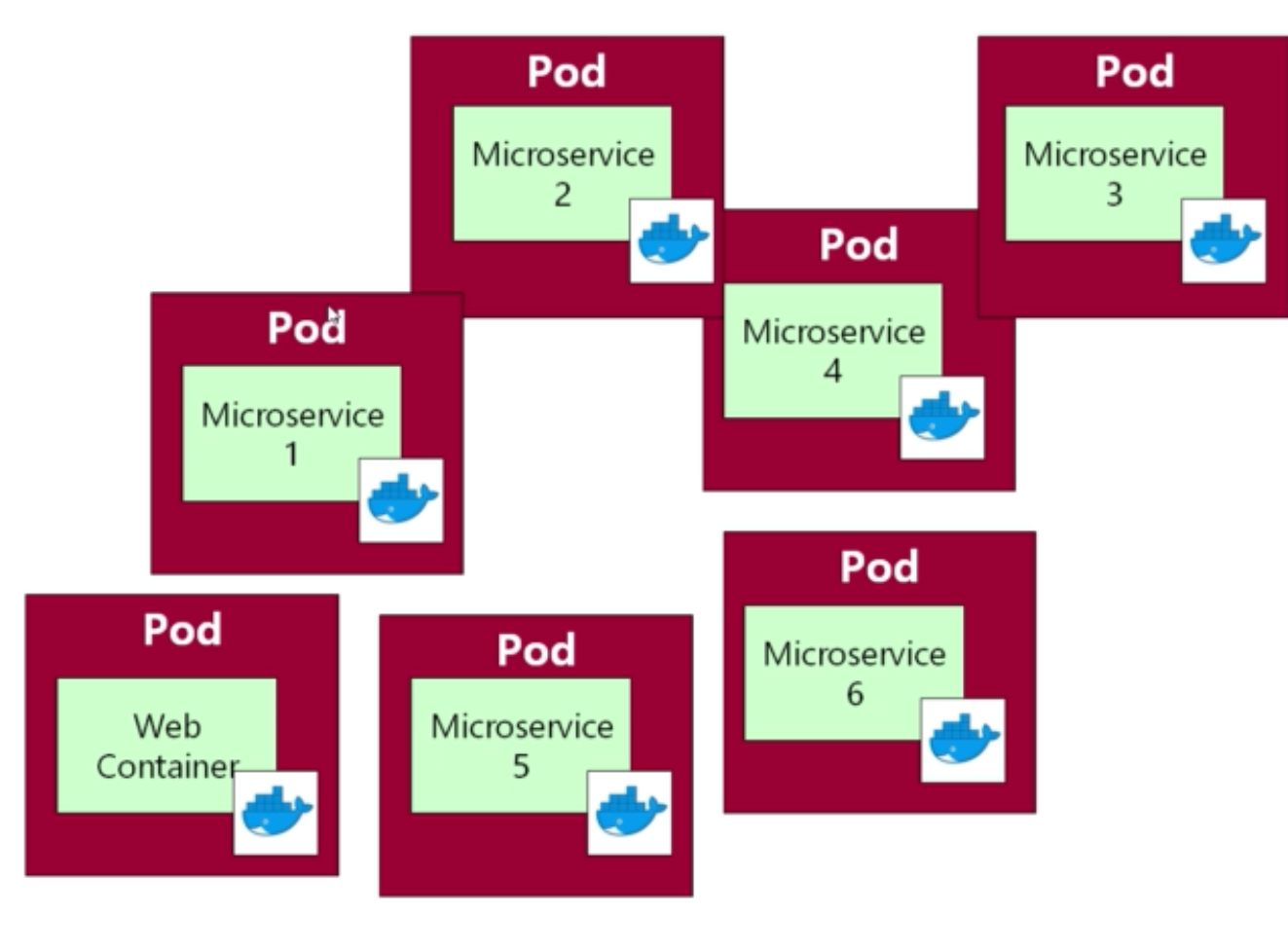
1. minikube start

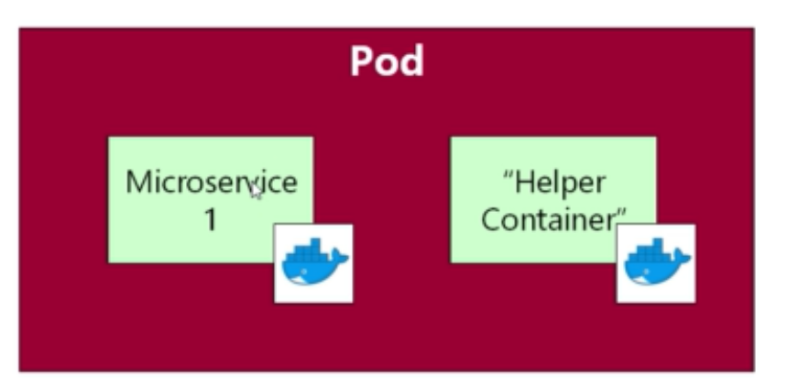


1. kubectl version: should show both client version and the server version.

**Pods:**

Group of one or more containers with shared storage/network and a spec on how to run the containers. For every container we wantto deploy, we create a pod, which is like a wrap-up for a container. WE can have more than 1 container inside a pod.





Like we may need a secondary container to do some processing in background. Like mongodb may use side-car containers.

Each pod must implement a single service. So Pod is the basic unit of deployment.

**Writing first pod:**

richardchesterwood/K8s-fleetman-webapp-angular: Tag: release0

We want to deploy this image and want Kubernetes to manage it so will create a pod for it.

<https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.16/#pod-v1-core>

This link tells how to create a pod.

apiVersion: v1

kind: Pod

metadata:

name: pod-example

spec:

containers:

- name: ubuntu

image: ubuntu:trusty

command: ["echo"]

args: ["Hello World"]

Containers we need in a pod, optionally we can specify command line arguments (args) and command.

Check is minikube is running: minikube status

**First pod:**

apiVersion: v1

kind: Pod

metadata:

name: webapp

spec:

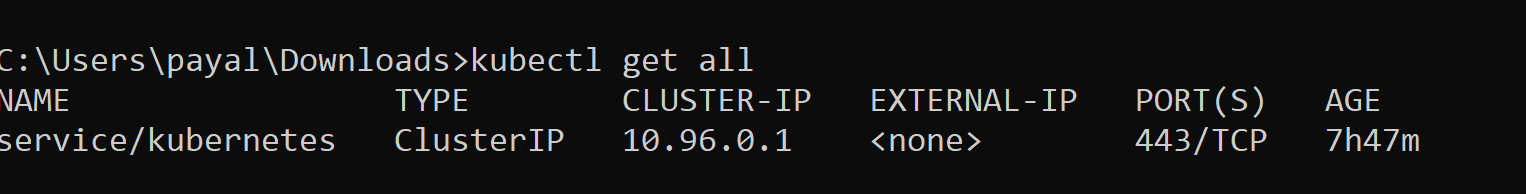
containers:

- name: webapp

image: richardchesterwood/K8s-fleetman-webapp-angular:release0

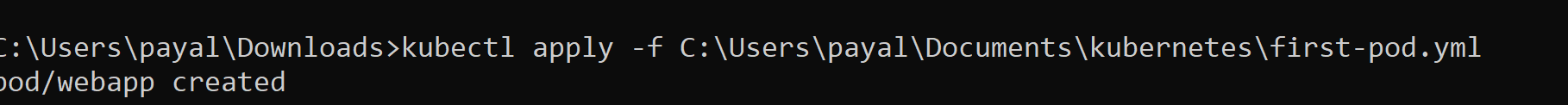
**Running pod:**

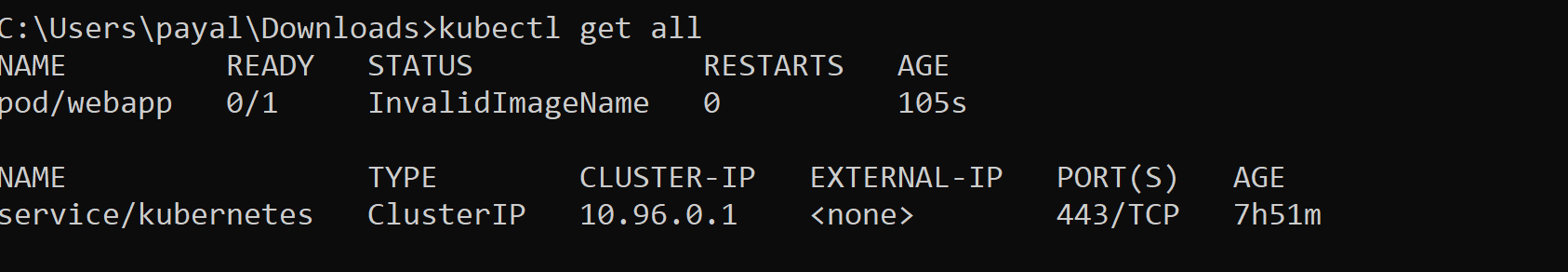
Kubectl get all: shows everything defined in Kubernetes cluster (pods)



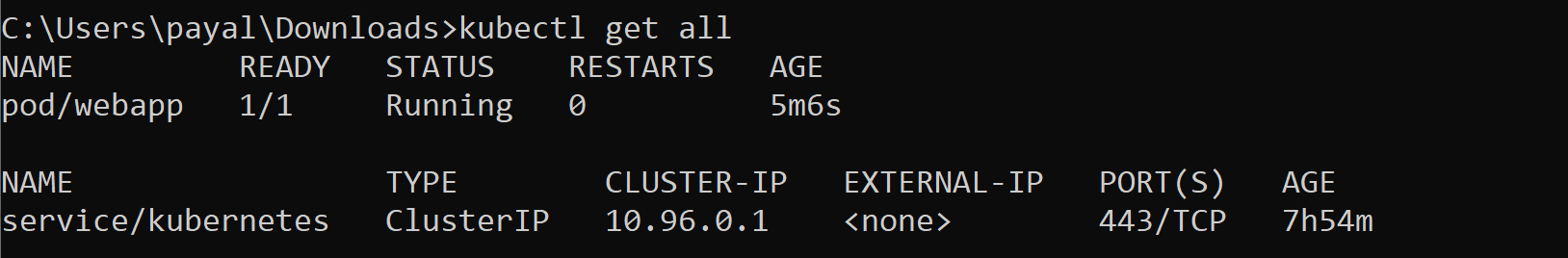
Rest service exposed by Kubernetes.

1. To deploy a pod to the cluster:





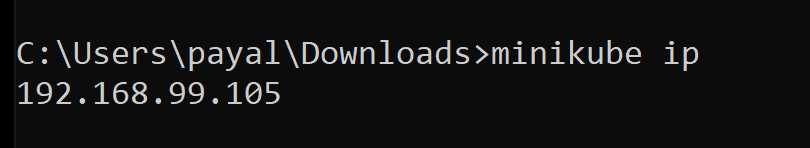
Currently not running.



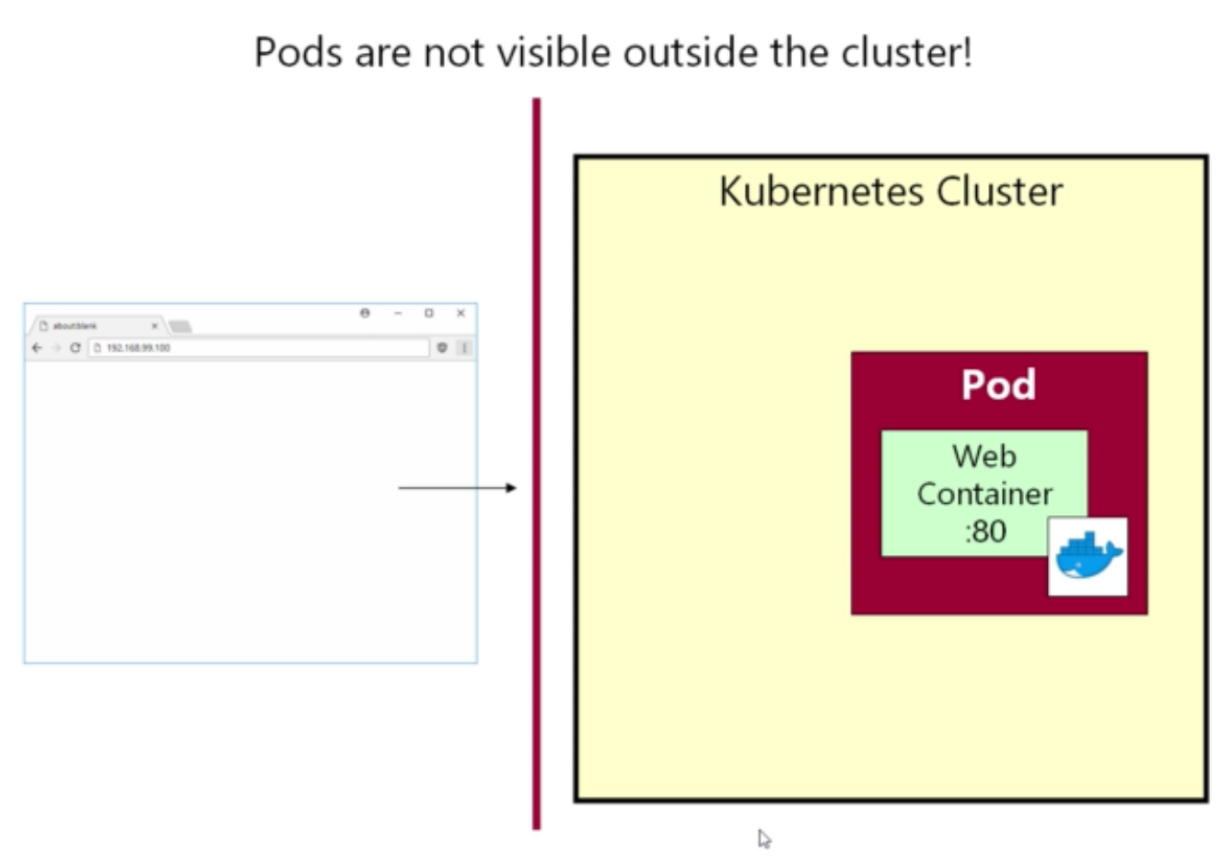
Now I running state.

1. Get the ip where minikube cluster is running Virtual machine

Minikube ip



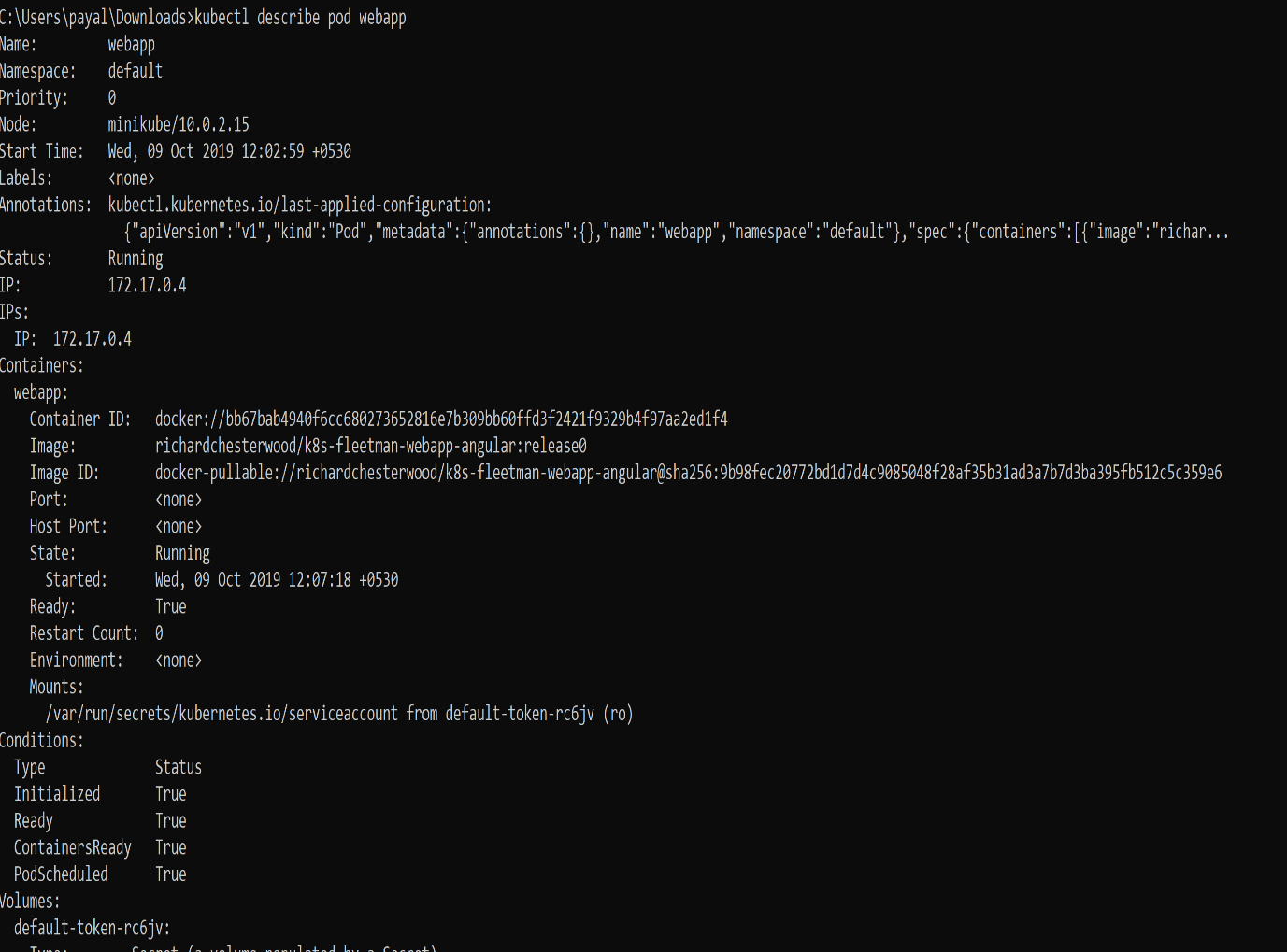
Pods are not visible outside the cluster. So in a web browser if you type in this ip, you will get “Site can’t be reached”.



Pods are only visible inside the cluster.

1. To get more information about the pod:

Kubectl describe pod <podname>



1. To execute command on the pod:

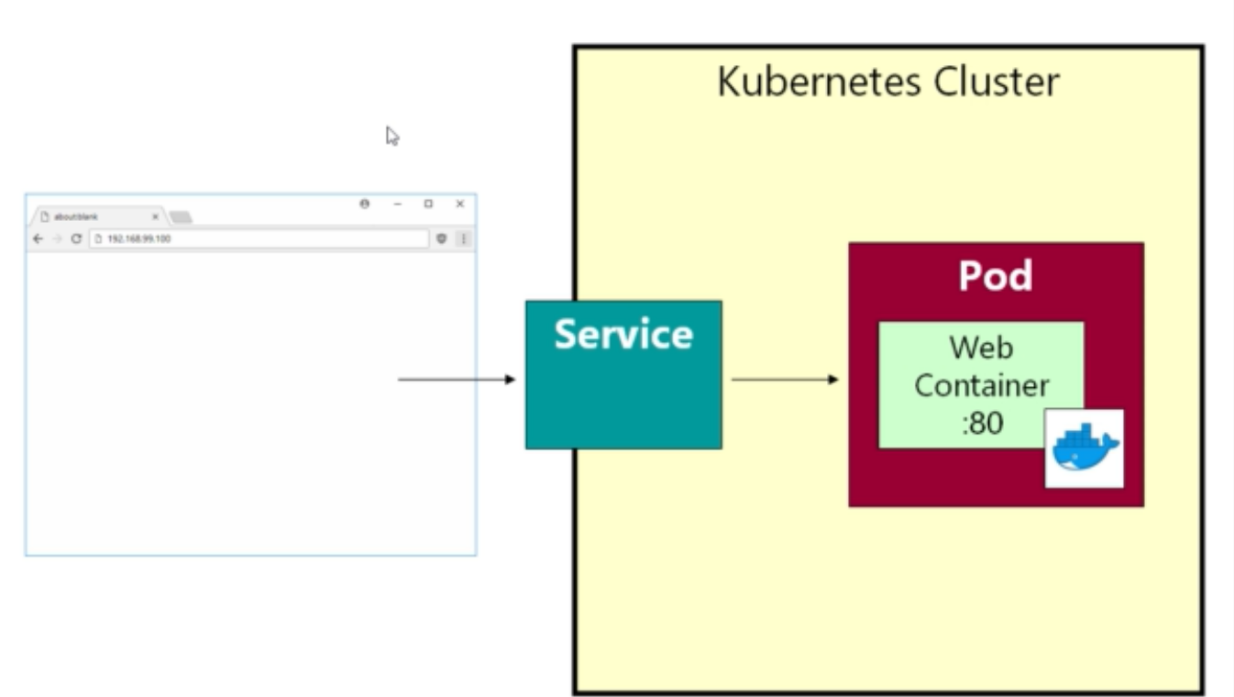
Kubectl exec <podname> <command>

Eg: kubectl exec webapp ls



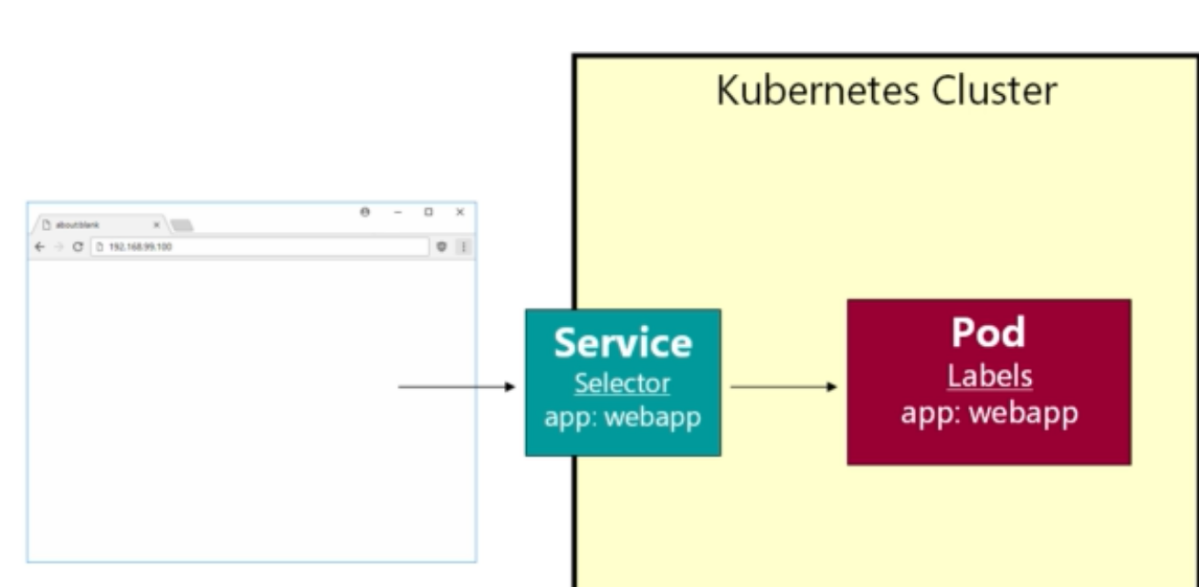
Accessing pod service inside the container..

Pods have short lifetime and they regularly die. Service is a long running object in kubernetes. Service will have ip address and fixed port. Can attach service to Pods.



**Pod Label:**

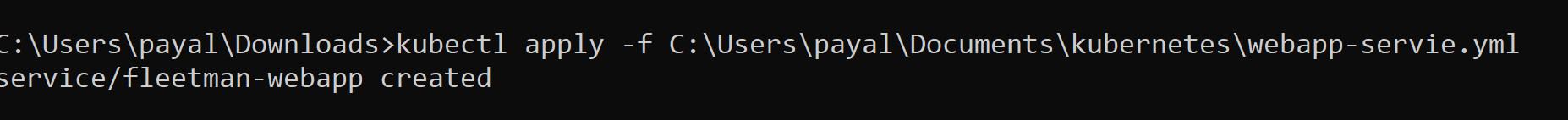
With a pod, we can set-up a series of key-value pairs. Like app:webapp as a key value pair. When we write a service, we provide a selector, which itself is like a series of key value pair. Service will look for any matching key-value pairs amongst the pods. If matches, will select that pod.

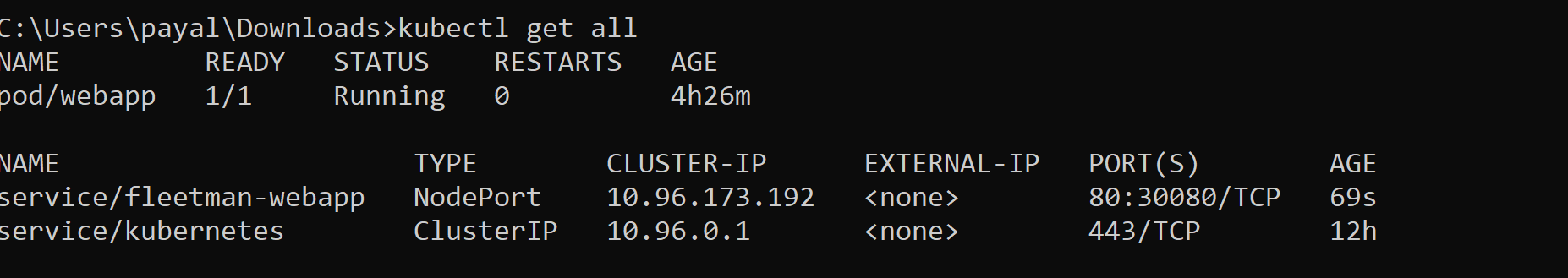


1. Create service to get networking endpoint for pod:

Kubectl get all

Create webapp-service.yml file





Minikube ip: try to open that ip on web browser: <http://192.168.99.105:30080/>. Doesn’t work yet.

Need to add label to a Pod:

**Webapp-servie.yml**

kind: Service

apiVersion: v1

metadata:

# Unique key of the Service instance

name: fleetman-webapp

spec:

ports:

# Accept traffic sent to port 80

- name: http

#accept traffic on port 80 from service

port: 80

#forward traffic to port 80 on containe

targetPort: 80

nodePort: 30080

#port available to browser >30000

selector:

# Loadbalance traffic across Pods matching

# this label selector. This selector will select all pods with key app and alue webapp

app: webapp

# Create an HA proxy in the cloud provider

# with an External IP address - \*Only supported

# by some cloud providers\*

#type: LoadBalancer

#type: ClusterIP

#implies service will be avaialable from only inside the cluster, acts as private service

type: NodePort

# available to whole Kubernetes cluster, expose a port through this node.

**Modify pod.yml to specify a label:**

apiVersion: v1

kind: Pod

metadata:

name: webapp

labels:

app: webapp

spec:

containers:

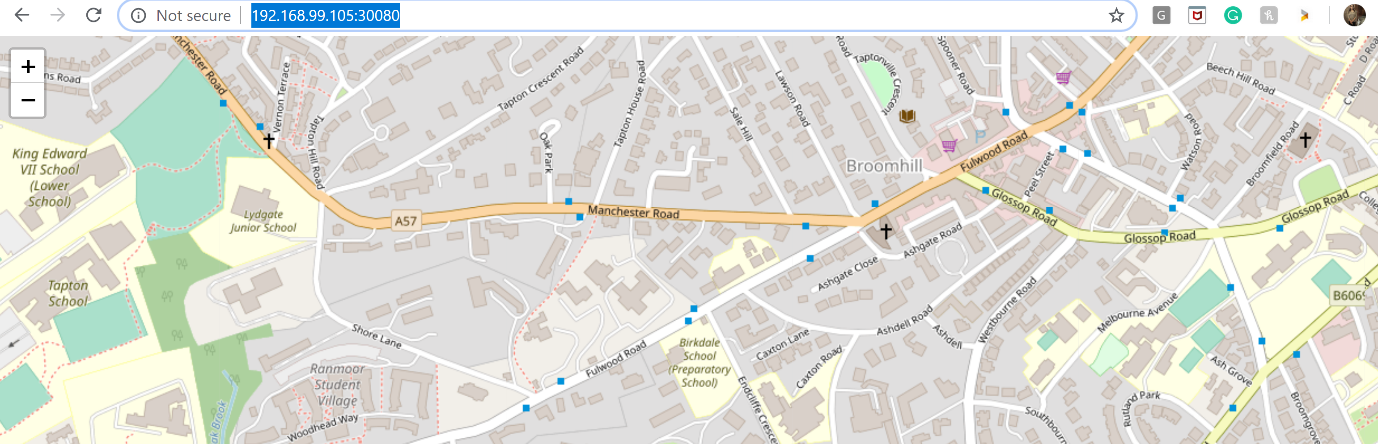
- name: webapp

image: richardchesterwood/k8s-fleetman-webapp-angular:release0

**Apply changes:**

kubectl apply -f C:\Users\payal\Documents\kubernetes\first-pod.yml

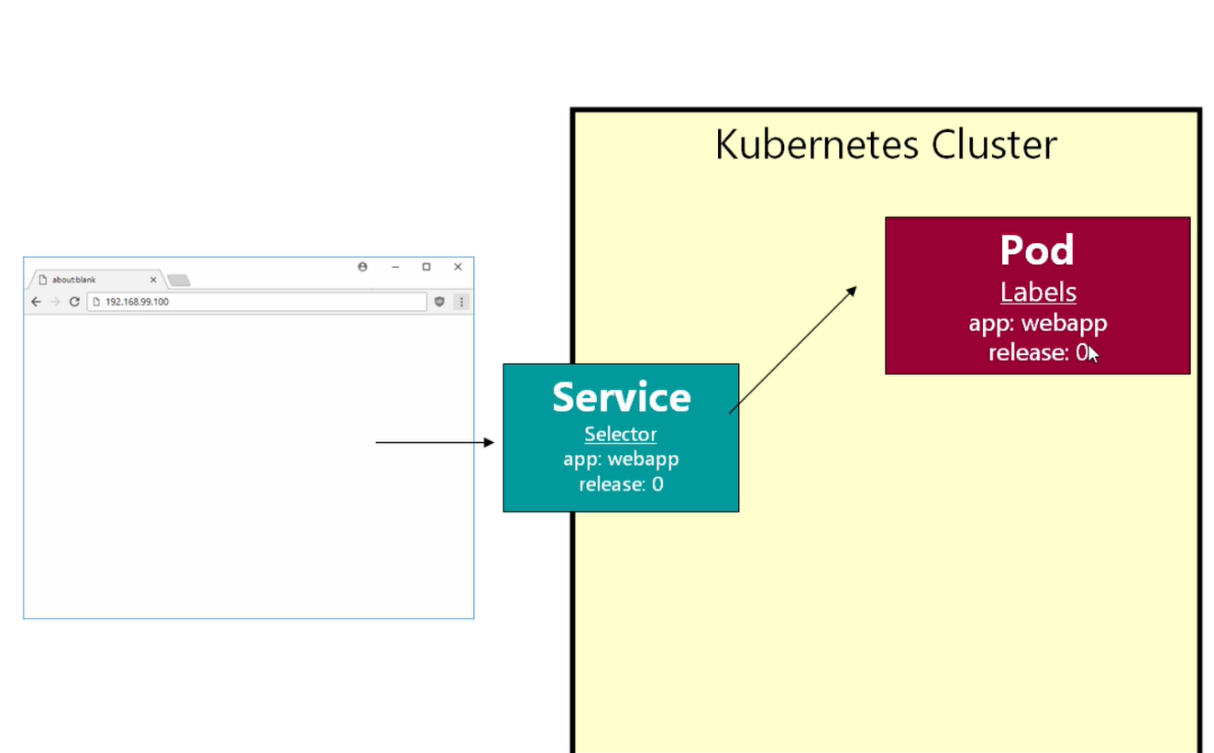
Now accessing app works <http://192.168.99.105:30080/>



**To deploy a new version for image:**

Don’t change image and apply kubectl. It will have some downtime.

To do with 0 downtime, add a new label



Does an and on the labels while matching.

First bring up another pod with a new version and then make the service point to new pod.

Make changes to pod.yml file:

apiVersion: v1

kind: Pod

metadata:

name: webapp

labels:

app: webapp

release: "0"

spec:

containers:

- name: webapp

image: richardchesterwood/k8s-fleetman-webapp-angular:release0

---

#3 dashes to define second pod

apiVersion: v1

kind: Pod

metadata:

name: webappv2

labels:

app: webapp

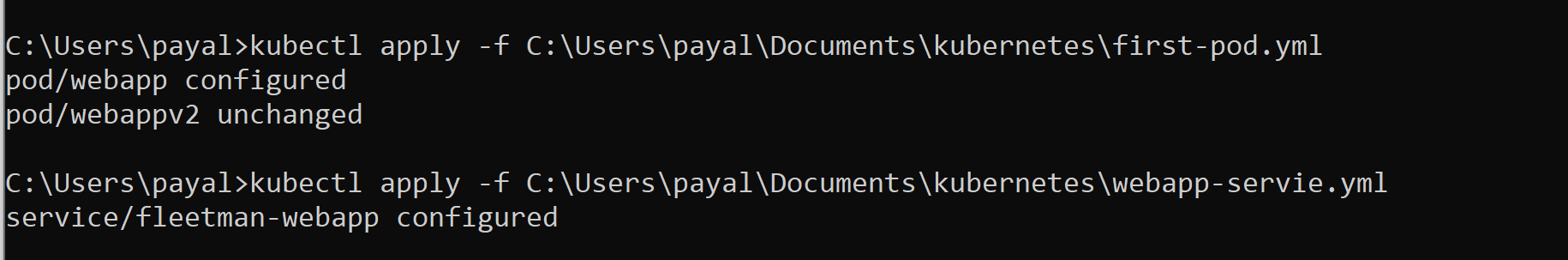
release: "0-5"

spec:

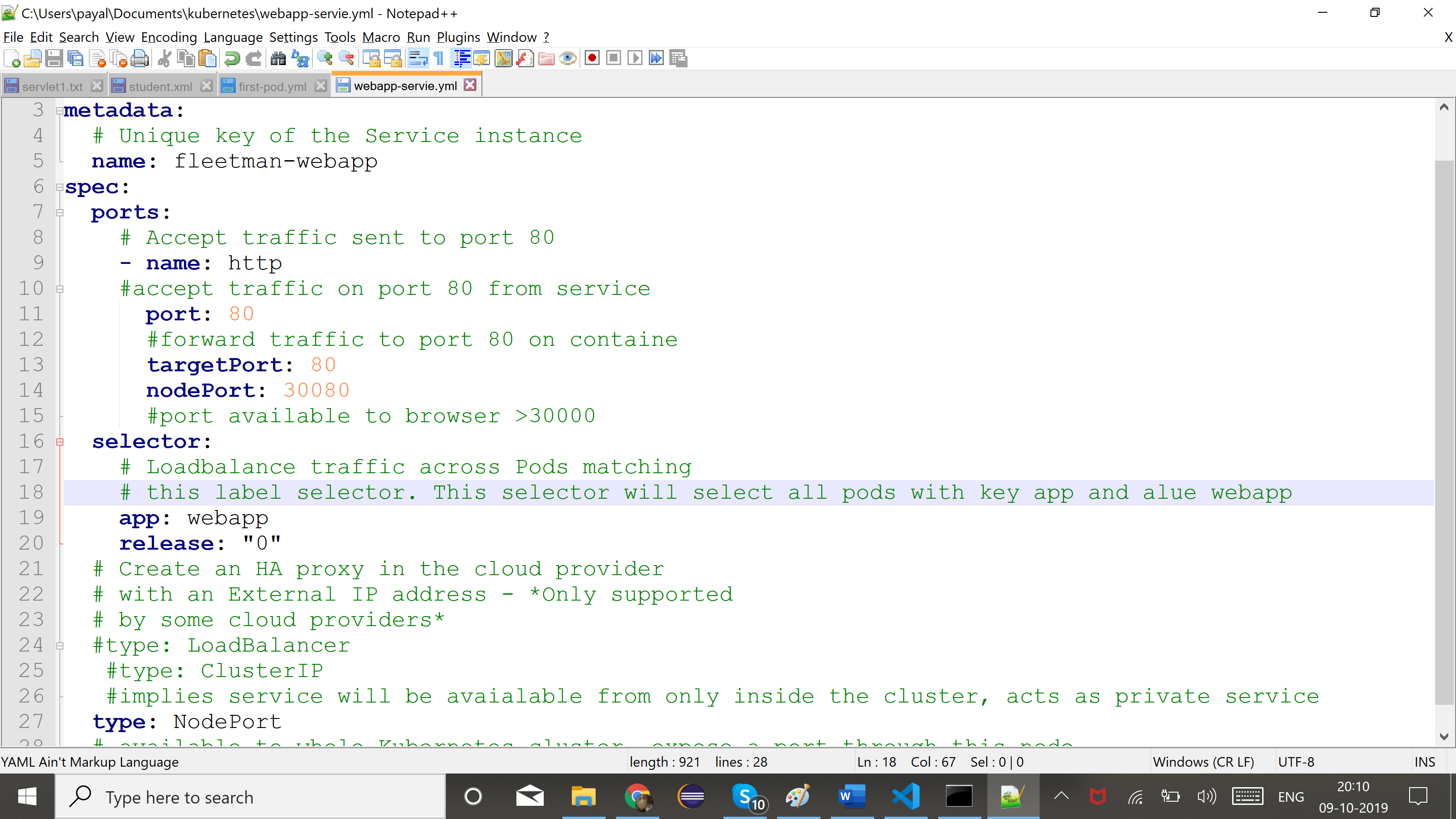
containers:

- name: webapp

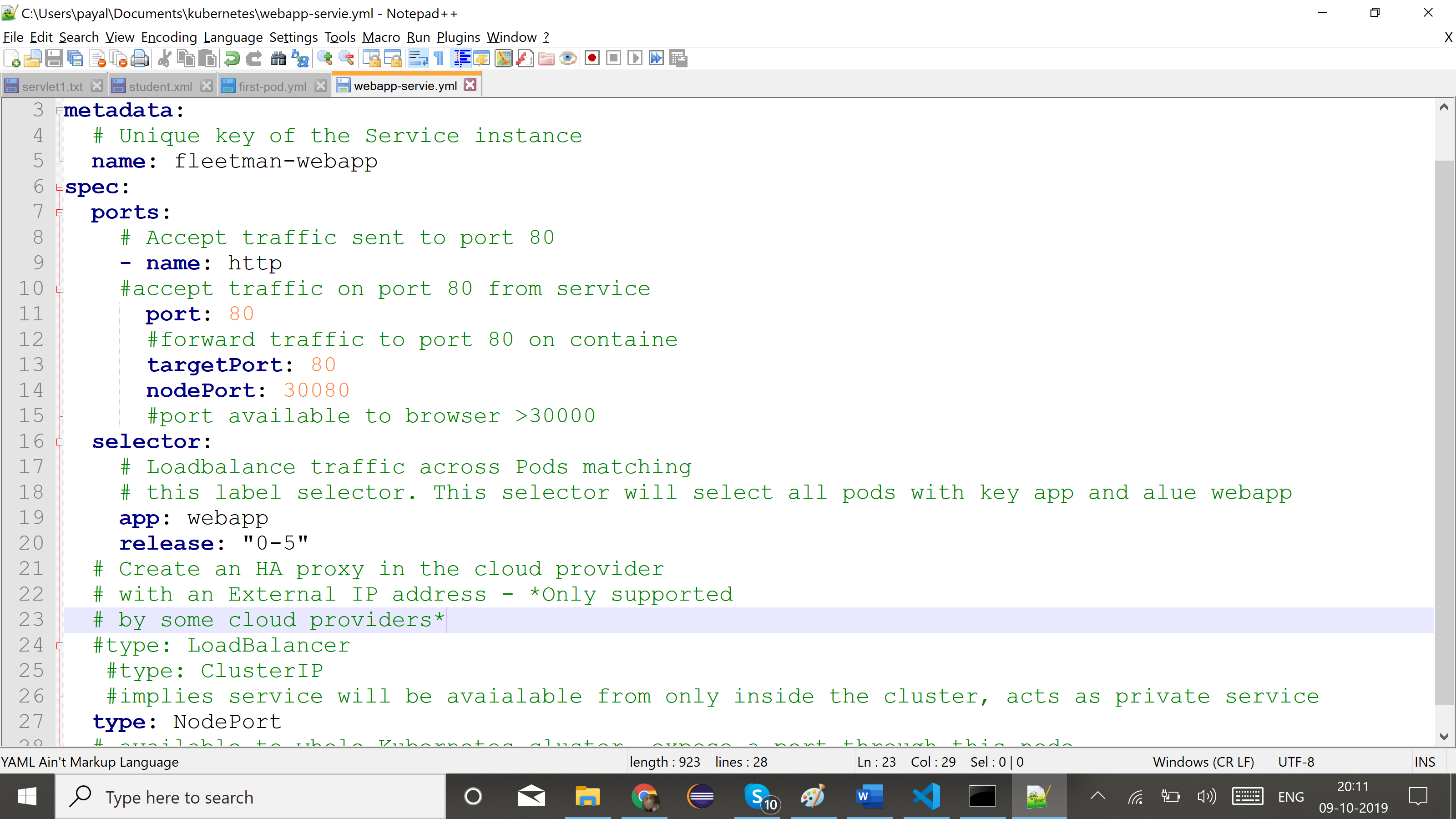
image: richardchesterwood/k8s-fleetman-webapp-angular:release0-5

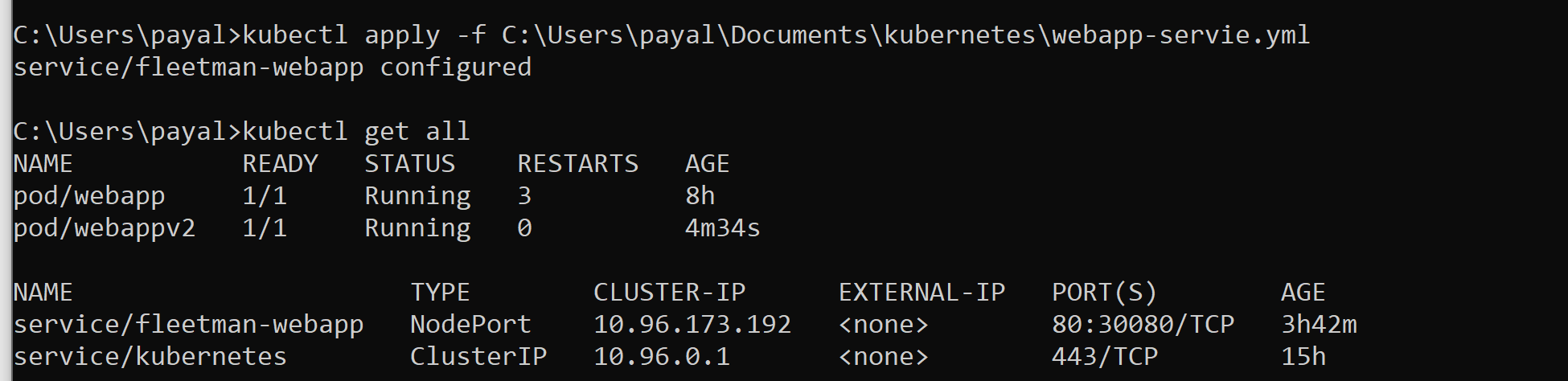


First service is pointing to release:0 label

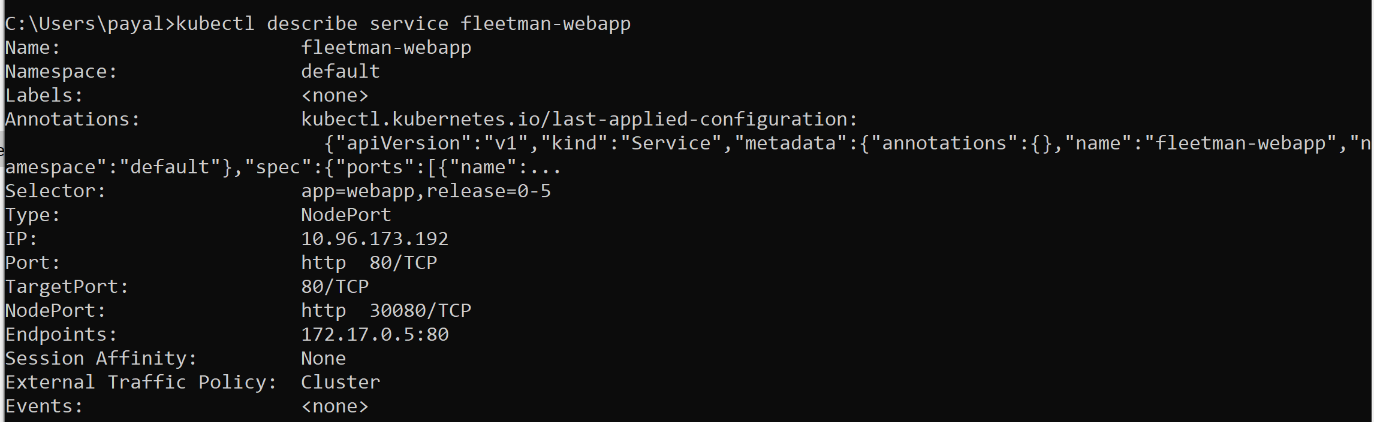


Then modify service.yml to release:0-5 when pod is up and running.





Kubectl describe service fleetman-webapp



Do ctrl+refresh on page to get new content. (Hard refresh)

Kubectl get pods: gets only the pods.

Kubectl get po : same as get pods.

Kubectl get po --show-labels

Kubectl get po --show-labels -l release=0

Kubectl apply -f .: apply all yaml files in the folder.