kubernetes

1. Overview Docker and Kubernates
2. Kubernates Architecture
3. Kubernates Cluster types
4. Kubernates types of Services
5. minikube and yml Setup (Not recommended)
6. EX 1: Create pod using yml
7. EX 2: Create pod using nodeport using yml
8. namespace
9. we can craete namespace by using yml file
10. we can craete namespace + pod by using yml file
11. k8S Resources - ReplicationController (RC)
12. k8S Resources - ReplicaSet
13. k8S Resources - Deployment in Kubernetes
14. EKS in AWS

kubernetes

Docker & Kubernetes Overview

What is Docker?

Docker is a free and open-source containerization software that allows you to package applications along with their dependencies into a single unit called a Docker Image. This image can run on any system that has Docker installed, making deployment easy and consistent across different environments.

Why use Docker?

✅ Portability – Run the same application on any machine, regardless of OS configuration.
✅ Dependency Management – Ensures that all required software (e.g., libraries, databases, and runtimes) is included within the image.
✅ Fast Deployment – No need to manually install dependencies every time you set up a new environment.
✅ Resource Efficiency – Uses fewer resources compared to traditional virtual machines.

How Docker Works?

Create a Docker Image – Package the app code + dependencies into a lightweight container image.

Run the Docker Container – Deploy this image as a container using the docker run command.

Execute Anywhere – The same image runs on any machine with Docker installed.

Once the image is built, it can run on any machine without requiring additional software setup.

Kubernetes (Container Orchestration Software)

What is Kubernetes?

Kubernetes (K8s) is a free and open-source orchestration tool developed by Google to manage containerized applications.

💡 Orchestration = Managing multiple containers efficiently

Kubernetes automates key tasks like:

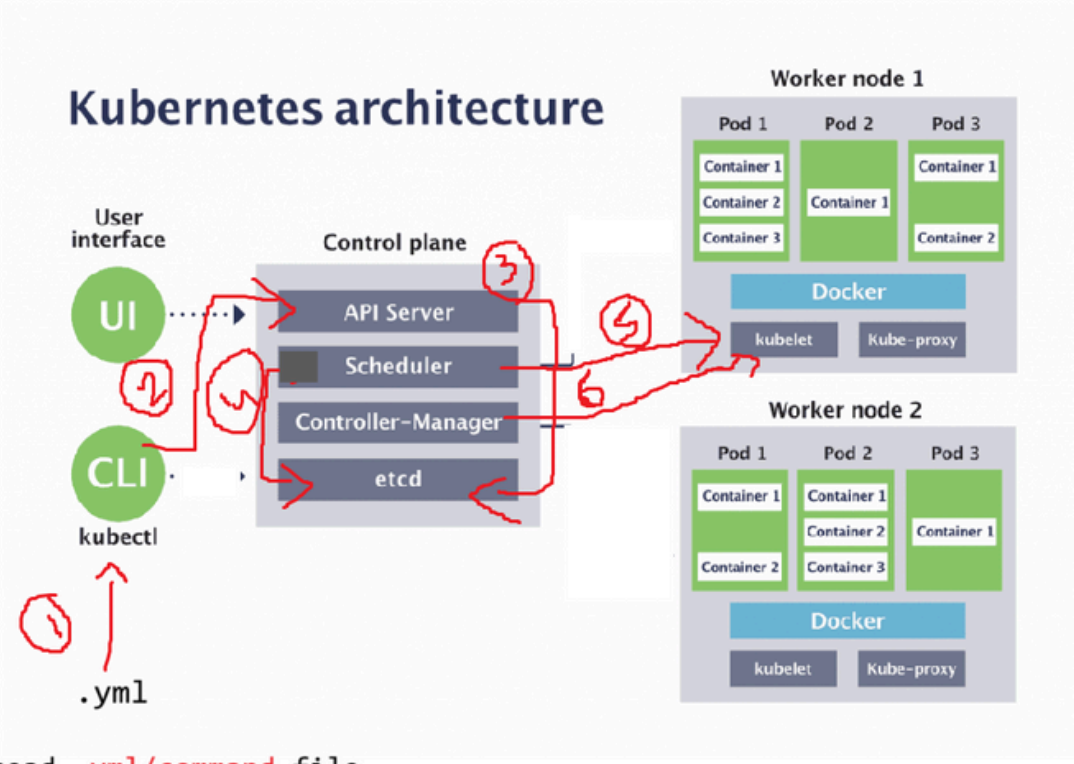
1. Creating, starting, and stopping containers.

2. Scaling up/down based on demand.

3. Handling failures automatically.

Why use Kubernetes?

✅ **Orchestration** – Efficiently manages multiple containers across a cluster of machines.
✅ **Self-Healing** – If a container crashes, Kubernetes automatically replaces it.
✅ **Load Balancing** – Distributes traffic across multiple containers to avoid overloading.
✅ **Auto Scaling** – Increases or decreases the number of running containers based on traffic load.
✅ **Automated Deployments** – Supports CI/CD for rolling updates and version control.



1. kubectl read .yml/command file
2. Then kubectl sends .yml details to API Server
3. Then API Server stores .yml details in etcd
4. Then Scheduler reads etcd
5. Scheduler& Controller-Manager it acts input kublet
6. Controller-Manager which will help us to do all pods and containers created or not it will check

Kubelet After input of 5 and 6 The Kubelet ensures the worker node is running the assigned workload

Kube Proxy The Kube Proxy manages networking for seamless cluster communication.

As we know The Controller Manager continuously monitors the cluster to ensure tasks run correctly.

Kubernetes Cluster Types

A Kubernetes Cluster is a group of servers working together to run containerized applications. It can be set up in two main ways:

A Kubernetes Cluster = Control Plane + Worker Nodes + Pods + Resources + Networking + Storage

1) Self-Managed Cluster
In this setup, we manually install and manage Kubernetes on our own infrastructure.

a) MiniKube (Single Node)
-> Runs a single-node cluster on a local machine.
-> Best for learning and practicing Kubernetes concepts.
-> Not suitable for production as it lacks high availability and scalability.

b) Kubeadm (Multi-Node)
-> A tool for setting up a multi-node cluster manually.
-> Requires configuring the control plane, worker nodes, and networking.
-> Gives full control over the cluster but requires deep Kubernetes expertise.
-> Used for on-premise or customized Kubernetes deployments.

2) Cloud Provider-Managed Cluster (Pre-configured, ready-to-use)
Cloud providers offer fully managed Kubernetes services, where they handle cluster maintenance, updates, and availability.

a) AWS EKS (Elastic Kubernetes Service)
-> A managed Kubernetes service on Amazon Web Services.

b) Azure AKS (Azure Kubernetes Service)
-> Microsoft Azure's managed Kubernetes offering.

c) GCP GKE (Google Kubernetes Engine)
-> Google Cloud's fully managed Kubernetes solution.

🚀 Kubernetes Services

-> A Kubernetes Service is used to expose a group of Pods so that they can be accessed reliably. Since Pods can be created and destroyed at any time (with changing IPs), a Service gives them a stable network identity.

🧭 Why Do We Use Services?

-> Pods are short-lived and can crash or restart.

-> Each time a Pod is created, it gets a new IP address.

-> Directly accessing Pods via IP is not reliable.

-> A Service gives a static IP to a group of Pods.

🌐 Types of Kubernetes Services

Kubernetes offers different types of services depending on how and where you want to expose your Pods:

🔹 1. ClusterIP (Default)
🔸 2. NodePort
🔹 3. LoadBalancer

🔐 **ClusterIP Service (Internal Access Only)**

📌 Key Points:

-> Pods are short-lived objects; if one crashes, Kubernetes replaces it with a new Pod.

-> Every new Pod gets a different IP address.

Note: 🛑 Never rely on Pod IPs to access an application.

-> A ClusterIP Service groups multiple Pods and assigns them a single static IP.

-> This static IP allows other components inside the cluster to access the group of Pods reliably, even when individual Pods change.

🚫 Access Scope:

-> Only accessible within the Kubernetes cluster.

-> Not reachable from the outside world (internet or external clients).

💡 Use Case:

-> Internal services such as databases, backend APIs, authentication services, etc.

Example: You don’t want to expose a database Pod to the internet, so you use a ClusterIP service to allow access only from other internal Pods.

🌐 **What is a NodePort Service in Kubernetes?**

-> A NodePort service is a type of Kubernetes Service that exposes your Pods outside the cluster using a port on each worker node (called a "NodePort").

🧭 **Why Use NodePort?**

By default, Pods and ClusterIP services are only accessible within the cluster.

NodePort makes them accessible externally by opening a static port (from 30000 to 32767) on each worker node.

It allows you to access your application using:

http://<NodeIP>:<NodePort>

Note: Here all traffic is routed to one worker node. Means load balancing cannot happen here.

🌐 **What is a LoadBalancer Service in Kubernetes?**

-> It not only provides external access to your app but also handles automatic traffic distribution across the backend Pods running on different worker nodes.

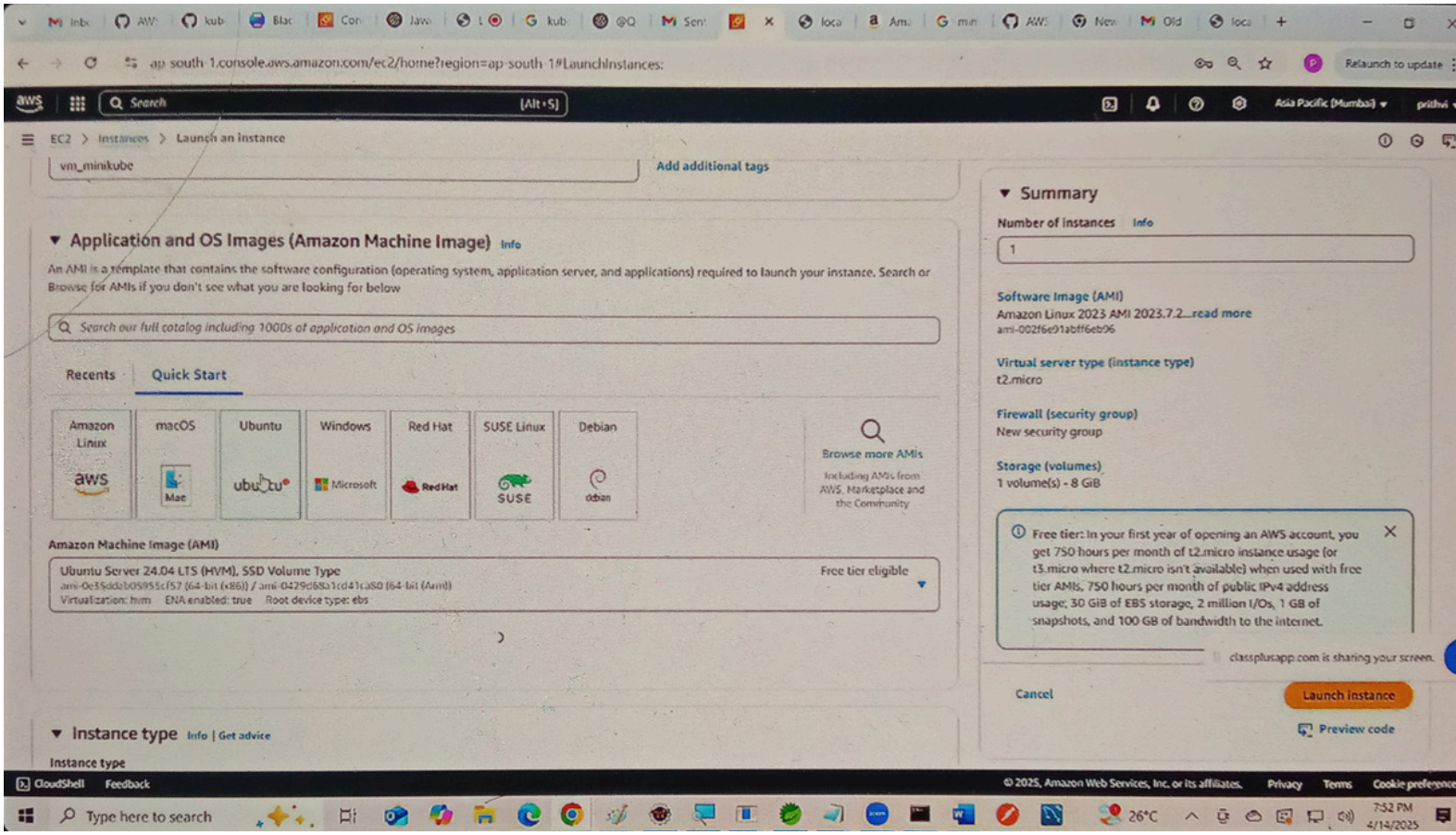
Step-1 : Setup Linux VM

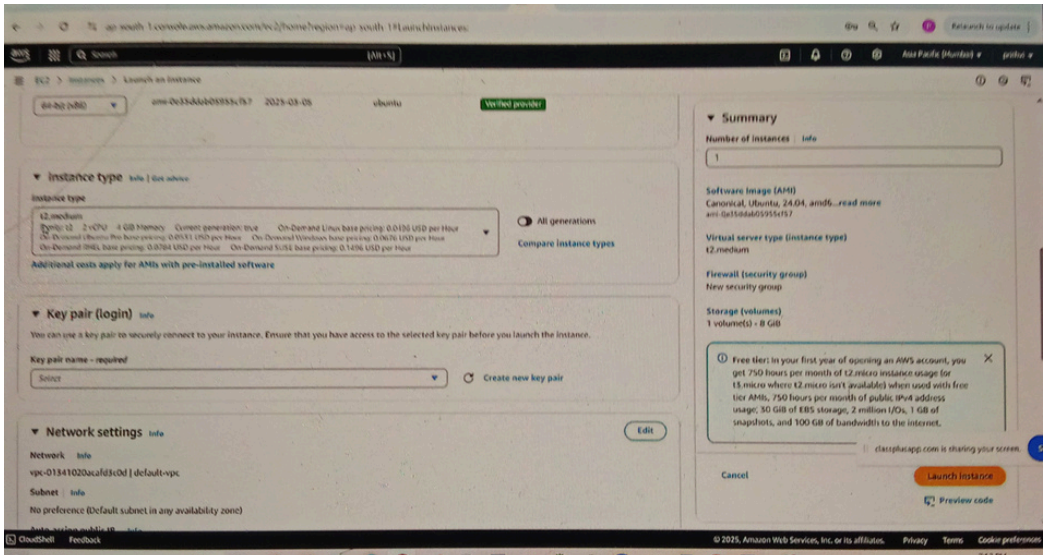
Login into AWS Cloud account

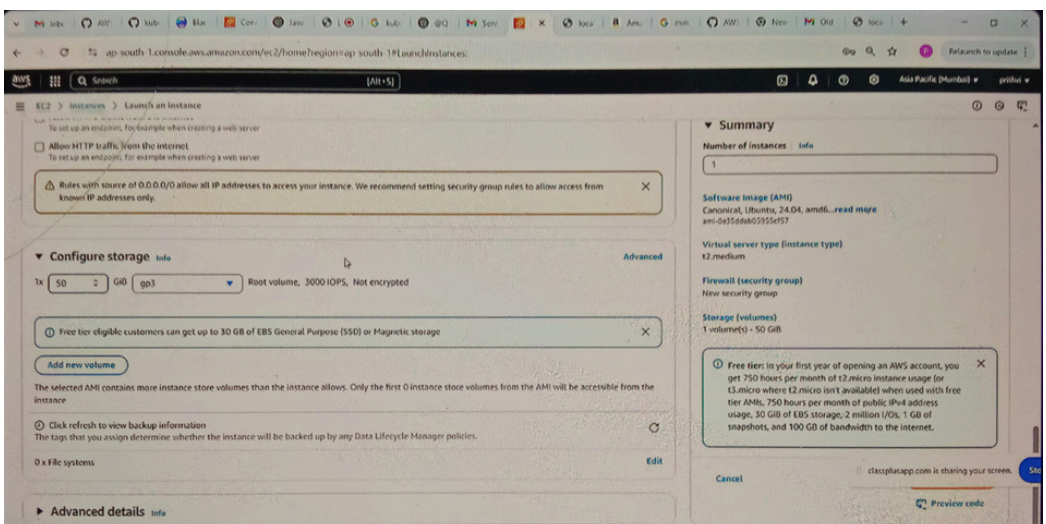
Create Linux VM with Ubuntu AMI (t2.medium or t3.medium)

Select Storage as 50 GB (Default is 8 GB only for Linux)

Create Linux VM and connect to it using SSH Client







click create/launch instance

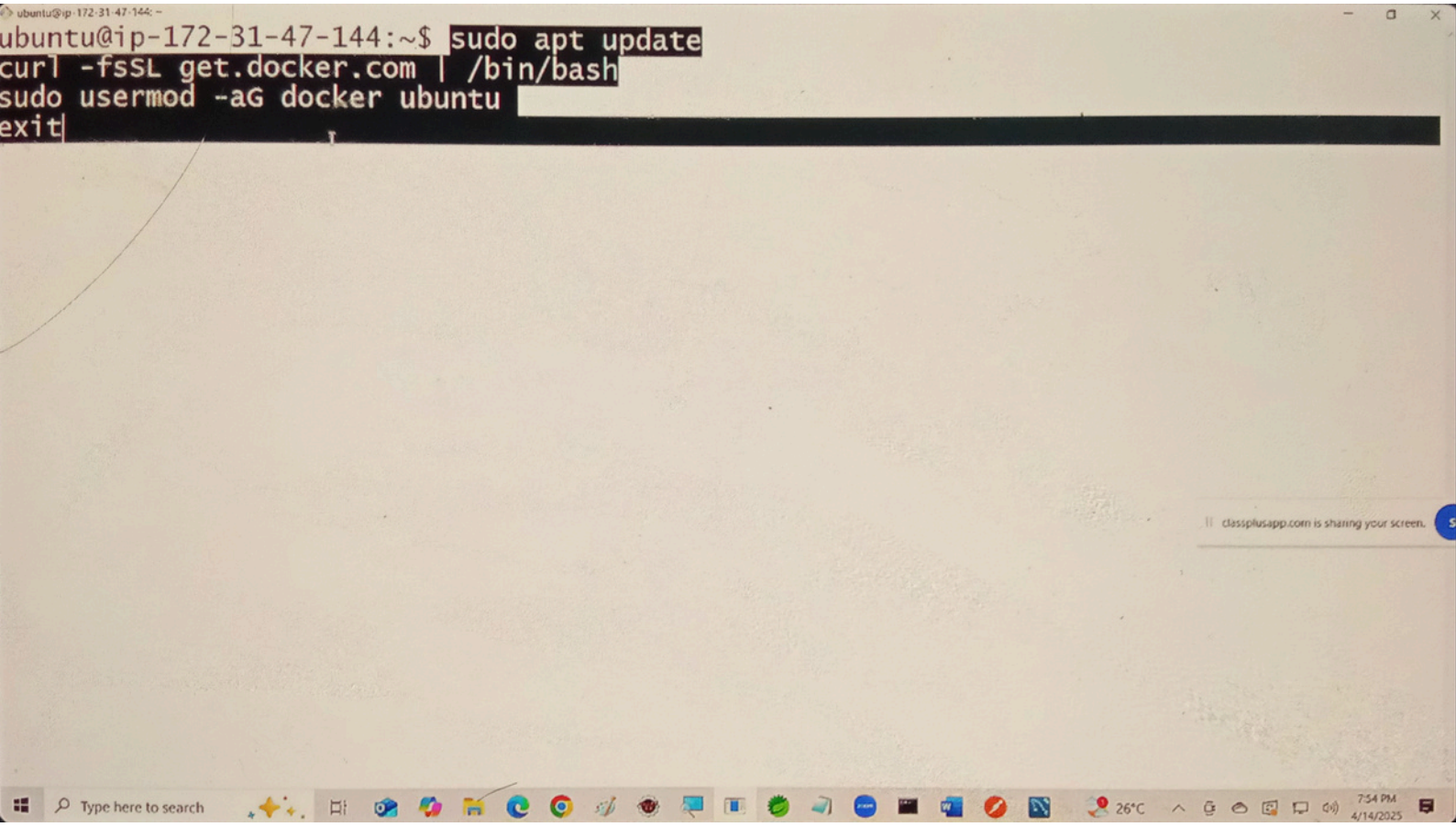
Step-2 : Install Docker In Ubuntu VM

sudo apt update

curl -fsSL get.docker.com | /bin/bash

sudo usermod -aG docker ubuntu

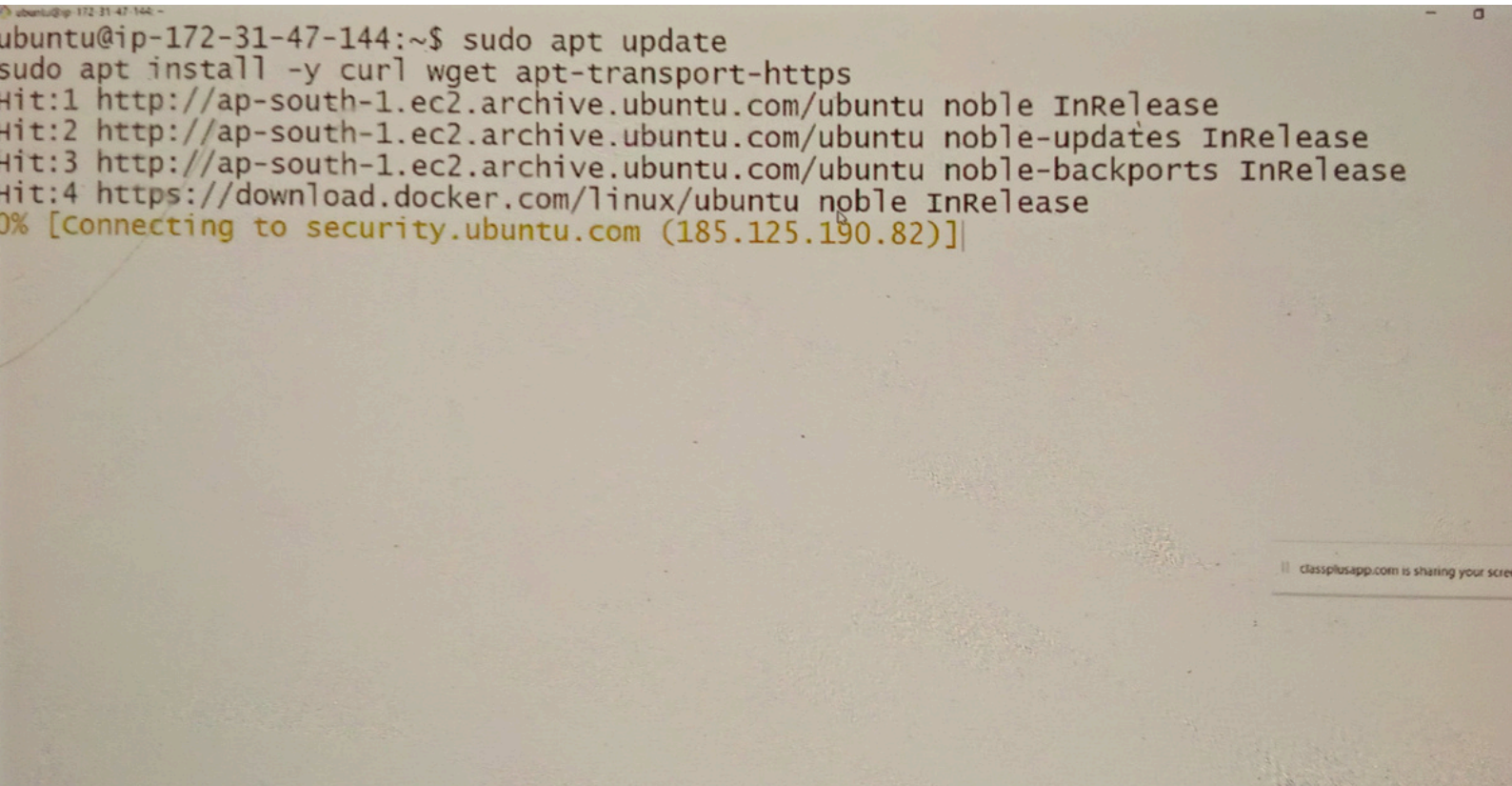
exit



Step-3 : Updating system packages and installing Minikube dependencies

sudo apt update

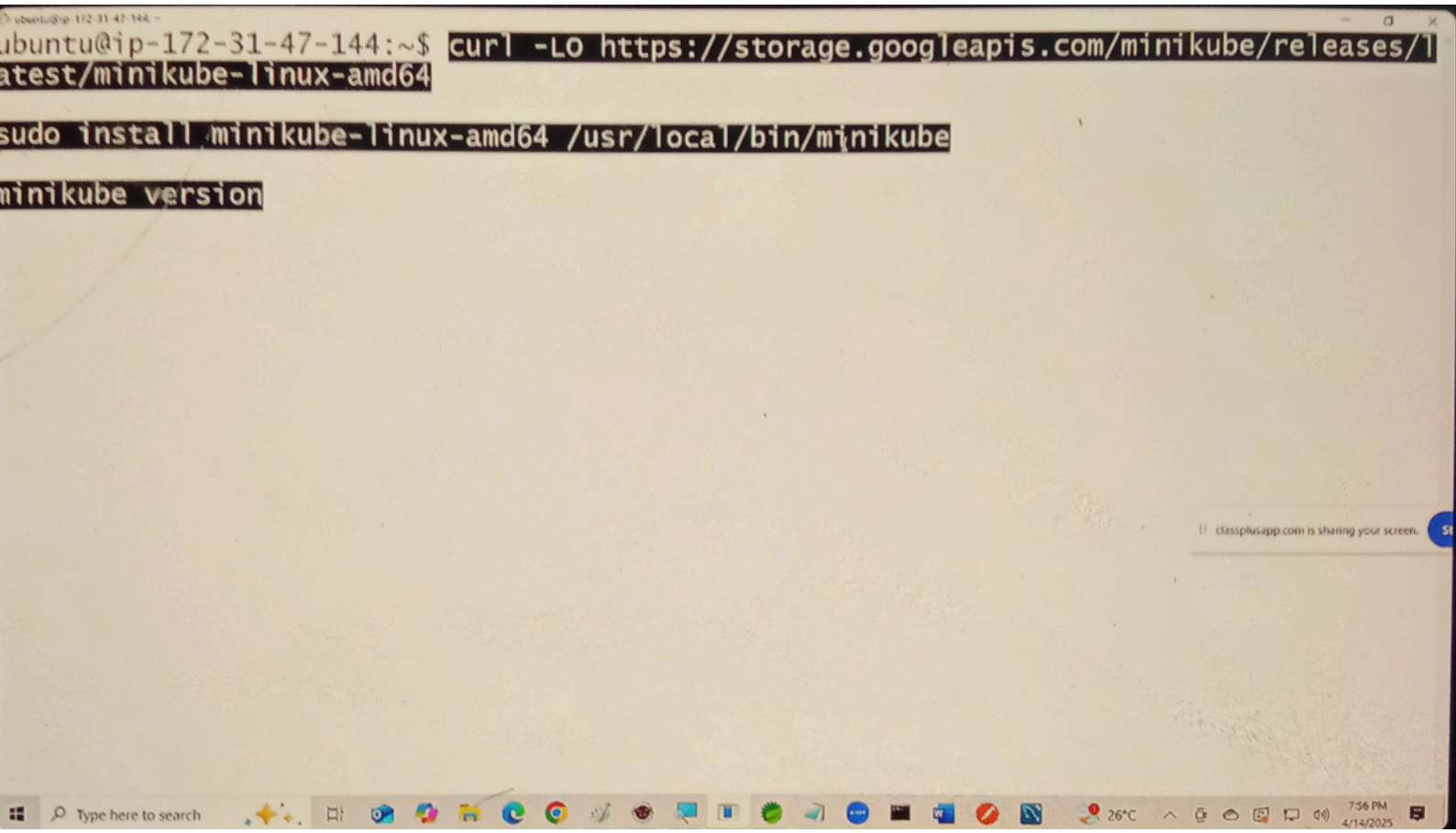
sudo apt install -y curl wget apt-transport-https



Step-4 : Installing Minikube
curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64

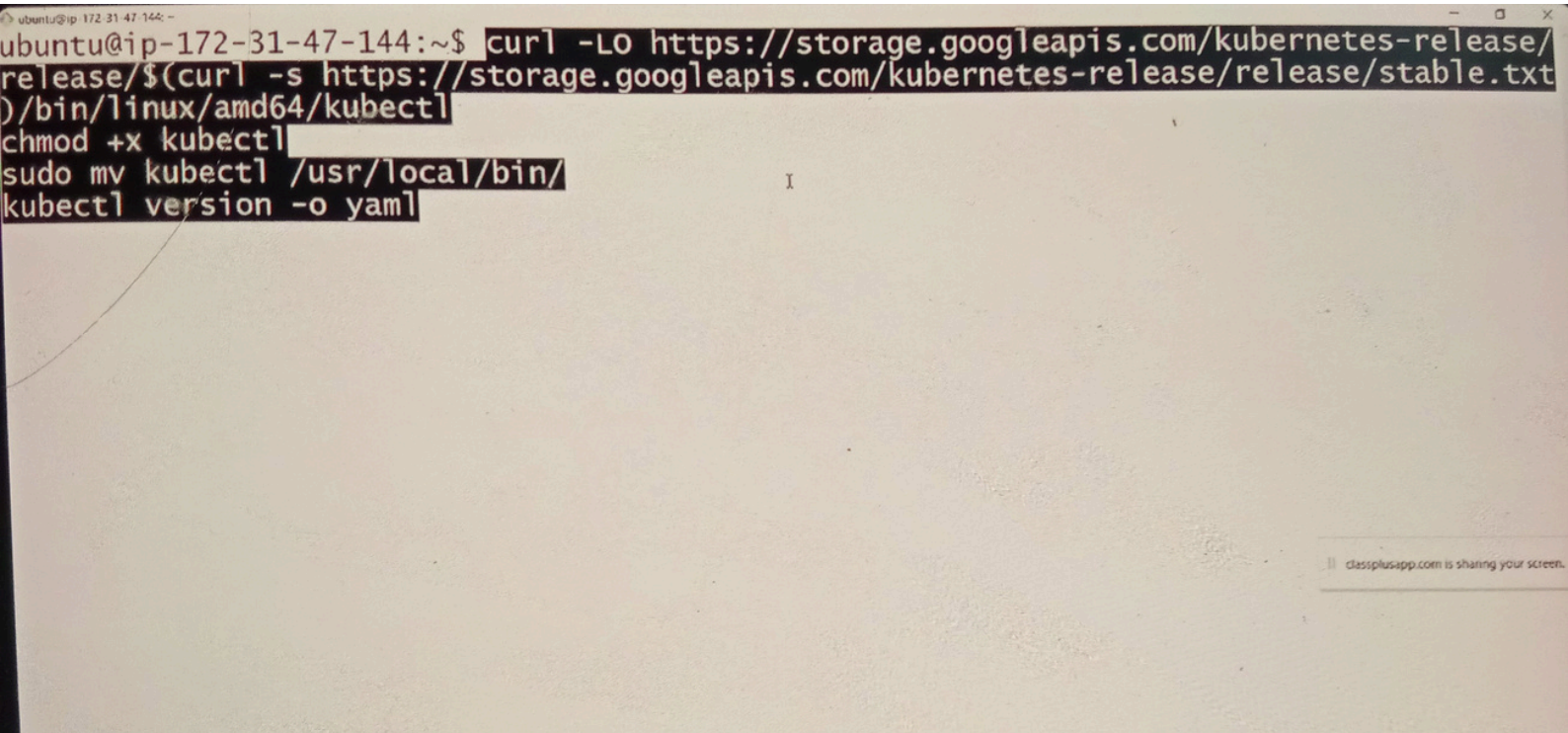
sudo install minikube-linux-amd64 /usr/local/bin/minikube

minikube version

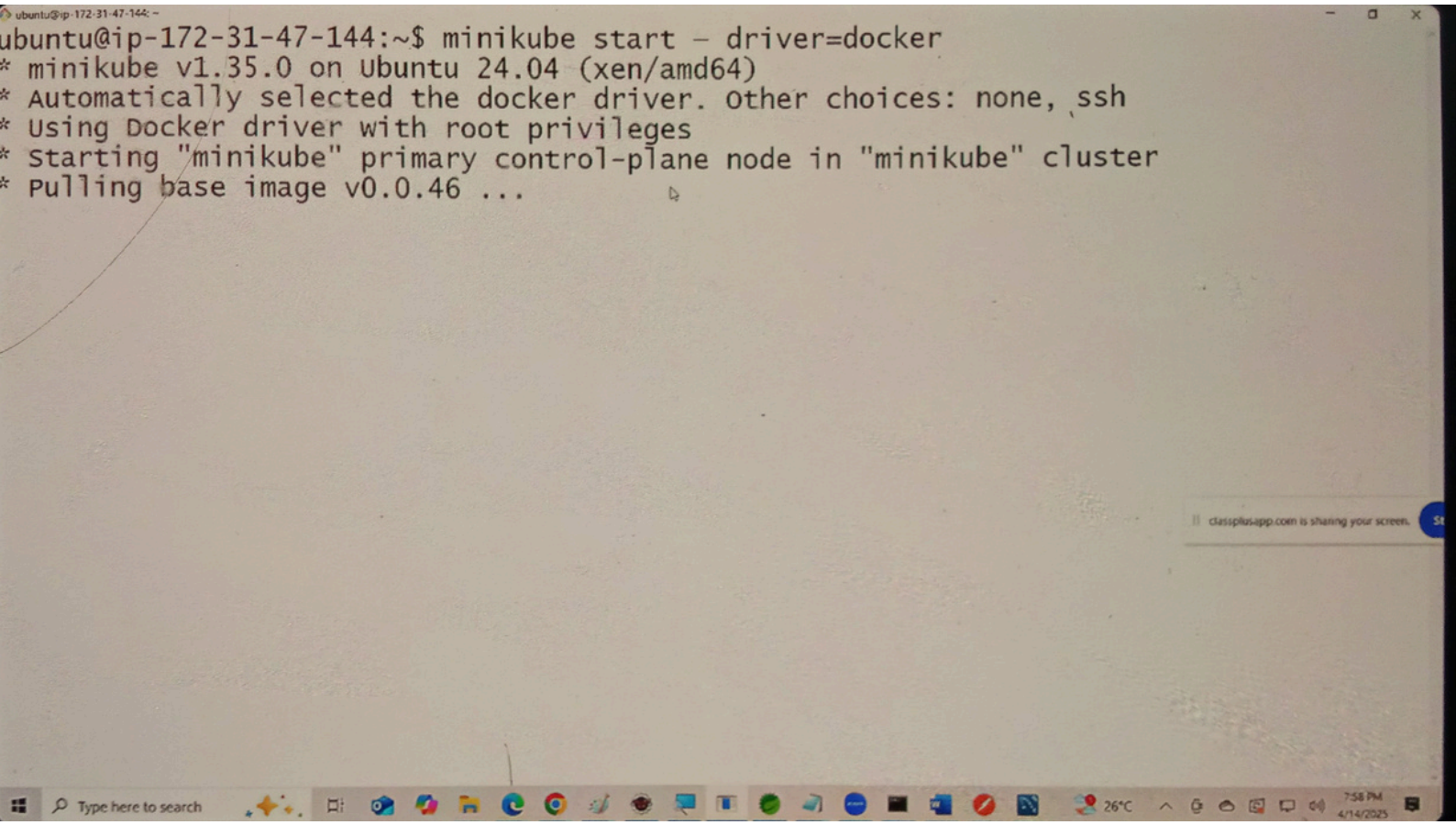


S
Step-5 : Install Kubectl (Kubernetes Client)

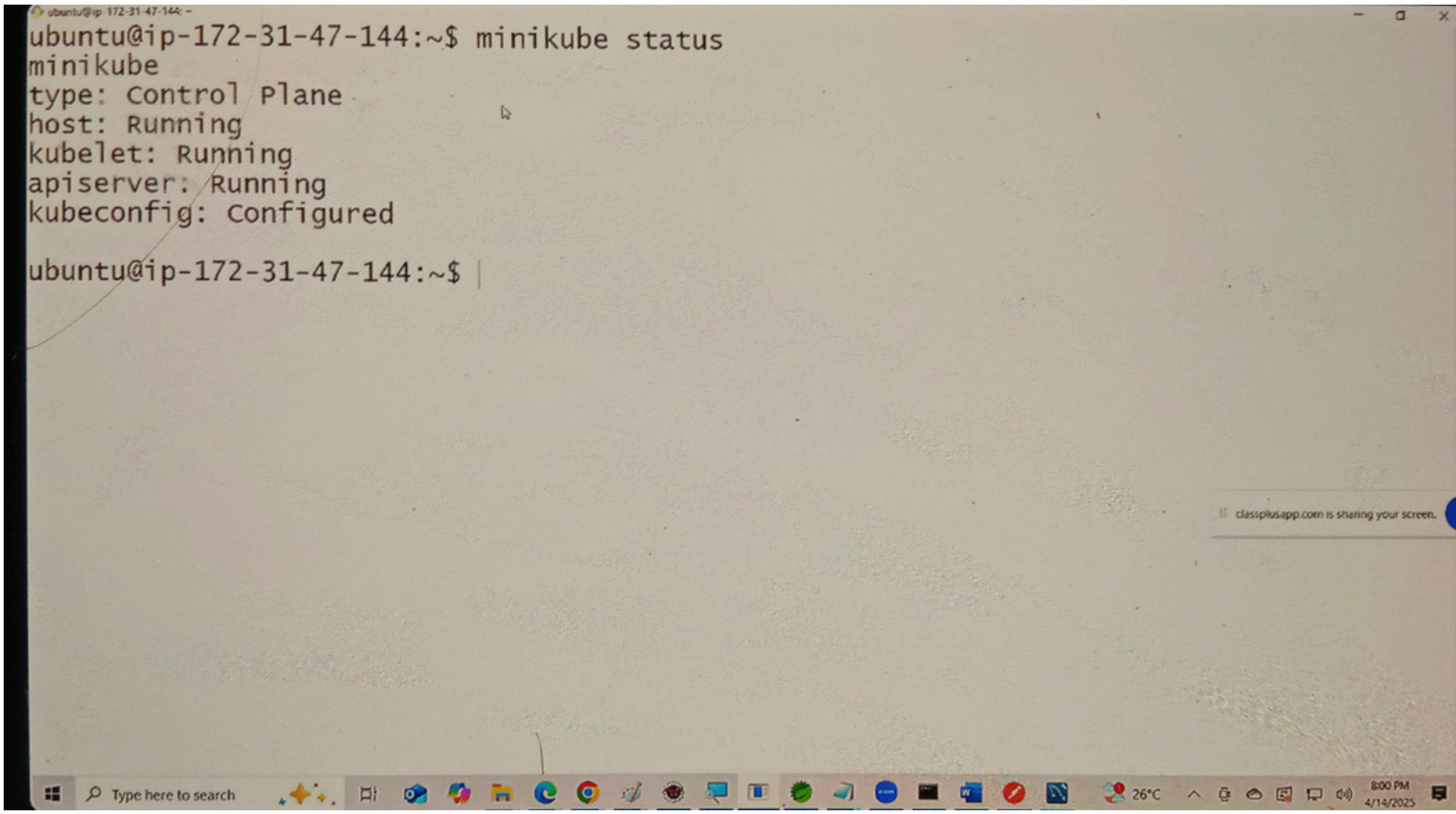
curl -LO https://storage.googleapis.com/kubernetes-release/release/$(curl -s https://storage.googleapis.com/kubernetes-release/release/stable.txt)/bin/linux/amd64/kubectl
chmod +x kubectl
sudo mv kubectl /usr/local/bin/
kubectl version -o yaml



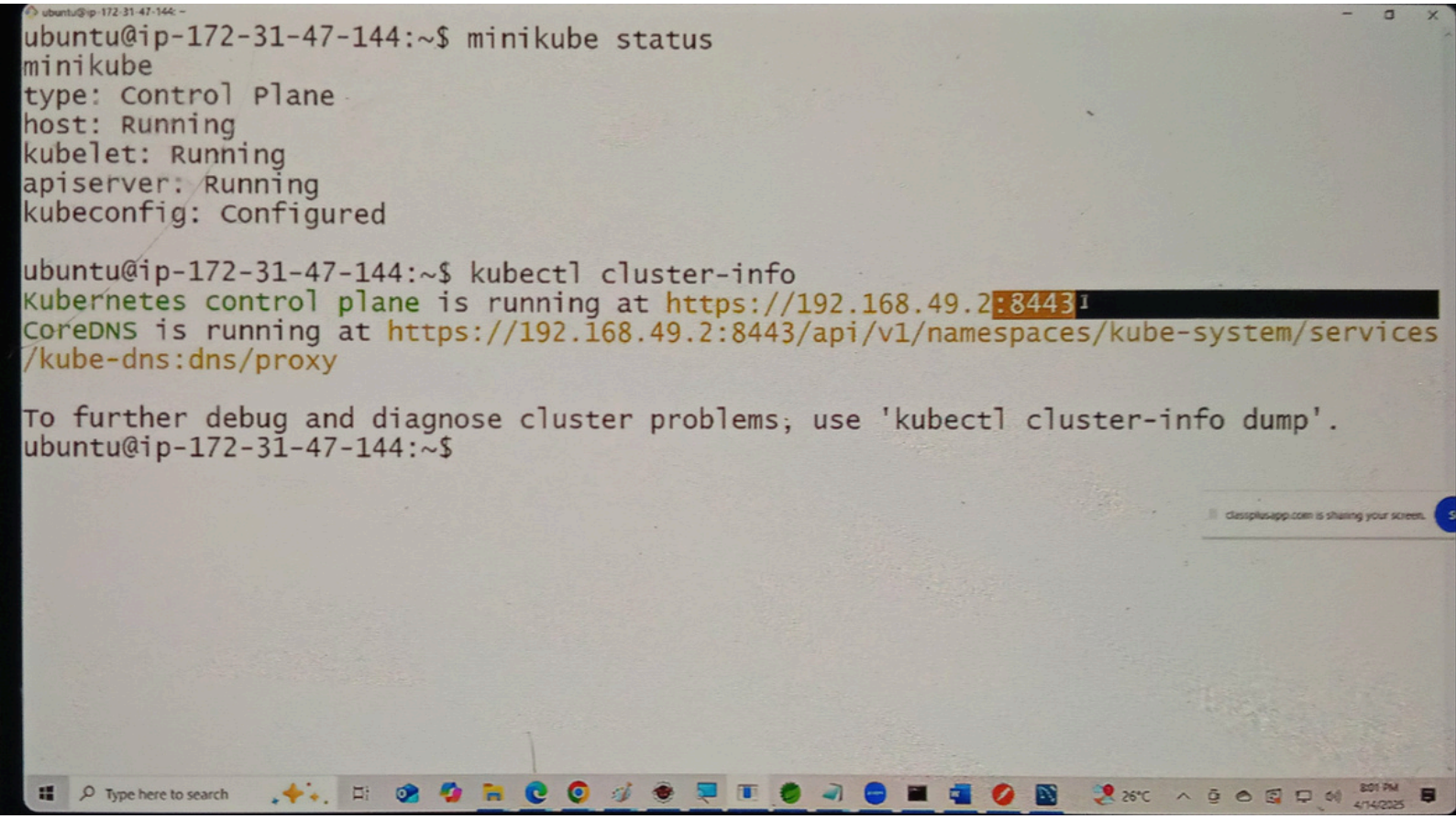
Step-6 : Start MiniKube Server
minikube start — driver=docker



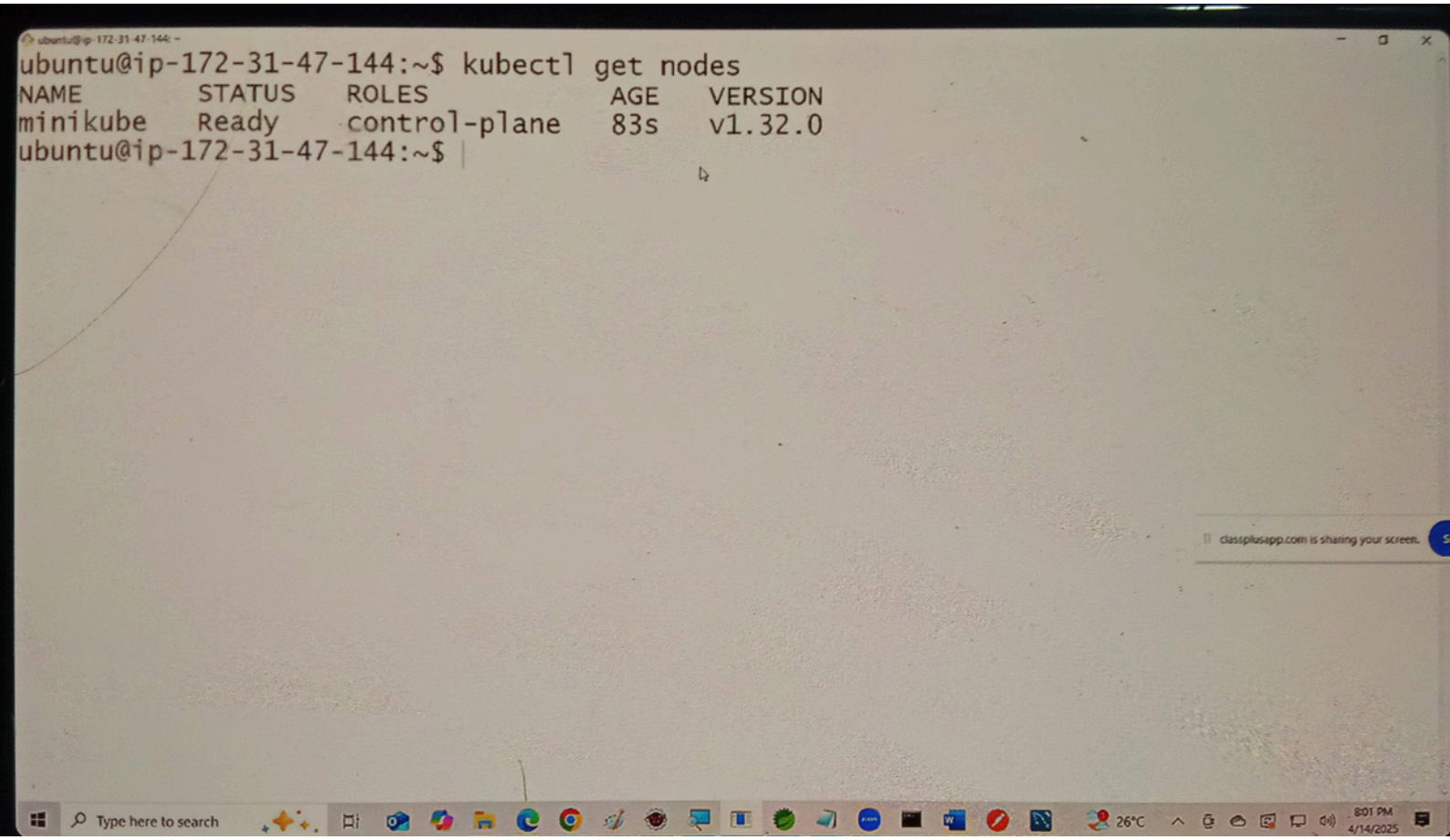
Step-7 : Check MiniKube status
minikube status



Step-8 : Access K8S Cluster
kubectl cluster-info

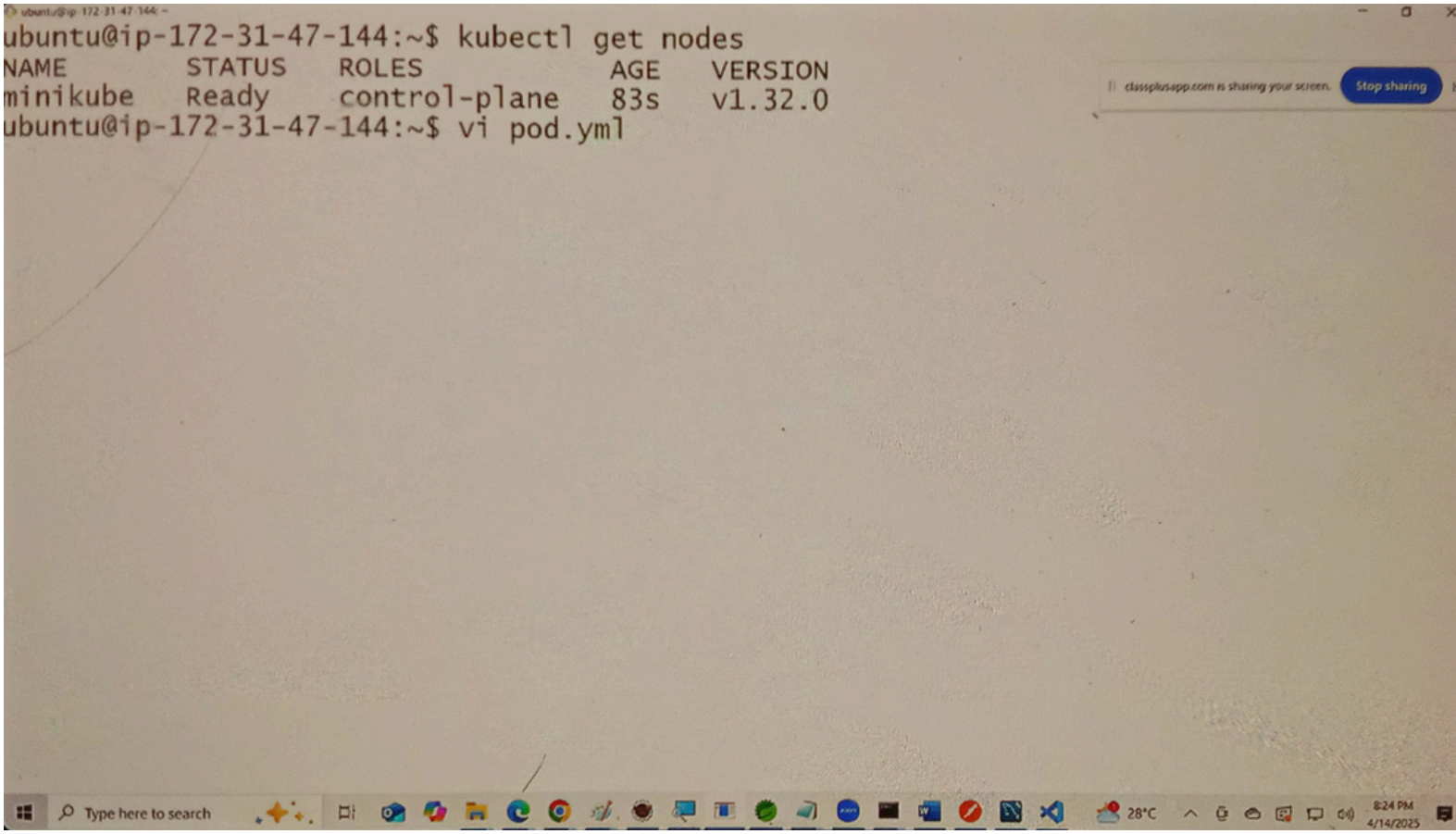


Step-9 : Access K8S Nodes
kubectl get nodes

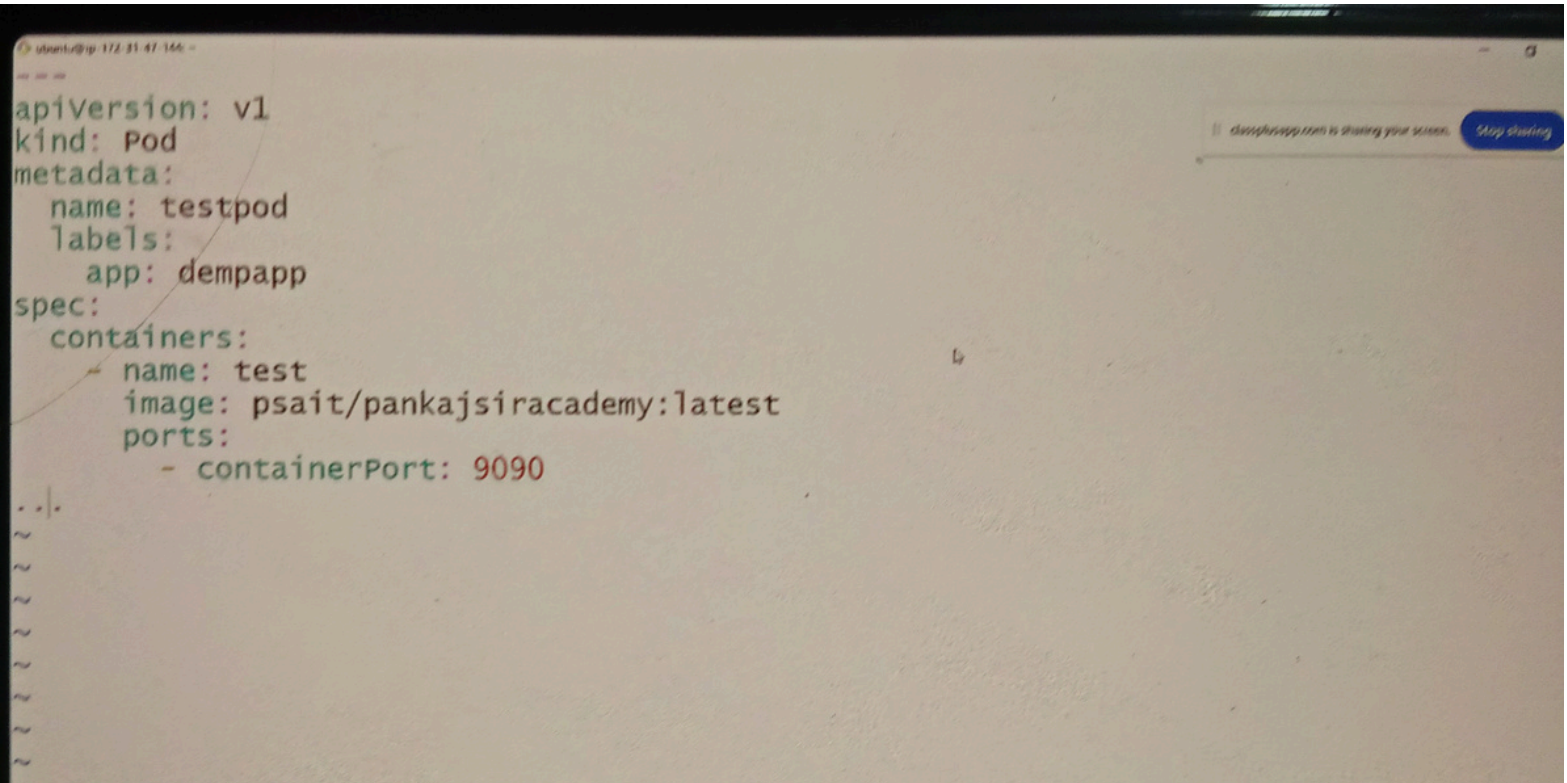


EX 1: Create pod using yml

Step 1: vi pod.yml



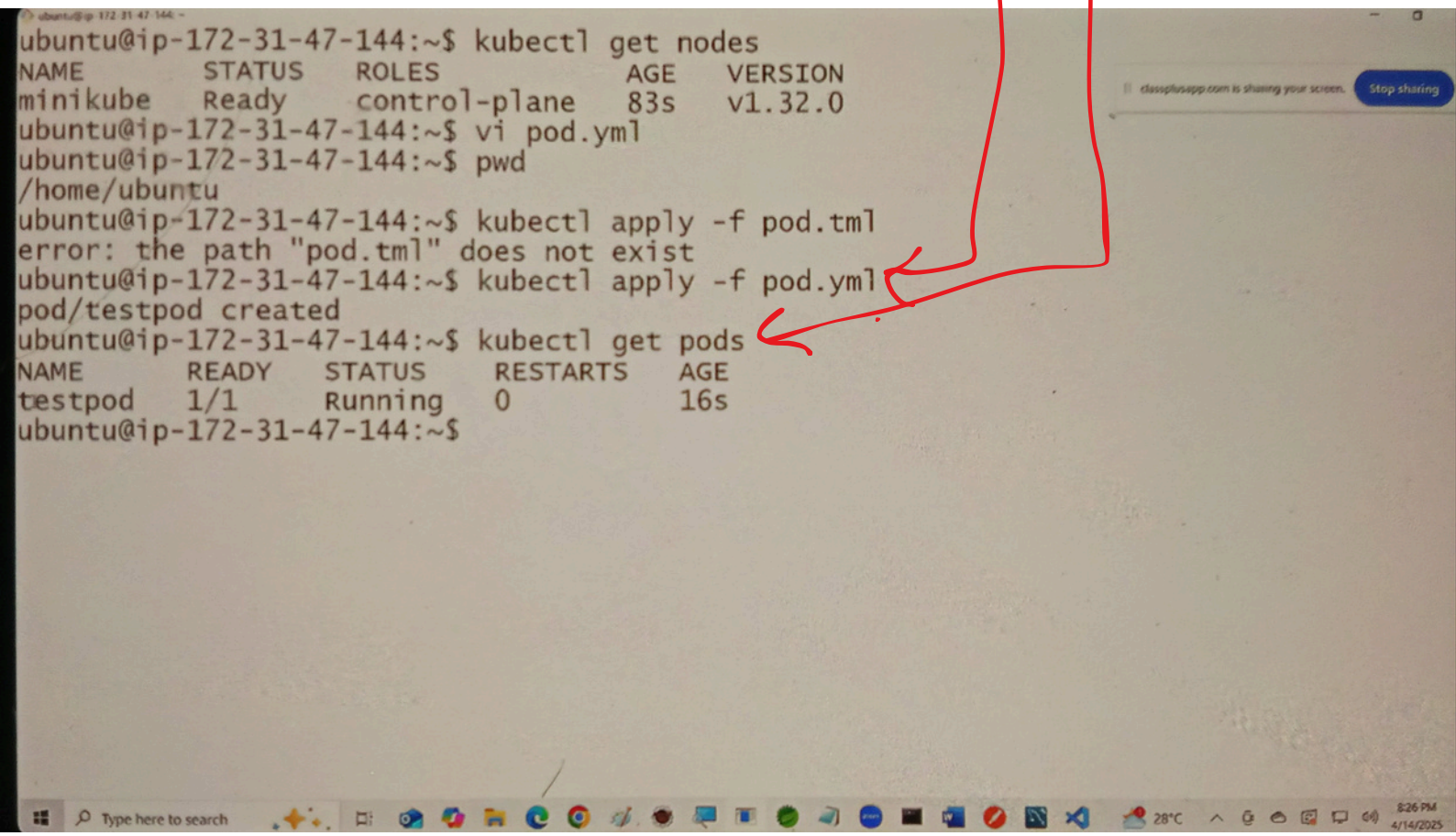
Step 2: write pod configuration in pod.yml



step 3: kubectl apply -f pod.yml

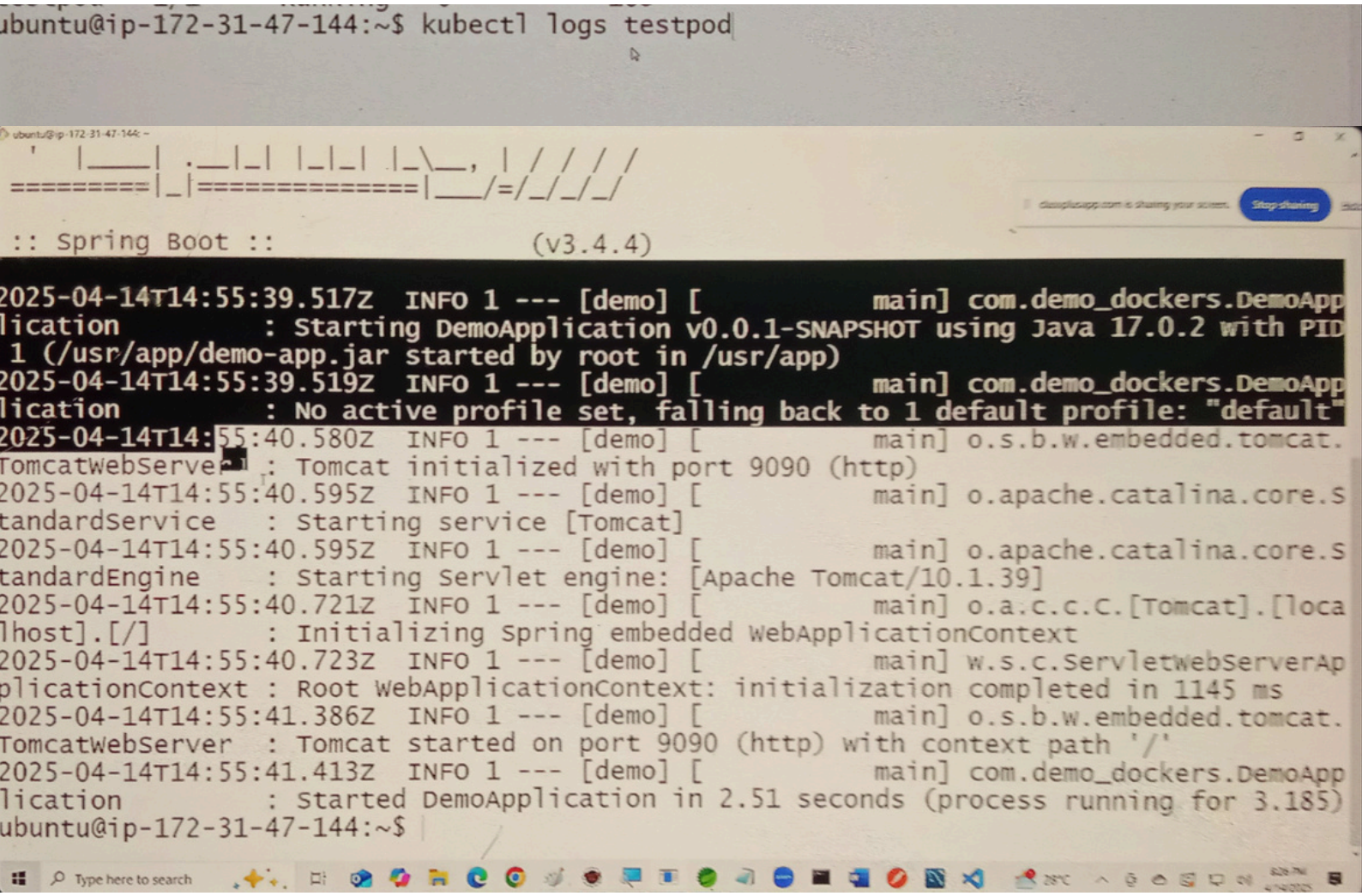
step 4: to check pod is created or not

kubectl get pods



It will checks logs of pods which you created eralier

kubectl logs testpod



EX 2: Create pod using yml

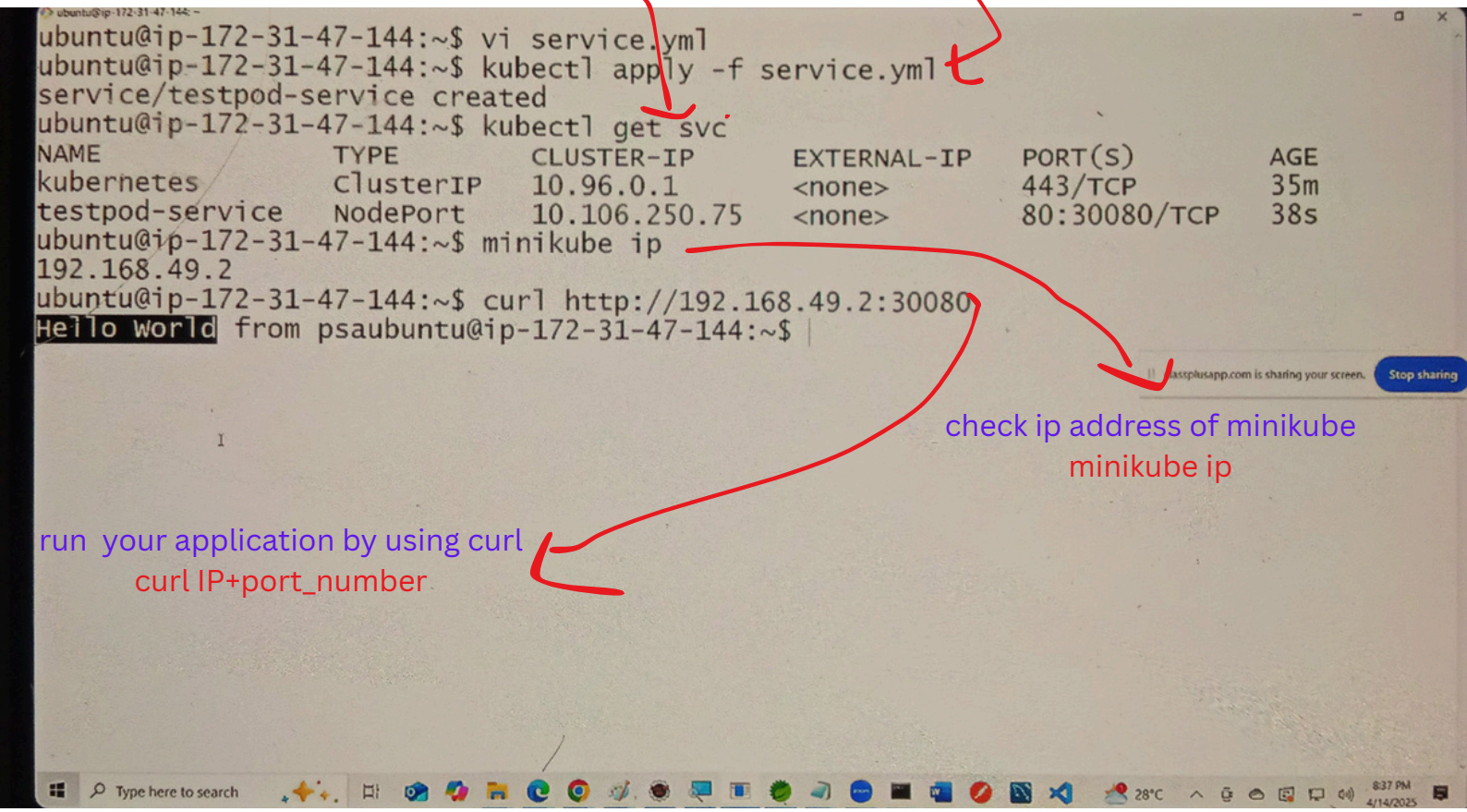
```
ubuntu@ip-172-31-47-144:~$ vi service.yml
```

```
---
apiVersion: v1
kind: Service
metadata:
  name: testpod-service
spec:
  type: NodePort
  selector:
    app: dempapp            # This must match the Pod's label
  ports:
    - port: 80              # Exposed port for external access
      targetPort: 9090      # Port on which the app is running inside the container
      nodePort: 30080       # External port exposed on each node
~
~
~
~
~
~
~
~
~
~
-- INSERT --                                              14,1          All
```

kubectl apply -f service.yml

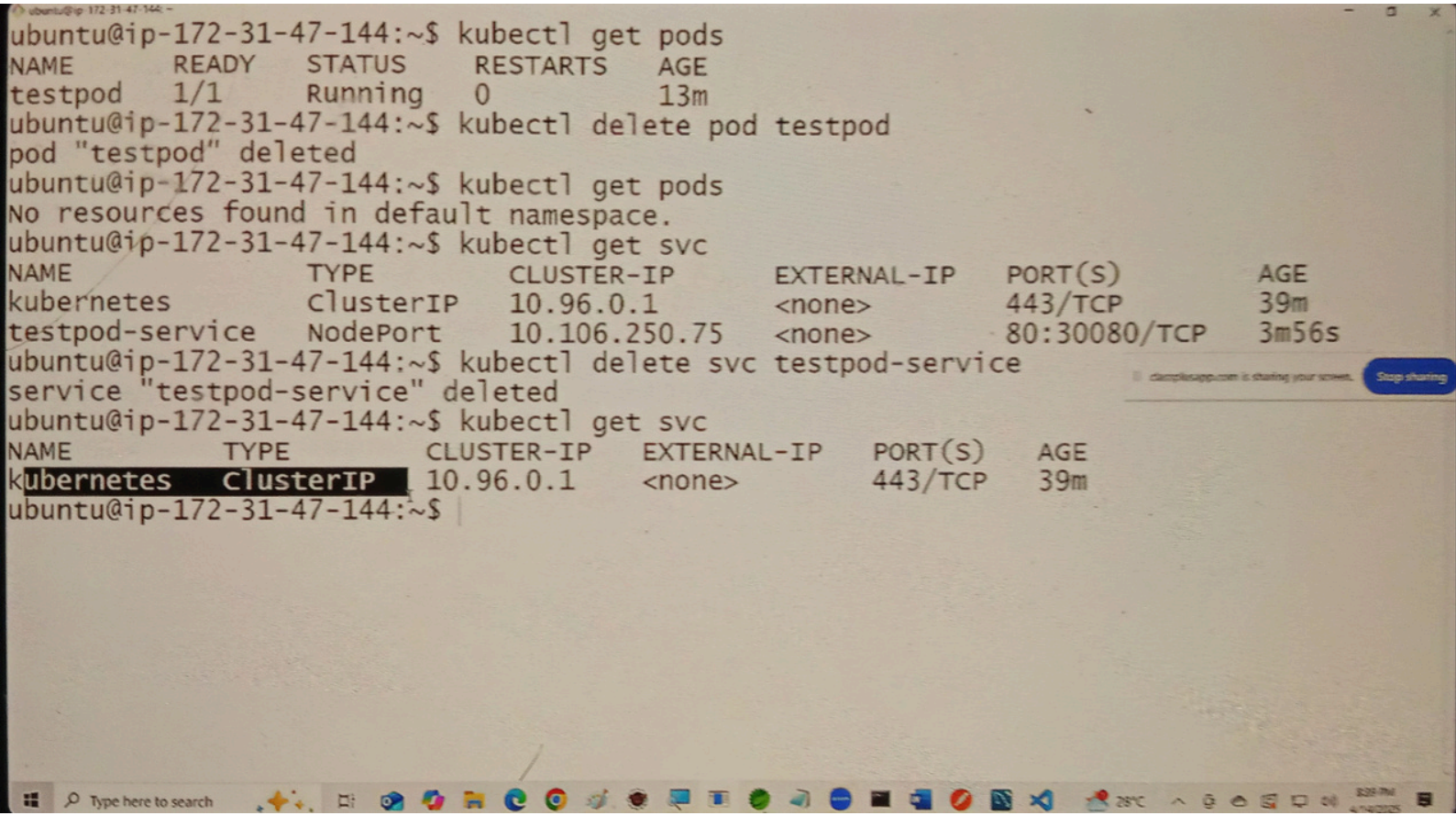
Check which service is running

kubectl get svc



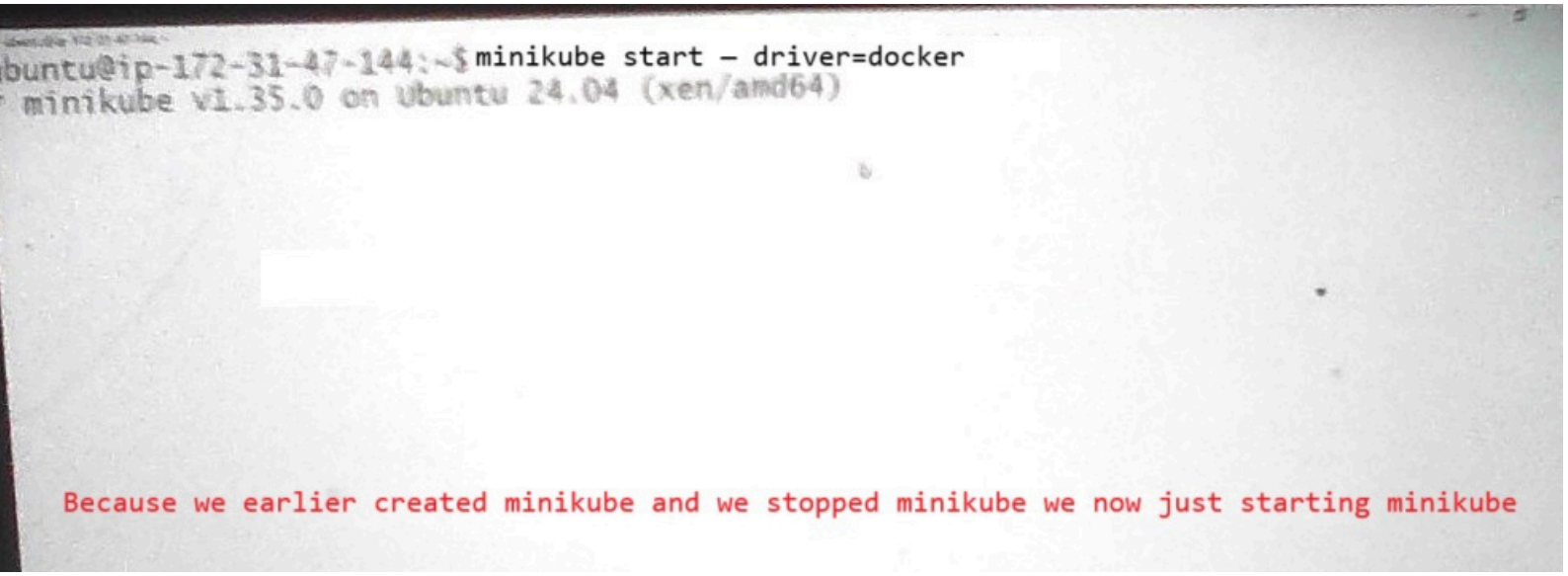
you can delete the service also lke below

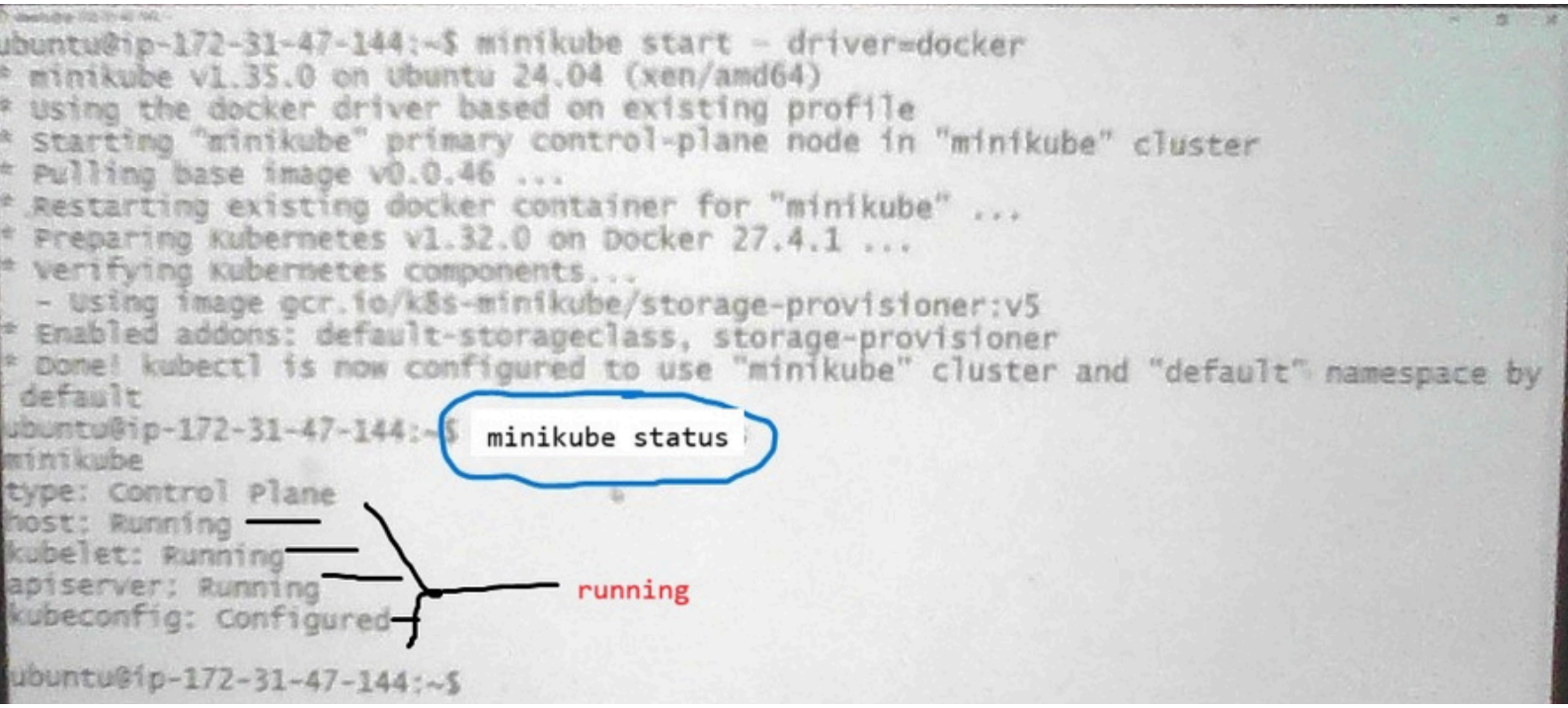
kubectl delete svc testpod-service



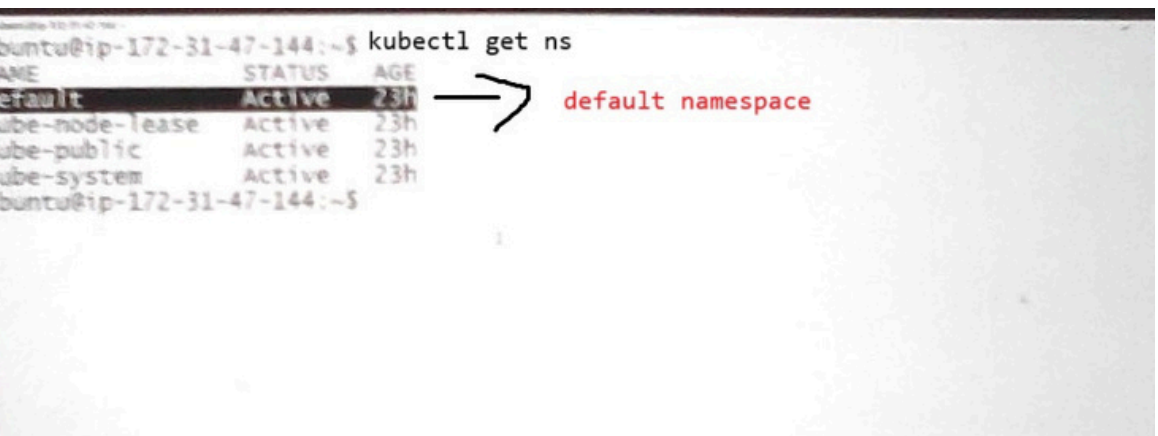
namespace

start minikube





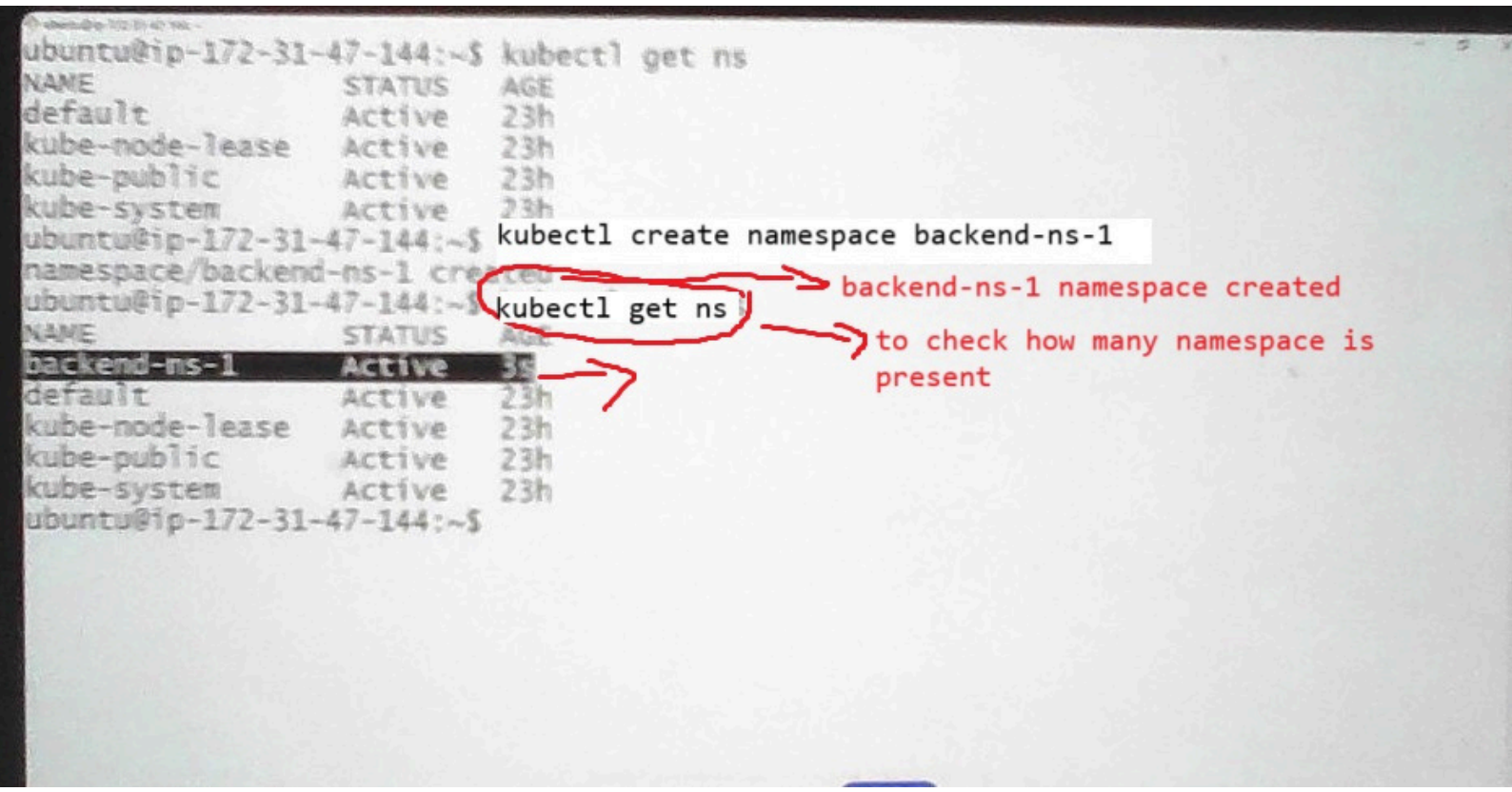
before create namespace how many namespace is present



namespace

A Namespace in Kubernetes is like a folder or directory in a computer's file system. It helps organize and group resources, like Pods, Services, Deployments, etc.

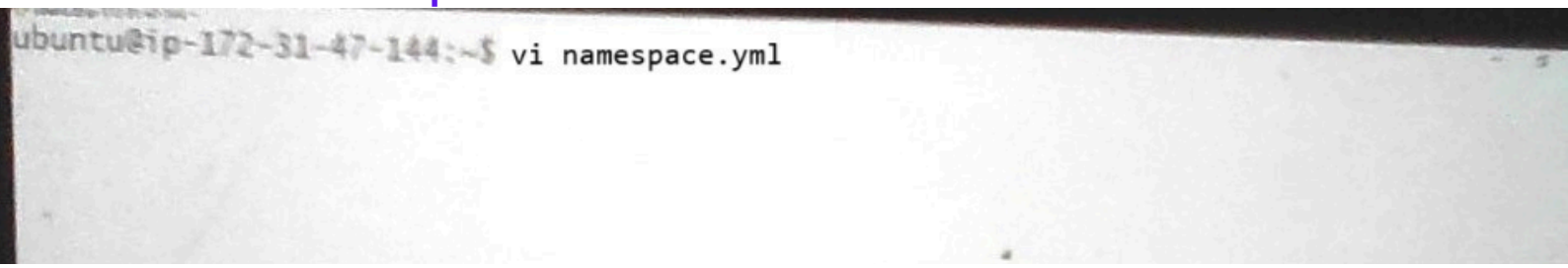
kubectl create namespace name



ex 1

we can craete namespace by using yml file like below

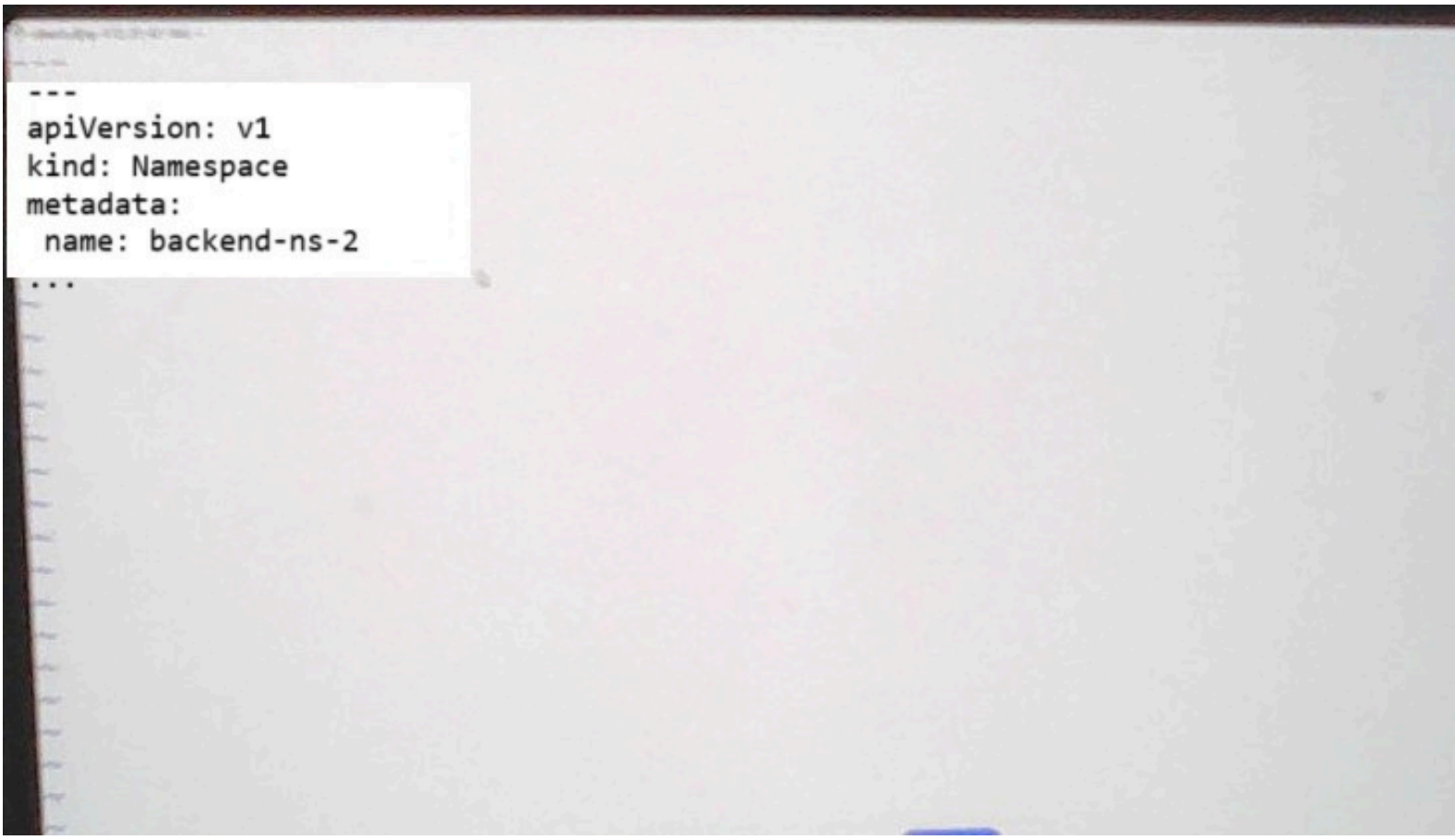
kubectl create namespace name



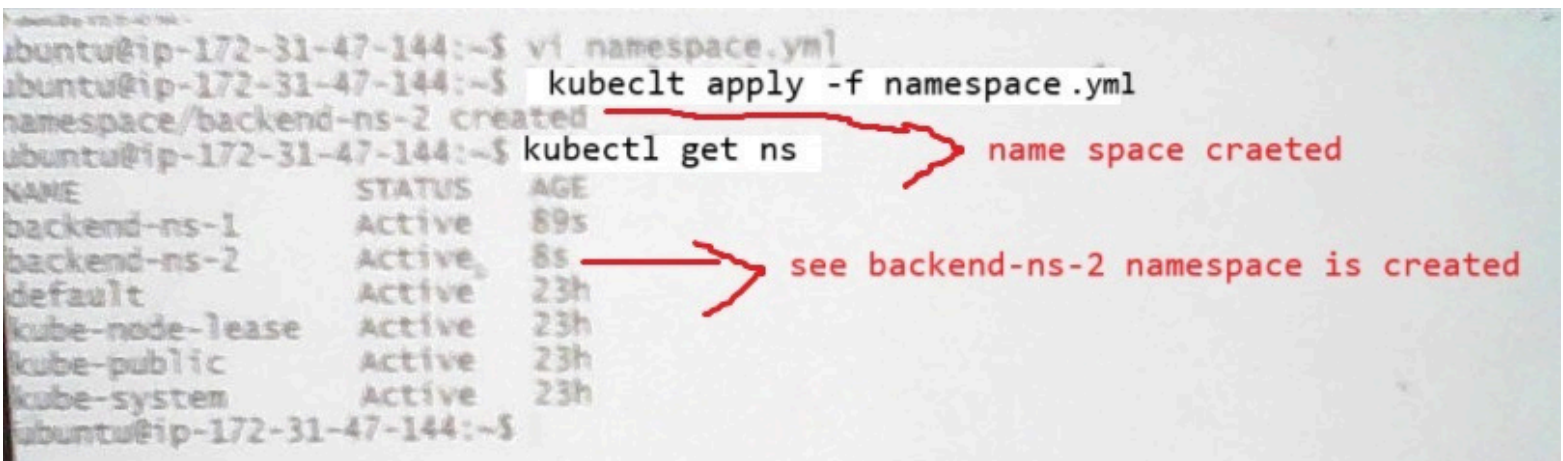
paste below code in .yml

```
---
apiVersion: v1
kind: Namespace
metadata:
 name: backend-ns2
...
```

note : before paste yml code check syntax



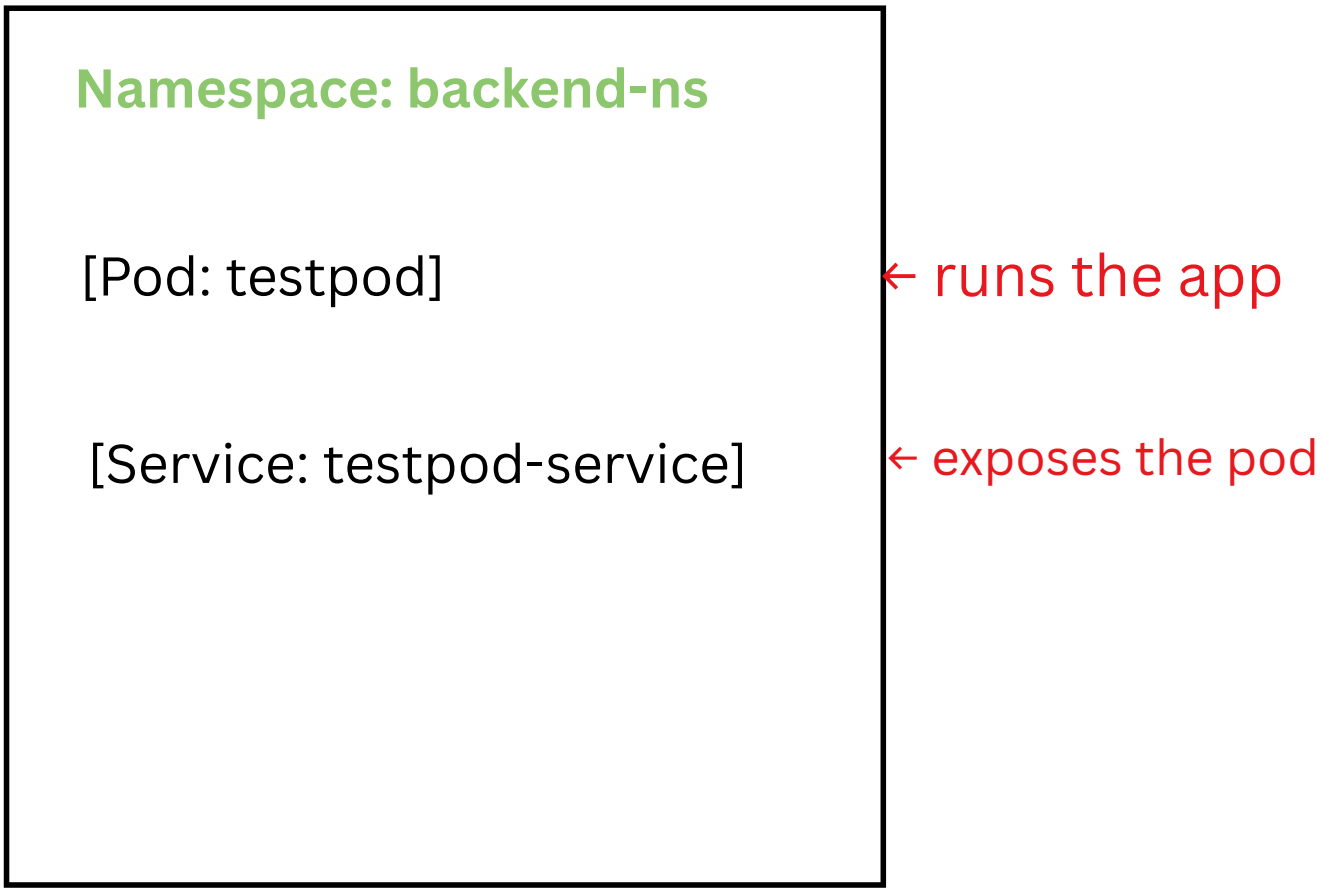
after creating yml file now run as shown below

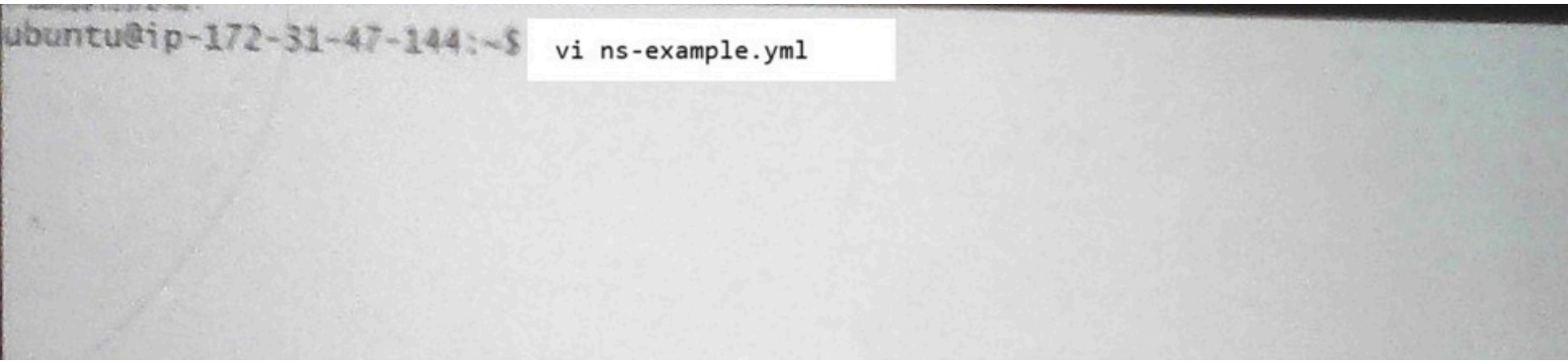


ex 2

we can craete namespace + pod by using yml file like below

kubectl create namespace name



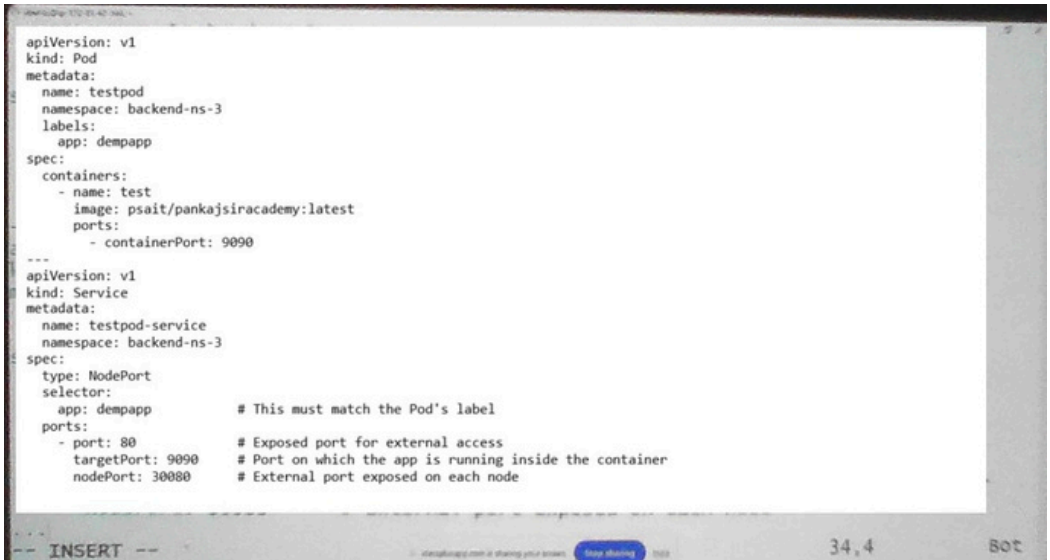


paste below code in .yml

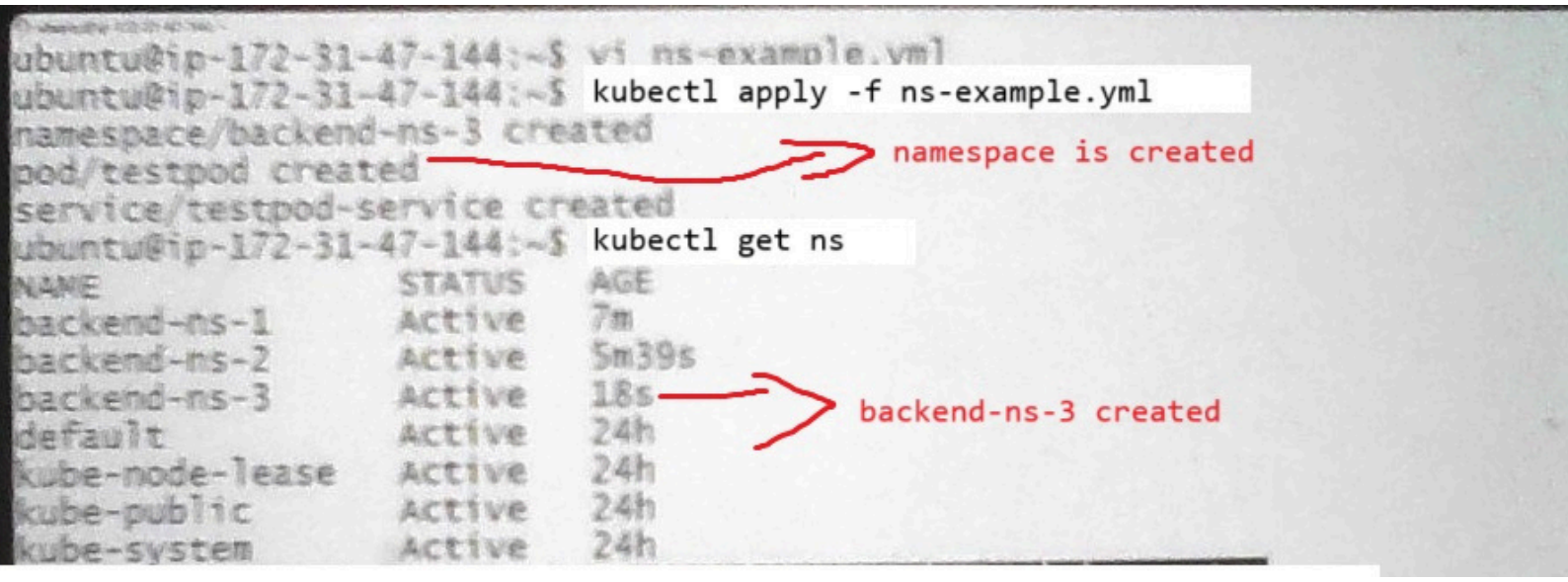
Namespace with POD with Service creation yml file

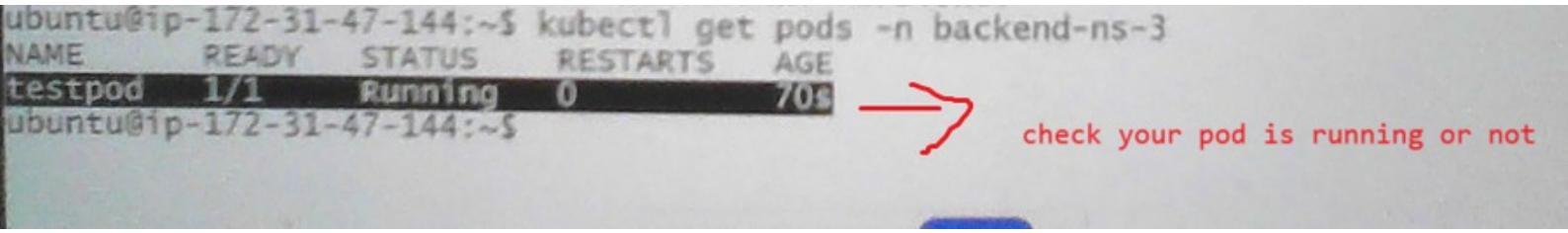
```
---
apiVersion: v1
kind: Namespace
metadata:
 name: backend-ns
---
apiVersion: v1
kind: Pod
metadata:
  name: testpod
  namespace: backend-ns
  labels:
    app: dempapp
spec:
  containers:
    - name: test
      image: psait/pankajsiracademy:latest
      ports:
        - containerPort: 9090
---
apiVersion: v1
kind: Service
metadata:
  name: testpod-service
  namespace: backend-ns
spec:
  type: NodePort
  selector:
    app: dempapp          # This must match the Pod's label
  ports:
    - port: 80            # Exposed port for external access
      targetPort: 9090    # Port on which the app is running inside the container
      nodePort: 30080     # External port exposed on each node
...
```

note : before paste yml code check syntax



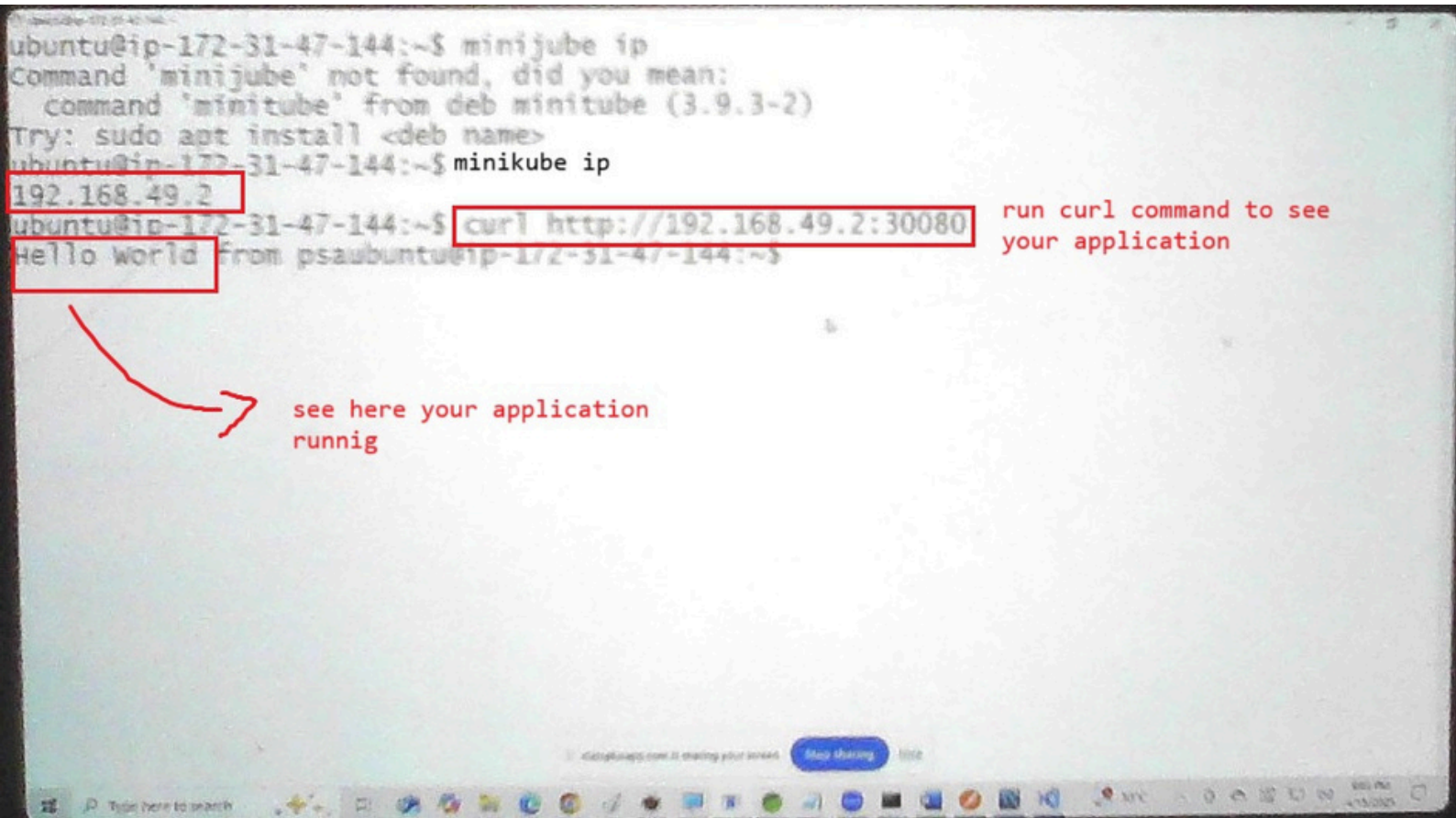
after creating yml file now run as shown below





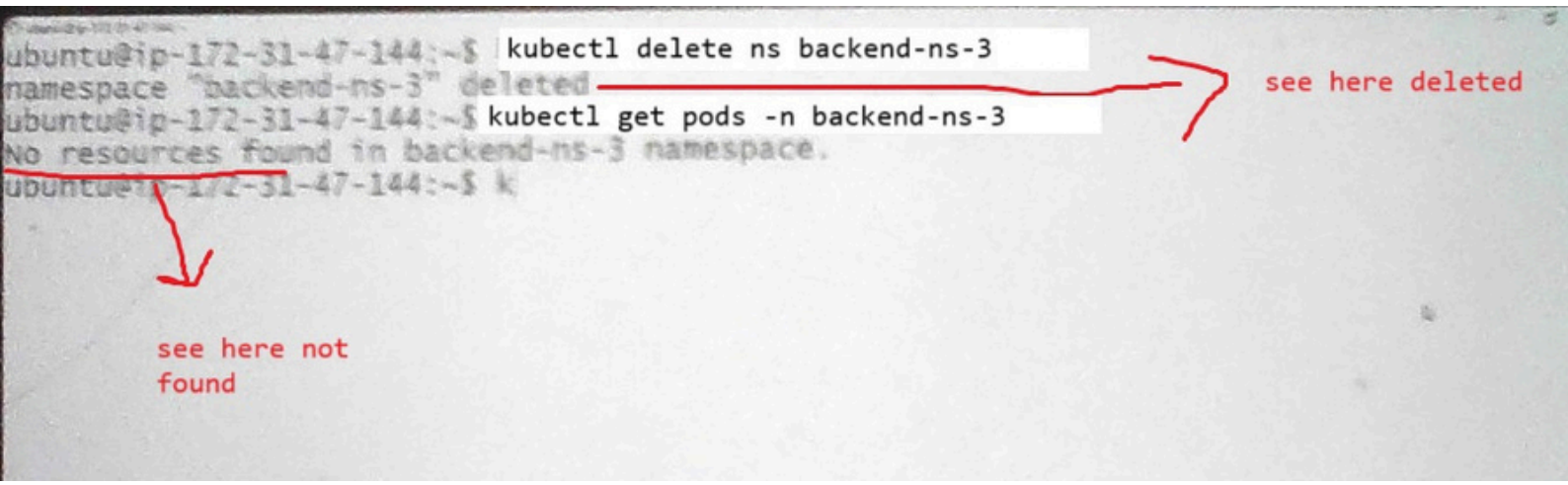
to check ip address minikube ip

to run application use curl +IP address

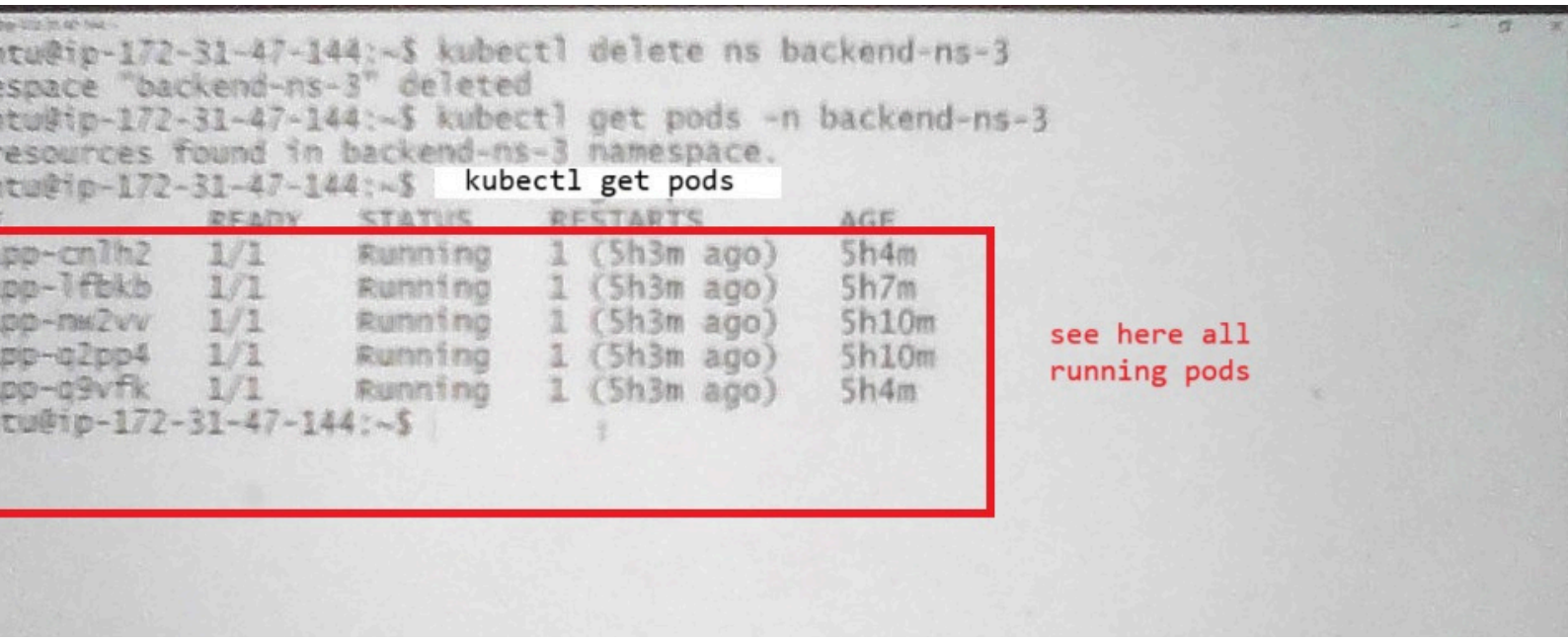


do delete namespace shown below

note: 🔥 If you delete a Namespace, all the resources inside it (like Pods, Services, Deployments, ConfigMaps, etc.) are also deleted automatically.



To check how many pods runing shown below



k8S Resources

-> When you create a Pod directly using kind: Pod, Kubernetes does not manage its lifecycle — if it crashes or is deleted, it's gone forever unless recreated manually.

-> K8S resources manages POD lifecycle

-> To let Kubernetes manage, restart, and scale Pods, we use higher-level controllers like the ones you listed.

🔁 **1) ReplicationController (RC)**
🔁 **2) ReplicaSet (RS)**
🚀 **3) Deployment**
💠 **4) DaemonSet**
💾 **5) StatefulSet**

🤖 ReplicationController (RC)

(RC is old, replaced mostly by ReplicaSet and Deployment, but still good to understand!)

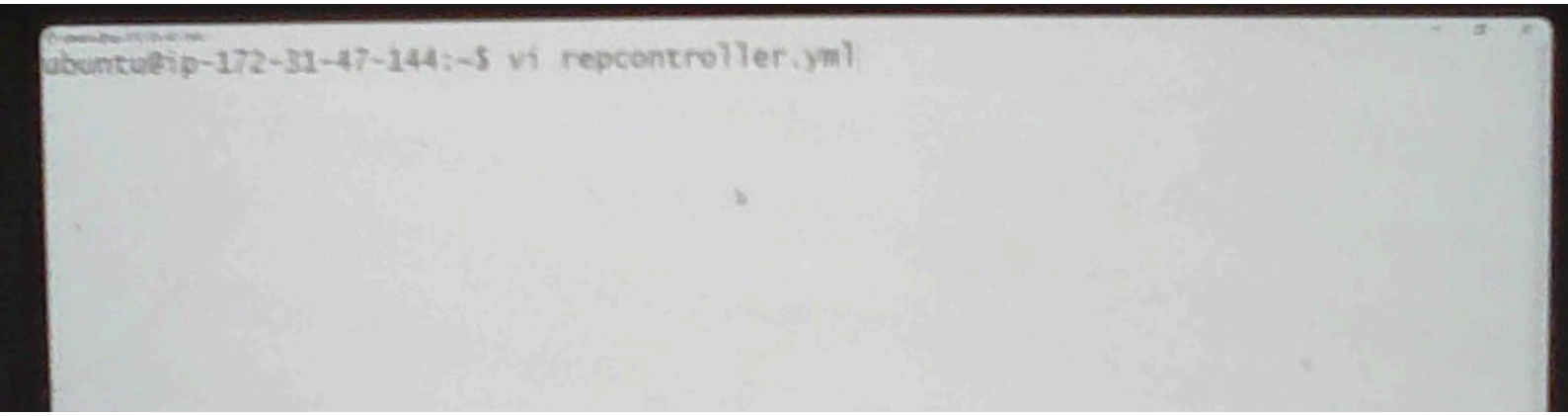
A ReplicationController ensures a specific number of Pods are always running.

It monitors the Pods and if one crashes or is deleted, it creates a new one automatically.

It's basically a Pod manager.

we here replace pod with ReplicationController(RC) in yml

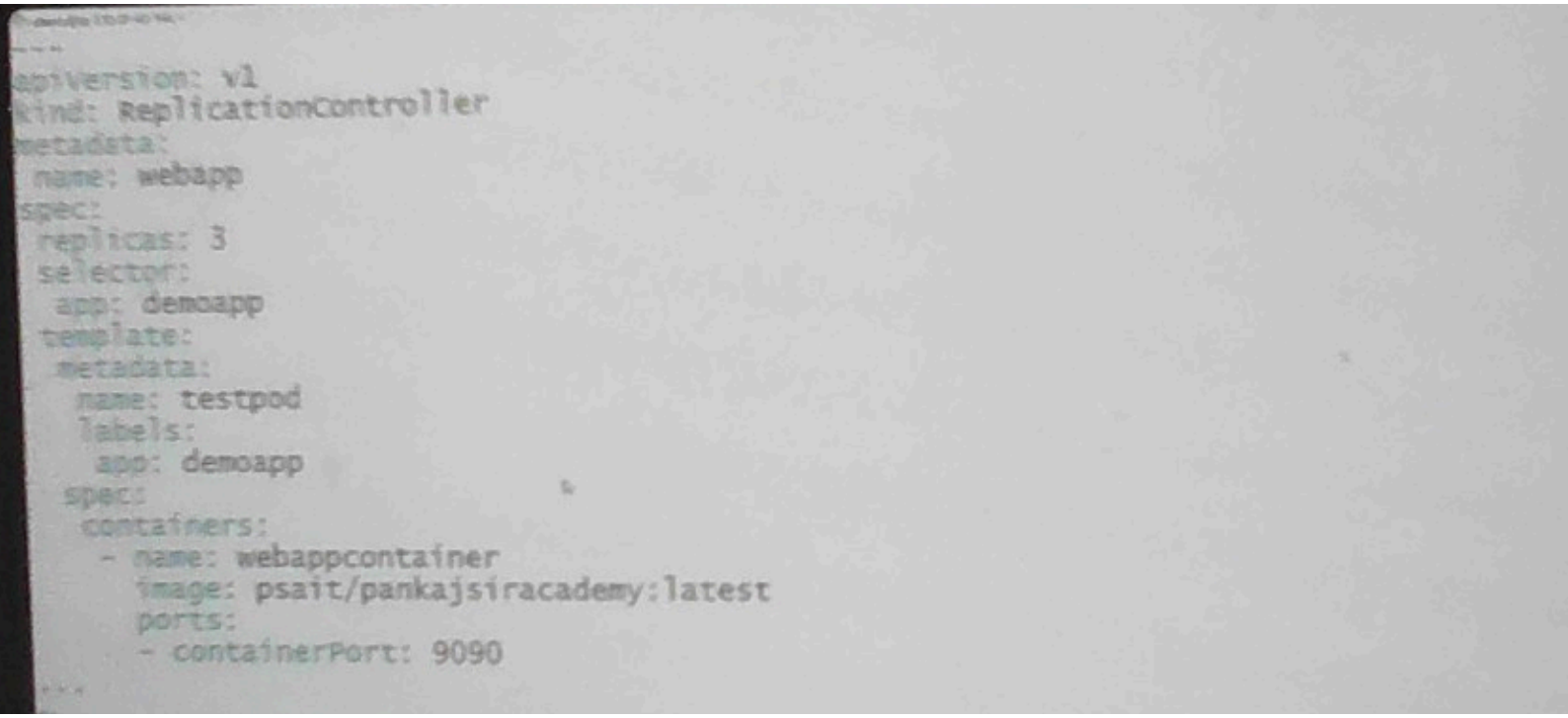
step 1 vi name.yml



step 2 write code in yml

```
---
apiVersion: v1
kind: ReplicationController
metadata:
 name: webapp
spec:
 replicas: 3
 selector:
  app: dempapp
 template:
  metadata:
   name: testpod
   labels:
    app: dempapp
  spec:
   containers:
    - name: webappcontainer
      image: psait/pankajsiracademy:latest
      ports:
      - containerPort: 9090
...
```

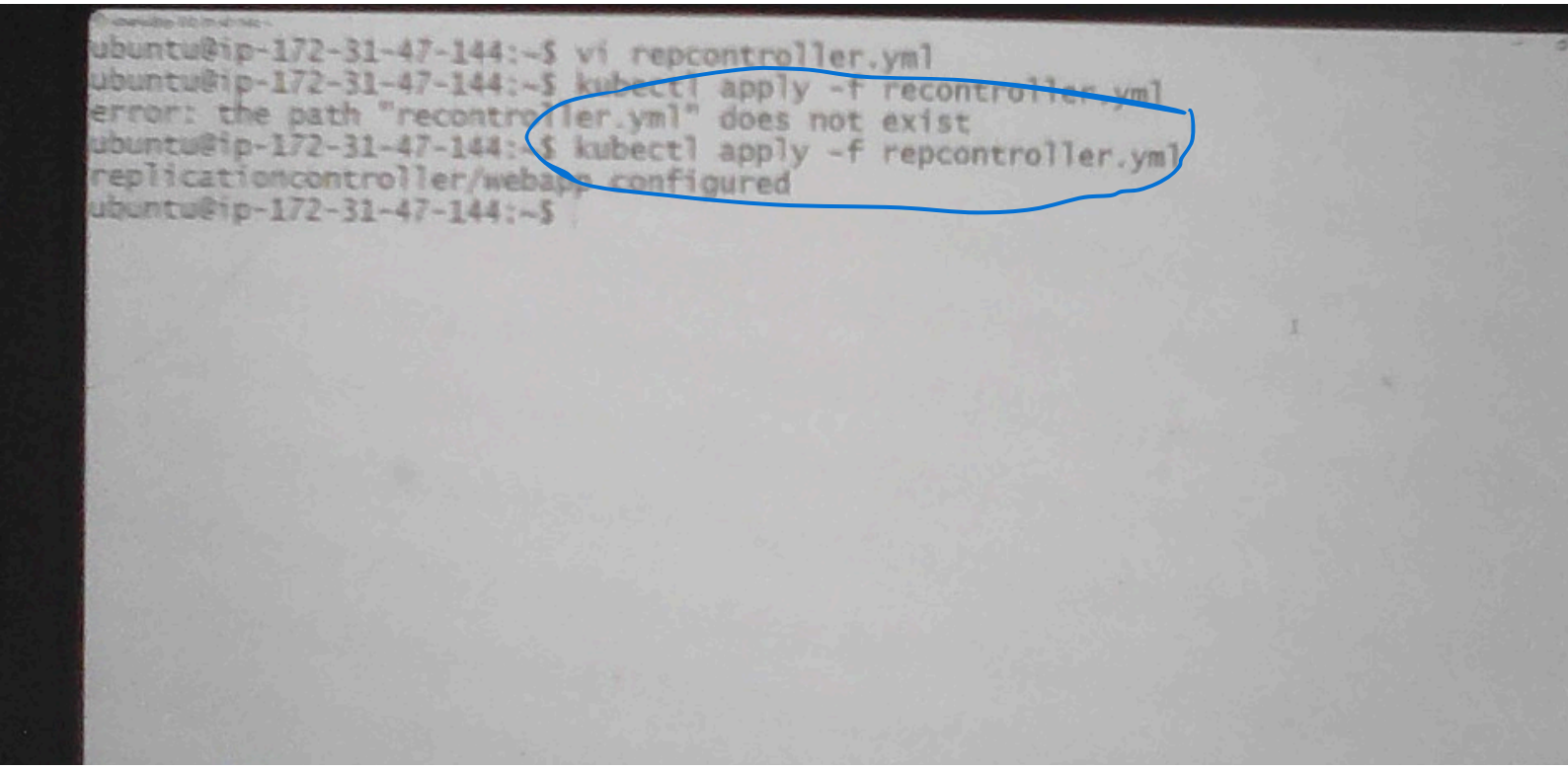
note : before paste yml code check syntax



step 3

after creating yml file now run as shown below

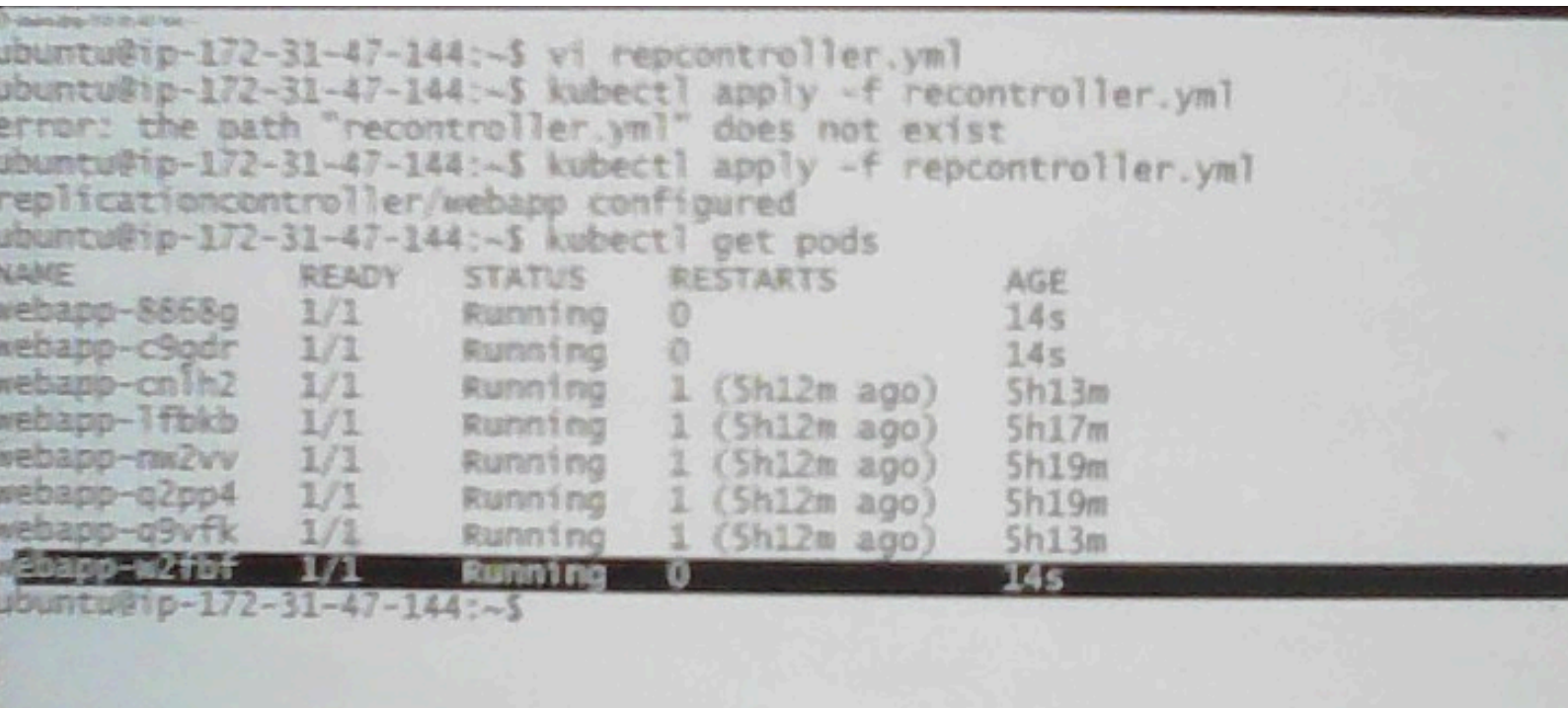
kubectl apply -f name.yml



step 4

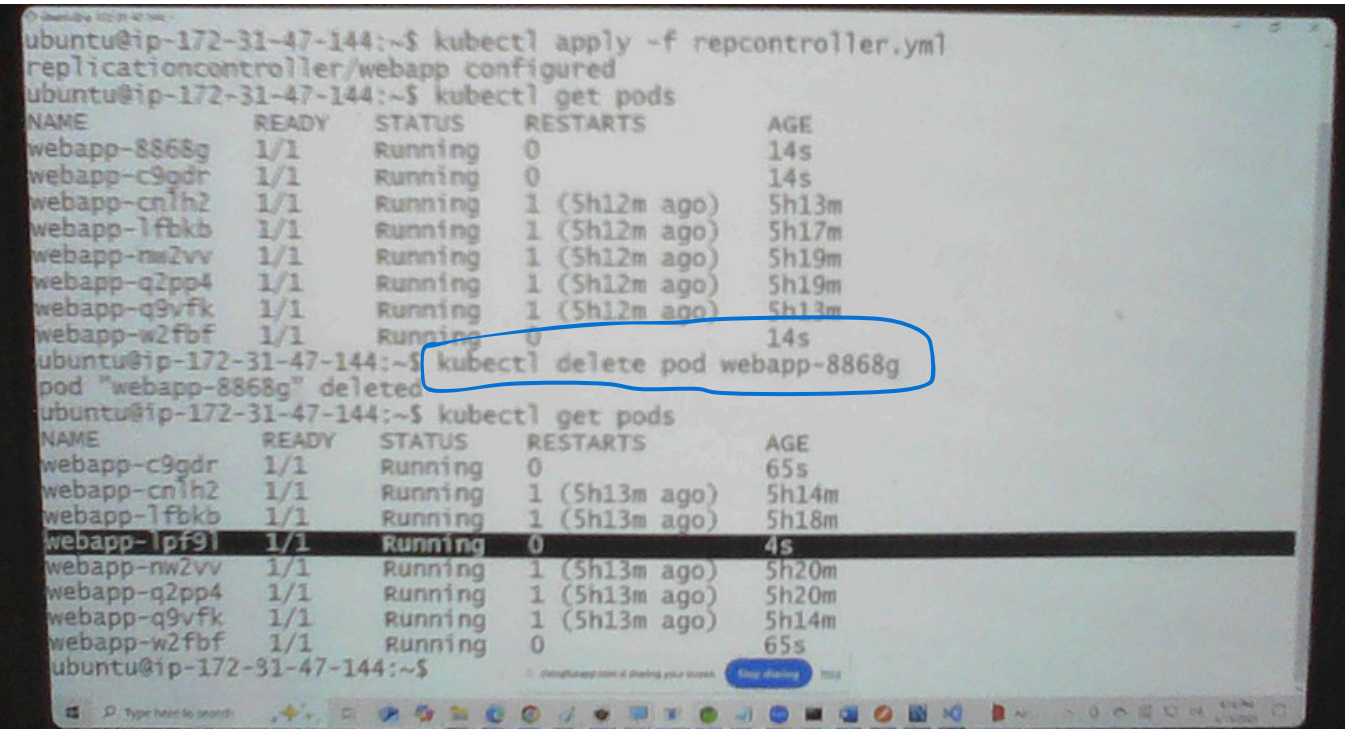
to check pod is created or not

kubectl get pods

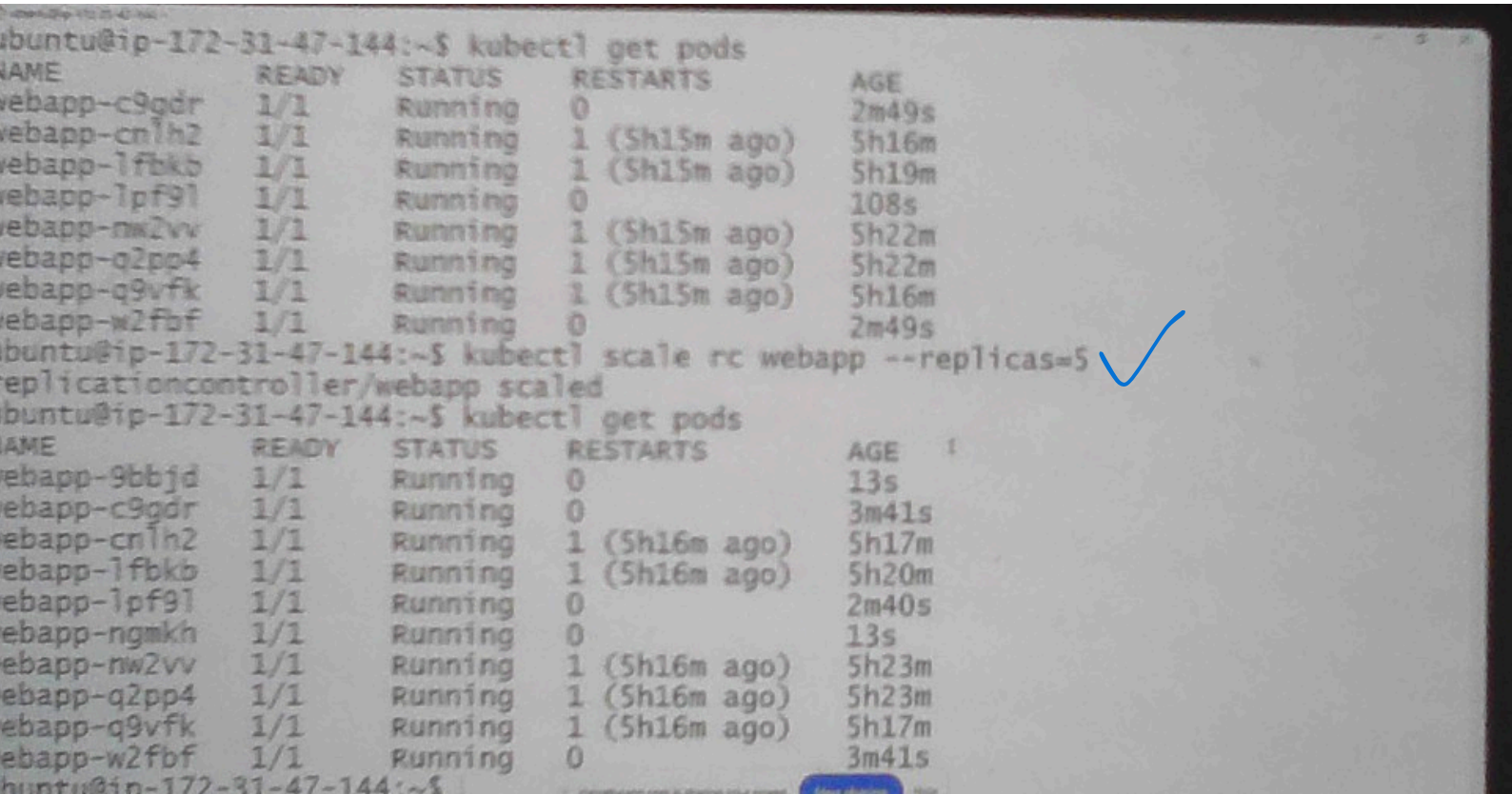


NOTE:
As per above yml we used ReplicaController(RC) it uses replicas 3 it means it created 3 pods even tho if you delete any pods

now you can delete any pod like below shown **but** it will regenerate pods automatically because of RC



You can increase replicas it means increasing number of pods like below



ReplicaSet in Kubernetes

🔹 What is a ReplicaSet?

A ReplicaSet (RS) is a Kubernetes resource that ensures a specified number of identical Pods are running at any given time.

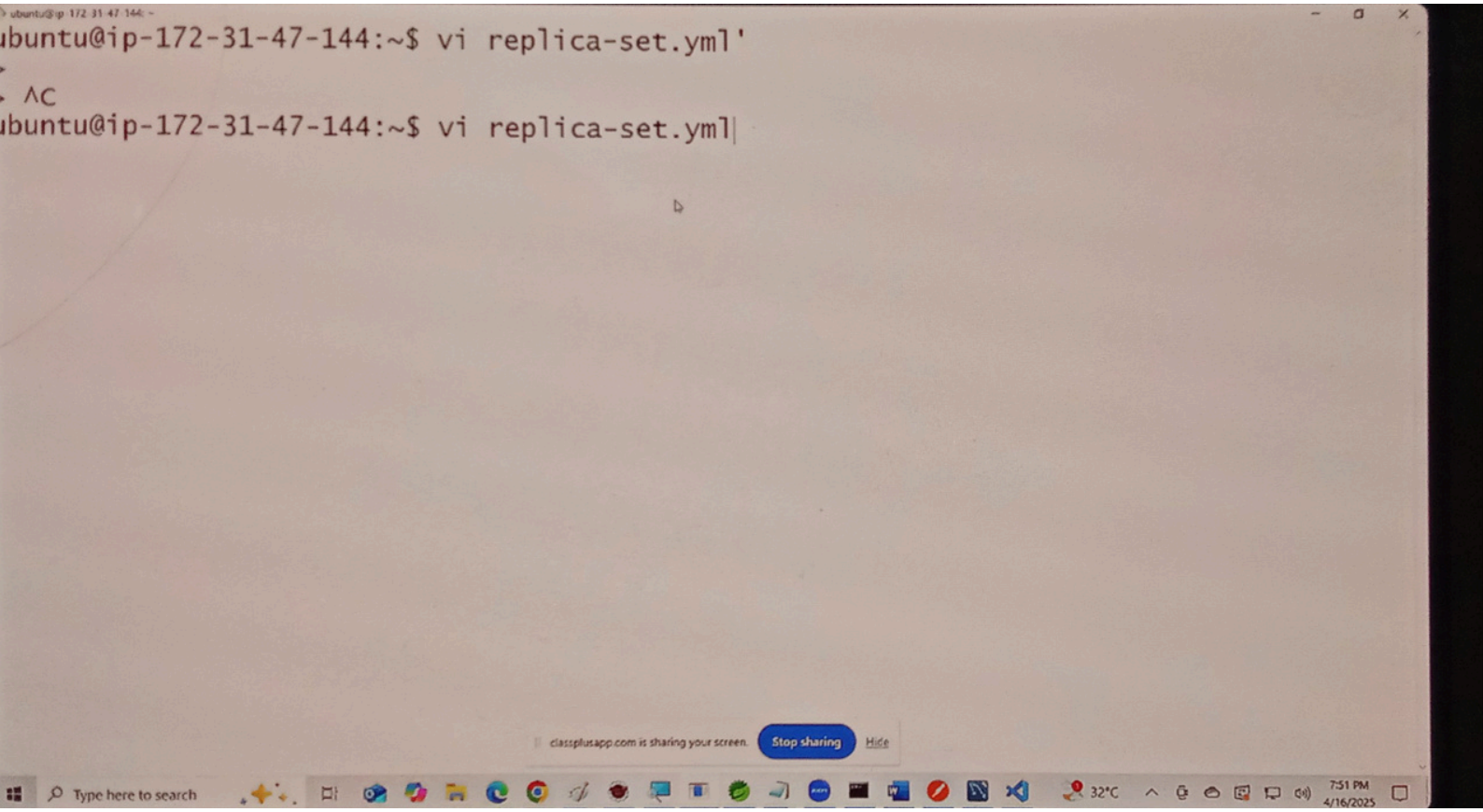
💡 Key Features:

Self-healing: If a Pod crashes or is manually deleted, the RS will automatically create a new Pod to maintain the desired number.

Scaling: You can increase or decrease the number of replicas (Pods) easily.

Selector-based matching: RS manages only those Pods that match its label selector.

step 1 create yml



step 2 write yml code

```yaml
---
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: webapp
spec:
  replicas: 3
  selector:
    matchLabels:
      app: dempapp        # Must match pod template labels
  template:
    metadata:
      labels:
        app: dempapp
    spec:
      containers:
        - name: webappcontainer
          image: psait/pankajsiracademy:latest
          ports:
            - containerPort: 9090
```

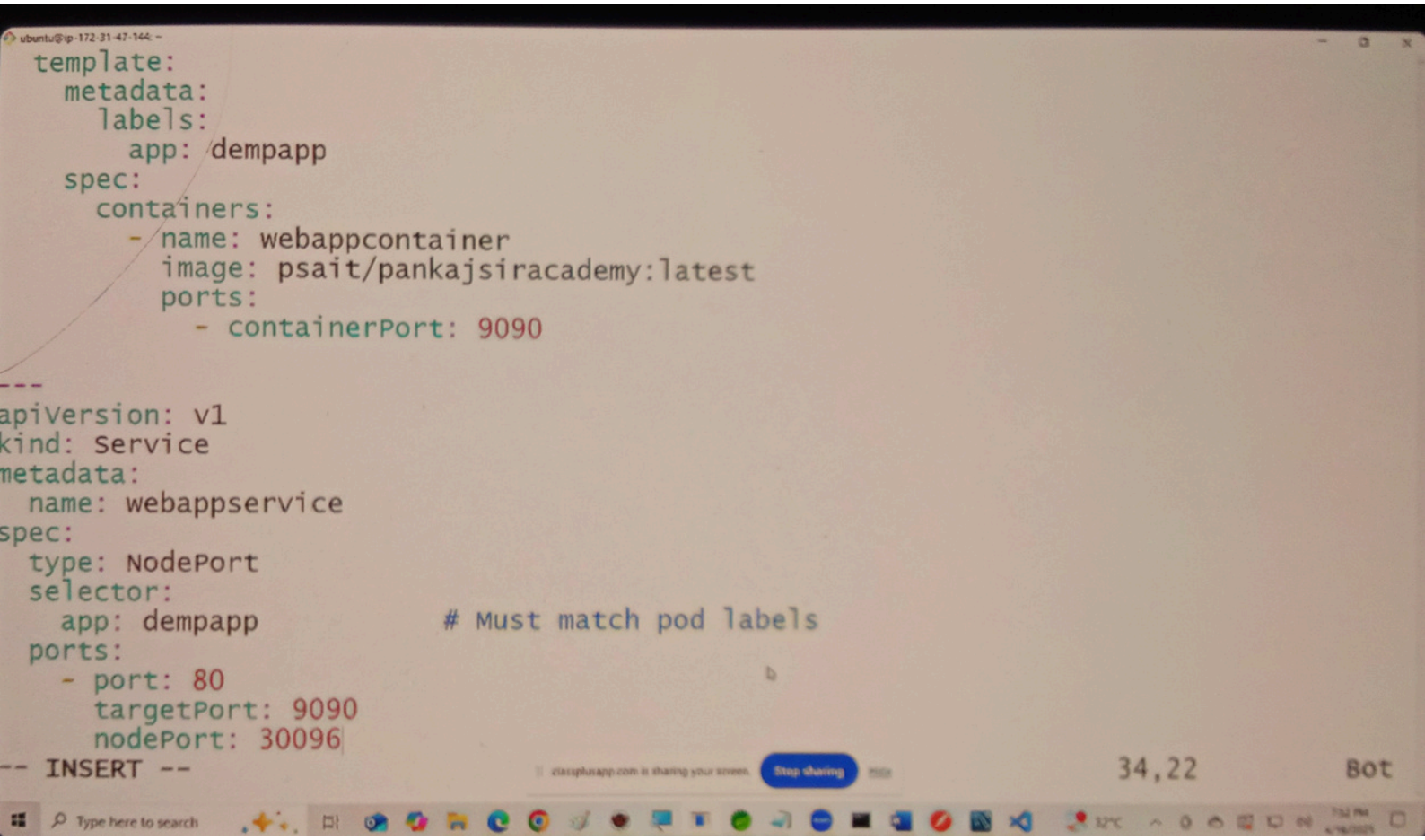
→ **Resource type**

note : before paste yml code check syntax

- **matchLabels** is a shorthand for equality-based match (e.g., app=web).
- **matchExpressions** gives you more flexibility (In, NotIn, Exists, DoesNotExist).
- You can use both together in a single ReplicaSet selector if needed.

```yaml
---
apiVersion: v1
kind: Service
metadata:
  name: webappservice
spec:
  type: NodePort
  selector:
    app: dempapp          # Must match pod labels
  ports:
    - port: 80
      targetPort: 9090
      nodePort: 30096
```

→ **service type**



step 3 : kubectl apply -f name.yml

```
ubuntu@ip-172-31-47-144:~$ kubectl apply -f replica-set.yml
replicaset.apps/webapp unchanged
service/webappservice configured
ubuntu@ip-172-31-47-144:~$
```

see here created

step 4: get -> minikube ip

step 5: run curl like below -> curl + minikube ip

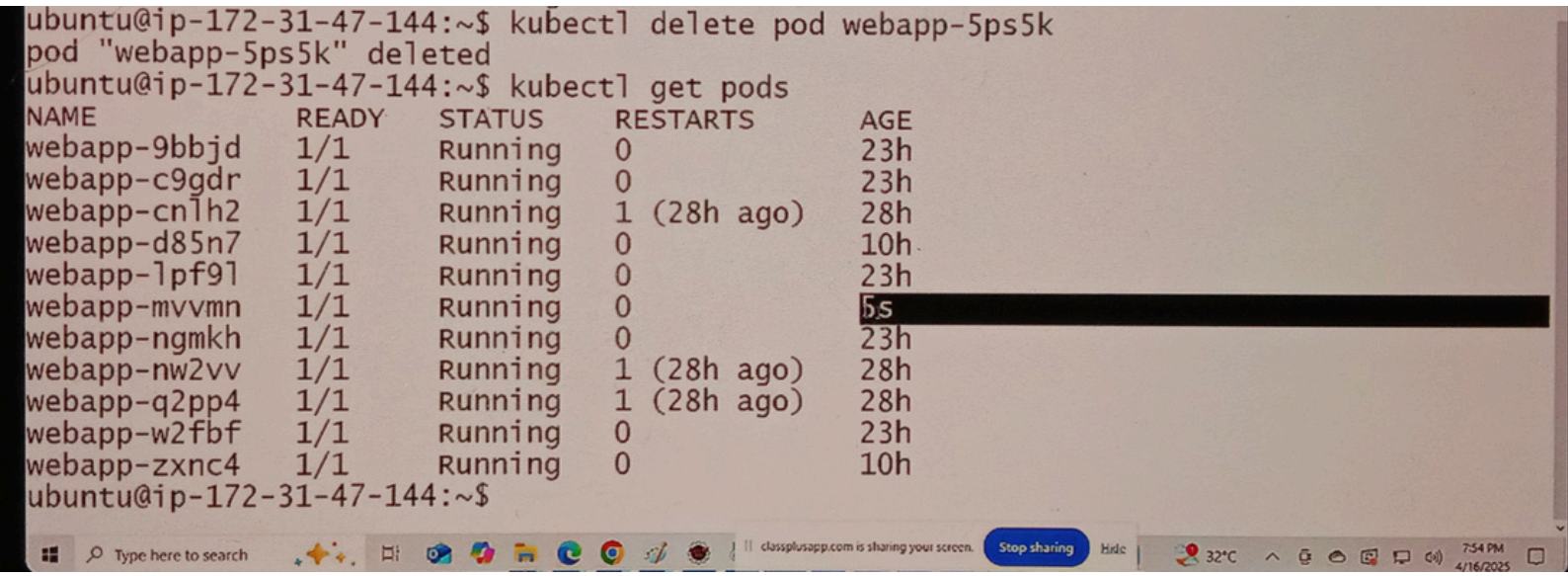
```
ubuntu@ip-172-31-47-144:~$ minikube ip
192.168.49.2
ubuntu@ip-172-31-47-144:~$ curl http://192.168.49.2:30096
Hello World from psaubuntu@ip-172-31-47-144:~$
```

step 6: check how many pods running -> kubectl get pods

```
Hello World from psaubuntu@ip-172-31-47-144:~$ kubectl get pods
NAME            READY   STATUS    RESTARTS      AGE
webapp-5ps5k    1/1     Running   0             10h
webapp-9bbjd    1/1     Running   0             23h
webapp-c9gdr    1/1     Running   0             23h
webapp-cnlh2    1/1     Running   1 (28h ago)   28h
webapp-d85n7    1/1     Running   0             10h
webapp-lpf9l    1/1     Running   0             23h
webapp-ngmkh    1/1     Running   0             23h
webapp-nw2vv    1/1     Running   1 (28h ago)   28h
webapp-q2pp4    1/1     Running   1 (28h ago)   28h
webapp-w2fbf    1/1     Running   0             23h
webapp-zxnc4    1/1     Running   0             10h
ubuntu@ip-172-31-47-144:~$
```

you can delete the specific pod also -> kubectl delete pod_name

Note : it will regenerate pod because of replicas



✅ **In Companies (Especially on AWS / EKS):**

Concept	What's Used in Practice
Pods	❌ Not created directly
ReplicaSets	⚙️ Used indirectly via Deployments
ReplicationControllers	❌ Deprecated / not used
Deployments	✅ Used to manage Pods & ReplicaSets
EKS (Elastic Kubernetes Service)	✅ Managed Kubernetes Control Plane
Helm Charts	✅ Used for templating & versioning apps
CI/CD Tools (e.g., ArgoCD, GitLab, Jenkins, CodePipeline)	✅ For automated deployment pipelines

Deployment in Kubernetes

Feature	ReplicaSet	Deployment
Manages Pods	✅ Yes	✅ Yes (via ReplicaSet)
Rolling Updates	❌ No	✅ Yes
Rollbacks	❌ No	✅ Yes
YAML Kind	ReplicaSet	Deployment
Use in Real World	Rare	Very Common

A Deployment in Kubernetes is one of the most recommended and used resources for managing Pod lifecycles. It ensures reliable application deployment with features like zero downtime, auto-scaling, rolling updates, and rollback capabilities.

🎯 **Key Advantages of Using Deployments**
Zero Downtime: Deployments ensure high availability by using strategies like rolling updates. Even when pods are being updated, the service remains available to users.

Auto Scaling: With Kubernetes Horizontal Pod Autoscaler, you can automatically scale your Pods based on CPU or memory usage (or other custom metrics).

Rolling Update & Rollback: Kubernetes allows rolling updates, which means it will gradually update Pods one by one, ensuring the application remains available throughout the process. If something goes wrong during the update, you can rollback to a previous stable version.

👨‍🔧 **When to Choose Which Strategy?**
Use **RollingUpdate**:

For most production workloads where high availability and zero downtime are required.

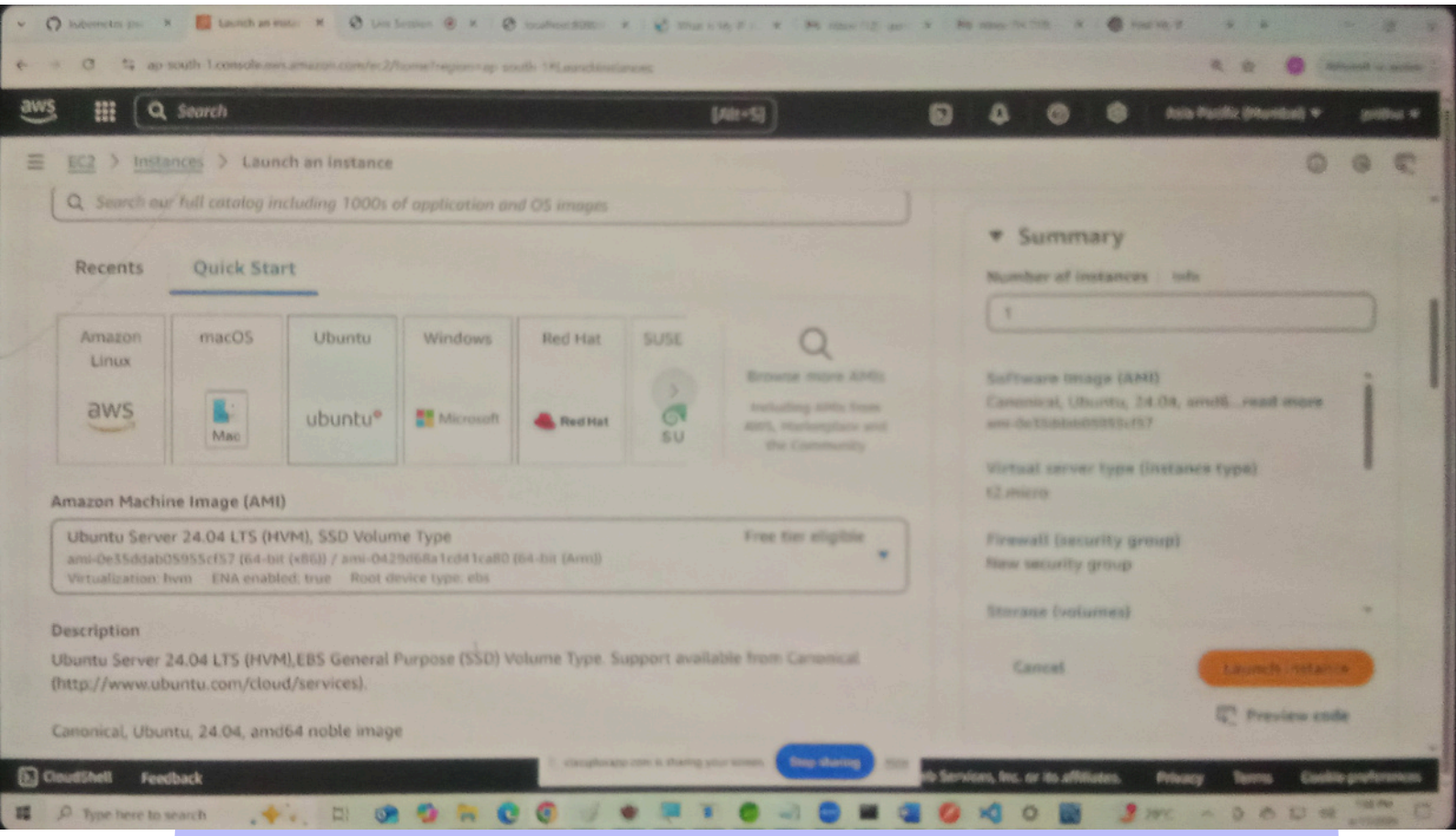
When you are gradually releasing new versions of your application and want to avoid service disruption.

Use Recreate:

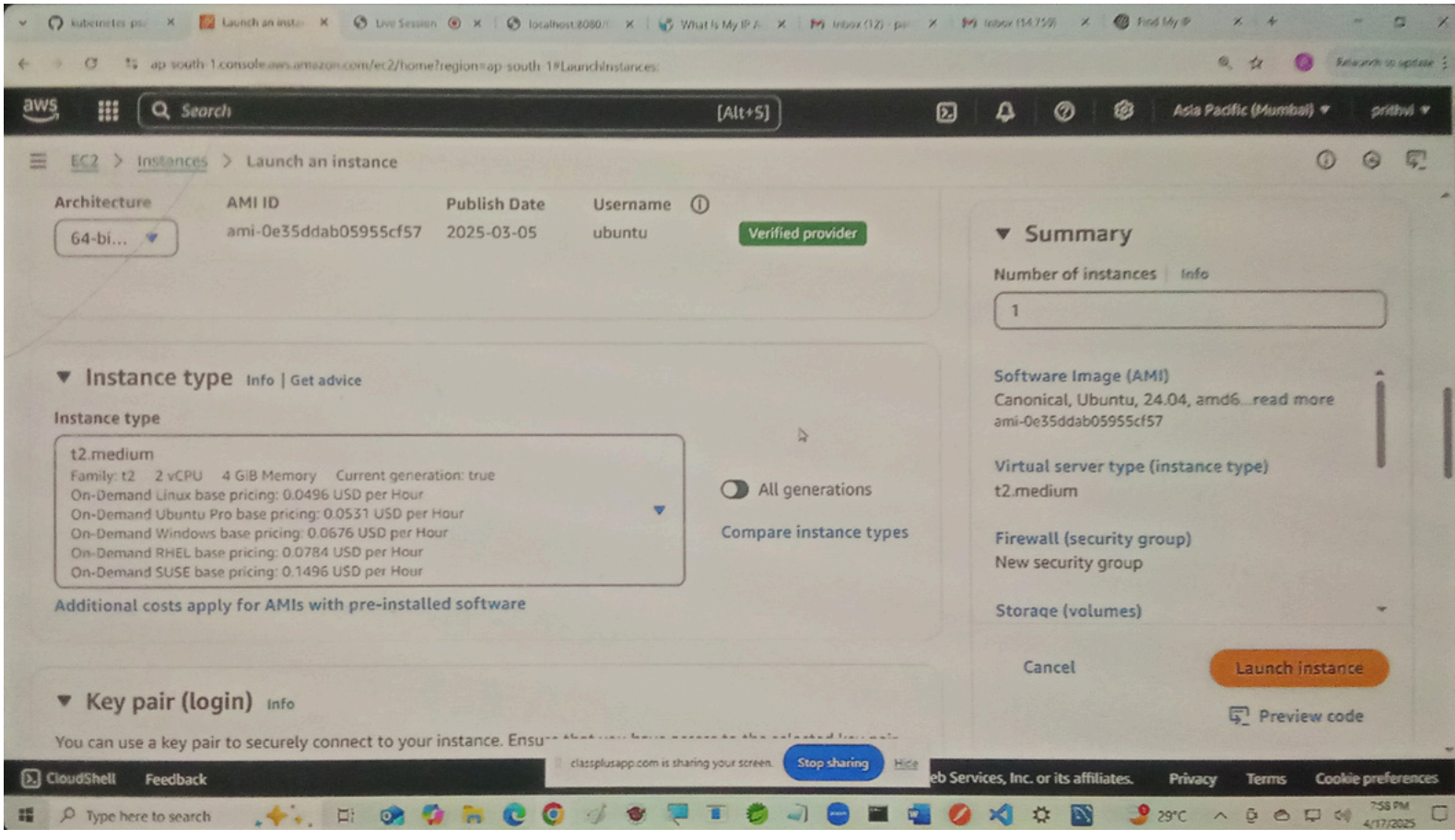
For non-critical applications or during maintenance windows where it's okay to have a temporary outage.

When you need to clear everything and redeploy fresh Pods (e.g., clearing persistent state or major upgrades).

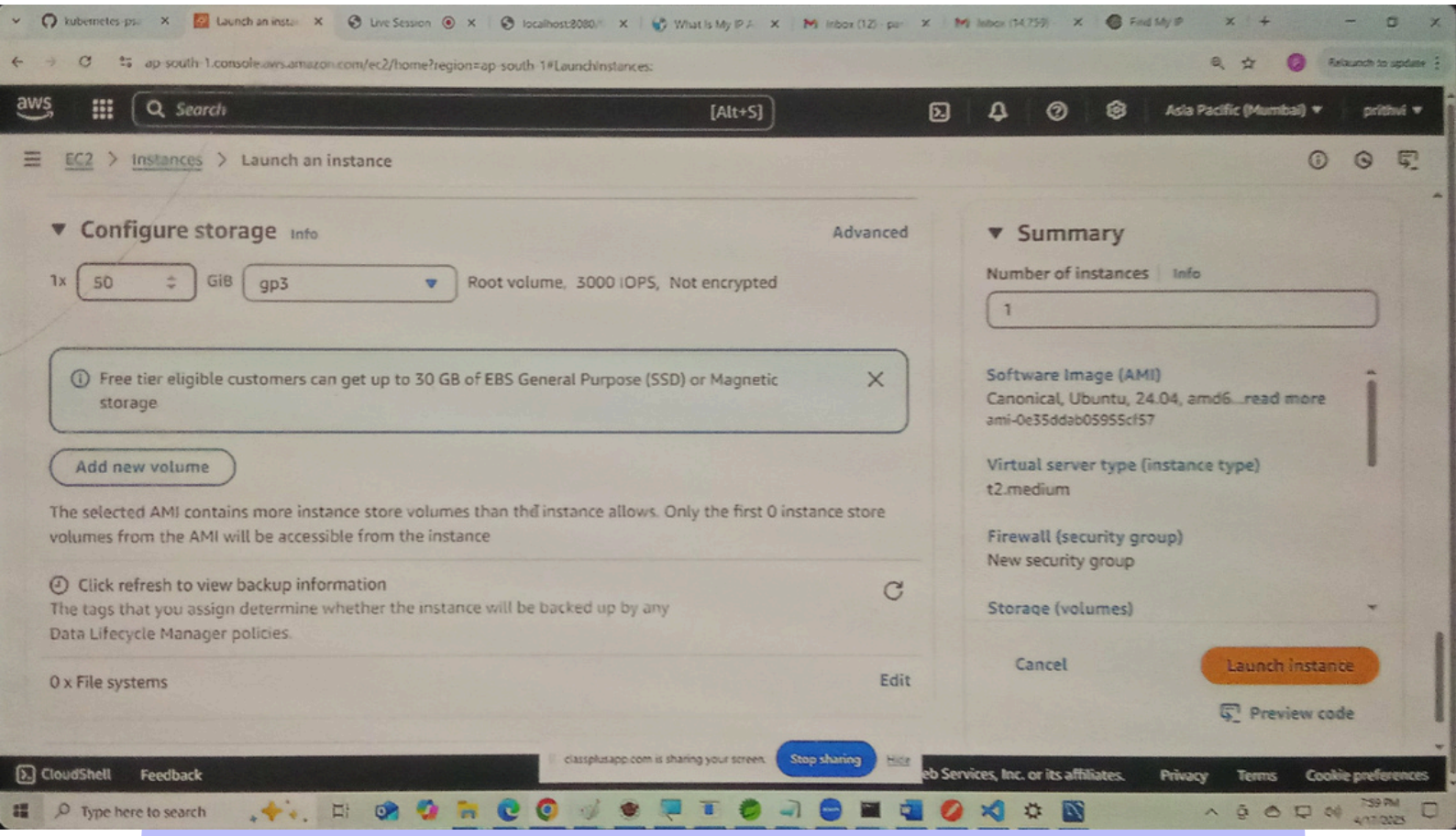
1 choose AMI ubantu or based on requirnment



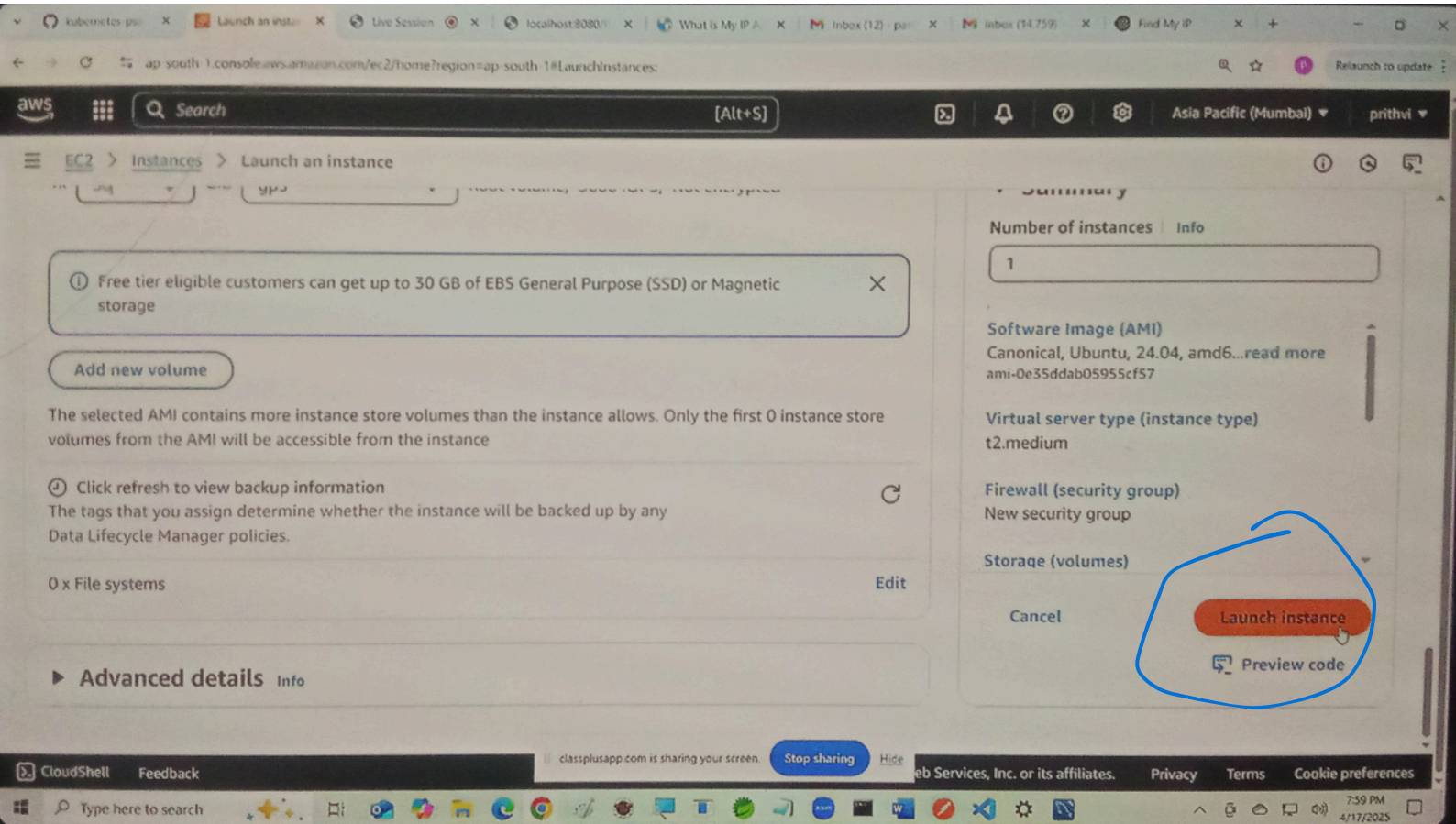
2 choose instance type t2.midium or higher



3 select storage minimum 50 gb

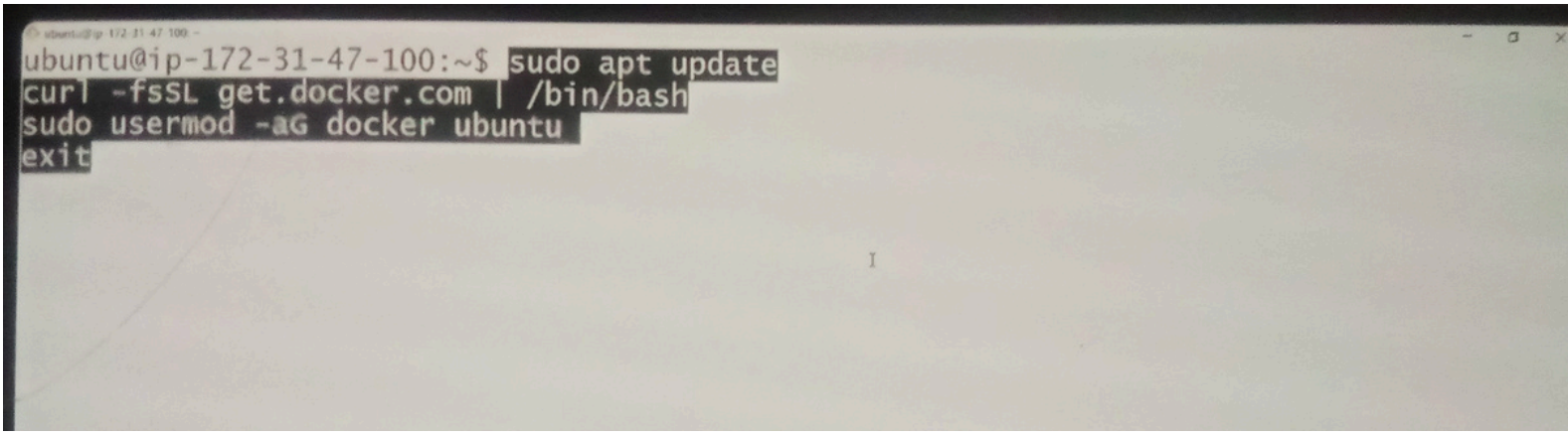


4 launch instance



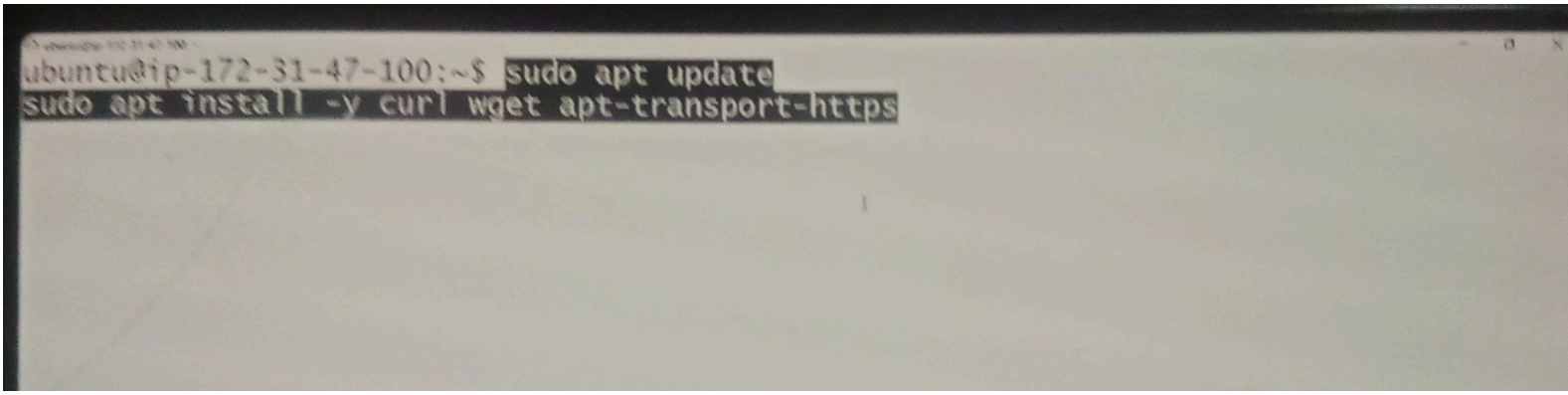
5 Install Docker In Ubuntu VM

sudo apt update
curl -fsSL get.docker.com | /bin/bash
sudo usermod -aG docker ubuntu
exit



6 Updating system packages and installing Minikube dependencies

sudo apt update
sudo apt install -y curl wget apt-transport-https

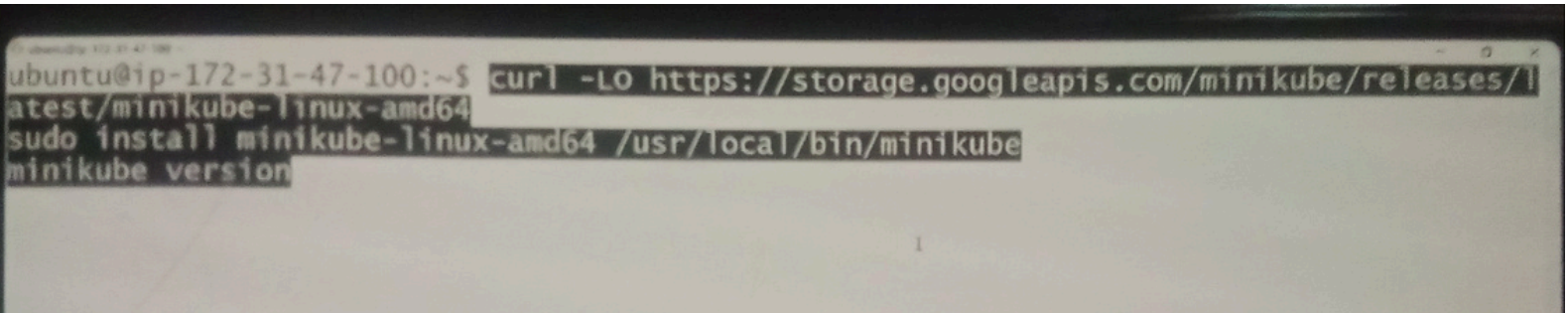


7 Installing Minikube

curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64

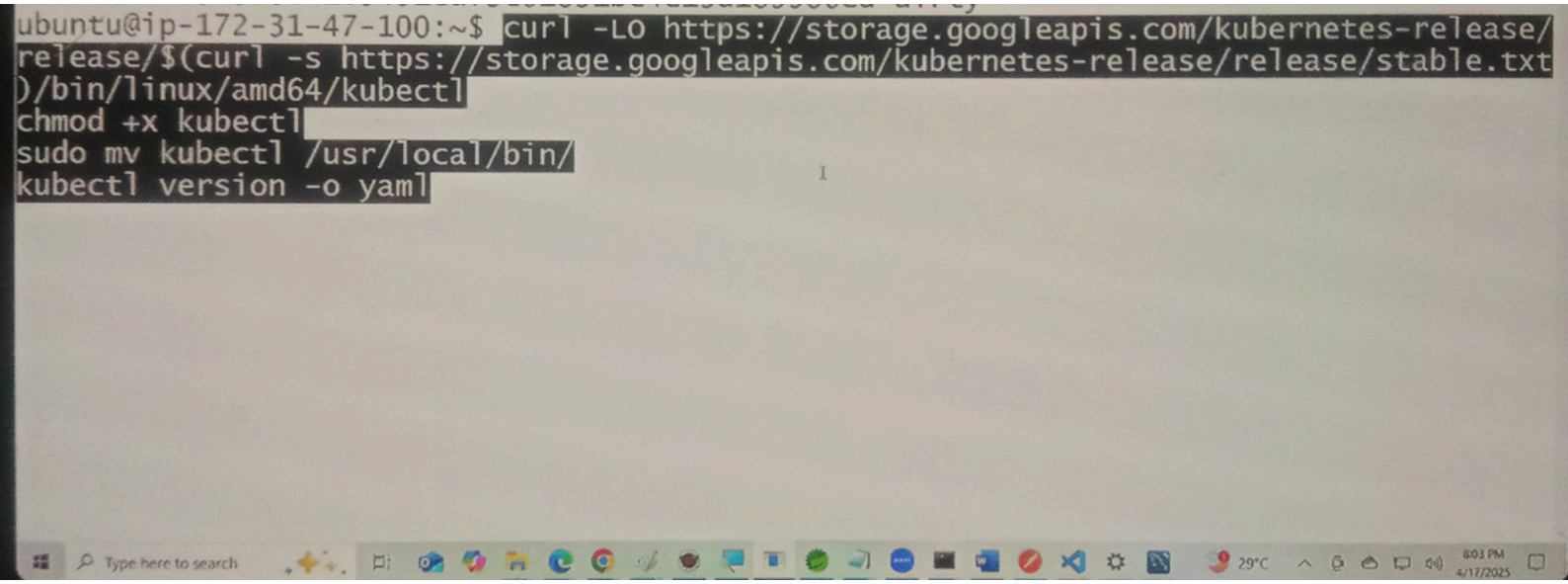
sudo install minikube-linux-amd64 /usr/local/bin/minikube

minikube version

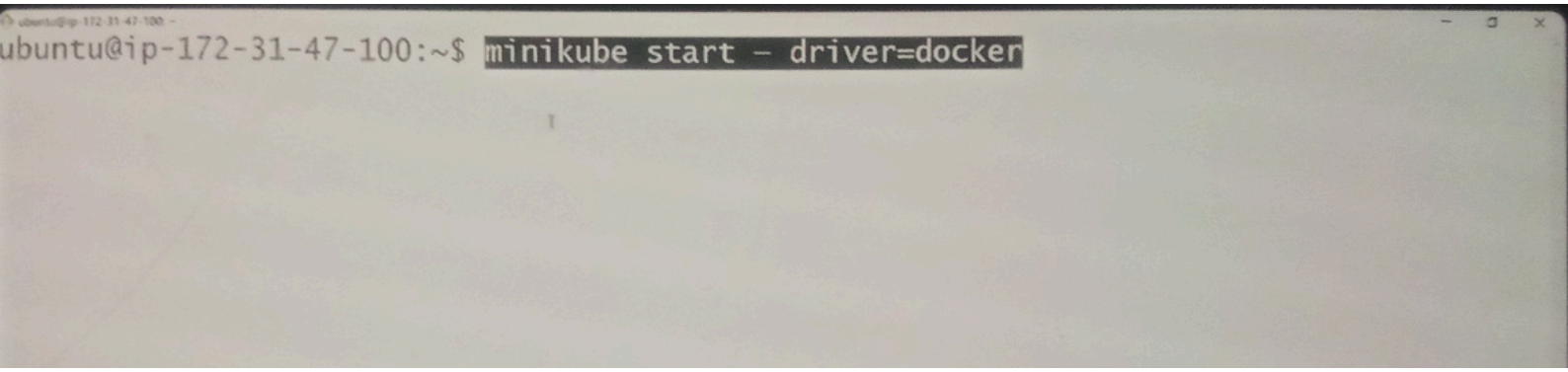


8 Install Kubectl (Kubernetes Client)

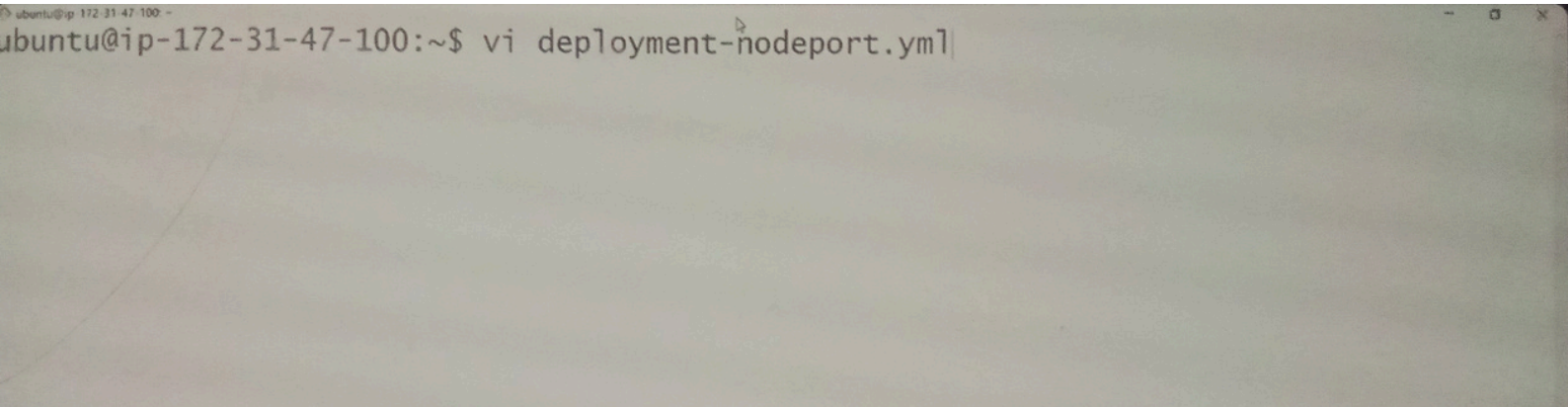
curl -LO https://storage.googleapis.com/kubernetes-release/release/$(curl -s https://storage.googleapis.com/kubernetes-release/release/stable.txt)/bin/linux/amd64/kubectl
chmod +x kubectl
sudo mv kubectl /usr/local/bin/
kubectl version -o yaml



9 Start MiniKube Server



10 create vi name.yml



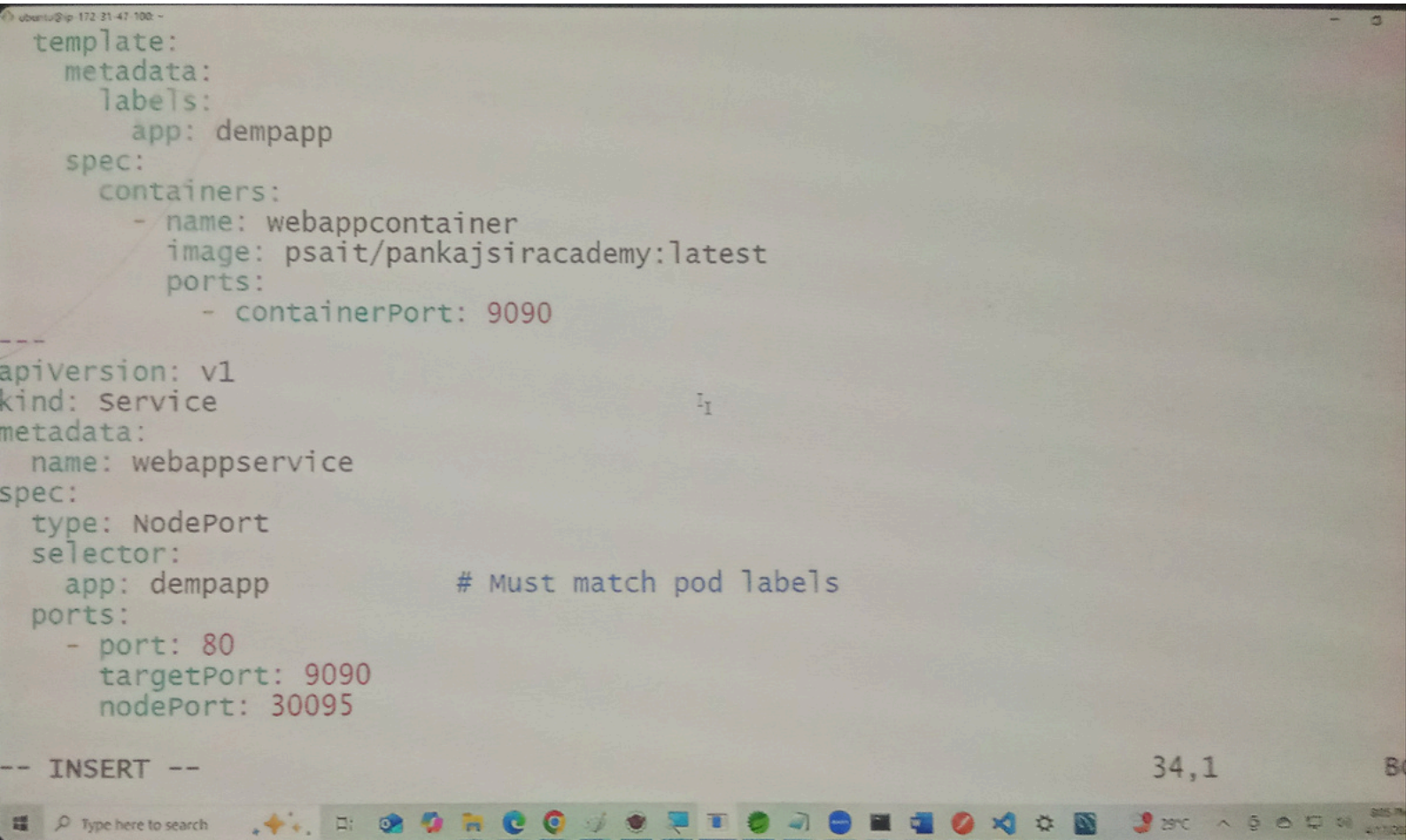
11 write code in yml

```yaml
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: webapp
spec:
  replicas: 3
  selector:
    matchLabels:
      app: dempapp        # Must match pod template labels
  template:
    metadata:
      labels:
        app: dempapp
    spec:
      containers:
        - name: webappcontainer
          image: psait/pankajsiracademy:latest
          ports:
            - containerPort: 9090
---
apiVersion: v1
kind: Service
metadata:
  name: webappservice
spec:
  type: NodePort
  selector:
    app: dempapp          # Must match pod labels
  ports:
    - port: 80
      targetPort: 9090
      nodePort: 30095
```

→ Resource type

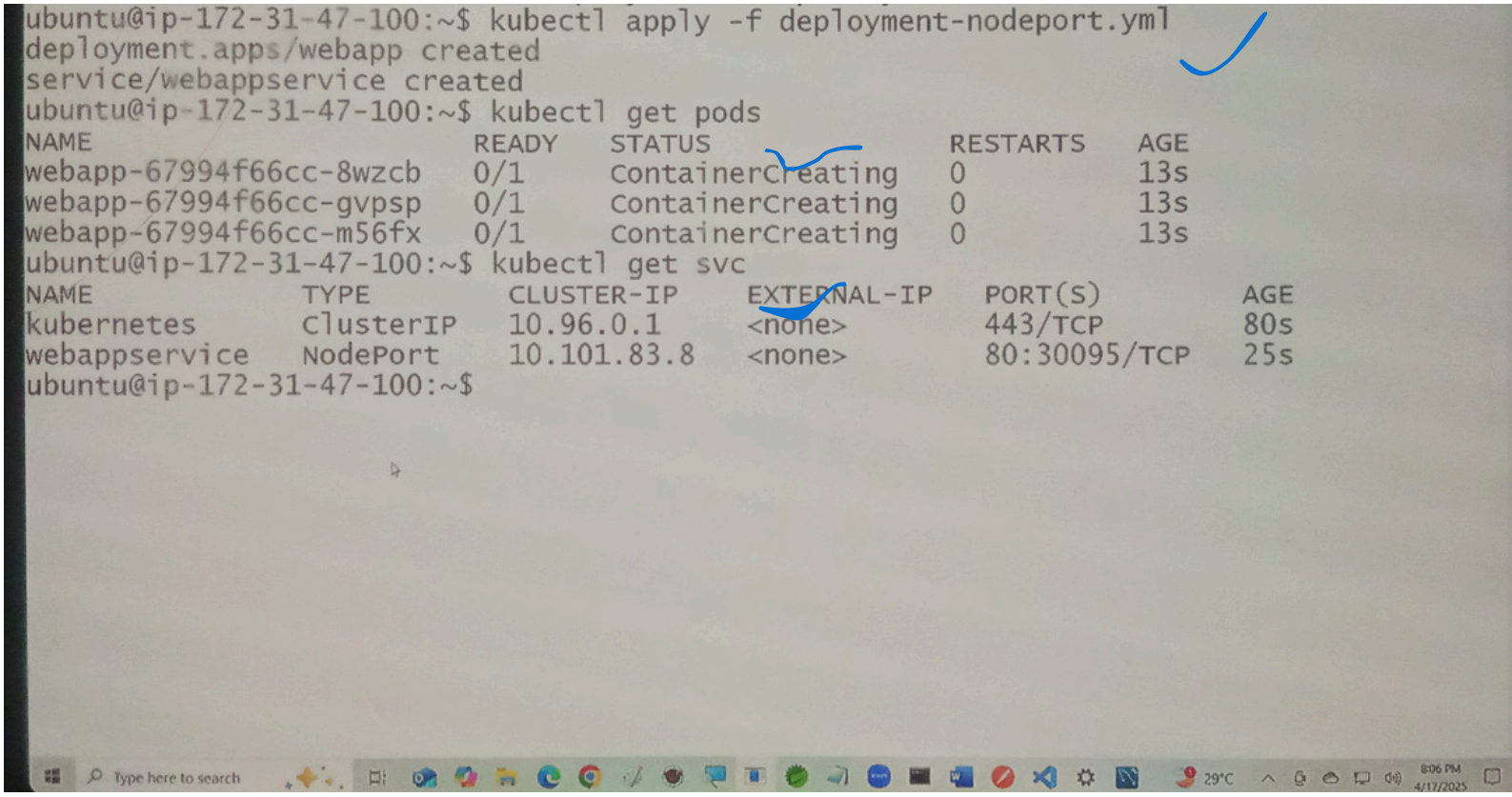
note : check yml syntax

→ service type



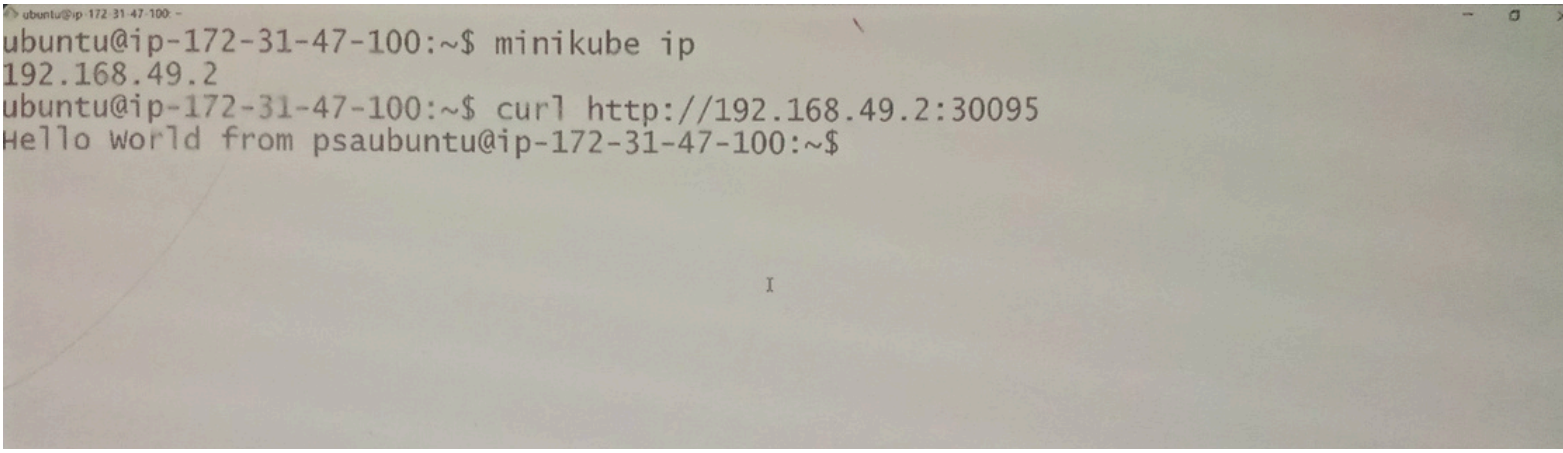
12

- to configure → kubectl apply -f name.yml
- to check pod--> kubectl get pods
- to check service → kubectl get svc



13

- check ip address of minikube--> minikube ip
- to hit your application use curl--> curl+ip-Address



14

- to check service → kubectl get svc
- to delete service(note service is deleting not source it means accessing application not possoble now)

→ kubectl delete svc service_name

- see here pod is not deted check--> kubectl get pods

```
ubuntu@ip-172-31-47-100:~$ kubectl get svc
NAME             TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)          AGE
kubernetes       ClusterIP   10.96.0.1      <none>        443/TCP          4m1s
webappservice    NodePort    10.101.83.8    <none>        80:30095/TCP     3m6s
ubuntu@ip-172-31-47-100:~$ kubectl delete svc webappservice
service "webappservice" deleted
ubuntu@ip-172-31-47-100:~$ kubectl get pods
NAME                      READY   STATUS    RESTARTS   AGE
webapp-67994f66cc-8wzcb   1/1     Running   0          3m31s
webapp-67994f66cc-gvpsp   1/1     Running   0          3m31s
webapp-67994f66cc-m56fx   1/1     Running   0          3m31s
ubuntu@ip-172-31-47-100:~$
```

15

- you can delete pod here (note but it will regenerate because of replicas)

→ kubectl delete pod pod_name

- see here pod again generated → kubectl get pods

```
ubuntu@ip-172-31-47-100:~$ kubectl delete pod webapp-67994f66cc-8wzcb
pod "webapp-67994f66cc-8wzcb" deleted
ubuntu@ip-172-31-47-100:~$ kubectl get pods
NAME                      READY   STATUS    RESTARTS   AGE
webapp-67994f66cc-gvpsp   1/1     Running   0          4m11s
webapp-67994f66cc-m56fx   1/1     Running   0          4m11s
webapp-67994f66cc-wkvwc   1/1     Running   0          3s
ubuntu@ip-172-31-47-100:~$
```

then how to delete pods ?

use source metadata name to delete pod permanently

source metadata name is present in yml

-- > kubectl delete resource_type metadata_name

NOTE : OR You can make replicas=0

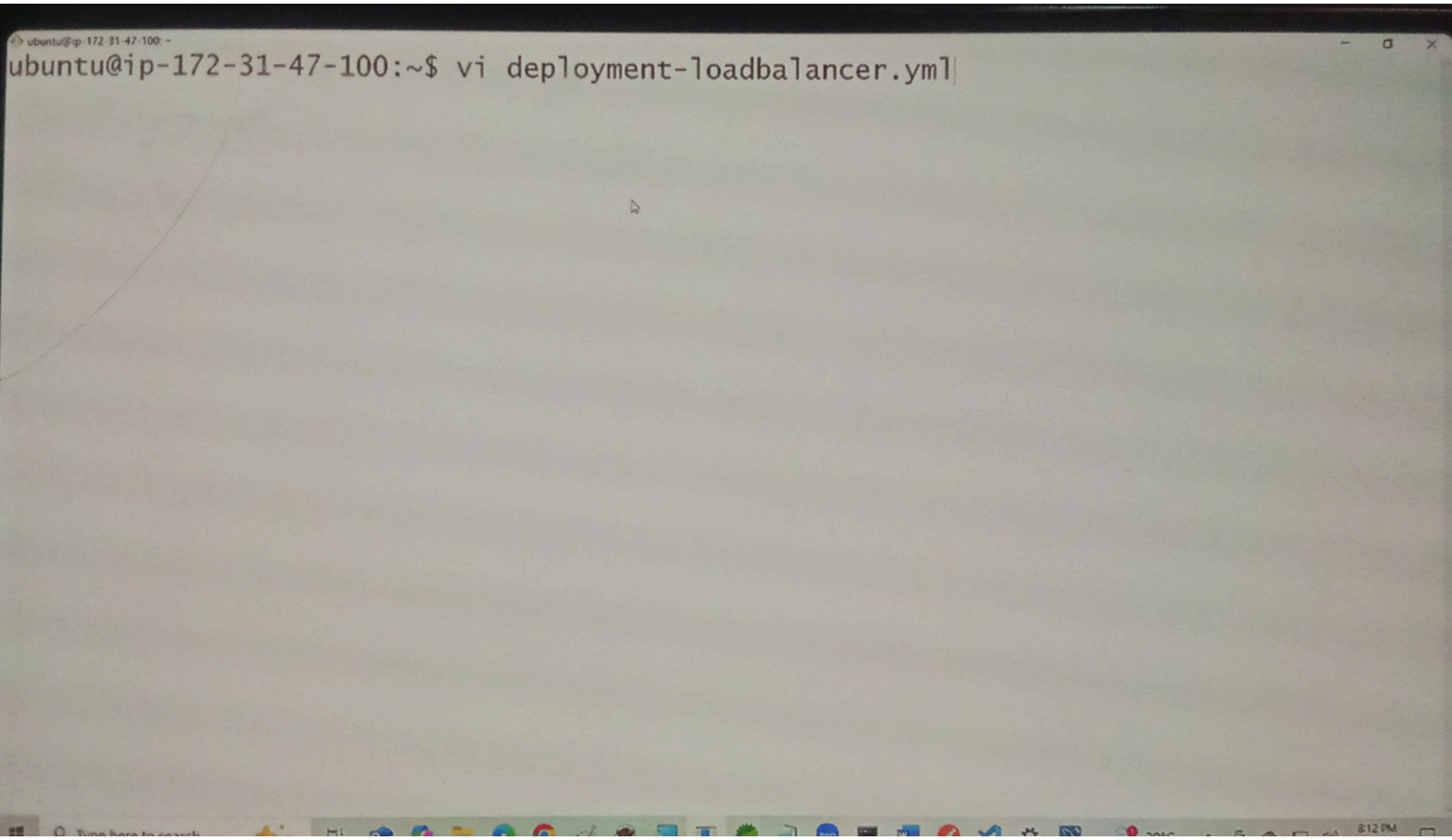
--> kubectl scale --replicas=0 <resource_type> <metadata_name>

```
ntu@ip-172-31-47-100:~$ kubectl delete deployment webapp
loyment.apps "webapp" deleted
ntu@ip-172-31-47-100:~$ kubectl get pods
 resources found in default namespace.
ntu@ip-172-31-47-100:~$
```

see here pod is not found

Till now we used **service type nodeport** it is publically accessible but not load balance works so now use **service type** is **LoadBalancer**

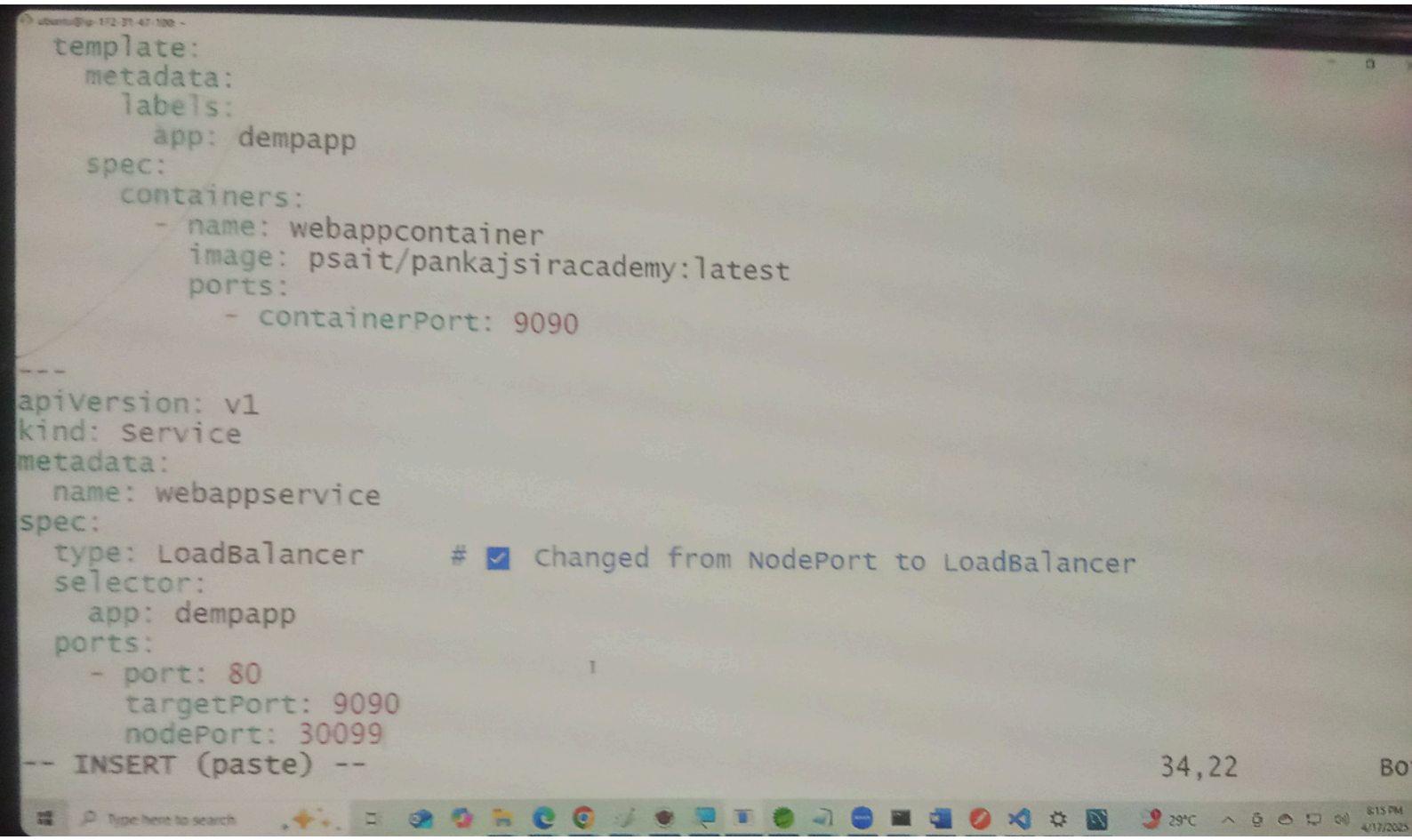
1 **create vi name.yml**



write code in yml

```yaml
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: webapp
spec:
  replicas: 3
  selector:
    matchLabels:
      app: dempapp        # Must match pod template labels
  template:
    metadata:
      labels:
        app: dempapp
    spec:
      containers:
        - name: webappcontainer
          image: psait/pankajsiracademy:latest
          ports:
            - containerPort: 9090

---
apiVersion: v1
kind: Service
metadata:
  name: webappservice
spec:
  type: LoadBalancer    # ✅ Changed from NodePort to LoadBalancer
  selector:
    app: dempapp
  ports:
    - port: 80
      targetPort: 9090
```



- to configure → kubectl apply -f name.yml
- check ip address of minikube--> minikube ip
- to hit your application use curl--> curl+ip-Address