Ques 1 : **Deploy Replica Set and Replication Controller, and deployment. Also learn the advantages and disadvantages of each.**

SOLN:

FIRST WE WILL DEPLOY THESE USING BASIC KUBERNETES LOCAL CLUSTER AND TRY TO CONFIGURE THE

1> REPLICATION CONTROLLER

2> REPLICA SET

3> DEPLOYMENT

Steps to Deploy ReplicationController, ReplicaSet & Deployment Using `kubeadm` Kubernetes Cluster :

Prerequisites

Ensure:

- **You have 2 Linux VMs (Ubuntu preferred) — 1 master + 1 worker**
- **`kubeadm`, `kubelet`, `kubectl`, `docker/containerd` are installed**

SO LETS GO WITH THE INSTALLATIONS :

COMMANDS I HAVE WRITTEN DOWN HERE FOR THE FULL INSTALLATIONS WE HAVE DONE IT IN ASSIGNMENT 5 FROM THE SRATCH

Kubernetes & Containerd Setup (All-in-One Script)

```
sudo apt update && sudo apt install -y apt-transport-https curl ca-certificates gnupg lsb-release && curl -fsSL https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo gpg --dearmor -o /usr/share/keyrings/kubernetes-archive-keyring.gpg &&
```

```
echo "deb [signed-by=/usr/share/keyrings/kubernetes-archive-keyring.gpg] https://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee /etc/apt/sources.list.d/kubernetes.list && sudo apt update && sudo apt install -y kubelet kubeadm kubectl containerd && sudo apt-mark hold kubelet kubeadm kubectl
```

🔗 **After that, configure containerd (required):**

```
sudo mkdir -p /etc/containerd && containerd config default | sudo tee /etc/containerd/config.toml > /dev/null && sudo systemctl restart containerd && sudo systemctl enable containerd
```

Local Kubernetes Setup (kubeadm)

1 Initial Setup on Master Node

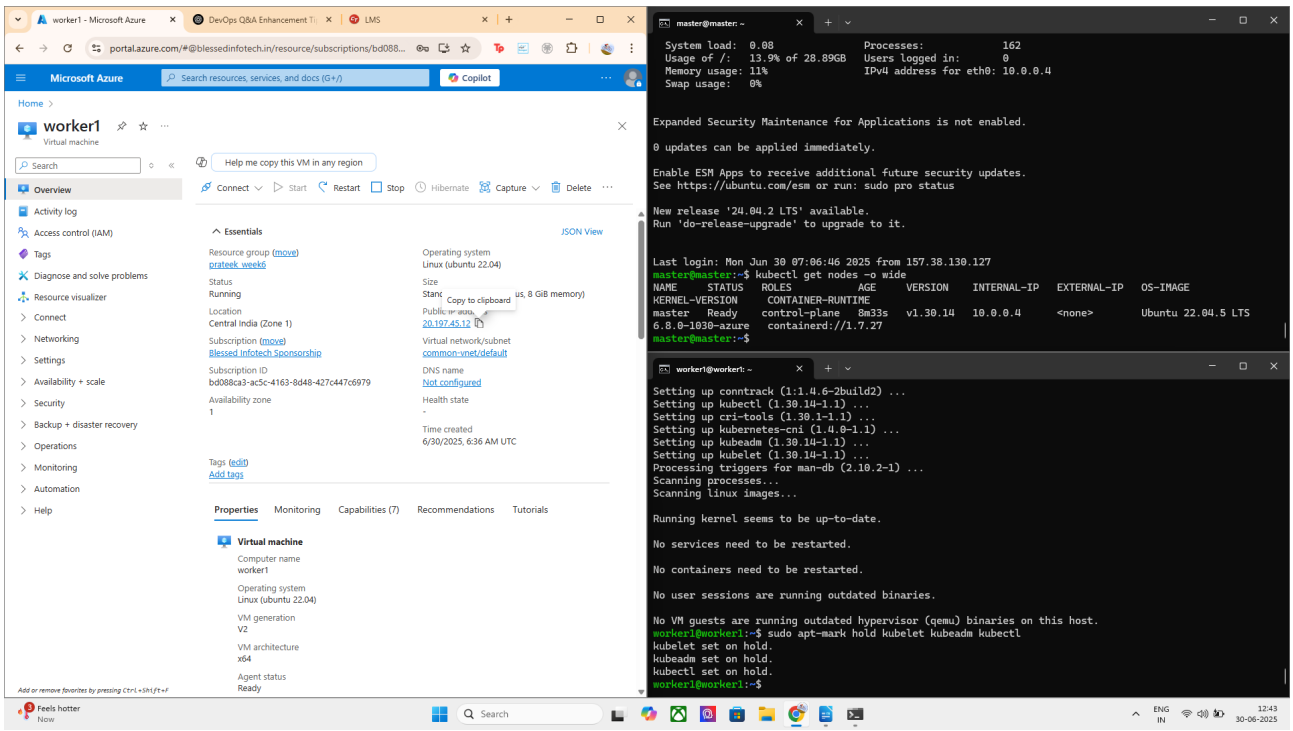
sudo kubeadm init –pod-network-cidr=192.168.0.0/16

2 Configure kubectl for Master

```
mkdir -p $HOME/.kube
sudo cp /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

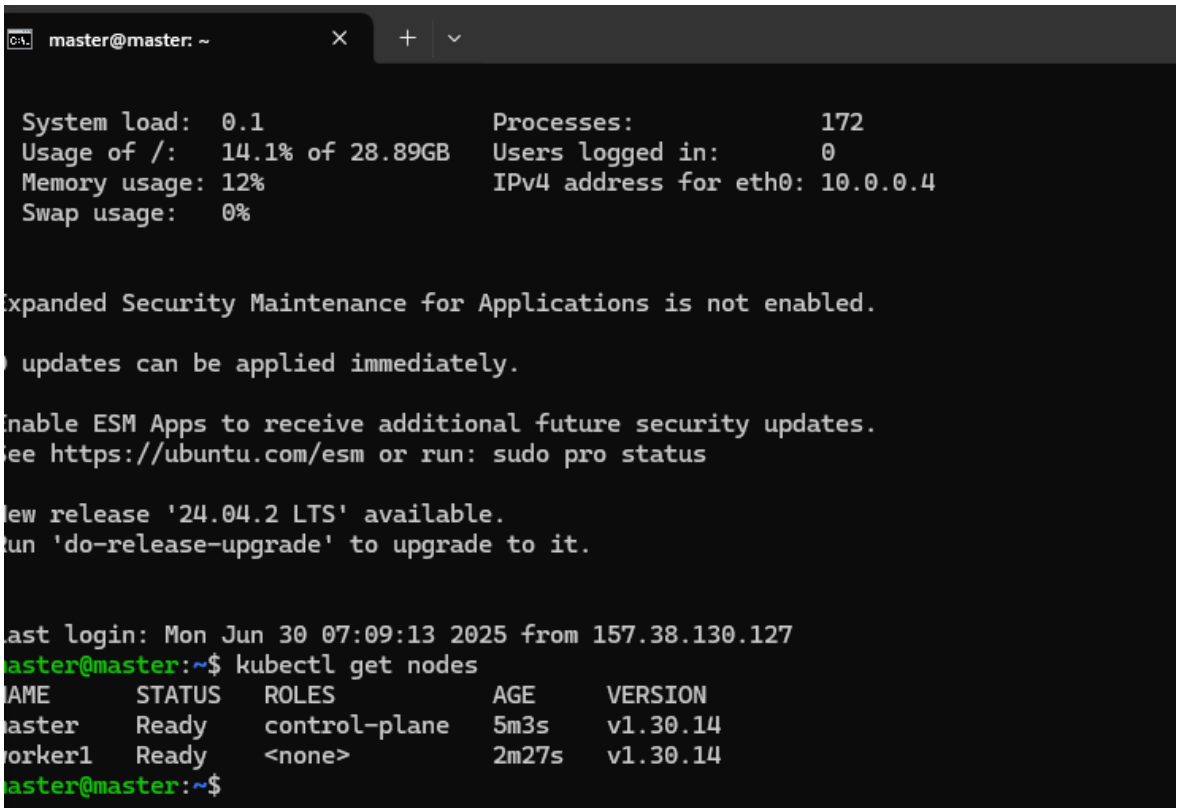
3 Install Pod Network (e.g., Calico or Flannel)

kubectlapply-f https://raw.githubusercontent.com/projectcalico/calico/v3.26.1/manifests/calico.yaml



Join Worker Node to Cluster
sudo kubeadm join <master-ip>:6443 --token <token> --discovery-token-ca-cert-hash sha256:<hash>

kubectl get nodes
kubectl get pods -A



```
master@master: ~
  GNU nano 6.2                    replicationcontroller.yaml *
apiVersion: v1
kind: ReplicationController
metadata:
  name: nginx-rc
spec:
  replicas: 2
  selector:
    app: nginx-rc
  template:
    metadata:
      labels:
        app: nginx-rc
    spec:
      containers:
      - name: nginx
        image: nginx:1.14
        ports:
        - containerPort: 80

^G Help      ^O Write Out  ^W Where Is   ^K Cut       ^T Execute    ^C Location   M-U Undo
^X Exit      ^R Read File  ^\ Replace    ^U Paste     ^J Justify    ^/ Go To Line M-E Redo
```

master@master:~$ kubectl apply -f replicationcontroller.yaml --validate=false
replicationcontroller/nginx-rc created

```
Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status

New release '24.04.2 LTS' available.
Run 'do-release-upgrade' to upgrade to it.


Last login: Mon Jun 30 07:25:48 2025 from 157.38.130.127
master@master:~$ kubectl get nodes
NAME      STATUS   ROLES           AGE   VERSION
master    Ready    control-plane   26m   v1.30.14
worker1   Ready    <none>          23m   v1.30.14
master@master:~$ kubectl apply -f replicationcontroller.yaml
error: error validating "replicationcontroller.yaml": error validating data: failed t
ttps://10.0.0.4:6443/openapi/v2?timeout=32s": dial tcp 10.0.0.4:6443: connect: connec
e to ignore these errors, turn validation off with --validate=false
master@master:~$ kubectl get nodes
NAME      STATUS   ROLES           AGE   VERSION
master    Ready    control-plane   26m   v1.30.14
worker1   Ready    <none>          24m   v1.30.14
master@master:~$ kubectl apply -f replicationcontroller.yaml --validate=false
replicationcontroller/nginx-rc created
master@master:~$
```

```
master@master:~$ kubectl get nodes
NAME      STATUS   ROLES           AGE   VERSION
master    Ready    control-plane   26m   v1.30.14
worker1   Ready    <none>          24m   v1.30.14
master@master:~$ kubectl apply -f replicationcontroller.yaml --validate=false
replicationcontroller/nginx-rc created
master@master:~$ kubectl get rc
NAME       DESIRED   CURRENT   READY   AGE
nginx-rc   2         2         2       110s
master@master:~$ kubectl get pods -o wide
NAME             READY   STATUS    RESTARTS   AGE    IP                NODE      NOMINATED NODE   READINESS GAT
ES
nginx-rc-f7vgw   1/1     Running   0          118s   192.168.235.130   worker1   <none>           <none>
nginx-rc-ff5b8   1/1     Running   0          118s   192.168.235.129   worker1   <none>           <none>
master@master:~$
```

Deploy Your Application

1. ReplicationController

```
kubectl apply -f replicationcontroller.yaml
```

2. ReplicaSet

```
kubectl apply -f replicaset.yaml
```

3. Deployment

```
kubectl apply -f deployment.yaml
```

4. Service

```
kubectl apply -f service.yaml
```

Step-by-Step: Create & Deploy `ReplicaSet`

📝 1. Create a file called `replicaset.yaml`

In your terminal:

```
nano replicaset.yaml
```

Paste the following content:

```
yaml
CopyEdit
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: nginx-rs
  labels:
    app: nginx
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:latest
        ports:
        - containerPort: 80
```

Save and exit:

- Press `Ctrl + O`, then `Enter` to save
- Press `Ctrl + X` to exit

2. Apply the ReplicaSet

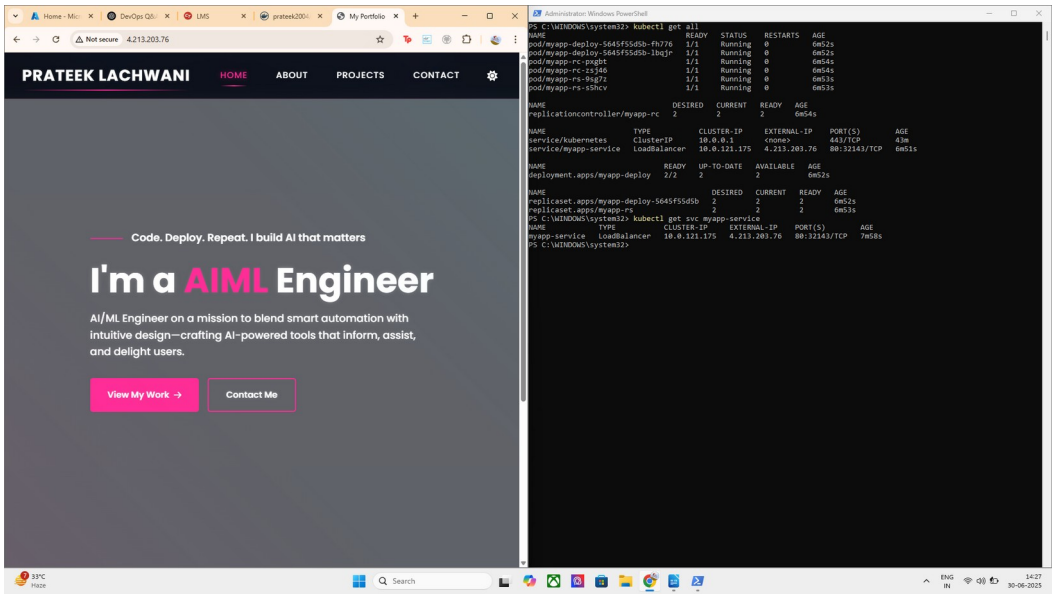
```
kubectl apply -f replicaset.yaml
```

3. Confirm it’s Running

```
kubectl get rs
kubectl get pods -l app=nginx
```

This should show 2 pods created by the ReplicaSet.

4. Bonus (Optional): Delete one pod an



Lets configure the same using cloud native development :

Using AKS (Azure Kubernetes services):

FIRST CREATE A AKS CLUSTER USING AZURE PORTAL AND THE CONFIGURATIONS AS GIVEN BELOW :

THEN

1. Connect to AKS Cluster

```
command used = az aks get-credentials --resource-group <your-resource-group> --name <your-cluster-name>
kubectl get nodes  # Confirm connection
```

```
PS C:\WINDOWS\system32> az aks get-credentials --resource-group prateek_cluster --name new_kuber
A different object named new_kuber already exists in your kubeconfig file.
Overwrite? (y/n): y
A different object named clusterUser_prateek_cluster_new_kuber already exists in your kubeconfig file.
Overwrite? (y/n): y
Merged "new_kuber" as current context in C:\Users\prate\.kube\config
PS C:\WINDOWS\system32> kubectl get nodes
NAME                                STATUS   ROLES    AGE   VERSION
aks-agentpool-92868701-vmss000000   Ready    <none>   29m   v1.31.8
aks-agentpool-92868701-vmss000001   Ready    <none>   29m   v1.31.8
aks-userpool-92868701-vmss000000    Ready    <none>   29m   v1.31.8
aks-userpool-92868701-vmss000001    Ready    <none>   29m   v1.31.8
PS C:\WINDOWS\system32>
```

2. Package Your App into a Docker Image

THAT I ALREADY HAD YOU CAN ALSO ACCESS THE IMAGE AT THIS

3. Write 3 Separate YAML Files

a. ReplicationController YAML

```
@"
apiVersion: v1
kind: ReplicationController
metadata:
  name: myapp-rc
spec:
```

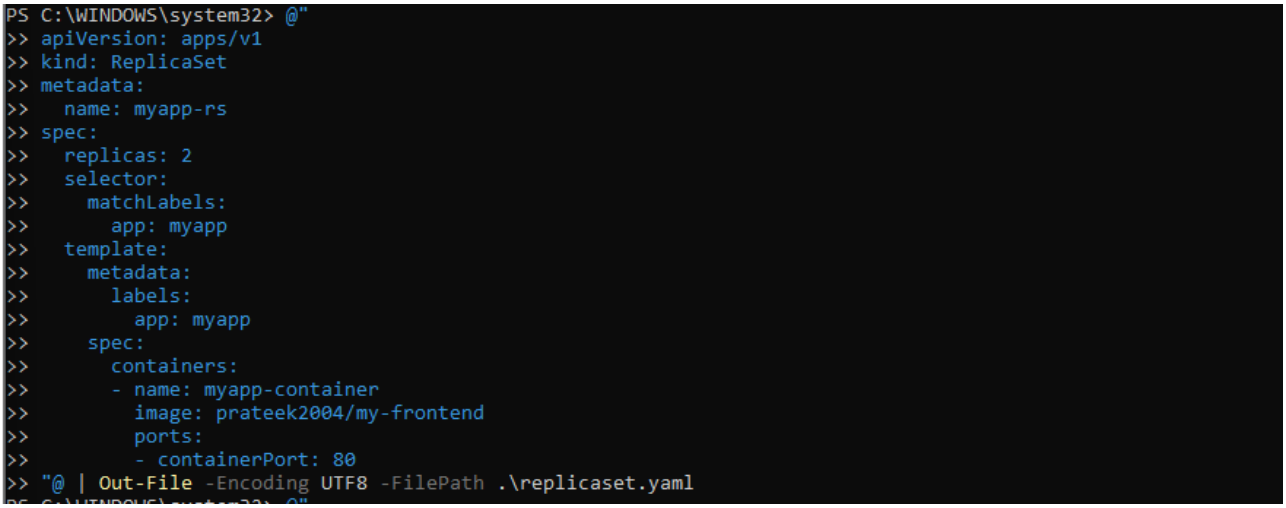
```
  replicas: 2
  selector:
    app: myapp
  template:
    metadata:
      labels:
        app: myapp
    spec:
      containers:
      - name: myapp-container
        image: prateek2004/my-frontend
        ports:
        - containerPort: 80
"@ | Out-File -Encoding UTF8 -FilePath .\replicationcontroller.yaml
```

```
PS C:\WINDOWS\system32> @"
>> apiVersion: v1
>> kind: ReplicationController
>> metadata:
>>   name: myapp-rc
>> spec:
>>   replicas: 2
>>   selector:
>>     app: myapp
>>   template:
>>     metadata:
>>       labels:
>>         app: myapp
>>     spec:
>>       containers:
>>       - name: myapp-container
>>         image: prateek2004/my-frontend
>>         ports:
>>         - containerPort: 80
>> "@ | Out-File -Encoding UTF8 -FilePath .\replicationcontroller.yaml
PS C:\WINDOWS\system32> @"
```

B. ReplicaSet YAML

```
@"
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: myapp-rs
spec:
  replicas: 2
  selector:
    matchLabels:
      app: myapp
```

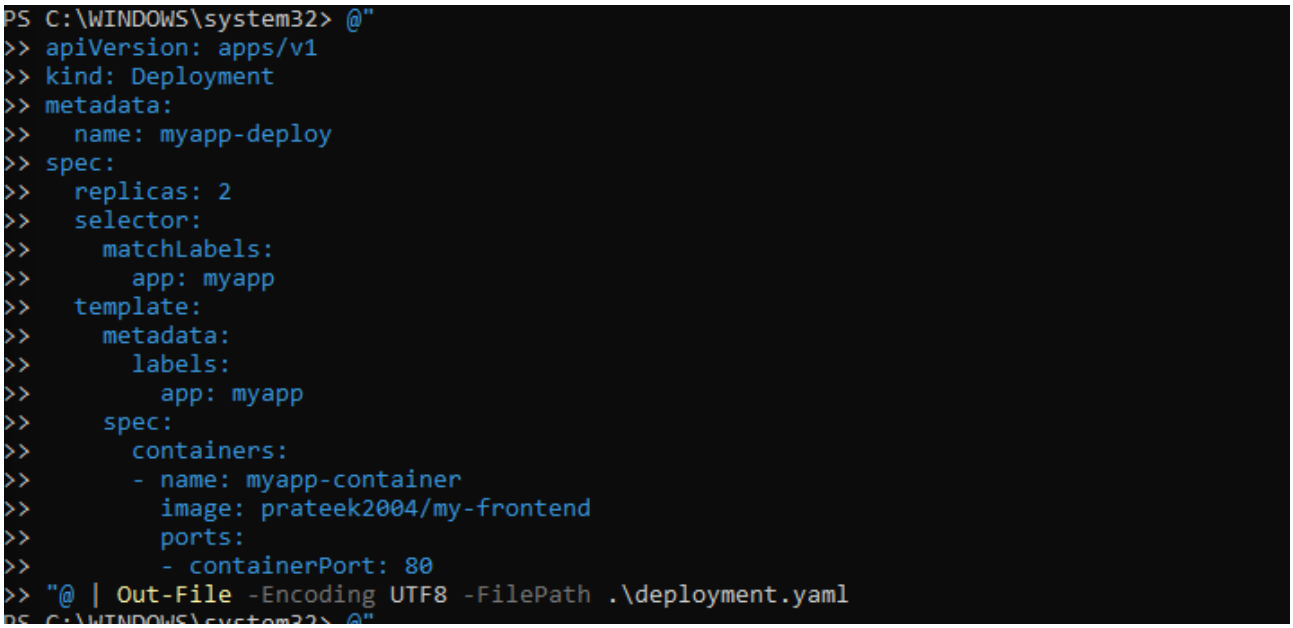
```
  template:
    metadata:
      labels:
        app: myapp
    spec:
      containers:
      - name: myapp-container
        image: prateek2004/my-frontend
        ports:
        - containerPort: 80
"@ | Out-File -Encoding UTF8 -FilePath .\replicaset.yaml
```



C. Deployment YAML

```
@"
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp-deploy
spec:
  replicas: 2
  selector:
    matchLabels:
      app: myapp
  template:
    metadata:
      labels:
```

```
      app: myapp
    spec:
      containers:
      - name: myapp-container
        image: prateek2004/my-frontend
        ports:
        - containerPort: 80
"@ | Out-File -Encoding UTF8 -FilePath .\deployment.yaml
```



NOW LETS ALSO CREATE A SERVICE CONFIGURATION FOR THE SAME DEPLOYMENT TO VISIBLE A PUBLIC IP :

Service YAML

```
@"
apiVersion: v1
kind: Service
metadata:
  name: myapp-service
spec:
  selector:
    app: myapp
  type: LoadBalancer
  ports:
    - port: 80
      targetPort: 80
"@ | Out-File -Encoding UTF8 -FilePath .\service.yaml
```

```
PS C:\WINDOWS\system32> @"
>> apiVersion: v1
>> kind: Service
>> metadata:
>>   name: myapp-service
>> spec:
>>   selector:
>>     app: myapp
>>   type: LoadBalancer
>>   ports:
>>     - port: 80
>>       targetPort: 80
>> "@ | Out-File -Encoding UTF8 -FilePath .\service.yaml
```

Apply Files to AKS :

command used :
kubectl apply -f .\replicationcontroller.yaml
kubectl apply -f .\replicaset.yaml
kubectl apply -f .\deployment.yaml
kubectl apply -f .\service.yaml

```
PS C:\WINDOWS\system32> kubectl apply -f .\replicationcontroller.yaml
>> kubectl apply -f .\replicaset.yaml
>> kubectl apply -f .\deployment.yaml
>> kubectl apply -f .\service.yaml
replicationcontroller/myapp-rc created
replicaset.apps/myapp-rs created
deployment.apps/myapp-deploy created
service/myapp-service created
PS C:\WINDOWS\system32>
```

Next Steps: Verify the Setup

1. ✅ Check All Resources

kubectl get all

EVERYTHING UPTO THE POINT :

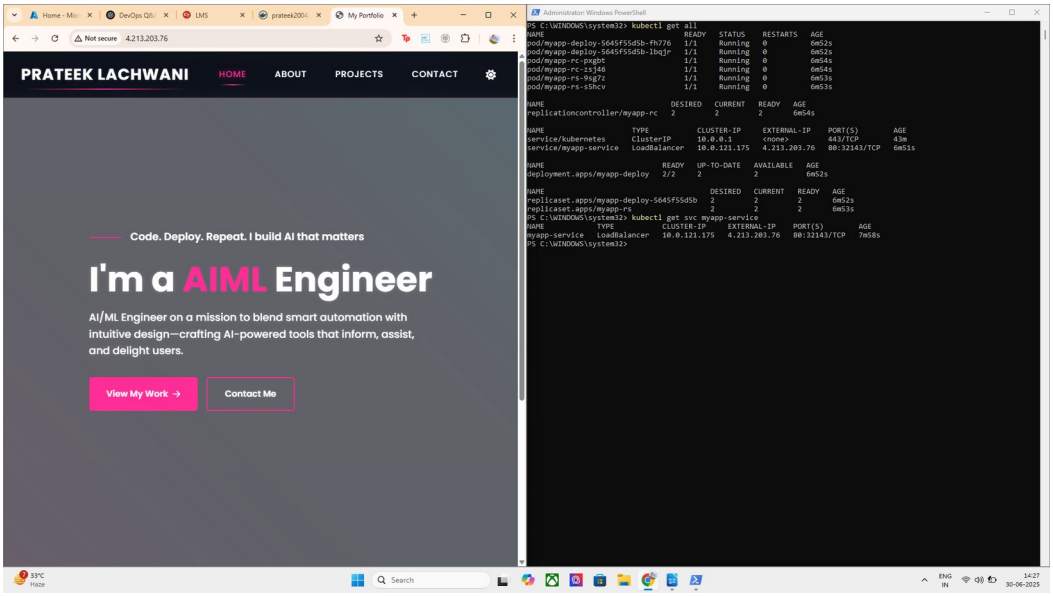
```
Administrator: Windows PowerShell
PS C:\WINDOWS\system32> kubectl get all
NAME                                    READY   STATUS    RESTARTS   AGE
pod/myapp-deploy-5645f55d5b-fh776       1/1     Running   0          6m52s
pod/myapp-deploy-5645f55d5b-lbqjr       1/1     Running   0          6m52s
pod/myapp-rc-pxgbt                      1/1     Running   0          6m54s
pod/myapp-rc-zsj46                      1/1     Running   0          6m54s
pod/myapp-rs-9sg7z                      1/1     Running   0          6m53s
pod/myapp-rs-s5hcv                      1/1     Running   0          6m53s

NAME                                DESIRED   CURRENT   READY   AGE
replicationcontroller/myapp-rc      2         2         2       6m54s

NAME                    TYPE           CLUSTER-IP     EXTERNAL-IP    PORT(S)        AGE
service/kubernetes      ClusterIP      10.0.0.1       <none>         443/TCP        43m
service/myapp-service   LoadBalancer   10.0.121.175   4.213.203.76   80:32143/TCP   6m51s

NAME                           READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/myapp-deploy   2/2     2            2           6m52s

NAME                                       DESIRED   CURRENT   READY   AGE
replicaset.apps/myapp-deploy-5645f55d5b    2         2         2       6m52s
replicaset.apps/myapp-rs                   2         2         2       6m53s
PS C:\WINDOWS\system32>
```

YOU CAN GET THE SERVICE USING :

kubectl get svc myapp-service

```
PS C:\WINDOWS\system32> kubectl get svc myapp-service
NAME            TYPE           CLUSTER-IP     EXTERNAL-IP    PORT(S)        AGE
myapp-service   LoadBalancer   10.0.121.175   4.213.203.76   80:32143/TCP   7m58s
PS C:\WINDOWS\system32>
```

URL: http://**4.213.203.76** it can be accessed on the web



Recap of What We've Done:

Component	Status	Description
ReplicationController	✅ Created	Manages legacy pods (less commonly used now)
ReplicaSet	✅ Created	Modern controller ensuring fixed number of identical pods
Deployment	✅ Created	Manages rollout, updates, rollback over ReplicaSets
Service	✅ LoadBalancer	Exposes app to the internet
Docker Image	✅ `prateek2004/my-frontend`	custom personal web app

Advantages and Disadvantages

Controller Type	Advantages	Disadvantages
ReplicationController	- Ensures desired pod count- Was Kubernetes' first controller	- **Deprecated** for most use-cases- No rolling updates or rollback
ReplicaSet	- Ensures desired number of pods- Supports label-based selection	- Doesn't support **rollbacks or declarative updates** on its own
Deployment	- Built on top of ReplicaSet- Supports **rolling updates**, rollback, pause/resume- Most widely used in real-world	- Slightly more complex YAML structure

To check the **use case and behavior** of each controller — **ReplicationController**, **ReplicaSet**, and **Deployment (manager)** — here's how you can **observe, test, and differentiate** them practically

1. **Check Which Pod Was Created by Which Controller**

kubectl get pods –show-labels
kubectl describe pod <pod-name>

```
PS C:\WINDOWS\system32> kubectl describe pod myapp-rc-pxgbt
Name:             myapp-rc-pxgbt
Namespace:        default
Priority:         0
Service Account:  default
Node:             aks-userpool-92868701-vmss000000/10.224.0.5
Start Time:       Mon, 30 Jun 2025 14:12:53 +0530
Labels:           app=myapp
Annotations:      <none>
Status:           Running
IP:               10.244.2.27
IPs:
  IP:           10.244.2.27
Controlled By:  ReplicationController/myapp-rc
Containers:
  myapp-container:
    Container ID:   containerd://24b2d74109f7e904bc55e20a8495084805d3461e2d58bf4f014112c49fa59de3
    Image:          prateek2004/my-frontend
    Image ID:       docker.io/prateek2004/my-frontend@sha256:58bedad0762aca5ffa1d2e7f2f9d275b5677bca263ceb0deec
15888b7a
    Port:           80/TCP
    Host Port:      0/TCP
    State:          Running
      Started:      Mon, 30 Jun 2025 14:13:05 +0530
    Ready:          True
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-
```

HENCE In the description, check the **`Controlled By`** section:

- If it says **ReplicationController**, the pod was managed by it.
- If it says **ReplicaSet**, same logic.
- For Deployment, it will show Deployment → ReplicaSet → Pods.

SO HERE ITS BY REPLICATION CONTROLLER

2. **LETS Simulate a Pod Failure**

Manually delete a pod and observe:

```
kubectl delete pod <pod-name>
```

Now check:

kubectl get pods

You'll see that the controller **automatically recreates** the deleted pod.
This demonstrates the **self-healing** use case.

myapp-rc-pdnb9 1/1 Running 0 18s

REGENERATED 18SECS BEFORE SHOWING SELF HEAL PROPERTY

3. **Check Ownership and Hierarchy**

kubectl get rs
kubectl describe rs <replicaset-name>

You'll see:

- If a ReplicaSet is **created by a Deployment**, it says so in "Controlled By".
- If it's **standalone**, it has no owner — useful when comparing RS vs Deployment.

```
PS C:\WINDOWS\system32> kubectl describe rs myapp-rs
Name:         myapp-rs
Namespace:    default
Selector:     app=myapp
Labels:       <none>
Annotations:  <none>
Replicas:     2 current / 2 desired
Pods Status:  2 Running / 0 Waiting / 0 Succeeded / 0 Failed
Pod Template:
  Labels:  app=myapp
  Containers:
   myapp-container:
    Image:         prateek2004/my-frontend
    Port:          80/TCP
    Host Port:     0/TCP
    Environment:   <none>
    Mounts:        <none>
  Volumes:         <none>
  Node-Selectors:  <none>
  Tolerations:     <none>
Events:
  Type    Reason            Age   From                   Message
  ----    ------            ----  ----                   -------
  Normal  SuccessfulCreate  26m   replicaset-controller  Created pod: myapp-rs-s5hcv
  Normal  SuccessfulCreate  26m   replicaset-controller  Created pod: myapp-rs-9sg7z
PS C:\WINDOWS\system32>
```

4. **Try a Rolling Update (Deployment Only)**

kubectl set image deployment/myapp-deploy myapp-container=nginx

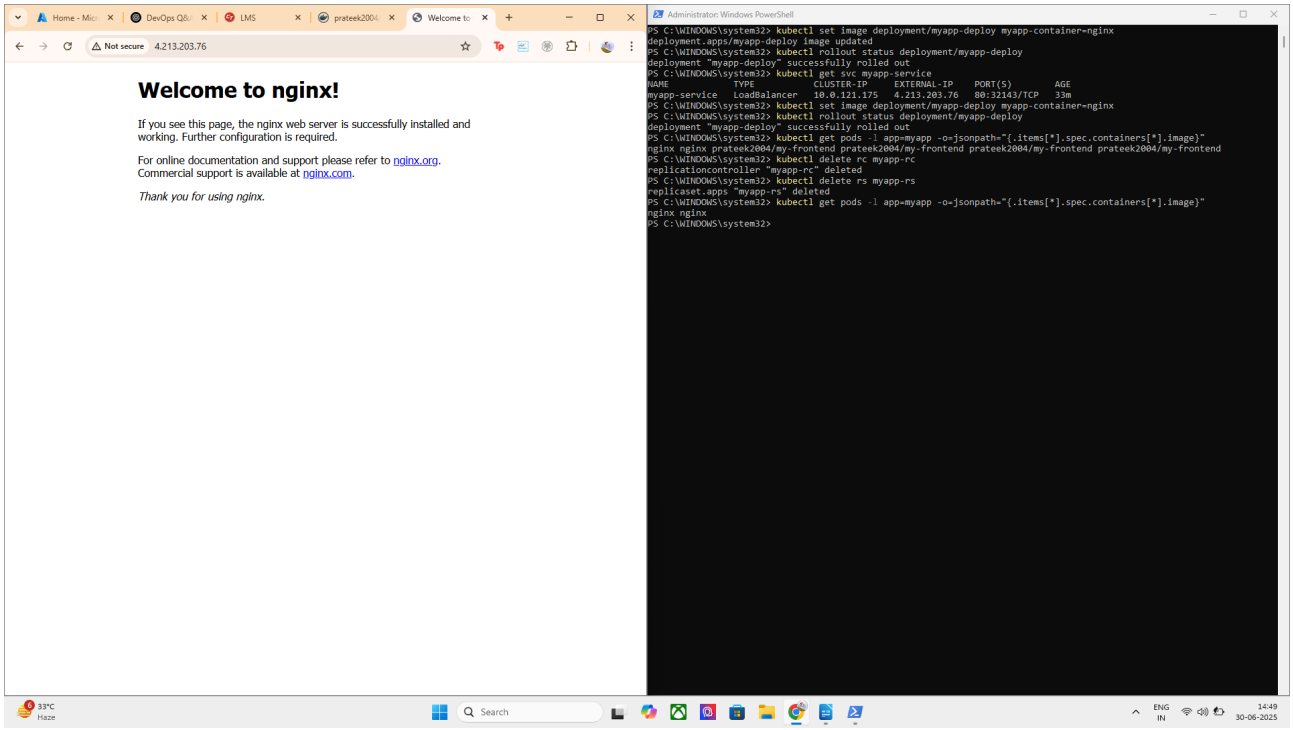
```
Administrator: Windows PowerShell
PS C:\WINDOWS\system32> kubectl set image deployment/myapp-deploy myapp-container=nginx
deployment.apps/myapp-deploy image updated
PS C:\WINDOWS\system32>
```

NOW CHECK:
kubectl rollout status deployment/myapp-deploy

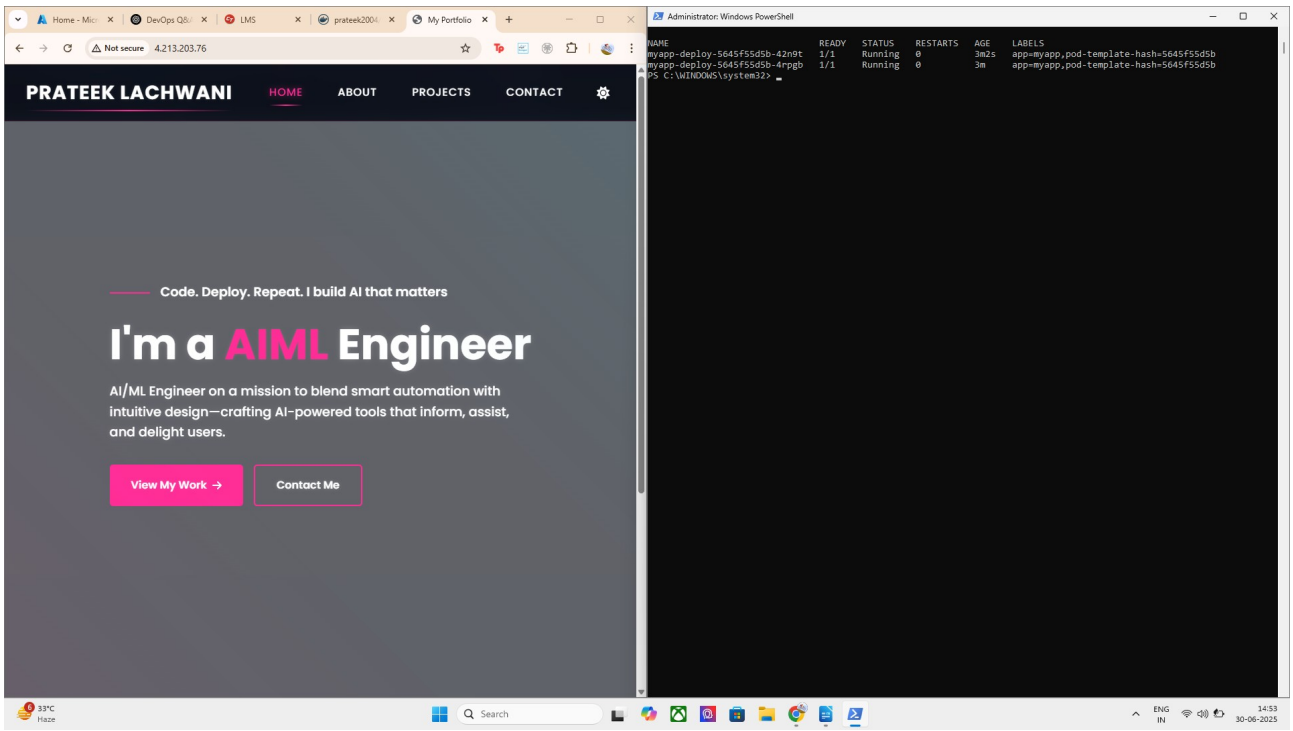
```
PS C:\WINDOWS\system32> kubectl rollout status deployment/myapp-deploy
deployment "myapp-deploy" successfully rolled out
PS C:\WINDOWS\system32>
```

```
Administrator: Windows PowerShell
PS C:\WINDOWS\system32> kubectl set image deployment/myapp-deploy myapp-container=nginx
deployment.apps/myapp-deploy image updated
PS C:\WINDOWS\system32> kubectl rollout status deployment/myapp-deploy
deployment "myapp-deploy" successfully rolled out
PS C:\WINDOWS\system32> kubectl get svc myapp-service
NAME            TYPE           CLUSTER-IP     EXTERNAL-IP    PORT(S)        AGE
myapp-service   LoadBalancer   10.0.121.175   4.213.203.76   80:32143/TCP   33m
PS C:\WINDOWS\system32>
```

Visit the `EXTERNAL-IP` — you'll now see the **NGINX welcome page**



ROLLED AGAIN TO ORIGINAL:



In this task, I deployed a **ReplicationController**, a **ReplicaSet**, and a **Deployment** in Azure Kubernetes Service (AKS) to manage my custom web application hosted on Docker Hub (`prateek2004/my-frontend`).
Each of these ensures high availability by maintaining multiple replicas (2 pods) of the application.
I used YAML files to define and apply all three controllers, and exposed them via a **LoadBalancer Service** for external access.
I also performed a **rolling update** on the Deployment to test seamless version changes (from my image to `nginx` and back).
To verify, I checked pod health, ReplicaSets, rollout history, and container images using various `kubectl` commands.
In addition, I learned the ownership difference between RC, RS, and Deployment, and how Deployment provides better lifecycle and update management

ExtraS :

- Switched from self-hosted kubeadm to Azure AKS due to reliability.
- Created Docker image of a personal project and hosted on Docker Hub.
- Understood and compared the behaviors of ReplicaSet vs ReplicationController vs Deployment.
- Used `kubectl get`, `set image`, `rollout status`, and `describe` for deeper insights.
- Created a LoadBalancer-type service to make the app publicly accessible.