## **Lab : Rolling updates:**

### **A. Create deployment for nginx: (app-deployment.yaml)**

# my-app.yaml

apiVersion: apps/v1

kind: Deployment

metadata:

name: my-app

spec:

replicas: 1

revisionHistoryLimit: 5 # (optional) How many past versions to keep

selector:

matchLabels:

app: my-app

template:

metadata:

labels:

app: my-app

spec:

containers:

- name: my-app

image: nginx:1.21 # Version 1 of app

### **B. Apply deployment:**

Kubectl apply -f app-deployment.yaml

### **C. Modify the deployment yaml to specify wrong version for image:**

# my-app.yaml

apiVersion: apps/v1

kind: Deployment

metadata:

name: my-app

spec:

replicas: 1

revisionHistoryLimit: 5 # (optional) How many past versions to keep

selector:

matchLabels:

app: my-app

template:

metadata:

labels:

app: my-app

spec:

containers:

- name: my-app

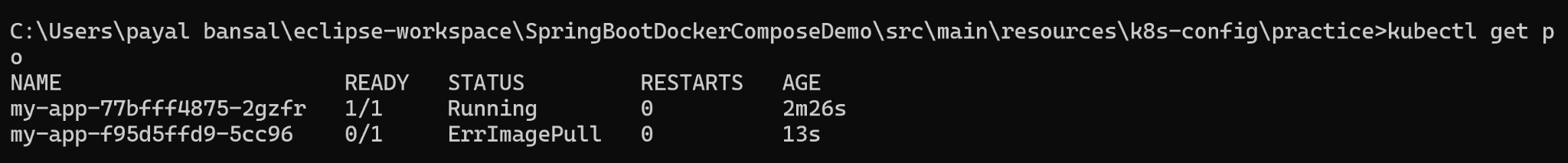
image: nginx:bad-version # Version 2 of app

### **D. Apply deployment:**

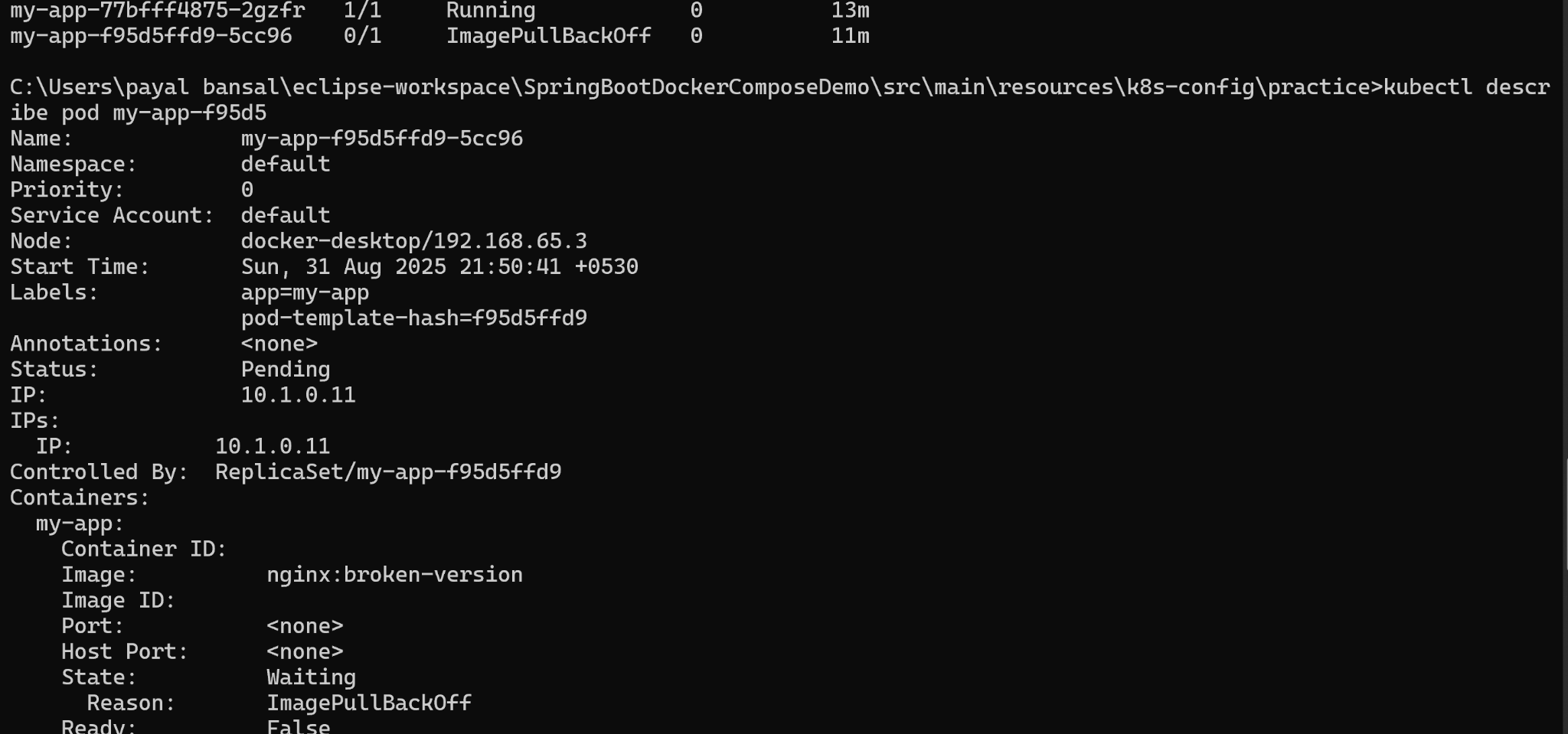
Kubectl apply -f app-deployment.yaml

### **E. Check pod status:**

kubectl get po



Check the failing pod



### **F. Rollback deployment:**

kubectl rollout undo deployment <name of deployment>



Kubectl get po:



That will:

* Delete the broken ReplicaSet
* Restore the **previous working Pod version**

### **G. Summary**

* Kubernetes keeps old Pods running during update until the new one works ✅
* If the new one fails (ErrImagePull), rollback to the old one manually
* Use kubectl rollout undo deployment <name> to fix fast