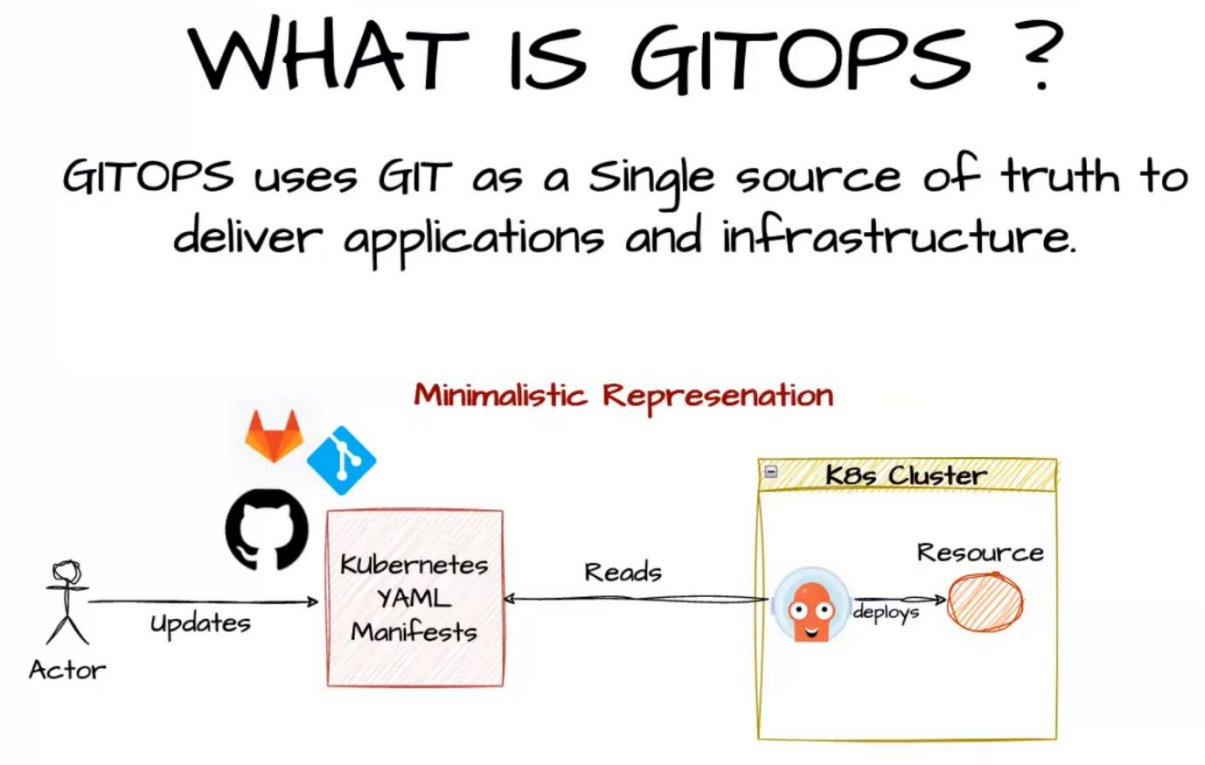
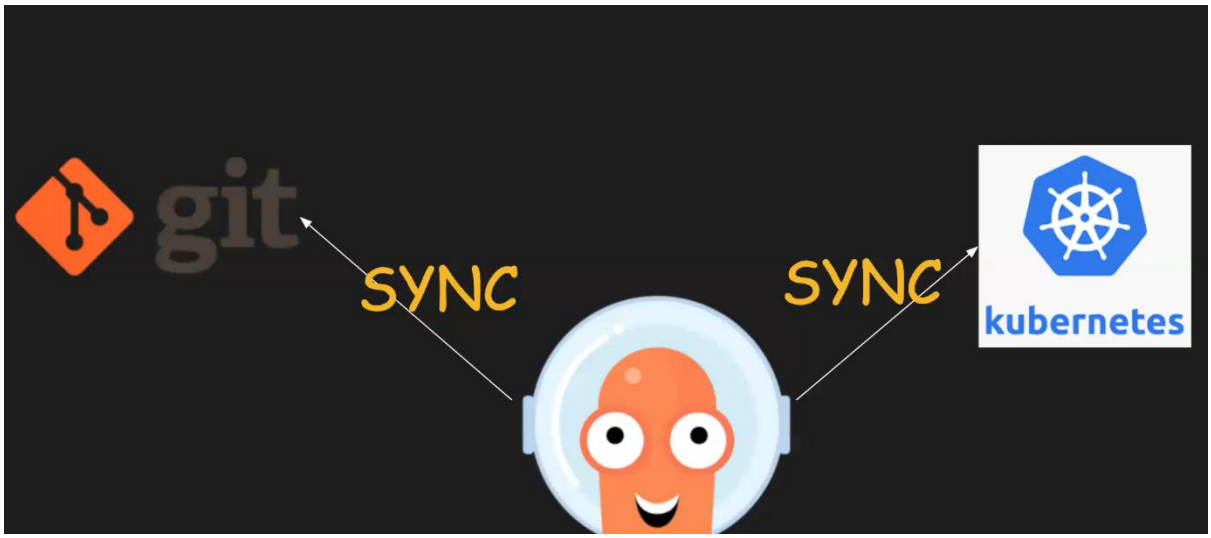
ARGOCD





Argo CD is a Kubernetes-native continuous deployment (CD) tool. Unlike external CD tools that only enable push-based deployments, Argo CD can pull updated code from Git repositories and deploy it directly to Kubernetes resources. It enables developers to manage both infrastructure configuration and application updates in one system.

Argo CD offers the following key features and capabilities:

- **Manual or automatic deployment of applications to a Kubernetes cluster.**
- **Automatic synchronization of application state to the current version of declarative configuration.**
- **Web user interface and command-line interface (CLI).**

- **Ability to visualize deployment issues, detect and remediate configuration drift.**
- **Role-based access control (RBAC) enabling multi-cluster management.**
- **Single sign-on (SSO) with providers such as GitLab, GitHub, Microsoft, OAuth2, OIDC, LinkedIn, LDAP, and SAML 2.0**
- **Support for webhooks triggering actions in GitLab, GitHub, and BitBucket.**

GitOps with Argo CD

GitOps is a software engineering practice that uses a Git repository as its single source of truth. Teams commit declarative configurations into Git, and these configurations are used to create environments needed for the continuous delivery process. There is no manual setup of environments and no use of standalone scripts—everything is defined through the Git repository.

A basic part of the GitOps process is a pull request. New versions of a configuration are introduced via pull request, merged with the main branch in the Git repository, and then the new version is automatically deployed. The Git repository contains a full record of all changes, including all details of the environment at every stage of the process.

Argo CD handles the latter stages of the GitOps process, ensuring that new configurations are correctly deployed to a Kubernetes cluster.

At a high level, the Argo CD process works like this:

1. **A developer makes changes to an application, pushing a new version of Kubernetes resource definitions to a Git repo.**
2. **Continuous integration is triggered, resulting in a new container image saved to a registry.**
3. **A developer issues a pull request, changing Kubernetes manifests, which are created either manually or automatically.**
4. **The pull request is reviewed and changes are merged to the main branch. This triggers a webhook which tells Argo CD a change was made.**
5. **Argo CD clones the repo and compares the application state with the current state of the Kubernetes cluster. It applies the required changes to cluster configuration.**
6. **Kubernetes uses its controllers to reconcile the changes required to cluster resources, until it achieves the desired configuration.**
7. **Argo CD monitors progress and when the Kubernetes cluster is ready, reports that the application is in sync.**
8. **ArgoCD also works in the other direction, monitoring changes in the Kubernetes cluster and discarding them if they don't match the current configuration in Git.**

How does Argo CD make it happen?

- **GitOps agent—Argo CD is responsible for pulling updated code from Git repositories and deploying it directly to Kubernetes resources. It manages both infrastructure configuration and application updates in one system.**
- **Custom Resource Definitions (CRD)—Argo CD operates in its own namespace within a Kubernetes cluster. It provides its own CRDs that extend the Kubernetes API and make it possible to define the desired application state in a declarative way. Based on the instructions in a Git repo or a Helm repo, Argo CD uses its CRDs to implement the changes within its dedicated namespace.**

- **CLI—Argo CD offers a powerful CLI that lets you create YAML resource definitions with a few simple commands. For example, the**

Argo CD app create

command lets you specify a few flags and create a valid

Application

object that describes your application, with no need to write YAML by hand.

STEP :1

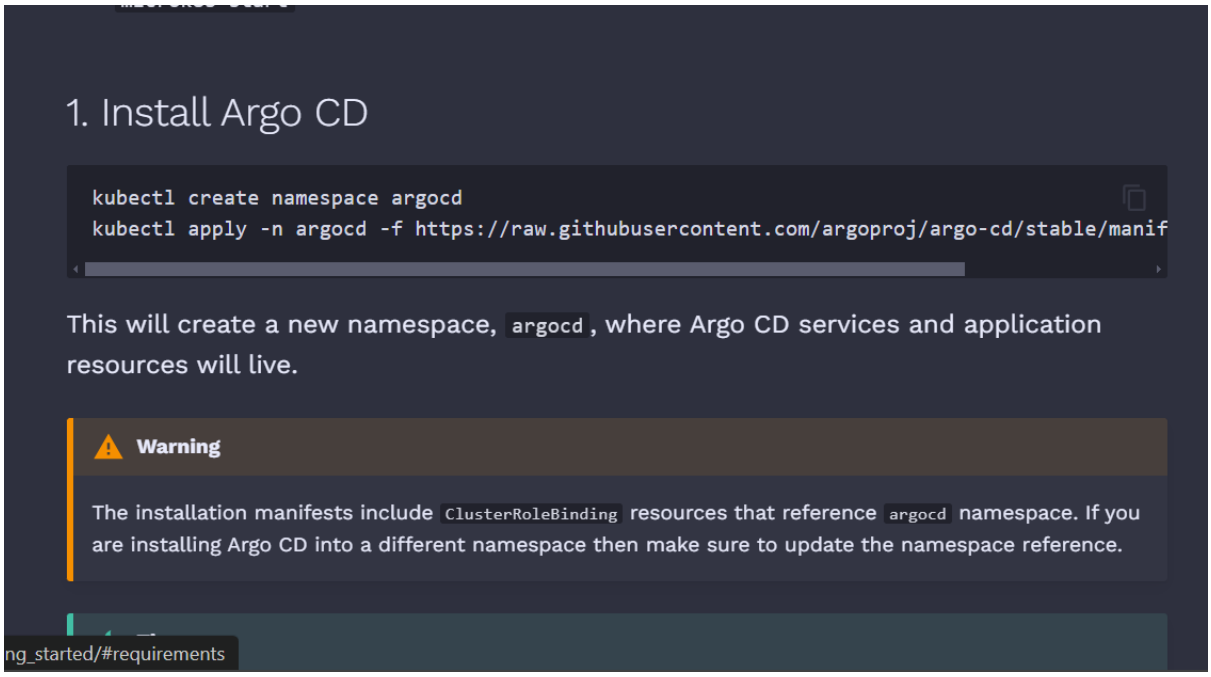
Installation of Argocd:

Go to Offical Documentation of Argocd:

https://argo-cd.readthedocs.io/en/stable/getting_started/

kubectl create namespace argocd

kubectl apply -n argocd -f https://raw.githubusercontent.com/argoproj/argo-cd/stable/manifests/install.yaml

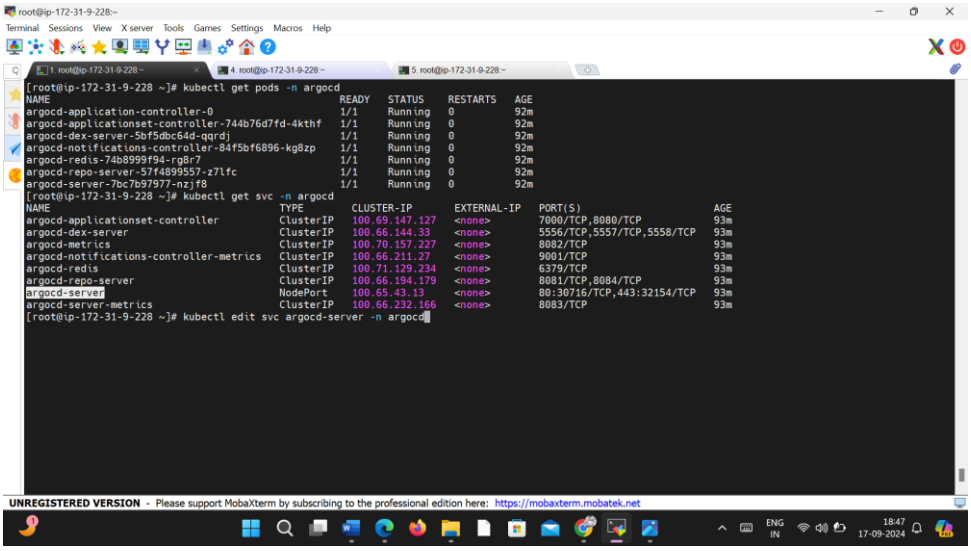


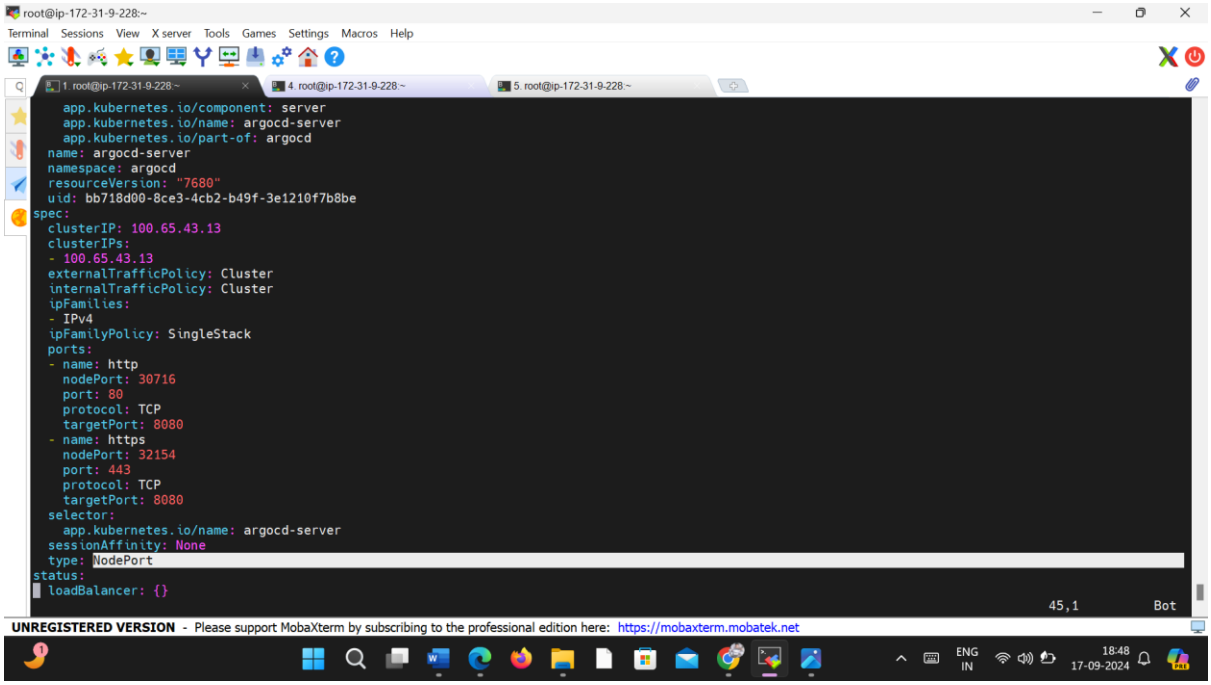
STEP 2:

After installing Argocd . We get below pods

```
[root@ip-172-31-9-228 ~]# kubectl get pods -n argocd
NAME                                                READY   STATUS    RESTARTS   AGE
argocd-application-controller-0                     1/1     Running   0          92m
argocd-applicationset-controller-744b76d7fd-4kthf   1/1     Running   0          92m
argocd-dex-server-5bf5dbc64d-qqrdj                  1/1     Running   0          92m
argocd-notifications-controller-84f5bf6896-kg8zp    1/1     Running   0          92m
argocd-redis-74b8999f94-rg8r7                       1/1     Running   0          92m
argocd-repo-server-57f4899557-z7lfc                 1/1     Running   0          92m
argocd-server-7bc7b97977-nzjf8                      1/1     Running   0          92m
[root@ip-172-31-9-228 ~]#
```

EXPOSE ARGOCD-SERVER SERVICE TO NODE PORT

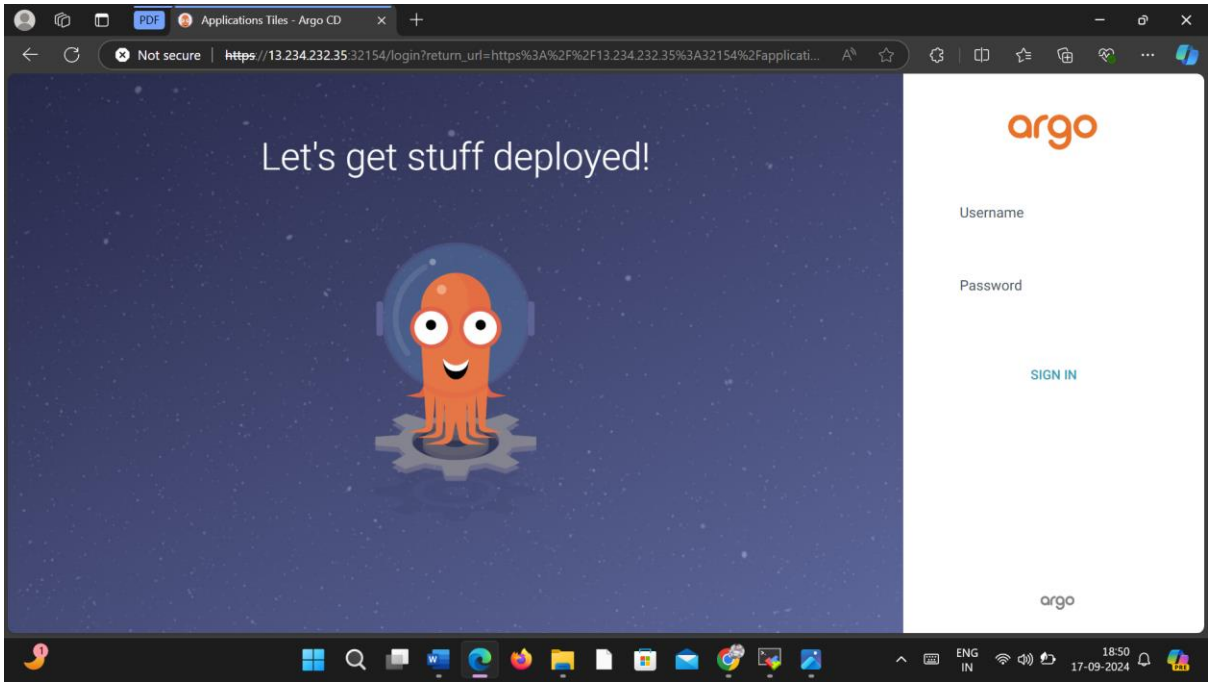




STEP 3:

To Login into Argocd UI

Copy PublicIP of EC2 instance along with NodePort and Paste in browser.

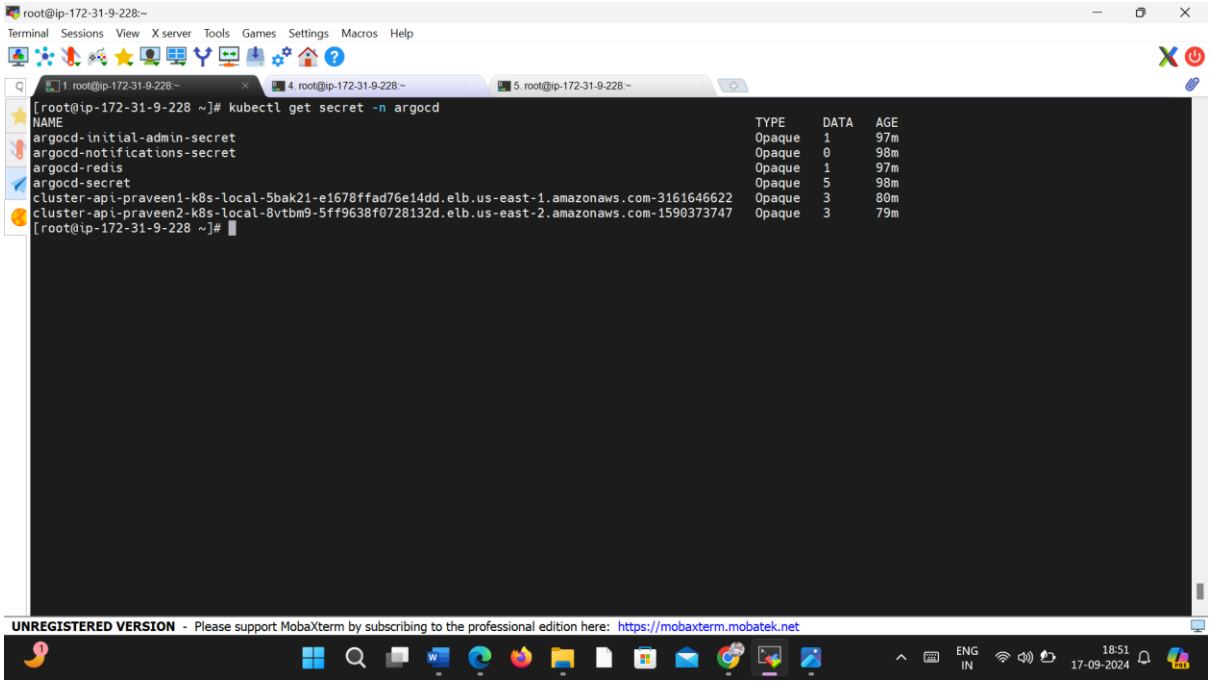


This is argocd DashBoard

UserName: admin

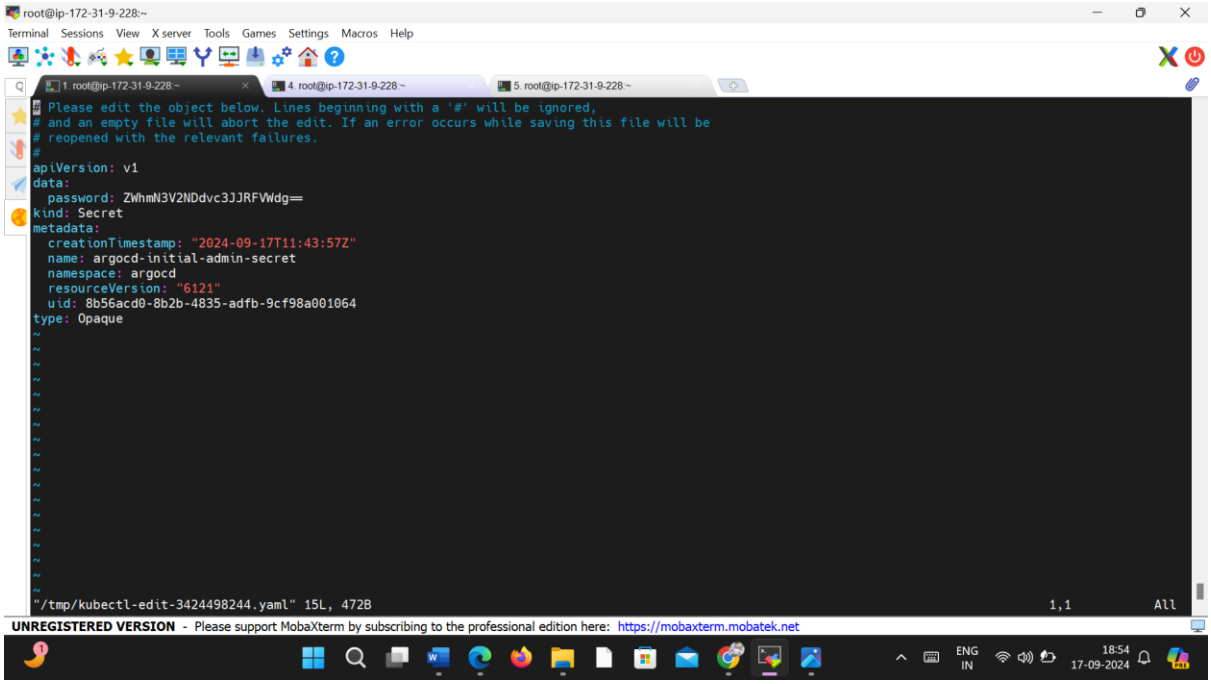
To get Password:

Go to argocd seceret

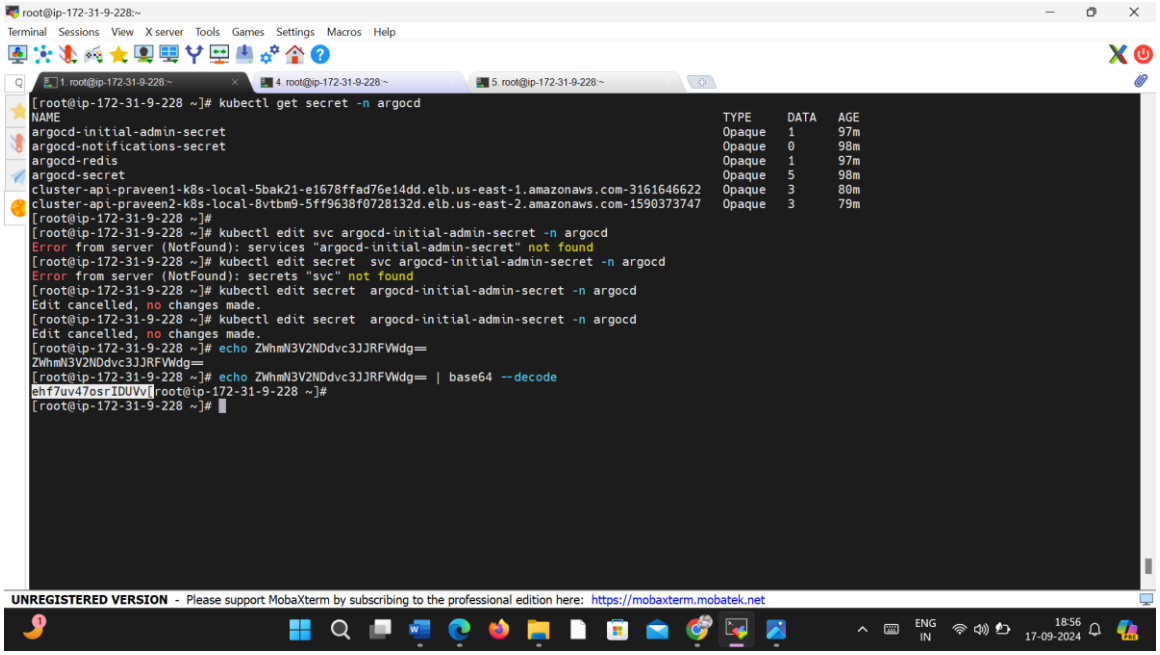


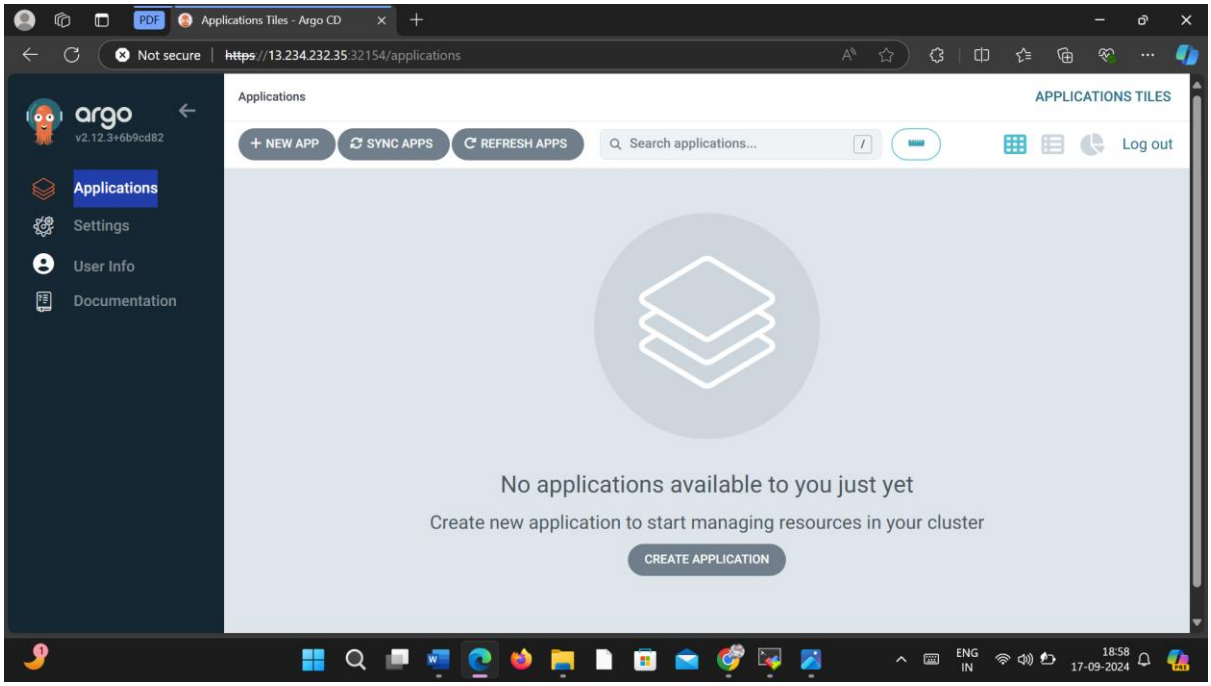
Edit argocd-initial-admin-secret

Kubectl edit secret argocd-initial-admin-secret -n argocd



Copy the password and decode using Base64

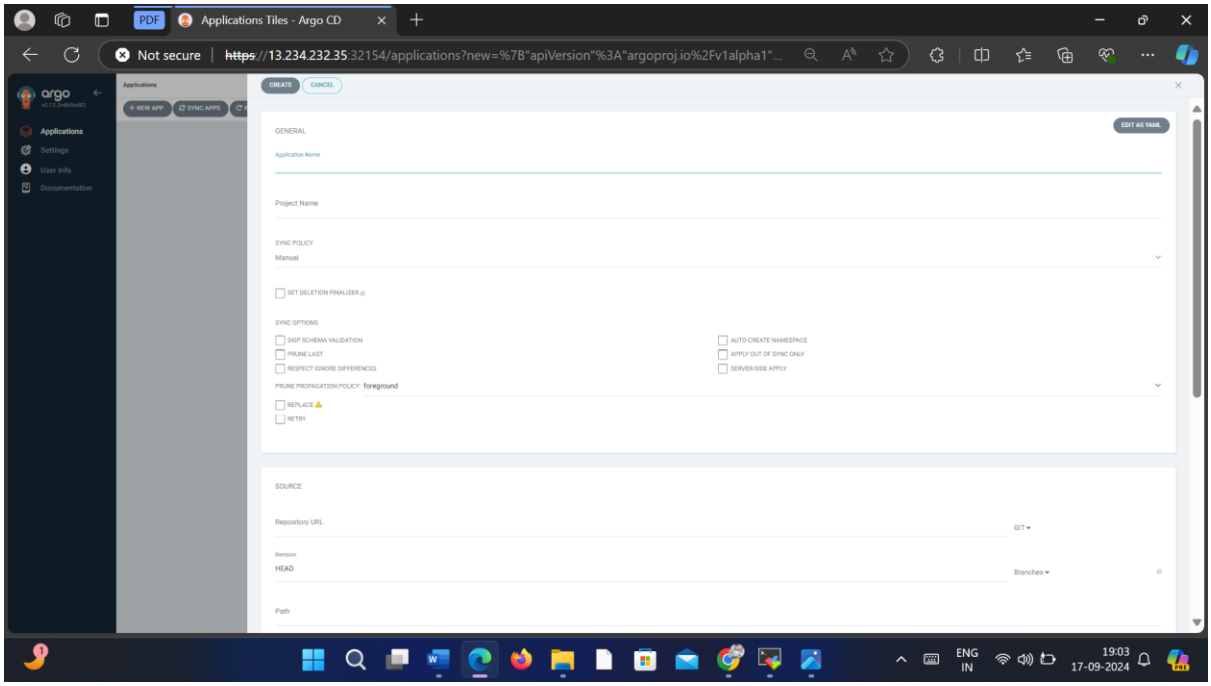




STEP 4:

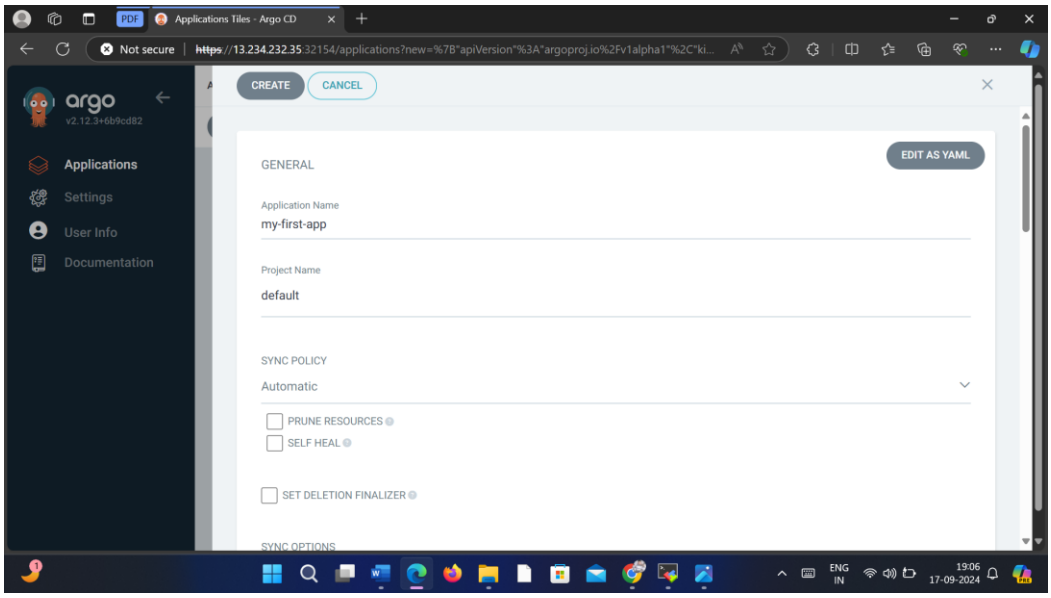
Create a Application:

- **Click On New app**



- **After Click on application . We get above UI.**
- **Configure the details.**

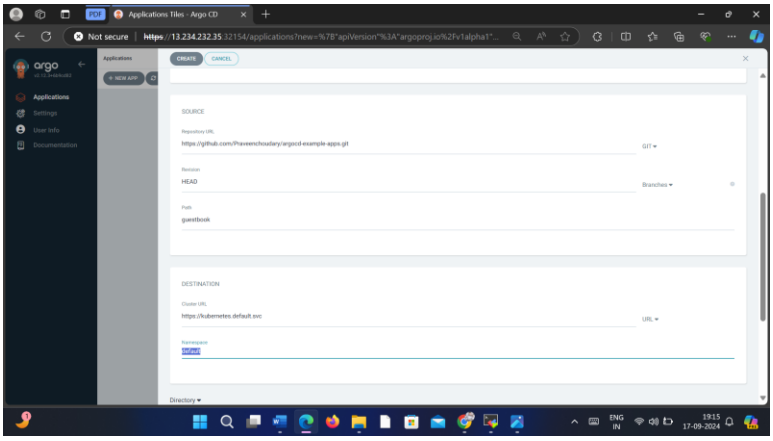
1. **Application Name: firstapp**
2. **Project Name: Default (In my case)**
3. **SYNC POLICY : Automatic**



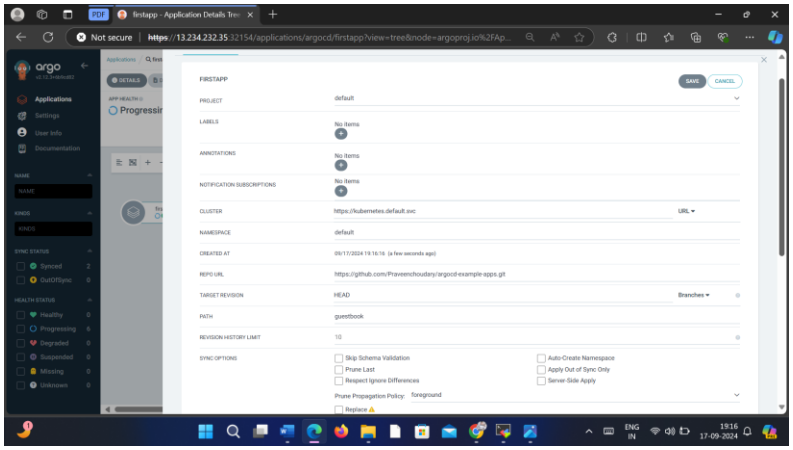
4. **Source: Paste Github repo url**

   https://github.com/Praveenchoudary/argocd-example-apps.git

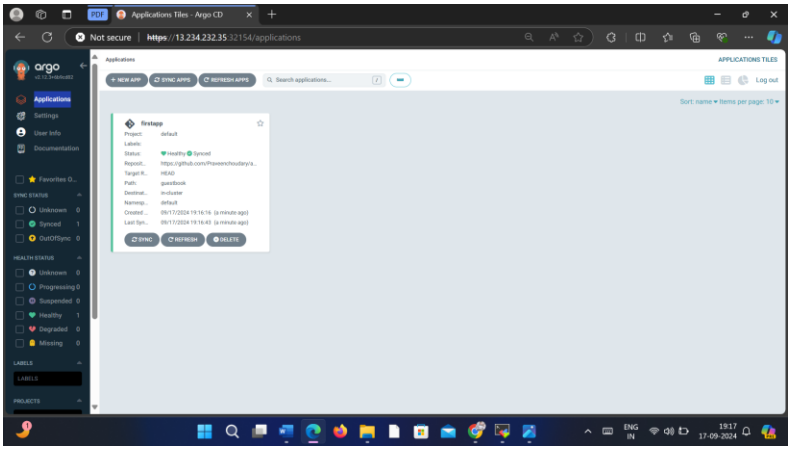
5. **Path: guestbook**
6. **Cluster url: https://kubernetes.default.svc (default url)**
7. **Name Space: default**



8. **Click on create**



Aplication is cretaed Successfully



deployment.yml

apiVersion: apps/v1

kind: Deployment

metadata:

** name: guestbook-ui**

spec:

** replicas: 3**

** revisionHistoryLimit: 3**

** selector:**

**   matchLabels:**

**   app: guestbook-ui**

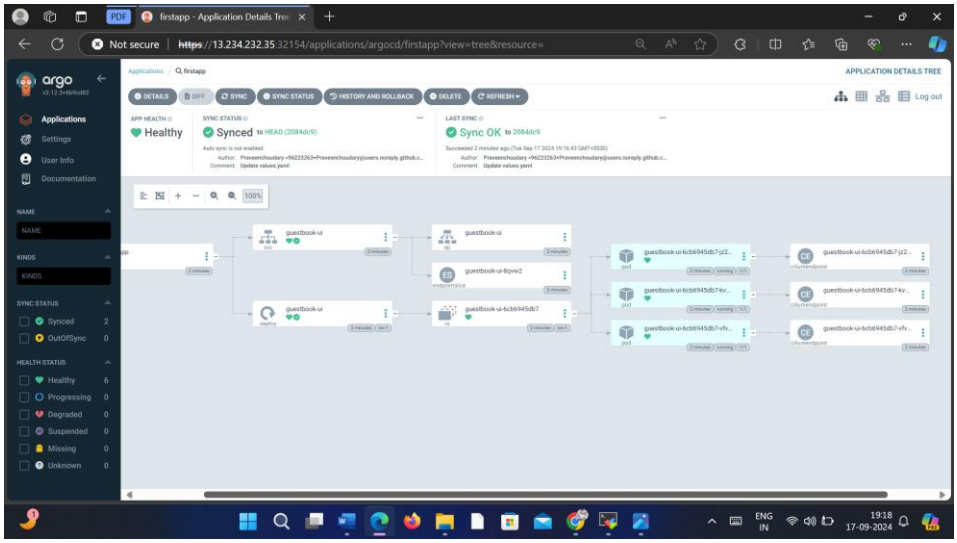
** template:**

```
  metadata:
    labels:
      app: guestbook-ui
  spec:
    containers:
    - image: praveen22233/netflix
      name: guestbook-ui
      ports:
      - containerPort: 80
```

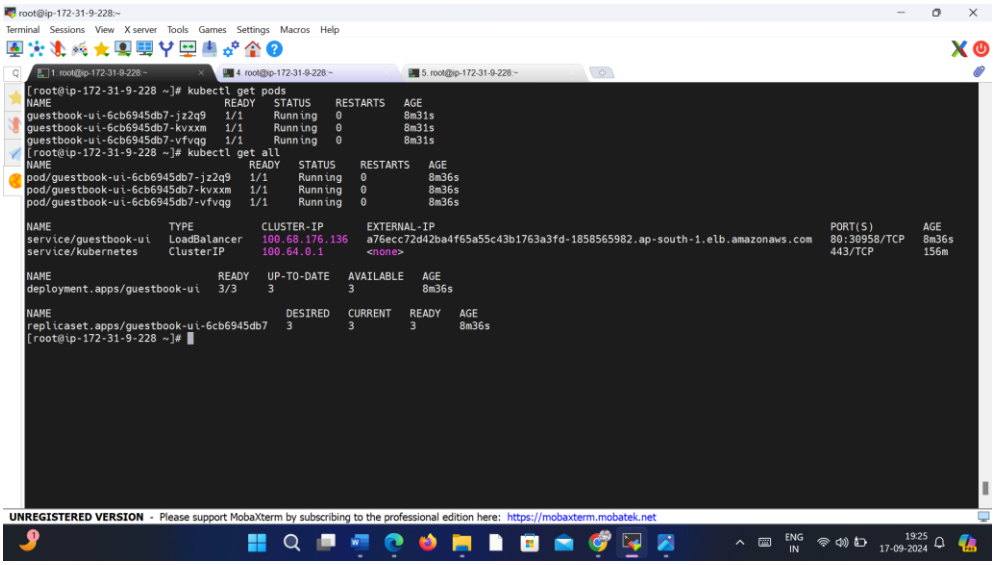
Three Pods are created as I mentioned replica count=3 in deployment.yml file and service LoadBalancer.

Service.yml

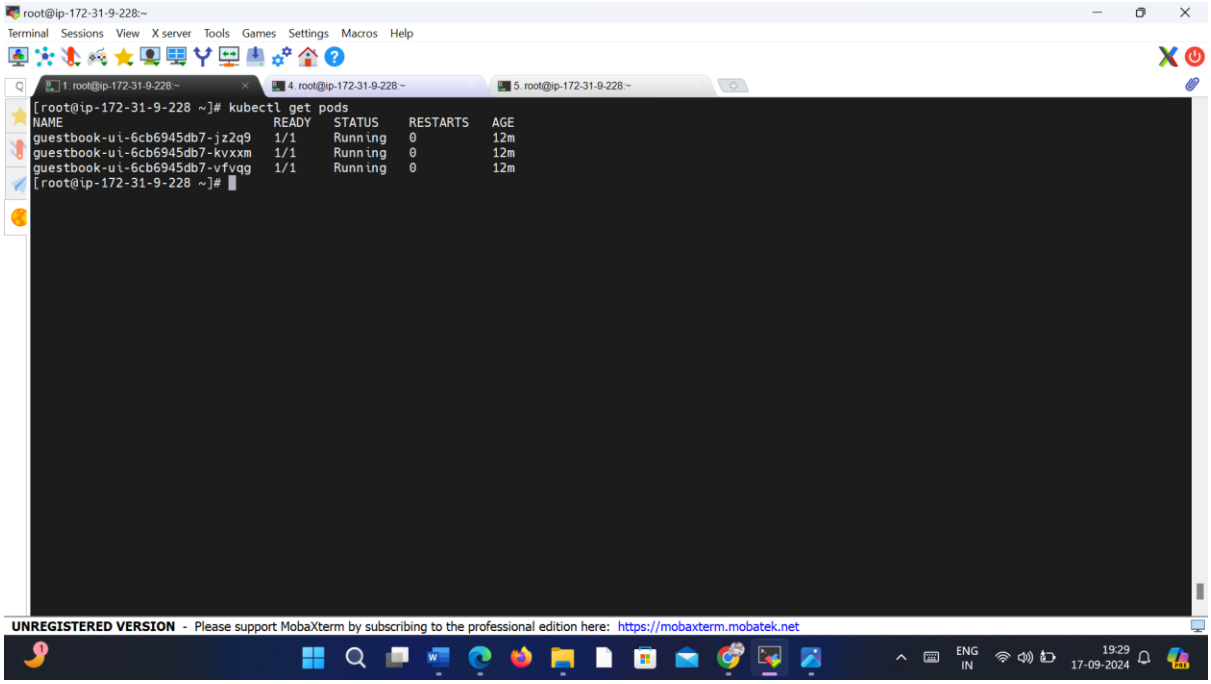
```
apiVersion: v1
kind: Service
metadata:
  name: guestbook-ui
spec:
  type: LoadBalancer
  selector:
    app: guestbook-ui
  ports:
  - port: 80
    targetPort: 80
```



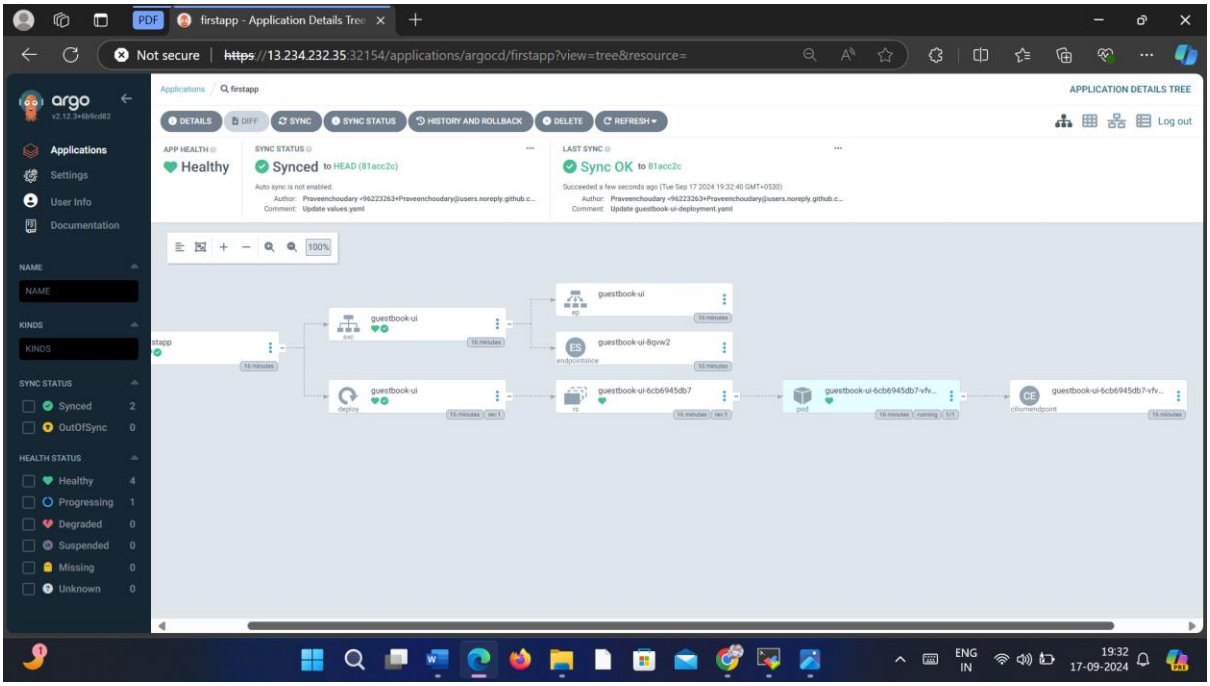
3 pods and service is created in k8s cluster. This is done by using argocd



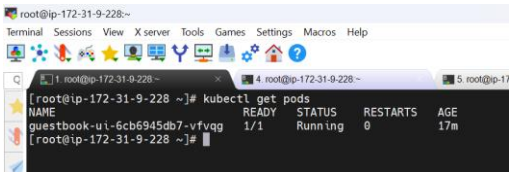
If we make any changes in yml files which are in github. Automatically the changes will reflected in k8s cluster.



- **In K8s cluster there is 3 pods are running.**
- **Now I make changes in replicas count to 1 in deployment.yml file in github and commit the changes .**
- **Automatically The pod size will be decreased to 1.**

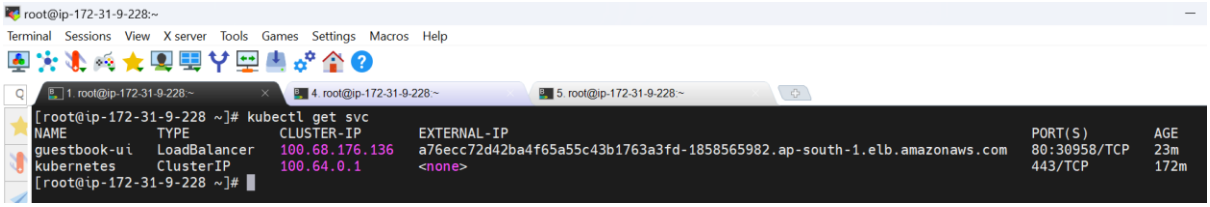


The replica count is decreased to one.

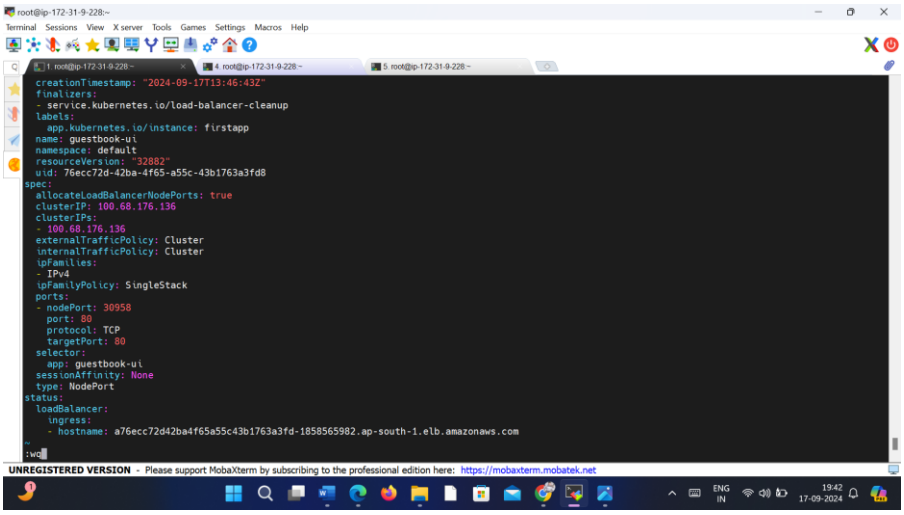


Note:

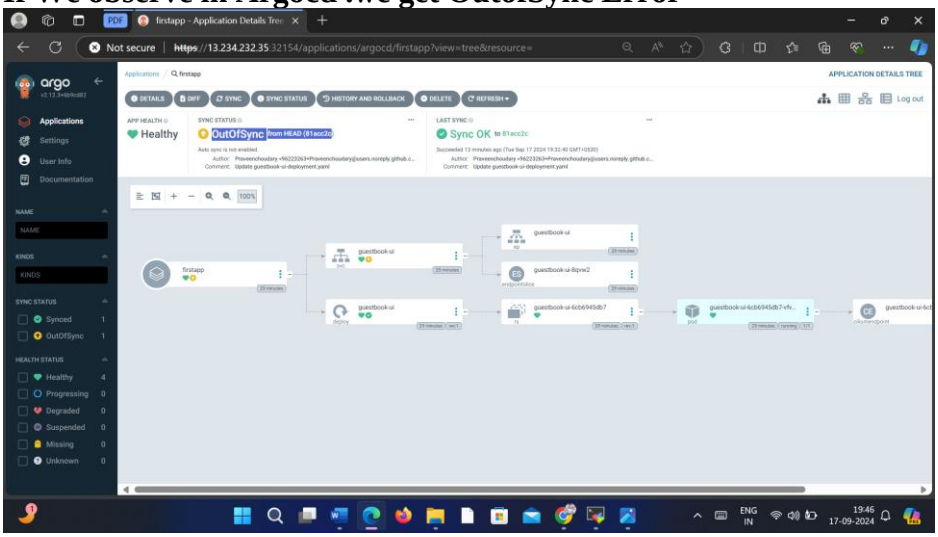
1. **Once if we implement ArgoCD, if we make any changes manually in our cluster using the kubectl command, Kubernetes will reject those request from that user. Because when we apply changes manually, ArgoCD will check the actual state of the cluster with the desired state of the cluster (GitHub)**



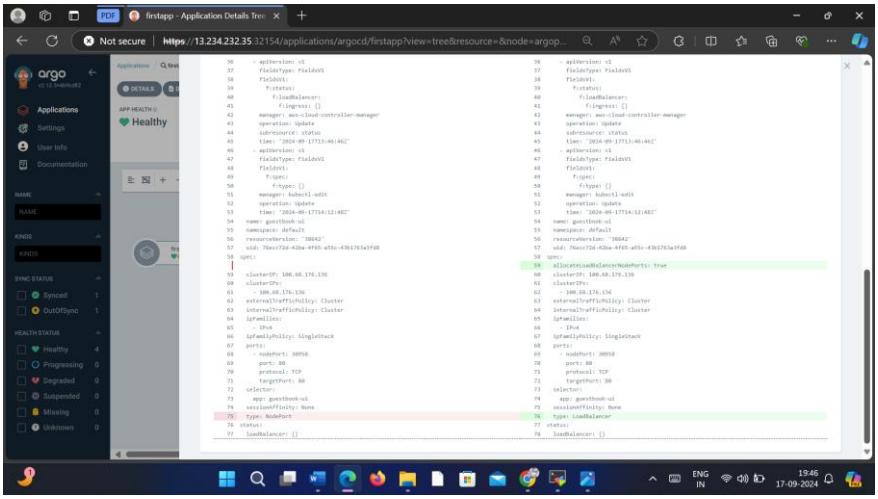
Here The svc type is LoadBalancer.so now I am going to change to NodPort directly in K8s cluster.



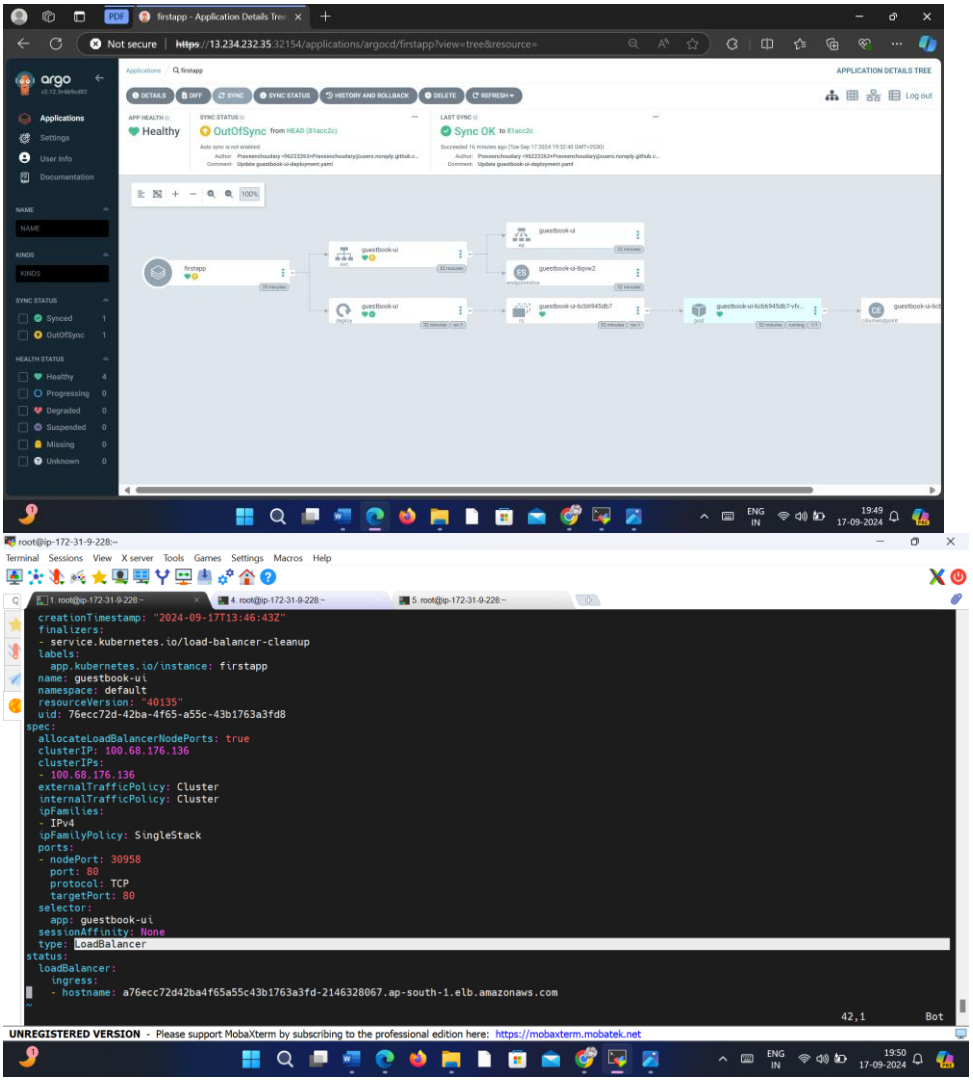
If We observe in Argocd .we get OutofSync Error



Argocd Clearly showing Changes in service.yml file



Suppose if You click sync in argocd. The changes will revert back in service .yml file.



- **Previously I Changed type to NodePort from LoadBalancer.**
- **The Argocd Reject that change We get error OutOfSync.and I clicked manually on sync it the changes is revertback**

STEP 5: Access application In my case the type is LoadBalancer.so I can access by using LB.