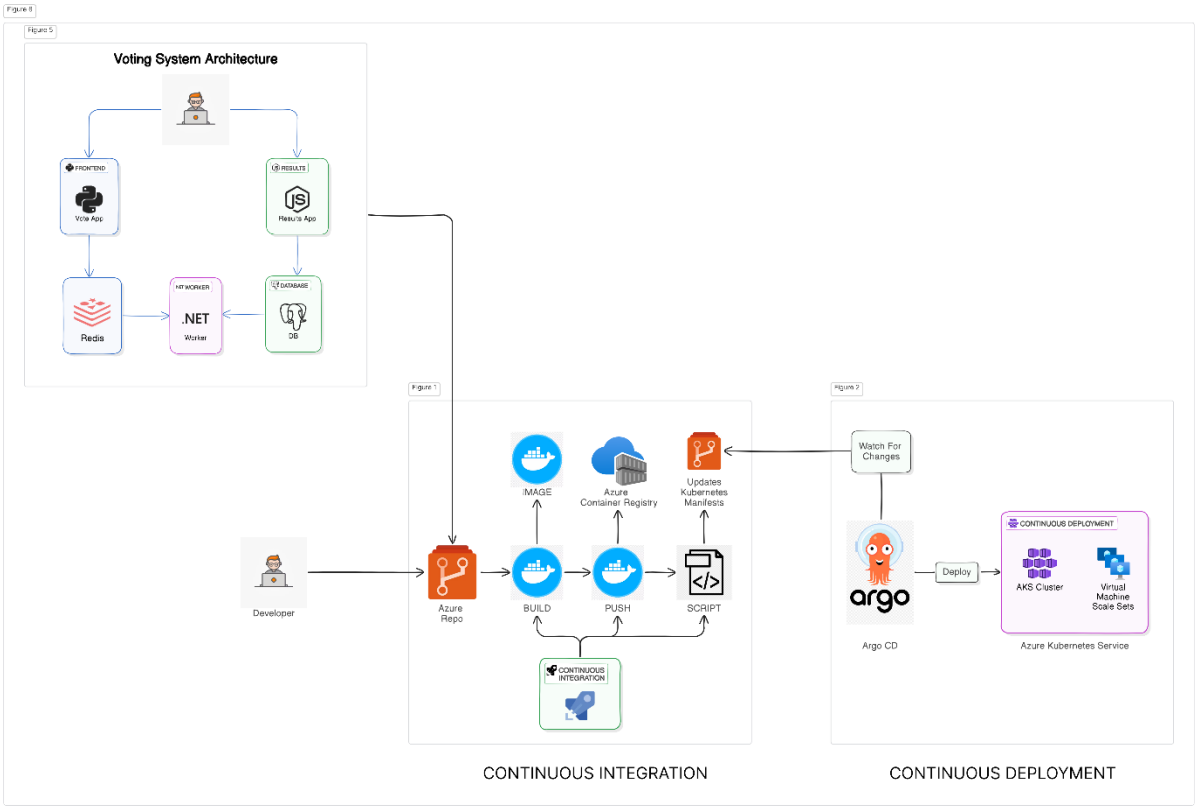
Complete End-to-End CI/CD Process with Azure DevOps and AKS Cluster Deployment



Project Objectives

Application with Microservices:

- **Front-End Web App**: Developed using Python, this application allows users to vote between two options.
- **Redis**: A data store that collects and manages incoming votes.
- **.NET Worker**: A service that processes and stores votes.
- **Postgres Database**: A relational database supported by a Docker volume, ensuring persistent data storage.
- **Node.js Web App**: Provides real-time updates of voting results through a dynamic web interface.

Implementing Continuous Integration Pipeline (CI):

- **Pipeline Creation**: Three separate pipelines were created for the worker, vote and result microservices. Each pipeline automates the build process and is responsible for:
    - Building Docker images from Dockerfiles located in the Azure Repository.
    - Pushing the built Docker images to Azure Container Registry.

Implementing Continuous Deployment (CD):

- **AKS Cluster Setup**:
    - An Azure Kubernetes Service (AKS) cluster was established, utilizing Virtual Machine Scale Sets as node pools to handle varying loads.
- **Deployment & Service Configuration**:
    - Created YAML files for deployments and services were configured to define the infrastructure and application setup within the AKS cluster.
- **ArgoCD Integration**:
    - ArgoCD was installed and configured to automate the deployment process. It synchronizes with the defined configurations to manage application deployments.
- **Shell Script for Image Update**:
    - A shell script was created to automatically update the Docker image name with the Build Number whenever a Vote pipeline is triggered. This script ensures that the new Docker image is deployed and replaces the previous version in the AKS cluster.

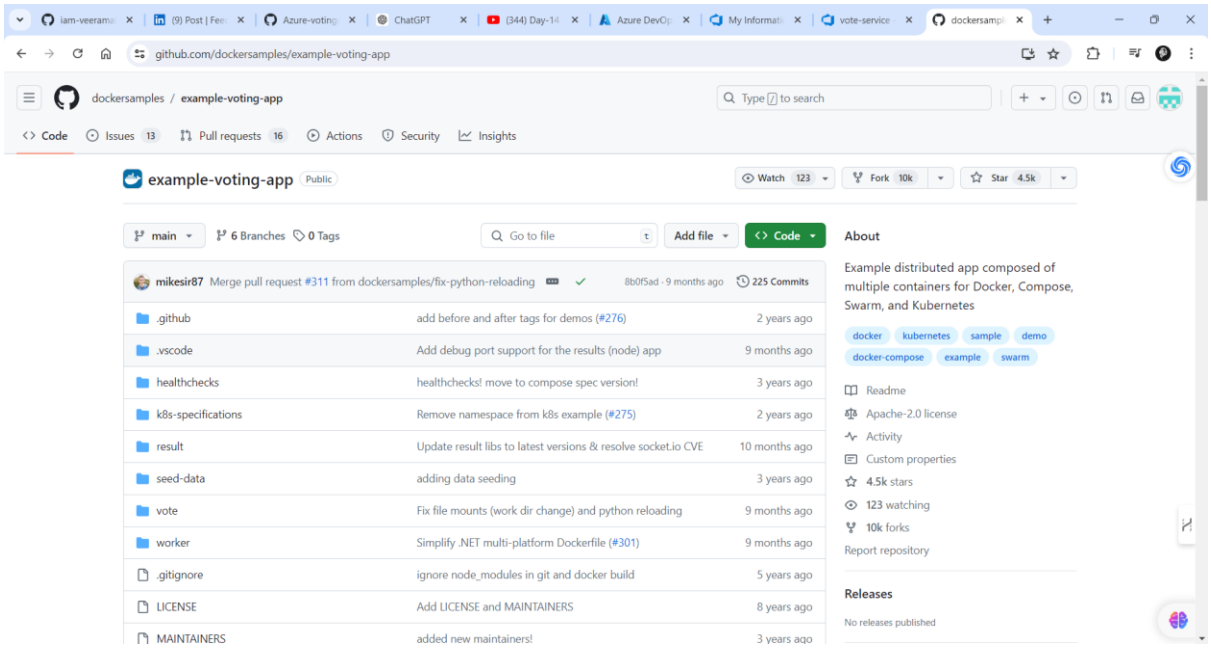
Implementation:

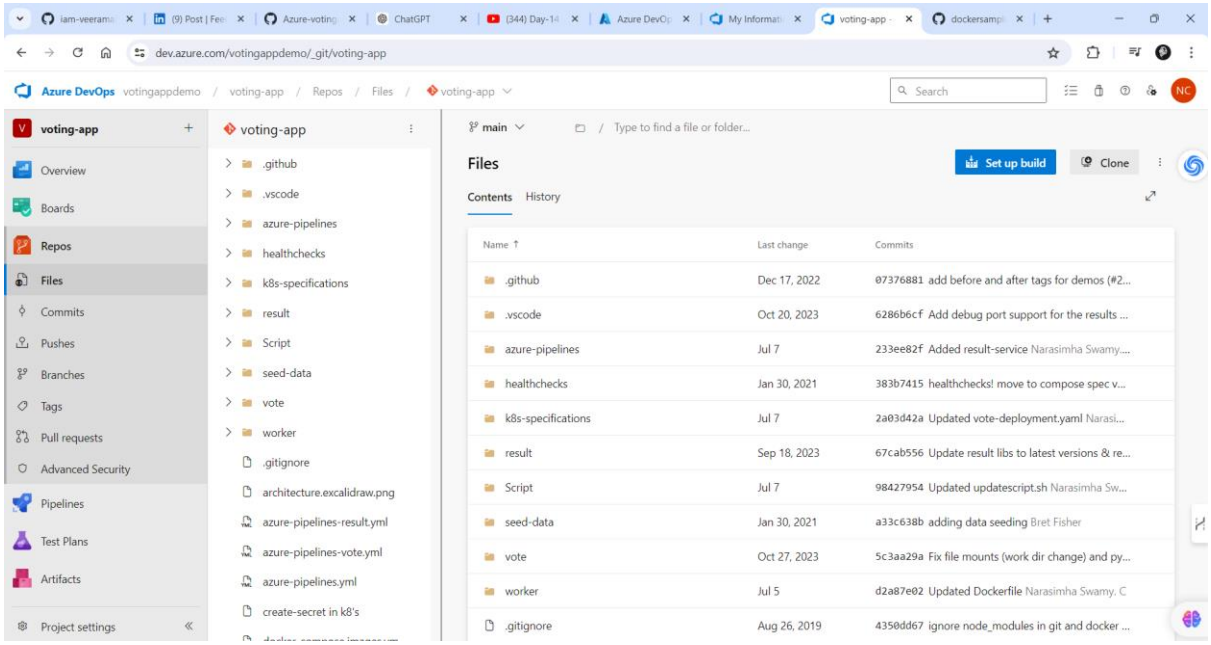
Step-1: Application with Microservices

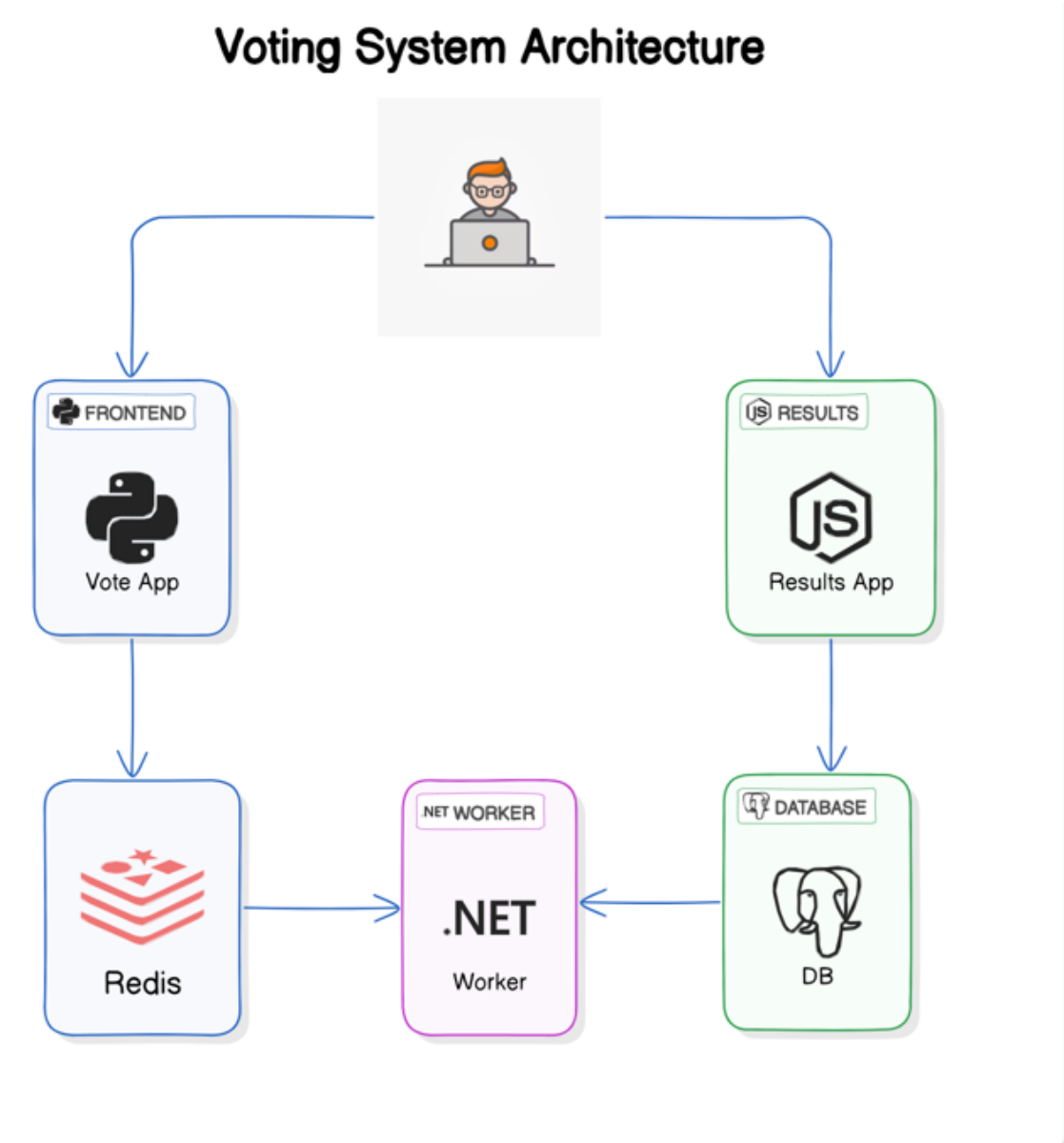
The application, consisting of multiple microservices, was sourced from the Docker team's multi-microservice application project. The code was forked from the Docker team's GitHub repository and integrated into my project. The forked code was then pushed into the Azure Repo for further development and deployment.

The application includes:

- A Front-End Web App developed using Python, allowing users to vote between two options.
- Redis, a data store that collects and manages incoming votes.
- A .NET Worker service that processes and stores votes.
- A Postgres Database, a relational database supported by a Docker volume to ensure persistent data storage.
- A Node.js Web App providing real-time updates of voting results through a dynamic web interface.


github.com/dockersamples/example-voting-app
dockersamples / example-voting-app
Type / to search
Code
Issues 13
Pull requests 16
Actions
Security
Insights
example-voting-app
Public
Watch 123
Fork 10k
Star 4.5k
main
6 Branches
0 Tags
Go to file
Add file
Code
About
mikesir87 Merge pull request #311 from dockersamples/fix-python-reloading
8b0f5ad · 9 months ago
225 Commits
.github add before and after tags for demos (#276) 2 years ago
.vscode Add debug port support for the results (node) app 9 months ago
healthchecks healthchecks! move to compose spec version! 3 years ago
k8s-specifications Remove namespace from k8s example (#275) 2 years ago
result Update result libs to latest versions & resolve socket.io CVE 10 months ago
seed-data adding data seeding 3 years ago
vote Fix file mounts (work dir change) and python reloading 9 months ago
worker Simplify .NET multi-platform Dockerfile (#301) 9 months ago
.gitignore ignore node_modules in git and docker build 5 years ago
LICENSE Add LICENSE and MAINTAINERS 8 years ago
MAINTAINERS added new maintainers! 3 years ago
Example distributed app composed of multiple containers for Docker, Compose, Swarm, and Kubernetes
docker
kubernetes
sample
demo
docker-compose
example
swarm
Readme
Apache-2.0 license
Activity
Custom properties
4.5k stars
123 watching
10k forks
Report repository
Releases
No releases published


dev.azure.com/votingappdemo/_git/voting-app
Azure DevOps votingappdemo / voting-app / Repos / Files / voting-app
Search
voting-app
Overview
Boards
Repos
Files
Commits
Pushes
Branches
Tags
Pull requests
Advanced Security
Pipelines
Test Plans
Artifacts
Project settings
voting-app
.github
.vscode
azure-pipelines
healthchecks
k8s-specifications
result
Script
seed-data
vote
worker
.gitignore
architecture.excalidraw.png
azure-pipelines-result.yml
azure-pipelines-vote.yml
azure-pipelines.yml
create-secret in k8's
main
Type to find a file or folder...
Files
Set up build
Clone
Contents
History
Name
Last change
Commits
.github Dec 17, 2022 07376881 add before and after tags for demos (#2...
.vscode Oct 20, 2023 6286b6cf Add debug port support for the results ...
azure-pipelines Jul 7 233ee82f Added result-service Narasimha Swamy....
healthchecks Jan 30, 2021 383b7415 healthchecks! move to compose spec v...
k8s-specifications Jul 7 2a03d42a Updated vote-deployment.yaml Narasi...
result Sep 18, 2023 67cab556 Update result libs to latest versions & re...
Script Jul 7 98427954 Updated updatescript.sh Narasimha Sw...
seed-data Jan 30, 2021 a33c638b adding data seeding Bret Fisher
vote Oct 27, 2023 5c3aa29a Fix file mounts (work dir change) and py...
worker Jul 5 d2a87e02 Updated Dockerfile Narasimha Swamy. C
.gitignore Aug 26, 2019 4350dd67 ignore node_modules in git and docker ...



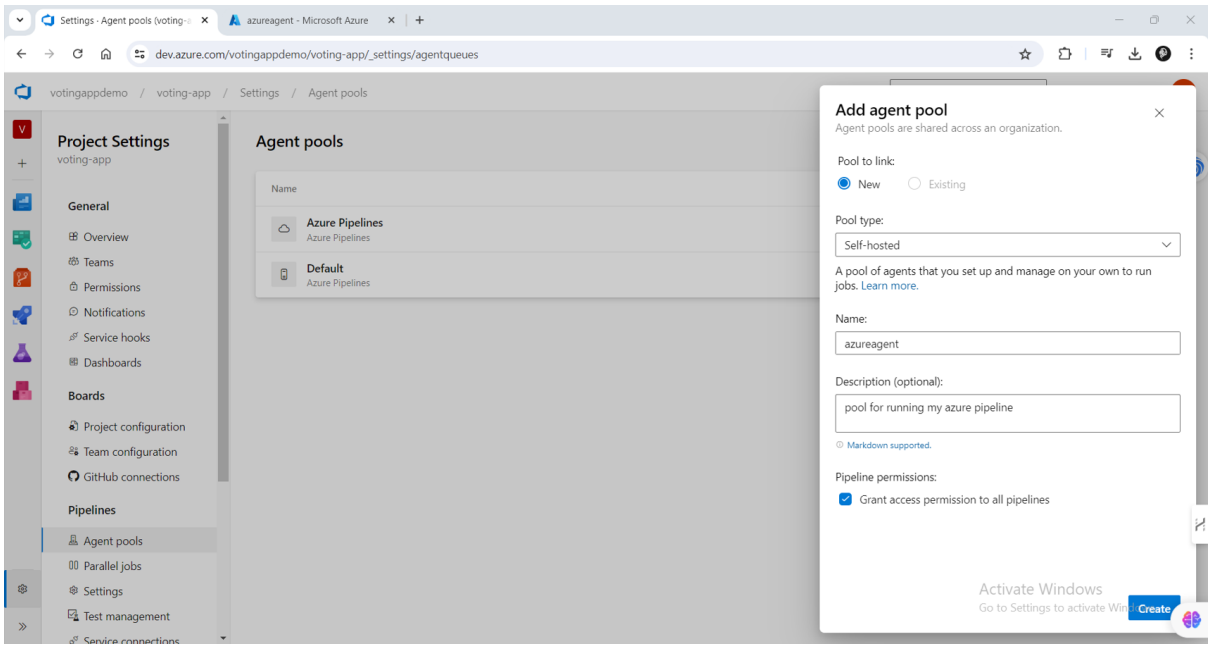
Step-2: Implementing Continuous Integration Pipeline (CI)

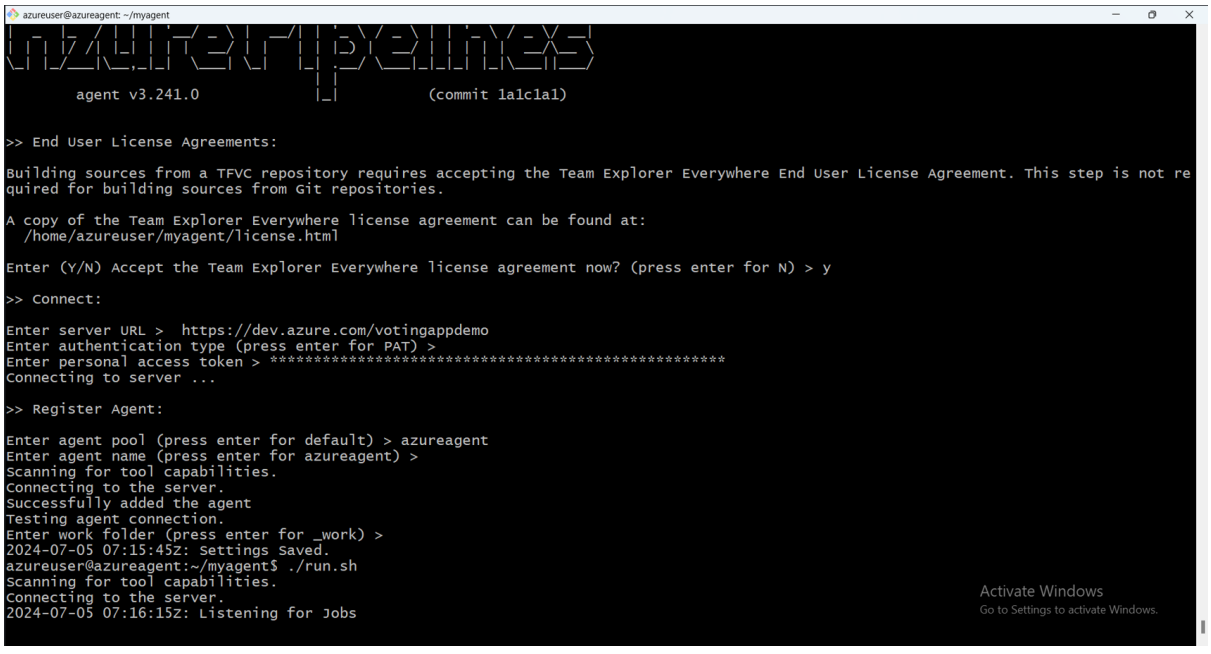
1) Create Agent Pools: A self-managed agent pool named azure agent was configured in Azure DevOps. This pool manages the build and deployment tasks, providing dedicated resources for executing the CI/CD pipelines.

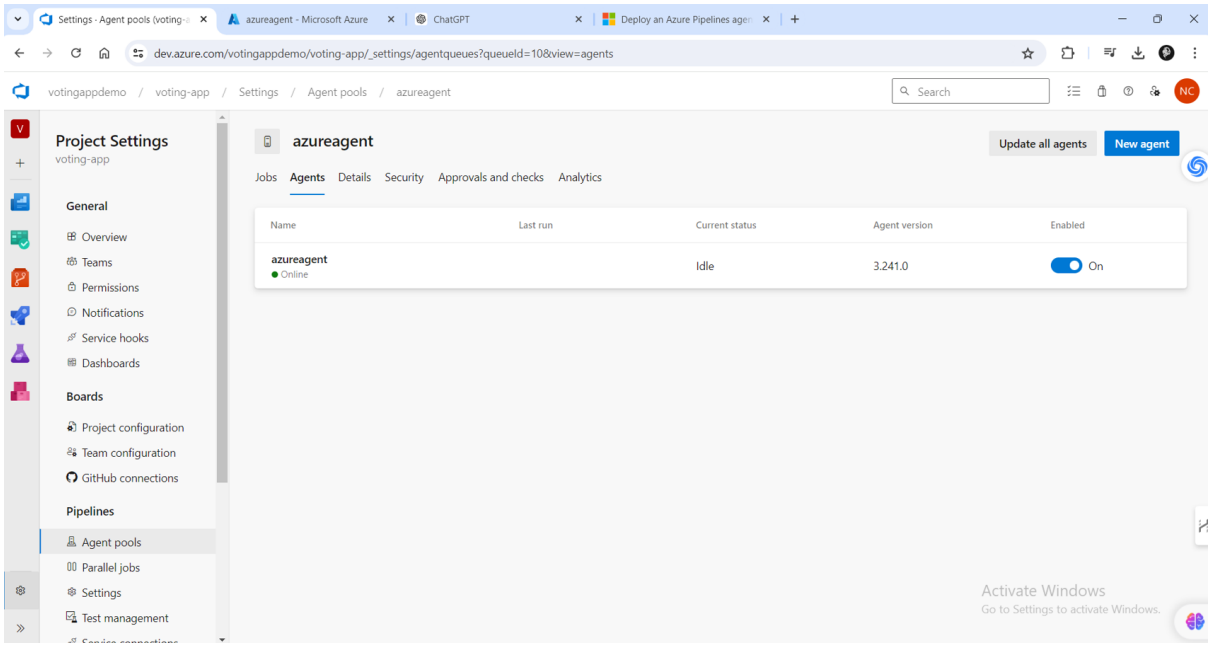
Steps to Create a Self-Hosted Agent Pool in Azure DevOps

- **Sign in to Azure DevOps**: Navigate to Azure DevOps portal, select your project, and go to project settings.
- **Create a New Agent Pool**: Under **Pipelines**, click **Agent pools**, then **Add pool**, name it, and click **Create**.
- **Download the Agent Package**: In agent pool details, click **New agent**, select your OS, and download the agent package.

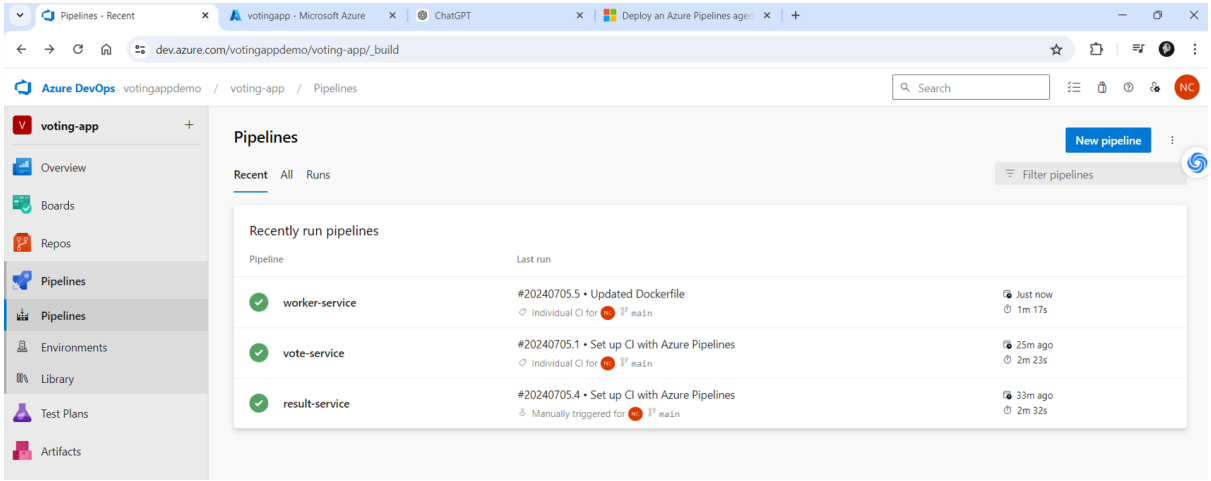
- **Configure the Agent on Your Laptop**: Extract the package, run the config script, and follow prompts for URL, PAT, and agent details.
- **Install and Start the Agent**: Run the run script to start the agent, optionally install as a service using svc commands.
- **Verify in Azure DevOps**: Go back to Azure DevOps portal, select your agent pool, and verify the agent is listed and online.



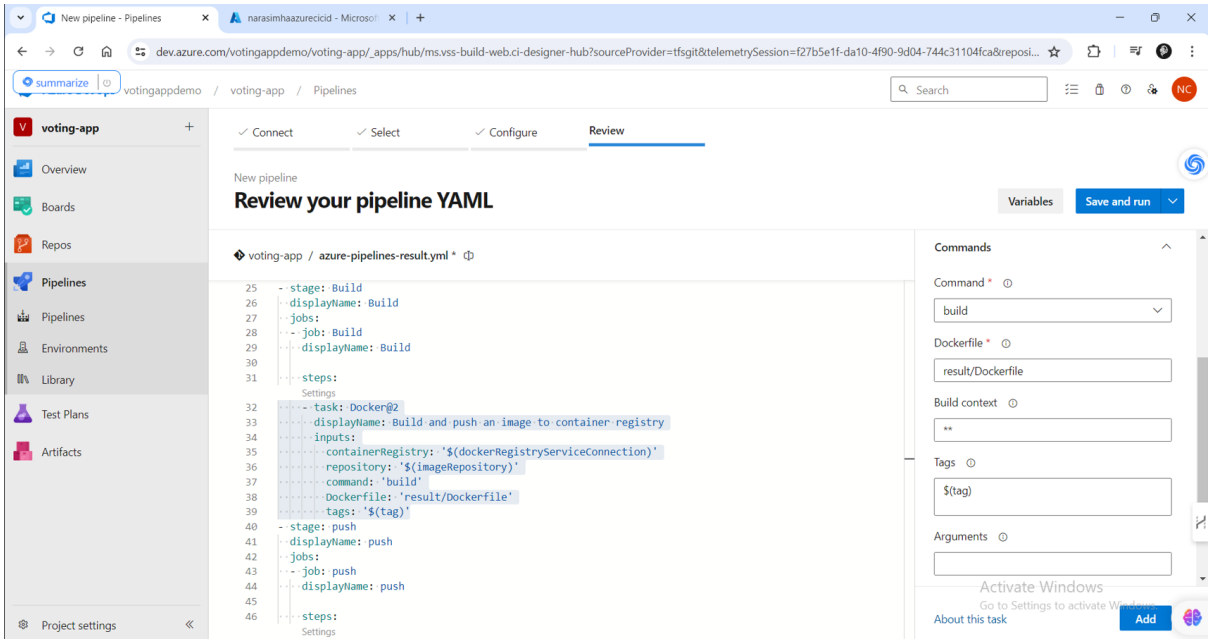




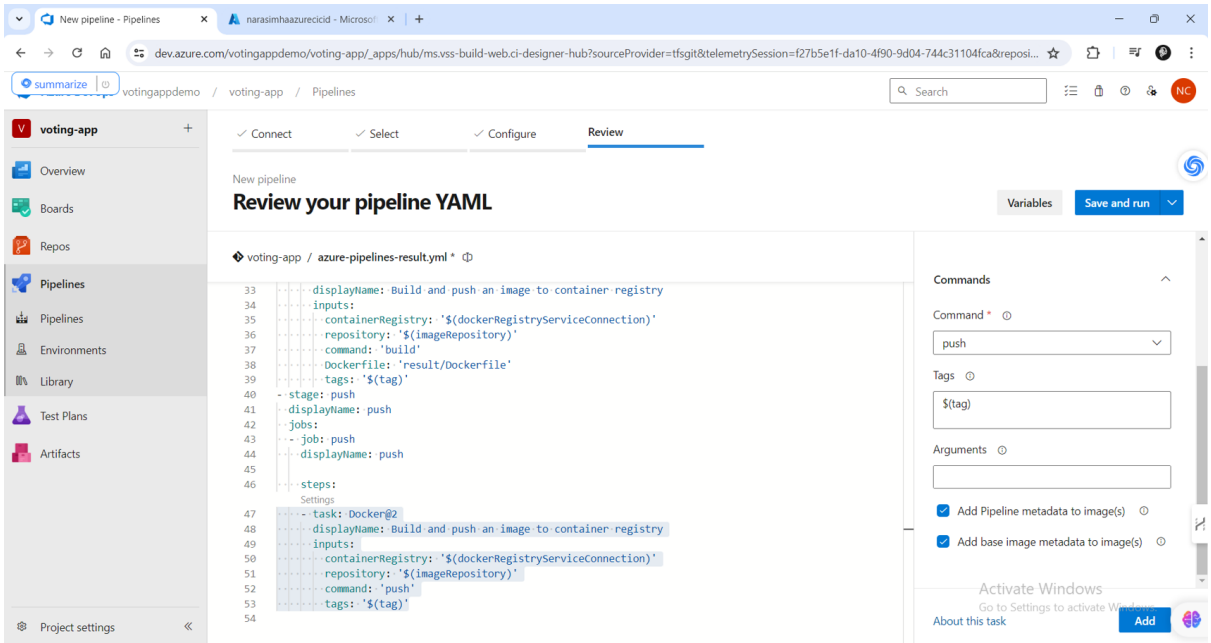
2) Create Three Pipelines: Separate pipelines were established for the **worker**, **vote** and **result** microservices. Each pipeline is dedicated to automating the build and push processes for its respective service.



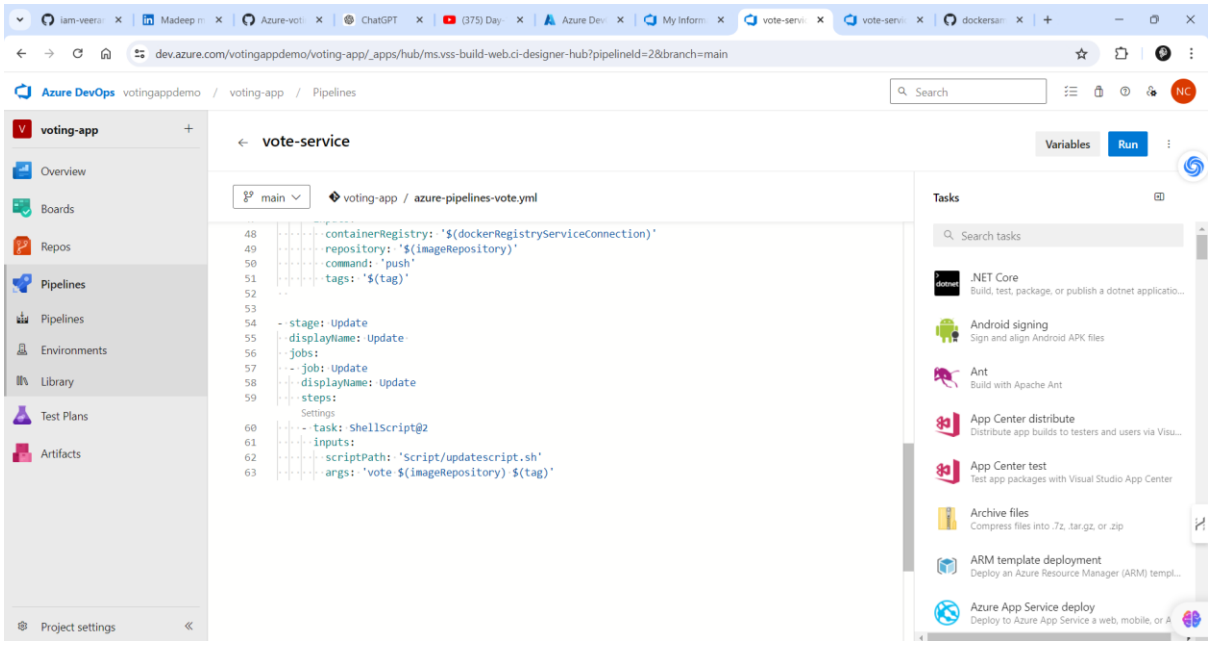
Automated Build Process: Each pipeline is designed to build Docker images from Dockerfiles located in the Azure Repository. This ensures that the latest code changes are compiled into a Docker image automatically.

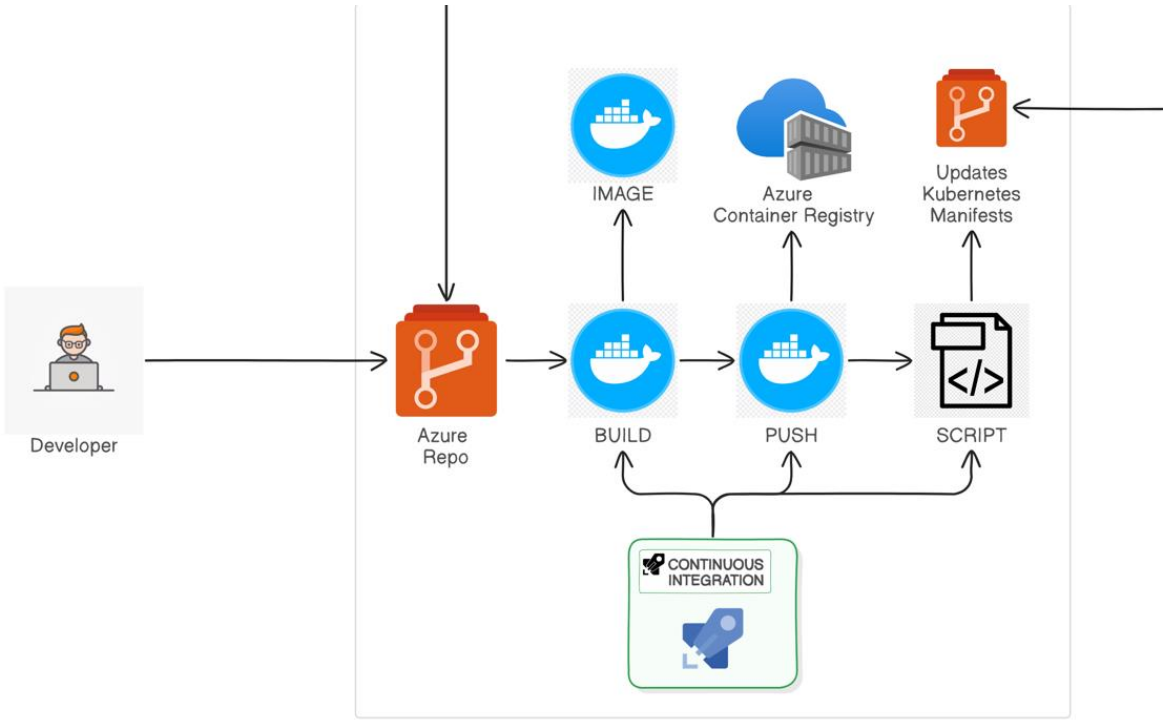


Automated Push Process: Once the Docker images are built, the pipelines automatically push these images to the Azure Container Registry. This step is crucial for maintaining an up-to-date image repository.



Vote Pipeline: Includes an additional Update stage to update deployments with the new image.

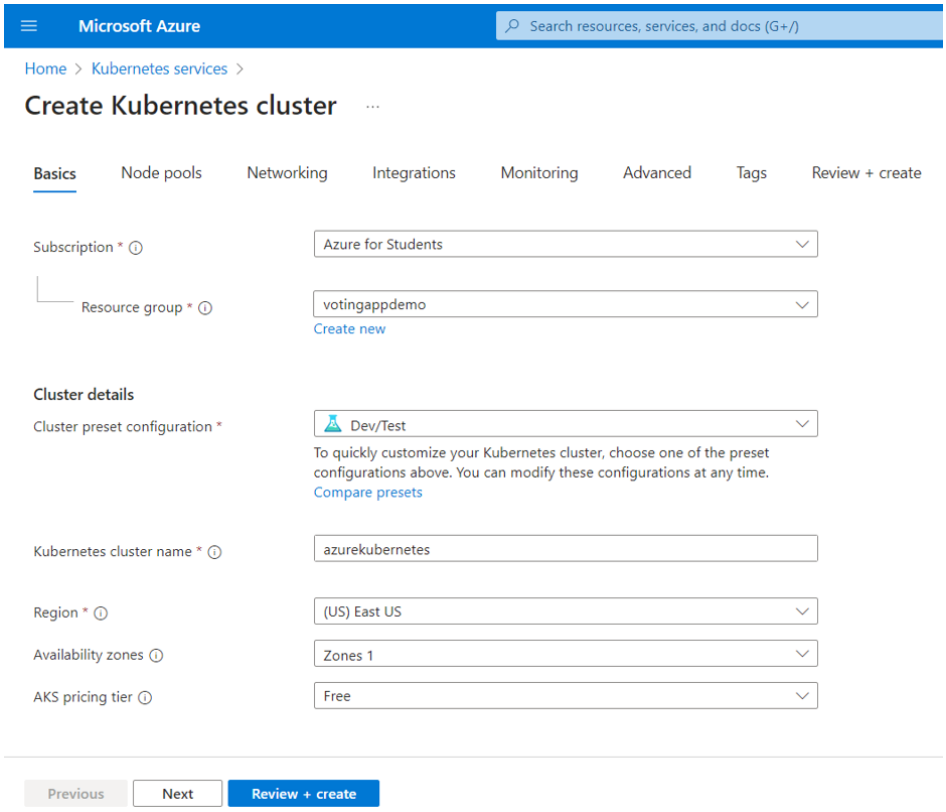


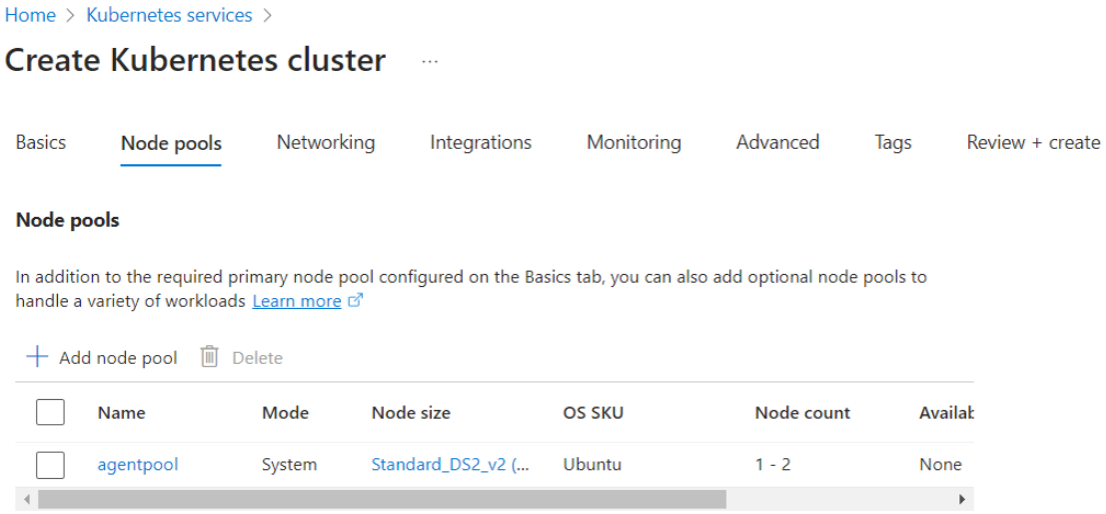


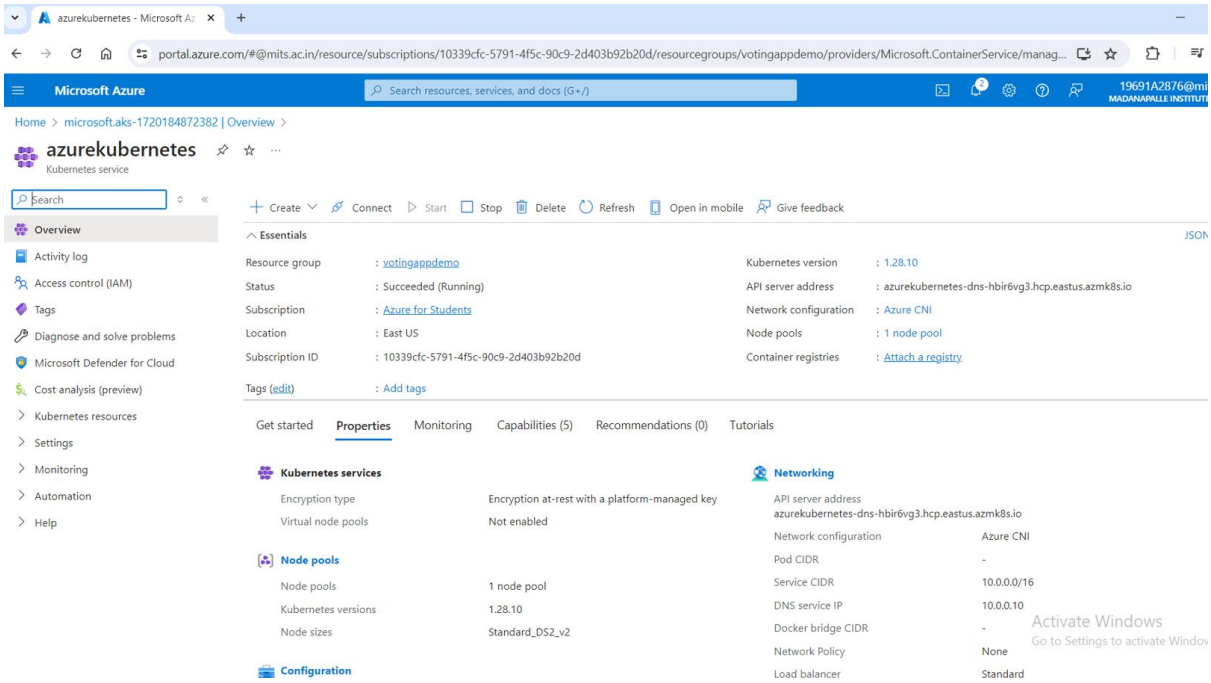
CONTINUOUS INTEGRATION

Step-3: Implementing Continuous Deployment (CD):

1) AKS Cluster Setup: An Azure Kubernetes Service (AKS) cluster was established to manage and deploy containerized applications. Virtual Machine Scale Sets were used as node pools to dynamically scale the cluster based on the workload.

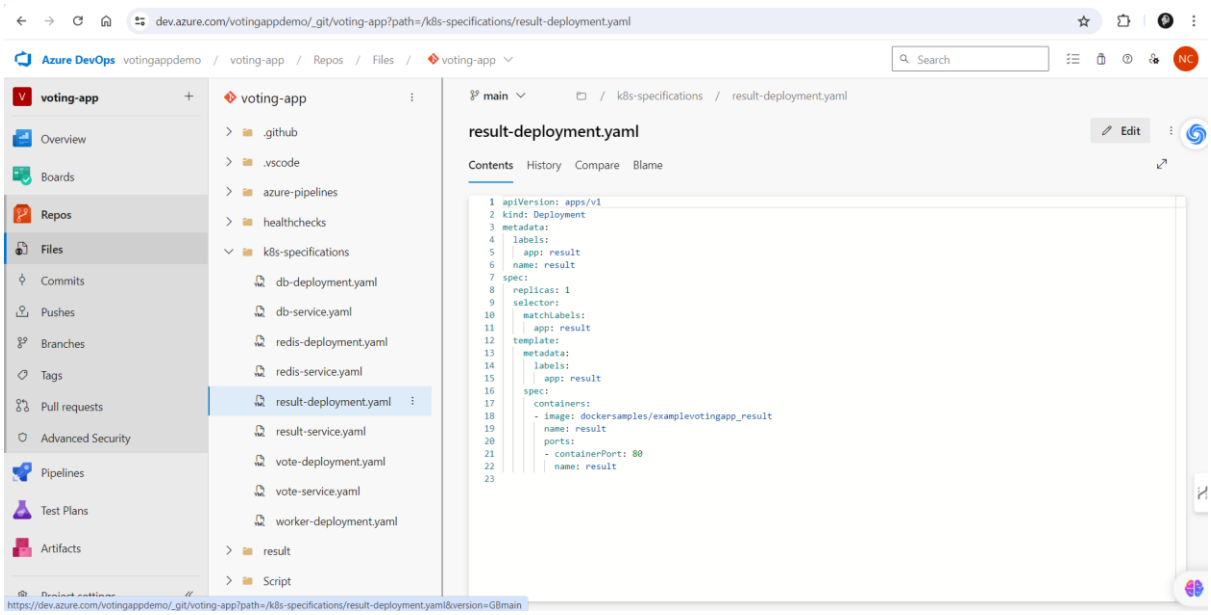






2) Deployment and Service Configuration: YAML files were created and configured for deployments and services within the AKS cluster. These files define the desired state and configuration of the infrastructure and applications, ensuring a consistent deployment environment.

- Deployment YAML files specify the desired state for each microservice, including the container image, replicas, and update strategies. They ensure that the correct version of the application is deployed with the desired number of replicas for high availability.
- Service YAML files define how each microservice is exposed within the AKS cluster. They specify the type of service (e.g., ClusterIP, LoadBalancer), port mappings, and selectors to route traffic to the appropriate pods, ensuring seamless communication between services and external access when needed.



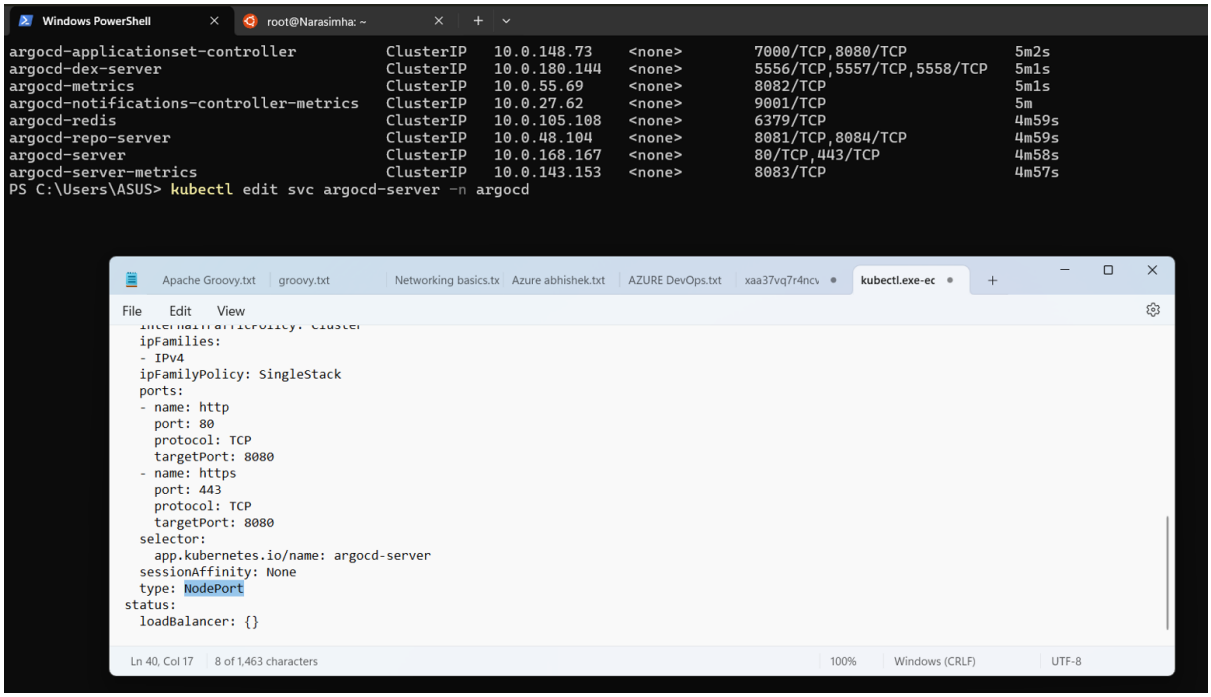
3) Install and configure the ArgoCD: ArgoCD was installed and configured to automate and manage the deployment process. It continuously monitors the Git repository for changes and synchronizes the defined configurations to maintain the desired application state within the AKS cluster.

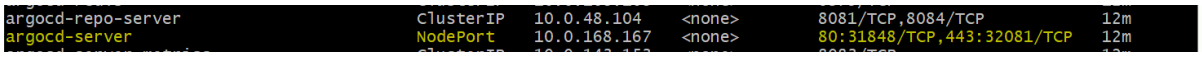
Steps to install and configure the Argocd

- **Install ArgoCD:** Deploy ArgoCD to your AKS cluster by applying the official ArgoCD installation manifest. This sets up the necessary components within a dedicated namespace, ensuring a clean and isolated environment for ArgoCD.

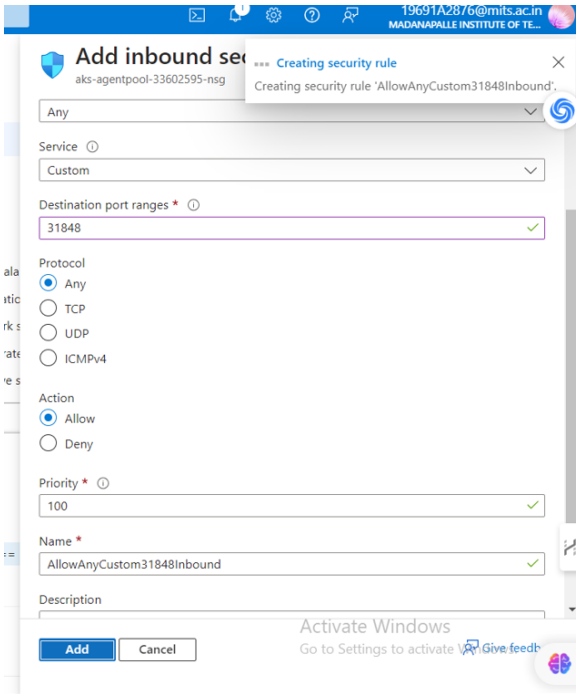
```
PS C:\Users\ASUS> kubectl create namespace argocd
namespace/argocd created
PS C:\Users\ASUS> kubectl apply -n argocd -f https://raw.githubusercontent.com/argoproj/argo-cd/stable/manifests/install.yaml
customresourcedefinition.apiextensions.k8s.io/applications.argoproj.io created
customresourcedefinition.apiextensions.k8s.io/applicationsets.argoproj.io created
customresourcedefinition.apiextensions.k8s.io/appprojects.argoproj.io created
serviceaccount/argocd-application-controller created
serviceaccount/argocd-applicationset-controller created
serviceaccount/argocd-dex-server created
serviceaccount/argocd-notifications-controller created
serviceaccount/argocd-redis created
serviceaccount/argocd-repo-server created
serviceaccount/argocd-server created
```

- **Create and Expose ArgoCD Namespace:** Create a namespace for ArgoCD (e.g., `argocd`) to manage its resources. Expose ArgoCD services by configuring either a LoadBalancer service or an Ingress resource, making the ArgoCD UI accessible externally.

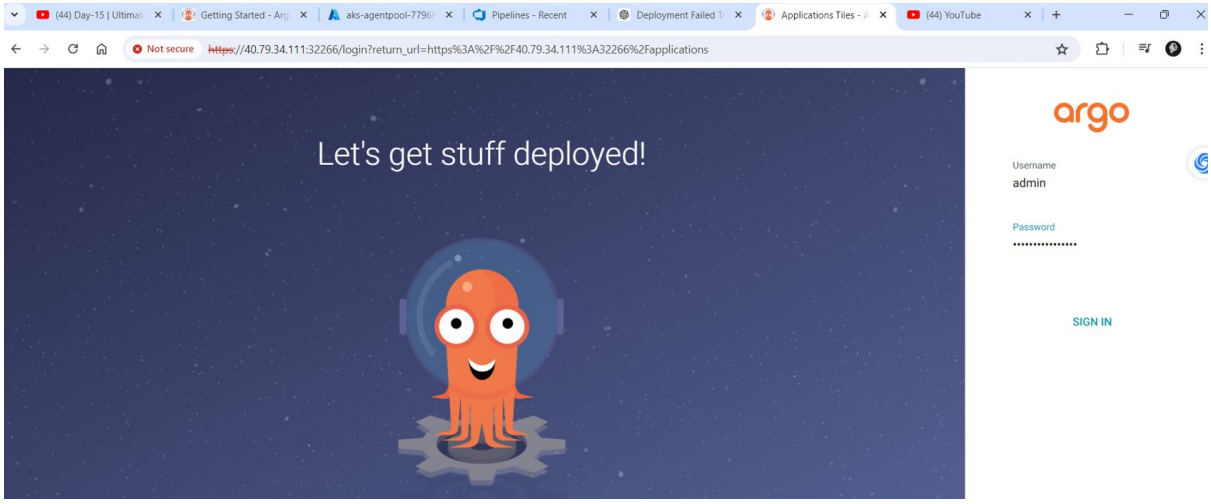




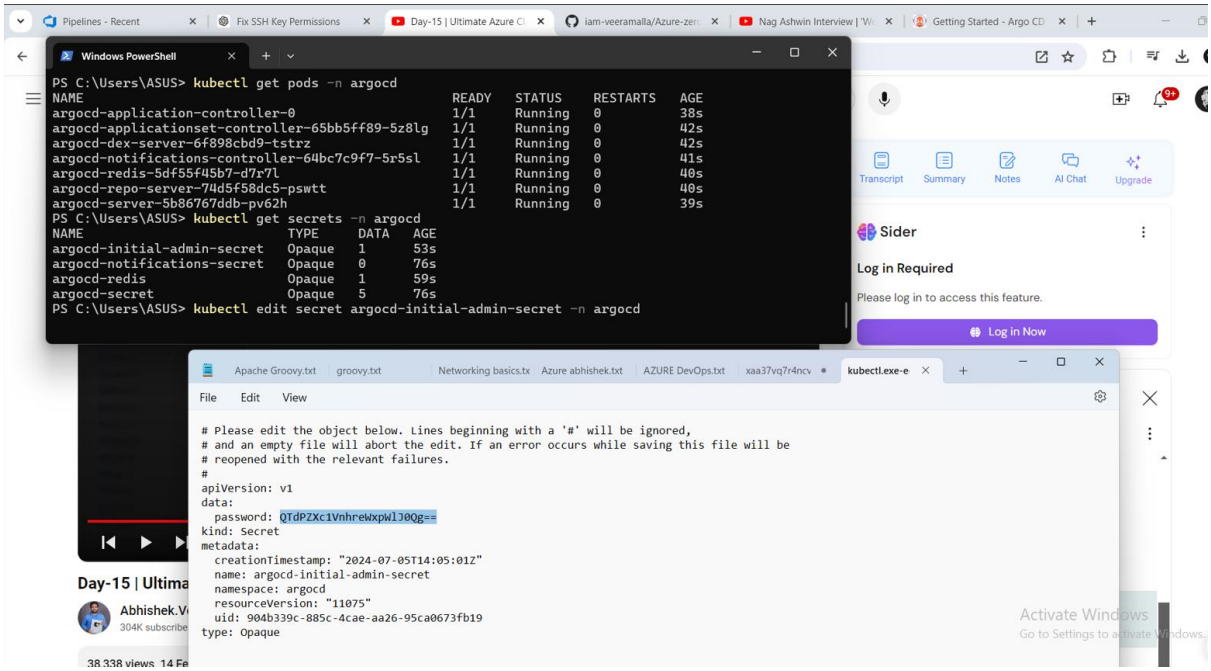
- **Access ArgoCD UI:** Obtain the initial admin password and use it to log in to the ArgoCD web UI. This allows you to manage and configure ArgoCD applications through its user interface.
- Add The inbound security rule for the AKS Agent pool to access the ArgoCD UI



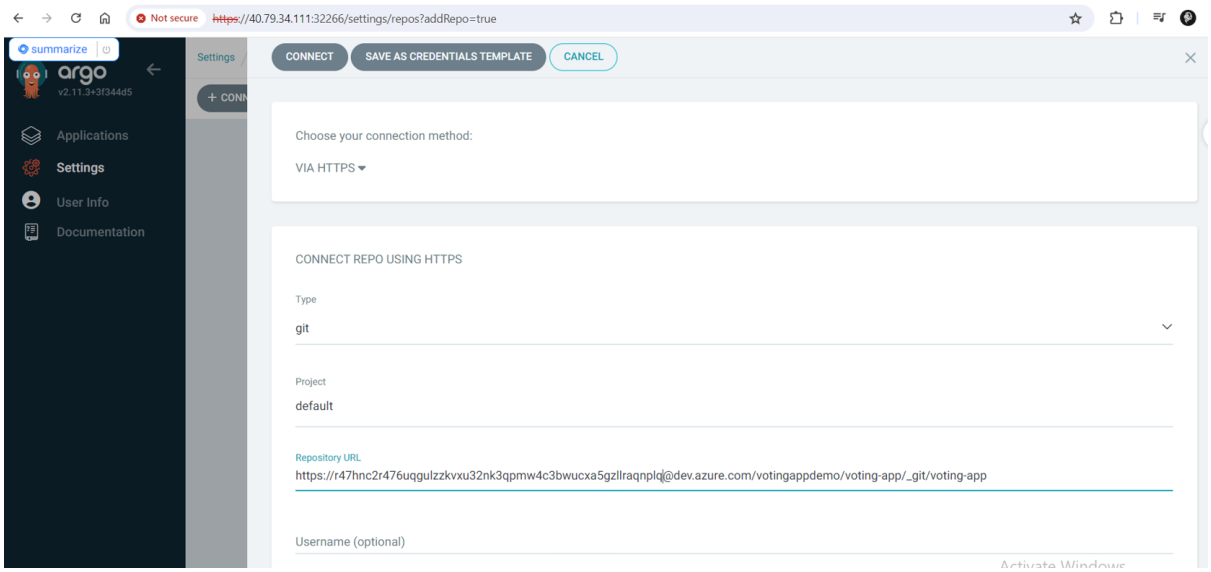
- Now Access the ArgoCD using : <IP of the machine>:<Argocd-Server Port>

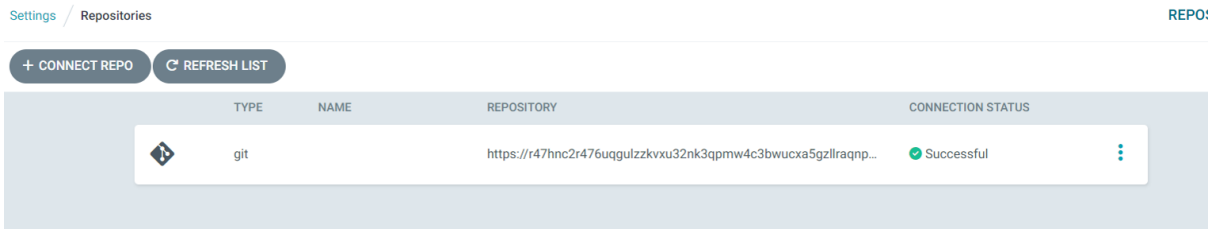


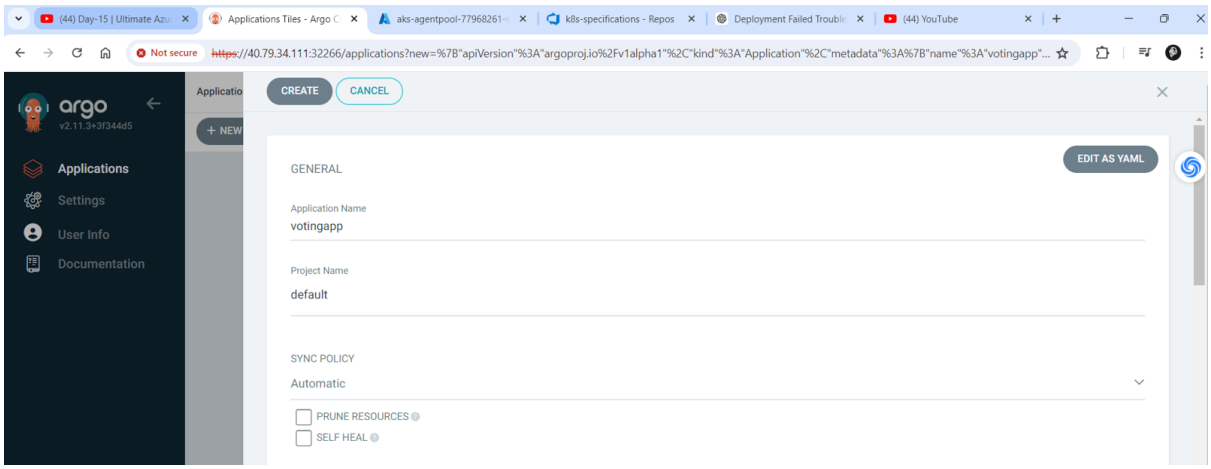
- Get the password from the secrets

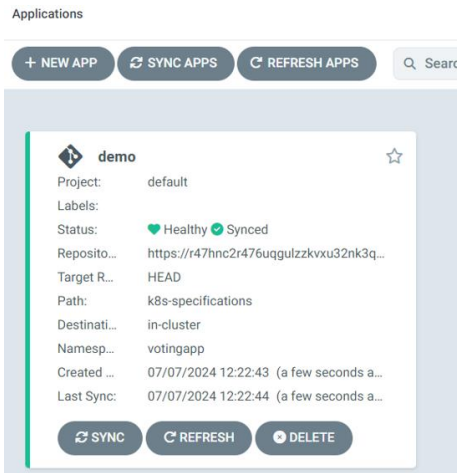


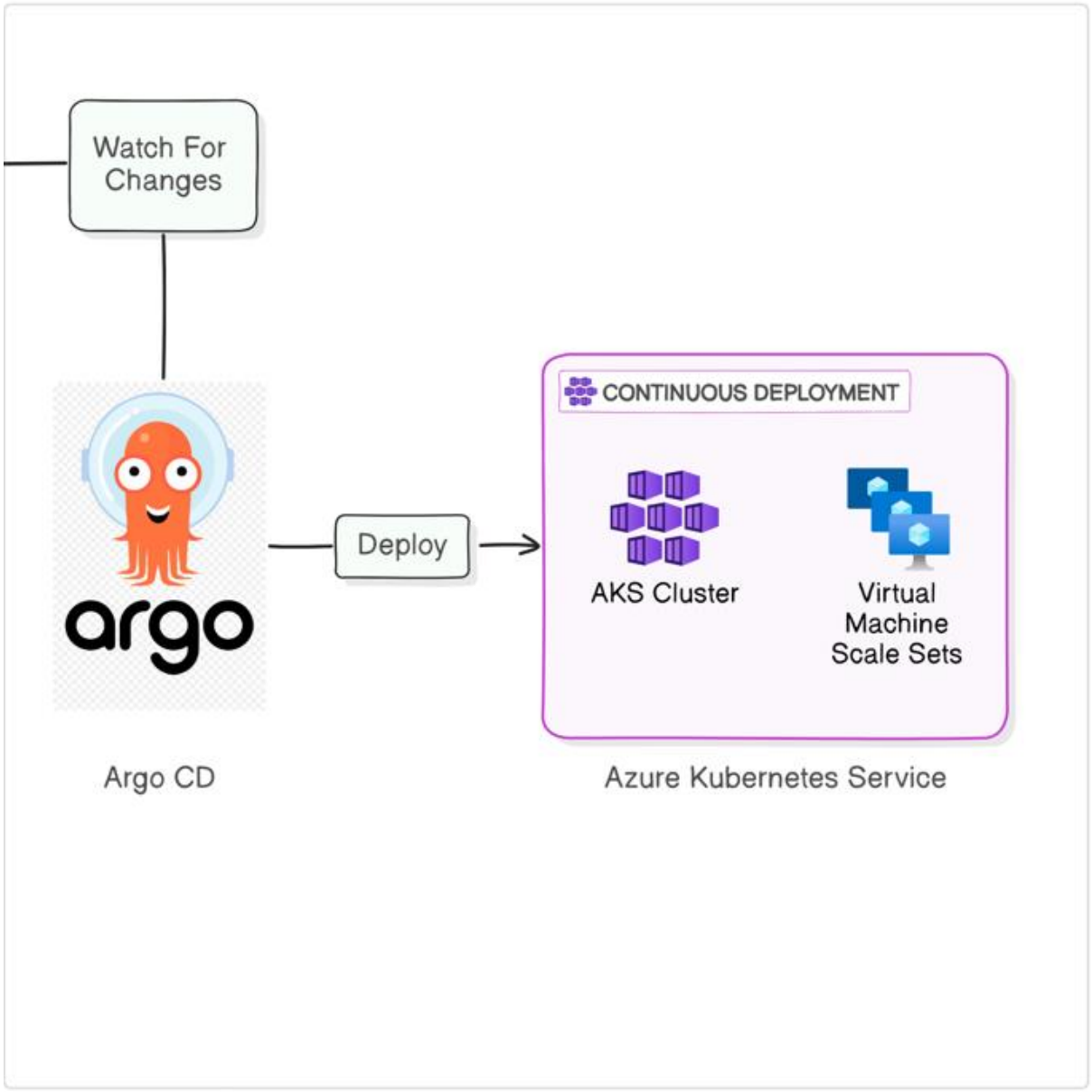
- **Connect to GIT Repository:** Configure ArgoCD to connect to your Git repository where Kubernetes manifests are stored. This enables ArgoCD to access, monitor, and synchronize your application's deployment configurations from the repository.
- **Create ArgoCD Applications:** Define ArgoCD applications to represent your deployments. Set up these applications to specify the source repository and path for your Kubernetes manifests and configure the target namespace within your AKS cluster.


Not secure https://40.79.34.111:32266/settings/repos?addRepo=true
summarize
argo
v2.11.3+3f344d5
Applications
Settings
User Info
Documentation
Settings
+ CONN
CONNECT
SAVE AS CREDENTIALS TEMPLATE
CANCEL
Choose your connection method:
VIA HTTPS
CONNECT REPO USING HTTPS
Type
git
Project
default
Repository URL
https://r47hnc2r476uqgulzzkvxu32nk3qpmw4c3bwucxa5gzllraqnplq@dev.azure.com/votingappdemo/voting-app/_git/voting-app
Username (optional)


Settings
Repositories
REPOS
+ CONNECT REPO
REFRESH LIST
TYPE
NAME
REPOSITORY
CONNECTION STATUS
git
https://r47hnc2r476uqgulzzkvxu32nk3qpmw4c3bwucxa5gzllraqnp...
Successful


Not secure https://40.79.34.111:32266/applications?new=%7B"apiVersion"%3A"argoproj.io%2Fv1alpha1"%2C"kind"%3A"Application"%2C"metadata"%3A%7B"name"%3A"votingapp"...
argo
v2.11.3+3f344d5
Applications
Settings
User Info
Documentation
CREATE
CANCEL
EDIT AS YAML
GENERAL
Application Name
votingapp
Project Name
default
SYNC POLICY
Automatic
PRUNE RESOURCES
SELF HEAL


Applications
+ NEW APP
SYNC APPS
REFRESH APPS
demo
Project: default
Labels:
Status: Healthy Synced
Reposito... https://r47hnc2r476uqgulzzkvxu32nk3q...
Target R... HEAD
Path: k8s-specifications
Destinati... in-cluster
Namesp... votingapp
Created ... 07/07/2024 12:22:43 (a few seconds a...
Last Sync: 07/07/2024 12:22:44 (a few seconds a...
SYNC
REFRESH
DELETE

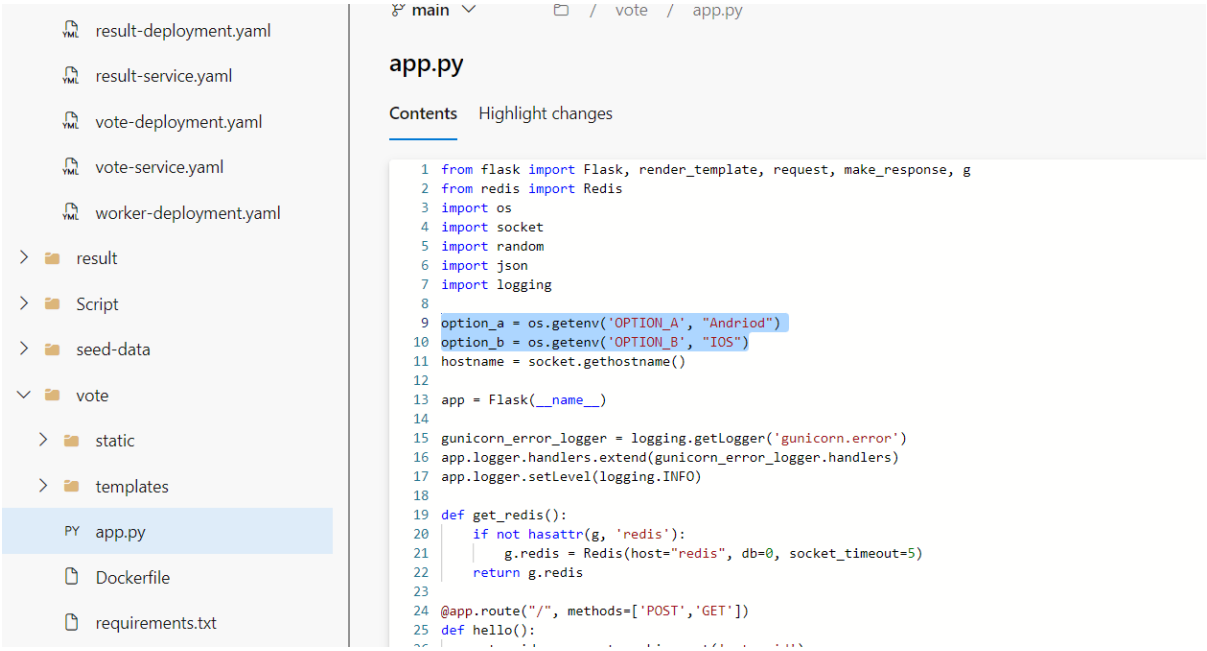


CONTINUOUS DEPLOYMENT

Testing the project:

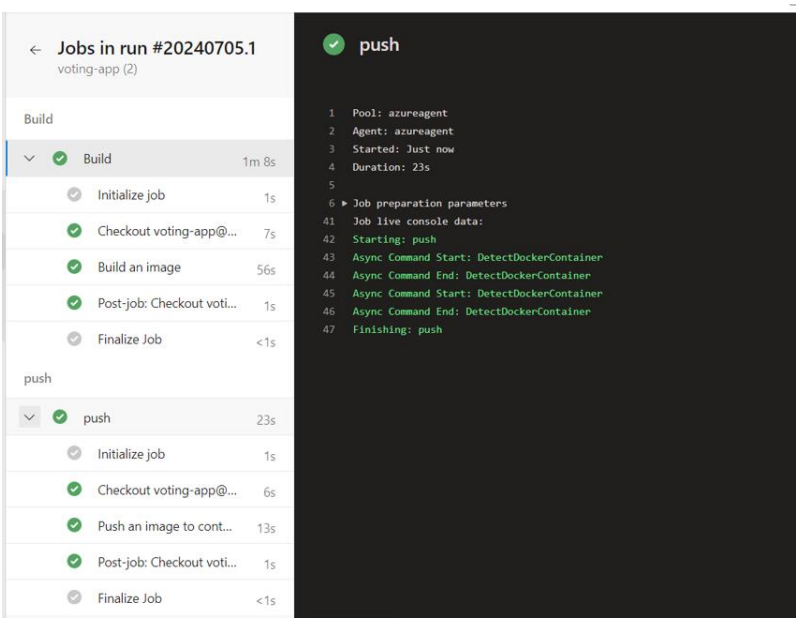
Modify Code in GitHub:

- Start by making changes to the application code in the GitHub repository. These changes could be updates to functionality, bug fixes, or new features.

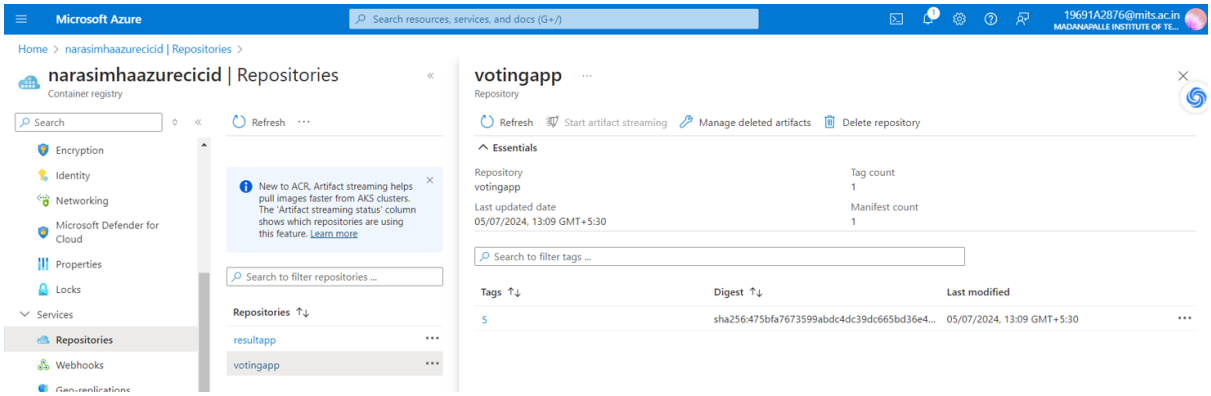


Triggers CI Pipeline:

- The code changes automatically trigger the Continuous Integration (CI) pipeline. This pipeline builds a new Docker image with the updated code and pushes it to the Azure Container Registry.

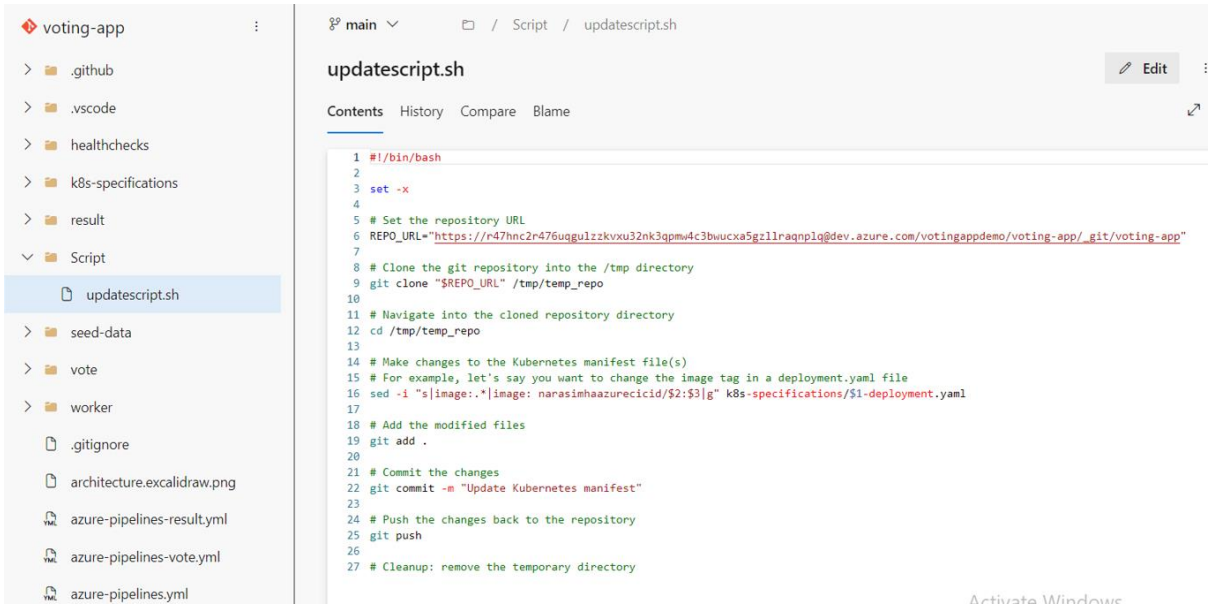


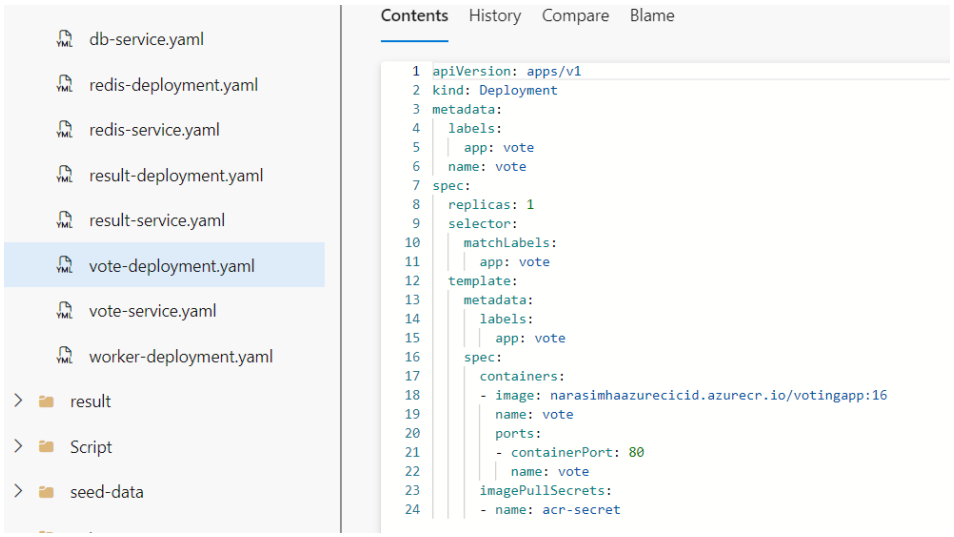
New image will be stored in the azure Container Registry



Updates Deployment Manifests:

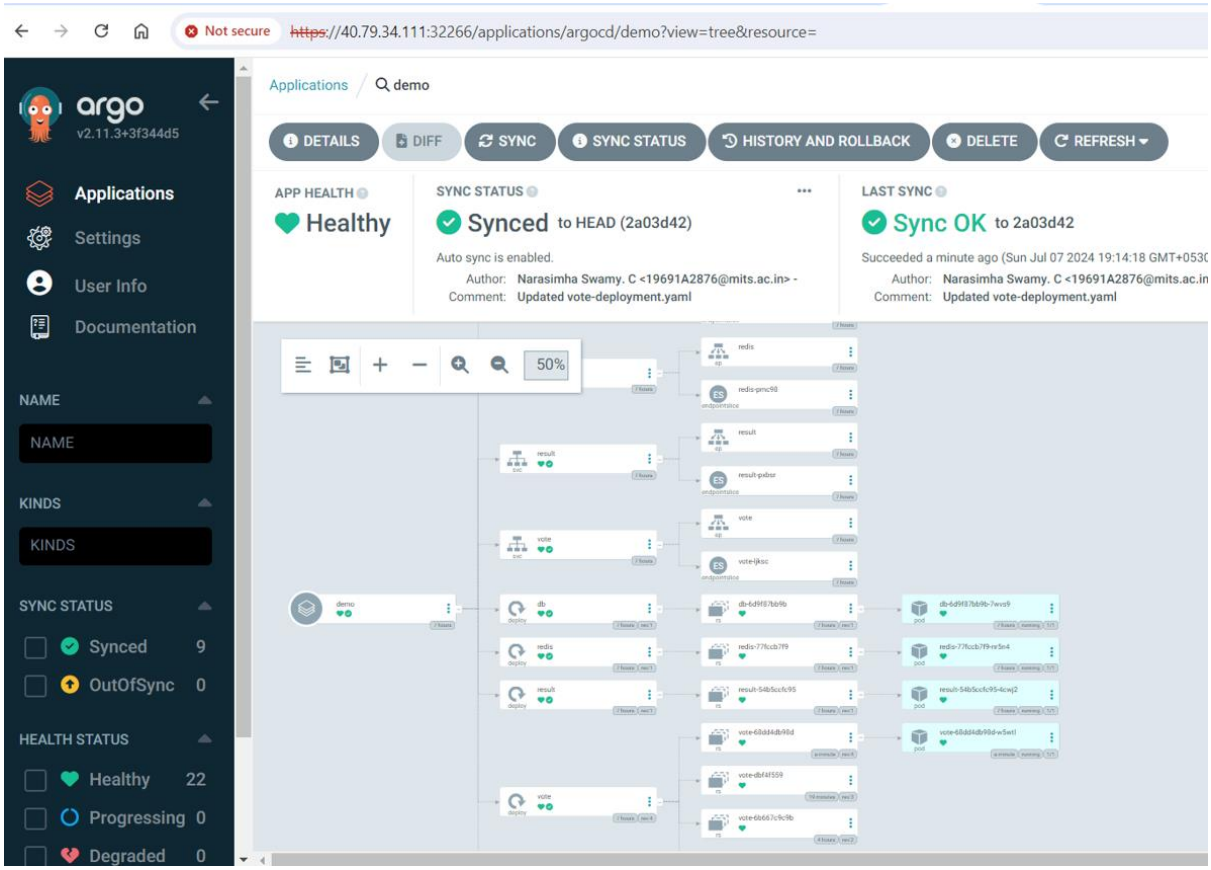
- A shell script updates the Kubernetes manifest files with the new Docker image details. This ensures that the AKS deployment configurations reflect the latest image.





Auto Deploys with ArgoCD:

- ArgoCD detects changes in the Git repository and synchronizes the updated manifests. It then deploys the new image to the AKS cluster, ensuring that the latest version of the application is live.



Then the new Applications will be deployed in the AKS cluster, and new application can be accessed through accessing the IP with port.