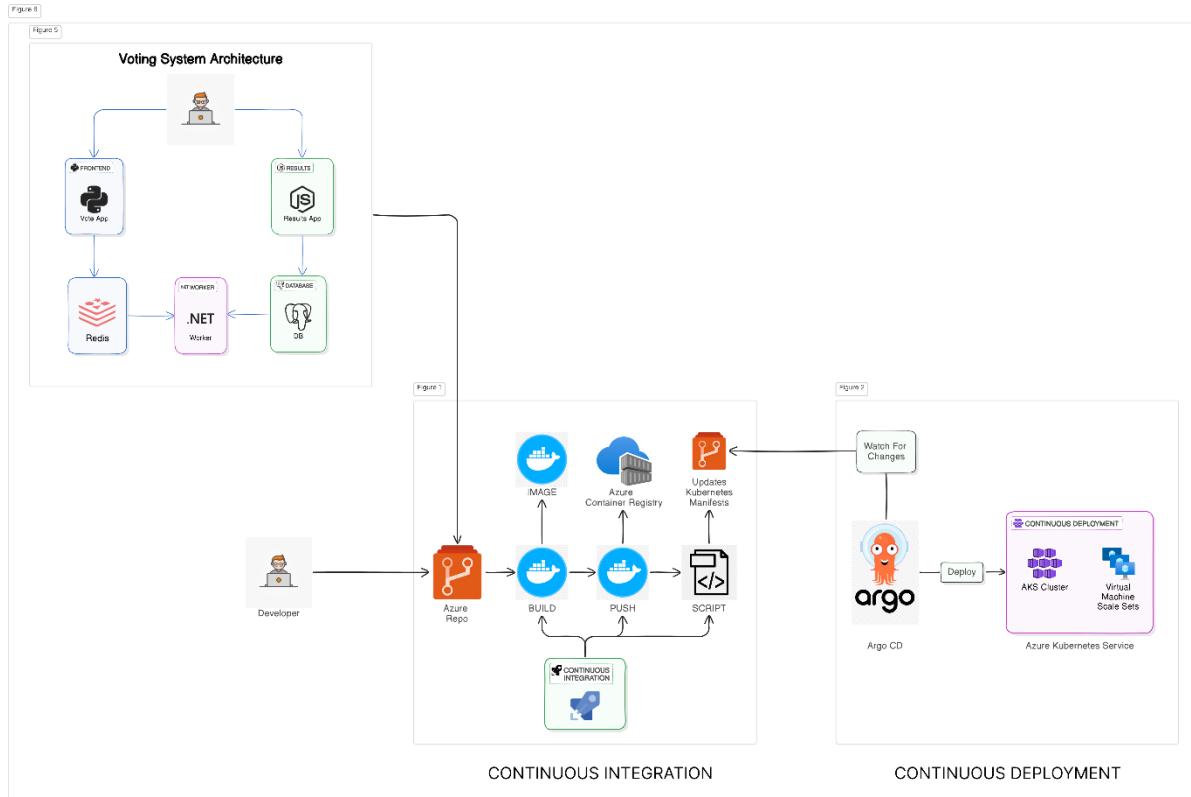


# Complete End-to-End CI/CD Process with Azure DevOps and AKS Cluster Deployment



## 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.

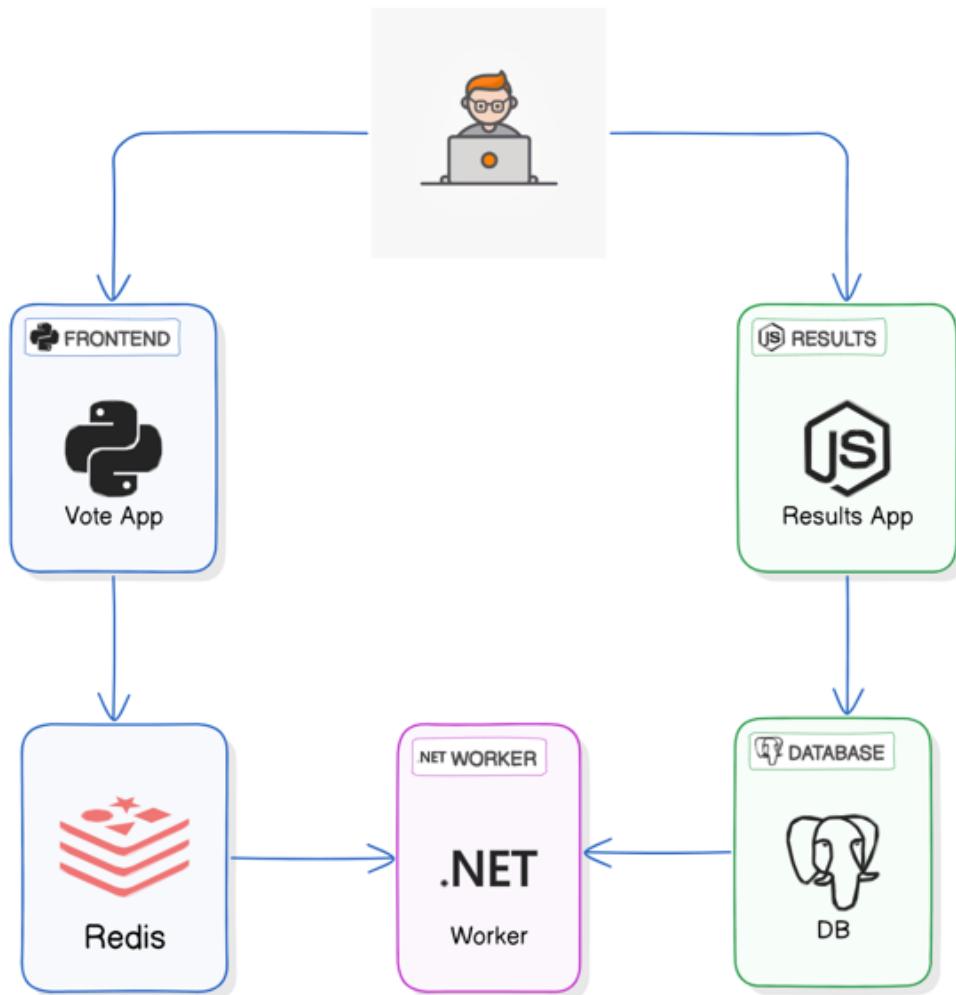
The application includes:

- 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.

The screenshot shows a GitHub repository page for 'example-voting-app'. The repository is public and has 6 branches and 0 tags. The main branch is selected. The repository description states: "Example distributed app composed of multiple containers for Docker, Compose, Swarm, and Kubernetes". It includes tags for docker, kubernetes, sample, demo, docker-compose, example, and swarm. The repository has 123 stars, 10k forks, and 4.5k issues. The commit history shows contributions from mikesir87, including a merge pull request (#311) and several commits related to Docker, Compose, and Kubernetes configurations. The repository also includes a Readme file, an Apache-2.0 license, activity logs, custom properties, and a report repository link.

The screenshot shows an Azure DevOps repository page for 'voting-app'. The repository is part of the 'votingappdemo' project. The 'Files' tab is selected, showing the contents of the 'main' branch. The contents include .github, .vscode, azure-pipelines, healthchecks, k8s-specifications, result, seed-data, vote, worker, .gitignore, architecture.excalidraw.png, azure-pipelines-result.yml, azure-pipelines-vote.yml, azure-pipelines.yml, create-secret in k8's, and docker-compose.yml. The repository has 123 watching, 10k forks, and 4.5k stars. The 'Set up build' button is visible at the top right of the file list.

# Voting System Architecture



## 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.

The screenshot shows the Azure DevOps portal interface. On the left, there's a sidebar with 'Project Settings' for 'voting-app'. Under 'Pipelines', 'Agent pools' is selected. The main area shows 'Agent pools' with two entries: 'Azure Pipelines' (Azure Pipelines) and 'Default' (Azure Pipelines). A modal window titled 'Add agent pool' is overlaid on the page. It contains fields for 'Name' (set to 'azureagent'), 'Pool type:' (set to 'Self-hosted'), and a 'Description (optional)' field containing 'pool for running my azure pipeline'. There are also tabs for 'Pool to link:' (New is selected) and 'Pipeline permissions:' (checkbox for 'Grant access permission to all pipelines' is checked). At the bottom right of the modal, there's an 'Activate Windows' message with a 'Create' button.

```

azureuser@azureagent:~/myagent
[REDACTED]
agent v3.241.0          (commit lalcial)

>> End User License Agreements:
Building sources from a TFVC repository requires accepting the Team Explorer Everywhere End User License Agreement. This step is not required for building sources from Git repositories.
A copy of the Team Explorer Everywhere license agreement can be found at:
  /home/azureuser/myagent/license.html

Enter (Y/N) Accept the Team Explorer Everywhere license agreement now? (press enter for N) > y

>> Connect:
Enter server URL > https://dev.azure.com/votingappdemo
Enter authentication type (press enter for PAT) >
Enter personal access token > *****
Connecting to server ...

>> Register Agent:
Enter agent pool (press enter for default) > azureagent
Enter agent name (press enter for azureagent) >
Scanning for tool capabilities.
Connecting to the server.
Successfully added the agent
Testing agent connection.
Enter work folder (press enter for _work) >
2024-07-05 07:15:45Z: Settings saved.
azureuser@azureagent:~/myagent$ ./run.sh
Scanning for tool capabilities.
Connecting to the server.
2024-07-05 07:16:15Z: Listening for jobs

```

The screenshot shows a terminal window with the output of the 'run.sh' script. It starts with the agent version and commit information. It then prompts for accepting the EULA, which is done by entering 'y'. It connects to the specified server URL using a PAT. It registers the agent with the pool 'azureagent' and successfully adds it. It then tests the connection and starts listening for jobs. At the bottom right, there's an 'Activate Windows' message with a 'Create' button.

The screenshot shows the 'Agent pools' section of the Azure DevOps interface. On the left, a sidebar lists project settings like General, Boards, Pipelines, and Agent pools. The 'Agent pools' item is selected. The main area displays the 'azureagent' pool details. It includes tabs for Jobs, Agents, Details, Security, Approvals and checks, and Analytics. Under the Agents tab, a table lists one agent named 'azureagent' which is online and idle, running version 3.241.0. There are buttons for 'Update all agents' and 'New agent'. A message at the bottom right says 'Activate Windows'.

**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.

The screenshot shows the 'Pipelines' section of the Azure DevOps interface. The sidebar shows the 'voting-app' project with the 'Pipelines' item selected. The main area displays the 'Pipelines' table under the 'Recent' tab. It lists three pipelines: 'worker-service', 'vote-service', and 'result-service', each with a green checkmark indicating they have been successfully run. The table includes columns for Pipeline name, Last run, and time since run.

Pipeline	Last run	
worker-service	#20240705.5 • Updated Dockerfile Individual CI for 🚧 main	Just now 1m 17s
vote-service	#20240705.1 • Set up CI with Azure Pipelines Individual CI for 🚧 main	25m ago 2m 23s
result-service	#20240705.4 • Set up CI with Azure Pipelines Manually triggered for 🚧 main	33m ago 2m 32s

**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.

```

25 - stage: Build
26   displayName: Build
27   jobs:
28     - job: Build
29       displayName: Build
30       steps:
31         - settings:
32           task: Docker@2
33             displayName: Build and push an image to container registry
34             inputs:
35               containerRegistry: '$(dockerRegistryServiceConnection)'
36               repository: '$(imageRepository)'
37               command: 'build'
38               Dockerfile: 'result/Dockerfile'
39               tags: '${tag}'
40     - stage: push
41       displayName: push
42       jobs:
43         - job: push
44           displayName: push
45           steps:
46             - settings:

```

**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.

```

33     - settings:
34       task: Docker@2
35         displayName: Build and push an image to container registry
36         inputs:
37           containerRegistry: '$(dockerRegistryServiceConnection)'
38           repository: '$(imageRepository)'
39           command: 'build'
40           Dockerfile: 'result/Dockerfile'
41           tags: '${tag}'
42     - stage: push
43       displayName: push
44       jobs:
45         - job: push
46           displayName: push
47           steps:
48             - settings:
49               task: Docker@2
50                 displayName: Build and push an image to container registry
51                 inputs:
52                   containerRegistry: '$(dockerRegistryServiceConnection)'
53                   repository: '$(imageRepository)'
54                   command: 'push'
55                   tags: '${tag}'

```

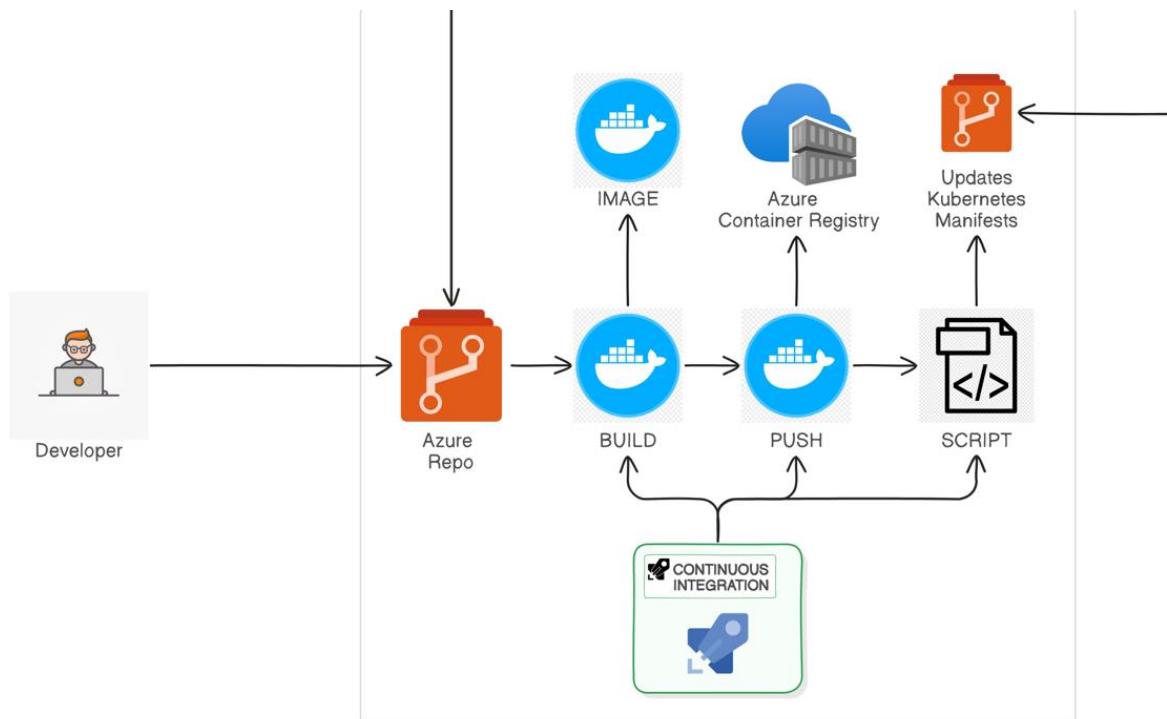
**Vote Pipeline:** Includes an additional Update stage to update deployments with the new image.

Screenshot of the Azure DevOps Pipelines interface showing the configuration of a pipeline named "vote-service" for the "voting-app" project. The pipeline file "azure-pipelines-vote.yml" is displayed, containing YAML code for building and pushing Docker images to a registry. The pipeline includes a stage named "Update" with a job named "Update" that runs a shell script. The pipeline also includes a "Settings" section with a task named "Shellscript@2". The right side of the screen shows a list of available tasks, including ".NET Core", "Android signing", "Ant", "App Center distribute", "App Center test", "Archive files", "ARM template deployment", and "Azure App Service deploy".

```

48   - containerRegistry: '$(dockerRegistryServiceConnection)'
49   - repository: '${{imageRepository}}'
50   - command: 'push'
51   - tags: '${{tag}}'
52
53
54   - stage: Update
55   - displayName: Update
56   - jobs:
57     - job: Update
58       displayName: Update
59       steps:
60         - task: Shellscript@2
61           inputs:
62             scriptPath: 'Script/updateScript.sh'
63             args: 'vote ${{imageRepository}} ${{tag}}'

```



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.

The screenshot shows the 'Create Kubernetes cluster' wizard in the Microsoft Azure portal. The 'Basics' tab is selected. The configuration includes:

- Subscription: Azure for Students
- Resource group: votingappdemo
- Cluster details:
  - Cluster preset configuration: Dev/Test
  - Kubernetes cluster name: azurekubernetes
  - Region: (US) East US
  - Availability zones: Zones 1
  - AKS pricing tier: Free
- Buttons at the bottom: Previous, Next, Review + create (highlighted in blue)

**Node pools**

In addition to the required primary node pool configured on the Basics tab, you can also add optional node pools to handle a variety of workloads [Learn more](#)

Add node pool	Delete					
<input type="checkbox"/>	Name	Mode	Node size	OS SKU	Node count	Availability
<input type="checkbox"/>	agentpool	System	Standard_DS2_v2 (...	Ubuntu	1 - 2	None

**Essentials**

Resource group	: votingappdemo	Kubernetes version	: 1.28.10
Status	: Succeeded (Running)	API server address	: azurekubernetes-dns-hbir6vg3.hcp.eastus.azmk8s.io
Subscription	: Azure for Students	Network configuration	: Azure CNI
Location	: East US	Node pools	: 1 node pool
Subscription ID	: 10339fcf-5791-4f5c-90c9-2d403b92b20d	Container registries	: <a href="#">Attach a registry</a>
Tags (edit)	: Add tags		

**Kubernetes services**

Encryption type	Encryption at-rest with a platform-managed key	Networking	
Virtual node pools	Not enabled	API server address	azurekubernetes-dns-hbir6vg3.hcp.eastus.azmk8s.io
Node pools	1 node pool	Network configuration	Azure CNI
Kubernetes versions	1.28.10	Pod CIDR	-
Node sizes	Standard_DS2_v2	Service CIDR	10.0.0.0/16
		DNS service IP	10.0.0.10
		Docker bridge CIDR	-
		Network Policy	None
		Load balancer	Standard

**Node pools**

Node pools	1 node pool
Kubernetes versions	1.28.10
Node sizes	Standard_DS2_v2

**Configuration**

**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.

The screenshot shows the Azure DevOps interface for a repository named 'voting-app'. The left sidebar is the navigation menu. The main area shows the file structure under 'k8s-specifications'. The 'result-deployment.yaml' file is selected and its content is displayed in the right pane.

```

1 apiVersion: apps/v1
2 kind: Deployment
3 metadata:
4   labels:
5     app: result
6   name: result
7 spec:
8   replicas: 1
9   selector:
10    matchLabels:
11      app: result
12   template:
13     metadata:
14       labels:
15         app: result
16     spec:
17       containers:
18         - image: dockersamples/examplevotingapp_result
19           name: result
20           ports:
21             - containerPort: 80
22               name: result
23

```

**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.apirextensions.k8s.io/applications.argoproj.io created
customresourcedefinition.apirextensions.k8s.io/applicationsets.argoproj.io created
customresourcedefinition.apirextensions.k8s.io/approjects.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.

```

Windows PowerShell
root@Narasimha: ~
argocd-applicationset-controller   ClusterIP  10.0.148.73    <none>      7000/TCP,8080/TCP  5m2s
argocd-dex-server                 ClusterIP  10.0.180.144   <none>      5556/TCP,5557/TCP,5558/TCP  5m1s
argocd-metrics                   ClusterIP  10.0.55.69    <none>      8082/TCP  5m1s
argocd-notifications-controller-metrics ClusterIP  10.0.27.62    <none>      9001/TCP  5m
argocd-redis                      ClusterIP  10.0.105.108   <none>      6379/TCP  4m59s
argocd-repo-server                ClusterIP  10.0.48.104   <none>      8081/TCP,8084/TCP  4m59s
argocd-server                     ClusterIP  10.0.168.167   <none>      80/TCP,443/TCP  4m58s
argocd-server-metrics              ClusterIP  10.0.143.153   <none>      8083/TCP  4m57s
PS C:\Users\ASUS> kubectl edit svc argocd-server -n argocd

```

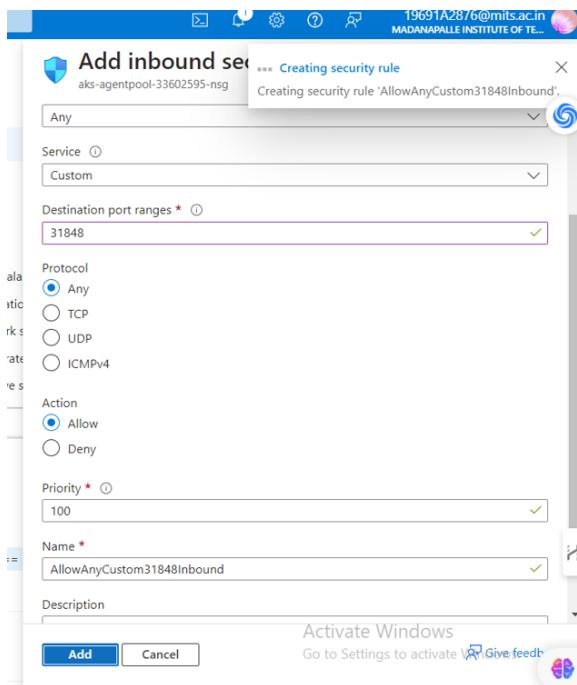
```

apiVersion: v1
kind: Service
metadata:
  name: argocd-server
spec:
  selector:
    app.kubernetes.io/name: argocd-server
  ports:
    - name: https
      port: 443
      protocol: TCP
      targetPort: 32081
    - name: http
      port: 80
      protocol: TCP
      targetPort: 32081
  type: NodePort
  status:
    loadBalancer: {}

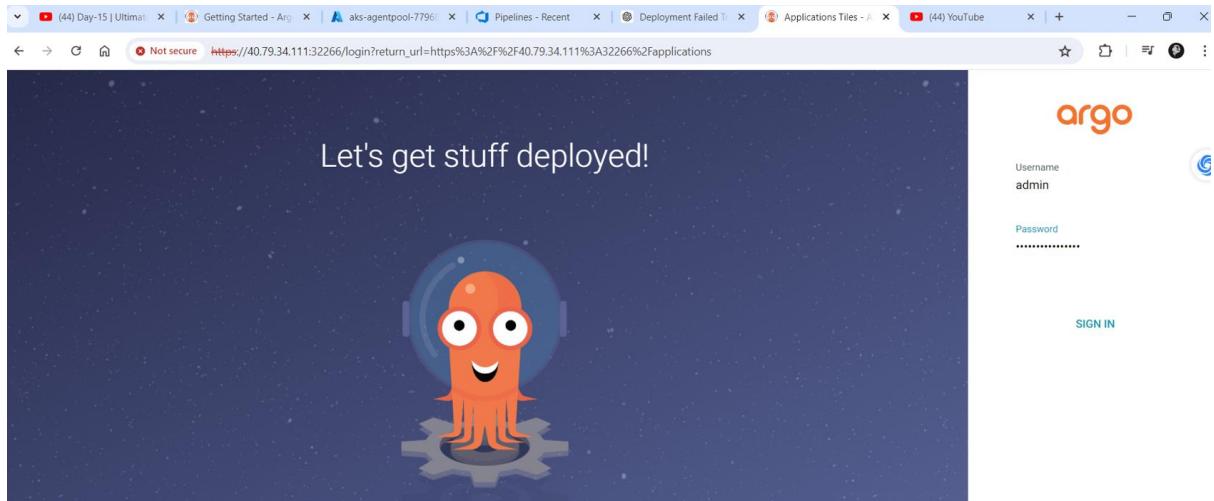
```

argocd-repo-server	ClusterIP	10.0.48.104	<none>	8081/TCP,8084/TCP	12m
argocd-server	NodePort	10.0.168.167	<none>	80:31848/TCP,443:32081/TCP	12m

- 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>



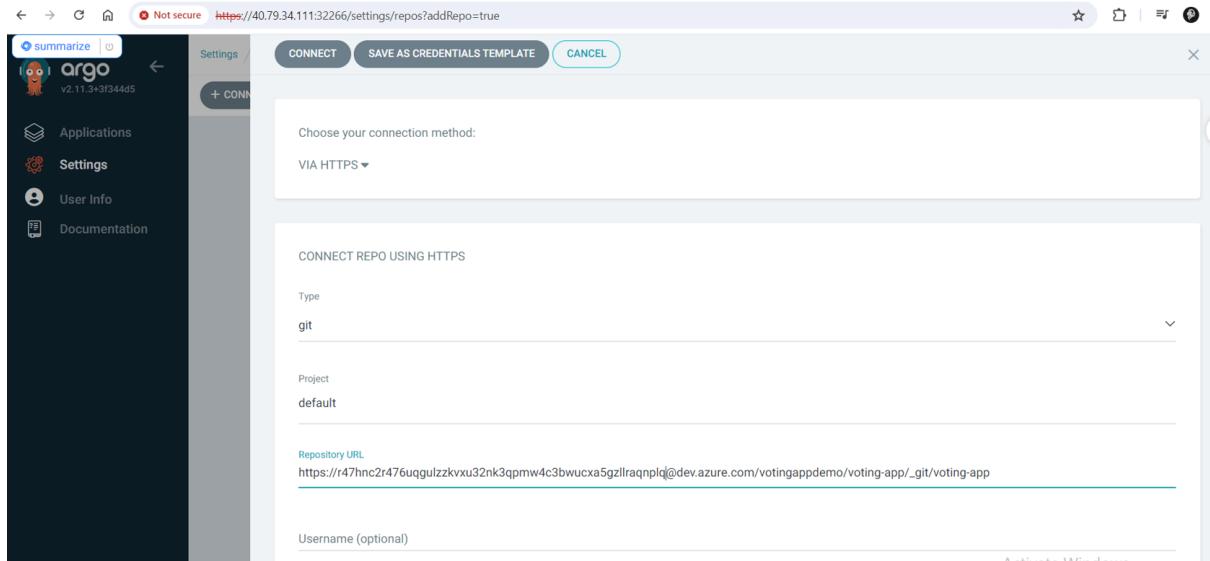
- Get the password from the secrets

```

PS C:\Users\ASUS> kubectl get pods -n argocd
NAME                               READY   STATUS    RESTARTS   AGE
argocd-application-controller-0     1/1     Running   0          38s
argocd-applicationset-controller-65bb5ff89-5z8lg   1/1     Running   0          42s
argocd-dex-server-6f898cbd9-tstrz   1/1     Running   0          42s
argocd-notifications-controller-64bc7c9f7-5r5sl   1/1     Running   0          41s
argocd-redis-5df55f45b7-d7r7l    1/1     Running   0          40s
argocd-repo-server-74d5f58dc5-pswtt   1/1     Running   0          40s
argocd-server-5b86767ddb-pv62h    1/1     Running   0          39s
PS C:\Users\ASUS> kubectl get secrets -n argocd
NAME           TYPE      DATA   AGE
argocd-initial-admin-secret   Opaque    1      53s
argocd-notifications-secret  Opaque    0      76s
argocd-redis     Opaque    1      59s
argocd-secret    Opaque    5      76s
PS C:\Users\ASUS> kubectl edit secret argocd-initial-admin-secret -n argocd

```

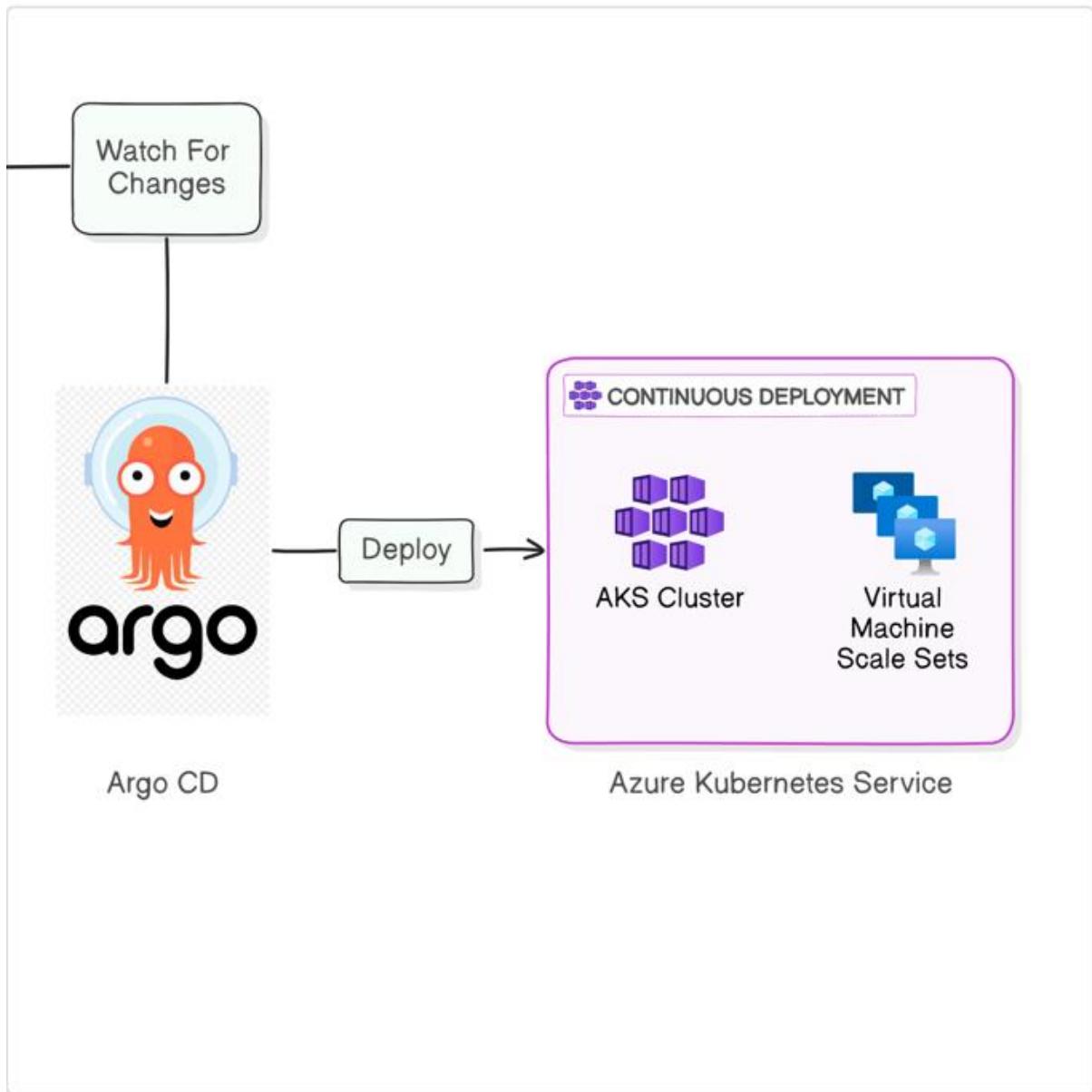
- Connect to GIT Repository:** Configure ArgoCD to connect to your Git repository where Kubernetes manifests are stored. This enables ArgoCD to access, monitor, and synchronize your application's deployment configurations from the repository.
- Create ArgoCD Applications:** Define ArgoCD applications 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.



This screenshot shows the 'REPOS' tab in the Argo UI. It lists a single repository entry: 'git' with 'NAME' 'git', 'REPOSITORY' 'https://r47hnc2r476uqgulzzkvxu32nk3qpmw4c3bwuxxa5gzllraqnplq@dev.azure.com/votingappdemo/voting-app/\_git/voting-app', and 'CONNECTION STATUS' 'Successful'. There are 'CONNECT REPO' and 'REFRESH LIST' buttons at the top of the table.

This screenshot shows the 'CREATE' dialog for a new application. The 'Application Name' is 'votingapp' and the 'Project Name' is 'default'. Under 'SYNC POLICY', 'Automatic' is selected. There are checkboxes for 'PRUNE RESOURCES' and 'SELF HEAL'. An 'EDIT AS YAML' button is visible in the top right corner.

This screenshot shows the 'Applications' tab. It lists one application named 'demo' with the following details: Project: default, Labels: Healthy Synced, Status: Healthy Synced, Repository: https://r47hnc2r476uqgulzzkvxu32nk3qpmw4c3bwuxxa5gzllraqnplq@dev.azure.com/votingappdemo/voting-app/\_git/voting-app, Target R...: HEAD, Path: k8s-specifications, Destination: in-cluster, Namespace: votingapp, Created ...: 07/07/2024 12:22:43, Last Sync: 07/07/2024 12:22:44. There are 'SYNC', 'REFRESH', and 'DELETE' buttons at the bottom of the card.



## 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.

The screenshot shows a file tree on the left and a code editor on the right.

**File Tree:**

- result-deployment.yaml
- result-service.yaml
- vote-deployment.yaml
- vote-service.yaml
- worker-deployment.yaml
- > result
- > Script
- > seed-data
- < vote
  - > static
  - > templates
- PY app.py
- Dockerfile
- requirements.txt

**Code Editor (app.py):**

```

1 from flask import Flask, render_template, request, make_response, g
2 from redis import Redis
3 import os
4 import socket
5 import random
6 import json
7 import logging
8
9 option_a = os.getenv('OPTION_A', "Android")
10 option_b = os.getenv('OPTION_B', "IOS")
11 hostname = socket.gethostname()
12
13 app = Flask(__name__)
14
15 gunicorn_error_logger = logging.getLogger('gunicorn.error')
16 app.logger.handlers.extend(gunicorn_error_logger.handlers)
17 app.logger.setLevel(logging.INFO)
18
19 def get_redis():
20     if not hasattr(g, 'redis'):
21         g.redis = Redis(host="redis", db=0, socket_timeout=5)
22     return g.redis
23
24 @app.route("/", methods=['POST', 'GET'])
25 def hello():
26     ...
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47

```

## 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.

The screenshot shows the Azure DevOps CI pipeline interface with two stages: Build and push.

**Build Stage:**

- Initialize job: 1s
- Checkout voting-app@...: 7s
- Build an image: 56s
- Post-job: Checkout voti...: 1s
- Finalize Job: <1s

**push Stage:**

- Initialize job: 1s
- Checkout voting-app@...: 6s
- Push an image to cont...: 13s
- Post-job: Checkout voti...: 1s
- Finalize Job: <1s

**Job Log (push stage):**

```

1 Pool: azureagent
2 Agent: azureagent
3 Started: Just now
4 Duration: 23s
5
6 ▶ Job preparation parameters
41 Job live console data:
42 Starting: push
43 Async Command Start: DetectDockerContainer
44 Async Command End: DetectDockerContainer
45 Async Command Start: DetectDockerContainer
46 Async Command End: DetectDockerContainer
47 Finishing: push

```

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.

```

#!/bin/bash
set -x
# Set the repository URL
REPO_URL="https://r47hnc2r476uogulzzkvxu32nk3qpmw4c3bwuca5gzllraqnplq@dev.azure.com/votingappdemo/voting-app/_git/voting-app"
# Clone the git repository into the /tmp directory
git clone "$REPO_URL" /tmp/temp_repo
# Navigate into the cloned repository directory
cd /tmp/temp_repo
# Make changes to the Kubernetes manifest file(s)
# For example, let's say you want to change the image tag in a deployment.yaml file
sed -i "s|image:.*|image: narasimhaazurecid/$2:$3|g" k8s-specifications/$1-deployment.yaml
# Add the modified files
git add .
# Commit the changes
git commit -m "Update Kubernetes manifest"
# Push the changes back to the repository
git push
# Cleanup: remove the temporary directory

```

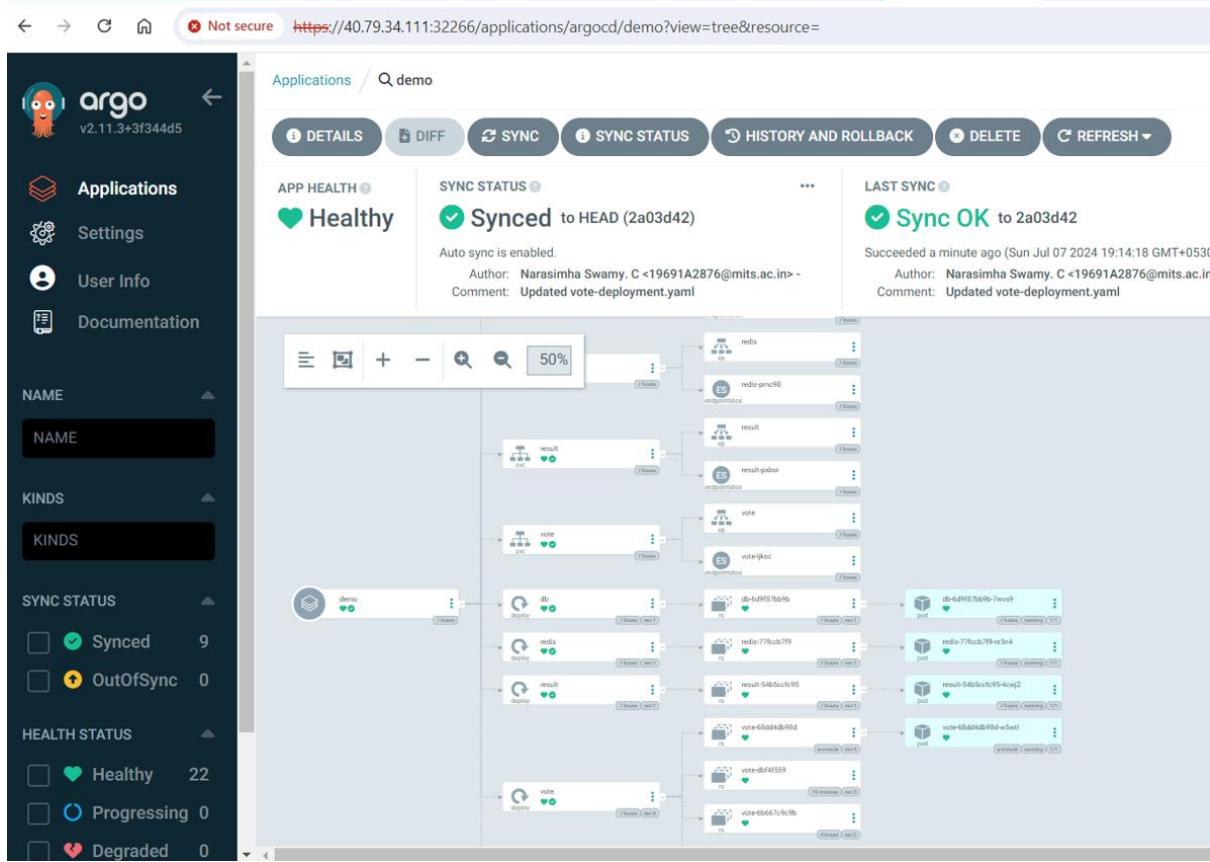
```

apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: vote
  name: vote
spec:
  replicas: 1
  selector:
    matchLabels:
      app: vote
  template:
    metadata:
      labels:
        app: vote
    spec:
      containers:
        - image: narasimhaazurecid.azurecr.io/votingapp:16
          name: vote
          ports:
            - containerPort: 80
              name: vote
          imagePullSecrets:
            - name: acr-secret

```

## 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.