Week-14 | DevOps Project from Absolute Zero | Part I | Project Repo Setup + CI Pipelines Implementation in Azure DevOps

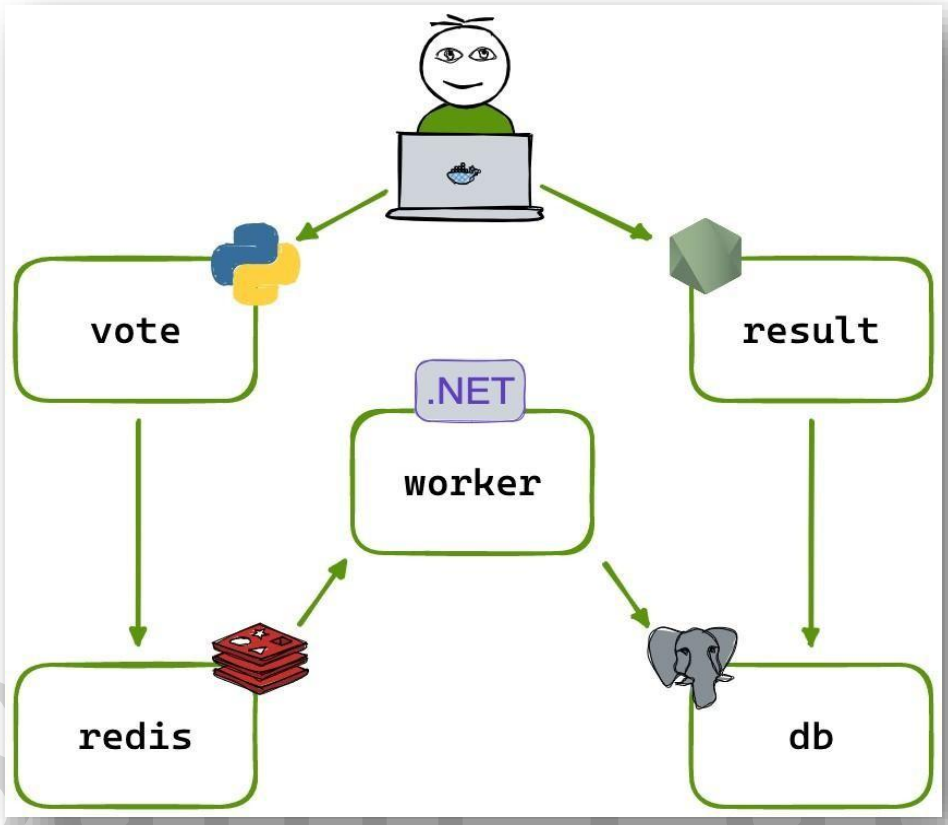


Table of Contents

Overview

Continuous Integration (CI) is a fundamental DevOps practice that ensures new code changes are integrated, built, and tested automatically. This document provides a structured approach to setting up a **DevOps project repository in Azure DevOps**, integrating version control, and implementing a **CI pipeline** to automate build and test processes. By following this guide, you will:

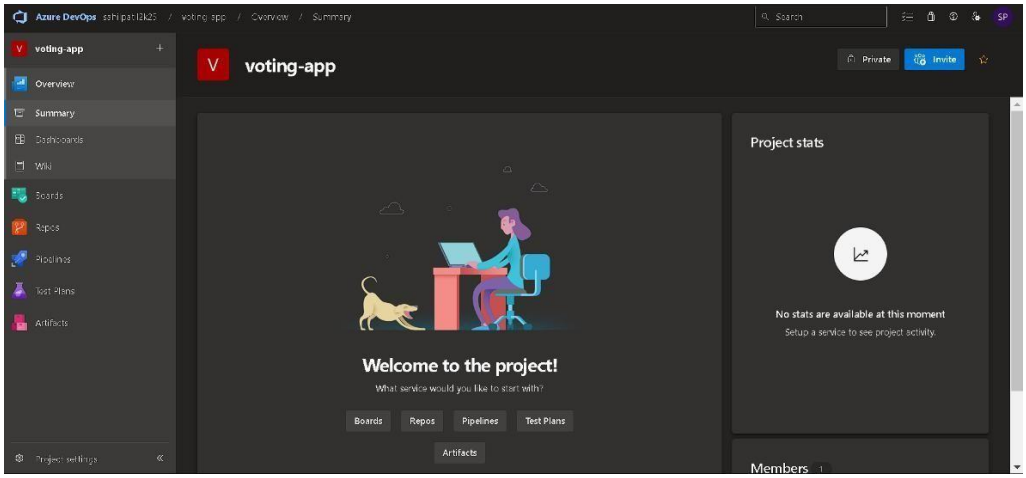
- ☑ Create a **new Azure DevOps project**.
- ☑ Set up a **Git repository** in Azure Repos.
- ☑ Implement a **CI pipeline** to automate code integration and validation.
- ☑ Configure **branch policies** to enforce quality standards.
- ☑ Learn best practices for maintaining an efficient CI workflow.

Step 1: Create a Project in Azure DevOps

Azure DevOps provides a centralized platform for version control, CI/CD pipelines, and project tracking.

1.1 Accessing Azure DevOps

1. Go to Azure DevOps and sign in with your Microsoft account.
2. Click on **Create New Project**.



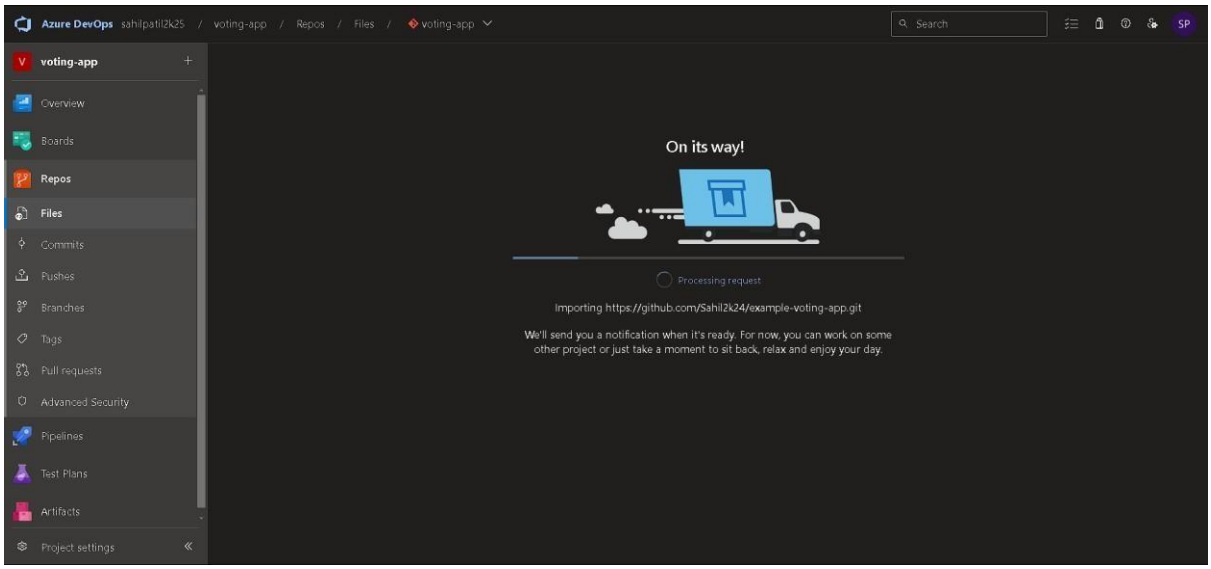
1.2 Project Configuration

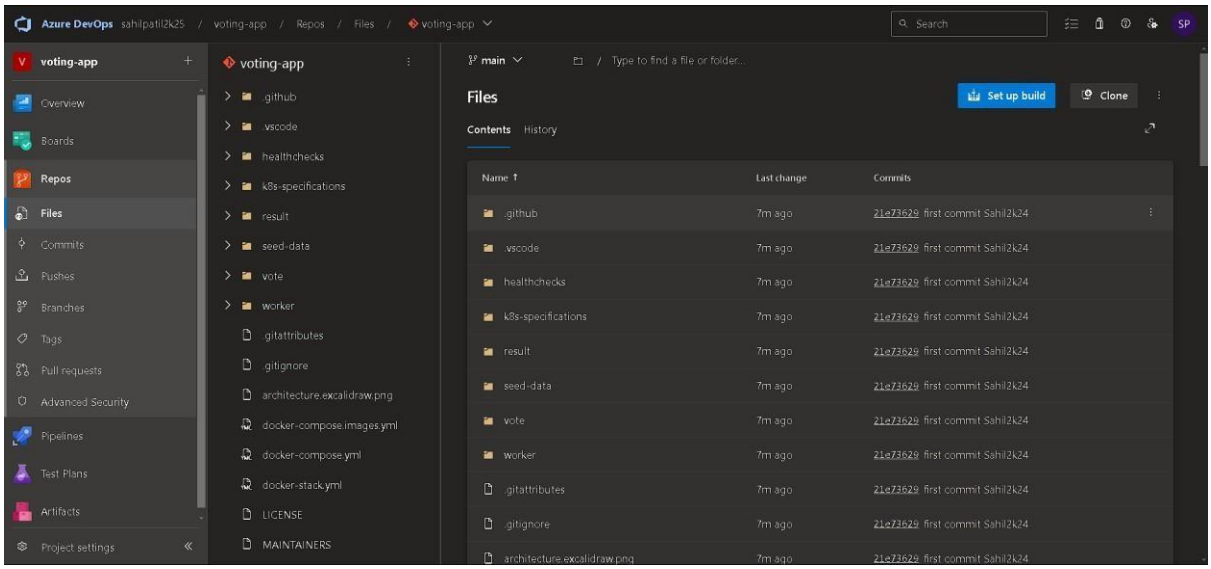
- **Project Name**: Choose a meaningful project name (e.g., DevOps-Project-Week-13).
- **Description**: Provide a brief description of the project.
- **Visibility**: Choose **Private** for security or **Public** if it's an open-source project.
- **Version Control System**: Select **Git** as the repository type.
- **Work Item Process**: Choose between **Agile, Scrum, or Kanban**, depending on your workflow.

Click **Create** to finalize the setup.

Step 2: Create a Git Repository in Azure Repos

Azure Repos provides Git-based version control to manage code and track changes.

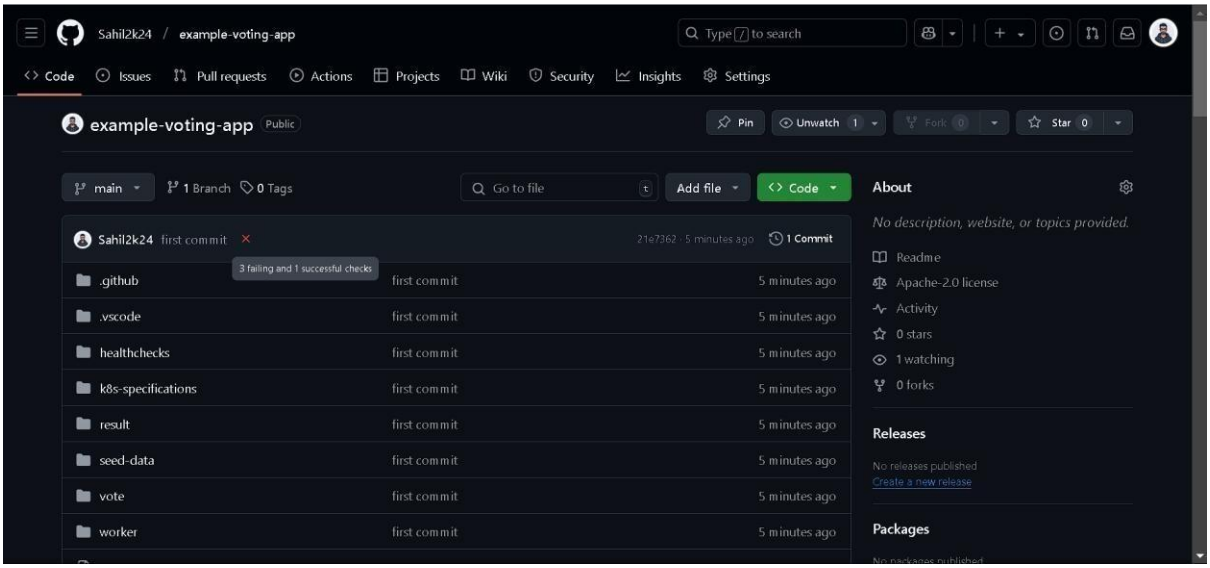




2.1 Initialize a New Repository

1. Navigate to **Repos** → **Files** in Azure DevOps.
2. Click on **Initialize Repository**.
3. Enter a repository name such as devops-project-repo.
4. Select a .gitignore template based on your technology stack (e.g., Node.js, Python, Java).
5. Optionally, add a **README.md** file for documentation.
6. Click **Create Repository**.

2.2 Clone the Repository Locally



Developers need to clone the repository to work on code locally. Use the following command to clone the repository: git clone https://dev.azure.com/<organization>/<project>/_git/devopsproject-repo cd devops-project-repo

After making changes, use these commands to commit and push updates:

git add . git commit -m "Initial commit" git push origin main

---

Step 3: Set Up a CI Pipeline in Azure DevOps

3.1 Understanding Continuous Integration (CI)

CI is the practice of automatically integrating new code changes into a shared repository. Every commit triggers an automated process that:

- ☑ **Builds the application** to detect errors early.
- ☑ **Runs tests** to ensure functionality remains intact.
- ☑ **Generates build artifacts** for future deployments.

3.2 Prerequisites for Setting Up CI Pipeline

Before creating a CI pipeline, ensure the following:

- A **Git repository** is available in Azure Repos.
- The project has a **build script** (if applicable).

- Necessary **dependencies** are listed in a configuration file (e.g., package.json for Node.js).

3.3 Configuring an Azure DevOps Pipeline 1.

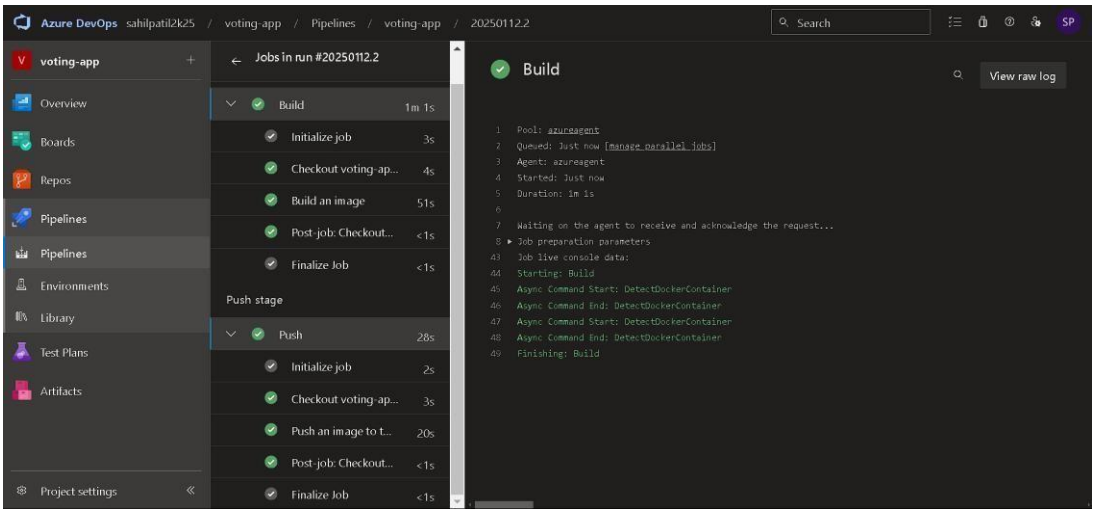
Navigate to **Pipelines** in Azure DevOps.

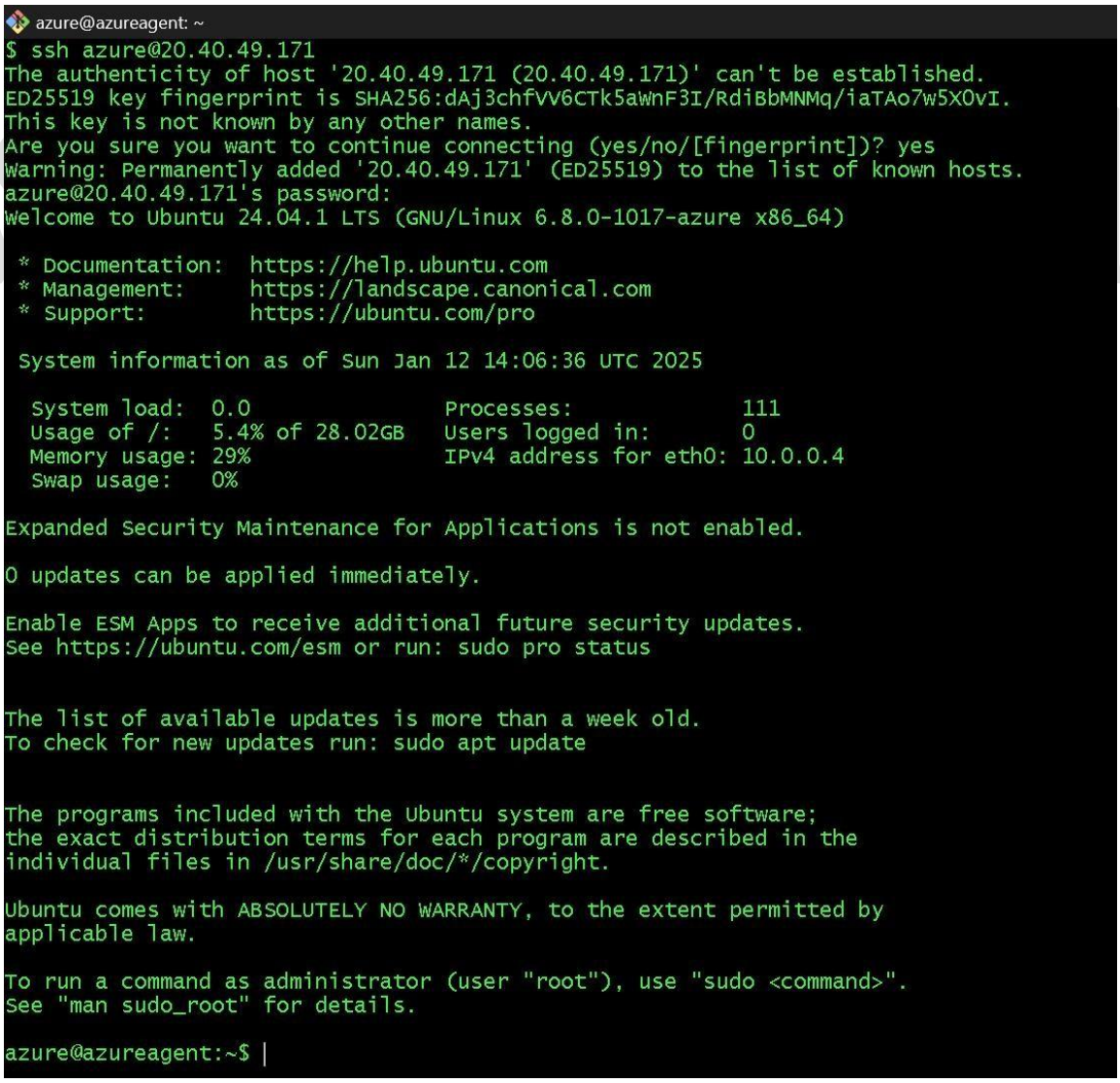
2. Click on **New Pipeline**.
3. Choose **Azure Repos Git** as the source.
4. Select the repository created in Step 2.
5. Choose **Starter Pipeline** or configure manually.

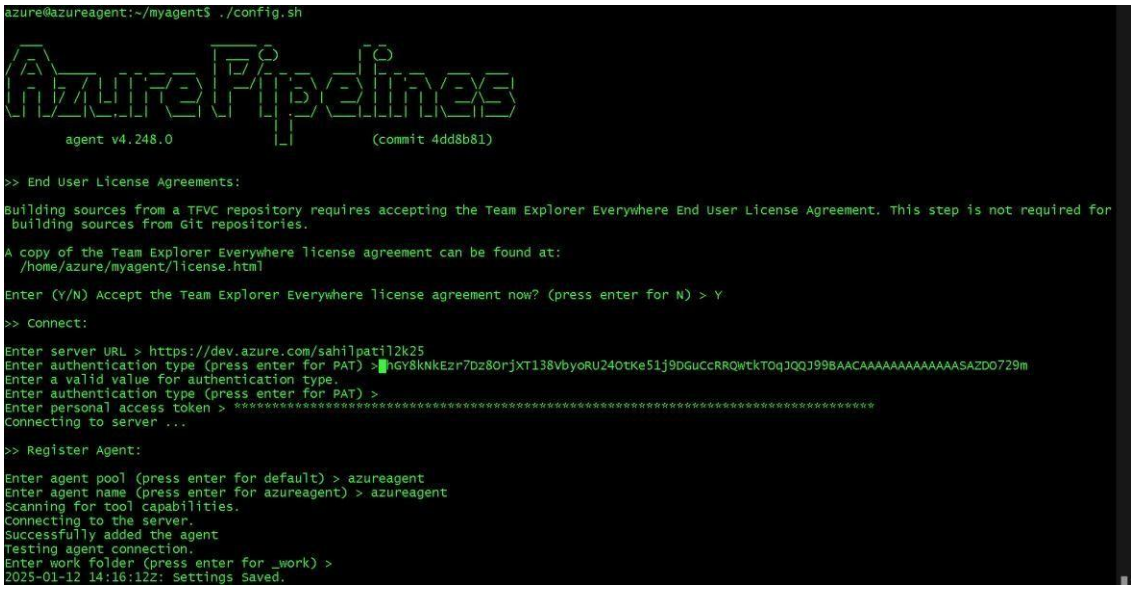
After setting up the pipeline, define the **build and test steps** based on your project's technology stack.

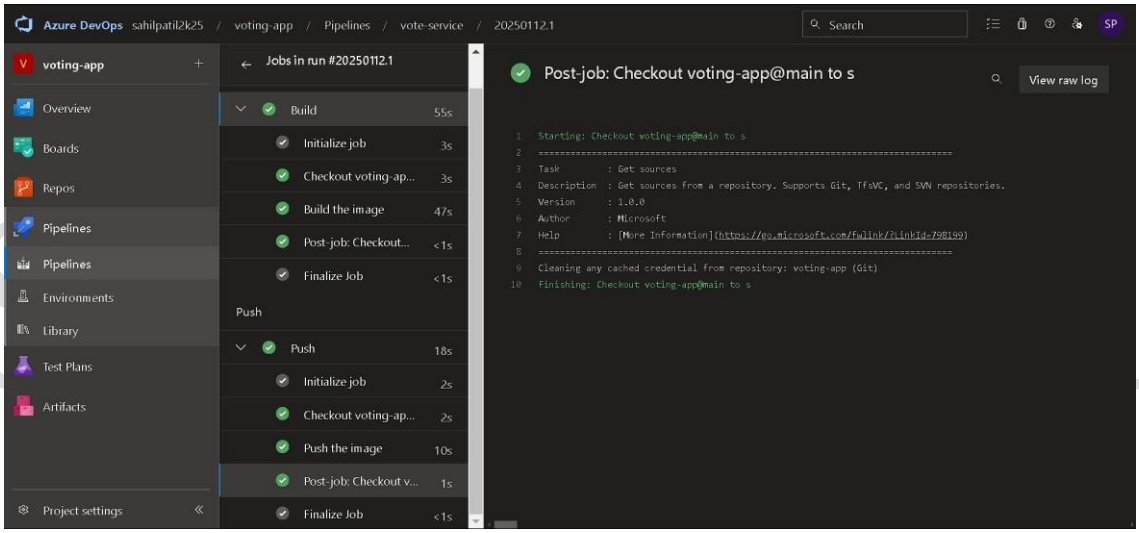
```yaml
# Docker
# Build and push an image to Azure Container Registry
# Reference: https://docs.microsoft.com/azure/devops/pipelines/languages/docker

trigger:
  paths:
    include:
      - vote/*

resources:
  - repo: self

variables:
  dockerRegistryServiceConnection: '****************'  # Azure service connection
ID
  imageRepository: 'votingapp'
  containerRegistry: 'votingapp2025.azurecr.io'
  dockerfilePath: '$(Build.SourcesDirectory)/vote/Dockerfile'
  tag: '$(Build.BuildId)'

pool:
  name: 'azureagent'

stages:
  - stage: Build
    displayName: Build
    jobs:
      - job: Build
        displayName: Build
        steps:
          - task: Docker@2
            displayName: Build the image
            inputs:
              containerRegistry: '$(dockerRegistryServiceConnection)'
              repository: '$(imageRepository)'
              command: 'build'
              Dockerfile: 'vote/Dockerfile'
              tags: '$(tag)'

  - stage: Push
    displayName: Push
    jobs:
      - job: Push
        displayName: Push
        steps:
          - task: Docker@2
            displayName: Push the image
            inputs:
              containerRegistry: '$(dockerRegistryServiceConnection)'
              repository: '$(imageRepository)'
              command: 'push'
              tags: '$(tag)'

  - stage: Update
    displayName: Update
    jobs:
      - job: Update
        displayName: Update
        steps:
          - script: |
              sudo apt-get install dos2unix -y
              dos2unix scripts/updateK8sManifests.sh
            displayName: 'Install packages and convert script'

          - task: ShellScript@2
            inputs:
              scriptPath: 'scripts/updateK8sManifests.sh'
              args: 'vote $(imageRepository) $(tag)'
```

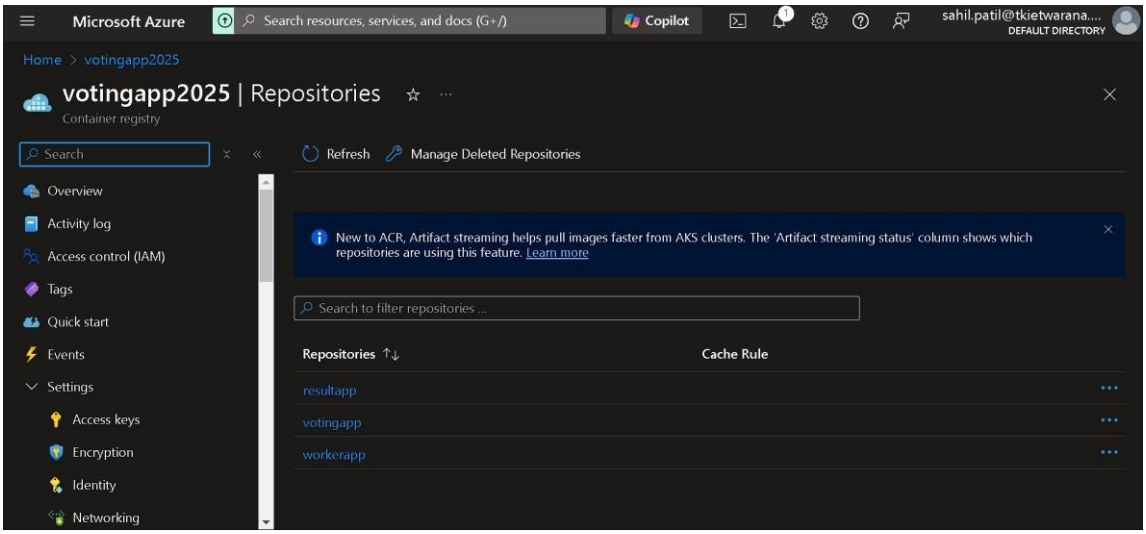
Similarly, you can create pipelines for the **worker** and **result** services by following the same structure. Just update the imageRepository, Dockerfile path, and relevant service details accordingly.

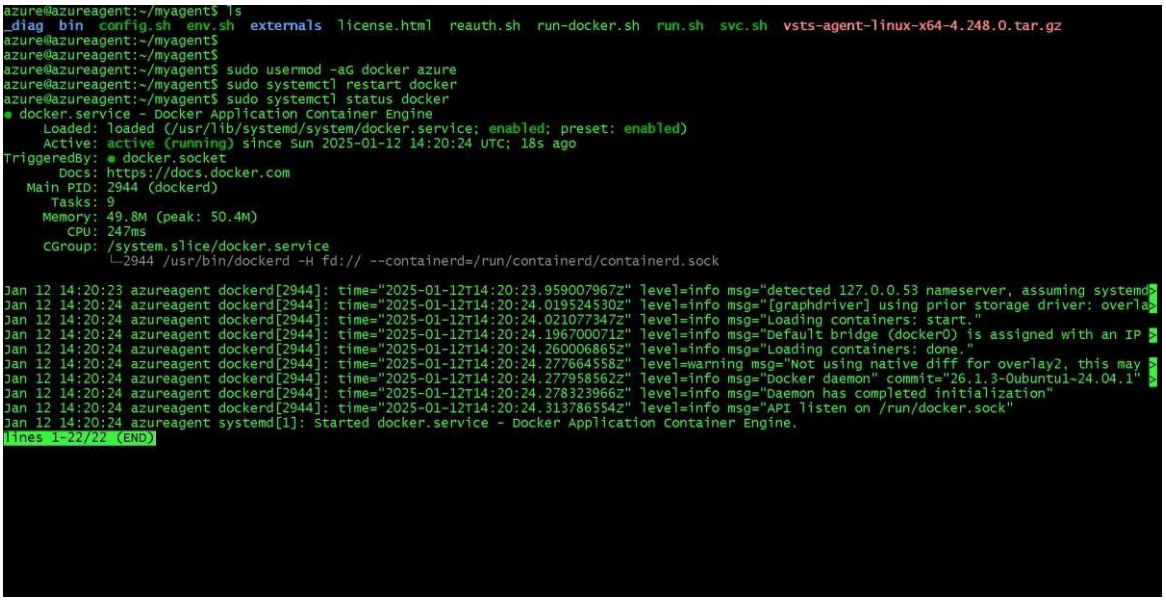


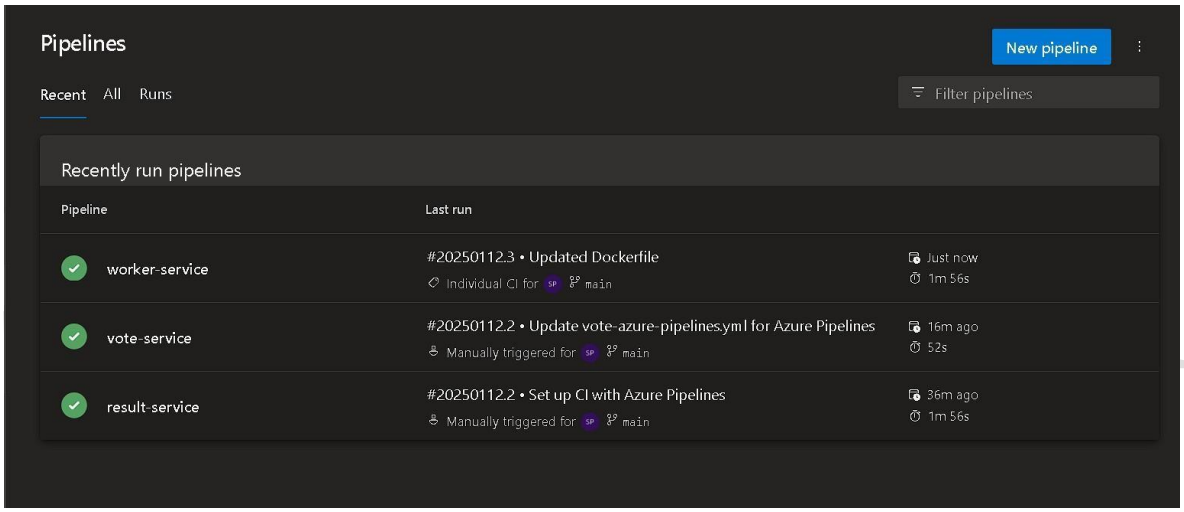



azure@azureagent:~/myagent$ ./config.sh
agent v4.248.0
(commit 4dd8b81)
>> End User License Agreements:
Building sources from a TFVC repository requires accepting the Team Explorer Everywhere End User License Agreement. This step is not required for building sources from Git repositories.
A copy of the Team Explorer Everywhere license agreement can be found at:
/home/azure/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/sahilpatil2k25
Enter authentication type (press enter for PAT) >
Enter a valid value for authentication type.
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) > azureagent
Scanning for tool capabilities.
Connecting to the server.
Successfully added the agent
Testing agent connection.
Enter work folder (press enter for _work) >
2025-01-12 14:16:122: Settings Saved.


Azure DevOps
sahilpatil2k25
voting-app
Pipelines
vote-service
20250112.1
Search
voting-app
Overview
Boards
Repos
Pipelines
Pipelines
Environments
Library
Test Plans
Artifacts
Project settings
Jobs in run #20250112.1
Build 55s
Initialize job 3s
Checkout voting-ap... 3s
Build the image 47s
Post-job: Checkout... <1s
Finalize Job <1s
Push
Push 18s
Initialize job 2s
Checkout voting-ap... 2s
Push the image 10s
Post-job: Checkout v... 1s
Finalize Job <1s
Post-job: Checkout voting-app@main to s
View raw log
Starting: Checkout voting-app@main to s
Task : Get sources
Description : Get sources from a repository. Supports Git, TfsVC, and SVN repositories.
Version : 1.0.0
Author : Microsoft
Help : [More Information]
Cleaning any cached credential from repository: voting-app (Git)
Finishing: Checkout voting-app@main to s


Microsoft Azure
Search resources, services, and docs (G+/)
Copilot
sahil.patil@tkietwarana....
DEFAULT DIRECTORY
Home > votingapp2025
votingapp2025 | Repositories
Container registry
Search
Refresh
Manage Deleted Repositories
Overview
Activity log
Access control (IAM)
Tags
Quick start
Events
Settings
Access keys
Encryption
Identity
Networking
New to ACR, Artifact streaming helps pull images faster from AKS clusters. The 'Artifact streaming status' column shows which repositories are using this feature. Learn more
Search to filter repositories ...
Repositories
Cache Rule
resultapp
votingapp
workerapp





Step 4: Enforcing Branch Policies for CI

To maintain code quality, enforce branch policies that require passing CI checks before merging changes.

4.1 Configure Branch Protection Rules

1. Go to **Repos** → **Branches** in Azure DevOps.
2. Select the main branch and click on **Branch Policies**.
3. Enable the following settings:
    - **Require a build to be successful before merging**.
    - **Require minimum number of reviewers**.
    - **Limit who can push directly to the branch**

4. Save the changes.

With these policies, only tested and approved code can be merged into the main branch.

Step 5: Setting Up Build Artifacts

Build artifacts are packaged outputs generated during the CI process. These can be stored for deployment in the later stages.

5.1 Configure Build Artifact Storage

1. In Azure Pipelines, add a task to **store artifacts**.
2. Define an output directory where build files are stored.
3. Enable artifact retention policies for efficient storage management.

Step 6: Monitoring and Debugging CI Pipelines

After running the CI pipeline, check for errors and optimize performance.

6.1 Tracking Pipeline Runs

1.Navigate to **Pipelines → Runs** to monitor execution.

2.Check logs for failed steps and error messages. **6.2**

Troubleshooting Common CI Issues

If the pipeline fails:

❌ **Missing dependencies** → Ensure all required packages are listed in a configuration file.

❌ **Incorrect YAML syntax** → Validate using an online YAML validator.

❌ **Pipeline not triggering** → Check branch settings in azure-pipelines.yml.

❌ **Test failures** → Run tests locally to identify potential issues.

Step 7: Best Practices for Implementing CI Pipelines

To maintain efficiency in CI/CD workflows, follow these best practices:

☑ **Optimize pipeline execution** → Only run necessary steps per commit.

☑ **Enable caching** → Reduce build time by caching dependencies.

☑ **Secure secrets and credentials** → Use Azure Key Vault to store sensitive data.

☑ **Implement parallel processing** → Speed up execution with concurrent jobs.

☑ **Automate testing** → Ensure every build includes unit and integration tests.

☑ **Use notifications** → Set up alerts for failed builds to notify developers.

Conclusion:

By following this guide, we have successfully:

☑ Created an **Azure DevOps project** and **Git repository**.

☑ Set up an **automated CI pipeline** to build and test code.

☑ Configured **branch protection policies** to enforce quality standards.

☑ Implemented **build artifact storage** for future deployments.

This forms the foundation for the **Continuous Deployment (CD)** phase, where applications are deployed to production environments automatically.
