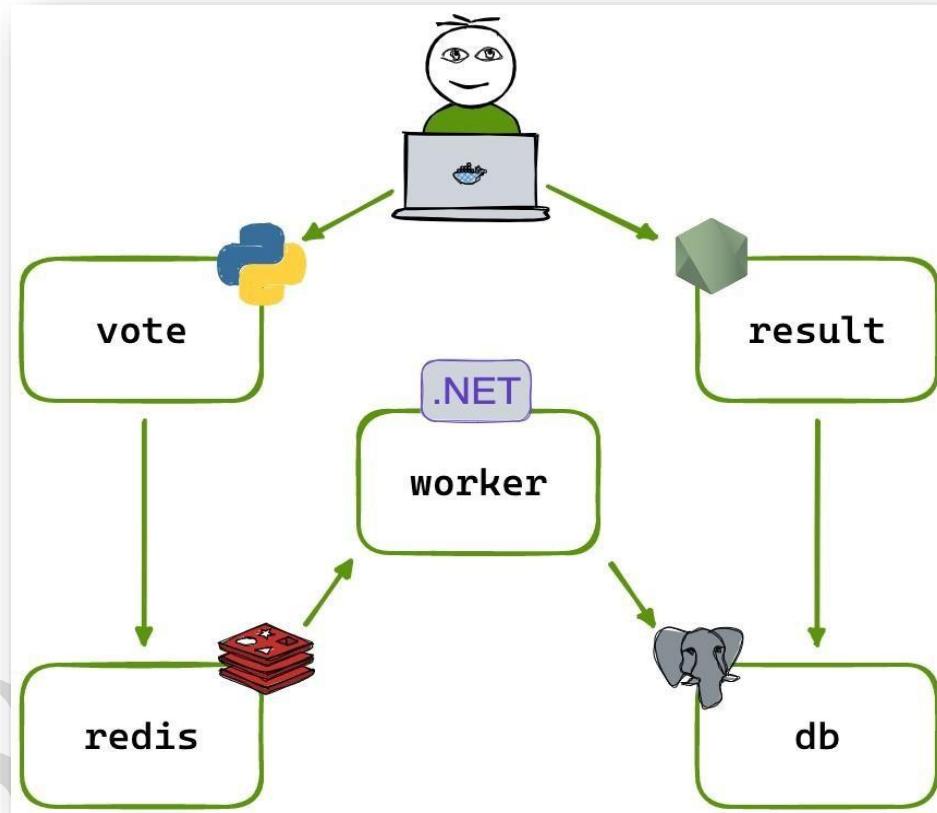


## Week-14 | DevOps Project from Absolute Zero | Part I | Project Repo Setup + CI Pipelines Implementation in Azure DevOps



---

### Table of Contents

- [Overview](#)
- [Step 1: Create a Project in Azure DevOps](#)
- [Step 2: Create a Git Repository in Azure Repos](#)
- [Step 3: Set Up a CI Pipeline in Azure DevOps](#)
  - [3.1 Understanding Continuous Integration \(CI\)](#)
  - [3.2 Prerequisites for Setting Up CI Pipeline](#)
  - [3.3 Configuring an Azure DevOps Pipeline](#)
- [Step 4: Enforcing Branch Policies for CI](#)
- [Step 5: Setting Up Build Artifacts](#)
- [Step 6: Monitoring and Debugging CI Pipelines](#)

- [Step 7: Best Practices for Implementing CI Pipelines](#)
  - [Conclusion](#)
- 

## 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.
- 

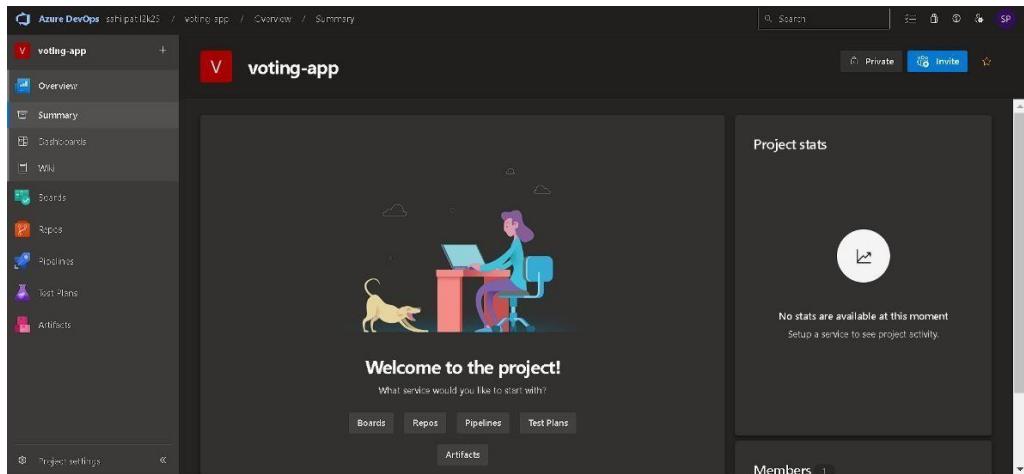
## Step 1: Create a Project in Azure DevOps



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.

The screenshot shows the Azure DevOps interface for a project named 'voting-app'. On the left, a sidebar lists various project management and development tools: Overview, Boards, Repos, Files, Commits, Pushes, Branches, Tags, Pull requests, Advanced Security, Pipelines, Test Plans, Artifacts, and Project settings. The 'Repos' option is selected. In the main area, a progress message 'On its way!' is displayed above a graphic of a truck carrying a trash bin. Below the graphic, it says 'Processing request' and 'Importing https://github.com/Sahil2k24/example-voting-app.git'. A note states: 'We'll send you a notification when it's ready. For now, you can work on some other project or just take a moment to sit back, relax and enjoy your day.'

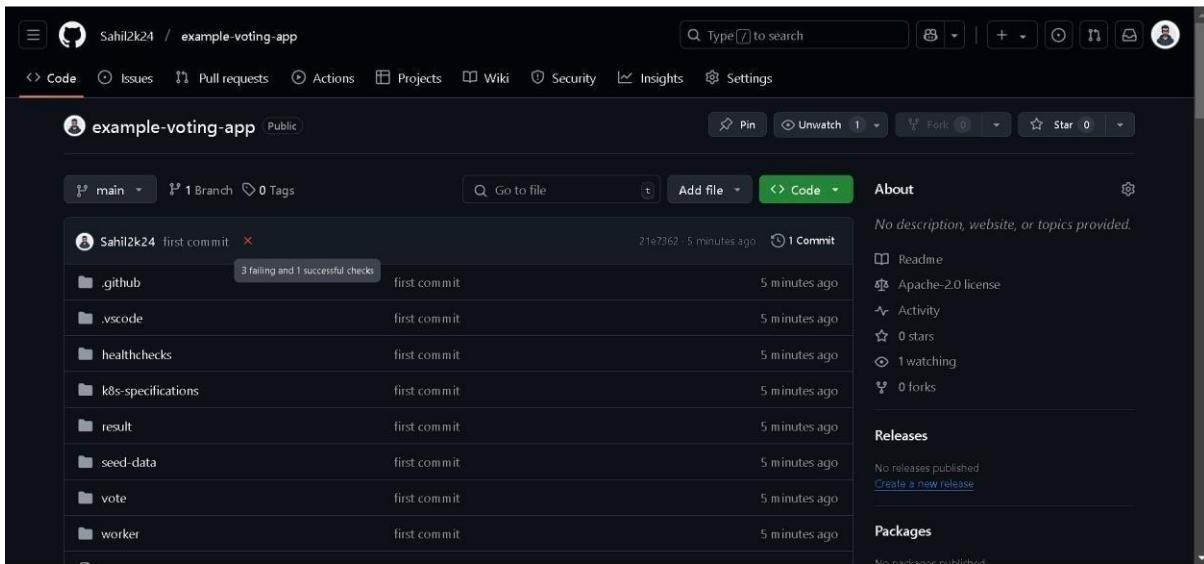
  

The second screenshot shows the same 'voting-app' repository after the import process has completed. The sidebar remains the same. In the main area, the 'Files' tab is selected, showing a list of files under the 'main' branch. The files listed include: .github, .vscode, .healthchecks, .k8s-specifications, .result, .seed-data, .vote, .worker, .gitattributes, .gitignore, architecture-excalidraw.png, docker-compose-images.yml, docker-compose.yml, docker-stack.yml, LICENSE, and MAINTAINERS. Each file entry includes its name, last change time (7m ago), and commit history, which all point to the first commit by 'Sahil2k24'.

## 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



The screenshot shows a GitHub repository page for 'example-voting-app'. The main branch is 'main' with 1 commit. The commit details are as follows:

File	Message	Time Ago
.github	first commit	5 minutes ago
.vscode	first commit	5 minutes ago
healthchecks	first commit	5 minutes ago
k8s-specifications	first commit	5 minutes ago
result	first commit	5 minutes ago
seed-data	first commit	5 minutes ago
vote	first commit	5 minutes ago
worker	first commit	5 minutes ago

The repository has no description, website, or topics provided. It includes a 'Readme' file and an 'Apache-2.0 license'. There is 1 watching and 0 forks.

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.

# Sahil Patil

```
# 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.

```

azur@azureagent:~ 
$ ssh azur@20.40.49.171
The authenticity of host '20.40.49.171 (20.40.49.171)' can't be established.
ED25519 key fingerprint is SHA256:dAj3chfVV6CTk5awnF3I/RdiBbMNMQ/iaTAo7w5X0VI.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '20.40.49.171' (ED25519) to the list of known hosts.
azur@20.40.49.171's password:
Welcome to Ubuntu 24.04.1 LTS (GNU/Linux 6.8.0-1017-azure x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/pro

System information as of Sun Jan 12 14:06:36 UTC 2025

System Load: 0.0          Processes:           111
Usage of /: 5.4% of 28.02GB   Users logged in:      0
Memory usage: 29%          IPv4 address for eth0: 10.0.0.4
Swap usage: 0%              

Expanded Security Maintenance for Applications is not enabled.

0 updates can be applied immediately.

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status

The list of available updates is more than a week old.
To check for new updates run: sudo apt update

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/*copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

azur@azureagent:~$ |

```

```

azure@azureagent:~/myagent$ ./config.sh
[?] Azure [?] Explorer Everywhere
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) > 1ngY8kNkEzr7dz80rjXT138vbyoRU24otKe51j9DGUCCRRQWtkT0qJQQJ99BAACAAAAAAAAAAASAZDO729m
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 14:16:12Z: Settings Saved,

```

The screenshot shows the Azure DevOps Pipelines interface. On the left, the project navigation bar includes 'Overview', 'Boards', 'Repos', 'Pipelines' (which is selected), 'Environments', 'Library', 'Test Plans', and 'Artifacts'. Under 'Pipelines', there is a list of runs: 'voting-app' (run #20250112.1). Clicking on this run reveals the detailed pipeline steps:

- Build**: Duration 55s, status green, steps: Initialize job (3s), Checkout voting-ap... (3s), Build the image (47s), Post-job: Checkout... (<1s), Finalize Job (<1s).
- Push**: Duration 18s, status green, steps: Push (18s), Initialize job (2s), Checkout voting-ap... (2s), Push the image (10s), Post-job: Checkout... (<1s), Finalize Job (<1s).

A log window titled 'Post-job: Checkout voting-app@main to s' displays the command output:

```

1 Starting: Checkout voting-app@main to s
2 -----
3 Task : Get sources
4 Description : Get sources from a repository. Supports Git, TfsVC, and SVN repositories.
5 Version : 1.0.0
6 Author : Microsoft
7 Help : [More Information](https://go.microsoft.com/fwlink/?LinkId=798199)
8 -----
9 Cleaning any cached credential from repository: voting-app (Git)
10 Finishing: Checkout voting-app@main to s

```

The screenshot shows the Microsoft Azure Container Registry (ACR) interface. The top navigation bar includes 'Home', 'votingapp2025', 'Search resources, services, and docs (G+)', 'Copilot', and user information 'sahil.patil@tkietwarana... DEFAULT DIRECTORY'. The main page for 'votingapp2025' shows the following details:

- Container registry**
- Repositories** (listing): resultapp, votingapp, workerapp.
- Cache Rule** (for each repository): three dots (...).
- Activity** (summary): 1 New to ACR, Artifact streaming helps pull images faster from AKS clusters. The 'Artifact streaming status' column shows which repositories are using this feature. A link to 'Learn more' is provided.
- Search**, **Refresh**, and **Manage Deleted Repositories** buttons.
- Settings** menu (with sub-options: Access keys, Encryption, Identity, Networking).

```

azure@azureagent:~/myagents$ ls
diag bin config.sh env.sh externals license.html reauth.sh run-docker.sh run.sh svc.sh vsts-agent-linux-x64-4.248.0.tar.gz
azure@azureagent:~/myagents
azure@azureagent:~/myagents$ azure@azureagent:~/myagents$ sudo usermod -aG docker azure
azure@azureagent:~/myagents$ sudo systemctl restart docker
azure@azureagent:~/myagents$ sudo systemctl status docker
● docker.service - Docker Application Container Engine
    Loaded: loaded (/usr/lib/systemd/system/docker.service; enabled; preset: enabled)
      Active: active (running) since sun 2025-01-12 14:20:24 UTC; 18s ago
TriggeredBy: ● docker.socket
  Docs: https://docs.docker.com
 Main PID: 2944 (dockerd)
   Tasks: 9
  Memory: 49.8M (peak: 50.4M)
    CPU: 247ms
   cGroup: /system.slice/docker.service
           └─2944 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock

jan 12 14:20:23 azureagent dockerd[2944]: time="2025-01-12T14:20:23.959007967Z" level=info msg="detected 127.0.0.53 nameserver, assuming systemd
jan 12 14:20:24 azureagent dockerd[2944]: time="2025-01-12T14:20:24.019524530Z" level=info msg="[graphdriver] using prior storage driver: overlay2"
jan 12 14:20:24 azureagent dockerd[2944]: time="2025-01-12T14:20:24.021077347Z" level=info msg="Loading containers: start."
jan 12 14:20:24 azureagent dockerd[2944]: time="2025-01-12T14:20:24.196700071Z" level=info msg="Default bridge (docker0) is assigned with an IP by
jan 12 14:20:24 azureagent dockerd[2944]: time="2025-01-12T14:20:24.260006865Z" level=info msg="Loading containers: done."
jan 12 14:20:24 azureagent dockerd[2944]: time="2025-01-12T14:20:24.277664558Z" level=warning msg="Not using native diff for overlay2, this may
jan 12 14:20:24 azureagent dockerd[2944]: time="2025-01-12T14:20:24.277958562Z" level=info msg="Docker daemon" commit="26.1.3-Ubuntu1~24.04.1"
jan 12 14:20:24 azureagent dockerd[2944]: time="2025-01-12T14:20:24.278323966Z" level=info msg="Daemon has completed initialization"
jan 12 14:20:24 azureagent dockerd[2944]: time="2025-01-12T14:20:24.313786554Z" level=info msg="API listen on /run/docker.sock"
jan 12 14:20:24 azureagent systemd[1]: Started docker.service - Docker Application Container Engine.
Lines 1--22/22 (END)

```

The screenshot shows the Azure DevOps Pipelines interface. At the top, there are buttons for 'New pipeline' and a filter input. Below that, a navigation bar with 'Recent', 'All', and 'Runs' tabs, and a 'Filter pipelines' search bar. The main area displays a table titled 'Recently run pipelines' with three rows:

Pipeline	Last run
worker-service	#20250112.3 • Updated Dockerfile ⌚ Individual CI for SP 🚧 main 🕒 Just now 🕒 1m 56s
vote-service	#20250112.2 • Update vote-azure-pipelines.yml for Azure Pipelines ⌚ Manually triggered for SP 🚧 main 🕒 16m ago 🕒 52s
result-service	#20250112.2 • Set up CI with Azure Pipelines ⌚ Manually triggered for SP 🚧 main 🕒 36m ago 🕒 1m 56s

## 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.

### 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.

---

# Sahil Patil