System Design & Scalability

Choose one of the following system design challenges:

1. Continuous Delivery Pipeline :

Design a CI/CD pipeline that includes: Automated builds and unit testing.

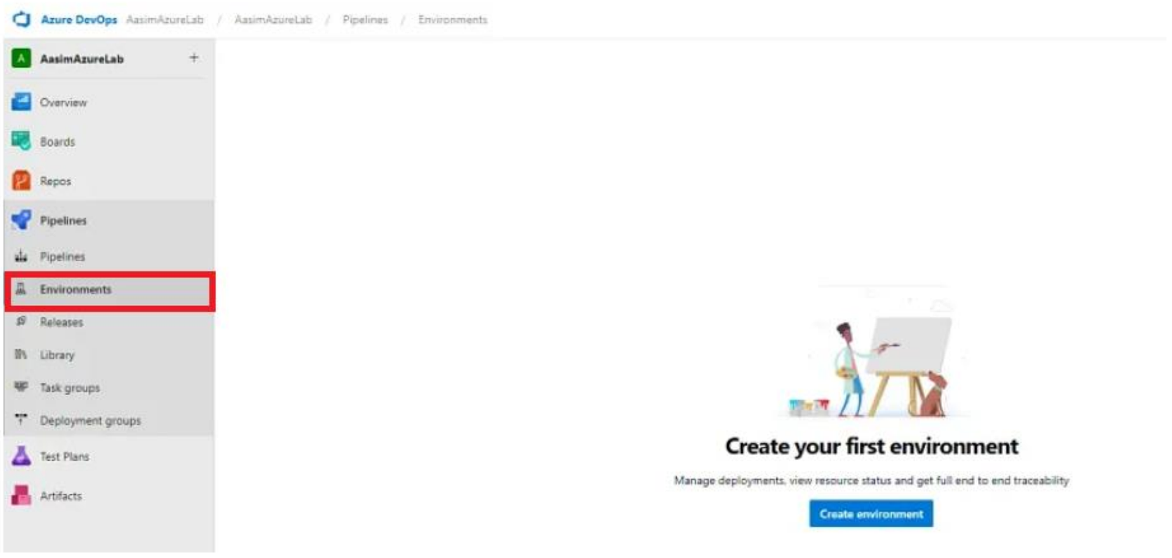
Security checks and quality gates.Multi-environment deployments (e.g., Dev, QA, Prod). Provide diagrams and documentation outlining the architecture.

Sign up with a Microsoft account –.
— Go to Azure DevOps

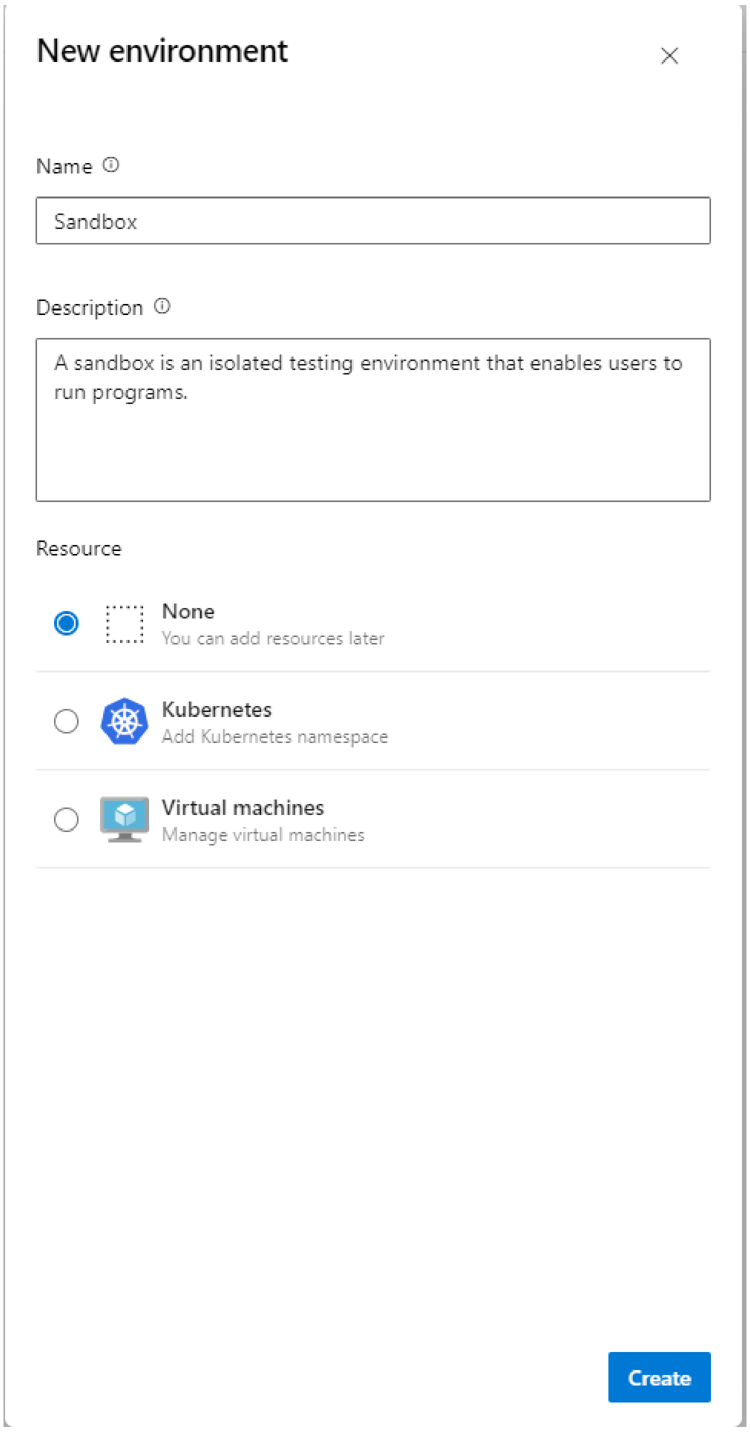
— Enter your login information and complete the sign-up procedure.

Environment — An environment is a set of resources that can be targeted with pipeline deployments. Dev, Test, QA, Staging, and Production are common environment names. An Azure DevOps environment is a logical destination for your pipeline's software deployment.

How to create Environment in Azure DevOps –
Step 1 — Go to the Pipeline Tab from the left navigation and click on it. After this, a new popup window will appear. That will ask you for the environment name.


Azure DevOps AasimAzureLab / AasimAzureLab / Pipelines / Environments
AasimAzureLab
Overview
Boards
Repos
Pipelines
Pipelines
Environments
Releases
Library
Task groups
Deployment groups
Test Plans
Artifacts
Create your first environment
Manage deployments, view resource status and get full end to end traceability
Create environment

Step 2– Enter the desired environment name, like Dev, QA, Stage, and Prod. Describe the description of the environment in the Description field. And then select the resource you want based on your requirements. Here, I do not want to use Kubernetes or a virtual machine, so i will select the first option, None. Later, if you would like to associate another resource.


New environment

Name ⓘ

Sandbox

Description ⓘ

A sandbox is an isolated testing environment that enables users to run programs.

Resource

[✓ None — You can add resources later]

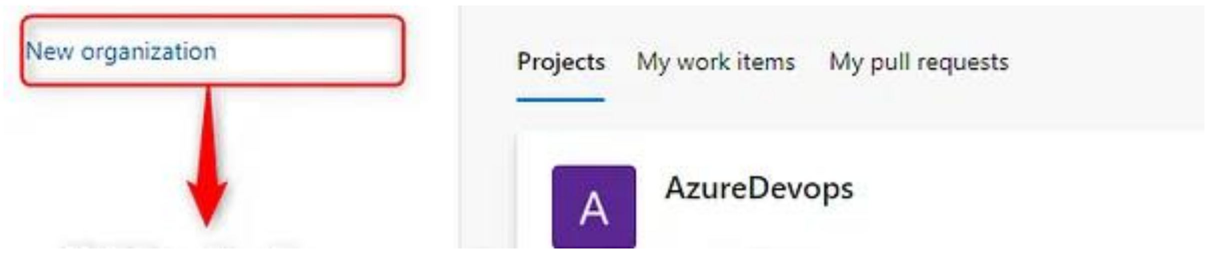
Kubernetes — Add Kubernetes namespace

Virtual machines — Manage virtual machines

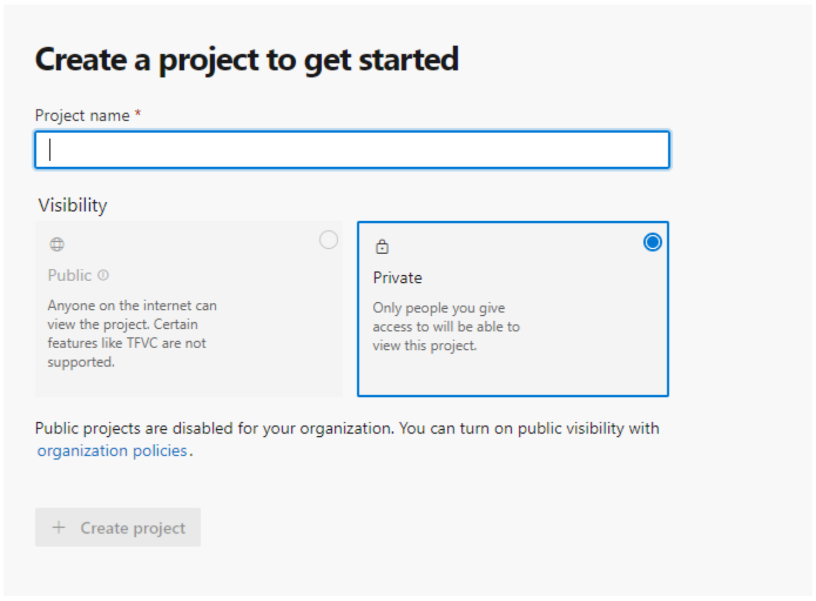
Organization — When you join an organization, you have access to Azure DevOps Services, where you can perform the following tasks: Use our cloud service to collaborate with others on application development. Plan and track your work, as well as code for bugs and issues. Install and configure continuous integration and deployment.

How to create new organization in Azure DevOps –

Step 1 — Click on the Azure DevOps logo and then click on the New Organization link that is placed in the left navigation. Once you click on it, it will ask for further details to provide the organization name, where you want to host, and a security-based captcha.



Step 2 — Here you can select the type of project, like public or private. Here I am with a private organization. After clicking on Create Project, it will ask you to accept privacy statements to continue.



Create a project to get started

Project name *

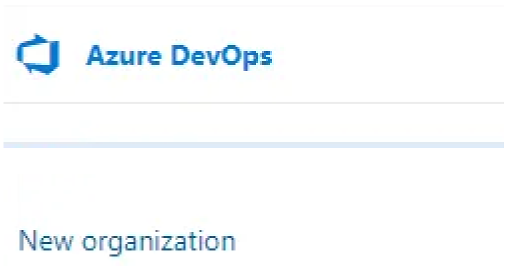
Visibility

Public
Anyone on the internet can view the project. Certain features like TFVC are not supported.

[✓ Private]
Only people you give access to will be able to view this project.

Public projects are disabled for your organization. You can turn on public visibility with organization policies.

Step 3 — After finishing setup, you can click on the Azure DevOps logo, or You can see all organizations in the left-side navigation. These organizations would be associated with the user based on permission.

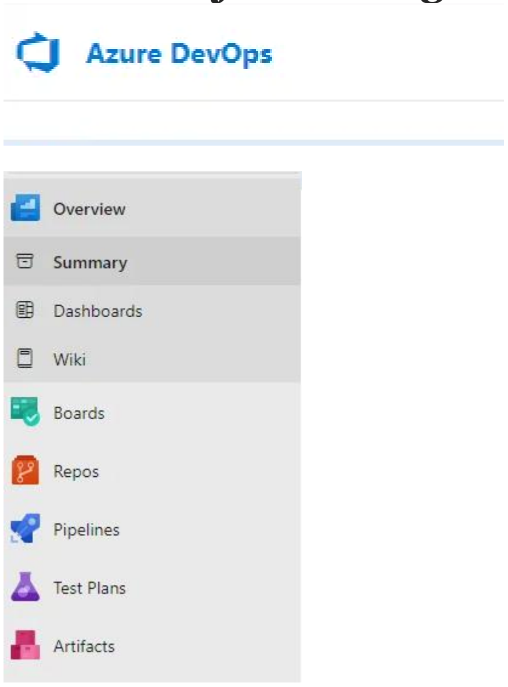


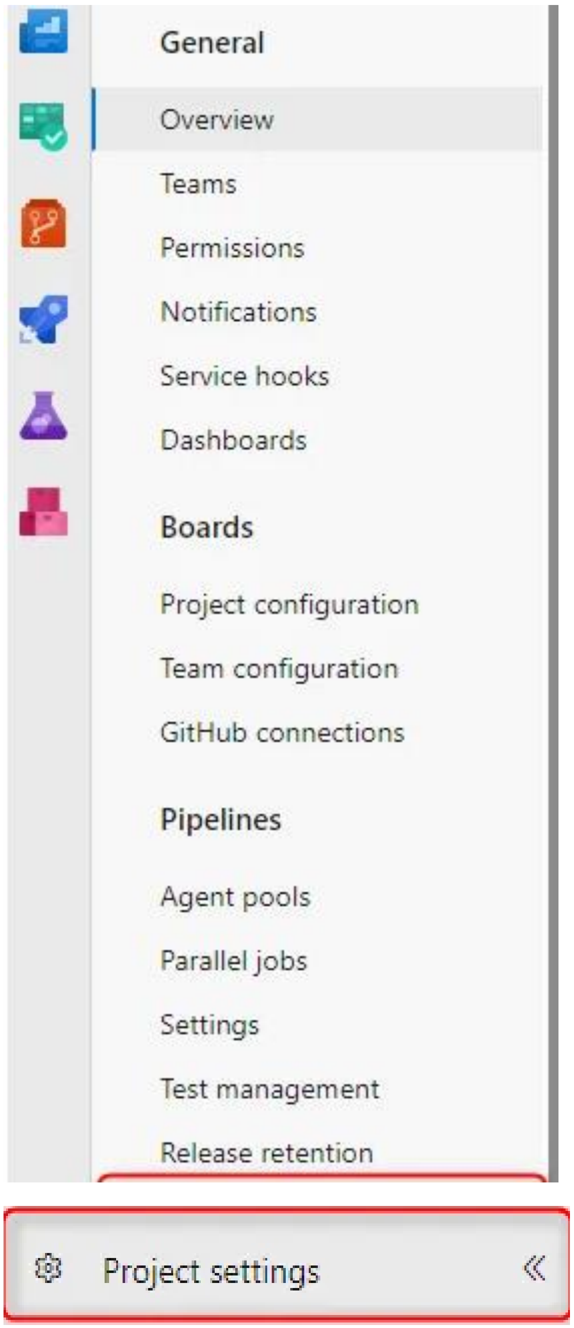
Subscription — In Azure, a subscription is a container that stores a set of related business or technical resources. The resources are used and billed collectively. An Azure account can have many subscriptions with different access management policies and invoicing procedures.

Make a subscription to the Microsoft Customer Agreement

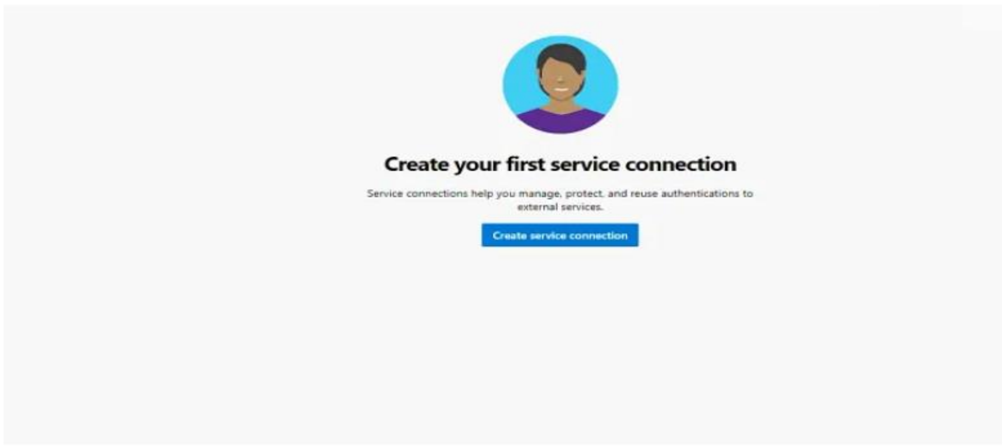
Service connection — Azure DevOps Pipelines leverage Service Connections to connect to external services such as Azure, GitHub, Docker, Kubernetes, and many others.

Choose **Project Settings** from the dashboard.



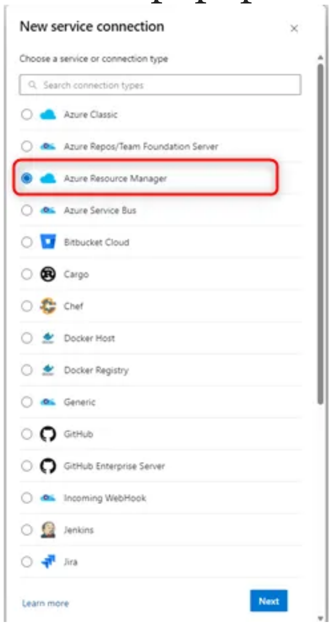


If there won’t be any service connection yet, a message will inform you. To start creating a new service connection, click **Create Service Connection**.


Create your first service connection

Service connections help you manage, protect, and reuse authentications to external services.

Once you click on it, you need to provide further details that will show as popup window.



New service connection

Choose a service or connection type

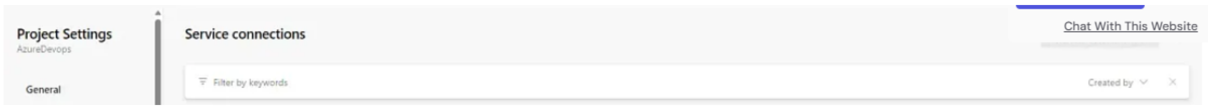
Search connection types

- Azure Classic
- Azure Repos/Team Foundation Server
- [✓ Azure Resource Manager]
- Azure Service Bus
- Bitbucket Cloud
- Cargo
- Chef
- Docker Host
- Docker Registry
- Generic
- GitHub
- GitHub Enterprise Server
- Incoming WebHook
- Jenkins
- Jira

Learn more

Next

After creation service connection it will show you list of service connection under service connection tab.



Project Settings
AzureDevops

General

Service connections

Chat With This Website

Filter by keywords

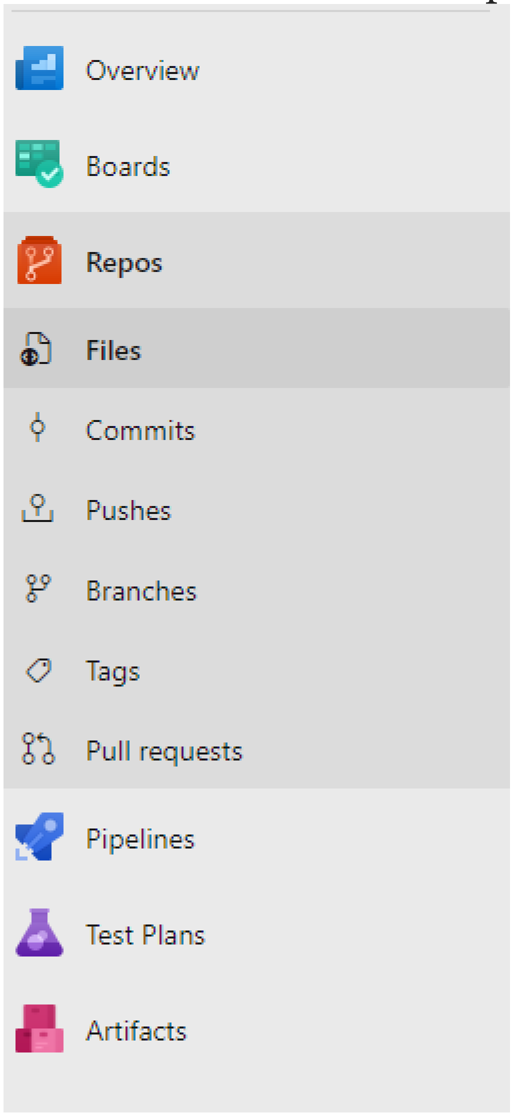
Created by

Version control system — Azure Pipelines requires you to have your source code in a version control system. Git and Azure Repos are the two types of versions control that Azure DevOps supports. Any changes you make to your version control repository are created and evaluated automatically.

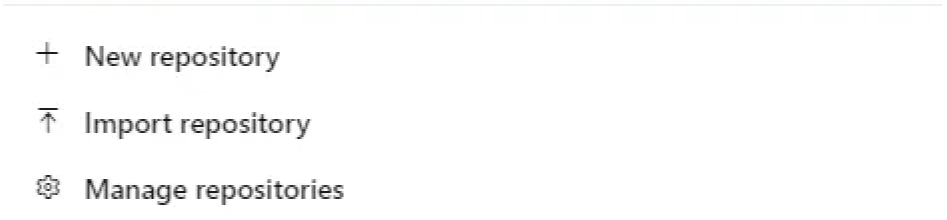
How to create new repos in Azure DevOps –

Step 1 — Select the project where you wish to build the repository after logging in. Skip this step if you are already working on the project.

There is normally a navigation menu on the left side of the project dashboard. To access the repositories area, click "Repos" or "Code".



There should be a button to start a new repository once you're in the Repos area. This is typically denoted by the word "Create" or a plus sign (+). To begin the creation process, click on it.

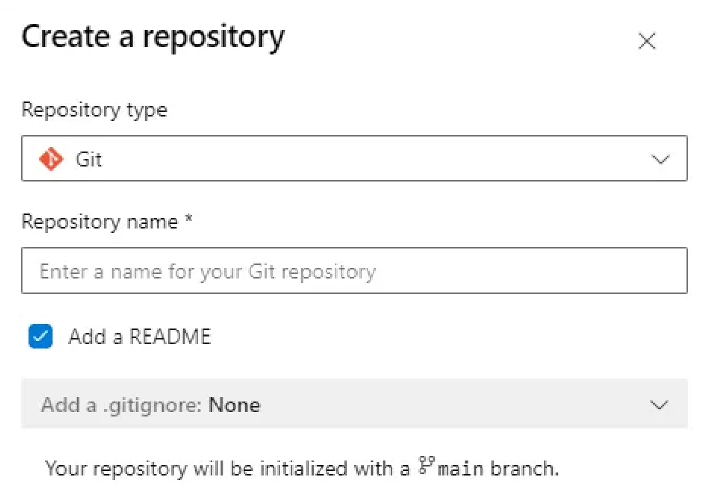


Different repository types, including Git and TFVC (Team Foundation Version Control), are available with Azure DevOps.

I have Choose "Git" because it is a well-liked and often used distributed version control system.

For the repository, you must provide the following configuration information:

– **Repository Name**: Give your repository a memorable name.
– Provide a **description** to clarify the repository's purpose, if you choose.
– Choose between **public visibility** (everyone can see it) and private visibility (only authorized users can see it).


Create a repository

Repository type: Git

Repository name *: Enter a name for your Git repository

☑ Add a README

Add a .gitignore: None

Your repository will be initialized with a main branch.

- You have the option to include a README file when starting the repository. This is useful for giving the project's overview information.
- Another option is a .gitignore template. This makes it easier to keep some files and directories out of version control.
- Choose a license template if your project is open source. This makes it easier to determine how your project will be licensed.
- After you've entered all the essential information, click "Create" to start building the repository.

Your repository will be generated in a brief period. The main page of the repository, where you can manage your code, branches, pull requests, and more, will be displayed. You must clone the repository in order to begin using it on your home computer. Copy the repository URL after clicking the "Clone" button. And you can see your repos in repos list.

```
git clone <repo path>
```

If you have already setup or configured the required elements, then you can skip that, and once you have finished all these steps, we will write our pipeline code using YAML syntax into our source code.

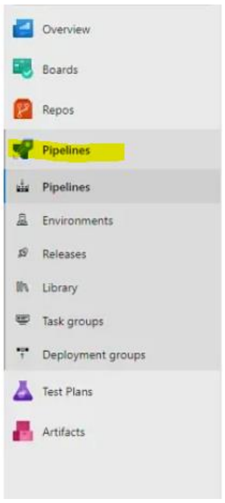
<yaml file >

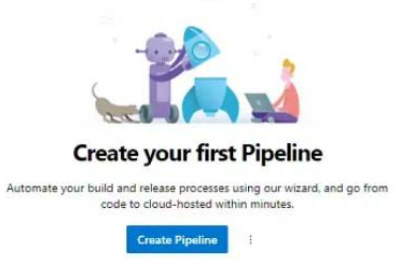
```
git add <files>
git commit -m "your message here"
git push
```

After pushing your code, you need to create the first pipeline in the pipeline section that will come from the left-side navigation of the Azure dashboard.

How to create or configure Azure Pipeline: –

Step 1 — Choose Pipelines from the dashboard, then click Builds. There won't be any build pipelines yet, a message will inform you. To start building the pipeline, click **create pipeline**.




Create your first Pipeline
Automate your build and release processes using our wizard, and go from code to cloud-hosted within minutes.
Create Pipeline

Step 2 — You will then be asked to specify the location of your code. The code for this project is kept in a GitHub repository. Decide on GitHub. The triggers for calling the build will originate from this location, where code is also kept.

Step 3 — After selecting repos, you will get new screen to choose pipeline template. Here we are going to select option. If you want to write from scratch, then you can select . `Existing Azure Pipeline YAML fileStarter Pipeline`

Step 4 — If you would select then it will ask for new values from another popup window. Like you need to provide name and the of YAML file. And click on continue.`Existing Azure Pipeline YAML filethe path`

Step 5 — In the final step you will do review your pipeline what you are going to run, and here are few other options such as variables. If you want to make it input parameter, you can add or set variables. And last, you can either save or run your pipeline.

```
trigger:
  branches:
    include:

      - main
pool:
  vmImage: 'ubuntu-latest'

variables:
  terraformVersion: '1.6.0'
  backendStorageAccount: 'terraformbackendsa'
  backendResourceGroup: 'tf-backend-rg'
  backendContainerName: 'tfstate'

stages:
- stage: Build
  displayName: 'Terraform Code Validation & Security Checks'
  jobs:
  - job: Validate_Terraform
    steps:
      - task: TerraformInstaller@0
        displayName: 'Install Terraform'
        inputs:
          terraformVersion: $(terraformVersion)

      - script: |
          terraform init
          terraform validate
        displayName: 'Terraform Validate'

      - script: |
          pip install checkov
          checkov -d .
        displayName: 'Run Checkov Security Scan'

      - script: |
          pip install tfsec
          tfsec .
        displayName: 'Run tfsec Security Scan'
```

Multiple Environments deployment steps

1. Dev Environment'

```
- stage: Deploy_Dev
  displayName: 'Deploy Dev Environment'
  dependsOn: Build
  jobs:
  - job: Terraform_Dev
    steps:
      - task: TerraformInstaller@0
        displayName: 'Install Terraform'
        inputs:
          terraformVersion: $(terraformVersion)

      - script: |
          terraform init -backend-config=backend.tfvars
          terraform plan -var-file=envs/dev/terraform.tfvars
          terraform apply -var-file=envs/dev/terraform.tfvars -auto-approve
        displayName: 'Terraform Apply Dev'
```

2. QA Environment'

```
- stage: Deploy_QA
  displayName: 'Deploy QA Environment'
  dependsOn: Deploy_Dev
  condition: succeeded()
  jobs:
  - job: Terraform_QA
    steps:
      - task: TerraformInstaller@0
        displayName: 'Install Terraform'
        inputs:
          terraformVersion: $(terraformVersion)

      - script: |
          terraform init -backend-config=backend.tfvars
          terraform plan -var-file=envs/qa/terraform.tfvars
          terraform apply -var-file=envs/qa/terraform.tfvars -auto-approve
        displayName: 'Terraform Apply QA'
```

3. Pd Environment'

```
- stage: Deploy_Prod
  displayName: 'Deploy Prod Environment'
  dependsOn: Deploy_QA
  condition: succeeded()
  jobs:
  - deployment: Deploy_Prod
    environment: Prod
    strategy:
      runOnce:
        deploy:
          steps:
          - task: TerraformInstaller@0
            displayName: 'Install Terraform'
            inputs:
              terraformVersion: $(terraformVersion)

          - script: |
              terraform init -backend-config=backend.tfvars
              terraform plan -var-file=envs/prod/terraform.tfvars
              terraform apply -var-file=envs/prod/terraform.tfvars -auto-approve
            displayName: 'Terraform Apply Prod'
```