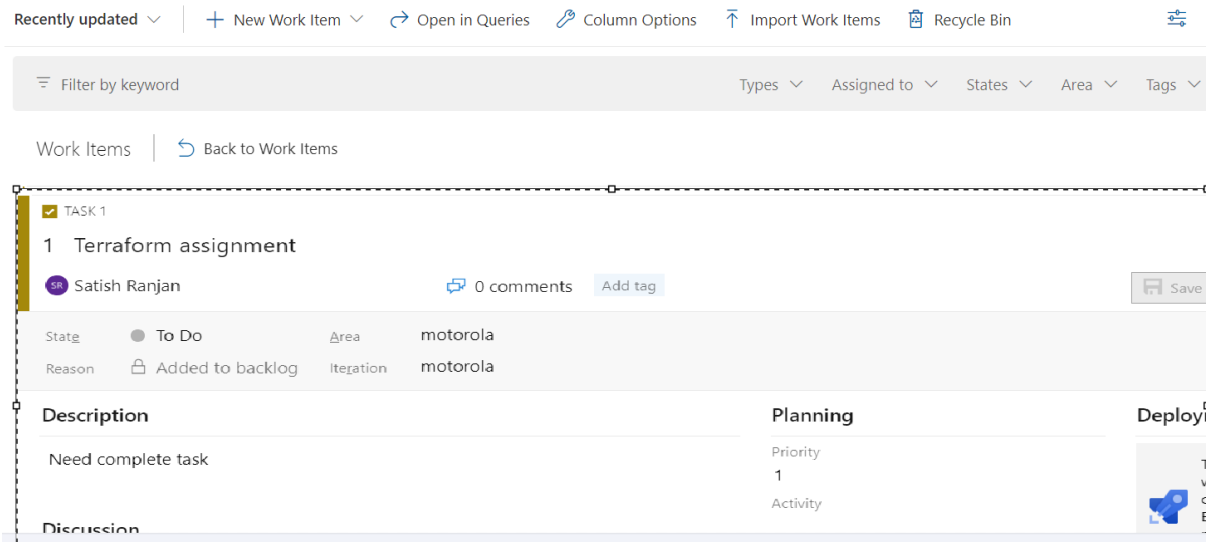
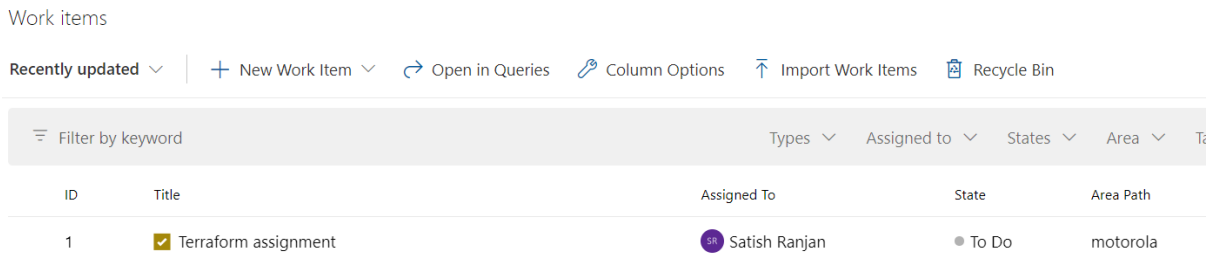
Azure CICD Pipeline

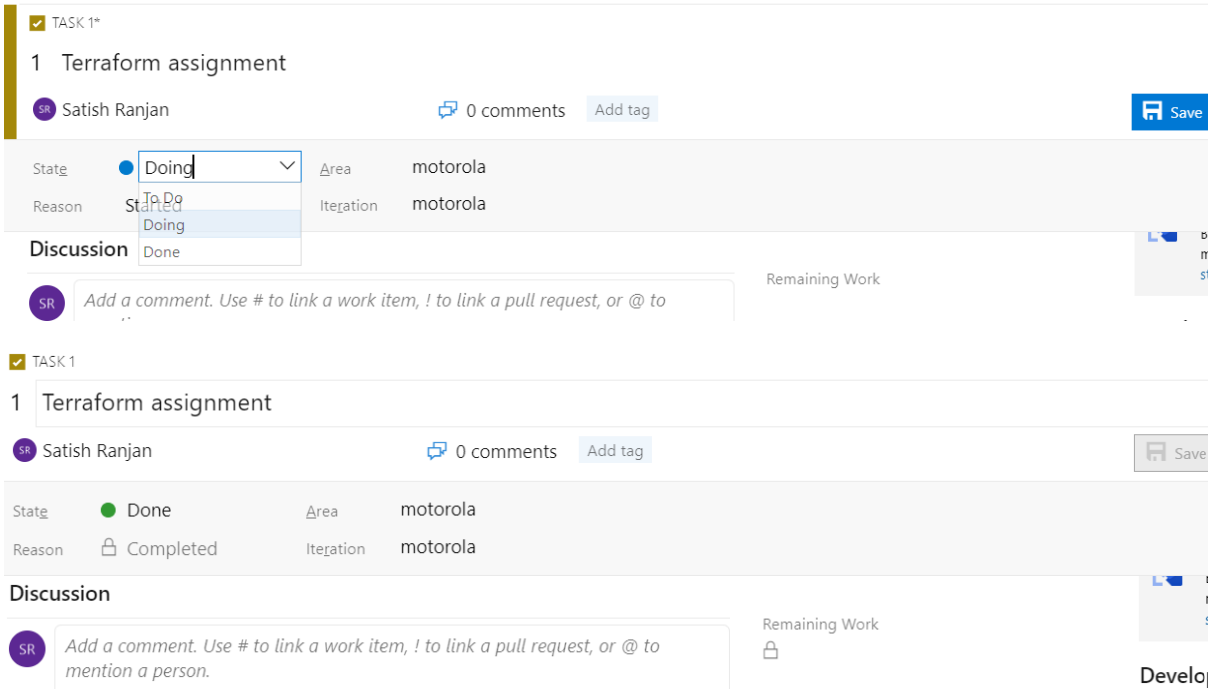
Azure Board ： Azure Boards is a tool in Microsoft Azure designed to help teams manage their projects and tasks effectively. Here's a simple breakdown:

1. Task Management: You can create, track, and manage tasks or work items. This helps teams know what needs to be done.
2. Visual Boards: It uses boards (like Kanban boards) to visualize tasks. You can move tasks through different stages (e.g., To Do, In Progress, Done).
3. Customizable: You can tailor boards to fit your team's needs, adding custom fields, tags, and workflows.
4. Collaboration: Teams can comment on tasks, attach files, and link related work items to keep everyone in sync.
5. Reporting: Azure Boards offers reporting tools to help track progress and performance, making it easier to see how the project is going.

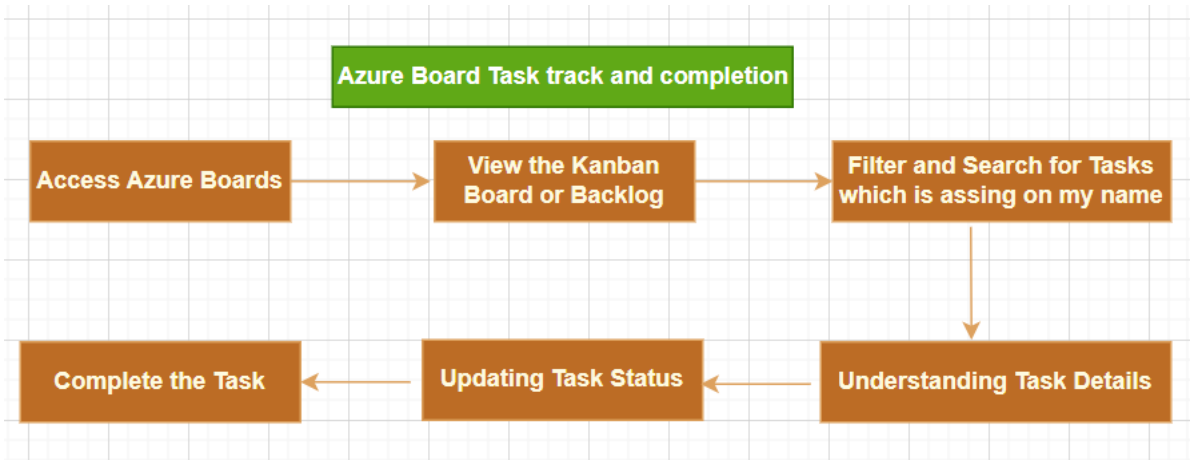
In short, Azure Boards helps teams plan, track, and discuss work all in one place!







Process to get Task In Azure Board:



1. Access Azure Boards

- We start by navigating to the our Azure DevOps project, click on Boards to access the board view

2. View the Kanban Board or Backlog

- **Kanban Board**: This displays tasks in different columns based on their status (e.g., To Do, In Progress, Done).
- **Backlog**: We can view the backlog, which lists all tasks that need to be completed, often organized by priority.

3. Filter and Search for Tasks which is assign on my name

- We can use filtering options to find specific tasks based on criteria such as assignee, priority, or tags.

- We can also use the search bar to quickly locate tasks by title or ID.

4. Understanding Task Details

- We should read the task description and any attached comments or files to understand the requirements.
- They can also view linked tasks, dependencies, and any relevant acceptance criteria.

5. Updating Task Status

- As work progresses, engineers can update the task's status by moving it to the appropriate column on the Kanban board (e.g., moving from "To Do" to "In Progress").
- They can add comments to provide updates or ask questions, and log hours or effort if that's part of their workflow.

6. Complete the Task

- Once the task is finished, the we can mark it as done. They should ensure that any necessary reviews or testing are completed as part of the process.
- After marking the task as complete, they may need to link it to any related pull requests or deployments.

Summary

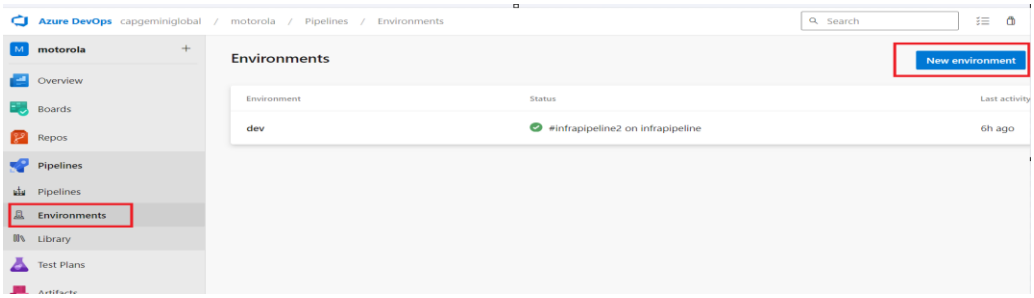
In summary, We can access Azure Boards to view and filter tasks, claim assignments, understand requirements, update statuses, and mark tasks as complete. This process fosters collaboration and transparency within the team.

Environment :

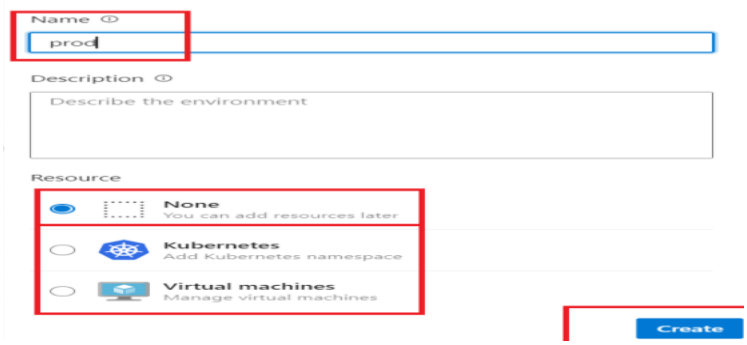
Environments in CI/CD pipelines are important for several simple reasons:

1. **Organization**: They separate different stages like Development, Staging, and Production, keeping things organized.
2. **Controlled Deployments**: You can test changes in a safe area before pushing them to the live site, reducing errors.
3. **Realistic Testing**: Environments mimic real-world settings, helping catch issues early.
4. **Configuration Management**: Each environment can have its own settings (like database connections) without changing the code.
5. **Security**: Sensitive information can be managed securely, with permissions controlling who can deploy.
6. **Approval Processes**: You can require approvals for changes, especially in Production, ensuring careful reviews.
7. **Easier Rollbacks**: If something goes wrong, it's easier to revert to a stable version.

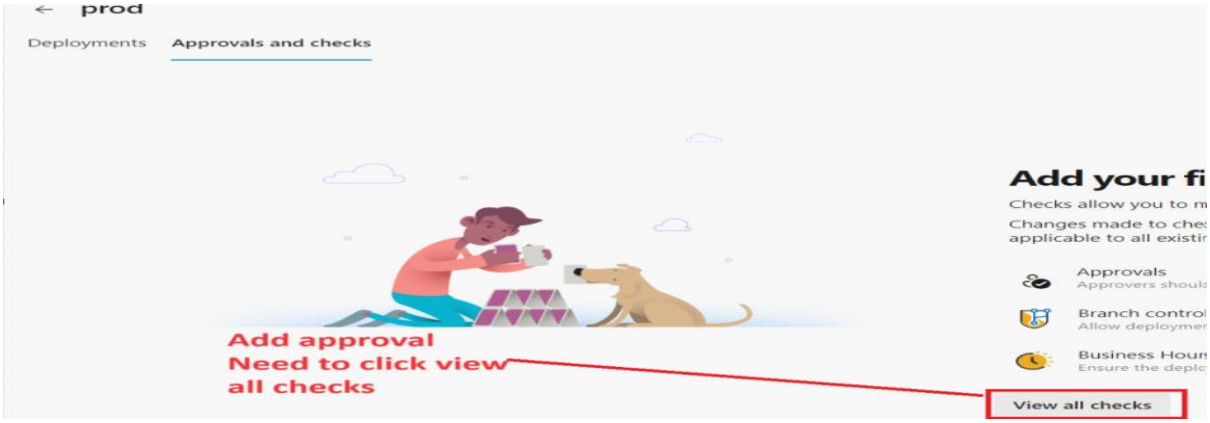
   Step1:



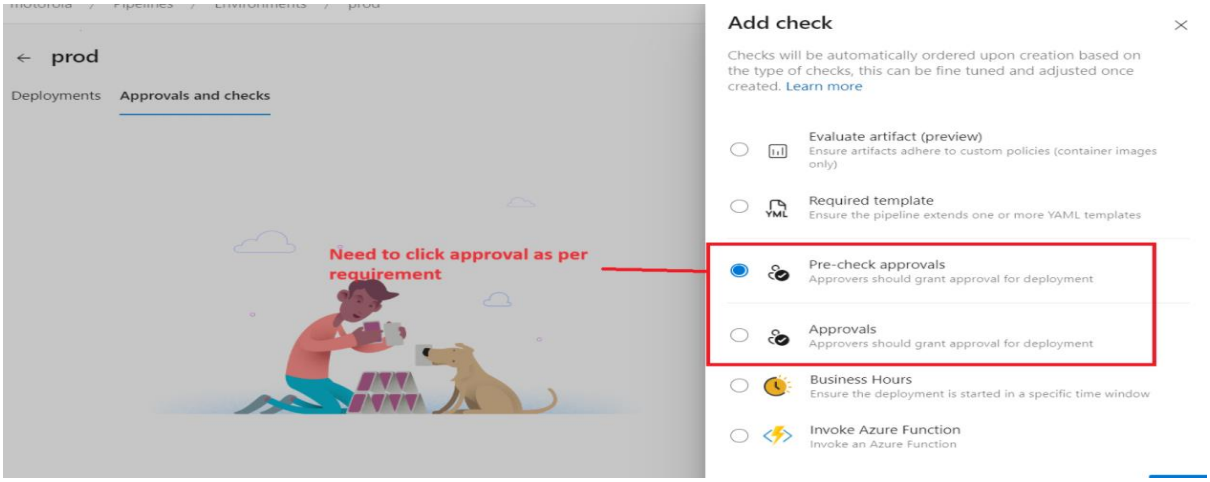
Step2:



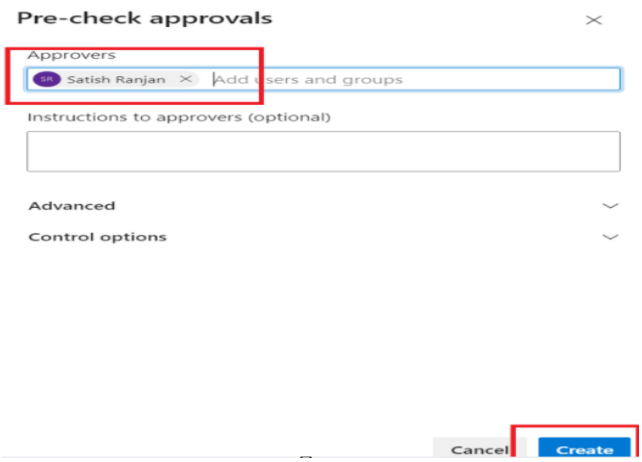
Step3:

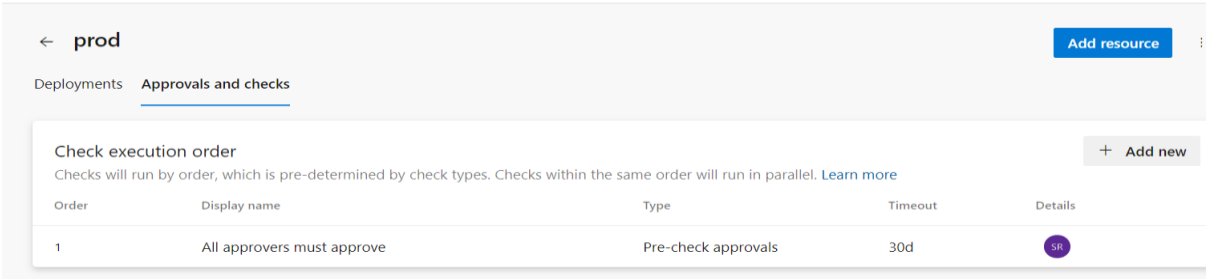


Step4:



Step5:





Step6: Now will implement this environment in CICD pipeline

Stage1:

```yaml
name: infrapipeline2
trigger: none
pool: satishpool
stages:
  - stage: terraforminstallandinit
    displayName: Terrafrom install and Init
    jobs:
      - job: terraforminstall
        displayName: Terraform install
        steps:
          - task: TerraformInstaller@1
            inputs:
              terraformVersion: 'latest'
      - job: terraforminit
        displayName: Terraform init
        steps:
          - task: TerraformTaskV4@4
            inputs:
              provider: 'azurerm'
              command: 'init'
              workingDirectory: '$(System.DefaultWorkingDirectory)/environment'
              backendServiceArm: 'satishranjanwindowshost'
              backendAzureRmResourceGroupName: 'satishrg'
              backendAzureRmStorageAccountName: 'satishranstorage'
              backendAzureRmContainerName: 'satishcontainer'
              backendAzureRmKey: 'terraform.tfstate'
```

Stage2:

```yaml
- stage: terraforminitandplan
  displayName: Terrafrom Init and Plan
  jobs:
  - deployment:
    environment: 'dev'
    strategy:
     runOnce:
      deploy:
       steps:
        - task: TerraformTaskV4@4
          inputs:
           provider: 'azurerm'
           command: 'init'
           workingDirectory: '$(System.DefaultWorkingDirectory)/environment'
           backendServiceArm: 'satishranjanwindowshost'
           backendAzureRmResourceGroupName: 'satishrg'
           backendAzureRmStorageAccountName: 'satishranstorage'
           backendAzureRmContainerName: 'satishcontainer'
           backendAzureRmKey: 'terraform.tfstate'
```

Step7: Now we can run pipeline, it will ask for approval before plan.

Template:

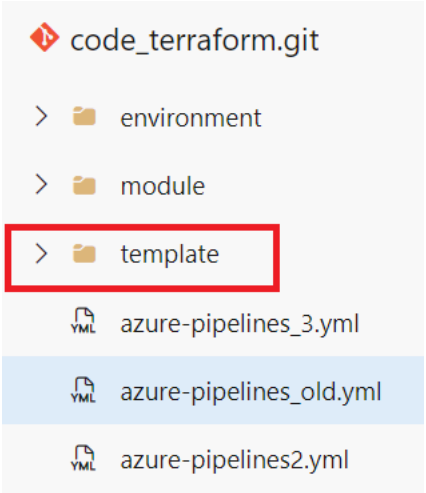
In Azure CI/CD pipelines, a **template** is a reusable piece of code that helps streamline the creation of pipelines. Think of it as a blueprint that you can use to define common tasks or settings across multiple pipelines. Here's a simple breakdown:

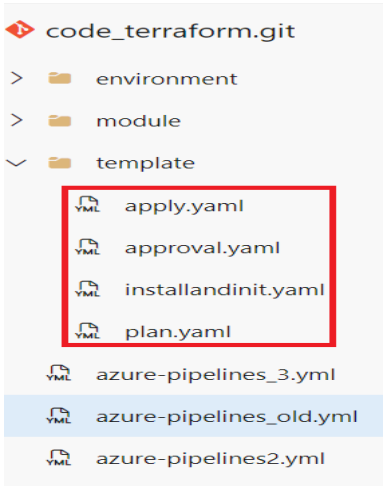
What Templates Do

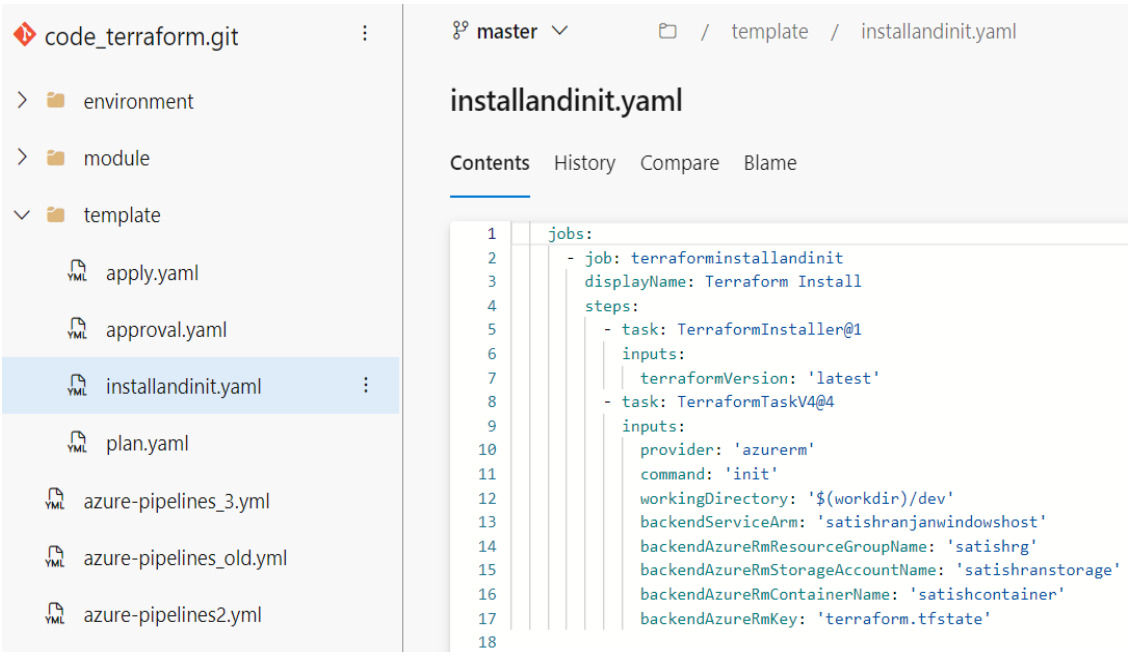
1. **Reusability**: Instead of writing the same steps multiple times for different pipelines, you create a template once and use it wherever needed.
2. **Consistency**: Using templates ensures that the same processes are followed in different projects, leading to fewer errors and more reliable builds.
3. **Simplified Maintenance**: If you need to change a common step (like deployment settings), you only need to update it in the template, and all pipelines using that template will reflect the change.

How to implement template:

Step1: Need to Prepare one folder in repo main file, inside template there will be multiple yaml file which will use in main yaml file.







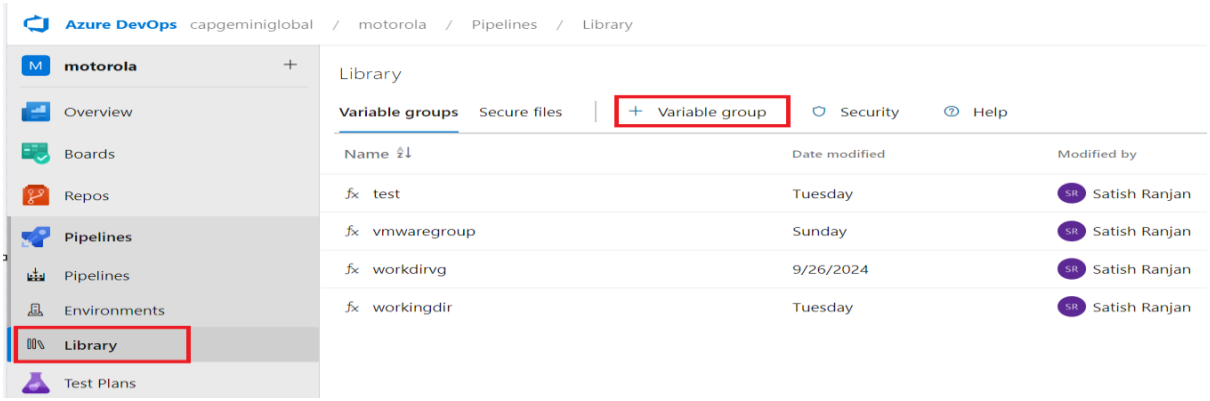
Step2: Now go to main pipeline code, and define template as per need

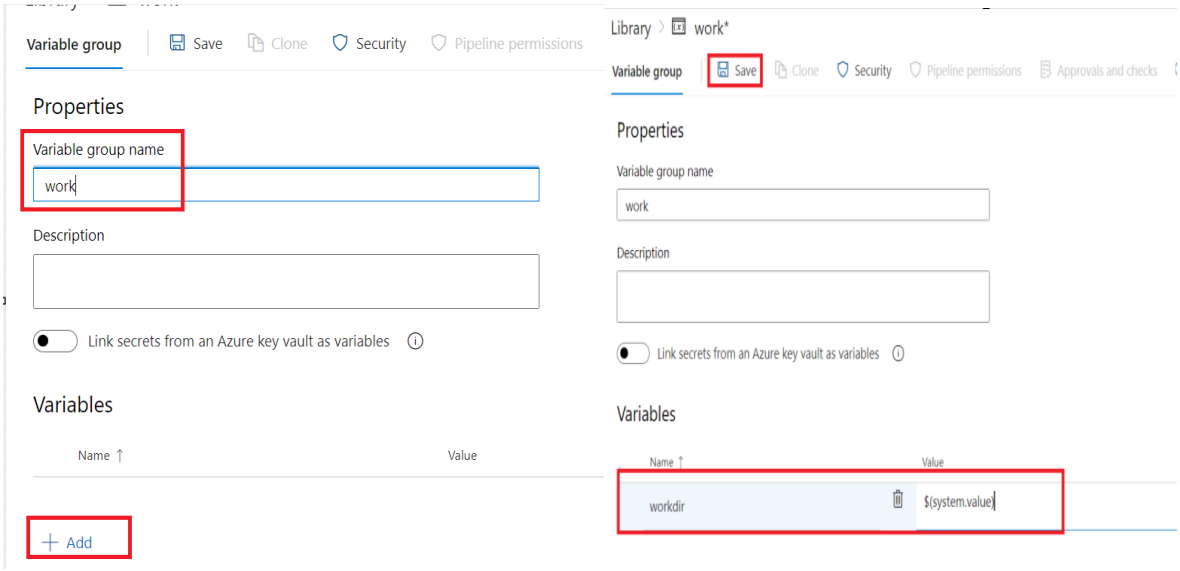
```yaml
# Template Use Case

trigger: none

pool: satishpool

variables:
 - group: workdirvg

parameters:
  - name: environment
    default: 'dev'
    type: string
    values:
      - 'dev'
      - 'prod'

stages:
# Terraform Installation and Initialization Stage
  - stage: terraforminstallandinit${{parameters.environment}}
    displayName: Terraform Installation and Initialization for ${{parameters.environment}}
    jobs:
      - template: template/installandinit.yaml

# Terraform Plan Stage
  - stage: tfplan${{parameters.environment}}
    dependsOn: terraforminstallandinit${{parameters.environment}}
    displayName: Terraform Plan for ${{parameters.environment}}
    jobs:
      - template: template/plan.yaml
# Manual Approval Stage before Apply
  - stage: ManualApproval${{parameters.environment}}
    displayName: Manual Approval Needed for ${{parameters.environment}}
    jobs:
      - template: template/approval.yaml

# Apply Stage Once Appoval Done
  - stage: terraformapply${{parameters.environment}}
    dependsOn: ManualApproval${{parameters.environment}}
    displayName: Terraform Apply for ${{parameters.environment}}
    jobs:
      - template: template/apply.yaml
```

Variable Group: In Azure CI/CD pipelines, a Variable Group is a way to manage a set of variables together. Think of it as a collection of related values that you can use across multiple pipelines.

Implement Variable Group:

Step1:





Step2: Use this variable group In pipeline yaml

```yaml
name: MultiStage Infra-Pipeline

trigger: none

pool:
 name: satishpool
 demands:
  - agent.name -equals ranjanofficeagent

variables:
 - group: workdirvg

parameters:
  - name: environment
    default: dev
    type: string
    values:
      - dev
      - prod
stages:
  - stage: terraforminstallationandinitfordev
    displayName: Terraform Installation and Initialization
    jobs:
      - job: terraforminstallation
```

```yaml
stages:
  - stage: terraforminstallationandinitfordev
    displayName: Terraform Installation and Initialization
    jobs:
      - job: terraforminstallation
        displayName: Terraform Installation
        steps:
          - task: TerraformInstaller@1
            inputs:
              terraformVersion: 'latest'
      - job: terraforminit
        displayName: Terraform Init
        steps:
          - task: TerraformTaskV4@4
            inputs:
              provider: 'azurerm'
              command: 'init'
              workingDirectory: '$(workdir)'
              backendServiceArm: 'satishranjanwindowshost'
              backendAzureRmResourceGroupName: 'satishrg'
              backendAzureRmStorageAccountName: 'satishranstorage'
              backendAzureRmContainerName: 'satishcontainer'
              backendAzureRmKey: 'terraform.tfstate'
```

Parameter:

Using parameters in Azure CI/CD pipelines makes your setup more flexible and easier to manage. Here's why they're helpful:

1. **Customization**: Change how the pipeline works without editing the code.
2. **Reusability**: Use the same pipeline for different projects or environments.
3. **Environment-Specific Settings**: Easily set values for different environments (like dev or production).
4. **Clarity**: Makes it clear what the pipeline needs to run.
5. **Dynamic Decisions**: Change pipeline behavior based on parameter values.
6. **Security**: Keep sensitive information safe by passing it securely.
7. **User Input**: Allow users to provide inputs when running the pipeline.

Overall, parameters make your pipelines more adaptable and user-friendly!

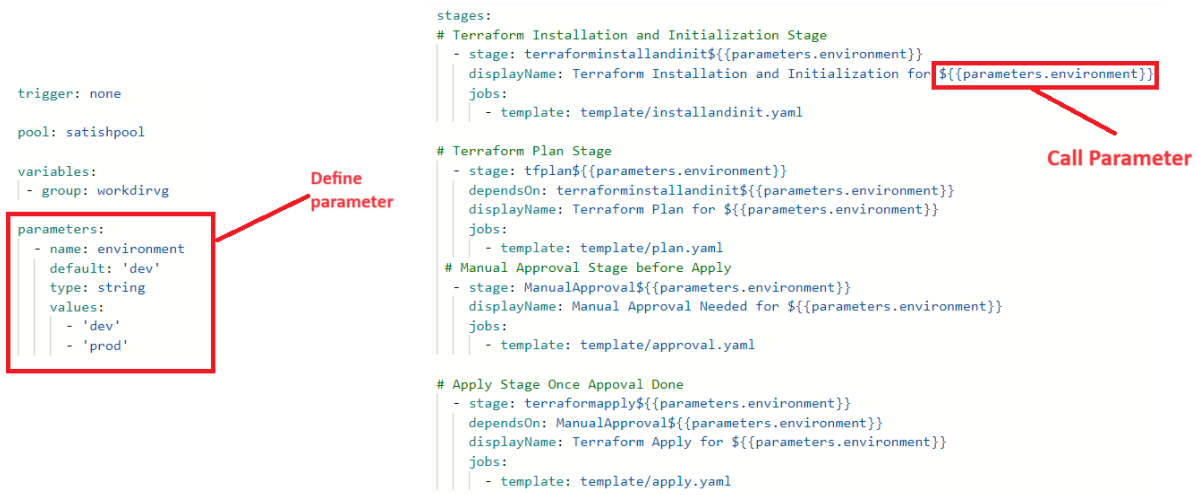
Azure CI/CD pipelines mein parameters ka istemal karne se aapki setup zyada flexible aur manageable ho jaati hai. Yahaan kuch key points hain:

1. **Customization**: Pipeline ka behavior bina code edit kiye badal sakte hain.
2. **Reusability**: Same pipeline ko alag projects ya environments ke liye use kar sakte hain.
3. **Environment-Specific Settings**: Alag environments (jaise dev ya production) ke liye values asaani se set kar sakte hain.
4. **Clarity**: Ye clear hota hai ki pipeline ko chalane ke liye kya chahiye.
5. **Dynamic Decisions**: Parameter values ke adhar par pipeline ka behavior badal sakta hai.
6. **Security**: Sensitive information ko securely pass karke safe rakh sakte hain.
7. **User Input**: Users se inputs lene ki facility milti hai jab pipeline run hoti hai.

Toh, parameters aapki pipelines ko zyada adaptable aur user-friendly banate hain!

Implementation of parameter:

Step1: Need to implement in pipeline code:



MultiStage Pipeline:

A Multi-Stage Pipeline allows you to define multiple stages in a single pipeline. Each stage can represent different phases of the development lifecycle, such as building, testing, and deploying your application.

Key Features:

1. **Separation of Concerns**: Each stage focuses on a distinct task.
2. **Parallel Execution**: Stages can run concurrently, speeding up the process.
3. **Conditions**: You can set conditions for when certain stages run based on the outcomes of previous stages.
4. **Environment Management**: Manage different environments (like dev, test, prod) within the same pipeline.

Example of a Multi-Stage Pipeline

Let's say you have a web application that you want to build, test, and deploy. Here's how a Multi-Stage Pipeline might be structured:

Stages:

- **Build Stage**: Compile the code and create artifacts.
- **Test Stage**: Run unit tests and integration tests.
- **Deploy to Staging**: Deploy the application to a staging environment for further testing.
- **Deploy to Production**: Deploy the application to the production environment after approval.

```yaml
stages:
  - stage: Build
    jobs:
      - job: BuildJob
        pool:
          vmImage: 'ubuntu-latest'
        steps:
          - script: echo "Building the application..."
          - script: dotnet build MyApp.sln
          - publish: $(System.DefaultWorkingDirectory)/bin
            artifact: drop

  - stage: Test
    dependsOn: Build
    jobs:
      - job: TestJob
        pool:
          vmImage: 'ubuntu-latest'
        steps:
          - script: echo "Running tests..."
          - script: dotnet test MyApp.Tests/MyApp.Tests.csproj

  - stage: DeployStaging
    dependsOn: Test
    jobs:
      - job: DeployStagingJob
        pool:
          vmImage: 'ubuntu-latest'
        steps:
          - script: echo "Deploying to Staging..."
          - script: az webapp deploy --name MyAppStaging --resou

  - stage: DeployProduction
    dependsOn: DeployStaging
    condition: succeeded()  # Only deploy if staging succeeded
    jobs:
```

Azure CICD Pipeline Terms

1. Build Pipeline: It takes your code and turns it into a usable application. This includes compiling the code, running tests, and creating build artifacts (like .dll or .zip files).

When It Runs: Usually runs automatically when you make changes to your code (like when you commit to Git).

Focus: Ensures that the code is working properly and can be integrated with other code.

Output: Produces files that are ready for deployment.

In simple terms, the Build Pipeline prepares your code

2. Release Pipeline: Takes the build artifacts produced by the Build Pipeline and deploys them to different environments (like testing, staging, or production).

When It Runs: Can be triggered automatically after a successful build or started manually when you're ready to deploy.

Focus: Ensures that the application is delivered correctly and runs in the target environment.

Output: Successfully deployed application in the chosen environment.

In simple terms, Release Pipeline puts it into action by build artifact generated by build pipeline

3. Artifact: An artifact is a package or file that is produced as the result of the build process. Artifacts can include various types of files needed for deployment or further testing.

Types of Artifacts: Common examples include:

- Compiled code (like .dll, .exe files)
- Libraries or packages (like .zip, .tar files)
- Docker images
- Configuration files
- Static files (like HTML, CSS, and JavaScript)

4. Agent: An agent is a software component that runs jobs and tasks in your CI/CD pipeline. It can be thought of as a worker that performs the actual building, testing, and deployment of your applications. There are two type of Agent:

a. Self hosted Agent: These are agents that you set up on your own machines or servers, giving you more control over the environment and tools installed.

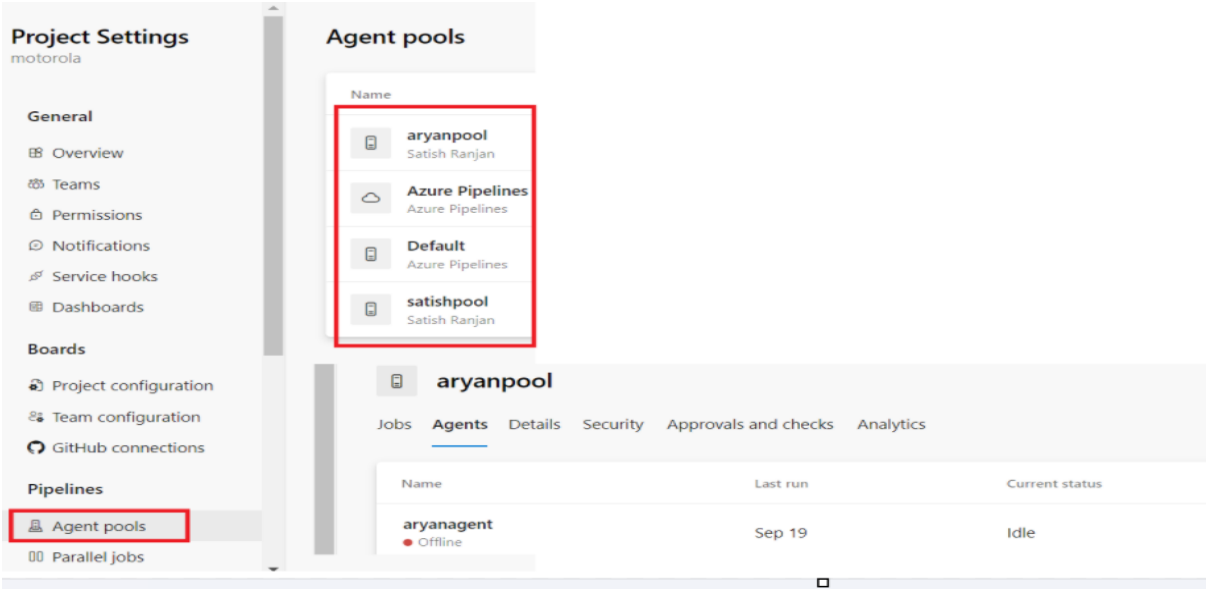
b. Microsoft Hosted Agent: These are pre-configured agents provided by Azure DevOps that run in the cloud.

How it Works in a Pipeline:

Job Assignment: When a pipeline runs, it requests an agent from the specified agent pool to execute its jobs.

Execution: The agent pulls the necessary code, executes tasks (like building or deploying), and then reports the results back to Azure DevOps.

5. Agent Pool: An agent pool is a grouping of these agents. It allows you to manage and share agents across multiple pipelines and projects.



6. Service Connection in Azure CICD Pipeline : A Service Connection allows Azure DevOps pipelines to authenticate and interact with external systems securely. It acts as a bridge between Azure DevOps and other services, enabling tasks like deployment, fetching code, or accessing resources. it helps Azure DevOps talk to other services safely. It lets your pipelines access resources or perform tasks without exposing sensitive information like passwords.

Key Features:

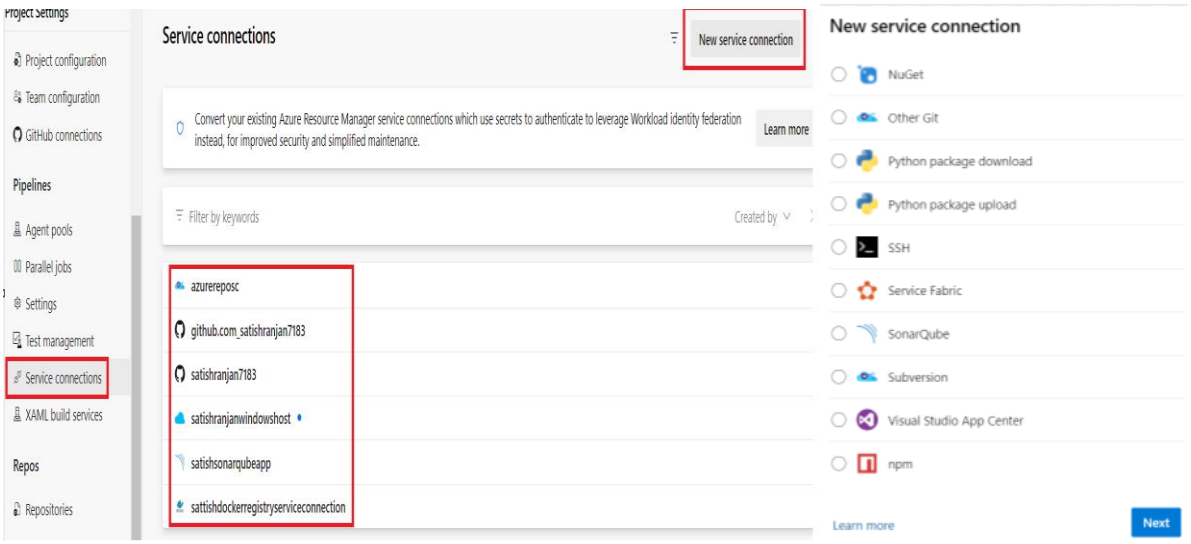
1. Authentication: It handles login details securely, so you don't have to write sensitive info in your pipeline code.
2. Reusability: Once you create a service connection, you can use it in different pipelines, making it easier to manage.
3. Security: You can control who can use the service connection, ensuring that only authorized users can access it.

How to Use Service Connections in a Pipeline:

Create a Service Connection: Go to Project Settings in Azure DevOps, select "Service connections," and set up the desired connection type.

Reference in Pipeline: Use the service connection in your YAML or classic pipeline by referencing it in tasks that require access to the external service.

Access Control: Manage permissions to control who can use the service connection in the pipelines.



7. Managed Identity: Managed Identity in Azure CI/CD pipelines is a feature that helps your applications securely access Azure resources without needing to manage credentials.

Key Points:

1. **No Credentials Needed**: You don't have to store passwords or secrets in your code; Azure handles authentication for you.
2. **Security**: Reduces the risk of exposing sensitive information, making your setup more secure.
3. **Access Control**: You can grant permissions to the managed identity, allowing it to access specific resources (like Azure Storage or Azure SQL Database).

How it Works in CI/CD:

- When your pipeline runs, it uses the managed identity to authenticate and interact with other Azure resources seamlessly.

Managed Identity Azure CI/CD pipelines mein ek feature hai jo aapki applications ko bina kisi password ya secret ke, securely Azure resources tak pahunchne mein madad karta hai.

Key Features:

1. **Credentials Ki Zaroorat Nahi**: Aapko passwords ya secrets apne code mein nahi rakhne hote; Azure khud authentication handle karta hai.
2. **Security**: Ye sensitive information ko exposure se bachata hai, jisse aapka setup zyada secure hota hai.
3. **Access Control**: Aap managed identity ko specific resources tak access dene ke liye permissions de sakte hain (jaise Azure Storage ya Azure SQL Database).

8. Publish Profile: In Azure CI/CD pipelines, a publish profile is a way to define how to deploy an application to Azure services (like Azure App Service). It contains settings that help automate the deployment process.

What is a Publish Profile?

- A publish profile is an XML file that contains configuration settings for deploying your application, such as:
    - Destination URL: Where the application will be deployed (e.g., an Azure App Service URL).

- Credentials: Authentication details like the user name and password (these are typically secure).
- Settings: Additional settings like the deployment method (e.g., Web Deploy).

Using Publish Profile in Azure CI/CD Pipeline:

1. **Create a Publish Profile**:
   - You can create a publish profile in Visual Studio when you publish your application, or you can manually create one in the Azure portal.
2. **Upload the Publish Profile**:
   - In the Azure portal, go to your App Service, find the "Publish profile" section, and download the publish profile file.
   - You can then securely store it in your Azure DevOps project.
3. **Set Up a Service Connection**:
   - Create a service connection in Azure DevOps to securely connect to your Azure subscription. This allows your pipeline to access Azure resources.
   - You can use a service connection with the publish profile to authenticate and deploy your application.
4. **Reference in Your Pipeline**:
   - In your Azure DevOps pipeline YAML or classic editor, you can reference the publish profile to specify deployment tasks. This tells the pipeline how to deploy the application using the settings defined in the publish profile.

Azure CI/CD pipelines mein **publish profile** ek tarika hai jisse aap apni application ko Azure services (jaise Azure App Service) par deploy karne ke liye settings define karte hain. Yahaan iske baare mein kuch simple points hain:

Publish Profile Kya Hai?

- **Publish Profile** ek XML file hoti hai jo deployment ke liye configuration settings contain karti hai, jaise:
  - **Destination URL**: Jahan application deploy hoga (jaise Azure App Service ka URL).
  - **Credentials**: Authentication details, jaise user name aur password (ye aam tor par secure hote hain).
  - **Settings**: Additional settings, jaise deployment method (jaise Web Deploy).

Azure CI/CD Pipeline Mein Publish Profile Kaise Use Karein:

1. **Publish Profile Create Karein**:
   - Aap Visual Studio mein apni application publish karte waqt publish profile bana sakte hain, ya phir ise Azure portal mein manually create kar sakte hain.
2. **Publish Profile Upload Karein**:
   - Azure portal mein apne App Service par jaakar "Publish profile" section mein jayein, wahan se publish profile file download karein.

- Is file ko aap apne Azure DevOps project mein securely store kar sakte hain.

3. **Service Connection Setup Karein**:
    - Azure DevOps mein ek service connection create karein taaki aap apne Azure subscription se securely connect kar saken. Ye aapki pipeline ko Azure resources tak access dene mein madad karta hai.
    - Aap publish profile ke saath service connection ka istemal karke authentication aur deployment kar sakte hain.
4. **Pipeline Mein Reference Karein**:
    - Apne Azure DevOps pipeline YAML ya classic editor mein publish profile ka reference de sakte hain, jisse aap deployment tasks specify kar sakte hain. Isse pipeline ko bataya jaata hai ki application ko publish profile mein defined settings ke according kaise deploy karna hai.