Azure DevOps Assignment – Step-by-Step Tasks

Azure DevOps Assignment – Step-by-Step Tasks

■ 1. Configure Dashboard and Queries for Work Items

■ *What It Is*

Azure Boards let you track and visualize work items (Tasks, Bugs, Epics, etc.). Dashboards and queries help your team **monitor progress, blockers, and assignments in real time**.

■ *How To Do It*

A. Create a Work Item Query

1. Navigate to **Boards → Queries**

2. Click **New Query**

3. Set filters, e.g.:

- `Work Item Type = Task`

- `Assigned To = @Me`

- `State != Closed`

4. Click **Save Query** → Choose folder: `Shared Queries`

B. Visualize the Query

1. Go to the query → Click **Charts**

2. Click **New Chart** → Choose type (Pie, Bar, Trend)

3. Group by: `State`, `Assigned To`, etc.

4. Save and name the chart

C. Add Charts to Dashboard

1. Go to **Project → Dashboards**

2. Click **New Dashboard** or select existing

3. Click **Edit** → Add widgets like:

- **Query Results**

- **Chart for Work Items**

4. Link them to your saved queries and charts

■ *Why It Matters*

Provides an at-a-glance view of work, assignments, and bottlenecks — great for agile planning or sprint reviews.

■ 2. Use Pipeline Variables While Configuring Pipelines

■ *What It Is*

Pipeline variables are **placeholders** used throughout a pipeline (for build config, environments, etc.).

■ *How To Do It*

A. Define Inline Variables in YAML

```
variables:
buildConfig: 'Release'
environment: 'dev'
```

B. Use Them in Tasks

```
- script: echo "Deploying to $(environment) with $(buildConfig)"
```

C. Stage or Job-Level Variables

```
stages:
- stage: Deploy
variables:
deployEnv: 'production'
```

D. System-Defined Variables

Some examples:

- `$(Build.BuildId)`

- `$(Agent.OS)`

- `$(Build.SourceBranch)`

■ *Why It Matters*

Variables help reuse logic across dev/test/prod environments with minimal edits.

■ 3. Use Variable and Task Groups & Set Scopes for Different Stages

■ *What It Is*

- **Variable Groups**: Shared key-value pairs

- **Task Groups**: Bundled reusable task sets

- **Scoped Variables**: Apply to specific jobs or stages

■ *How To Do It*

A. Create a Variable Group

1. Go to **Pipelines → Library → Variable Groups**

2. Click **+ Variable Group**

3. Add variables:

- `appName = myApp`

- `deployRegion = eastus`

B. Use in YAML

```
variables:
- group: dev-vars
```

C. Use Scoped Variables

```
stages:
- stage: QA
variables:
deployEnv: 'qa'
```

D. Create a Task Group (Classic UI Only)

1. Select repeated tasks → Right-click → **Create Task Group**

2. Name and parameterize values

■ *Why It Matters*

Reduces duplication and centralizes configuration for different stages.

■ 4. Create a Service Connection

■ *What It Is*

A **service connection** authorizes Azure DevOps to access Azure resources.

■ *How To Do It*

A. Create ARM Service Connection

1. Go to **Project Settings → Service Connections**
2. Click **New Service Connection → Azure Resource Manager**
3. Select **Service Principal (Automatic)**
4. Choose Azure subscription and Resource Group
5. Name it (e.g., `azure-prod-connection`) → **Verify + Save**

B. Use in Pipelines

```
- task: AzureCLI@2
inputs:
azureSubscription: 'azure-prod-connection'
scriptType: 'bash'
inlineScript: |
az group list
```

■ *Why It Matters*

Enables automated deployment and interaction with Azure resources.

■ 5. Create a Linux/Windows Self-Hosted Agent

■ *What It Is*

A self-hosted agent is a custom machine that runs builds instead of Microsoft-hosted agents.

■ *How To Do It*

A. Create Agent Pool

1. Go to **Project Settings → Agent Pools → Add Pool**
2. Name it (e.g., `self-hosted-linux`)

B. Configure Agent

On Windows:

```
config.cmd --url https://dev.azure.com/YOUR_ORG --auth PAT
run.cmd
```

On Linux:

```
./config.sh
./svc.sh install
./svc.sh start
```

C. Use in YAML

```
pool:
name: self-hosted-linux
```

■ *Why It Matters*

Useful for custom tooling, private networks, or restricted environments.

■ 6. Apply Pre and Post Deployment Approvers

■ *What It Is*

Manual approval checks before or after deployment.

■ *How To Do It*

A. Classic Release Pipelines

1. Go to **Pipelines → Releases → Edit**
2. Click ■ icon on environment
3. Add **Pre/Post-deployment approvers**

4. Assign users/groups

B. YAML-Based Pipelines

1. Go to **Pipelines → Environments → New Environment**
2. Name it (e.g., `prod-env`) → Add Approvals
3. Reference in YAML:

```
environment: prod-env
```

■ *Why It Matters*

Ensures accountability and review before production deployments.

■ 7. CI/CD Pipeline: Build & Push Docker Image to ACR and Deploy to AKS

■ *What It Is*

Builds a Docker image, pushes to Azure Container Registry, then deploys to AKS.

■ *YAML Example*

```
trigger:
- main

variables:
imageName: myapp
acrName: myregistry.azurecr.io

stages:
- stage: BuildPush
jobs:
- job: Docker
steps:
- task: Docker@2
inputs:
command: buildAndPush
containerRegistry: 'acr-connection'
repository: $(imageName)
tags: latest
```

```
- stage: Deploy
jobs:
- deployment: DeployAKS
environment: aks-prod
strategy:
runOnce:
deploy:
steps:
- task: Kubernetes@1
inputs:
azureSubscription: 'azure-connection'
kubernetesCluster: 'aks-cluster'
command: apply
configuration: 'manifests/deployment.yaml'
```

■ 8. CI/CD Pipeline: Docker to ACR → ACI

■ *What It Is*

Deploy Docker images from ACR to Azure Container Instances.

■ *YAML Snippet*

```
- task: AzureCLI@2
inputs:
azureSubscription: 'azure-connection'
scriptType: bash
inlineScript: |
az container create         --name mycontainer         --resource-group myrg         --image m
yregistry.azurecr.io/myapp:latest         --cpu 1 --memory 1.5         --registry-login-server
 myregistry.azurecr.io         --registry-username $(ACR_USERNAME)         --registry-password
 $(ACR_PASSWORD)
```

■ 9. CI/CD Pipeline: Build and Deploy .NET App to Azure App Service

■ *What It Is*

Publish and deploy a .NET app to Azure App Service.

■ *YAML Example*

```
- task: UseDotNet@2
inputs:
packageType: sdk
version: '6.x'

- task: DotNetCoreCLI@2
inputs:
command: 'publish'
arguments: '--configuration Release --output $(Build.ArtifactStagingDirectory)'

- task: AzureWebApp@1
inputs:
azureSubscription: 'azure-connection'
appName: 'dotnet-app'
package: '$(Build.ArtifactStagingDirectory)/**/*.zip'
```

■ 10. CI/CD Pipeline: Build React App and Deploy to Azure VM

■ *What It Is*

Builds a React app and copies it to an Azure VM over SSH.

■ *YAML Snippet*

```
- script: npm install
- script: npm run build

- task: CopyFiles@2
inputs:
sourceFolder: 'build'
targetFolder: '$(Build.ArtifactStagingDirectory)/build'

- task: SSH@0
inputs:
```

```
sshEndpoint: 'vm-ssh-connection'

commands: |

rm -rf /var/www/html/*

cp -R $(Build.ArtifactStagingDirectory)/build/* /var/www/html/
```
