DevOps Shack

Azure DevOps Workflow

A comprehensive Azure DevOps workflow involves integrating various Azure resources to automate processes, manage code repositories, build and test applications, and deploy them efficiently. In this guide, we'll explore a complete workflow in detail, including examples of using different Azure resources and their specific uses.

Table of Contents:

1. Introduction to Azure DevOps

Azure DevOps is a suite of services offered by Microsoft Azure to help development teams collaborate, plan, build, test, and deploy software applications. It provides a set of tools and services that support the entire software development lifecycle (SDLC), enabling teams to deliver high-quality software faster.

2. Setting Up Azure DevOps

To get started with Azure DevOps, you'll need an Azure account. Once logged in, you can create a new Azure DevOps organization and project. Within the project, you can set up teams, configure access permissions, and define work items such as user stories, tasks, and bugs.

3. Source Control Management with Azure Repos

Azure Repos is a Git-based version control system provided by Azure DevOps. It allows teams to host private Git repositories for their projects. Teams can collaborate on code, track changes, and manage branches effectively. Here's an example of using Azure Repos:

Example: Setting up a new repository for a web application and pushing code changes using Git commands:

```
# Initialize a new Git repository
git init

# Add Azure DevOps repository as remote
git remote add origin <repository_url>

# Add files to the repository
git add .

# Commit changes
git commit -m "Initial commit"

# Push changes to Azure Repos
git push -u origin master
```

4. Continuous Integration with Azure Pipelines

Azure Pipelines is a continuous integration (CI) and continuous deployment (CD) service that automates the build and deployment process. It allows teams to define build pipelines for their applications, including compiling code, running tests, and generating artifacts. Here's an example of using Azure Pipelines:

Example: Configuring a build pipeline for a Node.js application to run tests and package the application:

```
# azure-pipelines.yml

trigger:
- master

pool:
  vmImage: 'ubuntu-latest'

steps:
- task: NodeTool@0
  inputs:
    versionSpec: '10.x'
  displayName: 'Install Node.js'

- script: |
    npm install
    npm test
  displayName: 'npm install and test'

- task: ArchiveFiles@2
  inputs:
    rootFolderOrFile: '$(System.DefaultWorkingDirectory)'
    includeRootFolder: false
    archiveType: 'zip'
    archiveFile: '$(Build.ArtifactStagingDirectory)/$(Build.BuildId).zip'
    replaceExistingArchive: true
  displayName: 'Archive files'

- task: PublishBuildArtifacts@1
  inputs:
    PathtoPublish: '$(Build.ArtifactStagingDirectory)'
    ArtifactName: 'drop'
    publishLocation: 'Container'
```

5. Automated Testing with Azure Test Plans

Azure Test Plans is a comprehensive solution for manual and automated testing. It allows teams to plan, execute, and track tests, as well as analyze test results. Teams can create test plans, define test cases, and execute them across various configurations. Here's an example of using Azure Test Plans:

Example: Creating a test plan for a web application and executing automated tests using Selenium:

```
using OpenQA.Selenium;
using OpenQA.Selenium.Chrome;
using NUnit.Framework;

[TestFixture]
public class Tests
{
    IWebDriver driver;

    [SetUp]
    public void Setup()
    {
        driver = new ChromeDriver();
    }

    [Test]
    public void TestTitle()
    {
        driver.Url = "https://example.com";
        Assert.AreEqual("Example Domain", driver.Title);
    }

    [TearDown]
    public void TearDown()
    {
        driver.Quit();
    }
}
```

6. Artifact Management with Azure Artifacts

Azure Artifacts is a package management service that allows teams to create, host, and share packages. It supports various package types such as npm, NuGet, Maven, and Docker. Teams can publish and consume packages securely within their organization. Here's an example of using Azure Artifacts:

Example: Publishing a Node.js package to Azure Artifacts:

```
npm publish --registry <your_artifacts_feed_url>
```

7. Continuous Deployment with Azure Pipelines

Azure Pipelines supports continuous deployment to various Azure services such as Azure App Service, Azure Kubernetes Service (AKS), and Azure Functions. Teams can automate the deployment process and ensure consistent releases across environments. Here's an example of deploying a web application to Azure App Service:

Example: Configuring a release pipeline in Azure Pipelines to deploy a web application to Azure App Service:

```
# azure-pipelines.yml

trigger:
- master

pool:
  vmImage: 'ubuntu-latest'

steps:
- task: AzureWebApp@1
  inputs:
    azureSubscription: '<azure_subscription>'
    appName: '<app_name>'
    package: '$(Pipeline.Workspace)/drop/*.zip'
```

8. Monitoring and Feedback

Azure DevOps provides integrations with Azure Monitor and Application Insights for monitoring application performance and gathering telemetry data. Teams can set up alerts, monitor resource usage, and gain insights into application health. Additionally, Azure DevOps enables teams to collect feedback from stakeholders using features like Azure Boards and Azure Feedback.

9. Conclusion

In this guide, we've explored a complete Azure DevOps workflow, including setting up Azure DevOps, managing source code with Azure Repos, implementing continuous integration and deployment with Azure Pipelines, performing automated testing with Azure Test Plans, managing artifacts with Azure Artifacts, and monitoring applications with Azure Monitor and Application Insights. By leveraging these Azure resources effectively, development teams can streamline their processes, improve collaboration, and deliver high-quality software efficiently.