CI/CD baseline architecture with Azure Pipelines

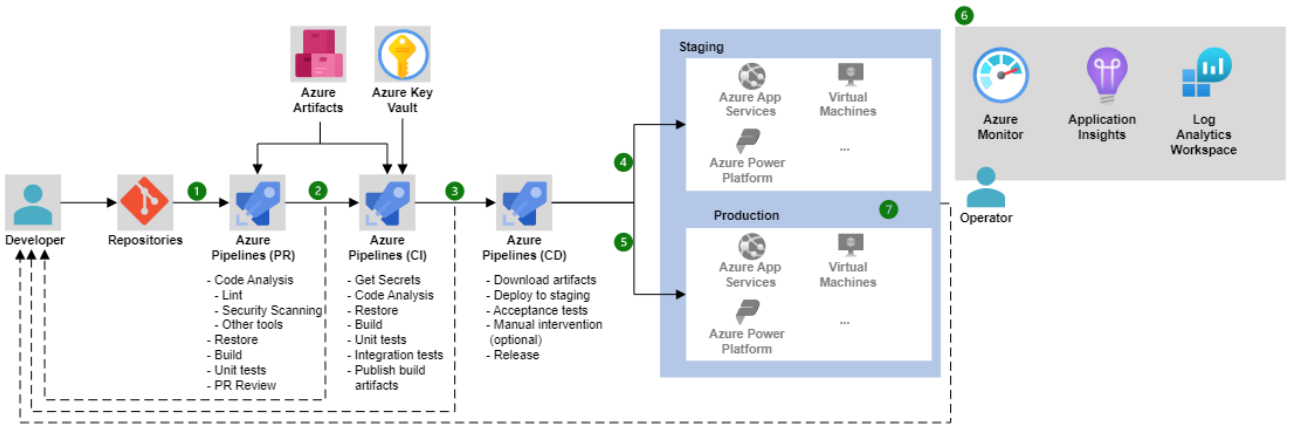
Application Insights
Azure DevOps
Pipelines
Repos
Web Apps

This article describes a high-level DevOps workflow for deploying application changes to staging and production environments in Azure. The solution uses continuous integration/continuous deployment (CI/CD) practices with Azure Pipelines.

Important

This article covers a general CI/CD architecture using Azure Pipelines. It is not intended to cover the specifics of deploying to different environments, such as Azure App Services, Virtual Machines, and Azure Power Platform. Deployment platform specifics are covered in separate articles.

Architecture



The diagram shows the following steps:
1. An engineer pushing code changes to an Azure DevOps Git repository.
2. An Azure Pipelines PR pipeline getting triggered. This pipeline shows the following tasks: linting, restore, build, and unit tests.
3. An Azure Pipelines CI pipeline getting triggered. This pipeline shows the following tasks: get secrets, linting, restore, build, unit tests, integration tests and publishing build artifacts.
4. An Azure Pipelines CD pipeline getting triggered. This pipeline shows the following tasks: download artifacts, deploy to staging, tests, manual intervention, and release.
5. Shows the CD pipeline deploying to a staging environment.
6. Shows the CD pipeline releasing to a production environment.
7. Shows an operator monitoring the pipeline, taking advantage of Azure Monitor, Azure Application Insights and Azure Analytics Workspace.

Note

Although this article covers CI/CD for application changes, Azure Pipelines can also be used to build CI/CD pipelines for infrastructure as code (IaC) changes.

Dataflow

The data flows through the scenario as follows:

1. **PR pipeline** - A pull request (PR) to Azure Repos Git triggers a PR pipeline. This pipeline runs fast quality checks. These checks should include:
    - Building the code, which requires pulling dependencies from a dependency management system.
    - The use of tools to analyze the code, such as static code analysis, linting, and security scanning
    - Unit tests

    If any of the checks fail, the pipeline run ends and the developer will have to make the required changes. If all checks pass, the pipeline should require a PR review. If the PR review fails, the pipeline ends and the developer will have to make the required changes. If all the checks and PR reviews pass, the PR will successfully merge.

2. **CI pipeline** - A merge to Azure Repos Git triggers a CI pipeline. This pipeline runs the same checks as the PR pipeline with some important additions. The CI pipeline runs integration tests. These integration tests

shouldn't require the deployment of the solution, as the build artifacts haven't been created yet. If the integration tests require secrets, the pipeline gets those secrets from Azure Key Vault. If any of the checks fail, the pipeline ends and the developer will have to make the required changes. The result of a successful run of this pipeline is the creation and publishing of build artifacts

3. **CD pipeline trigger** - The publishing of artifacts triggers the CD pipeline.
4. **CD release to staging** - The CD pipeline downloads the build artifacts that are created in the CI pipeline and deploys the solution to a staging environment. The pipeline then runs acceptance tests against the staging environment to validate the deployment. If any acceptance test fails, the pipeline ends and the developer will have to make the required changes. If the tests succeed, a manual validation task can be implemented to require a person or group to validate the deployment and resume the pipeline.
5. **CD release to production** - If the manual intervention is resumed, or there's no manual intervention implemented, the pipeline releases the solution to production. The pipeline should run smoke tests in production to ensure the release is working as expected. If a manual intervention step results in a cancel, the release fails, or the smoke tests fail, the release is rolled back, the pipeline ends and the developer will have to make the required changes.
6. **Monitoring** - Azure Monitor collects observability data such as logs and metrics so that an operator can analyze health, performance, and usage data. Application Insights collects all application-specific monitoring data, such as traces. Azure Log Analytics is used to store all that data.

Components

- An Azure Repos Git repository serves as a code repository that provides version control and a platform for collaborative projects.
- Azure Pipelines provides a way to build, test, package and release application and infrastructure code. This example has three distinct pipelines with the following responsibilities:
    - PR pipelines validate code before allowing a PR to merge through linting, building and unit testing.
    - CI pipelines run after code is merged. They perform the same validation as PR pipelines, but add integration testing and publish build artifacts if everything succeeds.
    - CD pipelines deploy build artifacts, run acceptance tests, and release to production.

- Azure Artifact Feeds allow you to manage and share software packages, such as Maven, npm, and NuGet. Artifact feeds allow you to manage the lifecycle of your packages, including versioning, promoting, and retiring packages. This helps you to ensure that your team is using the latest and most secure versions of your packages.
- Key Vault provides a way to manage secure data for your solution, including secrets, encryption keys, and certificates. In this architecture, it's used to store application secrets. These secrets are accessed through the pipeline. Secrets can be accessed by Azure Pipelines with a Key Vault task or by linking secrets from Key Vault.
- Monitor is an observability resource that collects and stores metrics and logs, application telemetry, and platform metrics for the Azure services. Use this data to monitor the application, set up alerts, dashboards, and perform root cause analysis of failures.
- Application Insights is a monitoring service that provides real-time insights into the performance and usage of your web applications.
- Log Analytics workspace provides a central location where you can store, query, and analyze data from multiple sources, including Azure resources, applications, and services.

Alternatives

While this article focuses on Azure Pipelines, you could consider these alternatives:

- Azure DevOps Server (previously known as Team Foundation Server) could be used as an on-premises substitute.
- Jenkins is an open source tool used to automate builds and deployments.
- GitHub Actions allow you to automate your CI/CD workflows directly from GitHub.
- GitHub Repositories can be substituted as the code repository. Azure Pipelines integrates seamlessly with GitHub repositories.

This article focuses on general CI/CD practices with Azure Pipelines. The following are some compute environments to which you could consider deploying:

- App Services is an HTTP-based service for hosting web applications, REST APIs, and mobile back ends. You can develop in your favorite language, and applications run and scale with ease on both Windows and Linux-based environments. Web Apps supports deployment slots like staging and

production. You can deploy an application to a staging slot and release it to the production slot.

- Azure Virtual Machines handles workloads that require a high degree of control, or depend on OS components and services that aren't possible with Web Apps (for example, the Windows GAC, or COM).
- Azure Power Platform is a collection of cloud services that enable users to build, deploy, and manage applications without the need for infrastructure or technical expertise.
- Azure Functions is a serverless compute platform that you can use to build applications. With Functions, you can use triggers and bindings to integrate services. Functions also support deployment slots like staging and production. You can deploy an application to a staging slot and release it to the production slot.
- Azure Kubernetes Service (AKS) is a managed Kubernetes cluster in Azure. Kubernetes is an open source container orchestration platform.
- Azure Container Apps allows you to run containerized applications on a serverless platform.

Scenario details

Using proven CI and CD practices to deploy application or infrastructure changes provides various benefits including:

- **Shorter release cycles** - Automated CI/CD processes allow you to deploy faster than manual practices. Many organizations deploy multiple times per day.
- **Better code quality** - Quality gates in CI pipelines, such as linting and unit testing, result in higher quality code.
- **Decreased risk of releasing** - Proper CI/CD practices dramatically decreases the risk of releasing new features. The deployment can be tested prior to release.
- **Increased productivity** - Automated CI/CD frees developers from working on manual integrations and deployments so they can focus on new features.
- **Enable rollbacks** - While proper CI/CD practices lower the number of bugs or regressions that are released, they still occur. CI/CD can enable automated rollbacks to earlier releases.

Potential use cases

Consider Azure Pipelines and CI/CD processes for:

- Accelerating application development and development lifecycles.
- Building quality and consistency into an automated build and release process.
- Increasing application stability and uptime.

Considerations

These considerations implement the pillars of the Azure Well-Architected Framework, which is a set of guiding tenets that can be used to improve the quality of a workload. For more information, see Microsoft Azure Well-Architected Framework.

Operational excellence

- Consider implementing Infrastructure as Code (IaC) to define your infrastructure and to deploy it in your pipelines.
- Consider using one of the tokenization tasks available in the VSTS marketplace.
- Use release variables in your release definitions to drive configuration changes of your environments. Release variables can be scoped to an entire release or a given environment. When using variables for secret information, ensure that you select the padlock icon.
- Consider using Self-hosted agents if you're deploying to resources running in a secured virtual network. You might also consider self-hosted agents if you're running a high volume of builds. In cases of high build volumes, self-hosted agents can be used to speed up builds in a cost efficient manner.
- Consider using Application Insights and other monitoring tools as early as possible in your release pipeline. Many organizations only begin monitoring in their production environment. By monitoring your other environments, you can identify bugs earlier in the development process and avoid issues in your production environment.
- Consider using separate monitoring resources for production.
- Consider using YAML pipelines instead of the Classic interface. YAML pipelines can be treated like other code. YAML pipelines can be checked in to source control and versioned, for example.

- Consider using YAML Templates to promote reuse and simplify pipelines. For example, PR and CI pipelines are similar. A single parameterized template could be used for both pipelines.
- Consider creating environments beyond staging and production to support activities such as manual user acceptance testing, performance and load testing, and rollbacks.

Cost optimization

Cost optimization is about looking at ways to reduce unnecessary expenses and improve operational efficiencies. For more information, see Overview of the cost optimization pillar.

Azure DevOps costs depend on the number of users in your organization that require access, along with other factors like the number of concurrent build/releases required and number of test users. For more information, see Azure DevOps pricing.

This pricing calculator provides an estimate for running Azure DevOps with 20 users.

Azure DevOps is billed on a per-user per-month basis. There might be more charges depending on concurrent pipelines needed, in addition to any additional test users or user basic licenses.

Security

- Consider the security benefits of using Microsoft-hosted agents when choosing whether to use Microsoft-hosted or self-hosted agents.
- Ensure all changes to environments are done through pipelines. Implement role-based access controls (RBAC) on the principle of least privilege, preventing users from accessing environments.
- Consider integrating steps in Azure Pipelines to track dependencies, manage licensing, scan for vulnerabilities, and keep dependencies to date.