Design a CI/CD pipeline using Azure DevOps

12/06/2018 • 5 minutes to read • Contributors all

In this article

This scenario provides architecture and design guidance for building a continuous integration (CI) and continuous deployment (CD) pipeline. In this example, the CI/CD pipeline deploys a two-tier .NET web application to the Azure App Service.

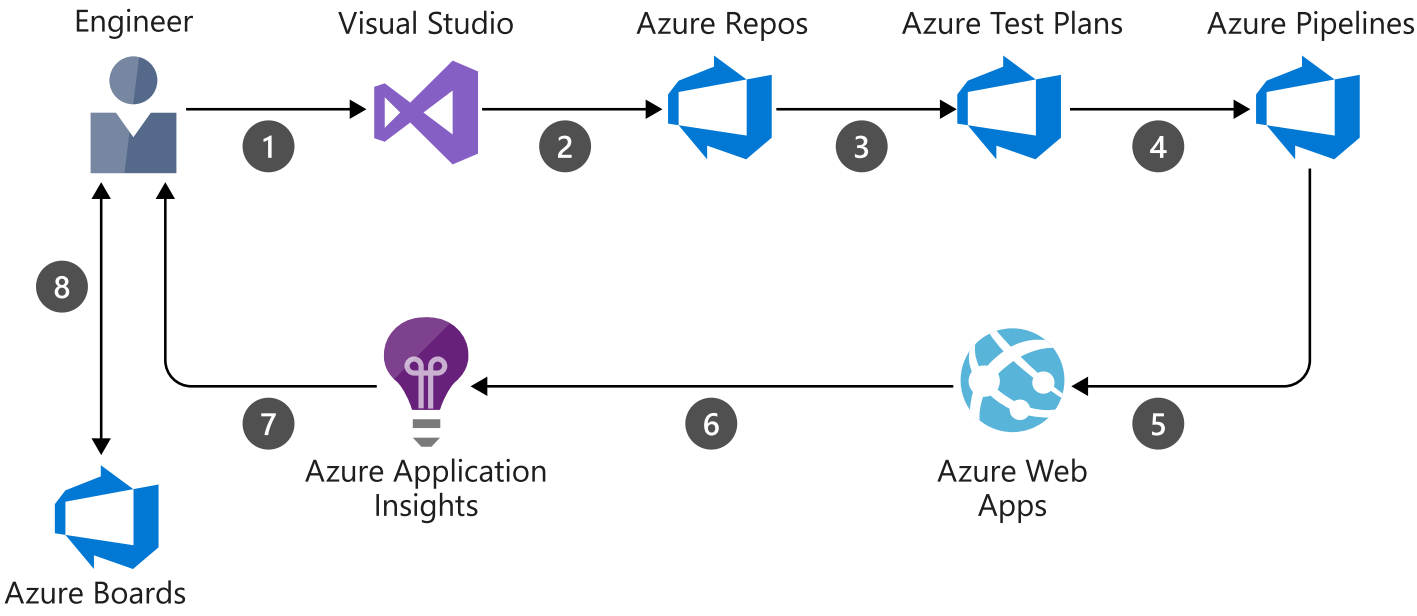
Migrating to modern CI/CD processes provides many benefits for application builds, deployments, testing, and monitoring. By using Azure DevOps along with other services such as App Service, organizations can focus on the development of their apps rather than the management of the supporting infrastructure.

Relevant use cases

Consider Azure DevOps 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.

Architecture



The data flows through the scenario as follows:

1. A developer changes application source code.
2. Application code including the web.config file is committed to the source code repository in Azure Repos.
3. Continuous integration triggers application build and unit tests using Azure Test Plans.

4. Continuous deployment within Azure Pipelines triggers an automated deployment of application artifacts *with environment-specific configuration values*.
5. The artifacts are deployed to Azure App Service.
6. Azure Application Insights collects and analyzes health, performance, and usage data.
7. Developers monitor and mange health, performance, and usage information.
8. Backlog information is used to prioritize new features and bug fixes using Azure Boards.

Components

- Azure DevOps is a service for managing your development lifecycle end-to-end—from planning and project management, to code management, and continuing to build and release.
- Azure Web Apps is a PaaS service for hosting web applications, REST APIs, and mobile back ends. While this article focuses on .NET, there are several additional development platform options supported.
- Application Insights is a first-party, extensible Application Performance Management (APM) service for web developers on multiple platforms.

Alternatives

While this article focuses on Azure DevOps, Azure DevOps Server (previously known as Team Foundation Server) could be used as an on-premises substitute. Alternatively, you could also use a set of technologies for an open-source development pipeline using Jenkins.

From an infrastructure-as-code perspective, Resource Manager templates were used as part of the Azure DevOps project, but you could consider other management technologies such as Terraform or Chef. If you prefer an infrastructure-as-a-service (IaaS)-based deployment and require configuration management, you could consider either Azure Automation State Configuration, Ansible, or Chef.

You could consider these alternatives to hosting in Azure Web Apps:

- Azure Virtual Machines handles workloads that require a high degree of control, or depend on OS components and services that are not possible with Web Apps (for example, the Windows GAC, or COM).
- Service Fabric is a good option if the workload architecture is focused around distributed components that benefit from being deployed and run across a cluster with a high degree of control. Service Fabric can also be used to host containers.
- Azure Functions provides an effective serverless approach if the workload architecture is centered around fine grained distributed components, requiring minimal dependencies, where individual components are only required to run on demand (not continuously) and orchestration of components is not required.

This decision tree for Azure compute services may help when choosing the right path to take for a migration.

Management and Security Considerations

- Consider leveraging one of the tokenization tasks available in the VSTS marketplace.
- Azure Key Vault tasks can download secrets from an Azure Key Vault into your release. You can then use those secrets as variables in your release definition, which avoids storing them in source control.
- 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.

- Deployment gates should be used in your release pipeline. This lets you leverage monitoring data in association with external systems (for example, incident management or additional bespoke systems) to determine whether a release should be promoted.
- Where manual intervention in a release pipeline is required, use the approvals functionality.
- Consider using Application Insights and additional 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.

Deploy the scenario

Prerequisites

- You must have an existing Azure account. If you don't have an Azure subscription, create a free account before you begin.
- You must sign up for an Azure DevOps organization. For more information, see Quickstart: Create your organization.

Walk-through

Azure DevOps Projects will deploy an App Service Plan, App Service, and an App Insights resource for you, as well as configure an Azure Pipelines pipeline for you.

Once you've configure a pipeline with Azure DevOps Projects and the build is completed, review the associated code changes, work items, and test results. You will notice that no test results are displayed, because the code does not contain any tests to run.

The pipeline creates a release definition and a continuous deployment trigger, deploying our application into the Dev environment. As part of a continuous deployment process, you may see releases that span multiple environments. A release can span both infrastructure (using techniques such as infrastructure-as-code), and can also deploy the application packages required along with any post-configuration tasks.

Pricing

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 may be additional charges depending on concurrent pipelines needed, in addition to any additional test users or user basic licenses.

Related resources

Review the following resources to learn more about CI/CD and Azure DevOps:

- What is DevOps?
- DevOps at Microsoft - How we work with Azure DevOps
- Step-by-step Tutorials: DevOps with Azure DevOps
- DevOps Checklist

- Create a CI/CD pipeline for .NET with Azure DevOps Projects