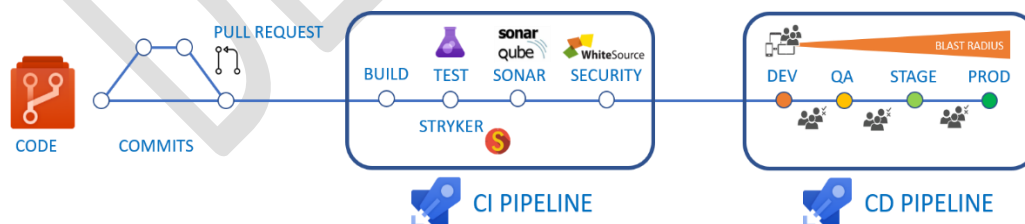


Continuous Integration using Azure Build Pipelines

- About Azure DevOps Pipeline
- Understanding the Build Process
- Create a Pipeline using Classic Editor
- Enable Continuous Triggers for Build Pipeline
- Add a status badge to Repository
- Working with Task Groups
- Validate Pull Request based on Build Pipeline result
- Add a Widget to Dashboard

About Azure DevOps Pipeline

- Azure Pipelines is a **cloud service** that you can use to **automatically build, test, and deploy** your code project. It is a fully featured continuous integration (CI) and continuous delivery (CD) service.
- Before you use continuous integration and continuous delivery practices for your applications, you must have your **source code in a version control system**. Azure Pipelines integrates with GitHub, GitHub Enterprise, Azure Repos Git & TFVC and Bitbucket Cloud.
- You can **automate** the build, testing, and deployment of your code to Microsoft Azure, Google Cloud Platform, or Amazon Web Services or On-Premise also.
- You can use many languages with Azure Pipelines, such as Python, Java, JavaScript, PHP, Ruby, C#, C++, and Go.
- Use Azure Pipelines to deploy your code to multiple targets. Targets include container registries, virtual machines, Azure services, or any on-premises or cloud target.
- To produce packages that can be consumed by others, you can publish NuGet, npm, or Maven packages to the built-in package management repository in Azure Pipelines. You also can use any other package management repository of your choice.

**Agent:**

A **build agent** builds or deploys the code. When your build or deployment runs, the system begins one or more jobs. An agent is **installable software** that runs one build or deployment job at a time.

Microsoft Hosted Agent: Each time we run a pipeline, we'll get a fresh virtual machine. There are six virtual machine images to choose from, including Ubuntu 16.04 and windows 2019.

Self Hosted Agent: Custom VM**Pipeline:**

A **pipeline** defines the continuous integration and deployment process for the app. It's made up of **one or more stages**. It can be thought of as a workflow that defines how your build, test, and deployment steps are run.

The pipeline runs when you submit **code** changes. You can configure the pipeline to run automatically, or you can run it manually. You connect your pipeline to a source repository like GitHub, Bitbucket, or Subversion.

Run

A run represents one execution of a pipeline. It collects the logs associated with running the steps and the results of running tests.

Trigger

A trigger is something that's set up to tell the pipeline when to run. You can configure a pipeline to run upon a push to a repository, at scheduled times, or upon the completion of another build. All of these actions are known as triggers.

Build Artifacts

An artifact is a collection of files or packages published by a run. Artifacts are made available to subsequent tasks, such as distribution or deployment.

The final product of the build pipeline is a **build artifact**. Think of an artifact as the smallest compiled unit that we need to test or deploy the app. For example, an artifact can be:

- A Java or .NET application packaged into a .jar or .zip file.
- A C++ or JavaScript library.
- Docker image.

Benefits of Continuous CI

Integration Continuous Integration (CI) provides many benefits to the development process, including:

- Improving code quality based on rapid feedback.
- Triggering automated testing for every code change.
- Reducing build times for rapid feedback and early detection of problems (risk reduction)
- Better managing technical debt and conducting code analysis.
- Reducing long, difficult, and bug-inducing merges.
- Increasing confidence in codebase health long before production deployment.

Understanding the Build Process

Requirement:

- The build machine is running Ubuntu 16.04.
- The build machine includes build tools like:
 - The .NET Core SDK.
 - NuGet, the package manager for .NET.

Create the Application Locally

1. `dotnet new sln -o HelloWorldApp`
2. `cd HelloWorldApp`
3. `dotnet new mvc -n HelloWorldApp.Web`
4. `dotnet sln add HelloWorldApp.Web\HelloWorldApp.Web.csproj`
5. `code .`

Here are the steps that happen during the build process:

1. **Print** build info to the wwwroot directory to help the QA team identify the build number and date.
`echo date > ./wwwroot/buildinfo.txt`
2. Run **dotnet restore** to install the project's dependencies.
`dotnet restore`
3. Run **dotnet build** to build the app under both Debug and Release configurations.
`dotnet build --configuration Debug` #Is used for debugging purpose so that we can get detailed messages/errors.
`dotnet build --configuration Release` #Is used in Production. Its performance is better.
4. Run **dotnet publish** to package the application as a .zip file and copy the results to a network share for the QA team to pick up.
`dotnet publish --configuration Debug --output /Debug`
`dotnet publish --configuration Release --output /Release`

This table associates the script commands with the new Azure Pipelines tasks:

Script command	Azure Pipelines task
<code>echo `date`</code>	CmdLine@2 or script
<code>dotnet restore</code>	DotNetCoreCLI@2
<code>dotnet build</code>	DotNetCoreCLI@2
<code>dotnet publish</code>	DotNetCoreCLI@2

Create a Pipeline using Classic Editor (Designer)

In Azure DevOps, you can use one of two methods to configure a pipeline:

1. **The visual designer.** Here, you drag tasks onto a form and then configure each task to do exactly what you need.
2. **A YAML file.** YAML is a compact format that makes it easy to structure the kind of data that's in configuration files. You typically maintain this YAML file directly with your application's source code.

Push the Sample Application to Repo

1. DevOps Portal → Create a Project (HelloWorldApp)
2. Project → **Repo** → Files → Copy Clone URL to Clipboard
3. D:\DevOpsDemos>git clone <Paste the URL>
4. D:\DevOpsDemos>dotnet new sln -o HelloWorldApp
5. D:\DevOpsDemos> cd HelloWorldApp
6. D:\DevOpsDemos\HelloWorldApp> dotnet new mvc -n HelloWorldApp.Web
7. D:\DevOpsDemos\HelloWorldApp> dotnet sln HelloWorldApp.sln add HelloWorldApp.Web\HelloWorldApp.Web.csproj
8. git add .
9. git commit -m "Initial Commit"
10. git push origin master
11. Switch back to the web portal and View the Committed Files.

Create a Pipeline

1. Pipeline → New Pipeline → **Use the classic editor**
2. Select your Source Code Repository and Branch → Continue
3. Select a Template: Select **ASP.NET Core** → Apply
4. This generates all the Tasks in a Job.
5. (Optional) Add Task: **File Creator** to create a file
File path = ./HelloWorldApp.Web/wwwroot/buildinfo.txt,
Content = "\$(Build.DefinitionName), \$(Build.BuildId), \$(Build.BuildNumber)"
6. (Optional) Add a Task: **Command line** → Add
Script = echo "\$(Build.DefinitionName), \$(Build.BuildId), \$(Build.BuildNumber)"
7. Click on **Save and queue**

Once the Pipelines completes the task – Check the Artifacts or check for the error.

Download the Artifacts

Select the Pipeline → Select the Run → On top expand button **Artifacts** → drop → download

Note: Check your email. You might have already received a build notification with the results of your run. You can use these notifications to let your team members know when builds complete and whether each build passed or failed.

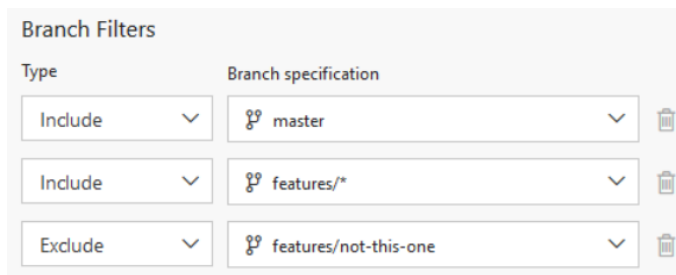
Enable Continuous Integration Triggers

Continuous integration (CI) triggers cause a build to run whenever a push is made to the specified branches or a specified tag is pushed.

Pipelines → **Pipelines** → Select Build Pipeline → Edit → **Tab Triggers** → **Continuous integration** → **Enable continuous integration**

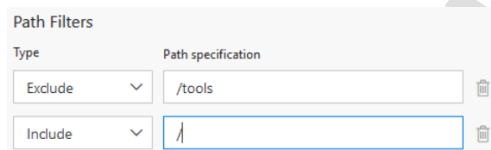
For example:

You want your build to be triggered by changes in master and most, but not all, of your feature branches.



Type	Branch specification
Include	master
Include	features/*
Exclude	features/not-this-one

You also don't want builds to be triggered by changes to files in the tools folder.



Type	Path specification
Exclude	/tools
Include	/

Batch changes

Select this check box if you have many team members uploading changes often and you want to reduce the number of builds you are running. If you select this option, when commits are made when a build is already running, the system waits until the build is completed and then queues another build of all changes that have not yet been built.

Add a status badge to your repository

Many developers like to show that they're keeping their code quality high by displaying a status badge in their repo.

1. Edit build → Options Tab → copy **Markdown** link
2. Paste in **ReadMe.md** of GitHub.
3. If required Enable anonymous access to badge
 - a. Project Settings → Pipelines → Settings → Disable "Disable anonymous access to badge"

Working with Task Groups (Classic)

A *task group* allows you to encapsulate a sequence of tasks, already defined in a build or a release pipeline, into a **single reusable task** that can be added to a build or release pipeline, just like any other task. You can choose to extract the parameters from the encapsulated tasks as configuration variables, and abstract the rest of the task information.

Create Task Group

1. For both Restore and Build unlink any parameters.
2. Select a **sequence of tasks** (restore and build tasks) in a build pipeline. Then open the shortcut menu and choose **Create task group**.
3. Specify a name and description for the new task group, and the category (tab in the Add tasks panel) you want to add it to.
4. After you choose **Create**, the new task group is created and replaces the selected tasks in your pipeline.
5. **All the '\$(vars)' from the underlying tasks, excluding the predefined variables, will surface as the mandatory parameters for the newly created task group.**
6. Save your updated pipeline.

Manage task groups

7. Azure Pipelines → **Task Groups** → Select the Task Group
8. Details page opens and you can now manage Tasks and Properties as per requirement.
9. **Hover** the Task Group → Use the **Export** shortcut command to save a copy of the task group as a JSON pipeline, and the **Import** icon to import previously saved task group definitions. Use this feature to transfer task groups between projects and enterprises, or replicate and save copies of your task groups.

Add a Widget to Dashboard

1. In Azure DevOps, select **Overview**, and then select **Dashboards**.
2. Select **Add a widget**.
3. In the **Add widget** pane, search for **Build History**.
4. Drag the **Build History** tile to the canvas.
5. Select the gear icon to configure the widget.
 - a. Keep the **Build History** title.
 - b. In the **Pipeline** drop-down list, select your pipeline.
 - c. Keep **All branches** selected.
6. Select **Save**.
7. Select **Done Editing**.

The **Build History** widget is displayed on the dashboard.
8. Hover over each build to view the build number, when the build was completed, and the elapsed build time.

Each build succeeded, so the bars on the widget are all green. If the build had failed, it would appear in red.

9. Select one of the bars to drill down into that build.

To display more widgets, select the **Extension Gallery** link at the bottom of the **Add Widget** pane.

DECCANSOFT