Build is CI and Release is used for CD/Continues Deployment.

Get started (important): <https://docs.microsoft.com/en-gb/azure/devops/pipelines/get-started-designer?view=vsts&tabs=new-nav>

<https://docs.microsoft.com/en-gb/azure/devops/pipelines/index?view=vsts>

Best place for details: <https://docs.microsoft.com/en-au/azure/devops/pipelines/get-started/?toc=/azure/devops/pipelines/toc.json&bc=/azure/devops/boards/pipelines/breadcrumb/toc.json&view=vsts>

Build a CI/CD pipeline DevOps: <https://www.youtube.com/watch?v=yNESSInONyk>

We need version control like git, to push code to some branch/master and from our branch setup build.

We can select existing templates which are collections of task for different type of projects including Azure Service Fabric Apps, Azure web apps, Azure Cloud Services, .net core, Asp.Net with Containers,

You can also go to Pipelines and Builds to add new/edit existing build.

Builds also called CI, usually produces an artifact drop that can then be deployed to various stages in a release (Select the **Utility** category, select the **Publish Build Artifacts** task).

Artifacts are the files that you want your build to produce e.g. .DLL, .EXE, scripts.

To produce artifacts, use tools such as copying with pattern matching (**Copy Files** Task), and a staging directory in which you can gather your artifacts before publishing them.

Copy Files: Use this task in a build or release pipeline to copy files from a source folder to a target folder using match patterns. Contents \*\*\bin\\*\* copies all files recursively from any bin folder, \*\*\\*.dacpac or \*\*\\*.filename.ps1 or \* copies all files in the root folder. \*\*/\* !.git/\*\*/\* (Copy everything from the source directory except the .git folder)

<https://docs.microsoft.com/en-au/azure/devops/pipelines/tasks/utility/copy-files?view=vsts&tabs=yaml#examples>

After a successful build => summary tab and you can download artifacts.

Enable continuous integration: builds => Edit => Triggers tab => Enable Continuous integration.

We can add tasks/agent job in both Builds and release like command line, copy files to build Aftifacts drop, publish Artifact drop, npm (to install and publish npm task from command line), run Bower, Azure CLI (to run a shell or bath script with Azure ClI), Azure Powershell, Azure SQL publish, deploy Ms SQL SSRS reports, Visual Studio Test to run automated test including Selenium with test adapters such as Mstest, NUnit, Qunit, Mocha, Jasmine. Send email, Assembly Info to set assembly version. Azure Dev Test labs tasks. Post to Slack, Zip and Unzip, SSIS build and deploy,

For each task get help from DevOps by clicking **learn more**.

We use build variables like Build.ArtifactStagingDirectory, System.DefaultWorkingDirectory etc.

We do gates builds which mean someone has to approve before the release/deployment. Or code coverage etc. we can also make our release to automatic on each build.

Builds: We do integration/CI and quality check/run tests, check build fail/succeeded.

Releases: We publish/deploy project in releases.

“refs/heads/master” is usually same as “refs/remotes/origin/master” in Git. because origin is the default name for the remote created by git clone and its primary branch is usually also named master.

**CI CD Steps detailed with Git branches (DevOps Git)**:

Use the visual designer: <https://docs.microsoft.com/en-gb/azure/devops/pipelines/get-started-designer?view=vsts&tabs=new-nav>

Prerequisites: You need a Git repository in Azure Pipelines, TFS, or GitHub with your app.

Create web project in VS => build => click “Add to Source Control” at bottom right.

Or right click solution => add solution to source control (local git).=> click Sync (publish Repository to Team Services)

* Publish Git Repo => advance => select Team project => select Team services domain => click **Publish Repository**.

This will create a new project in DevOps and add code repository.

Go to Azure Repos: dev.azure.com or visualstudio.com => select the project=> Repos

Create a build pipeline: dev.azure.com or visualstudio.com => select the project=> Pipelines => new pipeline => Select a source => Azure Repos Git => Select Project => select Repository => Click Continue => Select a template

Select a template: build usually produces an artifact drop and run some test but we publish in releases. So select **Build and Test Asp.Net Core Web App** => Apply => Save and Queue.

You can select branch from dropdown and put comments before running the build.

dev.azure.com or visualstudio.com => select the project=> Pipelines => builds => select the running build => logs, summary or tests

you can **download artifact drop** from build => select the build form list => summary tab (under Build artifacts published).

We can also create separate artefact like db (\*\*\\*.dacpac) or PowerShell (\*\*\\*.filename.ps1)

Enable continuous integration: builds => Edit => Triggers tab => Enable Continuous integration.

You can queues at any time manually.

Create separate artefact for DB:

Add Sql Server Database project in VS: got VS solution => Add new project => SQL Server => Sql Server Database project => Poc-Db

Right click properties => Target platform => Microsoft Azure SQL Database V12

Add folder dbo and add tables folder => add new table e.g. Persons => can copy past the sql to create a table from w3 schools

add post deployment script => add new => User Scripts => Post-Deployment Script

you can also set property of the sql script file to post deployment.

Ideally use separate sql scripts for data and use merge statements. Reference those script files in the Post-Deployment Script file. Like .\tables\DataScripts\Table1.Data.sql

You can get .dacpac from the drop folder and test on your local db.

Separate artefact for DB:

Go to bulds => Edit => Agent Job 1 => + (Add a Task to Agent Job 1) => find **Copy Files** task => Add

In Contents Put (\*\*\\*.dacpac) and on next line may be (\*\*\\*. PostDeployment1.sql)

In target put “$(build.artifactstagingdirectory)\database”

Advanced => Flatten Folders and save.

Publish Build Database Artifacts: Edit => Agent Job 1 => + (Add a Task to Agent Job 1) => find **Publish Build Artifacts** => Add

Path to publish = $(build.artifactstagingdirectory)\Db

Artifact name = db

Add task “**Visual Studio Build**” to build the Dacpac/dababase project and any other projects in the solutions. Select the VS version.

Please use Hosted Agent => **Hosted VS2017** to run the build, otherwise you may get error.

Plateform: $(BuildPlatform)

Configuration: $(BuildConfiguration)

**Review code with pull requests**:

Create pull requests to review and merge code in a Git project. Pull requests let your team review code and give feedback on changes before merging it into the master branch. Pull requests can come from either topic branches within the same repository or from a branch in a fork of the original repository. Reviewers can step through the proposed changes, leave comments, and vote to approve or reject the code.

Pull requests combine the review and merge of your code into a single collaborative process

<https://docs.microsoft.com/en-au/azure/devops/repos/git/pull-requests?view=tfs-2017&tabs=new-nav>

For pull request we need to first create a branch which we want to merge to the master.

Note: Protect branches with policies like master. Require pull requests to make any changes on these branches. Developers pushing changes directly to the protected branches will have their pushes rejected.

Create new branch: VS => Team Explorer => Branches => master => right click => select “**New local branch from...**” => enter branch name like “users/naveed/FixDbData” => Create Branch

Do you code changes push changes to local and Sync.

Go to dev.azure.com or visualstudio.com => select the project=> Repos => Branches => your branch (users/naveed => FixDbData) => mouse hover => click “New pull request” => next page

You can also create pull request from VS => Team Explorer => Home => Pull Request.

Also from Visual Studio from the Branches view in Team Explorer by right-clicking the branch name and selecting Create pull request while connected to your project.

from you branch to master (by default selected) => Enter title => Reviewers = naveed eamil => work items = 123 => create.

If you have **branch policies**, you can choose **Set auto-complete** to configure the pull request to close once all branch policies are met.

Configure **branch policies**:

<https://docs.microsoft.com/en-au/azure/devops/repos/git/branch-policies?view=tfs-2017>

Navigating to your project in the web portal (dev.azure.com or visualstudio.com) and selecting Repos, Branches => Locate your branch (master) => mouse hover => … (more action) => click to open context menus => select “**Branch polices**” => next page

Under **Build validation** => click + Add build policy => select the “**Build pipeline**” from dropdown.

New go to VS and do some code changes and push changes to the branch against a pull request is create or create a new pull request. It will run the build also.

Set auto complete and enable “Squash changes when merging”.

Rebase the current branch if master has changes:

Right click the branch in Team Explorer => Pull (get latest) (Pull = Fetch + merge)

git status

=> git checkout master

=> git pull

git checkout {users/username/description}

git checkout @{-1}

Rebase from master

git rebase master

git rebase -–continue

git push --force-with-lease

**Deploy a SQL DACPAC to Azure using Azure DevOps (publish artifacts)**: automatically deploy your database updates to Azure SQL database after every successful build.

Go to VisualStudio.com => sign in => User in top right corner => Visual Studio profile => Azure DevOps Organizations => New project/open existing project/Create project => Pipelines => Builds

The Build Definition follows the same basic steps.

Add separate Publish Artifact step/task one for each of the Projects contained in the Solution. In this instance, an Artifact for the Database project and one for the WebAPI Project.

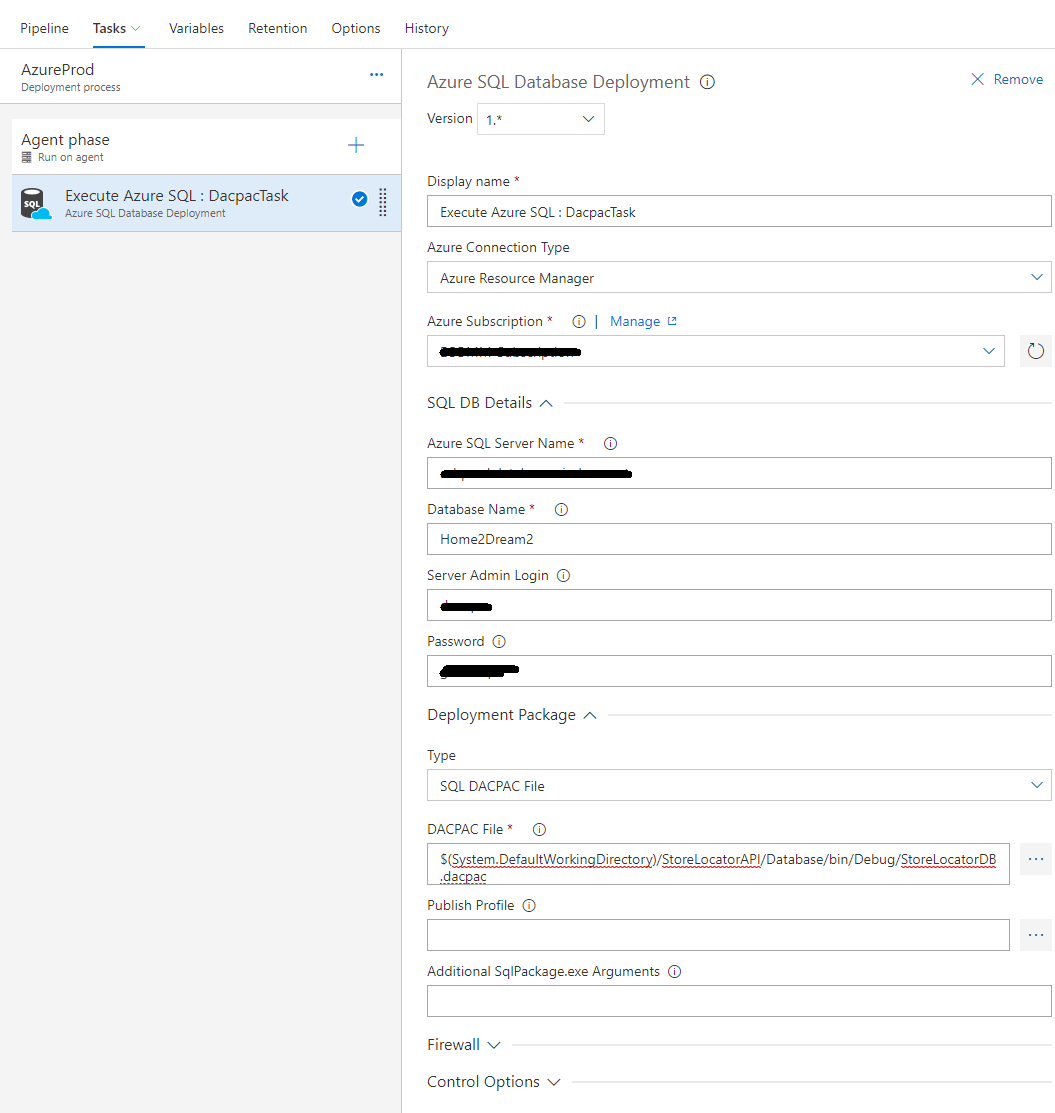
Tips:

Sql Database Project Targeting the Azure Platform. Open the Solution in Visual Studio and choose the Microsoft **Azure SQL Database V12** option in project settings => Target platform.

<https://docs.microsoft.com/en-us/azure/devops/pipelines/targets/azure-sqldb?view=vsts&tabs=yaml>

When setting up a build/CI pipeline for your Visual Studio database project, use the .NET desktop template. This template automatically adds the tasks to build the project and publish artifacts, including the DACPAC.

When setting up a release pipeline, choose Start with an empty pipeline, link the artifacts from build, and then add an Azure SQL Database Deployment task.



Go to VisualStudio.com and create new project. Go to Branches and clone.

Go to VS create a project => Connect to TFS/Git

Go to VisualStudio.com => branch/master => click on setup build.

Create new project in DevOps (if required): Go to VisualStudio.com => select organization => create project (right top corner)

You can also directly publish to Azure from VS.

* Go to dev.azure.com or visualstudio.com => select the project => Repos => click on Set up build (in right top corner) => select the APS.Net with Azure template
* Authorize Azure credentials on next page
* Add or remove build steps accordingly

A common workflow with Git is to create branches from your master branch. Eventually, you merge the code back to the master branch and delete the topic branch.

Follow the steps below to create a CI trigger that will run a build for feature branches.

1. Select Pipelines, and then choose Builds.
2. Locate the build pipeline that services your master branch. Select Edit.
3. Select the Triggers menu for your build. Ensure you have Continuous integration enabled.
4. Select the + Add icon under Branch filters.
5. Under the Branch specification dropdown, type features/\* in the Filter my branches text box and press Enter. The trigger now supports CI for all feature branches that match the wildcard as well as the master branch. Note that the filtered list of branches may not populate as you type \*. You can still press Enter and save the branch filter.
6. Select the Save & queue menu and then Select Save.

<https://docs.microsoft.com/en-us/azure/devops/pipelines/build/ci-build-git?toc=%2Fazure%2Fdevops%2Fdeploy-azure%2Ftoc.json&%3Bbc=%2Fazure%2Fdevops%2Fdeploy-azure%2Fbreadcrumb%2Ftoc.json&view=vsts&tabs=designer>

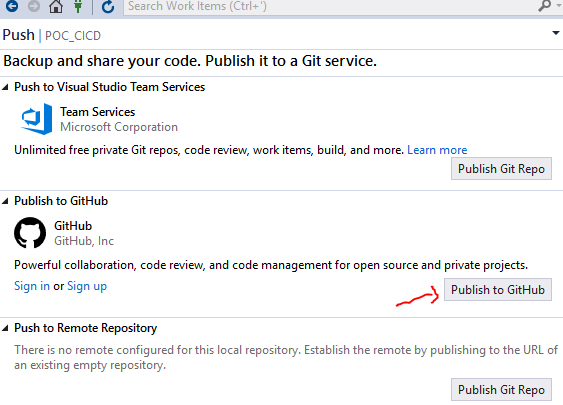
**CI CD Steps detailed with Git branches (GitHub – not complete)**:

Prerequisites: You need a Git repository in Azure Pipelines, TFS, or GitHub with your app.

Create web project in VS => build => click “Add to Source Control” at bottom right.

Or right click solution => add solution to source control (local git).=> click Sync

Team Explorer => home => Sync => Publish to GitHub/Git Repo Team Services (Sign in/Sign up Git Hub) => Push changes to git Hub.



Add master branch from GitHub to Dev Ops for CICD/Connect GitHub with Azure Boards:

<https://docs.microsoft.com/en-us/azure/devops/boards/github/connect-to-github?view=vsts>

Go to Azure DevOps account from <https://dev.azure.com>. Select Organization => Create Project (just enter the name) => Go to Project Settings => Boards => GitHub connections => enter GitHub credentials (must be an administrator) => Authorize Azure Boards and wait to redirect to dev.azure.com.

Note: use Edge or Chrome. Go to <https://visualstudio.microsoft.com/> sign in => select the project => Settings => Boards etc

Select/Add GitHub repositories => Save.

Right click the project solution in VS => Configure Continuous Delivery to Azure

In pop up click Re-enter your credentials or sign up for Azure.

GitHub personal access token (PAT) is required => follow the link => click Generate a personal access token => select all check boxes => copy token => past token in GitHub PAT.

Select Team services Domain => Select Team Project => select App Service => click ok and wait.

XXXXXXXXXXXXXXXXXXX

Integrate Your GitHub Projects With Azure Pipelines:

<https://www.azuredevopslabs.com/labs/azuredevops/github-integration/>

Go to <https://dev.azure.com> => select the project => Pipeline => Builds => New pipeline => select GitHub => Authorize using OAuth => follow step and select the Repository => continue.

Select a template => e.g. Azure Web App for Asp.Net => apply.

Select the step Azure App Service Deploy => Azure subscription click Manage => oh help icon I click and follow the steps (Generate Publish Settings) to download the file with details about the id and certificate etc. => copy past the subscription id, Name and certificate in the pop up (in Pipelines => Service Connections) => verify => ok.

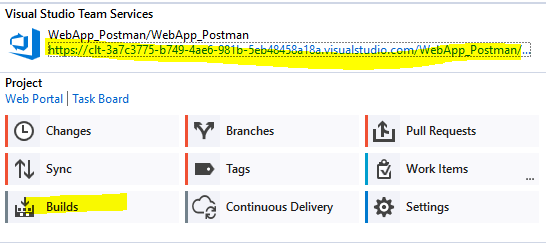
NOTE: The task **does not work with the Azure Classic service endpoint** and it will not list these connections in the parameters in the task. Select create a **Resource Manager service connection**.

<https://docs.microsoft.com/en-us/azure/devops/pipelines/release/azure-rm-endpoint?view=vsts>

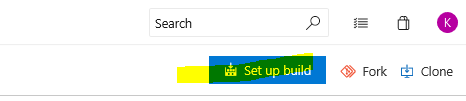
To deploy to Azure, an Azure subscription has to be linked to Team Foundation Server or to Azure Pipelines using the Services tab in the Account Administration section.

XXXXXXXXXXXXXXXXXXXXXX

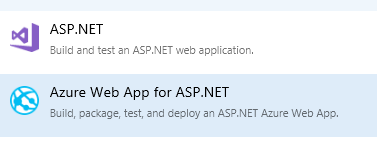
Create/Add code to Git repository



Click on the link to take you to DevOps.



Click on “Set up build”.



Select Azure Web App

Fill out the input fields and Save & Queue.

DevOps uses a Slack Integration to send a notification to a Slack channel so that we know if the tests passed or failed.

bind TASK RUNNER to build events

Update Node Version in Visual Studio 2017: Tools > Options > Projects and Solutions > Web Package Management > External Web Tools

Put “**C:\Program Files\nodejs**” on top.

task runners like Grunt and Gulp

Command Task Runner : https://marketplace.visualstudio.com/items?itemName=MadsKristensen.CommandTaskRunner

NPM Task Runner: https://marketplace.visualstudio.com/items?itemName=MadsKristensen.NPMTaskRunner

Batch files:

lines can be commented using REM or ::

A watch task keeps an eye on files and directories.

A **fork** is a complete copy of a repository, including all files, commits, and (optionally) branches. Once you're ready to share changes done in created fork, it's easy to contribute them back using pull requests.

For a very small team (2-5 developers), we recommend working in a single repo. Everyone should work in topic branches, and master should be protected with branch policies. As your team grows larger, you may find yourself outgrowing this arrangement and prefer to switch to a forking workflow.