Type script:

**let vs var**:

The **let** keyword is actually a newer JavaScript construct that TypeScript makes available. Many common problems in JavaScript are alleviated by using let, so you should use it instead of var whenever possible.

**const** is an augmentation of let in that it prevents re-assignment to a variable.

let isDone: boolean = false;

let color: string = "blue";

color = 'red';

let decimal: number = 6;

let hex: number = 0xf00d;

let binary: number = 0b1010;

let octal: number = 0o744;

**Template strings**:

You can also use template strings, which can span multiple lines and have embedded expressions. These strings are **surrounded by the backtick**/backquote (**`**) character, and embedded expressions are of the form **${ expr }**.

let fullName: string = `Bob Bobbington`;

let age: number = 37;

let sentence: string = `Hello, my name is ${ fullName }. I'll be ${ age + 1 } years old next month.`;

let list: number[] = [1, 2, 3];

let list: Array<number> = [1, 2, 3]; //another way

**Tuple** :

let xTuple: [string, number];

xTuple = ["hello", 10];

**Enum** :

enum Color {Red, Green, Blue}

let c: Color = Color.Green;

enum Color {Red = 1, Green , Blue = 4}

let c: Color = Color.Green;

let colorName: string = Color[2];

**Any**:

We may need to describe the type of variables that we do not know when we are writing an application. These values may come from dynamic content, e.g. from the user or a 3rd party library.

let notSure: any = 4;

notSure = "maybe a string instead";

notSure = false; // okay, definitely a boolean

let notSure: any = 4;

notSure.ifItExists(); // okay, ifItExists might exist at runtime

let list: any[] = [1, true, "free"];

list[1] = 100;

**Void**:

void is a little like the opposite of any: the absence of having any type at all. You may commonly see this as the return type of functions that do not return a value:

function warnUser(): void {

console.log("This is my warning message");

}

**Null and Undefined**:

In TypeScript, both undefined and null actually have their own types named undefined and null respectively.

let u: undefined = undefined;

let n: null = null;

you can assign null and undefined to something like number.

**Object**:

Object is a type that represents the non-primitive type, i.e. anything that is not number, string, Boolean, symbol, null, or undefined.

declare function create(o: object | null): void;

create({ prop: 0 }); // OK

create(null); // OK

create(42); // Error

create("string"); // Error

**JSX** is an embeddable XML-like syntax. It is meant to be transformed into valid JavaScript, though the semantics of that transformation are implementation-specific. JSX rose to popularity with the React framework, but has since seen other implementations as well. TypeScript supports embedding, type checking, and compiling JSX directly to JavaScript.

In order to use JSX you must do two things.

* Name your files with a .tsx extension
* Enable the jsx option

\*\*\*\_\_\_\_\_\_\_\_\_\_\_\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

React:

Name your files with a .tsx extension.

For class files should I use .ts or .tsx files for .jsx. JSX is optional and not required to use React.

It's kind of a convention to use x in the end when your JavaScript

Install Type Script in VS In "Get Tools and Features" under the Tools menu.

Video React CRUD <https://www.youtube.com/watch?v=noX75vnFrYc&list=PLN1yQDPWR2SMcMsO7QzMCC_g_sYr4K76Q>

Video CRUD with Web API <https://www.youtube.com/watch?v=noX75vnFrYc&list=PLN1yQDPWR2SMcMsO7QzMCC_g_sYr4K76Q>

CRUD api + js + code dotnetcore <https://dzone.com/articles/aspnet-core-crud-with-reactjs-and-entity-framework>

CRUD Employee: <http://ankitsharmablogs.com/asp-net-core-crud-with-react-js-and-entity-framework-core/>

<https://www.c-sharpcorner.com/article/my-first-react-application-with-asp-net-mvc/>

<https://www.c-sharpcorner.com/article/fetching-data-using-web-api-in-react/>

For MVC with React install NuGet “React.Web.Mvc4” package ReactJS.NET with ES6.

also install NuGet "Web.Optimization.React"

For ASP.NET Core, install React.AspNet instead! Please refer to project site (http://reactjs.net/) for more details

Add the required CDN files on schtml view.

Add FirstReactApp.jsx.

React is all about modular components.

We need to add HelloWorld.jsx file to create a new bundles.add() inside the bundleConfig file to compile our JSX to JavaScript and then minify it.

React.Web.Mvc4 supports the use of Microsoft's ASP.NET Bundling and Minification library to transform JavaScript via Babel and minify it along with all your other JavaScript.

App\_Start => BundleConfig.cs

**React Bootstrap** module implements all of the Bootstrap 3 components as React components, so they can be easily embedded into your application. It doesn't depend on jQuery,

npm install react-bootstrap

Others:

**Azure Monitor log queries**:

Azure Monitor logs are built on Azure Data Explorer, and Azure Monitor log queries use a version of the same Kusto query language.

Azure diagnostic logs vs activity:

An activity log is a report in which all the recorded azure resources events are sequentially ordered and displayed.

Diagnostsic log is helpful in investigating the application or APIs issue like any errors against the http request or headers etc. Can we use with diagnostic, Application Insight.

**Kusto query** is a read-only request to process data and return results

// works for Azure Log Analytics queries comments

<https://docs.microsoft.com/en-gb/azure/azure-monitor/log-query/get-started-queries?toc=%2Fazure%2Fazure-monitor%2Ftoc.json>

requests

| where timestamp > ago(12h)

| extend messageId = customDimensions['Request-message-id']

| where **messageId** == "b460af7e-c5e1-4744-8e5d-5f5dd9988be9"

**requests**

**| where StartTime >= datetime(2007-11-01) and StartTime < datetime(2007-12-01)**

**| where name has "admissions"**

**// comments**

**| where State == "FLORIDA"**

**| count**

**| take 10**

**| sort by TimeGenerated desc**

**Git branching strategies / contribution models** :

Trunk based development, Git Flow, GitHub Flow

In VS Code you can format your JSON document using Shift+Alt+F or Format Document from the context menu.

**Tip**: If you need any common code handler or helper for any class (className.ExtensionMethod()) use extension for that class like IServiceCollectionExtensions, DbContextExtensions, HttpClientExtensions, HttpResponseMessageExtensions, LinkExtensions for services.RegisterDbContext in Startup.css, DbContext, Http etc.

**Symbol servers** enable debuggers to automatically retrieve the correct program database (.pdb) files, also called symbol files without knowing product names, build numbers, or package names. When Visual Studio is pointed to the UNC share, it can find the symbols related to the binaries that are currently loaded. You can publish symbols to the symbol server in Azure Artifacts (azure devops).

**Ambient transactions** are defined as transactions that live in the current thread or Db context that anybody interested can query for their existence and use them to do work as part of them. An ambient transaction is one that works at the thread level. Thus, all operations that occur in that context will be part of the transaction.

**Kibana**, runs on top of Elasticsearch and is used primarily for analyzing log messages. Kibana is just a visualization tool and all data is stored in Elasticsearch.

**Amazon Elasticsearch Service** (Amazon ES) is a managed service that makes it easy to deploy, operate, and scale Elasticsearch clusters in the AWS Cloud. Elasticsearch is a popular open-source search and analytics engine for use cases such as log analytics, real-time application monitoring, and clickstream analysis.

**bcp Utility**: The bulk copy program utility (bcp) bulk copies data between an instance of Microsoft SQL Server and a data file in a user-specified format. The bcp utility can be used to import large numbers of new rows into SQL Server tables or to export data out of tables into data files.

public class ReferencesGenericRepository<TEntity> : IReferencesGenericRepository<TEntity>

where TEntity : class

{

public IQueryable<TEntity> GetAll()

{ ThrowExceptionIfOperationIsNotAllowed();

return DbContext.Set<TEntity>().AsNoTracking(); }

public async Task<TEntity> GetByCode(string code)

{ var result = this.GetAll();

var codeColumnName = DbContext.Model.FindEntityType(typeof(TEntity)).FindPrimaryKey().Properties.Select(x => x.Name).Single();

return await result.Where(e => (string)e.GetType().GetProperty(codeColumnName).GetValue(e) == code).SingleOrDefaultAsync();

} }

**GraphQL** is an open-source data query and manipulation language for APIs, and a runtime for fulfilling queries with existing data. GraphQL was developed internally by Facebook in 2012 before being publicly released in 2015

**Expression Bodied** =>technique can be used to define method in a single statement.

An expression body definition has the following general syntax:

memberName with Signature => expression;

// C# 5

public bool IsSquare(Rectangle rect)

{ return rect.Height == rect.Width; }

public bool IsSquare(Rectangle rect) **=>** rect.Height == rect.Width;

// Method that call another method

public void called() => Display();

**Get the remote Git** branch to local Git and create a local branch from your local remote branch

**git checkout features/features\_sprintX** (this will fetch the remote branch from cloud create a local origion branch and also a local branch and checkout it)

To confirm that you got the latest right branches check history for all the branches sto match.

git checkout features/tcsi-next

**Prune/Cleanup** the local references to remote branch:

git remote **prune** origin --dry-run

git remote prune origin

Azure Data Studio (similar to SSMS) is a new cross-platform tool that runs on macOS, Linux, as well as Windows.

**Code Coverage**: on the projects we want to see code coverage values

Run unit tests.

On the "Test Explorer" select passed unit tests, right click => "Analyze Code Coverage for Selected Tests".

You should see code coverage for your assemblies in the "Code Coverage Result window".

Just open the file/class and in Code Coverage Result window click on the **icon "Show code coverage colouring"**. Pink colour meaning not being covered.

**Abstract Class**: cannot be instantiated. An abstract class is used to inherit in the child class example is base class.

An abstract class can have a **static method**, which can be called directly, even if you do not have an instance of the abstract class.

**sAbstract methods** have no actual code in them, and child/subclasses have to override the method.

**Virtual methods** in an abstract class can have code, which is usually a default implementation of something, and any subclasses can override the method using the override modifier and provide a custom implementation.

It is not necessary to override virtual method in derived but it can be.

An abstract class cannot be inherited by structures. It can contains constructors or destructors. It can implement functions with non-Abstract methods. Abstract method declarations are only permitted in abstract classes.

It’s opposite to sealed class, the sealed class can’t be inherited while the abstract class can only be inherited by child classes (can’t have instances).

**Load/Performance/Stress testing tool:** We can use Soap UI open source or Pro version, Apache JMeter which is open source,

Microsoft included performance testing in their Visual Studio Enterprise till VS 2019.

Load Testing in Azure DevOps/Azure Test Plans in cloud.

Netling is by Tore Lervik and is a nice little load tester client for easy and quick web testing. It's open source and on GitHub.

In load testing, if with only 2 threads you get database deadlocks, we need to tune/create index on database tables using SQL Server profiler and Database Engine Tuning Advisor (for Indexes by default or non-clustered indexes/ index views).

The First type of testing normally done is Functional testing, which ensures that the application is working as expected.

In load testing we need to monitor the CPU, Memory and Network traffic/load.

We need to pick the small, medium and largest API endpoints like number of validations/db hit required.

Choose random test with say 5 threads/users.

Keep in mind the future projections/sate and how often load is expected.

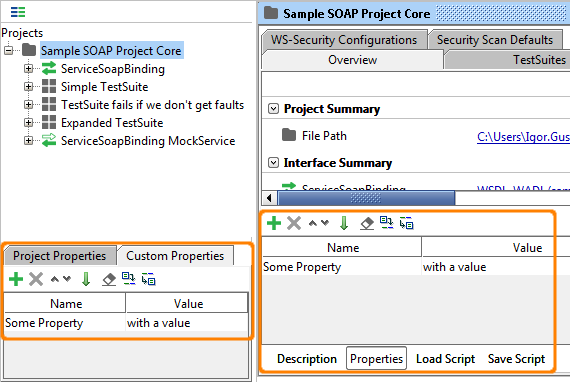
Have some long running load tests like running for number of days and check the error log file maintained etc.

Use Azure App Insight or Dynatrace etc.

In Soap UI properties can be used to hold the endpoints, authentication credentials of your services, to transfer and share session ids during test execution so multiple test steps or test cases can share the same sessions.

Properties are defined at several levels in soap UI Project, Test Suite and Test Case Properties.

For example a global password might be stored at the project level and accessed inside a request message with a standard property expansion: ${#Project#Password}.



Dynamic Properties: write groovy scripts directly inside a Property Expansion;

${=(int)(Math.random()\*1000)}

will be replaced with a random number between 0 and 999 every time it is evaluated.

${=request.name}

If you want to provide a current formatted timestamp:

${=import java.text.SimpleDateFormat ; new SimpleDateFormat("YYYY-MM-DDT00:00:00").format(new Date())}

**Improve SQL Server performance using profiler and Tuning Advisor**:

In SQL Server profiler select **Tuning Template** and can use filters. For filters go to Event Selection => Column Filters => Select Database Name and put name in like.

After running save the workload file as trace file on your disk.

Go to Tools => Select Tuning Advisor => find the same workload file/trace file => select Database => Click on start Analysis (green paly button)

If see any error fix the errors and rerun. To fix most of the error got to Tuning Options tab => Advance Options

You can also click one query from profiler, copy and paste into SQL Server Management Studio and test it. If work well, then call Database Tuning Advisor from Management Studio => Tools and tune the query.

Run tuning advisor to check weather Indexes are appropriate or not.

Create Statistics:

You can create query optimization statistics on one or more columns of a table or indexed view in SQL Server

When particular queries do not run very often, you can select to create column-level statistics rather than an index. Column-level statistics help Query Optimizer find better execution plans

**SQL Server query execution plans**:

Query item in the SQL Server Management Studio menu, you’ll see two options related to query plans – Display Estimated Execution plan and Include Actual Execution plan

Second option: Start SQL Server Profiler => In the File menu, select New Trace => In the Events Section tab, check Show all events => Expand the Performance node => Select Showplan XML

The SQL Server query plan is shown in the lower pane. It’s the same plan as shown when the Include Actual Execution Plan option is selected. You can see its details in the tooltip that appears on mouse over or save the whole trace as an XML file for later analysis.

**Windows performance counters** helps you analyse the performance data that applications, services, and drivers provide.

**APIM:** In APIM we can have Revisions (non breaking changes) or versions without effecting the end users after our testing we can enable the production/end-user’s version/revisions.

Can have mocked operations before the actual implementation like the UI developer can start using the mocked endpoint until the actual implementation.

Can be imported/deployed by Open API/Swagger or from as Azure App Service or WISDAL Soap service.

In code we can generate open APIs by Swash Buckle nugget package.

In APIM we can have products like starter/ unlimited/ guest with API policies restriction like number of request per minute/day or data amount.

In APIM we can directly test our API endpoints. Otherwise we can use separate Developer portal.

People need to sign-up for a product key to use Developer portal.

App Insight can be used to monitor APIs health (any 404/500 errors/ service fabric connection failure). Application map, Telemetry, error logs, queries etc.

\*\*----

**Imp Documentation**: <https://docs.microsoft.com/en-au/azure/devops/?view=azure-devops>

DevOps: <https://azure.microsoft.com/en-us/services/devops/>

<https://app.pluralsight.com/library/courses/tfs-visual-studio-2017-continuous-delivery-devops/table-of-contents>

Go to VSTS/ DevOps sportal => [www.visualstudio.com](http://www.visualstudio.com) => sign in =>

You can add new Azure DevOps Organization and create a project to get started.

Otherwise you can add/create a project in an existing Organization (if you are owner or member with enough right)

You can delete any organization or projects in that organization. Just select the organization => Organization Settings => Delete organization

Organization name could be Department + Branch/TeamNames

[**https://dev.azure.com**/Organization001TeamAbc/](https://dev.azure.com/Organization001TeamAbc/)

**Create new project =>** Project name => Private access => Version control Git/ Team Foundation Version Control =>

We can use Boards, Repos, Pipelines, Test Plans or Artifacts services in that project otherwise to keep it simple we can turn off some of the services in project settings for that project.

**Project Settings**: is the most important place to manage most of the things like adding new teams with existing security groups like Contributors, Build Administrators, Project Administrators, Readers etc. We can add groups or users in teams. We must add more than one Project Administrators.

Security tab: We can create new groups or teams like help group, code reviews, support etc. A group can have different members and team can have different groups.

Notifications tab: set notification for **Builds** (fails/completes), **Code Git** (commit pushes, pull request created/updated etc.), **Code TFVC** (code is checked in, a file with a specific extension/path is checked in, any code review changes), **Work** (work item created/changed/deleted/restored, or work item moved from this team project), **Release** (an approval for deployment is pending/completed/a manual intervention for a deployment is pending).

We can do team level or custom email address etc.

**Under Boards** tab we have other tabs like project configuration for boards (we can set Iterations with start and End dates). We can select days. Note we can add children like PI-1/2/3 (program increment). And each PI can have Sprint 1/2/3 and set start or end dates. (note: please set dates at sprint level instead of PI level).

DevOps is all about Boards, Repos, Pipelines, Test Plans and Artifacts

You can add existing project to Git/VSTS => open in VS => Right click the solution in Solution Explorer => Add Solution to Source Control.

In Team explorer => changes => click + (Add staging) => Commit (commit Staged). This is for local commit.

Otherwise just right click the solution and select Commit.

To push/commit changes to the server/origin => Team Explorer => Home => Sync => Push (Outgoing Commits)

You can again open project in the browser and do some changes & commit.

In VS Team Explorer => Home => Sync => Pull (in case of conflicts use Fetch and merge) or (get latest/pull for master/main branch switch to current branch and do rebase master in case of conflicts)

Rebase from master => git checkout master => git pull => git checkout {users/username/BranchNamedescription} or git checkout @{-1} s

=> git rebase master => git rebase **–continue** (resolve all conflicts) => gits push **--force-with-lease**

If you get lost at any stage during rebase just abort/undo => git rebase **--aborts**

**Builds/Automated Builds**:

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

Once we have code (Repos => Files/branches) in DevOps we can setup CI & CD using Pipelines.

DevOps => Repos => Files => Set up build (right top corner)

VSTS (older system) => Code => Files => Set up build (right top corner)

Or Go to DevOps => Pipelines => Builds => New pipeline => select code source like Azure Repos Git => Select a repository => Select a Template (e.g. ASP.Net core).

Note: we can always import an existing build/pipeline .yaml. go to pipeline => view => download/copy .yaml file and convert it to json format (past yaml <https://editor.swagger.io> and convert to json). Go to pipelines => builds => New => “Import a pipleline”.

You can export to json (if enough rights) go to Pipeline => select pipleline/build => … top right corner => Export

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

Releases: We publish/deploy project in releases (Dev/test/Pre Prod/Production) using release pipeline.

First we do CI/Continues Integration setup and then we can do publishing/CD in the same pipeline (by adding a task). But usually CD/Publishing is done in separately from artefacts (produce by CI).s

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)

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

You can download all logs

Build variables are used as $(variableName)

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

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.

<https://www.youtube.com/watch?v=yNESSInONyk>

<https://app.pluralsight.com/player?course=getting-started-visual-studio-team-services-2018&author=ben-day&name=7b010b04-fcc8-4359-b03d-1d8282bfa6d2&clip=1&mode=live>

++++++++++++++++++++++

To use Azure Pipelines, you need

1. An organization in Azure DevOps.
2. To have your source code stored in a version control system.

Release pipelines: <https://docs.microsoft.com/en-us/azure/devops/pipelines/release/define-multistage-release-process?view=azure-devops>

**Release pipelines:**

**Prerequisites for Release pipelines**: Deploy to an Azure Web App//Azure DevOps Project/ Deploy to IIS web server on Windows

Two separate targets like virtual machines, web servers or Azure App Services website instances etc. with unique names like QA and Production.

you can Use the Azure portal to create two new web app names QA and Production.

**Configure the triggers in your release pipeline**:

To add new Release pipelines go to Releases Tab => click New pipeline => Select a template (Azure App Service deployment or Azure Service Fabric deployment) => Enter Stage Name (Dev/Test/Pre Prod) => click **Add an artefact** => Select source type (Build/ Azure Rep/Git Hub/TFVC) best is select **Build** => Select form Project dropdown => Select from **Source (build pipeline)** => Source (build pipeline) is usually \_ build pipeline name => click Add.

1. In Azure Pipelines, open the Releases tab. Select your release pipeline and, in the right pane, choose Edit.
2. You may choose the Continuous deployment trigger icon in the Artifacts section to open the trigger panel. Make sure this is enabled so that a new release is created after every new successful build is completed.
3. Choose the Pre-deployment conditions icon in the Stages section to open the conditions panel. Make sure that the trigger for deployment to this stage is set to After release. This means that a deployment will be initiated automatically when a new release is created from this release pipeline.

**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)

**Pipelines**:

Build, test, and deploy Node.js, Python, Java, PHP, Ruby, C/C++, .NET, Android, and iOS apps. Run in parallel on Linux, macOS, and Windows.

Easily build and push images to container registries like Docker Hub and Azure Container Registry.

Explore and implement a wide range of community-built build, test, and deployment tasks, along with hundreds of extensions from Slack to SonarCloud.

Implement continuous delivery (CD) of your software to any cloud, including Azure, AWS, and GCP.

Support for YAML, test integration, release gates, reporting.

<https://dev.azure.com>

**Documentation**: <https://docs.microsoft.com/en-au/azure/devops/?view=azure-devops>

**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)

**Pull Requests**: After you set up a branch policy, you cannot directly push changes to the branch (master branch). Changes to the branch are only made through pull requests.

In some cases, you need to bypass policy requirements so you can push changes to the master branch directly or complete a pull request even if branch policies are not satisfied.

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

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 Security**” => opens dialog

Manage users and their access in Azure DevOps:

<https://docs.microsoft.com/en-us/azure/devops/organizations/accounts/manage-users-table-view?view=vsts&tabs=new-nav>

add users to your organization and specify the level of features they can use, such as Basic or Stakeholder.

Go to <https://dev.azure.com> => Organization settings => Users=> Add new users.

Select a user or group of users. Then, select the ... icon at the end of the Name column to open the context menu.

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

+++++++++++++++++++++++++++++++++++++++++++++++++++

**Release pipelines CD**:

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

Azure Pipelines has many out-of-the-box tasks to deploy a variety of applications. Otherwise consider using Shell scripts or PowerShell scripts.

It helps in monitoring and tracking for instance it tracks whether an issue fixed by a developer, or a product backlog item completed by your team, has been deployed to a specific stage.

Control of the deployments. It let you specify which users can change the configuration of a stage, or approve the release to be deployed into a particular stage.

It also provides a history of all the activity performed during each deployment.

You can roll back to a previous deployment, and provide all the logs in one place to help you debug the problem.

The agent downloads all the artifacts specified in that release, then runs all the tasks in the deployment job to deploy the app to the target servers for a stage.

Quick Start:

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

<https://docs.microsoft.com/en-us/azure/devops/pipelines/apps/cd/deploy-webdeploy-webapps?view=vsts>

Prerequisites: need a CI build and an Azure Web App where you will deploy the app.

CD release pipeline picks up the artifacts published by your CI build and then deploys them to your Azure web site.

1. Do one of the following to start creating a release pipeline:

* If you've just completed a CI build (see above), choose the link (for example, Build 20170815.1) to open the build summary. Then choose Release to start a new release pipeline that's automatically linked to the build pipeline.
* Open the **Releases tab in Azure Pipelines**, (**new pipeline**) open the + drop-down in the list of release pipelines, and choose Create release pipeline.

1. The easiest way to create a release pipeline is to use a template. Select the **Azure App Service Deployment template**. Then choose Apply.
2. If you created your new release pipeline from a build summary, check that the build pipeline and artifact is shown in the Artifacts section on the Pipeline tab. If you created a new release pipeline from the Releases tab, choose the + Add in Artifacts link and select your build artifact.
3. Choose the Continuous deployment **icon** in the Artifacts section, check that the continuous deployment trigger is enabled, and add a filter to include the master branch. Enabling the trigger will create a new release every time a new build is available.
4. Open the Tasks tab and, with Stage 1/Dev/Test selected, configure the task property variables as follows:
5. Azure Subscription: Select a connection from the list under Available Azure Service Connections or create a more restricted permissions connection to your Azure subscription. If you are using Azure Pipelines and if you see an Authorize button next to the input, click on it to authorize Azure Pipelines to connect to your Azure subscription. If you are using TFS or if you do not see the desired Azure subscription in the list of subscriptions, see Azure Resource Manager service connection to manually set up the connection.
6. App Service Name: Select the name of the web app from your subscription.

NOTE: Some settings for the tasks may have been automatically defined as stage variables when you created a release pipeline from a template. These settings cannot be modified in the task settings; instead you must select the parent stage item in order to edit these settings.

1. Save the release pipeline

Create a Release to deploy your app:

* Choose + Release and select Create a release.
* In the Create a new release panel, check that the artifact version you want to use is selected and choose Create.
* Choose the release link in the information bar message. For example: "Release Release-1 has been created".
* In the pipeline view, choose the status link in the stages of the pipeline to see the logs and agent output.
* After the release is complete, navigate to your site running in Azure.

Pre-deployment conditions: In Pipeline => release => select and edit release => Stages column/group mouse hover => click on Pre-deployment conditions **icon**.

You can enable/select Manual only deployment and also enable **Pre-deployment approvals** with Approvers user names

Customize web app deployment : <https://docs.microsoft.com/en-us/azure/devops/pipelines/targets/webapp?view=vsts&tabs=yaml>

We can define variables in Variables tab next to Tasks tab and use in Tasks like $(database.name).

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

Azure SQL Database Deployment Task: <https://docs.microsoft.com/en-us/azure/devops/pipelines/tasks/deploy/sql-azure-dacpac-deployment?view=vsts>

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

we can use Dacpac, SQL script file or inline SQL script,

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.

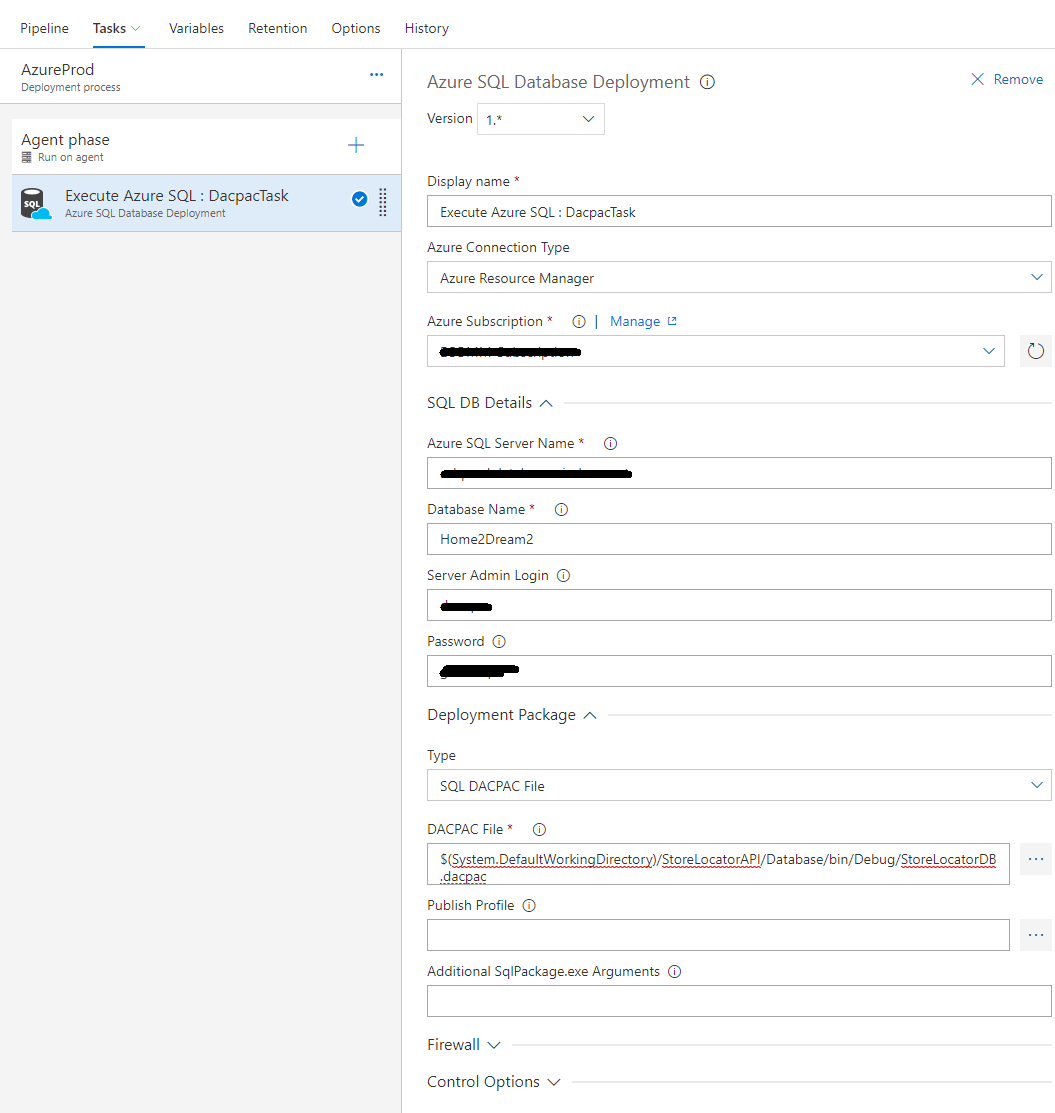
Tips:

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>

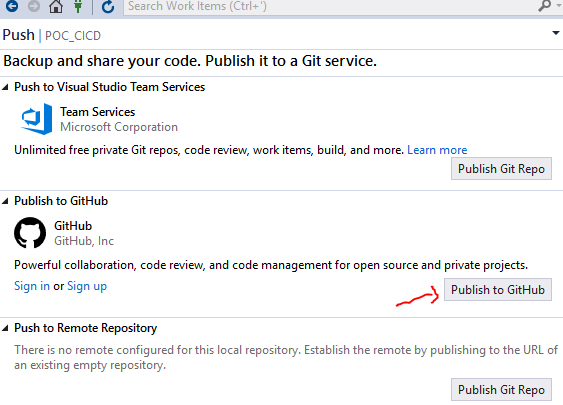
**CI CD Steps detailed with Git branches (GitHub – not complete)**:

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.

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 => on 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.

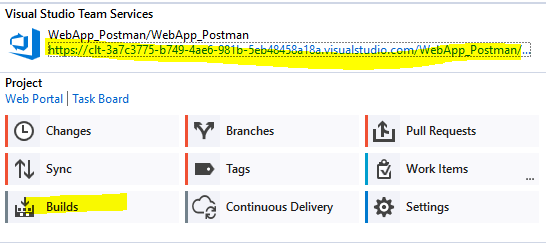
NOTE: The task **does not work with the Azure Classic service endpoint** and it will not list these connections in the parameters in the task. Select create a **Resource Manager service connection**.

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

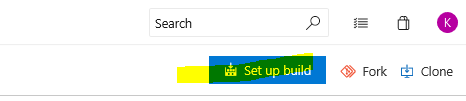
To deploy to Azure, an Azure subscription has to be linked to Team Foundation Server or to Azure Pipelines using the Services tab in the Account Administration section.

s

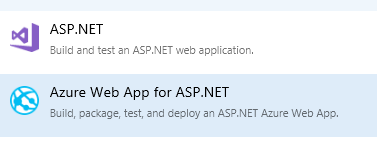
Create/Add code to Git repository



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.

**For CD/Release**:

Go to <https://portal.azure.com> search Web App => Create => give a name => wait until web app gets ready.

Select the created web app resource.

Come back to VSTS => Builds => select a successful build => Select **Create Release** => Choose **Azure App service Deployment** template => Apply => Give a name to Environment like test/staging/production.

In the Environments box click on 1 task link => choose Azure subscription => Authorize => chose App service name from dropdown.

From right side => Select Deploy Azure App Service => on right side scroll down to **Package or folder** select … and select the application folder under drop => Save => we can now run it by clicking on **Release** drop down and Create.

Click on Logs to see the progress.

You can select the deploy menu to trigger again the Release.

Pre-release approval like a user approval required.

**Build the Database Project on VSTS**:

Create a SQL Database on Azure: Go to Azure portal => Create a resource => databases => SQL Database

Connect using SSMS: we need the fully qualified server name, database name, and login information.

Select SQL Databases from the left-hand menu, and click your database on the SQL databases page.

Click overview => Server name

If you forget your server login information, navigate to the SQL Database server page (click on the server link in Database page) to view the server admin name. If necessary, reset the password.

On SQL server page find Reset admin password (right top)

Create a build definition that will create the DACPAC package.

build definition must have a task for “Build Solution” that will compile and package the database project in a DACPAC file and other projects in their appropriate form. It should also have a task to publish that DACPAC in an artifact on the hosted build agent. As you will observe, the value of Contents parameter is “\*\*\\*.dacpac”. These settings will ensure that any file that has extension .dacpac and is in any folder under the Build Directory on hosted agent will be published in the artifact named “DACPAC”.

**Selenium UI Tests**:

Use Nuget package and to load drivers use GetCurrentDirectory() to get the path. Other option is copy the Selenium drivers in the UI test project and change the property to copy always.

string startupPath = System.IO.Directory.GetCurrentDirectory();

string startupPathOpt2 = Environment.CurrentDirectory;

IWebDriver driver = new InternetExplorerDriver(startupPath);

driver.Navigate().GoToUrl("http://www.google.com");

Gated Check-in, Gated Builds

Branch policy

Centralized and Distributed Source Control Systems

Artifacts, Environments, and Gates

Branch, Merge, and Rebase

Main, Supporting, and Hotfix Branch

Manage access levels for different users

Artifact: Artifact is the published output of the build

+++++++++++++++++++++++++

You can publish to an Azure App Service using Visual Studio or to an IIS without CI CD (using VSTS).

VSTest task can be used to run tests on Build agent machines. Apart from MSTest based tests, you can also run tests written using test frameworks like NUnit, xUnit, Mocha, Jasmine, etc. using the appropriate test adapters to Visual Studio. The task uses vstest.console.exe to execute tests and the command-line options.

We can enable code coverage via a test task in build pipeline e.g. Visual Studio Test.

Create work item on failure of a build as bug.

++++++++++++++

For manual push/build from CI to PR (merge to master) go to pipeline => builds => “**ITS-Apps-Code-PR**” and Queue build. If successful it’s read to publish.

Note: PR build definition will soon be deleted.