Azure DevOps – Continuous Integration (CI) for SharePoint Development

Sep 2019

Revision History

Version	Name	Revision description	Date
1.0	Manjunath Poola	Final Version	10/09/2019

Table of Contents

1. Introduction

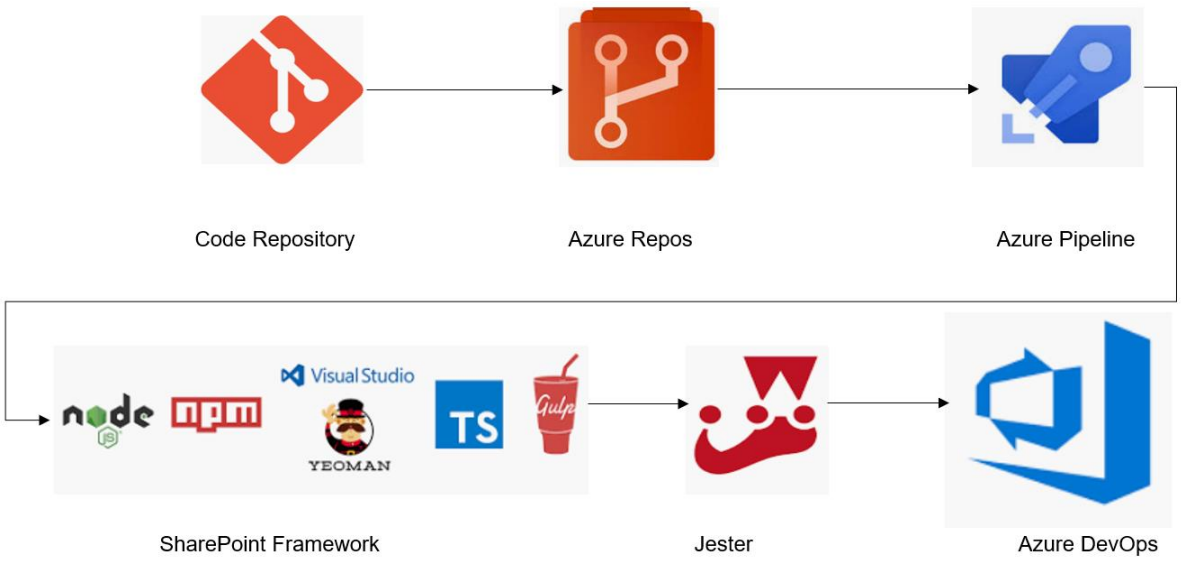
Azure DevOps (Visual Studio Team Services/Team Foundation Server) can be implemented for development projects where SharePoint Framework is used on O365.

Azure DevOps consists of tools that help developers to implement Continuous Integration, Continuous Development methodologies in their projects.

This article provides details to setup Azure DevOps with Continuous Integration (CI) to automate SharePoint Framework builds, unit testing and implementation.

1.1. Continuous Integration (CI)

Developers can integrate code into a shared repository by automatically verifying the build , unit tests and package the solution using Continuous Integration(CI), whenever there are new updates to code is submitted to repository.



Continuous Integration using DevOps.

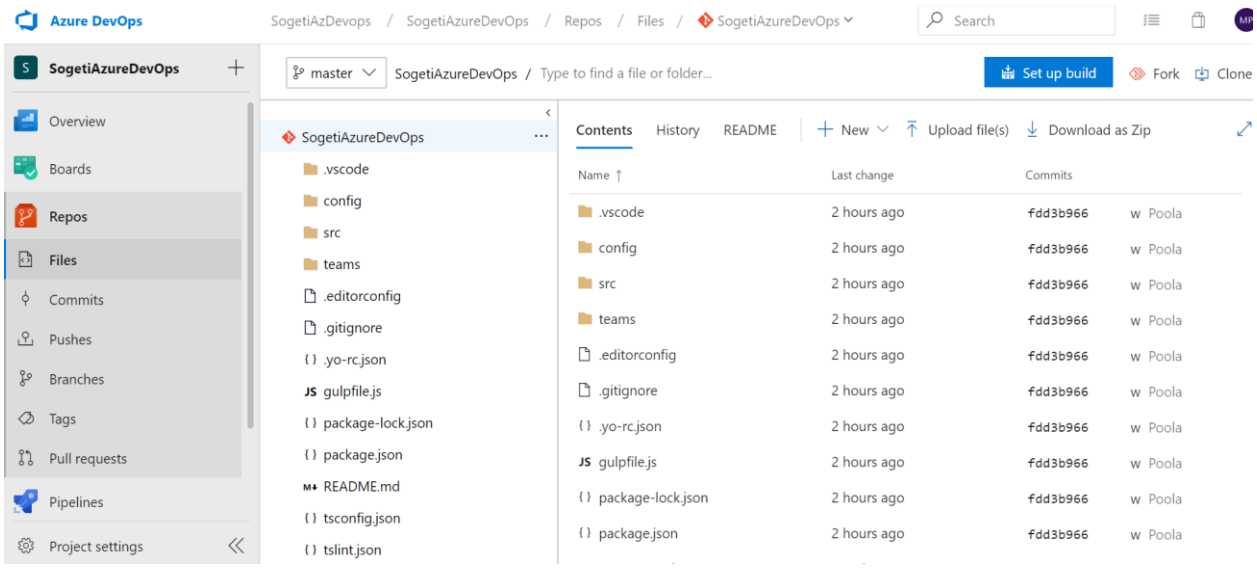
The steps involved to setup Azure DevOps for SharePoint Framework development are :

1. Build Definition Creation
2. Installing NodeJS and dependencies.
3. Execution Unit Test
4. Import Test results
5. Import code coverage information
6. Bundle the solution
7. Package the solution
8. Prepare the artifacts
9. Publish the artifacts

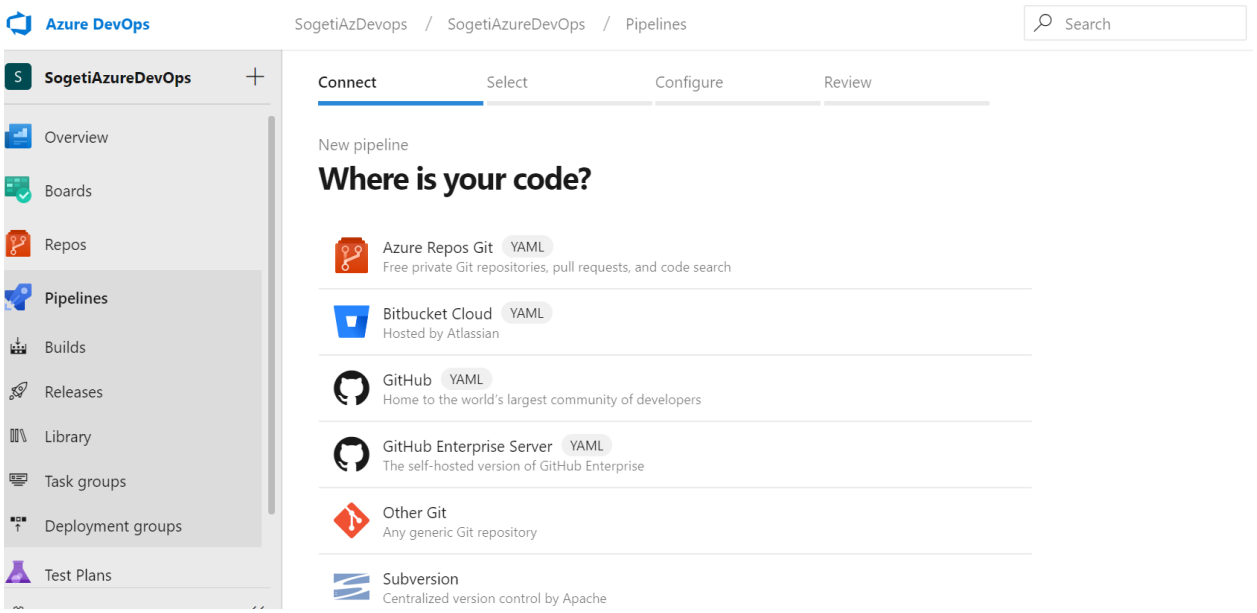
1.1.1. Build Definition Creation:

Build definitions can be defined as a process template.It is a set of configured tasks that need to be executed in a sequential manner on the source code every time a build is triggered.
Tasks can be grouped in phases, by default a build definition contains at least one phase.we can add new tasks to the phase by clicking on the big plus sign next to the phase name.

The repository can be in GitHub, and the files needs to be visible in Repos. The below screenshot provides snapshot of a sample Repos.



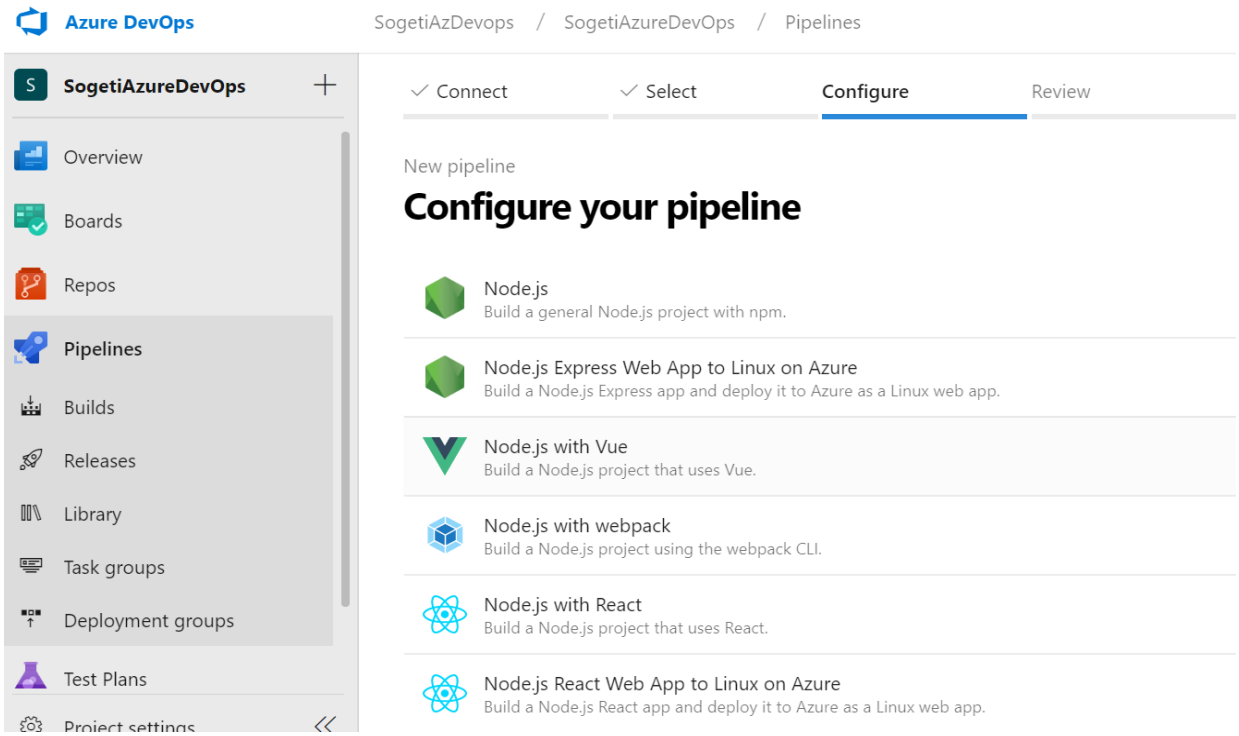
We can start Continuous Integration by creating new build definition and linking the same to repository.



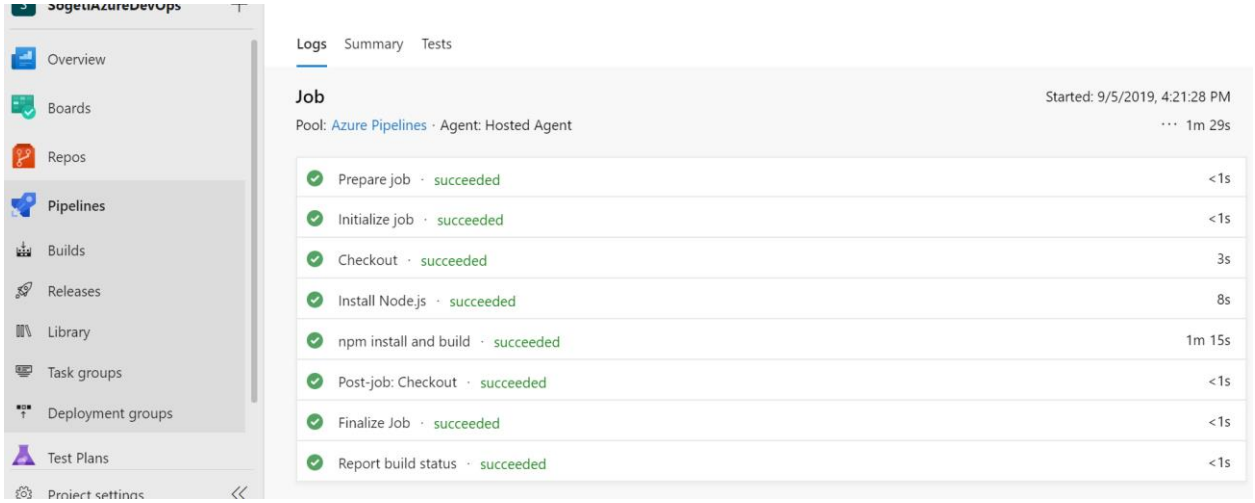
We can select Azure Repos Git option from the above screenshot, and link the repos to the Pipeline,

1.1.2. Installing NodeJS and dependencies.

Once the Build Definition has been created, we need to install NodeJS and related dependencies. We need to ensure to install version 10, as SharePoint Framework depends on it.



Once the installation is completed, the results are displayed as per the below screenshot.



1.1.3. Executing Test Cases:

The SharePoint Framework (Ver 1.8.0) doesn't provide a testing framework by default. We can leverage JEST# for creating Unit test cases. It is highly recommended at a minimum to test the business logic of the code to get feedback on any potential issues or regressions as soon as possible.

To have Azure DevOps execute your unit tests, add a **npm** task. Set the **command** to custom and **custom command** field, enter **test.** Then set the **Working Directory** option to **$(Build.SourcesDirectory).**

Jest is an open JavaScript testing library from Facebook. Its slogan is "Delightful JavaScript Testing". Jest can be used to test any JavaScript library especially with React and React Native

Configuring Jest

By default SharePoint Framework projects does not include a testing Framework. We can leverage Jest in this sample.

Install Jest by executing the below command -

```
npm i chai@4.X jest jest-junit @voitanos/jest-preset-spfx-react16 -D
```

To configure Jest, create a file **config/jest.config.json** and add the following content.

```
{
  "preset": "@voitanos/jest-preset-spfx-react16",
  "rootDir": "../src",
  "coverageReporters": [
    "text",
    "json",
    "lcov",
    "text-summary",
    "cobertura"
  ],
  "reporters": [
    "default",
    "jest-junit"
  ]
}
```

We need to configure the project to leverage jest when typing commands.
To configure we need to edit the **package.json** file and add/replace these two scripts with the following values:

```
"test": "./node_modules/.bin/jest --config ./config/jest.config.json",
"test:watch": "./node_modules/.bin/jest --config ./config/jest.config.json --watchAll"
```

We need to modify **package.json to** configure the Reporter.
Reporters are plugins that provide new export formats for test results to test runners. To do so edit **package.json** and add these lines after the **scripts** property.

```
"jest-junit": {
    "output": "temp/test/junit/junit.xml",
    "usePathForSuiteName": "true"
  }
```

Writing a unit test

For creating the first unit test, create a new file:
src/webparts/<webPartName>/tests/<webPartName.spec.ts> and add the following content:

```
/// <reference types="mocha" />
import {assert, expect} from 'chai';

describe("webPartName", () => {
  it("execute something", () => {
    assert.ok(true, 'should be true');
  });
  it("should add numbers Sync fluent", () => {
    const result:number = 1 + 3;
    expect(result).to.eq(4); // fluent API
  });
});
```

1.1.4. Importing test results

In order to get test results information attached with the build results, we need to import these test results from the test runner into Azure DevOps.
The following steps needs to be configured:

- Add a new **Publish Test Results** task.
- Set the Test results files field to **temp/test/junit/junit.xml**
- Set the **Searchfolder** to **$(Build.SourcesDirectory).**

1.1.5. Importing code coverage information

To get code coverage reported with the build status we need to add a task to import code coverage details. To configure the code coverage information, add the task - **publish code coverage results.** Ensure that we set the tool to:

- **Cobertura**, **Summary files** to **$(Build.SourcesDirectory)/temp/test/cobertura-coverage.xml**
- **Report Directory** to **$(Build.SourcesDirectory)/temp/test.**

1.1.6. Bundling the solution

We need to bundle the solution to get static assets that can be understood by a web browser.
Add another gulp task, set the **gulpfile** path, set the **Gulp Tasks** field to bundle and **add --ship in the Arguments.**

1.1.7. Packaging the solution

Now we have the static assets, the next step is to combine the assets into a package SharePoint will be able to deploy.

Add another **gulp** task, set the **gulpfile** path, set the **Gulp Tasks** field to **package-solution** and add **--ship** in the **Arguments**.

1.1.8. Preparing the artifacts

By default, an Azure DevOps build doesn't retain any files. To ensure that the required files needed for the release are retained, we need to explicitly indicate which files should be retained.
Add a **Copy Files** task and set the **Contents** to *****.sppkg** (the SharePoint Package created with the previous task) and the target folder to **$(build.artifactstagingdirectory)/drop.**

1.1.9. Publishing the artifacts

We have now collected all the files needed for deployment in a special artifacts folder, we still need to instruct Azure DevOps to keep these files after the execution of the build.

To do so add a **Publish artifacts** task and set the

Path to publish to **$(build.artifactstagingdirectory)/drop** and **Artifact name** to **drop.**

2. Conclusion

Building and Testing are the main activities of the continuous integration task. Continuous Integration helps to automate the build process when development of a solution by the project team, and when development cycles are undergoing continuous changes. Azure DevOps helps to automate the development builds of SPFx solution.

Some of the technical benefits of CI are :

- Helps to execute the test cases using the central code repository, and this helps to spot any integration issues.
- Better code control, and code management

3. References

S.No.	Reference
1.	AzureDevOps for CI/CD Implementation

About Sogeti

Sogeti is a leading provider of technology and engineering services. Sogeti delivers solutions that enable digital transformation and offers cutting-edge expertise in Cloud, Cybersecurity, Digital Manufacturing, Digital Assurance & Testing, and emerging technologies. Sogeti combines agility and speed of implementation with strong technology supplier partnerships, world class methodologies and its global delivery model, Rightshore®. Sogeti brings together more than 25,000 professionals in 15 countries, based in over 100 locations in Europe, USA and India. Sogeti is a wholly-owned subsidiary of Capgemini SE, listed on the Paris Stock Exchange.

Learn more about us at www.sogeti.com