B9IS121 - Networks & System Administration

A Report on

CI/CD PIPELINE USING AZURE DEVOPS

By

Shastri Telikicherla (10606539)

TABLE OF CONTENTS

1. ACRONYMS

CI: Continuous Integration

CD: Continuous Delivery/Continuous Deployment

DevOps: Development & Operations

.NET: Network Enabled Technology

MVC: Model View Controller

QA: Quality Assurance

API: Application Programming Interface

IaaS: Infrastructure as a service

PaaS: Platform as a service

SaaS: Software as a service

2. Introduction

DevOps is a set of practices that combines software development and IT operations to simplify the system development lifecycle and optimize the communication by providing CI and CD pipelines to build, test and deploy software to production on regular iterations like agile development lifecycle.

For the successful implementation of a software project, a continuous and improved collaboration is always needed between business, developers, and IT operations.

While agile processes like scrum, Kanban etc. are customer oriented and focuses on optimizing communication between end users and developers, DevOps is an internal practice within IT teams that optimizes communication between developers and operations team and covers software development, deployment, and maintenance as well.

This document represents a detailed report of the project “CI and CD pipeline using Azure DevOps”.

Patrick Debois originally coined the term “DevOps” in 2009. He is a Belgian consultant, agile practitioner, and project manager, who became one of the early DevOps leaders and formed this word by combining “Dev” as in development, and “Ops” as in operations.

Here, “Dev” doesn’t only mean developers, but everyone involved in developing a software product from disciplines like Q&A, development, testing, planning, etc. Similarly, “Ops” is an umbrella term for everyone involved in the operations team, including system engineers, database administrators, system administrators, security experts, network engineers, release engineers, operations staff, and others.

DevOps is a mindset or culture. And this mindset is not something that was carved out of a single rock.

DevOps practices affect developers throughout the software development life cycle:

- Developers must verify a system's provenance upon initialization. This verification determines whether the system has gone through the requisite gates with the requisite approvals.

- One practice is continuous deployment. A developer can place code into production without coordinating with members of other development teams. This affects the design choices and the overarching architectural style.

- Systems move through various environments on their way to production. This affects the use and management of configuration parameters.

- Systems are monitored after deployment, and changes might be rolled back. This affects the architectural style, the information that's exposed, and how that information is exposed.

In addition, DevOps practices rely heavily on tools of various kinds, including tools for container management, continuous integration, orchestration, monitoring, deployment, and testing. Increasingly, software engineers are the ones who maintain and configure such tools. In some organizations, such as Netflix, Google, and Amazon, they also develop those tools, whereas most organizations use existing tools. (Author et al., 2016)

3. Background Details

This project depicts the use case of an ecommerce website called *Fitness Factory* which is a web application built on .NET core MVC and is deployed on Microsoft azure app service. There are different teams involved in the complete lifecycle of this website like stakeholders, development, QA, operations etc. This project follows agile methodology for development and hence requires frequent UAT and Production releases.

Traditional Development and Operations Process:

Below diagram shows the different servers through which the code changes for this website undergo before reaching on to the production.

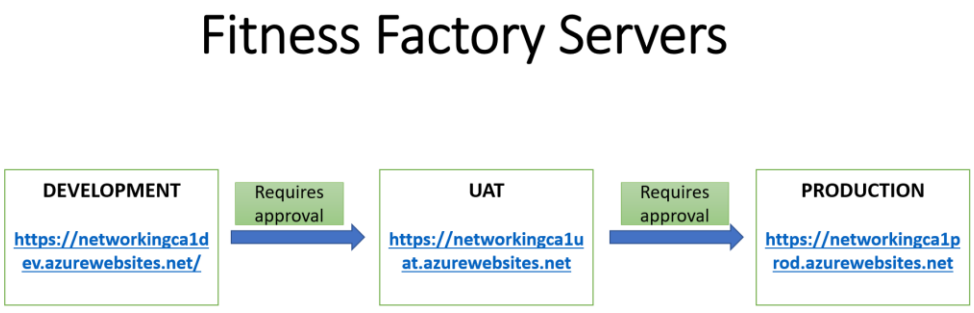

Figure1: Represents different servers for fitness factory website

A typical software development project following agile methodology involves multiple developers working parallelly on different user stories in a sprint and during the

development stage there is a continuous build and deployment on the development server.

At the end of the sprint a final build is performed in the development server and after getting the required stakeholder approval the code dependencies are extracted from the development server and are handed over to the operations team along with the deployment instructions who further move them on to the UAT server.

Once the changes are deployed on the UAT, the stakeholders perform testing and provide the feedback, if UAT is rejected the code changes are either dropped for re-development in the next sprint or will need an immediate fix and re-release to the UAT depending on the complexity. In case of UAT approval, the codebase is moved further from UAT to production servers.

Generally, the UAT and production deployment involves a downtime unless the web application is hosted on any cloud provider like Azure, AWS etc. where the concept of staging can be used to push the code changes into production without experiencing a downtime.

Process Improvement with DevOps implementation:

In a traditional development and deployment approach, collaboration between the development and deployment teams is essential, else it can lead to confusion and impact the software delivery. Also, this traditional approach is time consuming and involves manual steps throughout the code journey from development to production server.

It is often observed that system administrators and developers disagree on various grounds even though they agree that both must meet the customer expectations. This collaboration gets overwhelmed with the frequently changing and ever-evolving customer demands and in general the need arises to choose one option between dealing with an unstable environment and delivering quick changes and maintaining a highly stable production environment but stale. Both these scenarios are unacceptable as they do not offer a better solution to the end customer.

Hence there is a dire need for a system or practice that could balance both sides of the production environment, development, and operations. This is where DevOps comes to the rescue. By adopting DevOps, Fitness Factory can improve its products, serve the customers well with quick feedback and fixes, get a competitive edge in the market, and achieve the business objectives faster.

Organizations are introducing agile and lean software development techniques in operations to increase the pace of their software development process and to improve the quality of their software. They use the term DevOps, a portmanteau of development and operations, as an umbrella term to describe their efforts. (Author et al., 2017)

DevOps is created in such a way that if the developers design a faster product, operations can find ways to stabilize the system and it can integrate all the people associated with software development to its deployment that includes stakeholders, security engineers, developers, system administrators, QA etc.

Benefits of DevOps:

- High Development Speed
- System Stability
- Scalability
- Security
- Increased customer satisfaction

CI/CD:

It is important to note that CI/CD is not DevOps, and it is one of the critical DevOps practices.

Continuous integration is a software development practice where developers merge or integrate code changes into the primary code branch frequently. It includes automated testing that runs each time a new code is introduced to keep the primary code branch stable.

Next, continuous delivery involves automated deployment of software versions into the chosen production environment. DevOps teams can enable updates more frequently using automated deployment and reduce the number of issues occurring on deployment.

These two practices are together called continuous integration and continuous delivery (CI/CD). It includes complete automation of all the processes from coding to deployment and allows teams to remove operational overheads, human errors, and repetitive steps.

Hence, DevOps teams can emphasize building codes and deploying them quickly with lesser risks. Additionally, you can deploy more rapidly in small increments, become agile, and more confident and productive in running your codes.

There are lots of DevOps products like Azure DevOps, Jenkins, GitLab, Copado CI/CD, AWS CodePipeline etc.

One of the biggest benefits of Azure is the ability for organizations that adopt the public cloud to be exponentially agile. Services and solutions can be delivered in weeks and months instead of years. This accelerated life cycle for product and service delivery is based on agile concepts like continuous integration and continuous delivery (CI/CD). (Author et al., 2020)

Azure DevOps Server is a Microsoft product that provides version control, reporting, requirements management, project management, automated builds, testing, and release management capabilities. Although it is a complete DevOps toolchain, we will be focusing on the build and deployment pipelines.

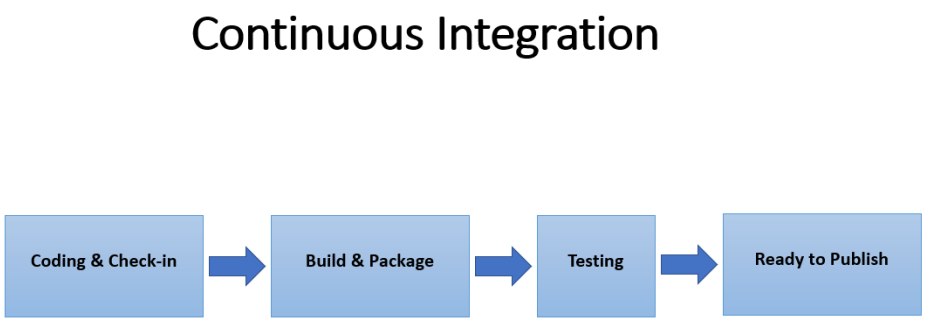

Figure2: Shows the steps involved in continuous integration

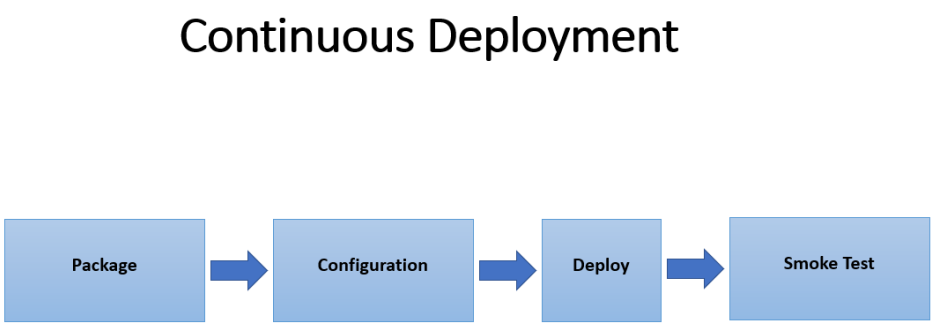

Figure3: Shows the steps involved in continuous deployment

4. Aim of the project:

The aim of this project is to demonstrate how CI/CD pipelines can be built for the Fitness Factory website by using Azure DevOps and show case the benefits build and release lifecycle automation for an application.

5. Technical project components:

Below is the list of various technologies/services and concepts that are used in this project:

- *.NET MVC*

  ASP.NET Core MVC is a rich framework for building web apps and APIs using the Model-View-Controller design pattern. The Fitness Factory website is built on this framework.

  The Model-View-Controller (MVC) architectural pattern separates an application into three main groups of components: Models, Views, and Controllers. This pattern helps to achieve separation of concerns. Using this pattern, user requests are routed to a Controller which is responsible for working with the Model to perform user actions and/or retrieve results of queries. The Controller chooses the View to display to the user and provides it with any Model data it requires. (Smith, 2018)

- *CI/CD pipelines*

  The CI/CD pipelines are created in Azure DevOps for this project which automate the entire lifecycle from development till deployment.

- *Azure Active Directory*

  Azure Active Directory token authentication via service principal is used to authenticate the Azure app service connection from Azure DevOps

- *SSH key authentication*

  Azure DevOps repo communicates from git via SSH key authentication. An SSH key is a secure access credential used in the Secure Shell (SSH) protocol. SSH keys use key pairs based on public key infrastructure (PKI) technology, the gold standard for digital identity authentication and encryption, to provide a secure and scalable method of authentication.

- *PaaS*

The app services created in Azure for this project are platform as a service offering. Platform as a service (PaaS) is a category of cloud computing services that allows customers to provision, instantiate, run, and manage a modular bundle comprising a computing platform and one or more applications, without the complexity of building and maintaining the infrastructure typically associated with developing and launching the application(s); and to allow developers to create, develop, and package such software bundles.

- *IaaS*

  The user agents in the CI and CD pipelines are Microsoft-hosted agents which run the jobs automatically and are an IaaS offering. Infrastructure as a service (IaaS) is a cloud computing service model by means of which computing resources are hosted in a public, private, or hybrid cloud.

- *SaaS*

  The Azure DevOps platform used in this project is a SaaS offering from Microsoft. Software as a service is a software licensing and delivery model in which software is licensed on a subscription basis and is centrally hosted. SaaS is also known as "on-demand software" and Web-based/Web-hosted software.

6. Project Implementation:

In this section, the step-by-step implementation details are provided:

- ***Created Azure Web App for Dev, UAT and Production environments***

Each resource or service belongs to a unique resource group in Azure, hence created a resource group in the Azure portal:

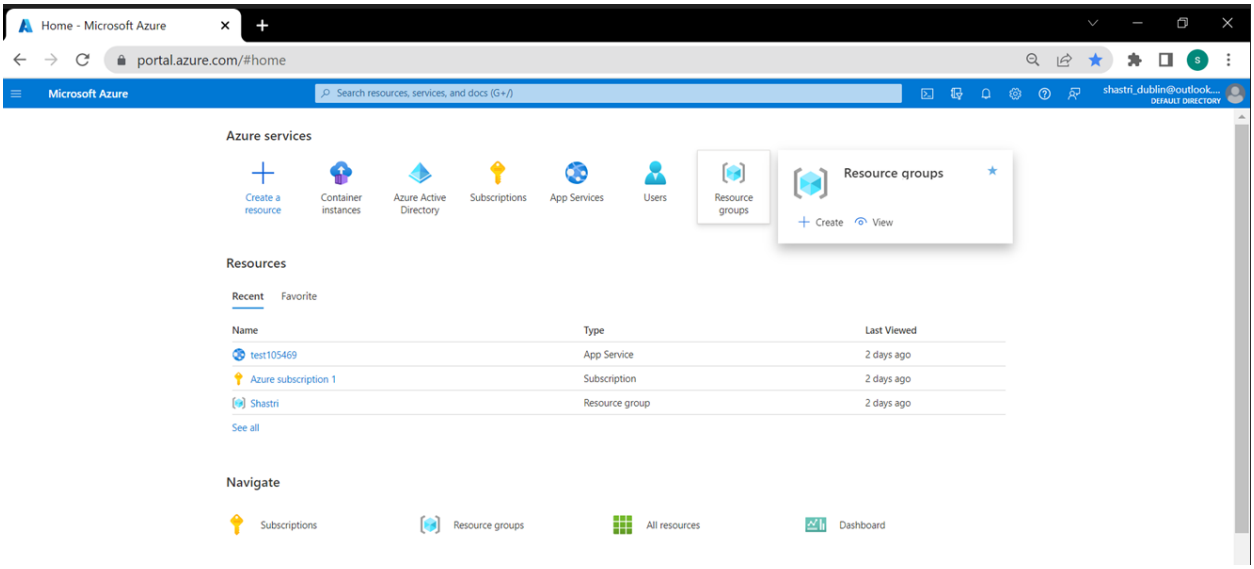

Figure4: displays the option to create resource group in azure portal

Clicked on the App Services to create a new app service or web application:

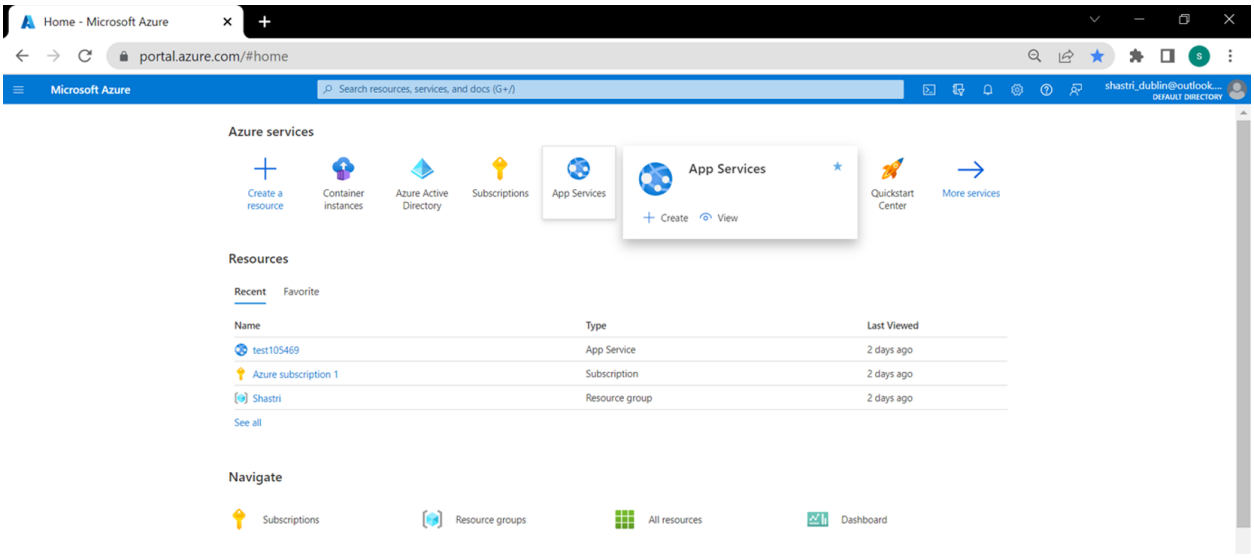

Figure5: displays the option to create app service in azure portal

In create a new app service form, specified the resource group, web application name, selected publish type as code and selected the run time stack as .NET 6 (LTS) as the local code was built on this version, selected the operating system as windows (one can also select Linux) and selected a close by region as per the current location:

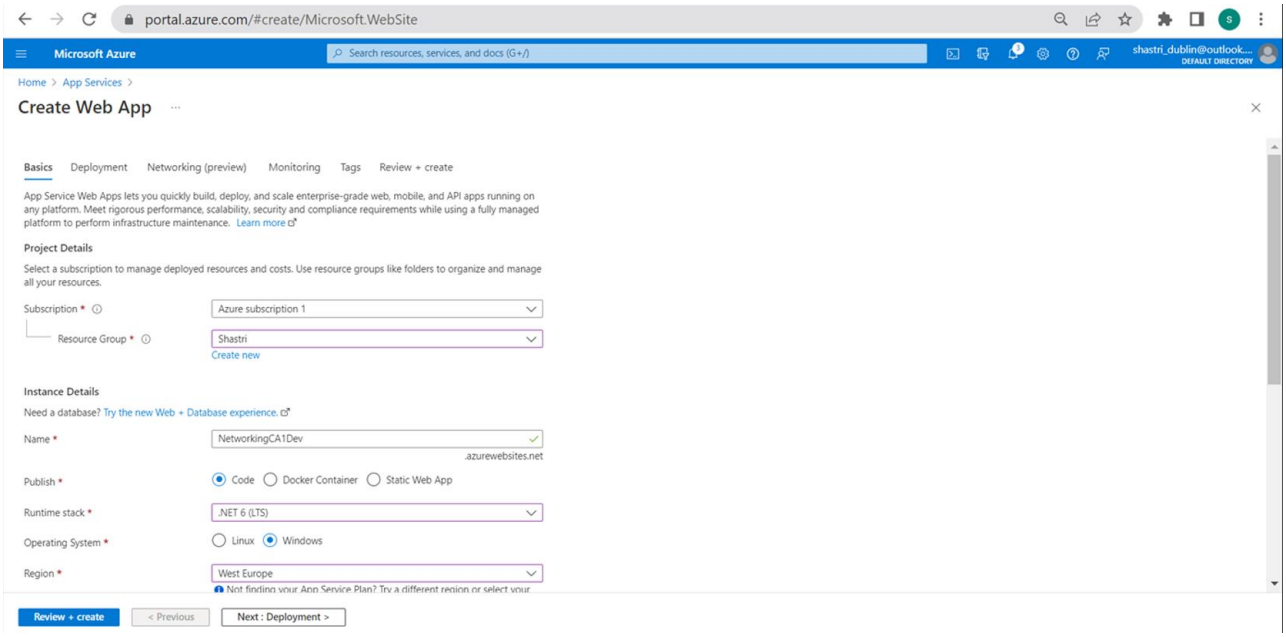

Figure6(a): options while creating app service in azure portal

In the Sku and size, selected a free tier server plan and clicked on create

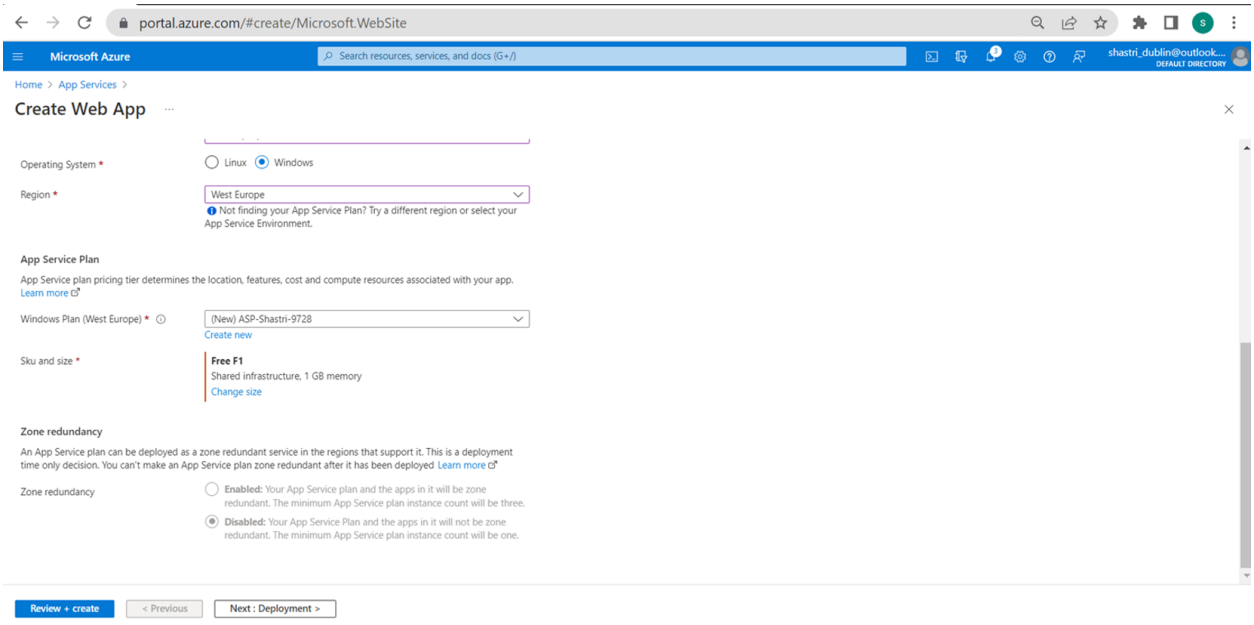

Figure6(b): options while creating app service in azure portal

By following the above process, created the web applications for Dev, UAT and Production environments. There is an option available to add a stage in the production web application, but due to the limitation with the subscription it was not selected.

Adding a stage helps to perform the continuous deployment onto production from UAT via staging without causing a production downtime.

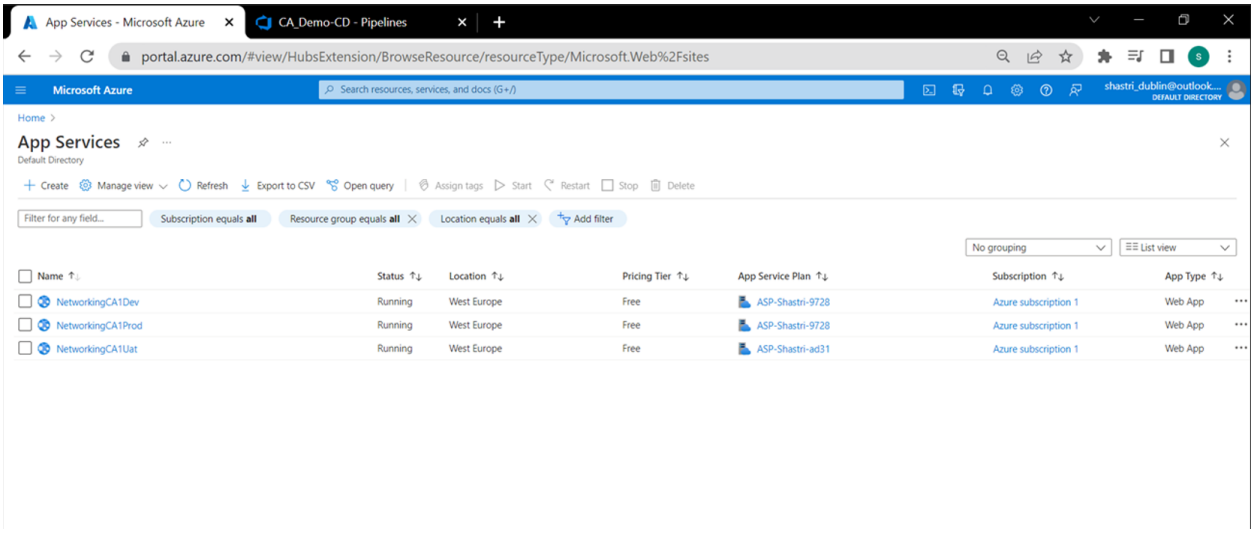

Figure7: lists the web apps related to dev, uat and prod in azure portal

- ***Registered and configured a new Azure DevOps account and raised a request to Microsoft for approval of Azure DevOps parallelism***

- ***Mapped Azure DevOps repo to local system via GitHub desktop***

Clicked on 'Clone repository' option from github desktop:

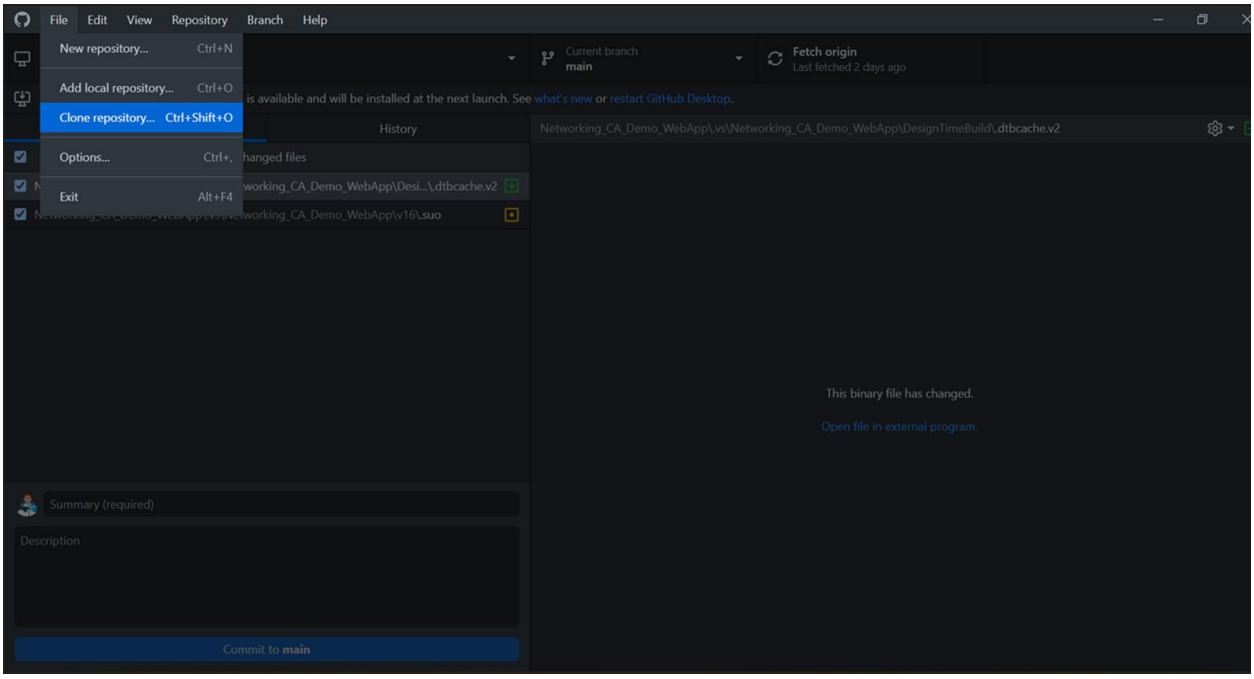

Figure8: displays option in github desktop to clone devops repository to git

Opened the Azure DevOps repos and clicked on generate Git credentials to generate the username and password for this repo

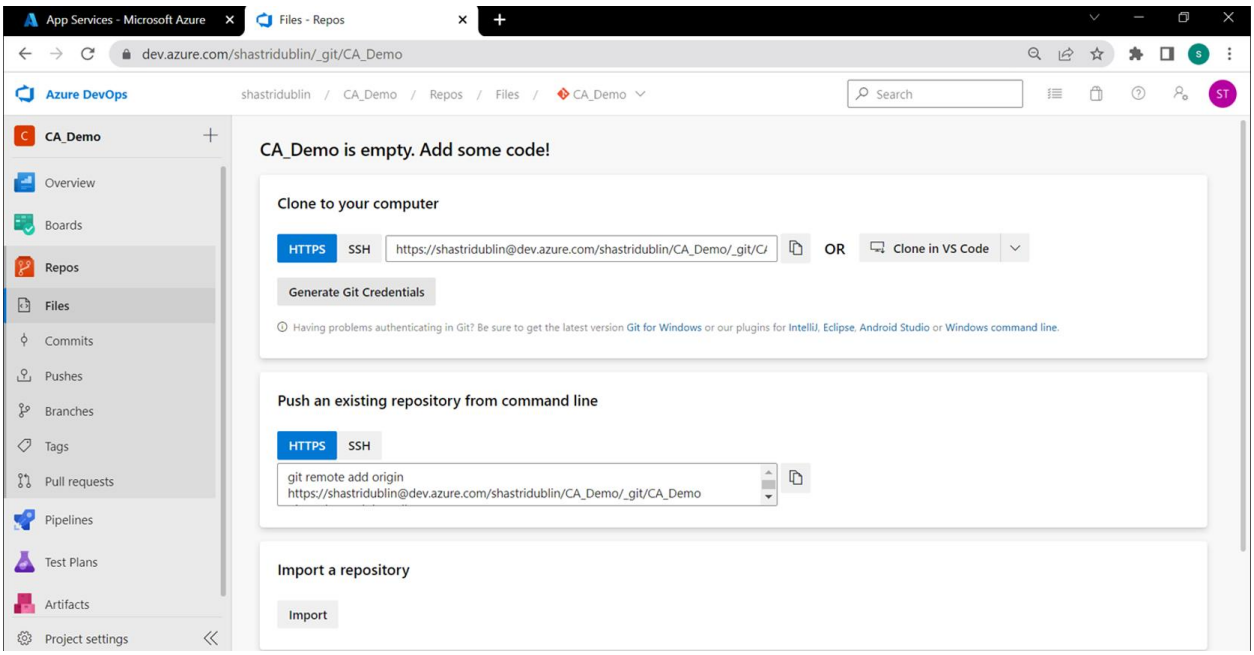

Figure9: displays azure devops repository link to clone devops repository

In the below image we can see the username and an auto generated password:

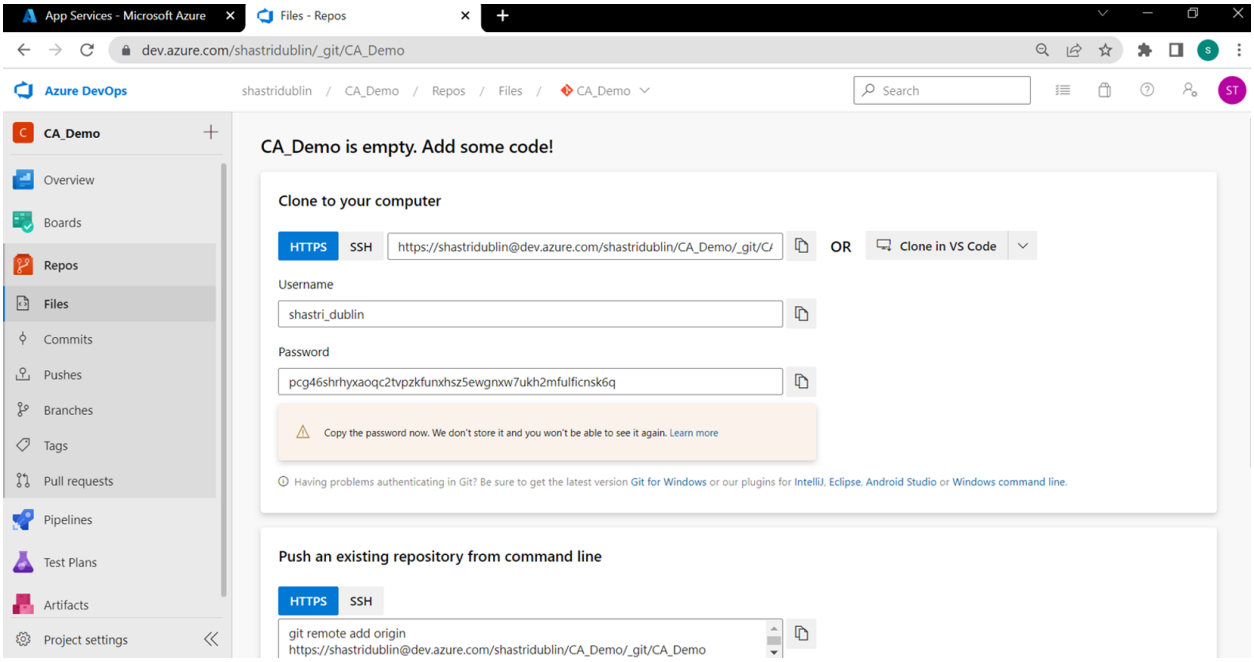

Figure10: displays azure devops repository link to clone devops repository

Entered the repo path and the above generated credentials in GitHub desktop:

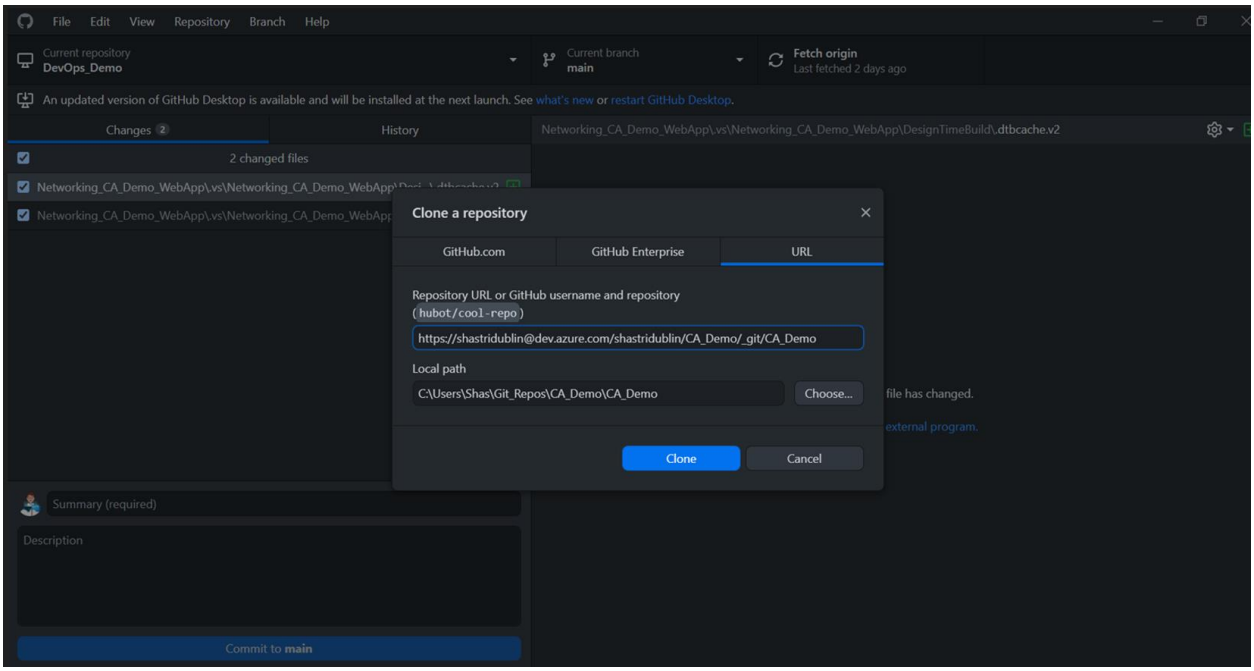

Figure11: displays azure devops repository link to clone devops repository

- ***Performed the code changes locally and checked in to repo from Github desktop***

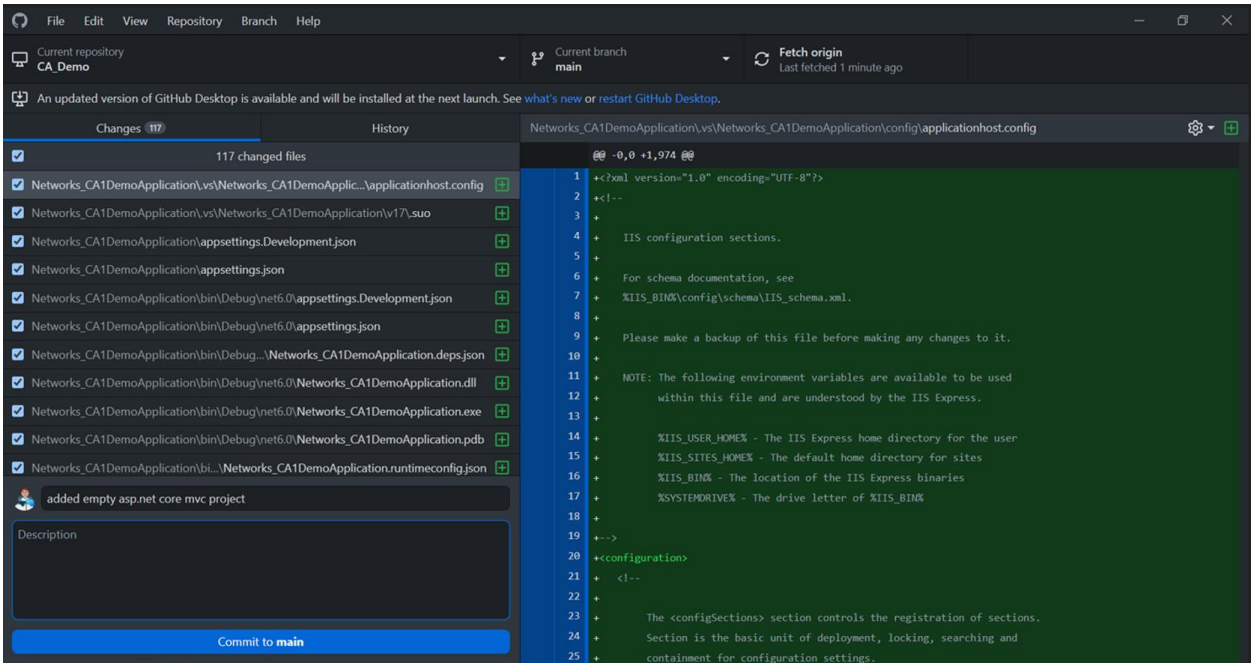

Figure12: displays code commit from git

Verified that the local changes started reflecting on the repo:

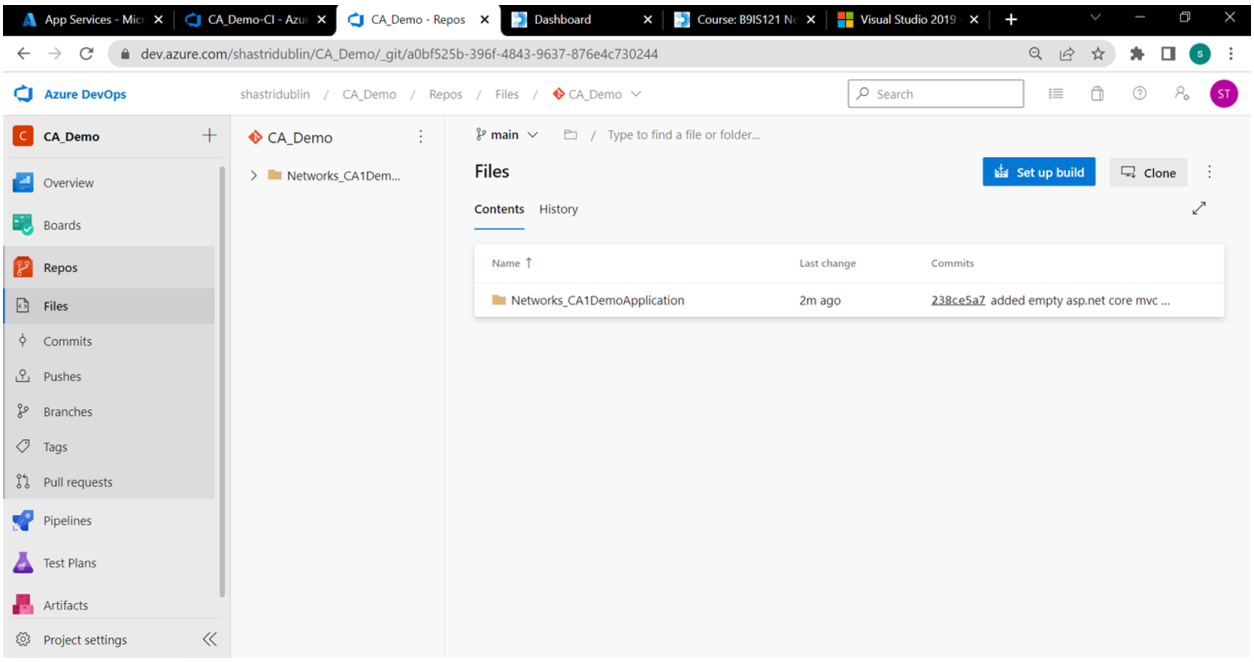

Figure13: displays azure repo parent folder

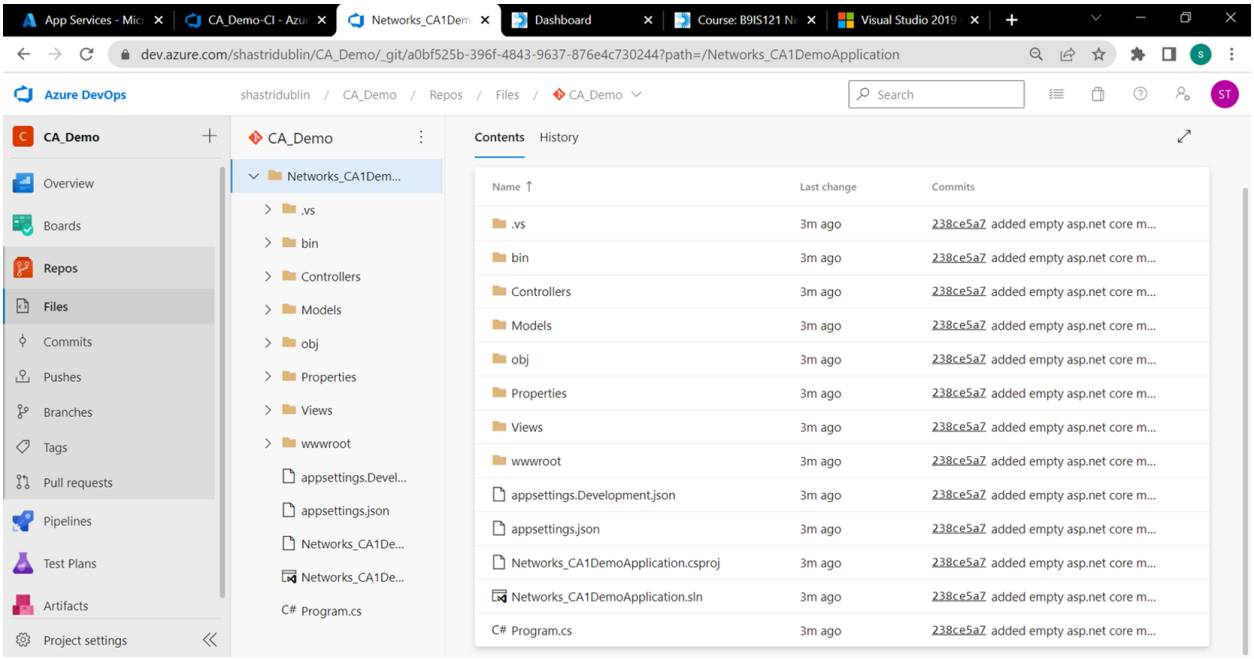

Figure14: displays code files under the azure repo parent folder

- ***Created a CI pipeline and verified the creation of artifacts required for deployment***

Clicked on pipelines > create pipeline

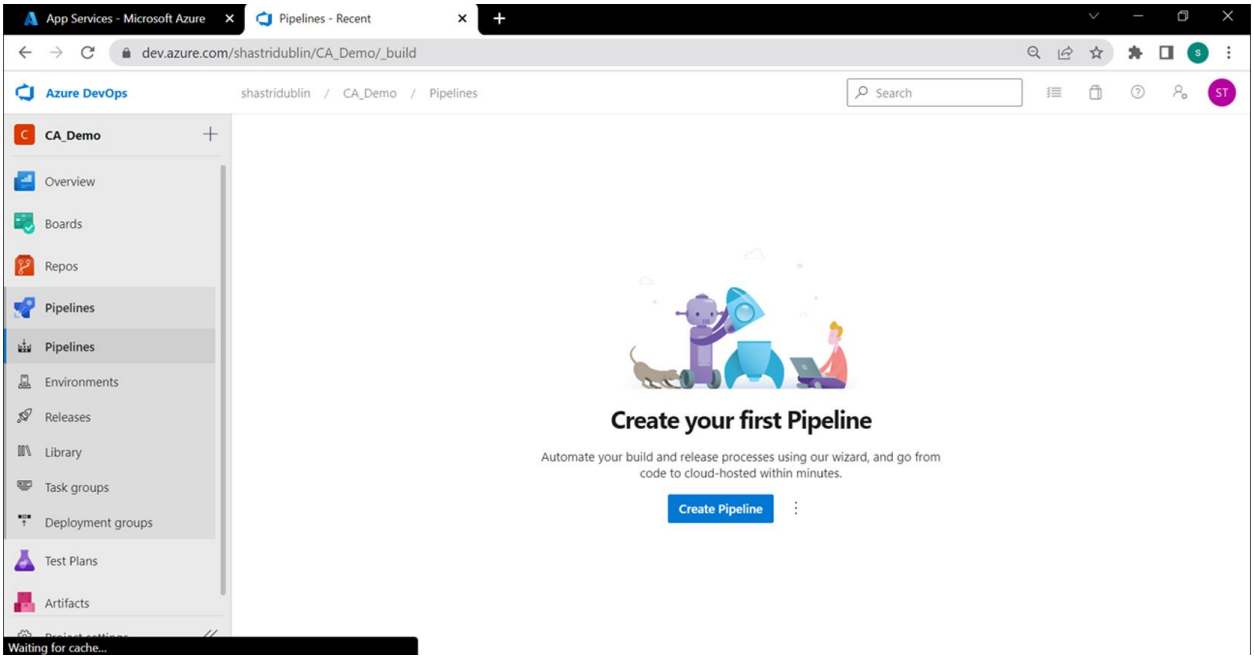

Figure15: displays the CI pipeline page

Clicked on the classic editor:

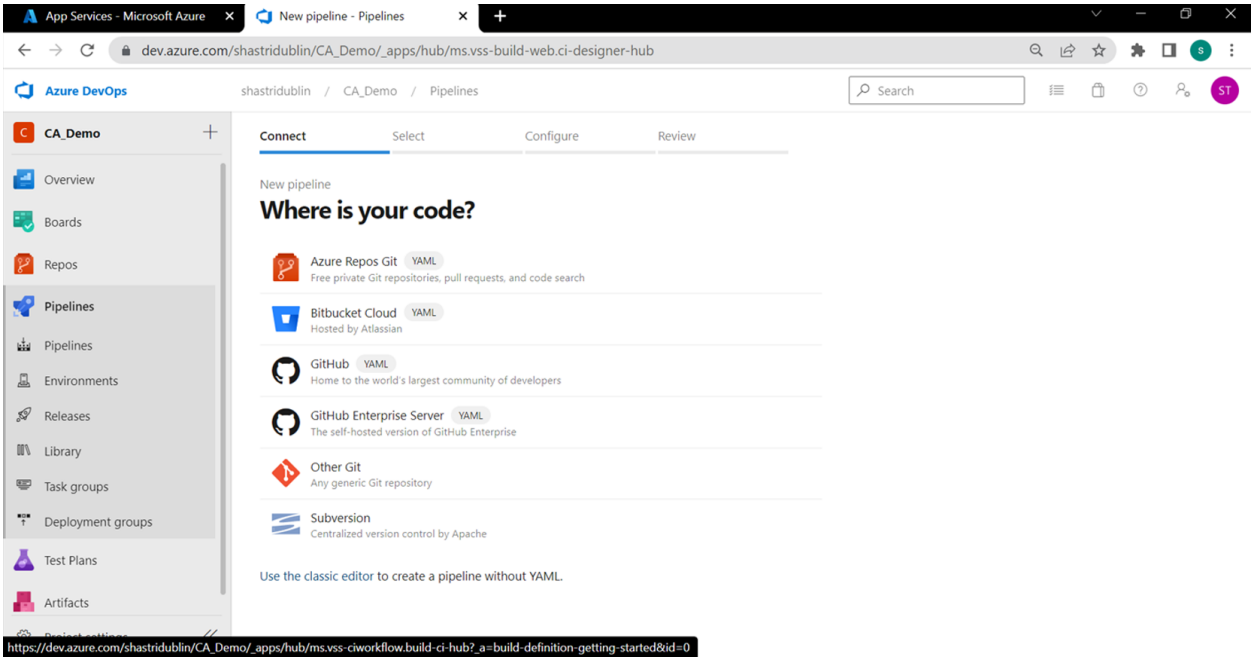

Figure16: display the initial step to create CI pipeline

Selected the source of the code repository:

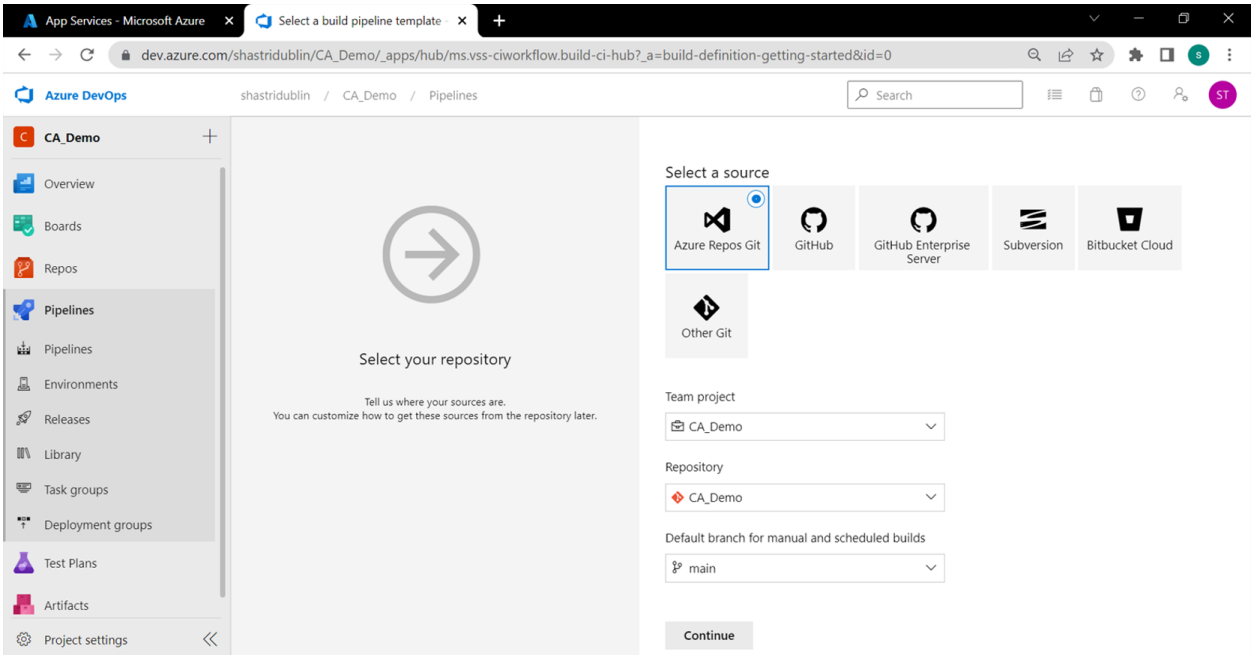

Figure17: displays the source code options in CI pipeline

Clicked on empty job to create a new pipeline (we can also use YAML to write the pipelines):

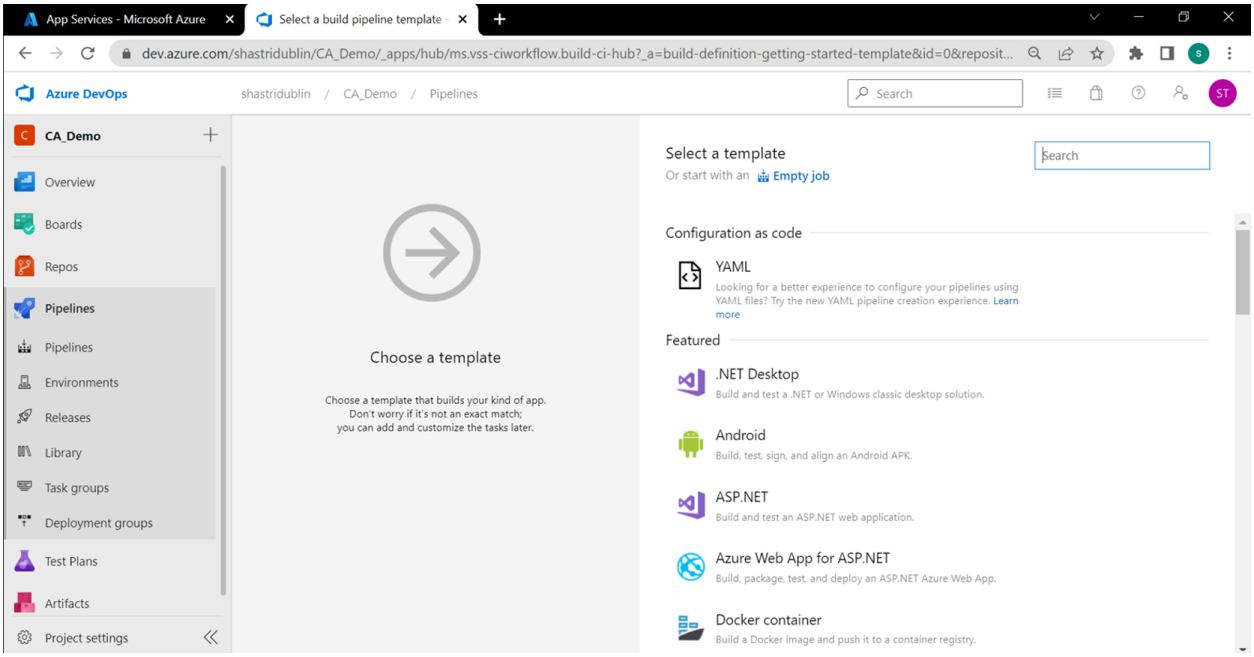

Figure18: displays the options to create a CI template

In pipeline clicked on the + icon next to agent job ('Agent job 1' – it is a computing infrastructure with installed agent software that runs one job at a time)

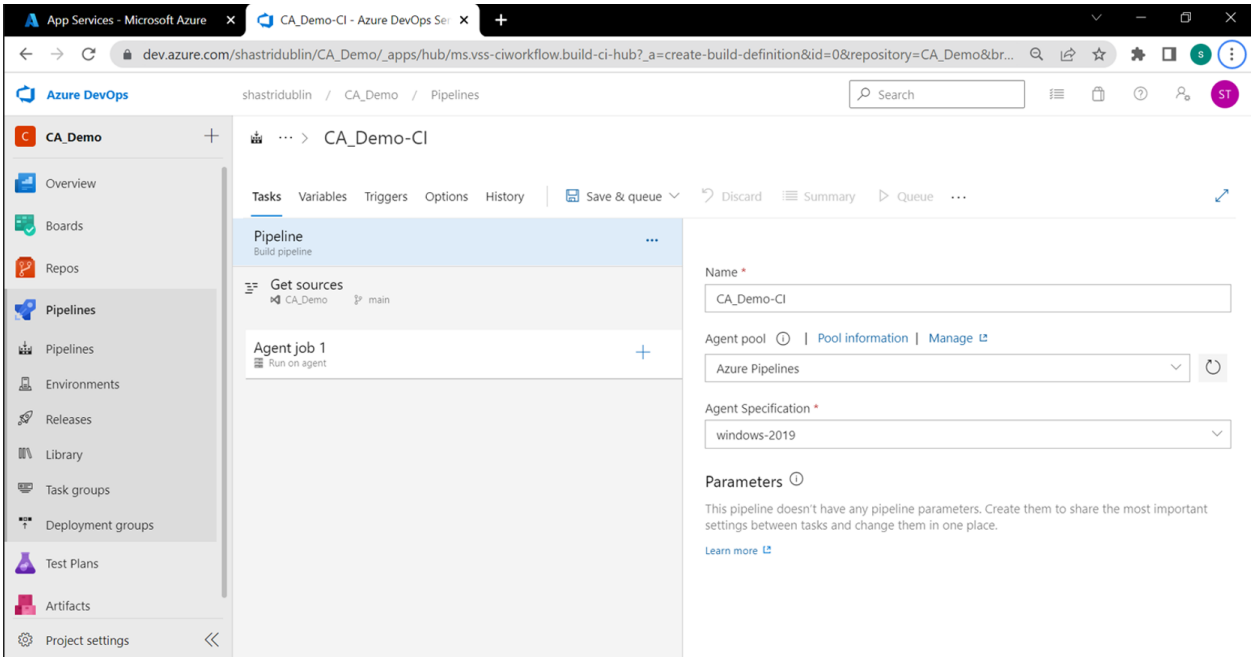

Figure19: displays the CI pipeline

Selected the use .net core option to specify the user agent about the .net version on which it needs to run the code:

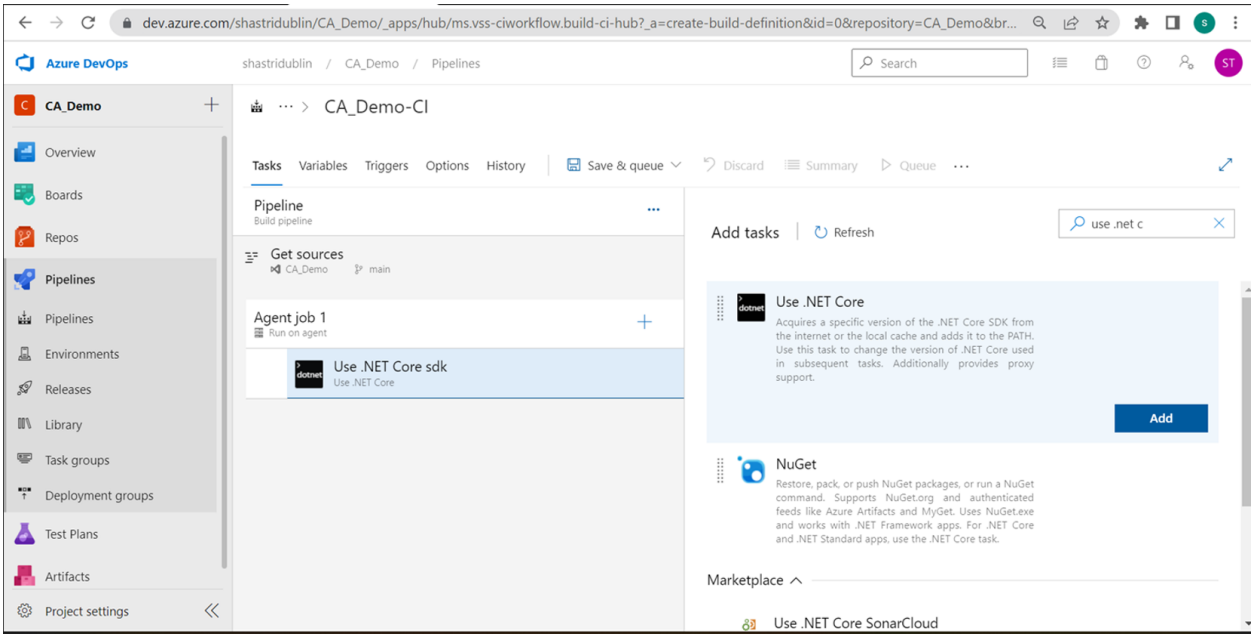

Figure20: displays the use .net core option as a step

Verified the target framework in the project properties:

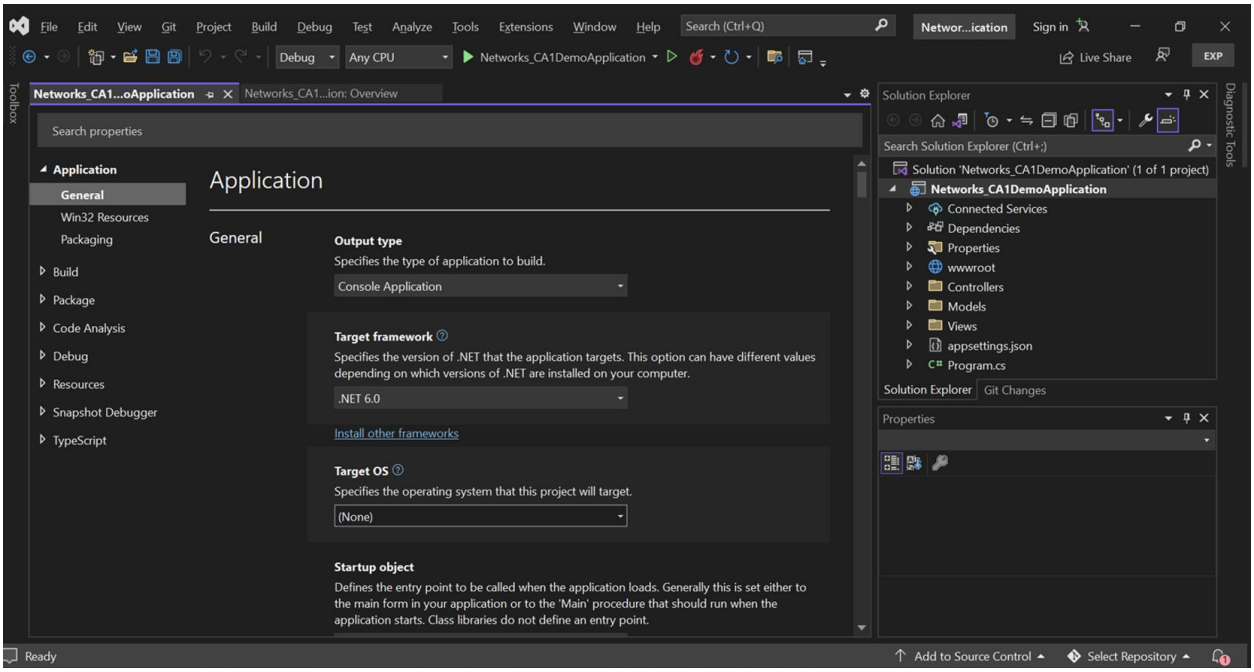

Figure21: displays the project properties of codebase

Selected the version as 6.0.x:

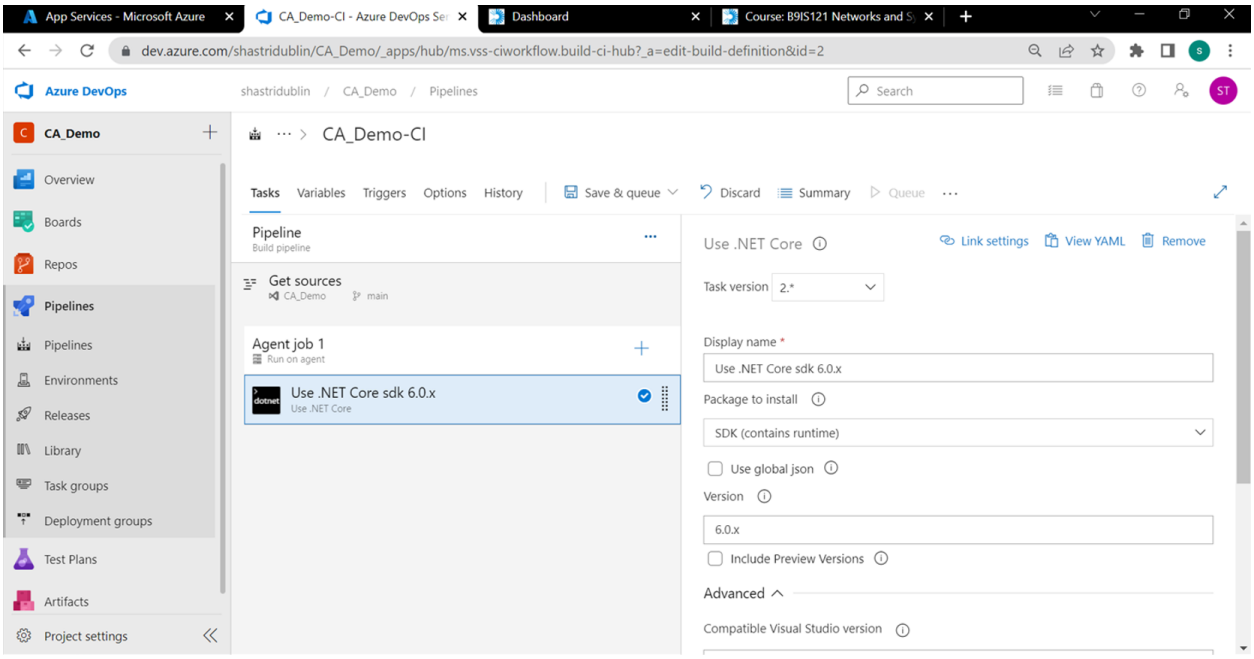

Figure22: displays the use .Net core step details

Added a new .NET Core step:

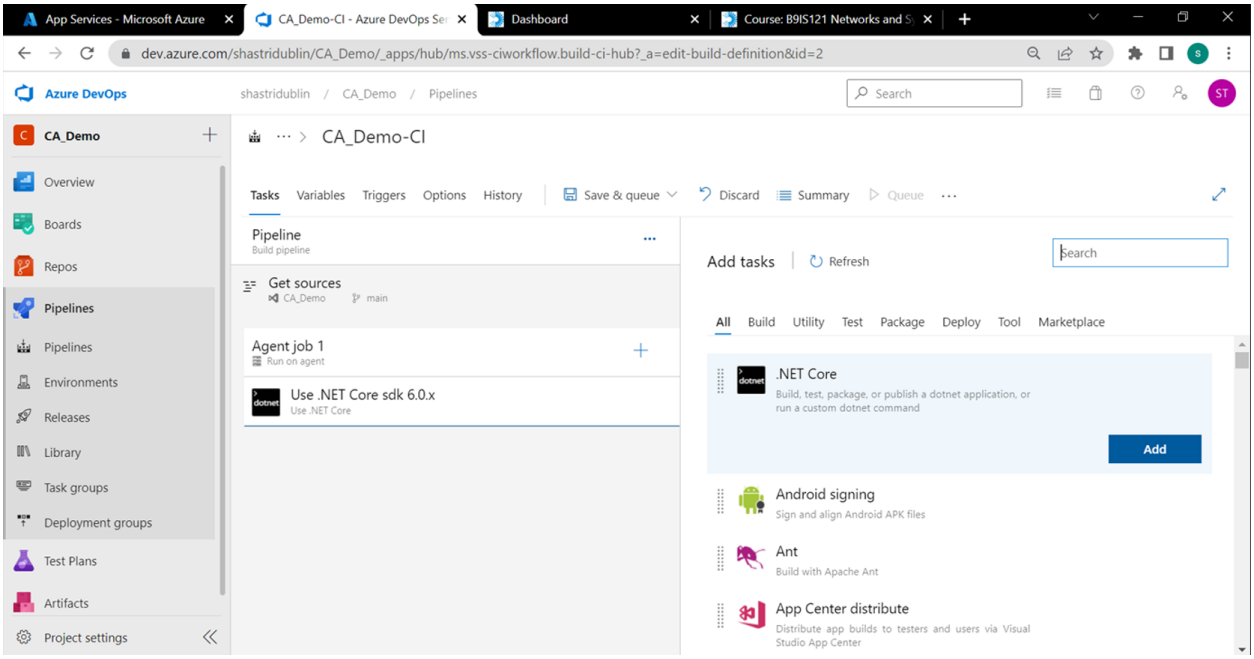

Figure23: displays .NET Core step in CI

The default option it provides is build, to build the .net application:

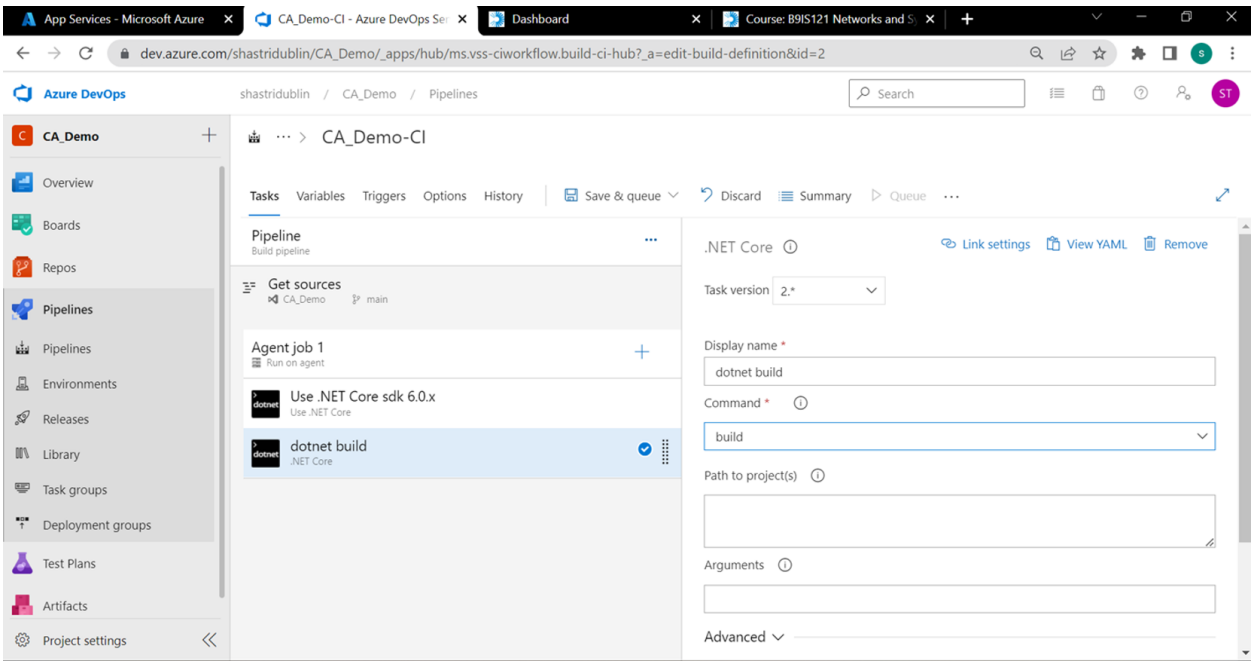

Figure24: displays the options after adding .net core step

In .net the build process comprises of the below three steps (restore > build > publish):

1. dotnet restore Networking_CA_Demo_WebApp.csproj

2. dotnet build Networking_CA_Demo_WebApp.csproj - -configuration Release

3. dotnet publish Networking_CA_Demo_WebApp.csproj - -configuration Release - - Output C:\OutputFiles

The above three steps are very specific for .net code and will change in case of other programming languages.

Added the first option restore and added the project path

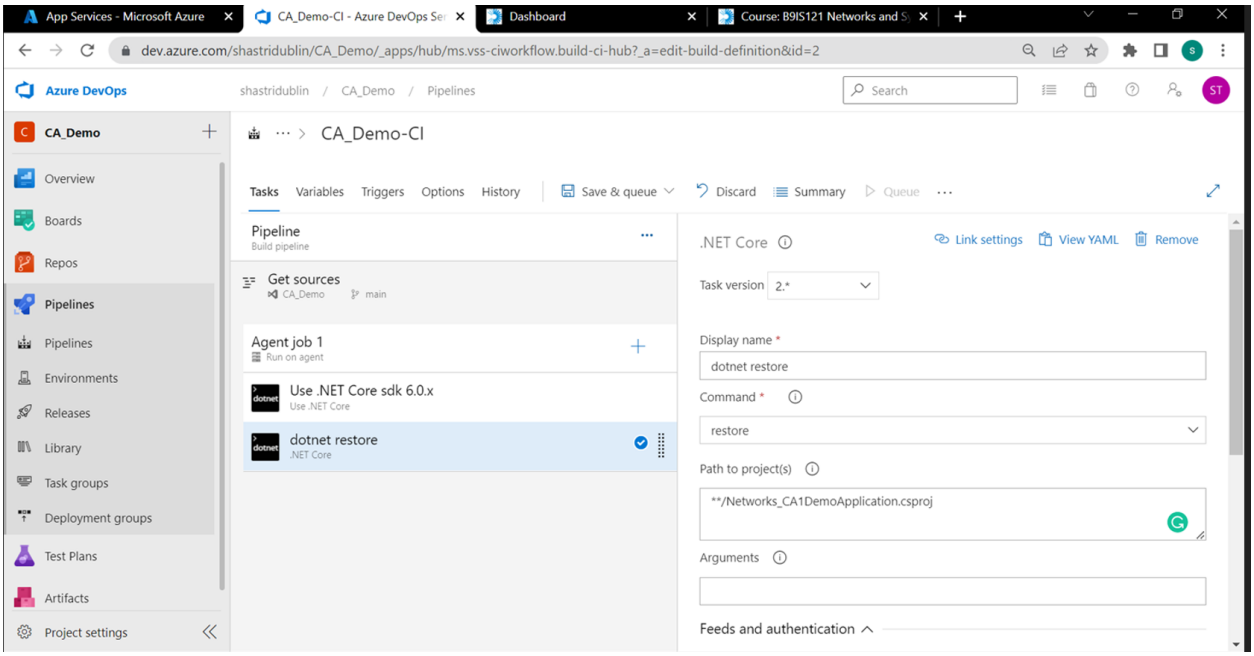

Figure25: displays renaming of the step to restore

Added the build step and added project path and arguments

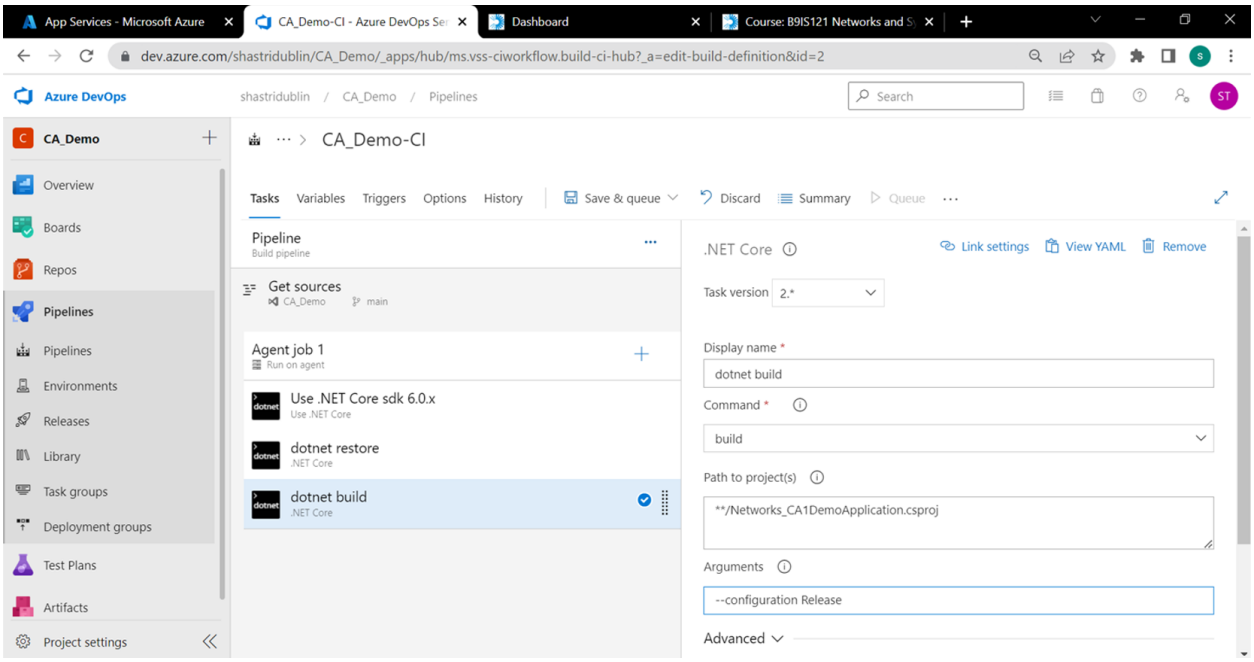

Figure26: displays addition of dotnet build

Clicked on the create variables option and clicked on the Add option:

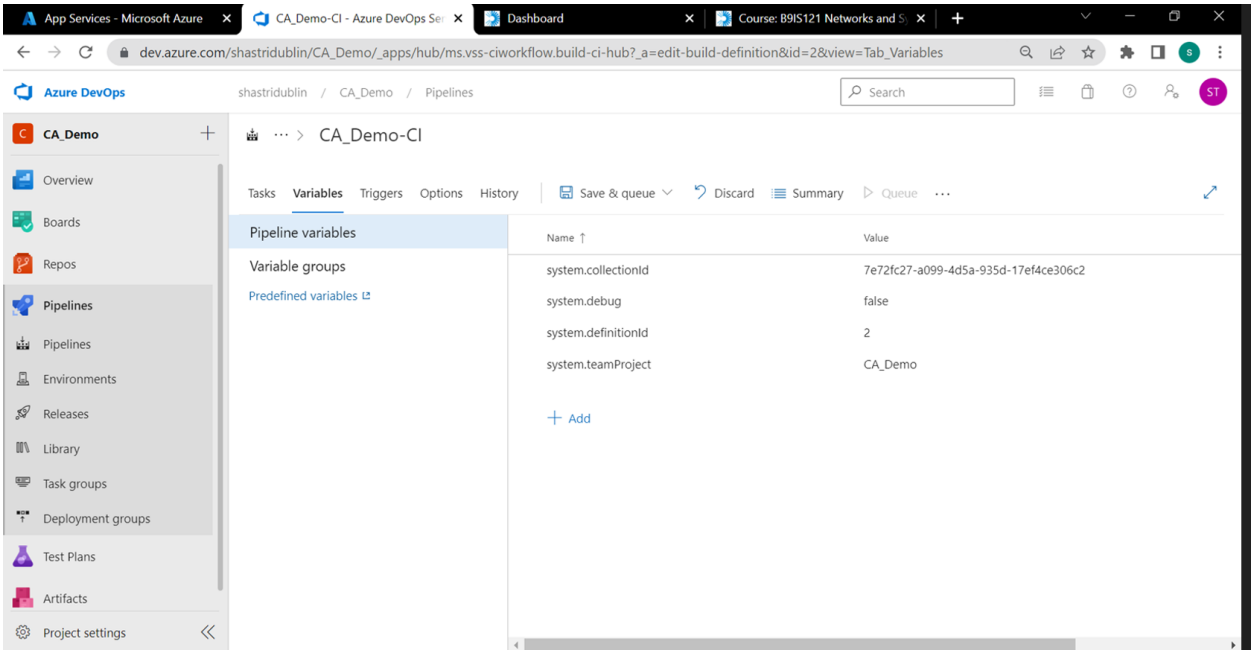

Figure27: displays the place to create a new variable

Created a new variable called ‘BuildType’ and specified its value as ‘Release’:

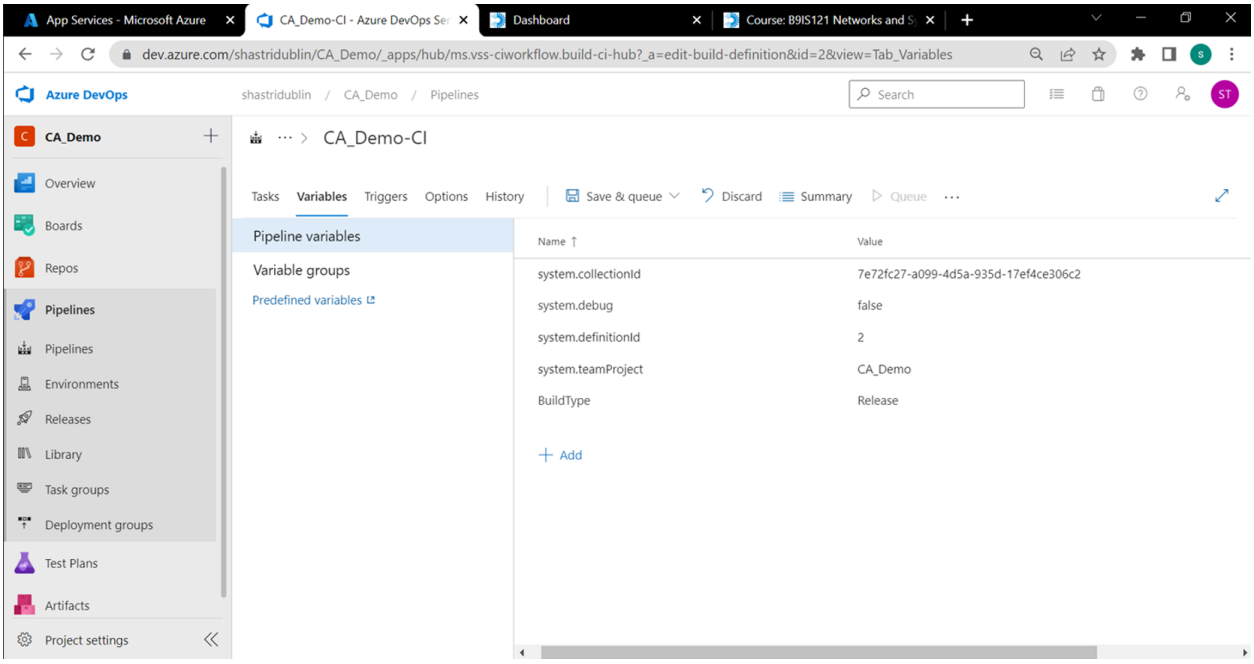

Figure28: displays creation of a new variable

Added publish which is the final step and specified the arguments (instead of hardcoding ‘Release’, referred it from the variable name that we created in the previous step)

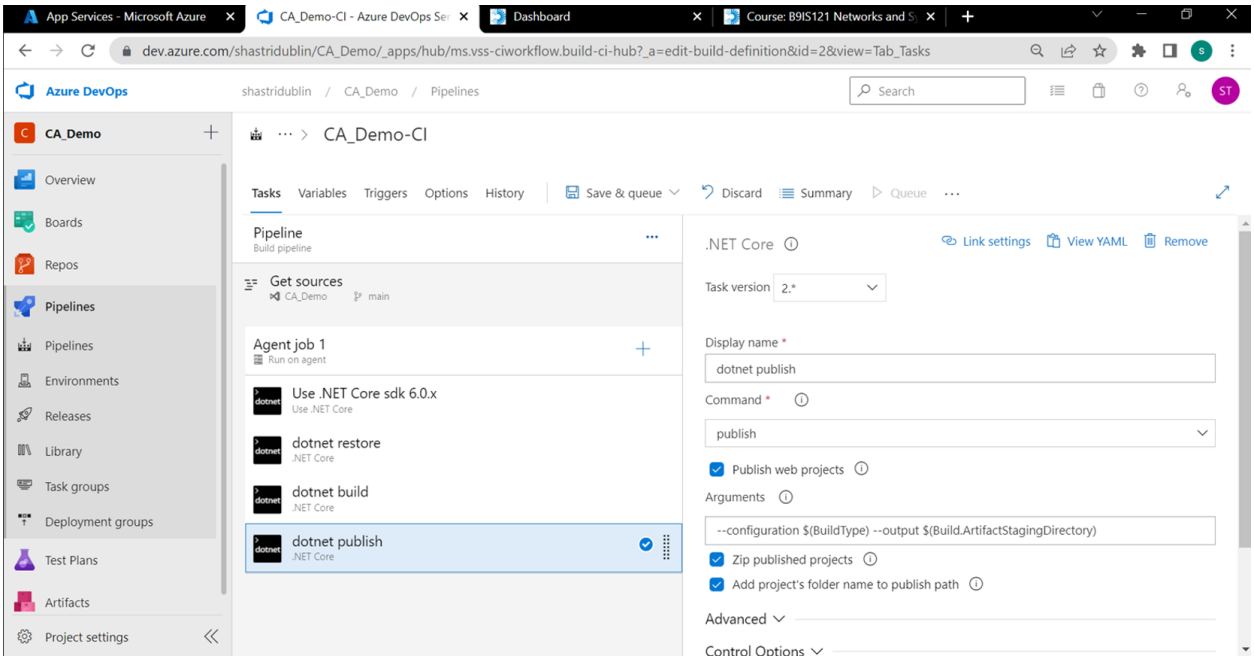

Figure29: displays the dotnet publish option

Added a step to publish the build artifacts (essential dependencies) which will be then deployed to the web application:

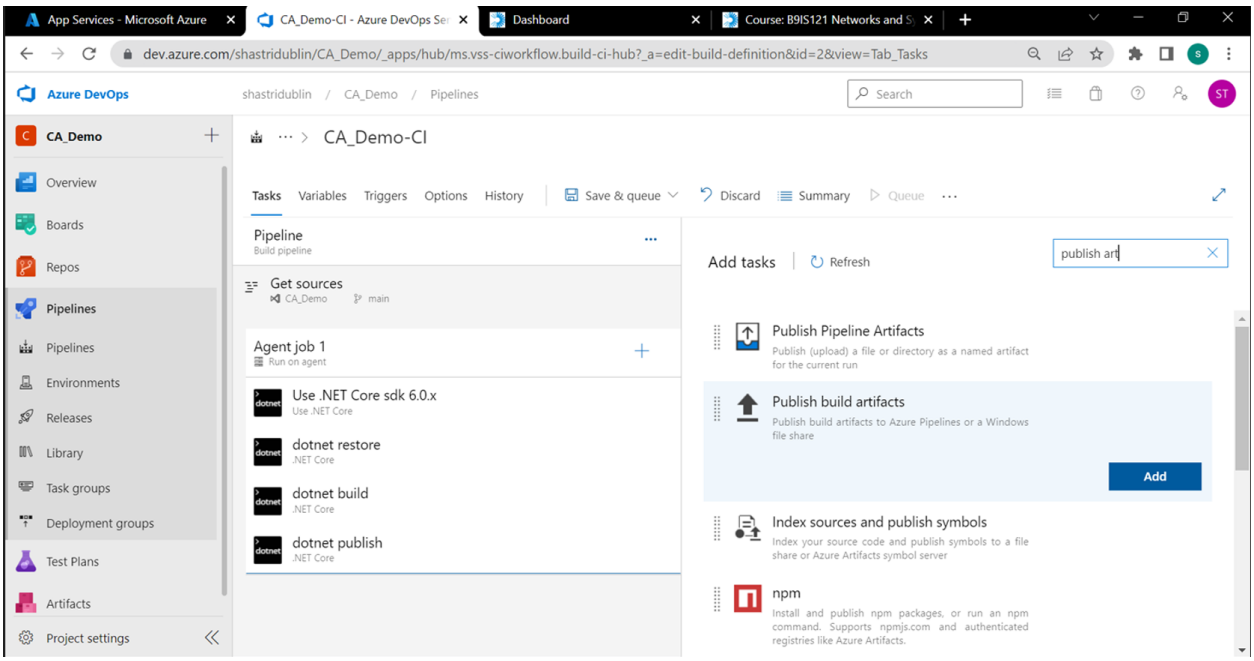

Figure30: displays publish build artifact option

Specified the path as 'ArtifactStagingDirectory' which is a default option:

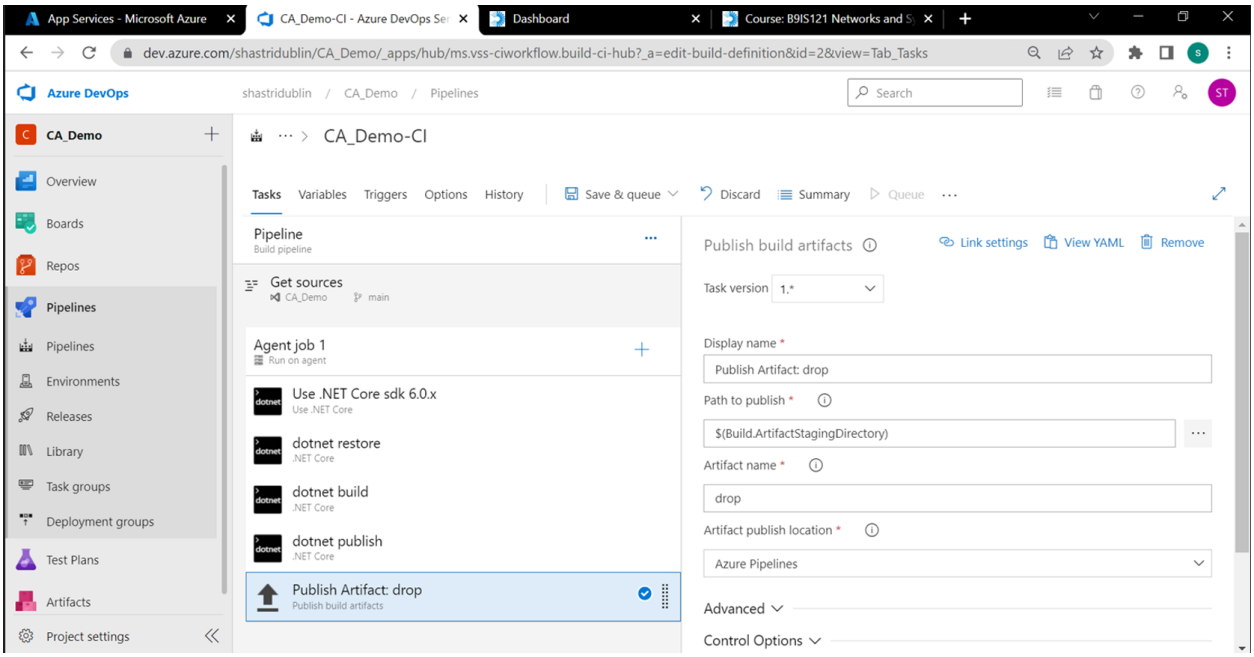

Figure31: displays details of publish artifacts

Clicked on Save & queue:

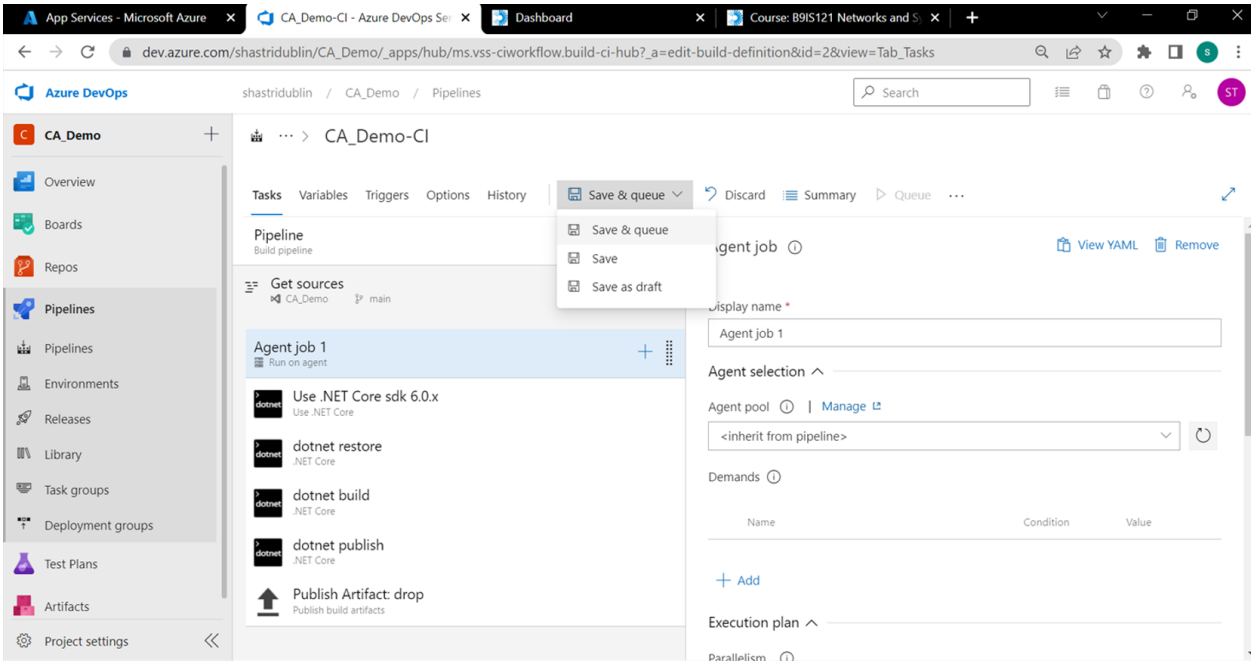

Figure32: displays save options for a CI pipeline

Once queued (based on the parallelism approved in Azure DevOps) the agent job kicks off, here we can see that it ran successfully through all the steps we listed above:

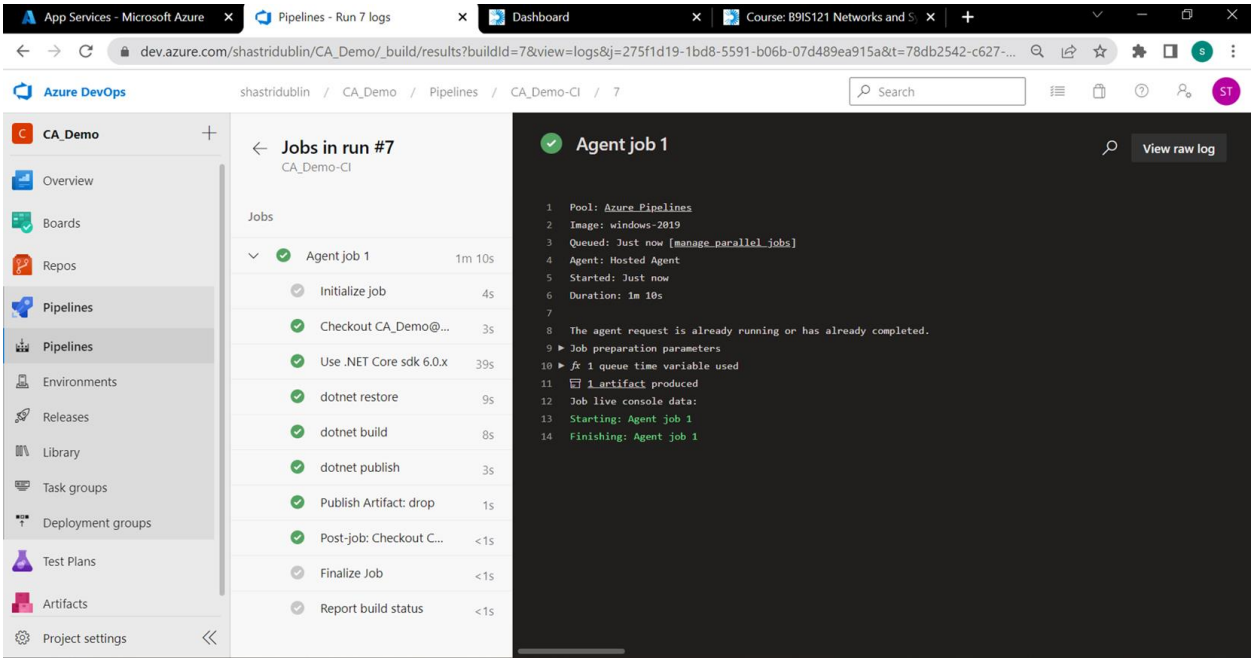

Figure33: displays queue status of CI pipeline

Clicked on the Artifacts and verified that the dependencies got placed here automatically in a zip file:

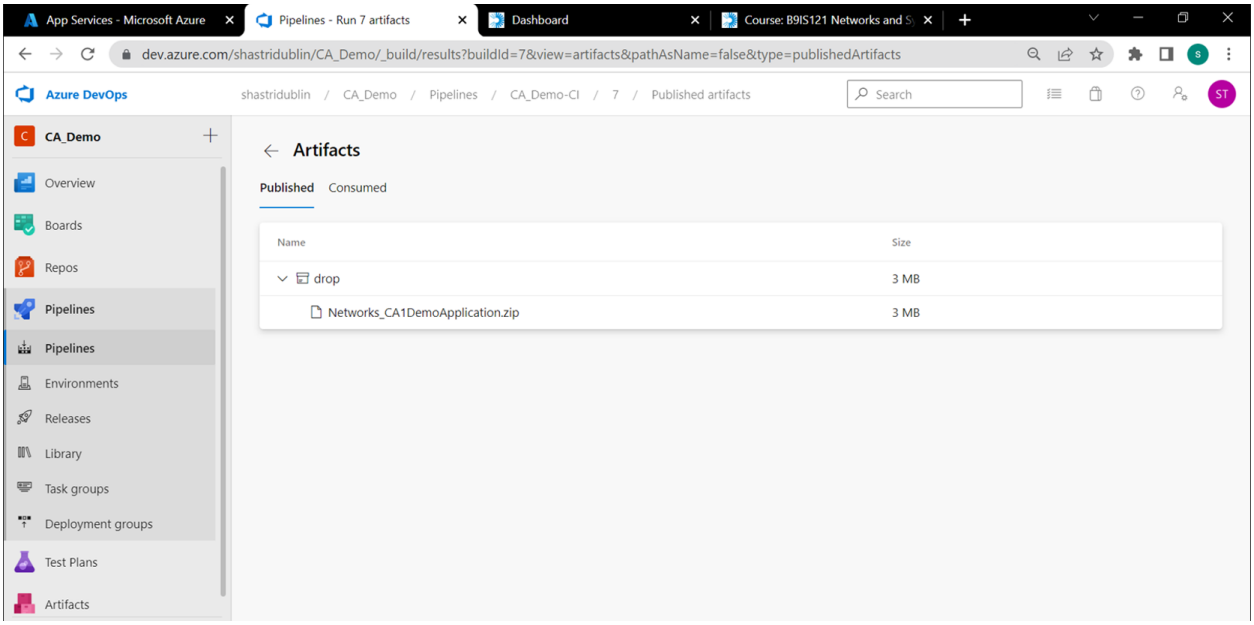

Figure34: displays azure devops artifacts

Under Triggers, we can tick the checkbox 'Enable continuous integration' – enabling this option will trigger the CI pipeline automatically whenever a code push is detected on the Azure DevOps repo:

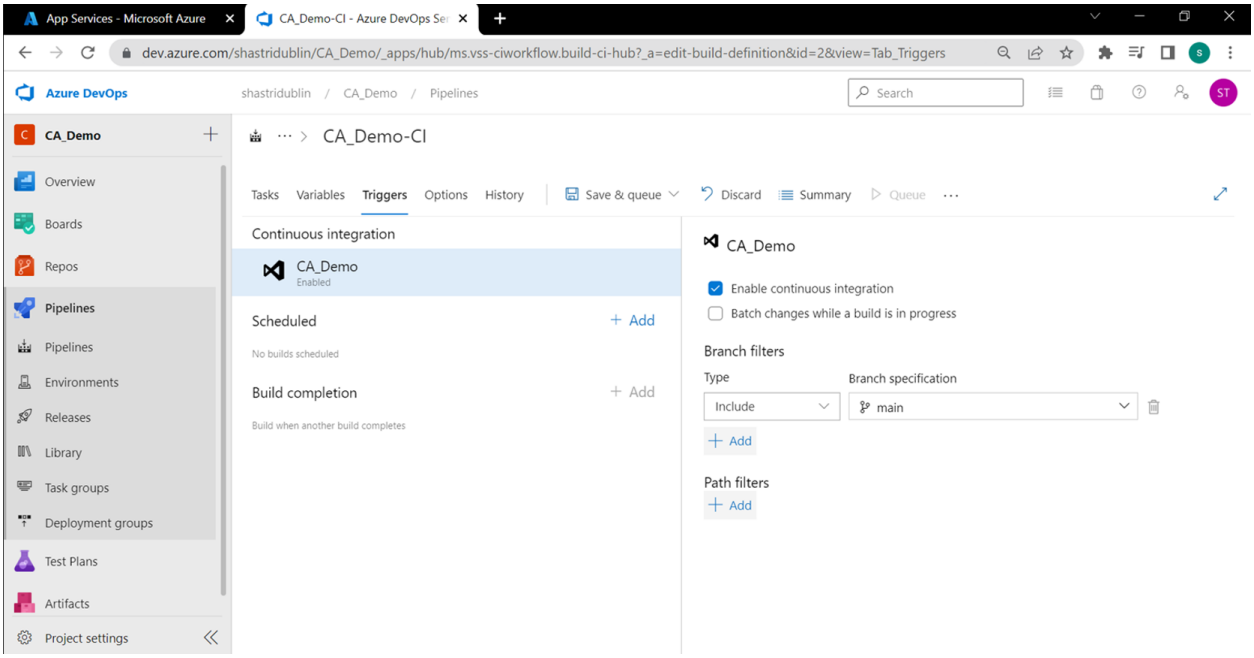

Figure35: option to auto enable CI

- ***Created a service connection to azure before creating a CD pipeline for deployment***

Clicked on project settings > service connections and clicked on create service connection:

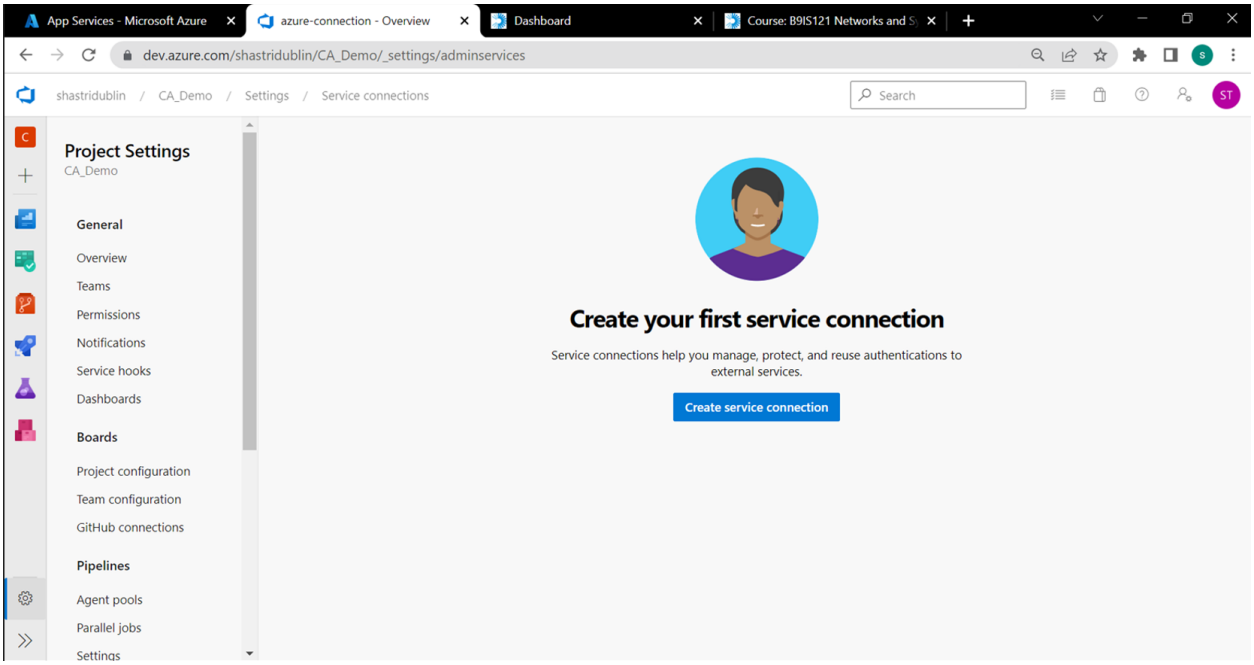

Figure36: option to create service connection

Since we had to connect to the azure web service, selected Azure Resource Manager:

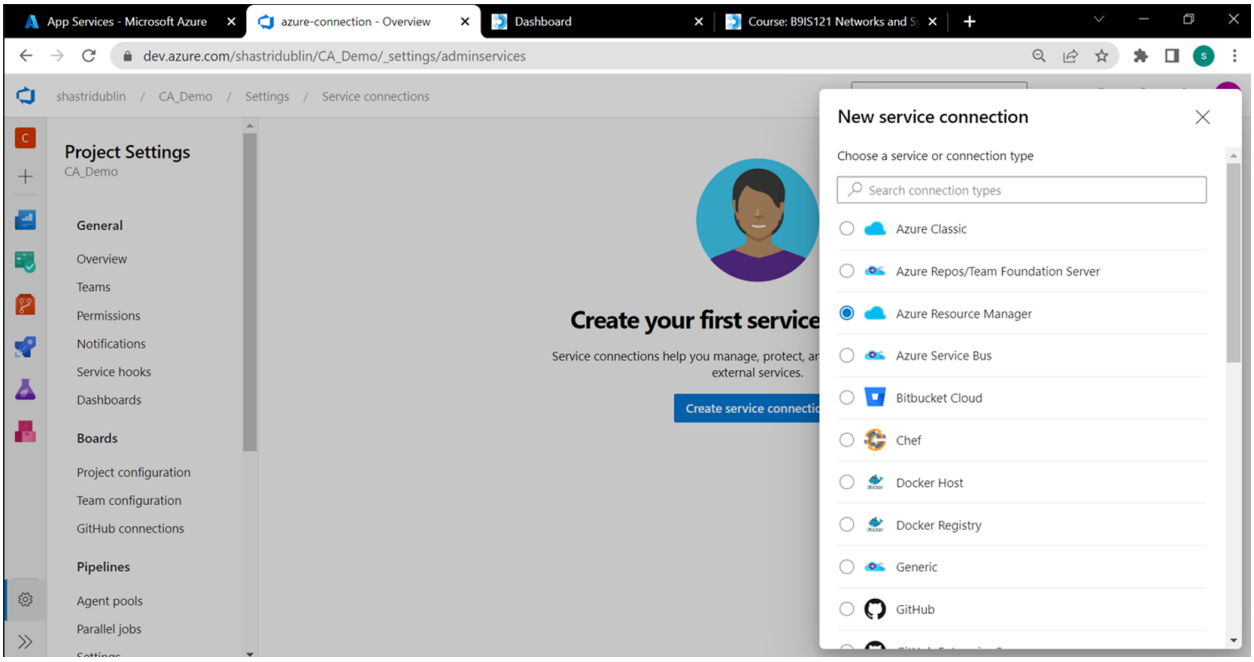

Figure37: displays selection of ARM to add app service connection

Clicked on the default authentication method and clicked next:

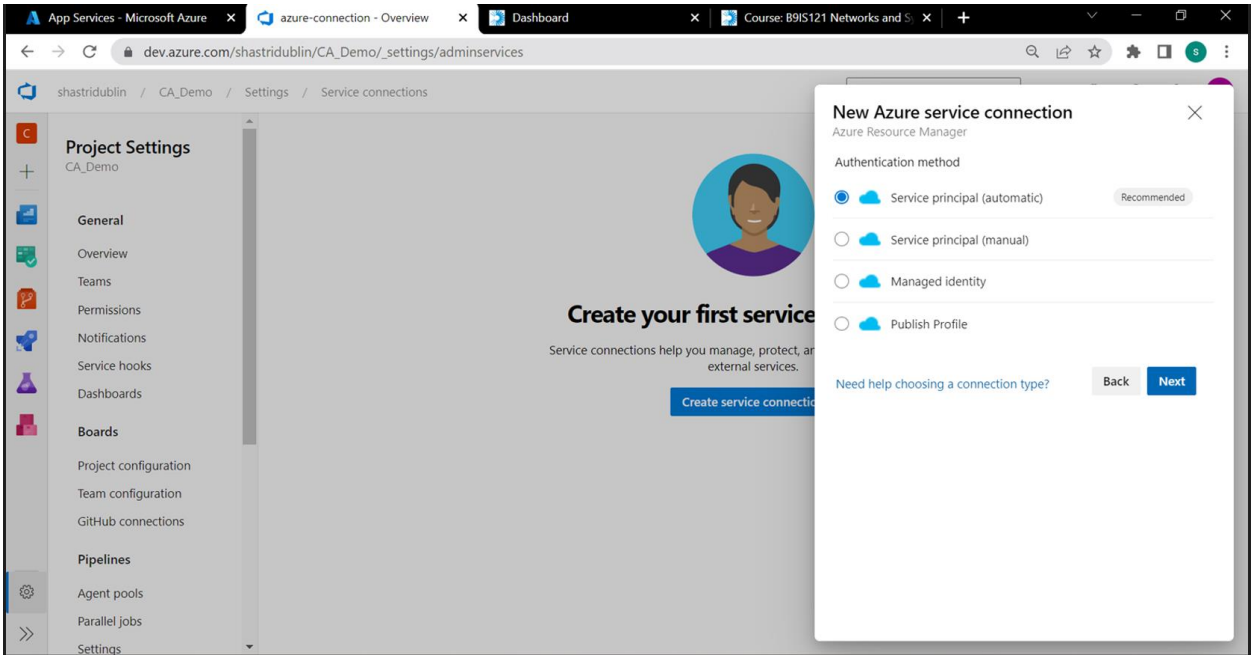

Figure38: displays default authentication method for service connection

The Azure subscription id started appearing automatically based on my user login the subscription dropdown, added a service connection name and clicked on save:

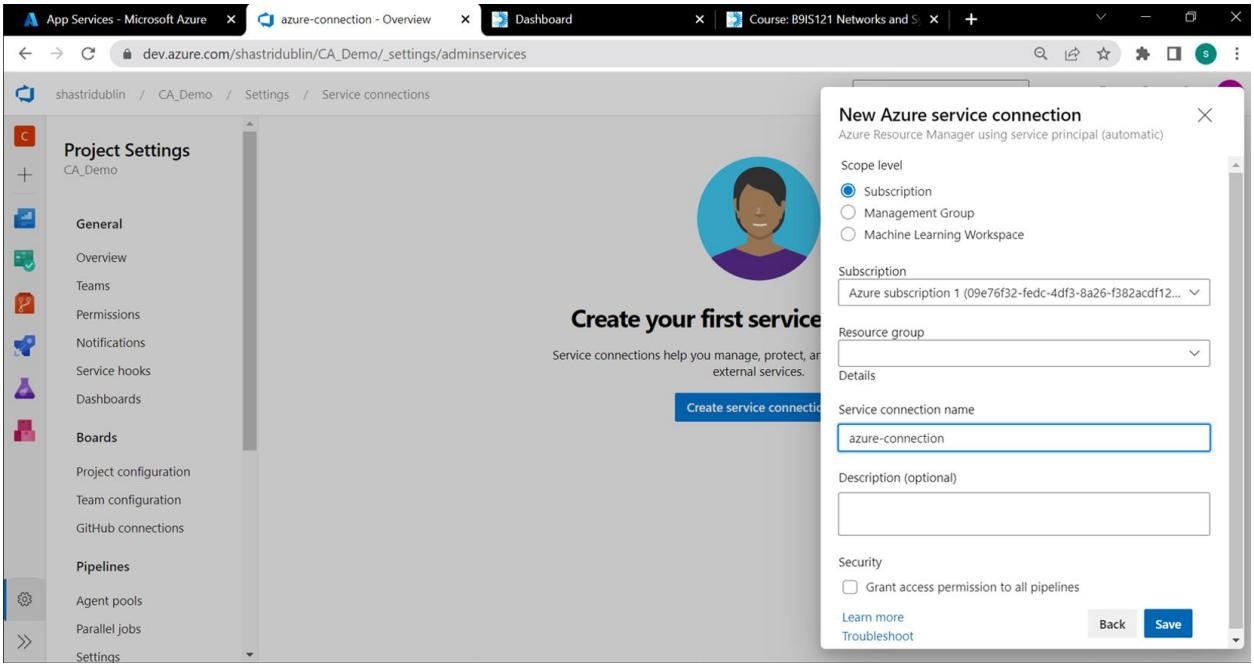

Figure39: displays option to select subscription and add service connection name

Verified that this new service connection was displaying:

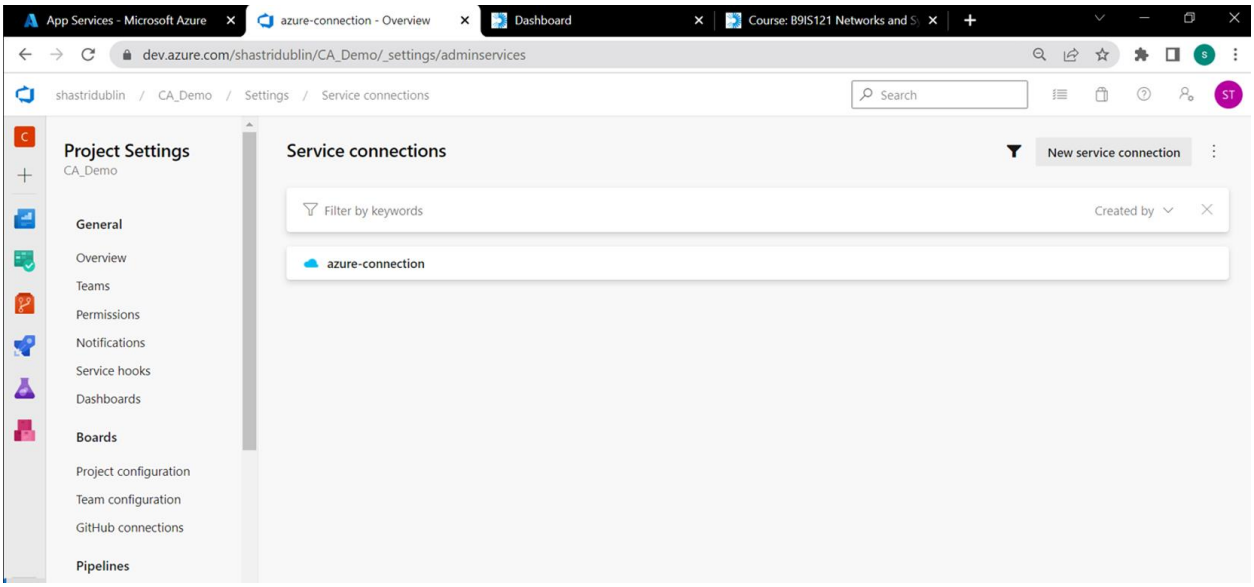

Figure40: verifying the creation of a service connection

- ***Created a CD pipeline and verified the deployment***

Clicked on Pipelines > Releases to create a new CD pipeline

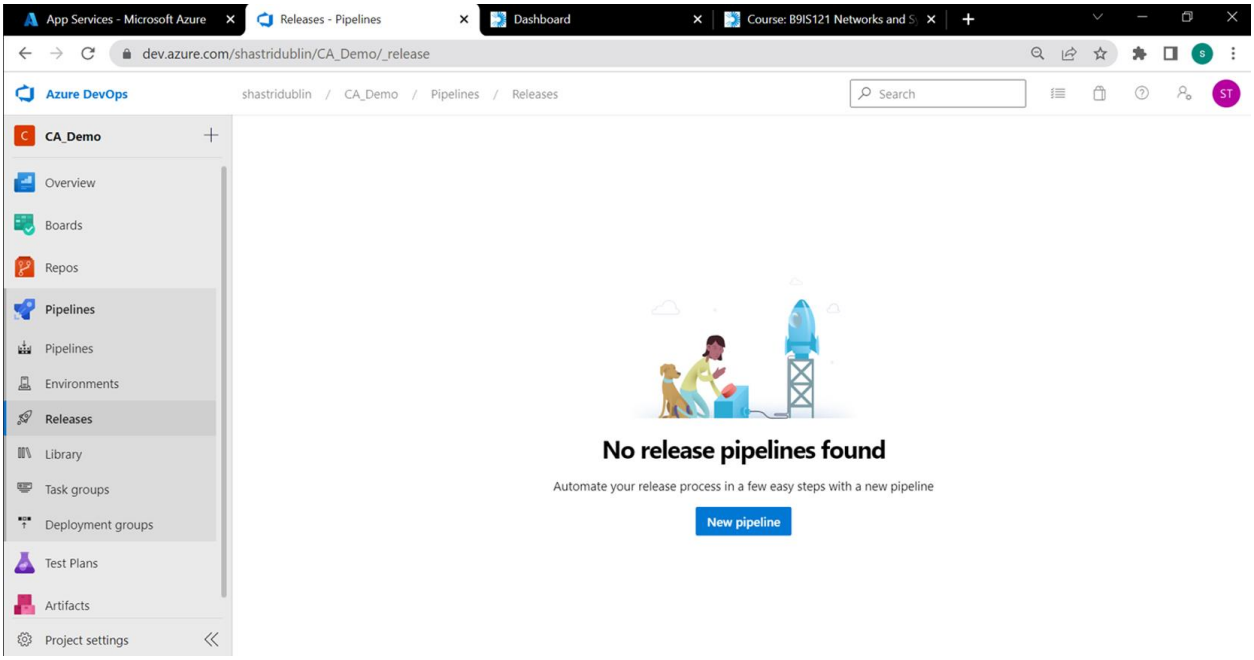

Figure41: place to create a new CD pipeline

Clicked on the empty job template to create the new pipeline:

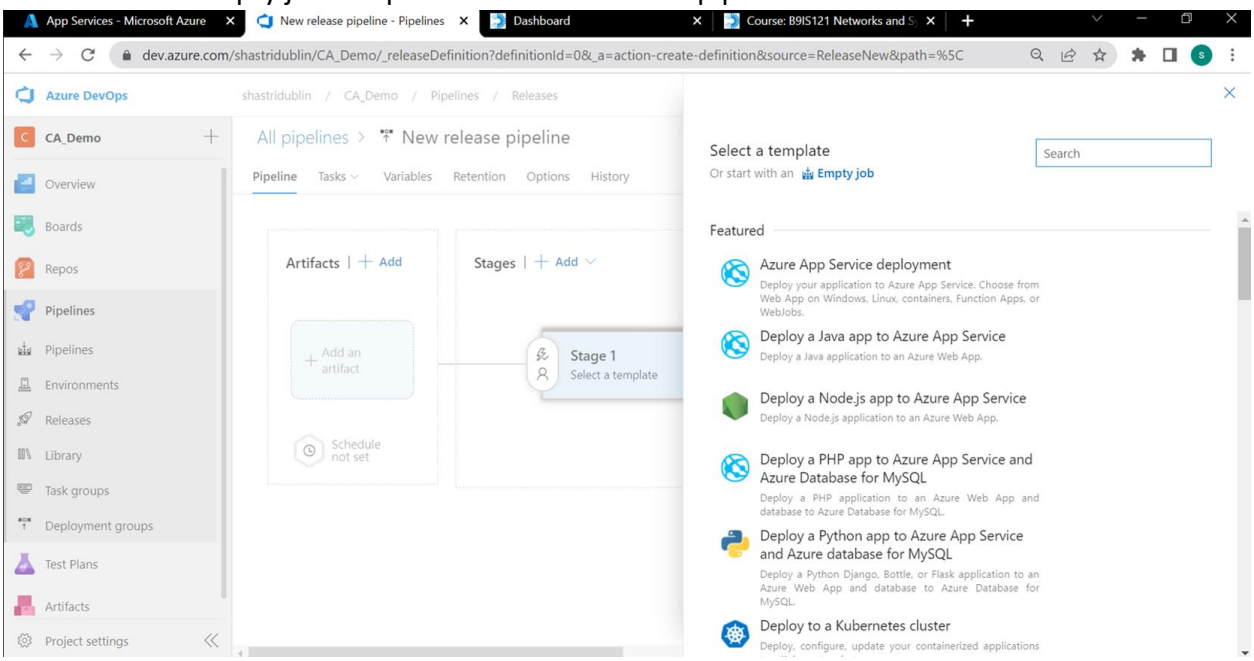

Figure42: selection of a CD pipeline template

Selected the source repo and provided the pipeline name:

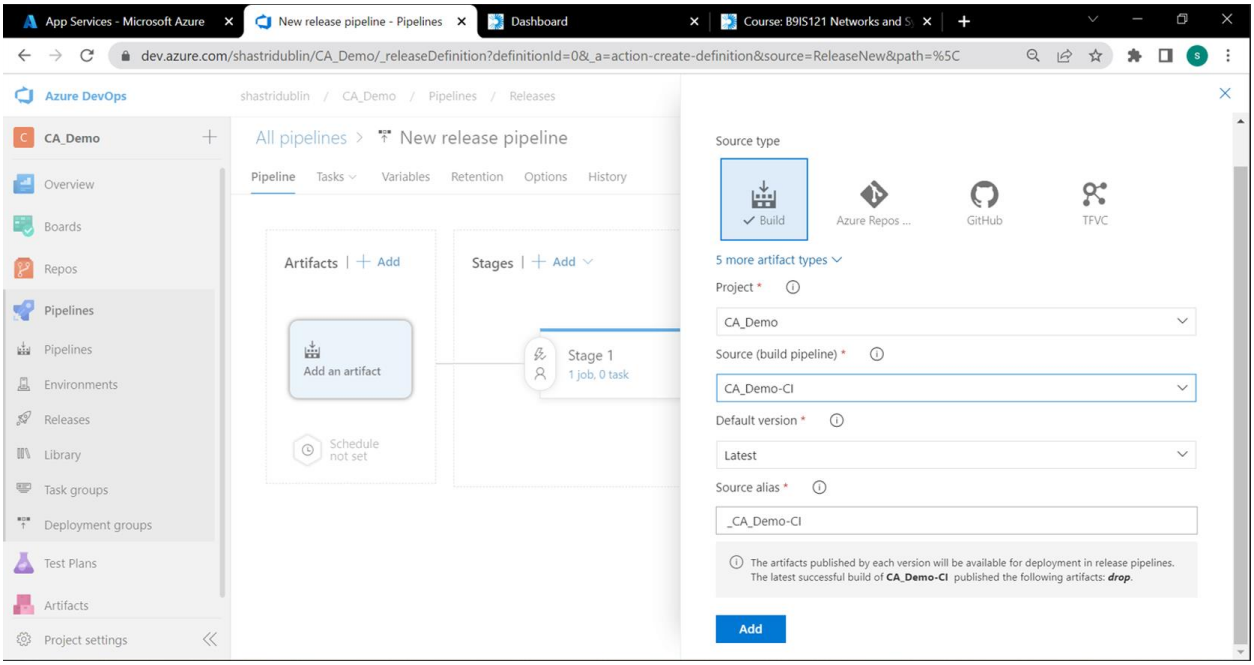

Figure43: default step in CD pipeline

Renamed the stage1 to development as we will first push the code base onto the development web app before proceeding to uat and production:

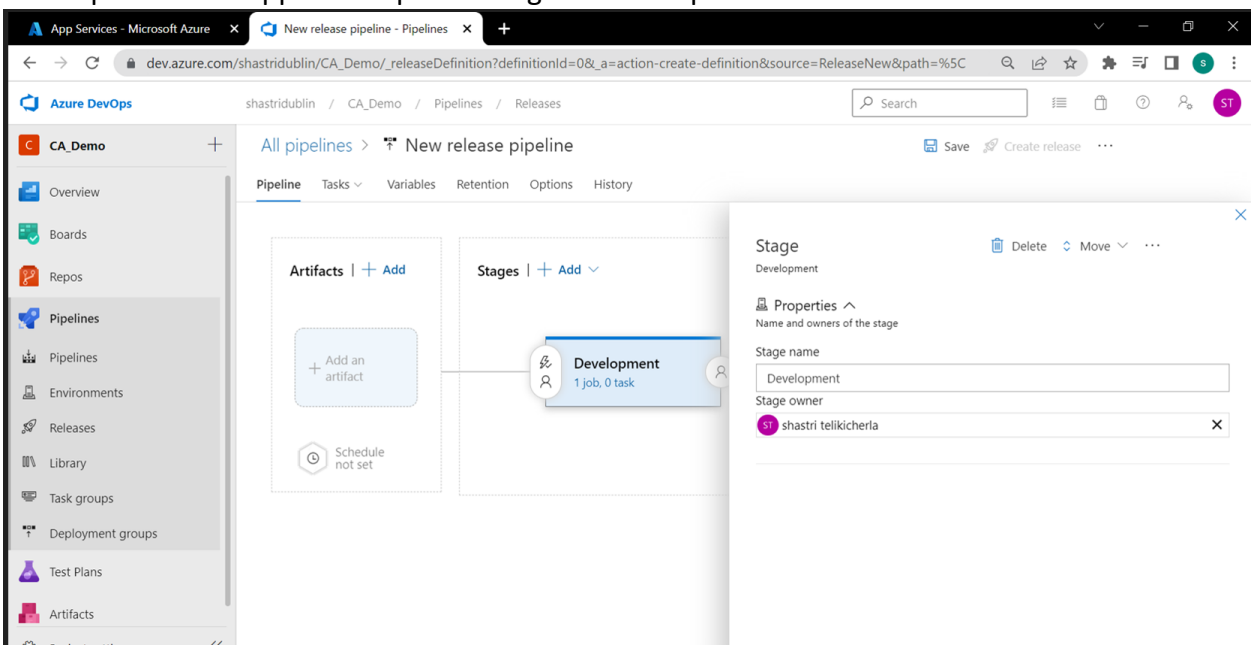

Figure44: renaming a stage in CD pipeline

Added a step Azure App Service Deploy under the agent job and specified the azure subscription:

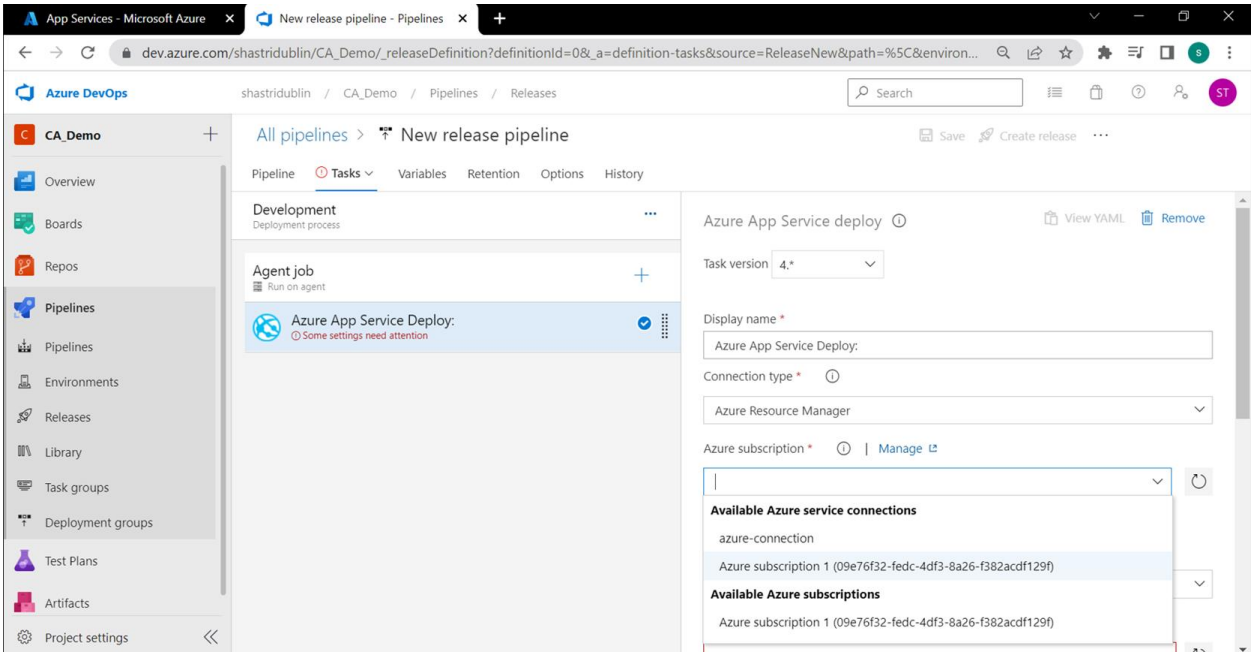

Figure45: selection of azure web app in service deploy

Selected the development web application:

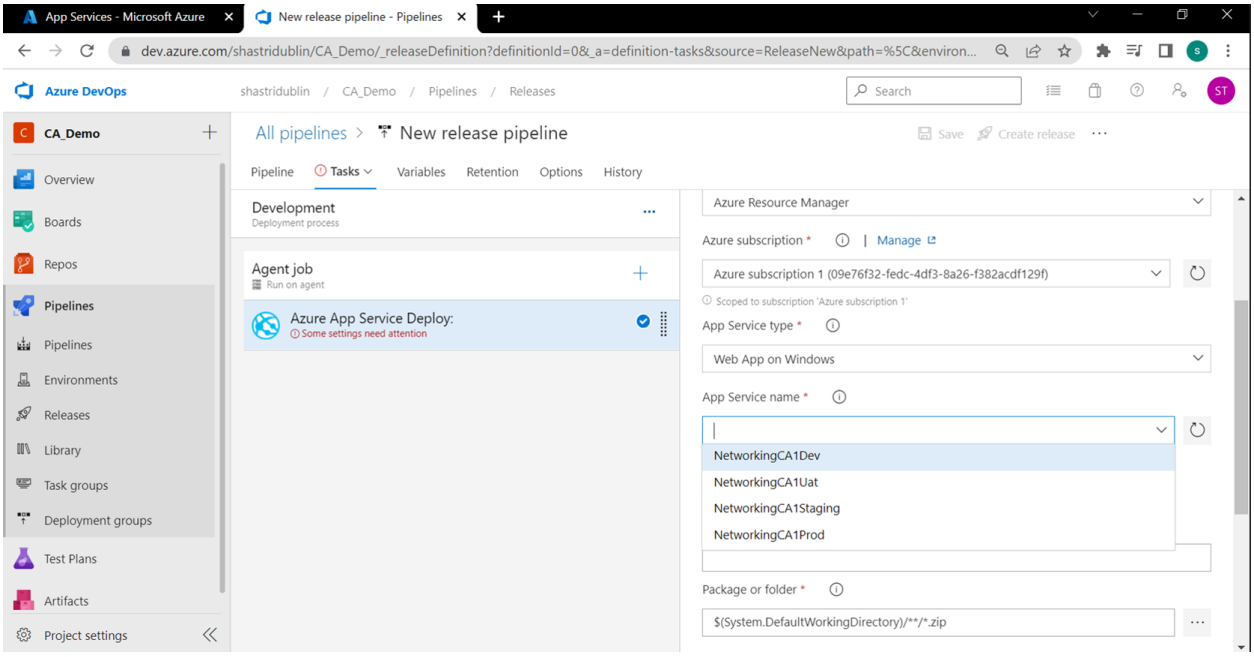

Figure46: selection of azure web app in service deploy contd.

Clicked on the clone option to create a copy of the development stage:

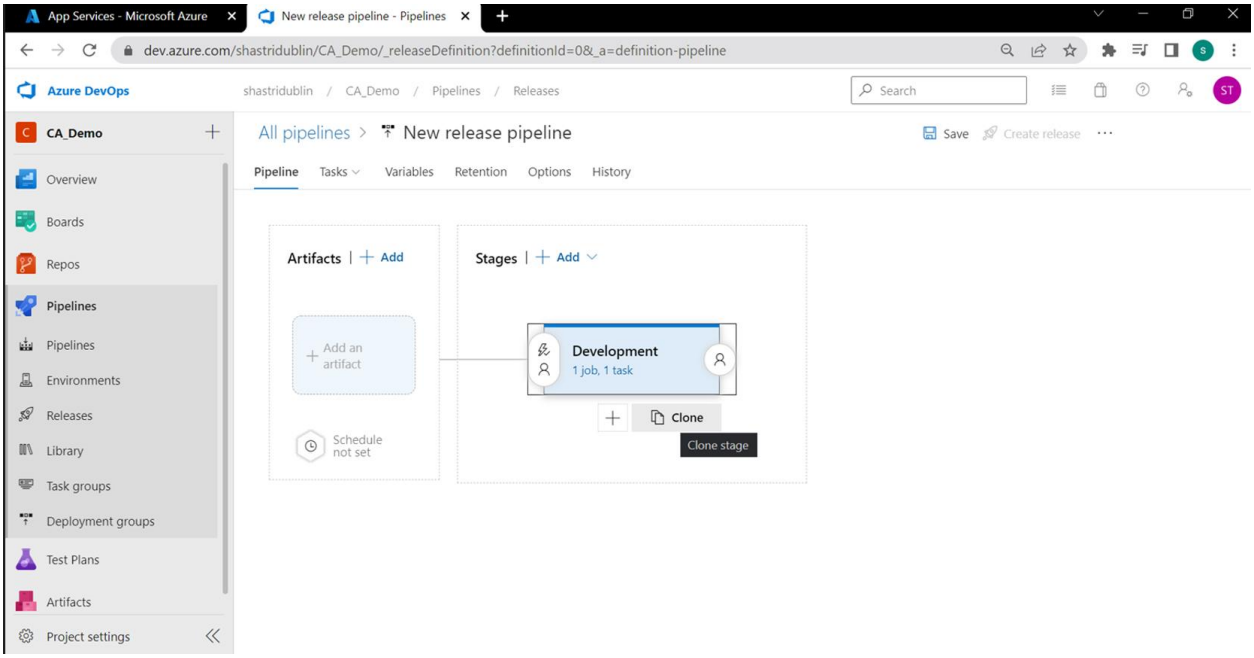

Figure47: displays option to clone a step

Configured this step for the UAT by selecting UAT link under app service name dropdown:

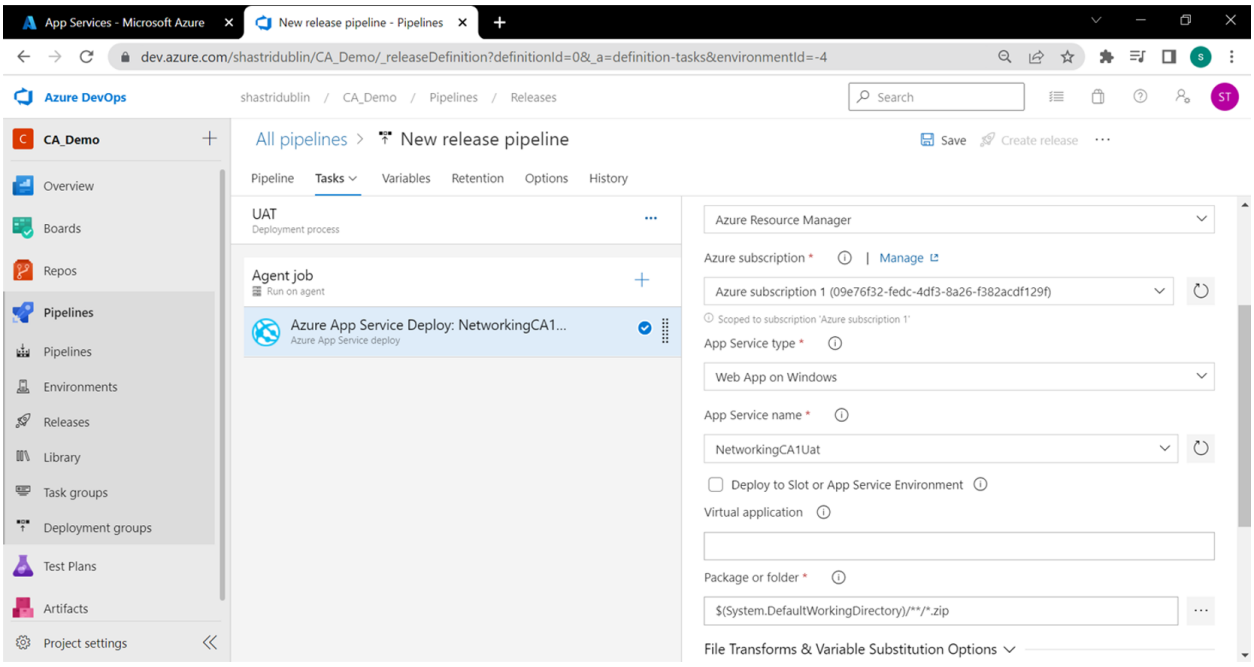

Figure48: selection of azure web app in service deploy

Since there is an approval required before the uat and production deployment, added a pre-deployment condition to the UAT stage and enabled approval required, added 'Shastri' as the approver:

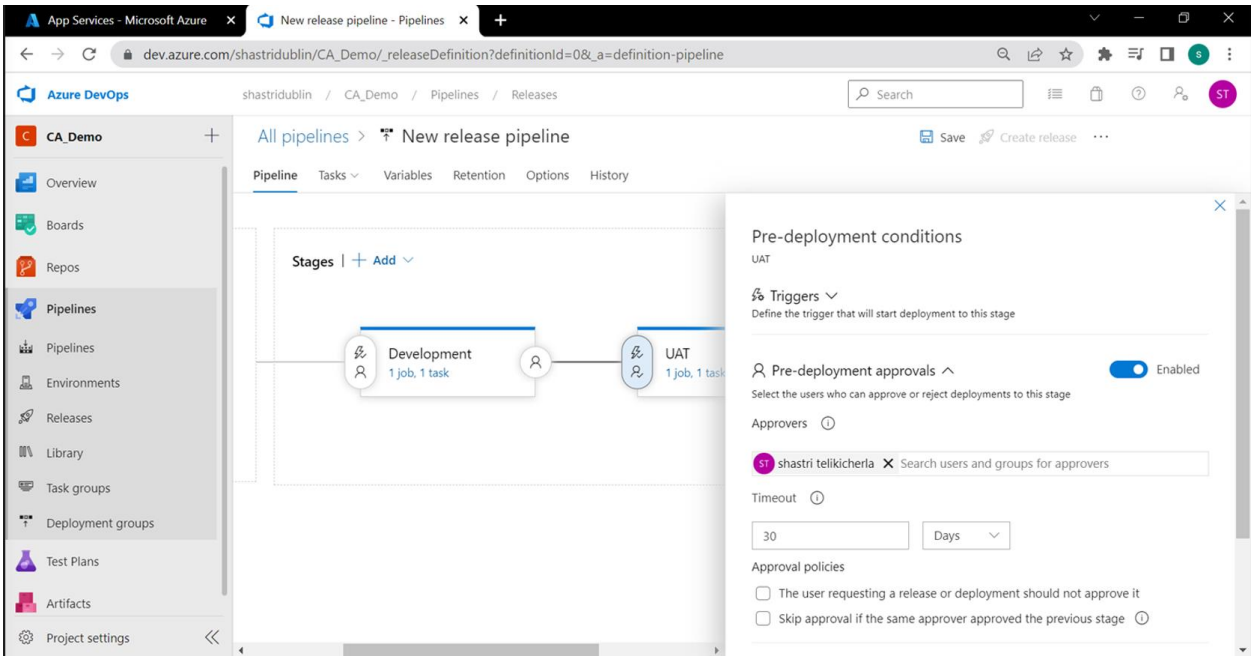

Figure49: option to enable pre deployment conditions

Enabled the continuous deployment trigger, which will automatically run this pipeline every time after a new artifact gets created from the previous CI pipeline:

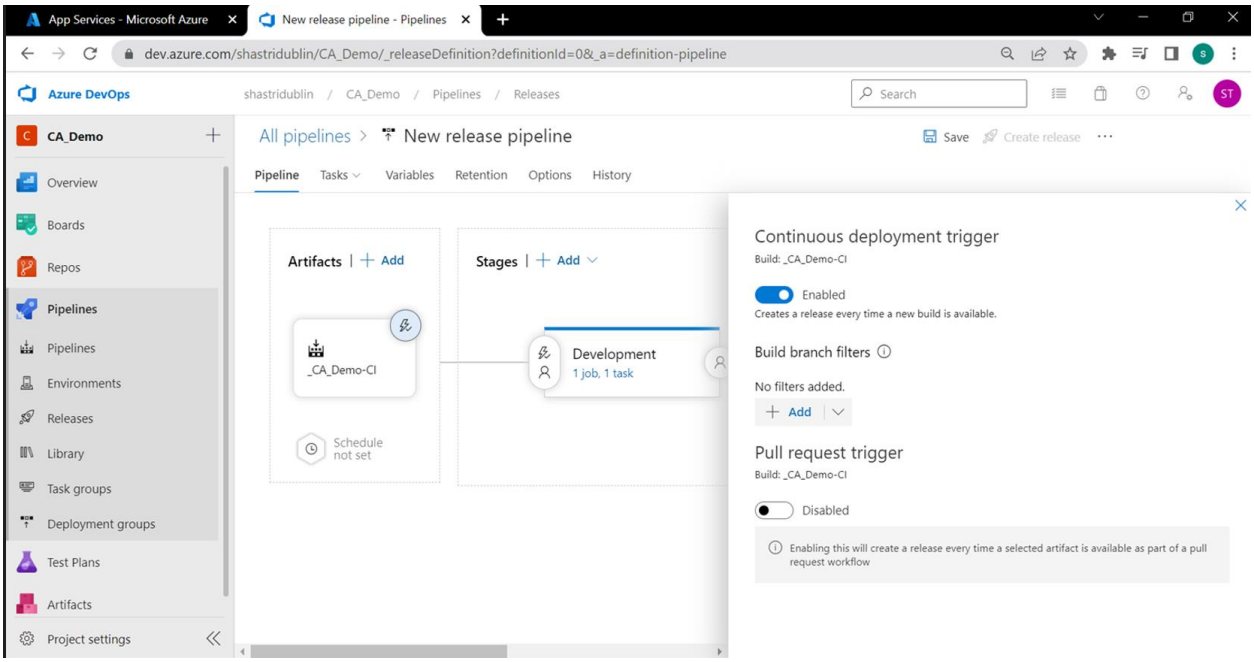

Figure51: displays option to auto enable CD pipeline trigger

- ***Web applications were compared from local and the dev, uat, prod environments and it was verified that the changes got deployed successfully via the CI and CD pipelines.***

7. Improvements:

During this project, few possible improvements were observed as listed below:

- Ability to demonstrate a CD pipeline with a staging environment without production downtime.

- Adding a full stack release deployment including a database and Web API along with a web application.

- Including other agile practices along with CI/CD to demonstrate DevOps.

- Usage of git command line to push/pull codebase to Azure DevOps repos.

8. Conclusion:

This project successfully demonstrated the CI/CD pipeline creation and automation of build and release cycles. It was an interesting and challenging project which helped deep dive into .net MVC web development, azure app services, azure devops and helped in understanding the capability of CI/CD and Azure DevOps service offering.

It also helped in understanding few self-improvement topics in terms of CLI usage, code optimization, application of object-oriented concepts, more challenging DevOps scenarios etc. which will become part of an offline implementation or in any of the upcoming projects.

9. References:

- Zhu, L., Bass, L. and Champlin-Scharff, G., 2016. DevOps and its practices. IEEE Software, 33(3), pp.32-34.

- Erich, F.M., Amrit, C. and Daneva, M., 2017. A qualitative study of DevOps usage in practice. Journal of software: Evolution and Process, 29(6), p.e1885.

- Soh, J., Copeland, M., Puca, A. and Harris, M., 2020. Continuous integration/continuous delivery with azure devOps. In Microsoft Azure (pp. 493-519). Apress, Berkeley, CA.

- Smith, S., 2018. Overview of ASP .NET Core MVC. Microsoft. Last modified January, 7, p.2018.

- Video credits:

  https://www.youtube.com/watch?v=xH5EY7FCFQw

  https://www.youtube.com/watch?v=H-R2bCXfz8I

  https://www.youtube.com/watch?v=GCe11rnjRsk

- Web credits:

  https://docs.microsoft.com/en-us/azure/devops-project/

  https://docs.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-6.0