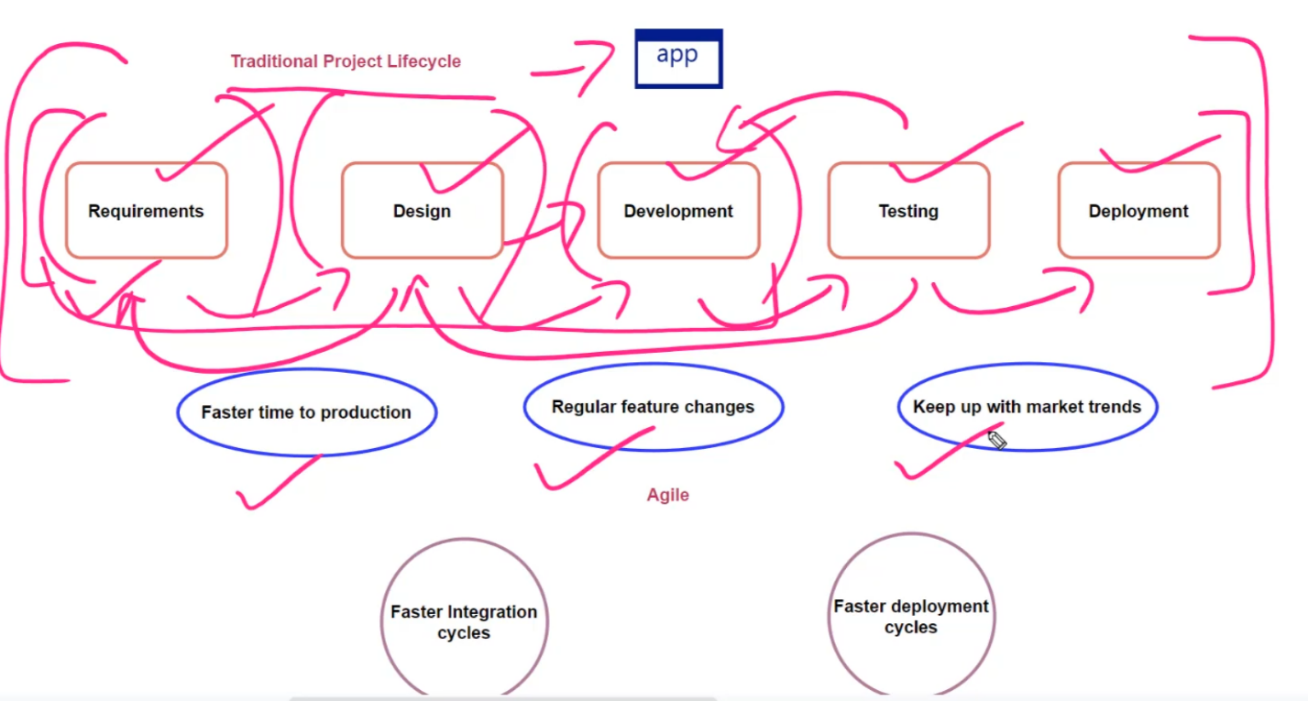
# Azure DevOps Fundamental

# 

1. **Why DevOps**

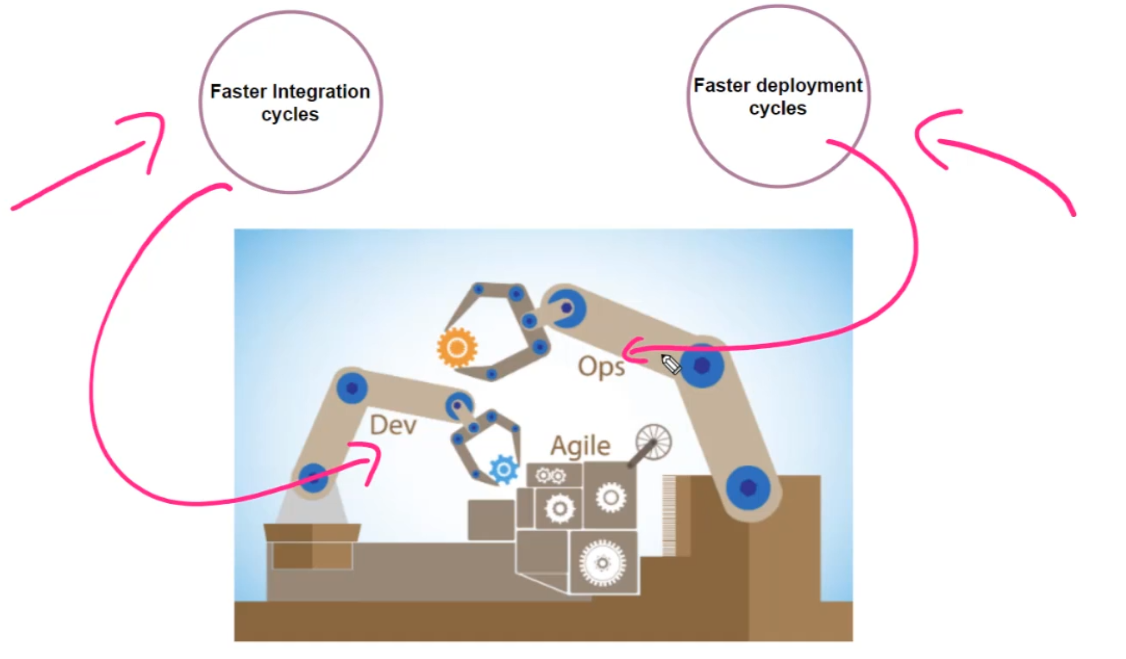
Usually we go through the SDLC process of Requirement -> Design -> Development -> Testing -> Deployment.

But doing it at once shot getting all the requirements, all the design, all relevant at one time itself This is an issue. So in the traditional project, life cycle, there were a lot of drawbacks that were being encountered. And also customers were giving regular feature changes onto the IT department. And also users want to ensure to keep up with market trends when it comes to their application changes. So then they came the adoption of the agile methodology.



So when it comes to agile, we are also looking at ensuring that we have faster integration cycles. So over here when a developer makes a change onto a code. It should be integrated very easily with the application so that those changes can be tested and also looking at faster deployment cycles.

So what we want to do the use of agile as the underlying methodology for the application lifecycle, and that's where we come to the term of devOps.



1. **Azure DevOps Tools:**

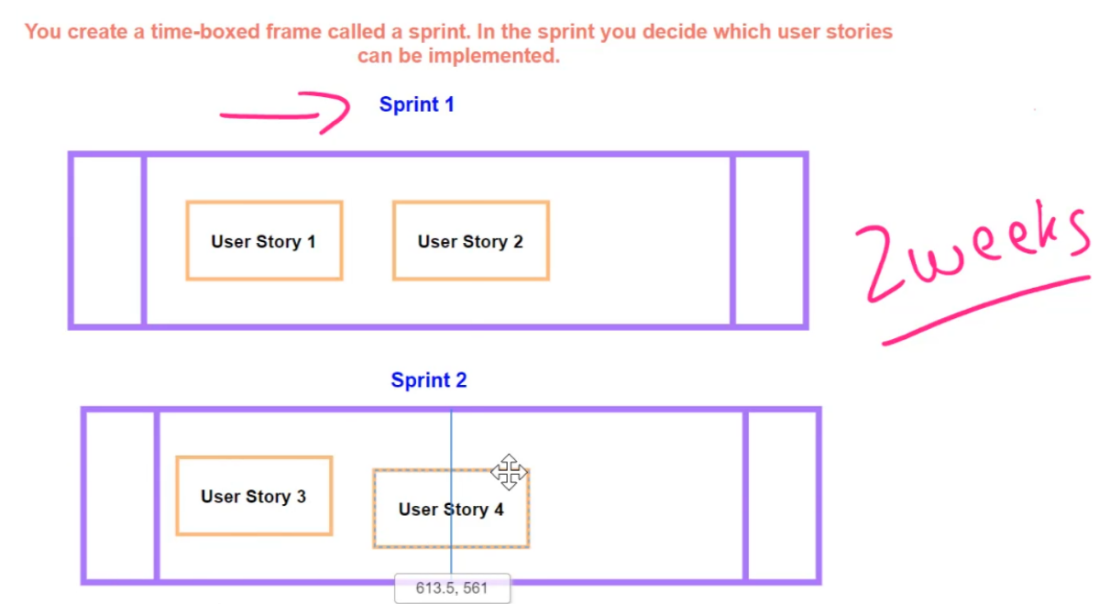
|  |  |
| --- | --- |
|  | So this helps you in the planning, tracking and working with work items across various teams. So this is similar to a project management solution that is available in Azure DevOps. |
|  | So this helps you with continuous integration and continuous deployment. |
|  | Used for managing your source code. Now Over here we are going to focus on Git. So GIT is an underlying tool that is available in Azure repos for managing your source code. |
|  | Used for manual and exploratory testing. |
|  | Used for managing your packages. |

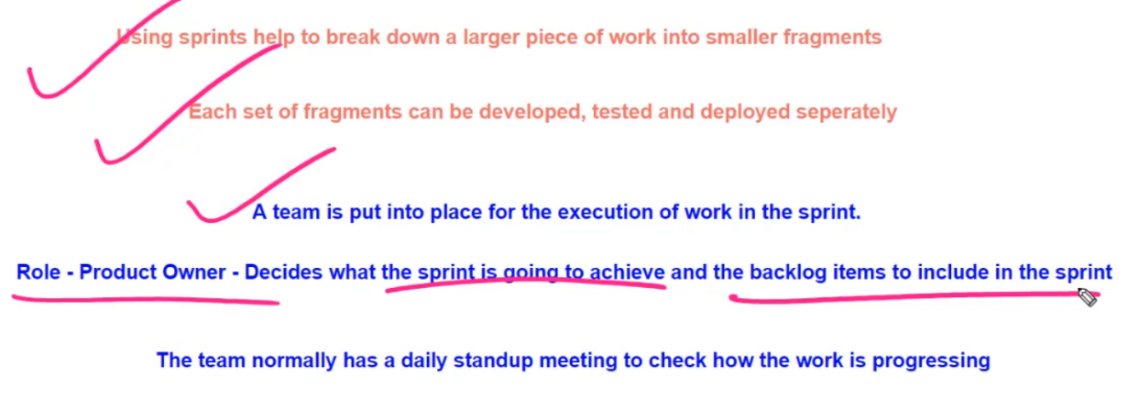
1. **Agile Project Management Process:**

So agile is a project management and a software development process, and the entire idea of agile is to go in and deliver software via small chunks of increments.

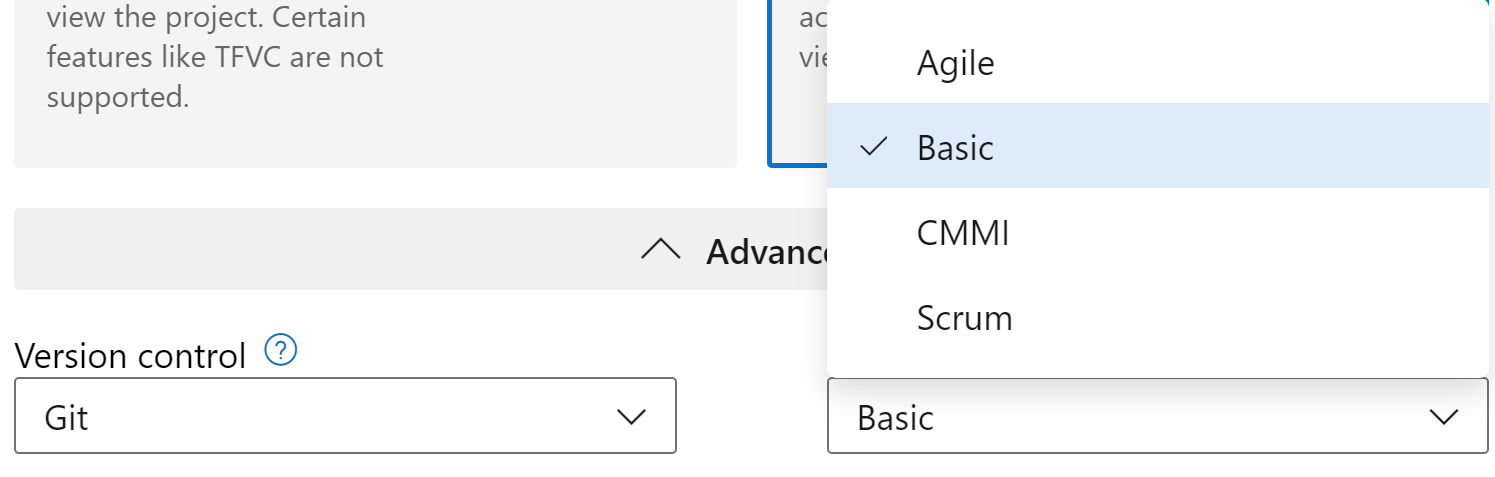
when we looked at the larger picture of the project lifecycle, which includes: the requirements phase -> the design phase -> the development phase -> the testing phase -> and the deployment phase over there Everything was sequential.

Now, in the agile approach, the entire idea is to kind of break it down. So you take some of the requirements -> You go ahead and develop something -> you go ahead and test it out -> and you deploy it. So on regular interval you give something to your customer.





1. **Work Item Process Available for Project:**

****

1. **Working with Git:**

Install the git on your machine -> Run command line

|  |  |
| --- | --- |
|  | Go to Kamal Workspace Directory -> create new folder “devops” -> enter into devops directory -> create two files  FileA.txt  FileB.txt |

|  |  |
| --- | --- |
|  | Dir -> to check the file added  Get more about Files  Git init command to initialise the git repository in local  Git add . -> to add the files into branch master  Git commit |

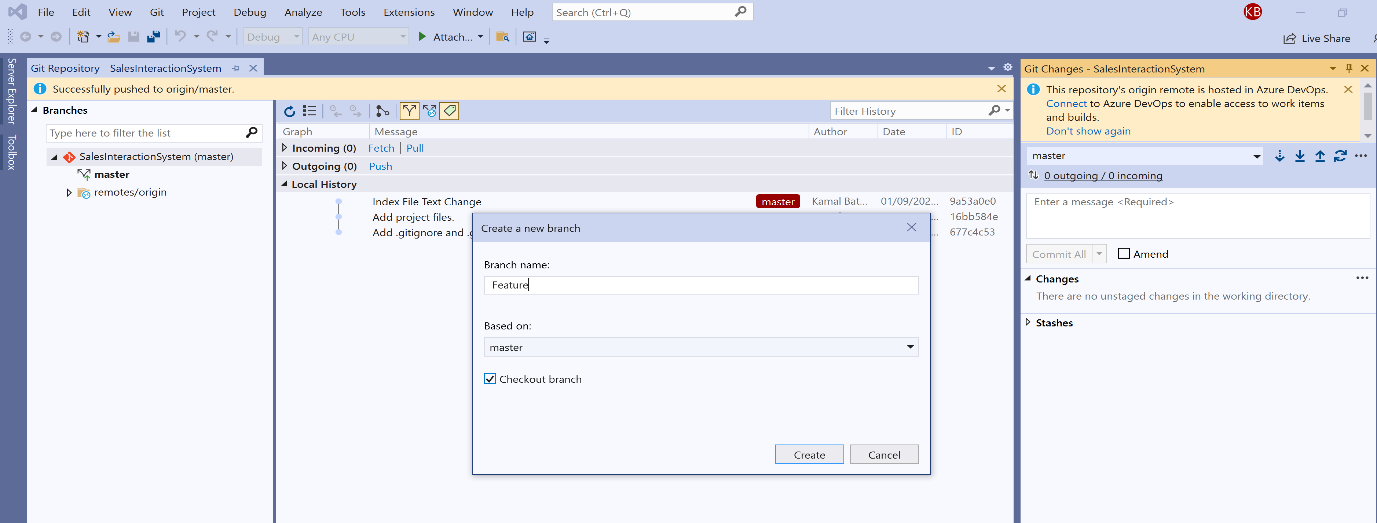
|  |  |
| --- | --- |
|  | Git status -> to check if nothing pending  Git log -> for version details  Git config –list -> to evaluate all configuration  If user Name and Email is missing then type last two command |

|  |  |
| --- | --- |
|  | Git checkout to revert the changes to previous commit |

In Order to publish the code that become available to other developer -> source code need to publish into central repository -> can be use

* Git Hub (on Internet)
* Git Hub Enterprise (for on-premise)
* Azure Repos ( On Azure DevOps Server)
  1. **Restrict Developer to Push the code directly into Master Branch**

Create a Feature Branch based on Master -> Then go to Azure DevOps

****

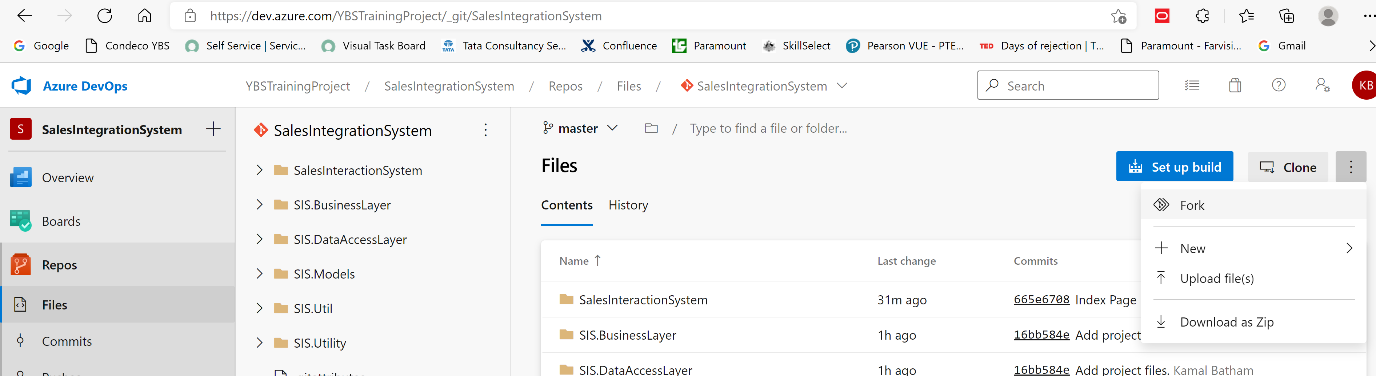
Select Master Branch -> Add Branch Policy

* 1. **To Avoid Specific file to check-in**

Use gitignore

* 1. **Forking the Repository**

**Your can fork the master branch, so that the developer can use this fork branch for new feature and push the changes.**

****

1. **Important Points**

The important commands are

* git init - This is to initialize an empty repository
* git add - This is to add files to the staging area from your working directory
* git commit - This is to commit files onto git
* git push - This is to push changes onto a remote repository
* git pull - This is to pull changes from a git repository
* git branch - This is for listing, creating and deleting branches
* git checkout - You can switch between branches with this command

#### Remember all the merge processes especially the squash merge

https://docs.microsoft.com/en-us/azure/devops/repos/git/merging-with-squash?view=azure-devops

#### Other configuration management tools

Giving the link for other configuration management tools

* Bitbucket - https://bitbucket.org/product
* Apache subversion - https://subversion.apache.org/
* GitHub Enterprise - https://github.com/enterprise

1. **How Azure Pipelines work Internally**
   1. **Starting with Azure Pipeline:**

So when Azure pipelines needs to go ahead and build your code, which is hosted in a git repository it requires a compute environment for building that application. It actually go ahead and temporarily build a new virtual machine on the cloud in Azure pipelines itself. You won't see this virtual machine, it'll be running in the background, as part of Azure pipelines, and then it will actually go ahead and carry out that build process.

So all of the code will be automatically taken from a git repository on this virtual machine. And then using the built tools, the code will actually be built on this virtual machine. And once the build is available, once a success or a failure, this virtual machine will be disposed.

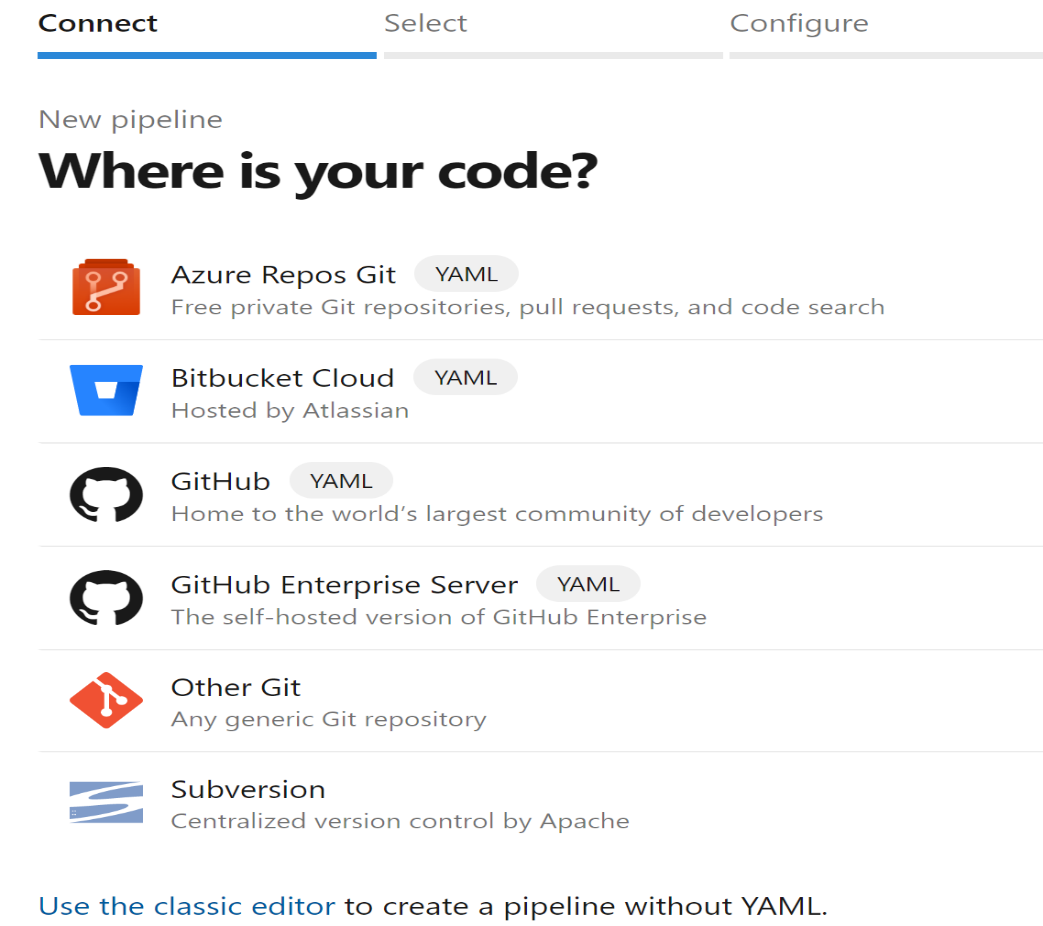
|  |  |
| --- | --- |
|  | This VM called as Agent |

So When you setup your Pipeline to build code -> you actually select one Windows Image -> This image will actually have visual studio already installed -> by this it build your code.

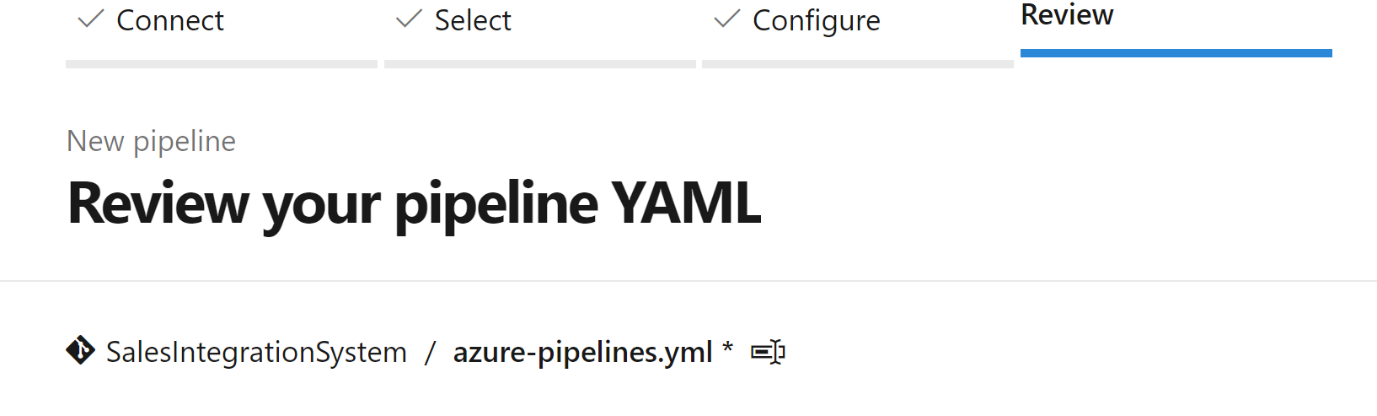
* 1. **Start Working with Azure Pipelines:**

Go to Pipeline -> Create Pipeline

It first ask -> where is your source code hosted in



Click Azure Repos -> Select Asp.Net Core (To Configure the Pipeline) -> .Yml file



# ASP.NET Core (.NET Framework)

# Build and test ASP.NET Core projects targeting the full .NET Framework.

# Add steps that publish symbols, save build artifacts, and more:

# https://docs.microsoft.com/azure/devops/pipelines/languages/dotnet-core

trigger: (This means- When any code change happen in master branch it actually

trigger this pipeline)

- master

pool:

  vmImage: 'windows-latest' (Use Windows latest image to build our code)

variables:

  solution: '\*\*/\*.sln'

  buildPlatform: 'Any CPU'

  buildConfiguration: 'Release'

steps:

- task: NuGetToolInstaller@1 (First step is to install the NuGet on machine

which use to restore all dependencies of your

source code)

- task: NuGetCommand@2 (Use the NuGet to restore the packages in your

solution)

  inputs:

    restoreSolution: '$(solution)'

- task: VSBuild@1 (Use Visual Studio to build your solution)

  inputs:

    solution: '$(solution)'

    msbuildArgs: '/p:DeployOnBuild=true /p:WebPublishMethod=Package /p:PackageAsSingleFile=true /p:SkipInvalidConfigurations=true /p:DesktopBuildPackageLocation="$(build.artifactStagingDirectory)\WebApp.zip" /p:DeployIisAppPath="Default Web Site"'

    platform: '$(buildPlatform)'

    configuration: '$(buildConfiguration)'

- task: VSTest@2

  inputs:

    platform: '$(buildPlatform)'

    configuration: '$(buildConfiguration)'

* 1. **Microsoft Hosted Agents**

The **Azure Pipelines** agent pool offers several virtual machine images to choose from, each including a broad range of tools and software.

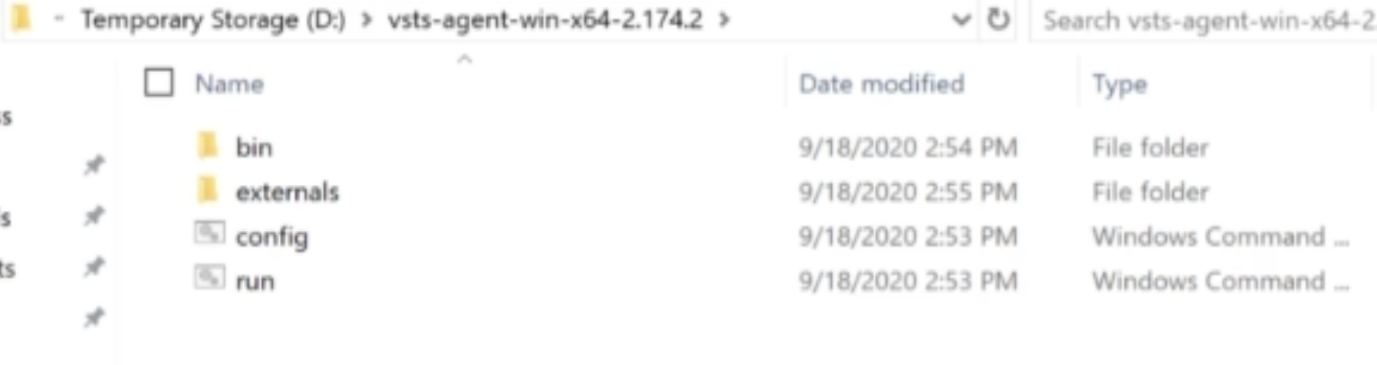
| **Image** | **YAML VM Image Label** | **Included Software** |
| --- | --- | --- |
| Windows Server 2022 with Visual Studio 2022 | windows-2022 | [Link](https://github.com/actions/virtual-environments/blob/main/images/win/Windows2022-Readme.md) |
| Windows Server 2019 with Visual Studio 2019 | windows-latest OR windows-2019 | [Link](https://github.com/actions/virtual-environments/blob/main/images/win/Windows2019-Readme.md) |
| Windows Server 2016 with Visual Studio 2017 | vs2017-win2016 | [Link](https://github.com/actions/virtual-environments/blob/main/images/win/Windows2016-Readme.md) |
| Ubuntu 20.04 | ubuntu-latest OR ubuntu-20.04 | [Link](https://github.com/actions/virtual-environments/blob/main/images/linux/Ubuntu2004-README.md) |
| Ubuntu 18.04 | ubuntu-18.04 | [Link](https://github.com/actions/virtual-environments/blob/main/images/linux/Ubuntu1804-README.md) |
| Ubuntu 16.04 | ubuntu-16.04 | [Link](https://github.com/actions/virtual-environments/blob/main/images/linux/Ubuntu1604-README.md) |
| macOS 11 Big Sur | macOS-11 | [Link](https://github.com/actions/virtual-environments/blob/main/images/macos/macos-11-Readme.md) |
| macOS X Mojave 10.14 | macOS-10.14 | [Link](https://github.com/actions/virtual-environments/blob/main/images/macos/macos-10.14-Readme.md) |
| macOS X Catalina 10.15 | macOS-latest OR macOS-10.15 | [Link](https://github.com/actions/virtual-environments/blob/main/images/macos/macos-10.15-Readme.md) |

Click on link and you can see what are the tools that are installed.

1. **Azure Pipeline – Self Hosted Agents**

Here, instead of Azure DevOps agents (VM) we want self-hosted VM to build source code. There are couple of software required to Install on this machine first.

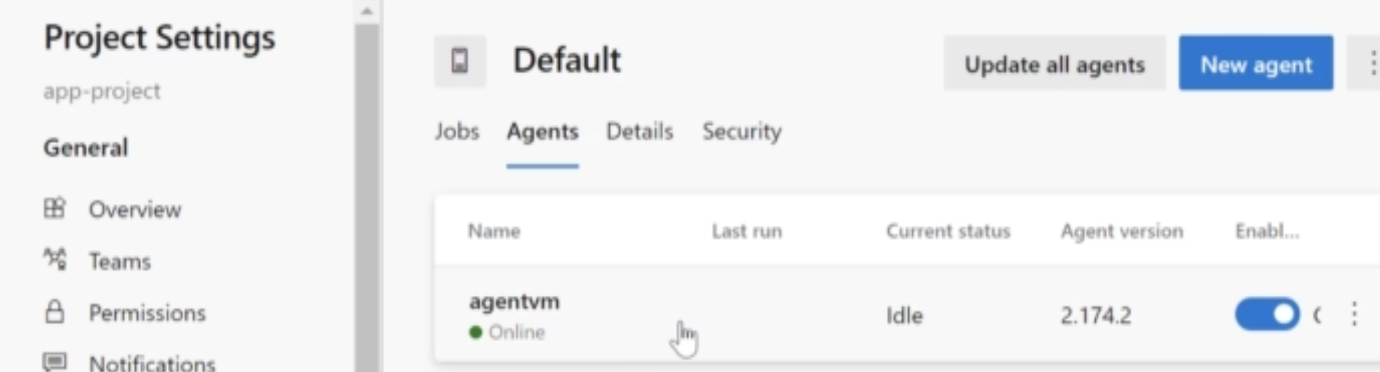
1. Install Git.exe -> This is required to checkout the code from git repository.
2. Install Nuget.exe -> This is used to restore the packages for the .Net Solution
3. Install Visual Studio and .Net Core for the Build of .Net Solution.
4. Azure Pipeline Agent -> It used to communicate with Azure pipelines in DevOps organization (Go to Project Settings -> Agent Pool -> New Agent -> Download)



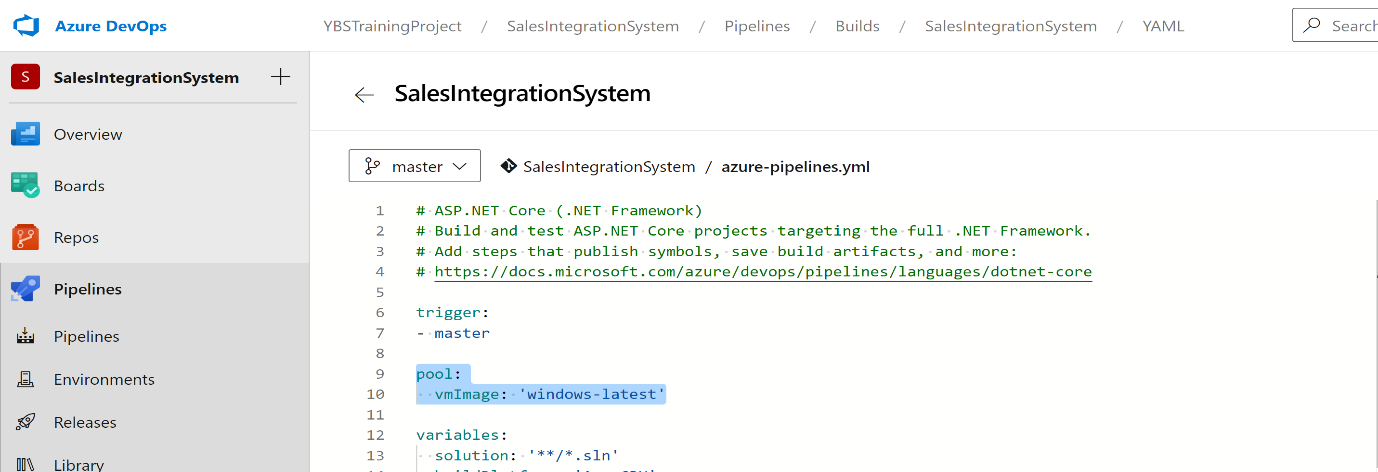
1. Now Open Powershell -> and run the command like below



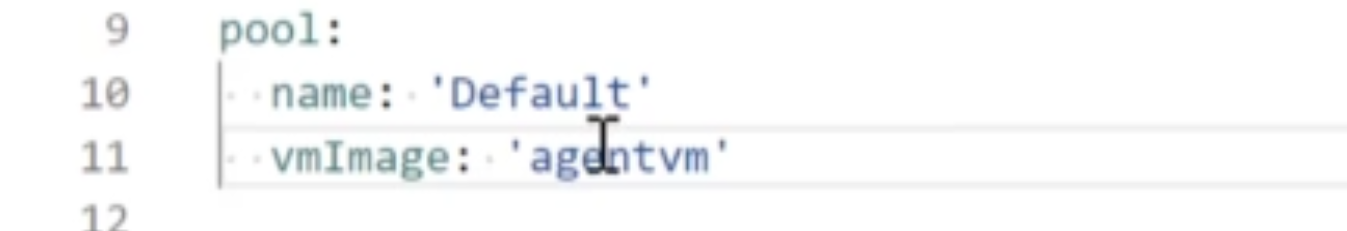
1. It will ask for PAT (Personal Access Token) -> Go to User Settings -> Click Personal Access Token -> Create a New Token -> Pass the PAT in above command
2. Once done, you will see the Agent created



1. Now use this agent for your build pipeline -> Go to your project Pipeline ->

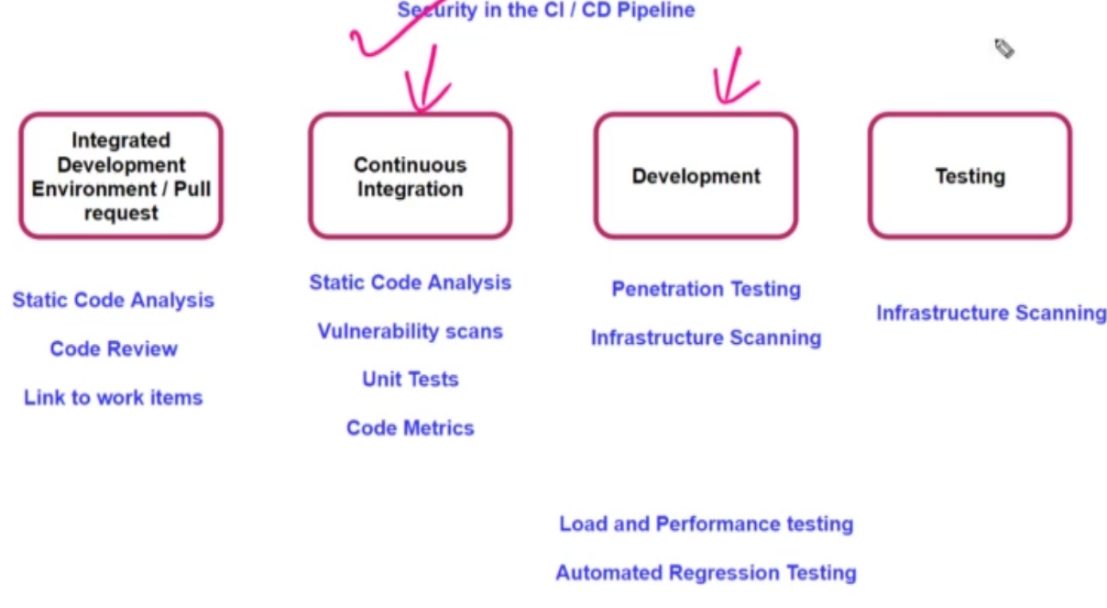


1. Change the pool above as below, and save -> Now when you make any change build process will done from Self Hosted Machine



1. **Security in CI/CD Pipeline**

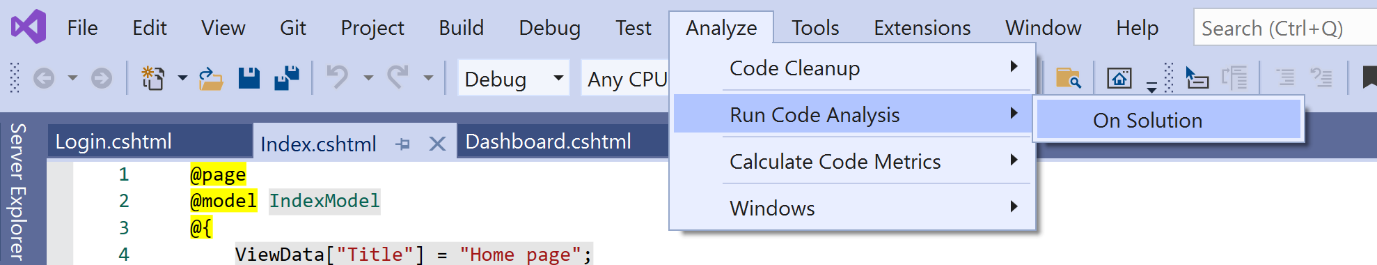
when it comes to automation, you also have important practices such as testing in place and also including the security of the application, which is being built in the CI/CD pipeline. Now, over here, I am showing you the different phases when it comes to the CI/CD pipeline, and security that we have to adhere in each phase.

****

1. **Source Code Analysis Tool:**

It can actually go ahead and give you either defects, issues or recommendations on how you can actually go ahead and improve on your code. So these rules are based on best practices in the industry.

We can do code analysis in VS as below

****

We can use other code analysis tools, which can give you more insights onto the code being developed. Like FX Cop

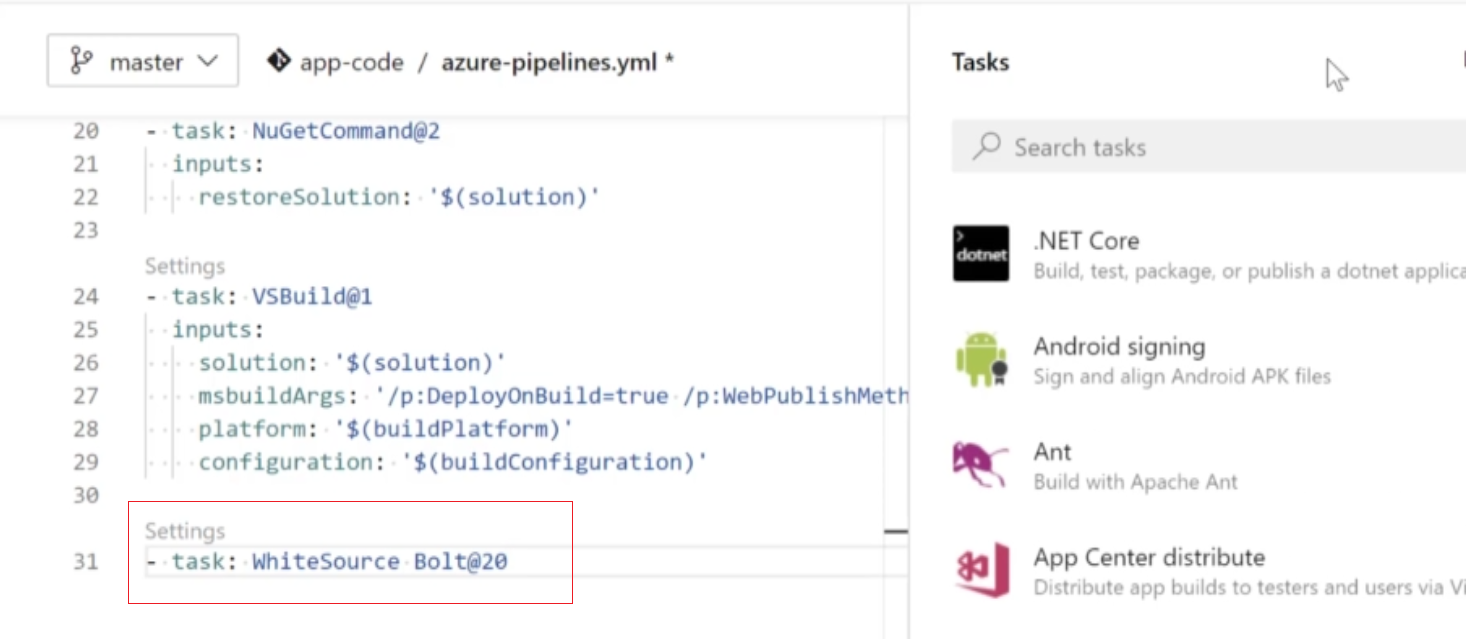
1. **WhiteSource Bolt & Black Duck Detect:**

During the development phase, your developers might be using open source packages, which are in place. now since these open source packages are free of cost. Your developer might go ahead and integrate those packages which the project itself.

But even if these open source components are free, it does not mean that they don't have any sort of security vulnerabilities because you are not responsible for those open source packages. You are not sure if those packages itself in the code within the packages are secure enough. Or maybe you're not even sure when it comes to the license considerations of these open source packages.

You can actually go ahead and use tools that can actually integrate, with Azure develops for going ahead and detecting or scanning for such vulnerabilities. So one of those tools which is available is White Source Bolt or black duck detect.

Go to DevOps organization -> Organization Settings -> Extension -> Browse Market Place -> search WhiteSource Bolt – Get it for your DevOps -> configure your yml file

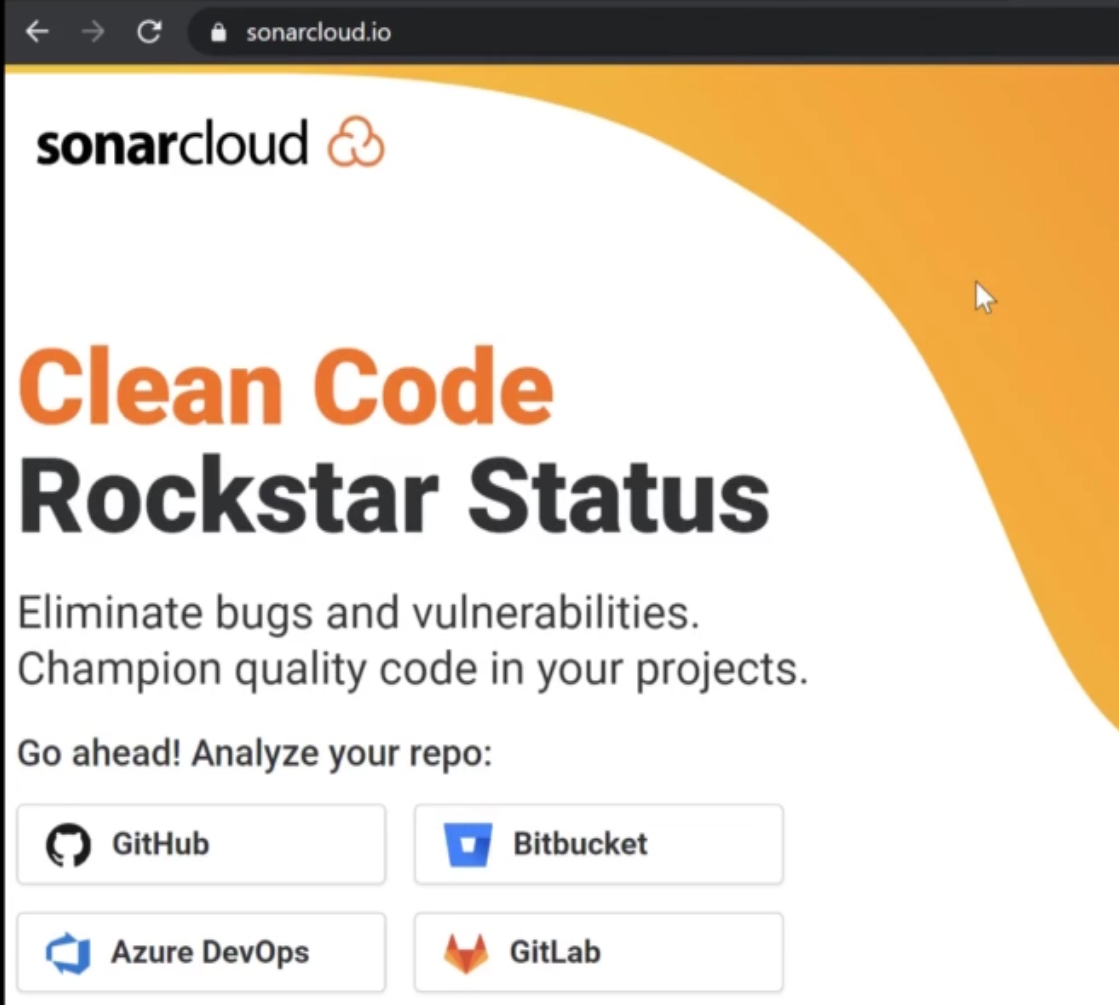


1. **Improving Code quality & Security**

SonarQube has the ability to go out and look at your code and give you recommendations on how to improve code quality and security. It has support for a large number of programming languages. So over here, you can get your bugs, your vulnerabilities by actually looking at your code.

You can install it on a virtual machine, on a server, and then you can go ahead and implement Sonar Qube in your continuous integration lifecycle or even in your integrated development environment.

**We also have SonalCloud** - it's available on the cloud. over here We don't have to download anything. We can directly do the integration of this tool with our code or with our continuous integration lifecycle.



**Step 1:** Click Login -> sign up using devops authentication.

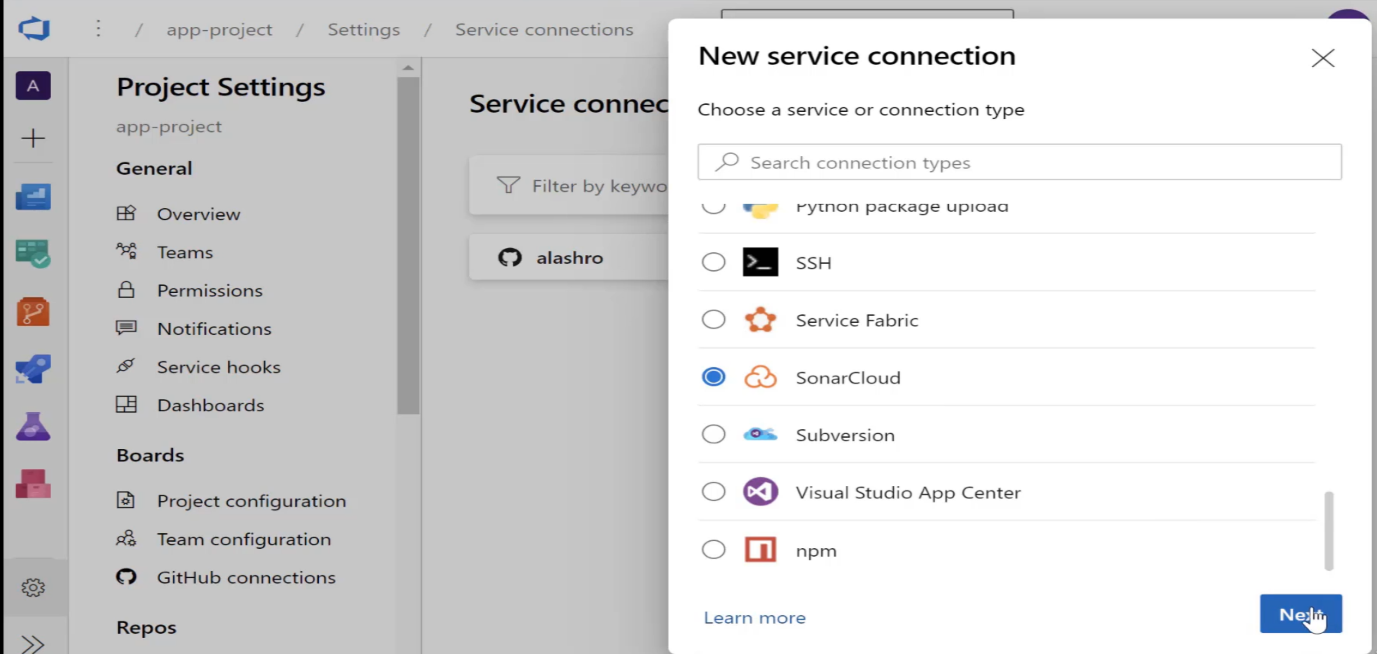


**Step 2:** Create organization -> Import Organization from Azure

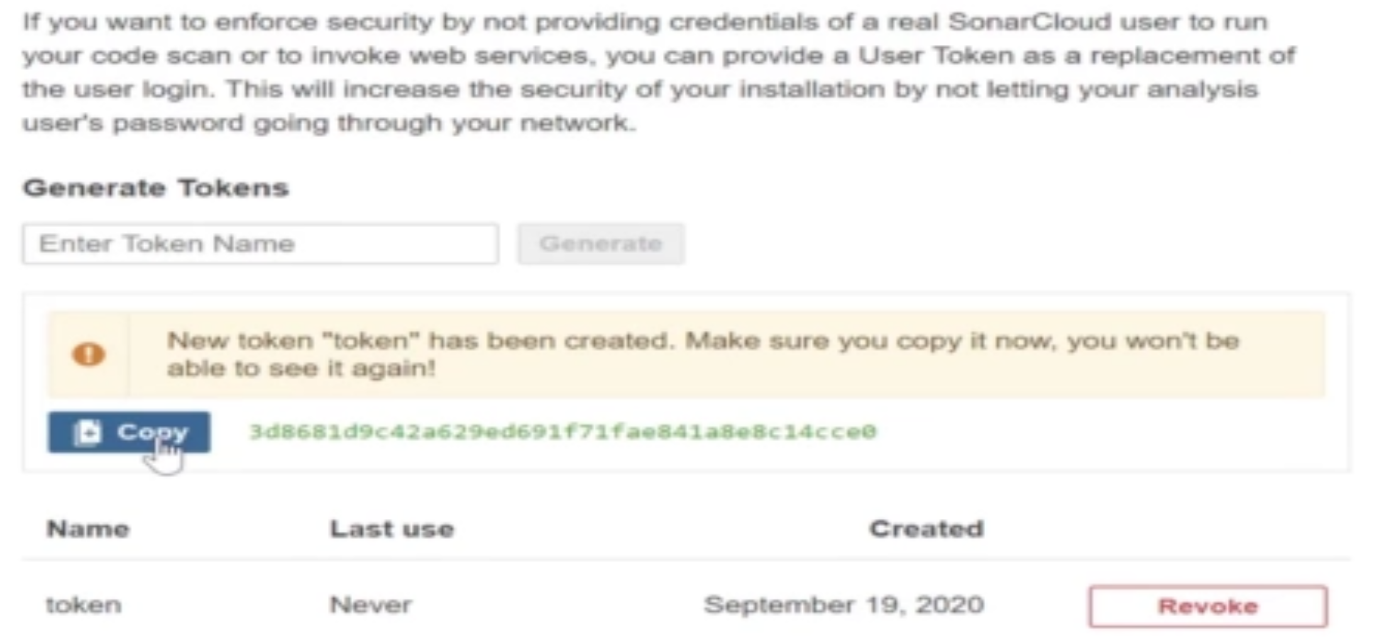
**Step 3:** Now for DevOps to interact with SonarCloud -> Go to your Organization Settings -> Extension -> Browse the Market Place -> Search SonalCloud -> Install It for Azure DevOps Organization

Now, once we have the extension in place, the next step is to go ahead and add a service connection for my project, our Azure DevOps project onto our instance of sonar cloud, that is on to our organization in Sonar cloud

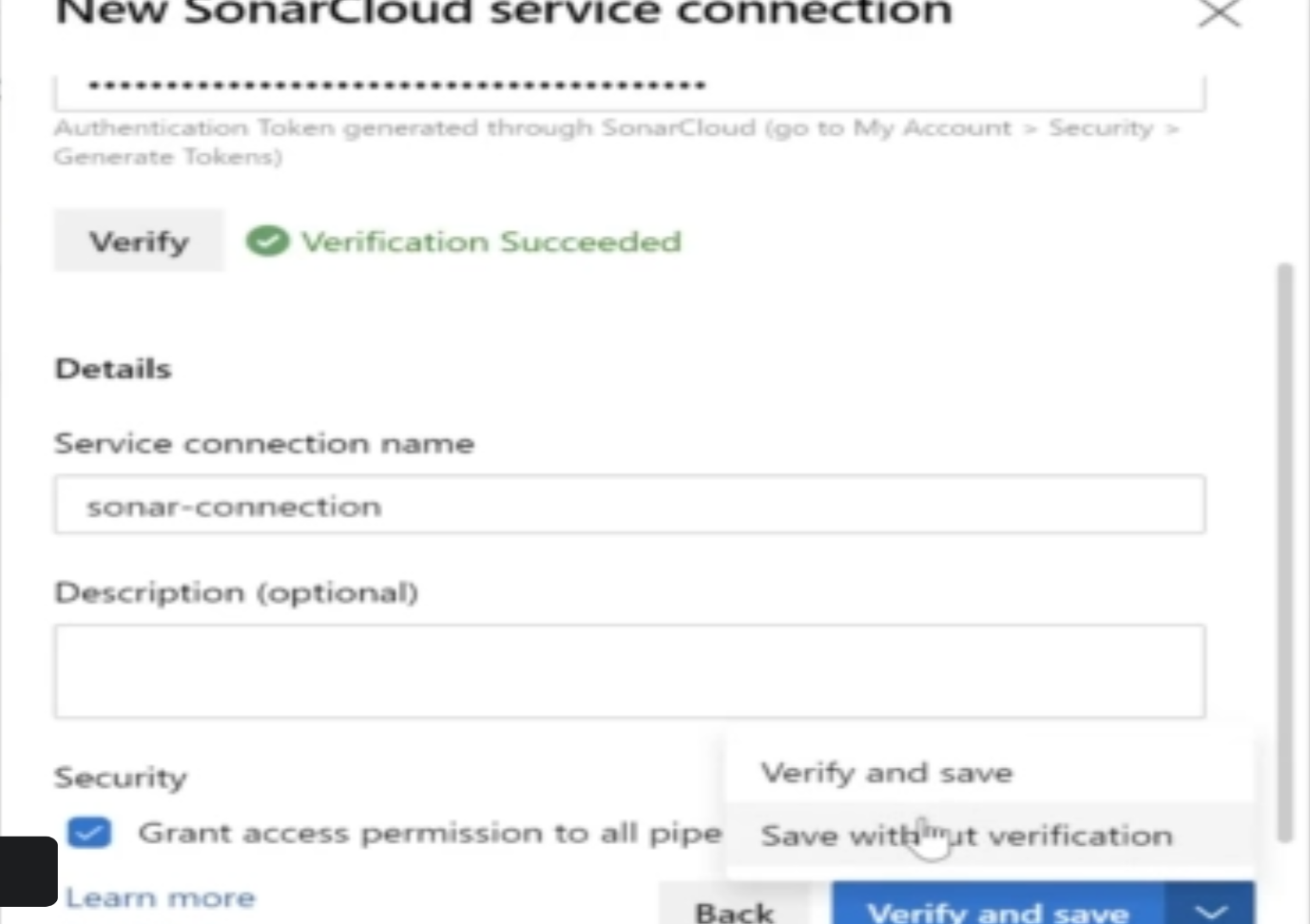
Go to Your Project -> Project Settings -> Click Service Connection -> New Service Connection -> Select SonalCloud -> Click Next



**Step 4:** It will ask for Sonal Cloud Token -> Go to SonalCloud, Click on My Account -> Security -> Here you can generate the Token

****

**Step 5:** Go back to DevOps -> input token and verify ->

****

**Step 6: Go to Project Pipeline YML file and Include required configuration as below**

 - task: NuGetCommand@2

   inputs:

     restoreSolution: '$(solution)'



 - task: SonarCloudPrepare@1

   inputs:

     SonarCloud: 'sonar-connection'

     organization: 'app-org'

     scannerMode: 'MSBuild'

     projectKey: 'app-project'

     projectName: 'app-project'

 - task: VSBuild@1

   inputs:

     solution: '$(solution)'

     msbuildArgs: '/p:DeployOnBuild=true /p:WebPublishMethod=Package /p:PackageAsSingleFile=true /p:SkipInvalidConfigurations=true /p:DesktopBuildPackageLocation="$(build.artifactStagingDirectory)\WebApp.zip" /p:DeployIisAppPath="Default Web Site"'

     platform: '$(buildPlatform)'

     configuration: '$(buildConfiguration)'



 - task: SonarCloudAnalyze@1

 - task: SonarCloudPublish@1

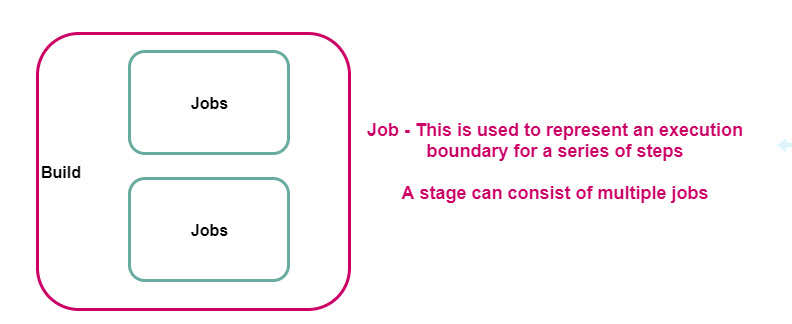
   inputs:

     pollingTimeoutSec: '300'

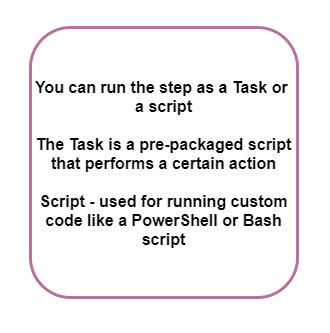
1. **Important Point**

Azure Pipelines can be used to automate your build and your release processes. At the moment we have looked at the Continuous Integration process that can be done with Azure Pipelines. Later on we will look at Continuous Delivery with the help of Azure Pipelines.

In Azure Pipelines, you have many stages. Each stage is a logical boundary in the pipeline. In each stage you can have a set of jobs



Each job can consist of PowerShell/Bash scripts or tasks



**Microsoft-Hosted agents and Self-Hosted agents**

Remember , you have various Microsoft Hosted agents that can be used to run your builds.

https://docs.microsoft.com/en-us/azure/devops/pipelines/agents/hosted?view=azure-devops&tabs=yaml

If these agents don't meet the build requirements of your application, you can decide to create your own self-hosted agent.

**Code Coverage Tools**

You can use Code Coverage tools to get a better idea on what code is not executed and required for your application.

For .Net based applications, you can use the .Net Core CLI task to generate code coverage reports.

For Java based applications, you can use the Cobertura tool

https://cobertura.github.io/cobertura/

**Technical Debt**

* This could be anything that could slow or hinder the entire development process.
* Especially when it comes to developing code, the amount of technical debt can increase over time.
* As the Technical Debt increases , it can become more difficult to make changes to code.

**What are the situations that can cause technical debt**

* When developers are under pressure to deliver and the application code is not up to the mark.
* Over time because of improper code , subsequent changes to code that uses bad coding practices can become more difficult.
* When the design of the application is not done properly.
* Then the subsequent code base will not be proper as well.
* When reviews are not conducted or refactoring – This helps to see how the code can be improved.
* Writing complex code that becomes difficult to change later on.

**Using Code Quality tools**

Remember , you can use SonarCloud and SonarQube tools for measuring code quality

https://sonarcloud.io/

https://www.sonarqube.org/

**Static Code Analysis Tools**

There are various Static Code Analysis Tools

For Java you have PMD - https://pmd.github.io/

For .Net you can use Visual Studio itself - https://docs.microsoft.com/en-us/visualstudio/code-quality/roslyn-analyzers-overview?view=vs-2019

1. **Custom Script Extension:**

Let's say that when you create a virtual machine in Azure , you want to ensure that there is some application or software that comes installed with the virtual machine. So let's say you want executable to run automatically when the machine is first booted up. so you can actually go out and make use of custom script extensions.

Example: IIS Installed on machine

**Step 1:** Create a file named IIS.ps1 with the following contents.

* import-module servermanager
* add-windowsfeature web-server -includeallsubfeature

If you are a windows local machine , create a file in notepad Add the above contents And then save the file as a PowerShell file

1) Choose the Save as type as "All Files"

2) Next give the file name as "IIS.ps1"

**Step 2:** Load this file to storage account -> Create New Storage Account on same location where VM -> Create a Container -> Upload IIS.ps1 file

**Step 3:** Go to your VM on Azure -> Extensions -> Add Extension -> Select Custom Script Extension -> Browse Script File from storage account

1. **PowerShell Desired State Configuration -**

You can do the same as above using PowerShell Desired state configuration. The core difference between power shell desired state configuration and custom script extensions is that, in custom script extensions, that script will run only once.

However, PowerShell Desired state configuration also run once but you can combine it with Azure Automation. And benefit is that, you can manage if something accidently corrupted in VM.

For example: If someone goes ahead and uninstalls internet information services, Azure automation will again go ahead and apply the script to install Internet information services. so you can go ahead and combine the power of power shell.

**Step 1:** Now for power shell desired state configuration for the implementation we need two files.

* 1. So the first file is basically the configuration file - basically specifies what should be the configuration of the target virtual machine.

Create a file named WebRole.ps1 and add the following contents

Configuration DScConfiguration {

Node $AllNodes.Where{$\_.Role -eq "WebServer"}.NodeName

{

WindowsFeature IISInstall {

Ensure = 'Present'

Name = 'Web-Server'

}

}}

You need to also zip this file.

* 1. Create a data configuration file as InstallWebconfig.psd1 and add the following contents

@{

AllNodes = @(

@{

NodeName = "localhost"

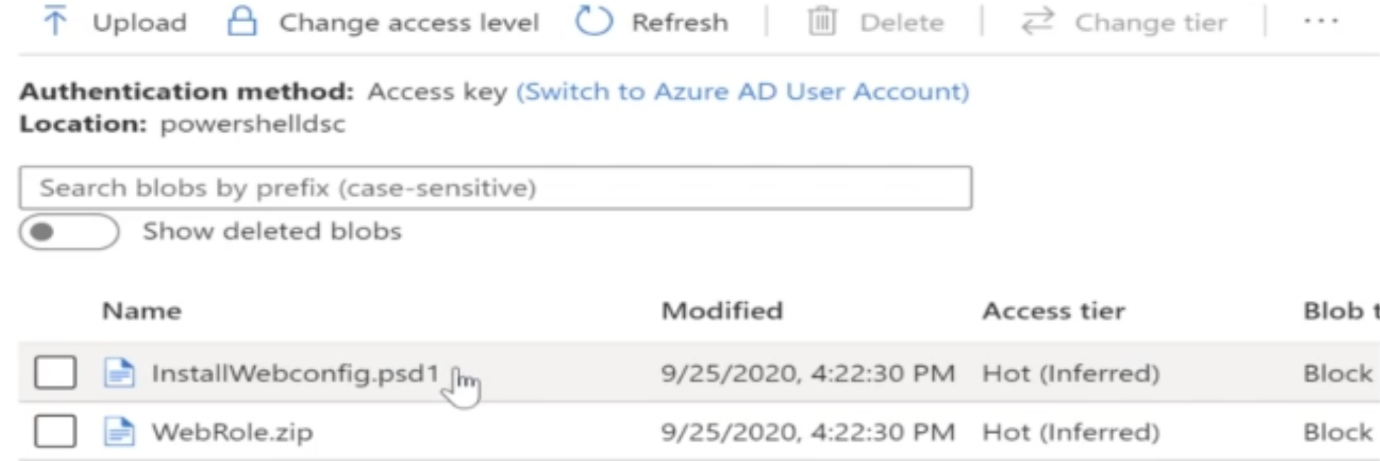
Role = "WebServer"

}

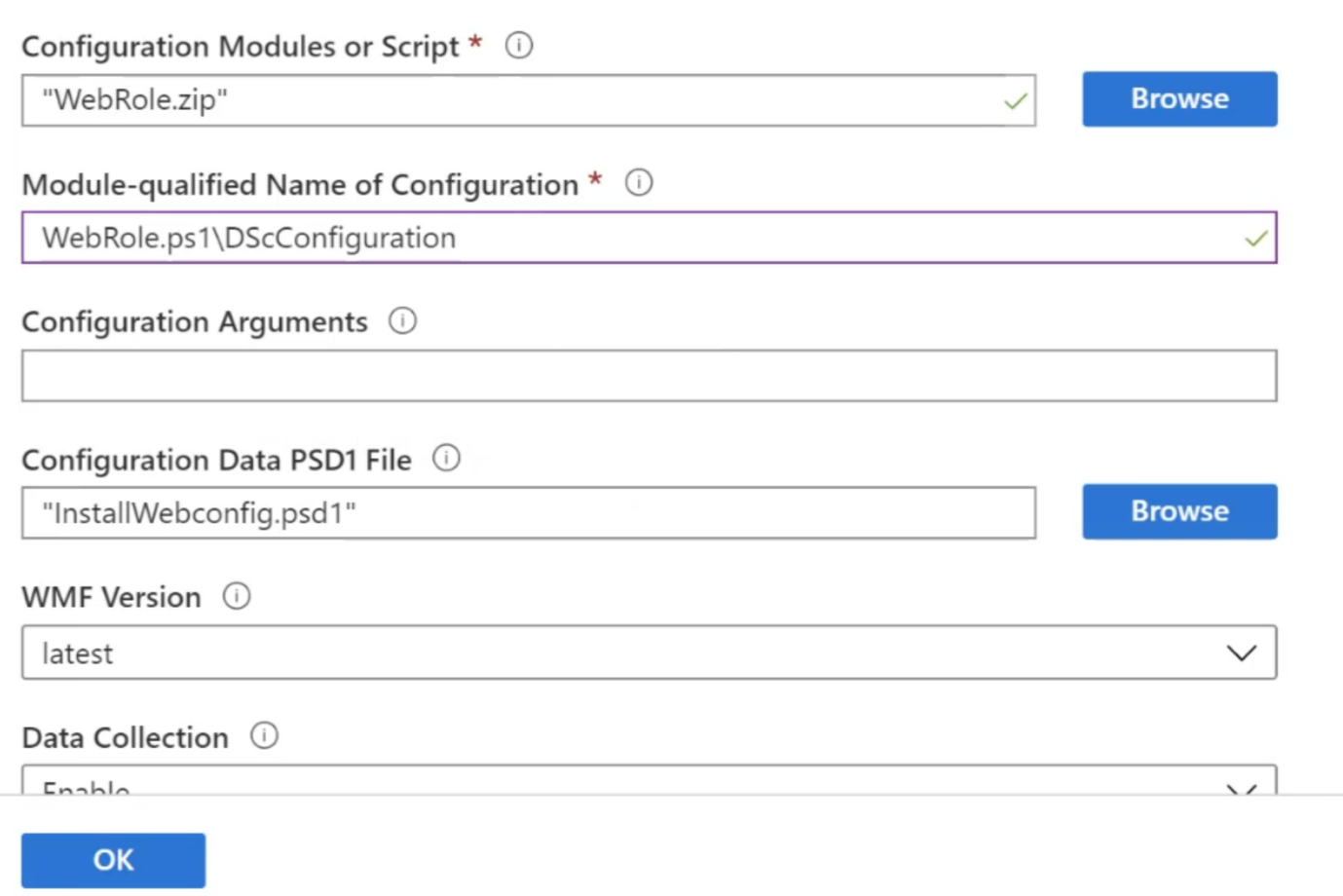
)

}

**Step 2:** Add these files into Storage Account



**Step 3:** Go to your VM on Azure -> Extensions -> Add Extension -> Choose PowerShell Desired State Configuration -> Browse Script File from storage account

****

1. **Azure Automation:**

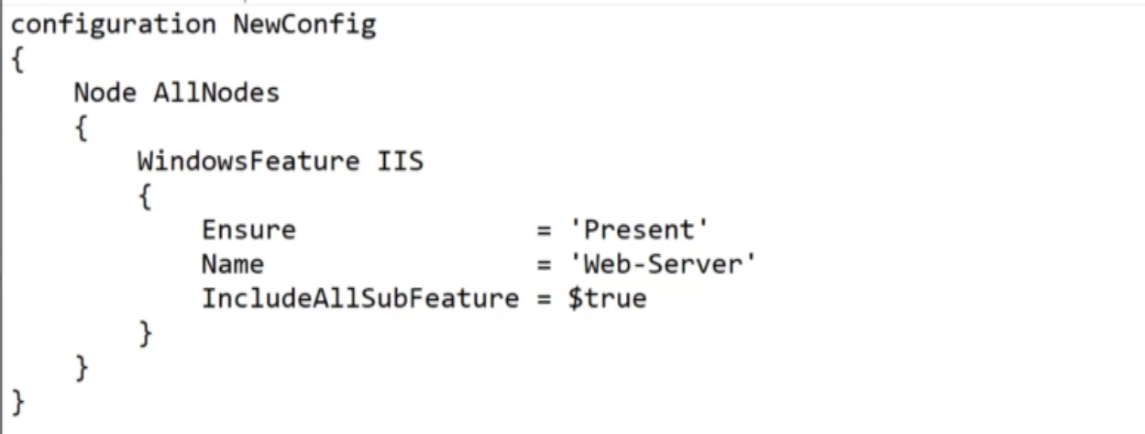
So let's say that you have a set of virtual machines in place and you want to go out and maintain the state of these virtual machines. We can use Azure Automation with PowerShell Desired State configuration to maintain the state of VM.

Azure automation use an internal desired state configuration pull server. It go ahead and look at the configuration of these virtual machines, and if it finds any drift or any mismatch in ideal configuration, then it'll go ahead and maintain the configuration of these virtual machines.

**Step 1:** Create a Azure Automation account

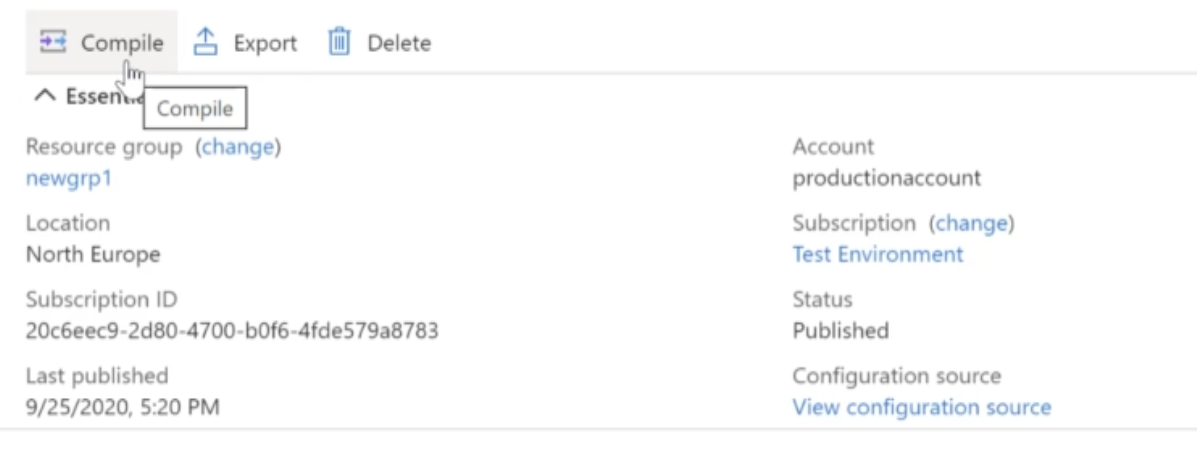
**Step 2:** Then will go ahead and upload a power shell DSC script. (This script will be bit different than above)

**Go to Azure Automation -**> Click State Configuration -> Go to Configuration Tab and click Add button -> Upload the below configuration file



**Step 3: Compile the script in Azure Automation**

Once above .ps1 file loaded -> Click on the file and compile it -> Once compiled go to compiled Configuration tab you can see script compiled

****

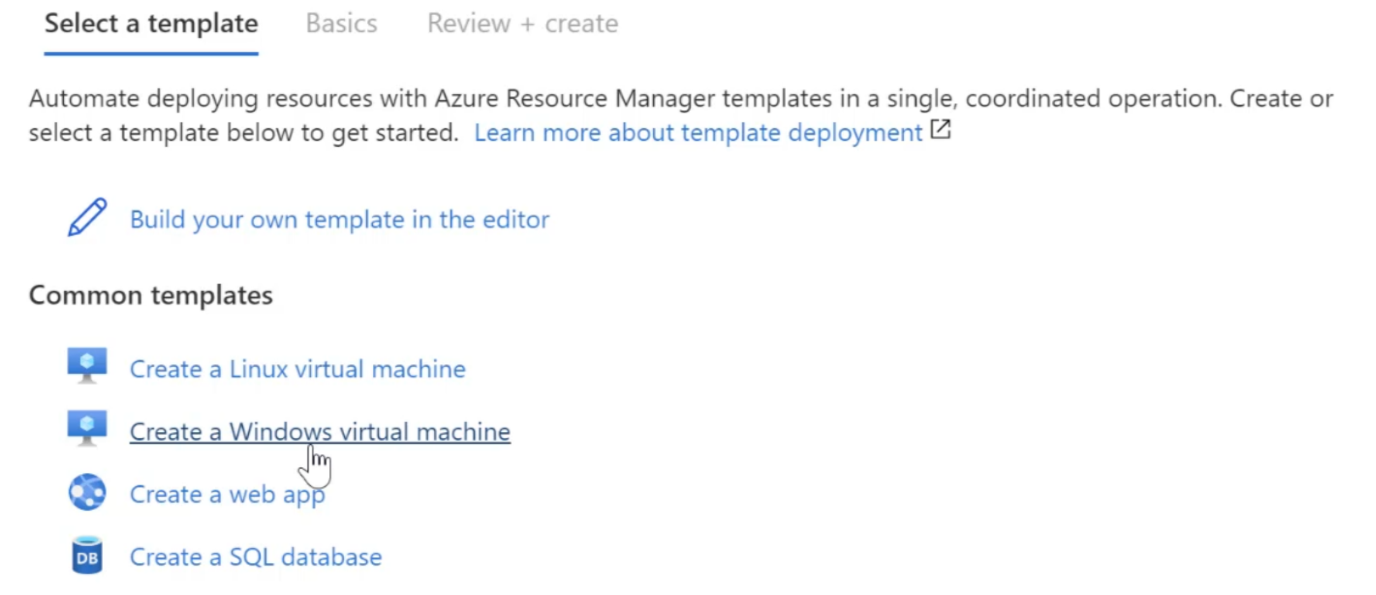
**Step 4: Associate the DSC configuration with VM**

Go to Nodes -> Add Node (Means add VM on which this need to be applied) -> complete the registration

|  |  |
| --- | --- |
|  | Click ok -> Now if you go and login VM and uninstall the ISIS -> it automatically install it. |

1. **ARM Template Deployment- VM:**

Go to Azure -> Search new Resource (Template Deployment) -> Click Create



Click Create Windows virtual machine-> now you need to edit the template and provide parameters

Note: Usually template contains the default value of all the parameter instead of (UserName, Password and DSNLabelPrefix), so either go manually to provide these details or use JSON file parameters.json as below) to offer these value to pass in template when go for deployment.

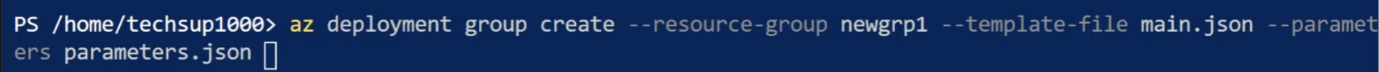


Download the Template file and store somewhere along with parameter.json file

Go to Azure -> Go to Virtual Machines -> Open Azure Cloud Cell (It needs storage account, if you don’t have create storage account here) -> Upload both file into Azure Cloud cell



Execute below command to deploy the VM



1. **Nested and Linked Template:**

Let's say that you want to go ahead and deploy a set of resources using ARM templates. So let's say you want to go ahead and deploy your virtual machine an Azure SQL database and an Azure storage account, Now, you could go ahead and create a single json template that can be used to deploy all of these resources.



But what if you have many resource need to deploy -> It would be difficult to maintain single template for all resource->

So what you can do to create a separate template for each resource -> Now one main template that could go ahead and call each one of these templates as linked templates.



So with the help of nested templates, we now have the ability to deploy these resources across different resource groups.

1. **ARM Template – Custom Script Extension:**

When you want to go ahead and apply some configuration onto the virtual machine -> So we are going to use same as above one ARM template and Parameters. Json file. The main difference I'm making is there is another section this is -> Microsot.Compute/VirtualMachines/Extensions.

Here actually mentioning the Type as custom script extension.

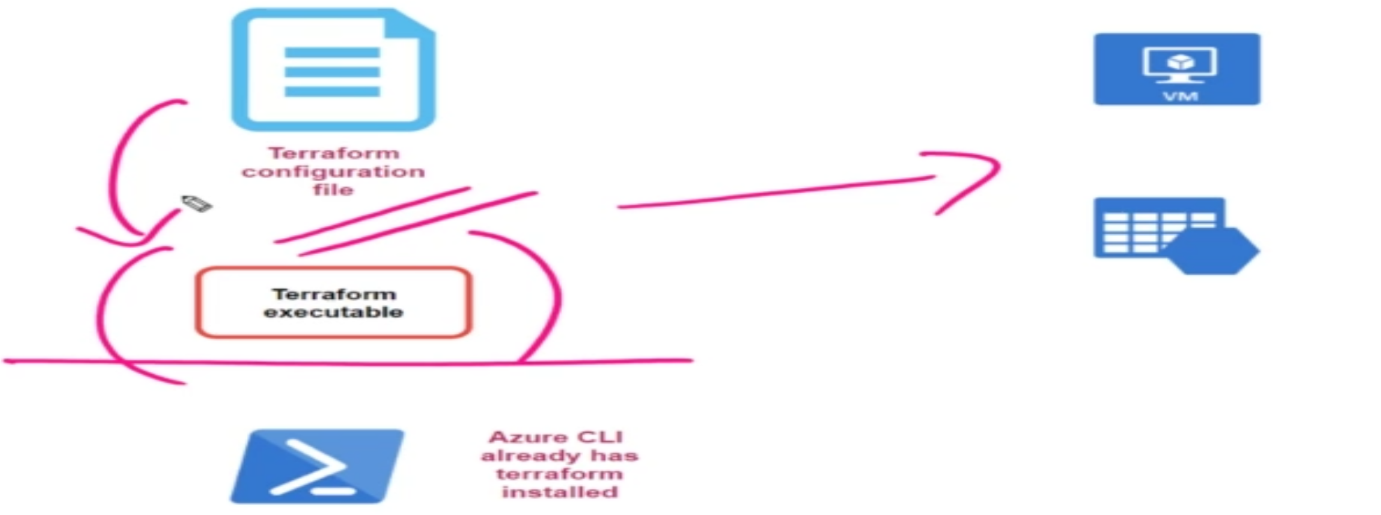


It's installing Internet information services on a virtual machine and install the hosting bundle, that Asp.Net hosting bundle on a virtual machine.



1. **Terraform:**

Using Terraform - you can also go ahead and design your templates, so this is another infrastructure as code solution. One of the advantages of using TerraForm is that you can go ahead and define templates that could deploy resources either onto Azure or onto other cloud providers such as Amazon Web Services.



In Azure, you have to go out and download the terraform executable, which is available on the terraform site in Azure So this executable will run on your local machine, you can go out and submit the terraform configuration file onto the terraform executable, and that executable will actually go ahead and deploy the resources accordingly.

Azure cloud shell already has terraform installed, so we can use Cloud shell.

1. **Setup Azure DevOps On-Premises:**

I am trying to setup Azure DevOps Server on my one Window 10 machine. Please find below Pre-requisite.

1. Install Git.exe – This is required to checkout the code from git repository. (Whether the code is in Git Repos or Git Hub)
2. Install Nuget.exe – This is used to restore the package for .Net Solution
3. Install Visual Studio and .Net Core for the build of .Net Solution.
4. Install Java 11+
5. Install Microsoft SQL Server – Used by Azure DevOps of creating required entities for repositories and pipelines
6. Download Azure DevOps Server and Install it on machine.

