# Azure DevOps – CI/CD

1. **Difference – Continuous Delivery & Deployment**

There are two terms that you should be aware of. So first is the term of continuous delivery and the other is of continuous deployment.



**Continuous delivery -** when it comes to continuous delivery, so here the pipeline needs to be able to deploy the application on to a production-based environment. This entire thing might not be automated.

**Continuous deployment -** which is the next level of continuous delivery. This is where the entire process from the code commit on to the deployment is completely automated.

You need to have a lot of trust in the code changes because over here we are saying that as soon as the code changes are pushed on to the repository, they should be deployed onto the production based environment.

1. **Azure Repository Commit Error:**

You may encounter two types of error if you configured DevOps on Https

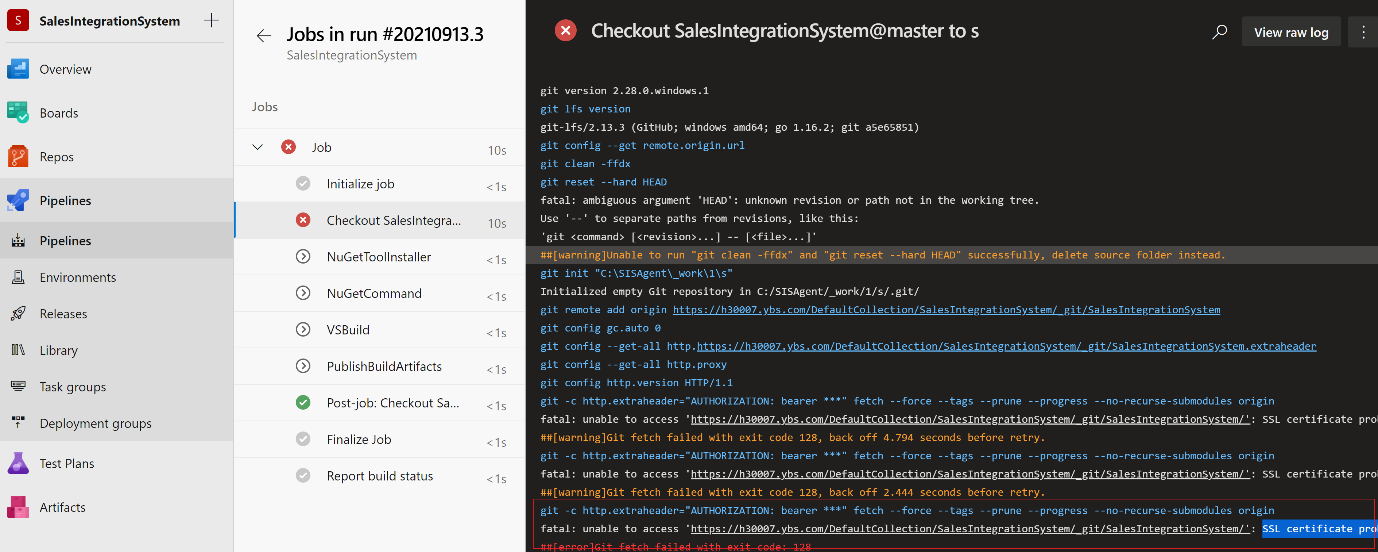
* 1. **Error When Pushing Code from Visual Studio**

When you created new repository and trying to push you code into it and error appear: SSL certificate problem: unable to get local issuer certificate

It means your DevOps using Https and local Git required SSL certificate.

Solution:

* Close all instances of Visual Studio
* Open a command prompt as administrator
* cd to the installation directory for VS2017’s Git -> C:\Program Files (x86)\Microsoft Visual Studio\2019\Professional\Common7\IDE\CommonExtensions\Microsoft\TeamFoundation\Team Explorer\Git\mingw32\bin
* Run the command: git config --system http.sslbackend schannel
* Reopen Visual Studio and try to push the code again.
  1. **Error When Build Pipeline trying to checkout the code from repository**



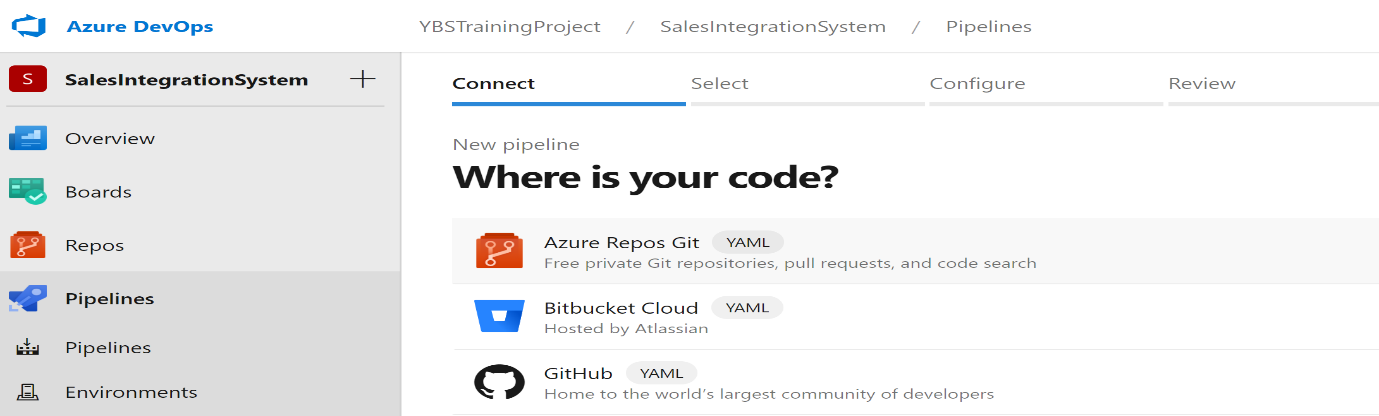
This is because git is unable to checkout your source code from Azure Repository for Build due to secure HTTP configuration.

Solution:

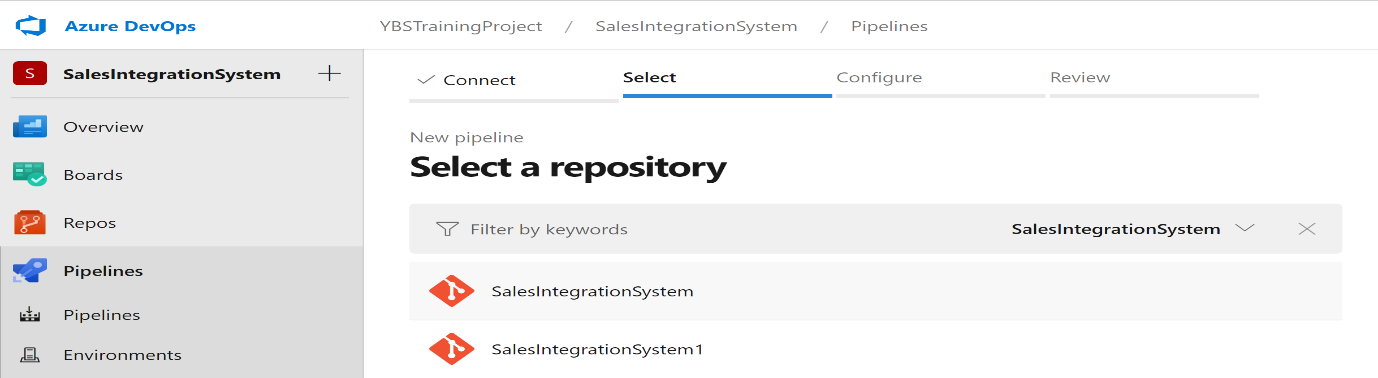
* Open a command prompt as administrator
* cd to the installation directory for Git -> C:\Program Files\Git\mingw64\bin
* Run the command: git config --system http.sslbackend schannel
* Retry to Run the Build Pipeline.

1. **Azure Build Pipeline:**

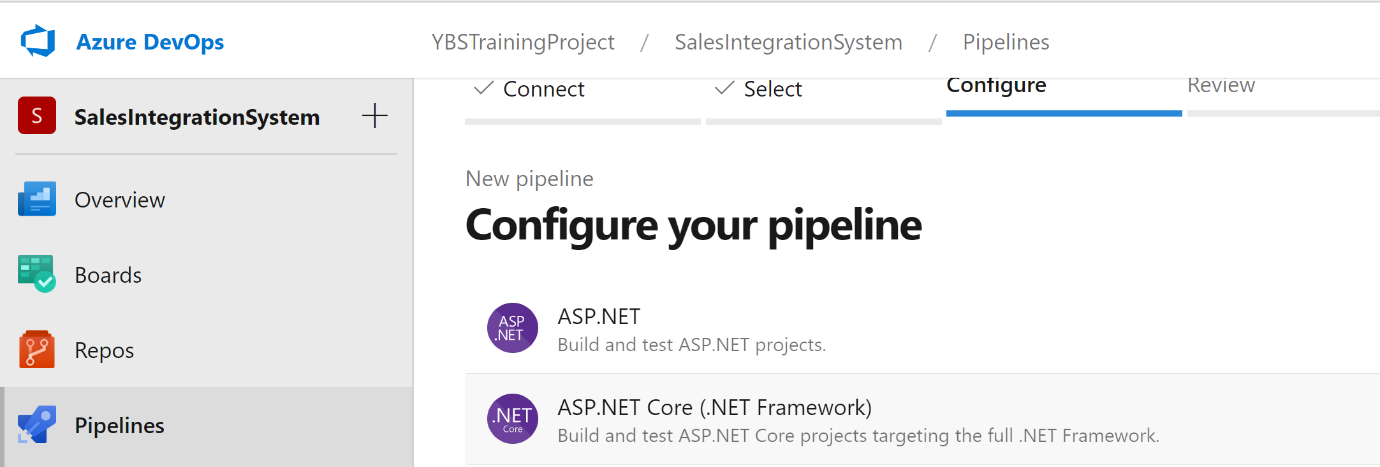
**Step 1:** First go ahead and create a new built pipeline -> hit on new pipeline -> choose Azure repose git



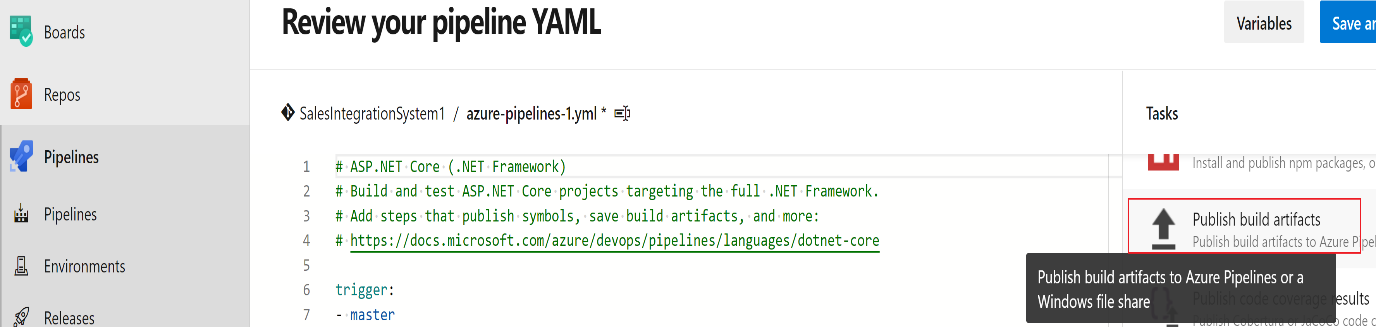
**Step 2:** Select your repository where your source code available



**Step 3:** Configure the version of you source code

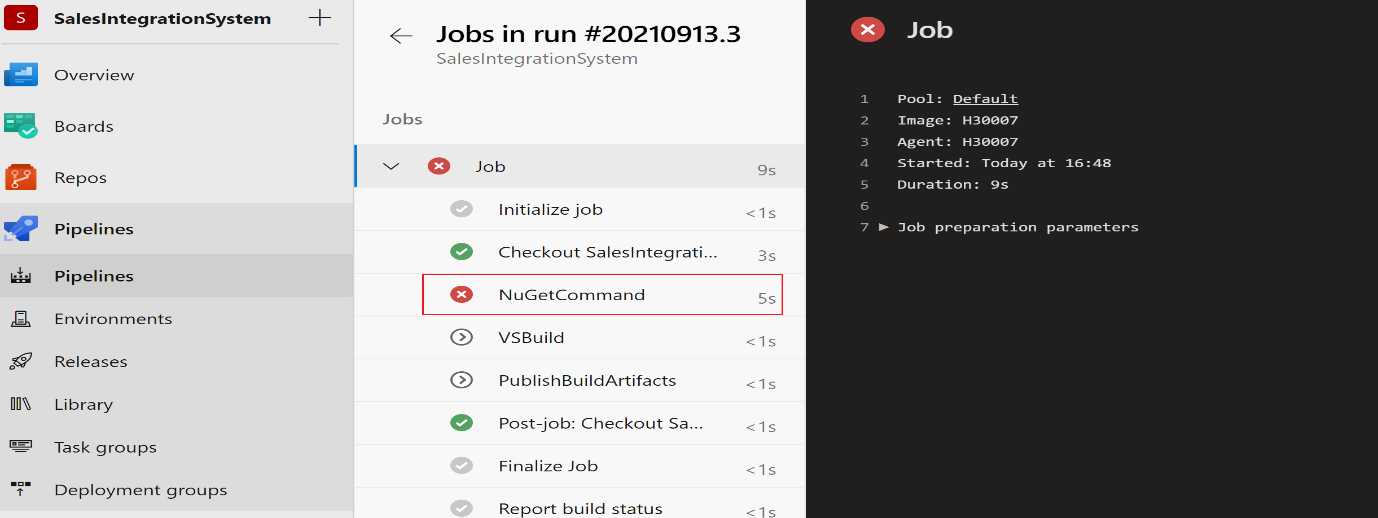


**Step 4:** Add new task into your build pipeline - this task is basically the published task, so we want to go ahead and publish the build artifacts.

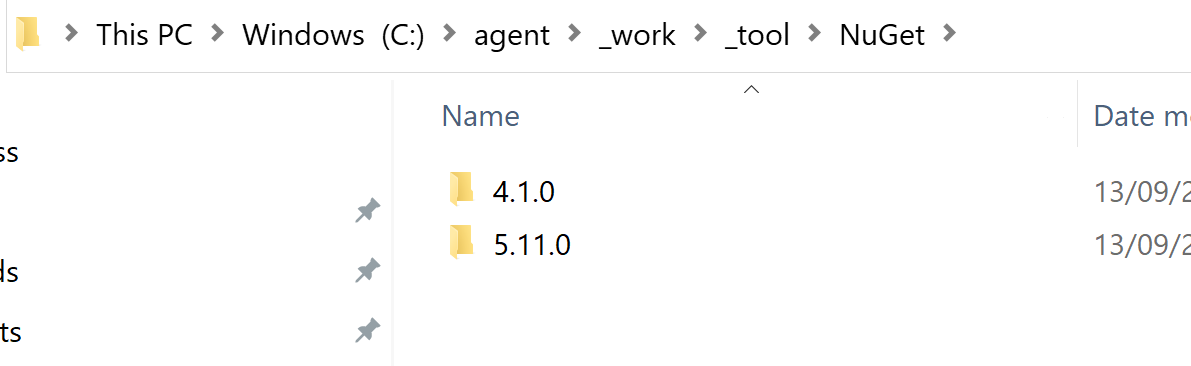


So over here, we're telling, built pipelines that once the build is complete, I want to go ahead and take that build whatever has been built the zip file and place it on to a location. So then our release pipeline can go ahead and take our application, take that zip file , take that build and deploy it onto a target environment.

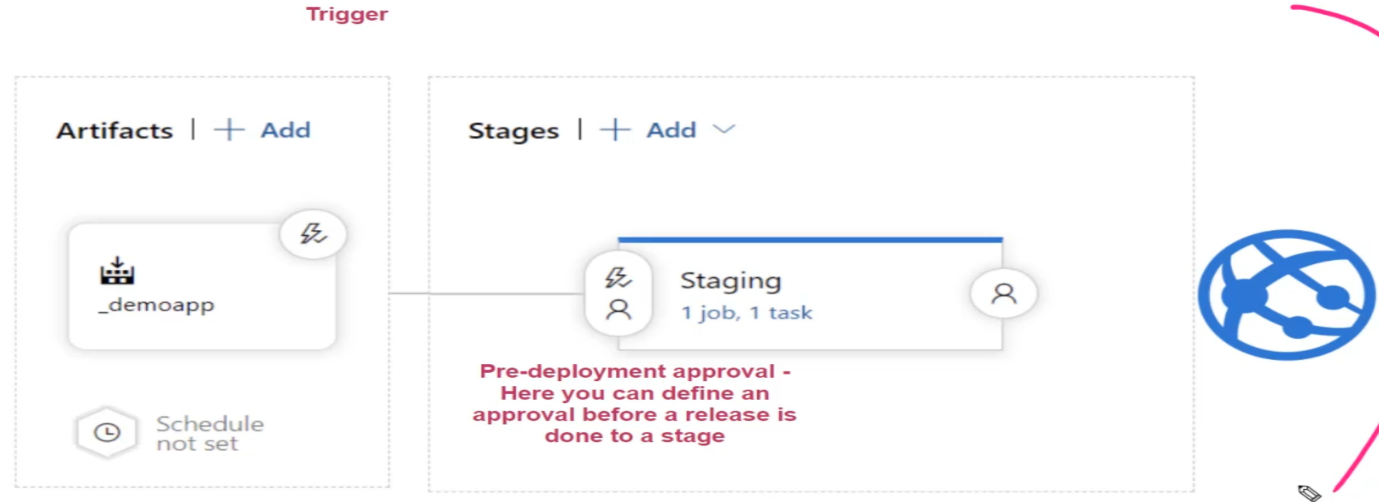
**Error Appear:**



Then go to -> C:\agent\\_work\\_tool and copy the NuGet Files manually -> Re-run the build pipeline



1. **Azure Release Pipeline:**

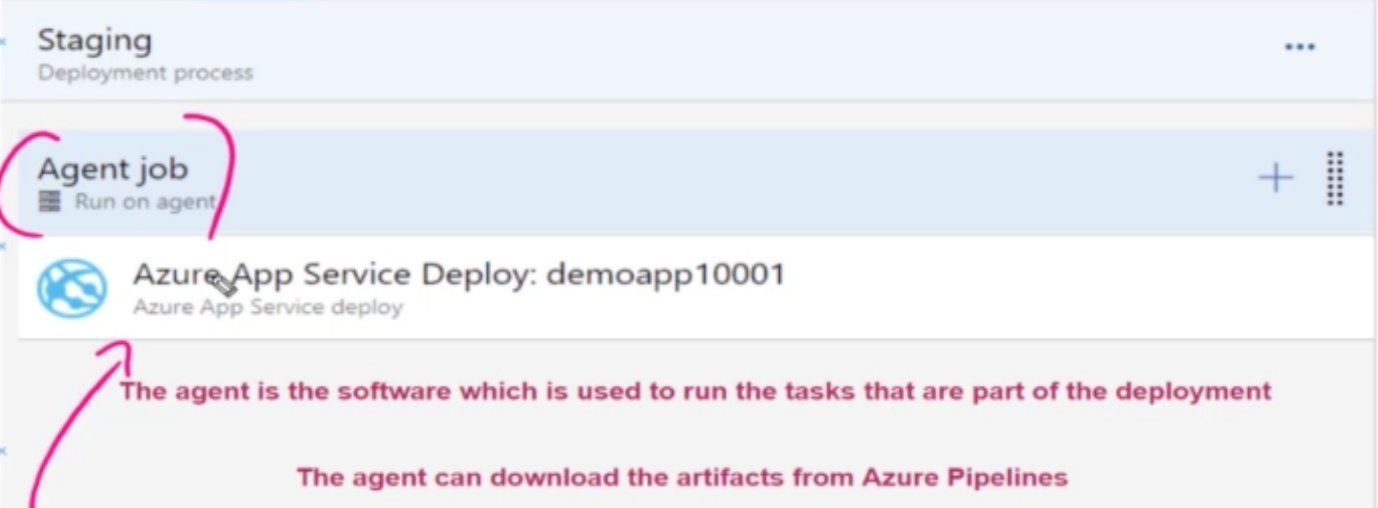
****

When you build your release pipeline –

**Stage 1:** Your first stage is basically your artifacts stage. So over here you can go ahead and pick up your artifacts. This is basically from the built pipeline.

Build pipeline -> build your application -> publish the built artifacts on to the release pipeline over here. -> The artifacts stage go ahead and pick up those artifacts. (This is a deployment file which is deployed on to a virtual machine.)

**Stage 2:** In release pipeline, we also have task under the Agent Jobs. Like Build pipeline, we also have Agents over here

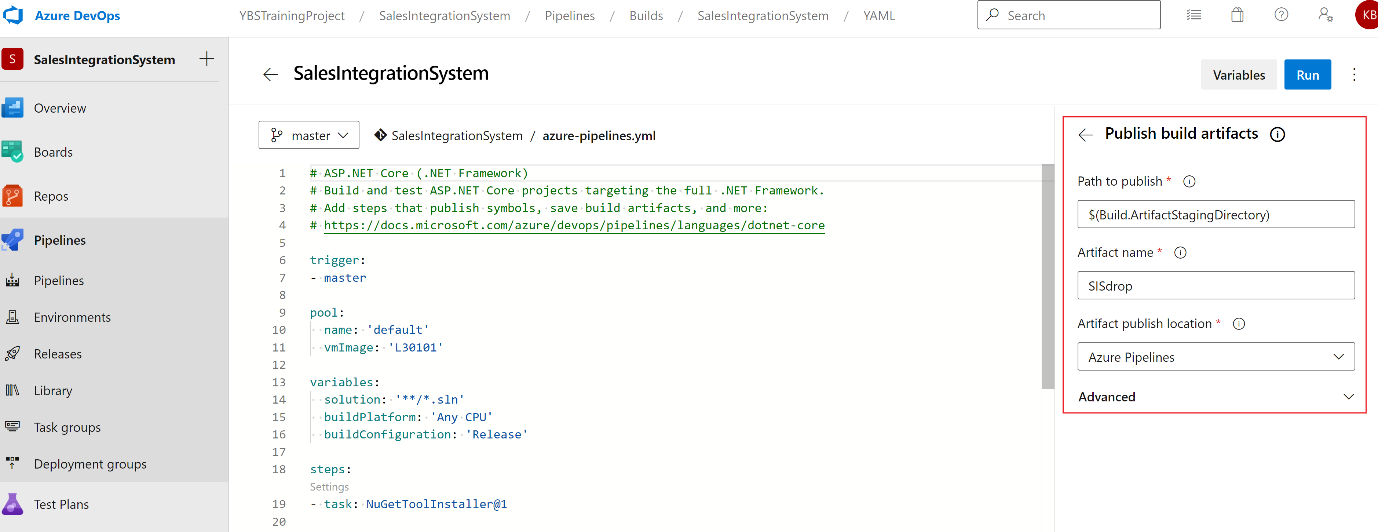


In Agent job -> use the plus symbol to create multiple tasks and these tasks will all be executed on this agent machine. So the agent is now the software, which is used to run the tasks that are part of the deployment. This agent will actually go out and download the artifacts from your Azure built pipelines and then run all of the tasks that are part of the release pipelines.

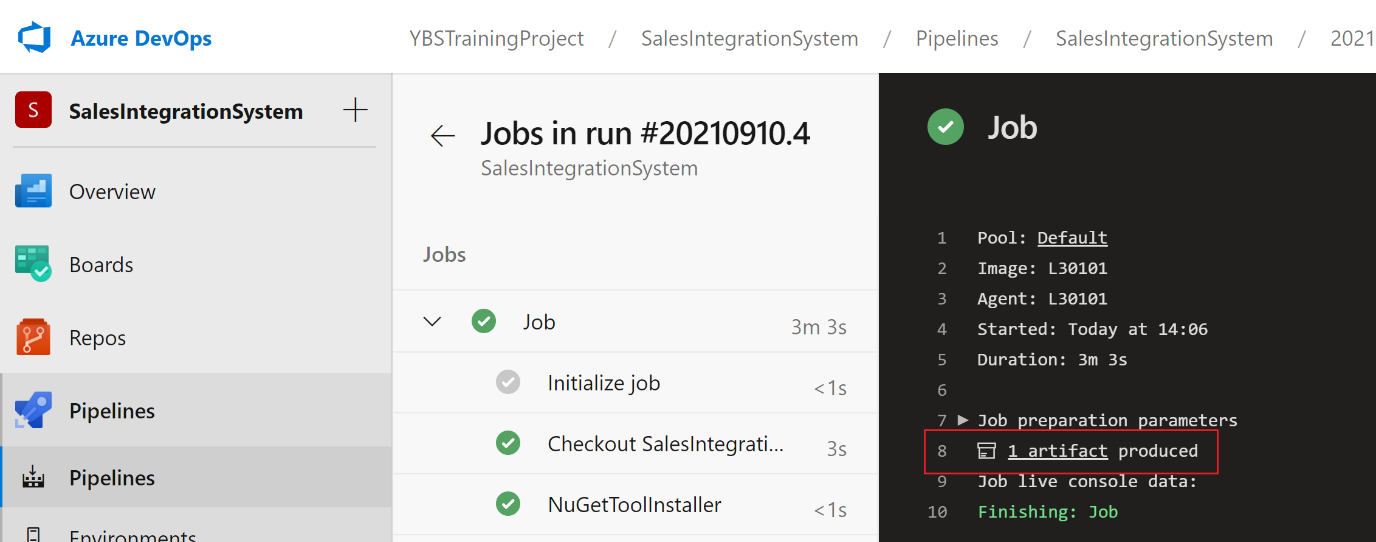
1. **DevOps Build Pipelines:**

Go to your Build Pipeline -> Add Publish Task

So over here, we're telling, built pipelines that once the build is complete, I want to go ahead and take that build whatever has been built the zip file and place it on to a location. So then our release pipeline can go ahead and take our application, take that zip file, take that build and deploy it onto a target environment.

****

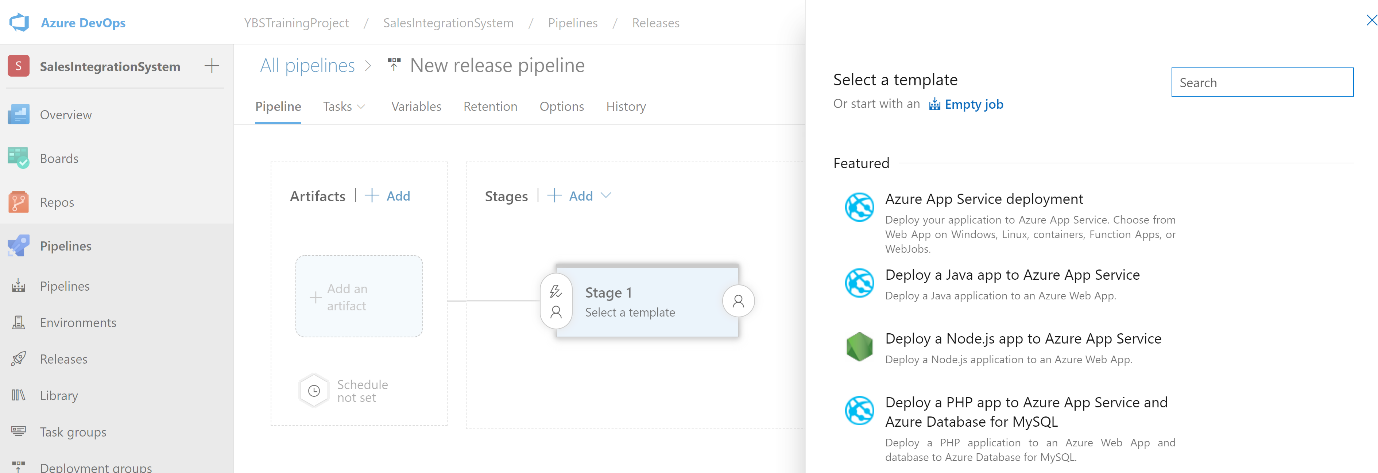
Save -> Run -> it will create a build Artifacts



1. **Create Release Pipeline:**

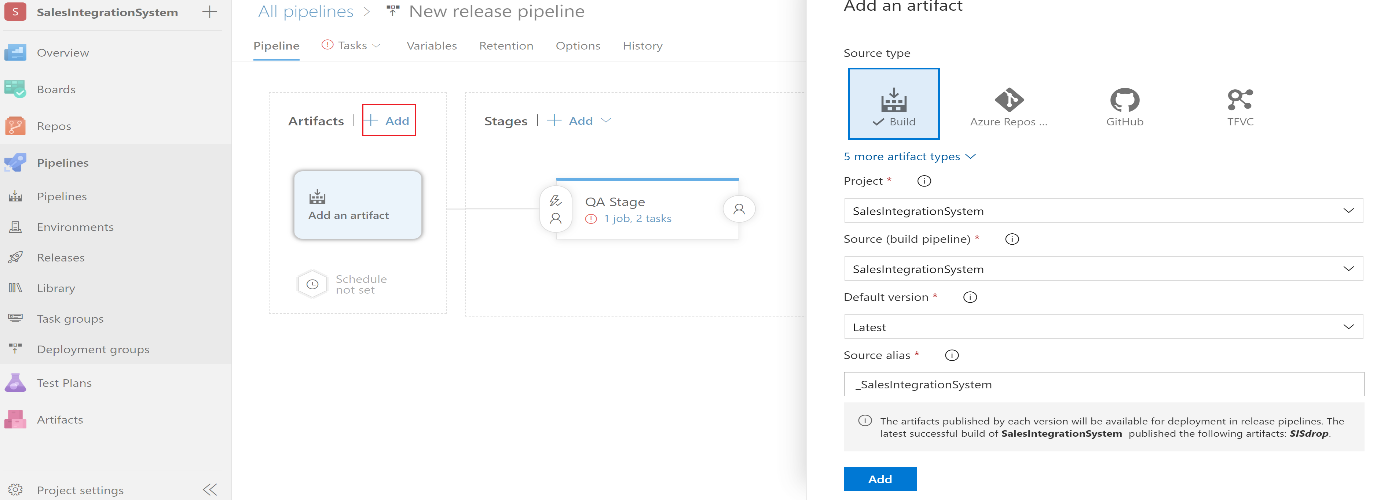
**Step 1:** Create a new Release Pipeline

So over here, you have different templates in place. So these templates will actually contain the task to basically perform a release over here.

****

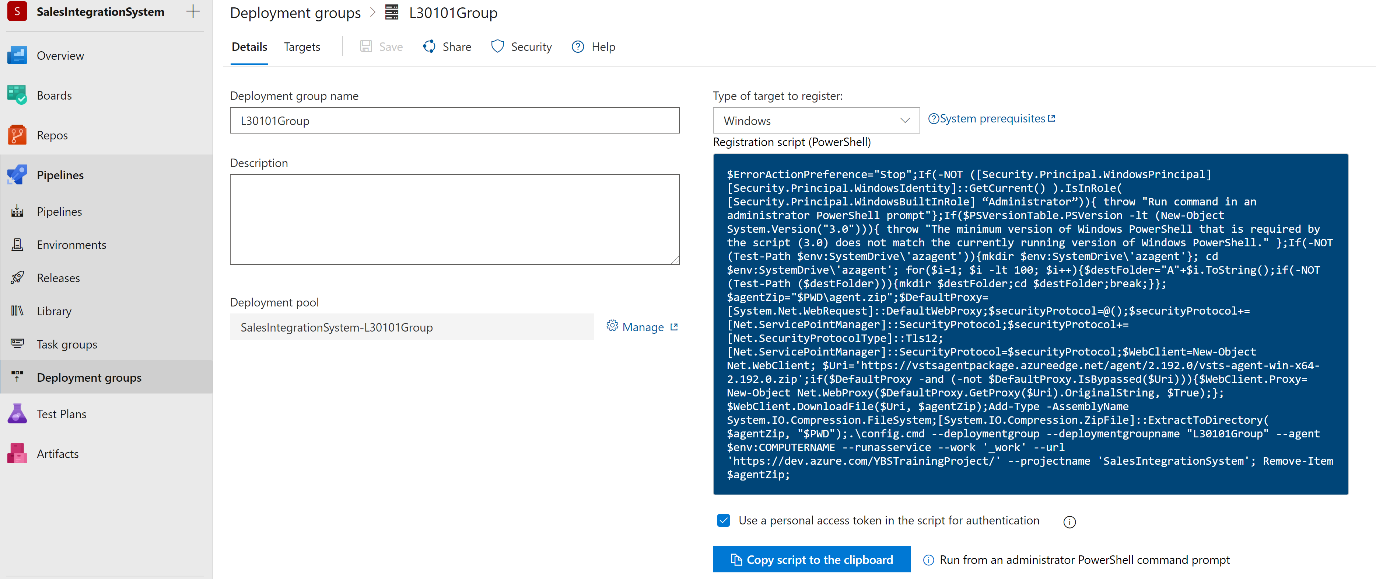
**Step 2:** Since you are going to deploy into ISIS -> so search IIS -> Add IIS Web Service

**Step 3:** Now Add Artifacts

****

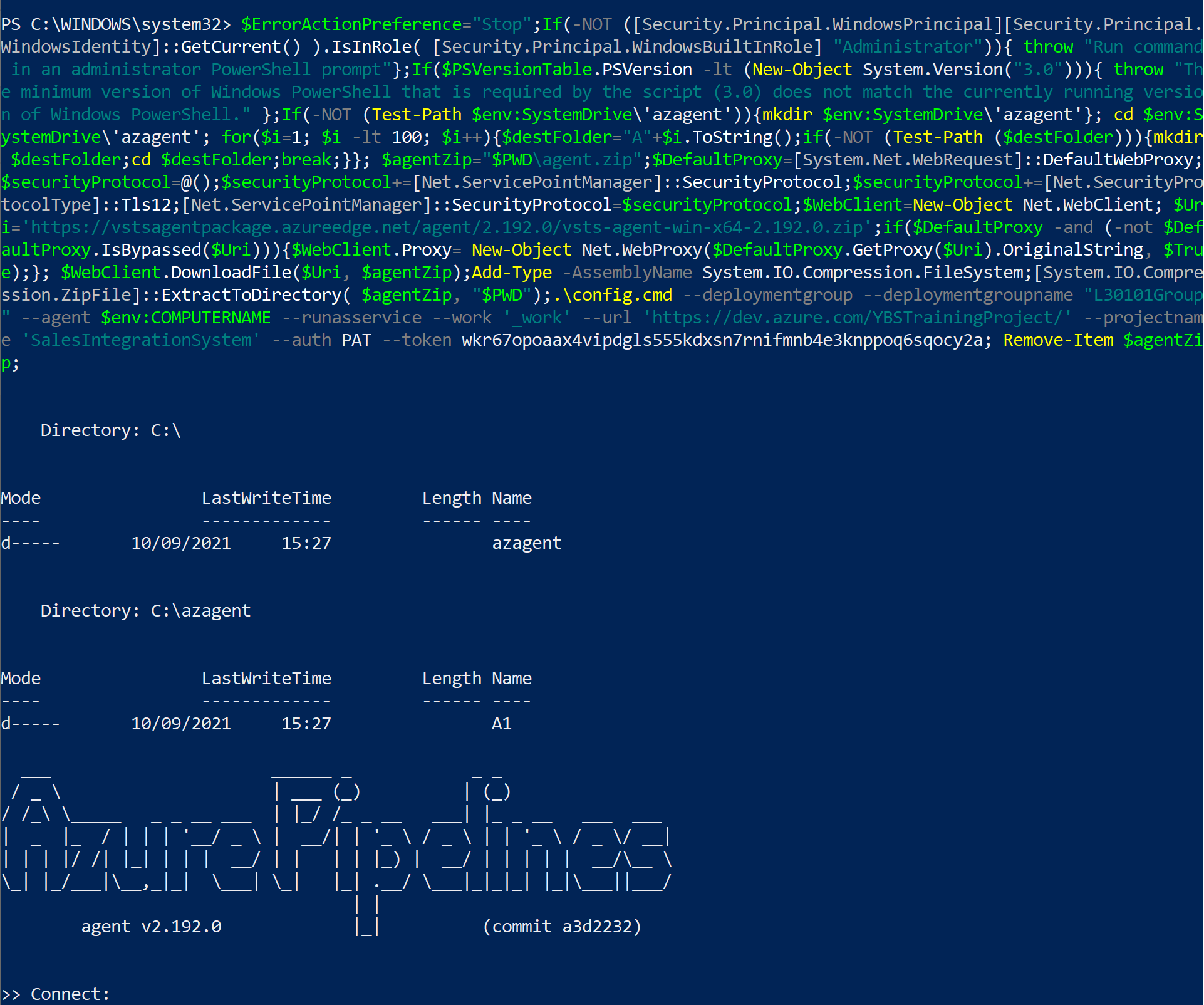
**Step 4: Add a Deployment Group**

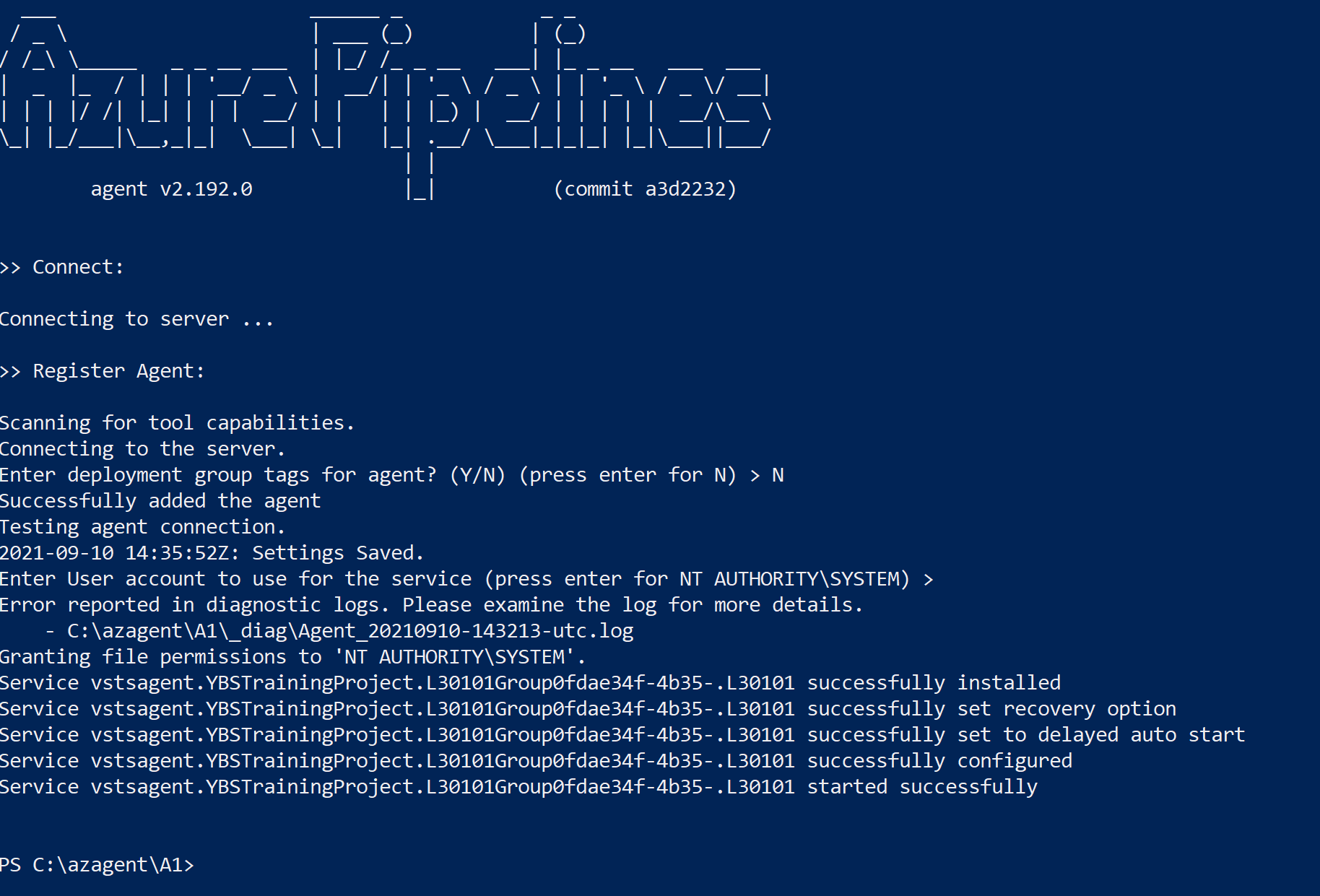
Create a new Deployment group -> You need to run this PowerShell script into your machine -> Then your machine will register on this Deployment Pipeline for particular target.

****

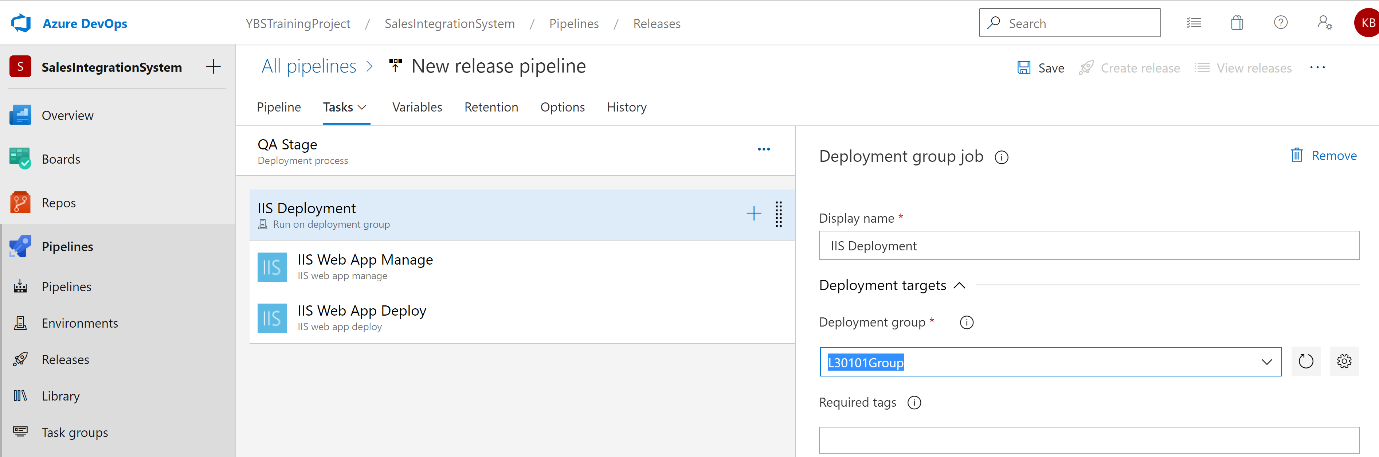
Deployment groups make it easy to define groups of target servers for deployment. A deployment group is a logical set of deployment targets that each have an agent installed.

Select the Check Box (Use PAT) -> Copy the script -> Open PowerShell ad Administrator -> Run the script -> It takes time





Now refresh Deployment Group circle and add Deployment Group

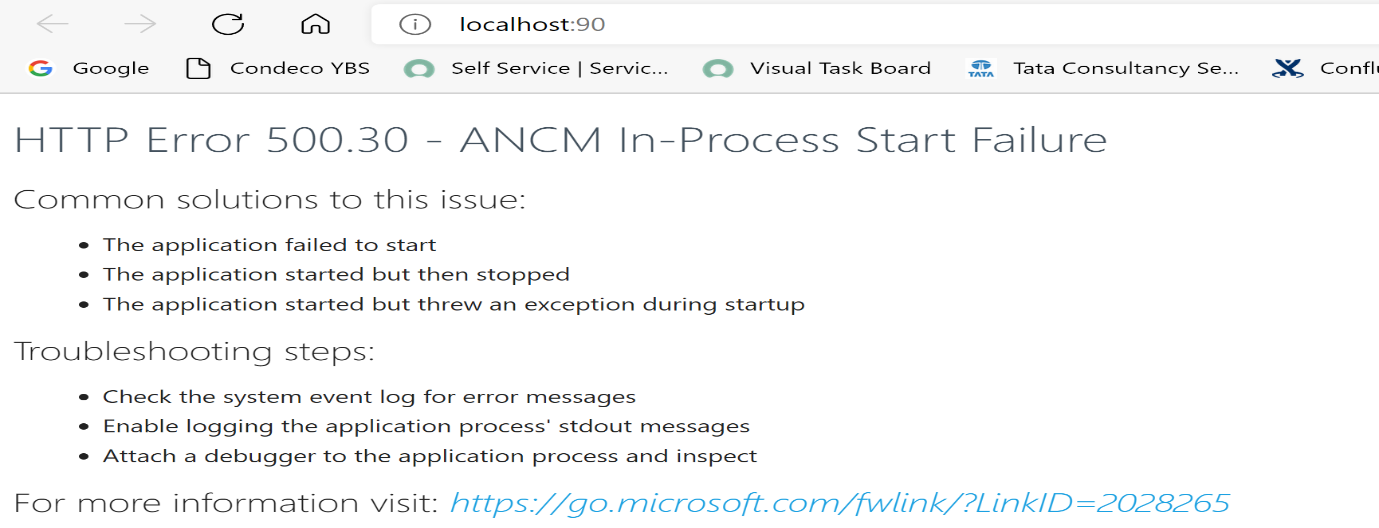


**Step 5:** In order to run your web application, you need to install - Install the .NET Core Hosting Bundle on hosting machine.

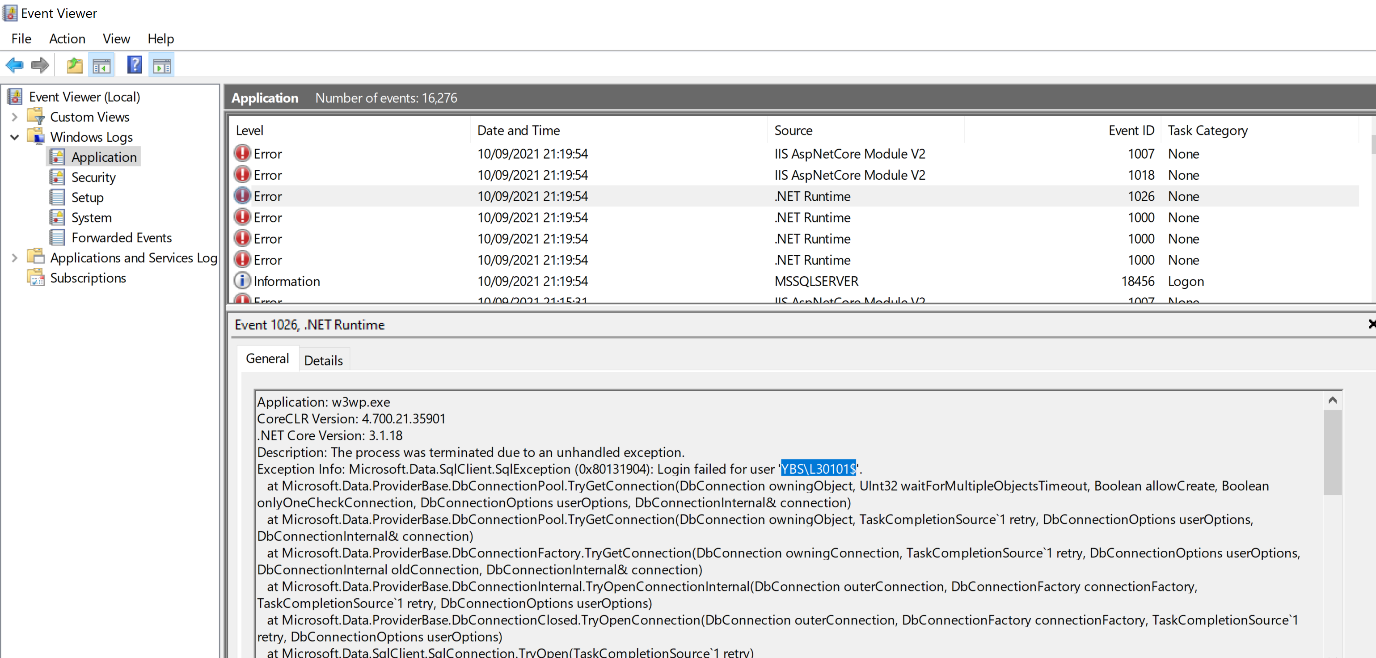
[Hosting Bundle | Microsoft Docs](https://docs.microsoft.com/en-us/aspnet/core/host-and-deploy/iis/hosting-bundle?view=aspnetcore-5.0)

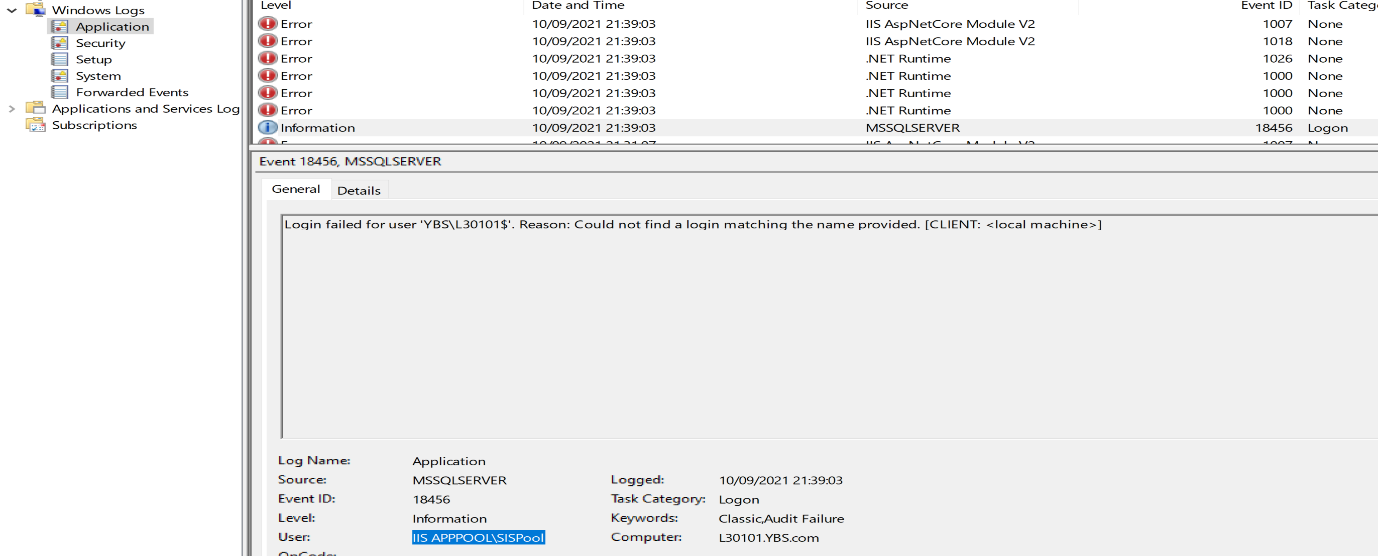
**Step 6:** Restart IIS and Run the application again -> If below error appears then open your Project in Visual Studio -> Remove Old version of Library ->

If still not resolve, then follow Step 7



**Step 7: Open Event Viewer -> See the .NET Runtime log**

****

****

**Open Database and create User**

1. **At Database Level**

**----------------------- Pool User -----In SalesIntegrationSystem ------------------------**

Use[SalesIntegrationSystem]

Go

CREATE USER [IIS APPPOOL\SISPool] for login [IIS APPPOOL\SISPool]

GO

EXECUTE sp\_addrolemember N'db\_owner', N'IIS APPPOOL\SISPool'

GO

**----------------------- Login User -----In SalesIntegrationSystem ------------------------**

Use[SalesIntegrationSystem]

Go

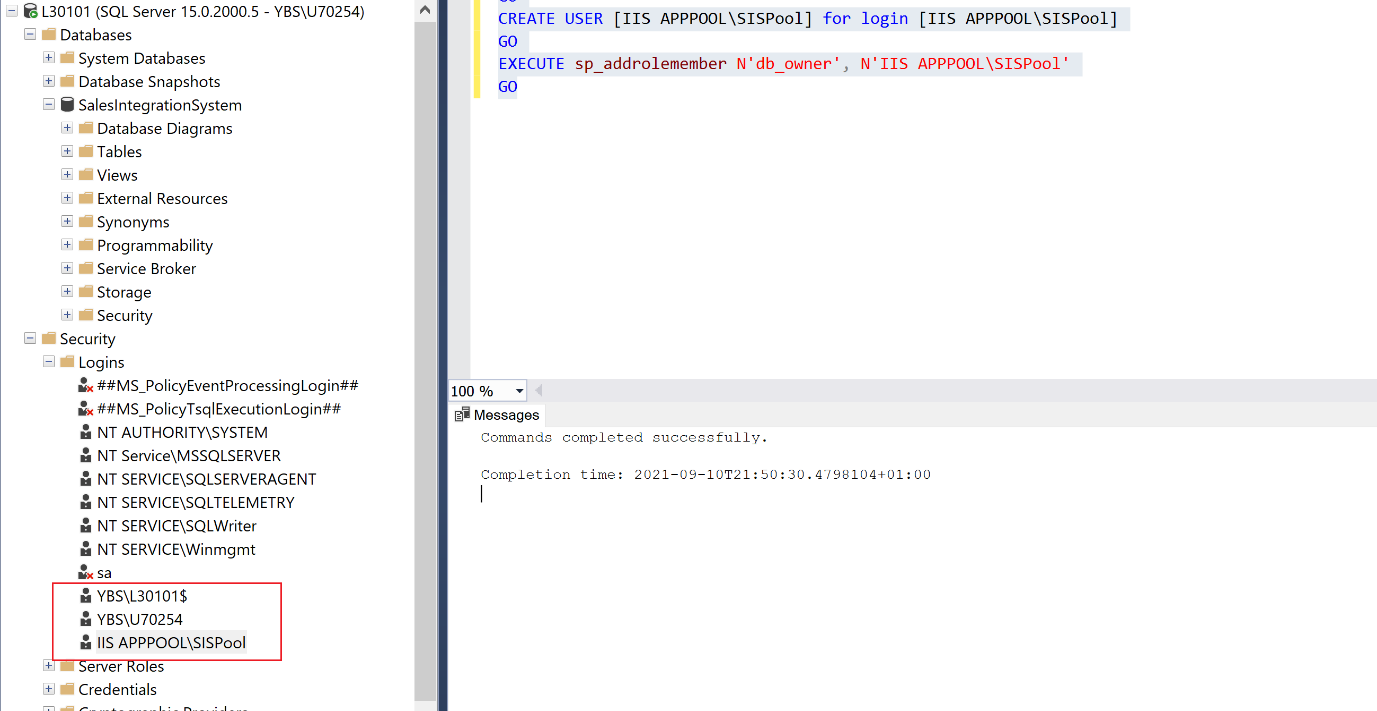
CREATE USER [YBS\L30101$] for login [YBS\L30101$]

GO

EXECUTE sp\_addrolemember N'db\_owner', N'YBS\L30101$'

GO

1. **At Root Label - Create Login User**

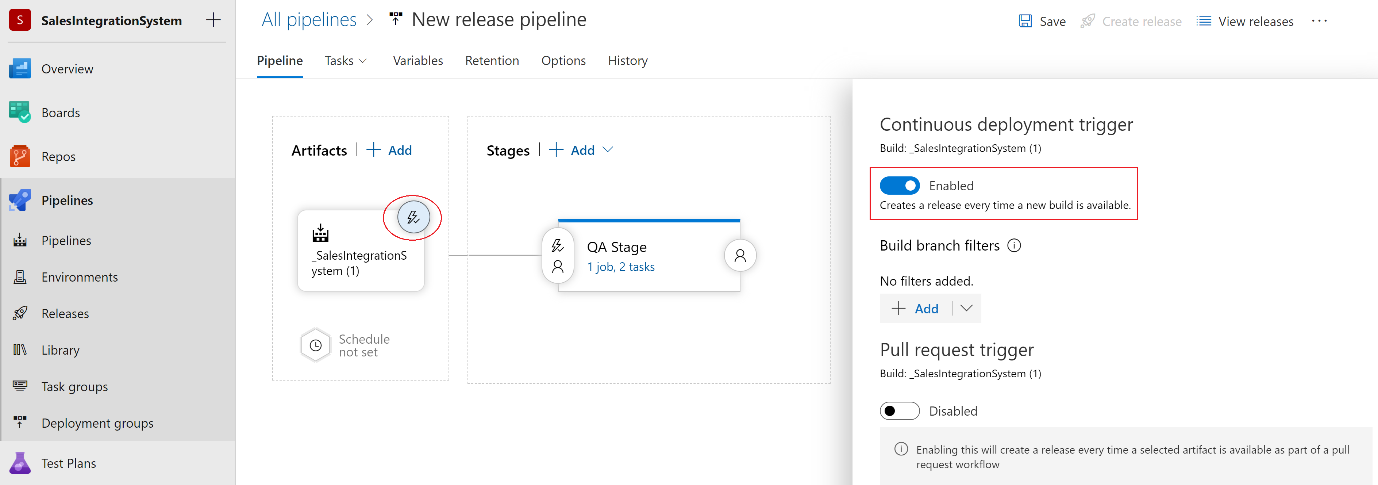
****

1. **Enable Remote Connection on SQL Server:**

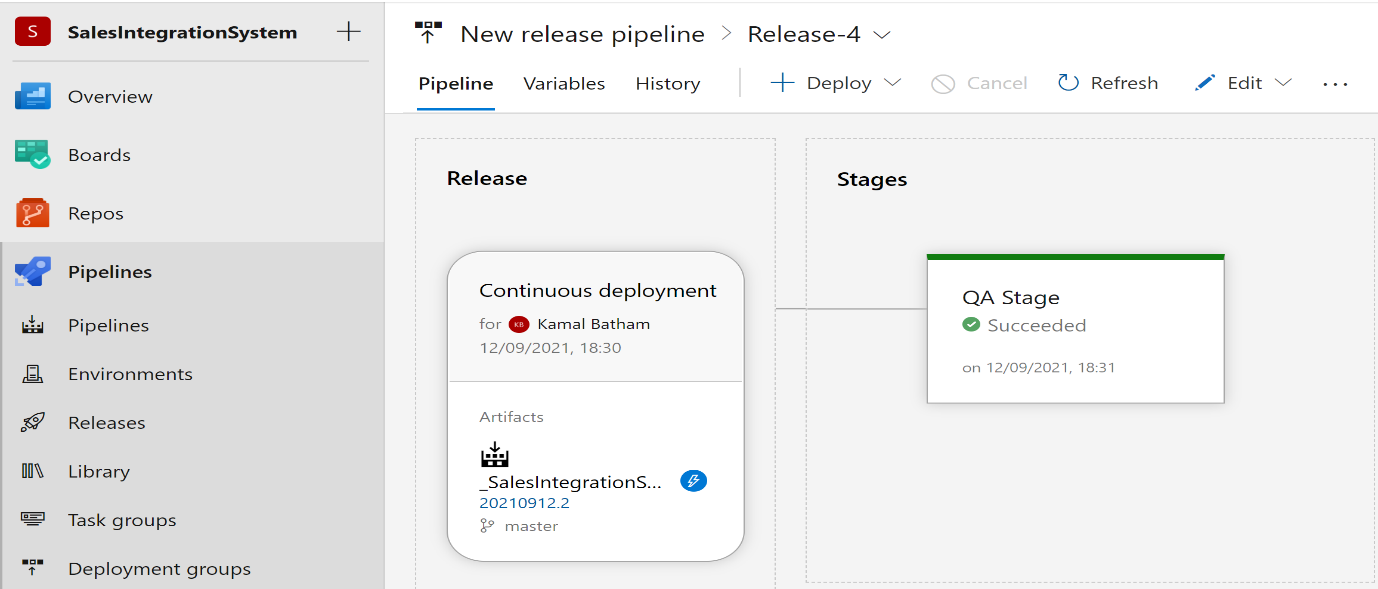
[How to Enable Remote Connections on SQL Server - TechNet Articles - United States (English) - TechNet Wiki (microsoft.com)](https://social.technet.microsoft.com/wiki/contents/articles/1533.how-to-enable-remote-connections-on-sql-server.aspx)

1. **Release Pipeline – Continuous Trigger:**

So this ensures that when the build is ready, a new build is ready from built pipelines, it will go ahead and trigger this release pipeline.



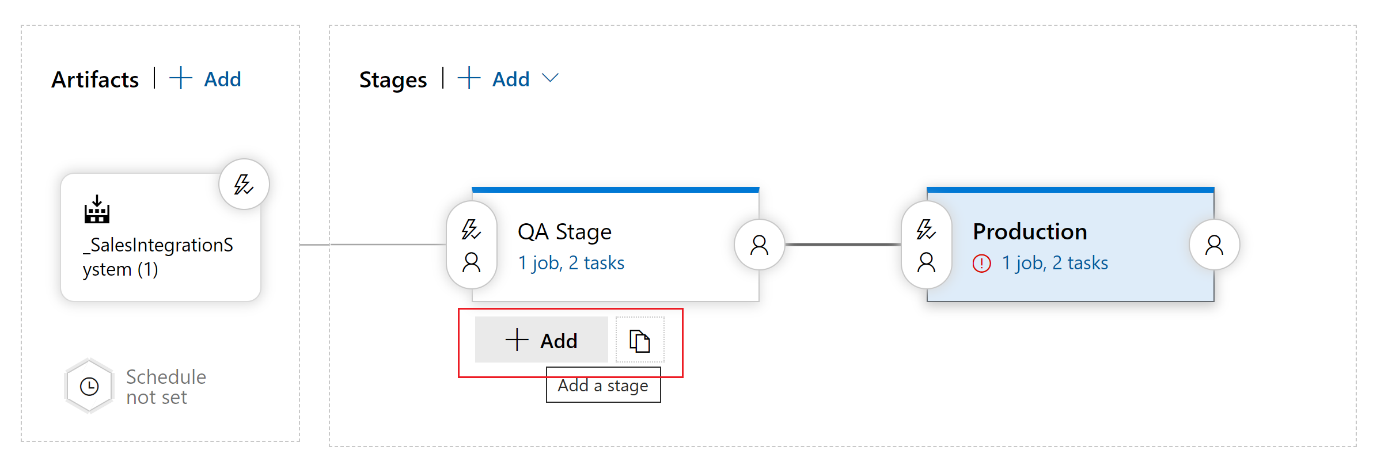
So over here, it has now become a continuous deployment, so this is not a manual trigger. So now it should have gone ahead and picked up our latest build and now it is going to go ahead and deploy it onto our QA stage and that is onto our L30101 (Windows 10) machine.



* 1. **Release Pipeline – Multiple Stages**

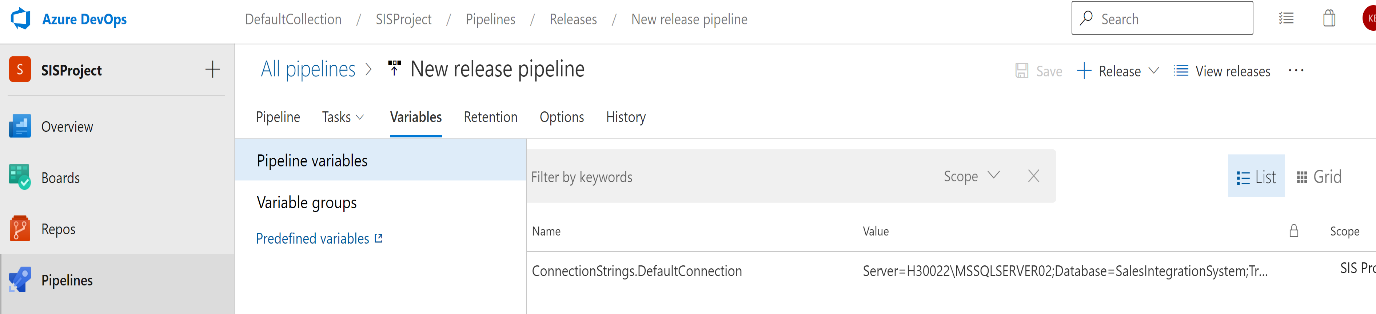
So currently we just have one stage in place that's the QA stage. now we can go ahead and add another stage as well.

Click on Add Button -> to add new stage



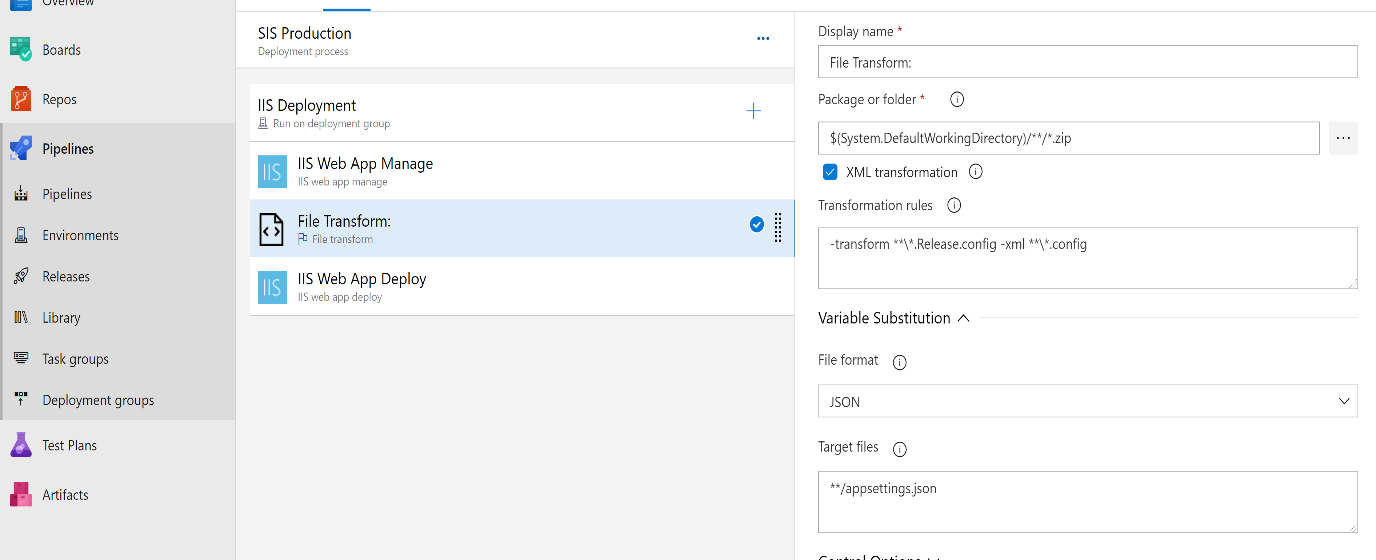
* 1. **Modify appsettings.config File for each stage**

**Step 1:** Go and create pipeline variable and define scope to apply it.

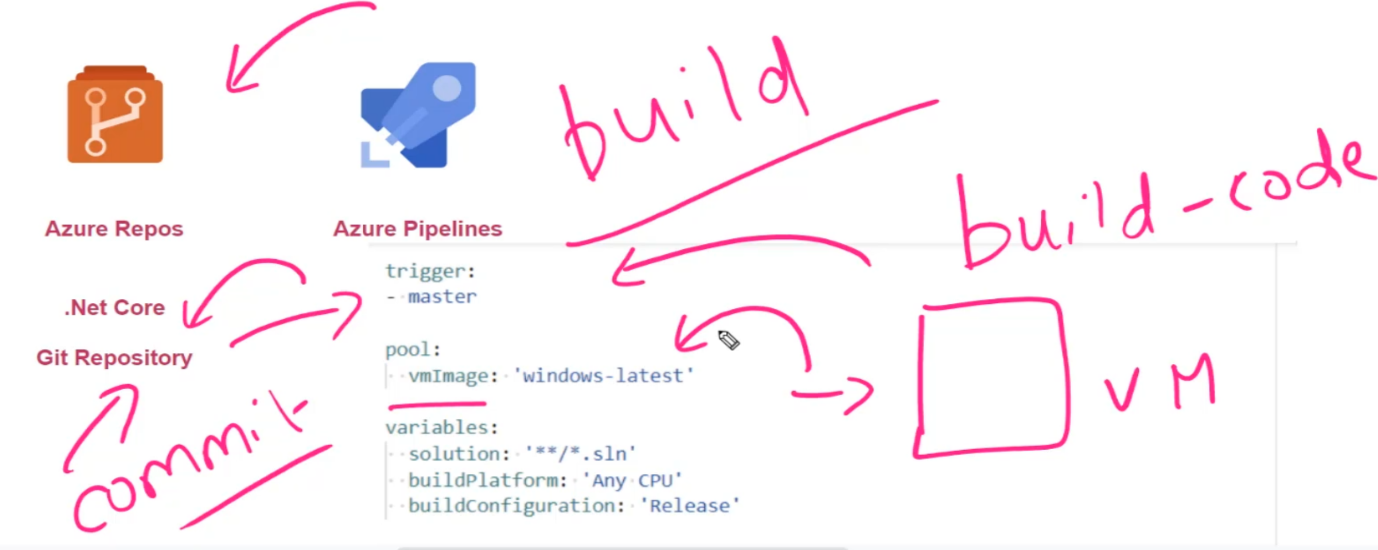
****

**Step 2:** You can create variable group if you want to use this variable for all release pipeline.

**Step 3:** Add File Transform task into your release pipeline, and configured it like below.



* 1. **Review of Release Pipeline**



**Step 1:** We have our code, which was hosted in our git repository in Azure repose **->** Then we created a built pipeline in Azure pipelines, this built pipeline would actually be triggered whenever there was a change made on to the master branch in our git repository. So whenever a commit was made on to the git repository, it would go ahead and trigger the build pipeline. (This build pipeline use self-hosted agent configured or Microsoft hosted agent in Azure to build your source code)

**Step 2: If Image is not self-Hosted -> it’s Azure VM image**

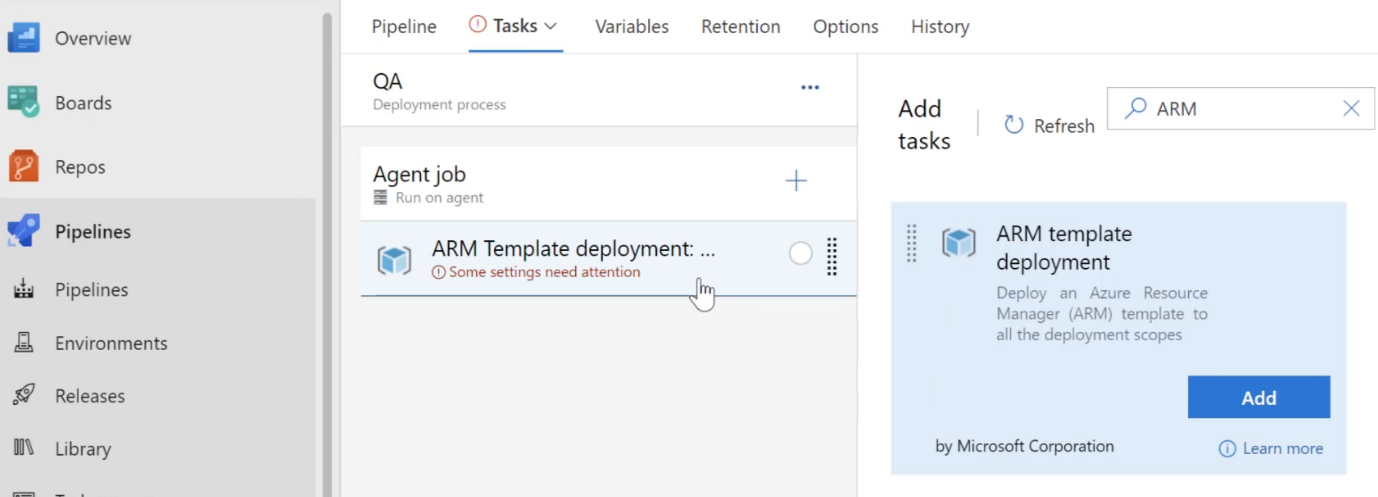
So it creates a virtual machine in the background based on the Windows latest image -> Remember, based on the image, there are certain tools that are already installed on the virtual machine, this is important because these tools will be used to carry out the build of your code. If these tools are not installed, then your build will fail. So ensure to choose your VM image accordingly.

Also note that once the build is complete, then this virtual machine will be disposed of, it will be deleted.

**Step 3:**

* 1. **ARM Template for release Pipeline**

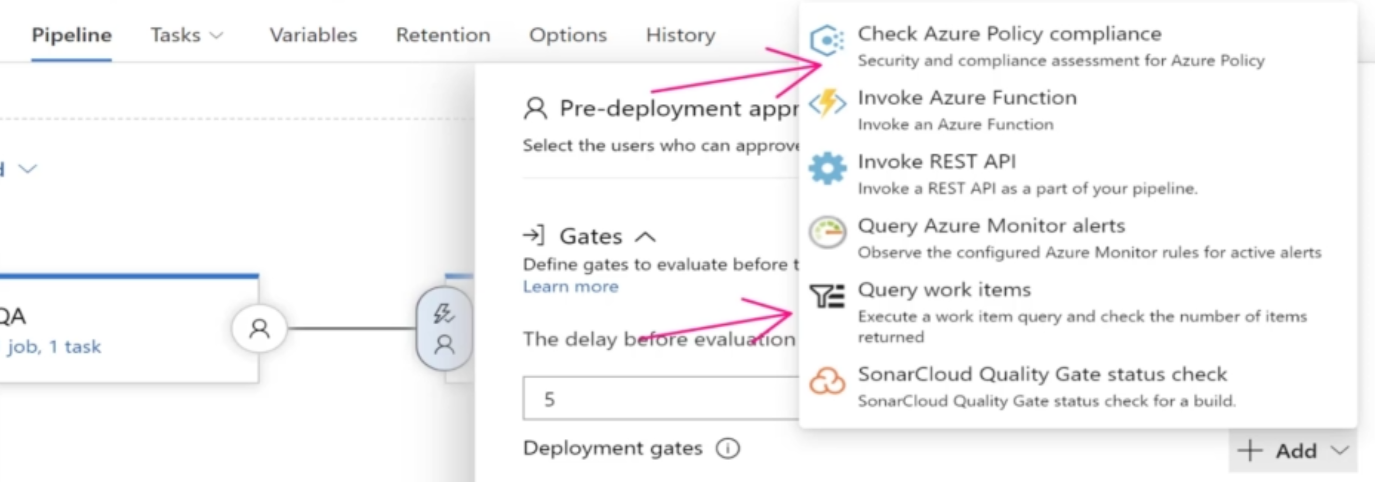
Your can also use ARM template for you deployment -> put the templated into storage account and provide the path here

****

1. **Approval & Gates**

The important concepts are

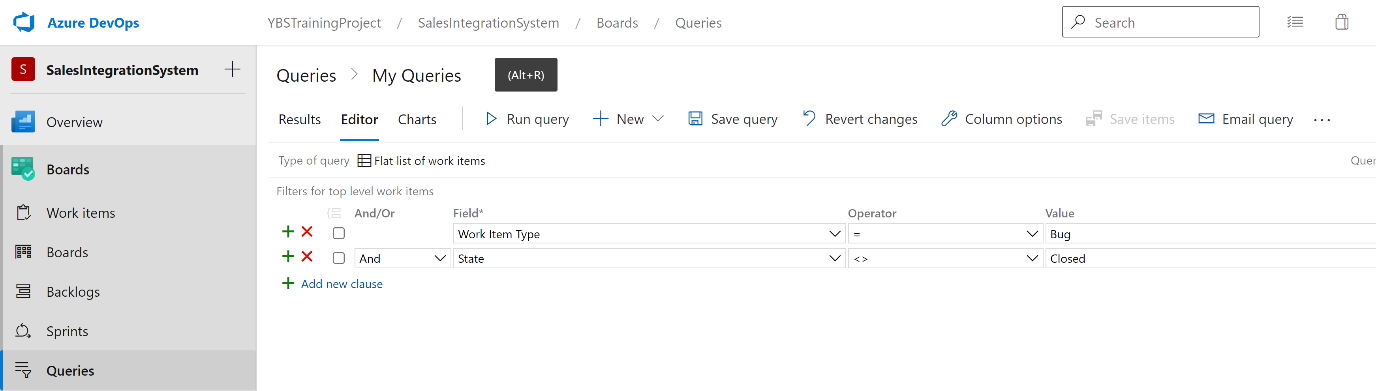
* Approval's and Gates can give you more control over the deployment pipeline.
* Here You can go ahead and define pre deployment and post deployment conditions in the pipeline.
* So, let's say that a user needs to first validate a change request before the deployment can be done onto a stage. Someone has to go out and validate whether the change request is appropriate or not. So we can go ahead and add a pre deployment approval.
* Another use case scenario let's say that the user needs to first validate the deployment, which has been done onto a stage before the deployment can be done onto another stage. So you can go ahead and at a post deployment approval.
* sometimes you might want to ensure that there are no active issues for the project before the deployment can be carried on to a stage, over here you can go ahead and use something known as a pre deployment gate.
* Another scenario, let's say an application has been deployed onto a staging environment and it needs to be ensured that there are no incidents recorded in the incident management system before the application is deployed onto a production based environment. Over here, you can go ahead and use a post deployment gate.
  1. **Gates:**



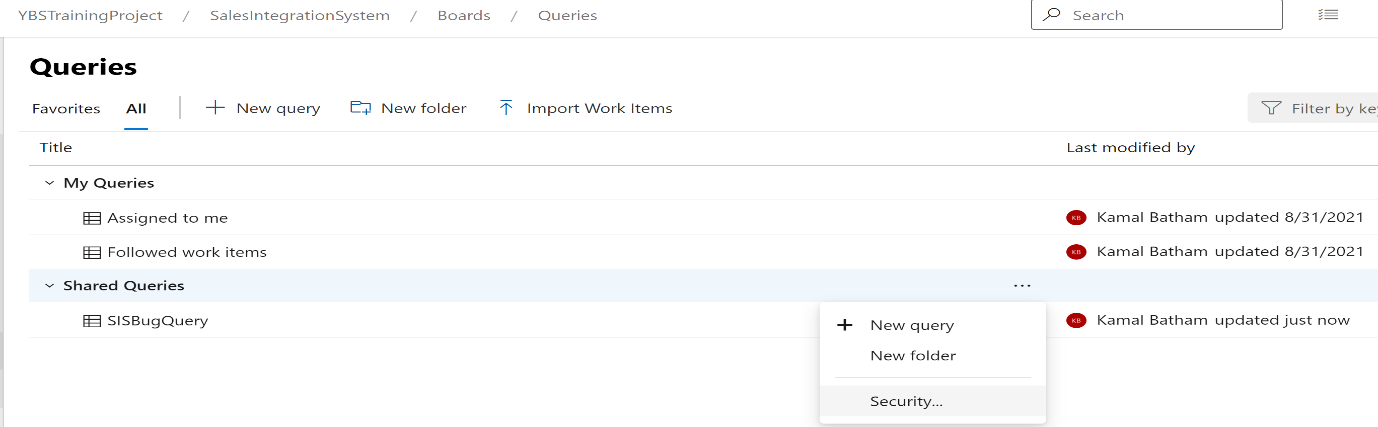
**Choose Check Azure Policy Compliance** - if your company has several Azure policies in place and this deployment must adhere to one of those policies. If it doesn't, then it should go ahead and not carry out this deployment.

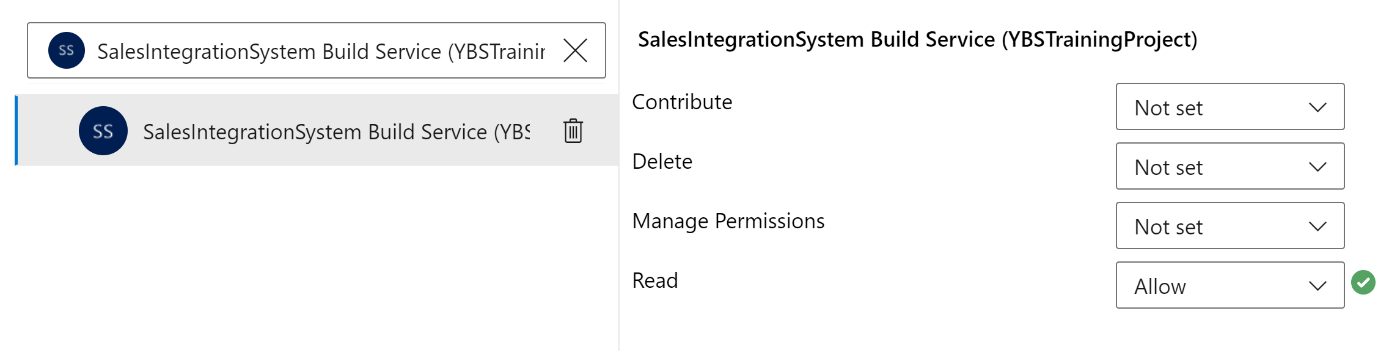
**Choose Query work Items** - you want to go ahead and ensure that there are no bugs when it comes to work items in the project, if there are no bugs, then only go ahead and do the deployment on to the production stage.

**Step 1:** if you select Query work Item -> First go to Queries in Azure Board -> Write condition to check list of Bugs available -> Save the query into Shared Query

****

**Step 2:** Now go to shared query -> Click on more action -> Click Security -> Add permission that will allow the pipeline to go ahead and access this Shared Query. -> Then Add Gates to evaluate this query to make decision for deployment.



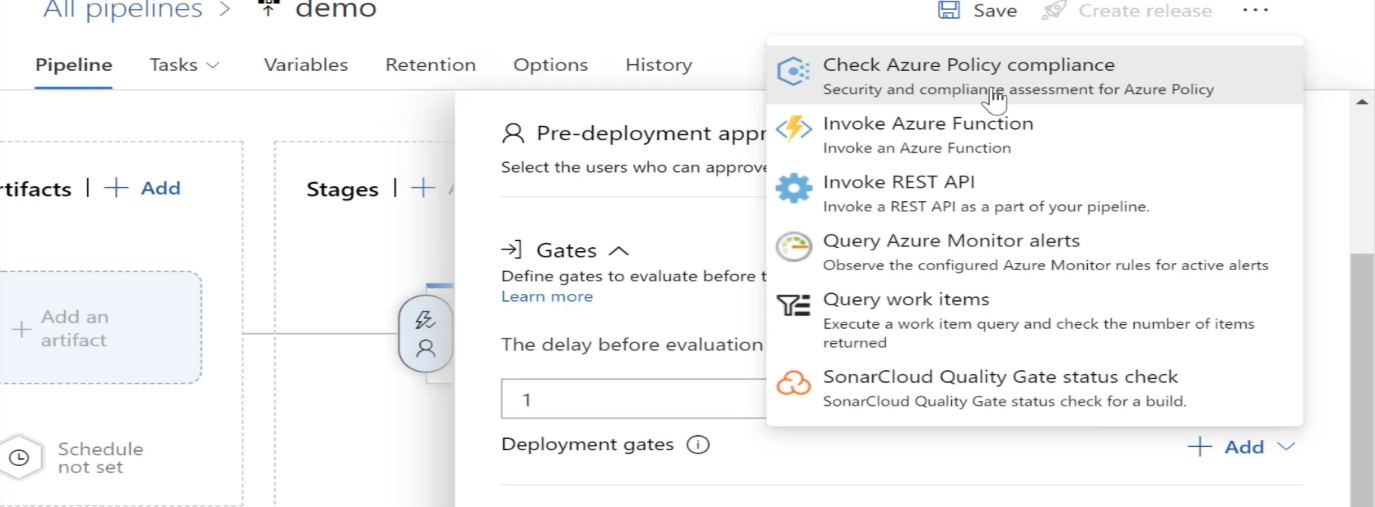


* 1. **Gates – Azure Policy:**

**Azure Policy -** Azure policy can be used to show the compliance of your existing resources, and when it comes to your new resources, you can go ahead and prevent the creation of new resources based on these policies.

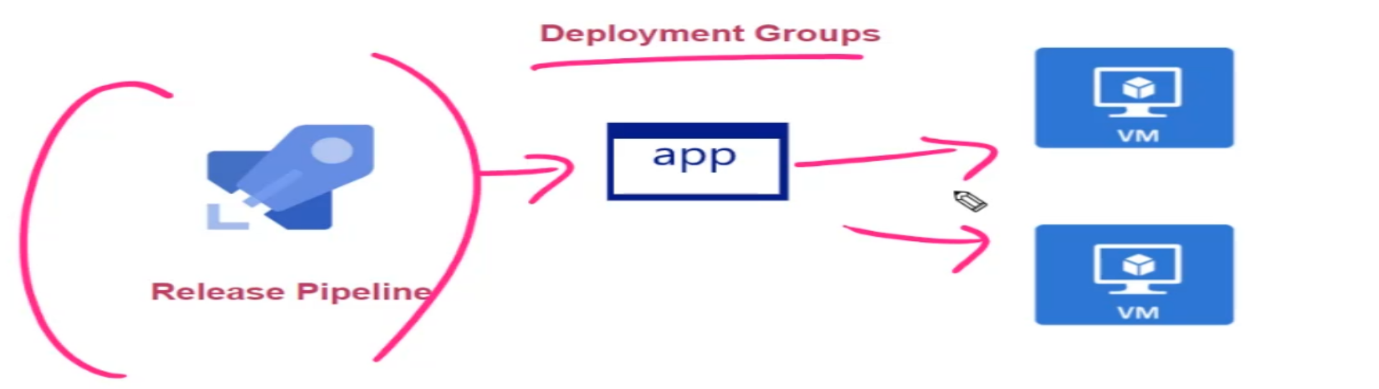
So, based on your organization – you can define azure policy for all new resource that suppose to be create – that resource need to follow the policy **For example:** Defined policy that Tag need to set when create new resource. Or define policy for SKU size of VM – so that when new VM created, it needs to follow that policy.

**Now Implement Gates with Azure Policy Compliance:** Select Azure Policy Compliance and apply the policy that you have created in Azure.

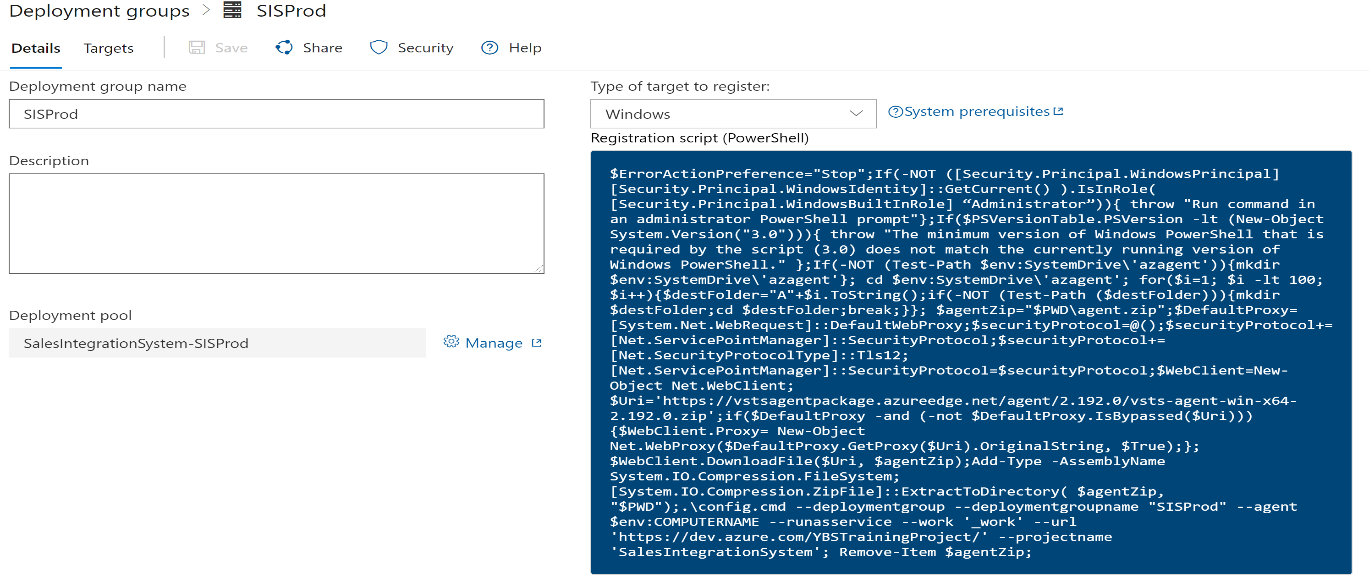


1. **Release Pipeline - Deployment Group:**

So, let's say that via your release pipeline, you want to go ahead and deploy an application on one virtual machine or onto a set of virtual machines, so these are an existing set of virtual machines.



So what you can do - install an agent on each of these virtual machines and then you could go ahead and make those virtual machines as part of the deployment group.



1. **Implementing OWASP- Primer:**

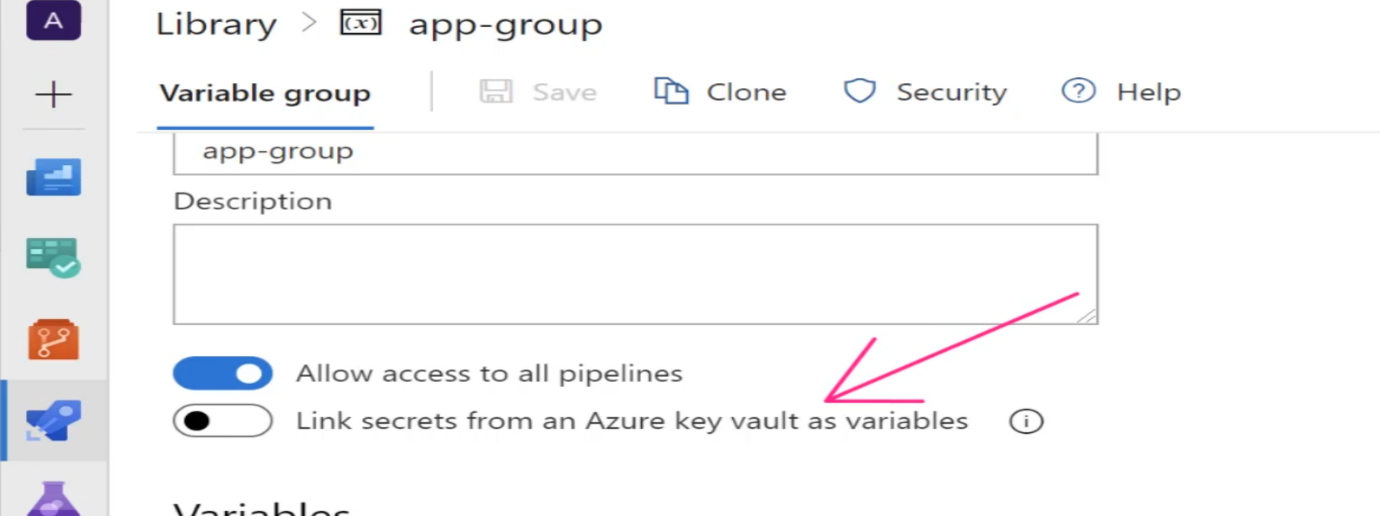


So let's say you have a built pipeline in place. The code is stored in Azure repose and let's say it goes out and deploys it via Azure release pipelines on to on-premise IIS.

Then after the QA stage, you can actually have another stage that maybe a security stage that would go ahead and actually implement this OWASP tool, that OWASP tool will actually go ahead on to the URL of your Web application. It will look for any sort of security vulnerabilities and then give you a report.

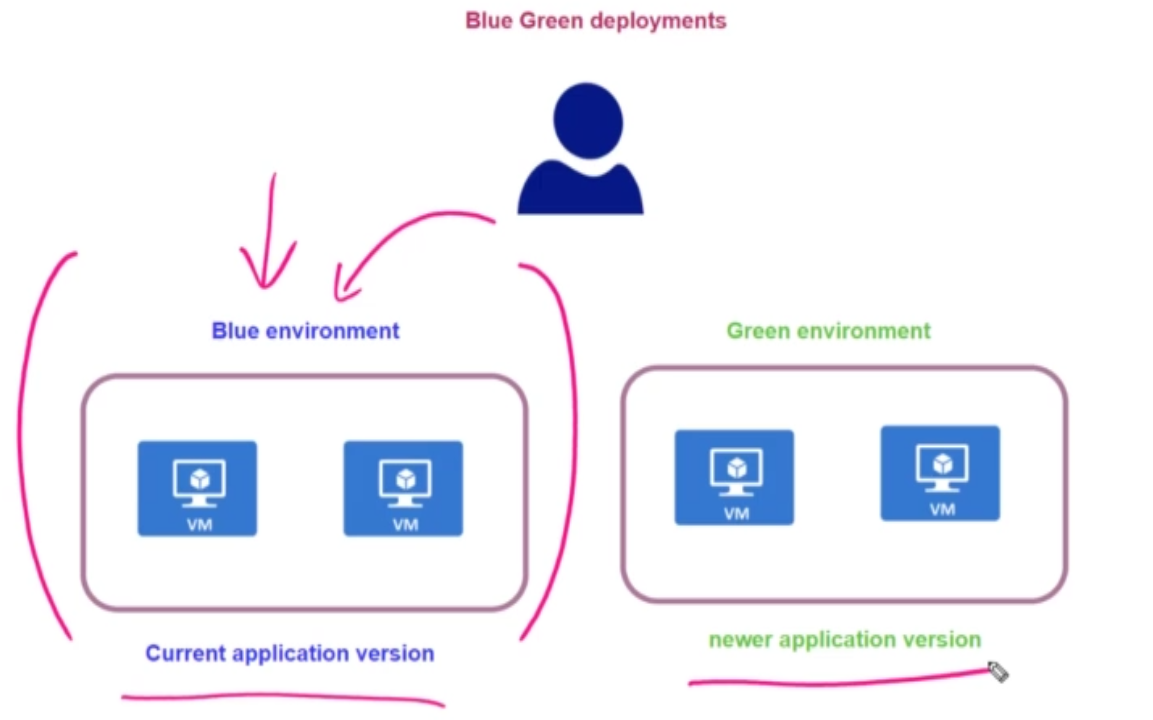
1. **Variable Groups- Azure Key Vault:**

**Step 1:** Create a new Azure Key Vault -> Defined new secret -> Go to your Azure DevOps Variable Group -> Here you have the ability to link secrets from as your keyboard has variables.



1. **Blue Green Deployment**

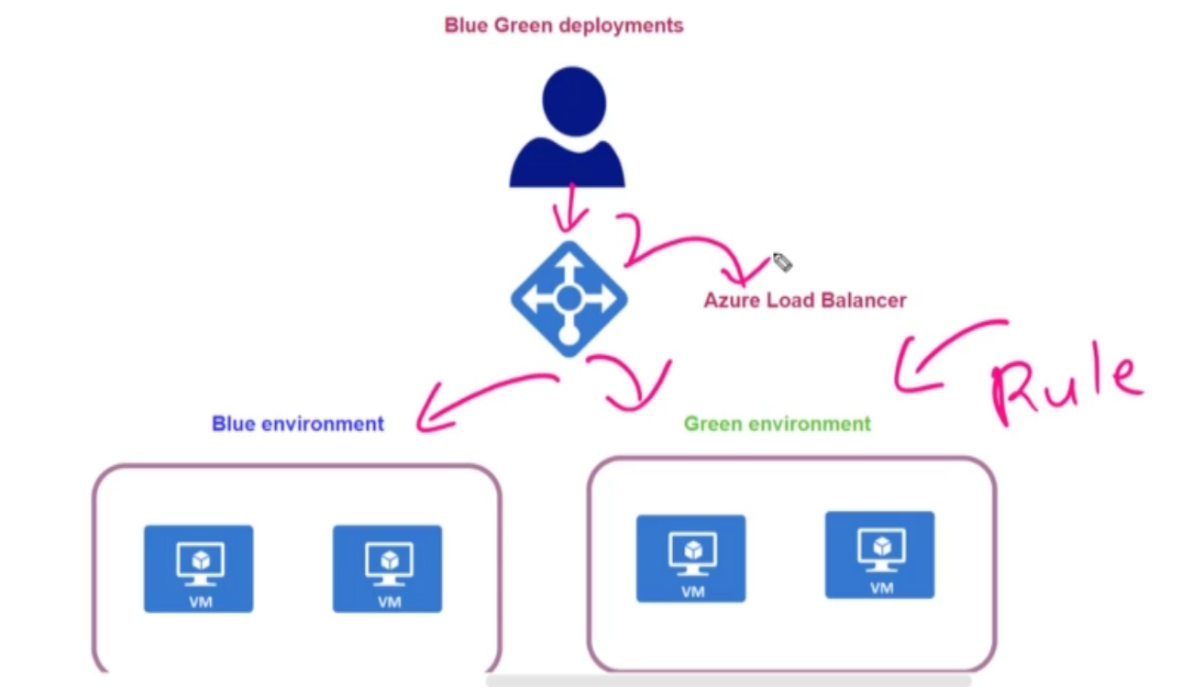
So this strategy is basically having two environments in place. So over here you could have one environment in place that's known has your blue environment. So this could contain your current application version. So overcome all of your users are currently going ahead. And let's see, accessing the application which is hosted on this environment.



Now, when it comes to go ahead and deploy a newer application version, one of the strategy is to go and spin up a duplicate environment and go ahead and deploy the new version of the application onto this environment itself. And this can be termed as green environment.

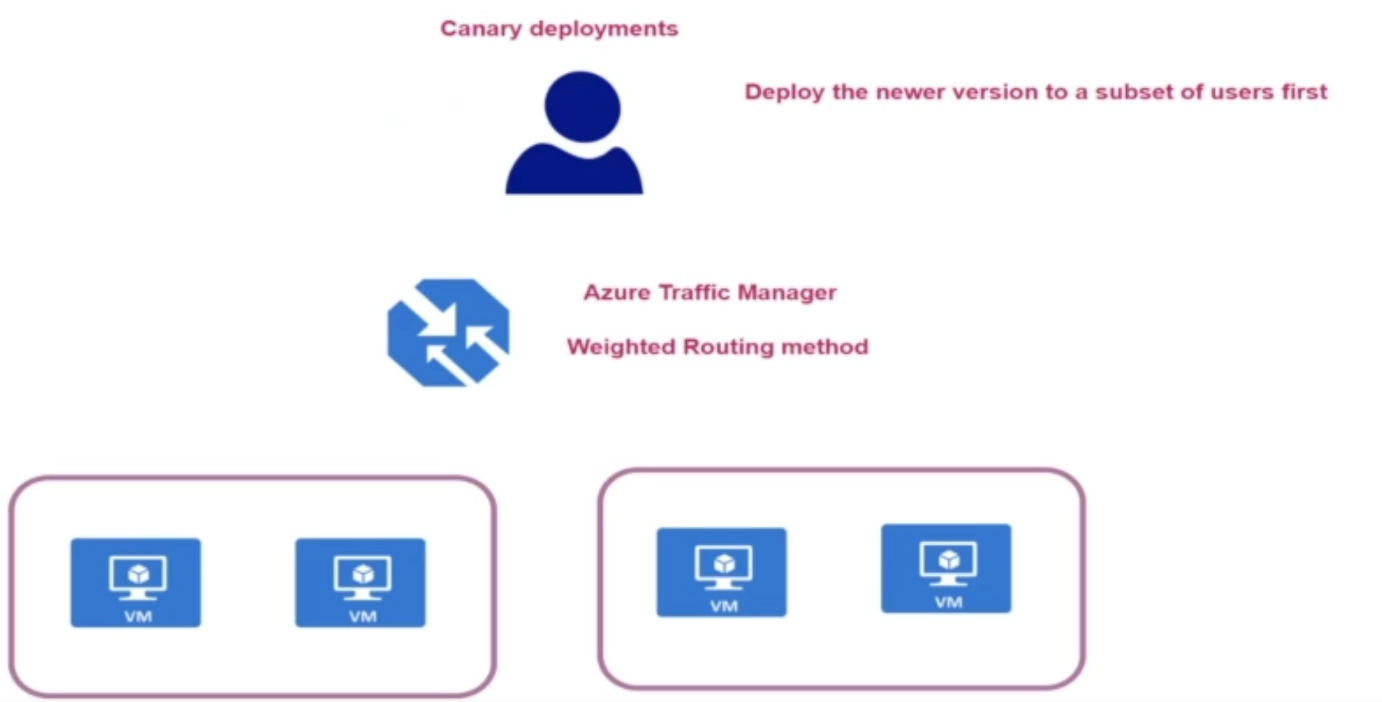
* Well, the first benefit is that it allows you to quickly switch over onto the newer environment. You also have time to test the application on the new environment before it goes live, and at any point in time you can actually go ahead and switch back onto the old environment.
* the disadvantage is that obviously the cost is double because you have to horse to one off your environments

You can manage the blue Green deployment using Load Balancer



1. **Canary Deployment**

You deploy the new version of your application on to a subset of users first. Once those users have actually gone ahead and tested the newer version of the application, then gradually can go ahead and ensure that all the users get the newer version of your application.



You can implement this using this service copy as your traffic manager.

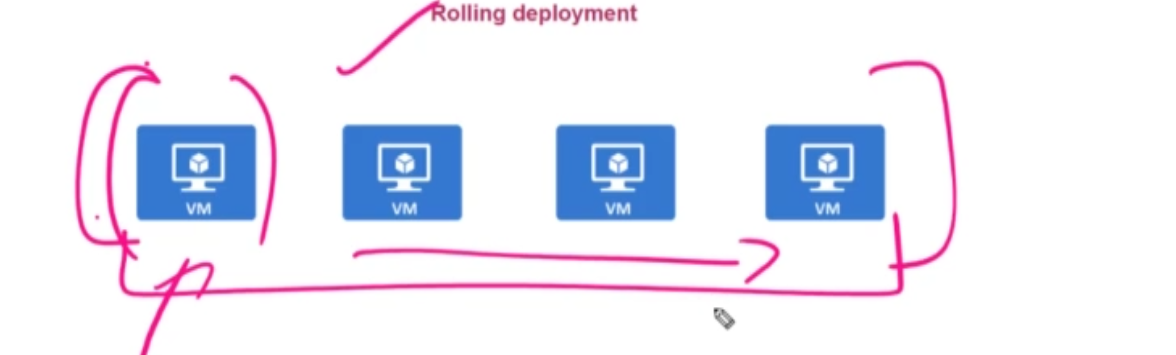
So over here, you might have, again, two environments in place, one that holds the current application version and the other one hosting the new application version. Now, there is a routing method that is available known as the weighted routing method.

you can go out and tell your traffic manager, if you're going out and perform any canary type of deployment, please go ahead and direct 80% of the traffic on to the current application portion and then direct around 20% of the traffic onto the new application version so that a subset of users will actually go ahead and get the newer application version.

In the end, when you want to go out and perform a complete switchover, you can actually go ahead and disable the environment altogether.

1. **Rolling Deployment**

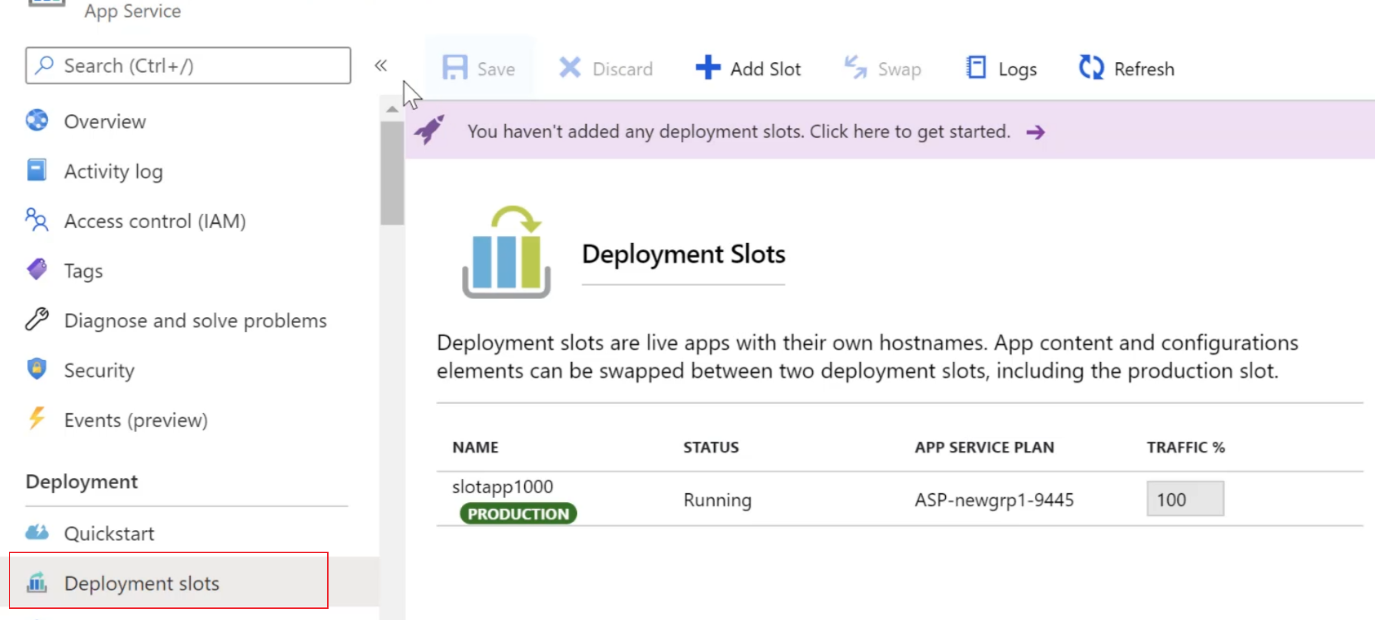
So over here, when you have a set of virtual machines that can be part of a virtual machine skill set.



So when you want to deploy a new application version on these virtual machines in a rolling deployment, you actually update the application version on each virtual machine in batches. So instead of actually deploying the new application version on all virtual machines at one time itself, your first update, one of the virtual machines with a newer application version.And slowly you start updating all of the voting machines.

1. **Deployment Slots:**

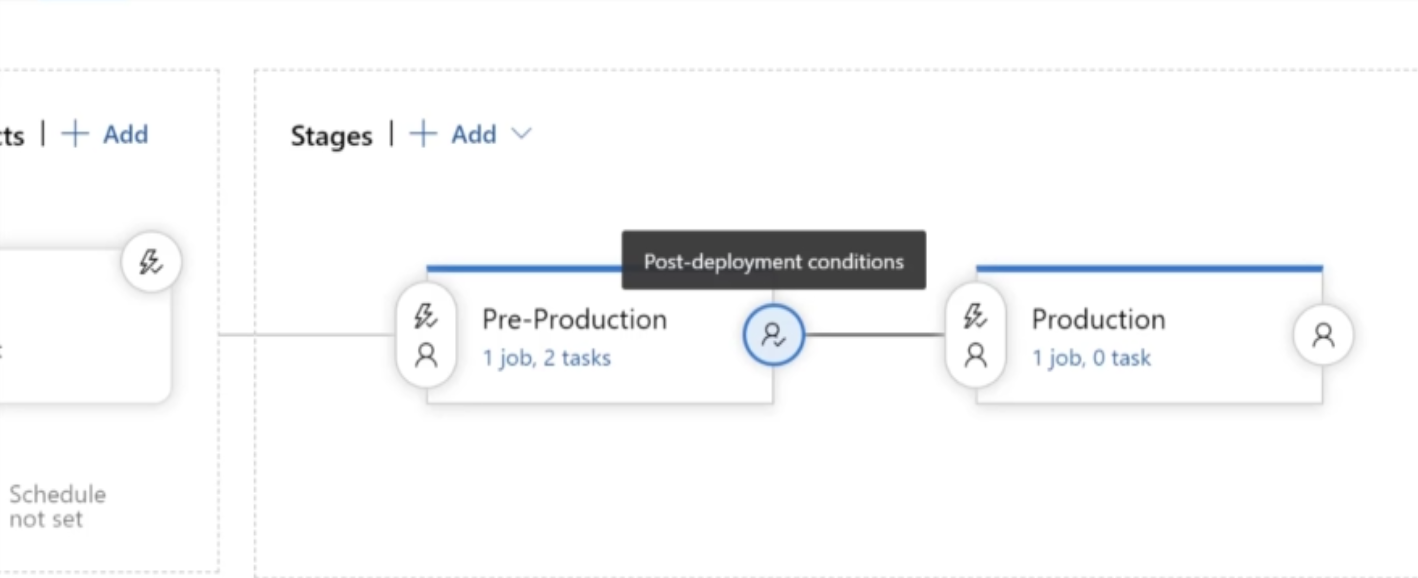
Azure -> Create Web App -> Go to Web App Resource -> Click Deployment Slot -> Add Slots



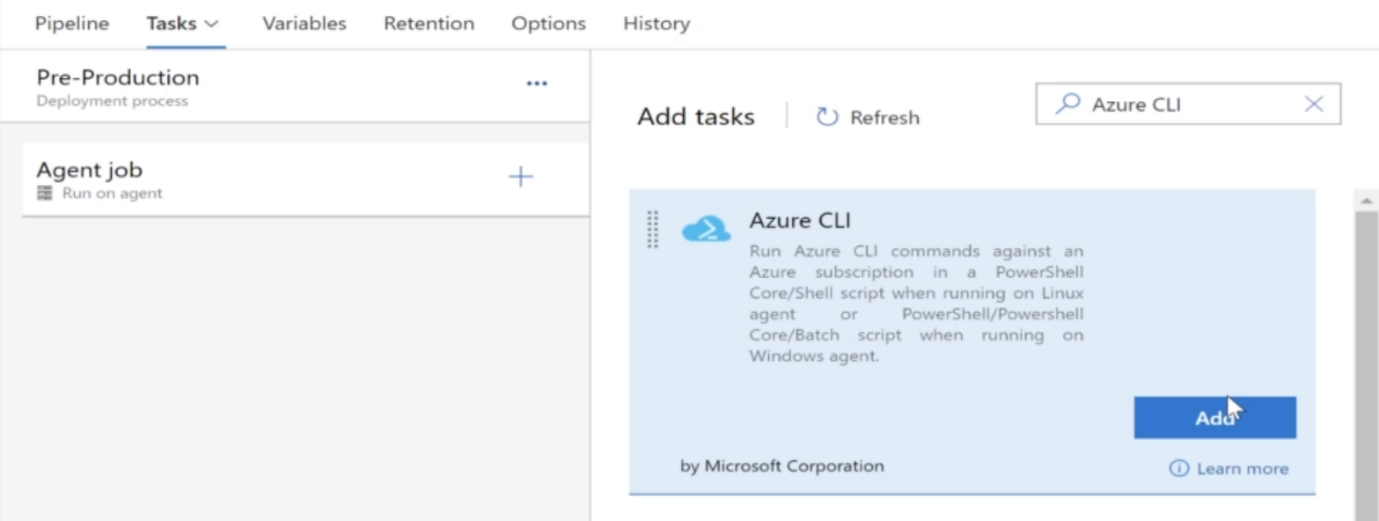
1. **Deployment Slots in Azure Pipeline:**

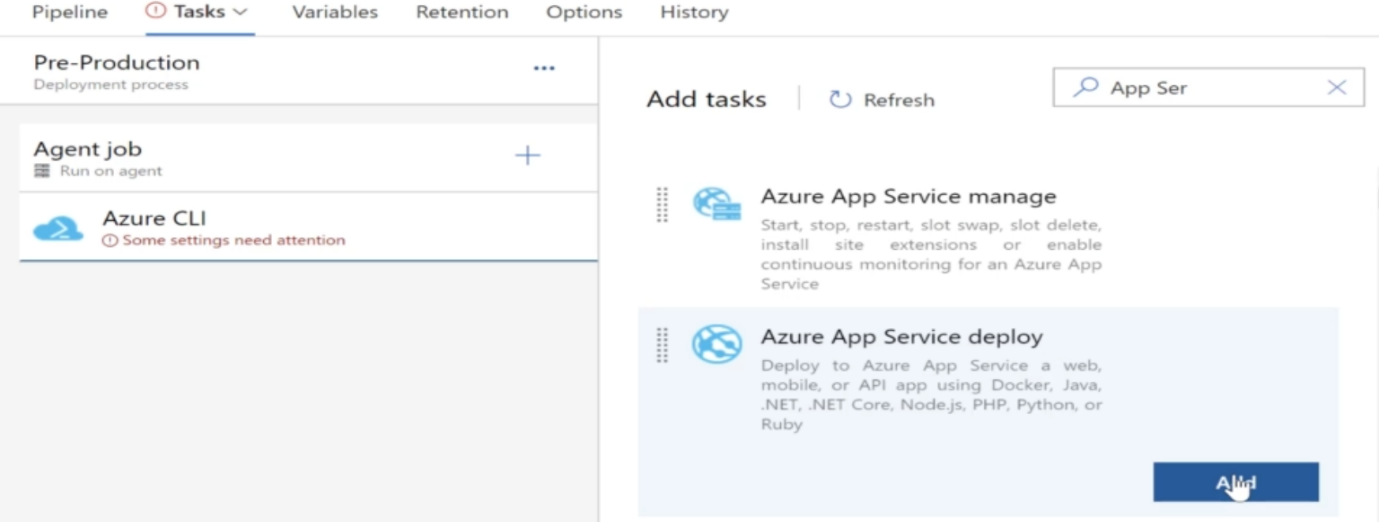
**Step 1:** Go to Azure DevOps -> Create Repository -> Push to code into Repository -> Now we will create deployment lot using CLI command

**Step 2:** Create new Build Pipeline -> Build the Project

**Step 3:** Create release Pipeline -> Add Task -> Set Pre-Deployment Condition on Pre-Production slot – > and based on the approval swap the Pre-production to Production.

**Step 4:** Add Azure CLI Task & Azure App Service deploy task

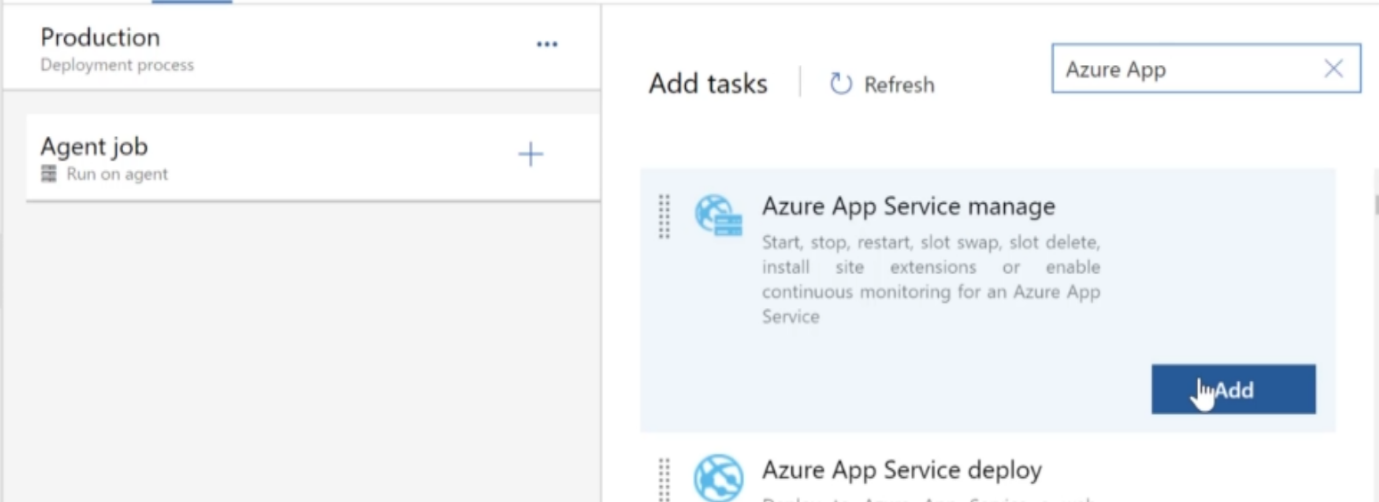
****

****

**Step 5:** Configure the Azure CLI andApp Service deploy task

****

**Step 6:** For Production Slot -> Add Azure App Service Manage-> And Set the swap slot from staging

****