# 😊 Self-Hosted Agent

**Self-Hosted Agent in Azure DevOps**

A **self-hosted agent** in Azure DevOps is a machine (on-premises or in the cloud) that runs build and deployment jobs for Azure DevOps pipelines. Unlike Microsoft-hosted agents, which are managed by Microsoft and run in the cloud, self-hosted agents are installed and maintained by users.

**Key Features of Self-Hosted Agents**

1. **Full Control Over Configuration**
    - Users can install any required dependencies, tools, or SDKs.
    - Custom environments can be configured for specific builds.
2. **Persistent File System and Caching**
    - Unlike Microsoft-hosted agents, self-hosted agents retain caches and files between jobs, reducing build times.
    - Useful for caching dependencies like NuGet, npm, or Maven.
3. **No Time Limits on Builds**
    - Microsoft-hosted agents impose time limits on free-tier usage, whereas self-hosted agents run indefinitely as long as they are available.
4. **Cost Efficiency**
    - Self-hosted agents can help reduce costs by leveraging existing hardware instead of paying for Microsoft-hosted agent minutes.
5. **Access to Private Networks**
    - Self-hosted agents can access internal resources, databases, or private services, making them ideal for enterprise environments.

**Use Cases for Self-Hosted Agents**

**1. Build and Deployment for Large Codebases**

- When dealing with large applications, build times can be optimized using high-performance self-hosted machines.

**2. Running Jobs on Specialized Hardware**

- If a project requires GPU acceleration, specific CPU architectures, or hardware-specific testing, self-hosted agents provide the necessary flexibility.

**3. CI/CD for On-Premises Applications**

- Organizations that maintain applications in on-premises data centers can use self-hosted agents to integrate with internal systems.

**4. Security and Compliance Requirements**

- Enterprises with strict security policies may require build agents to be inside a controlled network environment rather than using public cloud resources.

**5. Custom Software Dependencies**

- If the build process requires specific software that is not available on Microsoft-hosted agents, a self-hosted agent allows complete customization.

**Benefits of Using Self-Hosted Agents**

**Performance Optimization**

- Allows for the use of high-performance machines, reducing build and deployment times.
- Persistent storage helps avoid downloading dependencies in every build.

**Security and Control**

- Enables running jobs within a controlled environment with custom security policies.
- Reduces exposure to public cloud risks.

**Flexibility in Software Installation**

- Users can pre-install required SDKs, tools, and dependencies, ensuring compatibility with specific projects.

**Cost Reduction**

- Eliminates the cost associated with using Microsoft-hosted agents, especially for long-running jobs.
- Can be scaled as needed using existing infrastructure.

**Access to Private Resources**

- Self-hosted agents can interact with private APIs, databases, and other internal services without exposing them to the public internet.

**Conclusion**

Self-hosted agents in Azure DevOps provide greater control, security, and cost-efficiency compared to Microsoft-hosted agents. They are particularly useful for organizations with large codebases, custom infrastructure requirements, or strict compliance needs. By leveraging self-hosted agents, teams can optimize build performance, integrate with private networks, and maintain a more secure DevOps pipeline.

**Summary**

In this lab, we are setting up a **self-hosted agent** in Azure DevOps using a **Windows Server 2019** virtual machine (VM). The VM is configured with **Git and .NET 6.0 SDK** to build and deploy a .NET application.

**Steps Overview**

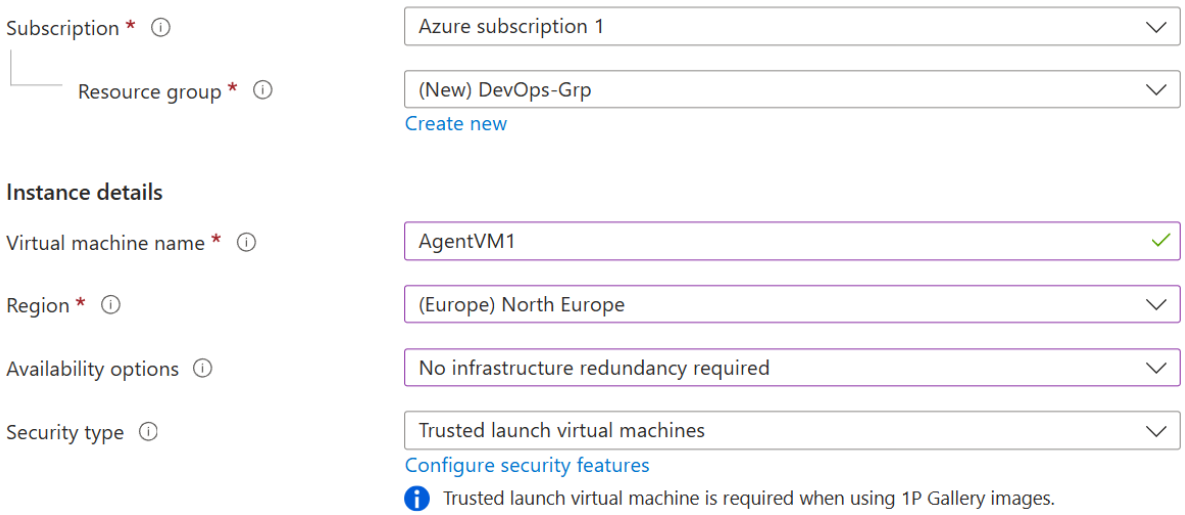
1. **Create a Windows Server 2019 VM** in Azure.
2. **Install necessary tools** (Git and .NET 6.0 SDK) on the VM.
3. **Generate a Personal Access Token (PAT)** in Azure DevOps for authentication.
4. **Configure the VM as an Azure DevOps agent**, register it in an Agent Pool, and verify connectivity.
5. **Modify the Azure DevOps pipeline** to use the self-hosted agent.
6. **Run the pipeline**, ensuring it builds and deploys using the configured VM.

**End Goal**

The objective is to create a **custom self-hosted agent** that allows greater **control, security, and flexibility** in running Azure DevOps pipelines, especially for building and deploying a .NET application. This ensures persistent caching, reduced build times, and access to private resources.

## 😄 To begin with the Lab

1. In this lab we will create a self-hosted agent, for that we need to open the Azure Portal. Then go and create a Virtual Machine based on Windows 2019 data center.
2. Choose your subscription, and resource group, give a name to the VM, and choose your region.




3. Then choose the image as Windows Server 2019 datacenter and the size based on your preference. After that give a username and password to the VM and move to the **review page to create your VM**.

Image * ⓘ — Windows Server 2019 Datacenter - x64 Gen2 (free services eligible)

See all images | Configure VM generation

VM architecture ⓘ — Arm64 / x64

Arm64 is not supported with the selected image.

Run with Azure Spot discount ⓘ

Size * ⓘ — Standard_D2s_v4 - 2 vcpus, 8 GiB memory ($145.27/month)

See all sizes

Enable Hibernation ⓘ

Hibernate is not supported by the size that you have selected. Choose a size that is compatible with Hibernate to enable this feature. Learn more

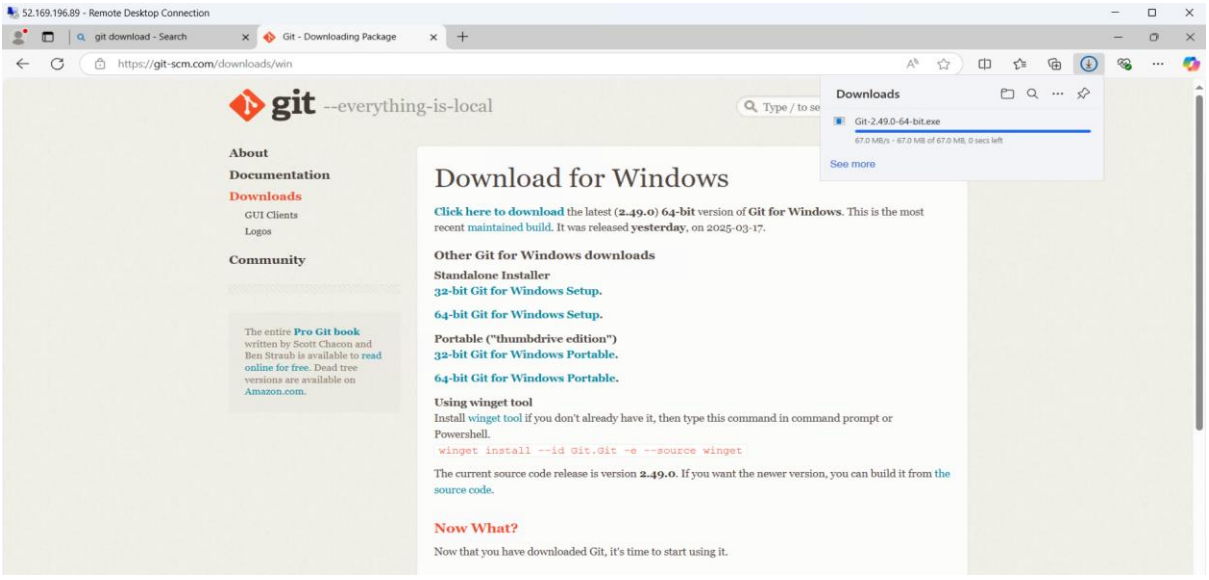
**Administrator account**

Username * ⓘ — demo

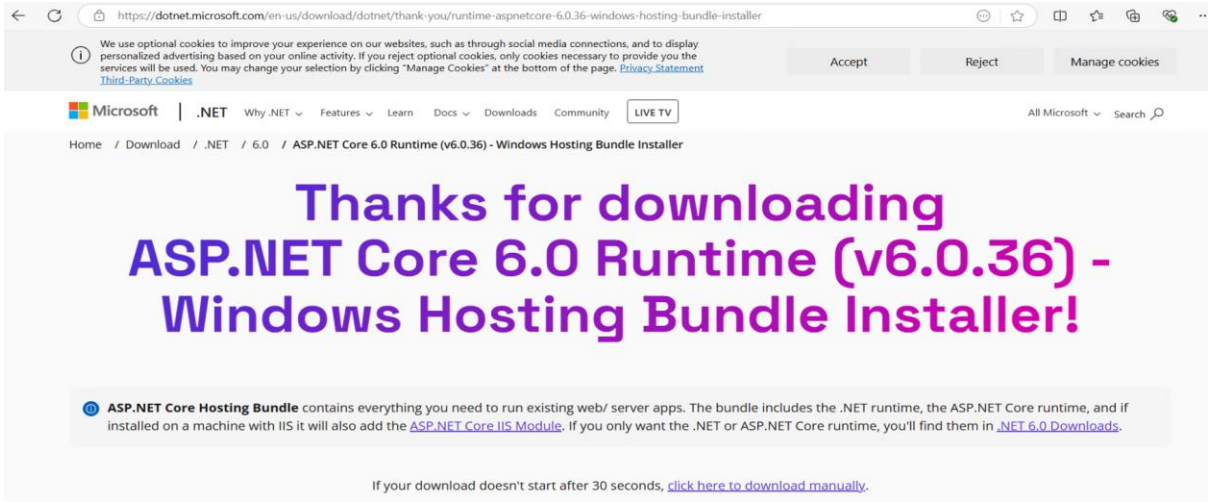
Password * — ••••••••••••

Confirm password * — ••••••••••••

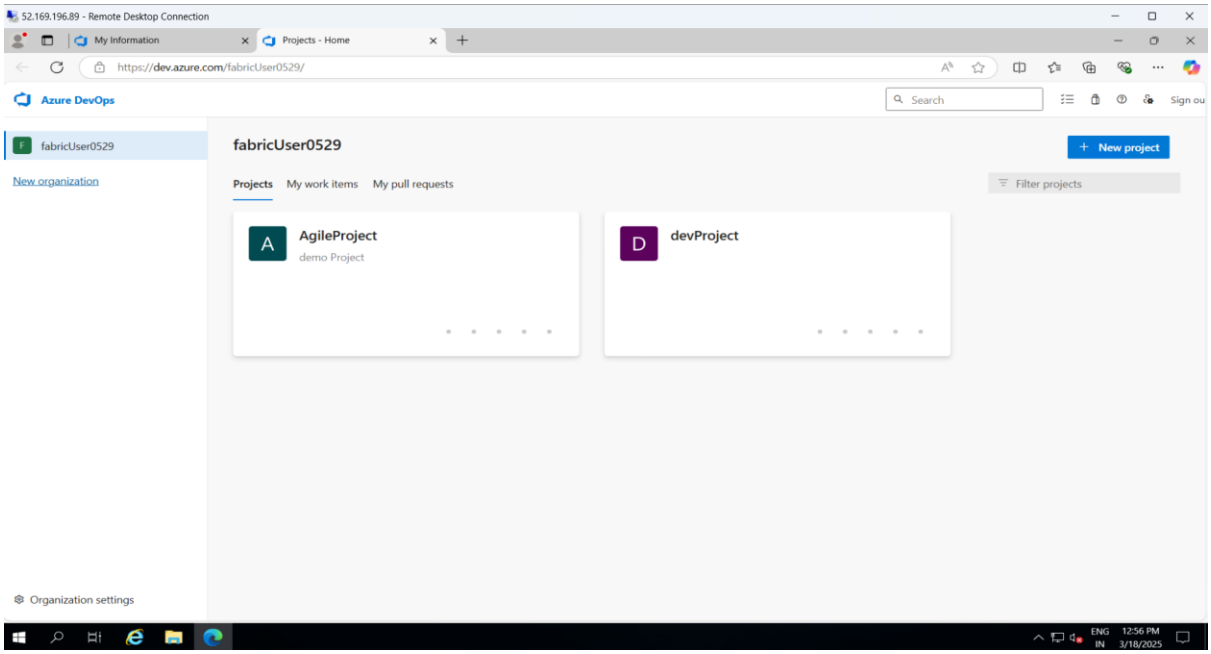
4. Once your Virtual Machine has been created then open it using RDP connection.
5. Now remember, this machine is going to be used now to build our .NET application code. So, what do we need to install on this machine? We need to install .NET 6.0 when it comes onto the SDK because this will be used for building the application. In addition to that, we also need to install the Git tool. The Git tool is going to be used to pull out the code from Azure onto this machine. So, Git is also required as a tool.
6. Once you are inside the VM open the Microsoft Bing browser in it and download the Git tool and install it.



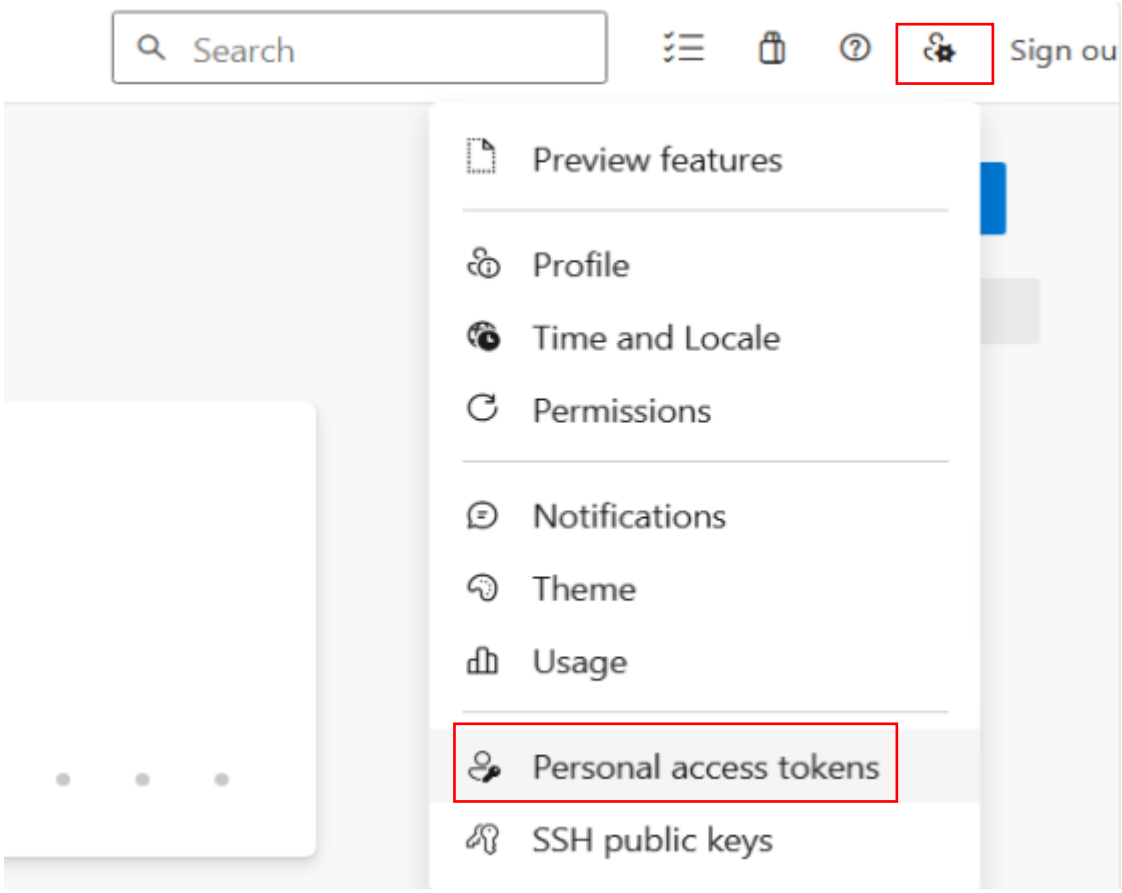

7. After installing Git, you need to download and install dot net 6 on your VM.



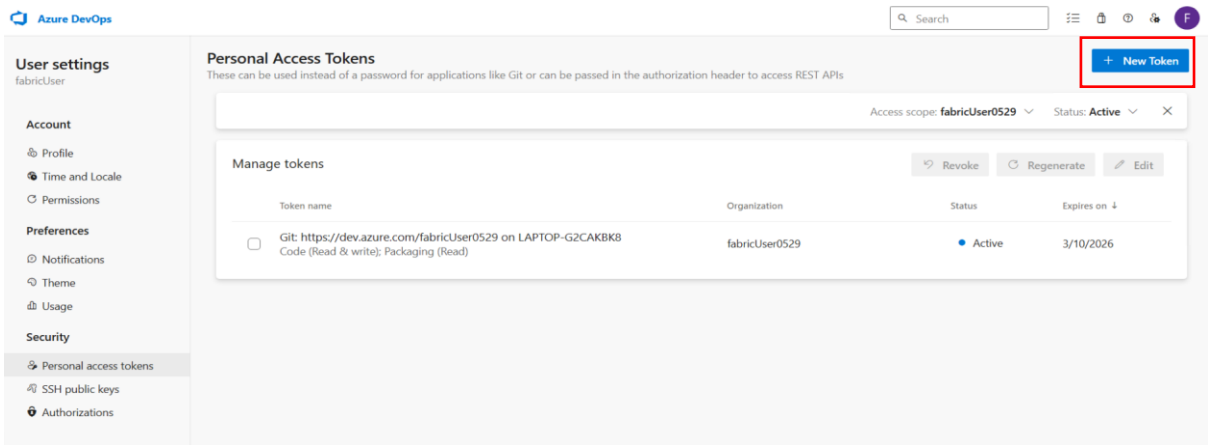

8. After installing both of the things on your VM you need to login to your Azure DevOps account inside your VM.
9. Here you can see that we are signed in successfully.



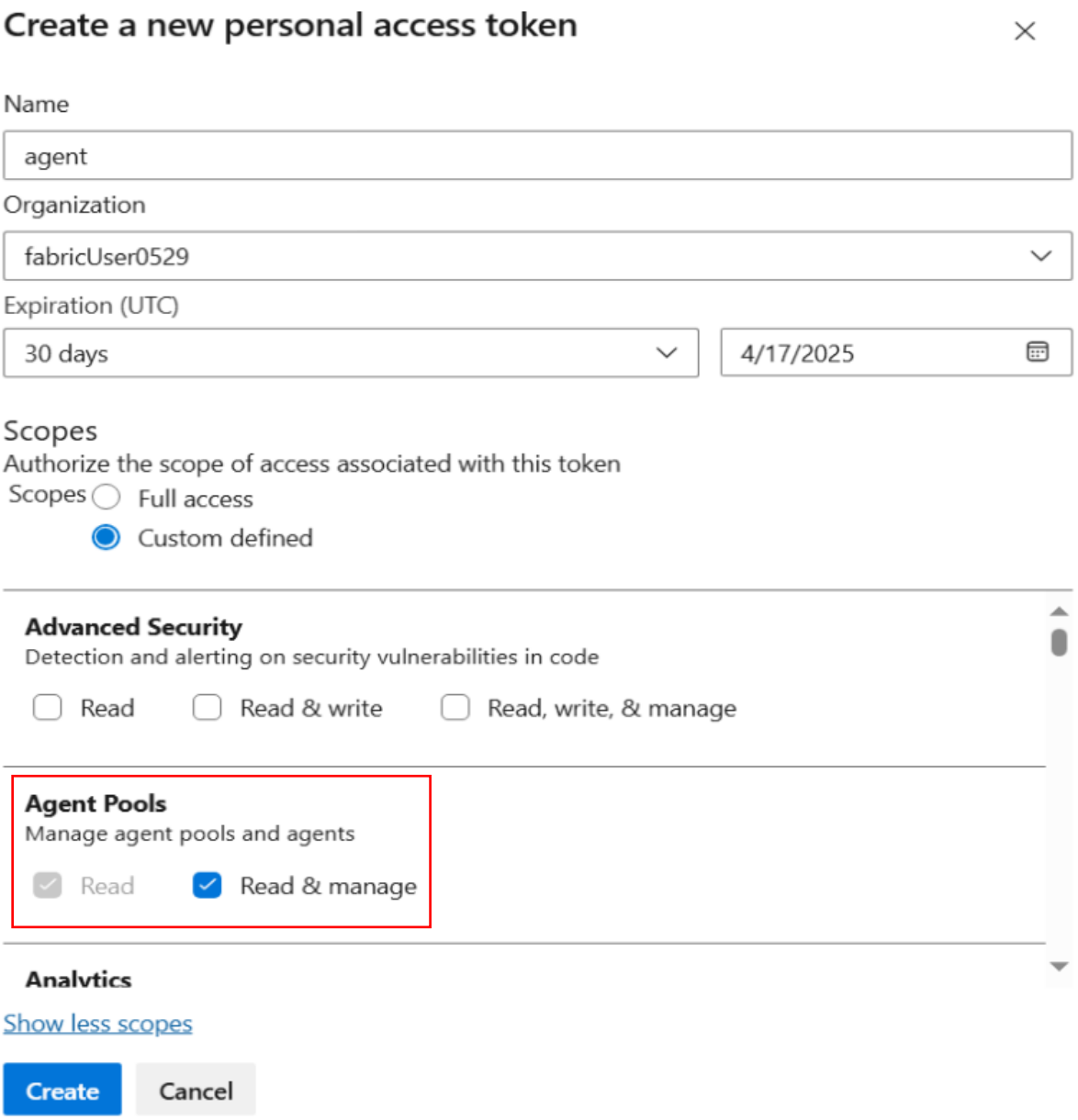

10. Now we need to click on our profile and choose to create a personal access tokens.



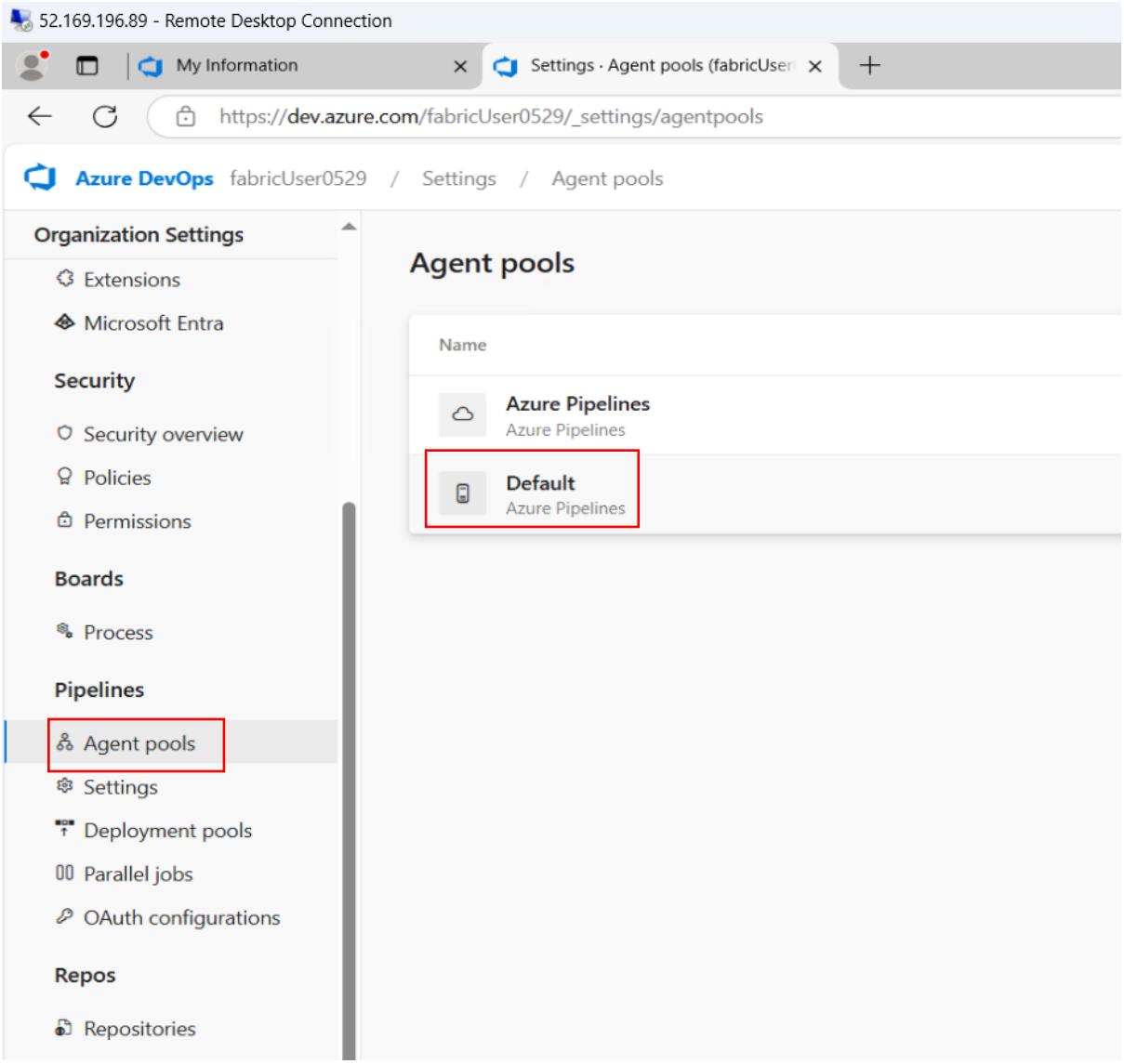

11. Then click on the new token. First, we'll start by giving it a name then in terms of the scope, we have to decide now what access we want to give as part of this personal access token. It's like a password, but this password carries some sort of permissions along with it. Here we have to search for agent pools



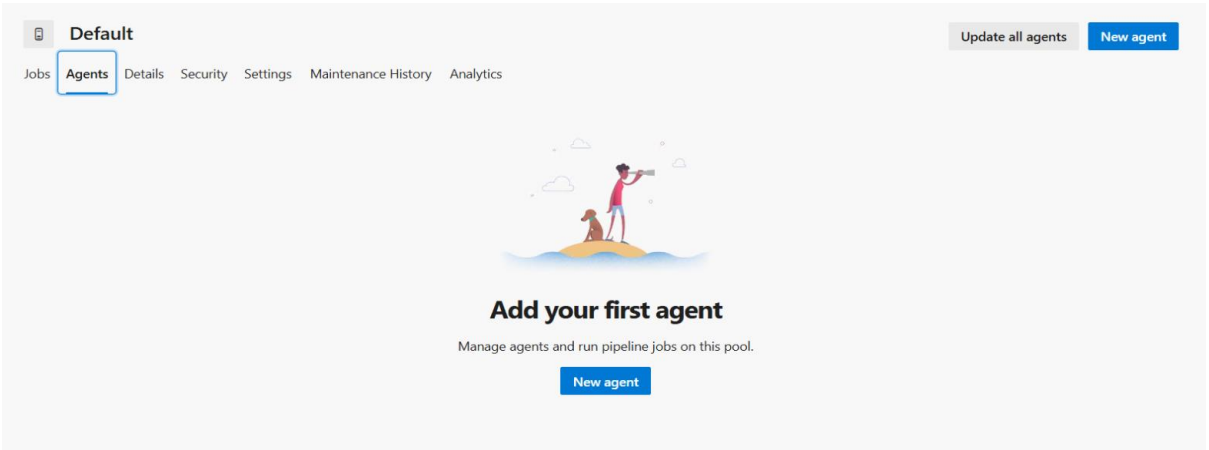

12. Search for all the scopes then choose agent pools and give access for both read and manage. Then just **copy the password** it has given to you and **copy it onto the notepad**.



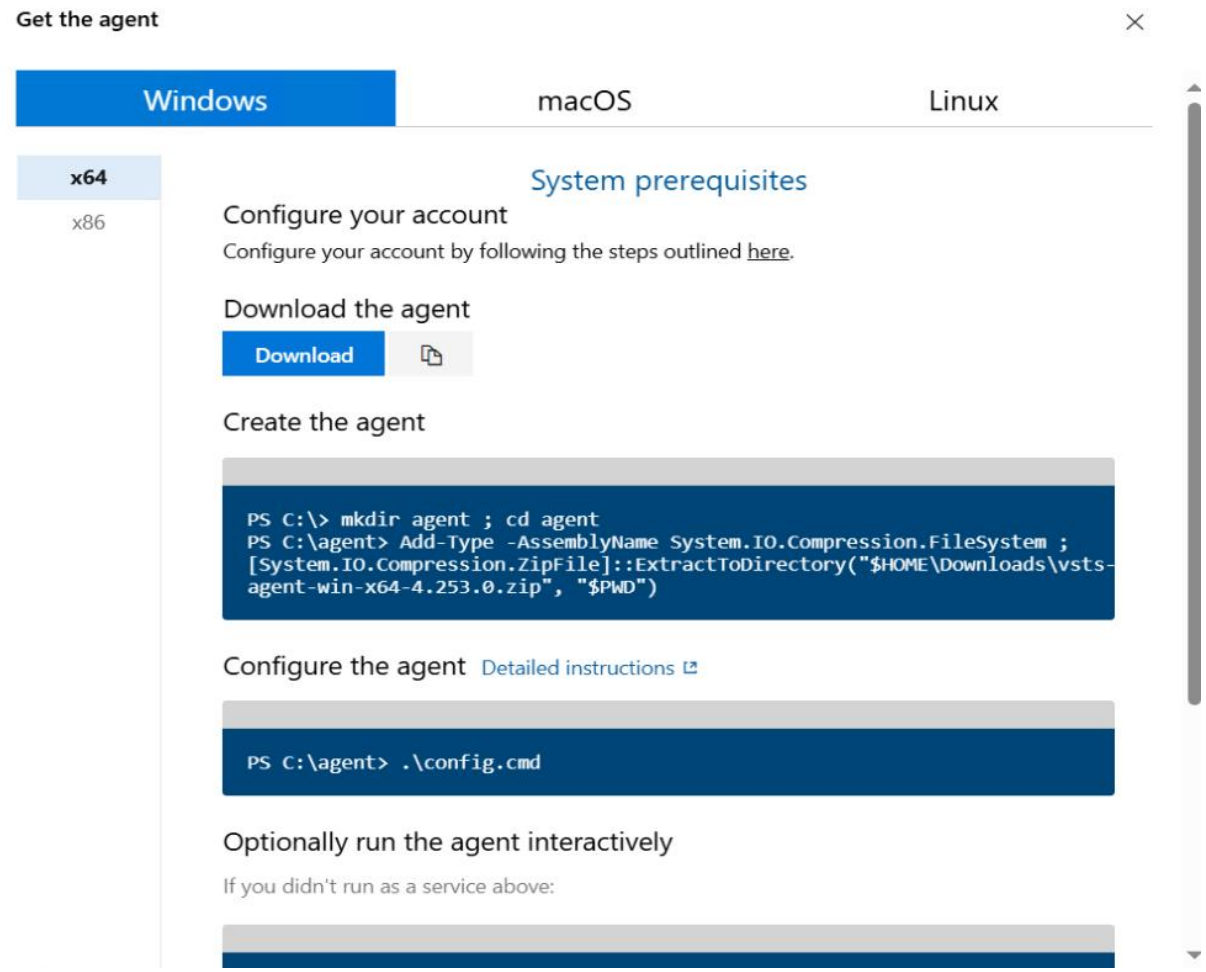

13. Now you need to navigate to **organization settings** and go to **Agent Pools** and choose the default Pool.



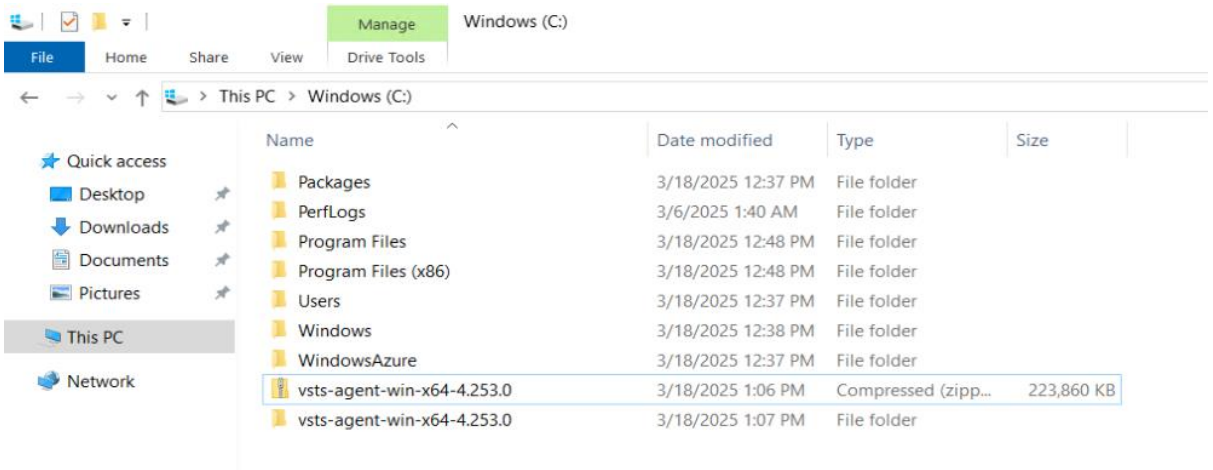

14. In the default Pool go to agents and click on new agent.

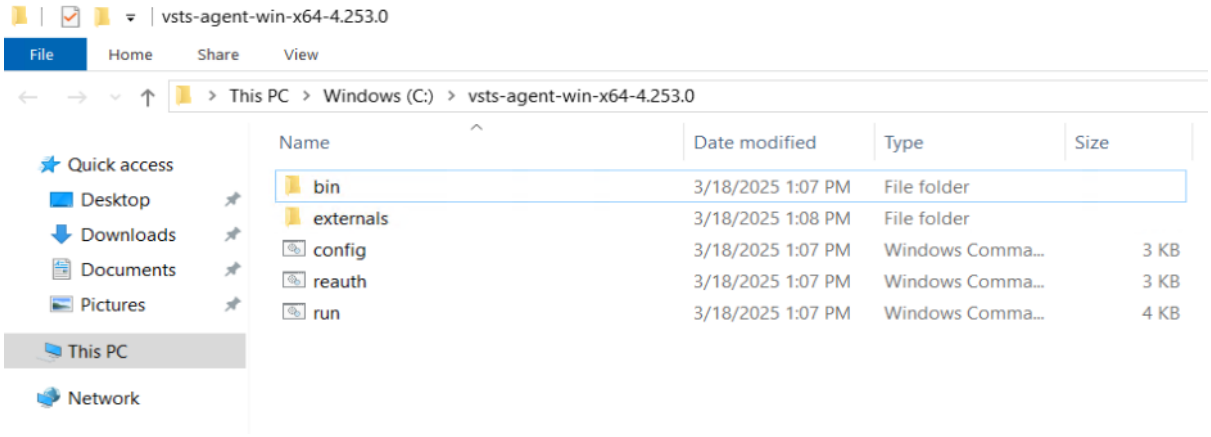



15. Then, based on whether you have a Windows machine a Mac OS machine, or a Linux machine, that is behaving as your agent, it'll let you know on the steps that you need to implement to ensure that this machine now behaves as an agent for your Azure DevOps organization. Click on download.

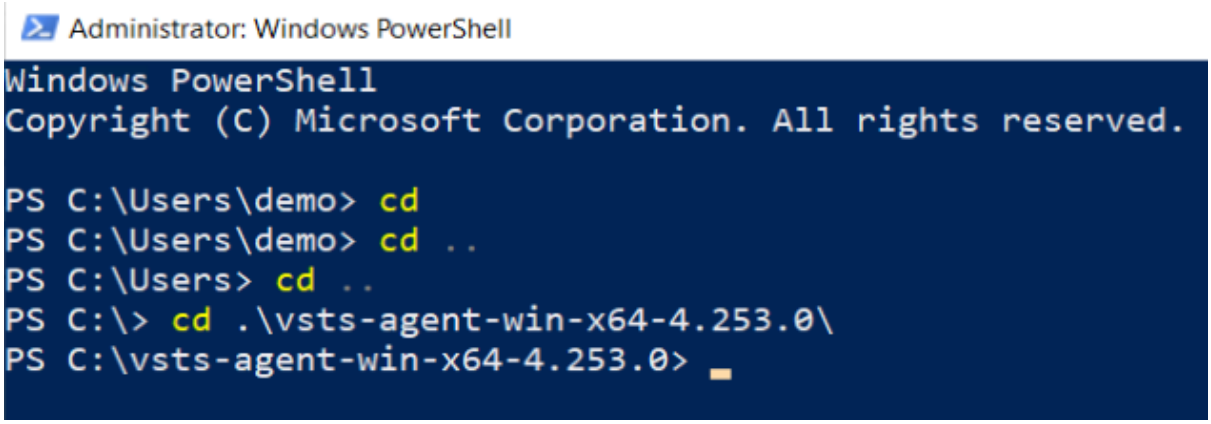



16. Once the folder has been downloaded you need to copy it and bring it into the **C drive inside your VM then extract it**. After extraction, you will see that it has some folders and files in place.






17. Now you need to open the PowerShell on your VM. Then open the agent folder in your PowerShell.




18. Then you need to run the config file as you can see in the snapshot. This config file will start the agent then you need to give the server URL for that you need to copy the URL from the browser.
19. After that you need to give the personal access token and register your Agent.

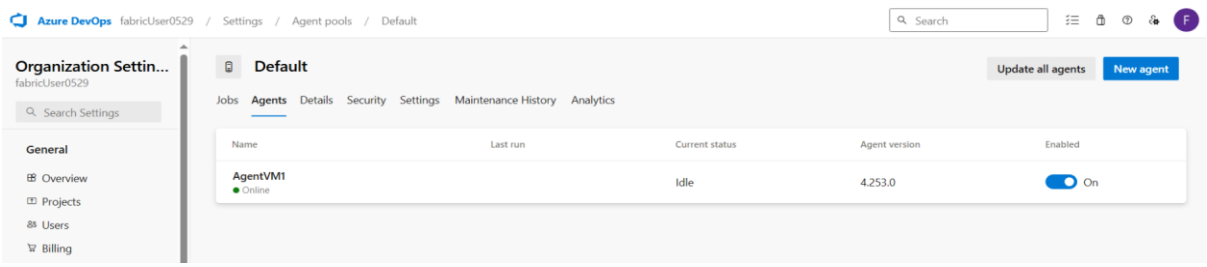
    **./config.cmd**

```
PS C:\vsts-agent-win-x64-4.253.0> ./config.cmd

 ___                    ______ _        _ _
 / _ \                   | ___ (_)      | (_)
/ /_\ \____  _ _ __ ___  | |_/ /_ _ __  | |_ _ __   ___ ___
|  _  |_  / | | '__/ _ \ |  __/| | '_ \ / _ \ | '_ \ / _ \ __|
| | | |/ /| |_| | |  __/ | |   | | |_) |  __/ | | | |  __/\__ \
\_| |_/___|\__,_|_|  \___| \_|   |_| .__/ \___|_|_|_| |_|\___||___/
                                    | |
       agent v4.253.0               |_|          (commit 5263f72)

>> Connect:

Enter server URL > https://dev.azure.com/fabricUser0529
Enter authentication type (press enter for PAT) >
Enter personal access token > ********************************************************************************************
Connecting to server ...

>> Register Agent:

Enter agent pool (press enter for default) >
Enter agent name (press enter for AgentVM1) >
Scanning for tool capabilities.
Connecting to the server.
Successfully added the agent
Testing agent connection.
Error reported in diagnostic logs. Please examine the log for more details.
    - C:\vsts-agent-win-x64-4.253.0\_diag\Agent_20250318-131214-utc.log
Enter work folder (press enter for _work) > C:\work
2025-03-18 13:15:11Z: Settings Saved.
```

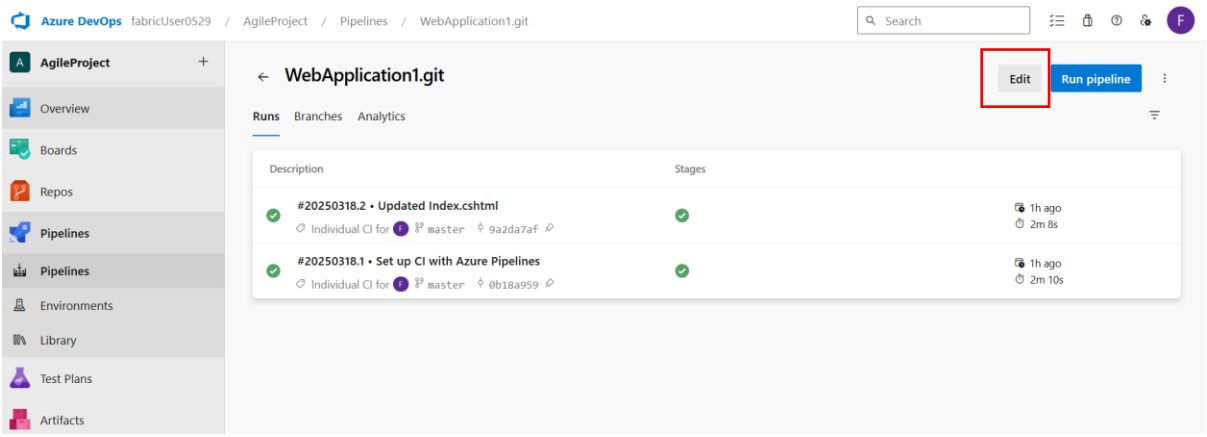
20. Follow the snapshot for the completion to start your agent.

```
Enter run agent as service? (Y/N) (press enter for N) > Y
Enter enable SERVICE_SID_TYPE_UNRESTRICTED for agent service (Y/N) (press enter for N) > Y
Enter User account to use for the service (press enter for NT AUTHORITY\NETWORK SERVICE) > Y
Enter a valid value for User account to use for the service.
Enter User account to use for the service (press enter for NT AUTHORITY\NETWORK SERVICE) >
Granting file permissions to 'NT AUTHORITY\NETWORK SERVICE'.
Service vstsagent.fabricUser0529.Default.AgentVM1 successfully installed
Service vstsagent.fabricUser0529.Default.AgentVM1 successfully set recovery option
Service vstsagent.fabricUser0529.Default.AgentVM1 successfully set to delayed auto start
Service vstsagent.fabricUser0529.Default.AgentVM1 successfully set SID type
Service vstsagent.fabricUser0529.Default.AgentVM1 successfully configured
Enter whether to prevent service starting immediately after configuration is finished? (Y/N) (press enter for N) >
Service vstsagent.fabricUser0529.Default.AgentVM1 started successfully
PS C:\vsts-agent-win-x64-4.253.0>
```

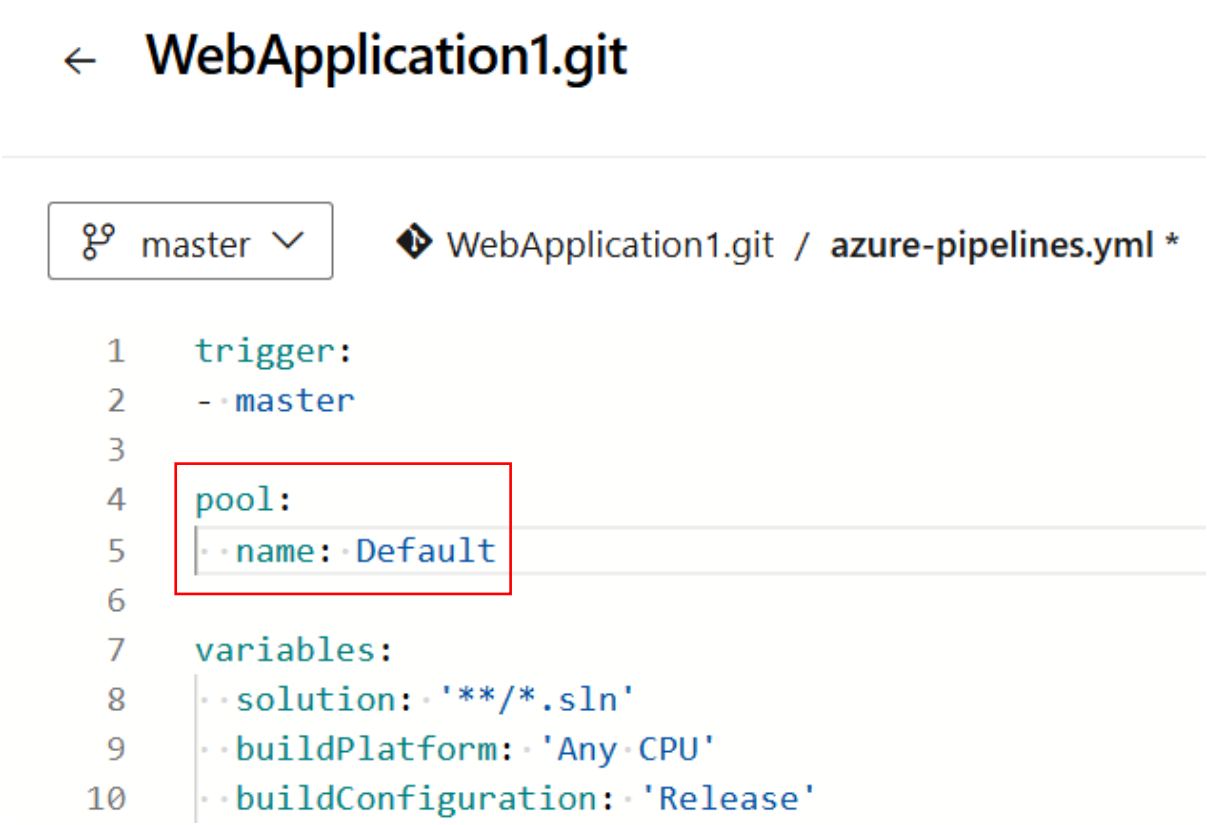
20. Go to the browser and you will see that your agent is online now. As our agent is online now, we can make use of it to run our pipeline.



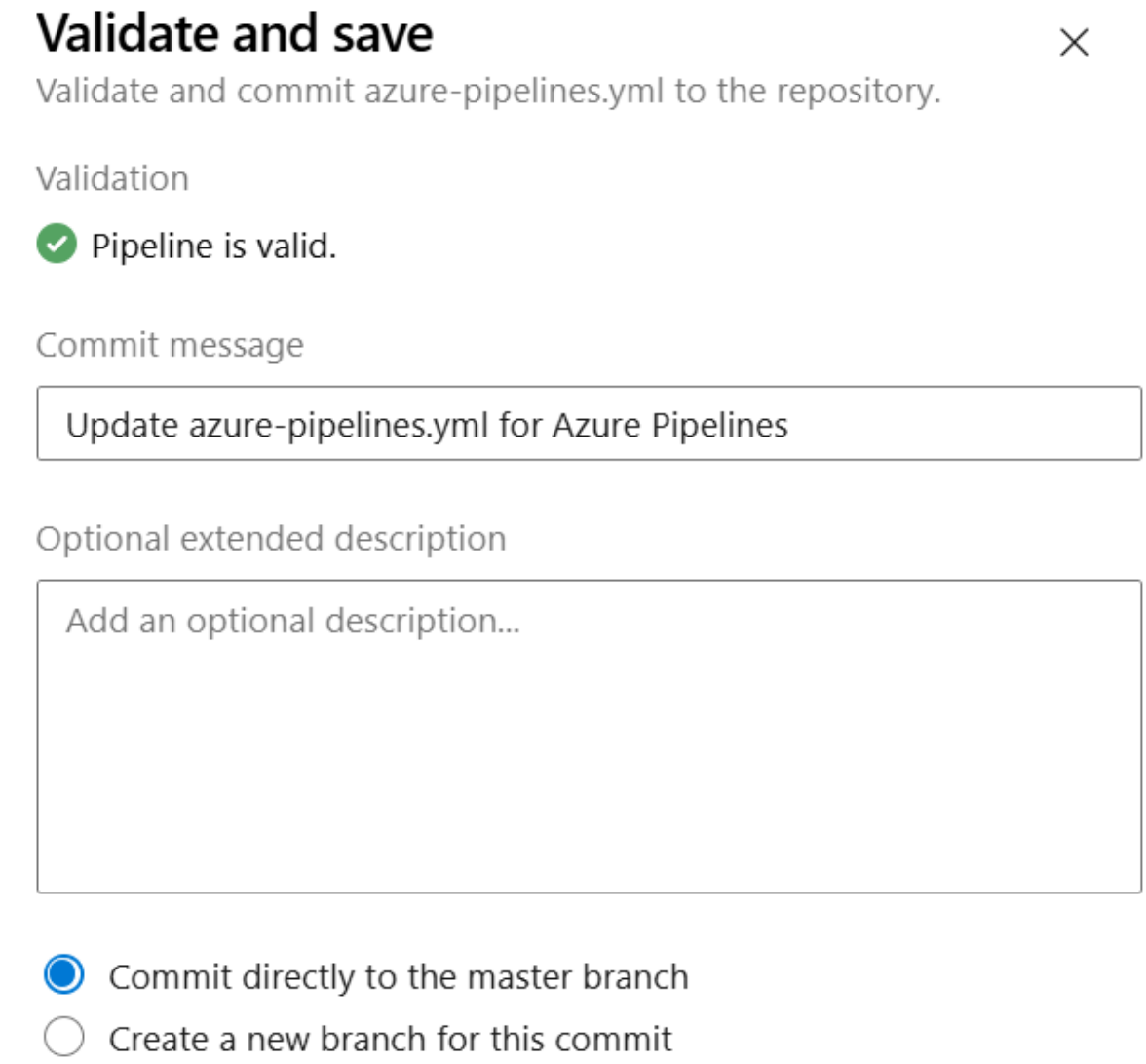

21. Now for the build pipeline that we have in our Azure DevOps, we can edit it.



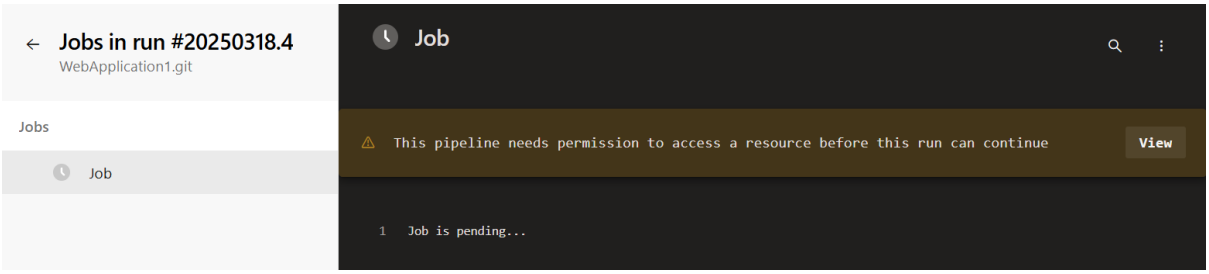

22. Here we just need to edit the pool on line number 4, and on line number 5 mention the pool as Default as you can see below, and leave everything as it is.

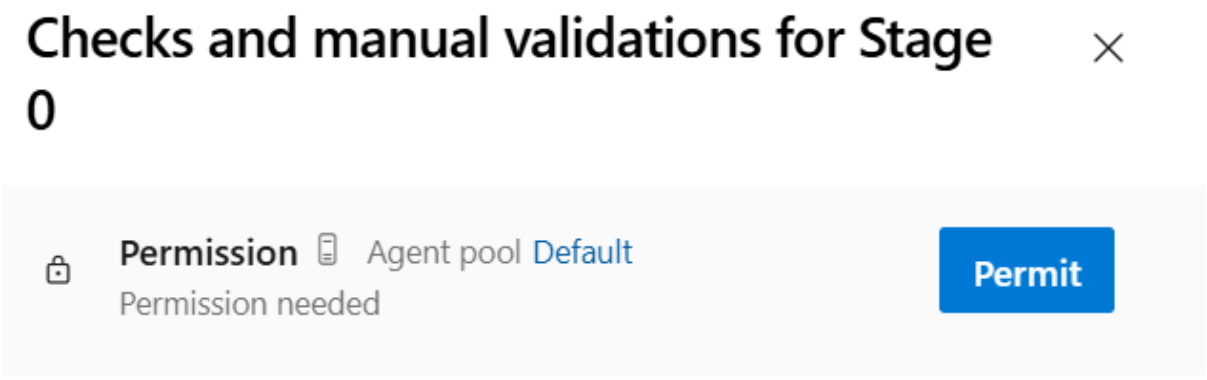



23. Then click on validate and save to save everything. In the end just run your pipeline.

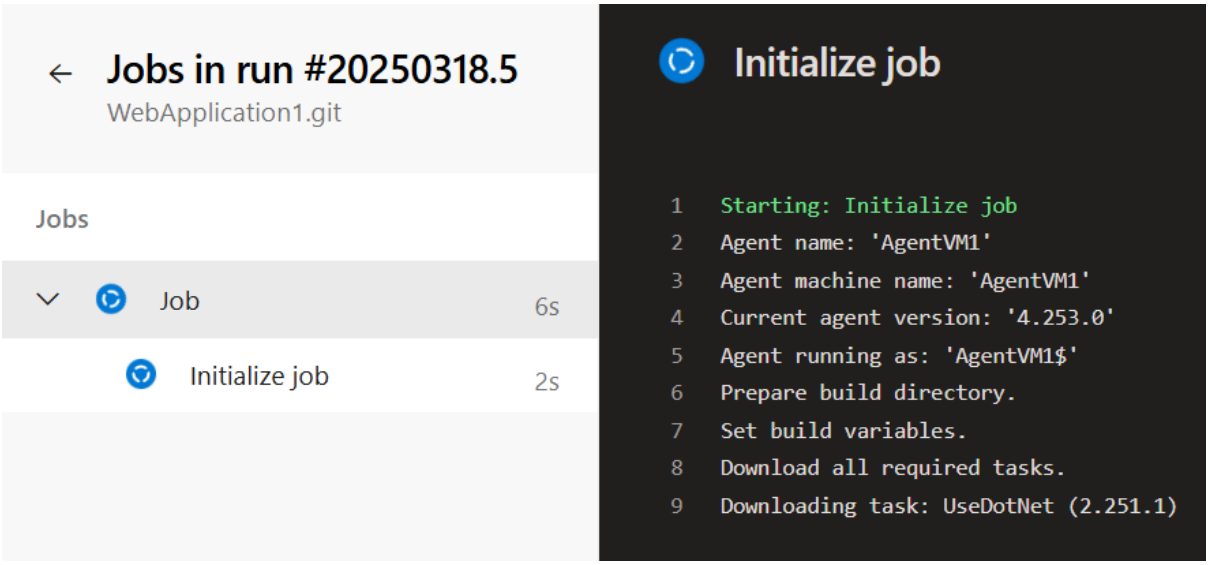



24. If you go to your jobs you will see that the pipeline needs permission, click on view and permit the pipeline to use your self-hosted agent to run the pipeline.

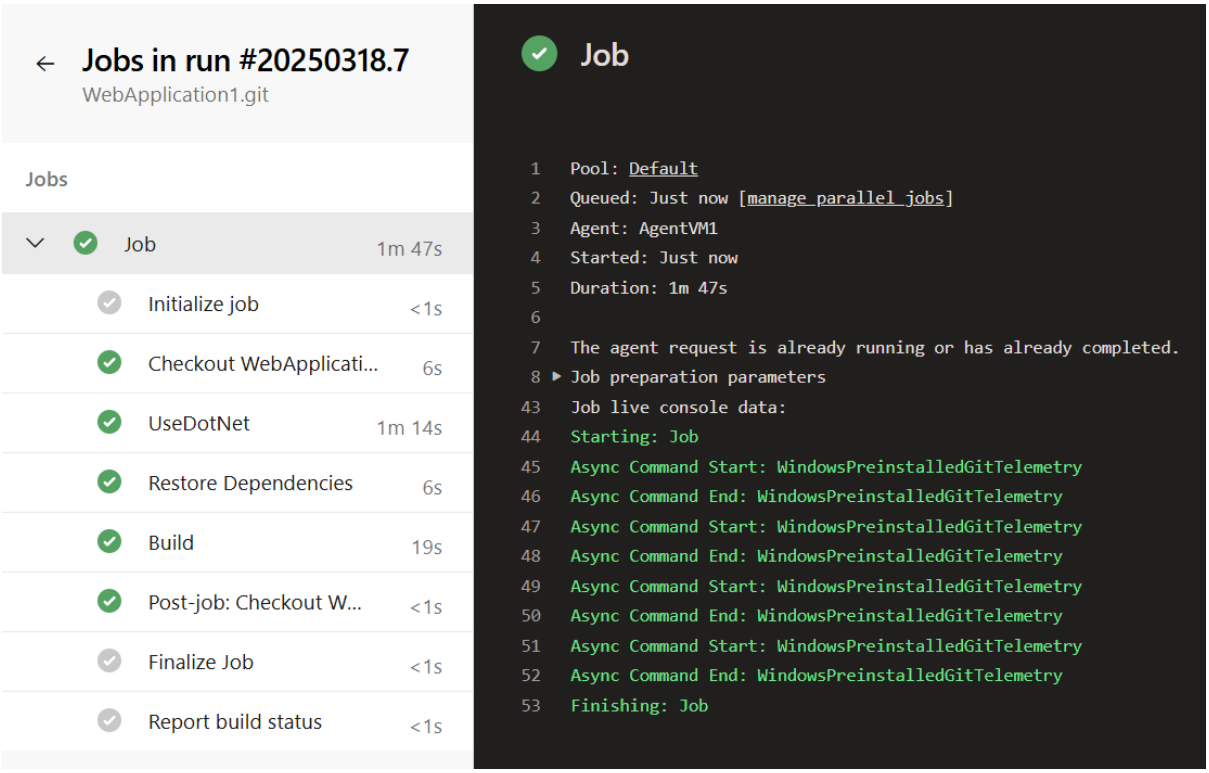





25. Here you can see that it has picked up the VM agent. As this is a self-hosted agent this pipeline may take more than 5 minutes.




26. Below you can see that our pipeline run is complete using the self-hosted environment.


Jobs in run #20250318.7
WebApplication1.git
Jobs
Job 1m 47s
Initialize job <1s
Checkout WebApplicati... 6s
UseDotNet 1m 14s
Restore Dependencies 6s
Build 19s
Post-job: Checkout W... <1s
Finalize Job <1s
Report build status <1s
Job
1 Pool: Default
2 Queued: Just now [manage parallel jobs]
3 Agent: AgentVM1
4 Started: Just now
5 Duration: 1m 47s
6
7 The agent request is already running or has already completed.
8 ▶ Job preparation parameters
43 Job live console data:
44 Starting: Job
45 Async Command Start: WindowsPreinstalledGitTelemetry
46 Async Command End: WindowsPreinstalledGitTelemetry
47 Async Command Start: WindowsPreinstalledGitTelemetry
48 Async Command End: WindowsPreinstalledGitTelemetry
49 Async Command Start: WindowsPreinstalledGitTelemetry
50 Async Command End: WindowsPreinstalledGitTelemetry
51 Async Command Start: WindowsPreinstalledGitTelemetry
52 Async Command End: WindowsPreinstalledGitTelemetry
53 Finishing: Job