Project Building CI/CD Pipeline

It contains source controller, CI, CD and build Mach learning applications and run on serverless environments

Azure Pipeline Agent

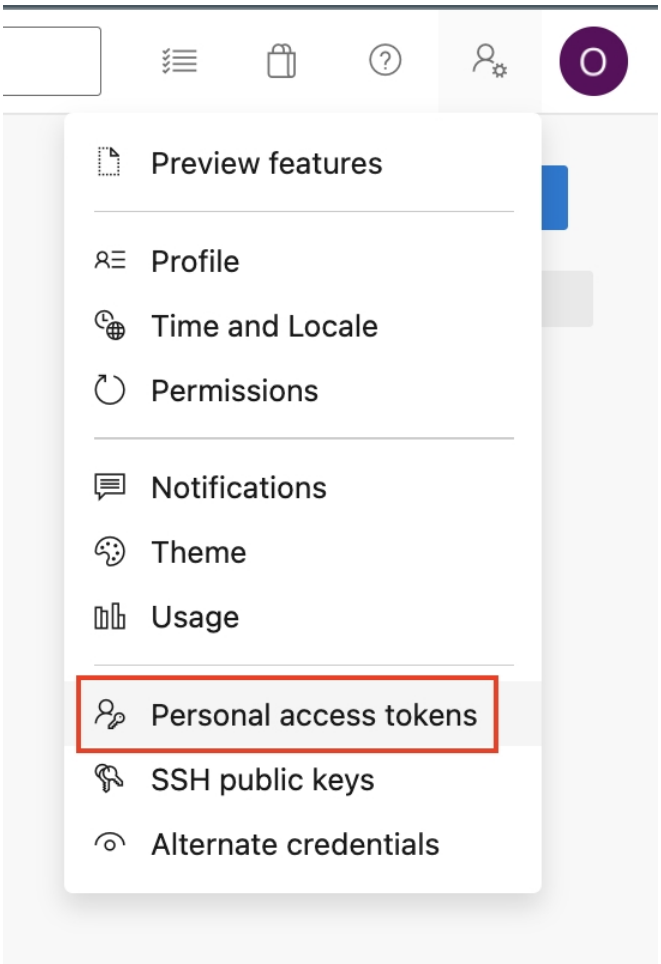
If you are using Udacity Cloud Lab, you will have to create your self-hosted Azure pipeline agent (a new Linux VM) that will build and deploy the code. Here are the steps to create an Azure pipeline agent.

Prerequisites

- You should have logged into the https://portal.azure.com/ and https://dev.azure.com/ in a separate browser tabs.
- You should have DevOps project, say **Flask-ML-Deploy**, available in your DevOps account.
- Your DevOps project should have a **service connection** created using the Azure Resource Manager and Service principal (manual)

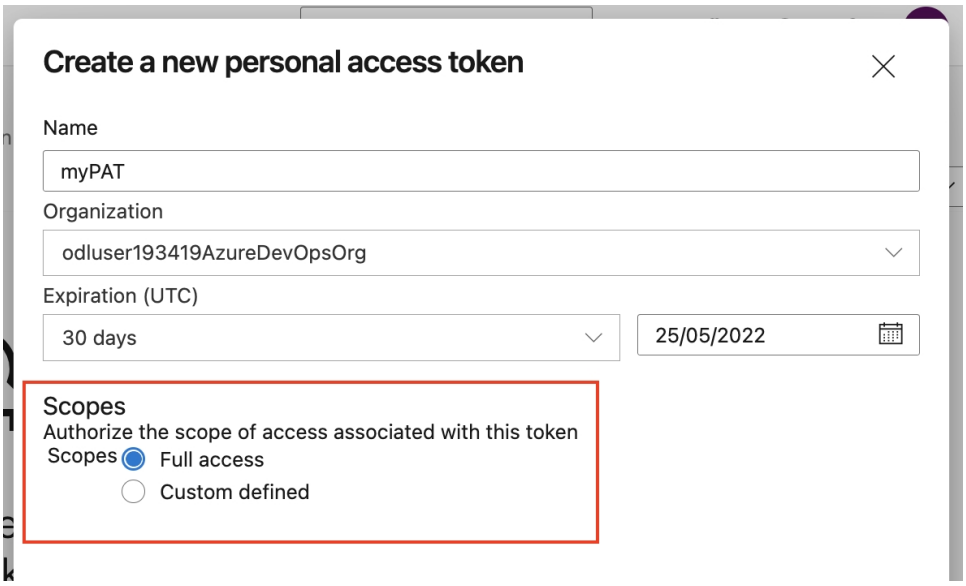
Task 1 - Create a Personal Access Token (PAT)

- Create a new Personal Access Token (PAT) that will be used instead of a password by the build agent (Linux VM) for authentication/authorization. To create a PAT, go to https://dev.azure.com/ home and click on the top-right user icon, as shown below.



Creating PAT in Azure DevOps

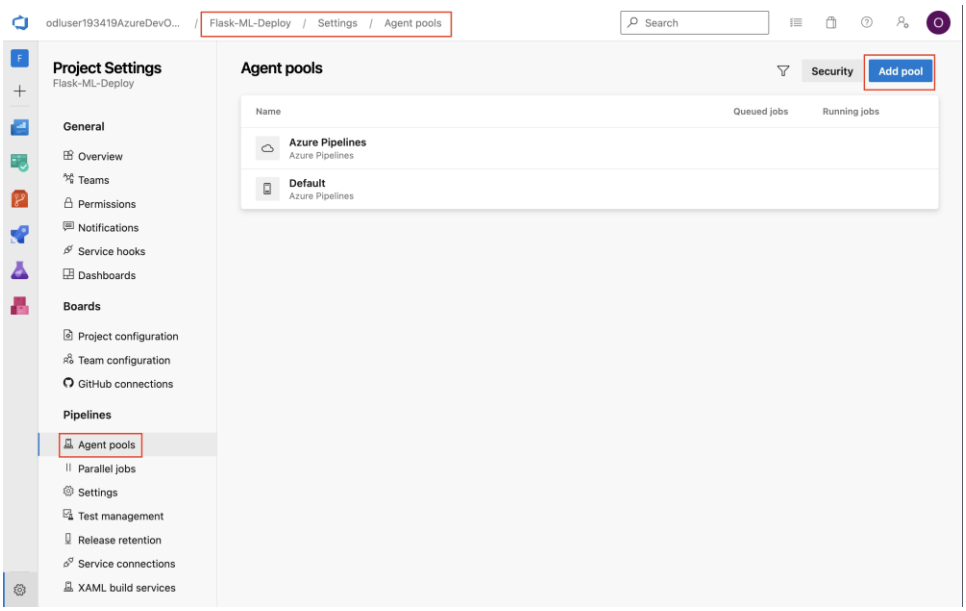
- Create a new PAT, and ensure that it has a "Full access" scope. **Save the PAT value for future use. You will not be able to view it again.**



Task 2. Create an Agent pool

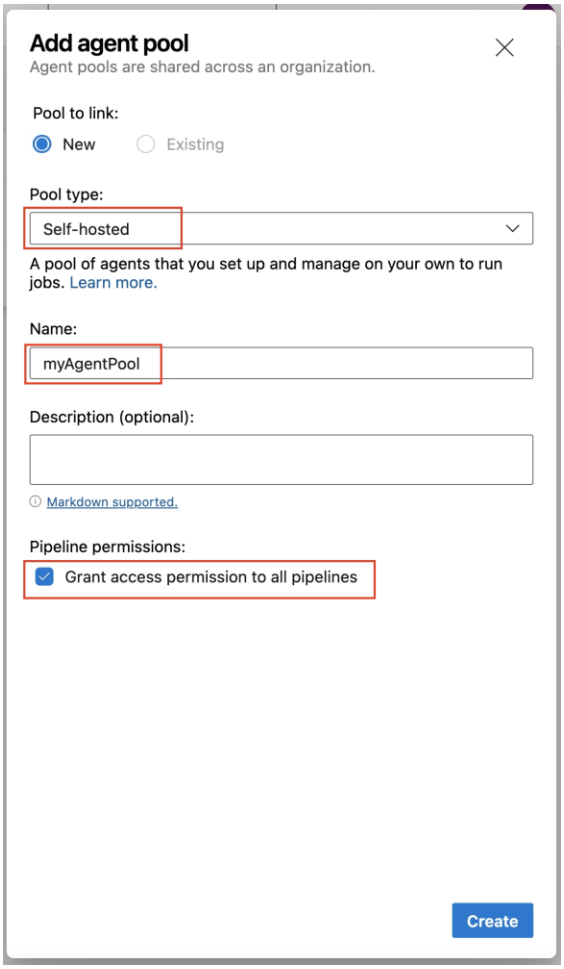
An Agent pool is a collection of the agents (VMs) that will build your code and deploy it to the Azure services. The agent is the machine that does the processing job of the pipeline.

- Go to the **Flask-ML-Deploy** DevOps project Settings >> Agent pools and add a new agent pool.



Adding a new Agent pool to the pipeline settings

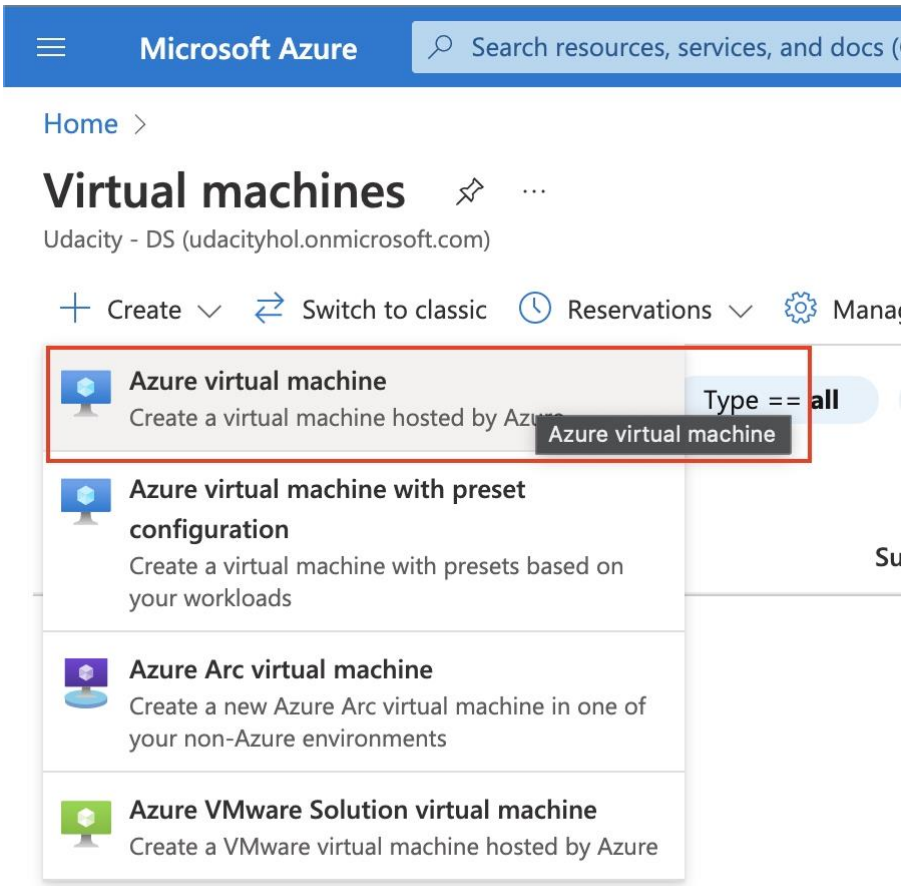
- Choose the agent pool as "Self-hosted". Provide the Agent pool a name and grant access permissions to all pipelines.



Creating self-hosted agent pool

Task 3. Create a Linux VM

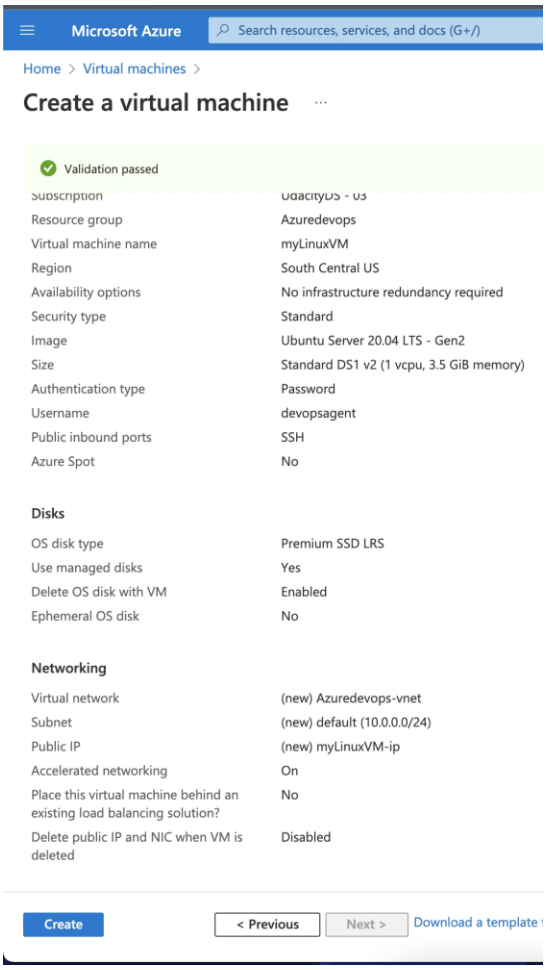
Navigate to the "Virtual machines" service in the Azure Portal, and then select "+ Create" to create a VM



- Use the following values in the Create a virtual machine wizard to create a VM:

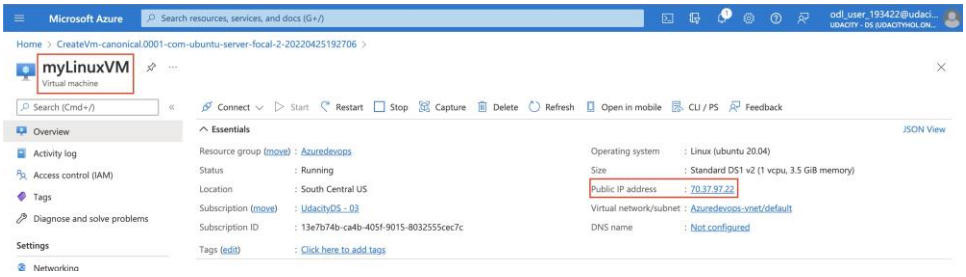
Field	Value
Subscription	Choose existing
Resource group	Choose existing, say **Azuredevops**
Virtual machine name	myLinuxVM
Availability options	No infrastructure redundency required
Region	Select the region same that of the resource group
Image	Ubuntu Server 20.04 LTS - Gen1
Size	Standard_D1s_v2
Authentication type	Password
Username	devopsagent
Password	DevOpsAgent@123
Public inbound ports	Allow selected ports Select inbound ports: SSH (22)

Leave th remaining fields as default. Review and create the VM. It will take a few minutes for the deployment.



Task 4. Configure the Linux VM as an Azure DevOps Build Agent

- Copy the Public IP address from the overview section of the virtual machine, say 70.37.97.22.

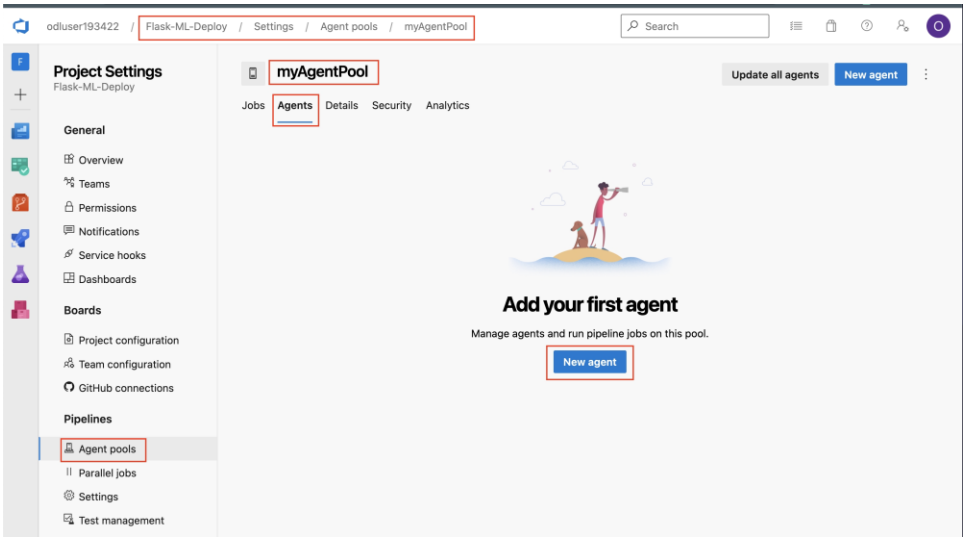


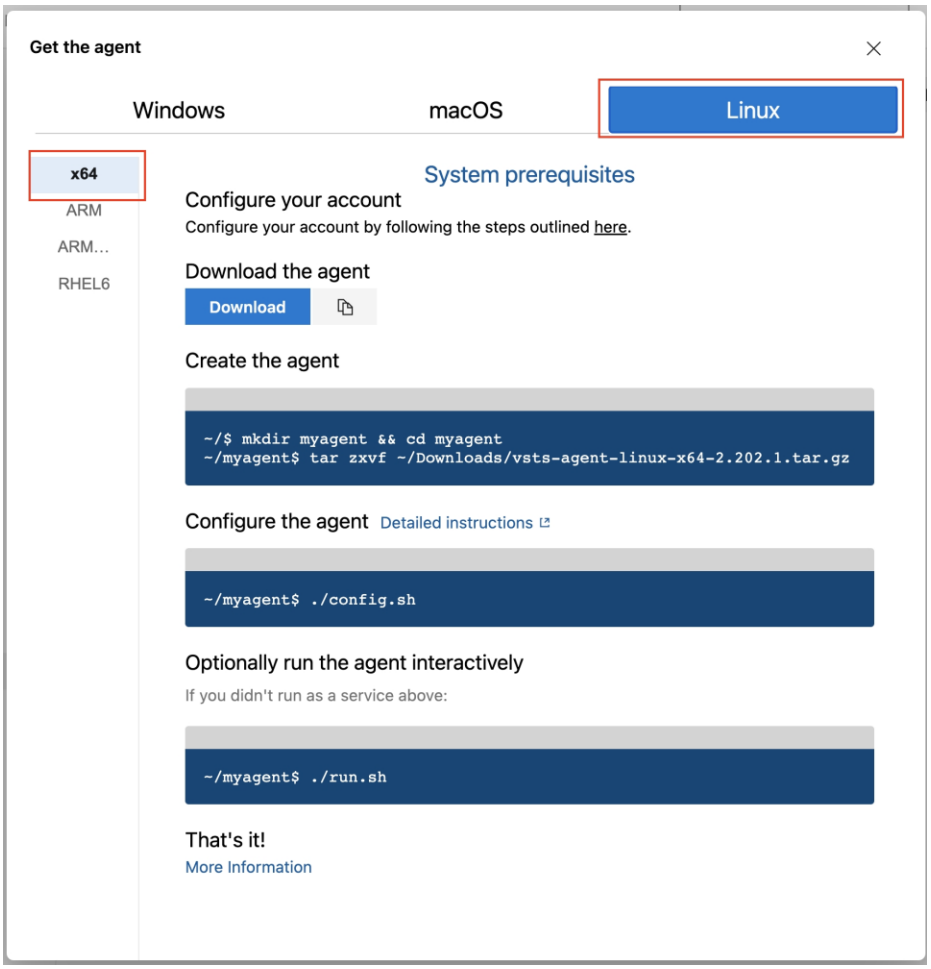
- Run the following commands from an Azure cloud shell or terminal or command prompt.*# Replace the IP address as applicable to you*
  ssh devopsagent@70.37.97.22
  Accept the default prompts and provide the username and password as you have set up in the last step above.

- After you SSH into the VM, install Docker as:
- sudo snap install docker
  # Check Python version because this agent will build your code
  python3 --version

- Configure the devopsagent user to run Docker as:
- sudo groupadd docker
  sudo usermod -aG docker $USER
  exit

- Restart the Linux VM from Azure portal to apply changes made in previous steps. Restarting the VM will log you out from the SSH log in. You will have to log back in using the same SSH command. Do note the new public IP, if it has been changed after the VM restart.
- Go back to the DevOps portal, and open the newly created Agent pool to add a new agent. The snapshot below will help you understand better.





- Copy the commands to download, create and configure the Linux x64 agent. The commands will be similar to the following:*# Download the agent*
  curl -O https://vstsagentpackage.azureedge.net/agent/2.202.1/vsts-agent-linux-x64-2.202.1.tar.gz
  # Create the agent
  mkdir myagent && cd myagent
  tar zxvf ../vsts-agent-linux-x64-2.202.1.tar.gz
  # Configure the agent
  ./config.sh

```
devopsagent@myLinuxVM:~/myagent$ ./config.sh

  ___                      ______ _            _ _
 / _ \                     | ___ (_)          | (_)
/ /_\ \_____   _ _ __ ___  | |_/ /_ _ __   ___| |_ _ __   ___  ___
|  _  |_  / | | | '__/ _ \ |  __/| | '_ \ / _ \ | | '_ \ / _ \/ __|
| | | |/ /| |_| | | |  __/ | |   | | |_) |  __/ | | | | |  __/\__ \
\_| |_/___|\__,_|_|  \___| \_|   |_| .__/ \___|_|_|_| |_|\___||___/
                                   | |
        agent v2.202.1             |_|          (commit a12c15b)


>> End User License Agreements:

Building sources from a TFVC repository requires accepting the Team Explorer Everywhere End User License
 Agreement. This step is not required for building sources from Git repositories.

A copy of the Team Explorer Everywhere license agreement can be found at:
  /home/devopsagent/myagent/license.html

Enter (Y/N) Accept the Team Explorer Everywhere license agreement now? (press enter for N) > Y

>> Connect:

Enter server URL > https://dev.azure.com/odluser193422
Enter authentication type (press enter for PAT) >
Enter personal access token > ****************************************************
Connecting to server ...

>> Register Agent:

Enter agent pool (press enter for default) > myAgentPool
Enter agent name (press enter for myLinuxVM) >
Scanning for tool capabilities.
Connecting to the server.
Successfully added the agent
Testing agent connection.
Enter work folder (press enter for _work) >
2022-04-25 14:37:47Z: Settings Saved.
devopsagent@myLinuxVM:~/myagent$
```

- The configuration will ask for the following prompts:

Prompt	Response
Accept the license agreement	Y
Server URL	Provide your Azure DevOps organization URL For example, https://dev.azure.com/organization-name or https://dev.azure.com/odluser193422
Authentication type	[Press enter]
Personal access token	[Provide the PAT saved above]
Agent pool (enter the value)	Choose the one created above, say **myAgentPool**
Agent name	[Press enter]
Work folder	[Press enter]

- Run the following commands to finish the set up

```
sudo ./svc.sh install
sudo ./svc.sh start
```

```
2022-04-25 14:37:47Z: Settings Saved.
devopsagent@myLinuxVM:~/myagent$ sudo ./svc.sh install
Creating launch agent in /etc/systemd/system/vsts.agent.odluser193422.myAgentPool.myLinuxVM.service
Run as user: devopsagent
Run as uid: 1000
gid: 1000
Created symlink /etc/systemd/system/multi-user.target.wants/vsts.agent.odluser193422.myAgentPool.myLinux
VM.service → /etc/systemd/system/vsts.agent.odluser193422.myAgentPool.myLinuxVM.service.
devopsagent@myLinuxVM:~/myagent$ sudo ./svc.sh start

/etc/systemd/system/vsts.agent.odluser193422.myAgentPool.myLinuxVM.service
● vsts.agent.odluser193422.myAgentPool.myLinuxVM.service - Azure Pipelines Agent (odluser193422.myAgentP
ool.myLinuxVM)
     Loaded: loaded (/etc/systemd/system/vsts.agent.odluser193422.myAgentPool.myLinuxVM.service; enabled
; vendor preset: enabled)
     Active: active (running) since Mon 2022-04-25 14:40:48 UTC; 19ms ago
   Main PID: 1563 (runsvc.sh)
      Tasks: 3 (limit: 4100)
     Memory: 856.0K
     CGroup: /system.slice/vsts.agent.odluser193422.myAgentPool.myLinuxVM.service
             ├─1563 /bin/bash /home/devopsagent/myagent/runsvc.sh
             └─1565 ./externals/node10/bin/node ./bin/AgentService.js

Apr 25 14:40:48 myLinuxVM systemd[1]: Started Azure Pipelines Agent (odluser193422.myAgentPool.my…uxVM).
Apr 25 14:40:48 myLinuxVM runsvc.sh[1563]: .path=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bi…ap/bin
Hint: Some lines were ellipsized, use -l to show in full.
devopsagent@myLinuxVM:~/myagent$
```

We have to install some additional packages to enable our agent build the Flask application code. These commands are specific to our sample Flask application, you can extend them per your application requirements:

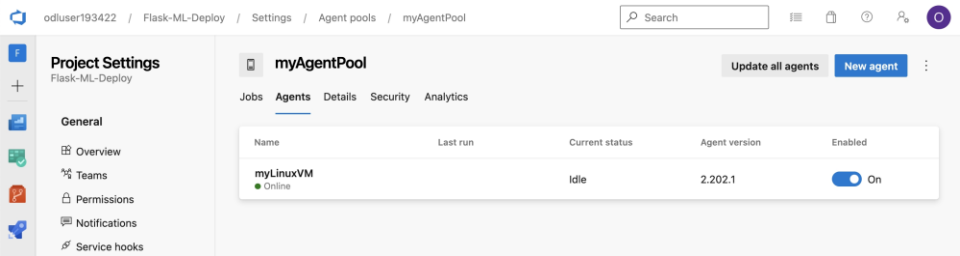
```
sudo apt-get update
sudo apt update
sudo apt install software-properties-common
sudo add-apt-repository ppa:deadsnakes/ppa
sudo apt install python3.7
sudo apt-get install python3.7-venv
sudo apt-get install python3-pip
python3.7 —version
pip —version
sudo apt-get install python3.7-distutils
sudo apt-get -y install zip
```

In addition, pylint is know to need an additional step, as mentioned in this stackoverflow thread:

```
# Shows no output because the Path is not set explicitly
which pylint
pip show --files pylint
# Shows Files:
#  ../../../bin/epylint
# ../../../bin/pylint
echo $PATH
export PATH=$HOME/.local/bin:$PATH
```

```
echo $PATH
which pylint
```

In you Azure DevOps, navigate to Organization Settings >> Agent Pools >> myAgentPool and then select the Agents tab. Confirm that the self-hosted agent is online.



Successfully added the self-hosted agent to the agent pool

Set up the DevOps Pipeline

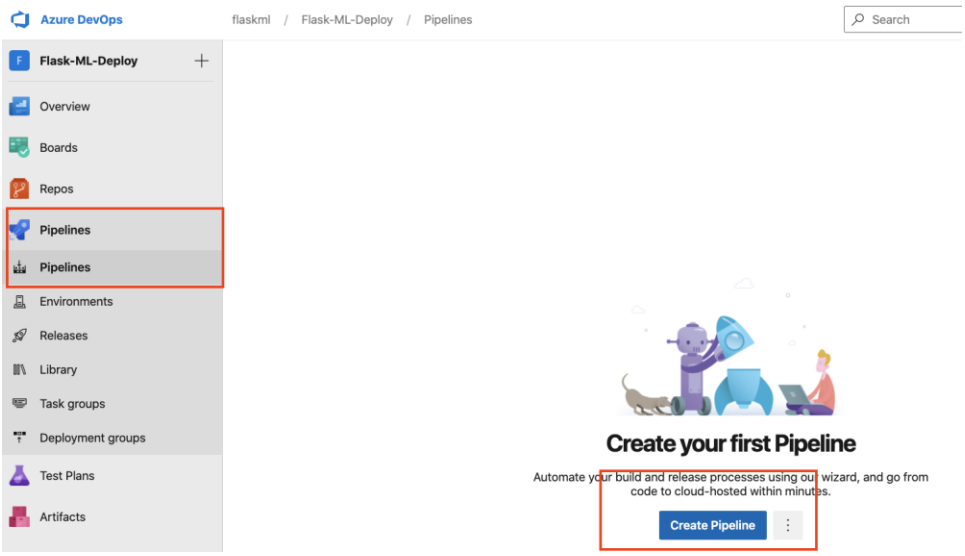
Next, we'll create a CI/CD pipeline. The screenshots below show the steps, but if you need to, you can also refer to the official documentation for more detail - [Use CI/CD to deploy a Python web app to Azure App Service on Linux](#).

1. Create a project

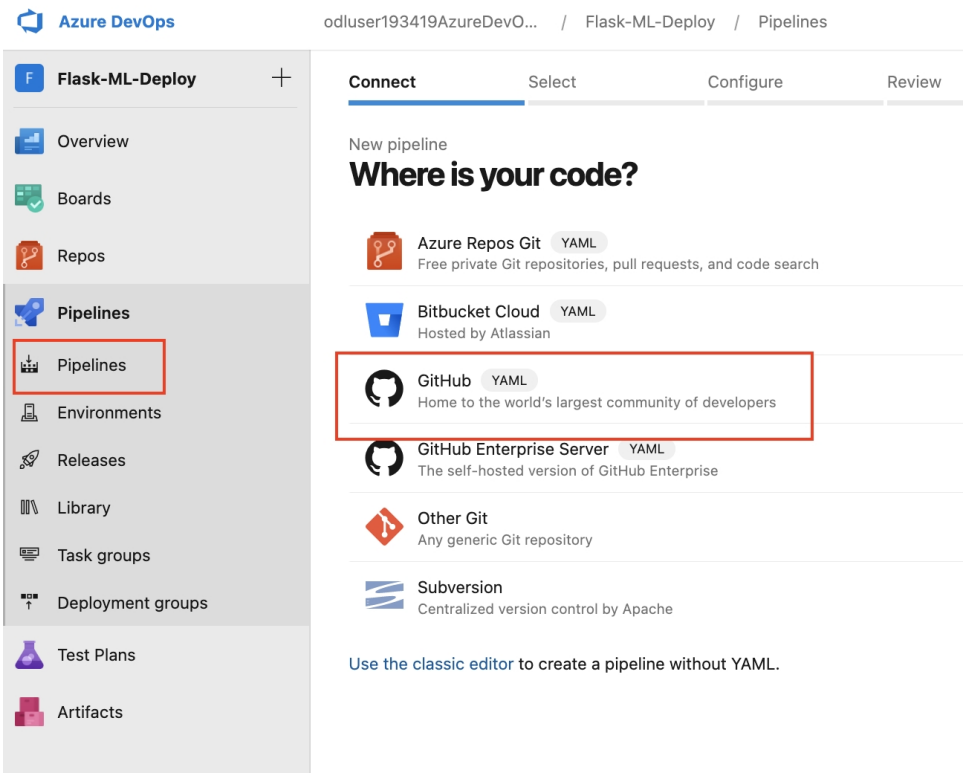
We are assuming that you already have a Azure DevOps org, and a publicly visible project **Flask-ML-Deploy** available.

2. Create a Pipeline

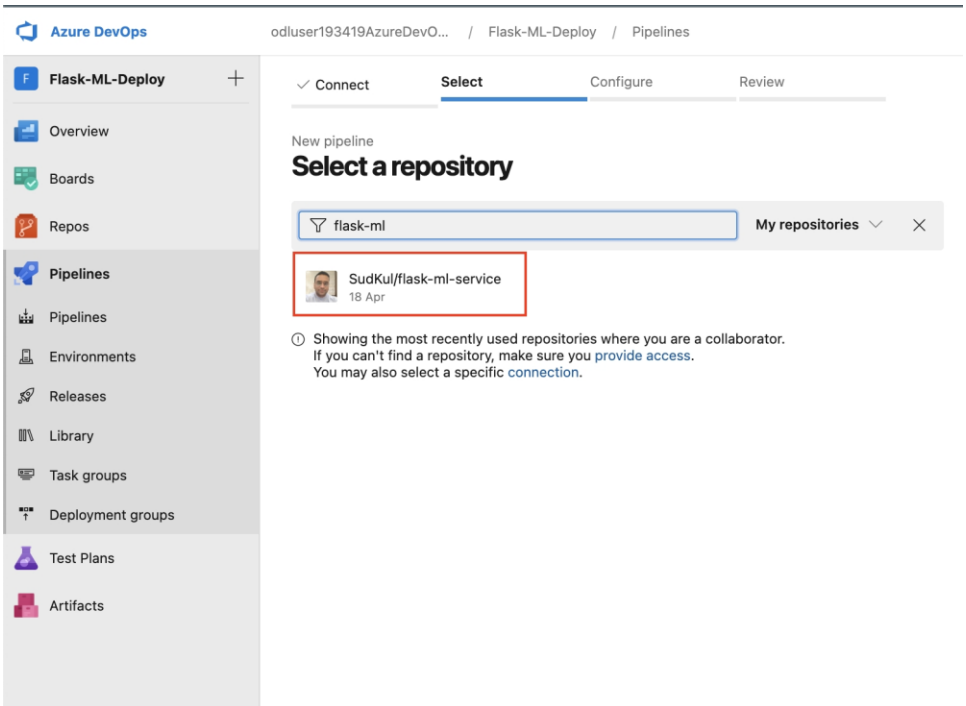
- Go back to the DevOps project, select **Pipeline** and create a new one.



- Choose the Github repository as the source code location.

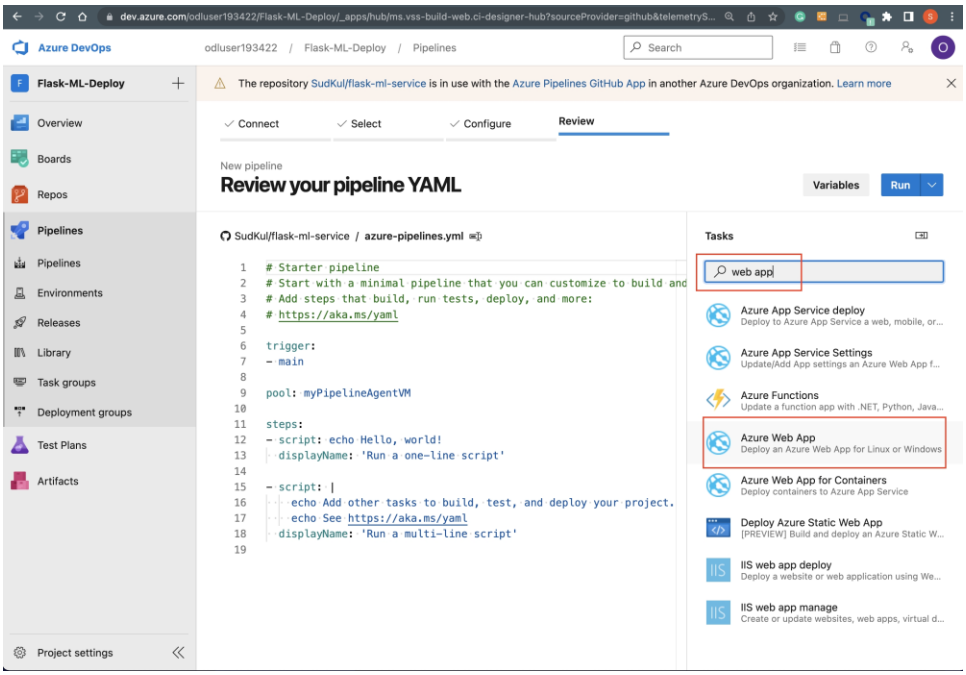


Connect the pipeline to Github

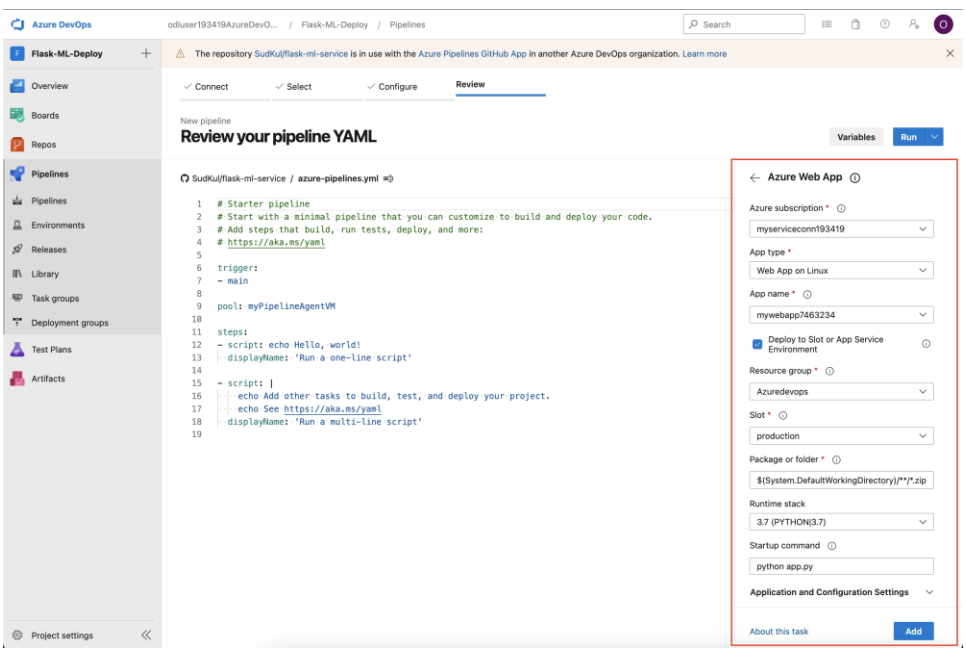


Select the repository

- Note that the starter code already has *requirements.txt* and an *app.py* files. Therefore, the Pipeline wizard will automatically identify the application environment and present you with a sample *azure-pipelines.yml*.
- The *azure-pipelines.yml* file defines the pipeline environment, stages, jobs, tasks, build agent, and other types of configuration used to build the code from the Github and deploy it to the Azure services. The Pipeline wizard provides an assistant (see the snapshot below) to configure some environment variables.



Using the assistant in the Pipeline wizard



3. Update *azure-pipelines.yml* file

- The *azure-pipelines.yml* file format is indentation sensitive and is easy to understand; for example, we will define two stages: Build stage and Deploy

Web App. Each stage will have its job(s) set, and each job can have a bunch of scripts/tasks. Writing the *azure-pipelines.yml* file is the core of setting up a CI/CD pipeline.

Update the pipeline using the sample code present in [azure-pipelines-for-self-hosted-agent.yml]() file. **For starter, feel free to delete any stages/jobs/script/tasks as you need to make your pipeline work**

***Do you know**: If your pipeline throws an error, you can try out individual script/tasks in the pipeline build agent (recall that we learned how to SSH log into the myLinuxVM) we created earlier.*

```yaml
trigger:
- master

pool: myAgentPool

variables:
  azureServiceConnectionId: 'Your_service_connection_name'
  webAppName: 'Your_existing_Web_app_name'
  environmentName: 'flask-ml-service'
  # Project root folder
  projectRoot: $(System.DefaultWorkingDirectory)

stages:
- stage: Build
  displayName: Build stage
  jobs:
  - job: BuildJob
    pool: myAgentPool
    steps:
    - script: |
        python3.7 -m pip install --upgrade pip
      workingDirectory: $(projectRoot)
      displayName: 'Your_choice'

    - task: ArchiveFiles@2
      displayName: 'Archive files'
      inputs:
        rootFolderOrFile: '$(projectRoot)'
        includeRootFolder: false
        archiveType: zip
        archiveFile: $(Build.ArtifactStagingDirectory)/$(Build.BuildId).zip
        replaceExistingArchive: true

    - upload: $(Build.ArtifactStagingDirectory)/$(Build.BuildId).zip
      displayName: 'Upload package'
      artifact: drop
```

Provide the name of your custom agent pool

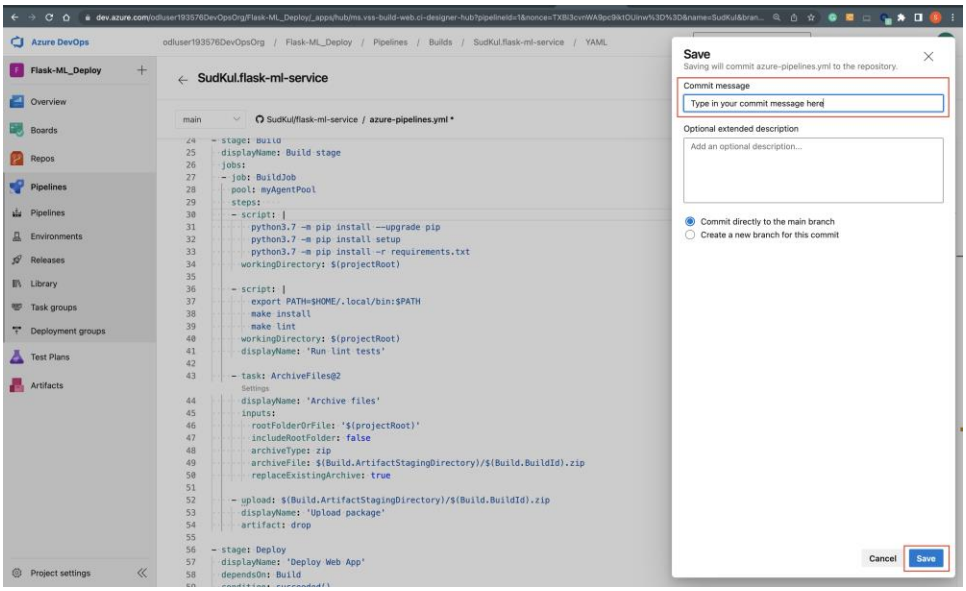
Environment variables

One job having three steps: a script, a task, and an upload task

You can provide multiple commands here

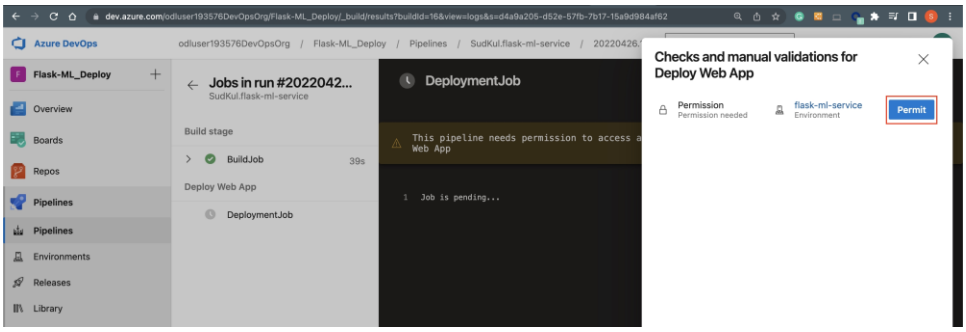
Standard task for archiving a project

Standard task for uploading the artifact

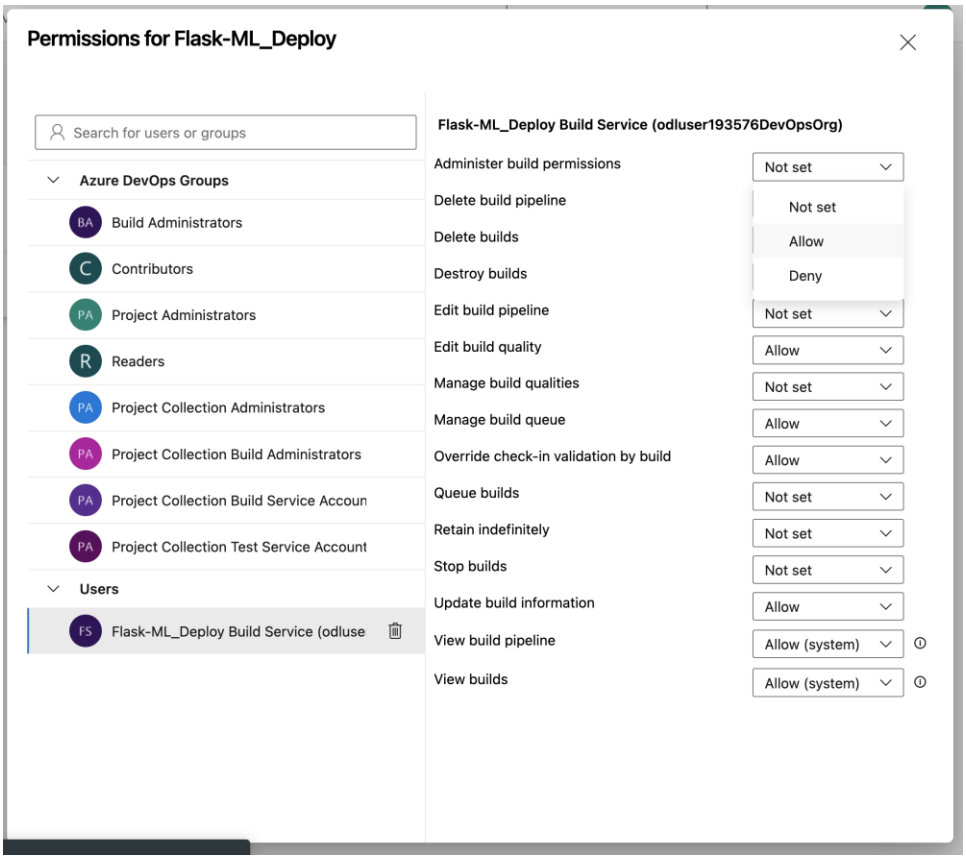


Save your changes, provide a commit message, and run the pipeline

- The Deploy Web App stage in our example will need your permission to deploy the build artifact to the Azure Web App. You can update the pipline permissions for the current user in the Pipelines >> More actions as [outlined here] and shown in the snapshots below.



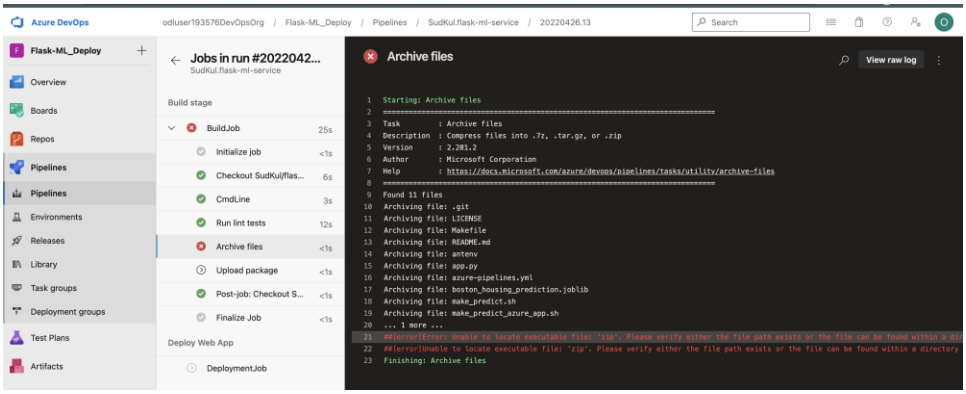
Update pipeline permissions at the project-level for the current user



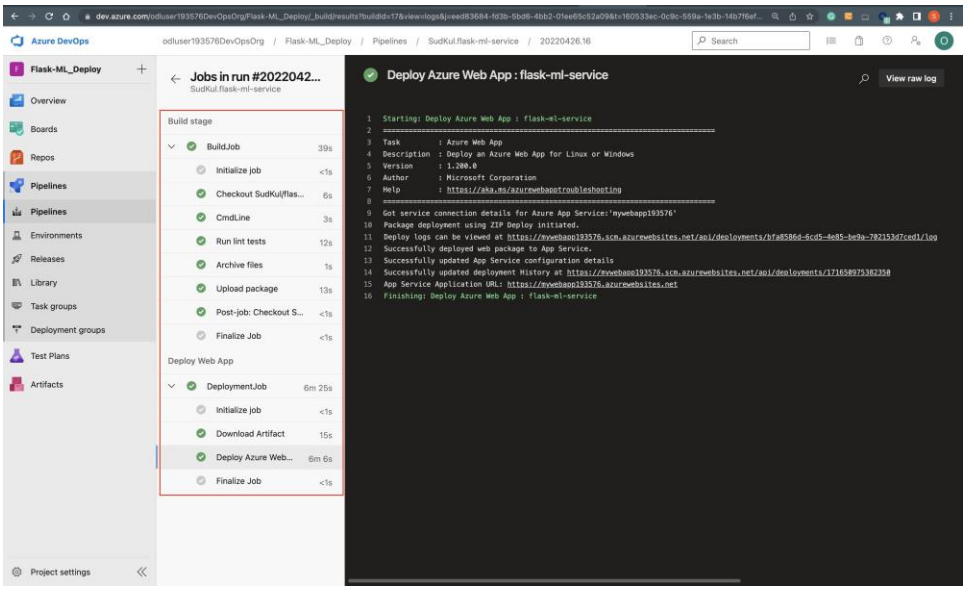
Allow all permissions for the current user

4. Troubleshoot

- In case of a pipeline error, we recommend you to look into the individual failed stages and read through the error message. The error message will help you understand the root cause, which could be a format error, dependency not present on the build agent, or the syntax error in the scripts/tasks.



Examining the failed task due to the Zip dependency not present on the build agent



Successful pipeline deployment

- Reference:
  - [Azure Pipeline YAML documentation](#).
  - [Command Line task](#)
  - [Archive Files task](#)
  - [Publish Pipeline Artifacts task](#)