# Use Case

We have provided the steps to deploy .net application to EKS. We have automated it using azure devOps.

# Azure DevOps

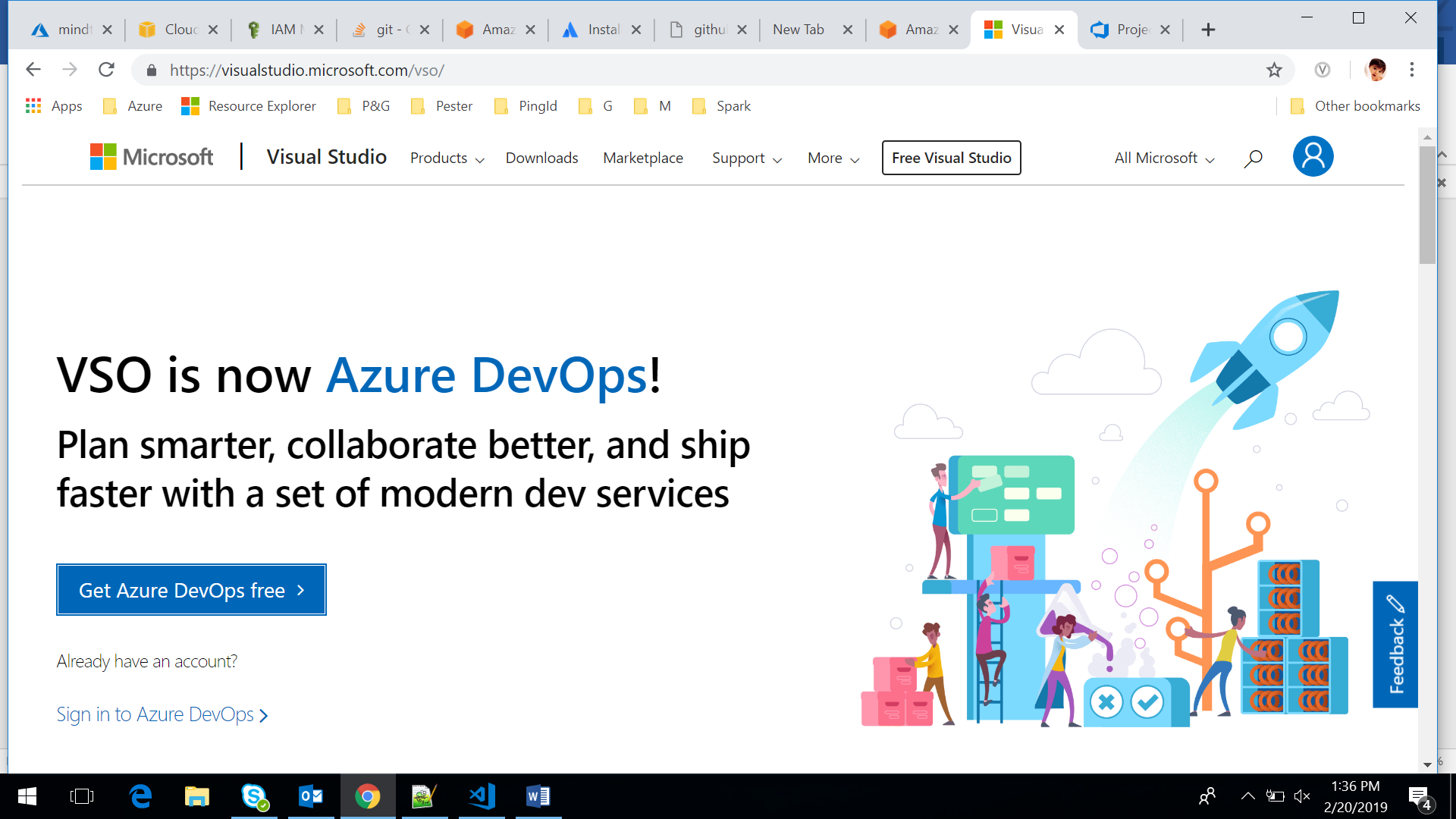
It sets up everything you need for developing, deploying and monitoring your application. DevOps Projects dashboard used to monitor code commits, builds, and deployments, all from a single view in the Azure portal

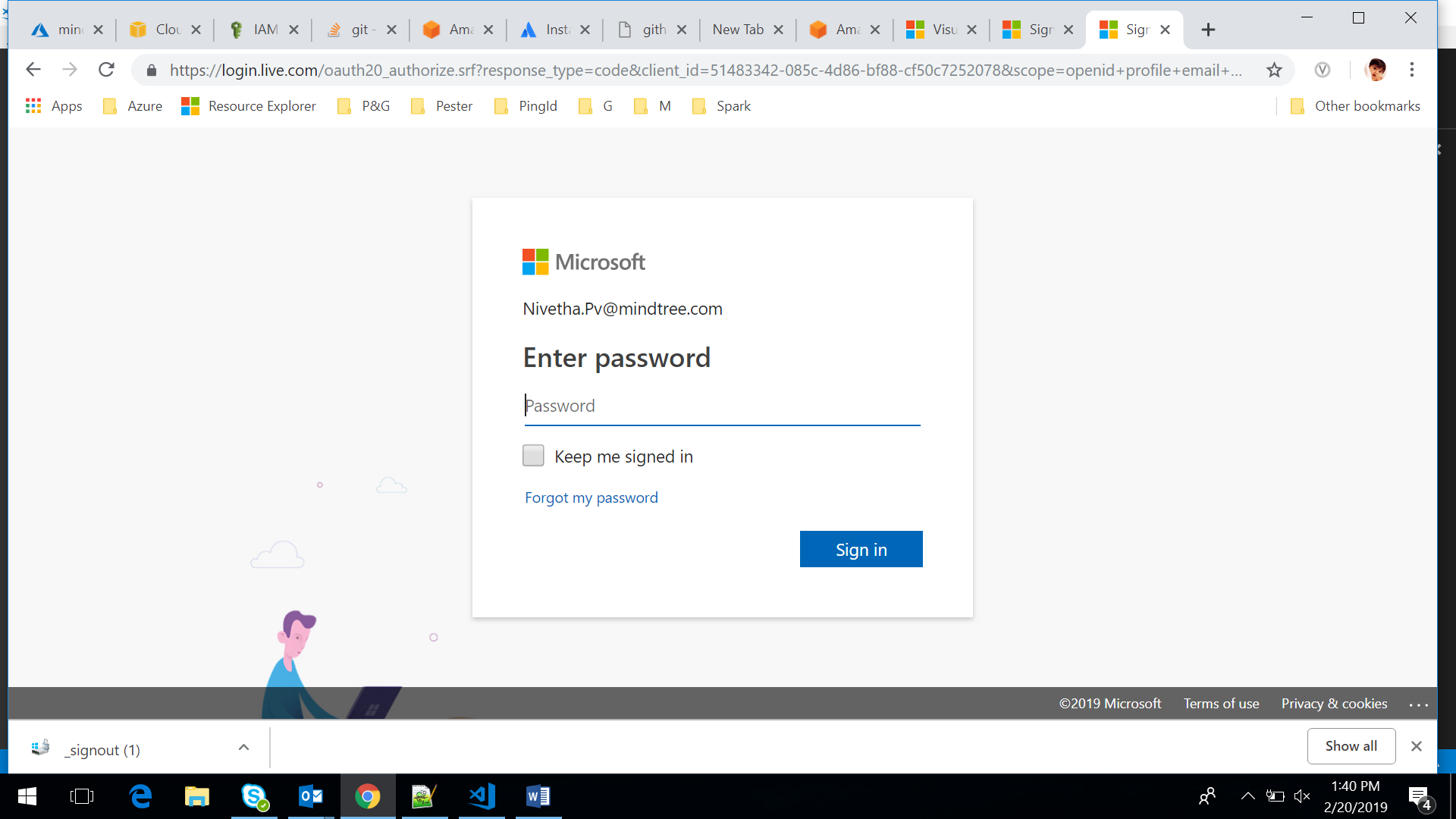
## Need for Azure DevOps

* Quickly deploy your application to various Azure services such as Virtual Machines, App Service, Azure Kubernetes Services (AKS), Azure SQL Database, Azure Service Fabric and etc.
* Configuration of a DevOps pipeline, from setting up the initial Git repository, configuring CI/CD pipeline, creating an application insights resource for monitoring, and providing a single view of the entire solution with the creation of a DevOps dashboard in the Azure portal.

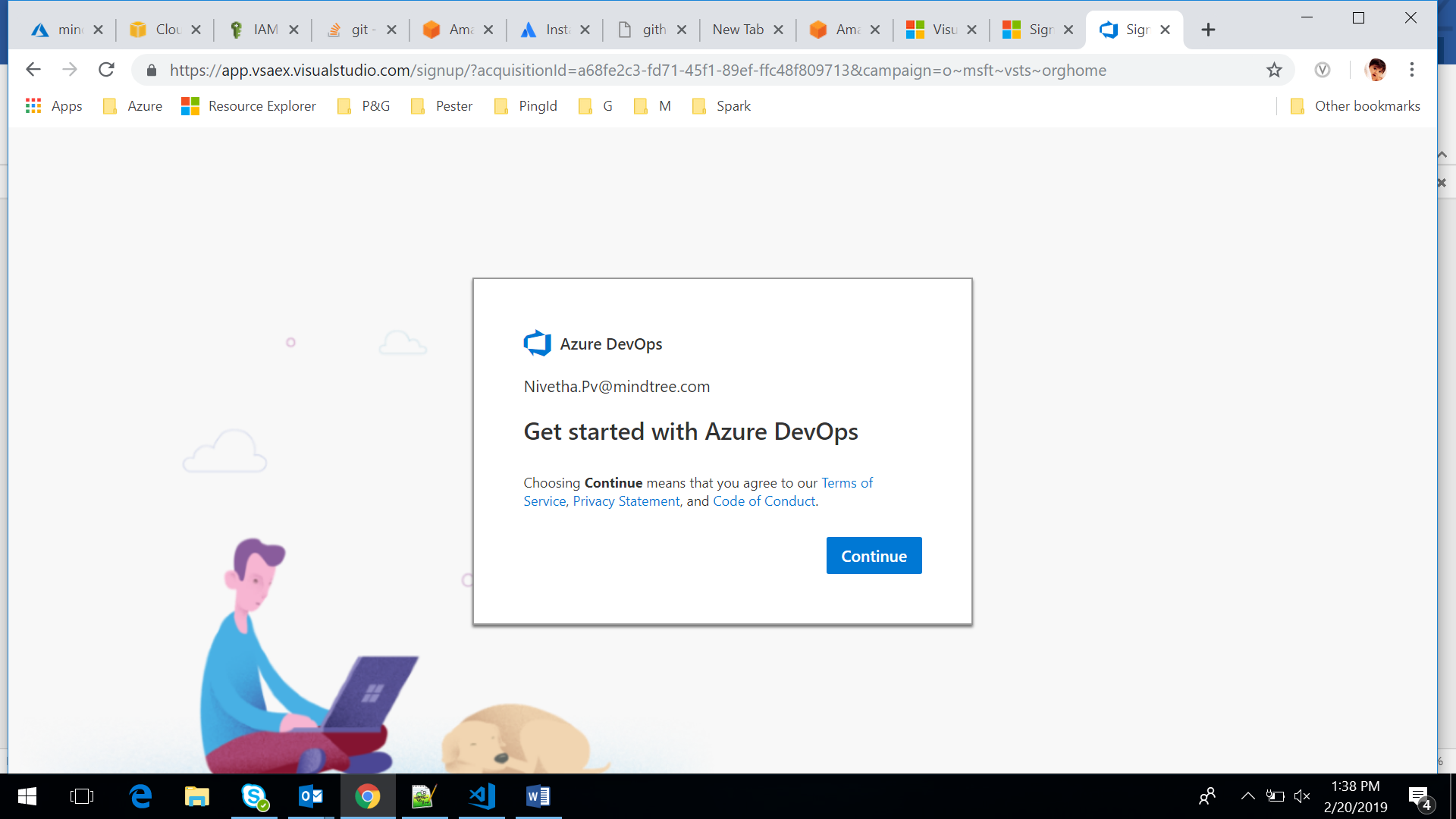
## Steps to be followed to set up CICD

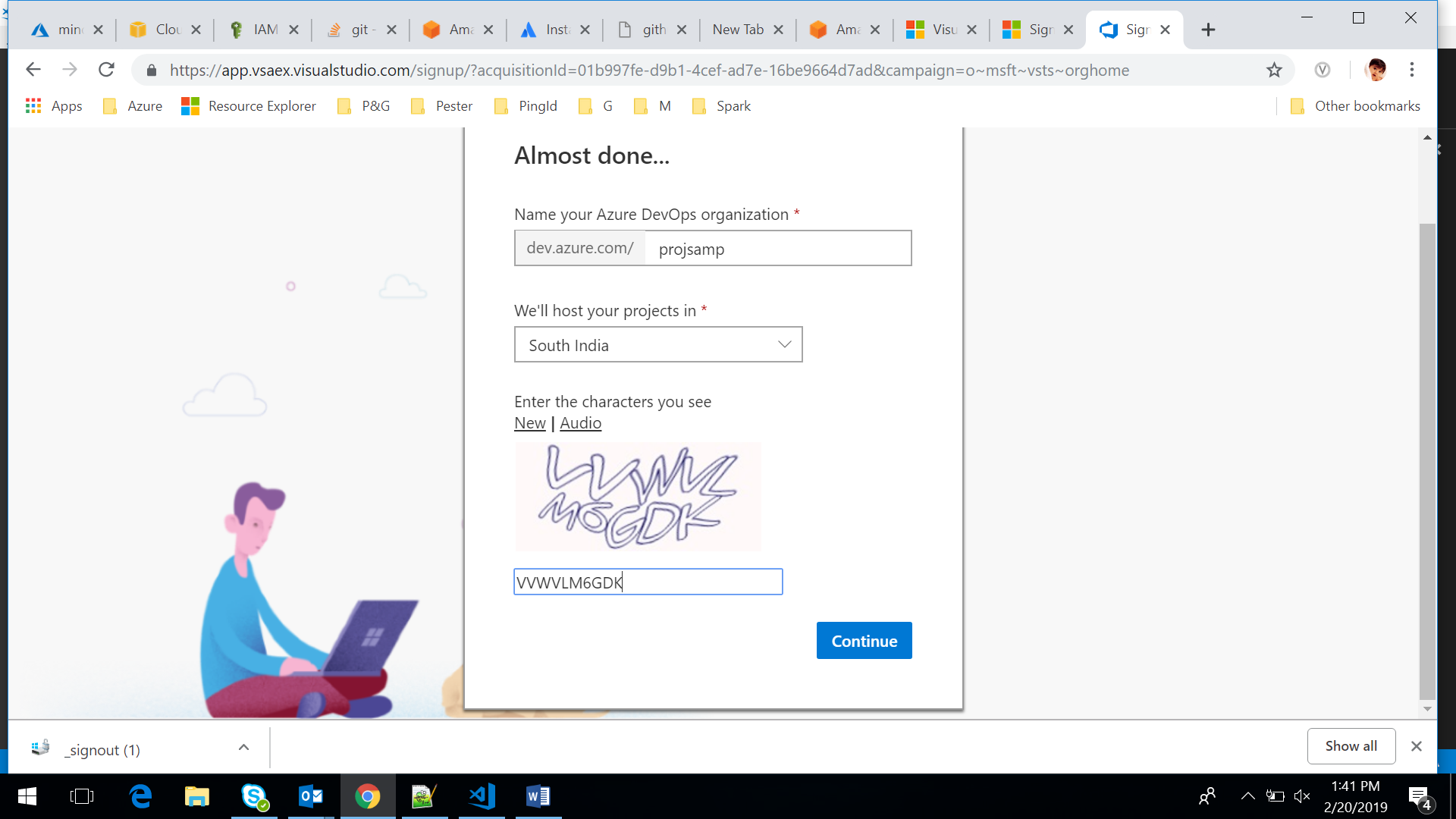
Step 1: Login to Azure DevOps <https://visualstudio.microsoft.com/vso/>



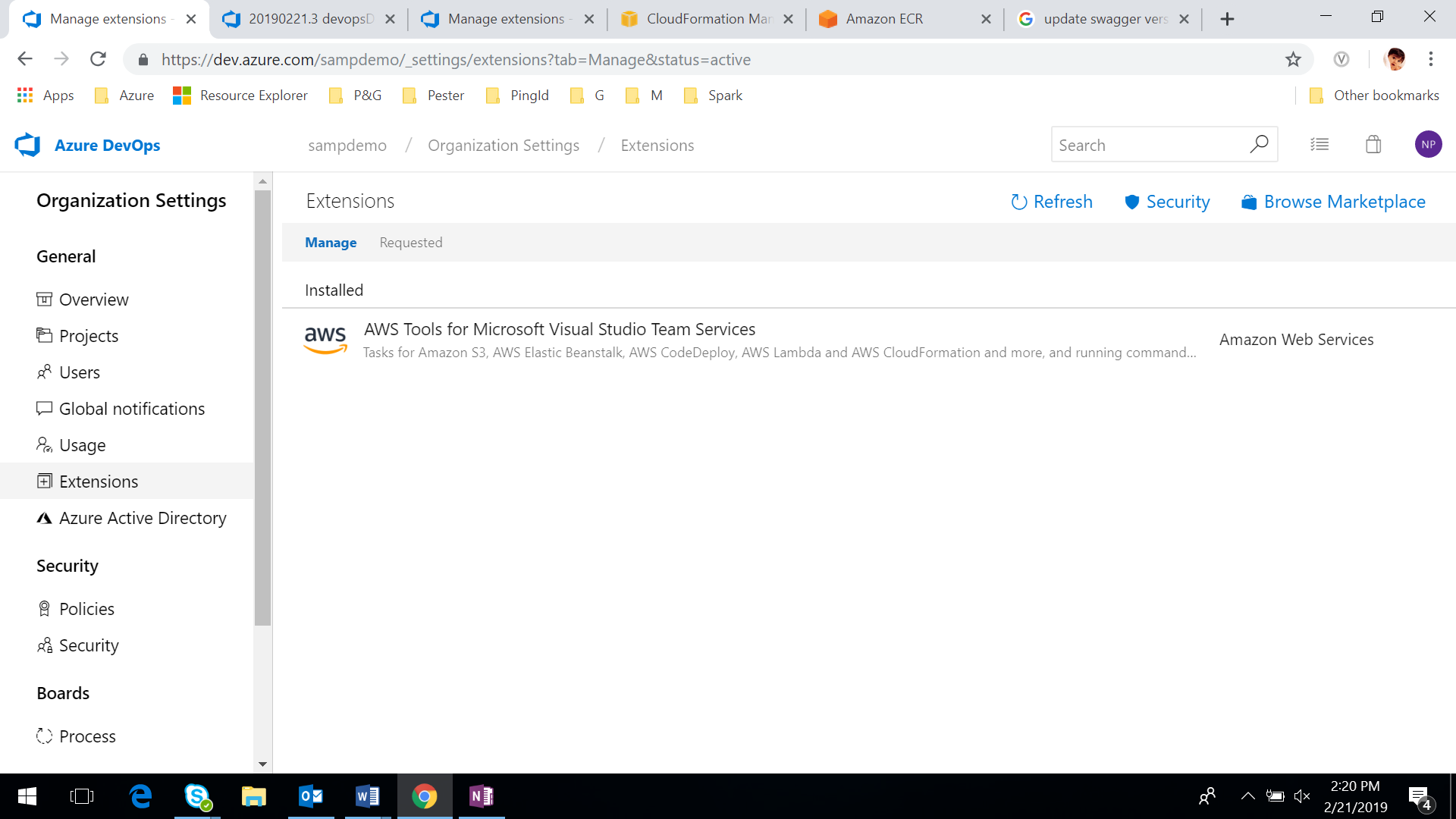
Step2: Sign in to your Azure DevOps <https://dev.azure.com>

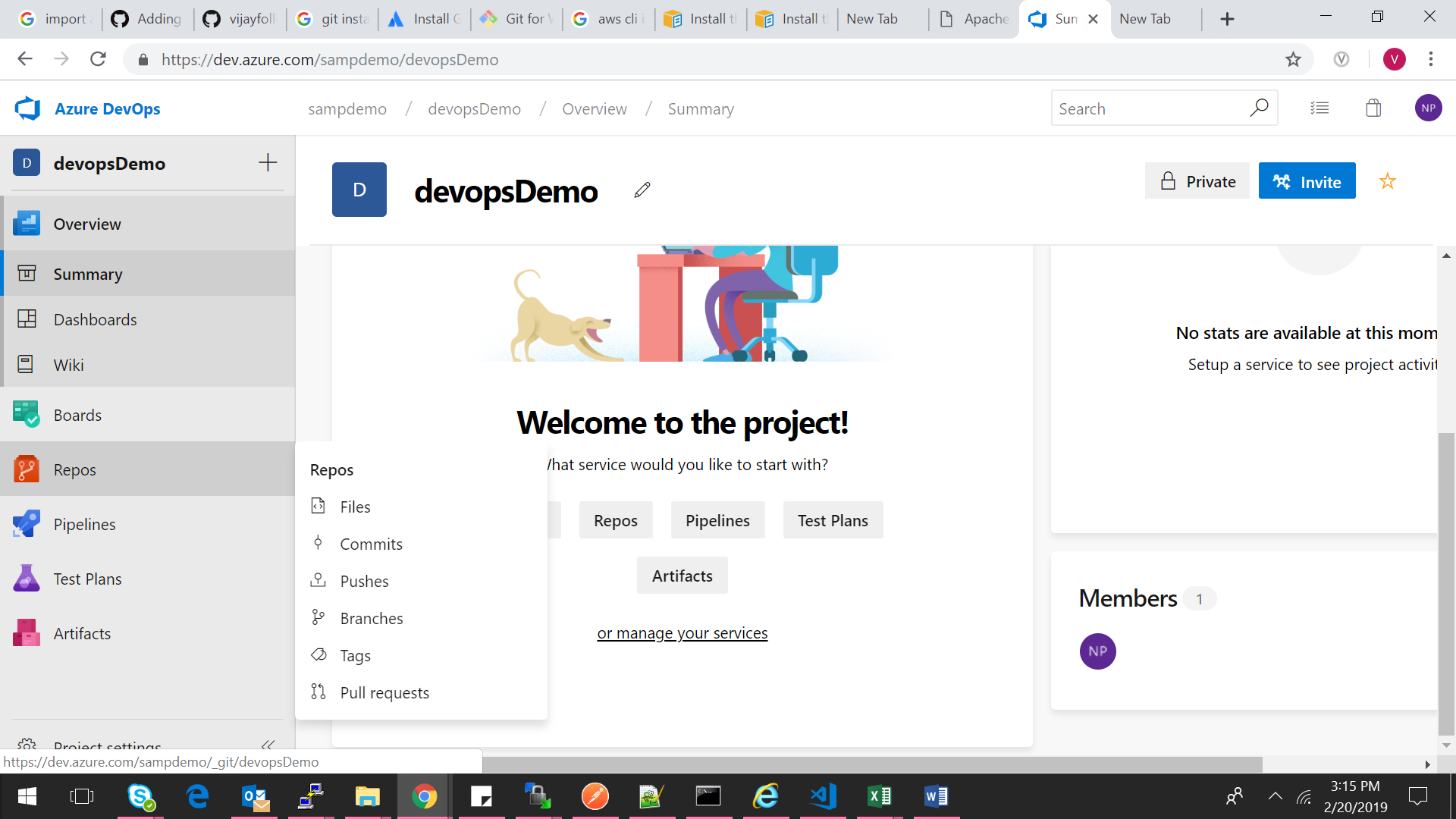
Step3: Create a new organizational account in it

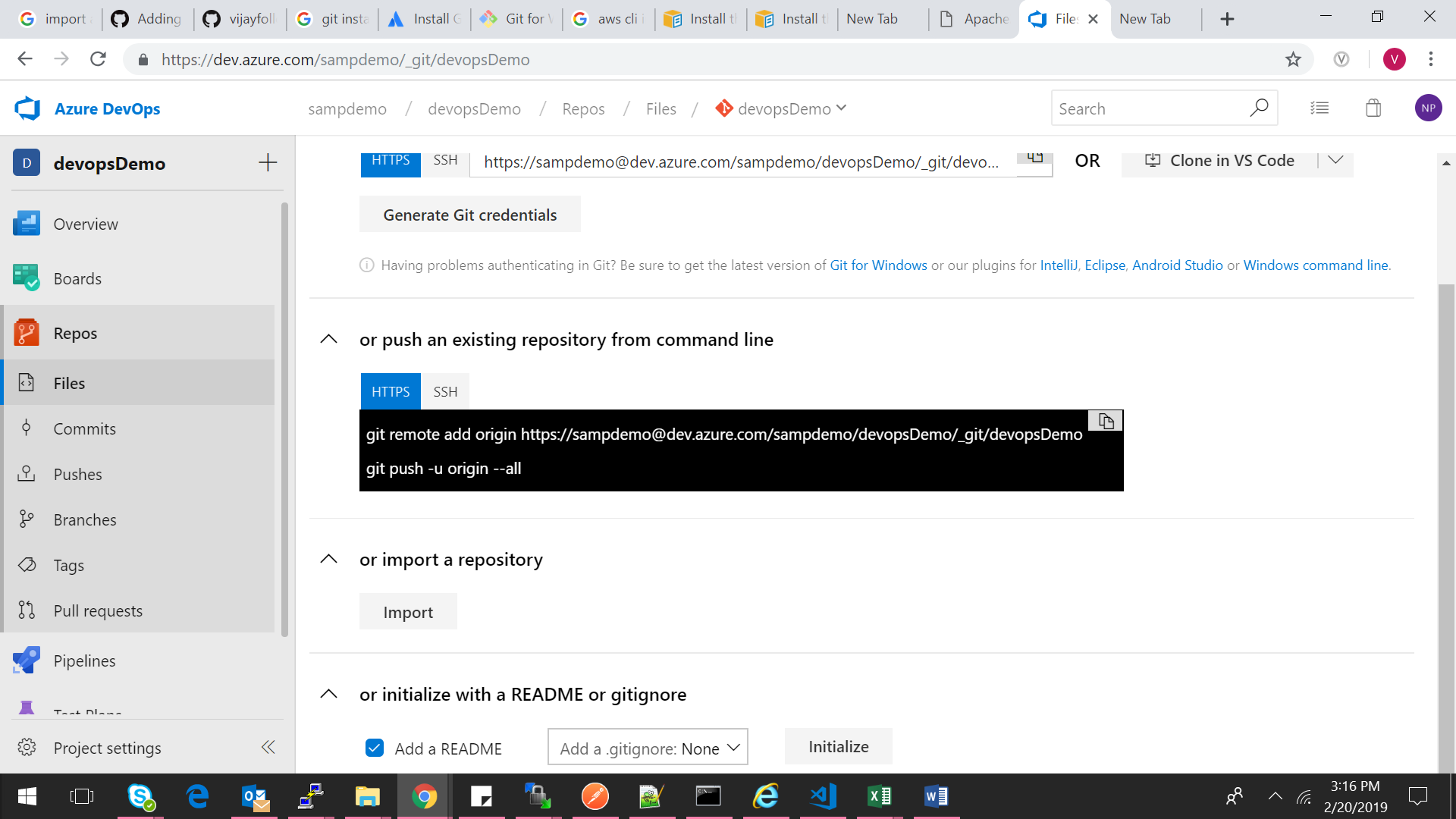


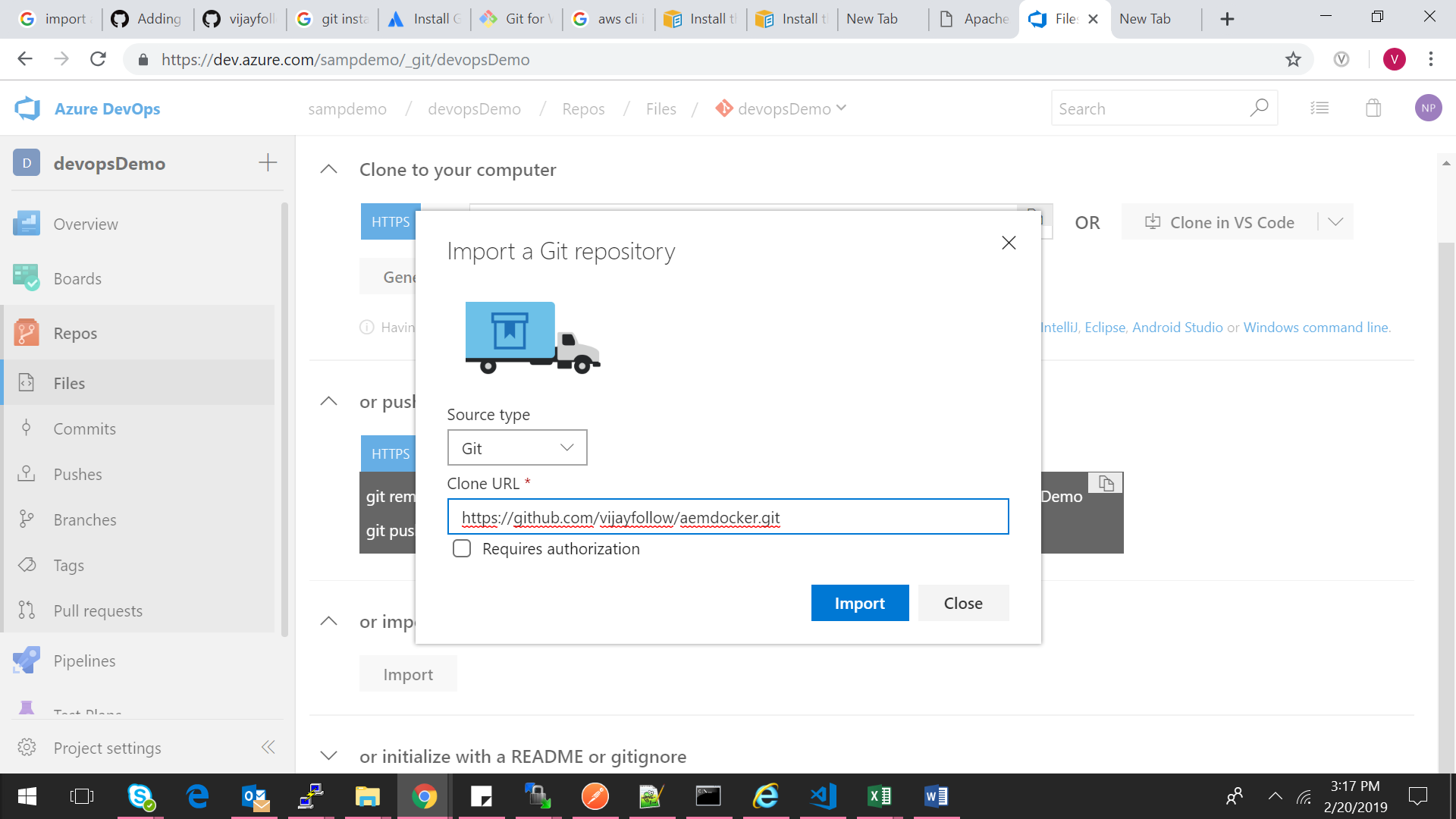


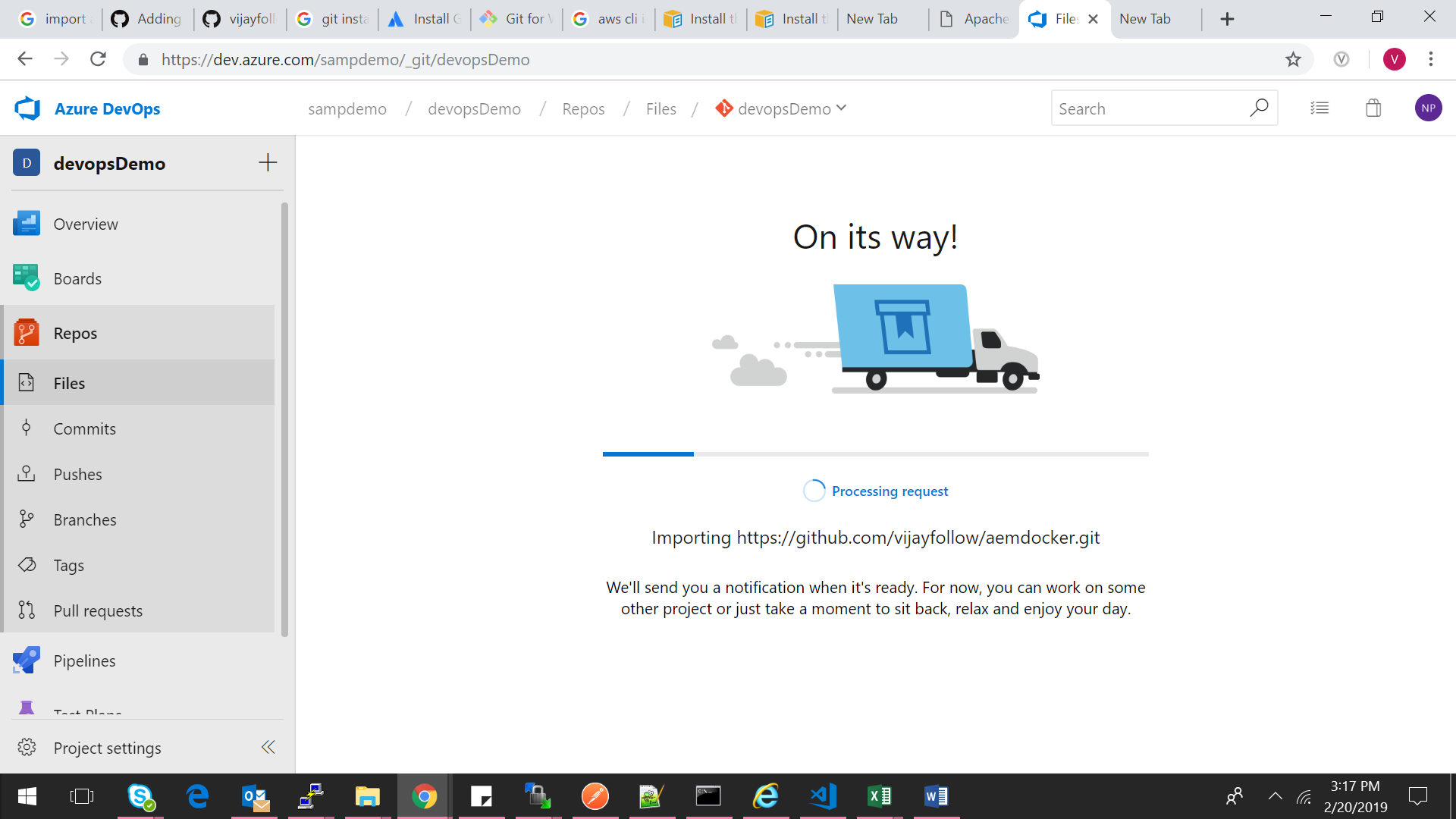
Step5: Install “AWS Tools” from marketplace as we need AWS services for our project “Organisation Settings -> Browse from Marketplace -> Choose for AWS tools”



Step 4: Create a project under your organization. 

Step 5: Inside Project, Choose Repos->click Import to import your git URL that has the source code

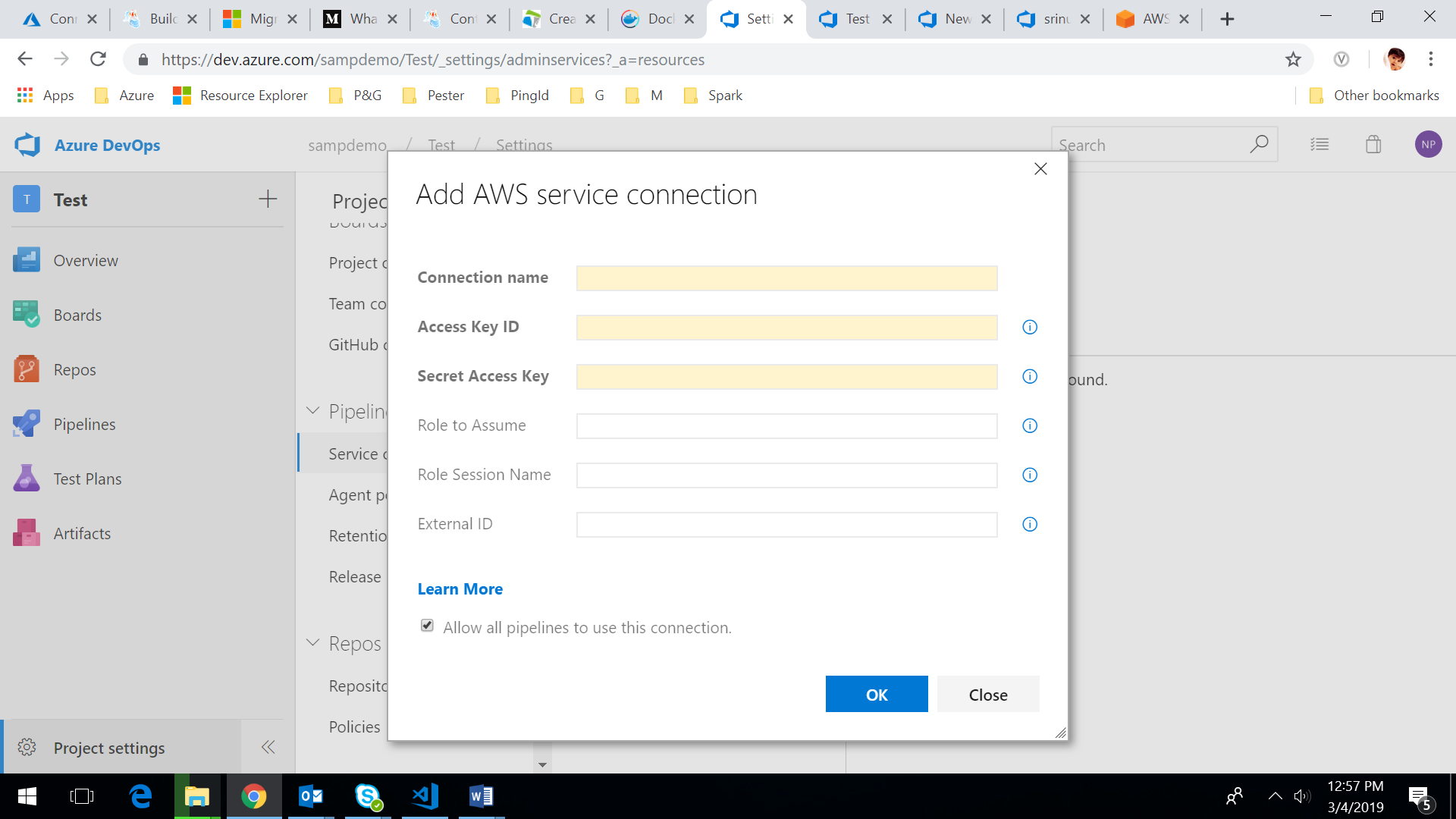




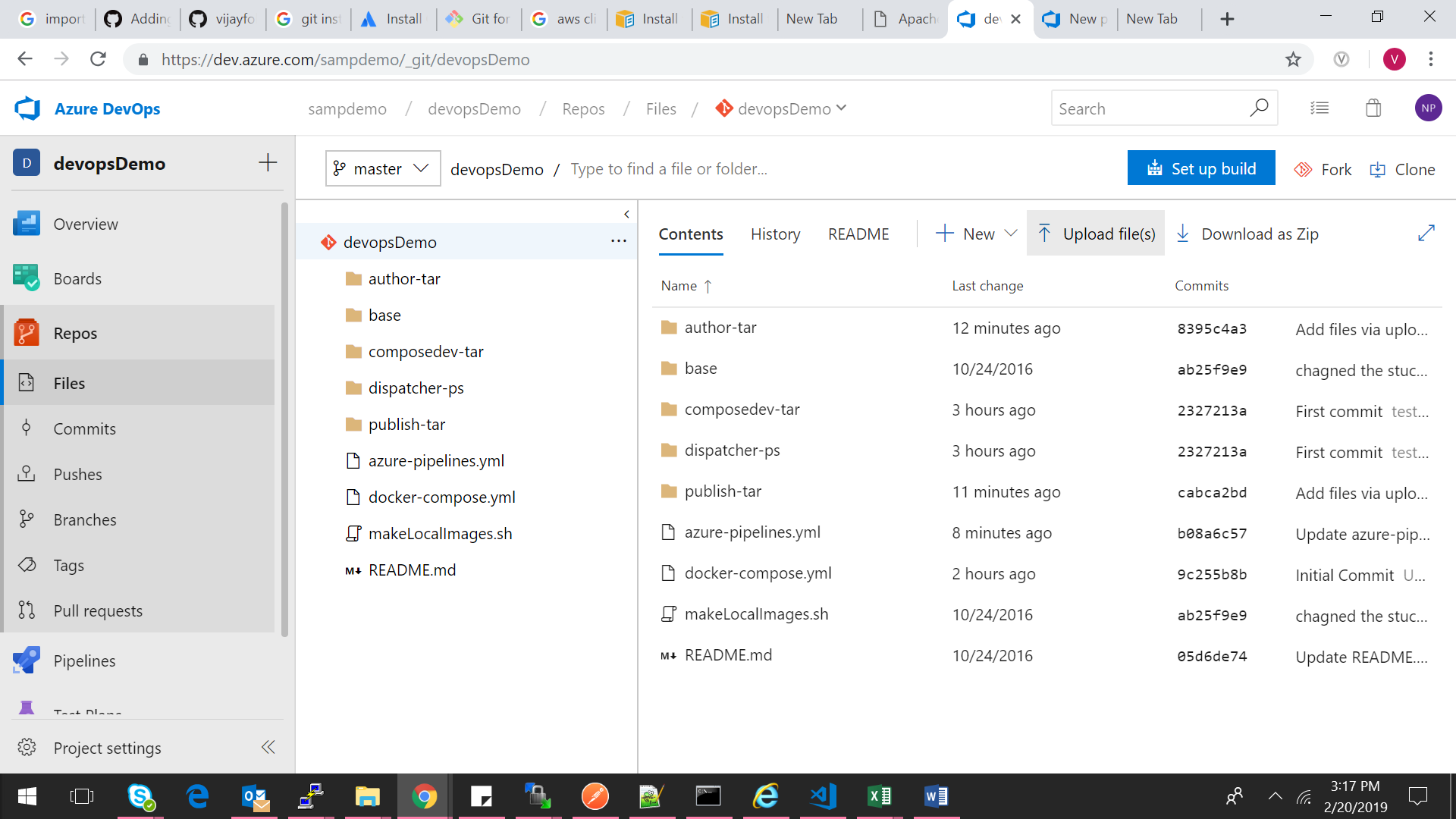
### **Build Pipeline [Continuous Integration]**

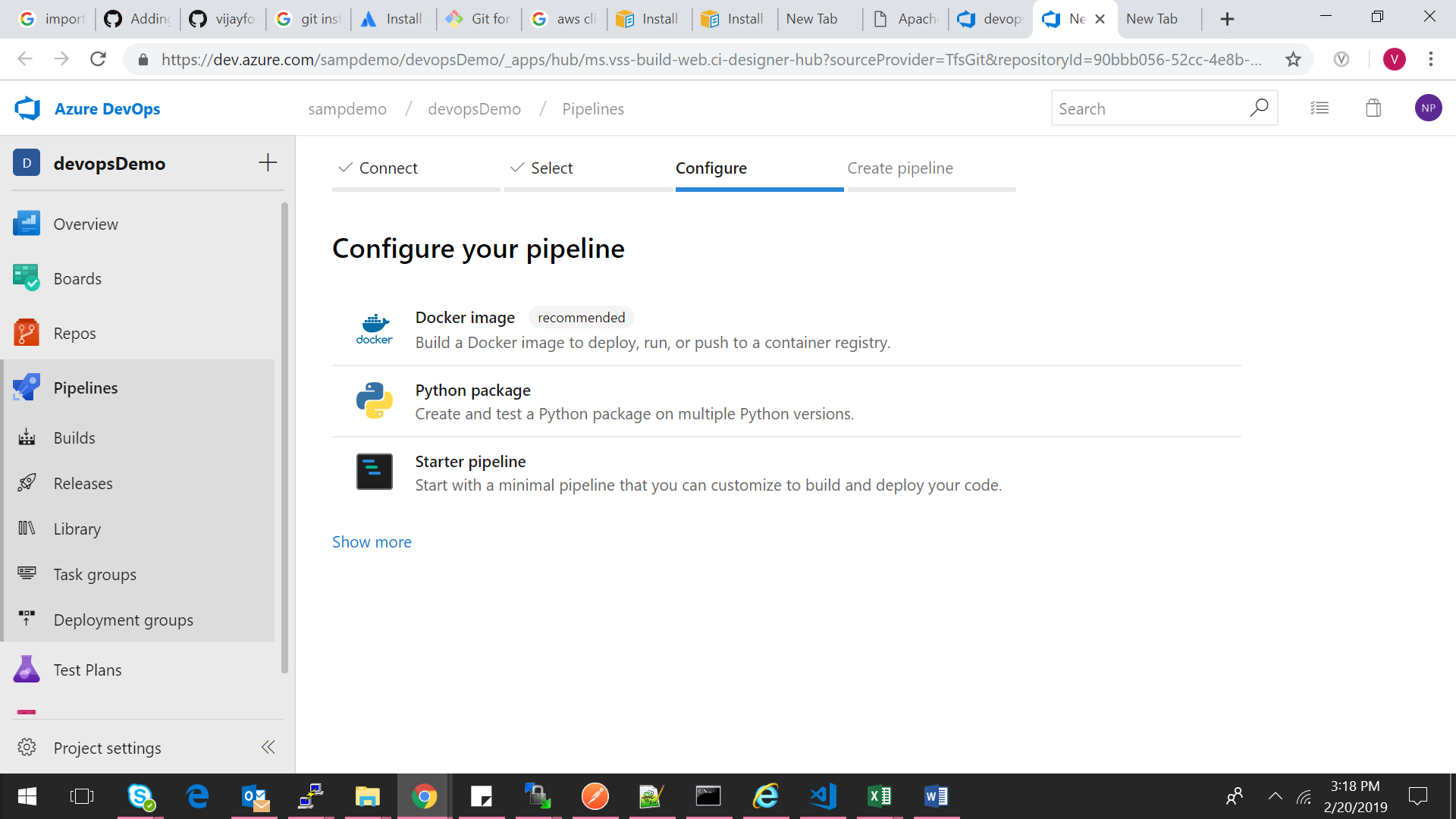
To build the docker image and push the image to ecr repository

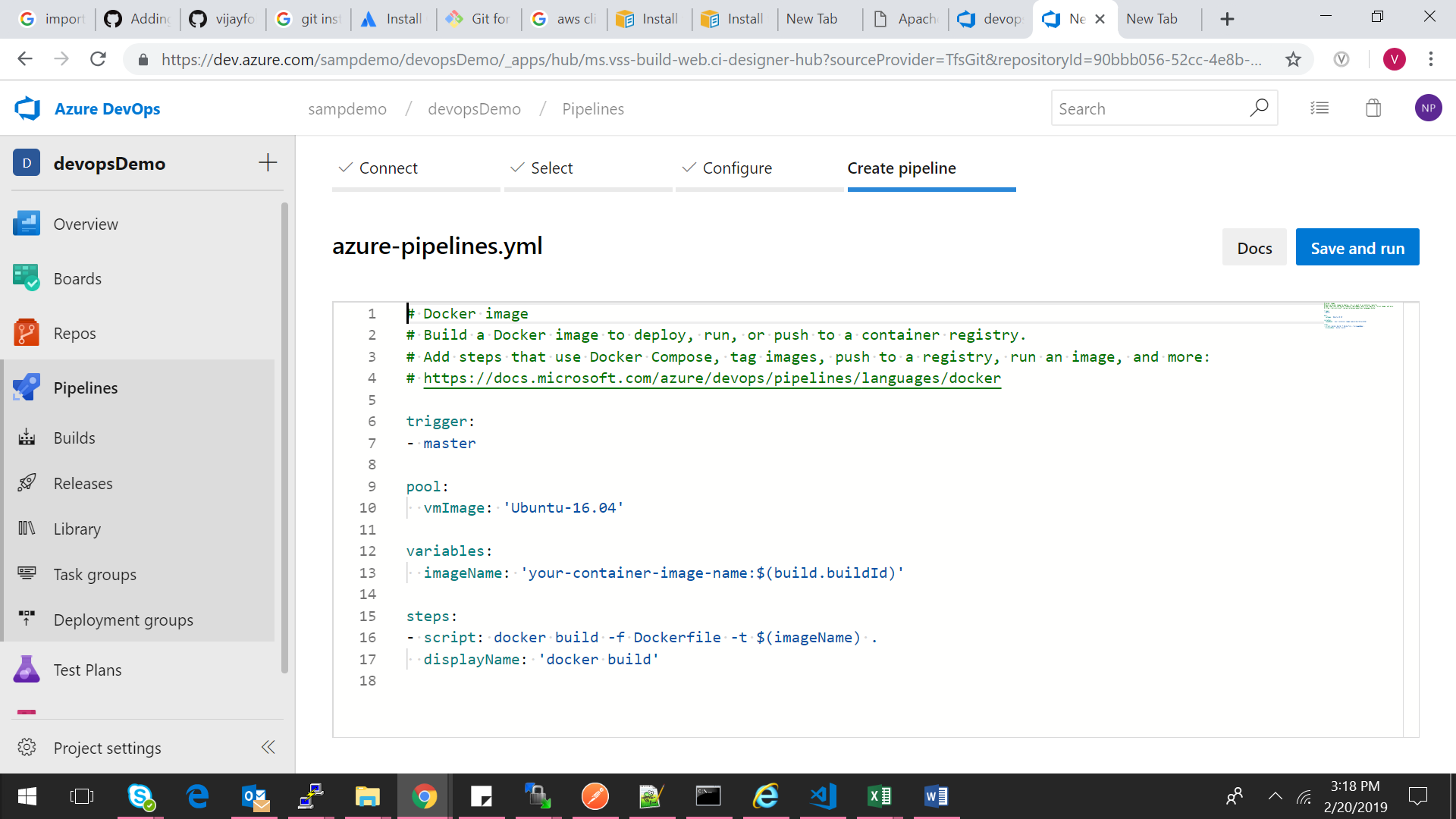
Step1: To be able to connect to AWS and push the Docker image to ECR, Service connection need to be added for AWS with IAM user details (Access key ID and Secret Access Key)



Step 2: Add the build step [click Set up build] and configure your pipeline [Choose Docker image in it]. It will have default code snippet to build a Docker images.







Step7: Modify the code snippet as follows.

It builds the Docker image and push it to ECR

# Docker image

# Build a Docker image to deploy, run, or push to a container registry.

# Add steps that use Docker Compose, tag images, push to a registry, run an image, and more:

# https://docs.microsoft.com/azure/devops/pipelines/languages/docker

trigger:

- master

pool:

vmImage: 'Ubuntu-16.04'

variables:

imageName: '$(build.buildId)'

steps:

- script: docker build -f webapptest/DockerFile -t $(imageName) .

displayName: 'docker build'

- script: 'docker images'

- task: AmazonWebServices.aws-vsts-tools.ECRPushImage.ECRPushImage@1

displayName: 'Push Image: '

inputs:

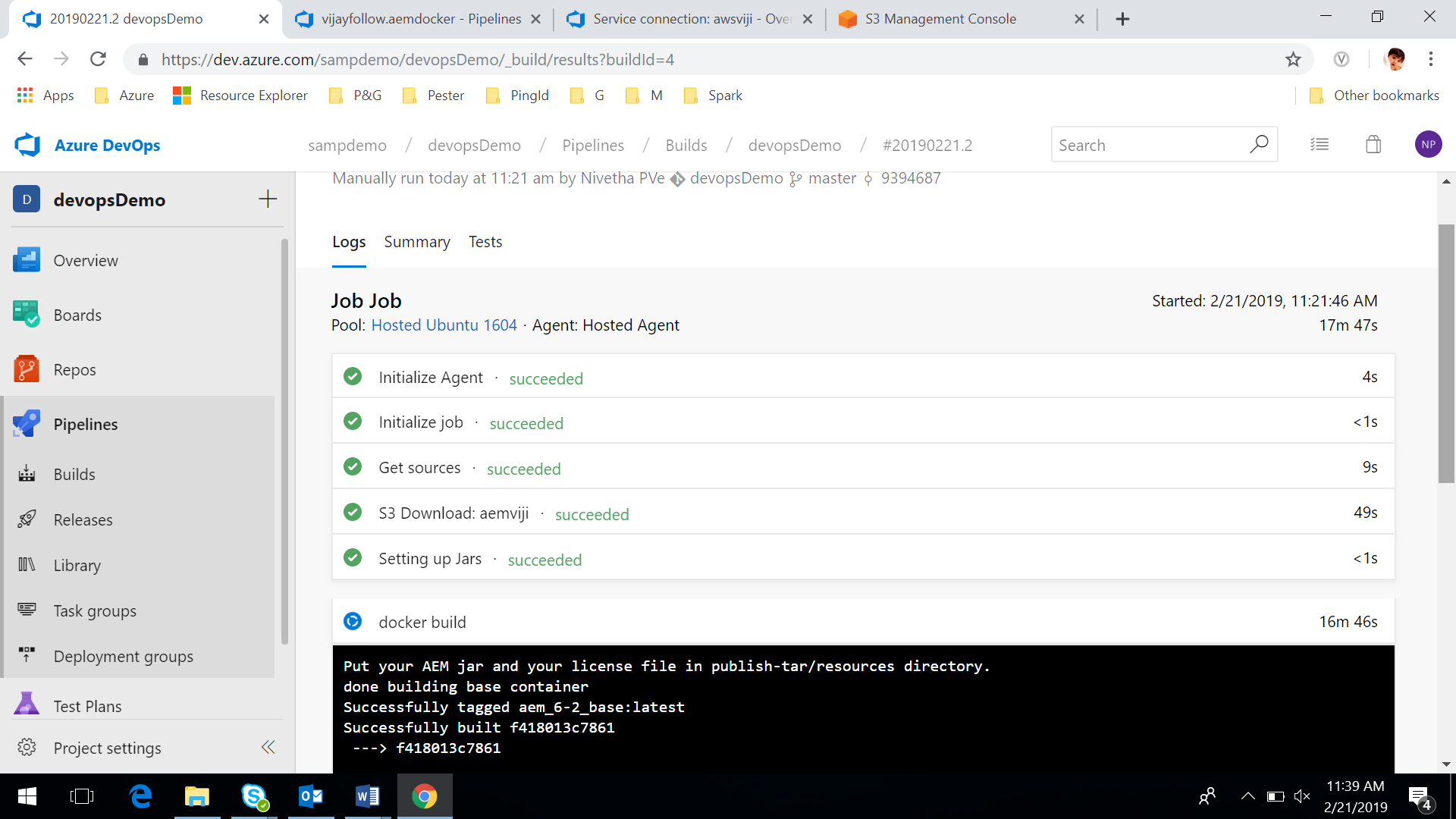
awsCredentials: awsserviceconn #We need to pass the service connection name

repositoryName: 's\_mymvcweb' #ECR Repository

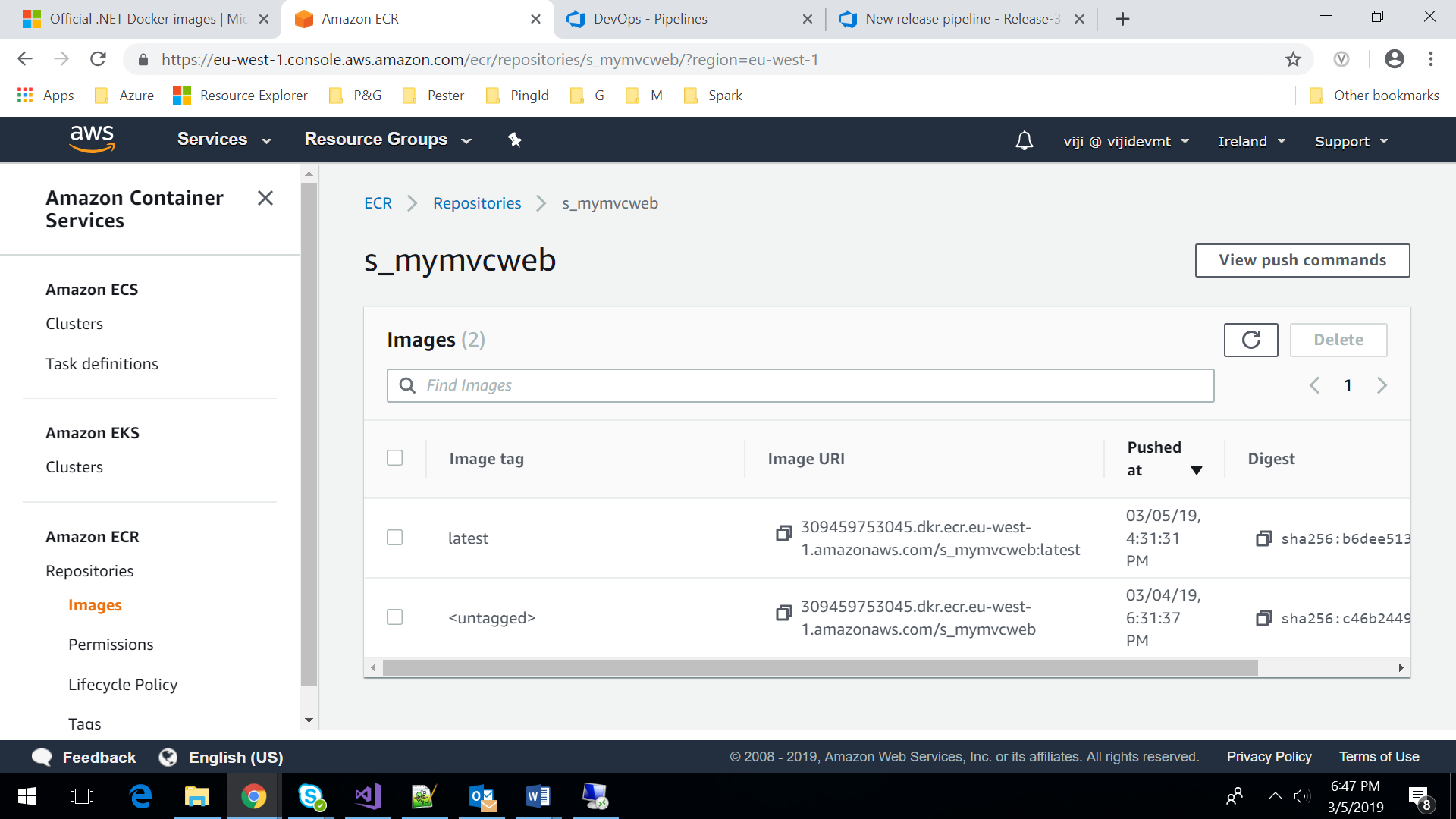
pushTag: 'latest'

autoCreateRepository: true

Step 7: Save and run the build. It will add the images to ecr repository



Images are pushed to the repository



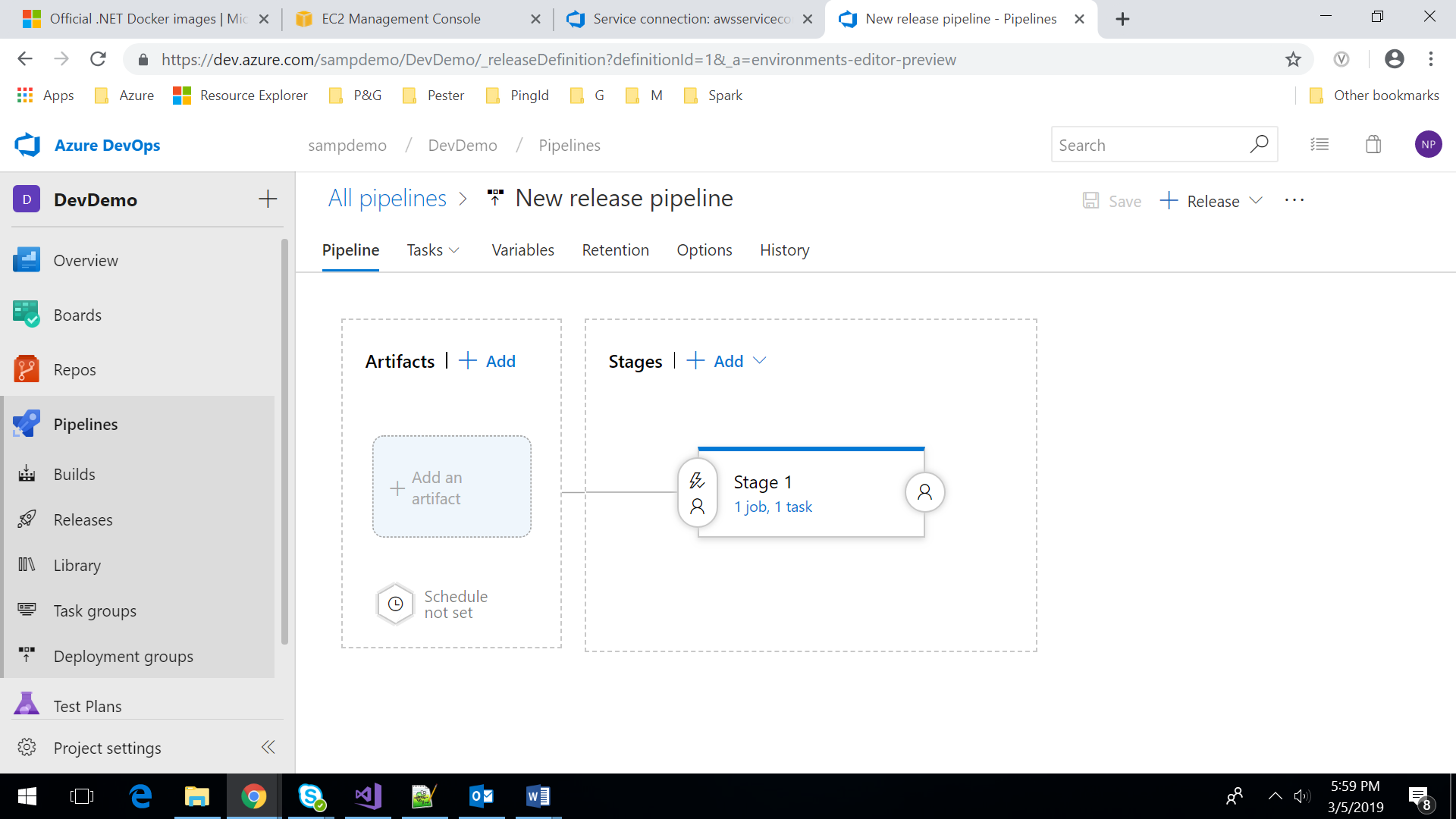
### **Release Pipeline [Continuous Deployment]**

To deploy the application is eks

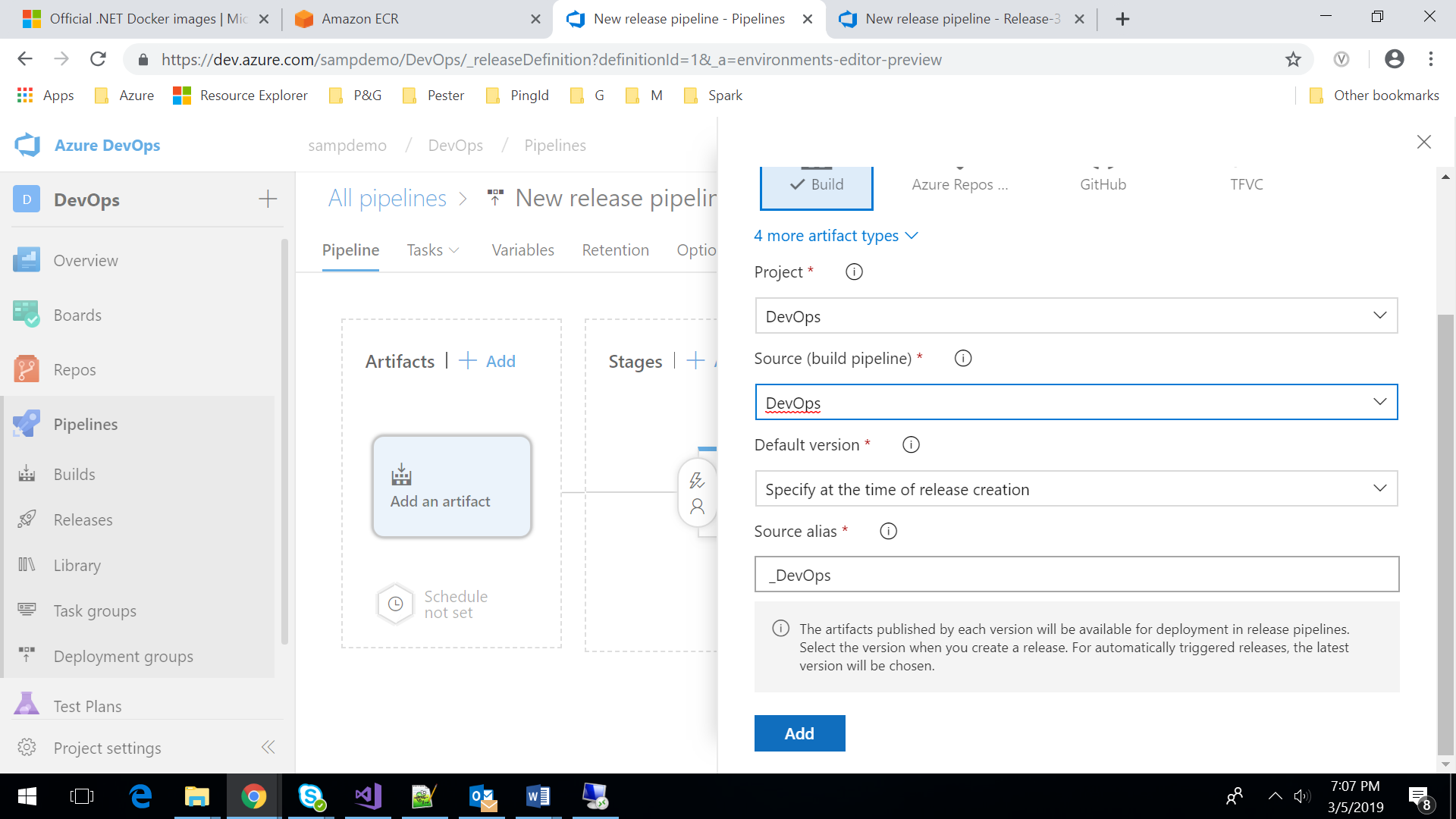
Step1: Manually deploy EKS by using cloud formation template. It will create one Baston VM and worker nodes in EKS.

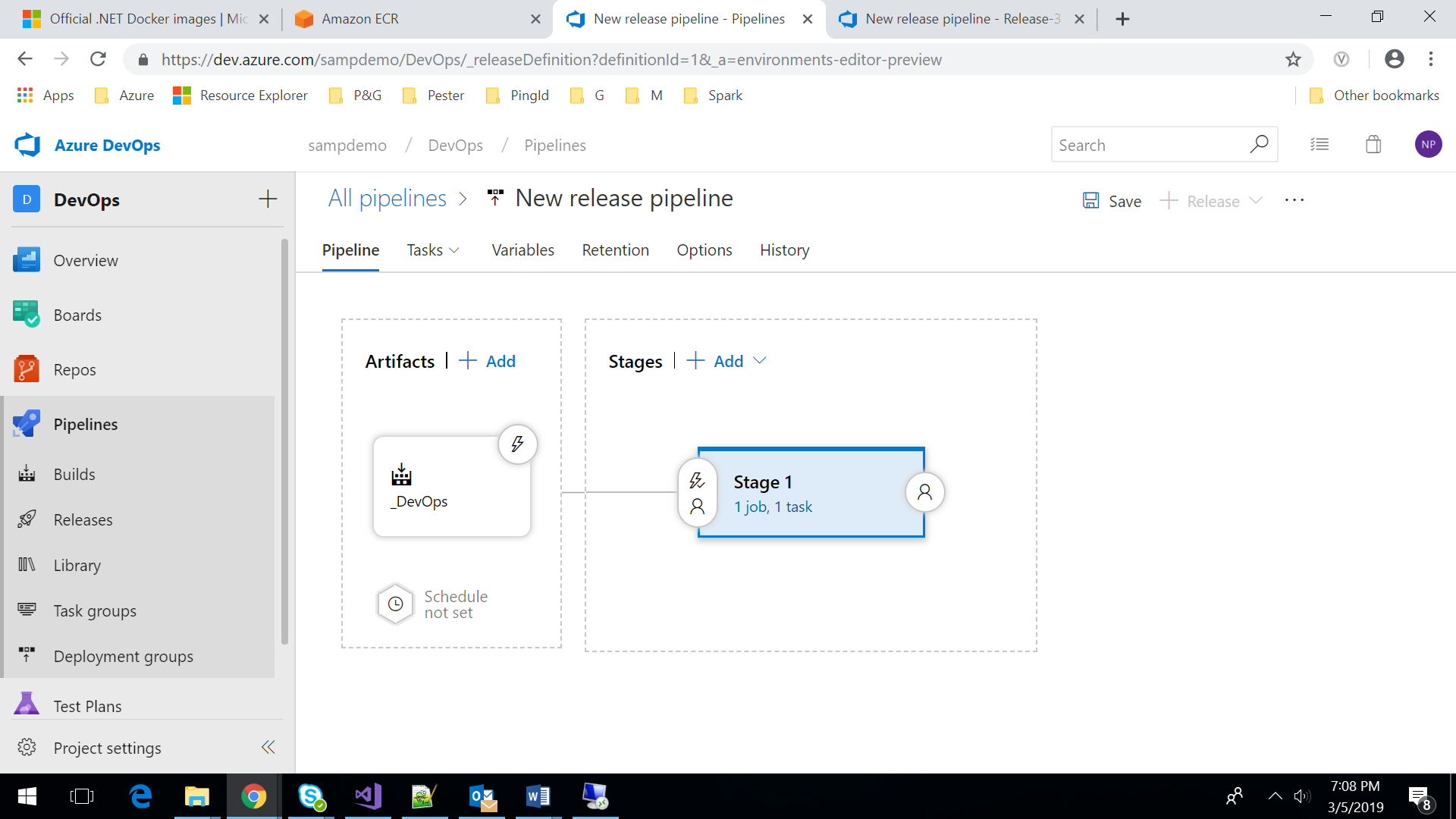
We can have the eks deployment yml files in s3 and manually execute it from there

Step2: Add a release step, Go to pipelines -> releases



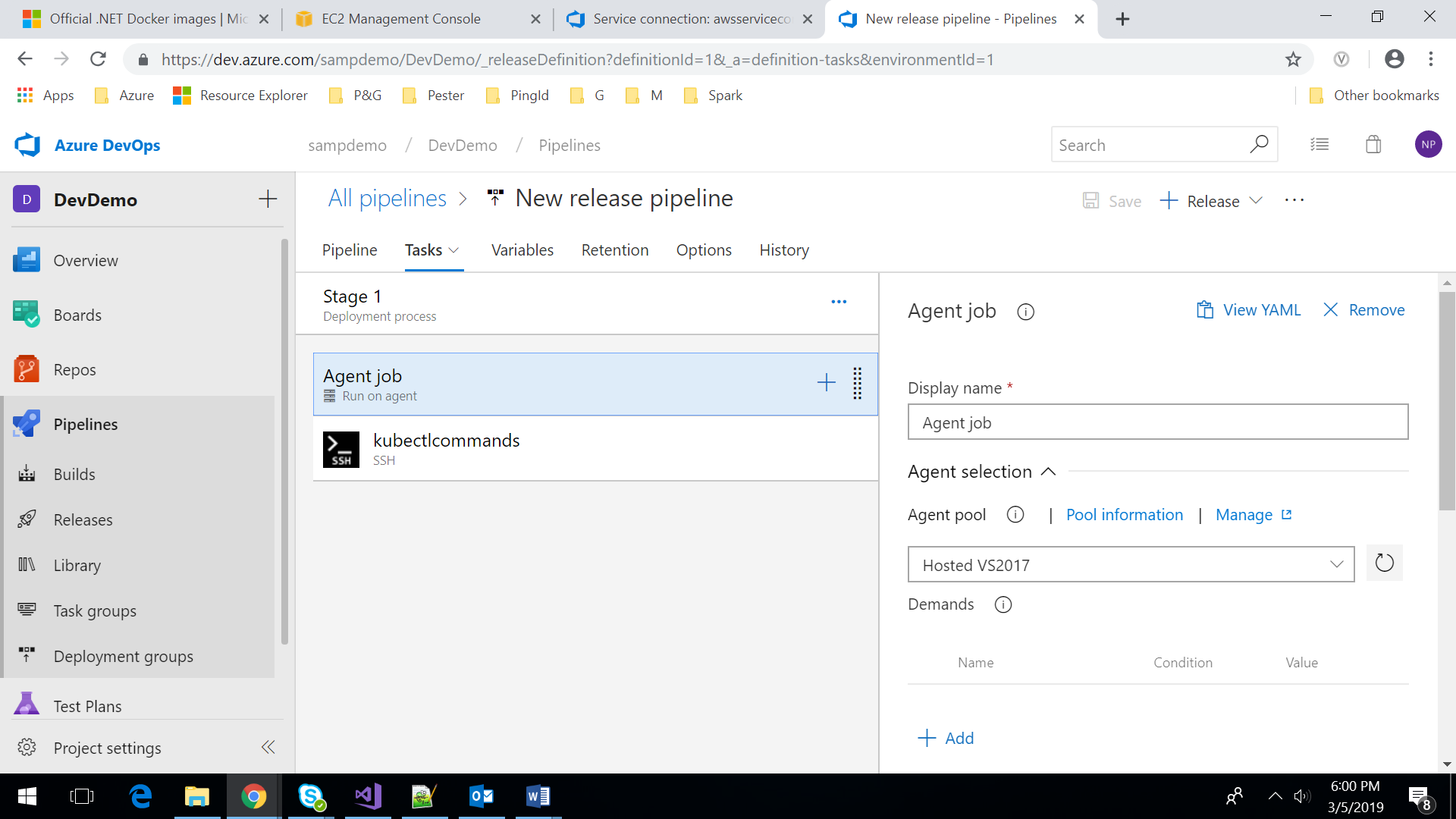
Step3: Integrate Build pipeline to release by adding artifacts as below

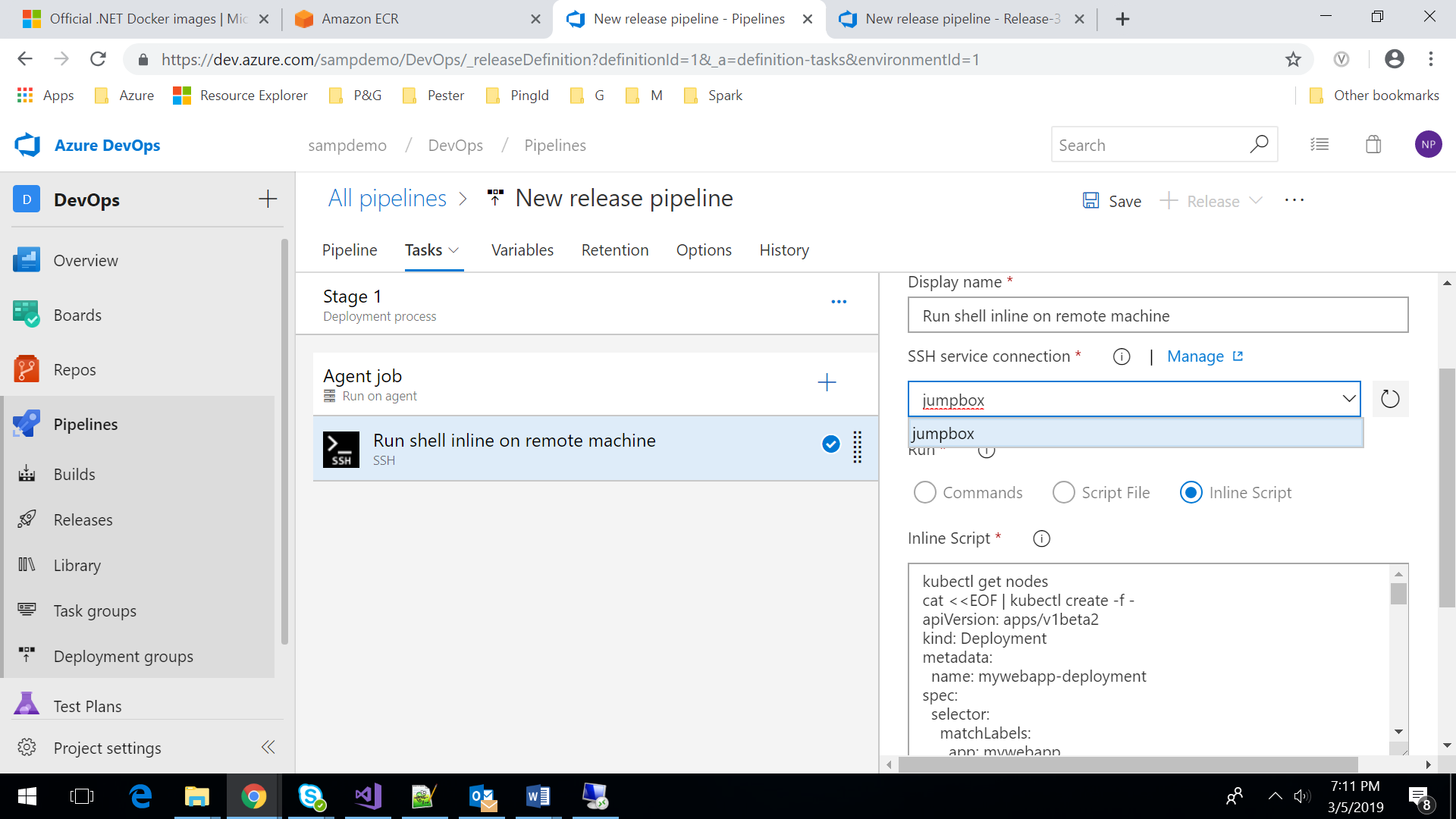
 



Note: We need to give the existing name of the build pipeline

Step4: Add an empty stage and click the next section (Tasks) add task “ssh”, ssh to baston OS





Step6: Add the following inline script to deploy the docker images to eks

kubectl get nodes

cat <<EOF | kubectl create -f -

apiVersion: apps/v1beta2

kind: Deployment

metadata:

name: mymvcweb-deployment

spec:

selector:

matchLabels:

app: mymvcweb

tier: mymvcweb-deployment

replicas: 3

template:

metadata:

labels:

app: mymvcweb

tier: mymvcweb-deployment

spec:

containers:

- name: mymvcweb

image: 309459753045.dkr.ecr.eu-west-1.amazonaws.com/s\_mymvcweb:latest

resources:

requests:

cpu: 200m

memory: 200Mi

env:

- name: GET\_HOSTS\_FROM

value: dns

# Using `GET\_HOSTS\_FROM=dns` requires your cluster to

# provide a dns service. As of Kubernetes 1.3, DNS is a built-in

# service launched automatically. However, if the cluster you are using

# does not have a built-in DNS service, you can instead

# access an environment variable to find the master

# service's host. To do so, comment out the 'value: dns' line above, and

# uncomment the line below:

# value: env

ports:

- containerPort: 80

EOF

cat <<EOF | kubectl create -f -

apiVersion: v1

kind: Service

metadata:

name: mymvcweb-service

labels:

app: mymvcweb

tier: mymvcweb-deployment

spec:

# comment or delete the following line if you want to use a LoadBalancer

# type: NodePort

# if your cluster supports it, uncomment the following to automatically create

# an external load-balanced IP for the frontend service.

type: LoadBalancer

ports:

- port: 5000

selector:

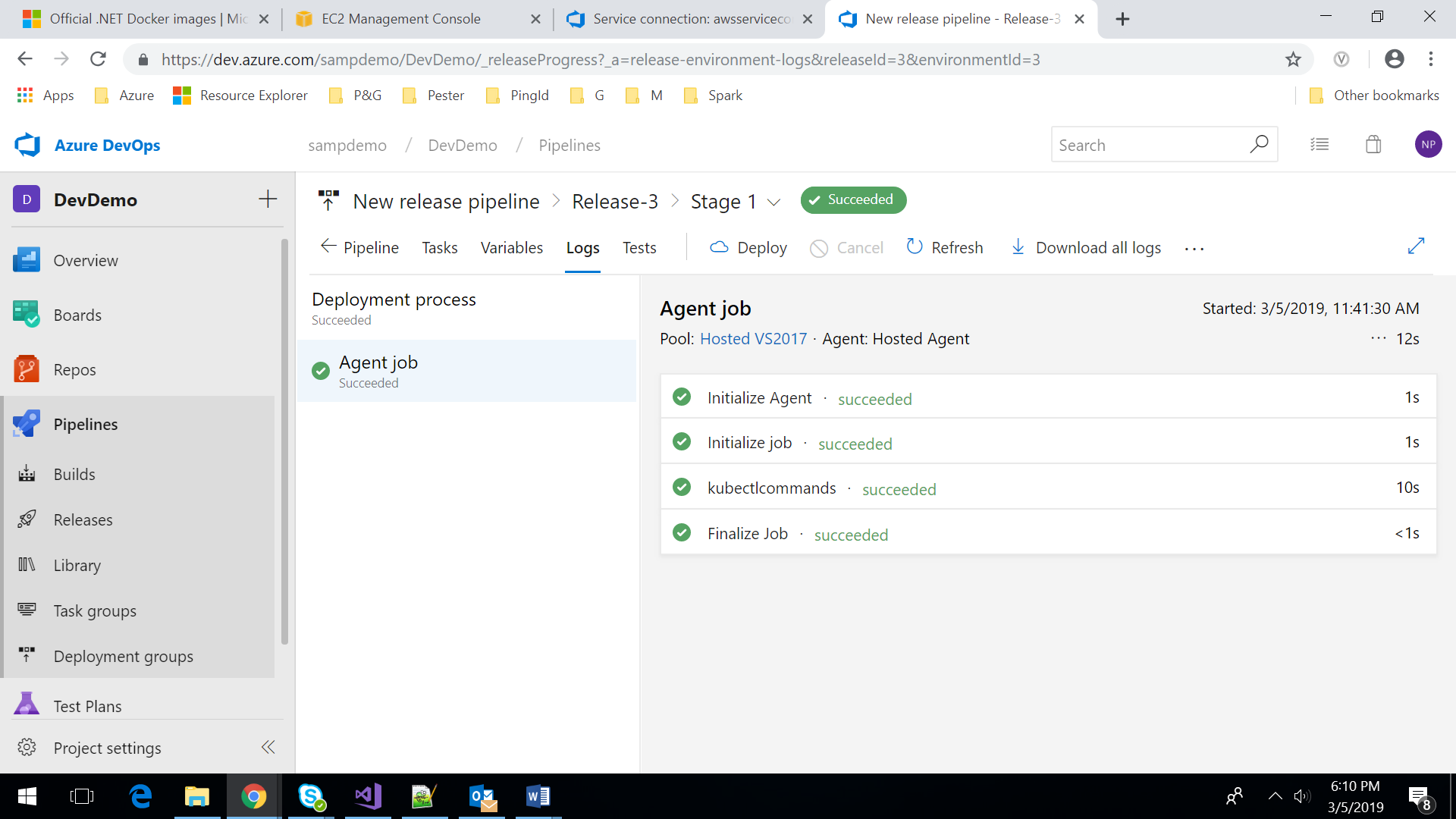
app: mymvcweb

tier: mymvcweb-deployment

EOF

kubectl get services

Step7: Save and create a release



Step8:

After the script is executed, Load balancer will be provisioned where we can fetch the url. After accessing the URL we can get the desired output

