DEVOPS HANDS-ON

Objectives:

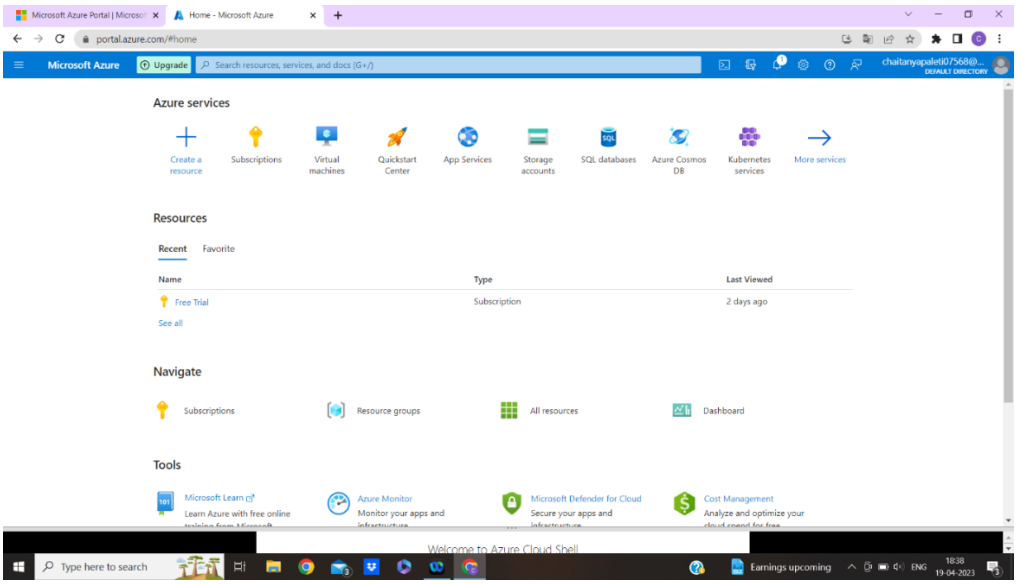
- Quickly set up two identical Azure environments using a (predefined) template.
- Build a web application package in a Continuous Integration (CI) Build Pipeline.
- Deploy the web application package to Azure environments with a Continuous Delivery (CD) Release Pipeline.
- Track functional changes throughout the CI/CD Pipelines

Prerequisites:

- An active Azure Subscription
- A modern web browser (Edge, Chrome or Firefox preferred, Internet Explorer 11 to some degree) with internet access.

Exercise 1: Log on to the Azure Cloud Shell

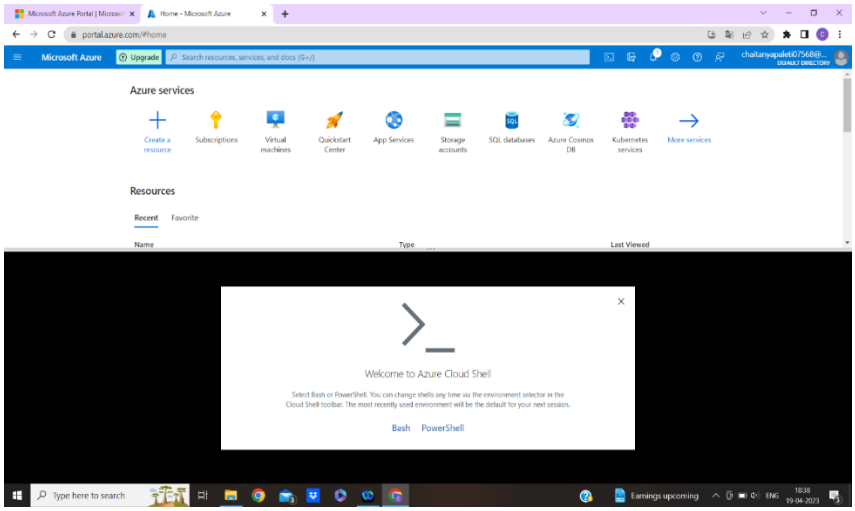
1. Upon successful creation of Azure Free Trial Subscription, you will be redirected to Azure Portal home page as follows:



- You can check your account credentials on the top right corner.

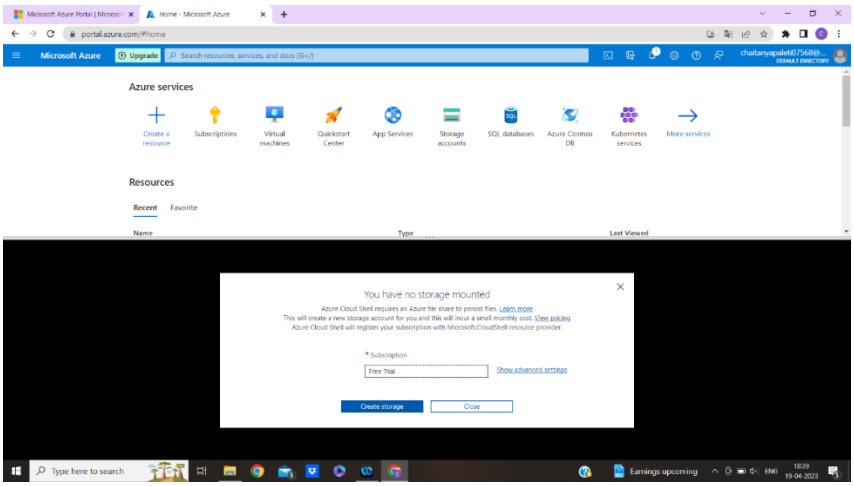
eg. chaitanyapaleti07568@gmail.com (Default directory), which is my free trial account.

2. Start the Azure Cloud Shell (Bash) by clicking the console icon (>_) in the top bar of the portal. You will get like this:

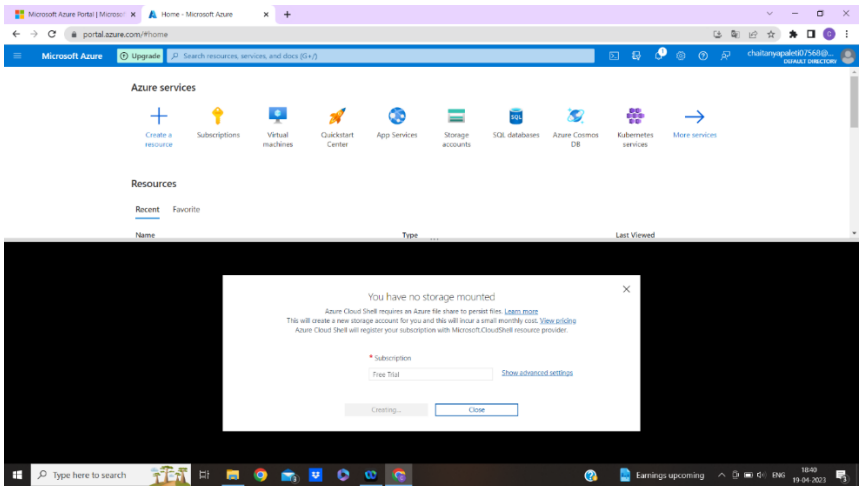


- If you are first time using Azure Cloud Shell, you will be asked a few questions.

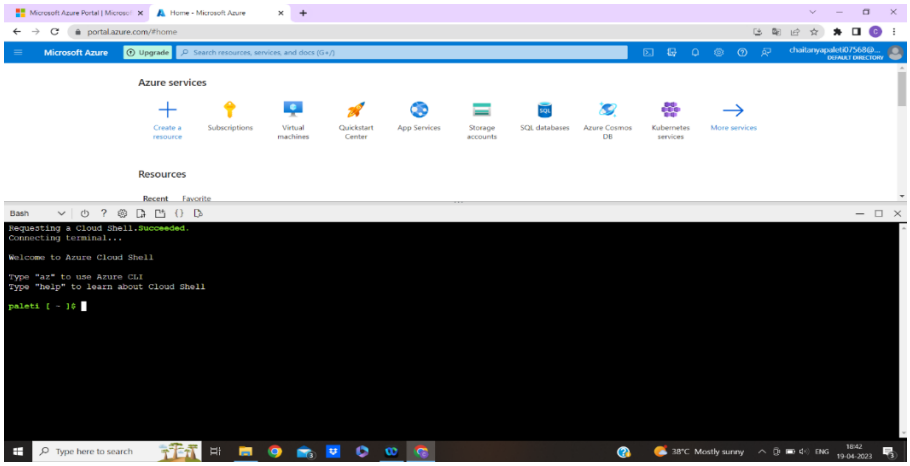
3. Click on **Bash**.



4. Click on **Create Storage**.



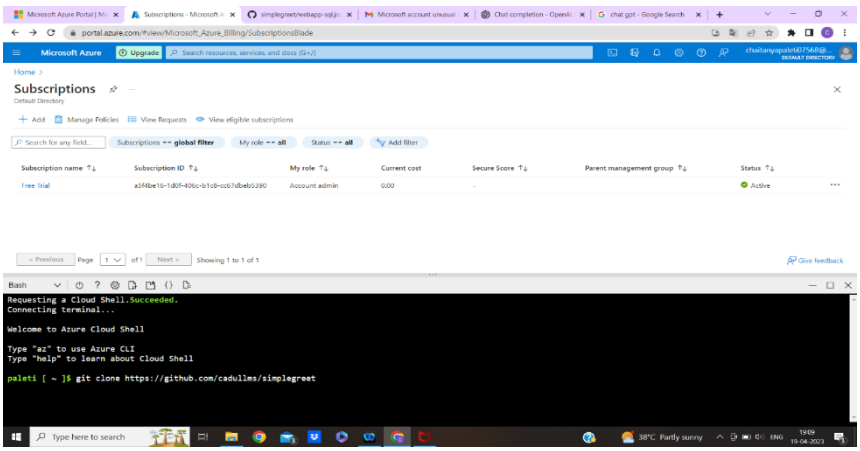
- Your console should then look like this:



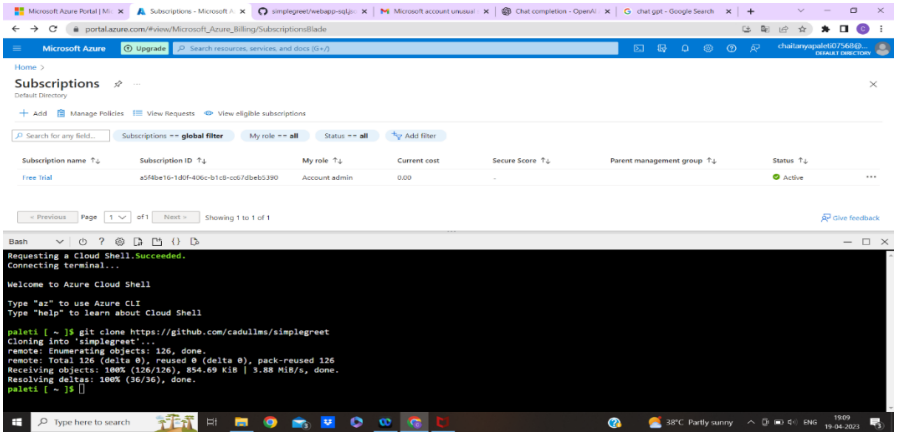
Exercise 2: Create the Azure Environments

- We will start by creating the environments, to which we will deploy our application later on. Our environment will consist of one app service (containing a Java Web App) and an Azure SQL Server database.
- We will have one **development** environment and one **production** environment, so that we will be able to safely test any change in an isolated environment that is as structurally similar to production as possible. The key is not experiencing production releases that fail badly because some detail in production was different from what we tested against.
- The approach we are using is **Infrastructure as Code (IaC)**. In this approach, changes to an environment are always performed through scripts and templates that are version controlled and require no human interaction other than passing in some parameters.
- For Azure, the API called Azure Resource Manager (ARM) and with ARM templates we can easily create our two environments in exactly the same way. In our case, our templates are already predefined for the exact needs of our application.

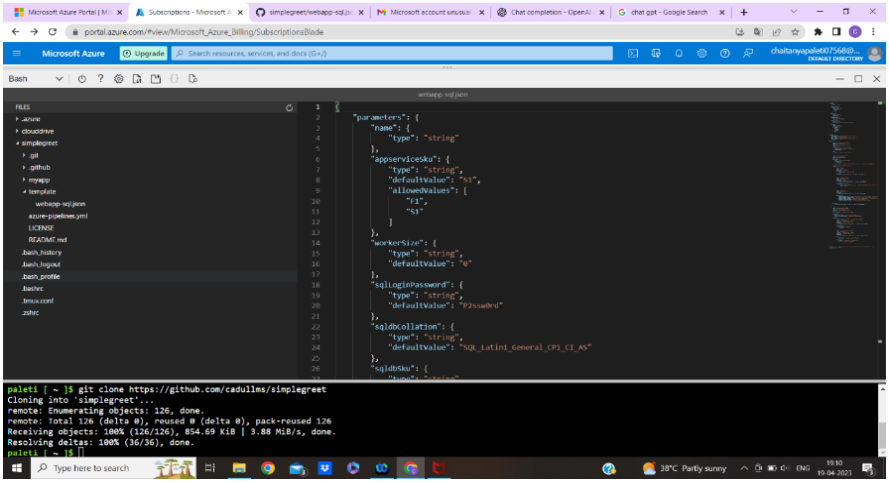
1. In the Cloud Shell, type:
   git clone https://github.com/cadullms/simplegreet



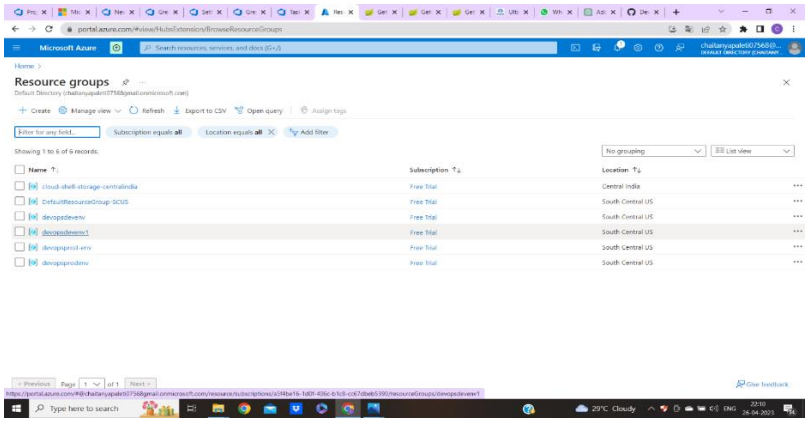
- This command will download(clone) all the codes and templates required to deploy our application from github.



- We can explore what we just downloaded (cloned) from github.
2. In the toolbar of the shell click Open editor ({}). This opens a text editor above the shell.
3. In the editor, navigate to the file **simplegreet/template/webapp-sql.json**. This is the Azure Resource Manager (ARM) template we are going to use to set up our environments.

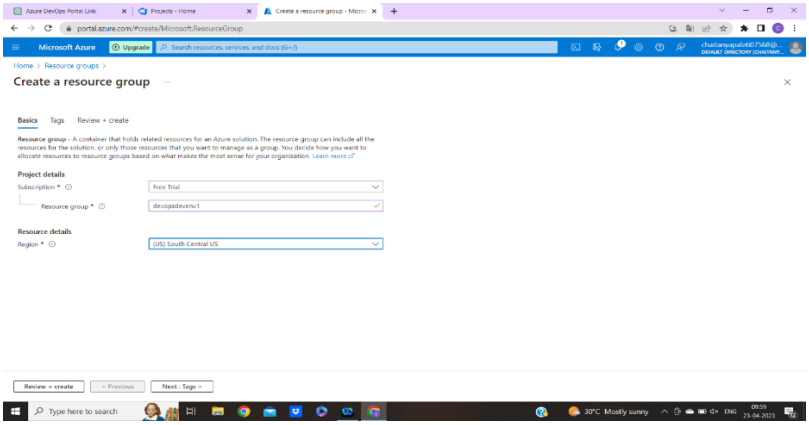


4. Create two resource groups : one for dev,one for prod
- You can find resource group in azure portal home page or you can search resources in the home page itself.
  Resource grooups -> +create -> create resource group

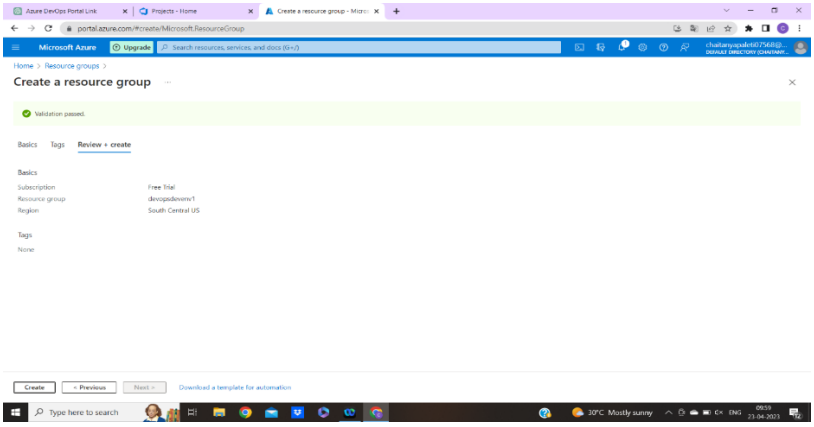


5. Provide Subscription details (here **Free Trial**) , Resource group name for Dev(**eg-devopsdevenv1**) and Region (**eg-South Central US**).

- The reason for specifying the location of a resource group is to specify a location for data/metadata for the deployment to be stored in.

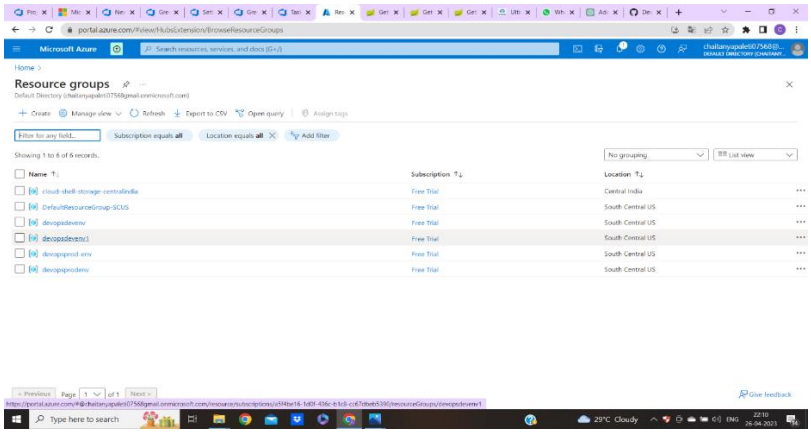


6. Leave the default and Click on **Review+create**.



7. Once validation is done click on **Create**.

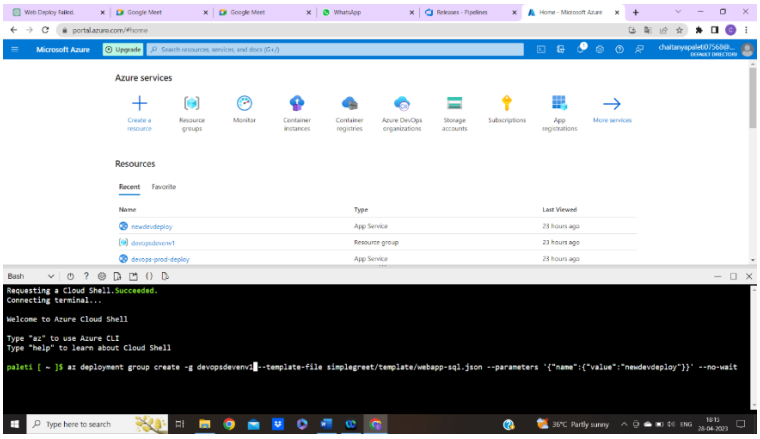
8. Once it is created you can check in the Resource groups lists.



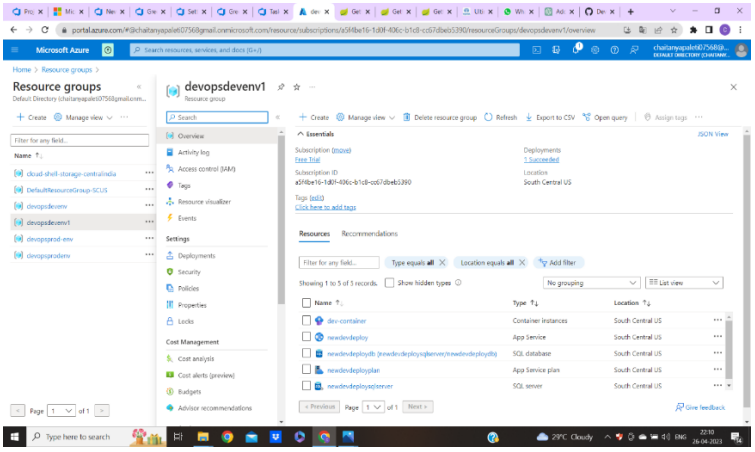
9. Repeat the same for creating the resource group for Prod.

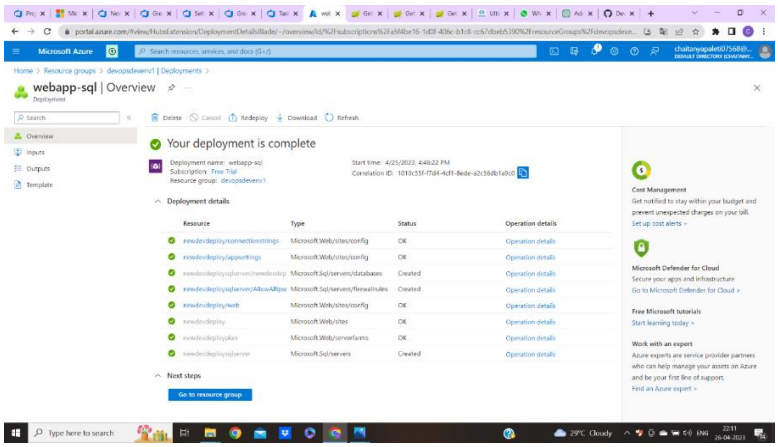
- **devopsdevenv1** and **devopsprod-env** are the resource groups I created for **Dev** and **Prod** respectively.

10. In the cloud shell, execute the command:
    az deployment group create -g <resource group name> --template-file simplegreet/template/webapp-sql.json --parameters '{"name":{"value":"<a unique name>"}}' --no-wait
    This command deploys a resource group deployment using an Azure Resource Manager template.
    - **az deployment group create -** This is the Azure CLI command for creating a resource group deployment. az deployment group create
    - **-g <resource group> -** This parameter specifies the name of the resource group where the deployment will be created. The name of my **Dev** resource group is **devopsdevenv1.**
    - **--template-file simplegreet/template/webapp-sql.json -** This parameter specifies the location of the Azure Resource Manager template file that will be used for the deployment. In this case, the template file is located at "simplegreet/template/webapp-sql.json".
    - **--parameters '{"name":{"value":"<a unique name>"}}' -** This parameter specifies the parameters for the deployment. In this case, the parameter named "name" is being set to a unique value. This value will be used to name the resources created by the deployment.
    - **<a unique name>** is a name in lowercase letters that you can freely choose, but that must still be available as <a unique name>.azurewebsites.net. **newdevdeploy** is the unique name I have given.
    - **--no wait -** This parameter specifies that the command should not wait for the deployment to complete before returning.



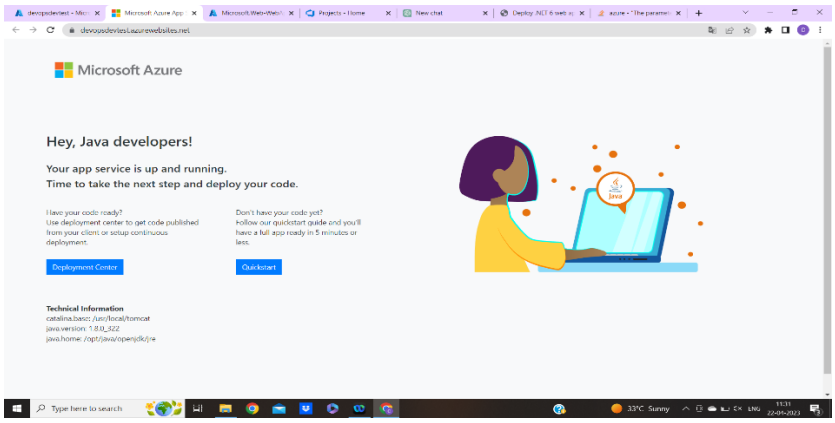
- You can check the deplyment status in the following path:
  Resource group name -> Essentials ->Deployments
  OR
  Click on Deployments under Settings as shown in the below screenshot.



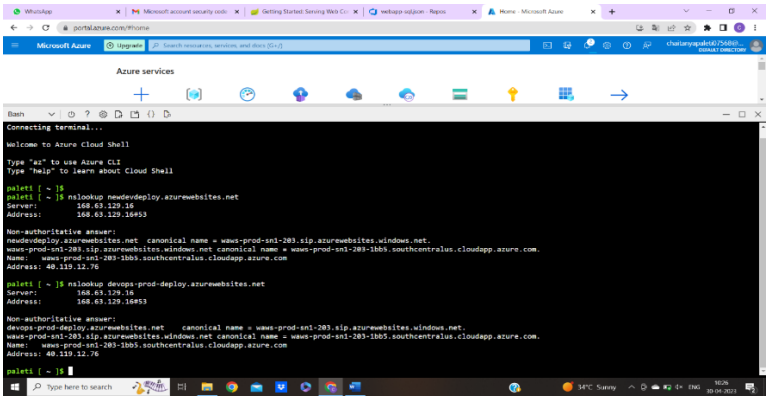


11. Repeat the preceding steps with the **Prod** resource group and another unique name.
    - I have given **devops-prod-deploy** as unique name here.
    - We can check our new website in a browser at https://<a-unique-name>.azurewebsites.net.
    - For Dev environment https://newdevdeploy.azurewebsites.net.

- For Prod environment https://devops-prod-deploy.azurewebsites.net.
- You will get a default page like this in both the websites for now.



12. You can check the availability of the name by typing **nslookup<a unique name>.azurewebsites.net** in any shell and check whether that returns an IP already.
    - Here, **nslookup newdevdeploy.azurewebsites.net** for Dev environment and **nslookup devops-prod-deploy.azurewebsites.net** for Prod environment.

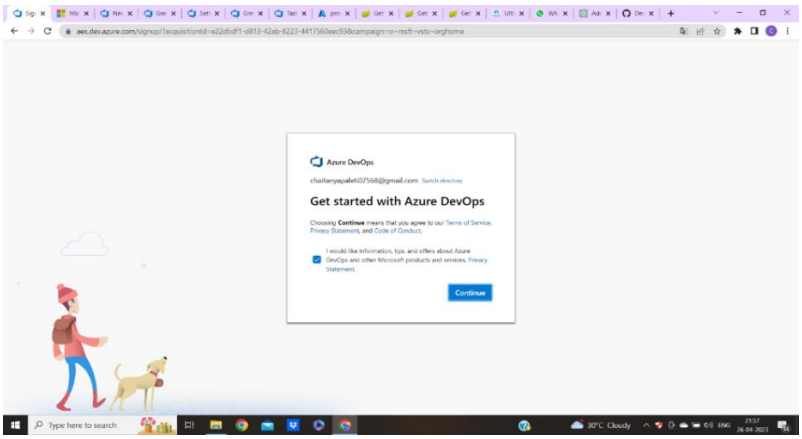


Exercise 3: Set up Azure DevOps

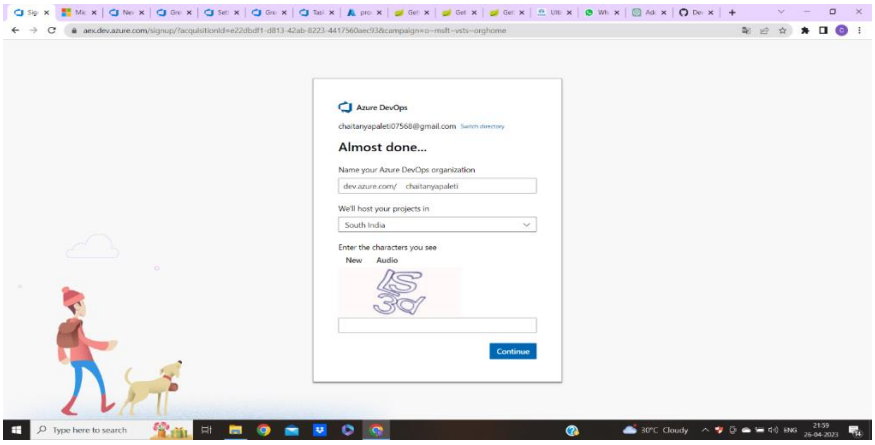
1. In additional browser tab, navigate to **https://dev.azure.com**.

- You will see the start free page if you are using the first time. If you already have an account sign in to Azure DevOps.

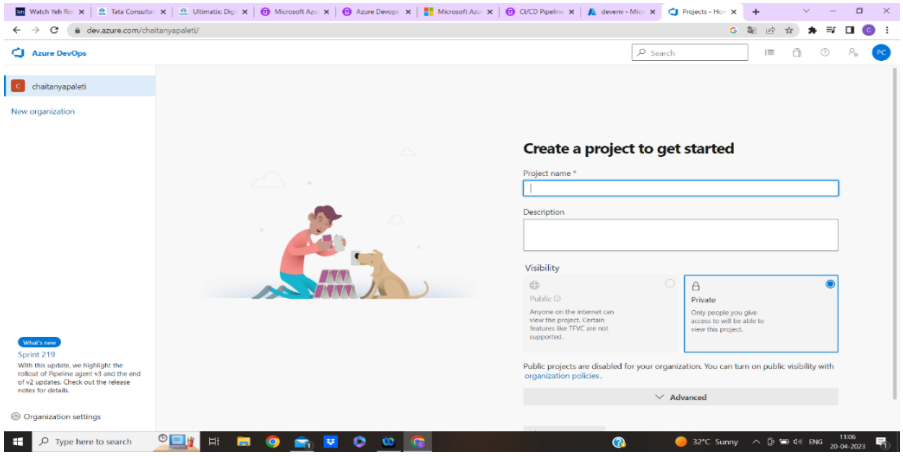
2. When asked, click continue.



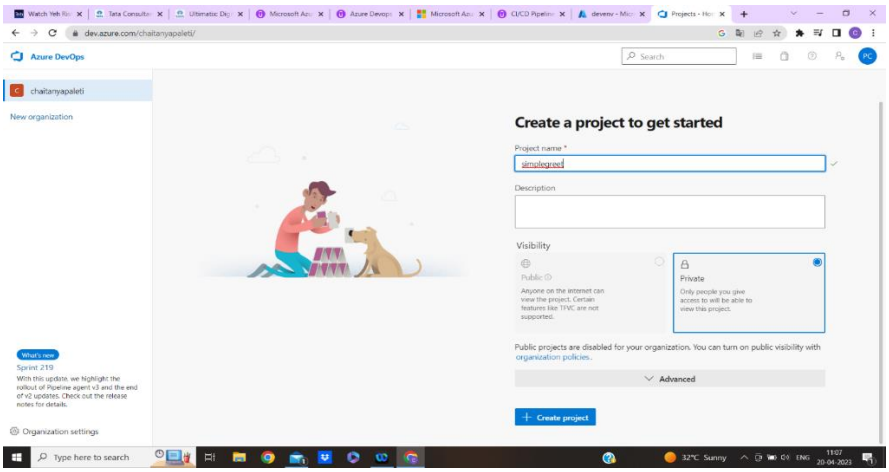
- This will create an organization with the name of your account (ex-chaitanyapaleti). But I have created my project in **chaitanya-paleti** organization.
- The organization is the root object for your work in Azure DevOps and contains projects.



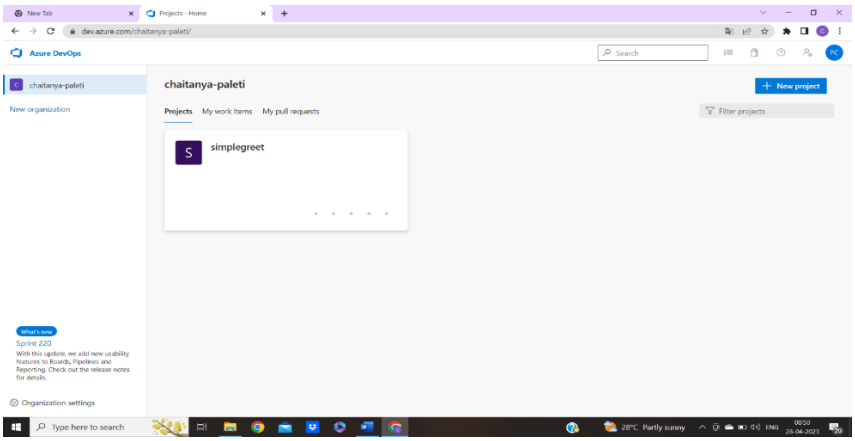
3. When you click continue, you will arrive at a screen that asks you to **Create a project to get started**.



4. At that screen enter "**simplegreet**" as the project name, leave everything as default, then click **Create project**.



- You will see the page like this:

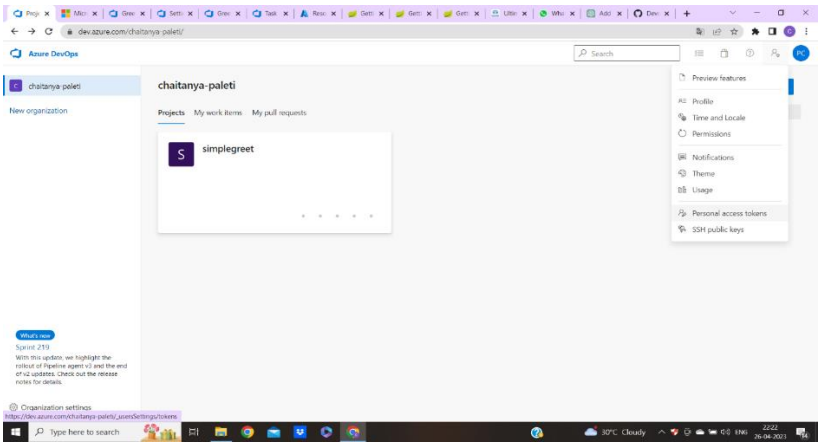


Exerccise 4 : Create a build and release agent

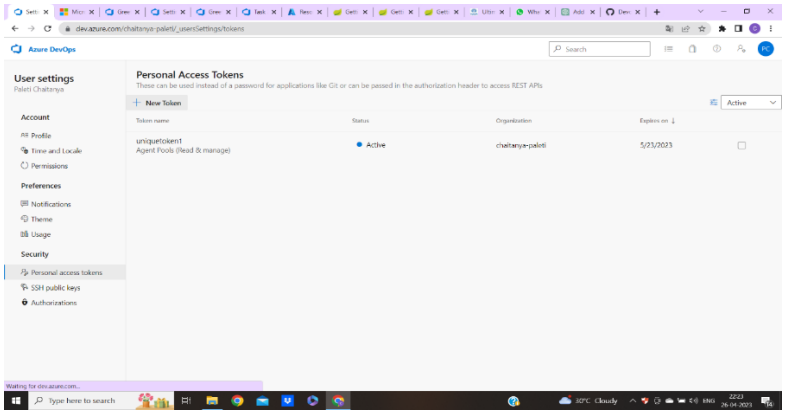
- The agent is a piece of cross-platform software that can run anywhere in the cloud or on premises and connects to Azure DevOps to wait for jobs to be executed, like building a piece of software, running tests, deploying software, and so on.
- We are using private agent using our build agent container image from Docker Hub. For the creation of an agent we always need to authenticate against Azure DevOps. In the container we will be not be able to do that interactively, thus we will be using another authentication option for Azure DevOps, which is **Personal Access Token (PAT)**.
- Then, our containerized build agent will run in Azure container Instances (ACI), a service in which we can run single container without any up-front setup with just one command.

1. Create a **PAT** for your Azure DevOps organization

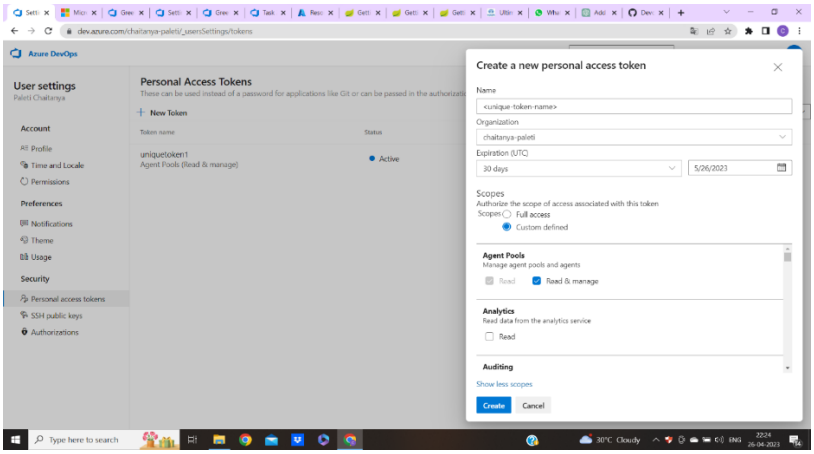
- In Azure DevOps portal, click on the icon of profile as show in the screenshot and click on **Personal access tokens**.



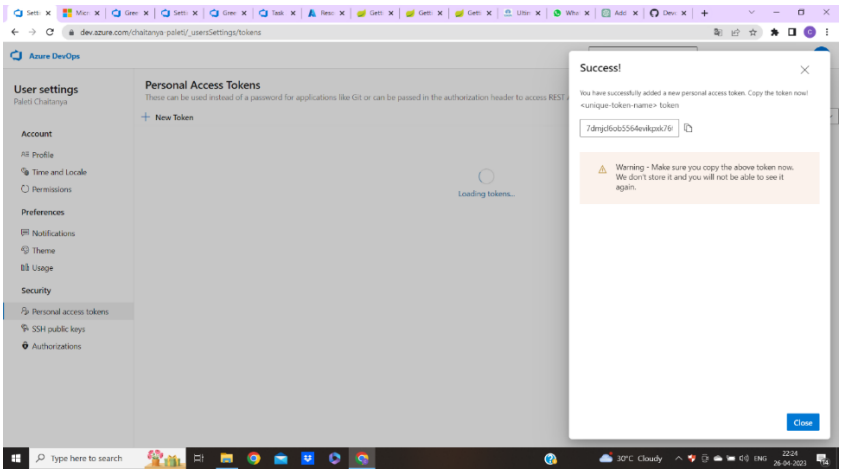
2. Click on **+New Token**.



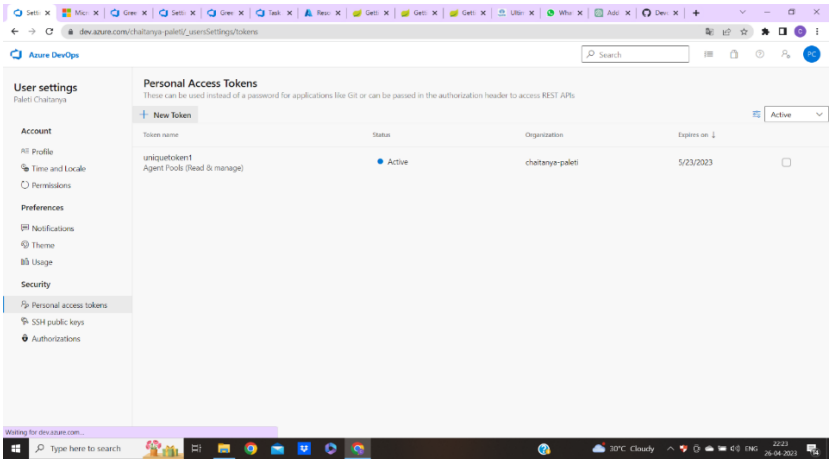
3. It will enable you a **Create a new personal access token** drawer.
4. Give the name to the token (**eg-uniquetoken1**), organization (**eg-chaitanya-paleti**).
5. For Scopes, select Custom defined and use the scope **Agent Pools (Read & Manage)** for the PAT.
6. Click on **Create**.



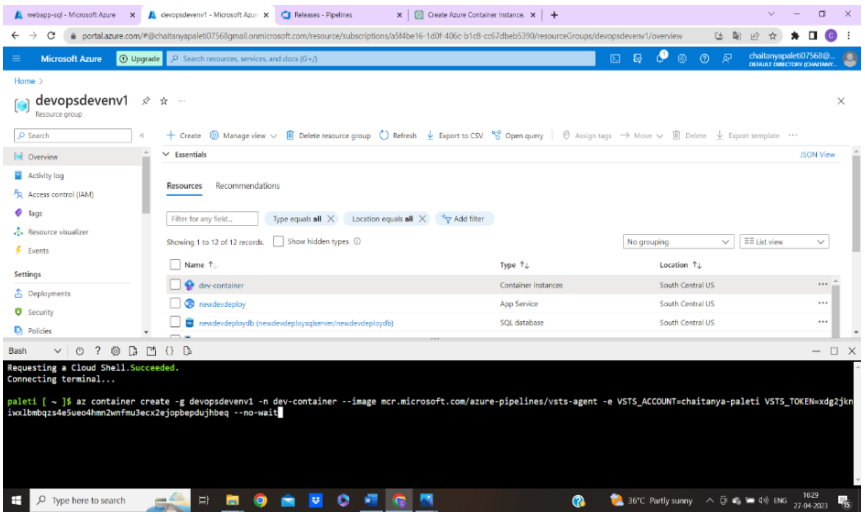
7. Copy the PAT token and save it in a note pad for future use and click close.



- You can see the PAT token you have created in the list.



8. In the Cloud Shell, execute this command:
   az container create -g <resource group> -n <some name> --image mcr.microsoft.com/azure-pipelines/vsts-agent -e VSTS_ACCOUNT=<azure devops organization name> VSTS_TOKEN=<pat> --no wait
   - **az container create -** This is the Azure CLI command for creating a container instance.
   - **-g <resource group> -** This parameter specifies the name of the resource group where the container instance will be created. I used my **Dev** resource group **devopsdevenv1.**
   - **-n <some name> -** This parameter specifies a name for the container instance. Here **dev-container** is the name of the container instance.
   - **--image mcr.microsoft.com/azure-pipelines/vsts-agent -** This parameter specifies the image to use for the container instance. In this case, the image is "mcr.microsoft.com/azure-pipelines/vsts-agent", which is the Azure Pipelines VSTS Agent.
   - **-e VSTS_ACCOUNT=<azure devops organization name> VSTS_TOKEN=<pat> -** This parameter specifies environment variables to pass to the container instance. In this case, it sets the **VSTS_ACCOUNT** and **VSTS_TOKEN** variables, which are used to authenticate the VSTS Agent with an Azure DevOps organization. Replace <azure devops organization name> with your Azure DevOps Organization. My organization name is **chaitanya-paleti.**

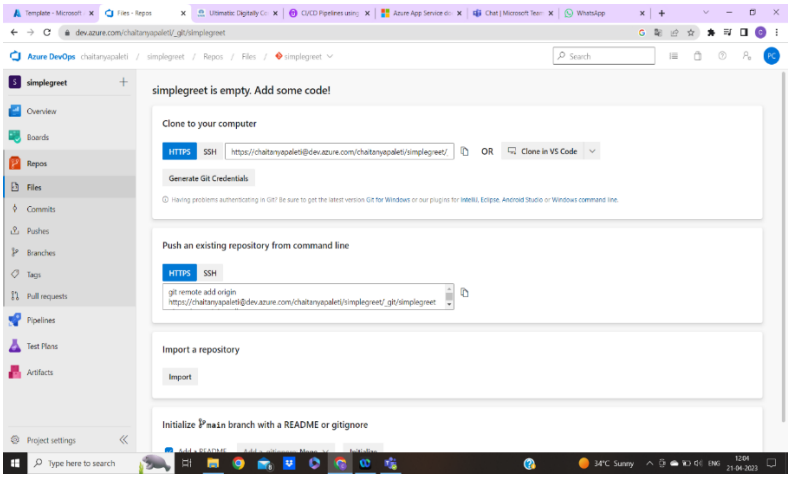


9. Once the above command is executed successfully, a container instance will be created for that resource group.
10. You can see in the above screenshot **dev-container** under Resources as container instances or search Container Instances from home page.

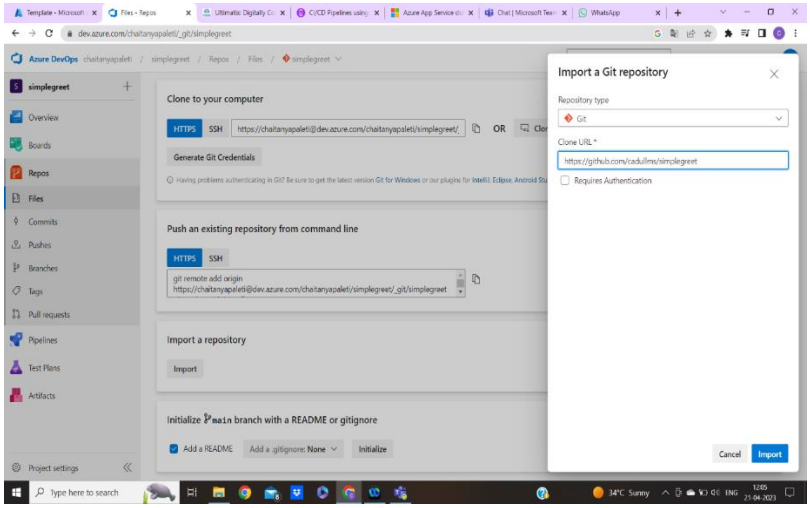
Exercise 5: Import Code and Create the Azure DevOps Build Pipeline

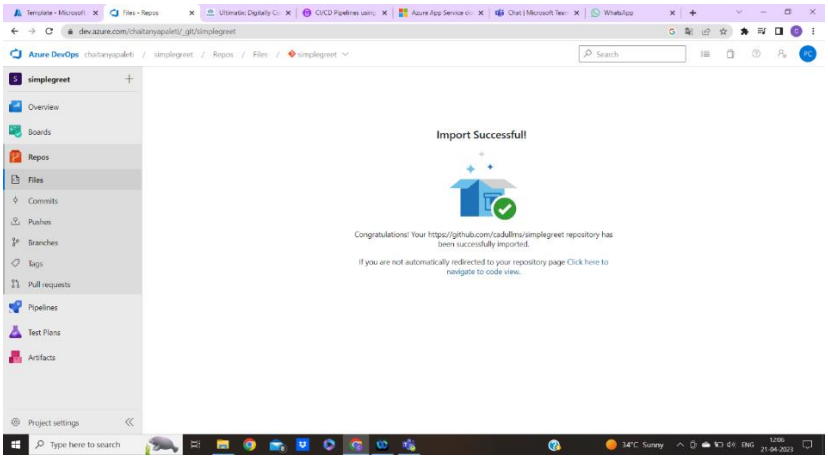
- We need code to build something in a Build Pipeline. So we will just import the code, then we will create the pipeline with yaml-definition (contains all information about how our code will be compiled and packaged in Azure DevOps).

1. In your browser, navigate to your Azure DevOps project.
    - Click **Repos**:

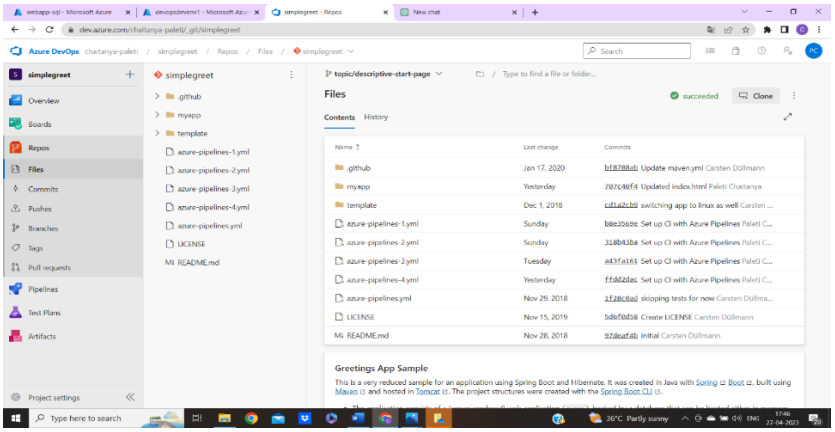


2. In the repository view, under **import a repository,** click **import** and enter the url https://github.com/cadullms/simplegreet and click **Import**.



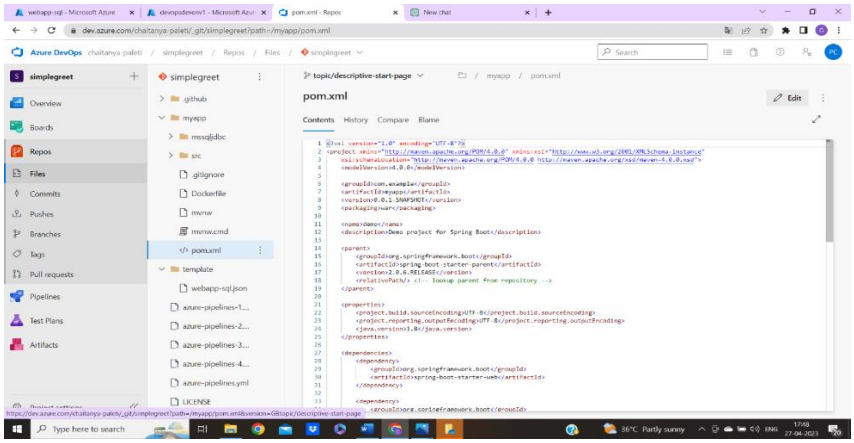


- The imported repository is the same we cloned to get our infrastructure - it contains the code of our application as well.

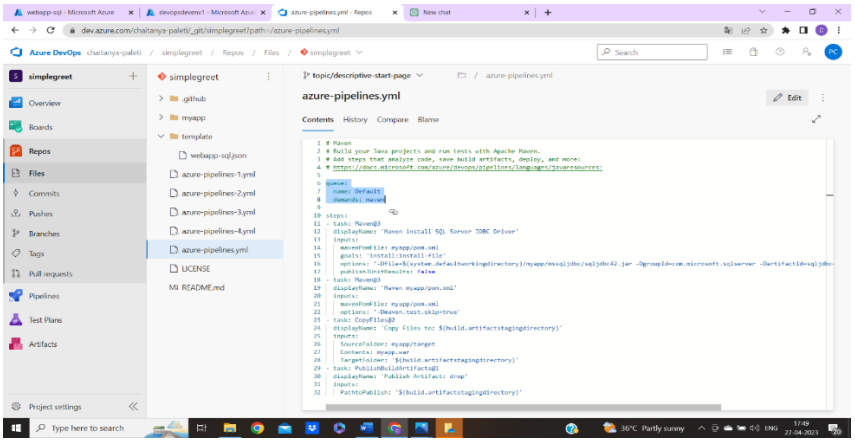


3. Once the import has finished, your browser should show a code explorer view of the imported code as shown in the screenshot.

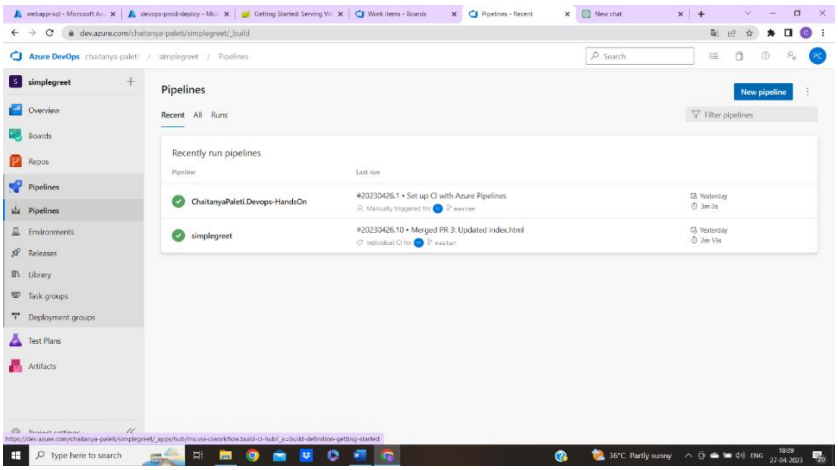
- **pom.xml** - which contains all information we need to build the application with Maven.



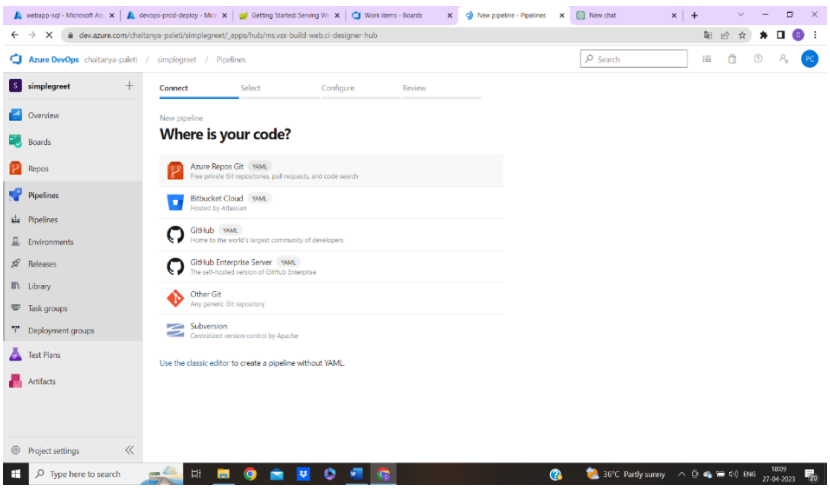
- **yaml file** - which contains additional information like on which type of agent the build run on and where the result of the build should be put.
- In our azure-pipelines.yaml, it is defined that we should run in the Default agent queue (to which we added our private agent in the previous exercise)

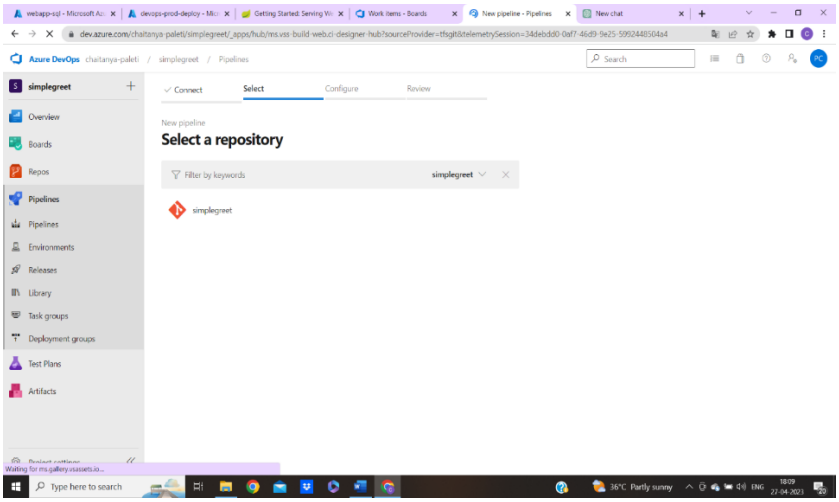


4. To actually run the build, in the menu, click **Pipelines**:



5. Click **New Pipeline**. When asked, where your code is, choose **Azure Repos** and then select our Repo (**simplegreet**).

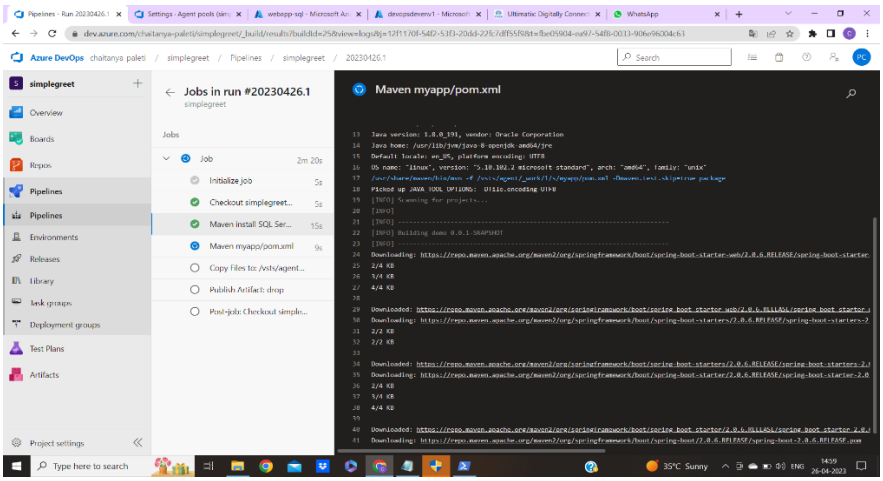




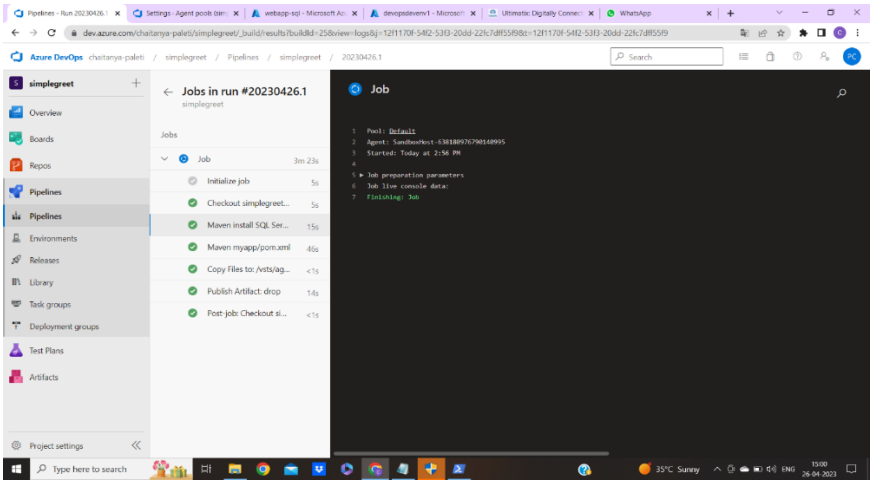
- You should now see the content of the **azure-pipelines.yaml** file. This is because by convention, if a file with that name exists in the root folder of a repo, by convention Azure DevOps assumes that this must be our build definition.

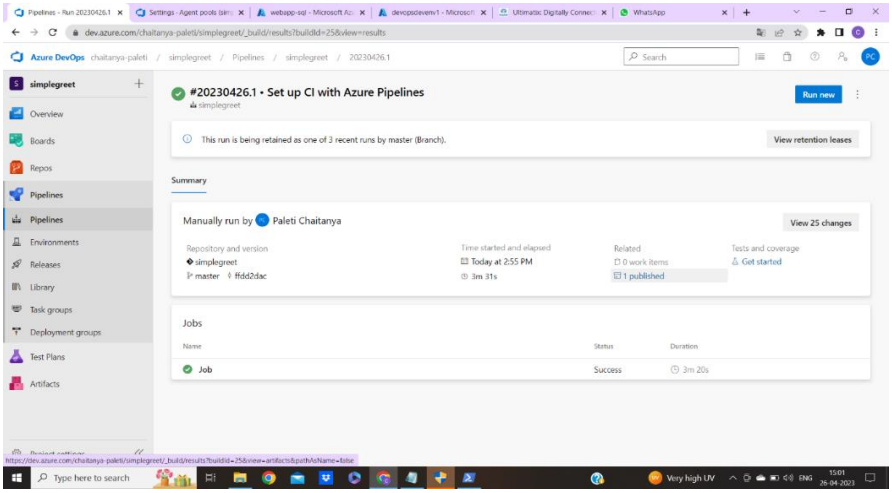
6. Click **Run**.

- You can now see a build page where you can follow the progress of the build.

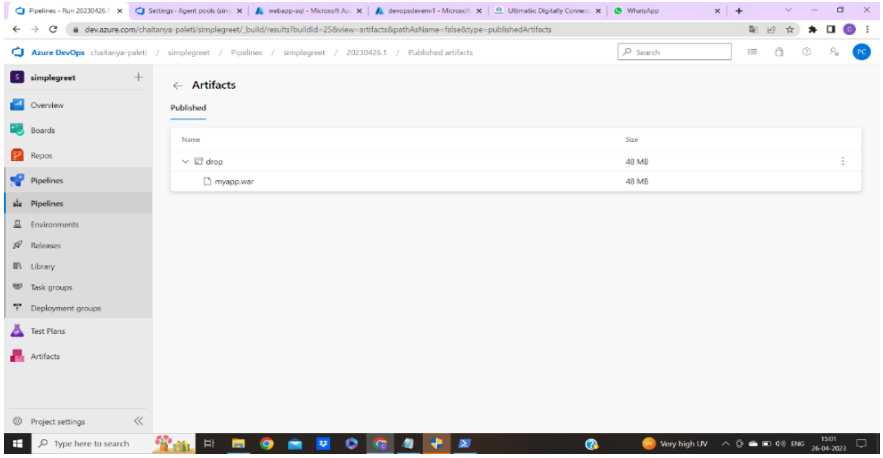


- The build is now defined as a Continuous Integration (CI) Build, which means that whenever new code is pushed to our repository, the build will run again.





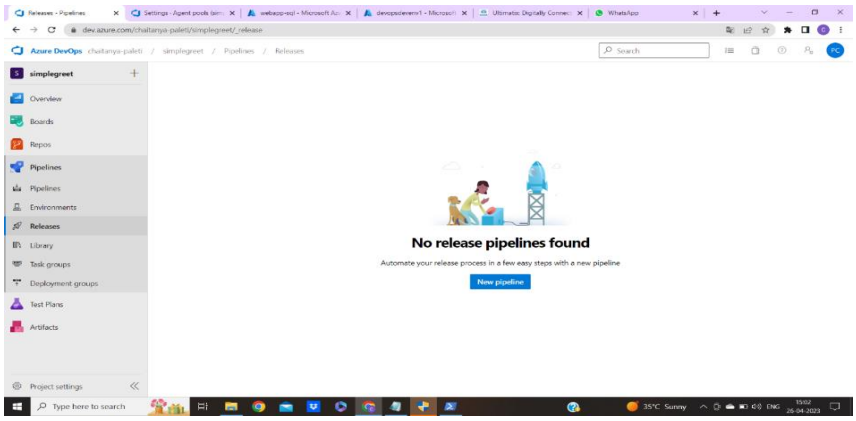
7. Click on **1 published**. You can see the artifact (myapp.war file), our build produced. This artifact is now we will use in Release Pipeline.



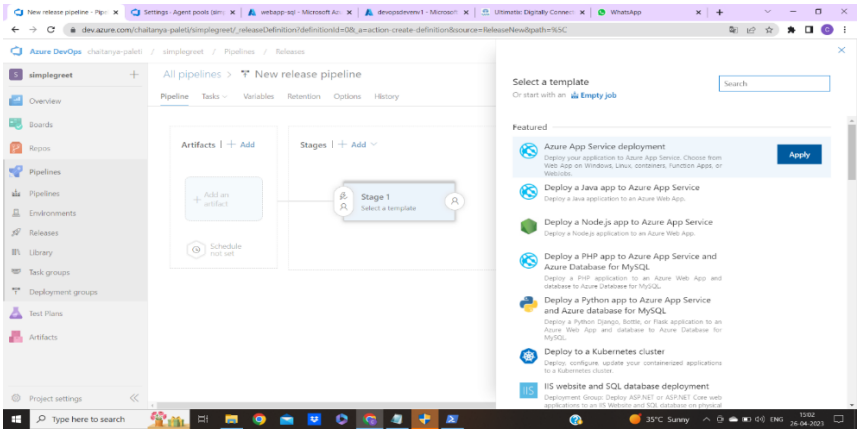
Exercise 6: Create the Azure DevOps Release Pipeline

- The output of the CI part of CI/CD (.war file we just built) should always be independent of the actual environment we are deploying to.
- Yet to actually deploy to the different environments (in this case, one Dev and one Prod environment) of course we need to make the environments known to the system and define the steps of how we can actually deploy the generic package to the specific environment.

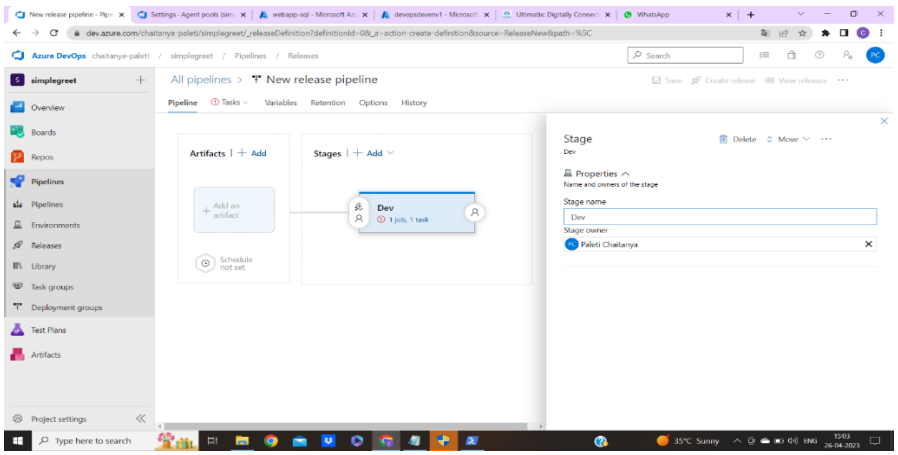
1. In your project, navigate to **Pipelines -> Releases**. This will open an empty list of Release Pipelines (in my case since this is the first time creating Release pipeline).



2. Click **New pipeline**. This will open a **Select a template** drawer to the right.

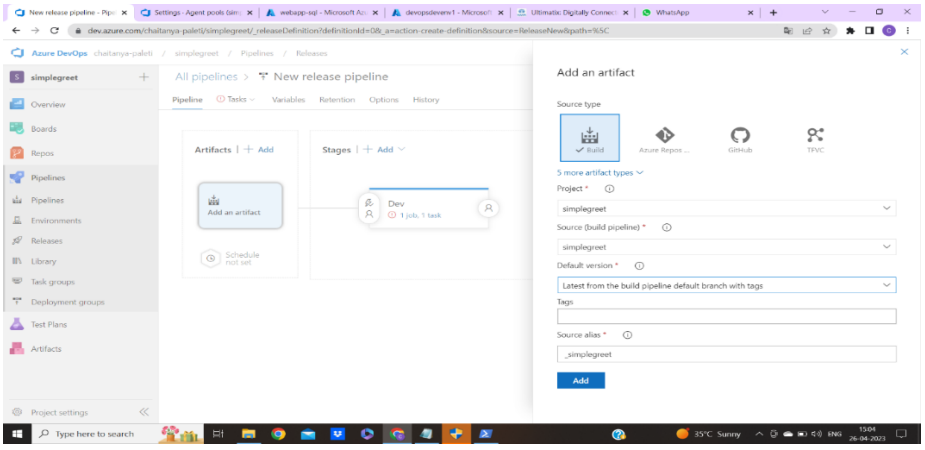


3. Choose **Azure App Service Deployment** and click **Apply**.

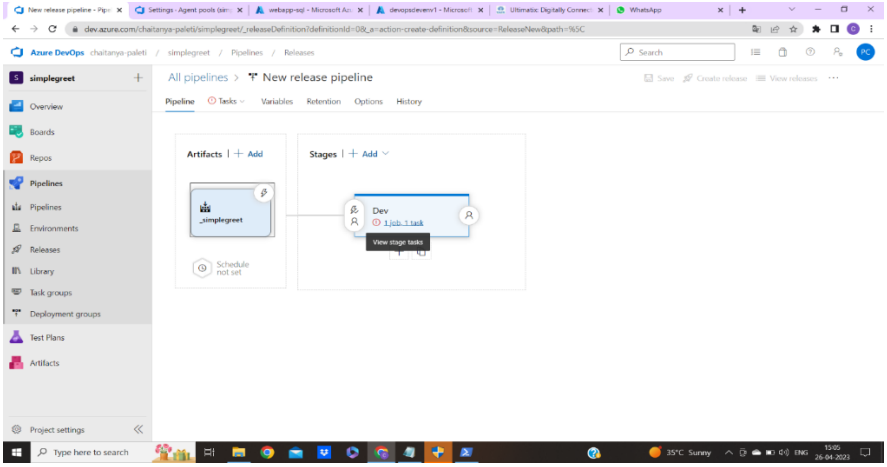


4. Name the stage “**Dev**”, then close this drawer with the “x”.
5. In the pipeline overview, under Artifacts, click **Add an artifact**.

- Select our Build Pipeline, for the **Default version** choose **Latest from the build pipeline default branch with** tags and click **Add**.

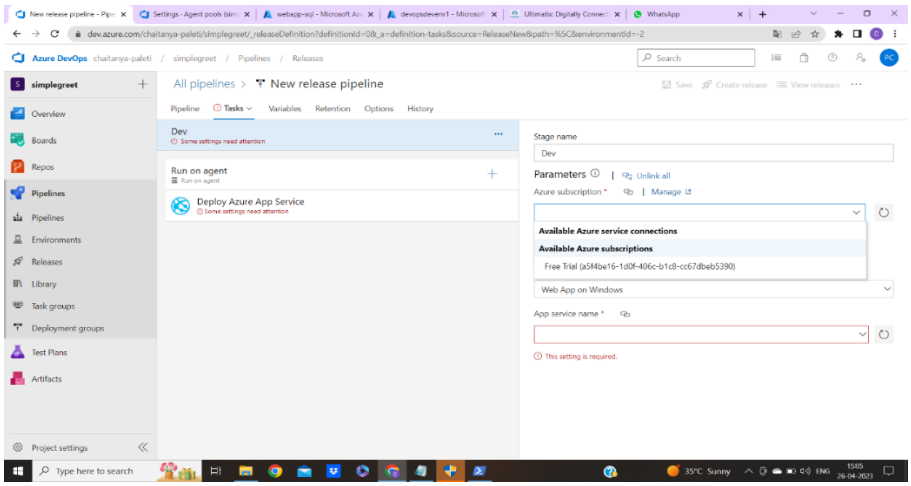


6. In the pipeline overview, under Stages, in the **Dev** environment, click the **task link**. This opens the actual steps for automating the deployment to this stage/environment.

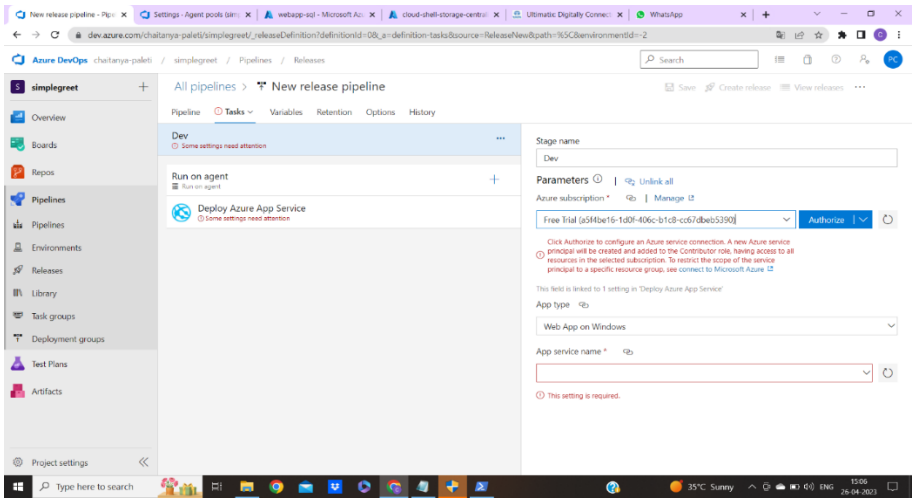


7. Here we will now fill in the details of how we can connect to our Azure resources.

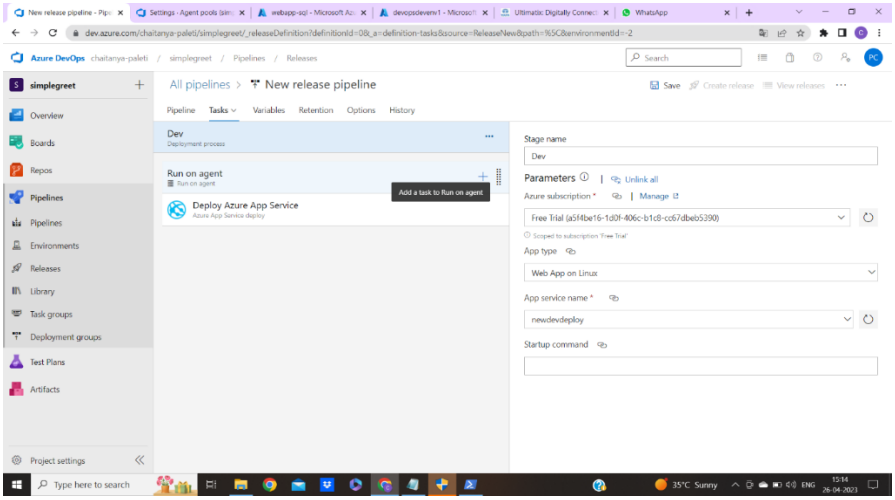
- Under **Parameters, Azure Subscription**, choose your subscription.



8. Click **Authorize**. Azure DevOps then tries to automatically create a Service Principle that our automation can use to authenticate against the Azure API, so that we can create resources and deploy apps.

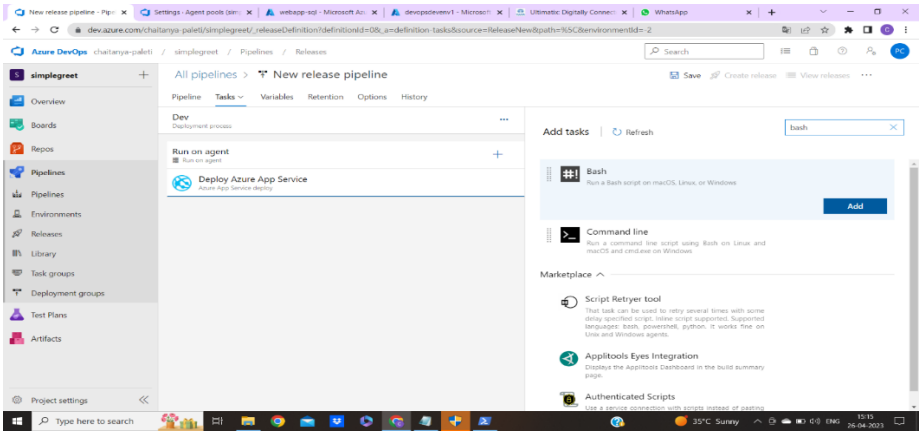


9. For App type choose **Web App on Linux** and for App Service Name click the dropdown and you should see the app service we created in the beginning – select it.
   Here **newdevdeploy** for **Dev** resource group **devopsdevenv1**.
10. Click the **+** button above the task list.



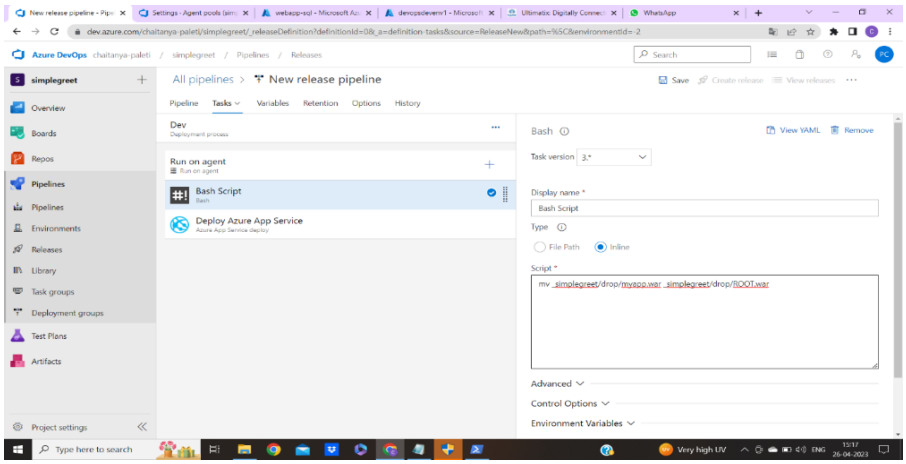
- This opens a drawer to the right containing many available tasks, ranging from simple operations like copying files to complex integrations like creating VM's in VMWare or deploying to third party services like the Apple or Google App Stores.

11. In the **Add tasks** drawer search box, type "Bash", select the **Bash** task and click **Add**.

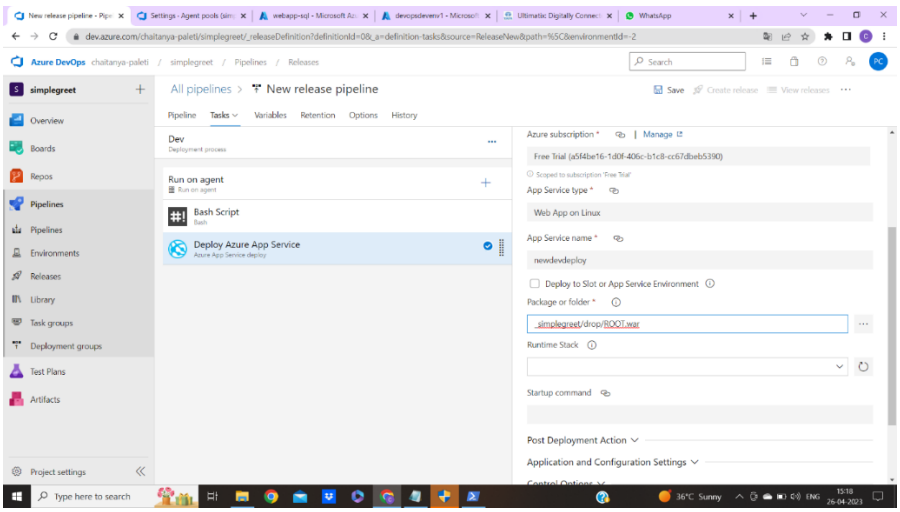


- As we are creating a Java application that is running in Tomcat, the easiest option to run our app as the root app is to rename its **.war** file to **ROOT.war**. We will do so with another step that will always be executed immediately before the deploy task that was already created by the template.

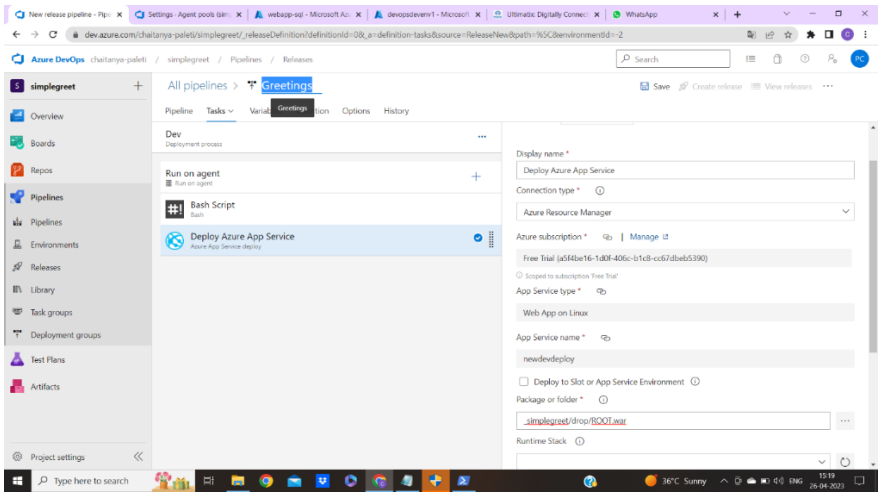
12. Drag the Bash task above the **Deploy Azure App Service** task.
13. In the Bash task, check the **Inline** radio button and enter the following command: **mv_simplegreet/drop/myapp.war _simplegreet/drop/ROOT.war**



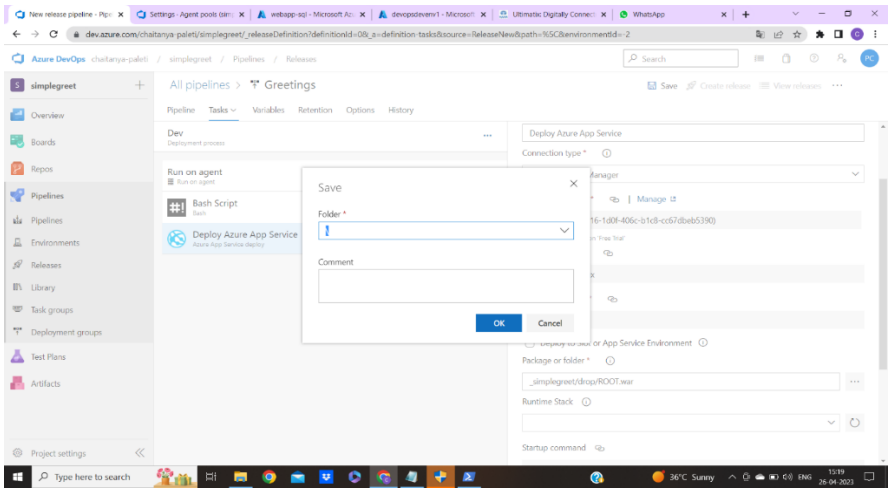
14. In the Deploy Azure App Service task, for **Package or folder** enter **_simplegreet/drop/ROOT.war**



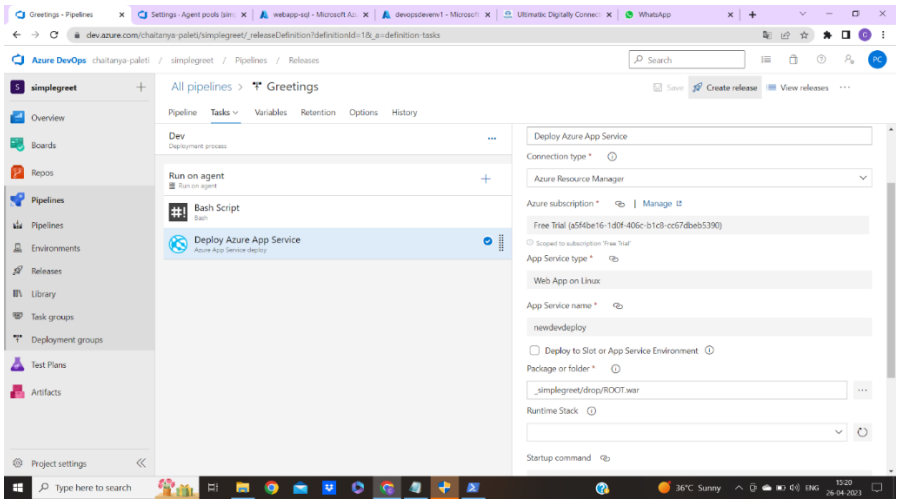
15. Rename the pipeline to **“Greetings”**.



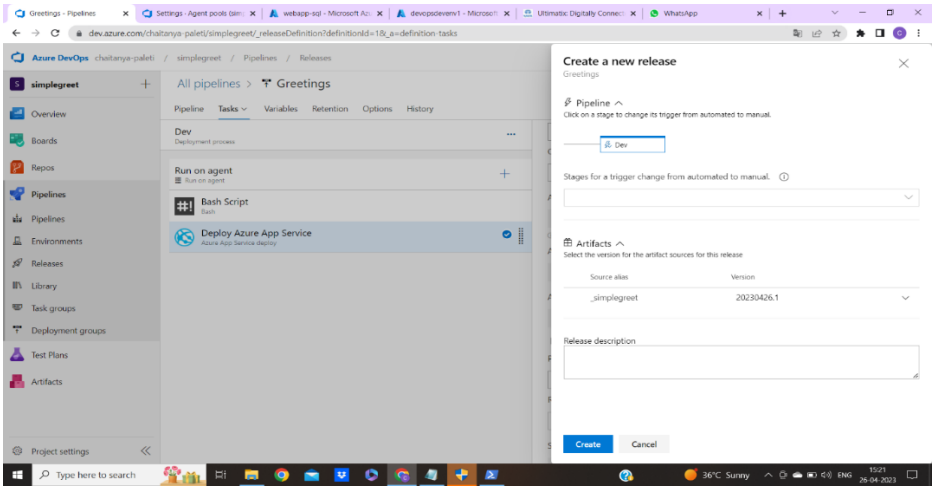
16. Click **Save**. In the Save dialog, enter a command if you want and click **OK**.

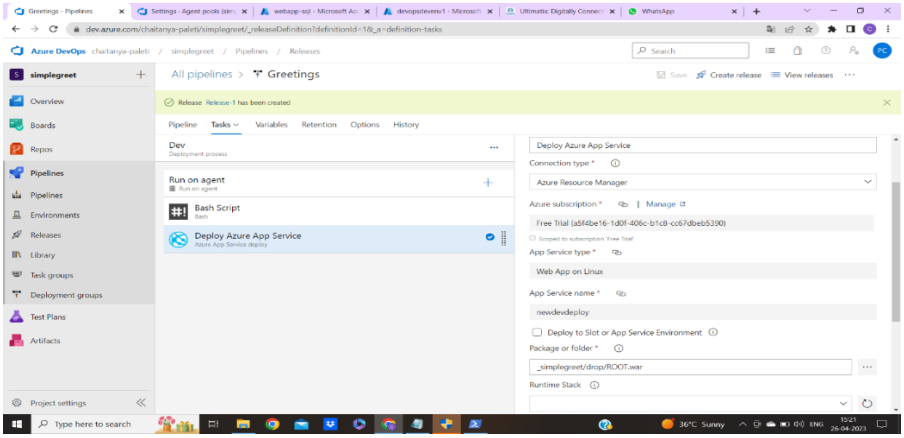


17. Next to Save button, click **Create Release**.

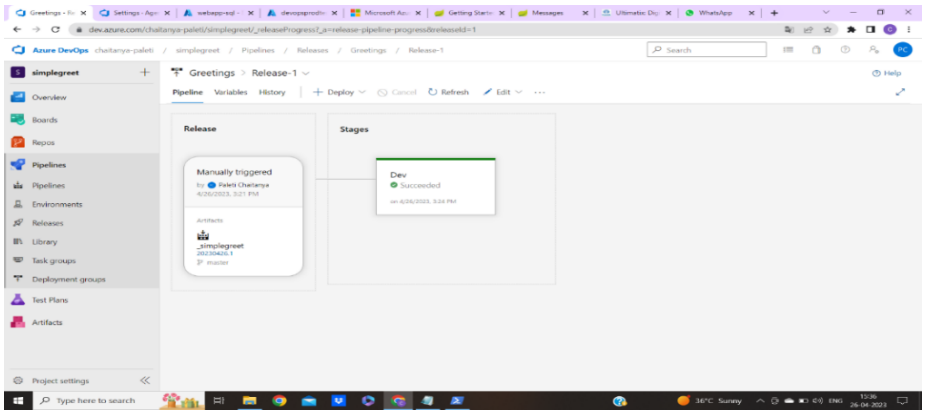


18. In the drawer coming up, click **Create**.

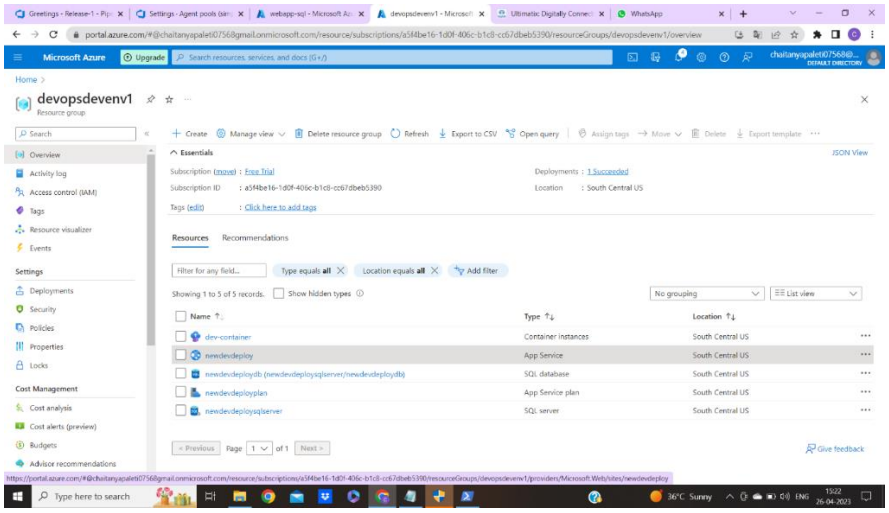




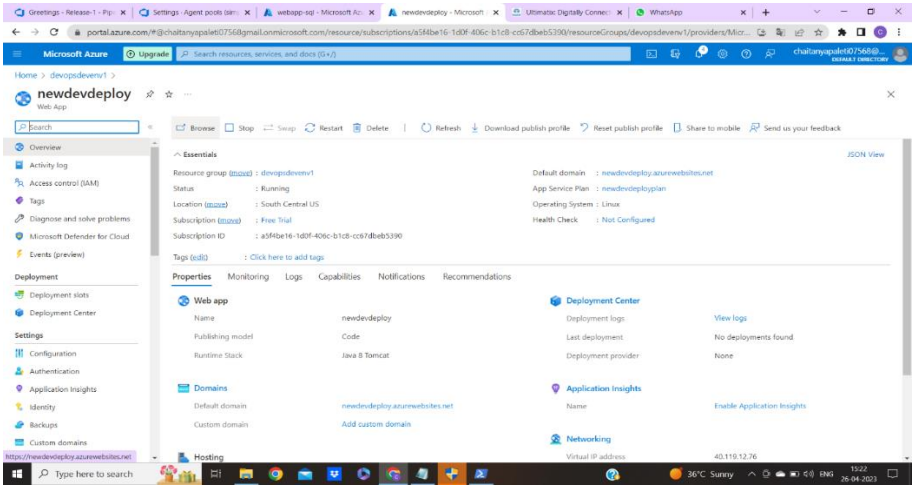
- In contrast to Build Pipelines, in Release Pipelines we are not directly queueing the tasks to be performed immediately, instead we create a release, which keeps the exact versions of all artifacts in one place, so that this exact package can be deployed to the different stages at any time.
- By default, the single stage we have defined so far will be deployed automatically, but we can choose to not have any deployments be triggered automatically.
- The following screenshot says that your deployment in Dev is Succeeded.



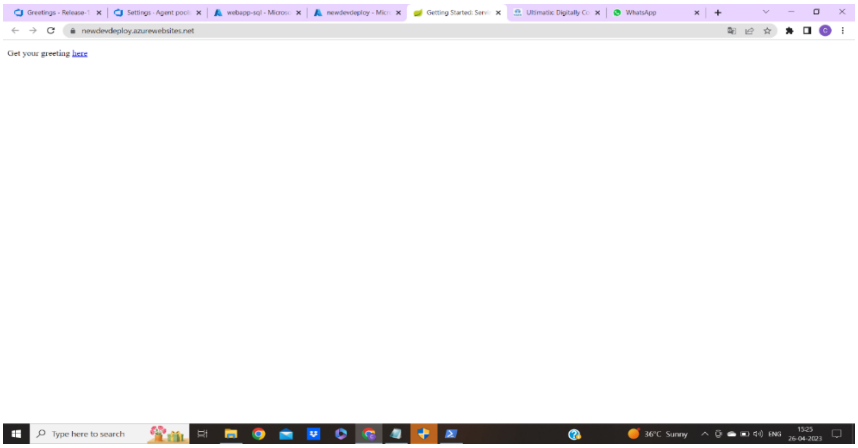
19. Now verify our deployment so far using the Azure portal.
    - Navigate to https://portal.azure.com, in the menu on the left, choose Resource Groups and select your dev resource group (**devopsdevenv1**).



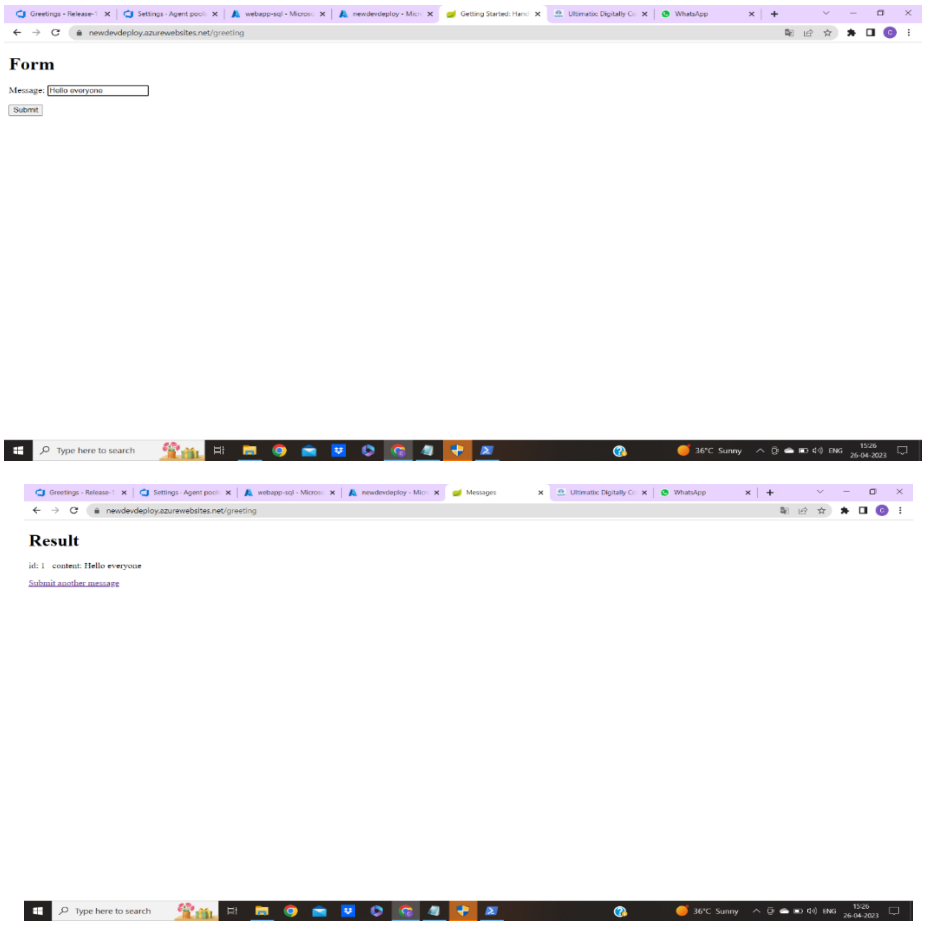
- In the resource group you should see an app service plan, an app service and Azure SQL Server resources. Click on the **app service** and then click the **Browse** button.



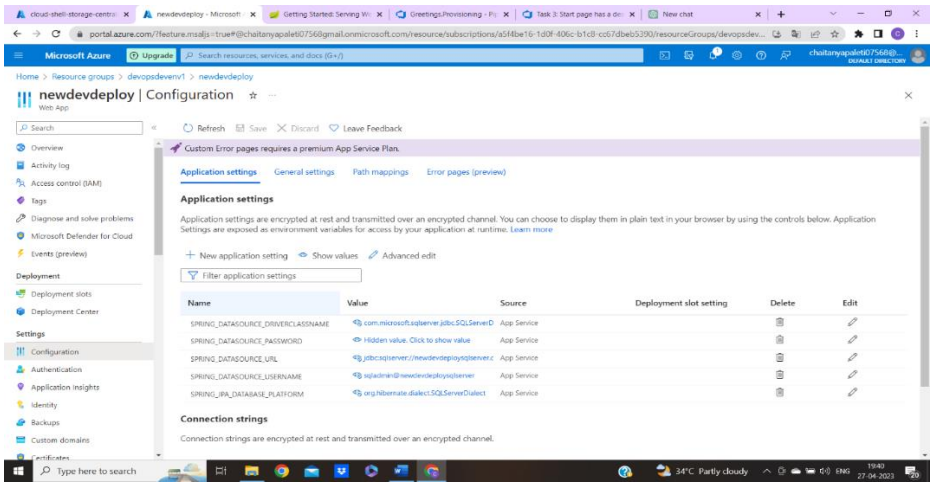
- This will take you to the web site you just deployed at [https://<some-name>.azurewebsites.net](). In this case its [https://newdevdepl oy.azurewebsites.net](). It will take a while for the app to come up at initial start, but eventually you should see an app inviting you to create a few greetings.



- Once you click the **here**, like in the above screenshot, you will get following output pages.



20. Back in Azure portal, go to **Application Settings** part of your app service, scroll down to application Settings, and click **Show Values.** This will show you the environment specific values for this stage.



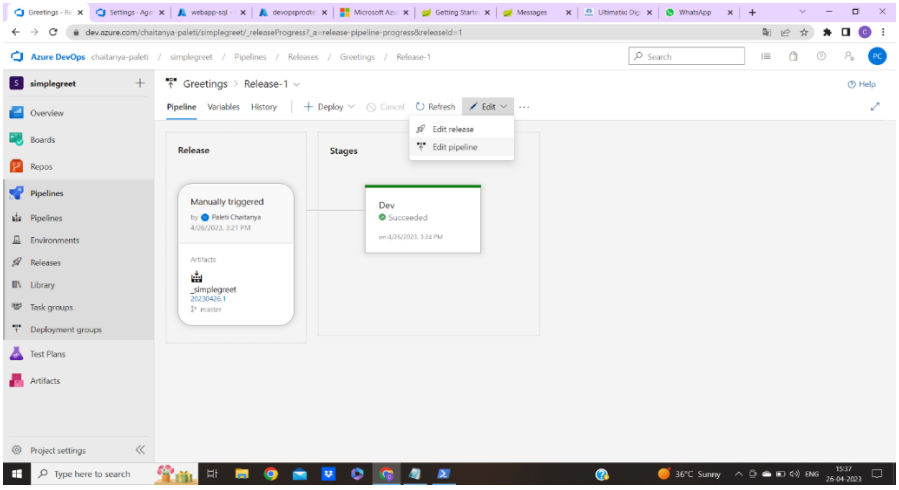
- These settings make sure that the web app code is using the correct JDBC driver with the correct connection string and a few more settings. These settings are passed to the application as environment variables, and fortunately Spring Boot allows overriding its application settings via these environment variables.

- The values itself come directly from the ARM template we deployed right in the beginning.

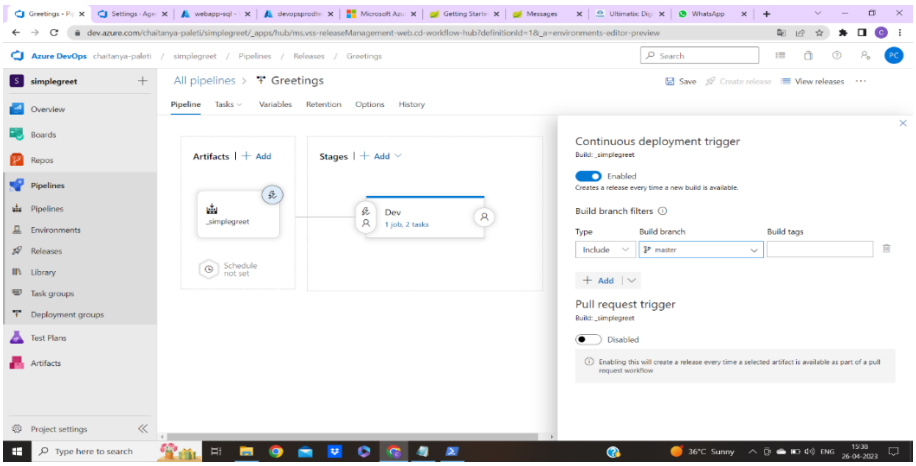
Exercise - 7: Continuously deliver to Production

- The first thing to “continuous” is that the transition from CI (building the product) to CD (delivering the product) needs to happen automatically.
- For Continuous Delivery it is not necessary to actually deploy every change in our code to production – but at least every change that is pushed to our master branch should be picked up as quickly as possible to be deployed and tested in an environment that is as close to production as possible.
- To actually deploy to production, we will extend our pipeline by adding a Prod stage. Deployment to that stage will be triggered automatically as well, but each release will need to be approved, to prevent changes rolling uncontrolled into production.

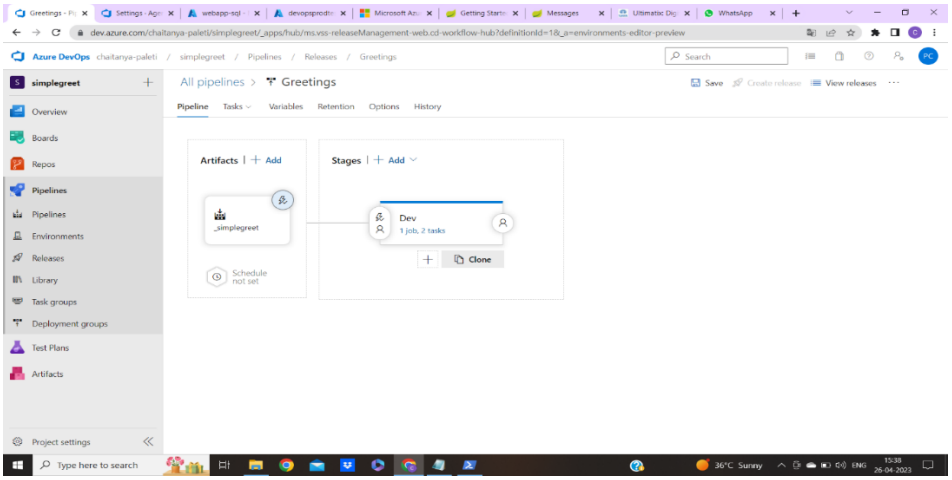
1. Navigate to your release pipeline by clicking **Edit pipeline**.



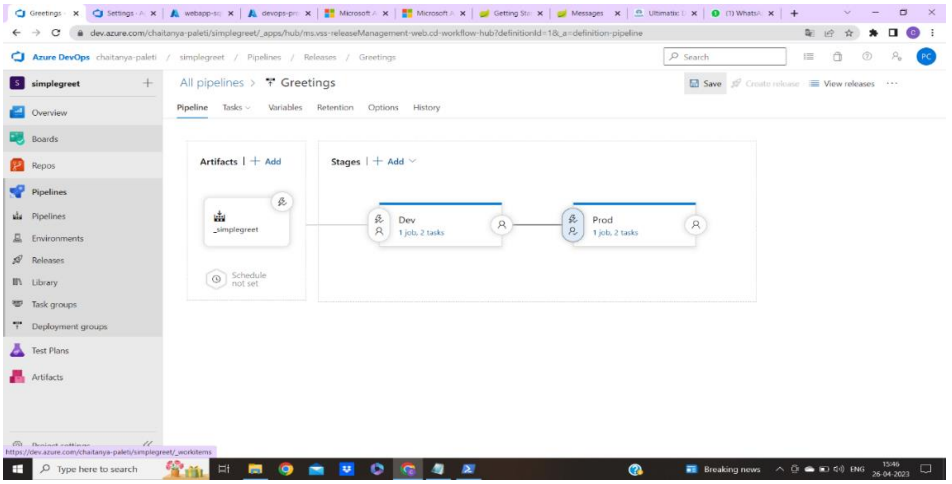
2. In the **Continuous deployment trigger** drawer, set the trigger to **Enabled**, then under **Build branch filters** click **Add** and add an **Include** filter for the **master** branch. This way any topic/feature or other not-so-stable branches will not enter our release definition.

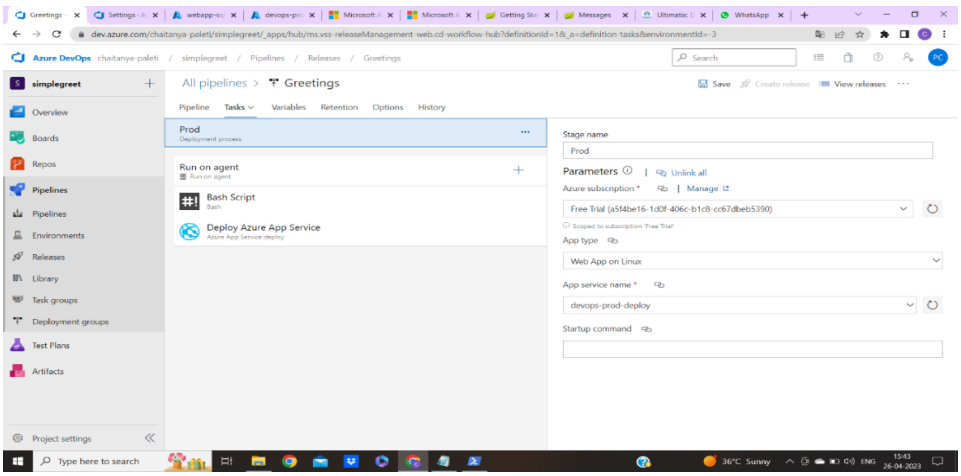


3. Clone the **Dev** Stage by clicking on **Clone**.

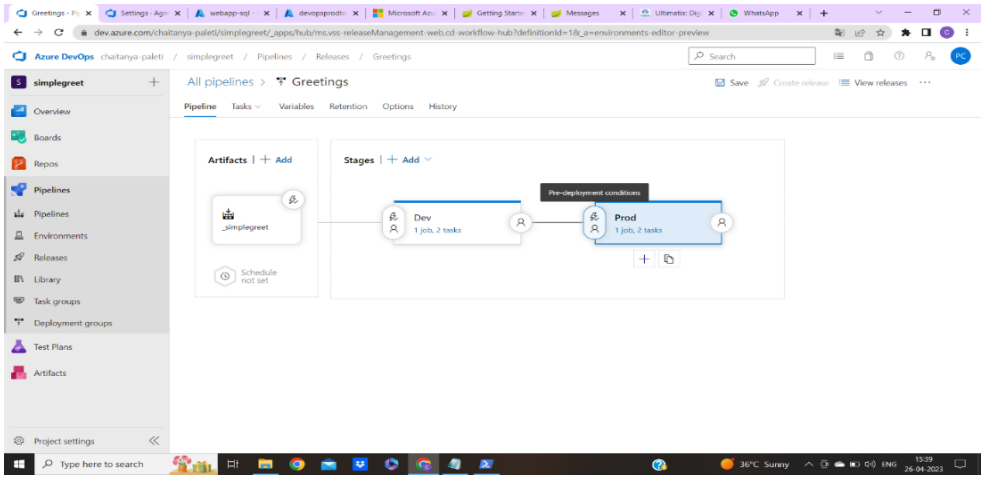


4. Click the cloned stage, name it **Prod** and in the Deployment process section choose the production resource group (**devopsprod-env**) and app service (**devops-prod-deploy**).

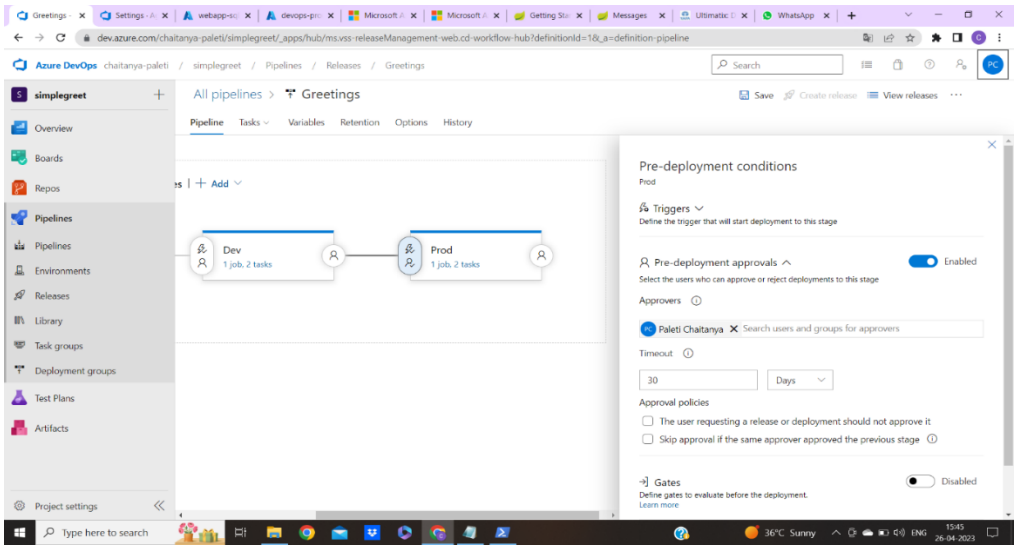




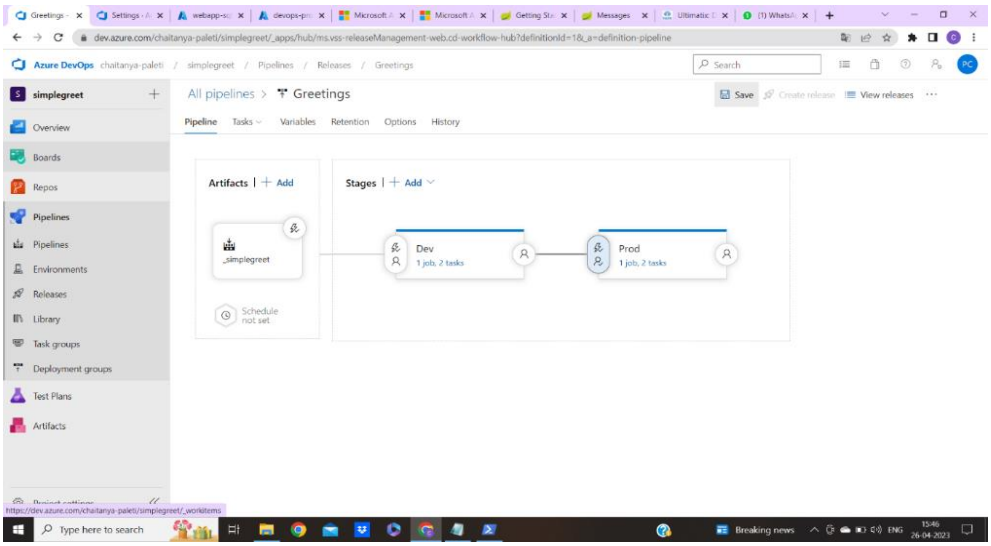
5. Click the **Pre-deployment conditions** for the production stage (Prod).



6. In the **Pre-deployment conditions** drawer, set **Pre-deployment approvals** to enabled and add your account name (here **Paleti Chaitanya**) as approver. The approvers of a stage will be notified via email when a new approval is pending



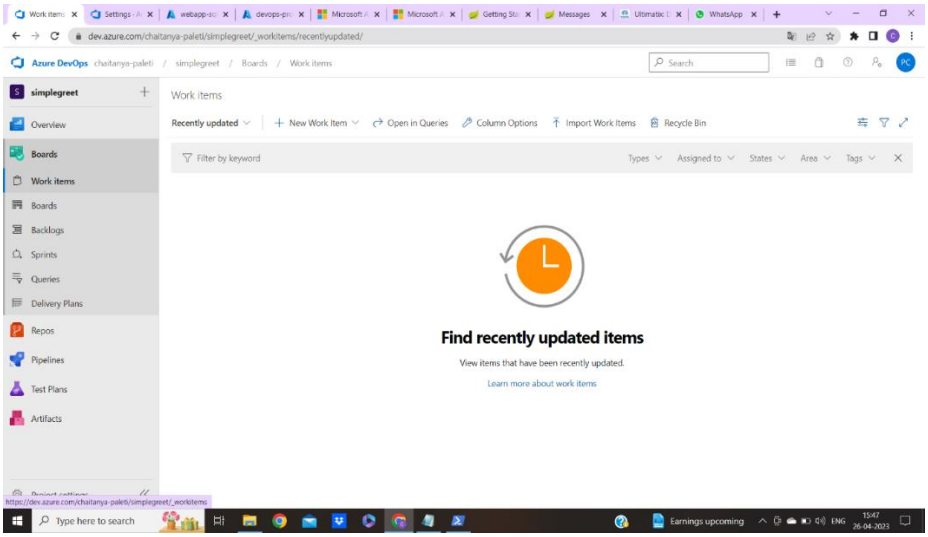
7. Click **Save** to save our changes.



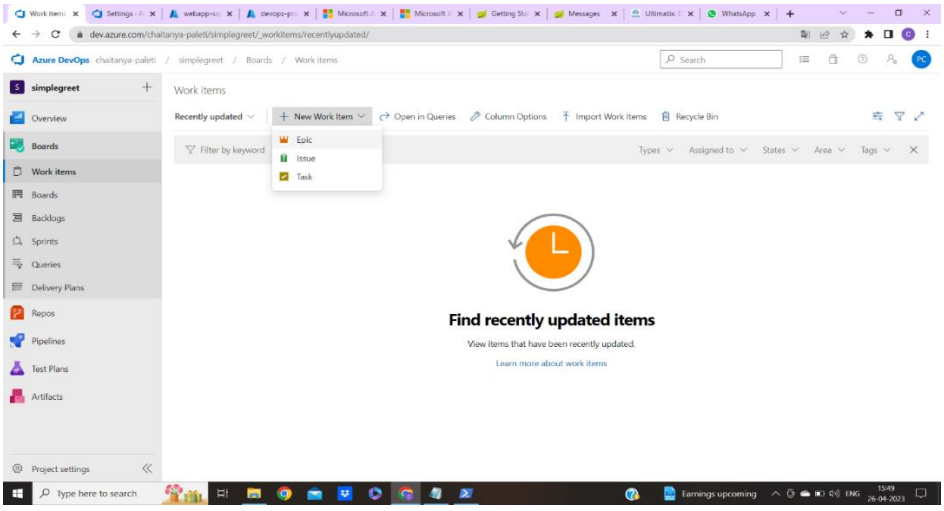
Exercise 8 : Track functional changes throughout the CI/CD Pipelines

- Azure DevOps can do much more than just CI/CD, starting from organizing work in Backlogs and Boards. Typically, in Azure DevOps any change in our software is planned or prioritized through work items (like User Stories or Product Backlog Items).
- Additionally, work items are fully integrated to code repositories, build and release and more, so that all activities for a change can be tracked, even through branches and pull requests in version control. Work items can even drive the development workflow.

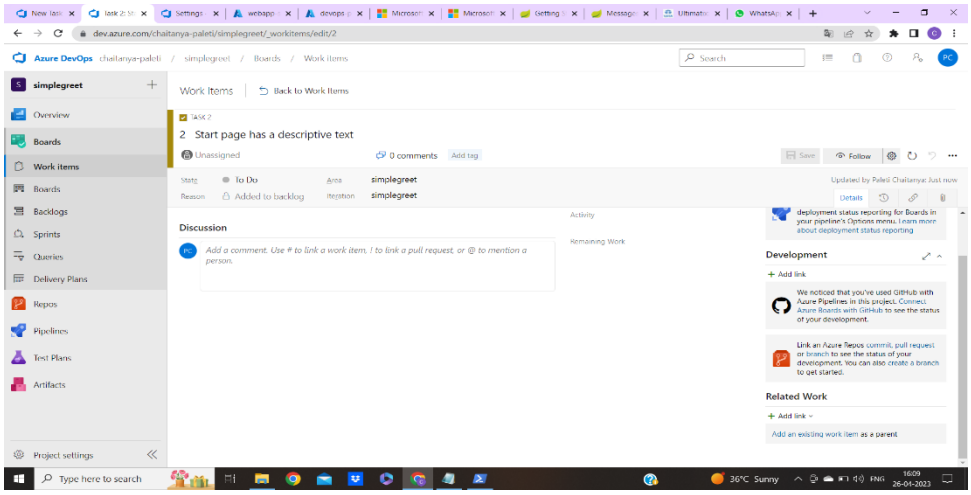
1. In the Azure DevOps, select **Azure Boards**.



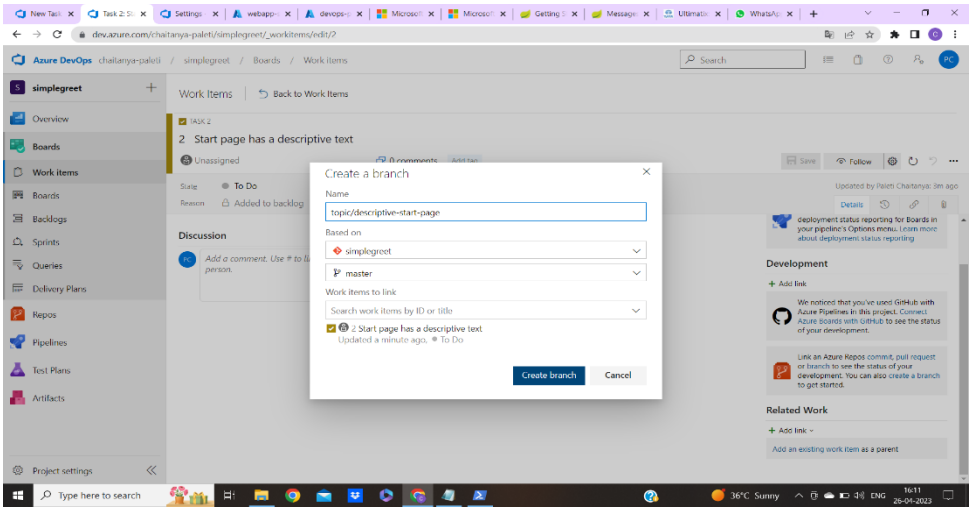
2. Click **+New Work** Item and select the item to track our code change.
   Here I have chosen **Task**.



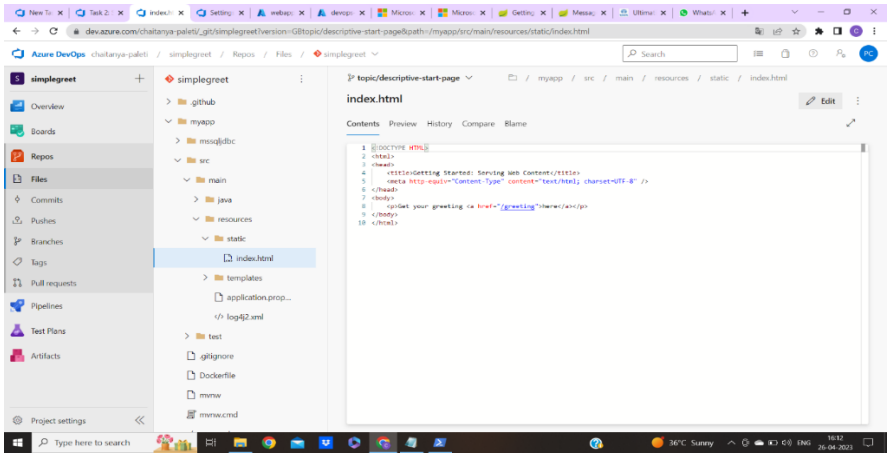
3. In the Task form, enter the title "**Start page has a descriptive text**" and click **Save**
4. In the **Development** section of the **Task**, click **Create a new branch**.



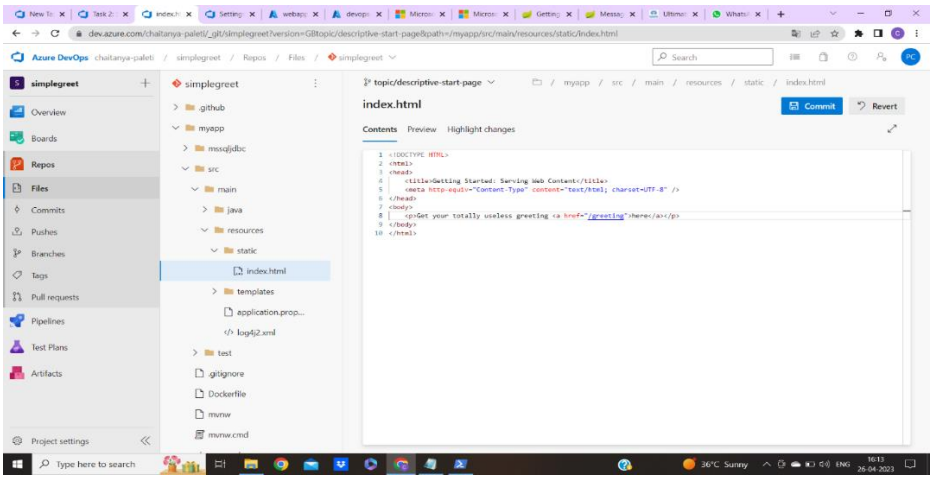
5. In the **Create a branch dialog**, enter **“topic/descriptive-start-page”** as the name of the branch and select the **master** branch of our repository, then click **Create branch**.



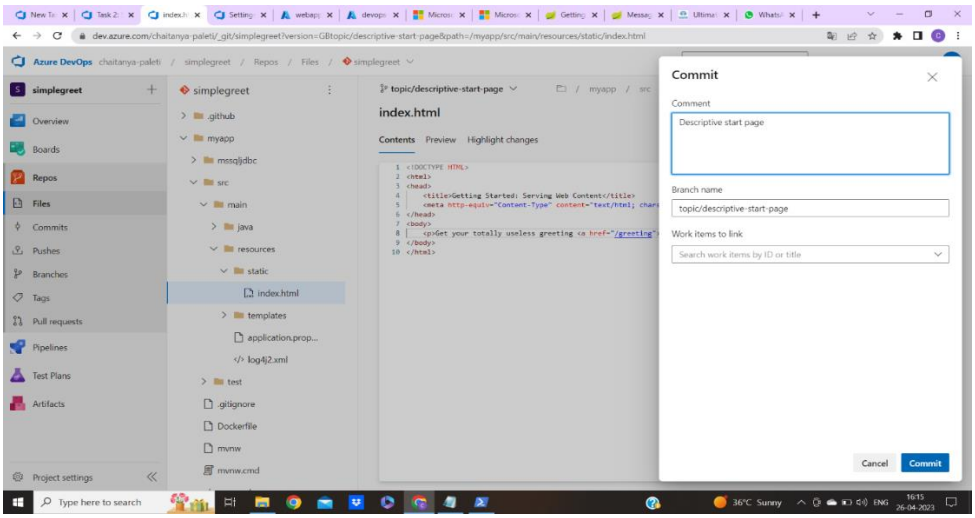
6. The new branch is automatically linked to the work item, so that we can track the code changes we will implement in that branch to the work item.
    - In the mean time, Azure DevOps has automatically opened the new branch for you in the browser, so we can immediately start our work.



7. Navigate to **myapp/src/main/resources/static/index.html** and click **Edit.** This enables us to edit our code.
8. Locate the string "**Get your greeting**" in the html code and change it to "**Get your totally useless greeting**" or any string you wish.



9. Click **Commit**. In the Commit dialog enter "**Descriptive start page**" as comment and click **Commit.**

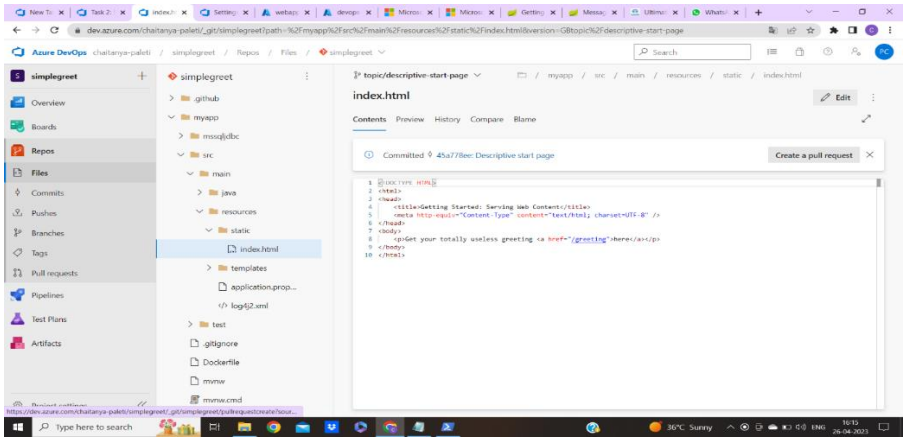


- We could make more changes in this way or do some real development on our development machine, pushing intermediary commits to our topic branch. All of these changes might trigger a CI builds, but will not be automatically deployed, because we filtered our release definition to only include the master branch.

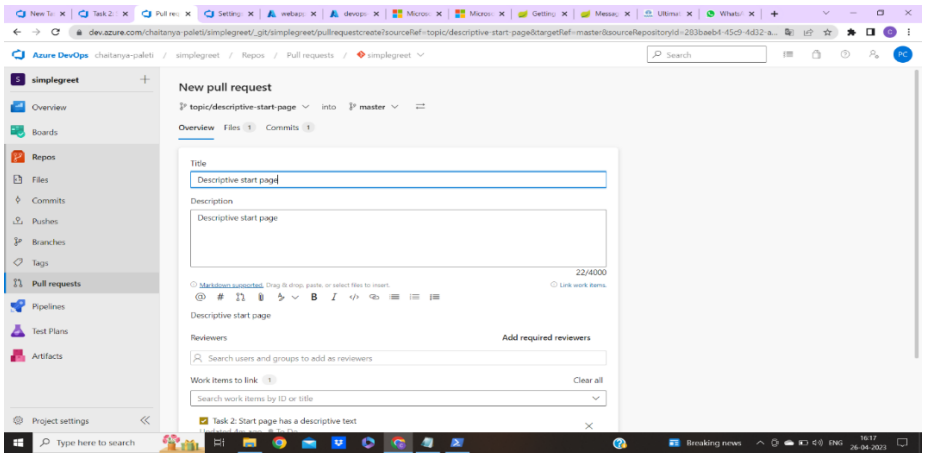
10. Thus, to get our code with the new start page deployed, we need to merge it into the master branch. For this we use **pull request**.
    - A pull request automatically documents all the changes that were done for a topic/feature and provides a great workflow for reviewing changes.

    - As we committed some code recently, Azure DevOps conveniently offers you to create a pull request already in the current view.

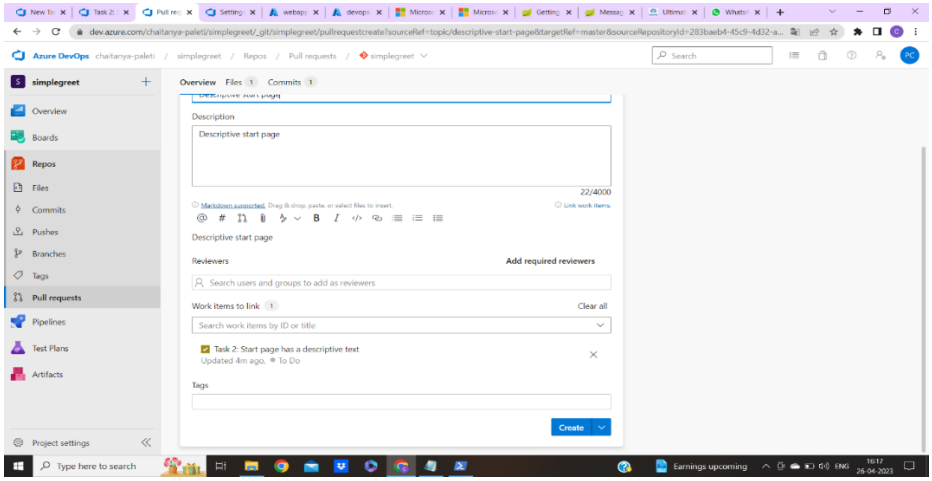
- Click **Create a pull request.** This takes you to a pull request page summarizing all info about our efforts to implement the user story to make the start page more descriptive.



- You could enter reviewers here and start a conversation using direct code comments about the quality of your code.

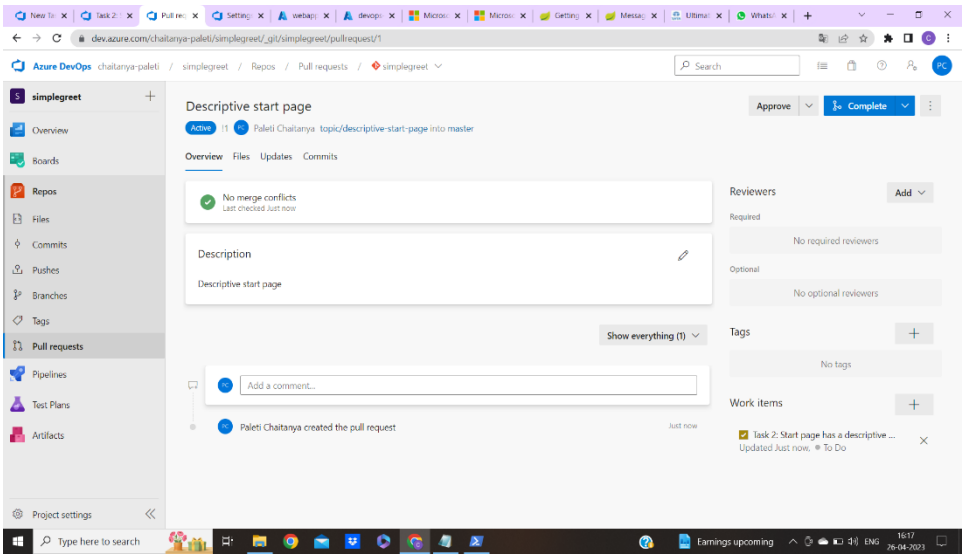


11. For now just accept the defaults, click **Create**.

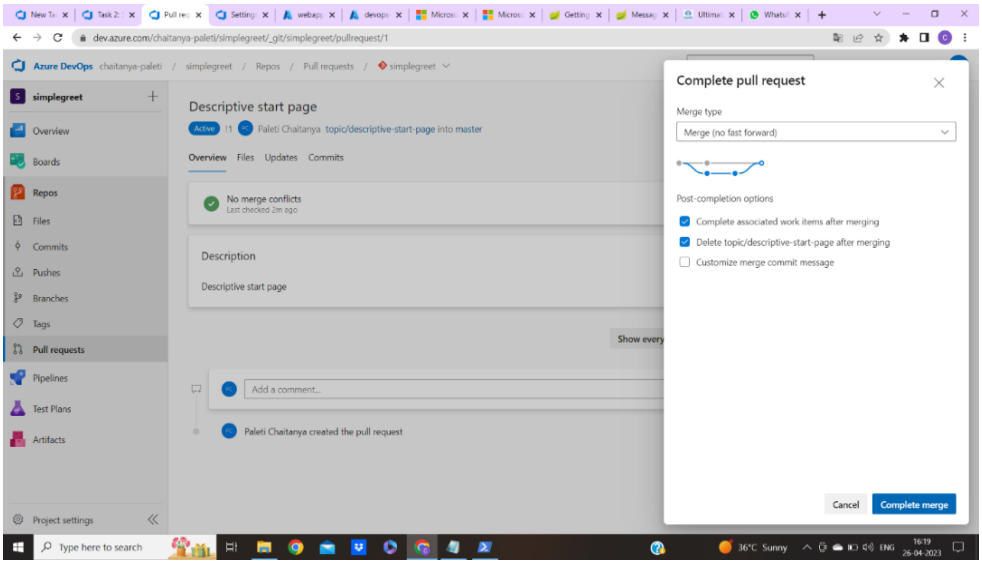


- This will create the pull request and publish it to all reviewers you might have entered. As I don't have any reviewers, I can directly go on to finish the pull request.

12. In the pull request, click **Complete**.

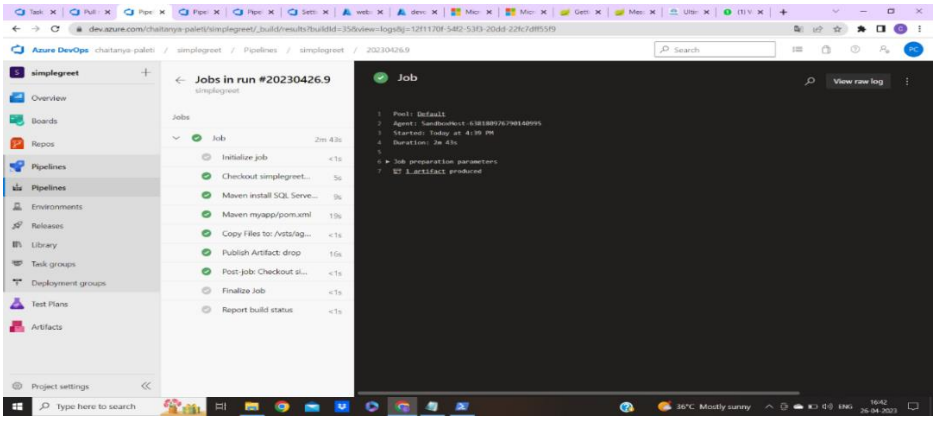


13. In the Complete pull request dialog, click **Complete merge**.

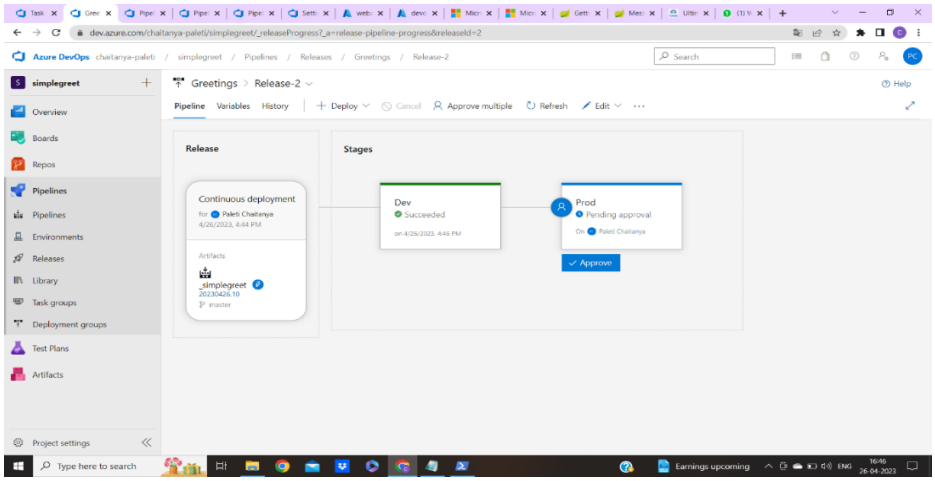


14. This will initiate a series of processes being carried out automatically:

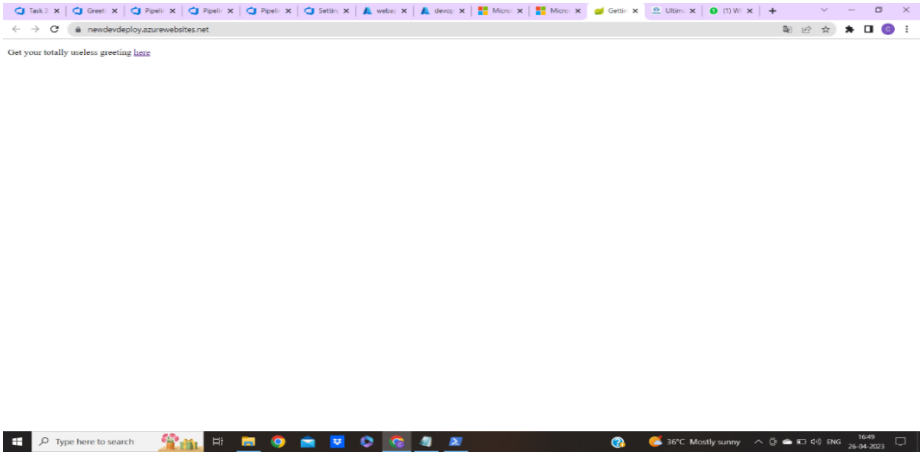
- The topic branch with all its code changes will be merged into the **master** branch.
- The commit to the **master** branch will trigger a CI build, building a new version of our application (a new **myapp.war** file).



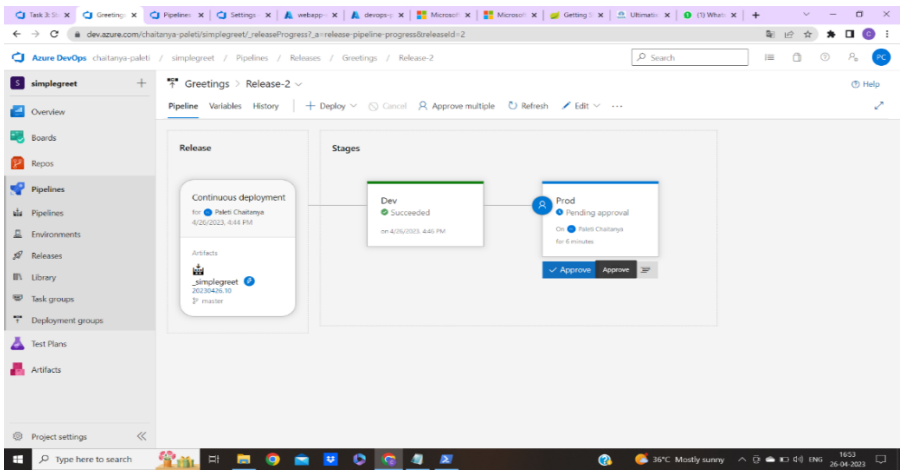
- As the CI build was for the master branch, if the build was successful, a new release will automatically be created in the release definition.
- The new release will automatically pick up application package (our myapp.war file) and deploy it to the Dev stage.
- In case the deployment to Dev went fine, the approval request for Prod stage will be sent out.



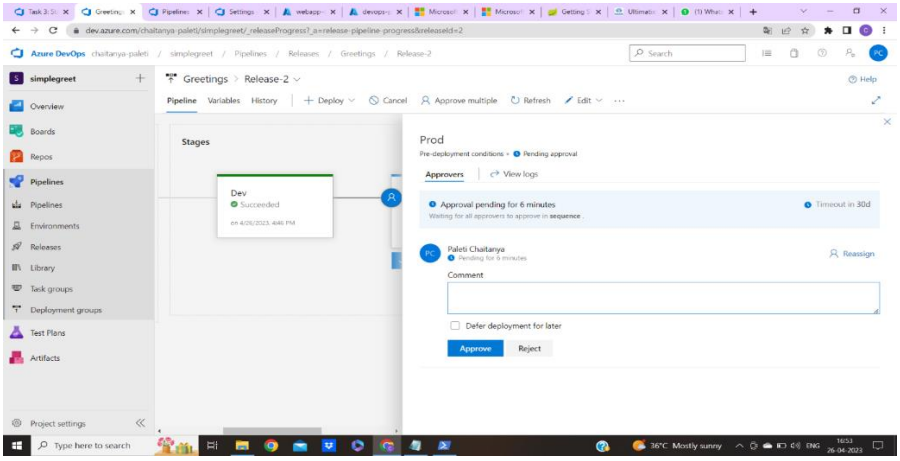
15. After the deployment to Dev finishes, check the result in your Dev application, by navigating to https://<dev-app-service-name>.azurewebsites.net.
    - Here https://newdevdeploy.azurewebistes.net.

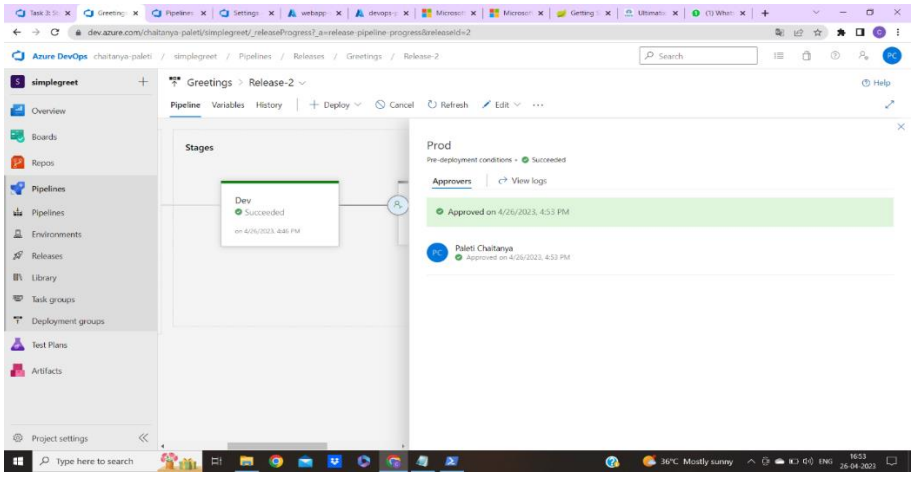


16. If the Dev version is ok, you can approve the pending Prod deployment.

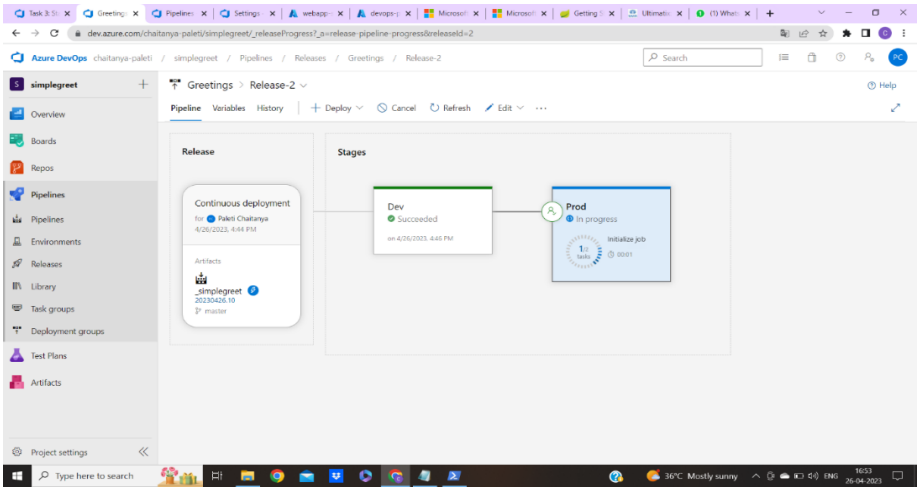


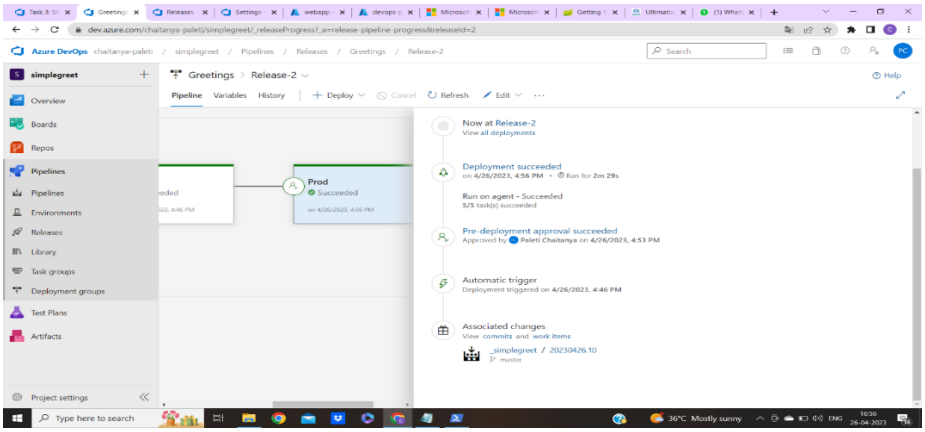
- In the drawer, click **Approve**. Your approval will be documented, and the deployment will start.



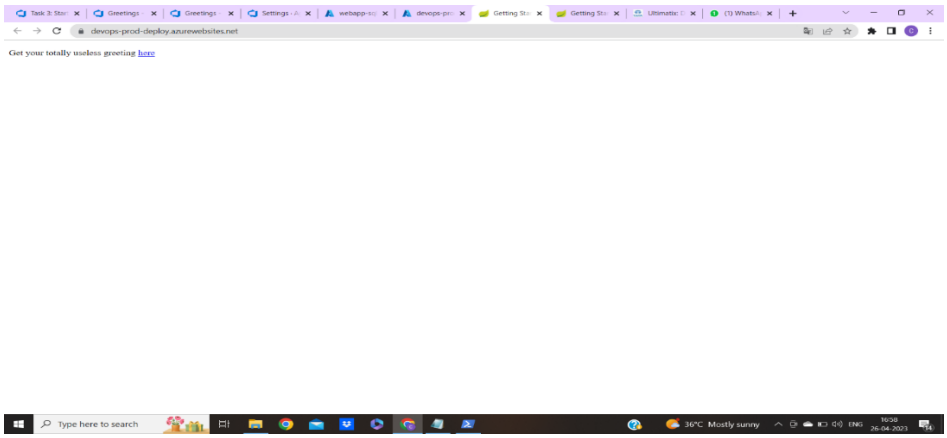


- Once the deployment to **Prod** is approved, the new package will finally be deployed to production.





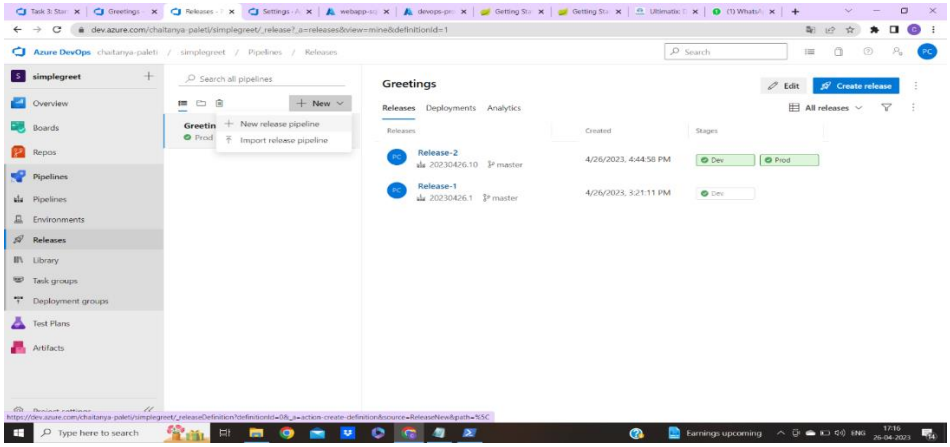
17. Once deployment finishes, navigate to your production application at https://<prod-app-service-name>.azurewebsites.net.
    - Here it is https://devops-prod-deploy.azurewebsite.net.
    - After a bit of initialization time, you should see our new start page there.



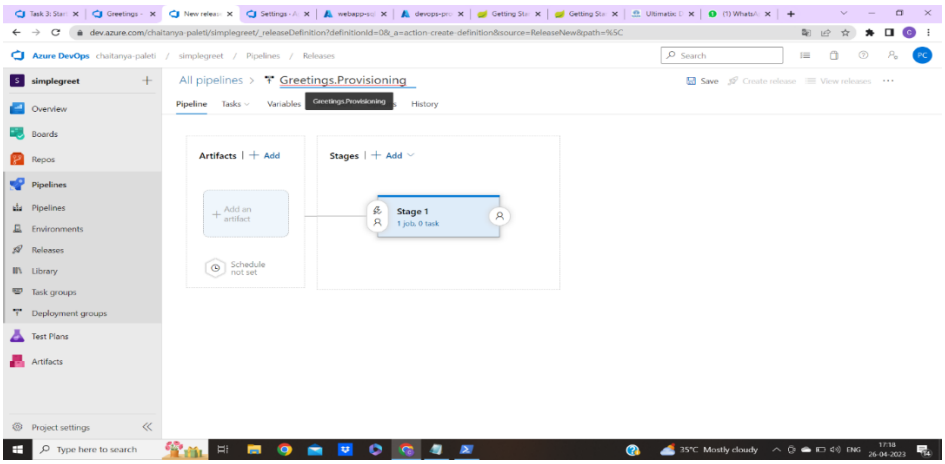
- This concludes the continuously delivering software with Azure DevOps.

Extra Challenge: Environment Provisioning Pipeline

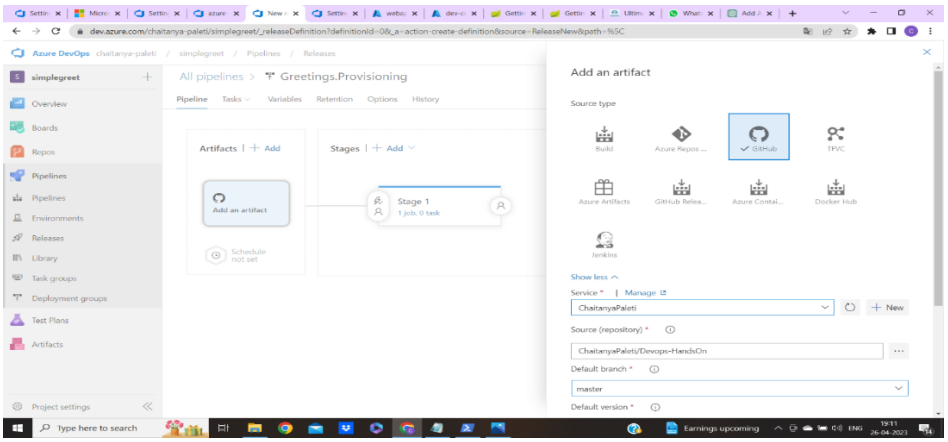
1. Create an additional release pipeline with Empty template. Click on New **Release Pipeline**.

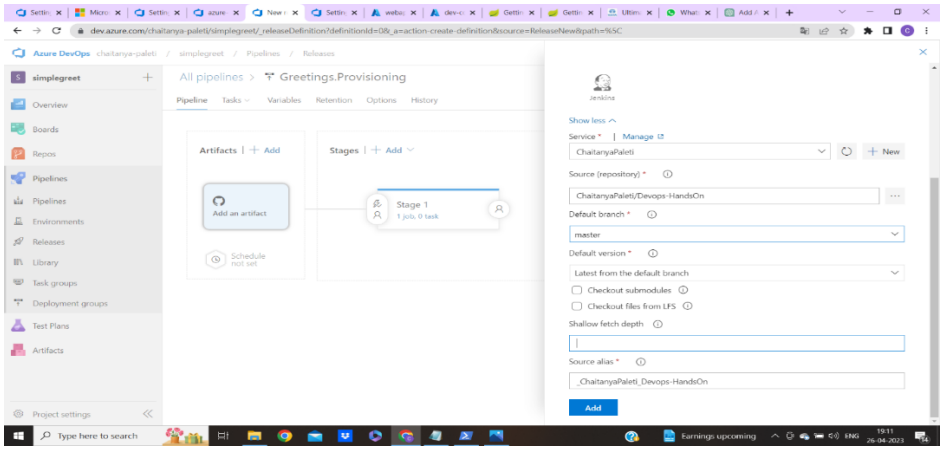


2. Name it as "**Greetings.Provisioning**".

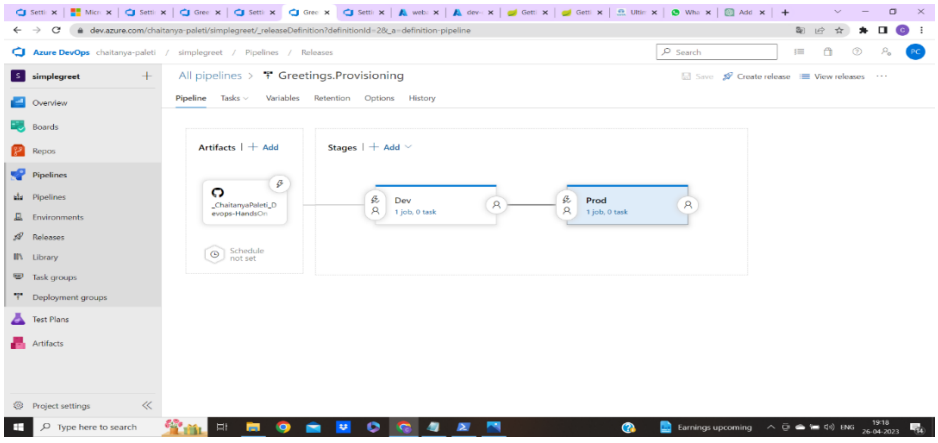


3. Use our git repository as the artifact for the pipeline. Click **Add**.

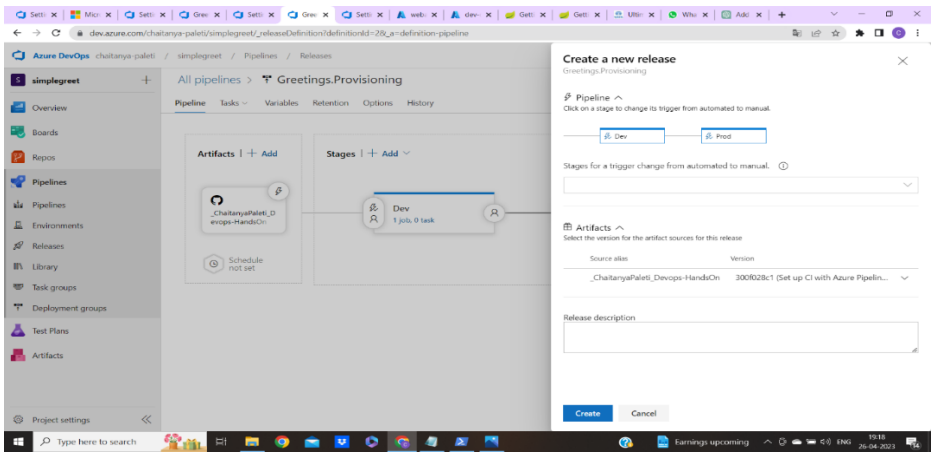


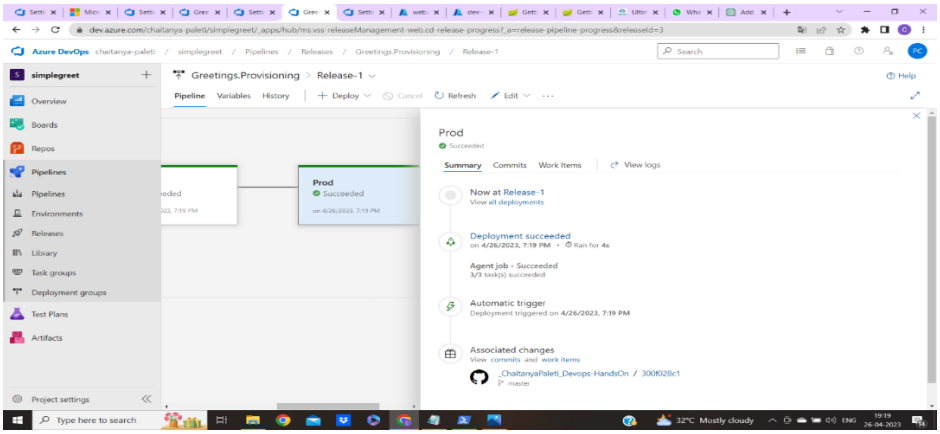


4. Create stages in the environment that mirror the stages of our product pipeline. Here **Dev** and **Prod** are the stages. Then click on **Create Release**.



5. Create a release for the pipeline and apply it to all environments again. Click on **Create**.





- The system will detect that there are no changes and should leave the existing environments untouched.