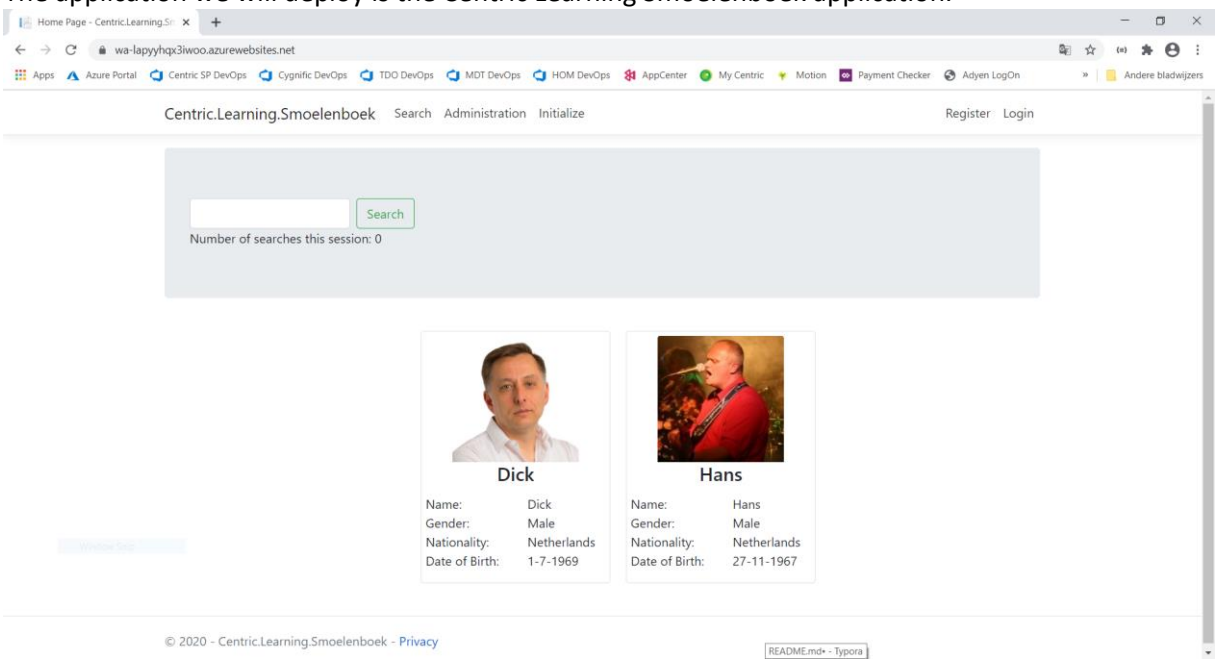
Azure DevOps- Continuous Deployment

Introduction

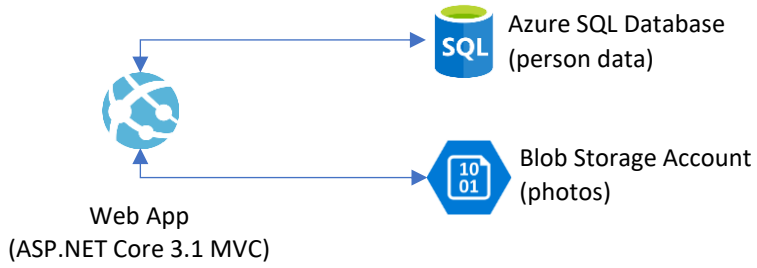
This lab is about Continuous Deployment with Azure DevOps pipelines. In this lab we will:

- create an ARM template for deploying a web application to Azure.
- quickly setup an automated build pipeline in Azure DevOps.
- setup a release pipeline for the web application.

The application we will deploy is the Centric Learning Smoelenboek application.



The components that this application will run on in Azure are as follows:



Pre-requisites

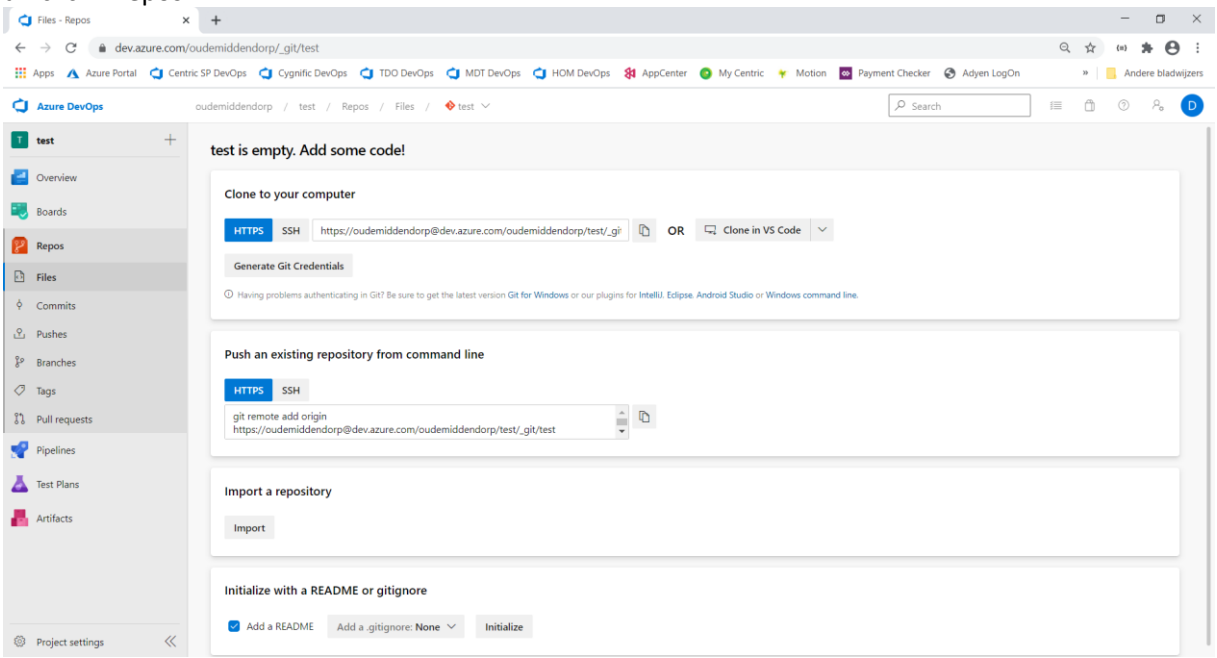
To follow this lab you will need the following:

- Access to two Azure Resource Groups that you own (one for testing and one for production)
  This is already set up for you because you have registered for the workshop
- Visual Studio Code
  Get Visual Studio Code for free from https://code.visualstudio.com/.
  Install "Azure Resource Manager (ARM) tools" from
  https://marketplace.visualstudio.com/items?itemName=msazurermtools.azurerm-vscode-tools.
- Azure CLI
  Get it from https://docs.microsoft.com/nl-nl/cli/azure/install-azure-cli-windows?view=azure-cli-latest
- Azure DevOps Organization owned by you (created with your centric.eu account)
  You have to create this yourself. See the document "Azure DevOps – Continuous Deployment – Lab Preparation".

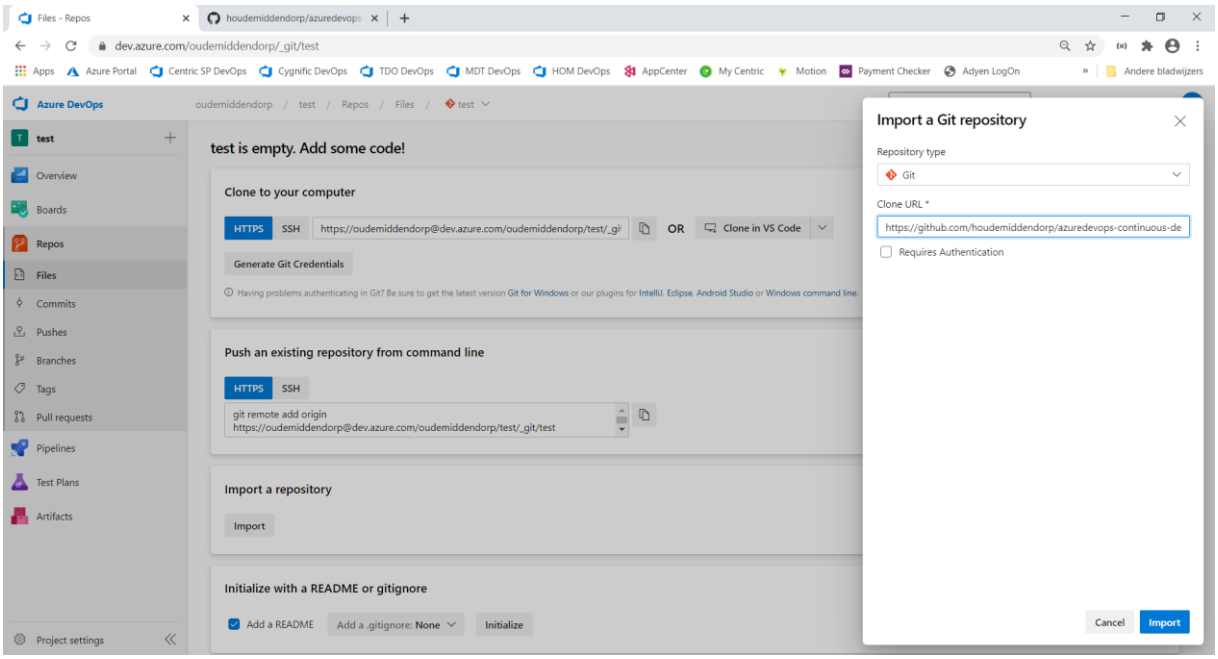
Last Step of Preparation

When you have set up your Azure DevOps project the last step of this work is to copy the code of the lab to your own repository.

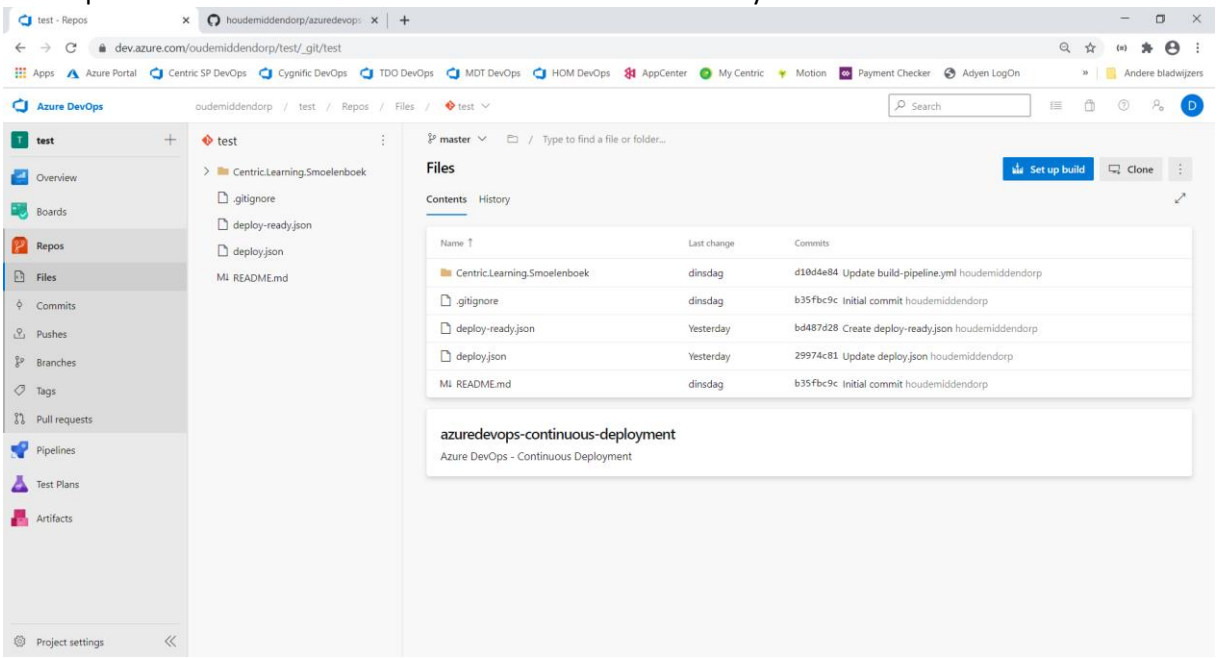
Navigate to your Azure DevOps project. On the left side of the screen you see a number of icons. Find an click "Repos".



And then click "Import" below "Import a repository". Enter the clone url of the lab in github (https://github.com/houdemiddendorp/azuredevops-continuous-deployment.git) in the "Clone URL" text box. And click "Import".



The import is started and after a few seconds it should be ready.



At this moment, all preparation for the lab and the workshop in Azure DevOps is done.

Setup a build pipeline

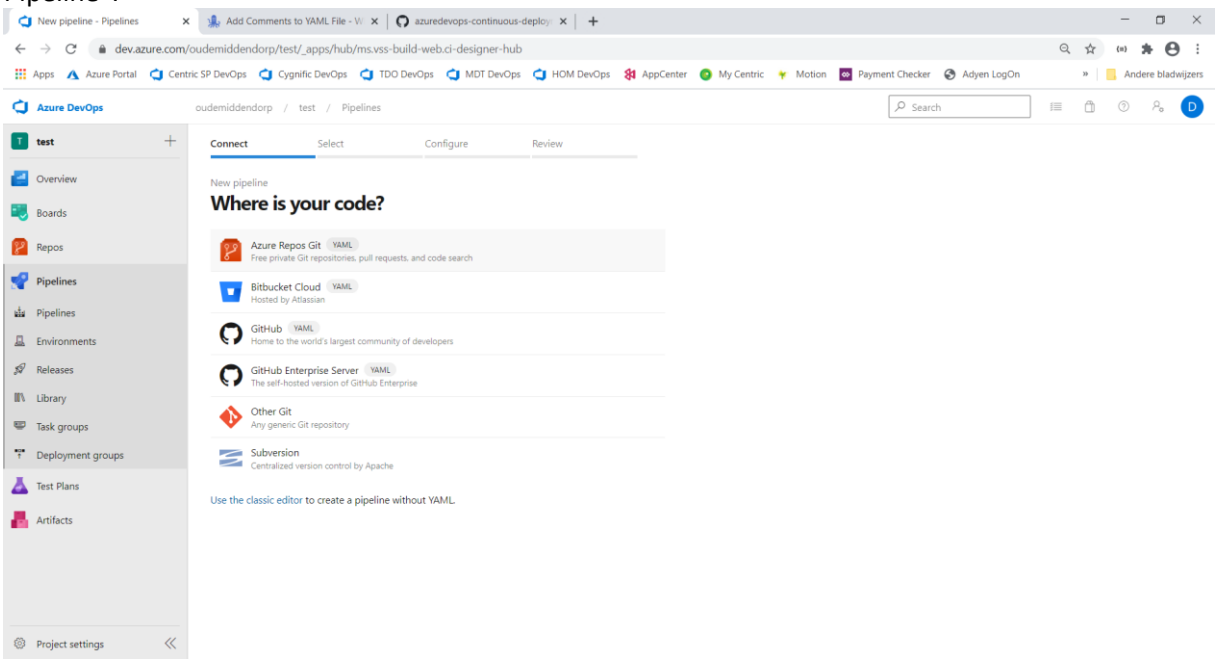
Because we want to implement a continuous deployment pipeline, we also need a continuous integration pipeline (or build pipeline). This pipeline will build the application and publish the artifacts that are necessary for the release pipeline to Azure DevOps Artifacts.

The build pipeline is defined in a yaml file which is already in the repository; the file “build-pipeline.yml” under “Centric.Learning.Smoelenboek”.

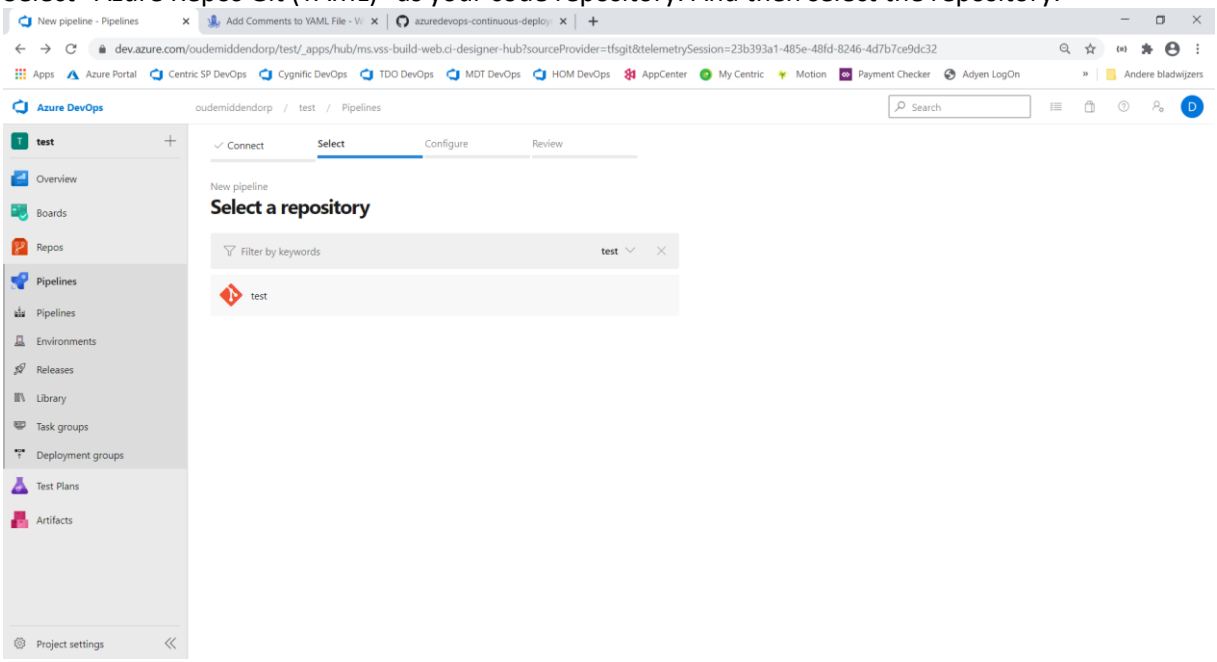
```
# the name of the build, avilable in the variable $(Build.BuildNumber)
# will be like "yyyy.mm.dd.revision"
name: $(date:yyyy).$(date:MM).$(date:dd)$(Rev:.r)

# the pipeline will automatically start when the master branch is changed
# i.e. when a pull request is done to the master branch
trigger:
- master

# the pipeline will run on a hosted Linux machine
pool:
  vmImage: 'ubuntu-latest'

# the variable $(solution) contains the name of the solution to be build (wildcard)
variables:
  solution: '**/Centric.Learning.Smoelenboek.csproj'
  system.debug: false

# now the steps for the build
steps:

# first step: build the app! using dotnet and publish to artifact staging
- task: DotNetCoreCLI@2
  displayName: 'Build Application and publish to artifact staging directory'
  inputs:
    command: 'publish'
    projects: '$(solution)'
    arguments: '-- output $(Build.ArtifactStagingDirectory)
                            /p:Version=$(Build.BuildNumber) /p:AssemblyVersion=$(Build.BuildNumber)'


# second step: copy the ARM template (is initially empty) to artifact staging
- task: CopyFiles@2
  displayName: 'Copy ARM template to artifact staging directory'
  inputs:
    Contents: 'deploy.json'
    TargetFolder: '$(Build.ArtifactStagingDirectory)'

# last step: publish artifact staging to artifact store
- task: PublishBuildArtifacts@1
  displayName: 'Pubish artifact staging to artifact store'
  inputs:
    PathtoPublish: '$(Build.ArtifactStagingDirectory)'
    ArtifactName: 'drop'
    publishLocation: 'Container'
```

We will not go into detail on this file, but you probably can read it and understand what is happening.

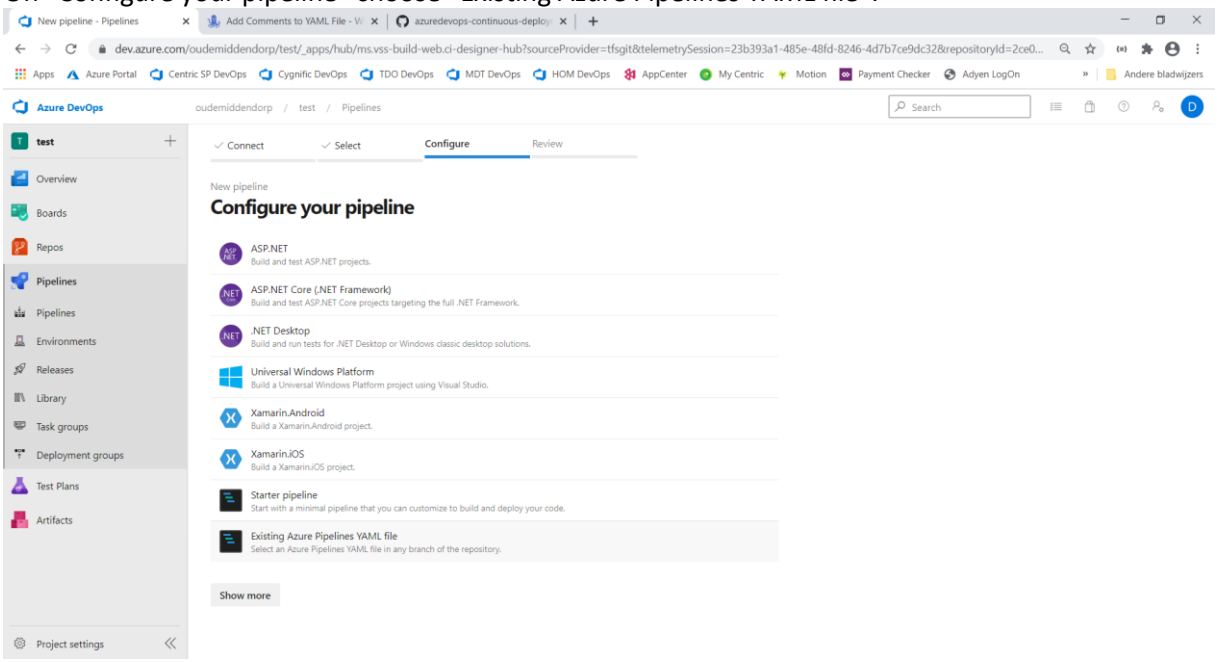
Now, from the left menu in Azure DevOps, choose “Pipelines” (the blue “rocket”) and then “Create Pipeline”.



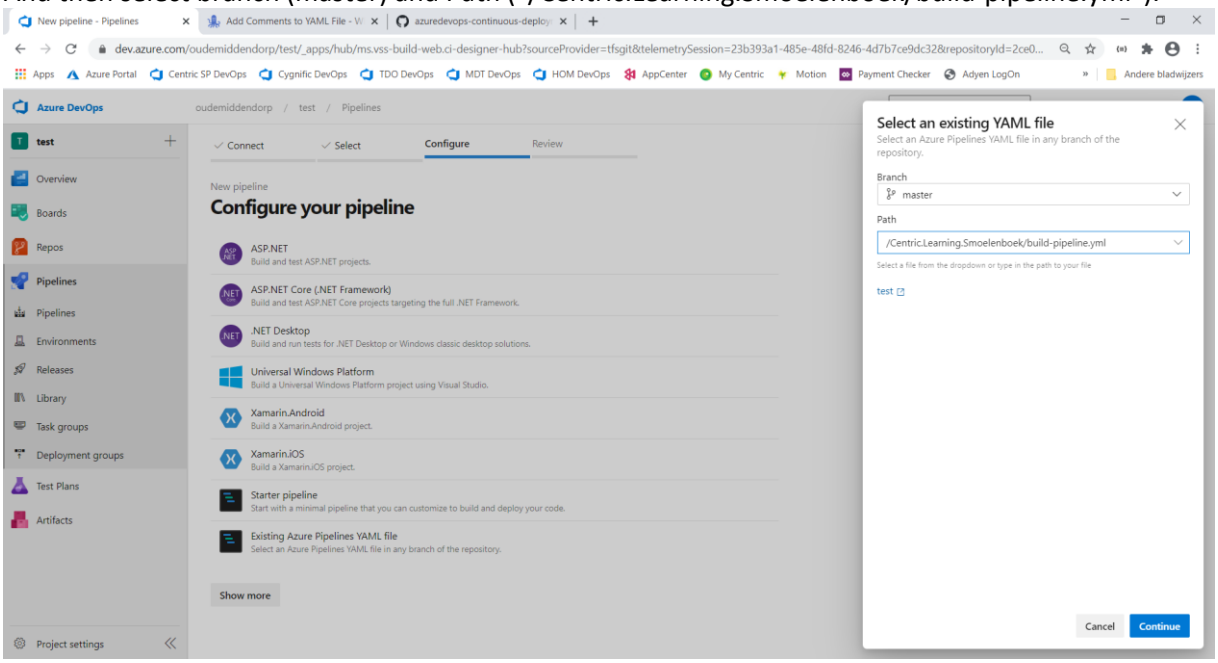
Select “Azure Repos Git (YAML)” as your code repository. And then select the repository.



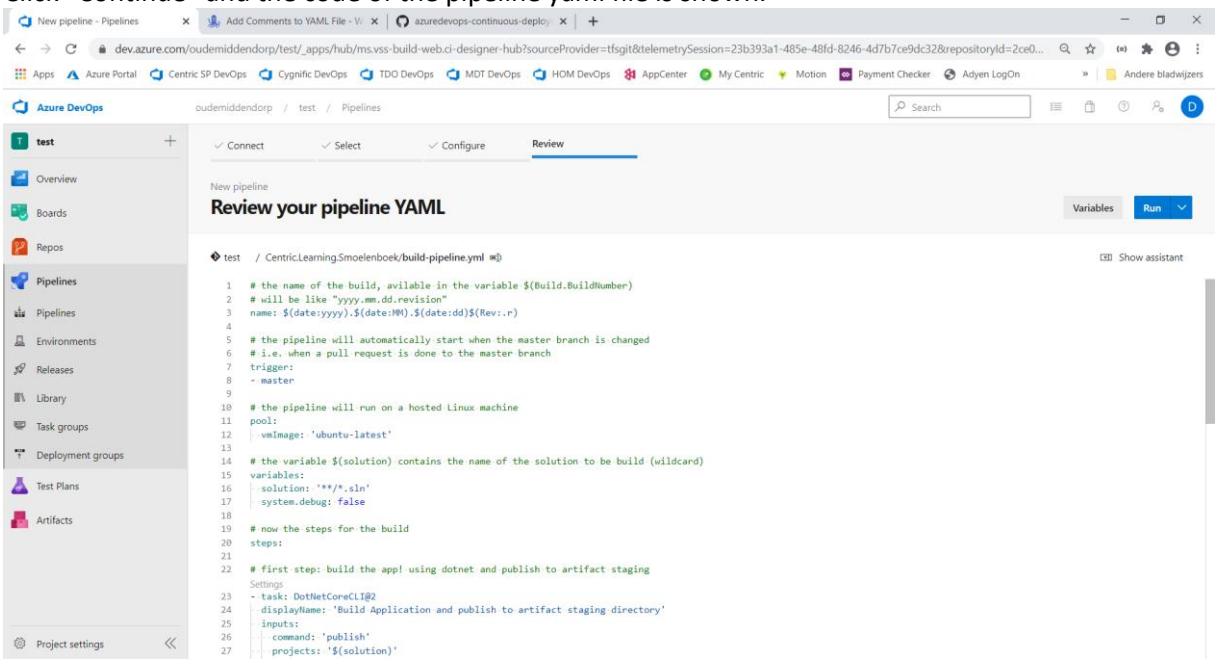
On “Configure your pipeline” choose “Existing Azure Pipelines YAML file”.



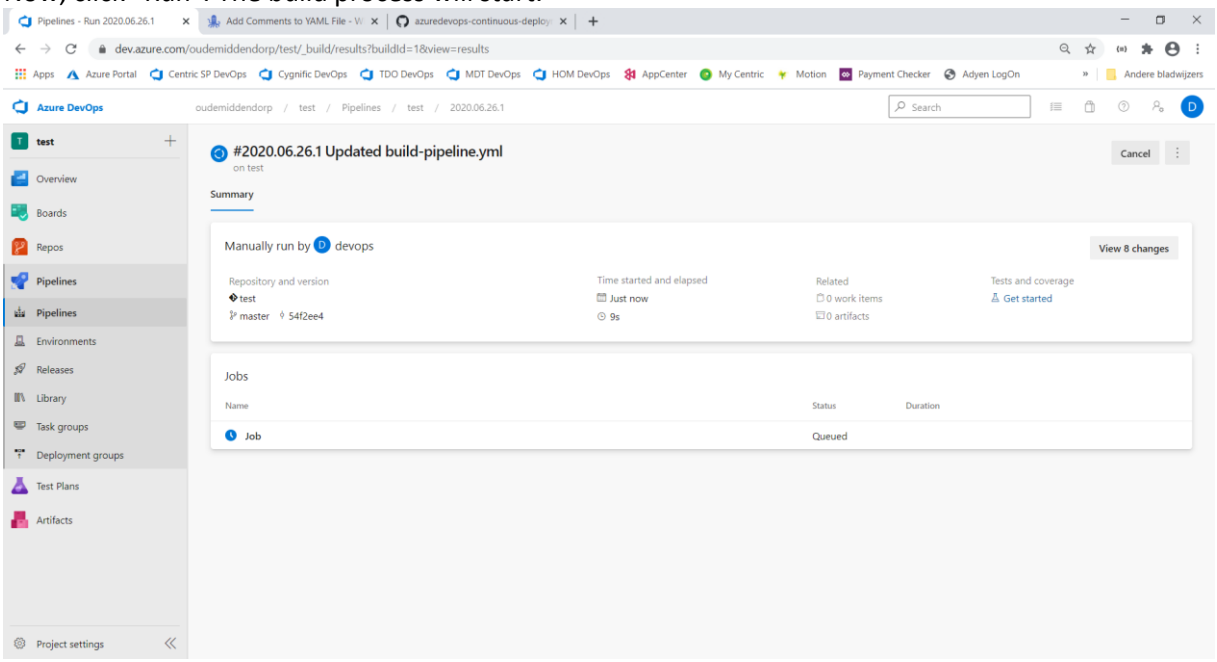
And then select branch (master) and Path (“/Centric.Learning.Smoelenboek/build-pipeline.yml”).



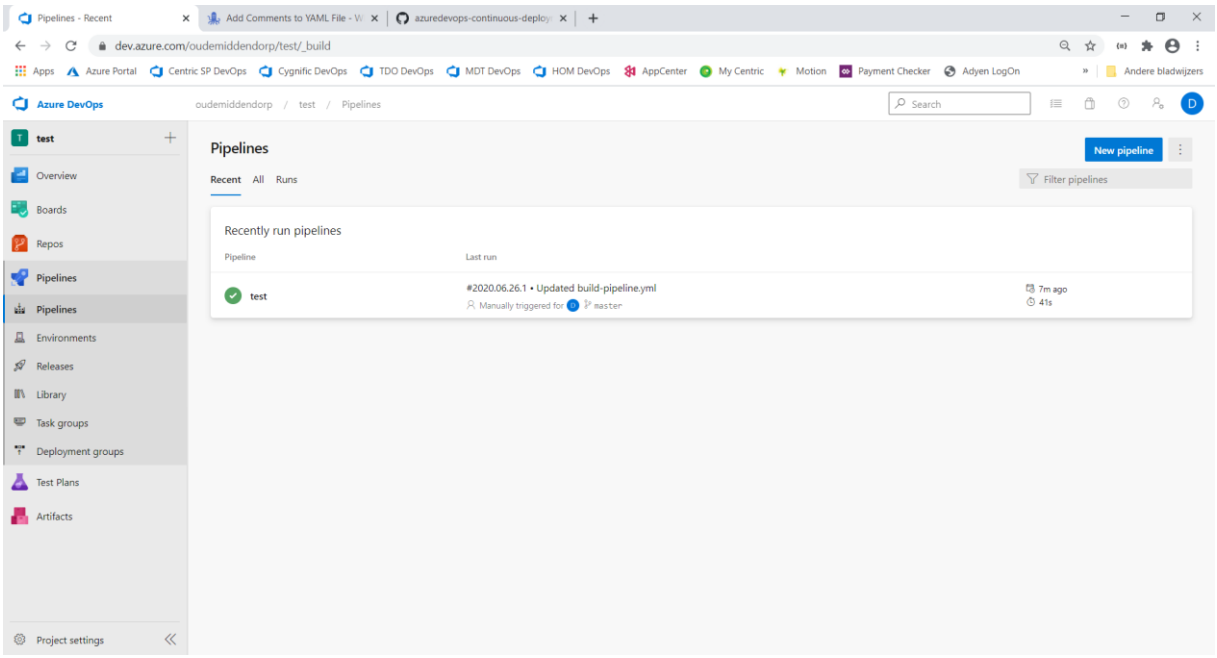
Click “Continue” and the code of the pipeline yaml file is shown.



Now, click “Run”. The build process will start.



You can follow it by clicking on “Job”. What you will see is the output of the machine that is running the build process. When this is ready, click on Pipelines in the main menu again. You will see that there is a recent build available (in this case the name of this pipeline is “test”).

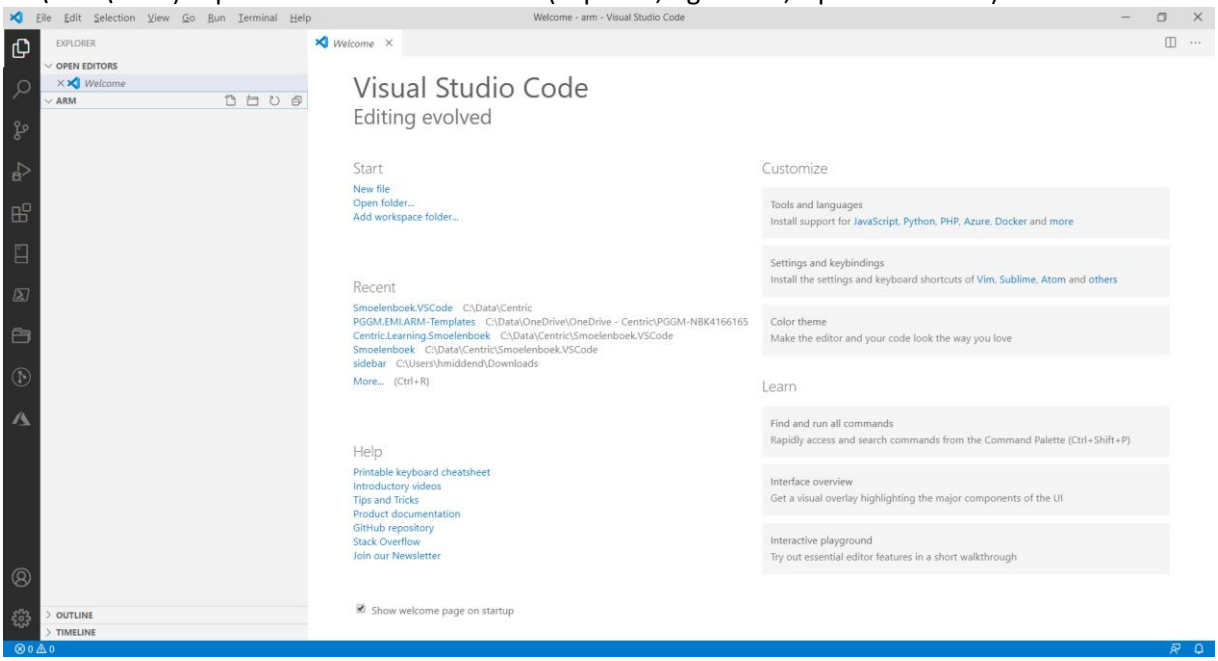


Create an ARM template

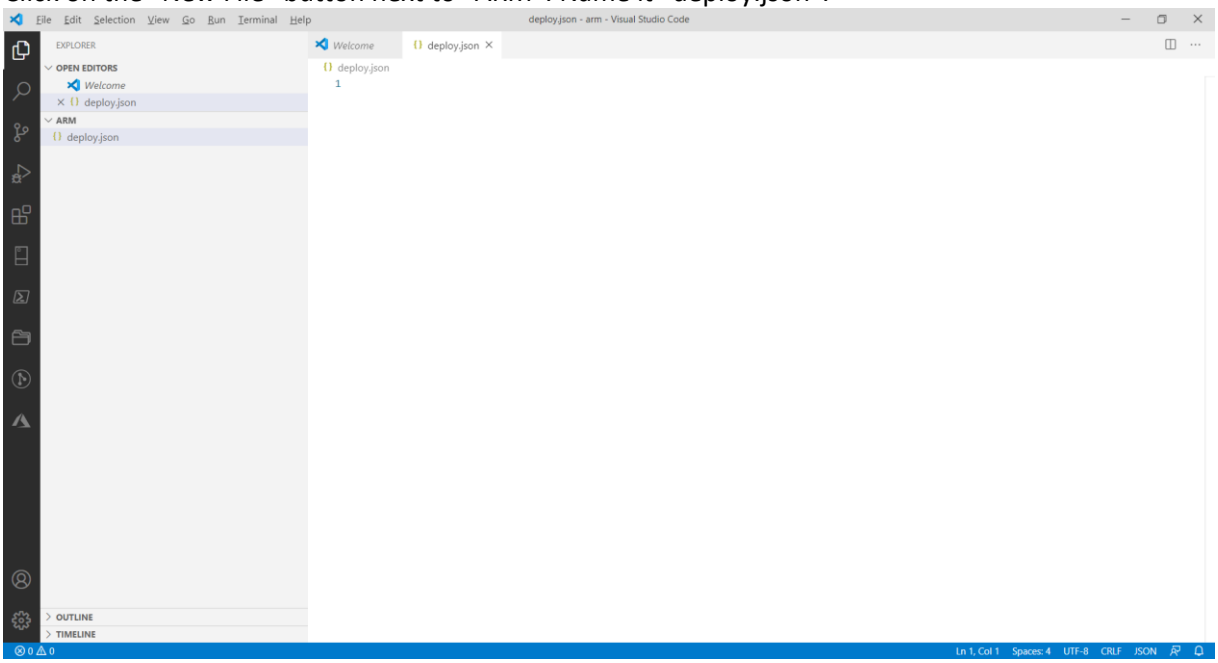
We have our build process in place and now must create the definition of the infrastructure that our application will run on. We will create an ARM template for this.

Construct the ARM template

Go to your explorer and create a new folder in which you will create the ARM template JSON file (i.e. “c:\data\arm”). Open this folder with “Code” (explorer, right click, Open with Code).



Click on the “New File” button next to “ARM”. Name it “deploy.json”.



Now we will start to create the ARM template. We will do this by using the shortcuts that the “ARM Resource Manager (ARM) tools” extension installed for us.

Type “arm!” and then enter. The following code is added to the file:

```
{
    "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {},
    "functions": [],
    "variables": {},
    "resources": [],
    "outputs": {}
}
```

We will create a Azure SQL database that is secured by a username and password. We will generate this username and password so that we not need to enter this information during a release. Next to that we do need to generate a SQL server name and a SQL database name. This is all stored in variables.

Change the “variables” tag so that is looks like this:

```
    "variables": {
        "sqlServerName": "[concat('ss-', uniqueString(resourceGroup().id))]",
        "sqlServerAdmin": "[concat('l', uniqueString(resourceGroup().id, 'usrsecretinformation'))]",
        "sqlServerAdminPwd": "[concat('P', uniqueString(resourceGroup().id, 'pwdsecretinformation'), 'x
', '!')]",
        "sqlDatabaseName": "[concat('db-', uniqueString(resourceGroup().id))]"
    },
```

Now, we can add our first resource to the template; the SQL server. Place the cursor between the brackets after “resources”, create a new line and type “arm-sql-server”. The following code is added:

```
    "resources": [
    {
        "name": "sqlServer1",
        "type": "Microsoft.Sql/servers",
        "apiVersion": "2014-04-01",
        "location": "[resourceGroup().location]",
        "tags": {
            "displayName": "sqlServer1"
        },
```

```
        "properties": {
            "administratorLogin": "adminUsername",
            "administratorLoginPassword": "adminPassword"
        },
        "resources": [
            {
                "type": "firewallRules",
                "apiVersion": "2014-04-01",
                "dependsOn": [
                    "[resourceId('Microsoft.Sql/servers', 'sqlServer1')]"
                ],
                "location": "[resourceGroup().location]",
                "name": "AllowAllWindowsAzureIps",
                "properties": {
                    "startIpAddress": "0.0.0.0",
                    "endIpAddress": "0.0.0.0"
                }
            }
        ]
    }
    ],
```

Change the following tags:

- name: [variables('sqlServerName')];
- tags.displayName: SQL Server;
- properties.administratorLogin: [variables('sqlServerAdmin')];
- properties.administratorLoginPassword: [variables('sqlServerAdminPwd')].

Your code should now like this:

```
    "resources": [
    {
        "name": "[variables('sqlServerName')]",
        "type": "Microsoft.Sql/servers",
        "apiVersion": "2014-04-01",
        "location": "[resourceGroup().location]",
        "tags": {
            "displayName": "SQL Server"
        },
        "properties": {
            "administratorLogin": "[variables('sqlServerAdmin')]",
            "administratorLoginPassword": "[variables('sqlServerAdminPwd')]"
        },
        "resources": [
            {
                "type": "firewallRules",
                "apiVersion": "2014-04-01",
                "dependsOn": [
                    "[resourceId('Microsoft.Sql/servers', variables('sqlServerName'))]"
                ],
                "location": "[resourceGroup().location]",
                "name": "AllowAllWindowsAzureIps",
                "properties": {
                    "startIpAddress": "0.0.0.0",
                    "endIpAddress": "0.0.0.0"
                }
            }
        ]
    }
    ],
```

Next, we have to add the database resource. Enter a comma after the last accolade of the last element in the "resources" array and start a new line. Enter "arm-sql-db". The following code appears:

```
    {
        "name": "sqlServer1/sqlDatabase1",
        "type": "Microsoft.Sql/servers/databases",
        "apiVersion": "2014-04-01",
        "location": "[resourceGroup().location]",
        "tags": {
```

```
            "displayName": "sqlDatabase1"
        },
        "dependsOn": [
            "[resourceId('Microsoft.Sql/servers', 'sqlServer1')]"
        ],
        "properties": {
            "collation": "SQL_Latin1_General_CP1_CI_AS",
            "edition": "Basic",
            "maxSizeBytes": "1073741824",
            "requestedServiceObjectiveName": "Basic"
        }
    }
```

Change the following tags:

- name: [concat(variables('sqlServerName'), '/', variables('sqlDatabaseName'))]
- tags.displayName: SQL Database.

And in the "dependsOn", tags change the value to:
[resourceId('Microsoft.Sql/servers', variables('sqlServerName'))]

Your code should now look like this:

```
    {
        "name": "[concat(variables('sqlServerName'), '/', variables('sqlDatabaseName'))]",
        "type": "Microsoft.Sql/servers/databases",
        "apiVersion": "2014-04-01",
        "location": "[resourceGroup().location]",
        "tags": {
            "displayName": "SQL Database"
        },
        "dependsOn": [
            "[resourceId('Microsoft.Sql/servers', variables('sqlServerName'))]"
        ],
        "properties": {
            "collation": "SQL_Latin1_General_CP1_CI_AS",
            "edition": "Basic",
            "maxSizeBytes": "1073741824",
            "requestedServiceObjectiveName": "Basic"
        }
    }
```

We also need a BlockBlob storage account that will store the photos of the application. Again, enter a comma and new line after the last accolade of the resources array and enter "arm-storage". The following code should appear:

```
    {
        "name": "storageaccount1",
        "type": "Microsoft.Storage/storageAccounts",
        "apiVersion": "2019-06-01",
        "tags": {
            "displayName": "storageaccount1"
        },
        "location": "[resourceGroup().location]",
        "kind": "StorageV2",
        "sku": {
            "name": "Premium_LRS",
            "tier": "Premium"
        }
    }
```

We need to create a variable for generating the name of the storage account. Go to the variables tags and change it to this:

```
    "variables": {
        "sqlServerName": "[concat('ss-', uniqueString(resourceGroup().id))]",
        "sqlServerAdmin": "[concat('l', uniqueString(resourceGroup().id, 'usrsecretinformation'))]",
        "sqlServerAdminPwd": "[concat('P', uniqueString(resourceGroup().id, 'pwdsecretinformation'), 'x
', '!')]",
        "sqlDatabaseName": "[concat('db-', uniqueString(resourceGroup().id))]",
        "storageAccountName": "[toLower(concat('sa', uniqueString(resourceGroup().id)))]"
    },
```

After that, change the following tags for to storage account resource you just created:

- name: [variables('storageAccountName')];
- tags.displayName: Storage Account;
- kind: BlockBlobStorage.

Your code should look like this:

```
    {
        "name": "[variables('storageAccountName')]",
        "type": "Microsoft.Storage/storageAccounts",
        "apiVersion": "2019-06-01",
        "tags": {
            "displayName": "Storage Account"
        },
        "location": "[resourceGroup().location]",
        "kind": "BlockBlobStorage",
        "sku": {
            "name": "Premium_LRS",
            "tier": "Premium"
        }
    }
```

We web application will run in a Web App resource. For this we need to set up an App Service Plan and a Web App. These also must have a name that we will generate in the variables section. Add the following lines to the variables section (do not forget to enter a comma after the last variable you defined):

```
        "appServicePlanName": "[concat('asp-', uniqueString(resourceGroup().id))]",
        "webAppName": "[concat('wa-', uniqueString(resourceGroup().id))]"
```

Add a comma and a new line after the last resource in the resources array and enter “arm-plan”. The following code appears:

```
    {
        "name": "appServicePlan1",
        "type": "Microsoft.Web/serverfarms",
        "apiVersion": "2018-02-01",
        "location": "[resourceGroup().location]",
        "sku": {
            "name": "F1",
            "capacity": 1
        },
        "tags": {
            "displayName": "appServicePlan1"
        },
        "properties": {
            "name": "appServicePlan1"
        }
    }
```

Change the following tags:

- name: [variables('appServicePlanName')];
- tags.displayName: App Service Plan;
- properties.name: [variables('appServicePlanName')].

Your code should look like this:

```
    {
        "name": "[variables('appServicePlanName')]",
        "type": "Microsoft.Web/serverfarms",
        "apiVersion": "2018-02-01",
        "location": "[resourceGroup().location]",
        "sku": {
            "name": "F1",
            "capacity": 1
        },
        "tags": {
            "displayName": "App Service Plan"
        },
        "properties": {
            "name": "[variables('appServicePlanName')]"
        }
    }
```

Now, enter another comma and newline and add the Web App resource by typing “arm-web-app”. The following code appears:

```
    {
        "name": "webApp1",
        "type": "Microsoft.Web/sites",
        "apiVersion": "2018-11-01",
        "location": "[resourceGroup().location]",
        "tags": {
            "[concat('hidden-
related:', resourceGroup().id, '/providers/Microsoft.Web/serverfarms/appServicePlan1')]": "
Resource",
            "displayName": "webApp1"
        },
        "dependsOn": [
            "[resourceId('Microsoft.Web/serverfarms', 'appServicePlan1')]"
        ],
        "properties": {
            "name": "webApp1",
            "serverFarmId": "[resourceId('Microsoft.Web/serverfarms', 'appServicePlan1')]"
        }
    }
```

Change the following tags:

- name: [variables('webAppName')]
- tags.”[concat('hidden-related:', resourceGroup().id, '/providers/Microsoft.Web/serverfarms/appServicePlan1')]": "Resource" to "[concat('hidden-related:', resourceGroup().id, concat('/providers/Microsoft.Web/serverfarms/', variables('appServicePlanName')))]": "Resource"
- tags.displayName: Web Application
- properties.name: [variables('webAppName')]
- properties.serverFarmId: [resourceId('Microsoft.Web/serverfarms', variables('appServicePlanName'))]

The web application not only depends on an app service plan, but also the database server and the storage account must exist. Change the “dependsOn” section to this:

```
        "dependsOn": [
            "[resourceId('Microsoft.Web/serverfarms', variables('appServicePlanName'))]",
            "[resourceId('Microsoft.Sql/servers', variables('sqlServerName'))]",
            "[resourceId('Microsoft.Storage/storageAccounts', variables('storageAccountName
'))]"
        ],
```

We are almost ready with the ARM template. We only need to tell the web application where the data is. We do this by configuring the connection strings to the database and the storage account. Do this by adding the following lines to the “properties” section (do not forget the comma at the end of the “serverFarmId” line:

```
            "siteConfig": {
                "connectionStrings": [
                    {
                        "name": "DefaultConnection",
                        "connectionString": "[concat('Server=tcp:', variables('sqlServerNam
e'), '.database.windows.net,1433;Initial Catalog=', variables('sqlDatabaseName'), ';Persist
 Security Info=False;User ID=', variables('sqlServerAdmin'), ';Password=', variables('sqlSe
rverAdminPwd'), ';MultipleActiveResultSets=False;Encrypt=True;TrustServerCertificate=False;
Connection Timeout=30;')]",
                        "type": "SQLAzure"
                    },
                    {
                        "name": "StorageConnection",
                        "connectionString": "[concat('DefaultEndpointsProtocol=https;Accoun
tName=',variables('storageAccountName'),';AccountKey=',listKeys(resourceId('Microsoft.Stora
ge/storageAccounts', variables('storageAccountName')), providers('Microsoft.Storage', 'stor
ageAccounts').apiVersions[0]).keys[0].value)]",
                        "type": "Custom"
                    }
                ]
            }
```

As you see, the connection strings are generated from all variables we defined.

After the ARM template is implemented in Azure we will need to deploy the code of the web application to the web app resource. We will need the name of the Web App resource to deploy the code to it. Therefore, the ARM template must generate some output that contains the name of the Web App. This can be defined in the outputs section.

Change the output section to this:

```
    "outputs": {
        "webappName": {
            "value": "[variables('webappName')]",
            "type": "string"
        }
    }
```

Validate the ARM template

Before we make a change to the code so we can use our ARM template in a deployment, we need to validate it.

Open a command window or a powershell session
Navigate to the folder you created the ARM template in

```
c:
cd \data\arm
```

Login to Azure:

```
az login
```

All subscriptions you have access to are shown in JSON format.

Set the correct subscription:

```
az account set -s Centric-Training-SP-xx
```

Validate the ARM template

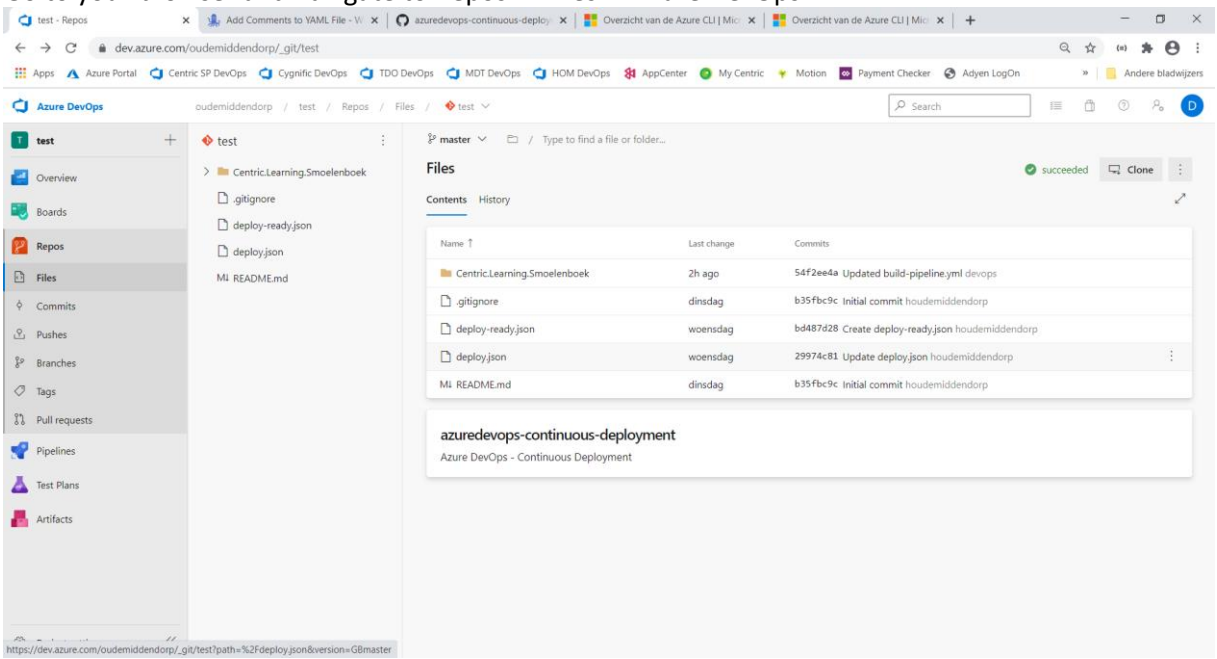
```
az deployment group validate –resource-group <name of your resource group> --
template-file deploy.json
```

You will get information of the outcome of the validation process. Hopefully it is ok 😊
If it is not ok, you can use the completed code which is in the repository “deploy-ready.json” (copy and paste).

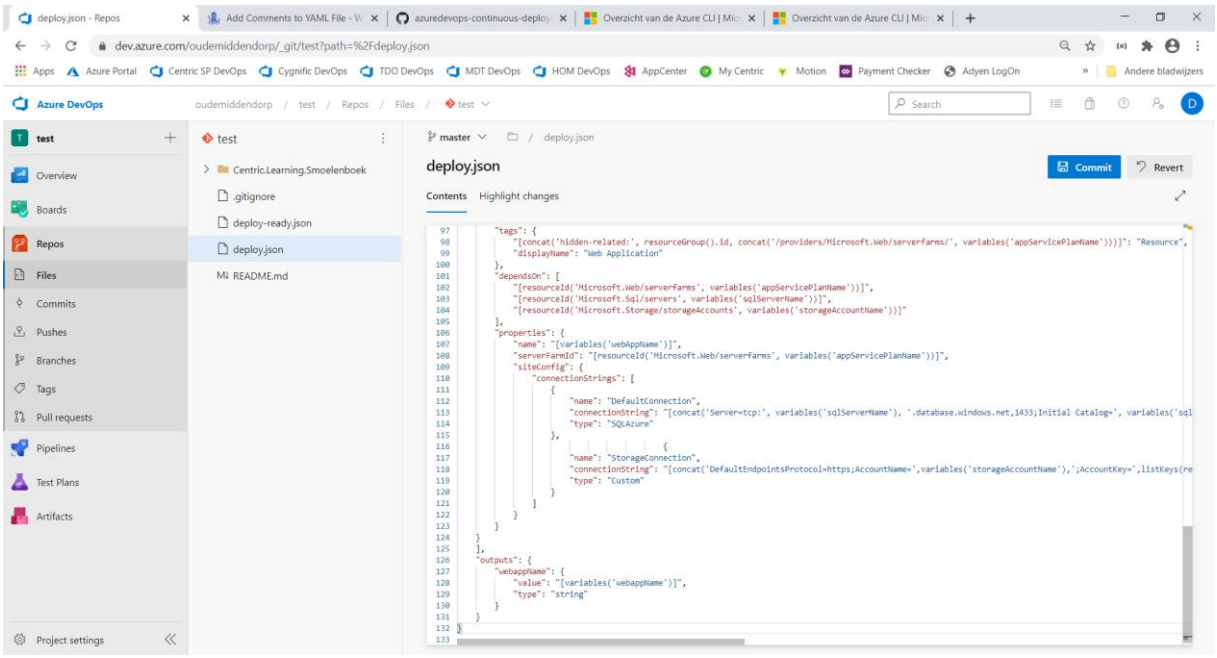
Commit the code to the repository

Here is the part where an IT pro does same things as developers are used to do for ages. Because we do not want to bother you with complex Git things at this moment, we will use the shortcut.

Go to your browser and navigate to Repos -> Files in Azure DevOps.

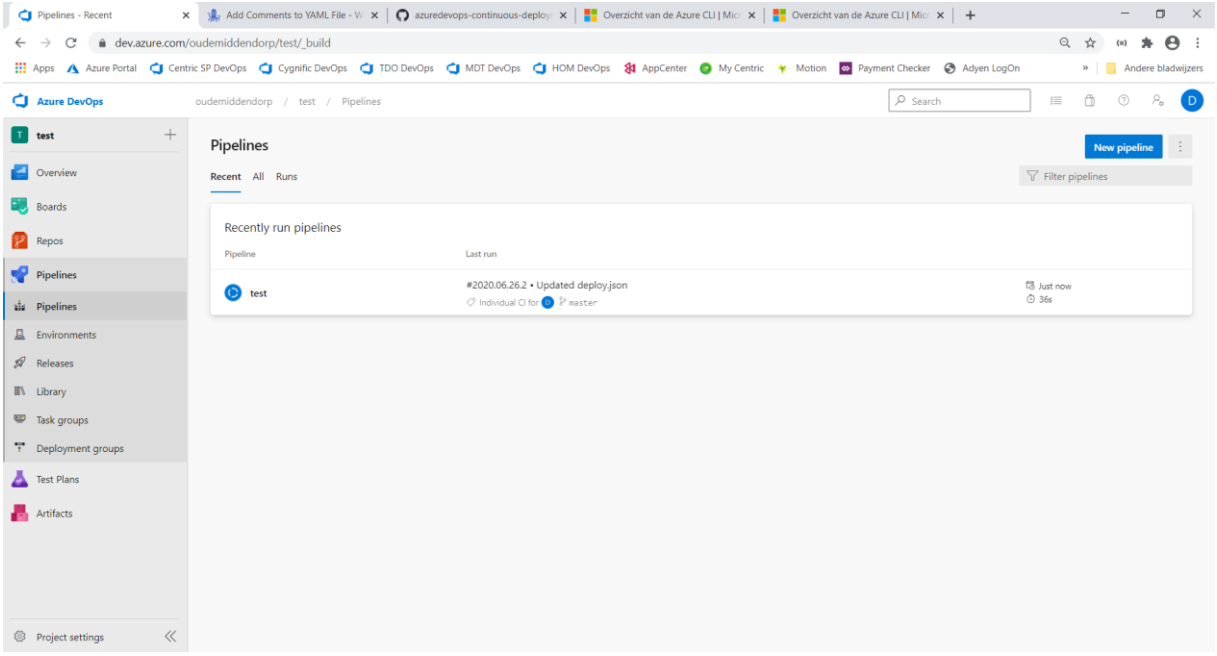


Click on “deploy.json” and then “Edit”. Alt-Tab to Visual Studio Code and copy the contents of the ARM template. Past it in the browser window.



Now we make the actual change to the repository by committing this change. Click “Commit” and enter a brief description. Note that you can also link work items to this change. With that you can link product backlog items to code changes and make traceability of PBI’s to code changes and build and deployments possible. For now, we leave this empty, so click “Commit”.

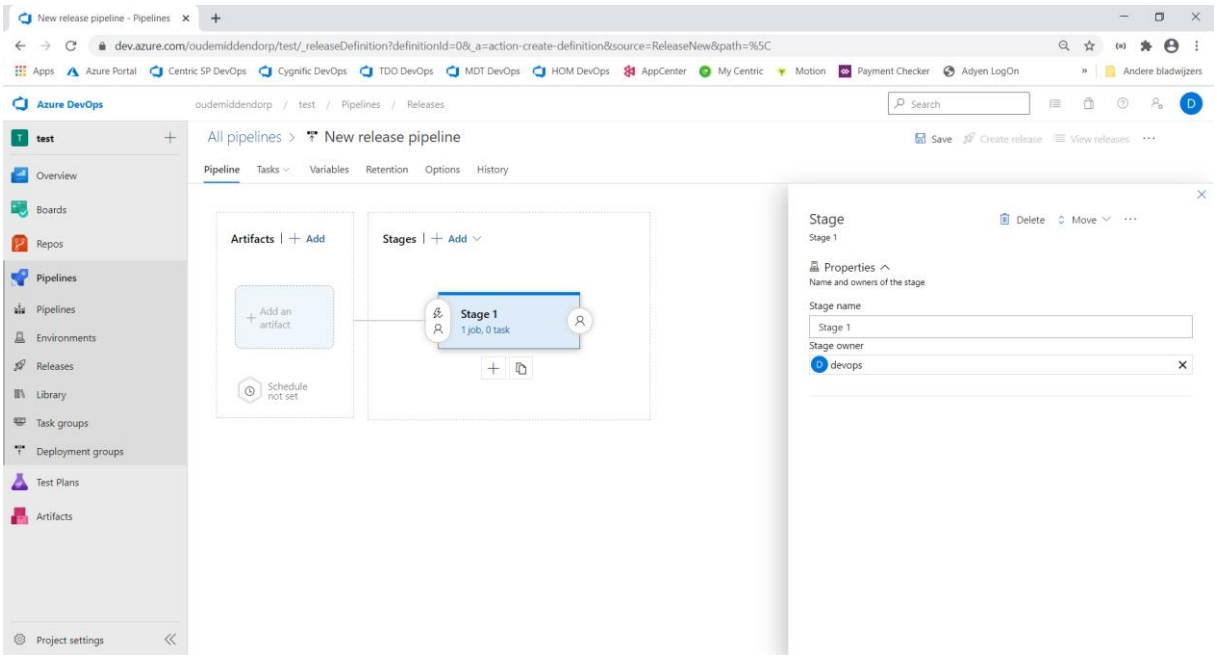
Because you committed this change to the master branch, a new build is started automatically. Navigate to “Pipelines” and see that your build is running because of the change you made.



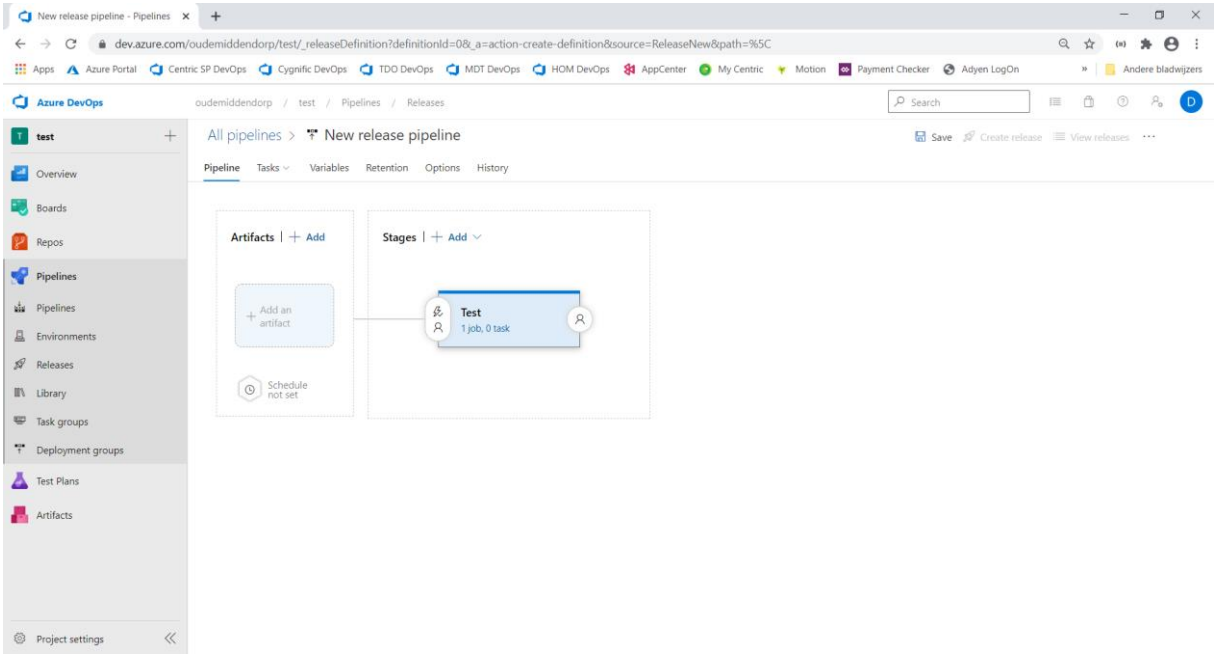
Setup a release pipeline

The last step in this lab is to define the release pipeline. This pipeline is started automatically if the build succeeds. It will first deploy to a test-stage and after that to a production stage.

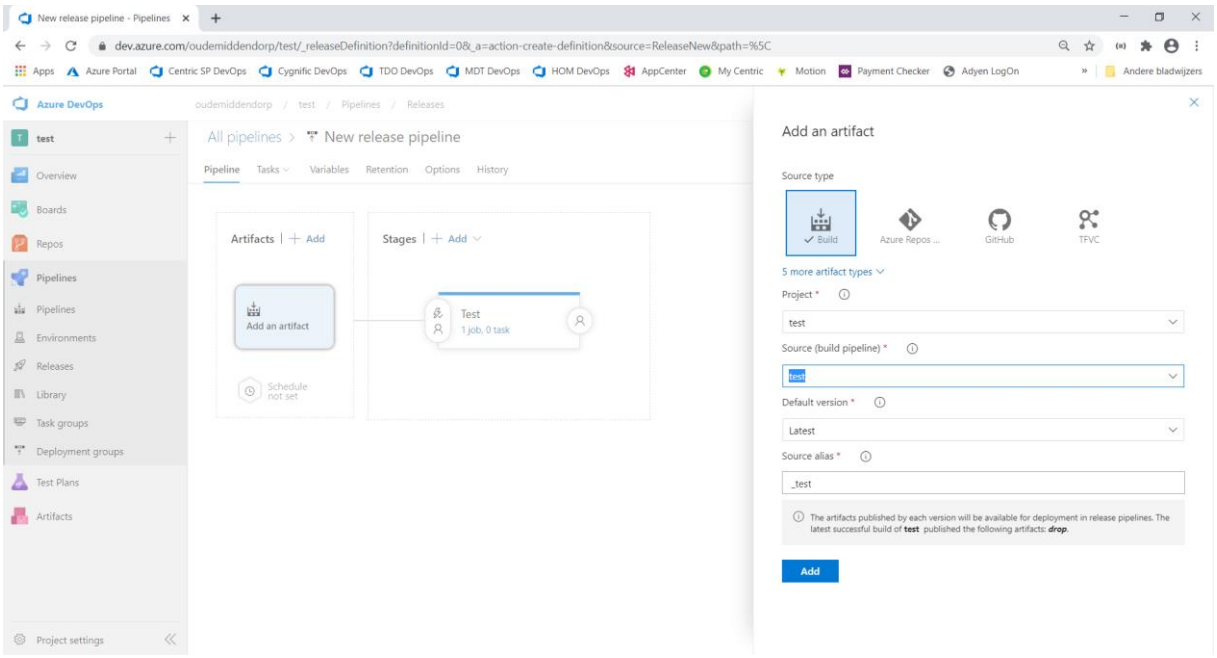
First navigate in Azure DevOps to Pipelines -> Releases and click “New pipeline”. Next, do not select a template, but start with an “Empty job”.



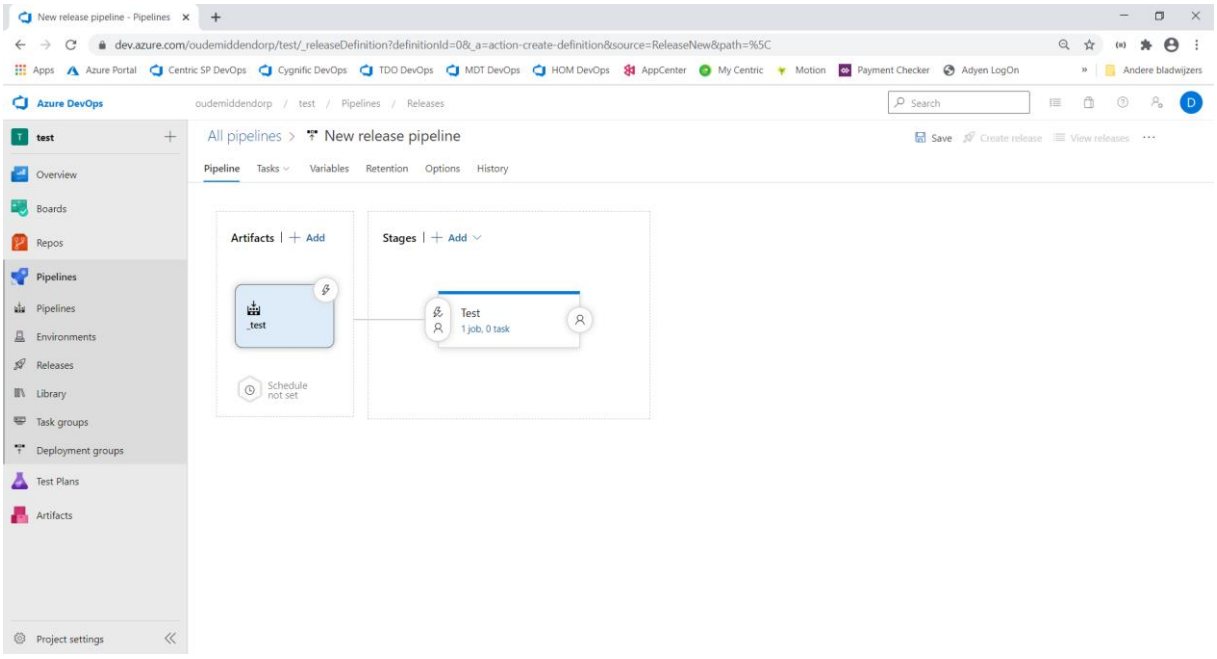
Rename “Stage 1” to “Test” and close the Stage properties window.



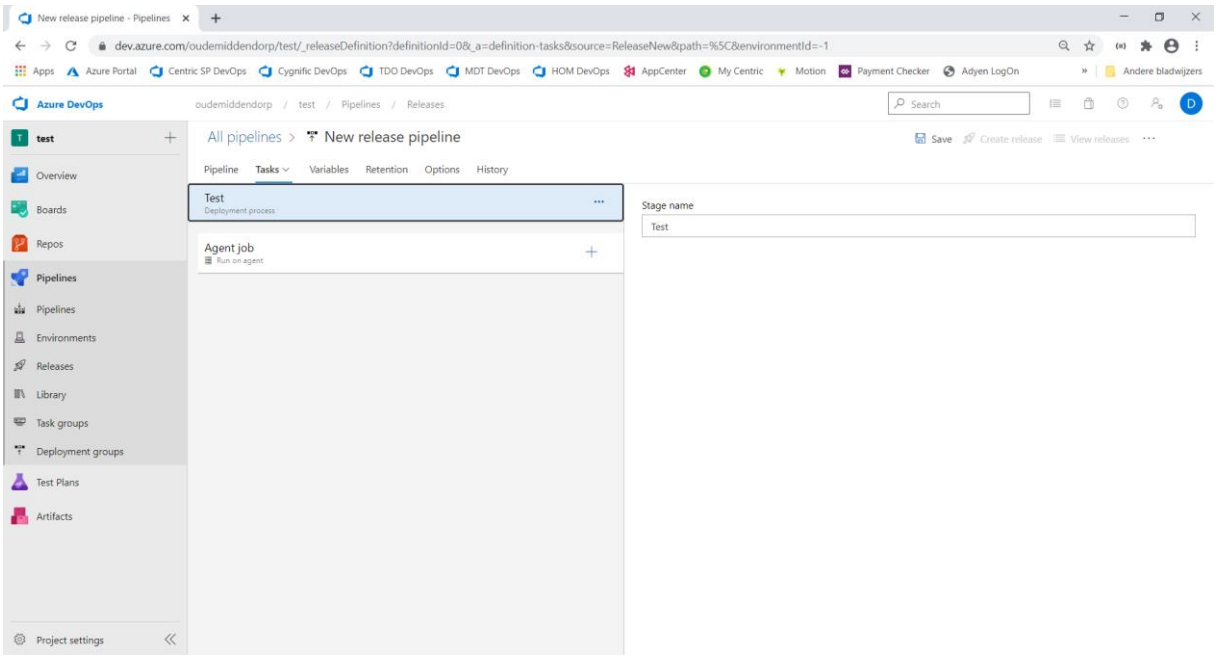
Add the artifact from the build pipeline. Click “+ Add” next to Artifacts and then select the build pipeline you defined earlier.



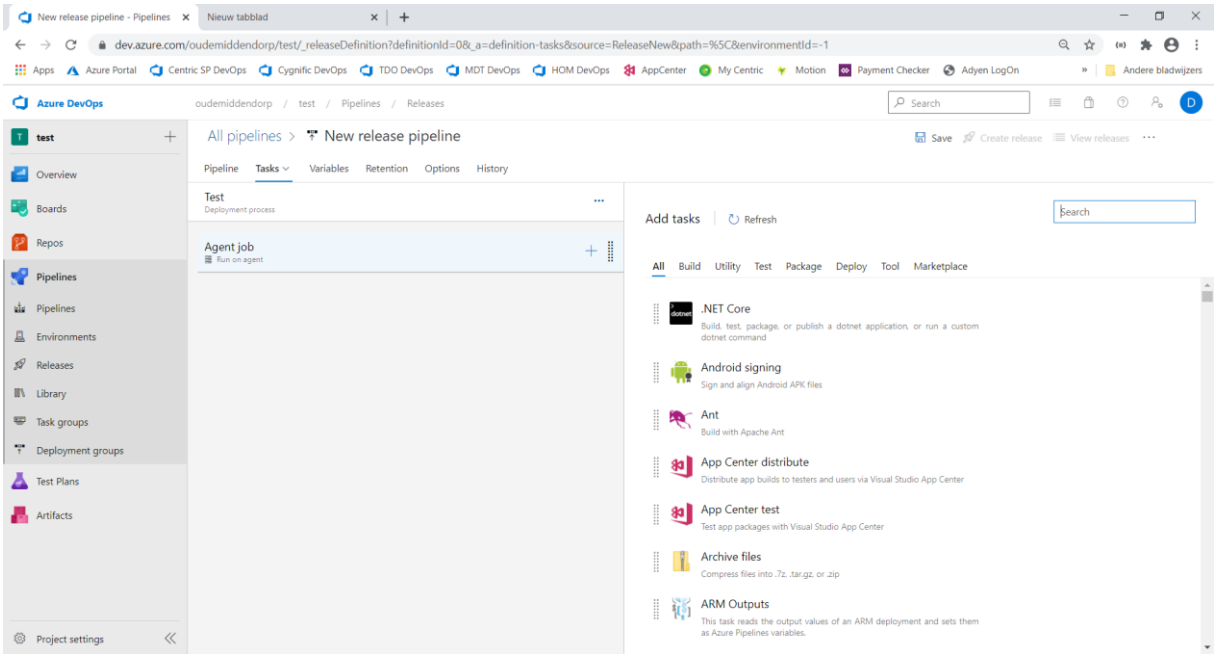
Click “Add”.



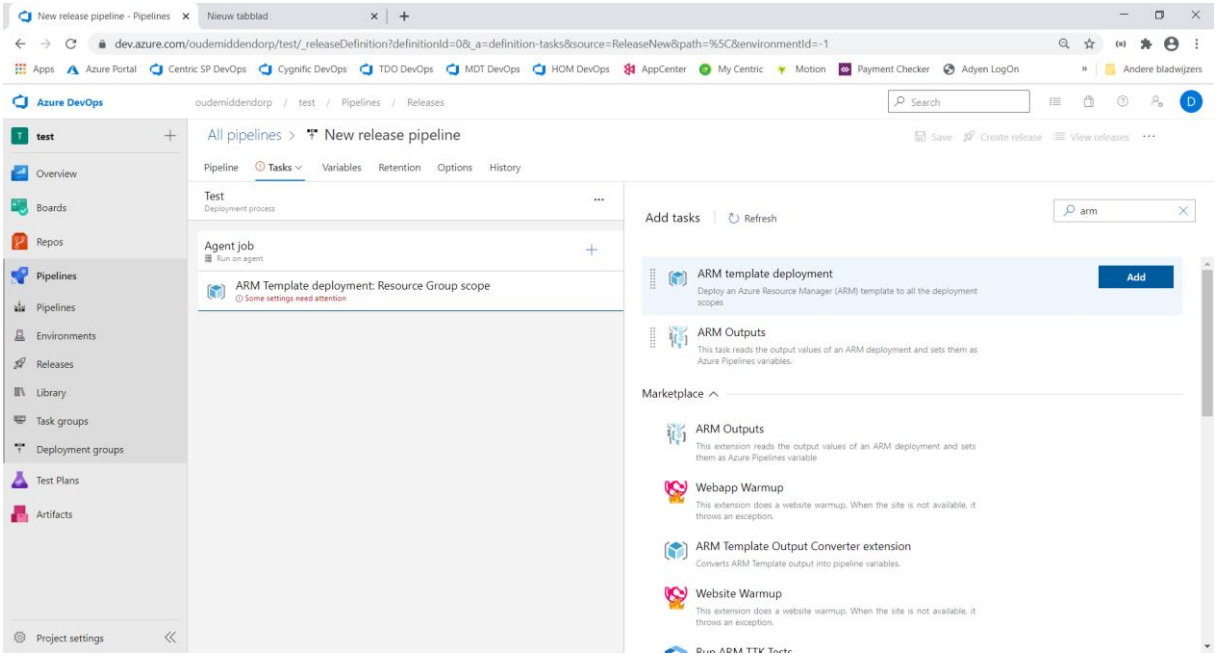
Now we will define the test deployment stage. Click on the link “1 job, 0 task” below “Test”.



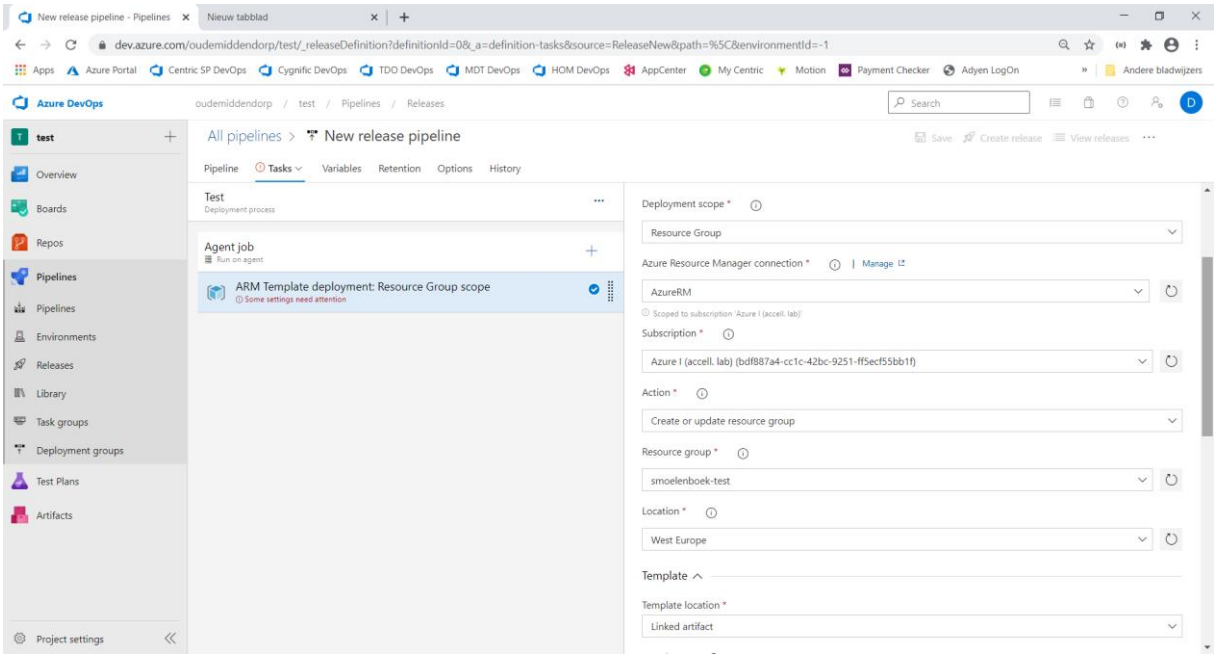
Add a task to the agent job, click no “+” next to “Agent job”.



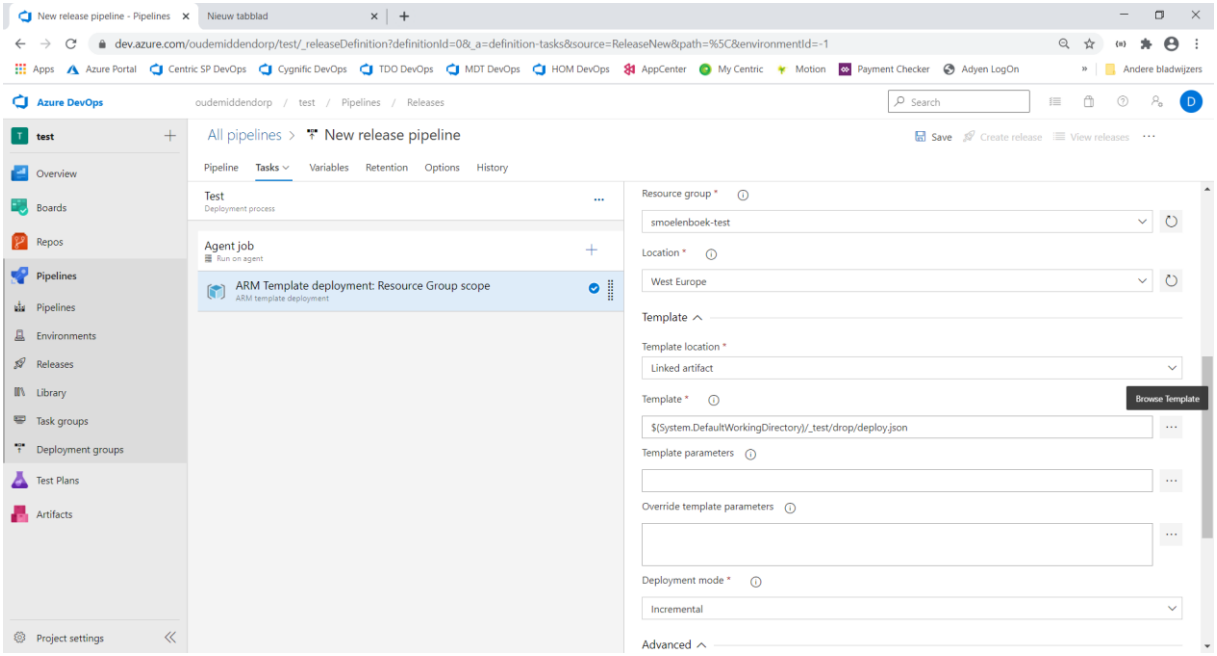
Select “ARM template deployment” from the available tasks and click “Add”.



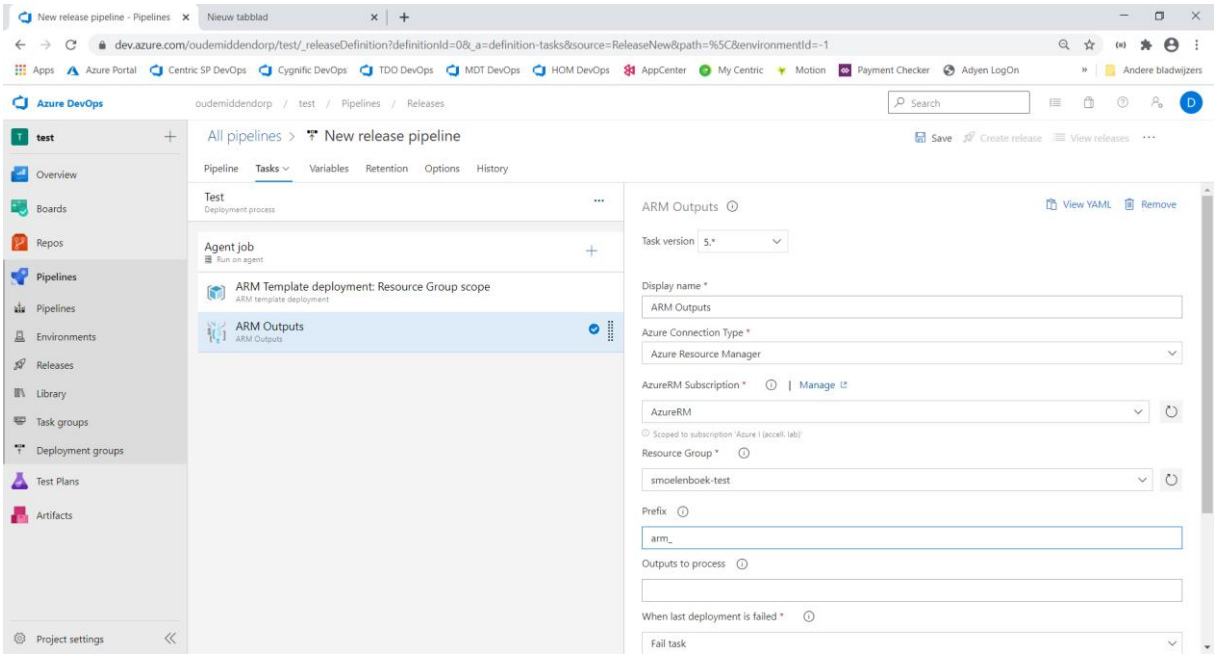
Click on the newly created task and fill the field on the right side of the screen:



Scroll down and select the template.

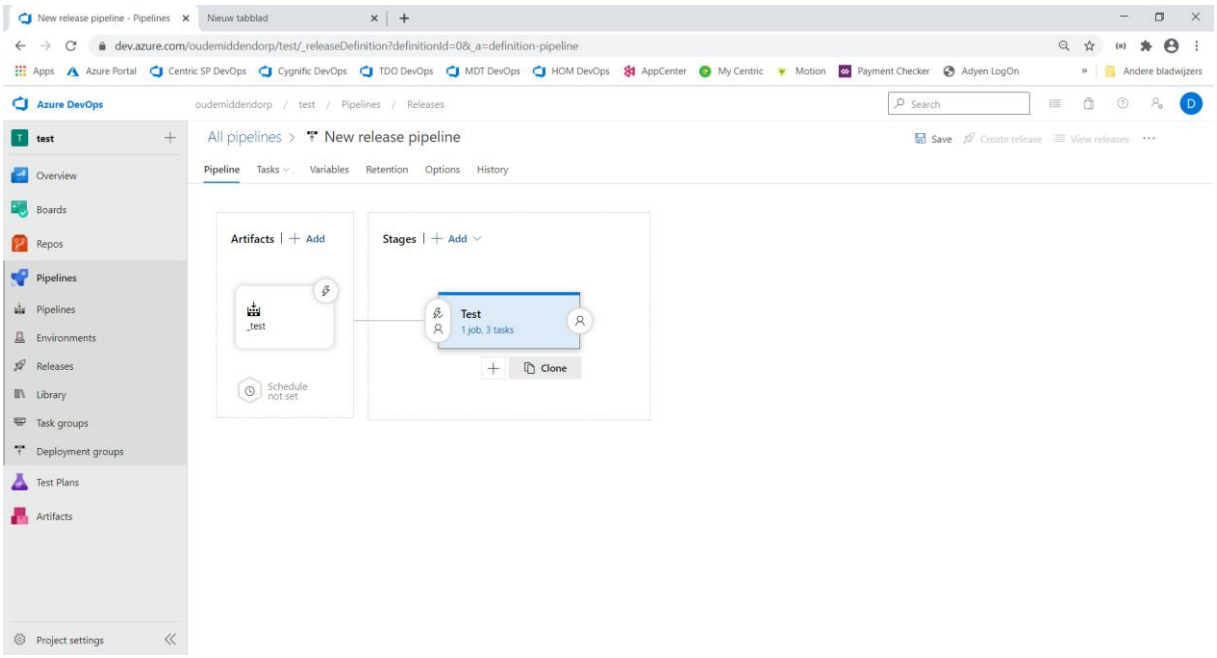


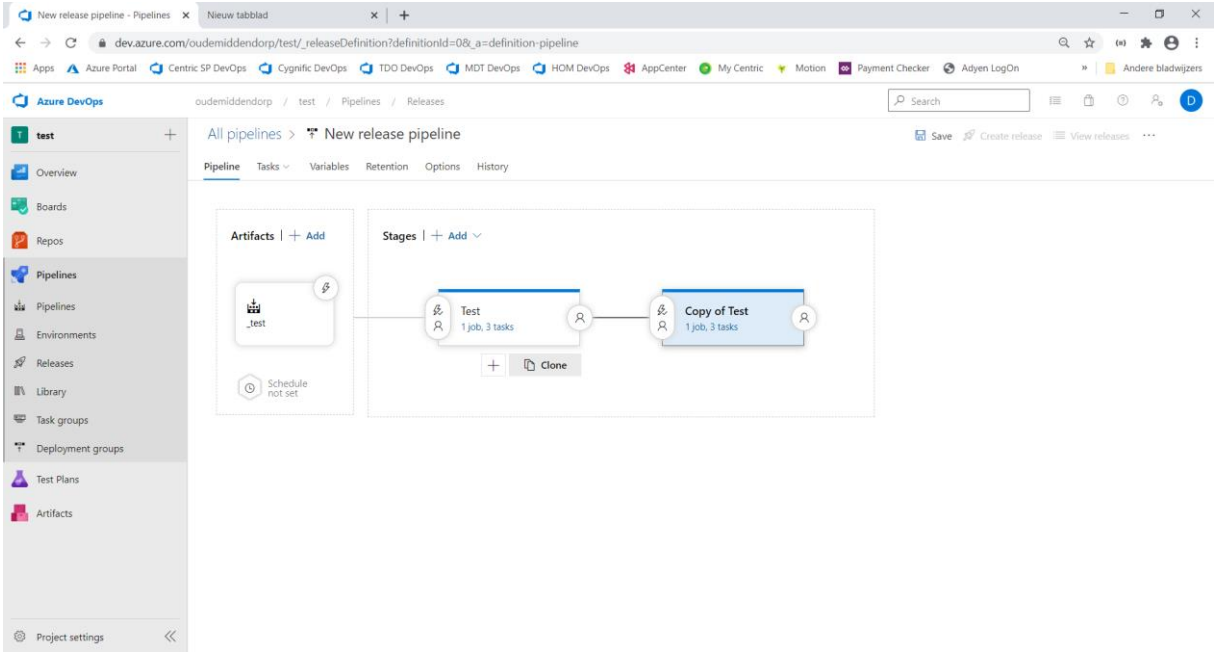
Add another task to the agent job, select “ARM outputs”. Fill the form. Enter “arm_” as “Prefix”.



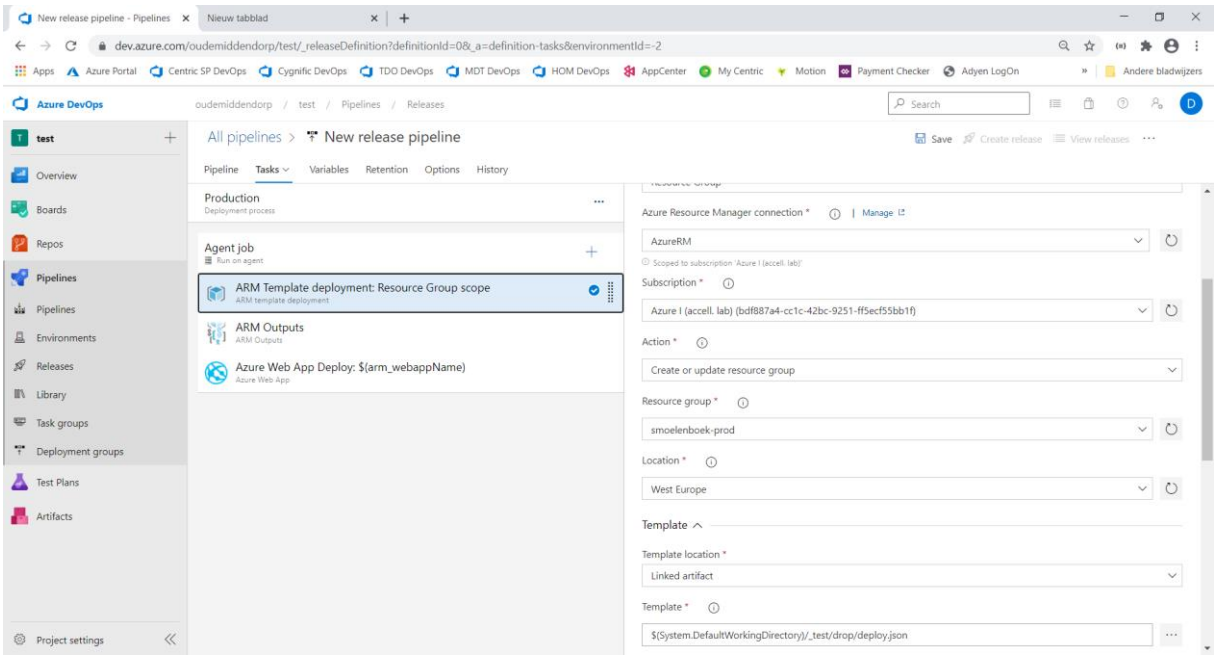
Add the task for deploying the application to the agent job. Select “Azure Web App”. Fill in the form. Select “Web App for Windows” as “App type” and enter “$(arm_webappName)” in the “App name” field.

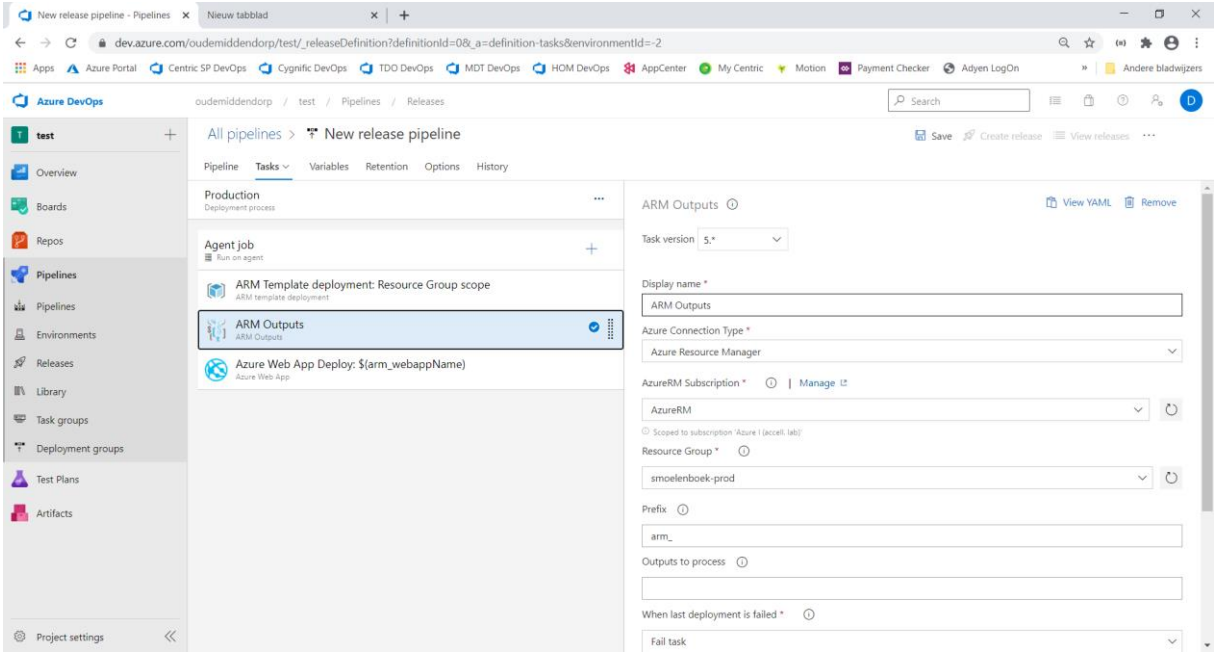
Now the tasks for the deployment to the test stage are defined. We will now copy this stage to the “production” stage. Click on the “Pipeline” tab above the task lists. Move your mouse over the test stage and click “Clone”.





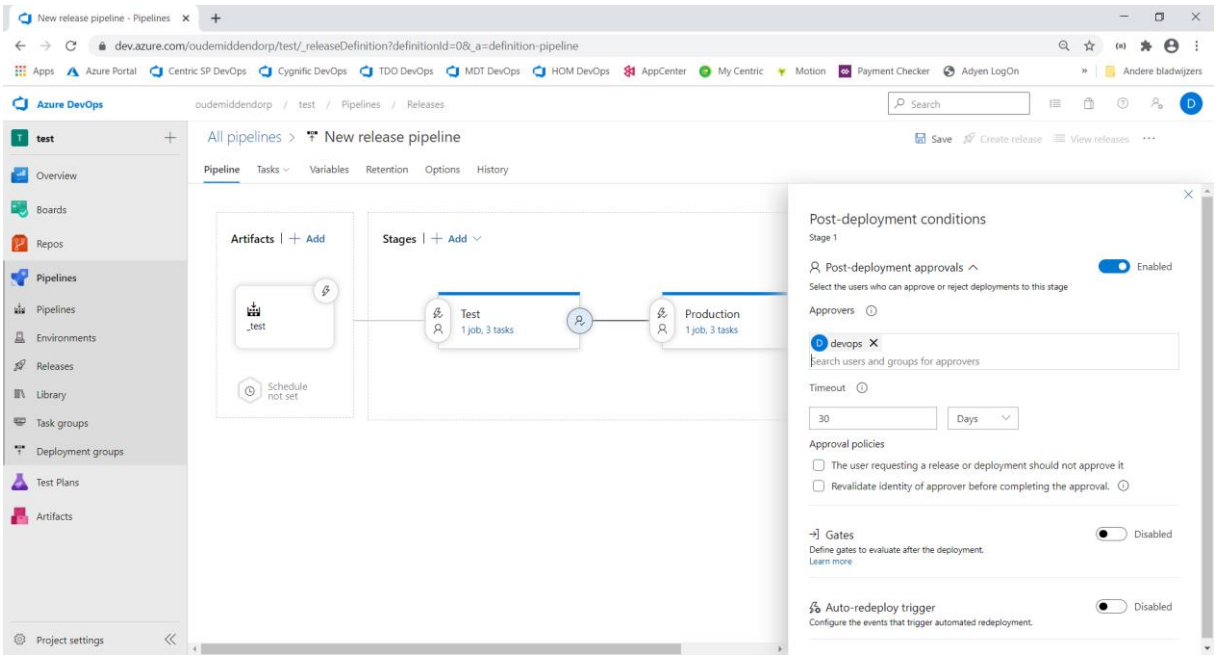
Note that this stage will deploy after the test stage. Rename “Copy of test” to “production” and then open the task list for the “production” stage. In the first 2 tasks, rename the resource group to the second resource group you have available for this lab.



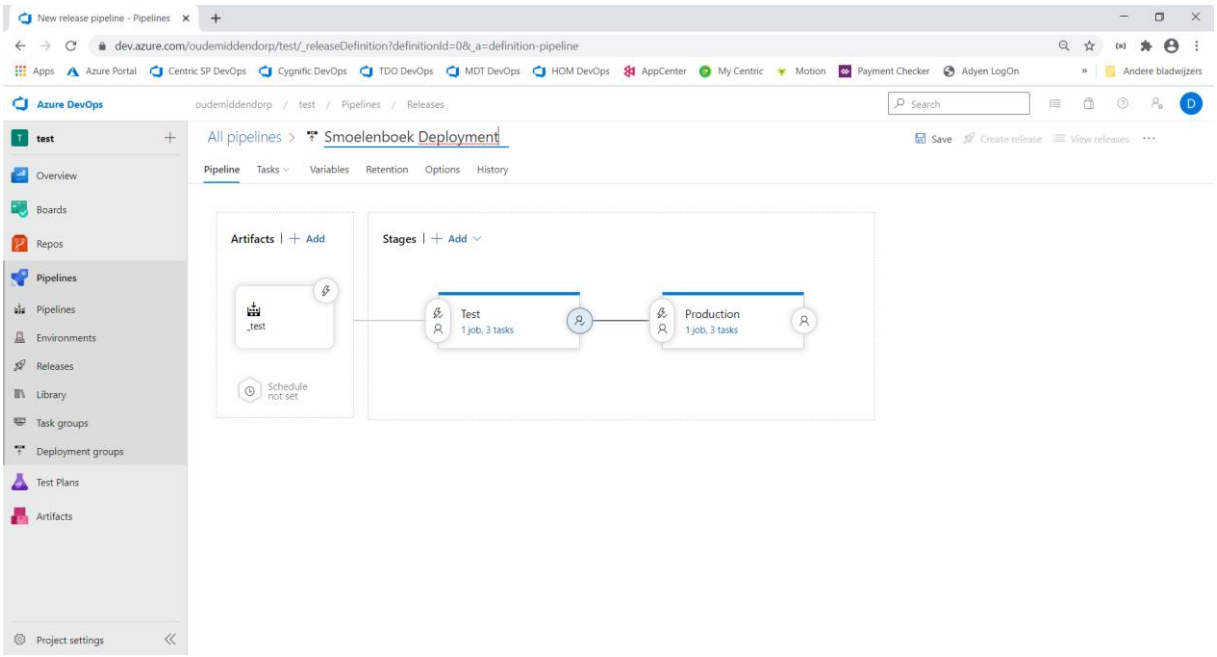


Navigate to the “Pipeline” tab again. Make sure that there is no form open (close the “Stage” form if necessary).

We will now add an approval step to the process; when the test stage is successfully deployed and tested, someone must approve this stage before it will be deployed to production. To implement this, click on the “person” icon on the right side of the “Test” stage block (post-deployment conditions) and enable “Post-deployment approvals”. Select yourself as approver.

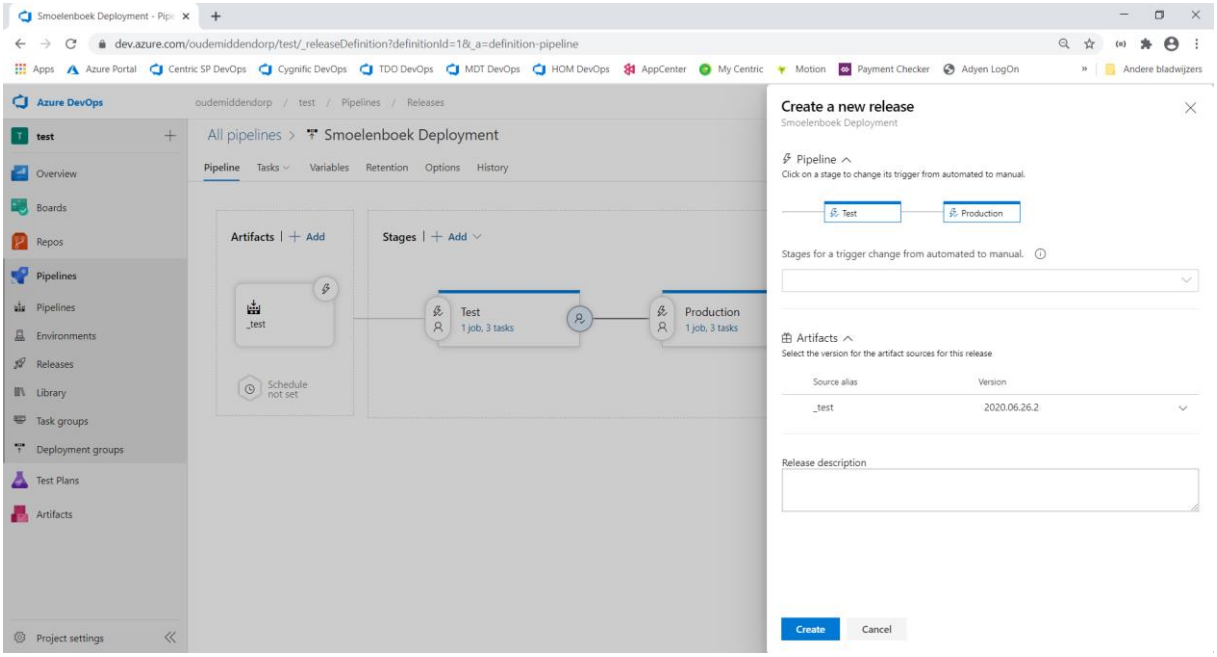


Close this popup. The release pipeline is now ready and can be saved. Before saving, give it a meaningful name and then click “Save”. Leave the defaults in the popup and press “OK”.

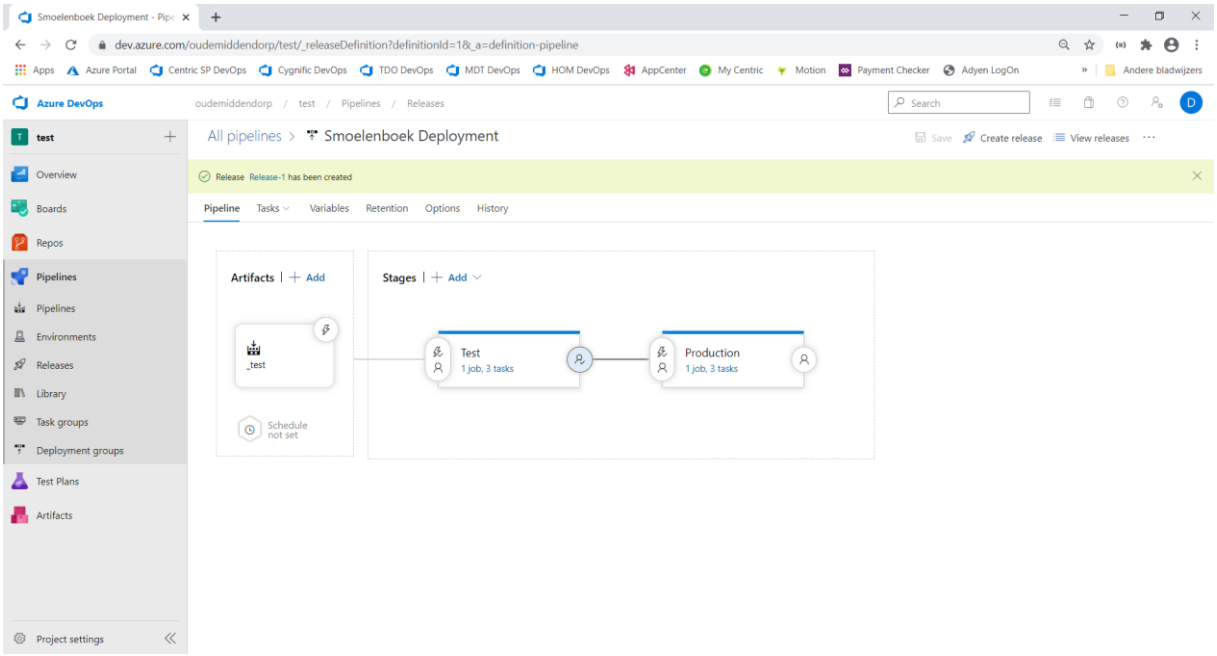


Give it a try!!!

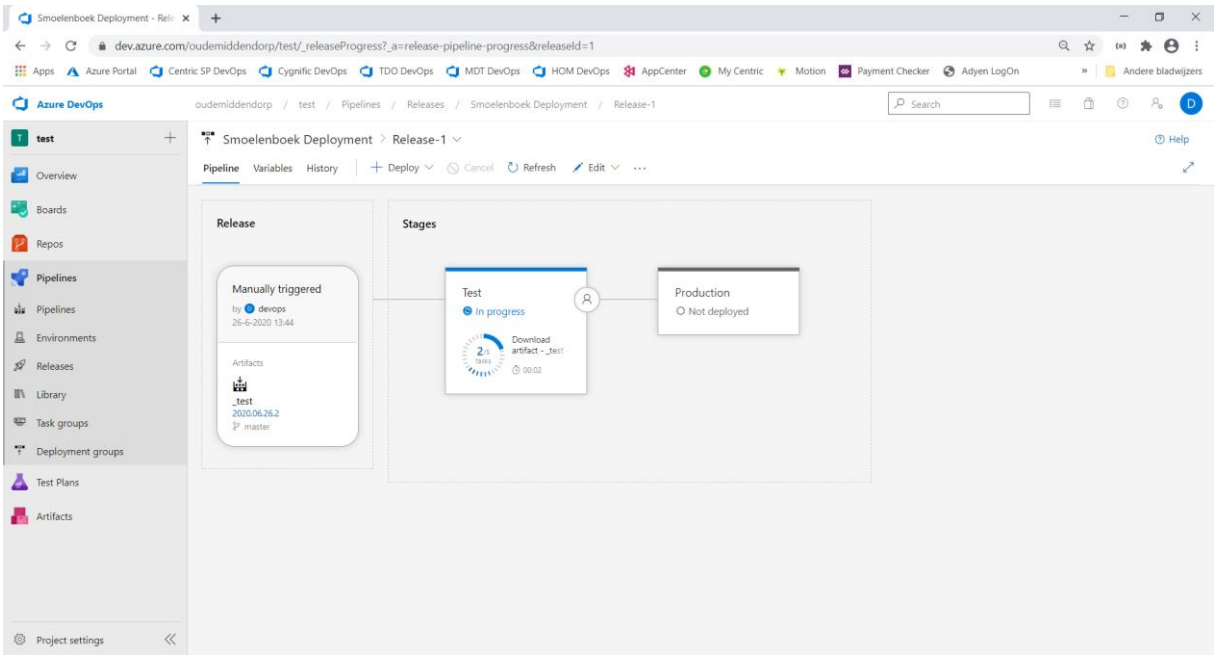
After the pipeline is saved, the “Create release” button becomes available. Click it and then click on “Create”.



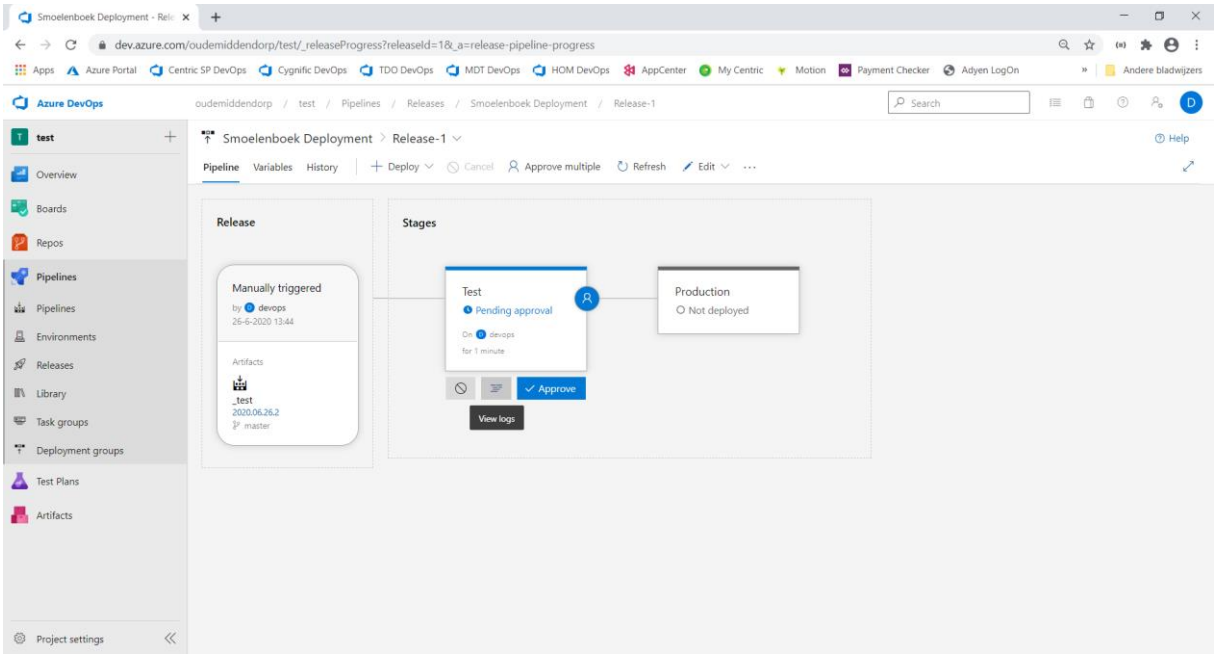
Click on the link in the green box.



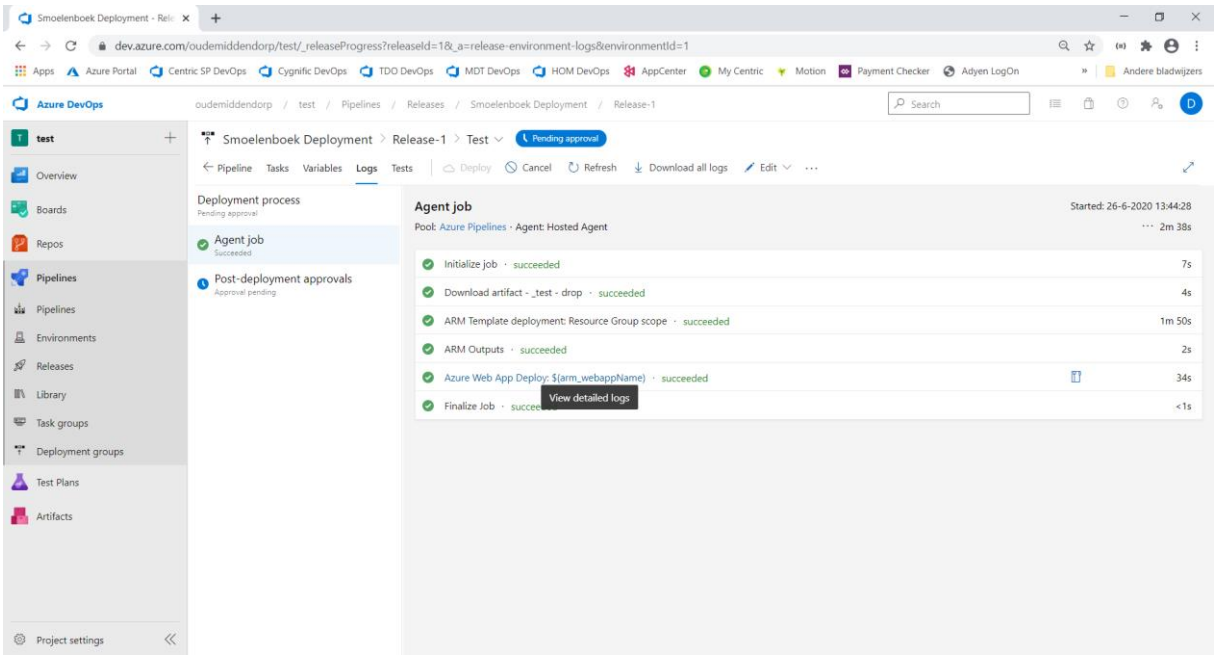
You can follow the deployment process from here.



It will take some time to deploy. When it is ready view the logs.



Click on “Agent job” and inspect the “Azure Web App Deploy …” task.



At the end of the log of this task a URL is given for the web application you created during this deployment. Try it to test if the application works (Ctrl-hover mouse).