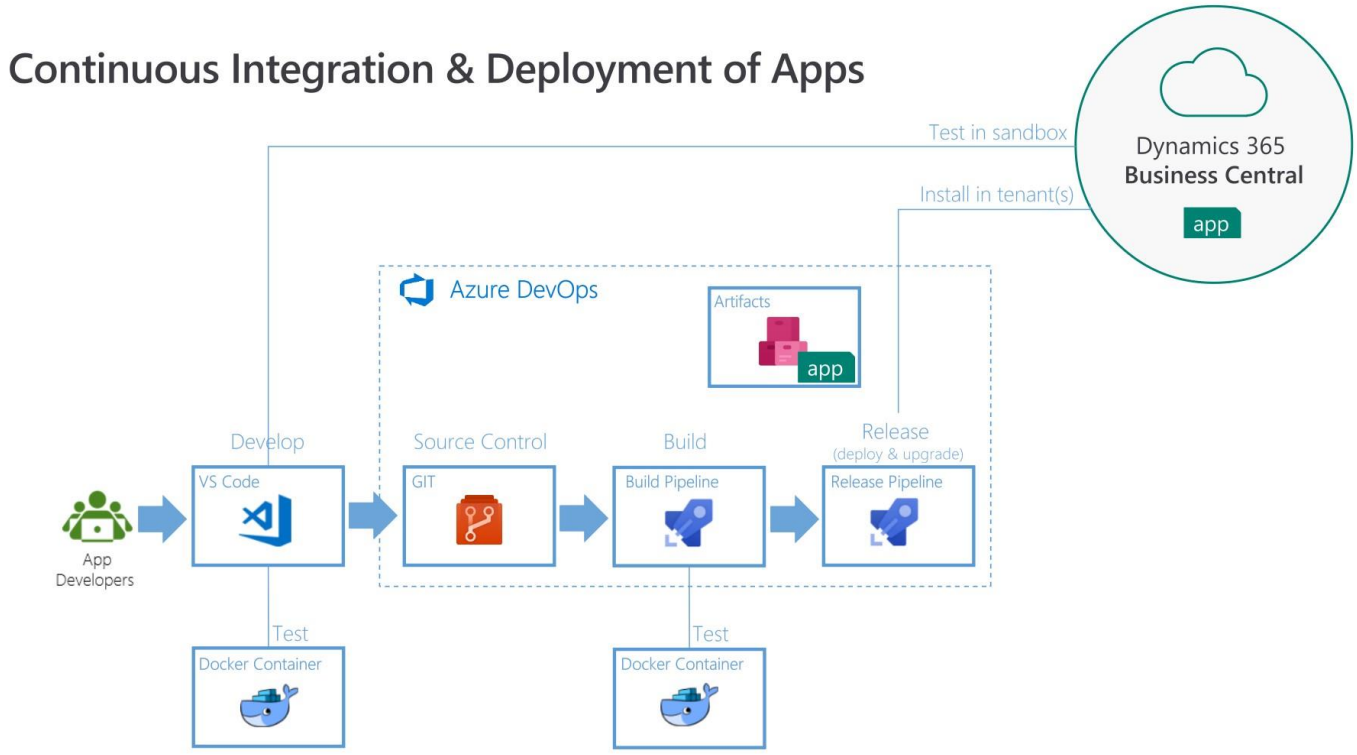
Azure DevOps – CI/CD Workshop



This workshop will help you setup a project on Azure DevOps, including Continuous Integration, Continuous Deployment and requirements for the developer to submit Pull Requests when checking in.

Workshop repositories

This workshop uses two repositories. The scripts and pipelines are very similar, but one is a Per Tenant Extension and using the Per Tenant Extension cop + number ranges and the other one is an AppSource app, using the AppSourceCop + appsource number ranges + prefixes and breaking change notifications.

https://dev.azure.com/businesscentralapps/Old.HelloWorld
or

https://github.com/businesscentralapps/Old.HelloWorld

This is the Per Tenant Extension version of the app.

- Number range defined in app.json is 50100 to 50149.
- PerTenantExtensionCop and UICop are enabled during build (in scripts\settings.json)
- PreviousApps points to .zip file containing previous versions of apps

https://dev.azure.com/businesscentralapps/Old.HelloWorld.AppSource
or

https://github.com/businesscentralapps/Old.HelloWorld.AppSource

This is the AppSource version of the app.

- Number range defined in app.json is 70074169 to 70074218 (my allocated number range)
- Logo and various URL's in app.json set
- TranslationFile feature enabled
- AppSourceCop and UICop are enabled during build (in scripts\settings.json)
- PreviousApps points to .zip file containing previous versions of apps

The per tenant extension version is the easiest to use for the workshop.

Workshop environment

To complete this workshop, you need a computer with at least 16Gb of RAM running Windows 10 or Windows Server 2019 with the latest Windows updates applied, the latest Visual Studio Code update and the latest Docker version installed and running Windows Containers.

If your own computer doesn't meet these requirements, you can create a virtual machine on Azure using http://aka.ms/getbc. After logging into your Azure subscription, you should **at least** specify valid values for these properties:

- Resource group: **<name of resource group>**
- Vm Name: **<name of VM>**
- Accept Eula: **Yes**
- Remote Desktop Access: *** (to allow all IP addresses to connect to remote desktop)**
- Admin Password: **<my admin password>**
- Artifact Url: **bcartifacts/sandbox//us/latest**
- License File Uri: **<secure url to a developer/training licensefile>**
- Final Setup Script Url: **additional-install.ps1 (or full url to script)**
- Contact E Mail for Let's Encrypt: **<my email address>**
- Add Traefik: **Yes (if you want to be able to access containers in the VM from the internet)**

Read this blog post https://freddysblog.com/2017/02/26/create-a-secure-url-to-a-file/ to learn more about how to create a secure url.

Note the Final Setup Script Url, which will invoke the script additional-install.ps1 script after the VM is final, which will install chocolatey and use that to install Git, Microsoft Edge, Google Chrome, Firefox and some VS Code extensions.

The remaining properties can be left to their default values. Accept the terms and conditions and press **Purchase**.

Now, the deployment starts. It will take around ~30 minutes until the VM is ready. You can monitor the deployment status on the landing page (http://<vmname>.<region>.cloudapp.azure.com). The Landing page should state: Installation Complete before you continue.

Prerequisites

Software

If you are using a workshop environment created by https://aka.ms/getbc, you will have this software installed already automatically. If you are using your own computer/laptop, you need to check/install this software:

- BcContainerHelper PowerShell module
- Az PowerShell module
- VS Code
- AL Language Extension
- Git
- Microsoft Edge or Google Chrome

Get an Azure DevOps Account

The Azure DevOps account and organization is where you will create your projects and store your source code. Open https://azure.microsoft.com/en-us/services/devops/ to create a free account. You will be able to create public or private projects in Azure DevOps.

Install and configure GIT

Git is the source code management tool used by Visual Studio Code to connect to your Azure DevOps repository.

If GIT isn't already installed, navigate to https://www.git-scm.com/download/win and click the download link to download and install Git. Select Visual Studio Code as Git's default editor during installation Wizard and select to commit as-is and checkout as-is when asked.

Configure your username and email in git by starting a **Command Prompt** and type:

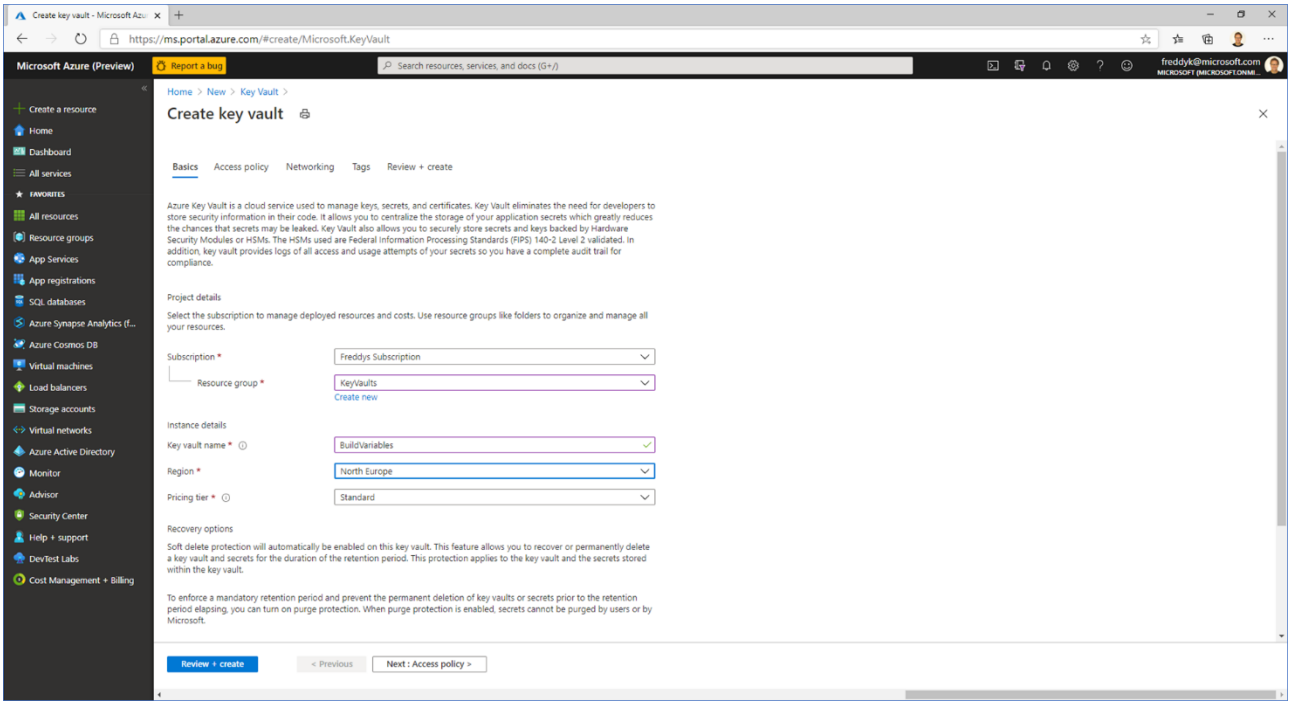
```
git config --global user.name "<your username>"
git config --global user.email "<your email>"
git config --global credential.helper manager
```

You should use the same email here, as the one used for your Azure DevOps Account.

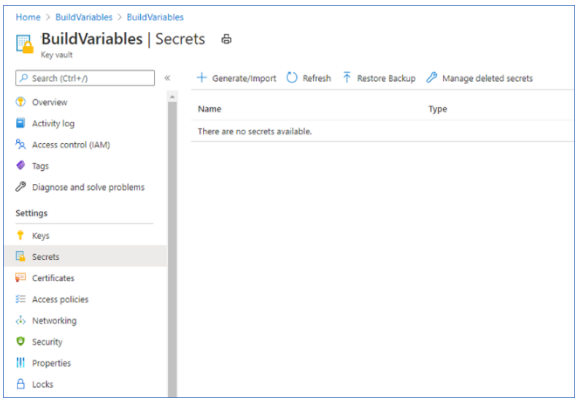
Create a Key vault for your secrets

Secrets belong in key vaults and since this workshop will contain license file secrets, passwords and a shared access signature for insider builds, the key vault is needed.

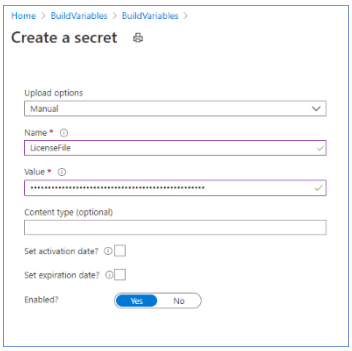
Navigate to https://portal.azure.com and login to your subscription. Click Create a resource and select Key vault. Fill out the values as needed and create the Key Vault.



Navigate to the Key Vault resource and navigate to secrets:



Click Generate/Import and fill out the values:



And press create.

For this workshop you need to create a licensefile secret with a secure url to your license file and a password secret.

In PowerShell, accessing these secrets are very simple using the Az PowerShell module.

First, you need to connect your Windows User to your Azure Account, run:

```
Connect-AzAccount
```

login to your Azure Account and now you can use:

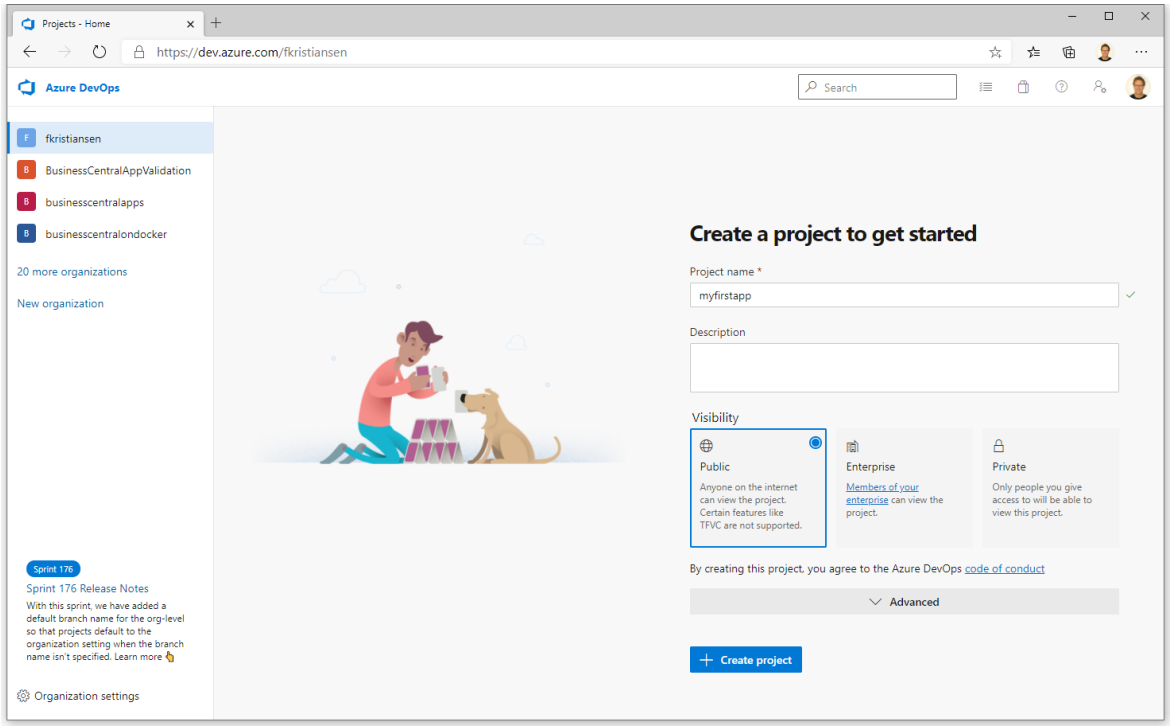
```
$vaultName = "BuildVariables"
$passwordSecret = Get-AzKeyVaultSecret -VaultName $vaultName -Name "Password"
$password = $passwordSecret.SecretValue
```

To read the password as a SecureString. If you want to see the actual password, you need to convert the password to text, which can be done using:

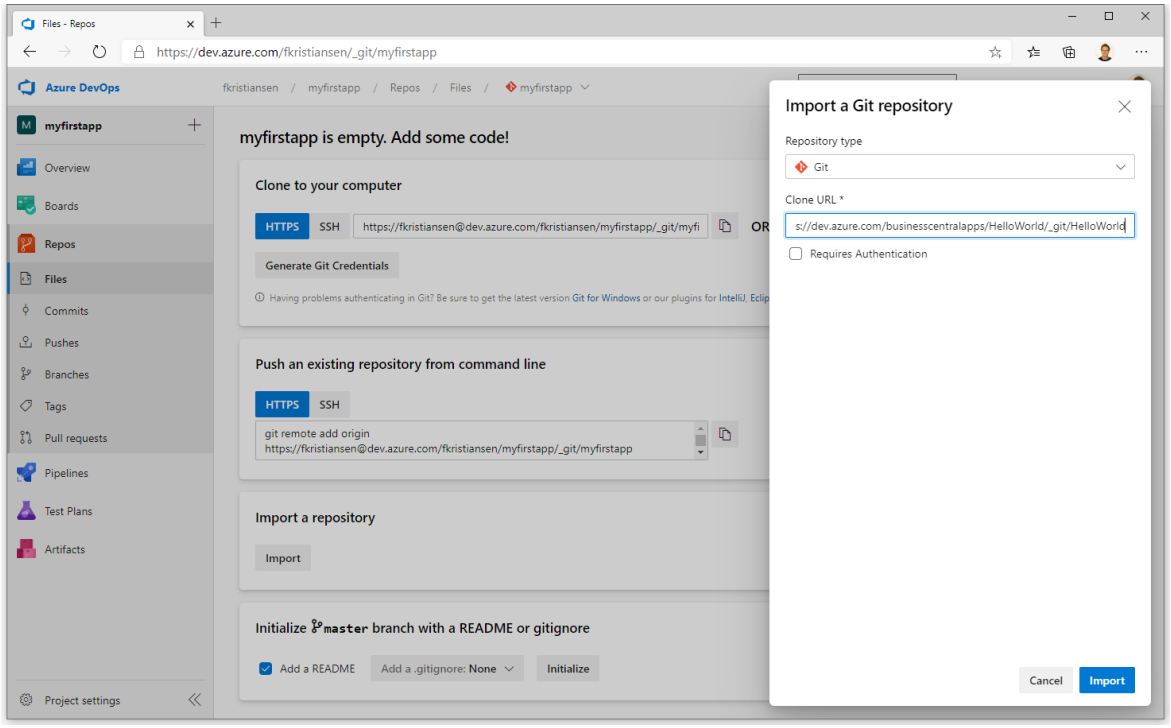
```
[Runtime.InteropServices.Marshal]::PtrToStringAuto([Runtime.InteropServices.Marshal]::SecureStri
ngToBSTR($Password))
```

Create your organization and your first project

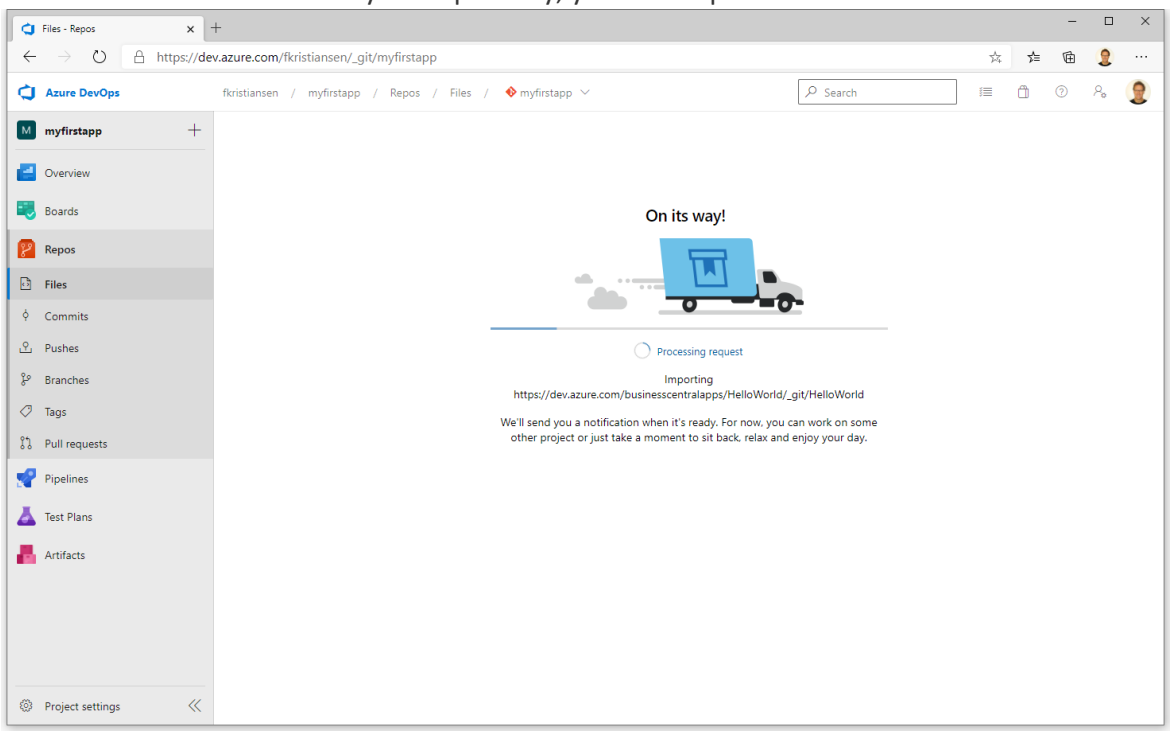
Navigate to https://devops.azure.com and login to your DevOps account. Create your organization, which is the location in which you will create your projects. In your organization, create your first project:



In the project navigate to the **Repos** -> **Files** area, click **Import** and enter https://dev.azure.com/businesscentralapps/Old.HelloWorld/_git/Old.HelloWorld (or Old.HelloWorld.AppSource for the AppSource version) in the Clone URL field. The sample repository is also available on github here: https://github.com/BusinessCentralApps/Old.HelloWorld (add .AppSource for AppSource)



After the truck has delivered your repository, you can inspect the content.



Inspect the content of the repository

The repository consists of 4 project folders: **base**, **app**, **test** a **scripts** folder.

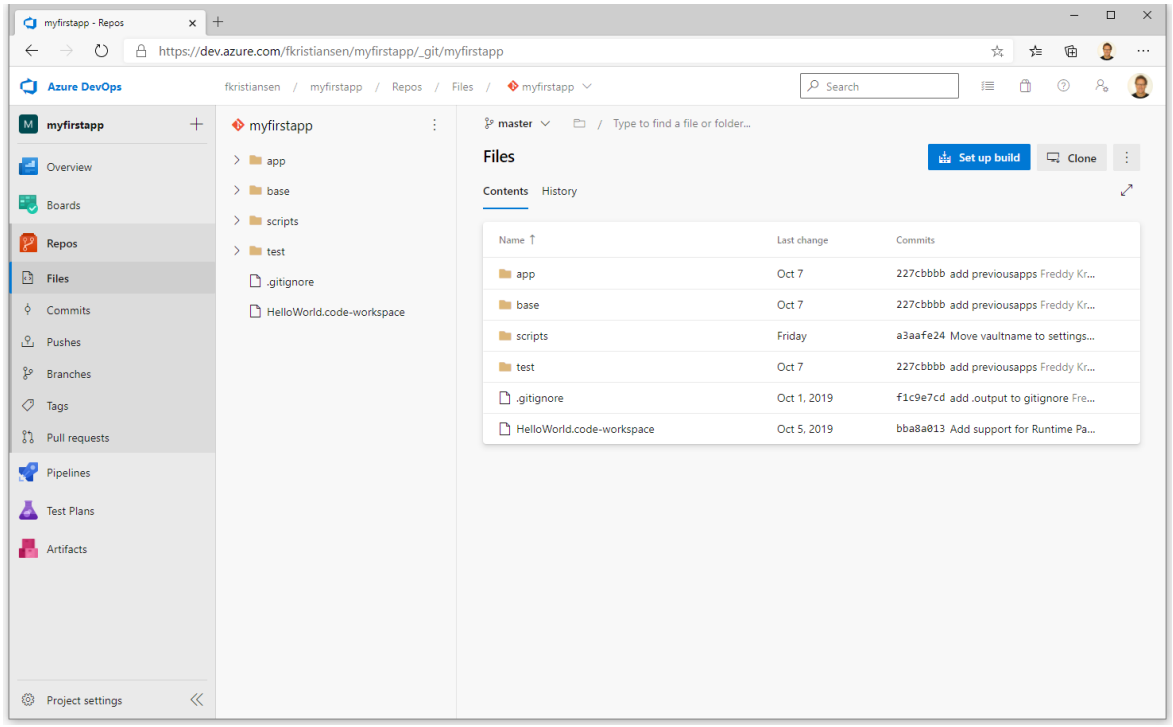
base contains a single codeunit with a single function, which returns **App Published: Hello World Base!**

app has a dependency to **base** and consists of a **Customer List Page Extension**, which will pop up the Hello World message on the **OnOpenPage Trigger**.

test is the test app with a dependency to **app**, containing **a single test**, which opens the **Customer List** and tests that the **Hello World message appears**. **scripts** is a set of scripts/files used for **CI/CD** and setup of dev environments.

The **.gitignore** file is known to everybody who are using GIT as a description of which files GIT should ignore.

The **HelloWorld.code-workspace** is the workspace you want to open with VS Code.

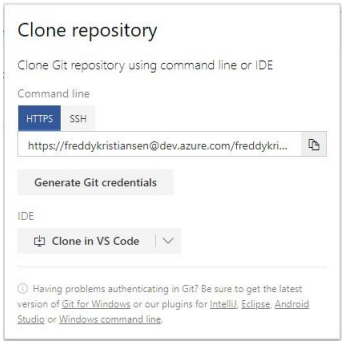


Note: The template will constantly be changed/improved, and the content of the template repository might vary.

If you want to add multiple apps to the project, the idea is to create folders for each app in the root folder.

Clone the project

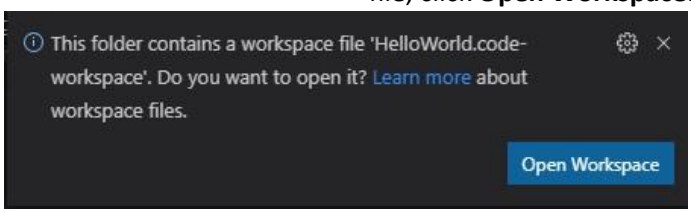
In order to work with the project, we need to clone the project to our work machine. You can use the Workshop VM as work machine, or you can use your personal computer/laptop.



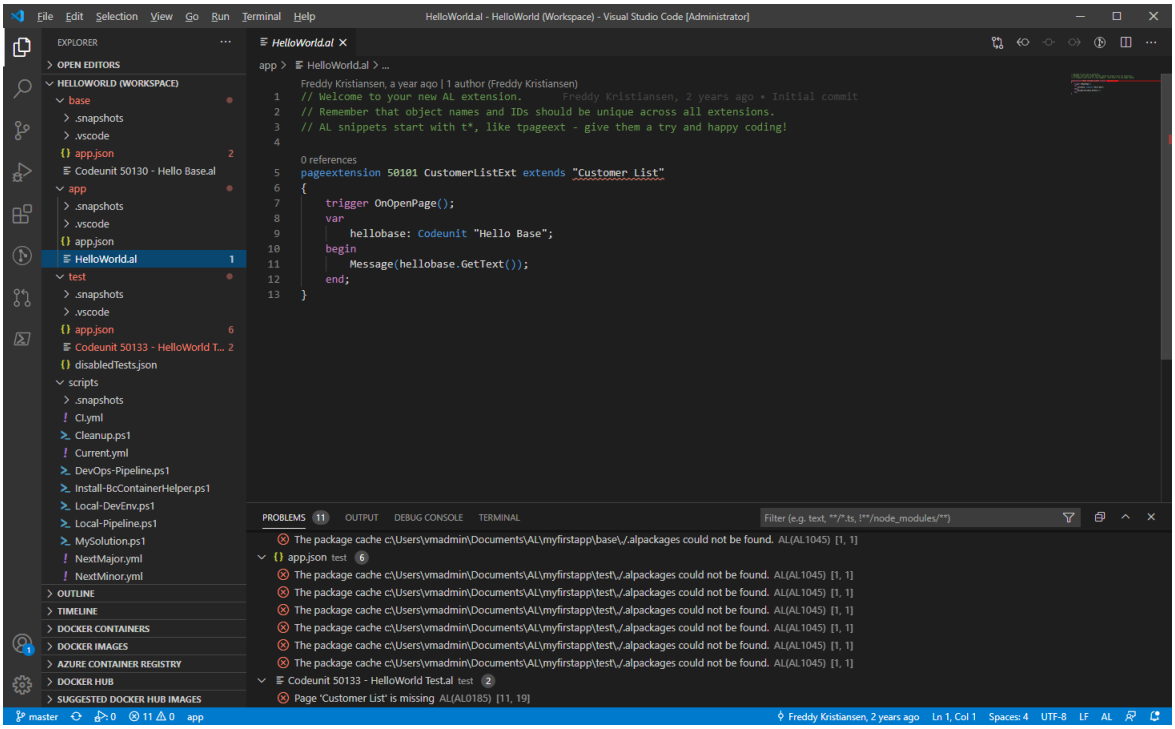
Make sure VS Code is running and click the **Clone** button in the upper right Corner and select **Clone in VS Code**.

Allow the browser to Open VS Code and select a location (f.ex. Documents\AL) for the repository and sign-in to your Azure DevOps account if asked to do so. Say Yes to open the repository.

After opening the repository, VS Code asks whether you want to open the workspace file, click **Open Workspace**.



And you should “almost” be ready to start working:



Make it your project

The project has been setup with some default **object ids**, **app ids**, **publisher** and **name**. Since this workshop is going to deploy your app to a cloud tenant, you should change these.

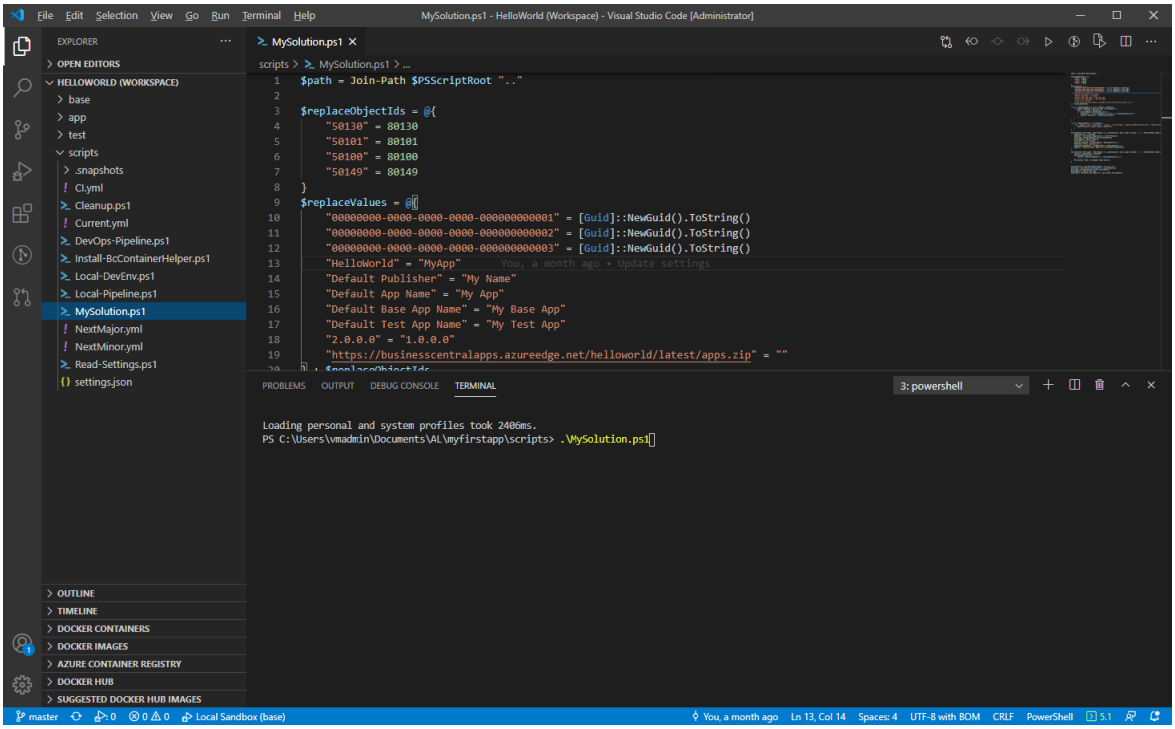
You can either fix the solution manually or you can run a small script to fix the solution automatically.

Fix the solution manually

You can modify app.json in the base project, the app project and the test project manually, setting the app id, app name, app publisher and app versions. Remember that the app project has a dependency on the base project and the test project has a dependency on the app project. Your ids must match. If you are using the AppSource version of HelloWorld, you should also modify the object ids.

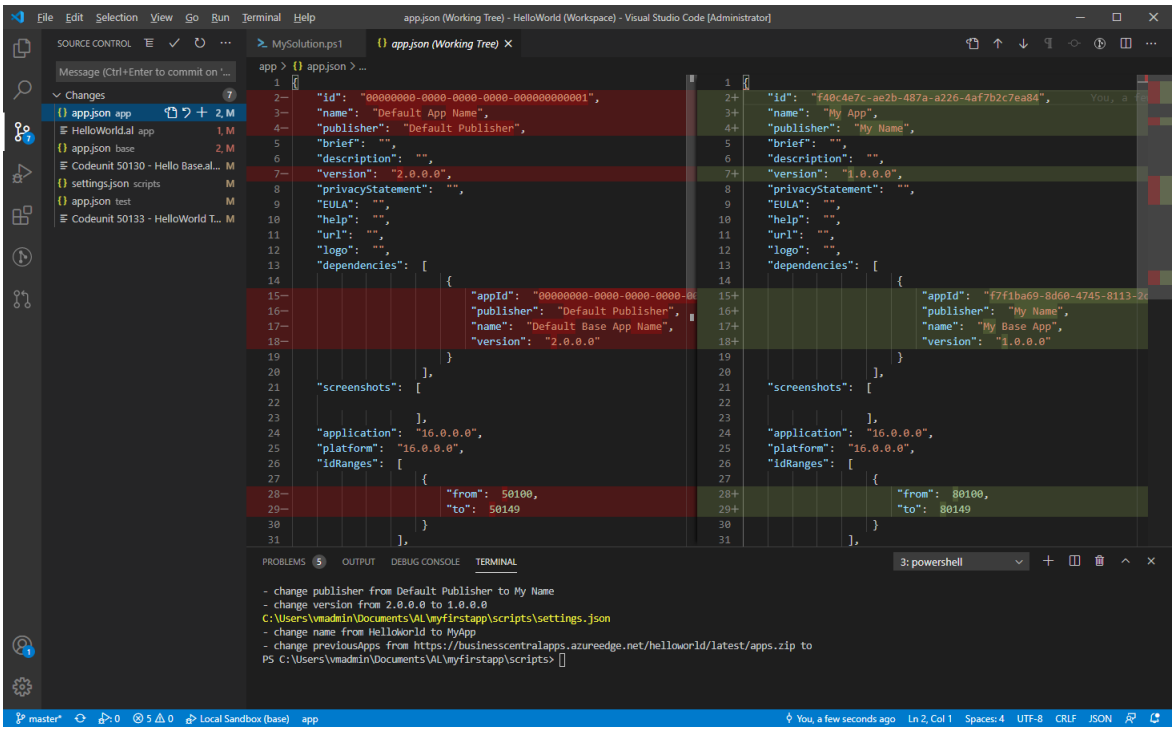
Fix the solution automatically

In VS Code, open the **scripts** folder and open the **MySolution.ps1** file.



Modify **$replaceValues** array if needed to indicate the values you want to use on the right side, save the script and run the script. **Note:** Running the script can be tricky, I cannot get F5 to work…, so right-click the PowerShell file and **Open in Integrated Terminal** – or use **Ctrl+Shift+P** and **Open Current File in PowerShell ISE**.

You should see an output indicating which files are modified and if you click the source control symbol, you should see which files have changed and by clicking a file, you can see the changes.



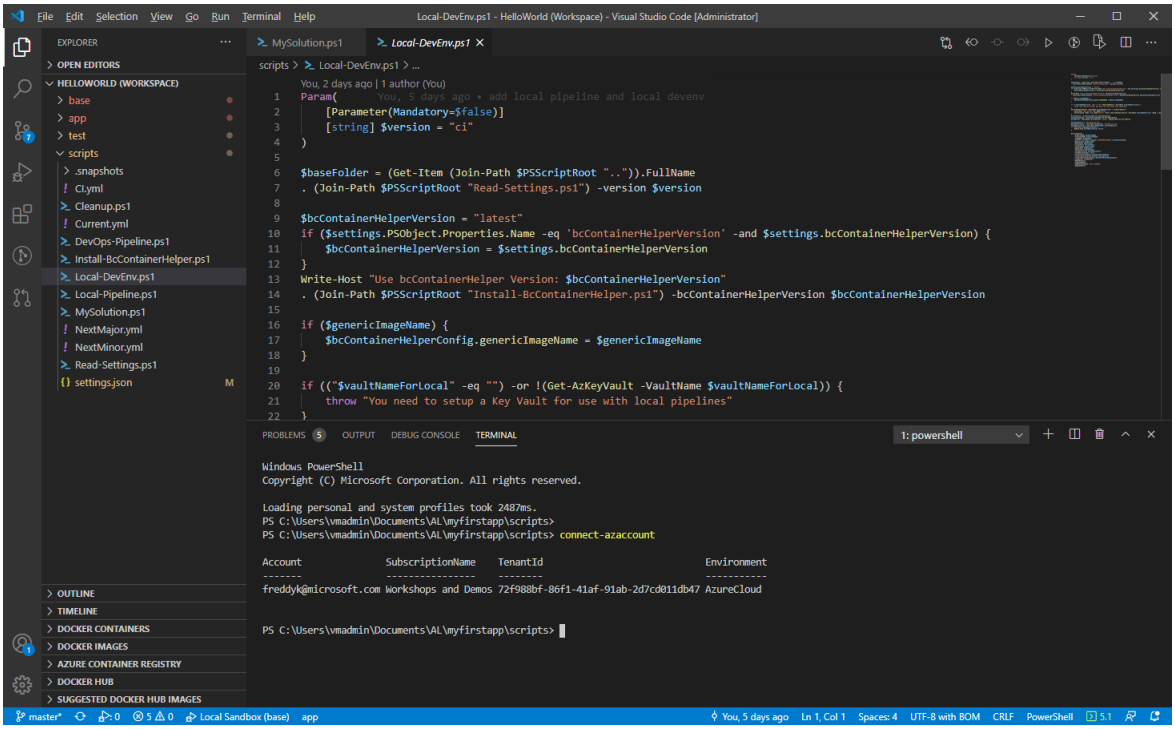
Build the solution manually

To get into a situation where RAD (Rapid Application Development works), you need to create a development environment and build+publish your projects. Starting with the base, you would need to download symbols, compile and publish all apps after creating a container with the right version.

Build the solution automatically

In VS Code, open the **scripts** folder and open the **Local-DevEnv.ps1** file.

Open the PowerShell terminal and run **Connect-AzAccount** to allow PowerShell access to your Azure Account and **Select-AzSubscription** to get access to the Subscription where your KeyVault was created.

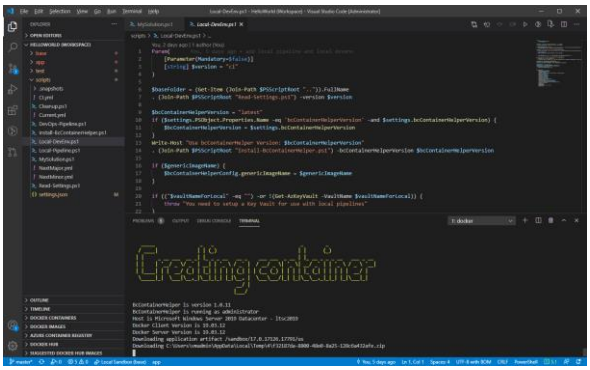


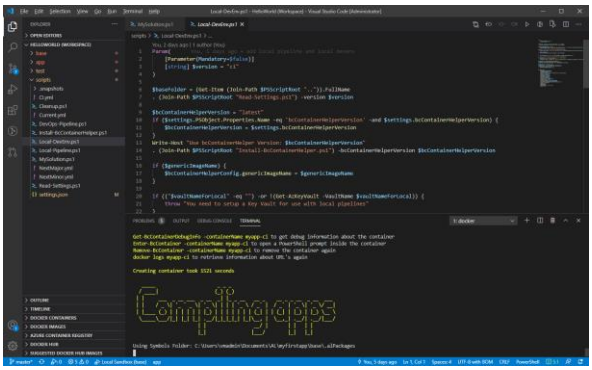
Now run **.\Local-DevEnv.ps1**

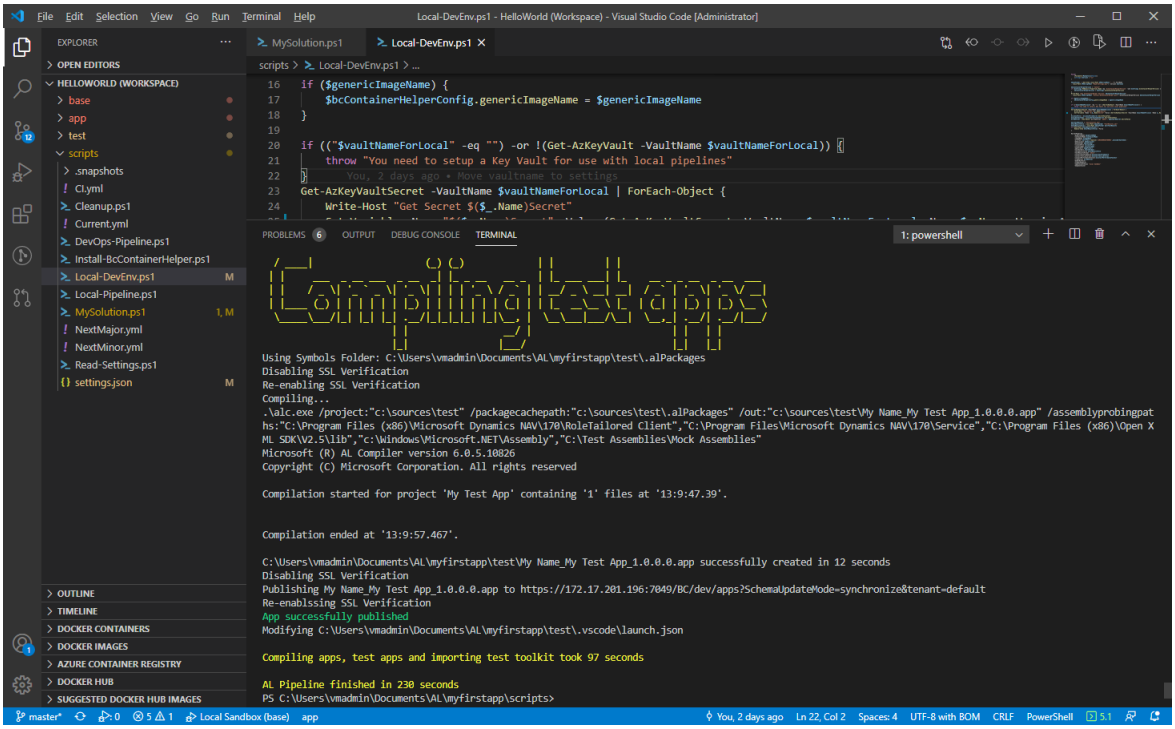
This script will create a devopment container, compile and publish all apps using the Dev Endpoint and leave the container running, allowing you to modify and publish individual apps afterwards.

Note: that the local-devenv script assumes that you have a key vault called **BuildVariables** with at least a **licensefile** and a **password** secret.

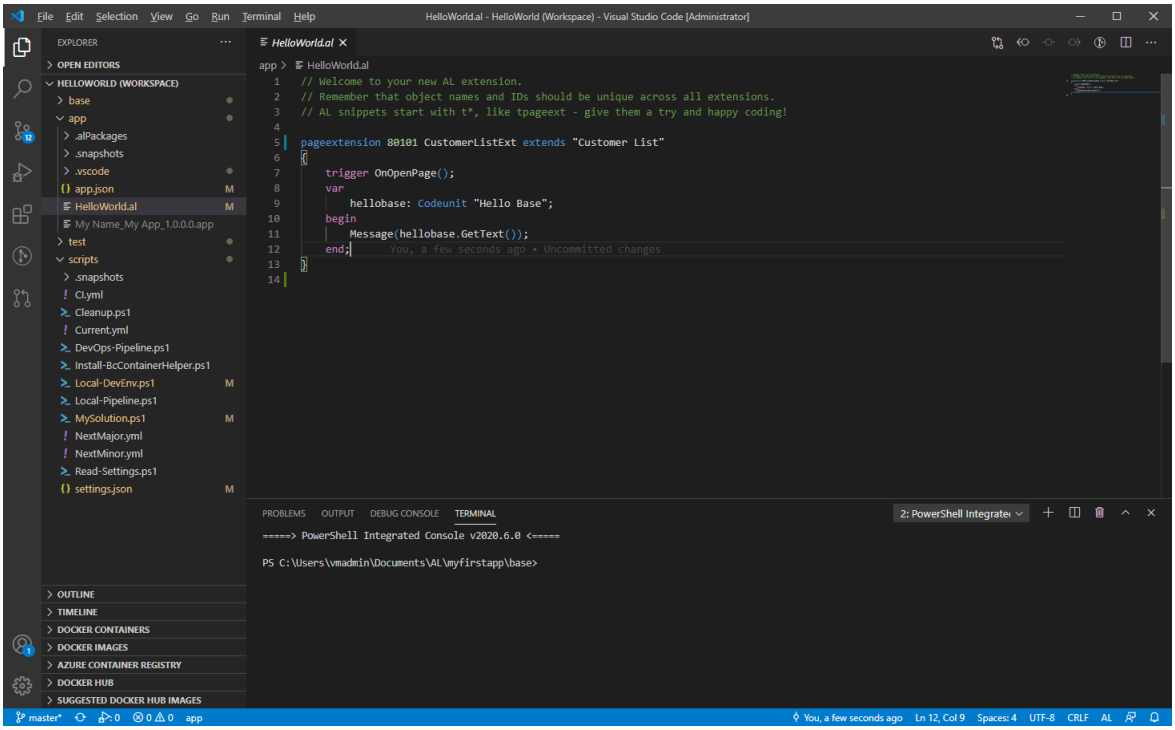
You should see the script running, creating the container, compiling and publishing apps and modifying the launch.json file to prepare your VS Code for Rapid Application Development.



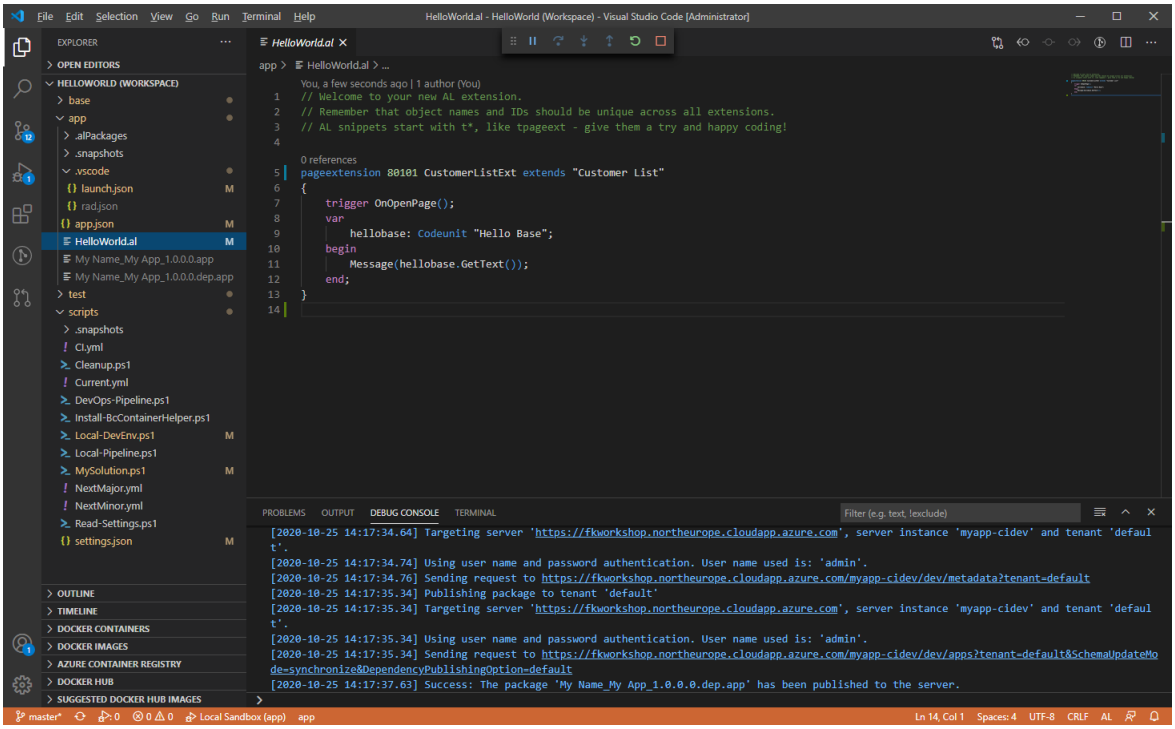




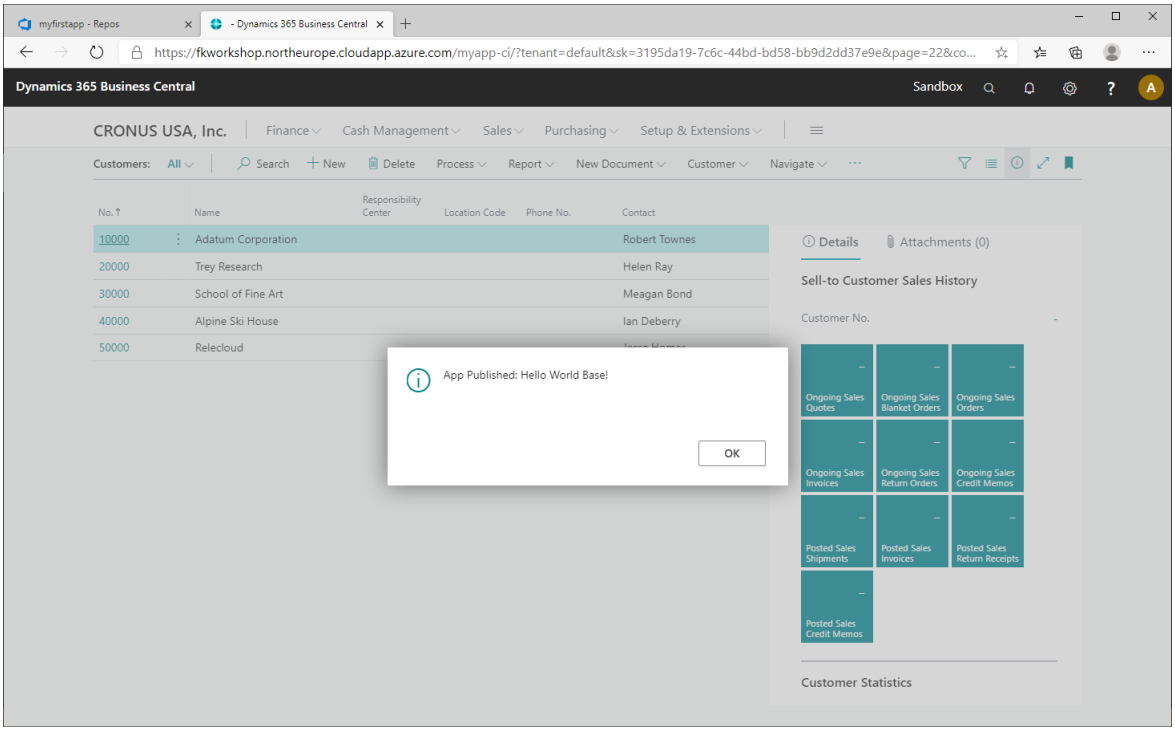
Press Ctrl+Shift+P and execute Developer: **Reload Window to get rid of cached compiler errors.**



Now you can navigate to HelloWorld.al and press F5.



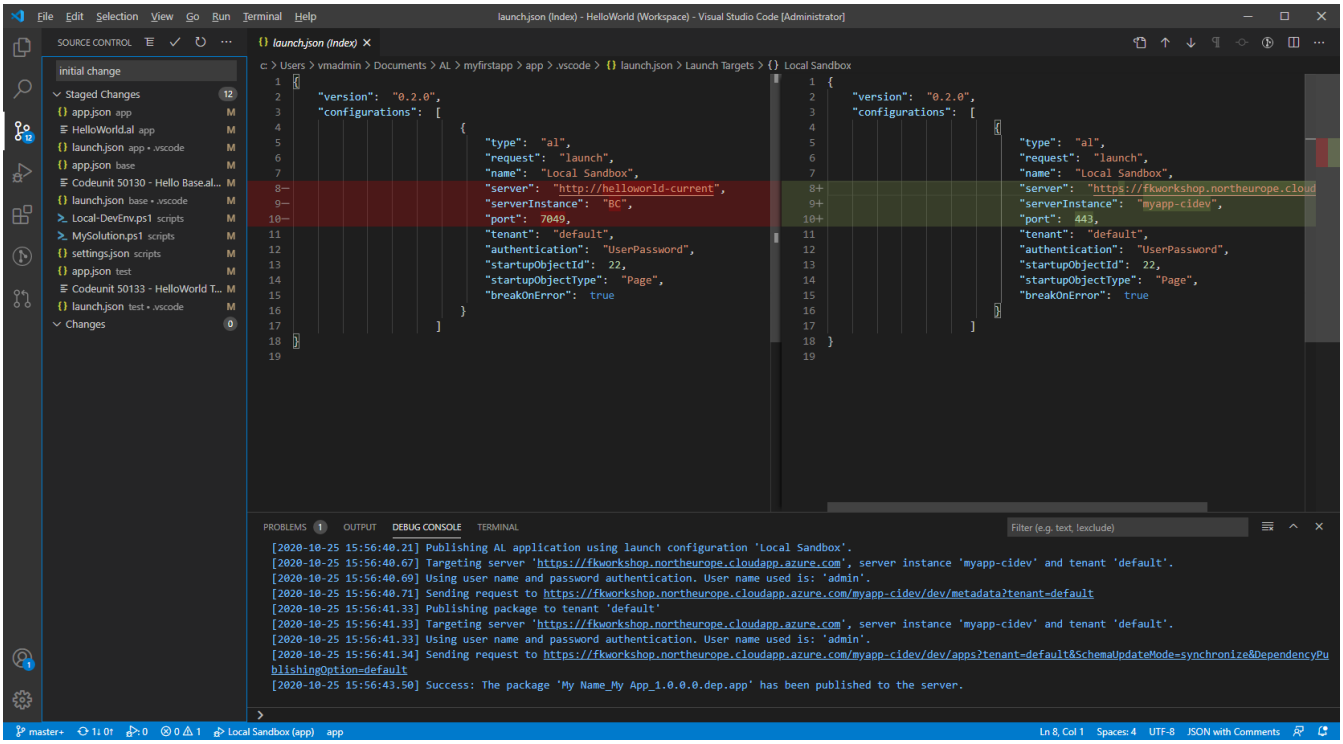
And then it should automatically launch the Web Client.



Check-in your changes

Typically, we do not check-in **launch.json**, in VS Code, you can add these to **.gitignore** in order to avoid tracking them if you like, in this workshop we will just check in everything.

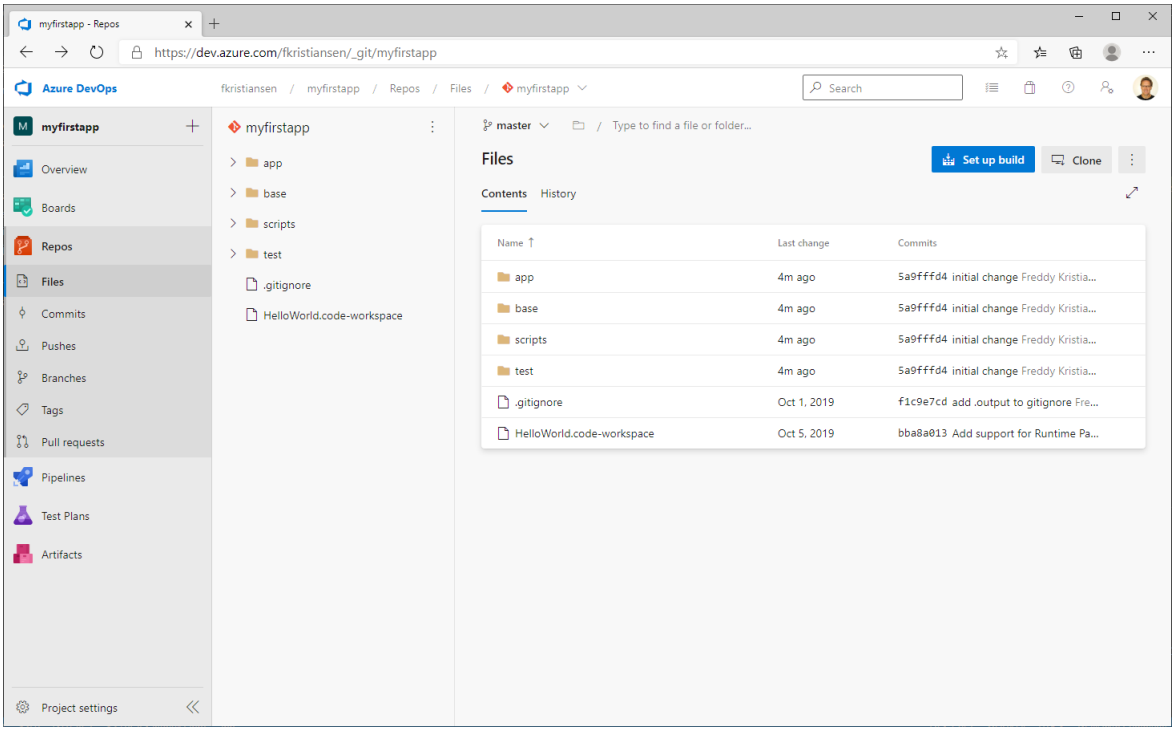
Click the Source Control icon and press **+** on the changes line to stage all your changes.



Staged changes are changes you want to commit. Enter a commit message and hit Commit (or use **Ctrl+Enter**). In the bottom left corner, identify the **Synchronize Changes** symbol and hit that. You might be asked to login to your devops account in order to push your changes.

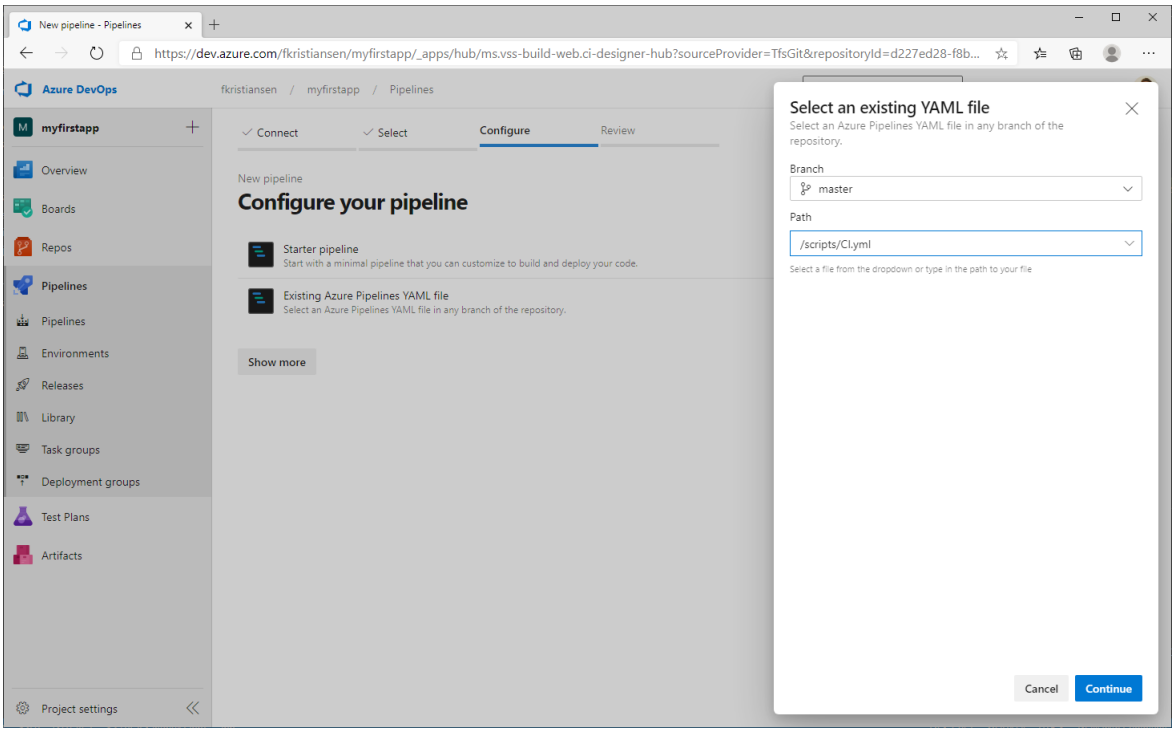
Create a Build Pipeline

In **Azure DevOps**, under your **Project**, **Repos** -> **Files**, you will see a button called **Set up build**. **Click it**.

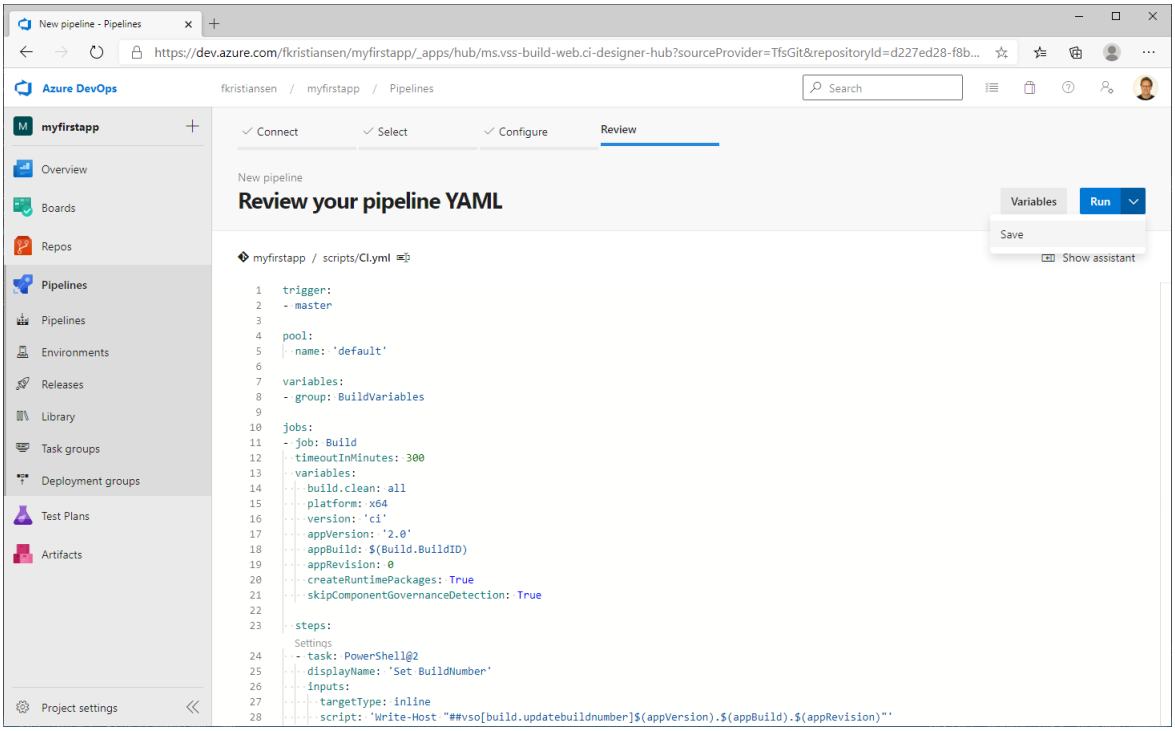


In the **Configure your pipeline**, select **Existing Azure Pipelines YAML file**.

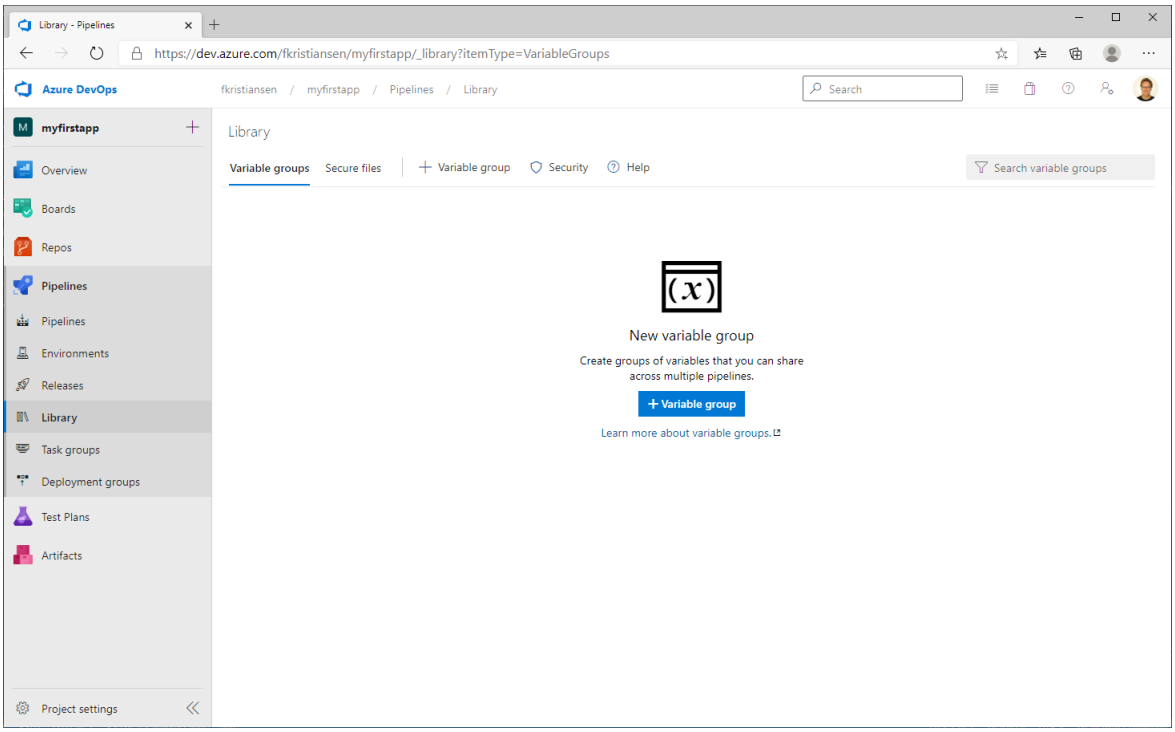
Select **/.azureDevOps/CI.yml** in the path for the YAML file and press **Continue**.



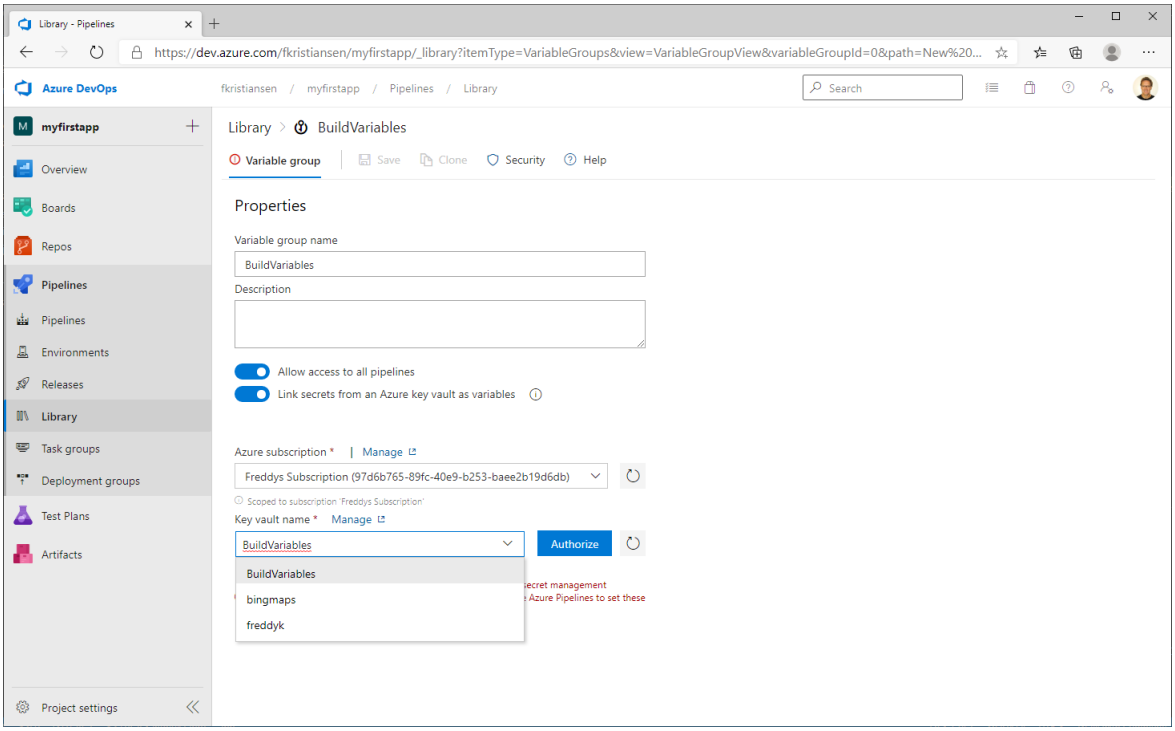
In the Review your Pipeline YAML, click the arrow next to the Run button and Save the pipeline.



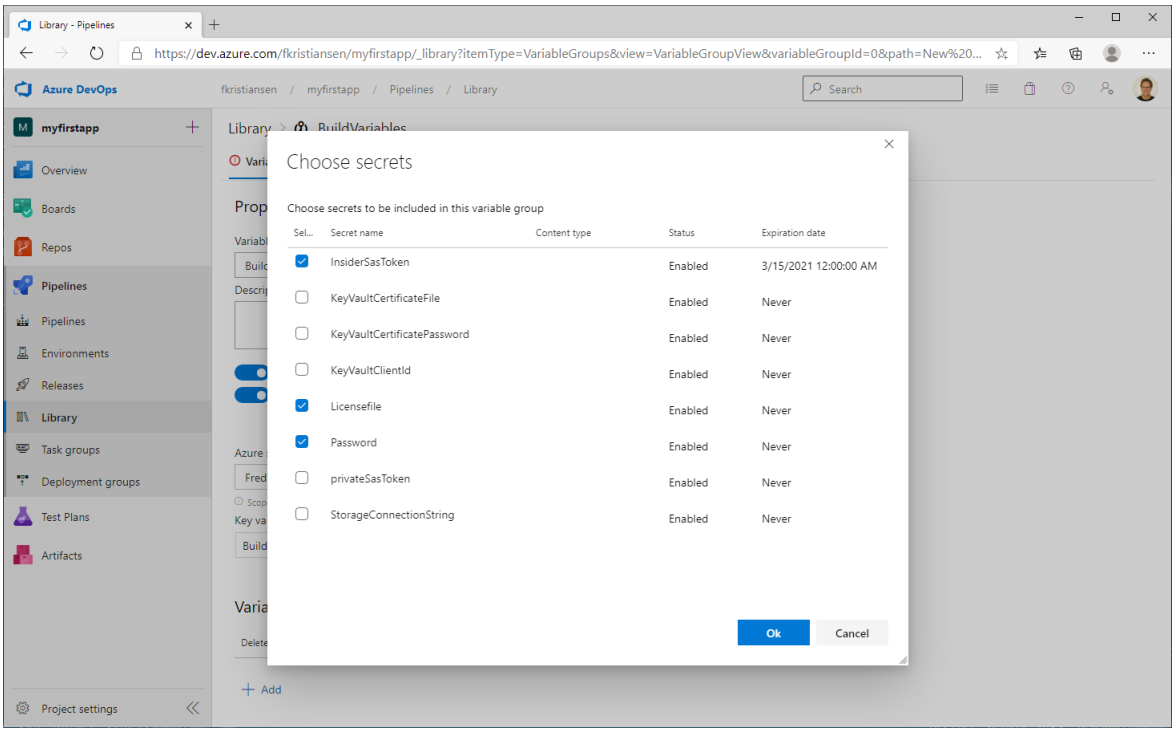
The pipeline needs access to your Key Vault secrets. Navigate to Library under Pipelines:



Create a variable group called BuildVariables. Link Secrets from an Azure Key Vault as variables and authorize the pipeline to access your subscription and your Key Vault:



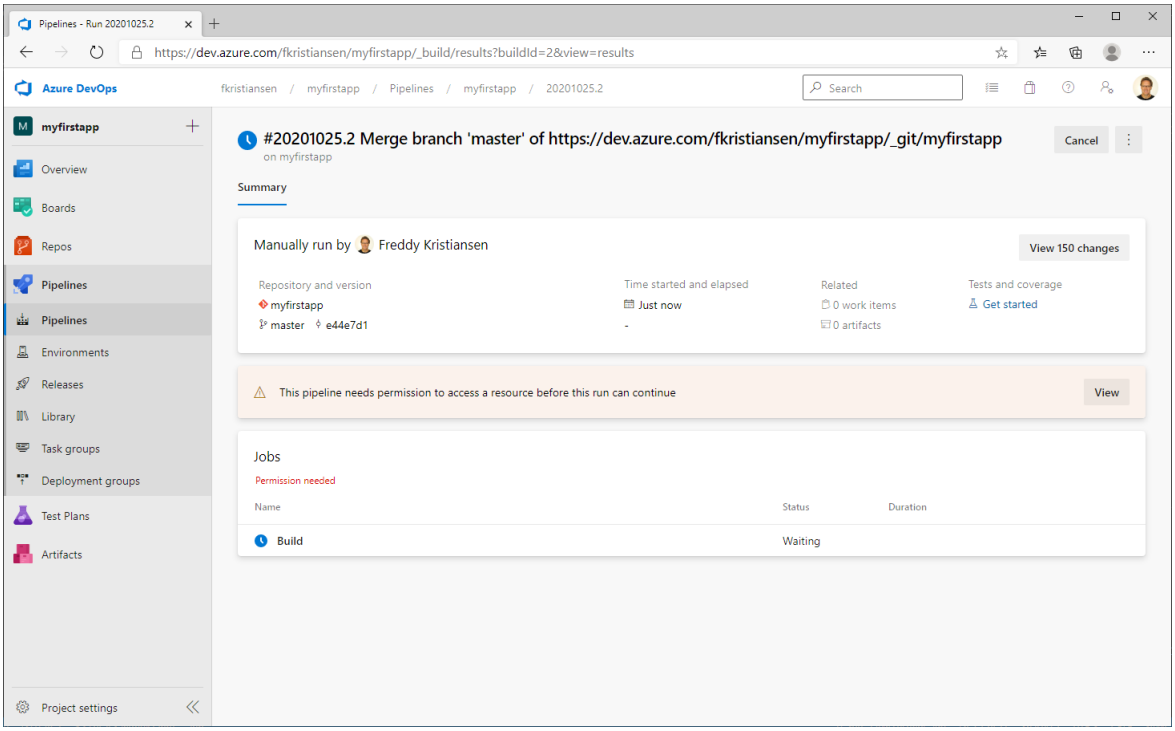
After that, add the variables needed by the pipeline:



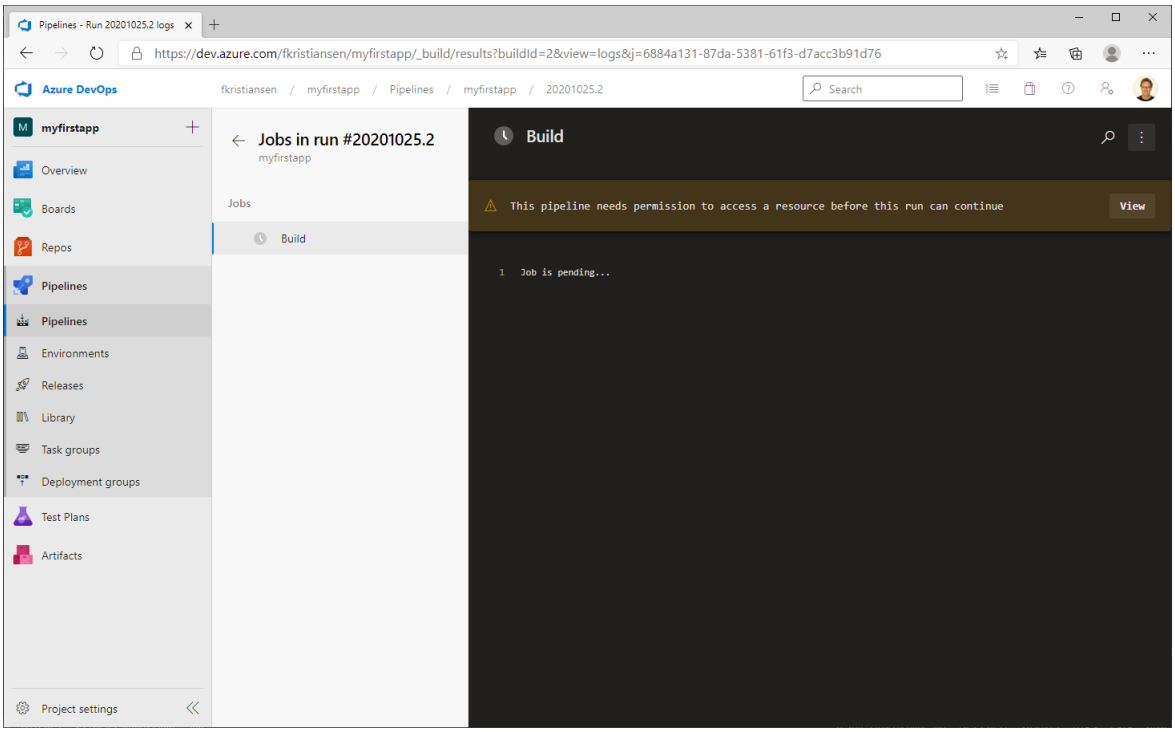
Click Ok and Save!

Go back to pipelines, click your pipeline and select **Run Pipeline**.

The pipeline will ask for permissions to access a resource (your Key Vault) before running. This can either be in the pipeline window, like here:

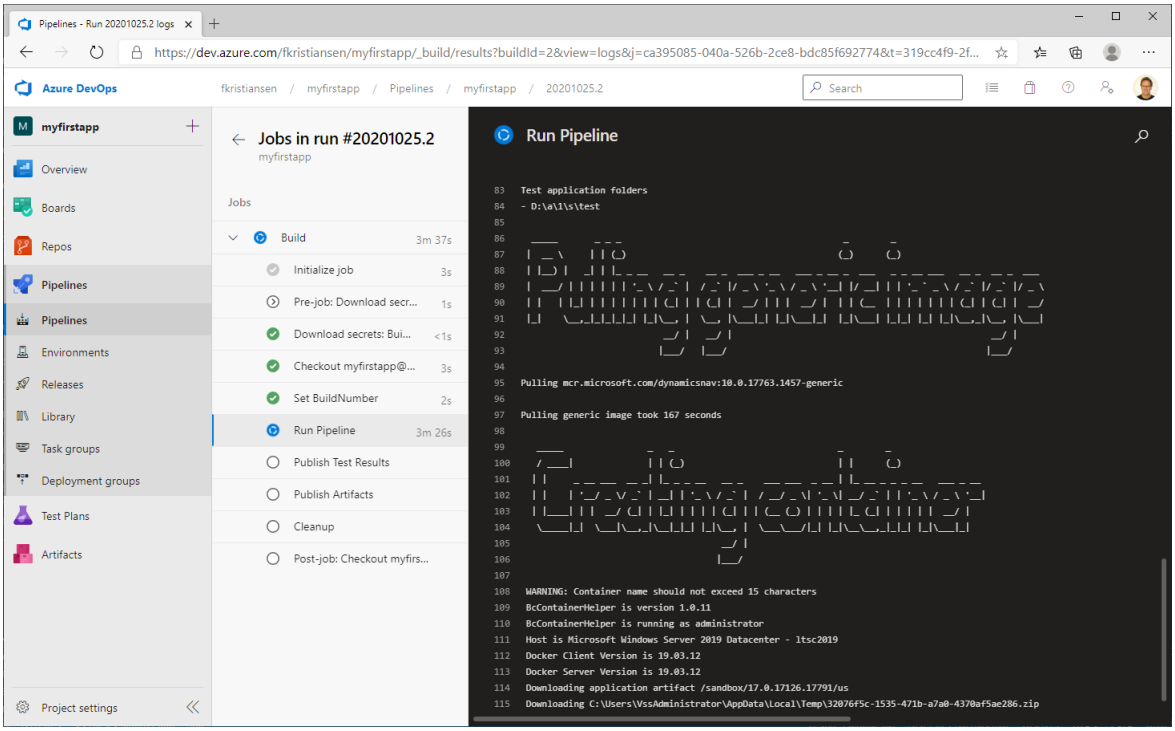


or inside the actual build window, like here:

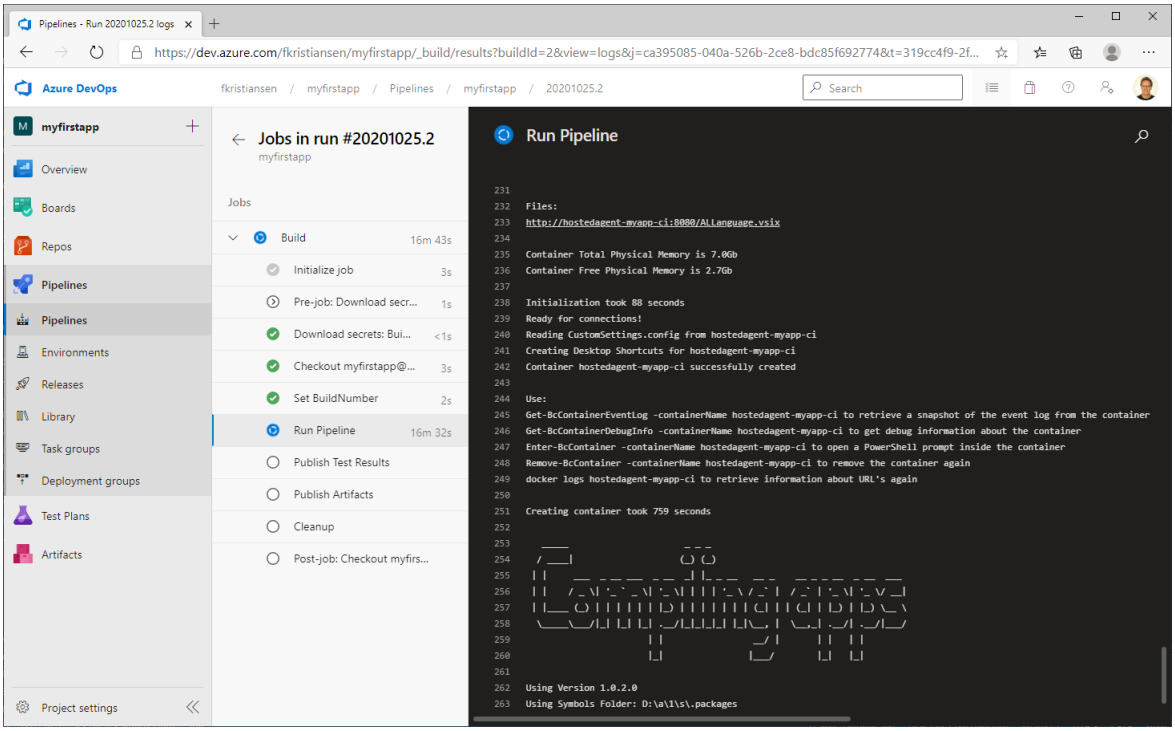


Click **view** and **grant permissions.**

Follow the progress of the pipeline by clicking the pipeline:



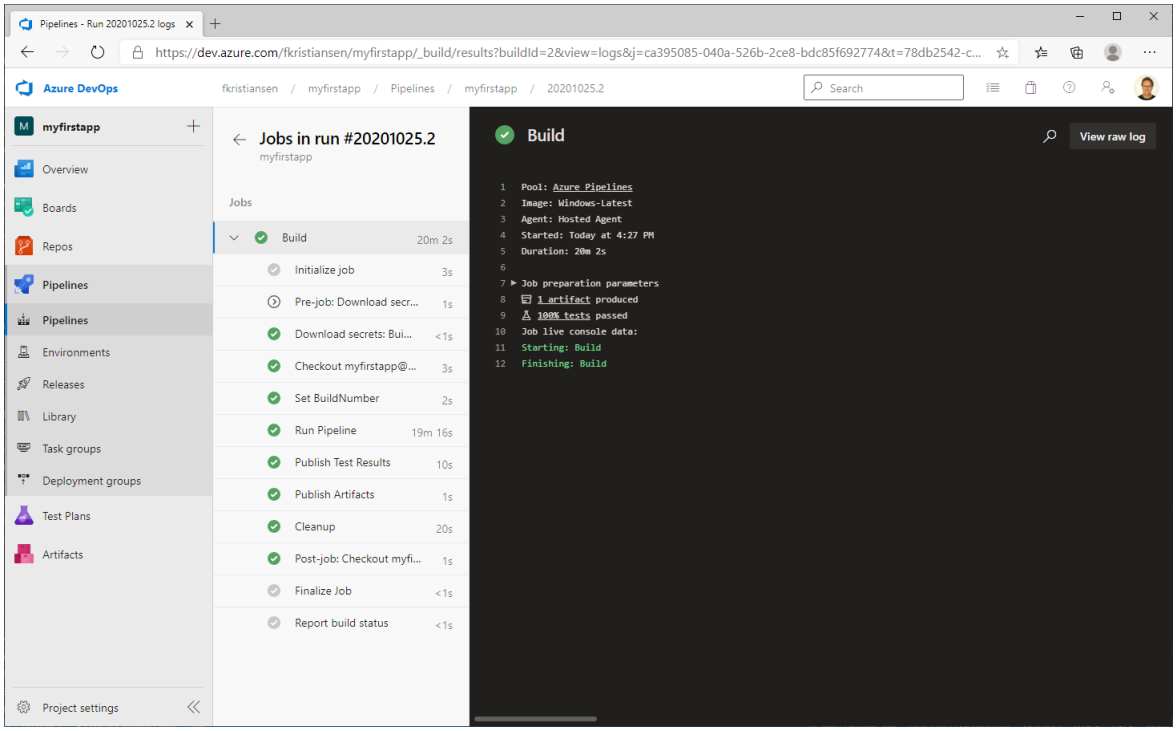
The Run Pipeline is the task, which basically performs the same steps as you did when running the local devenv pipeline earlier and you will see similar output.



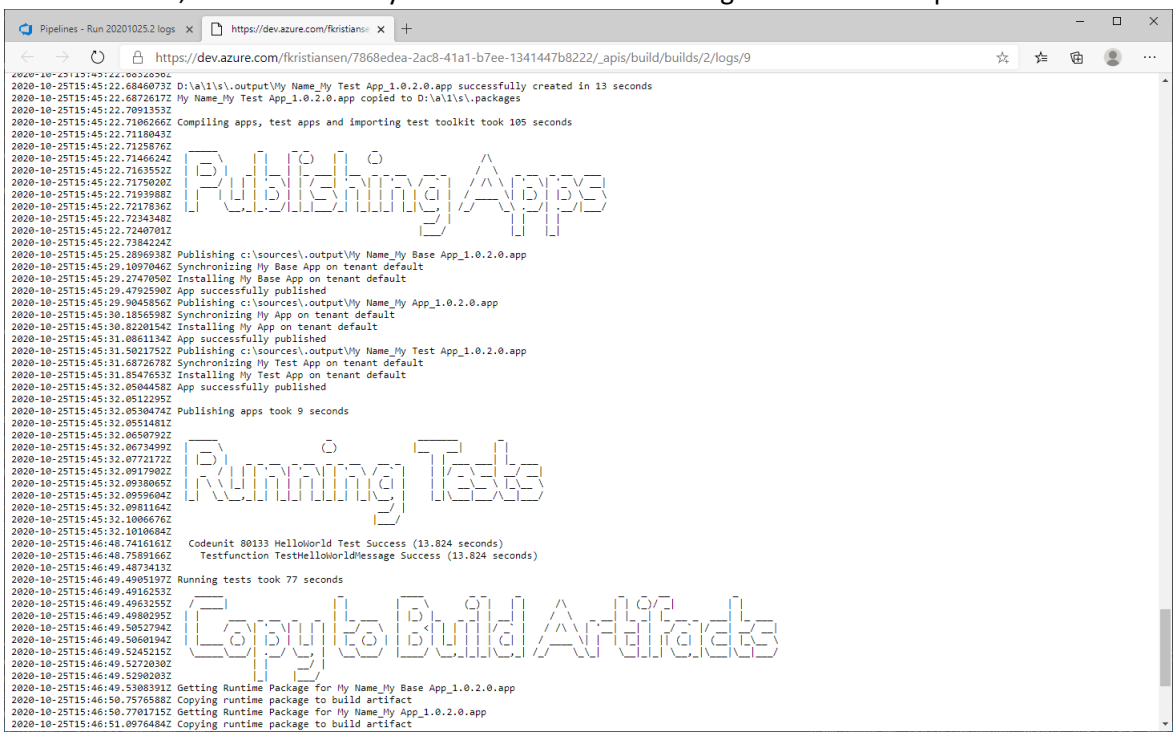
The pipeline is by default setup to use Azure Hosted Agents, which will take approx. 20 minutes to complete a build and nothing can really be reused between builds.

You can also setup your own build agents (either on an Azure VM or a local computer) for use with the build, which will greatly increase speed but will also come with a cost of running and maintaining that machine.

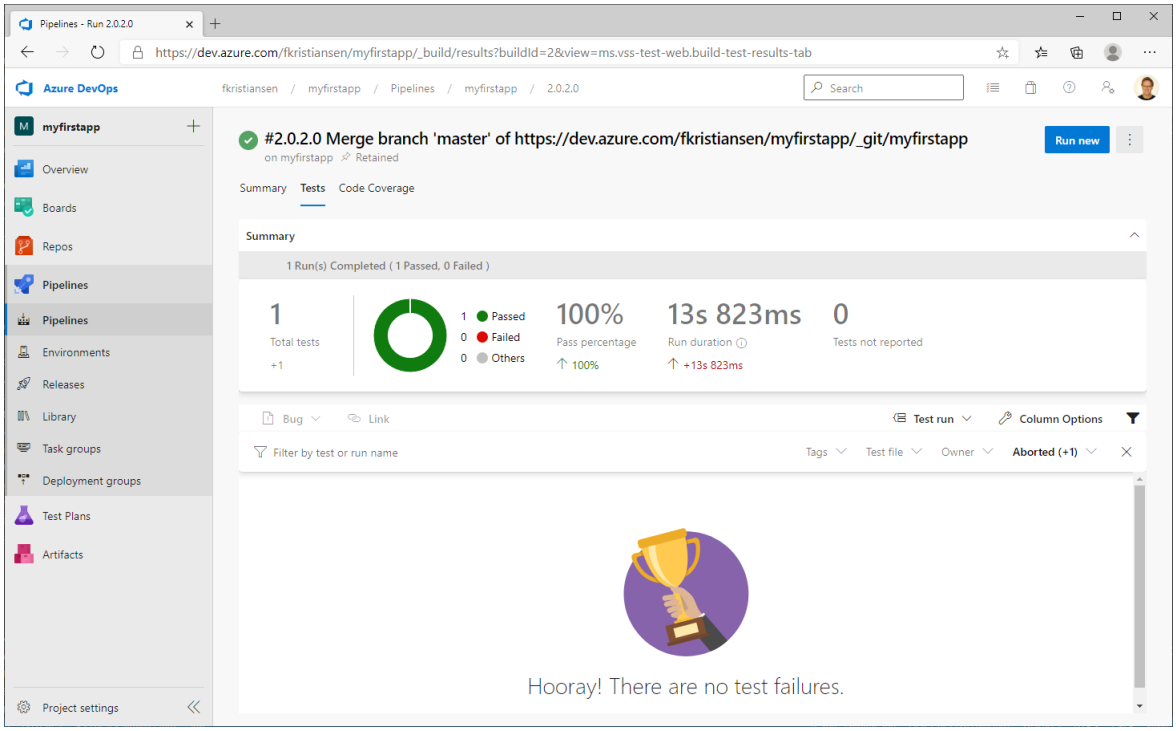
When the build is complete, you should see:



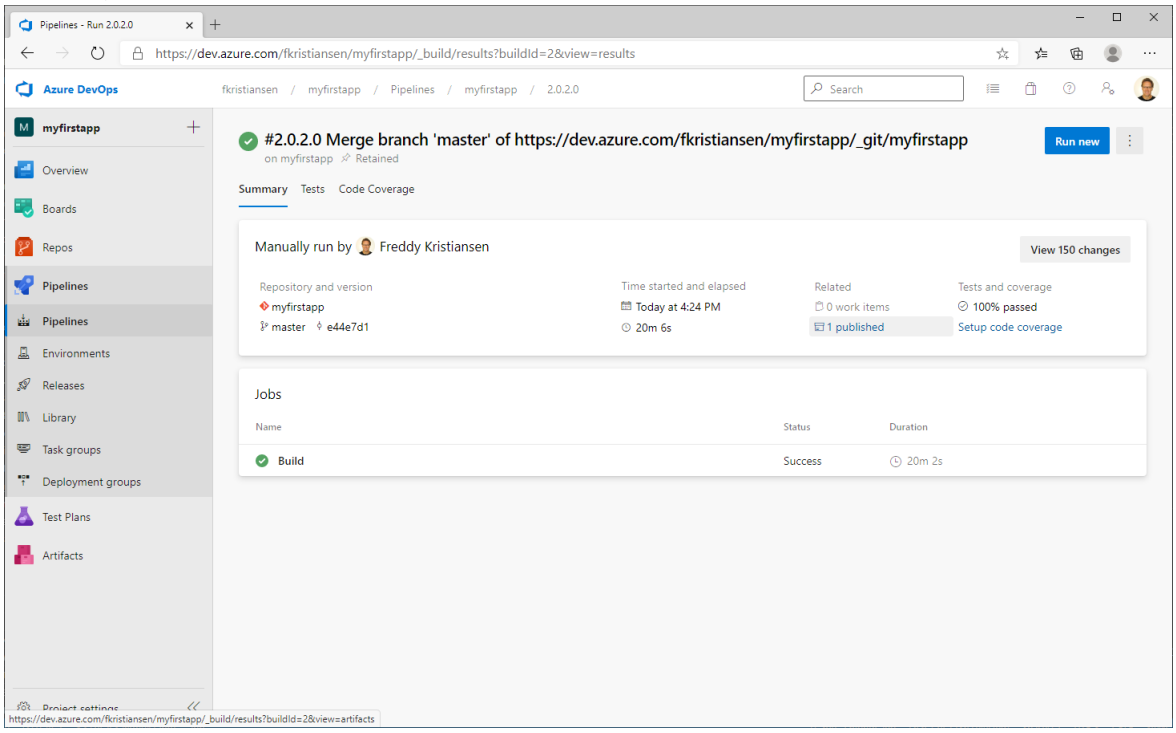
And clicking **view raw log** on the **Run Pipeline** will give you a nice output of the pipeline, which also can be used to search for into, and should always be included when creating issues on Run-Pipeline.



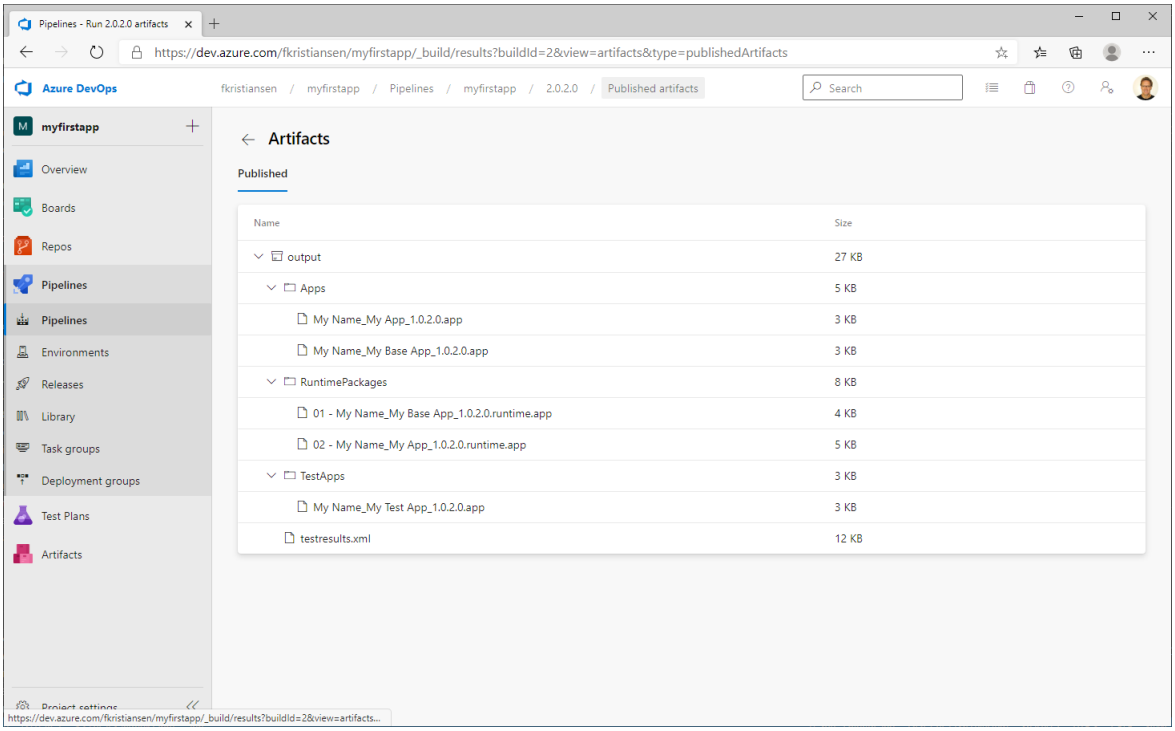
You can inspect the test results:



And in the build summary, you can download the published app artifacts (not to be confused with Business Central artifacts):



The app is available as app and runtime package. The test app and the test results are also published.
Note that the runtime packages are prefixed with a number series, which indicates the order in which the should be installed (dependencies first), as Sort-AppFilesByDependencies doesn't work on runtime packages.



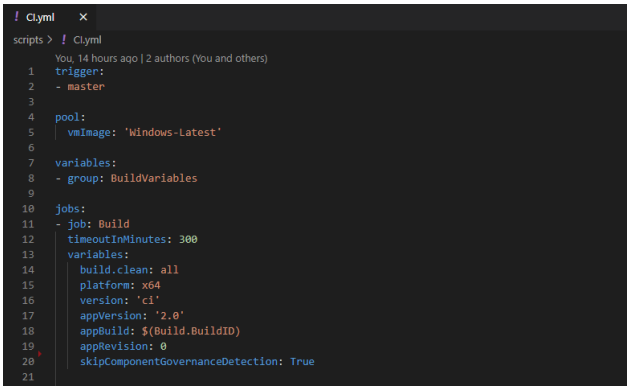
Congratulations – you have run your first Build pipeline.

Inspect the pipeline

Let’s have a look at the pipeline and some of the steps.

Initialization

In CI.yml you have some settings and variables at the start:



Trigger: master means that the pipeline will trigger whenever a change is made to the master branch.

Pool: vmImage: 'windows-latest' means that the pipeline will use Azure Hosted agents of the latest Windows version. Change this to **name: 'default'** if you want to use self-hosted agents in the default pool. (You can use https://aka.ms/getbuildagent to create a self-hosted agent and add it to the default pool).

Variables: group: buildVariables will request access to the variable group buildVariables and make the variables defined in that available in the pipeline.

timeoutInMinutes: 300 determines that the build will run in up to 300 minutes before timing out.

build.clean: all means that DevOps will clean all symbols caches and binary folders before every build – do not attempt to reuse anything.

platform: x64 indicates that we want to run 64 bit mode (not 32 bit x86) **pool: name: Default** says that we want to use a build agent from the Default pool (in which we places our agent)

version: 'ci' determines which settings from **settings.json** to use. The settings points out which Business Central artifacts to use for the build, which apps to install, the App Folders, the Test Folders, which test framework to install, which cops to enable etc. etc.

AppVersion, AppBuild and AppRevision determines the version numbering. There are a lot of different ways to make version numbering of you app. I have implemented one very simple model, where the three variables **appVersion, appBuild and appRevision** will be combined into a version number of the build. **appBuild** is set to the **Build_buildID** which is a unique build ID (auto incrementing) for this Organization.

The individual apps compiled by the pipeline will get the same **appBuild** and **appRevision**, but will keep their **appMajor** and **appMinor** from **app.json**. This allows for dependency apps to be compiled from various pipelines and get unique build numbers.

Steps

Set BuildNumber

Inline PowerShell script to set the build number for Azure DevOps.

Run Pipeline

Invoke the function DevOps-Pipeline to perform the actual pipeline. InsiderSasToken, LicenseFile and code signing certificate are transferred from variable group to function in environment variables as the build agent won’t have

access to key vaults, the pipeline does. AppBuild and AppRevision are transferred as parameters together with version, which indicates whether this should be a ci build, a current build, a nextminor build or a nextmajor build.

The DevOps-Pipeline will also check that the right version of BcContainerHelper is installed and imported and it will read the settings file to determine the right settings for invoking the Run-AlPipeline in BcContainerHelper

Publish Test Results

Publish the test results, which are saves in XML files by the pipeline. The format used for the test output is Junit.

Publish Artiacts

Publish the build artifacts (not to be confused with artifacts for running a container) to devops. This is the build result. The apps, the runtime packages, the test apps and the test results.

Cleanup

Cleanup the environment by invoking cleanup.ps1. When using hosted agents, this is really not necessary, they get cleaned up automatically. When using self-hosted agents, the cleanup function will remove containers, artifacts and images left over after failed builds. Artifacts and images, which haven't been used for 2 days are removed.

Settings

The settings file in the scripts folder determines the settings for running the pipeline. The file is read and used only by the Read-Settings file, which reads the file and sets a number of variables based on settings.json. Read-Settings takes a version parameter indicating which version to build (ci, current, nextminor or nextmajor). Settings are read from the specific version section if available, else from the main section.

This means that if you want to run **AppSourceCop** in all versions but the **ci** version – you would set this to **true in the main section** and **false in the ci section**.

Read-Settings will set the following variables

agentName is blank if running local or set to $ENV:AGENT_NAME if running a devops agent.

pipelineName is the name of the pipeline, including the version (e.g. HelloWorld-ci)

containerName is set to the name of the build container. The agent name is included in the name if running a devops agent.

The following list are the settings, which will be turned into variables with the same name

installApps is a comma separated list of secure url's to dependencies (apps or .zip files containing apps) to be installed before compiling apps and test apps. The apps can be apps or runtime packages. If they are apps, they are sorted after dependencies before installing. Runtime packages are sorted and installed alphabetically.

previousApps is a comma separated list of secure url's to previous versions of the apps (or .zip files containing previous versions of the apps) to be used as previous versions of apps for **AppSourceCop** breaking change detection and upgrade test. When **previousApps** are specifying, these apps are published and installed before the newly build apps and upgrade is run before the tests.

appFolders is a comma separated list of folders which should be compiled as apps. The folders will be sorted after dependencies before compiled, published and installed. Apps in app folders are compiled before the test framework/libraries are published/installed and they are signed (if signing certificate is specified) unless doNotSignApp is set to true.

appSourceCopMandatoryAffixes is a comma separated list of affixes to be used as mandatory affixes in AppSourceCop settings.

appSourceCopSupportedCountries is a comma separated list of supported countries to be used as supported countries in AppSourceCop settings.

testFolders is a comma separated list of folders which should be compiled and used as test apps. The folders will be sorted after dependencies before compiled, published and installed. Apps in test folders are compiled after the test framework/libraries are published/installed and tests in these apps are used for test execution.

memoryLimit determines the amount of memory available in the build container during the pipeline run.

additionalCountries is a comma separated list of country codes. During the pipeline, a container with this local version of Business Central will be spun up and the apps produced will be published, installed and tested. Test results will be gathered for all versions. Default is no additional countries.

genericImageName is the generic image name to use for creating the container. Default is the default generic image in BcContainerHelper configuration.

vaultNameForLocal is the name of the key vault to use for secrets like licensefile, insiderSasToken and passwords. This setting is only used in **Local-DevEnv.ps1** and **Local-Pipeline.ps1**.

bcContainerHelperVersion determines which version of BcContainerHelper to use. Latest is the default setting, which probably is fine for most. Preview means grab the latest preview version and a specific version number will grab that exact version from the PowerShell Gallery. This setting can also be a local path on the build agent or a URL to a github repository, where the desired version of the containerhelper can be downloaded.

installTestFramework is a Boolean setting determining whether to install the Test Framework before compiling the test apps.

installTestLibraries is a Boolean setting, determining whether to install the Test Libraries before compiling the test apps. Test Libraries includes the Test Framework.

installPerformanceToolkit is a Boolean setting, determining whether to install the Performance Toolkit before compiling the test apps. Performance toolkit include the Test Framework.

enableCodeCop is a Boolean setting, determining whether Code Cop is enabled.

enableAppSourceCop is a Boolean setting, determining whether AppSource Cop is enabled.

enablePerTenantExtensionCop is a Boolean setting, determining whether Per Tenant Extension Cop is enabled.

enableUICop is a Boolean setting, determining whether UI Cop is enabled.

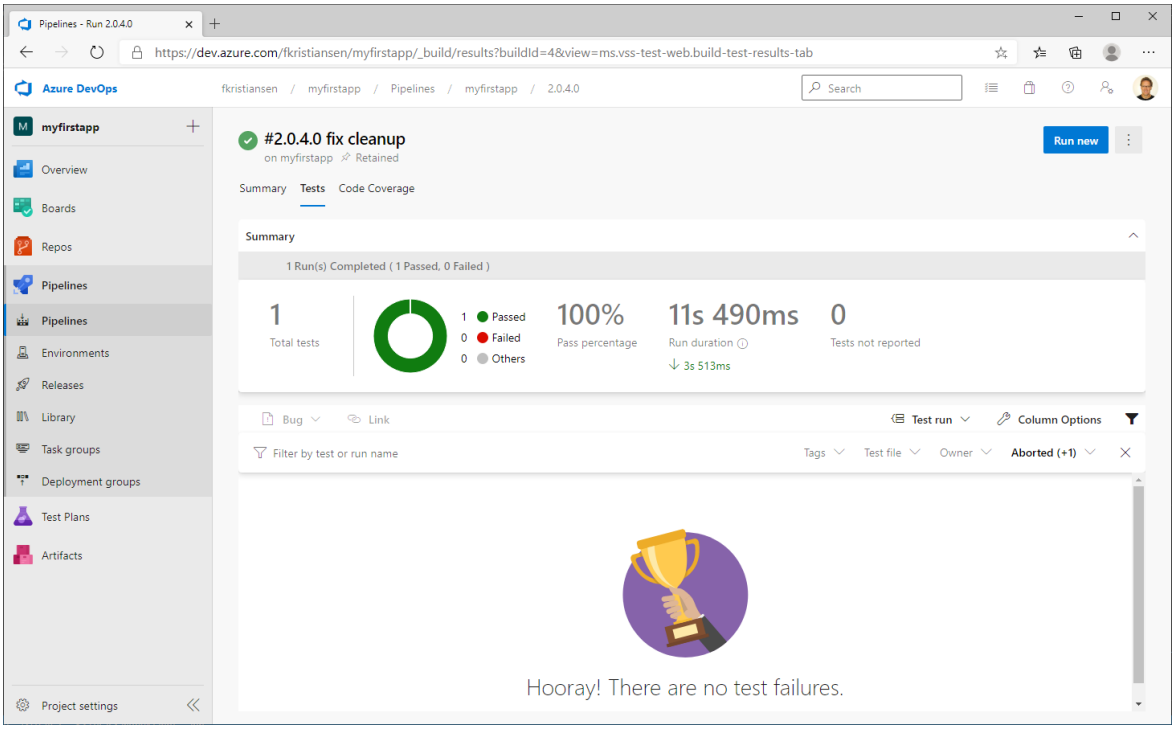
doNotSignApps is a Boolean setting which can be set to true if you do not want to sign Apps.

doNotRunTests is a Boolean setting, which can be set to true if you do not want to run Tests.

cacheImage is a Boolean setting, which determines whether an image will be cached before creating the build container. By default cacheImage is set to true on ci pipelines which is typically reusing the same Business Central version and false on other pipelines as versions changes a lot.

Publishing Test Results

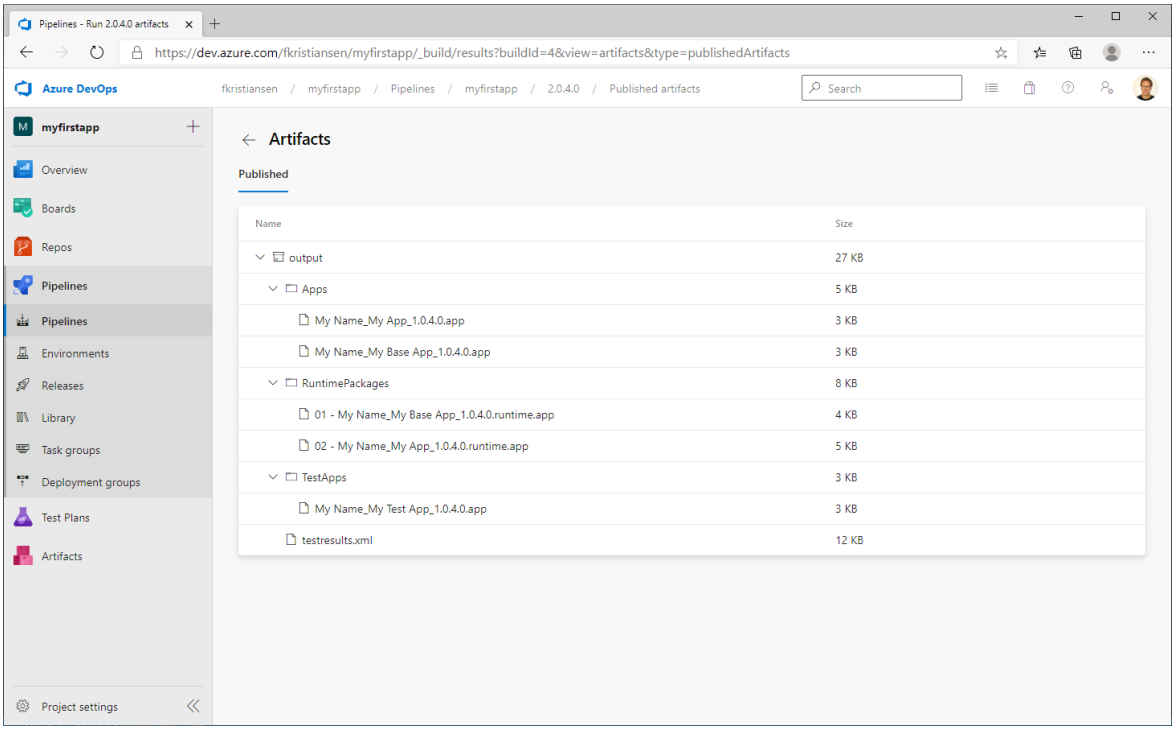
The **Publish Test Results** step will publish the **JUnit** compatible test results file to **Azure DevOps**, giving you the opportunity to investigate failing tests, see stack traces and creating work item for fixing failing tests.



Publish Artifacts

The **Publish Artifacts** step will publish the generated build artifacts to make them available for a release pipeline. The artifacts contains the **.app files generated** (**including tests**), runtime packages of the same apps and the test results.

The artifacts are easily consumed by release pipelines.



Cleanup

The Cleanup step will invoke the **cleanup.ps1** script, which will remove the container (if present) and cleanup up artifacts cache and container images, which hasn't been used the last 2 days.

Create a Release Pipeline

I will be working with a few different release pipelines.

1. Two release pipelines for releasing the build artifacts to blob storage (preview and production). You can also use Azure DevOps Artifacts, but what I really like about blob storage is, that I can secure access to the artifacts as I decide and I can download the artifacts using a simple Url (no PowerShell commands with special Az modules needed). The preview release pipeline should be invoked after every successful build, the production one should run on demand.
2. Two release pipelines for releasing a Per Tenant Extension to an online environment for a customer to test in sandbox environment or run in production. The pipeline releasing to a sandbox environment could be setup to run after every successful build, the one releasing to production should run on demand.
3. **Later versions** of this workshop will also include how to **deploy the App directly to AppSource** and through that **to end-customers** who have installed your app.

Releasing to Blob Storage

The way I have structured the blob storage for my apps is:

https://storageaccount/appname/version/apps.zip

appname could be **bingmaps**, **bingmaps-preview**, **helloworld** or **helloworld-preview**. **Version** is either a **specific version number** or **latest**.

Examples:

https://businesscentralapps.blob.core.windows.net/bingmaps/16.0.10208.0/apps.zip

gives me version 16.0.10208.0 of the BingMaps app, and

https://businesscentralapps.blob.core.windows.net/bingmaps-preview/latest/runtimepackages.zip

gives me the runtime packages from the latest version of the bingmaps preview, and

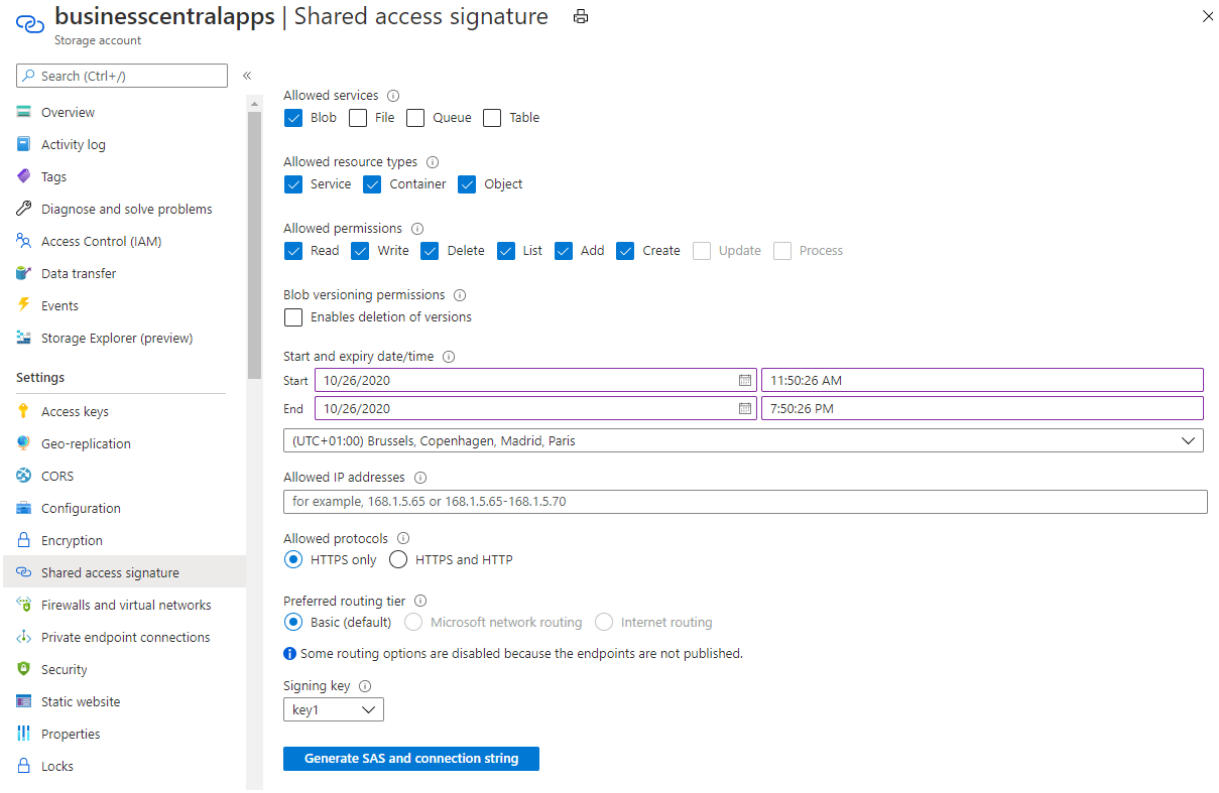
https://businesscentralapps.blob.core.windows.net/helloworld/latest/apps.zip

gives me the latest production release of the helloworld app.

You might have noticed that I always use the latest production release of my apps as **previousapps** setting for AppSourceCop to get breaking change notification. So, let's setup the two release pipelines.

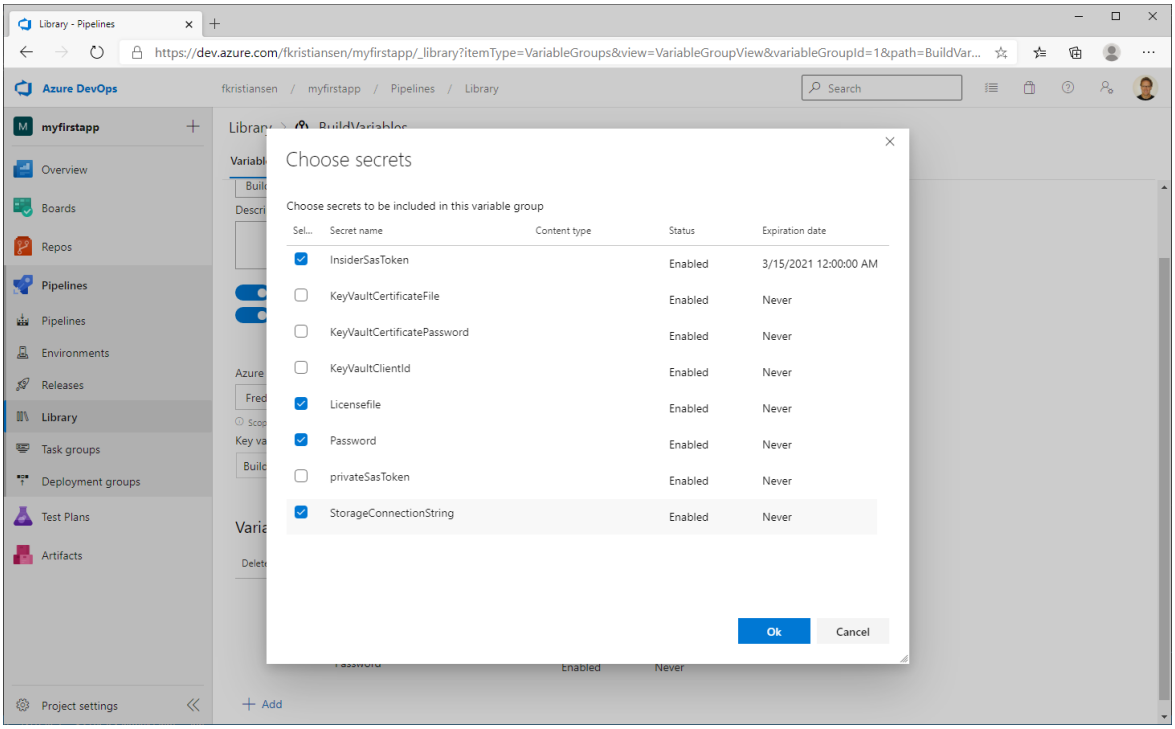
Create a Blob Storage and get the Connection String

First we need a Blob Storage and a connection string. The Blog Storage Account is created in the Azure Portal and under shared access signature you define the access requirements for the connection string and generate the connection string:



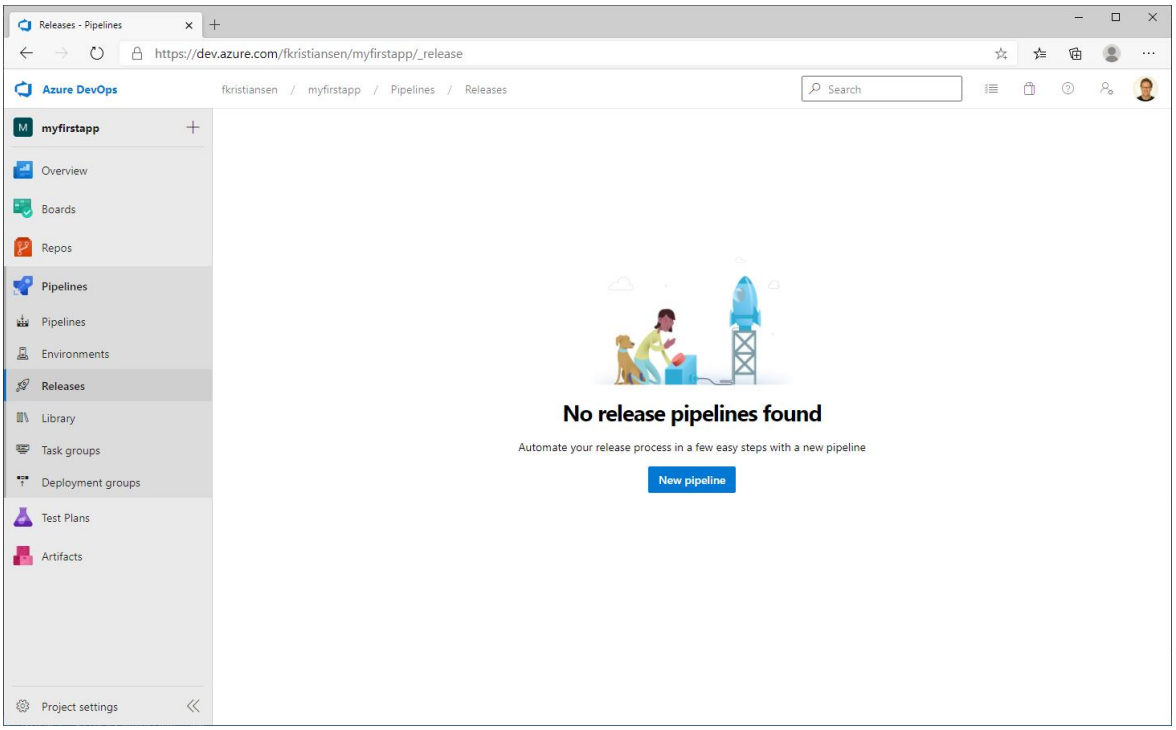
After pressing Generate SAS and connection string, you get three values:

Click the **Copy to Clipboard** next to the **Connection String** and add this as a secret in your key vault from the first section. In the library section under Pipelines, add the new secret to the secrets available to the pipelines, press Ok and Save.

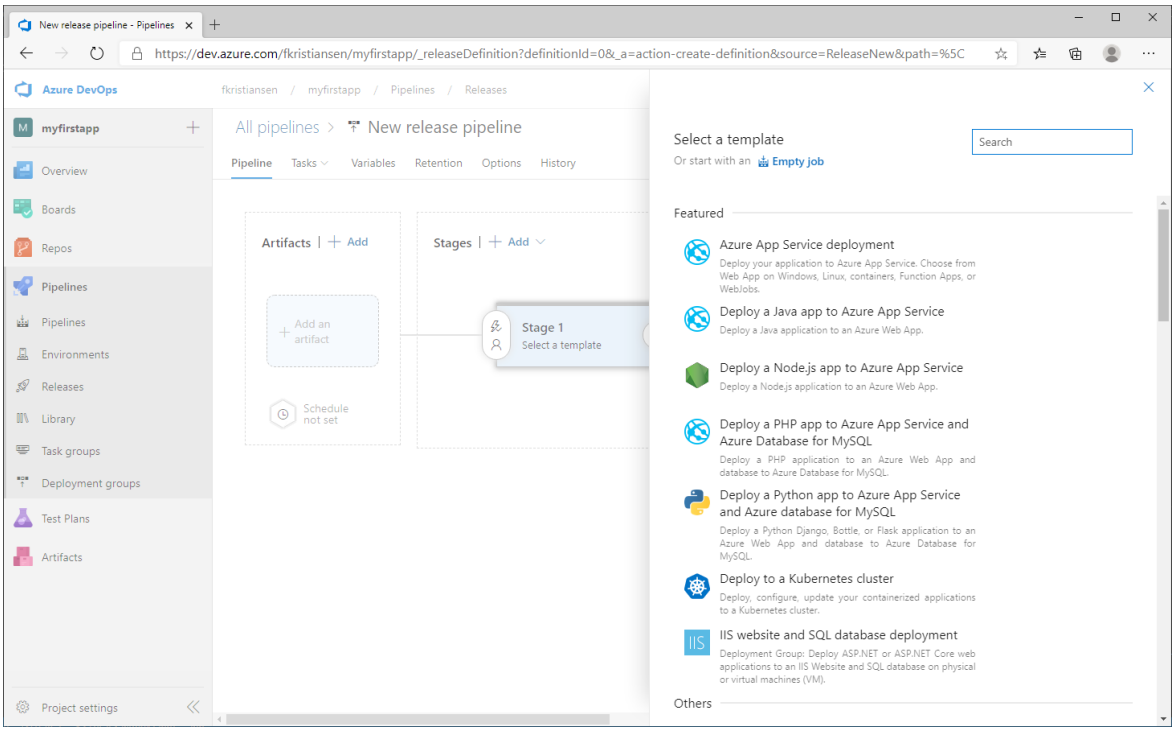


Creating the release Pipeline

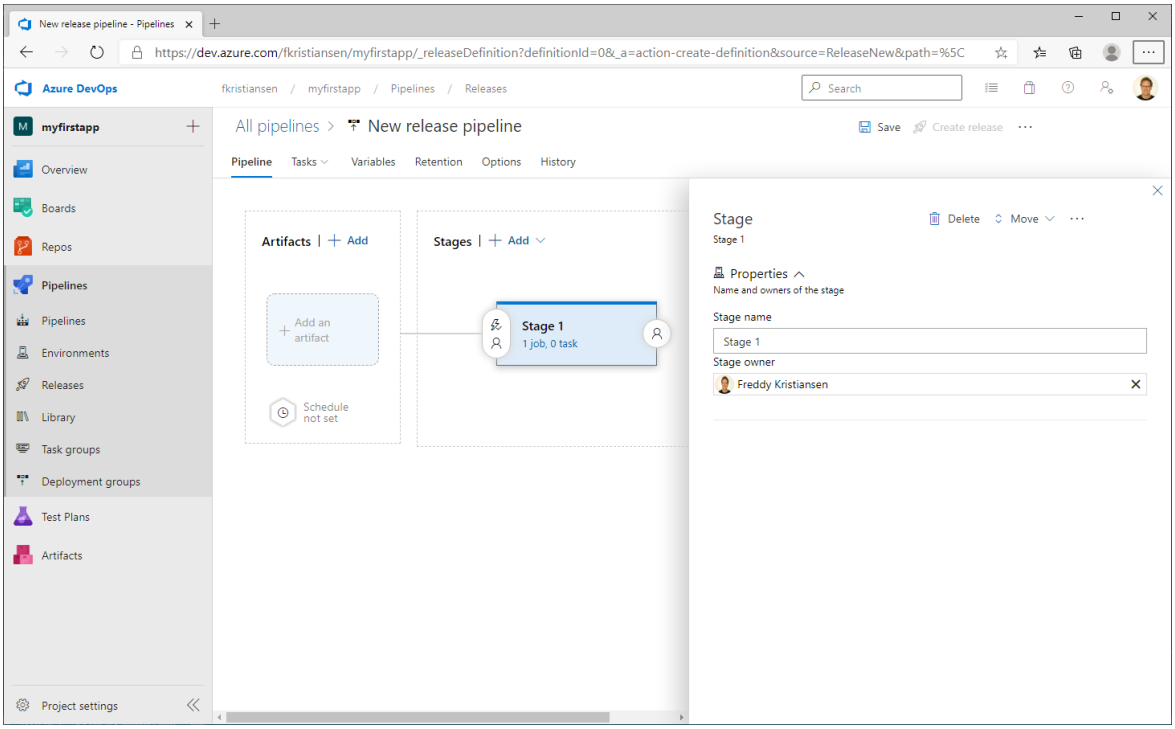
In **Azure DevOps**, under **Pipelines**, click **Releases** and select **New pipeline**.



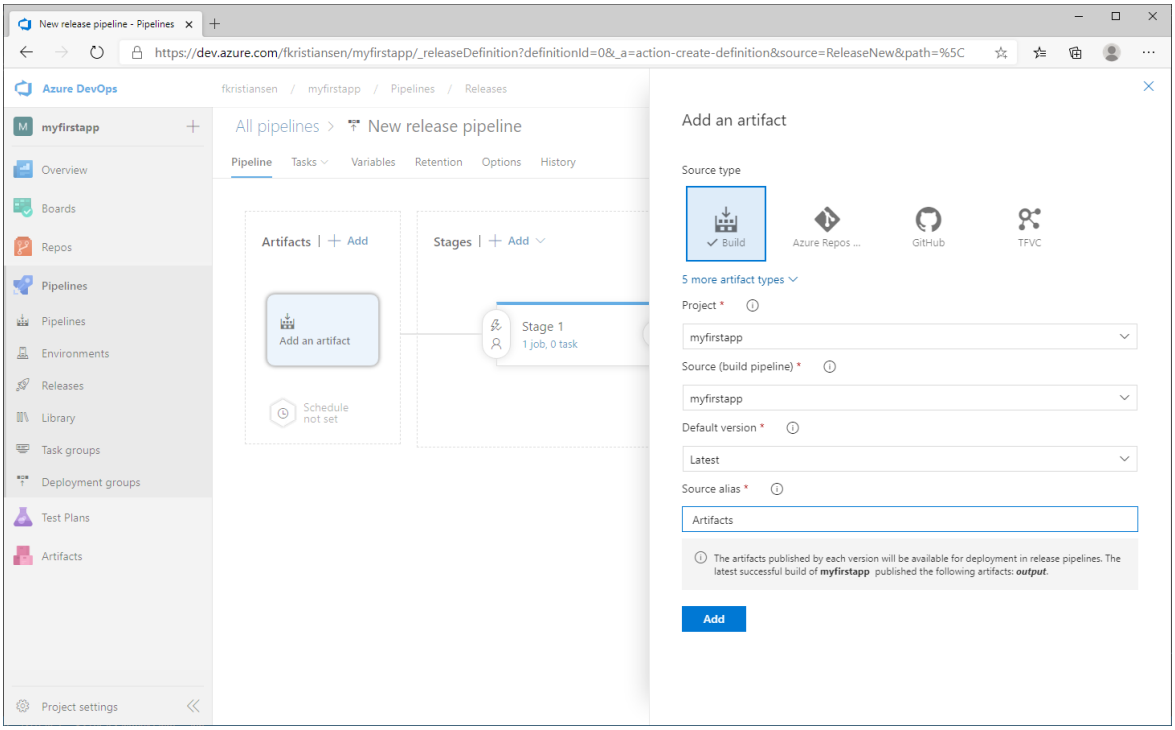
Select **Empty job** as template



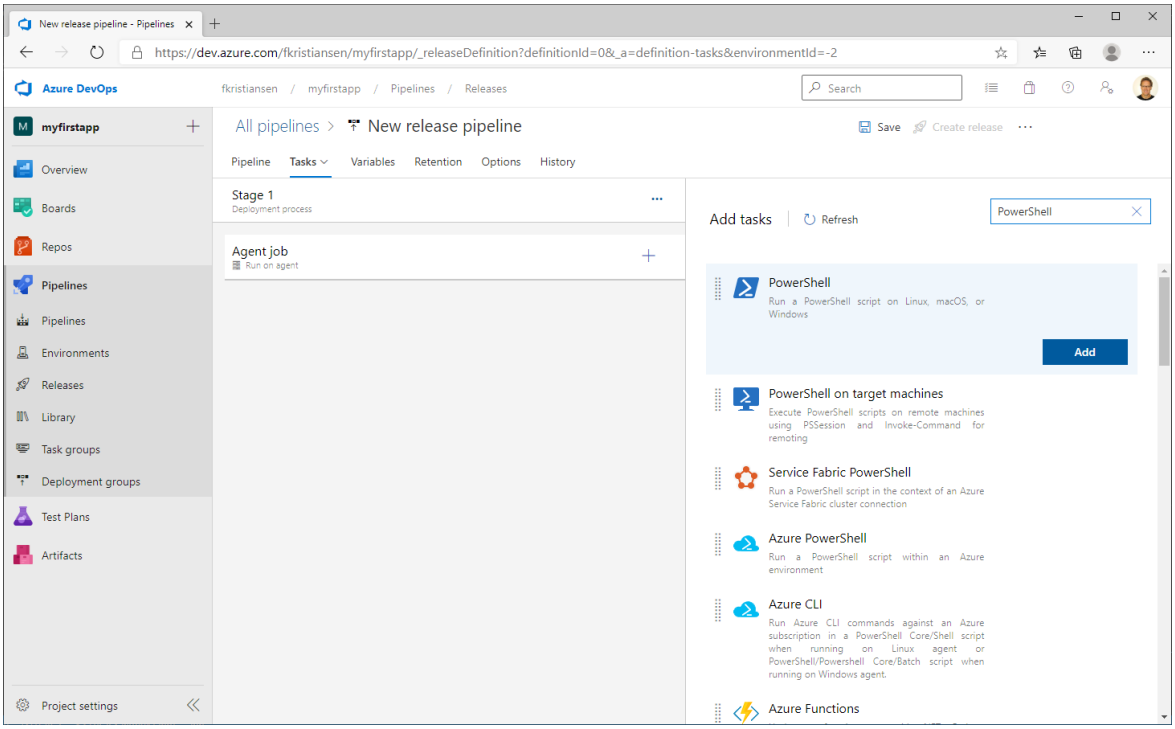
Click **Add an artifact**



Use the **build pipeline** as source and name the artifacts **Artifacts.**



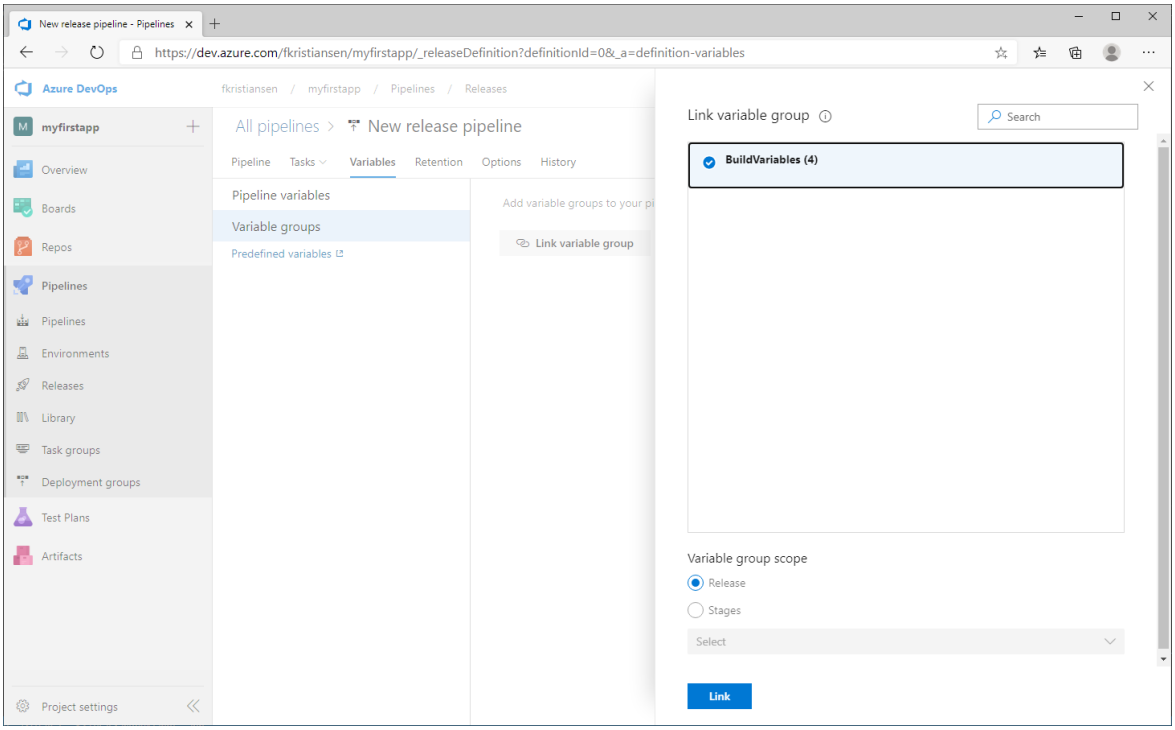
Click Tasks and click the + next to the Agent Job. Search for PowerShell and add a PowerShell task to the job.



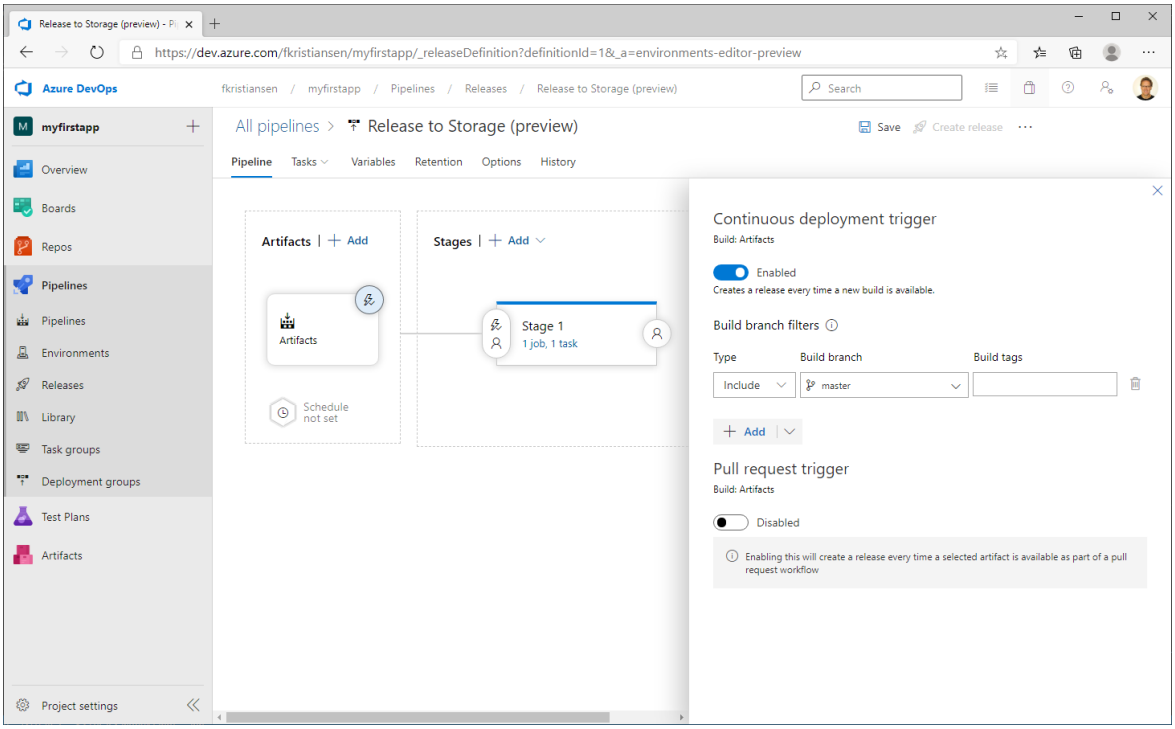
The **PowerShell Script task** displays that some settings needs attention. Click the **PowerShell Script task.** Select Inline and Paste the following code into the script editor:

```
Write-Host "Installing BcContainerHelper"
Install-Module BcContainerHelper -Force

Write-Host "Publishing to Storage"
Publish-BuildOutputToStorage `
    -storageConnectionString "$(StorageConnectionString)" `
    -projectName "$($ENV:BUILD_PROJECTNAME)-preview".ToLowerInvariant() `
    -appVersion "$ENV:BUILD_BUILDNUMBER" `
    -path "Artifacts\output" `
    -setLatest
```

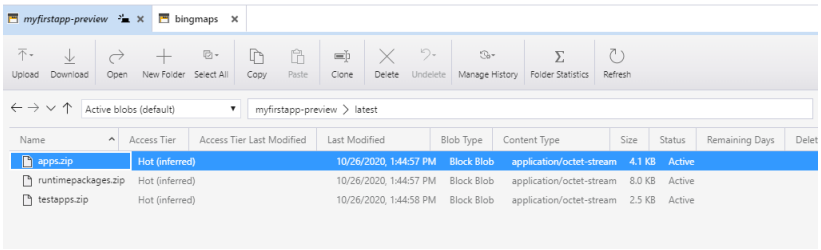
Click **Variables**, **Variable Groups**, select **BuildVariables** and click **Link** to link the variable group to the release pipeline.



Set the name of the release Pipeline to **Release to Storage (preview).** Click the small lightning bolt icon in artifacts to enable Continuous deployment trigger on the preview release pipeline.



Click **Save**. After successfully saving the release pipeline, click **Create Release** to create a release and you should get your artifacts published to blob storage:



Now redo the entire process again, where you remove **-preview** in the script and replace **(preview)** with **(prod)** in the name and I can now use https://businesscentralapps.blob.core.windows.net/myfirstapp/latest/apps.zip as previousapps in my settings file.

Releasing a Per Tenant Extension to an online environment

To release per tenant extension apps to an online environment, we will use the service-to-service authentication for automation APIs, which was shipped in Business Central 2020 release wave 2 (v17). AJ did a very detailed description of this feature on his blog: https://www.kauffmann.nl/2020/09/14/service-to-service-authentication-for-automation-apis-in-business-central/.

The partner creates an AAD Application for authentication. The Customer registers the partners AAD Application in their Business Central tenant and assigns permissions that the app can do automation and extension management.

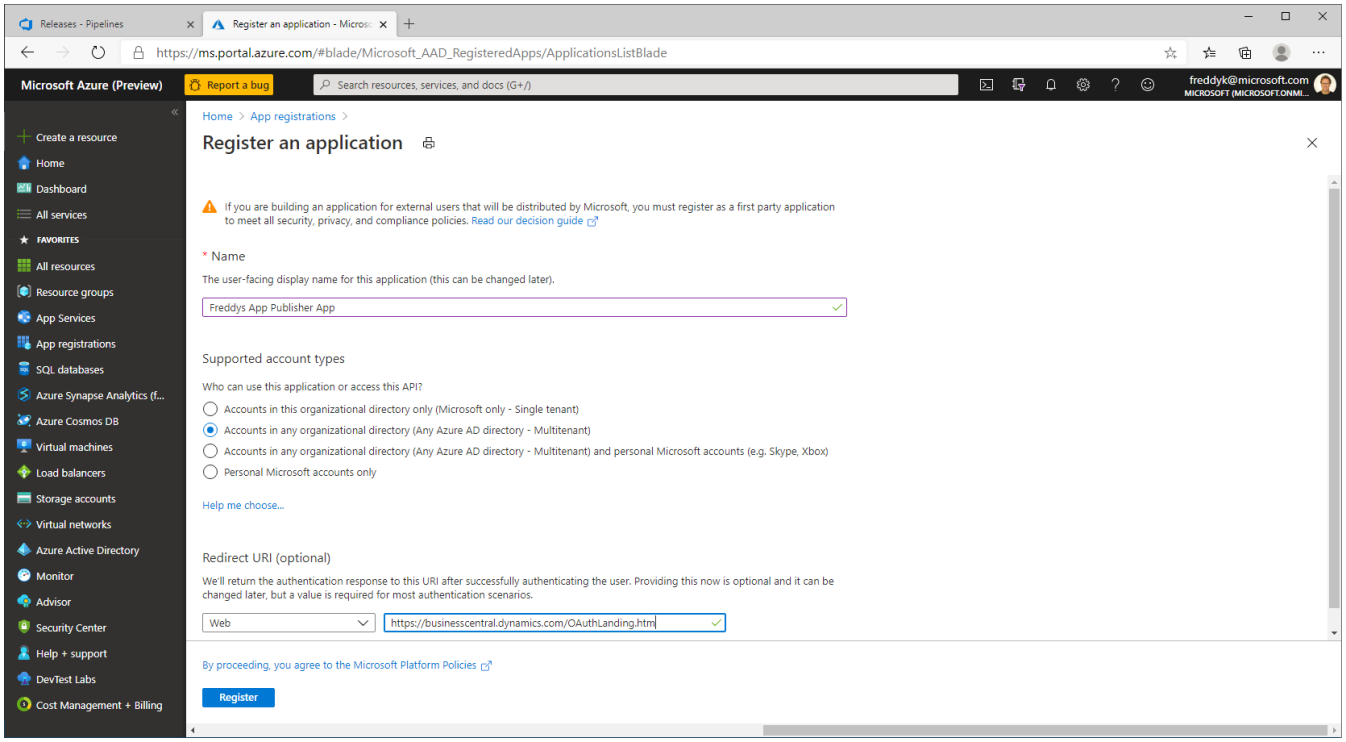
So, we need to setup a few things before we can create a release pipeline:

- An AAD Application for authenticating to Business Central
- An online tenant of Business Central v17 or higher
- Permissions in Business Central to allow the AAD App to do Automation and Extension Management

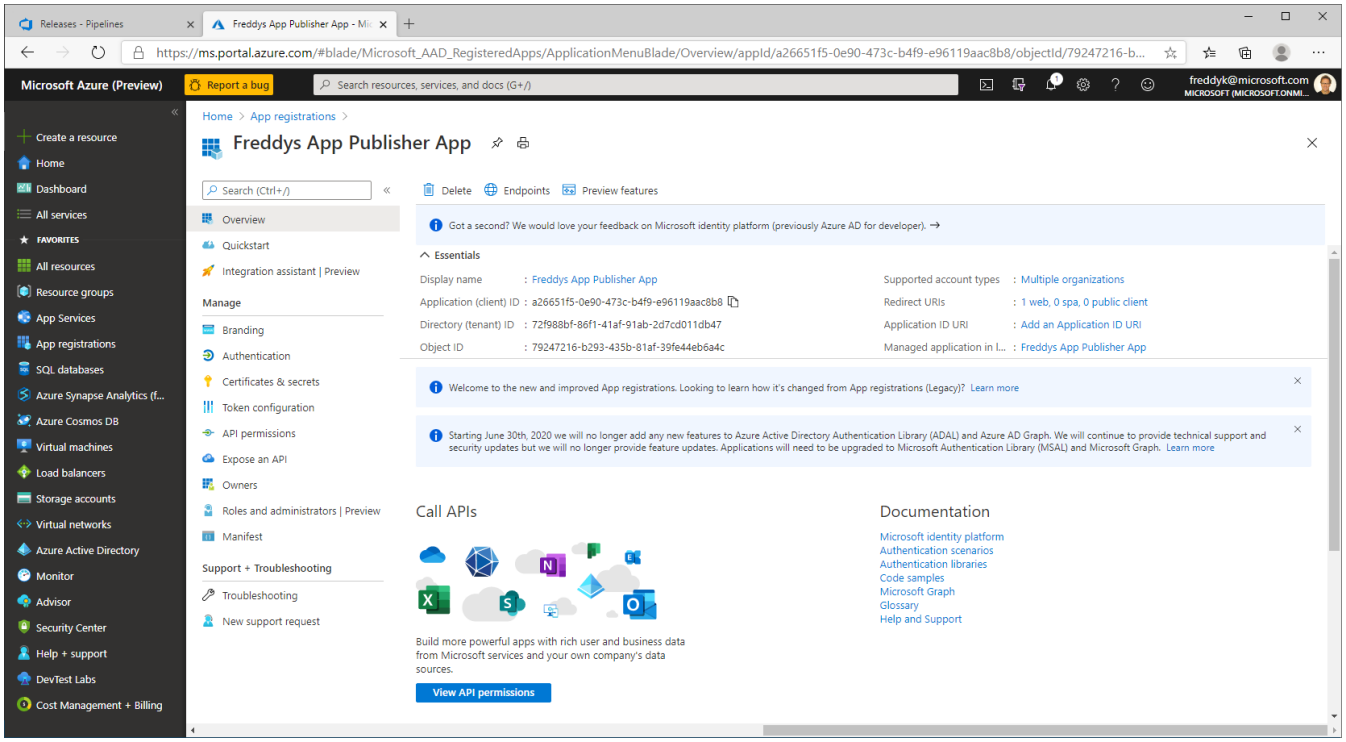
An AAD Application for authenticating to Business Central

Every partner will probably create just one multitenant AAD Application, which they will use for managing extensions or their customers, so I am not going to create a script for this (not at this time 😊)

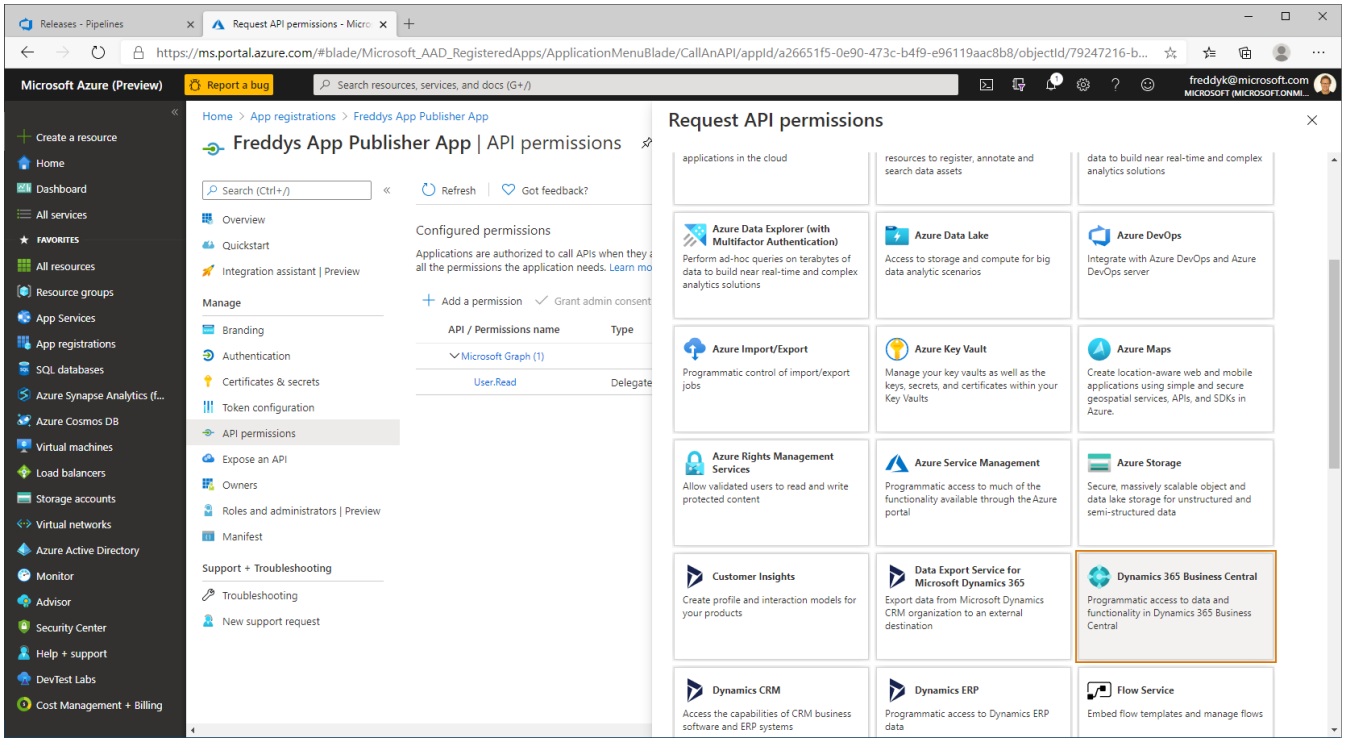
Open the **Azure Portal** (https://portal.azure.com), search for **App Registrations**, create a **New App Registration**. Give your app a **friendly display name**, set the app to **multitenant**, set the **redirect URI** to https://businesscentral.dynamics.com/OAuthLanding.htm. Click Register.



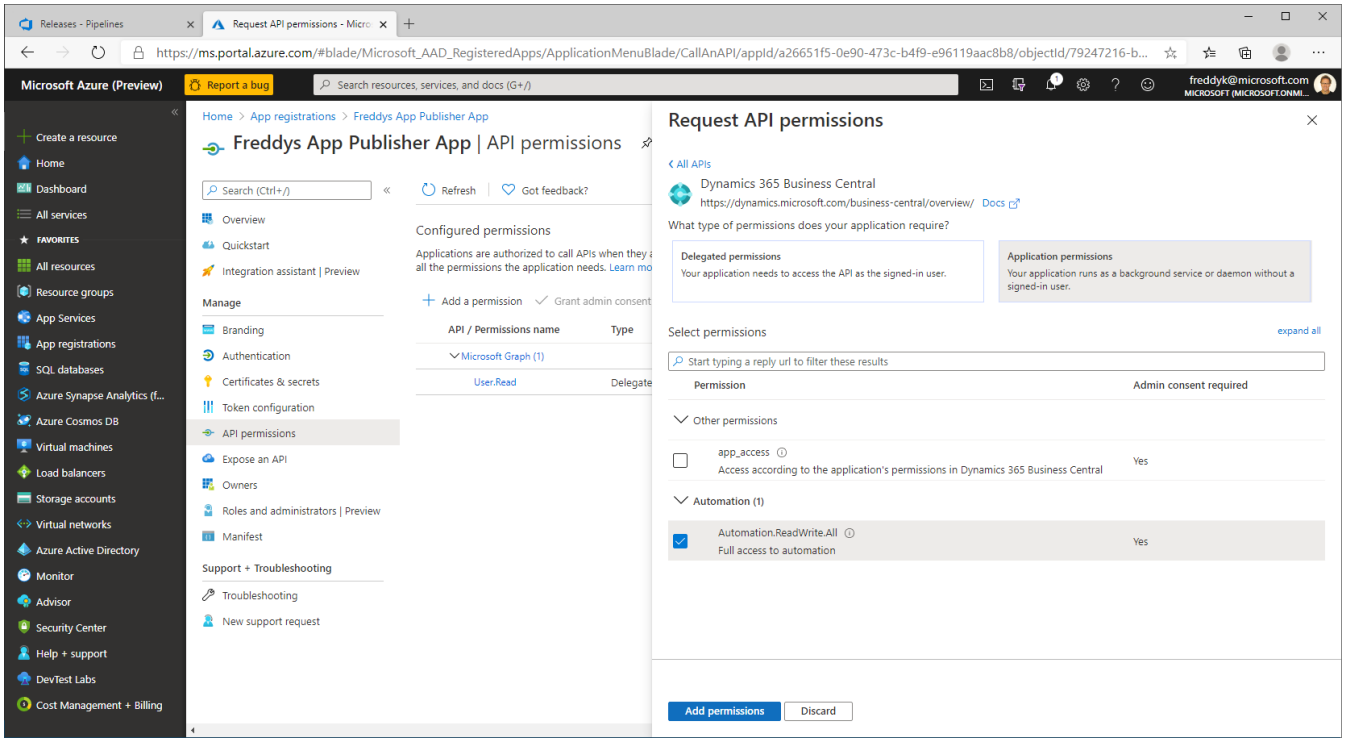
Make a copy the Application (Client ID).



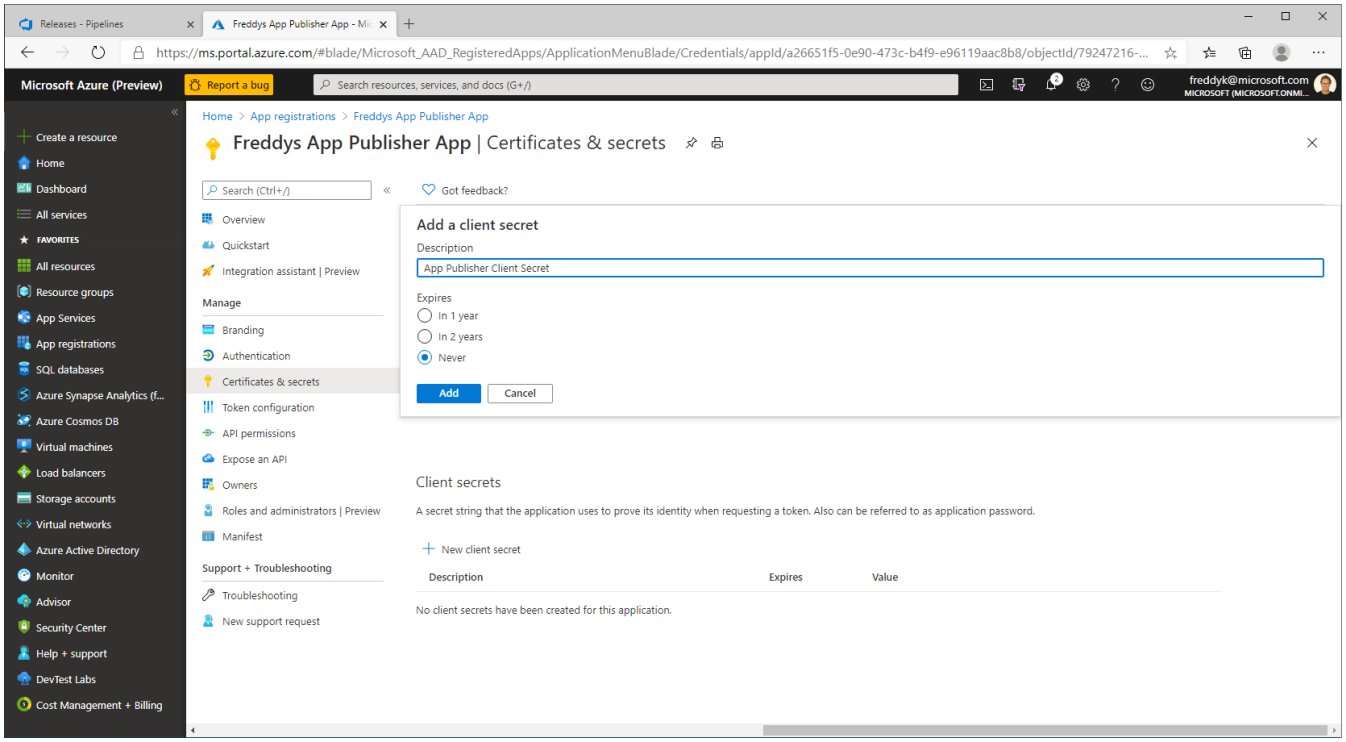
Click **API Permissions** and select **+ Add a permission**, locate Dynamics 365 Business Central and select that.



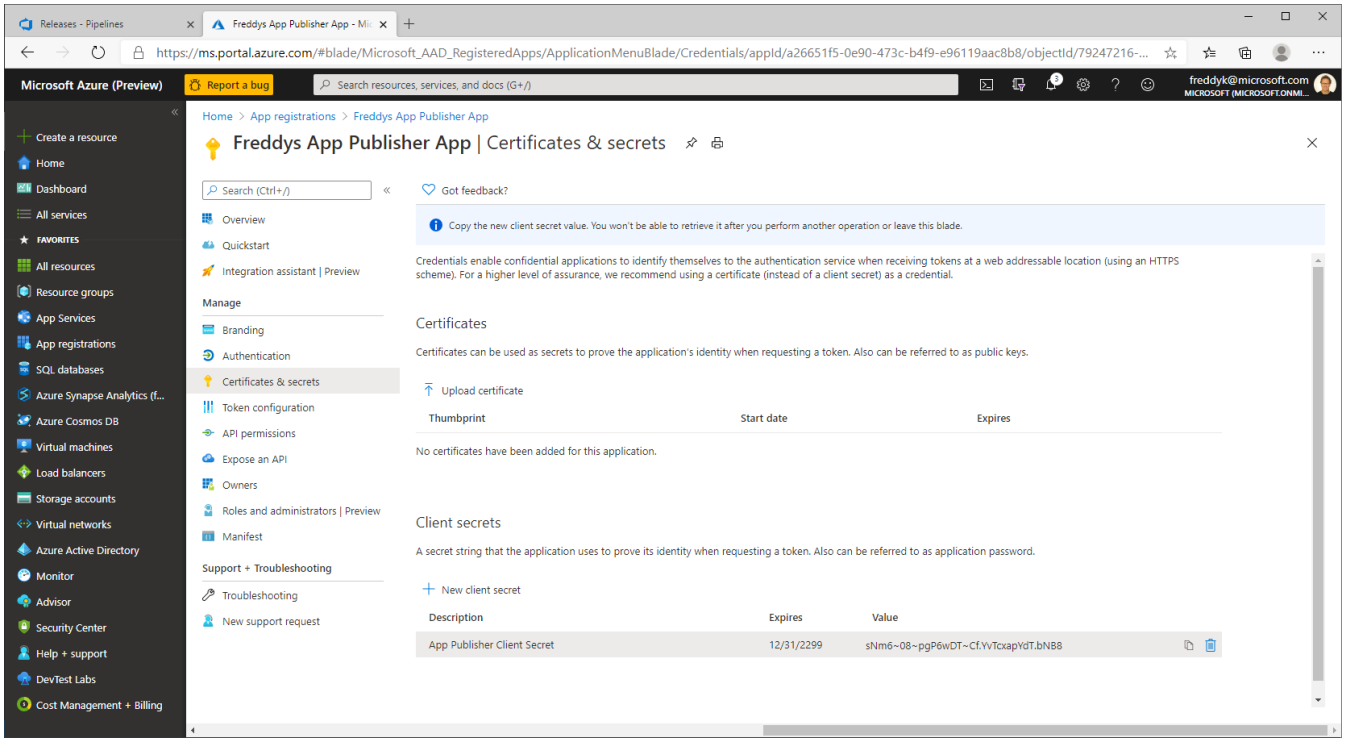
Select **Application Permissions**, check **Automation.ReadWrite.All** and click **Add permissions**



Select **Certificates & secrets**, click **+ New client secret**, give it a **name**, set the **expiration date** and click **Add**

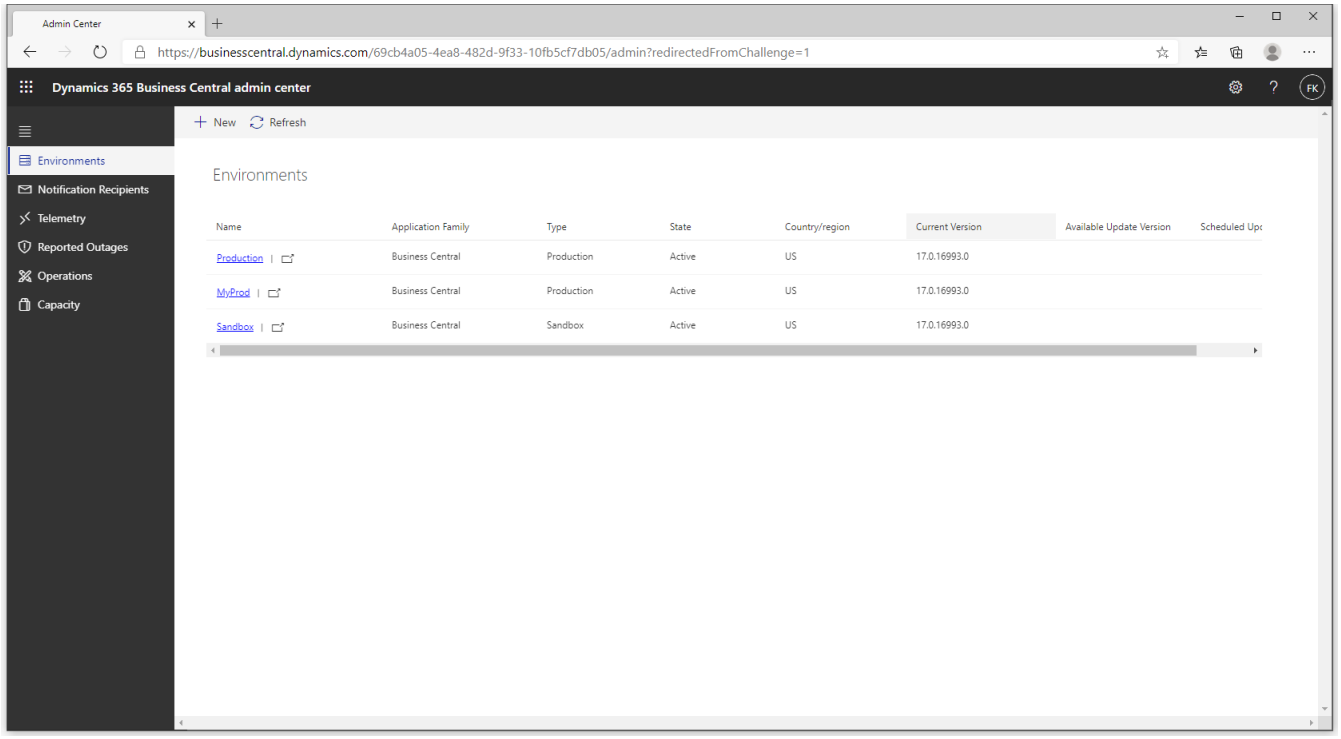


Make a **Copy of the Client Secret**, it is only shown once



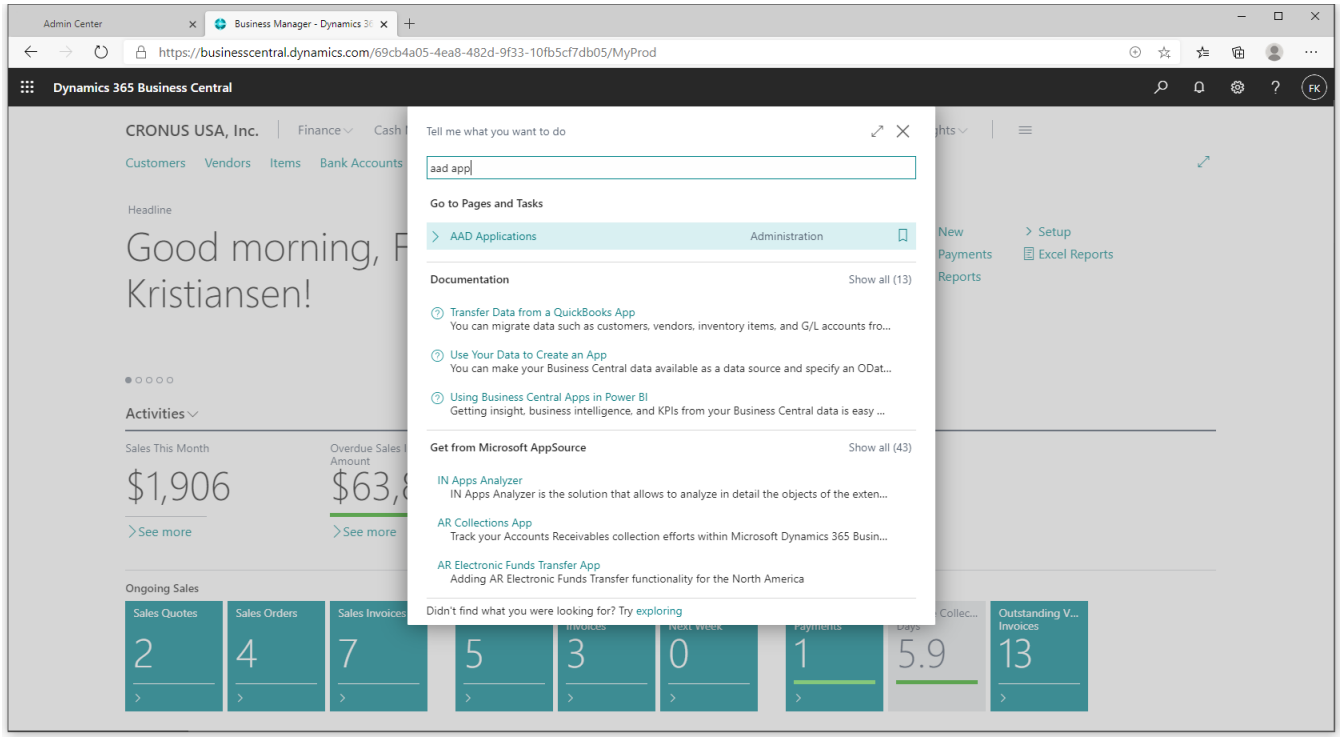
Online tenant of Business Central

Next thing we need is an online tenant of Business Central, and we need to assign permissions to the partner AAD Application. In my admin center, I have created a production environment (MyProd) and a Sandbox environment (Sandbox), which I will use for this workshop.

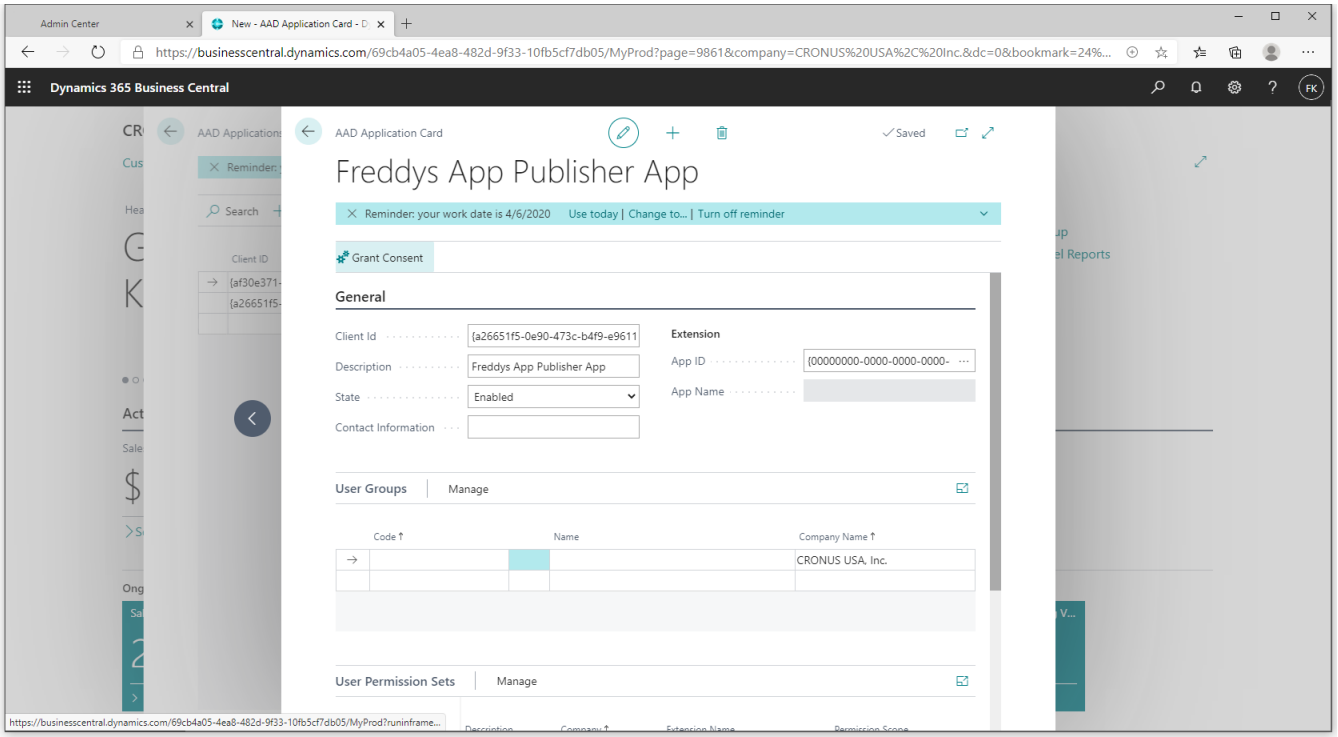


Permissions in Business Central to allow the AAD App to do Automation and Extension Management

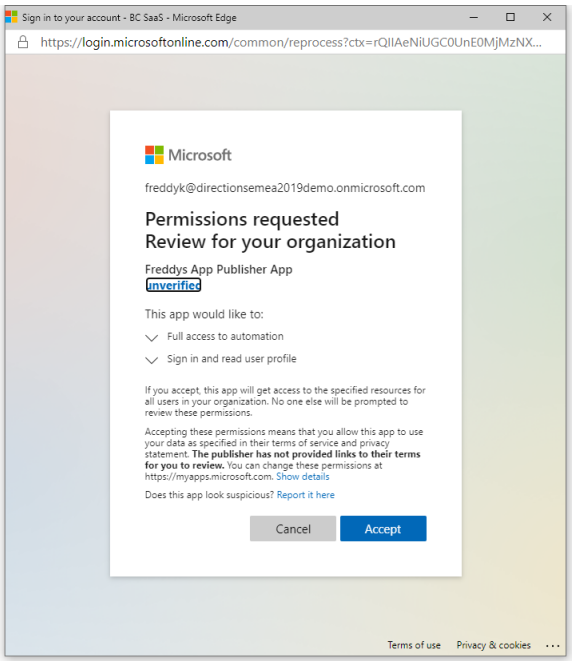
Navigate to your **Business Central environment**, click search and enter **aad app**



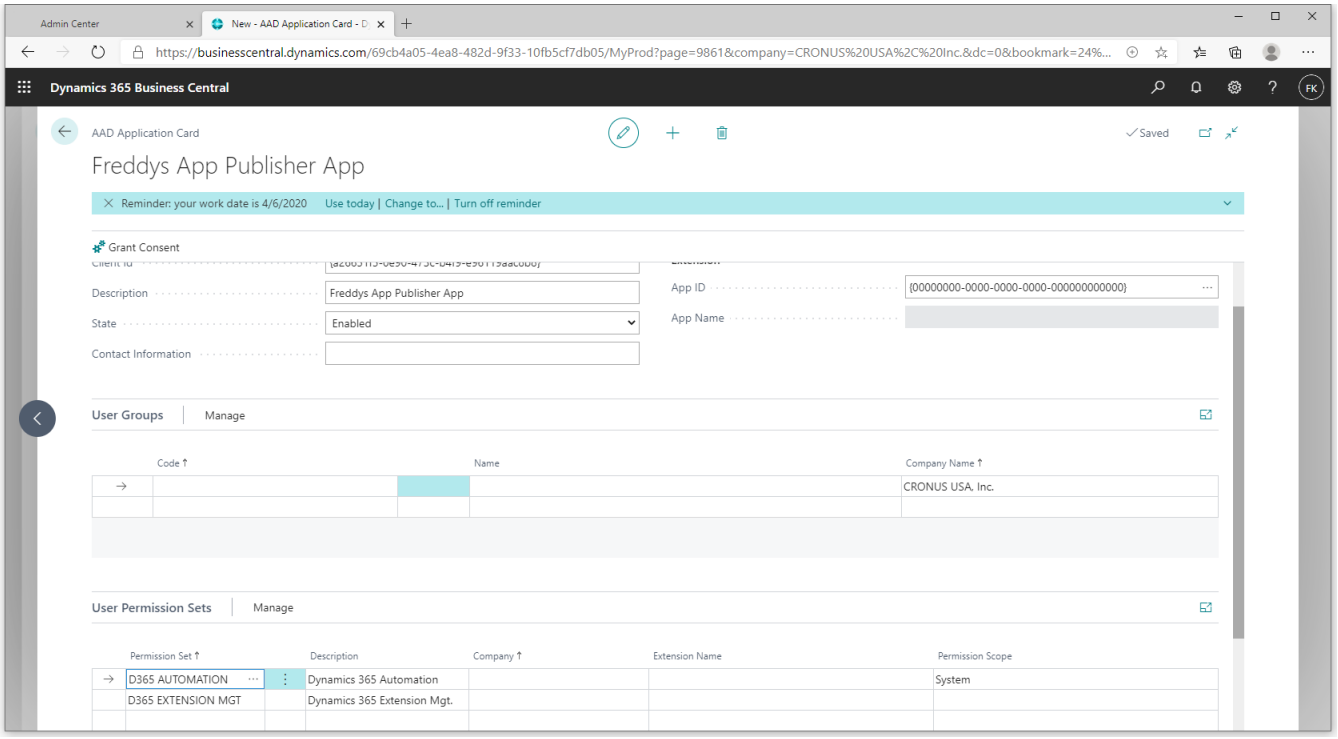
Select **AAD Applications**, click **+ New**, enter the **Client ID** and a **description** and click **Grant Consent**



You will now be asked to authenticate and give access to Freddys App Publisher App to do automation and sign in and read user profile



Click **Accept**, and assign two **User Permission Sets**: **D365 AUTOMATION** and **D365 EXTENSION MGT**

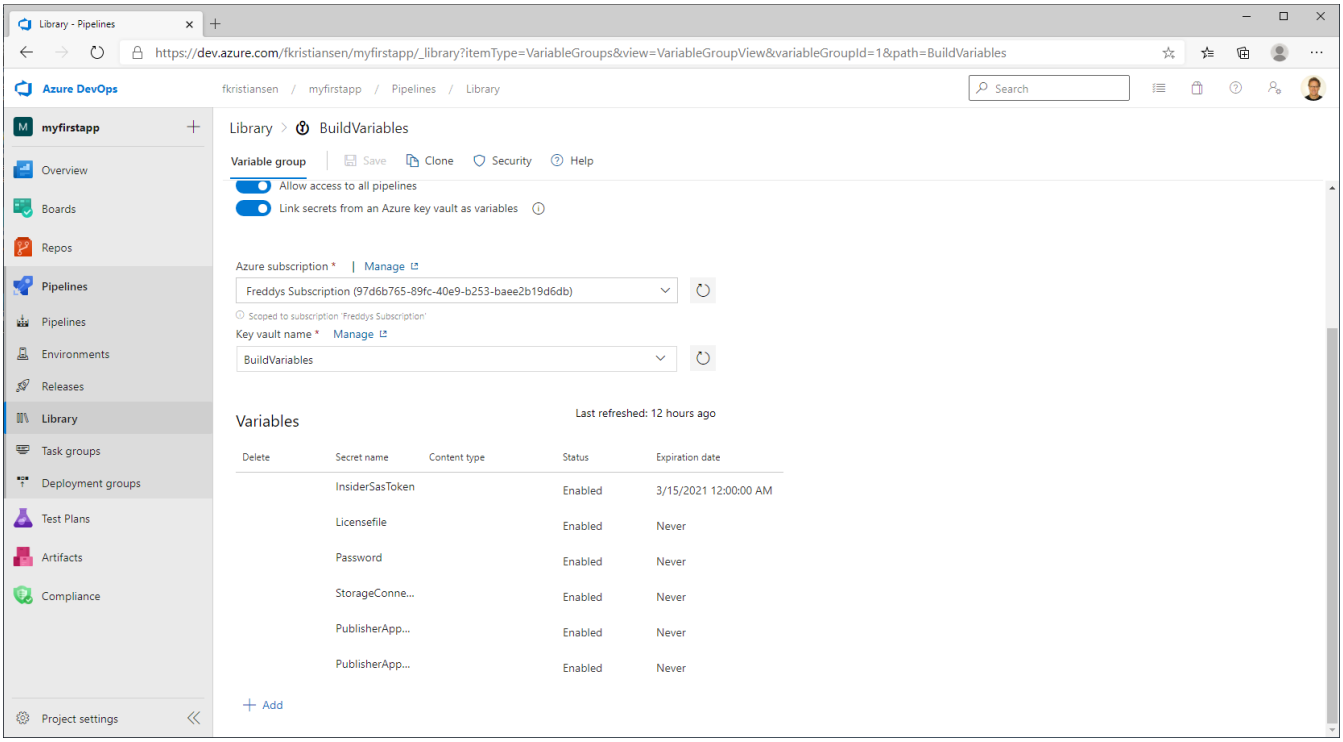


Perform the same steps in the **Sandbox environment**.

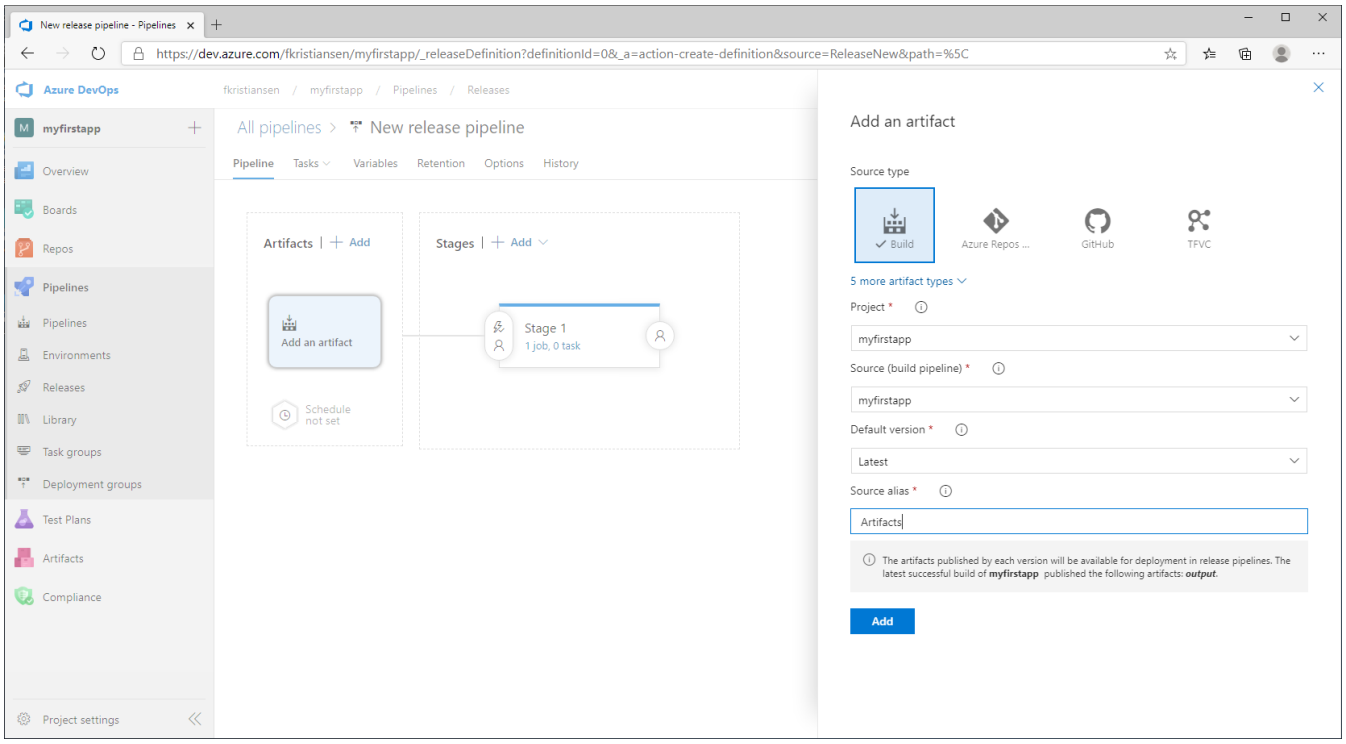
Creating the release Pipeline

Open the Azure Portal and locate your Key Vault. Add secrets for **PublisherAppClientID** (Client ID from your AAD App) and **PublisherAppClientSecret** (Client Secret from your AAD App).

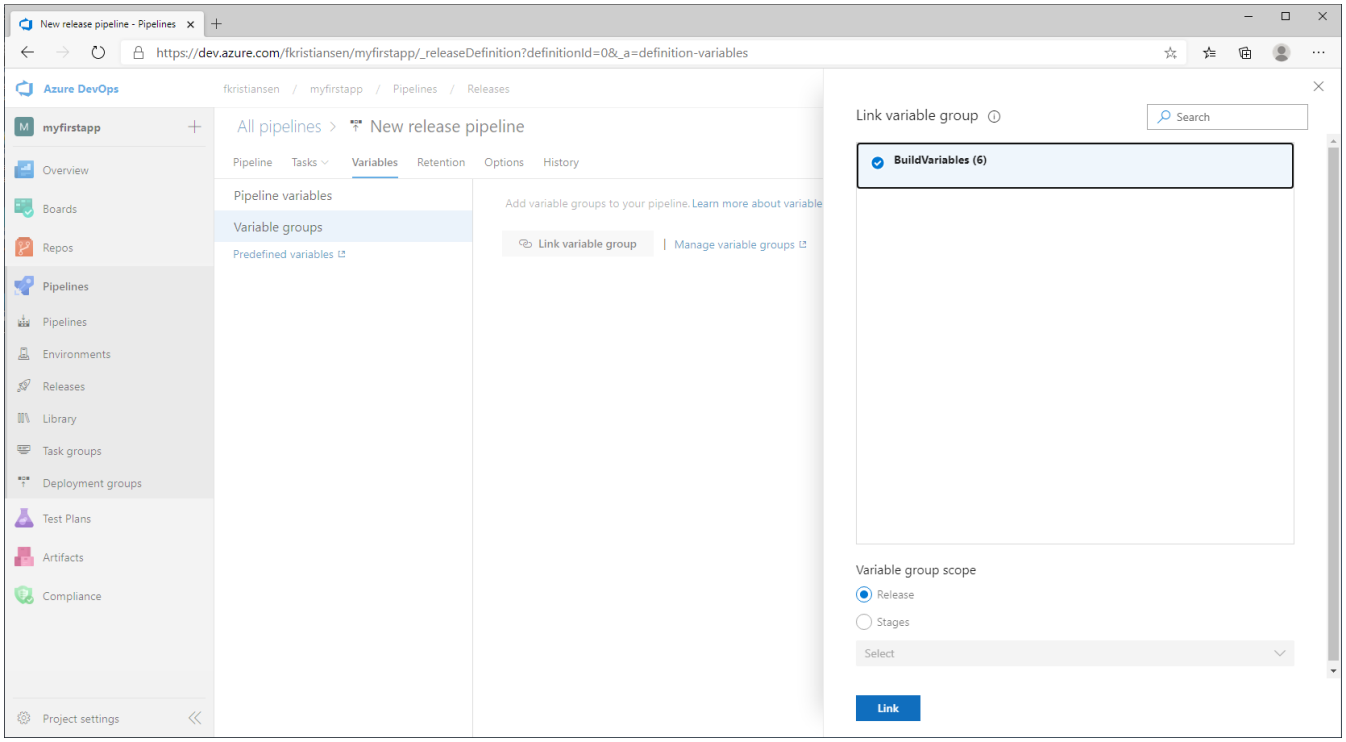
In Azure DevOps, navigate to Pipelines -> Library, modify build variables and add **PublisherAppClientID** and **PublisherAppClientSecret** to get access to these values for the pipelines and **Save**.



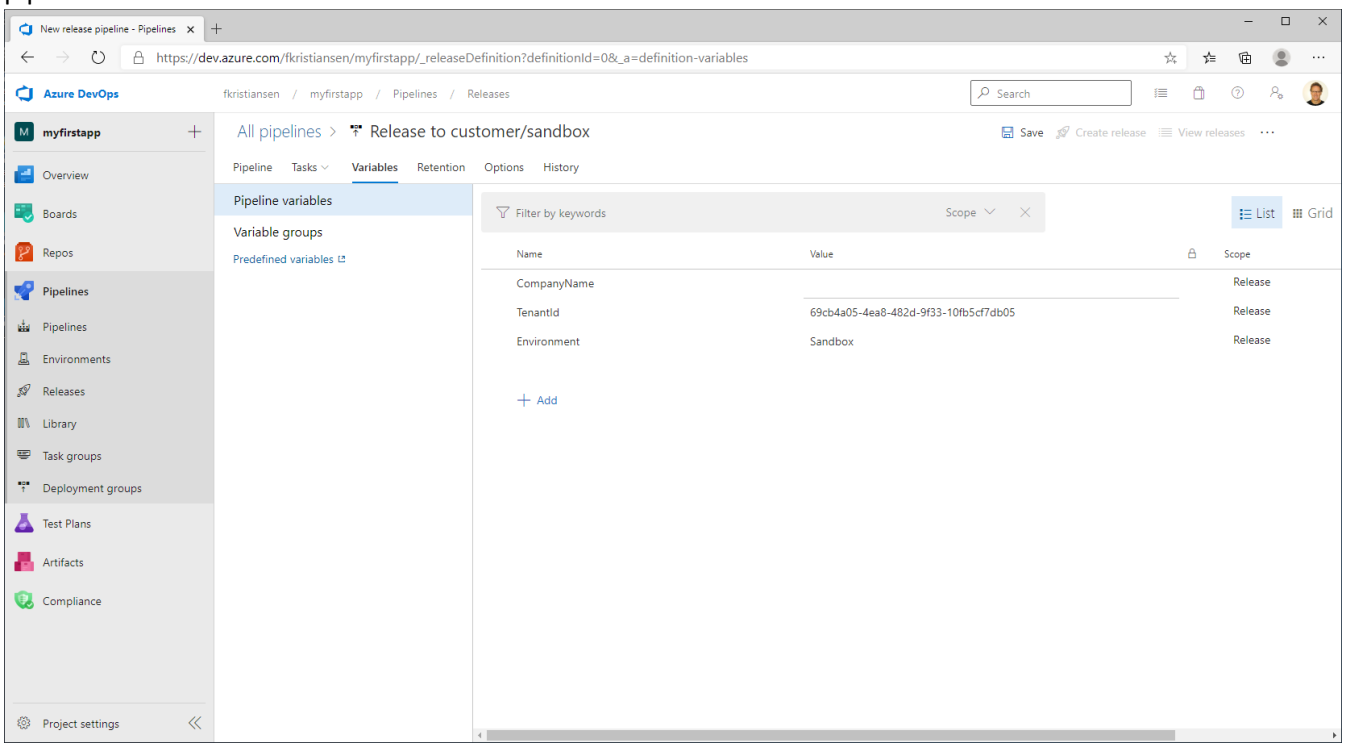
In **Releases**, click **+ New**, add a new **Release Pipeline**, click **Add an artifact**, **select the source** and set the source alias to **Artifacts**, click **Add**.



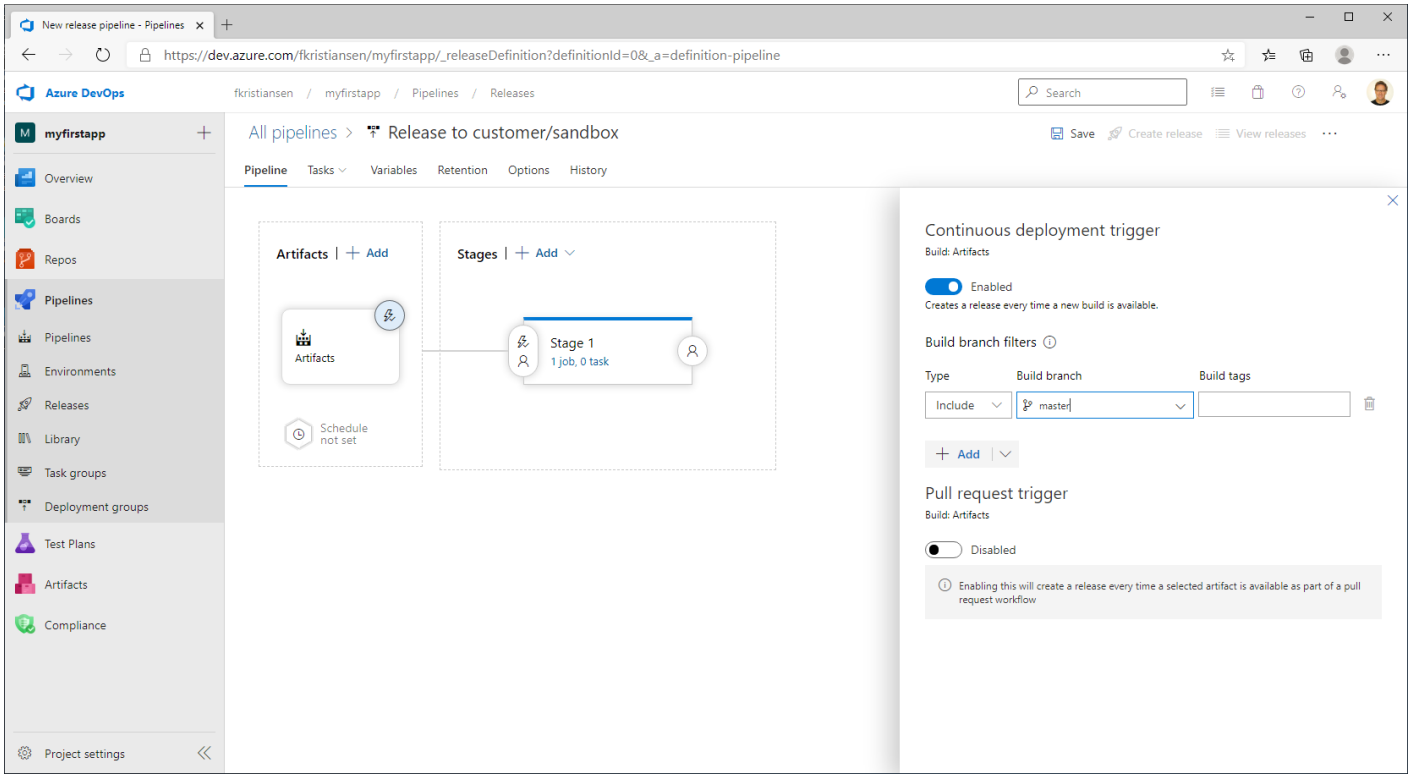
In **Variables**, **Variable Groups**, link the variable group **BuildVariables**



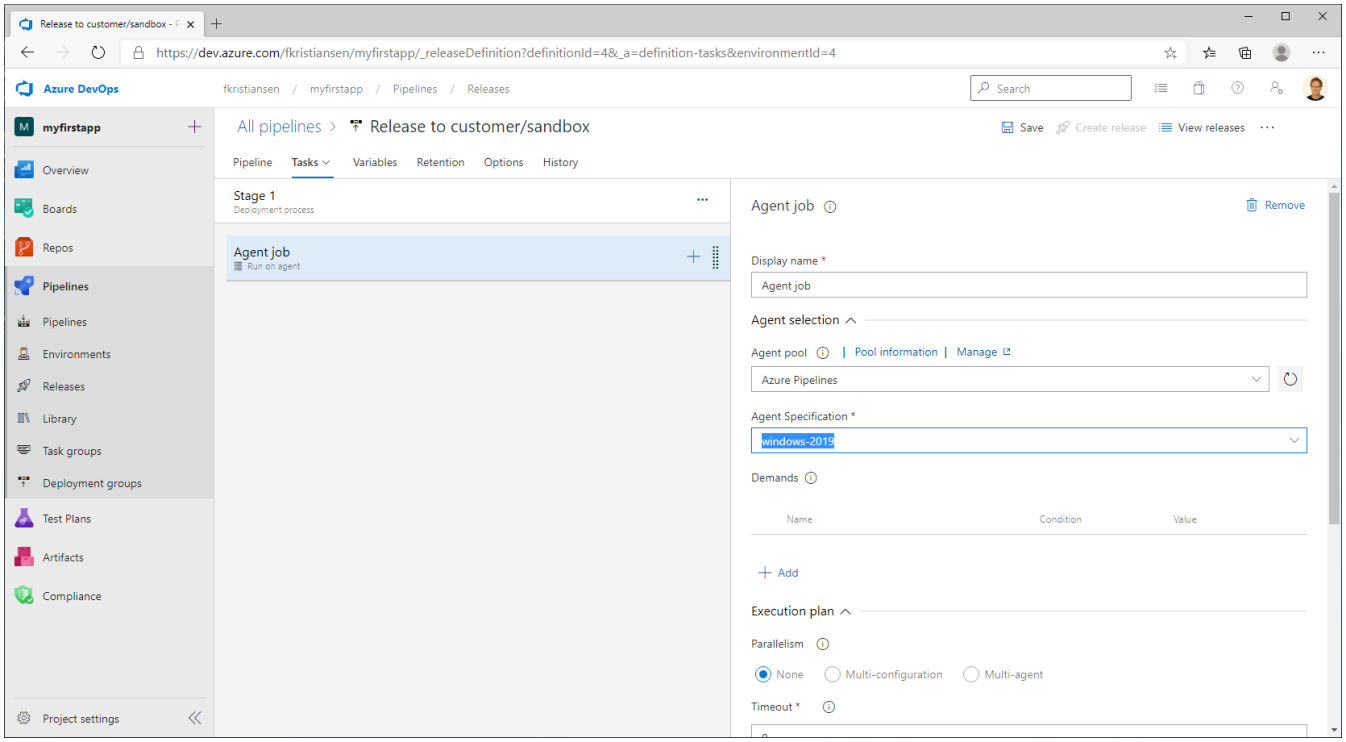
Under pipeline variables, define 3 variables: **CompanyName**, **TenantId** and **Environment**. Set the name of the pipeline to **Release to customer Sandbox**.



Under **Pipeline**, set the **Continuous Deployment Trigger** and **Save** the pipeline.

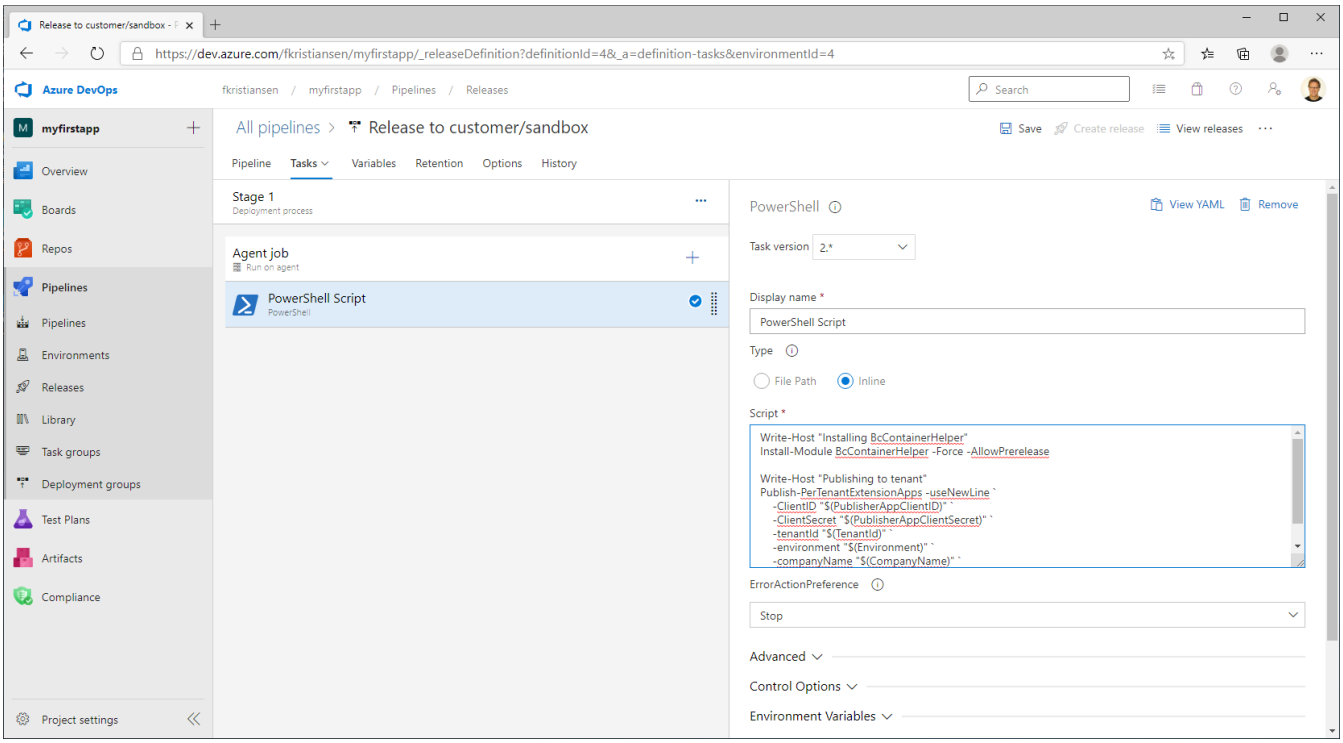


Select **Tasks**, click **Agent Job** and select **Windows-2019** as **Agent Specification**.

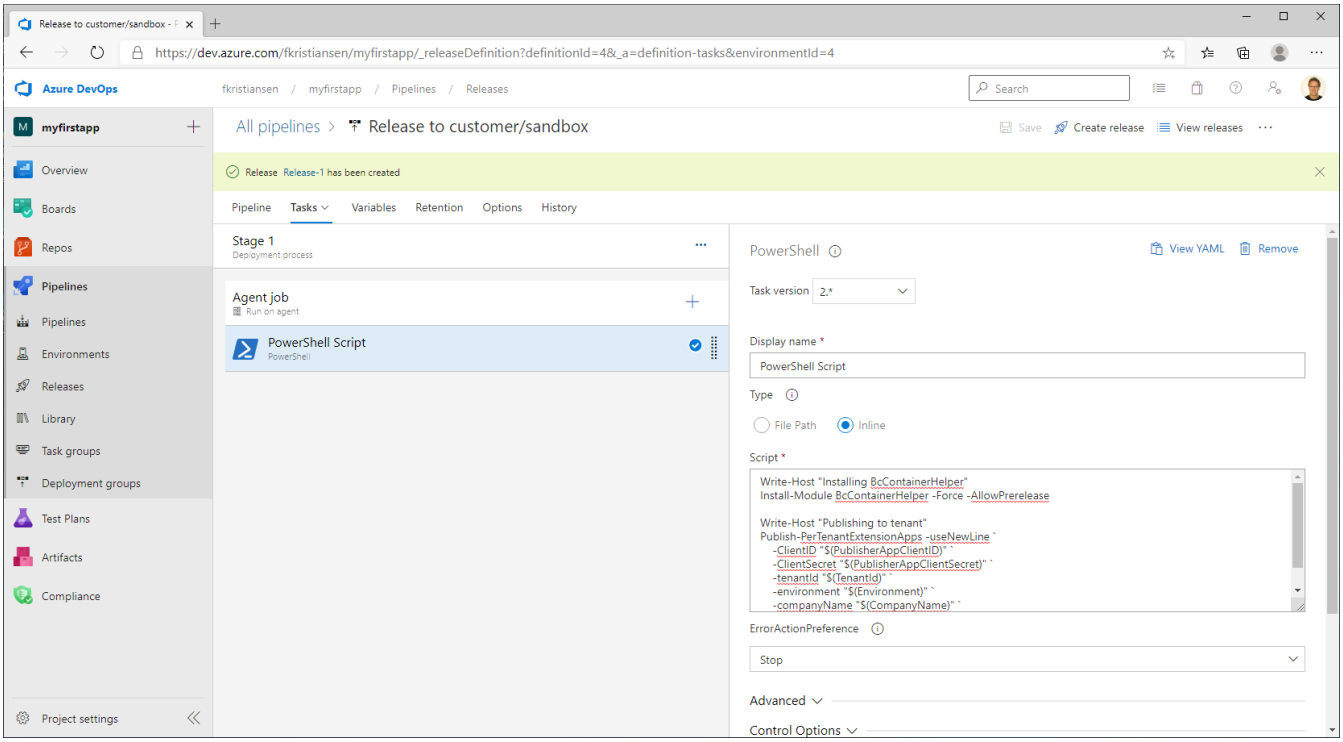


Click the + in Agent job to add a job, search for PowerShell and add a PowerShell Task. Click the task line and set the type to be inline and copy/paste the following script:

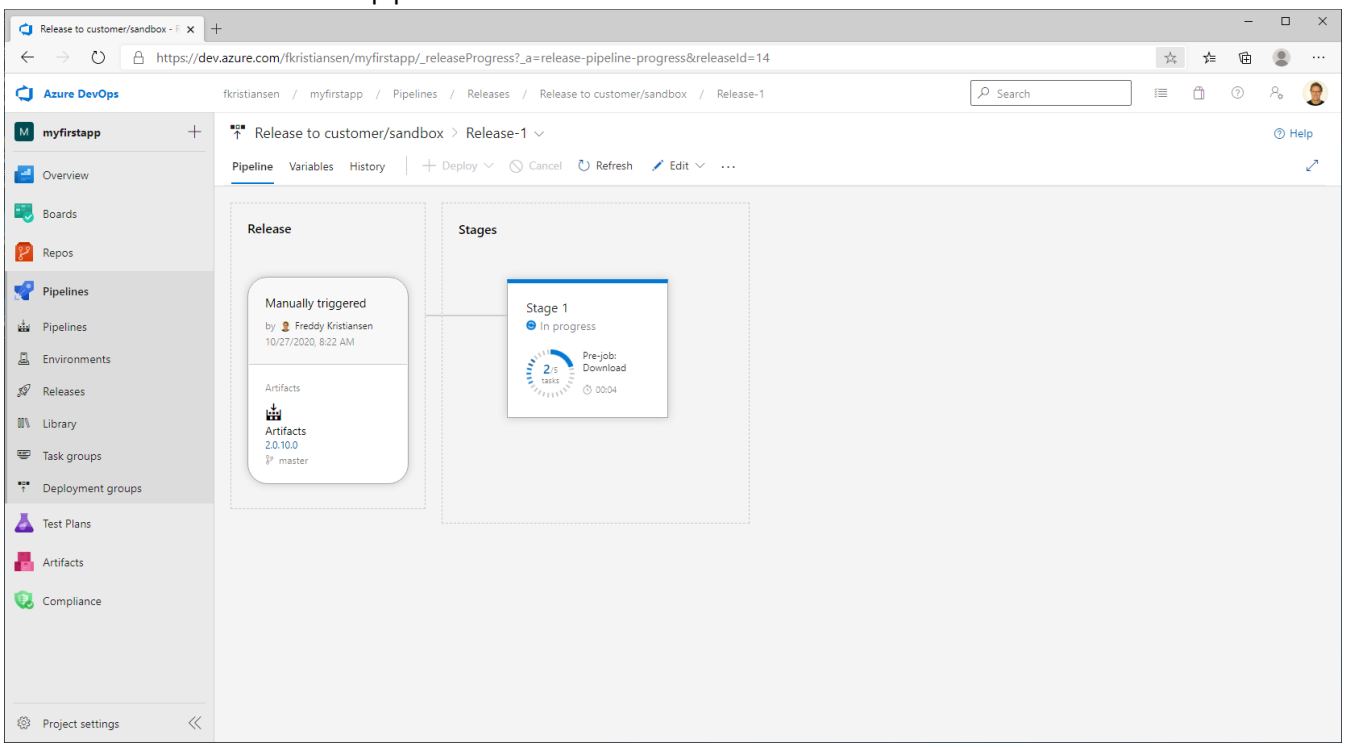
```
Write-Host "Installing BcContainerHelper"
Install-Module BcContainerHelper -Force -AllowPrerelease

Write-Host "Publishing to tenant"
Publish-PerTenantExtensionApps -useNewLine `
    -ClientID "$(PublisherAppClientID)" `
    -ClientSecret "$(PublisherAppClientSecret)" `
    -tenantId "$(TenantId)" `
    -environment "$(Environment)" `
    -companyName "$(CompanyName)" `
    -appFiles @(Get-Item "Artifacts/output/Apps/*.app" | % { $_.FullName })
```



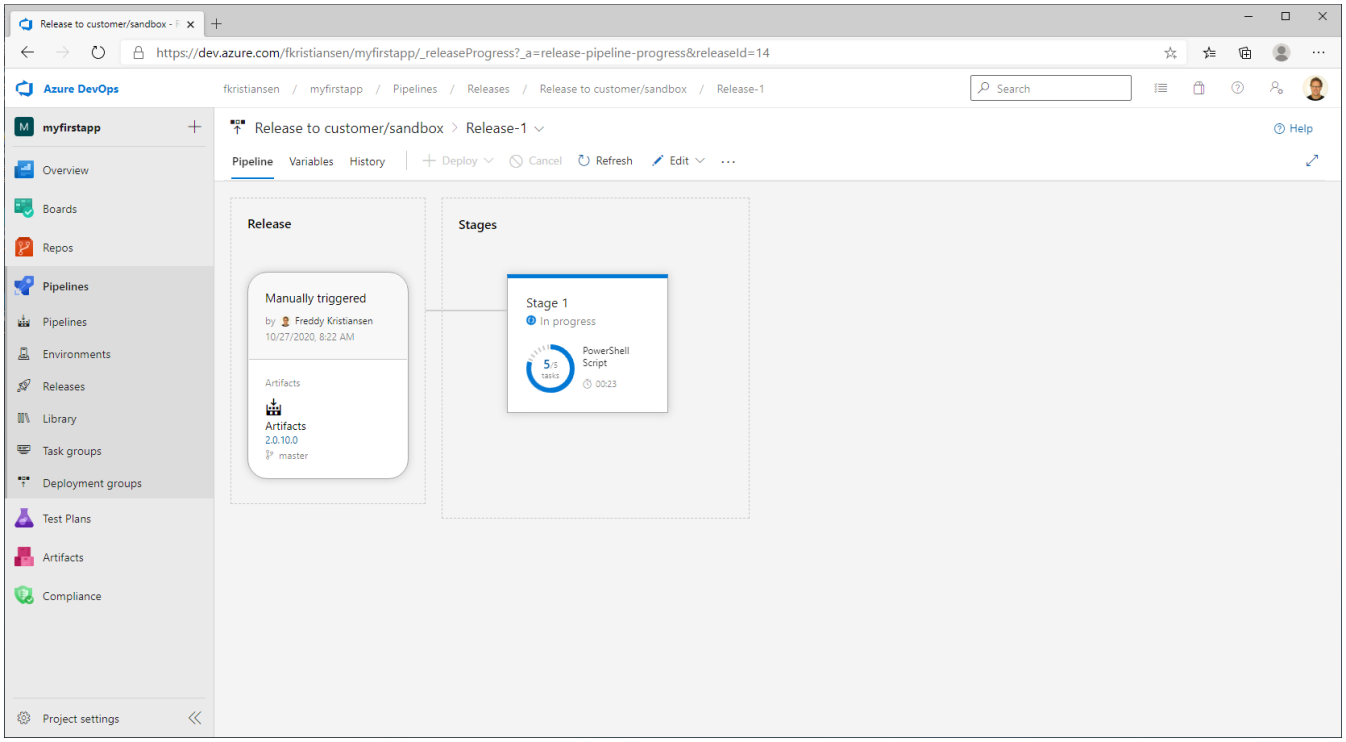
Save the pipeline and press **Create Release** to publish the latest version of your app to the **Sandbox** environment.



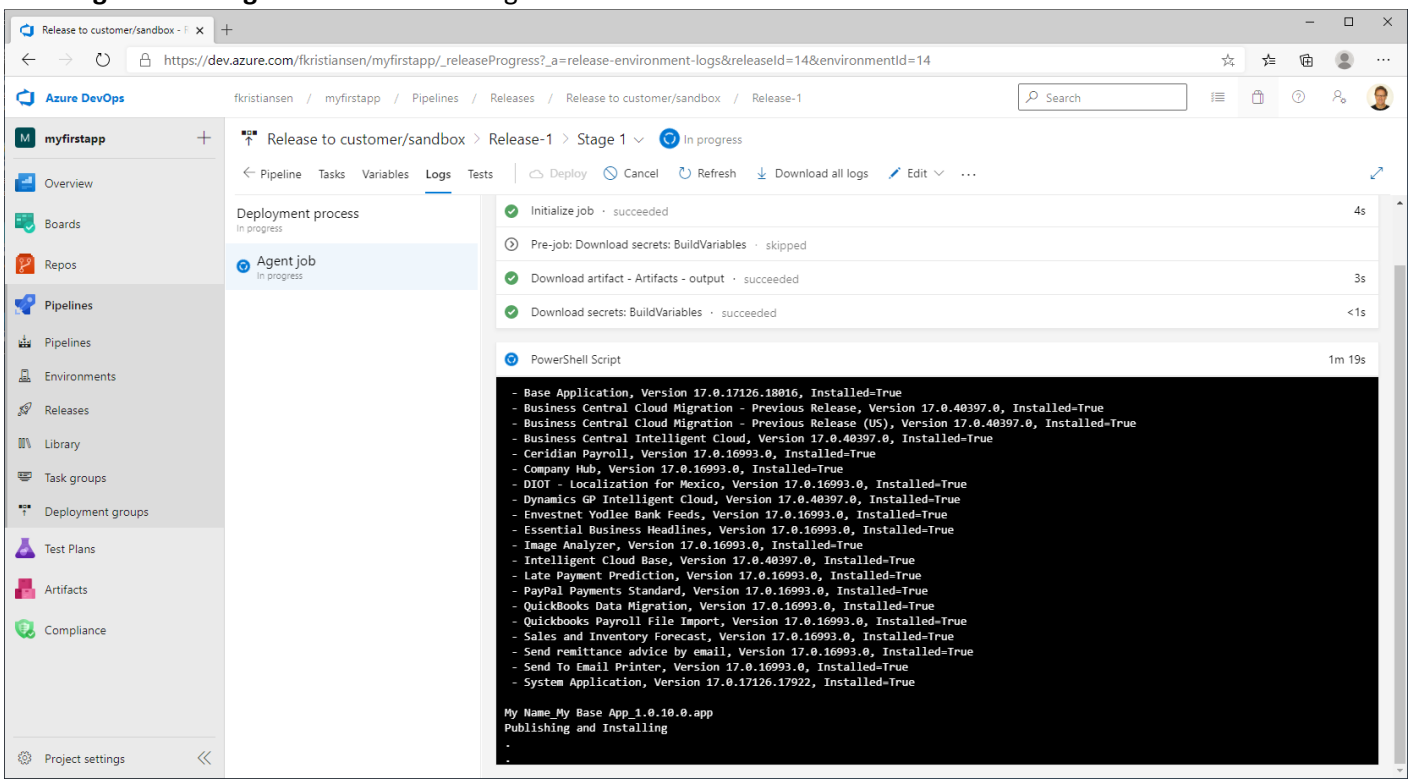
Click **Release-1** to monitor the pipeline



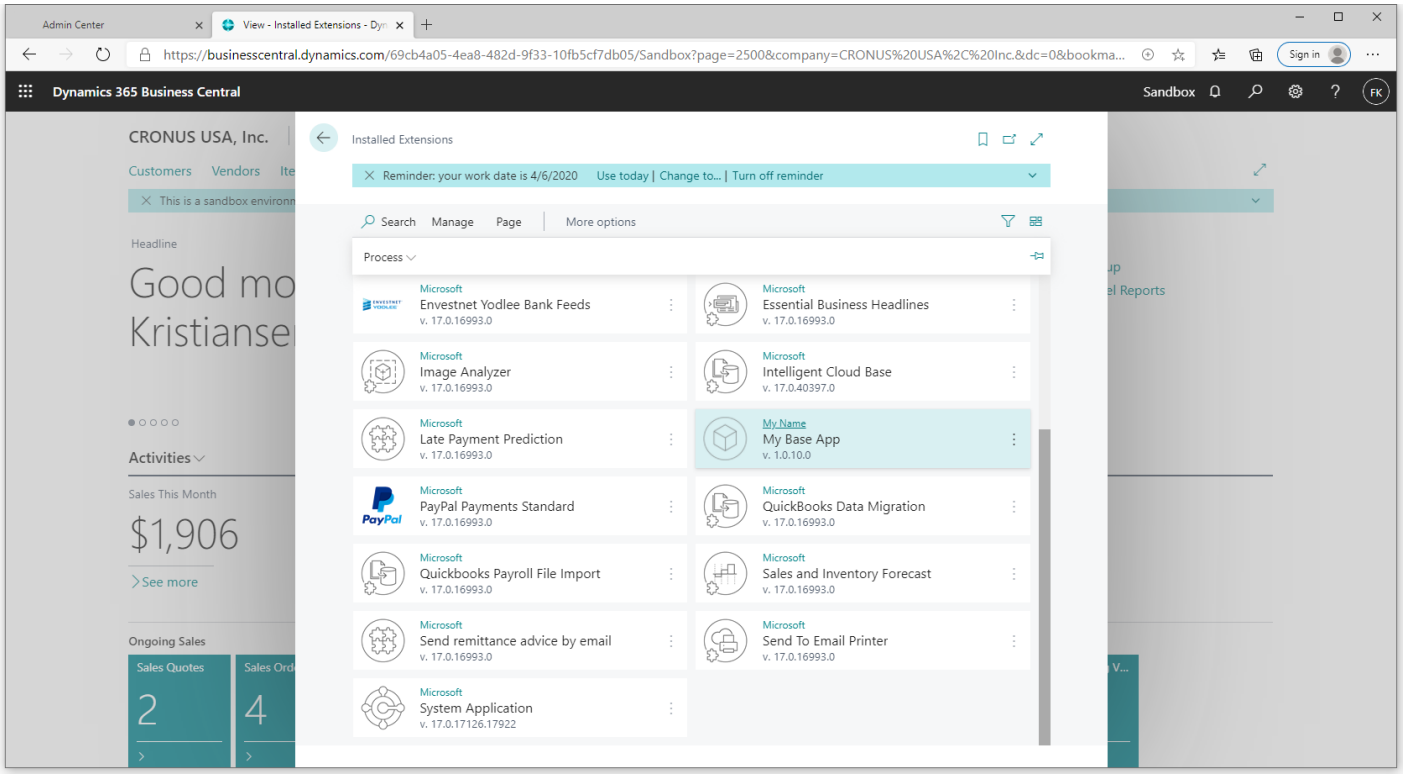
Running PowerShell Script



Click **Logs** below **Stage 1** to monitor the log



You can also login to your **sandbox environment** and see that the apps are getting installed



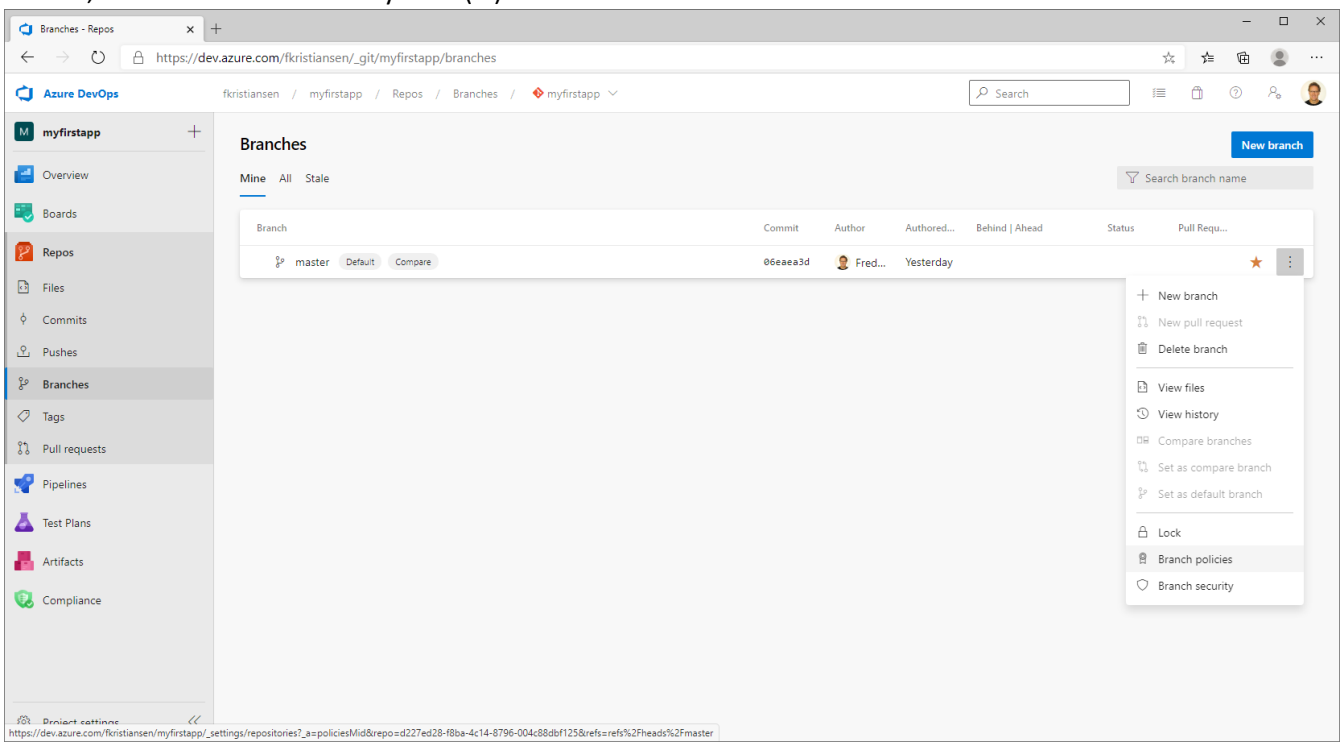
You can repeat the process for the production tenant, without the continouos deployment trigger and deploy the app on demand.

Congratulations, you have successfully published your app to an online tenant.

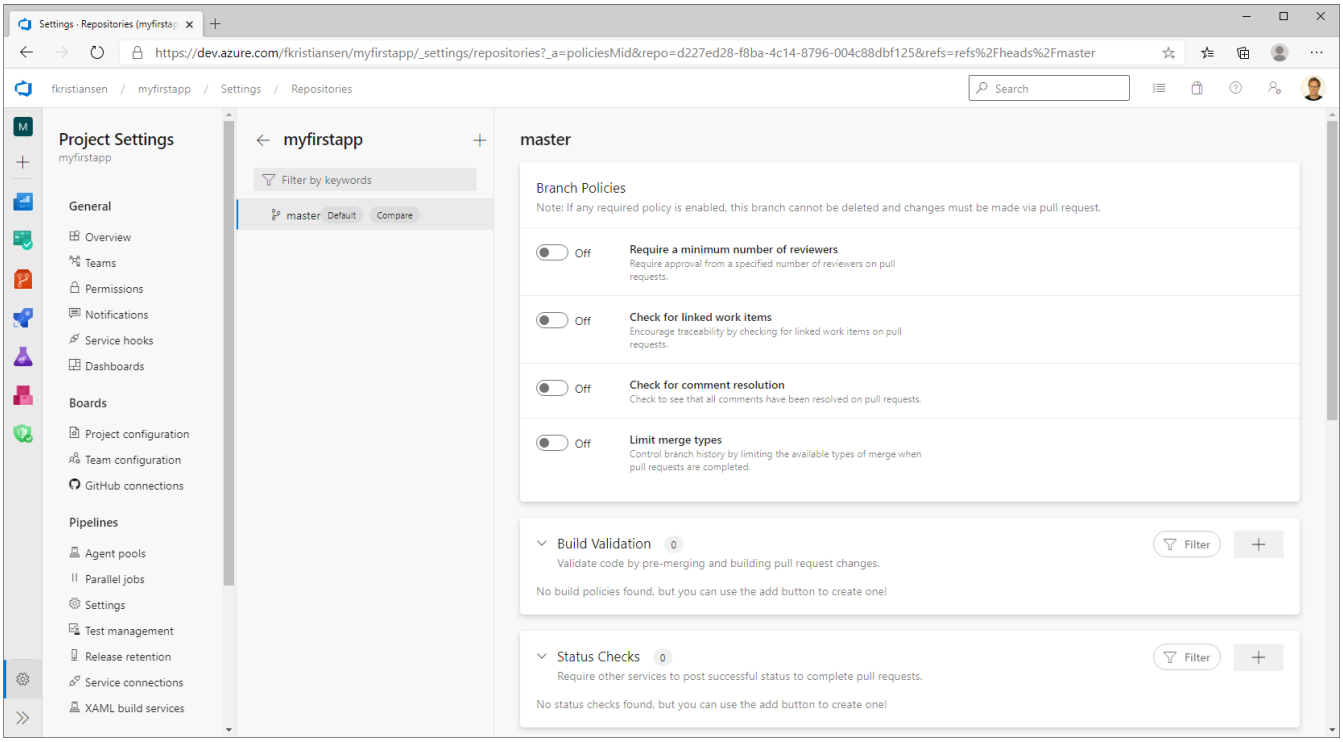
Branch Policies

At this time, we have an Azure DevOps project with a build pipeline, which we can kick off manually. We also have a release pipeline, which helps us release the product, but in order to raise the quality bar even more, we should setup branch policies to demand code reviews and successful builds before a fix makes it into the repository.

Open your Azure DevOps organization (eg. https://dev.azure.com//) and select **Repos** -> **Branches**. Select the **master** branch, click the **more actions** symbol (…) and choose **Branch Policies**.



You should now be able to setup branch policies for your branch

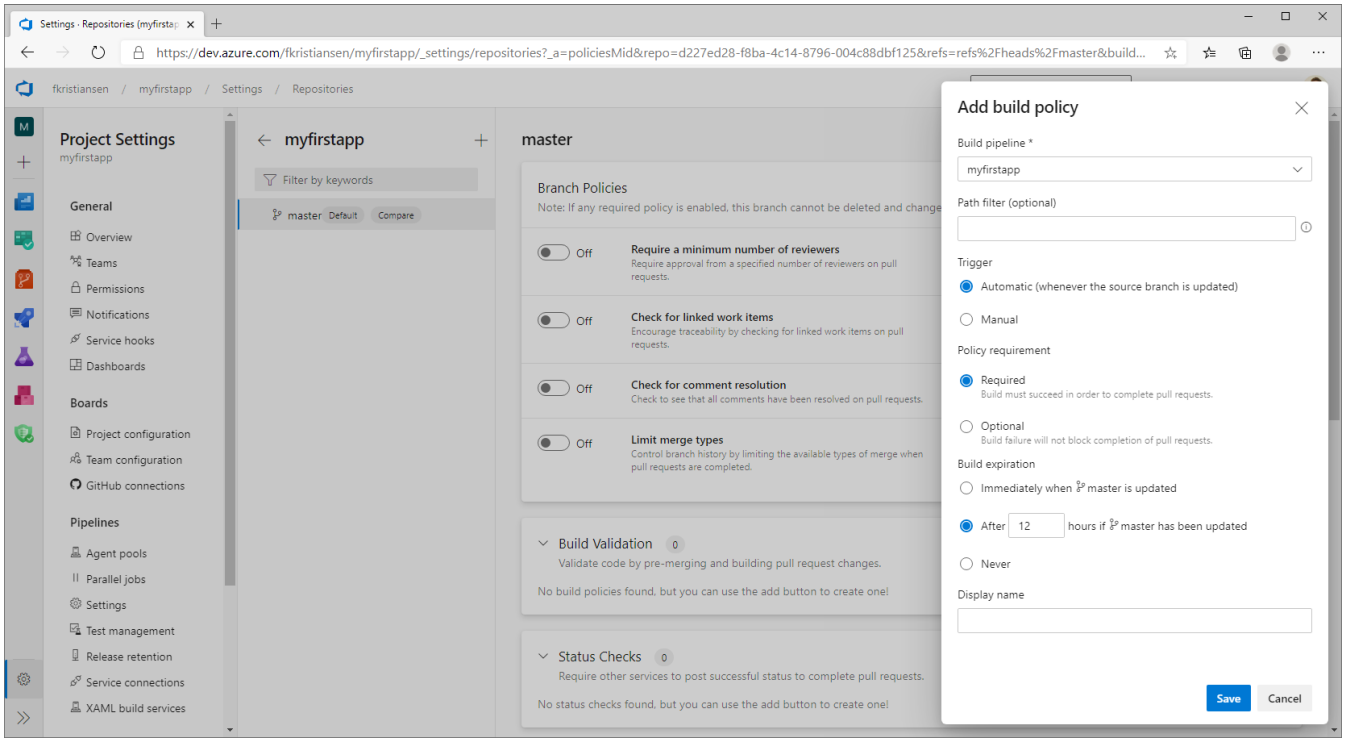


The first thing you will notice is, that setting **any Required policy** will **enforce** the use of **pull requests** and will **disallow direct checkins** to the **master** branch. It will also prevent the branch from unintended or evil deletion.

You can read much more about the policies here: https://docs.microsoft.com/en-us/azure/devops/repos/git/branch-policies.

What I want, is to ensure that my CI build pipeline runs whenever somebody checks something in and that the checkin cannot be completed if the build isn't successful. This is called Build validation and by adding a build policy

with our CI pipeline, we should be ready to go.



Congratulations

I know you think you are just getting started here, but with this, you have successfully completed this Hands On Lab.