# **Azure DevOps YAML Pipeline Templates Example**

Quinn Meagher | CSA App Innovation

## **Use-Case for YAML Templates?**

When creating an Azure DevOps pipeline, you can choose between the classic or YAML experience. The classic experience offers an interface in which a user can visually implement the series of tasks required to facilitate a deployment. Tasks can be grouped together into stages to control execution flow, often for multiple environments. Under the hood Azure DevOps uses a schema-based definition in YAML to represent the user’s pipeline configuration.

A YAML pipeline definition can be configured and executed as a pipeline without the need for a visual interface. The YAML experience provides an interface to reference, edit, and execute YAML schema definitions in code. By using YAML you gain access to valuable features such as pipeline versioning, standardized approvals/gates, and the use of YAML pipeline templates.

A YAML pipeline template is a small schema definition for a single task or collection of tasks that can be used in the composition of a complex pipeline. These templates are often stored in a centralized place in Azure DevOps and can be shared across projects.

Templates should be implemented such that organizational standards are built in. Consumers will directly reference a template in code along with the template’s required input parameters.

Some common use cases for pipeline templates include enforcing static code analysis, PR build validation definition of done, and zero-downtime deployments.

## **Overview**

For this example, we will be deploying a basic Azure Function App using YAML pipeline templates. By creating composable YAML pipelines using templates you can ensure that all development teams adhere to a standard for common application deployment needs.

The YAML templates used in this example showcase the ability to deploy an Azure Function with built-in zero downtime deployments using Azure App Service Deployment Slots.

Two environments are configured for deployment within our multi-stage YAML pipeline. Pipeline stages for higher level environments should be gated behind approvals.

## **Infrastructure**

For this example, there are a few cloud resources that need to be provisioned. Included with this documentation is a set of Azure CLI scripts that will create and destroy the resources for you. You can alternatively just create these resources manually in the Azure Portal.

Only one subscription is used in this example to reduce complexity. As such, environments are separated by resource groups. In a real-life implementation each environment would deploy to individual subscriptions using multiple Azure DevOps Service Connections

Required resources:

* demo-rg-dev (Resource Group)
  + demo-func-dev (Azure Function App – Consumption tier)
    - secondary-slot (Azure App Service Deployment Slot)
* demo-rg-prod (Resource Group)
  + demo-func-prod (Azure Function App – Consumption tier)
    - secondary-slot (Azure App Service Deployment Slot)

To run the included Azure CLI script, ensure that you have the Azure CLI downloaded.

[How to install the Azure CLI | Microsoft Learn](https://learn.microsoft.com/en-us/cli/azure/install-azure-cli)

Following a successful install, open a terminal window in the example directory.

To interact with the CLI we will need to authenticate to Azure. This can be accomplished using the command below.

*Az login*

[Sign in with Azure CLI — Login and Authentication | Microsoft Learn](https://learn.microsoft.com/en-us/cli/azure/authenticate-azure-cli)

The setup script has two required arguments **subscriptionId** and **location**.

To execute the included scripts, use the command below. Script requires bash/shell to run.

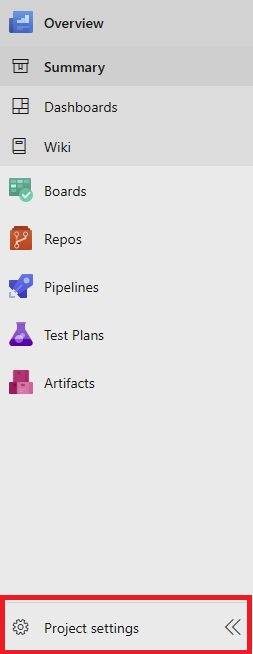
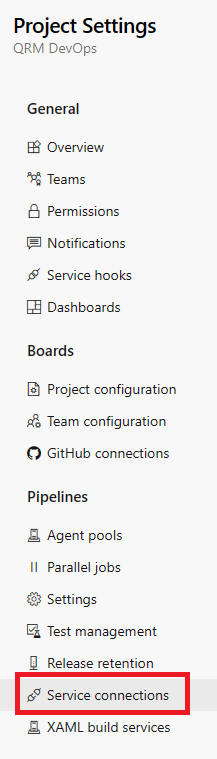
*./setup.sh* ***subscriptionId******location***

The required infrastructure should be deployed now. When the example is complete there will be a script to tear down these resources.

## **Service Connection**

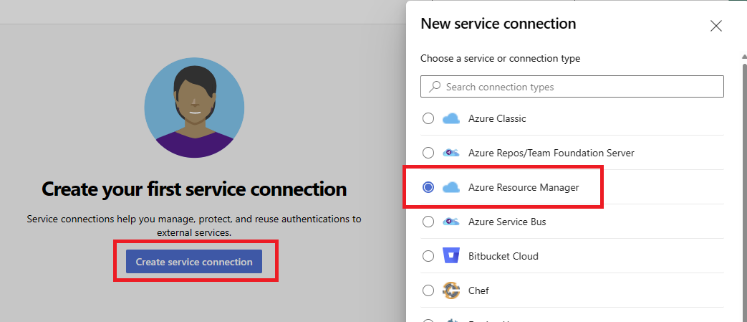
Before our Azure DevOps pipeline can deploy resources to Azure we need to set up a Service Connection. A Service Connection manages authorization to external services for your Azure DevOps project. To deploy to our Azure Function Apps we need to authorize our pipelines to interact with Azure Resource Manager (ARM) using a Service Connection.

For this demo we will be manually creating a Service Connection and the corresponding Azure Application Registration.

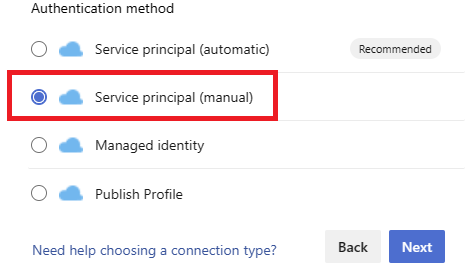


To create a new Service Connection, navigate to Project settings. Service Connections are configured for the entire project but can be authorized for specific pipelines.

From the Project Settings menu, select the Service connections blade.



Create a new service connection and select Azure Resource Manager as the type.



Service Principal (automatic) is identical to what we will configure in this example.

For the sake of understanding we will be manually creating the underlying configuration.

Graphical user interface, text, application, email

Description automatically generated

To create our Service Connection we will need some configuration from Azure.

Azure allows you to expose connections to the Microsoft Identity Platform using Azure Application Registrations.

Subscription Id, Subscription Name, Tenant Id, and Service Connection Name can be filled in at this time.

The Service Principal Id (Client Id) and Service Principal key (Client Secret) will come from the Azure App Registration we will create.

### 

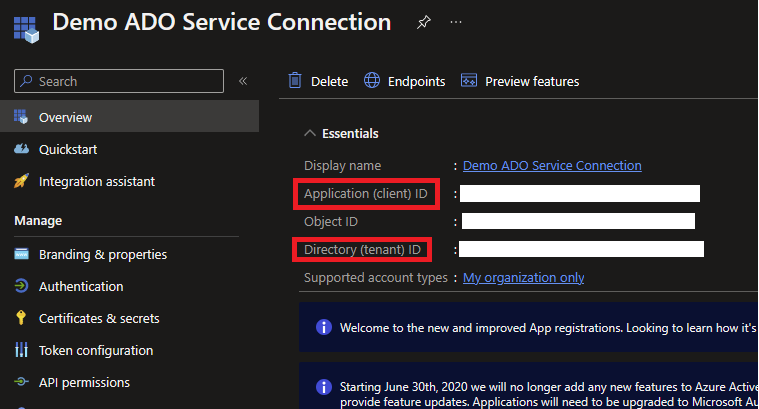
### **Create an Azure Application Registration**

To create the Azure App Registration, navigate to the App Registration Blade from within your Azure Active Directory.

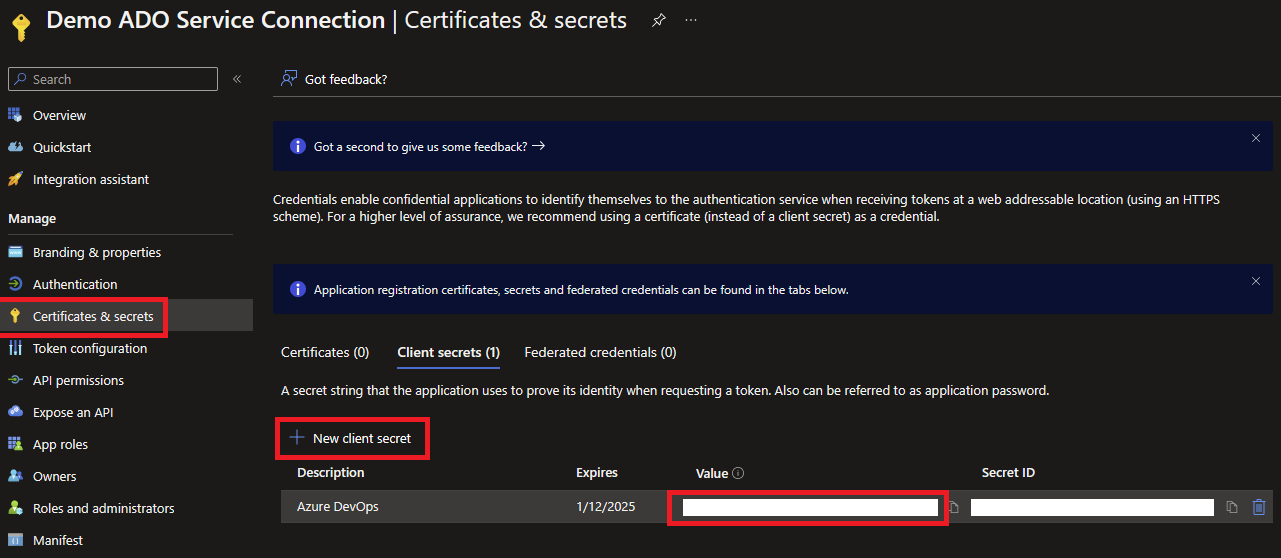
Select New registration.

A name is required for our Azure Application Registration.

No additional configuration here is required, go ahead and register our Azure Application Registration.



You can find the Service Principal Id (Client Id) and Tenant Id values in the Overview blade.



To create the Service Principal Key (Client Secret), navigate to the Certificates & secrets blade.

Select New client secret to create a new entry.

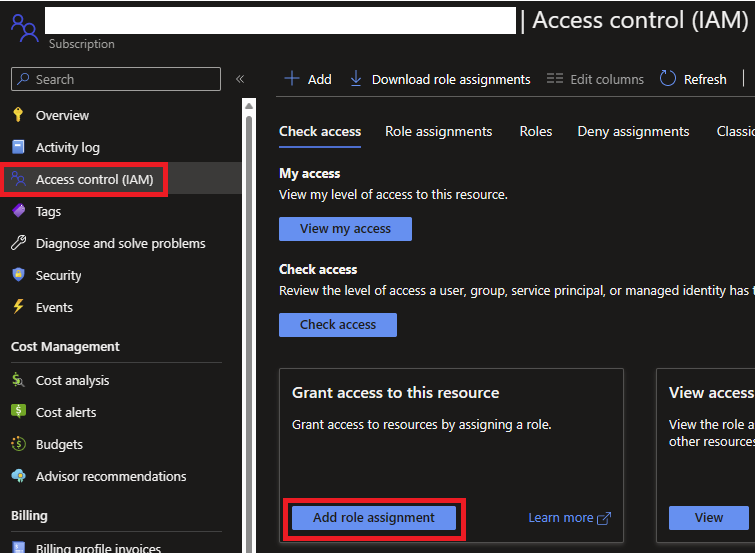
We now have all the configuration needed to complete the configuration for our Azure DevOps Service Connection.

Return to Azure DevOps and create the Service Connection.

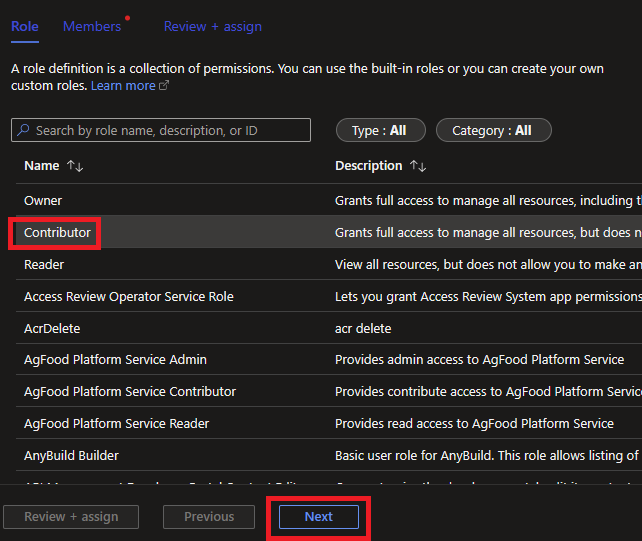
### **Configure the RBAC Assignment**

The last step before our pipelines can deploy to the Azure Function Apps is to configure a Role Based Access Control (RBAC) assignment for our Azure Application Registration so that it has the necessary authorization to deploy.

To configure the RBAC assignment, select your desired subscription from the Subscriptions blade.

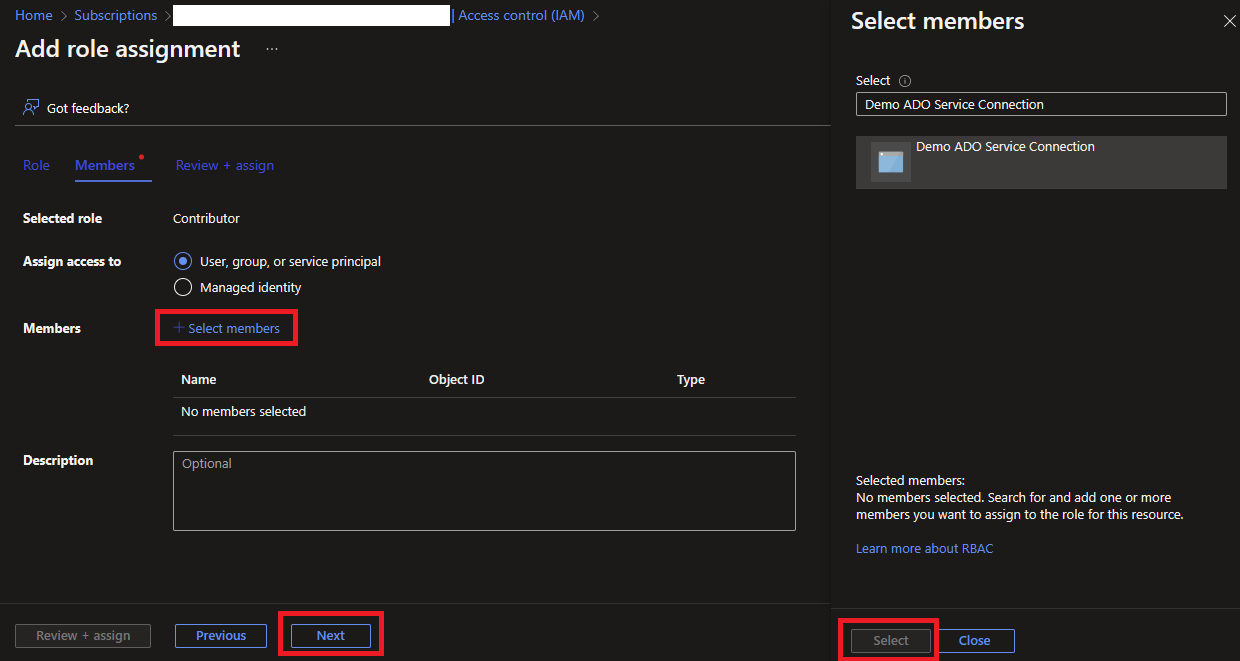
From your Subscription, select the Access control (IAM) blade.

Select Add role assignment.



For this demo we are using the Contributor role but it is recommended to use the Principle of least privilege.

Select Next.



Click the Select Members button to add members to this assignment.

When selecting an Azure Application Registration for a role assignment you must search for its name or else it will not show up on the list.

No additional configuration is required, the assignment can be completed.

We are now ready to set up and execute our pipelines. This demo uses two repositories, PipelineTemplates, and DemoFunction.

At this time move the PipelineTemplates and DemoFunction folders into their own Azure DevOps repositories.

## **PipelineTemplates Repository**

The PipelineTemplates repository contains YAML templates which will build up our pipeline. Templates should enforce enterprise standards and reduce the composition time for new pipelines.

This repository has a single top-level folder named YAML. Inside the YAML folder are modules for different resource types, each representing an enterprise deployment standard.

Modules contain Build Validation, Continuous Integration (CI), and Continuous (CD) templates. This demo only contains templates for Azure Function Apps. The description is intended to show general reusability.

### **Build Validation Template**

Text

Description automatically generatedThe build validation template is named azure-pipelines-build.yaml

This is used for build validation; a pipeline consuming this template can be configured as a pull request gate in the Build Policies menu.

A build validation pipeline consuming this template can be found in the DemoFunction repository.

Text

Description automatically generated

The Continuous Integration (CI) template is named azure-pipelines-ci.yaml

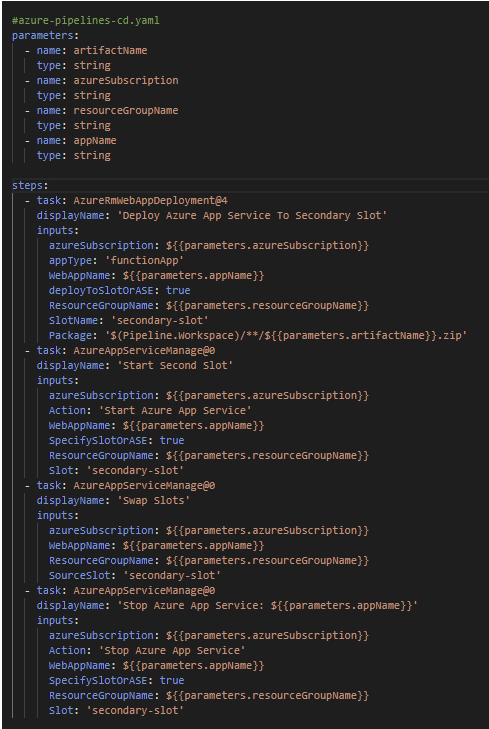
This is used in a multi-stage pipeline to build and publish the artifact to be deployed.

In a multi-stage pipeline this would be configured as the first stage. Stages after this would pick up the published artifact.

The Continuous Deployment (CD) template is named azure-pipelines-cd.yaml

This is used in a multi-stage pipeline to deploy the artifact created by our continuous integration.

In a multi-stage pipeline this would be configured once for each environment.



The CD template showcases how you can enforce enterprise standards using pipeline templates.

This template has zero downtime deployments baked in using Azure App Service Deployment Slots.

Consumers of this template can deploy to an Azure Function App quicker by passing in a thin layer of configuration and have baked in zero downtime deployments.

## **DemoFunction Repository**

The DemoFunction repository contains two pipelines. A single stage pipeline for build validation and a multi-stage pipeline for CI/CD.

These pipelines leverage Azure DevOps Environments which can configured so that approval is required for each stage to execute.

Higher level environments would normally be gated behind manual approvals or quality gates. This demo will not be using any approvals but they can be easily configured in the Environments blade from within your Azure DevOps project.

### 

### **Build Validation Pipeline**

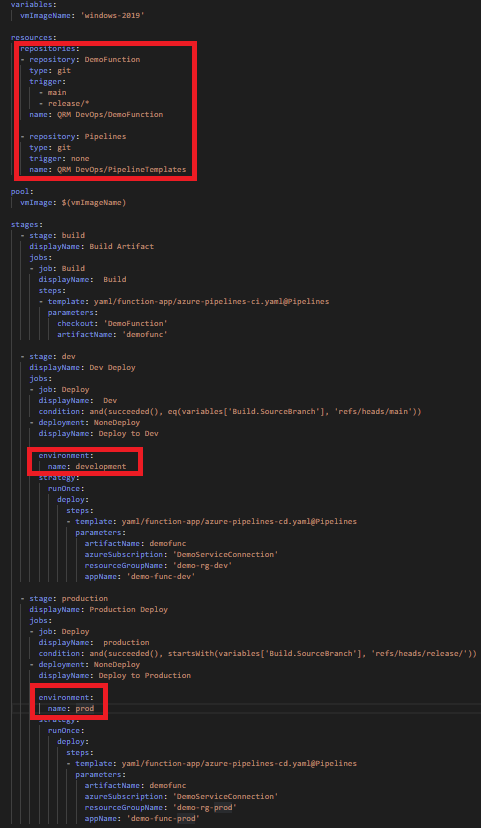
The build validation pipeline consumes the build validation template from our PipelineTemplates repository.

The connection to both repositories and template configuration can be seen highlighted in red.

Templates can be referenced using the file path. By specifying @Pipelines at the end you are telling the agent to look in that repository for the pipeline code.

The build validation pipeline uses trigger: none because it is manually executed by the PR rather than triggered by a branch operation.

### **Multi-Stage Pipeline**

The multi-stage pipeline consumes the CI and CD templates from our PipelineTemplates repository.

The connection to both repositories and template configuration can be seen highlighted in red.

Templates can be referenced using the file path. By specifying @Pipelines at the end you are telling the agent to look in that repository for the pipeline code.

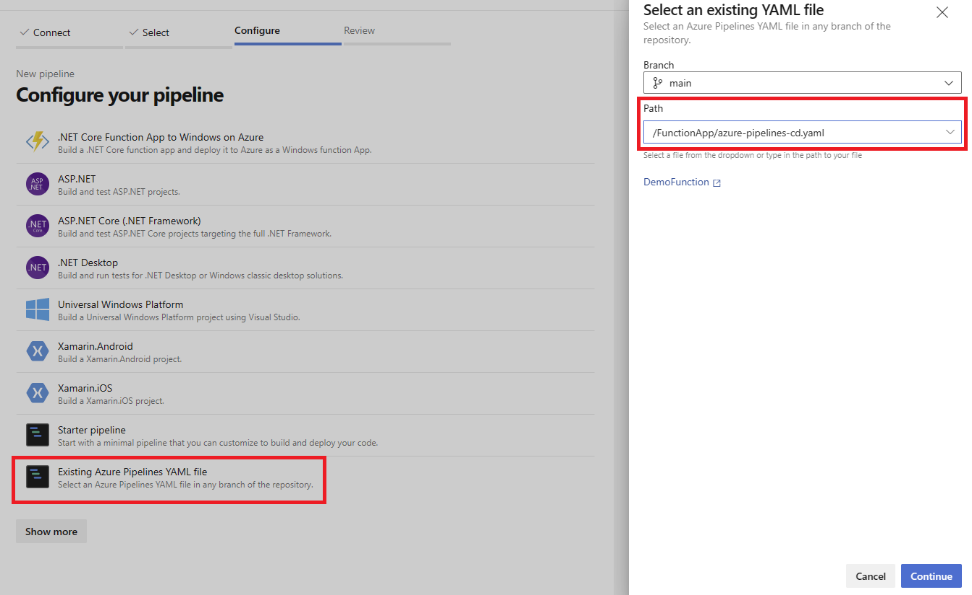
The multi-stage pipeline uses trigger: none because it is manually executed by the PR rather than triggered by a branch operation.

Two stages are configured, one for each environment. This is how pipelines can hook into environment specific approvals or quality gates.

The azureSubscription parameter is the name of our Service Connection. Normally you would have a Service Connection for each subscription.

### **Creating the Pipeline**

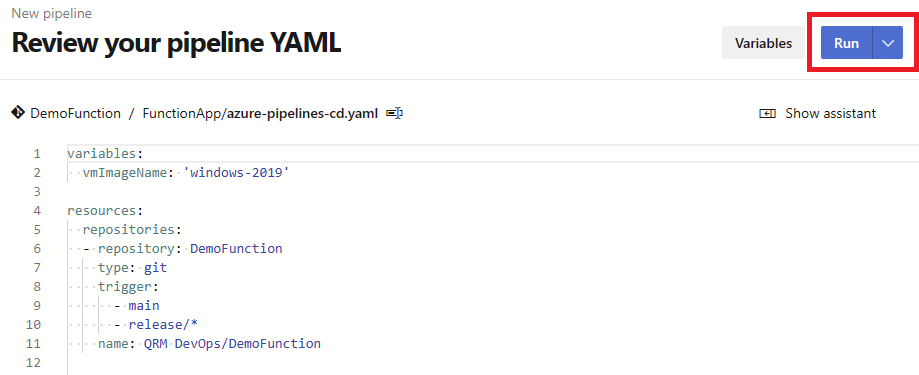
To create the multi-stage pipeline, navigate to the Pipelines blade in your Azure DevOps project.



Select New Pipeline and choose Azure repos Git (YAML).

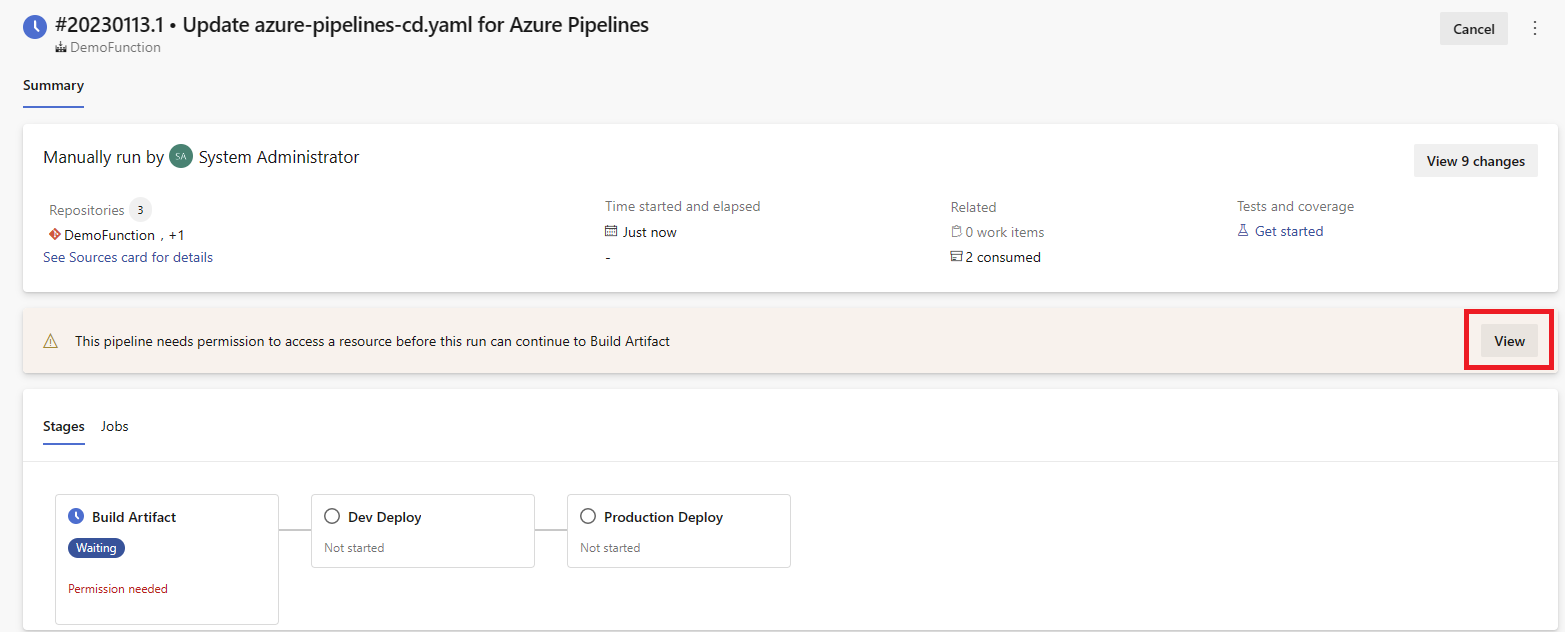
We can then select our DemoFunction repository.

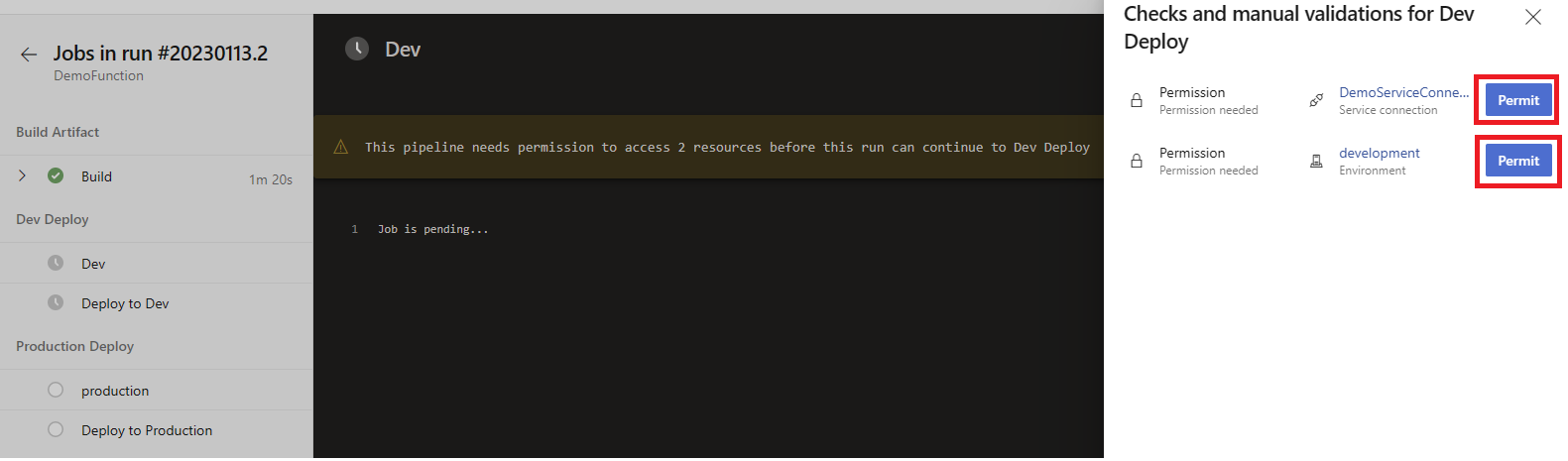
The example comes with the multi-stage pipeline so we will select Existing Azure Pipelines YAML file.



The review pipeline screen lets you made view and edit your YAML pipeline file.

Select Run to save and execute this pipeline.

New pipelines require one-time permissions to allow access to Service Connections and Environments.



Once everything has been authorized in the pipeline, we are able to deploy our resources.

Graphical user interface, application

Description automatically generated

The teardown script has one required argument, **subscriptionId**.

To execute the included teardown script, use the command below. Script requires bash/shell to run.

*./teardown.sh* ***subscriptionId***