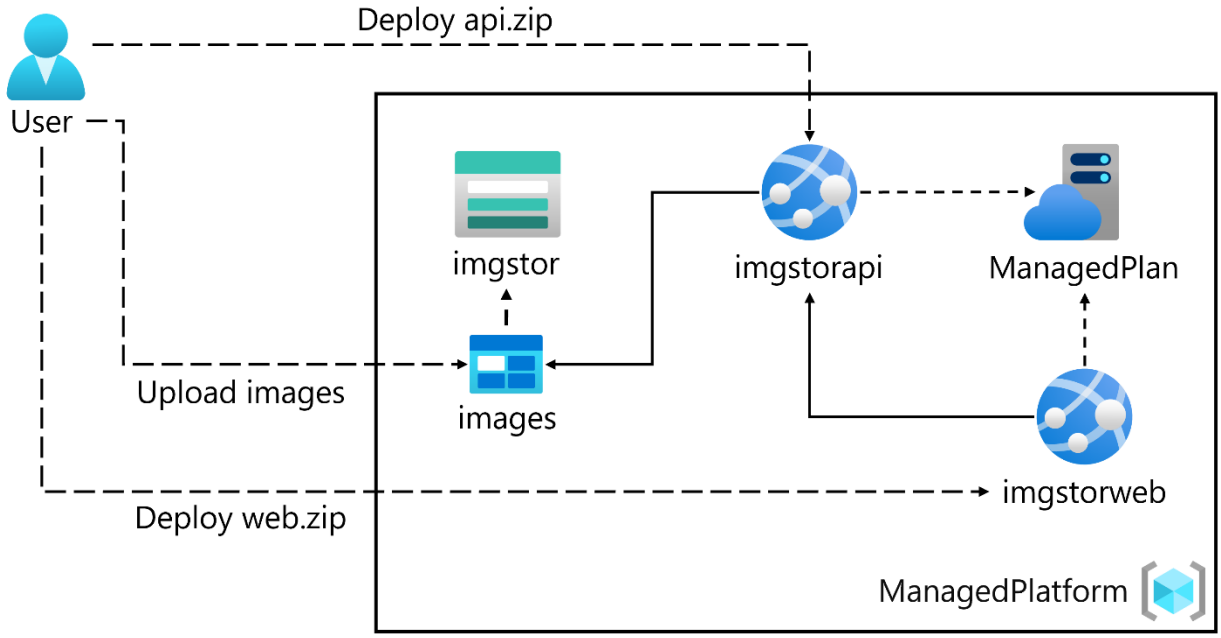
Exercise 1 Build Web App using PAAS

Architecture diagram



Part1: Build a backend API by using Azure Storage and the WebApp Service

Task 1: Open the Azure portal

Task 2: Create a Storage account

Task 3: Upload a sample blob

1. On the **Storage Account** blade, in the **Data storage** section, select the **Containers** link.
2. On the **Containers** blade, select **+ Container**.
3. In the **New container** window, perform the following actions:

Setting	Action
Name text box	Enter **images**
Public access level list	Select **Blob (anonymous read access for blobs only)**, and then select **Create**

4. On the **Containers** blade, select the newly created **images** container.
5. Upload any existing image inside images container

Task 4: Create a web app

1. Create WebApp with these configurations:

Name text box	Enter **imgapi***[yourname]*
Publish section	Select **Code**
Runtime stack drop-down list	Select **.NET 6 (LTS)**

Task 5: Configure the web app

1. On the **App Service** blade, in the **Settings** section, select the **Configuration** link.
2. In the **Configuration** section, perform the following actions, select **Save**, and then select **Continue**.

Setting	Action
Application settings tab	Select **New application setting**
Add/Edit application setting pop-up dialog	In the **Name** text box, enter **StorageConnectionString**
Value text box	Paste the storage connection string that you previously copied to Notepad
Deployment slot setting text box	Retain the default value, and then select **OK** to close the pop-up dialog and return to the **Configuration** section

3. On the **App Service** blade in the **Settings** section, select the **Properties** link.
4. In the **Properties** section, copy the value of the **URL** hyperlink, and then paste it to Notepad. You'll use this value later in the lab.

Task 6: Deploy an `ASP.NET` web application to Web Apps

1. Open **Visual Studio Code** icon.
2. On the **File** menu, select **Open Folder**.
3. In the **File Explorer** window, browse to **Starter\API**, and then select **Select Folder**.
4. On the **Explorer** pane of the **Visual Studio Code** window, expand the **Controllers** folder, and then select the **ImagesController.cs** file to open the file in the editor.
5. In the editor, in the **ImagesController** class on line 26, observe the **GetCloudBlobContainer** method and the code used to retrieve a container.
6. In the **ImagesController** class on line 36, observe the **Get** method and the code used to retrieve all blobs asynchronously from the **images** container.
7. In the **ImagesController** class on line 55, observe the **Post** method and the code used to persist an uploaded image to Storage.
8. On the taskbar, select the **Windows Terminal** icon.

- `az login`

- In the **Microsoft Edge** browser window, enter the email address and password for your Microsoft account, and then select **Sign in**.

- Enter the following command, and then select Enter to deploy the **api.zip** file to the web app that you created previously in this lab:

```
az webapp deployment source config-zip --resource-group ManagedPlatform --src api.zip --name <name-of-your-api-app>
```

Note: Replace the <*name-of-your-api-app*> placeholder with the name of the web app that you created previously in this lab.

Wait for the deployment to complete before you continue with this lab.

16. On the Azure portal's **navigation** pane, select the **Resource groups** link.
17. On the **Resource groups** blade, select the resource group that you created previously in this lab.
18. select the **imgapi***[yourname]* web app that you created previously in this lab.
19. From the **App Service** blade, select **Browse**.

Note: The **Browse** command will perform a GET request to the root of the website, which returns a JavaScript Object Notation (JSON) array. This array should contain the URL for your single uploaded image in your Storage account.

Part 2: Build a front-end web application by using Azure Web Apps

Task 1: Create a web app

1. **Create a web app**

Setting	Action
Publish section	Select **Code**
Runtime stack drop-down list	Select **.NET 6 (LTS)**

Task 2: Configure a web app

1. On the **App Service** blade, in the **Settings** section, select the **Configuration** link.
2. In the **Configuration** section, perform the following actions, select **Save**, and then select **Continue**:

Setting	Action
Application settings tab	Select **New application setting**
Add/Edit application setting pop-up dialog	In the **Name** text box, enter **ApiUrl**
Value text box	Enter the web app URL that you copied previously in this lab. **Note**: Make sure you include the protocol **https://**, in the URL that you copy into the **Value** text box for this application setting
Deployment slot setting text box	Retain the default value, and then select **OK**

Task 3: Deploy an `ASP.NET` web application to Web Apps

1. On the taskbar, select the **Visual Studio Code** icon.
2. On the **File** menu, select **Open Folder**.
3. In the **File Explorer** window, browse **Starter\Web**, and then select **Select Folder**.
4. On the **Explorer** pane of the **Visual Studio Code** window, expand the **Pages** folder, and then select the **Index.cshtml.cs** file to open the file in the editor.
5. In the editor, in the **IndexModel** class on line 30, observe the **OnGetAsync** method and the code used to retrieve the list of images from the API.
6. In the **IndexModel** class on line 41, observe the **OnPostAsync** method and the code used to stream an uploaded image to the backend API.
7. On the taskbar, select the **Windows Terminal** icon.
8. At the open command prompt, enter the following command, and then select Enter to sign into the Azure CLI:

Use the following command to deploy a zip to webapp.

- `az webapp deployment source config-zip --resource-group <name-of-your-rg> --src web.zip --name <name-of-your-web-app>`

Note: Replace the <*name-of-your-web-app*> placeholder with the name of the web app that you created previously in this lab. You recently queried this app's name in the previous steps.

Wait for the deployment to complete before you continue with this lab.

- On the Azure portal's **navigation** pane, select **Resource groups**.

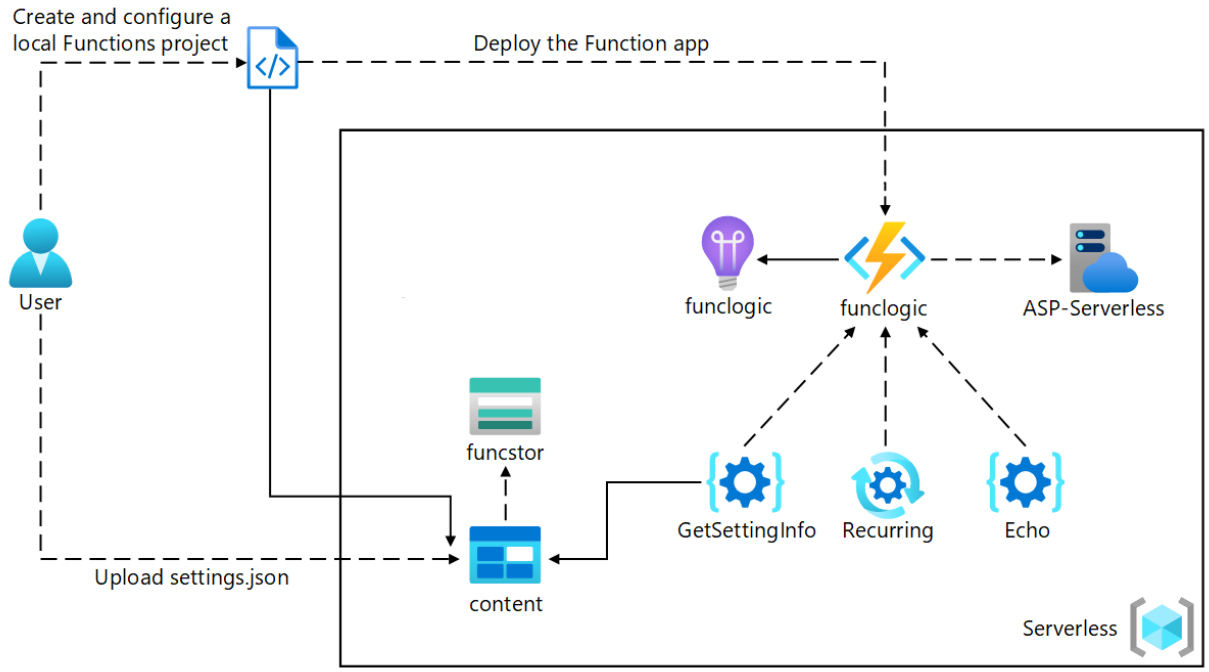
- On the **Resource groups** blade, select your resource group that you created previously in this lab and select the **imgweb***[yourname]* web app that you created previously in this lab.

Click **Browse** and Observe the list of images in the gallery. The gallery should list a single image that was uploaded to Storage previously in the lab.

b. Try uploading any existing image and observe that the list of gallery images has updated with your new image.

Exercise 2 **Azure Pipelines & Functions**

Architecture diagram



Exercise 1: Create Azure resources

Task 1: Open the Azure portal

Task 2: Create an Azure Storage account

Resource group section	Select **Create new**, enter **Serverless**, and then select **OK**
Storage account name text box	Enter **funcstor***[yourname]*

1.
2. On the **Overview** blade, select the **Go to resource** button to navigate to the blade of the newly created storage account.
3. On the **Storage account** blade, in the **Security + networking** section, select **Access keys**. Copy connection string to Notepad

Task 3: Create a function app

1.

Setting	Action
Resource group section	Select **Serverless**
Function App name text box	Enter **funclogic***[yourname]*
Publish section	Select **Code**
Runtime stack drop-down list	Select **.NET**
Version drop-down list	Select **6**
Region drop-down list	Select the **East US** region

2. Go to the function app created ➔ Configuration
3. Add new app Setting ➔ AzureWebJobsStorage and enter the value of the connection string of the storage account previously copied

   Note: Wait for the creation task to complete before you move forward with this lab.

Review

In this exercise, you created all the resources that you'll use in this lab.

Exercise 2: Configure a local Azure Functions project

Task 1: Initialize a function project

1. Create empty Directory **Starter\func** empty directory:
2. In the same directory initiate a the below command to create a new local Azure Functions project in the current directory using the **dotnet** runtime:

```
func init --worker-runtime dotnet --force
```

Task 2: Configure a connection string

1. On the **Start** screen, select the **Visual Studio Code** tile.
2. On the **File** menu, select **Open Folder**.
3. In the **File Explorer** window that opens, browse to **Allfiles (F):\Allfiles\Labs\02\Starter\func**, and then select **Select Folder**.
4. On the **Explorer** pane of the **Visual Studio Code** window, open the **local.settings.json** file.
5. Observe the current value of the **AzureWebJobsStorage** setting:

   code

```
5.  "AzureWebJobsStorage": "UseDevelopmentStorage=true",
6.
```

7. Change the value of the **AzureWebJobsStorage** element to the **connection string** of the storage account that you recorded earlier in this lab.
8. Save the **local.settings.json** file.

Task 3: Build and validate a project

1. On the taskbar, select the **Windows Terminal** icon.
2. Run the following command to build the project inside the directory of the function:

```
dotnet build
```

Exercise 3: Create a function that's triggered by an HTTP request

Task 1: Create an HTTP-triggered function

• Run the following command to use the **Azure Functions Core Tools** to create a new function named **Echo** using the **HTTP trigger** template:

code

```
3.  func new --template "HTTP trigger" --name "Echo"
```

Task 2: Write HTTP-triggered function code

1. Replace **Echo.cs** with the below:

c#

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.Azure.WebJobs;
using Microsoft.AspNetCore.Http;
using Microsoft.Extensions.Logging;
public static class Echo
{
    [FunctionName("Echo")]
    public static IActionResult Run(
        [HttpTrigger("POST")] HttpRequest request,
        ILogger logger)
    {
        logger.LogInformation("Received a request");
        return new OkObjectResult(request.Body);
    }
}
```

Select **Save** to save your changes to the **Echo.cs** file.

Task 3: Test the HTTP-triggered function by using httprepl

1. **Make sure than you're in Starter\func** directory
2. Run the following command to run the function app project:

```
func start --build
```

Exercise 4: Azure DevOps Repos and Pipelines

Task1: Add your changes to Azure DevOps

1. create a new repository
2. Clone it to a folder on your Desktop
3. Add previous working directory
4. Add & commit your git changes

Task2: Create a new Build Pipeline

1. On top right create new pipeline
2. Choose classic editor to create a pipeline without YAML
3. Bind it to the repository that was already created in Task1
4. Start with empty Job
5. Make sure you specify the self-hosted agent
6. Add a new Task using + Icon
7. Search for .NET Core
8. For pipeline needed steps **follow the Image guidance** at end of lab

   Build Steps should contain the following

1. Use .NET version 6.0.x (This step will install DotNet6 to the agent)
2. .NET Core step to restore
3. .Net Core to build
4. .Net Core to publish
5. Public Artifact: drop

Save and trigger the build pipeline in case it succeeded proceed to Task3

Task3: Add a new Service Connection:

Project Settings → Service connections → New Service Connection

Choose Azure Resource Manager(manual) and fill it with the below info:

Subscription ID: 8abda4b1-ce7a-4059-a145-0107b284f68a

Subscription Name: UA-CloudComputing

Service Principal Id: 63ba738a-84bd-4679-b207-43db99db5e46

Service Principal key: JRX8Q~yIQcas.RitYsdISTsQ3tcBNITMTZW5mbdd

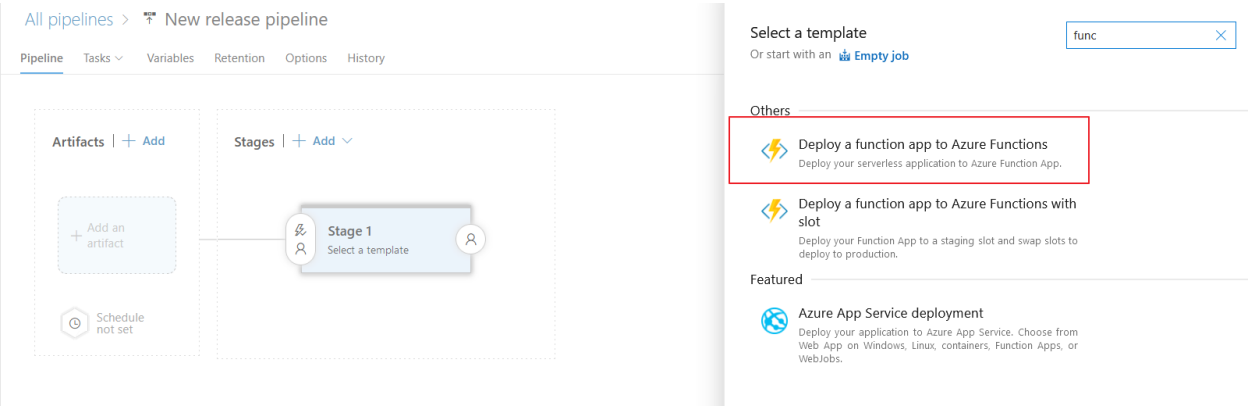
Tenant ID: 262bb0ea-0d60-4819-9b9b-095782cd7499

Give your Service Connection a Name

Click verify and Grant access permission to all pipelines checkbox and Save

Task 4: Create Release Pipeline

Under Releases click on the new button to create a new release pipeline and use Deploy a function as showing in the below screenshot



In the add an artifact chose the build pipeline.

Inside Stage 1 Fill the needed information as following:

Under Azure Subscription choose the Service connection which was created in the previous step

For Azure Functions App name field choose Function app that was created earlier in azure Portal will show below.

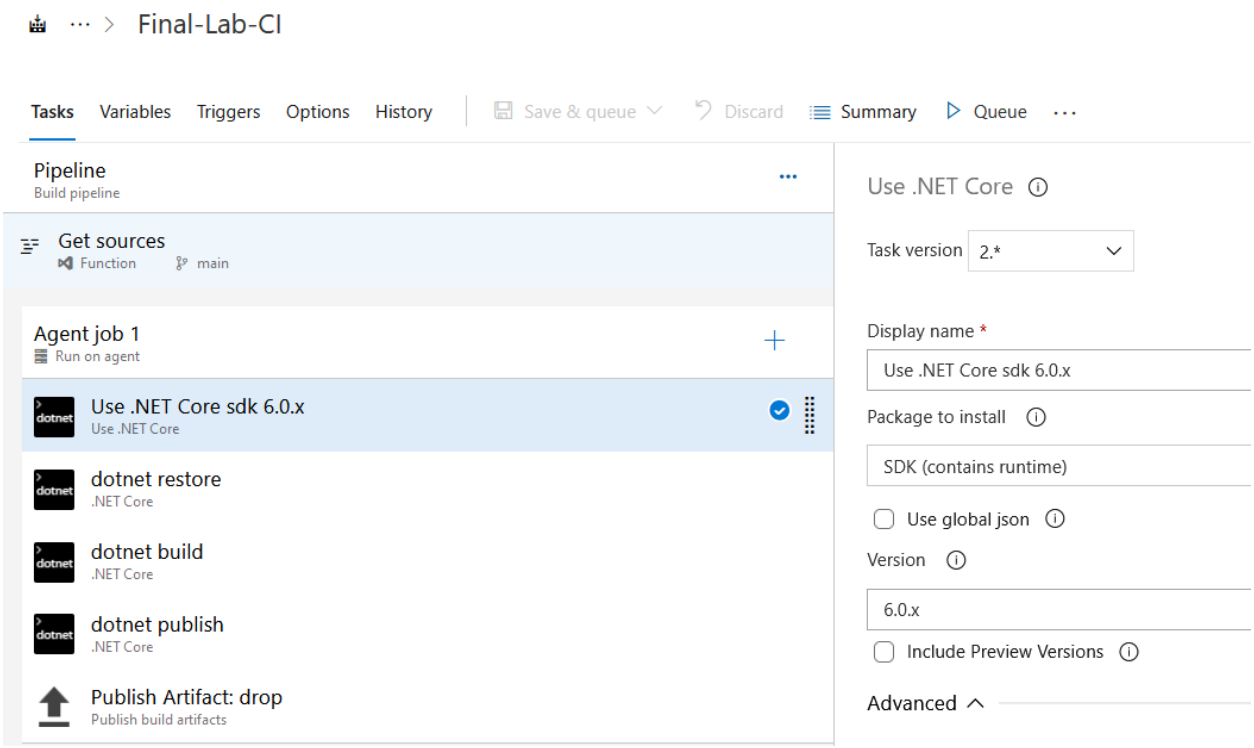
After Saving, click on Create Release which will deploy the code that was built in the Build Pipeline to the Function App which was created in Azure Portal.

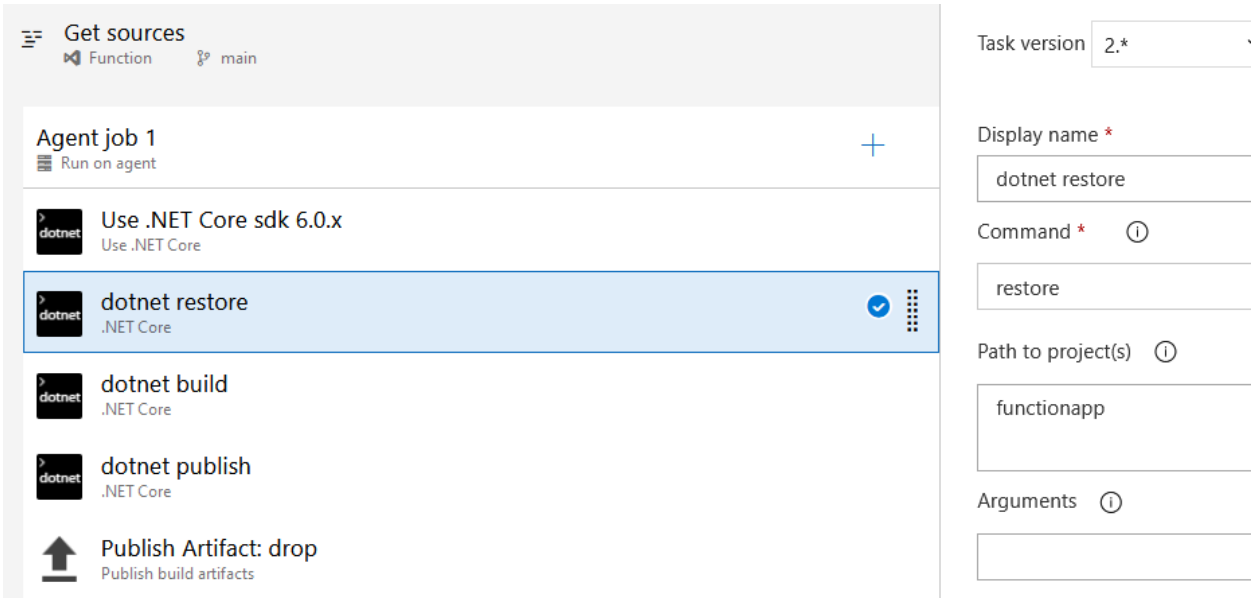
Task5: Verify Functionality of the Function APP in Azure Portal By triggering it using HTTP Trigger.

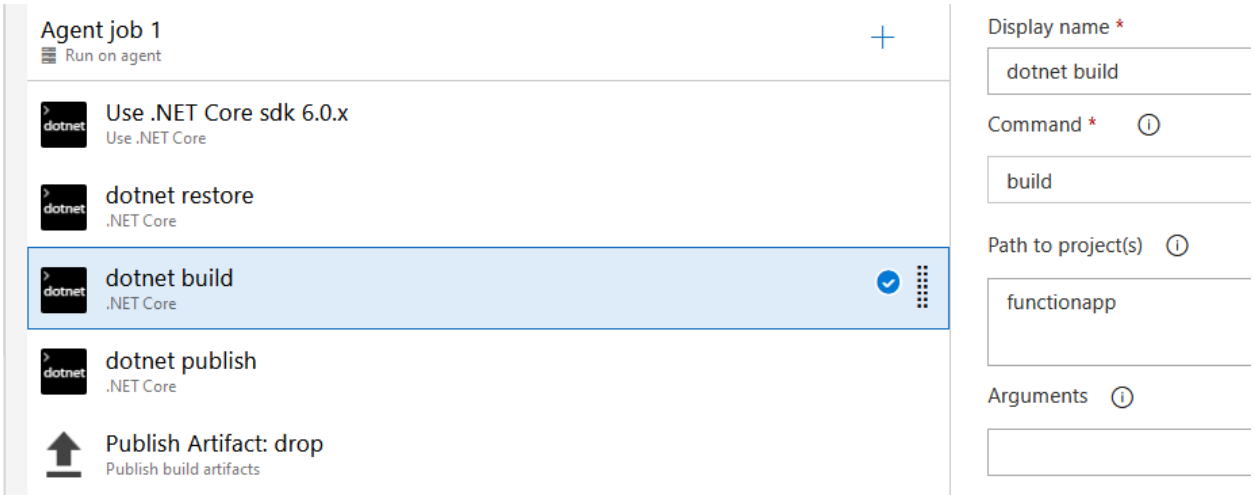
Click Function App → Specify the Function that was deployed

Code + Test Section → Test/RUN Button

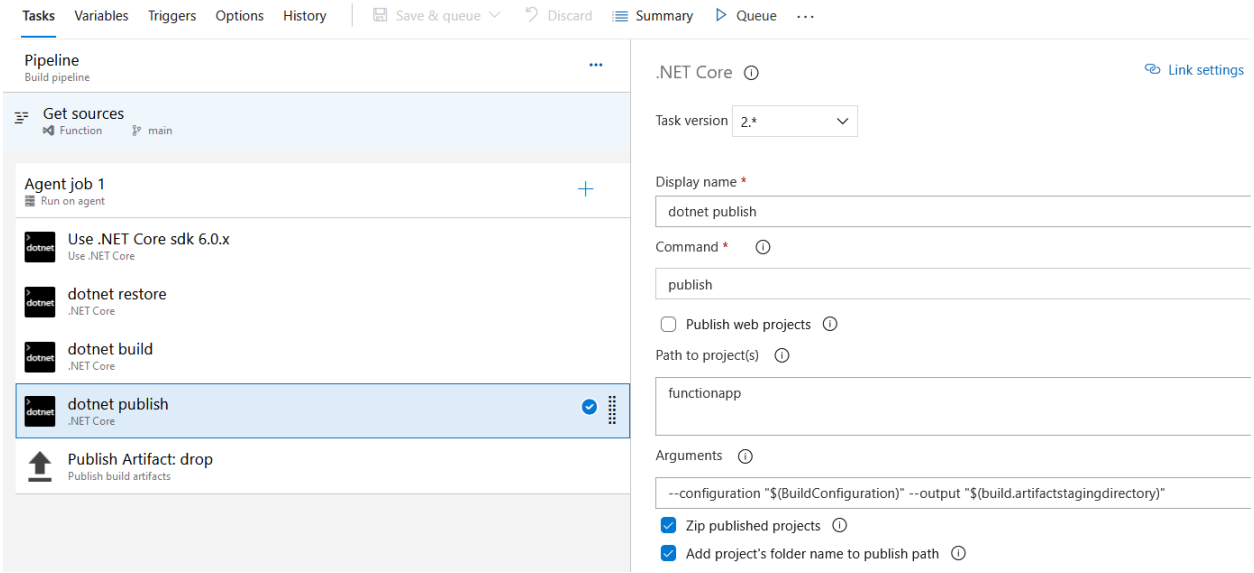
Reference for Build Pipeline Steps

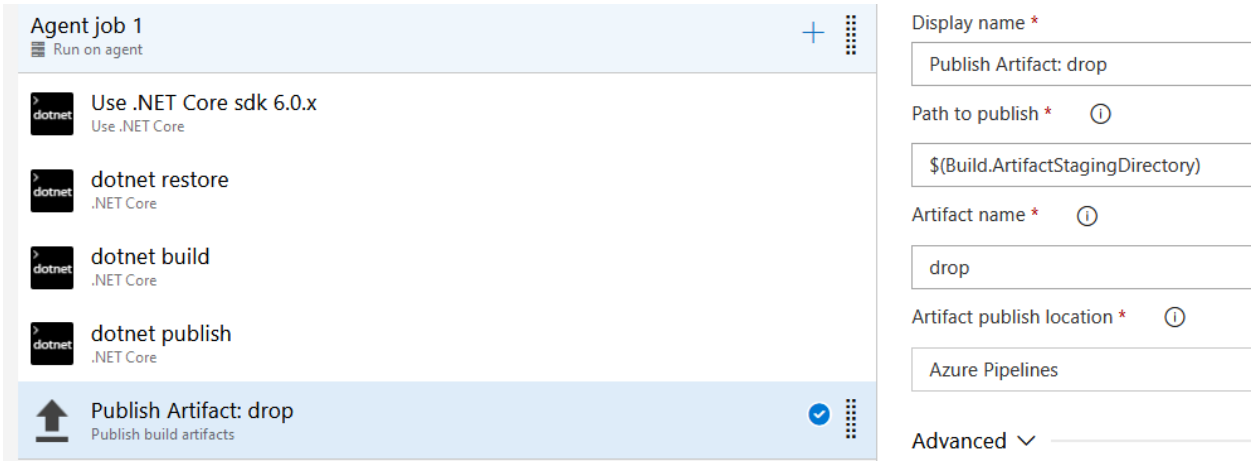





Agent job 1
Run on agent
Use .NET Core sdk 6.0.x
Use .NET Core
dotnet restore
.NET Core
dotnet build
.NET Core
dotnet publish
.NET Core
Publish Artifact: drop
Publish build artifacts
Display name *
dotnet build
Command *
build
Path to project(s)
functionapp
Arguments

Ste


Tasks
Variables
Triggers
Options
History
Save & queue
Discard
Summary
Queue
Pipeline
Build pipeline
Get sources
Function
main
Agent job 1
Run on agent
Use .NET Core sdk 6.0.x
Use .NET Core
dotnet restore
.NET Core
dotnet build
.NET Core
dotnet publish
.NET Core
Publish Artifact: drop
Publish build artifacts
.NET Core
Link settings
Task version
2.*
Display name *
dotnet publish
Command *
publish
Publish web projects
Path to project(s)
functionapp
Arguments
--configuration "$(BuildConfiguration)" --output "$(build.artifactstagingdirectory)"
Zip published projects
Add project's folder name to publish path


Agent job 1
Run on agent
Use .NET Core sdk 6.0.x
Use .NET Core
dotnet restore
.NET Core
dotnet build
.NET Core
dotnet publish
.NET Core
Publish Artifact: drop
Publish build artifacts
Display name *
Publish Artifact: drop
Path to publish *
$(Build.ArtifactStagingDirectory)
Artifact name *
drop
Artifact publish location *
Azure Pipelines
Advanced