# Practice M4: Azure for Developers

For this module we will need a PC running recent version of **Windows**, **Linux**, or **macOS**. It can be either physical or virtual machine

## Part 1: Container Instances

Navigate to <https://portal.azure.com>

Enter your credentials

### Working with Containers

#### Azure Portal

##### Resource group

Search for resource groups

Click on **+ Add** to create a new resource group

For **Resource group** set **RG-Containers**

Select the region to be **West Europe**

Click on **Review + create**

Then on **Create**

Once the deployment is done, click on the **Go to resource group** button

##### Container instance

Click on the **+ Add** button to create new resource

Type **Container instances** in the top searchfield and hit **Enter**

Once there, click on either **+ Create** or **Create container instances**

Ensure that the correct **Subscription** and **Resource group** are selected

Enter a unique (in the resource group) name for the **Container name.** For example, **aze-hello**

Change the **Image source** selection to **Docker Hub or other registry**

For **Image type** leave the default selection – **Public**

In the **image** text field, we must paste the path to the image. This could be a repository in any registry to which we have access. It can be either Docker Hub, or Microsoft’s container registry

For this exercise use just **shekeriev/aze-image-1** because the container image is in **Docker Hub** and is publicly available

Leave the **OS type** to **Linux**

You can click on **Change size** if you want to lower for example the memory

For now, leave it as it is

Click on **Next: Networking** button

In the **DNS name label** field enter a string that must be unique for the region. It will become part of the FQDN of the container. Enter for example **aze-hello**

Accept the default values for **Networking type** and **Ports**

Click on **Next: Advanced**

Accept all default values

Click on **Review + create** button

Click on **Create**

##### Interact with a container

Once the deployment is done, click on the **Go to resource** button

Being in the **Overview** section, we can copy either the **IP address** or the **FQDN** and paste it into a browser window

And voila, we can see that our container is working and reachable as expected

##### Inspect a container

Return to the **Overview** of the container instance in the portal

Now, we can see that the graphs for **CPU**, **Memory**, and others are showing some activity

Click on the **Containers** option under **Settings**

Here, we can see that there is one container in running state

Bellow the container list, we can see four different sections

First one, the **Events** section, is selected by default. It shows us, what stages the container went through

In **Properties** we can see some more details about our container

**Logs** are showing us the HTTP GET requests generated when we accessed the application vie the browser

Click on **Connect** to establish an interactive session to the container

Select **/bin/sh** and click **Connect**

Now, we are in the container. Here we can issue some commands. For example, let’s try **ls**, **hostname**, **uname**, etc

Once, we are done exploring, we can execute the **exit** command to close the connection

##### Remove a container

Return to the **Overview** section

Click on the **Delete** button

Confirm with **Yes**

#### Azure CLI

Let’s continue with our experiments in the existing resource group

If using a local shell, login first by issuing:

**az login**

##### Container instance

In order to start a container like the one we created in the previous part we must execute the following:

**az container create --resource-group RG-Containers --name aze-hello --image shekeriev/aze-image-1 --dns-name-label aze-hello --ports 80**

##### Interact with a container

Once, our container is created, we can ask for details like the public IP address, the FQDN, etc.

**az container show --resource-group RG-Containers --name aze-hello**

We can always narrow down the received information and change the output style with:

**az container show --resource-group RG-Containers --name aze-hello --query "{FQDN:ipAddress.fqdn,IP:ipAddress.ip}" --output table**

Note, that depending on the shell in use, you may need to enclose the query argument in double quotes

We can copy either the **IP address** or the **FQDN** and paste it into a browser window

And voila, we can see that our container is working and reachable as expected

##### Inspect a container

Let’s get container logs. For this to happen, we must execute:

**az container logs --resource-group RG-Containers --name aze-hello**

We can attach to the container’s output streams and monitor what is happening in real-time:

**az container attach --resource-group RG-Containers --name aze-hello**

If we return to the browser window and refresh a few times, we will notice that events are appearing immediately on the stream

Let’s detach with **Ctrl+C**

When asked, confirm with **y**

Now, let’s execute a command, for example to see the hostname of the container:

**az container exec --resource-group RG-Containers --name aze-hello --exec-command hostname**

In a similar fashion, we can start a shell and connect to it:

**az container exec --resource-group RG-Containers --name aze-hello --exec-command /bin/sh**

Now, we are in the container. Here we can issue some commands. For example, let’s try **ls**, **hostname**, **uname**, etc

Once, we are done exploring, we can execute the **exit** command to close the connection

##### Remove a container

Let’s delete the container with:

**az container delete --resource-group RG-Containers --name aze-hello**

Confirm with **y**

#### Azure PowerShell

Let’s continue with our experiments in the existing resource group

If using a local shell, login first by issuing:

**Connect-AzAccount**

##### Container instance

Let’s start a container from the same image as we did in the previous part:

**New-AzContainerGroup -ResourceGroupName RG-Containers -Name aze-hello -Image shekeriev/aze-image-1 -DnsNameLabel aze-hello -Port 80**

##### Interact with a container

Now, that we have a running container, we can ask for detailed information with:

**Get-AzContainerGroup -ResourceGroupName RG-Containers -Name aze-hello**

Of course, we can narrow-down the received information with:

**Get-AzContainerGroup -ResourceGroupName RG-Containers -Name aze-hello | Select IpAddress, Fqdn**

We can copy either the **IP address** or the **FQDN** and paste it into a browser window

And voila, we can see that our container is working and reachable as expected

##### Inspect a container

Should we want to get the logs, we can execute:

**Get-AzContainerInstanceLog -ResourceGroupName RG-Containers -Name aze-hello -ContainerGroupName aze-hello**

We can add the **-Tail** option to get only the last **X** lines. For example **-Tail 5** to get the **last 5 lines**

##### Remove a container

Let’s delete the container with:

**Remove-AzContainerGroup -ResourceGroupName RG-Containers -Name aze-hello**

### Container Images

For this part we will need an access to a Docker server, either installed locally (for example Docker Desktop), or remotely

Additionally, we will need a Docker CLI to interact with the Docker server

For the tasks that follow, we will assume, that we have Docker Desktop installed on our host machine

#### Prepare the project

Download and extract the accompanying file **docker-image-2.zip**

Let’s assume that you extracted it in a folder **C:\TMP\docker-image-2**

If you examine the contents, you will see that there is one file named **Dockerfile**, a file named **file.png** and a folder **web** with one HTML file and one PNG file

Examine the **Dockerfile**

Copy or move **file.png** to the folder **web**

Go to the **web** folder and open the **index.html** file in a text editor

Create an empty row on **line 10** and paste this block of code:

**<br />**

**<h1>And this is a cat ;)</h1>**

**<img src="file.png" />**  
Save and close the file

There is no need to modify the **Dockerfile** as it will copy the whole **web** folder including the new file to the image

#### Create a container image

Open a terminal session and make sure that you are in the folder that contains the **Dockerfile** from the previous task

Now, build the new image with:

**docker build . -t aze-image-2**

List the available images

**docker images**

Run the app locally:

**docker run -d -p 8080:80 aze-image-2**

Open a browser window and navigate to <http://localhost:8080> to test the app

#### Publish a container image

We can publish our image to any Docker registry to which we have the appropriate access. For example, this could be the **Docker Hub**, or our own **Azure Container Registry**

Let’s go with the second option

##### Azure Portal: Create Azure Container Registry

Go to the resource group created earlier (RG-Containers)

Click on the **+ Add** button

Search for **Container Registry** in the **Marketplace** search box

Click on **Create**

For **Registry name** enter **azecr2021** (or something else that is not taken \*)

Set the **Location** to **West Europe**

Change the **SKU** to **Basic**

Click on **Review + create** and then on **Create**

After the registry is created, navigate to its **Overview** mode

There, click on **Access keys** in the **Settings** section

Switch the **Admin user** option to **Enable**

\* *Note that in case you used a different name for the registry then you should adjust the commands that follow*

##### Azure CLI: Create Azure Container Registry

A container registry can be created with other means as well. For example, we can create it with **Azure CLI**

We need a single command to accomplish this task:

**az acr create --resource-group RG-Containers --name azecr2021 --sku Basic --admin-enabled true**

##### Publish the image

No matter how we did create the registry, let’s push the image

Return on the console session used earlier to build the container image

Tag the image against the new registry:

**docker tag aze-image-2 azecr2021.azurecr.io/aze-image-2:v1**

Next, log in to the registry:

**az acr login --name azecr2021**

Then push the image to the registry

**docker push azecr2021.azurecr.io/aze-image-2:v1**

#### List repositories in a registry

Now, we can list all repositories available in our registry

If you prefer the portal way of doing things, you can navigate to the **azecr2020** registry

Then click on **Repositories** under **Services**

There you will see our repository **aze-image-2**

If you click on it, you will see how many different tags (or versions) are available

If you prefer the command line way, you can execute:

**az acr repository list --resource-group RG-Containers --name azecr2021 --output table**

Besides the wanted output you will notice a deprecation warning about the **resource-group**. The command can be:

**az acr repository list --name azecr2021 --output table**

This should return the list of repositories. In order to see how many tags are available in a repository, execute:

**az acr repository show-tags --resource-group RG-Containers --name azecr2021 --repository aze-image-2 --output table**

The same applies here and the new version of the command should look like:

**az acr repository show-tags --name azecr2021 --repository aze-image-2 --output table**

#### Deploy the new application

Knowing at least three ways to deploy a container application, choose the one you like most and use it to deploy the application

However, this time there is a slight complication. Our image is in a private registry, so we must provide credentials

##### Azure Portal

Here, the main difference is that during the creation process, you must select **Azure Container Registry** for **Image Source**

Then in **Registry** drop-down list, select the registry created earlier

Make sure that the appropriate **Image** and **Image tag** are selected

The rest of the steps are the same as before

##### Azure CLI

Extended version of the container creation command would look like:

**az container create --resource-group RG-Containers --name aze-hello --image azecr2021.azurecr.io/aze-image-2:v1 --cpu 1 --memory 1 --registry-login-server azecr2021.azurecr.io --registry-username azecr2021 --registry-password "Rj+92PBqfat2d9mbHhGQCHpJK2PN79d7" --dns-name-label aze-hello --ports 80**

If you wonder where we took the registry password from, remember that when we enabled the admin user, there were two passwords (or access keys) generated

## Part 2: Azure App Services

In this part we will create a simple **HTML** web application deployed in two ways – via the command line and portal

Next, we will create a little bit more “sophisticated” **PHP** application that interacts with an **Azure SQL** database

Let’s navigate to <https://portal.azure.com>

Enter your credentials

### Web Apps

Navigate to **resource groups** section

Create a new resource group, for example **RG-WebApps** in the **West Europe** region

Enter the resource group

#### Azure Portal: HTML web app

##### Create a Web App

Click on the **+ Add** button to add new resource

In the search bar enter **App Services** and hit **Enter**

Click either on the **+ Create** or the **Create app service** button to create a new app service

Make sure that the **Subscription** and the **Resource Group** are correctly set

For **Name** of the instance enter **azewebapp**

Be sure to select **Code** in the **Publish** section

For the current task, the **Runtime stack** is of no importance. Let’s select **PHP**

Set the **Operating System** to **Linux**

Select **West Europe** for the **Region**

Click **Create new** link under the drop-down list in the **App Service Plan** section

In the text field enter a name you like, for example **Linux-WebApp-Plan**

Change the **Sku and size** to **Free F1**

Click on **Review + create**

Click on **Create**

##### Deploy the project

Extract the accompanying archive file **web-app-html.zip** to a folder of your choice

Once the web app is ready, click on the **Go to resource**

Click on the **Deployment Center** option under **Deployment**

Select **FTPS Credentials** tab

We will need an FTP application

**FileZilla** as well as any other FTP application will do the job

Copy the **FTPS Endpoint** string and paste it in the **Host** field of the FTP application

Use the provided **Username** and **Password**

Click on the **Quickconnect** button

Accept the certificate

Navigate to the extracted files and copy them to the destination folder in the right part of the screen

Close the FTP session and application

Return to the **Azure Portal**

Go to the **Overview** section

Copy the value for **URL** and paste in a new browser window or click over it to open a new window

You should see a familiar web page

##### Change and re-deploy a project

Navigate back to the folder where the files for the page are stored

Open the **index.html** file with a text editor and modify it. For example, add or remove text

Once you are done with the manipulation save and close the file

Open again the FTP application and copy the URL, username, and the password from the **FTPS Credentials** option under the **Deployment Center**

Upload the new version of the files

Close the FTP application

Return to the **Overview** section of the web application in the **Azure Portal** and click on the URL

Now you should see the new version of the page

#### Azure CLI: HTML web app

Let’s repeat a similar deployment, but this time on the console

If using a local shell, login first by issuing:

**az login**

##### Deploy a HTML web app

As static HTML web apps are hosted only on Windows app service plan, we must create a new plan

Unfortunately, we must create it in a separate resource group as the two (Linux and Windows) app service plans cannot co-exist in one group

**az group create --name RG-WebApps-Win --location westeurope**

Now, navigate to the folder where you extracted the accompanying archive file **web-app-html.zip**

Execute the following command to deploy the web application:

**az webapp up --resource-group RG-WebApps-Win --location westeurope --name azewebapp1 --html --plan Windows-WebApp-Plan --sku F1**

Note the **app\_url** parameter and use the value to test the application

If by any chance you omit the URL, you can always use the following command to list all web applications with their parameters:

**az webapp list --resource-group RG-WebApps-Win**

Or to narrow down the results:

**az webapp list --resource-group RG-WebApps-Win --query "[].{Name:name,URL:defaultHostName}" --output table**

Please note that depending on the shell, you may need to use double quotes around the query value

##### Change and re-deploy a project

Change the **index.html** file

Execute the same command that you used for the initial deployment to redeploy the app:

**az webapp up --resource-group RG-WebApps-Win --location westeurope --name azewebapp1 --html**

You can navigate to the browse window and check if the changes got reflected

#### Azure Portal: PHP + SQL web app

Let’s first extract the contents of the **web-app-php.zip** file to a folder of our choice

Now, we should prepare the database server and the database

##### Preparation

Let’s navigate to <https://portal.azure.com>

Create a new resource group. For example, the **RG-WebApps-DB** group

Enter the resource group

Click on the **+ Add** button and search for **Azure SQL**

Click on **+ Create**

Then click on **Create** in the **SQL databases** tile

Ensure that the **Subscription** and the **Resource group** are filled correctly

For **Database name** enter **azedb**

Click on **Create new** in the **Server** section

In the **Server name** filed enter **azedbsrv**

Enter **demosa** for the **Server admin login**

Use a password that conforms to the rules, for example **DemoPassword-2021**

Set the location to **West Europe**

Click on **OK**

Click on **Configure database**

Select **Basic** plan and click **Apply**

Click on **Next: Networking** to configure the connectivity to the database

Under **Connectivity method** select **Public endpoint**

For the **Allow Azure services and resources to access this server** select **Yes**

Do the same for **Add current client IP address**

Click on **Review + create**

Click on **Create**

##### Load the data

Once the deployment is done, click on **Go to resource**

Click on the **Query editor (preview)**

Enter the login information and click on the **OK** button

Paste the contents of the file **cities-database.sql** in the query panel

Click on the **Run** button

You can check if the data was loaded fine

##### Configure the application

Navigate to the **Connection strings** under the **Settings** section

Switch to the **PHP** tab

Copy the information related to the **SQL Server Extension**

Open the **index.php** file

Paste the copied information after the **// CONNECTION INFORMATION BELLOW**

Adjust the password

Save and close the file

##### Create the web application

Go to the resource group (**RG-WebApps-DB**)

Click on the **+ Add** button to add new resource

In the search bar enter **App Services** and hit **Enter**

Click either on the **+ Create** or the **Create app service** button to create a new app service

Make sure that the **Subscription** and the **Resource Group** are correctly set

For **Name** of the instance enter **azewebapp2**

Be sure to select **Code** in the **Publish** section

For the current task set the **Runtime stack** to **PHP 7.4**

Set the **Operating System** to **Linux**

Select **West Europe** for the **Region**

Select the existing Linux plan (we can have only one free plan per region)

Click on **Review + create**

Click on **Create**

##### Deploy the project

Once the web app is ready, click on the **Go to resource**

Click on the **Deployment Center** option under **Deployment**

Switch to **FTPS Credentials**

Again, we will need an FTP application

**FileZilla** as well as any other FTP application will do the job

Copy the **FTPS Endpoint** string and paste it in the **Host** field of the FTP application

Use the provided **Username** and **Password**

Click on the **Quickconnect** button

Accept the certificate if prompted to do so

Navigate to the extracted files and copy them to the destination folder in the right part of the screen

Close the FTP session and application

Return to the **Azure Portal**

Go to the **Overview** section

Copy the value for **URL** and paste in a new browser window or click over it to open a new window

You should see a web page showing information about the top 10 cities by population in Bulgaria

## Part 3: Azure Functions

In this part we will create a function app with two functions. Then will extend one of the functions to store messages in a queue. After this, we will experiment with logic apps. At first, we will create one stand-alone application as a proof of concept. Then we will create a second one, that works with the same queue used by the function app

Let’s navigate to <https://portal.azure.com>

Enter your credentials

### Azure Functions

Navigate to **resource groups** section

Create a new resource group, for example **RG-Functions** in the **West Europe** region

Enter the resource group

#### Create a Function App

Click on the **+ Add** button

Search for **Function App** in the main search bar and hit **Enter**

Click on the **+ Create** button

Ensure that the **Subscription** and **Resource Group** are correctly filled in

Enter **azefuncapp** for **Function App name**

Make sure that **Code** is selected in the **Publish** option

Select **.NET** for **Runtime stack**

Change the **Region** to **West Europe**

Click on **Next: Hosting**

Move forward by clicking on **Next: Monitoring**

Click on **Review + create**

Click on **Create**

Once the deployment is done, click on the **Go to resource** button

#### Create a time triggered function

Once in the **Overview** mode of the function application, click on **Functions** option in the **Functions** section

Click on the **+ Add** button

Select **Timer trigger** template

Let’s change the schedule to **0 \*/1 \* \* \* \***

This will cause the function to be executed every minute instead of every five minutes

Click on **Add** button

After the function is created, you will be brought in its **Overview** section

Explore the options under the **Developer** section

Return to the function by clicking on its name

When in the **Monitoring** section, switch to the **Logs** view

After a while you will see some log messages proving that the function is working as expected

#### Create a HTTP triggered function

Return in the **Overview** mode of the function app

Select **Functions** and click on the **+ Add** button

Select **HTTP trigger** and click on the **Add** button

Once the function is created, click on **Code + Test** to examine the code

Then, click on the **Get function URL** button

Click on **Copy** to copy it to the clipboard

Open a web browser, paste the URL and hit **Enter**

You will see a message that we must pass a name either via a query string value or in the request body

Let’s pass it as a query string

Add **&name=Demo** at the end of the URL

Now the result should become **Hello, Demo. This HTTP triggered function executed successfully.**

Alternatively, we may use an application like **Postman** to test for example sending a **body** instead of URL parameter

#### Modify the app to use a Queue

Return to the function window and switch to the **Integration** mode

Then click on **+ Add output**

Explore the options. Select the **Azure Queue Storage** in the **Binding Type** drop-down list

Copy the **Message parameter name**, which in our case should be **outputQueueItem**

We can select a new storage account or accept the default proposal

Confirm by clicking on the **OK** button

Return to the function’s code by clicking on its name

Modify the signature to Run(HttpRequest req, **ICollector<string> outputQueueItem**, ILogger log)

Just before the **return** statement add this one:

outputQueueItem.Add("Name passed to the function: " + name);

Click on **Save**

Let’s test the function again, click on the **Get function URL** button

Click on **Copy** to copy it to the clipboard

Open a web browser, paste the URL, add **&name=Demo** at the end of the URL, and hit the **Enter** key

The result should not differ from the earlier executions

Let’s go and check the queue

Navigate to the storage account you specified earlier (or if used the proposed, navigate to the one in the resource group)

Then, go to the queues and enter the queue specified during the modification, it should be **outqueue**

There you should see the message

You can execute the function a few more times and see what happens

#### Functions with SQL backend

Let’s create a new **HTTP trigger** function but this time it will interact with a database

We will reuse the database server created earlier and will just add a new database with some data

##### Create SQL server and database

Return to the resource group that contains the database (for example **RG-WebApps-DB**)

Click on the **+ Add** button

In the search field enter **SQL Database** and hit **Enter**

Click on **Create**

Check the **Subscription** and **Resource group** values

For **Database name** enter **dbfunc**

In **Server** field make sure that the existing **SQL Sever** is selected

Click on **Configure database** link under **Compute + storage** to change the default value

Follow the **Looking for basic, standard, premium?** link in the top-left section

Select the **Basic** option

Click **Apply** to accept the proposed **5 DTU** size

Click on **Review + create**

Click on **Create**

##### Load sample data

Navigate to the database

Go to **Query editor (preview)** option

Enter the appropriate credentials and click **OK**

Paste the contents of the **cities-database-ext.sql** file to the **Query 1** window

Click on the **Run** button

You can check if the data was loaded fine

##### Create a HTTP triggered function

Return to the **Function App** created earlier

Select **Functions** and click on the **+ Add** button

Then, select the **HTTP trigger** option and click on the **Create Function** button

Once the function is created, examine the code

Open the file **function-with-db.txt** and copy the text

Return to the function body and substitute the code with what you copied from the file

Click on **Save**

Navigate to the **SQL Database**

Click on **Connection strings** under **Settings**

Copy the code for **ADO.NET**

Return to the function

Navigate to **line 22** and paste what you have copied as a value for the **str** variable

Substitute the string **{your\_password}** with the actual password for the **SQL Server** you set earlier, for example **DemoPassword-2021**

Click on the **Save** button

Check in the **Logs** for a string saying **Compilation succeeded**

Then, click on the **Get function URL** button

Click on **Copy** to copy it to the clipboard

Open a web browser, paste the URL and hit **Enter**

You will see a message that we must pass a city code either via a query string value or in the request body

Let’s pass it as a query string

Add **&citycode=BG03** at the end of the URL

Now the result should show some information about **Varna**

If the page remains blank, then go to the **Logs** and check

It you see a message that the sni.dll can not be loaded, then go to the function app and lower the runtime version

Navigate to **Configuration** in the **Settings** section

Switch to **Function runtime settings**

Set the **Runtime version** to **~2**

Click the **Save** button and then on **Continue**

Retry the last test. Now it should be successful

Let’s try it with city code **BG99**

Now, we should see a message that nothing has been found for this code

Our application is working as expected

### Azure Logic Apps

Return to the resource group created earlier (**RG-Functions**) or create a new one

#### Create Logic App #1

Once in the resource group, click on the **+ Add** button

Then search for **Logic apps** in the main search bar and hit the **Enter** key

Click on the **+ Add > Consumption** button

For **Name** enter **azelogic1**

Ensure that the **Subscription** and **Resource Group** are filled correctly

Click **Review + create** and then on **Create**

Once the deployment is done, click on **Go to resource**

Click on **Recurrence** tile in the **Start with a common trigger**

Change the interval to **1 minute**

Click on the **+ New step** button

Type **send email** in the **Search connectors and actions** field

You will see that there are many platforms supported. Depending on where you have account, make the appropriate selection. I will choose **Send an email** from **Outlook.com**

Click on **Sign in**

Select a profile and/or enter the required details

Once the initialization is done, let’s fill in some details for **To**, **Subject**, and **Body**

Click on **Save**

Return to the **Overview** section of our logic application

After a while you will see a notification

Look after the **Runs last 24 hours** field or the **Runs history** table

You can go to the mailbox and check for any new messages

Once, we are aware that the application is working, we can click on **Disable** to stop it

#### Create Logic App #2

Now, let’s create a second logic app

This time it will interact with our Function app from the previous part. It will look for messages in a Queue and if any found, it will notify us via email and then delete the queued message

Return to the resource group

Once in the resource group, click on the **+ Add** button

Then search for **Logic apps** in the main search bar and hit the **Enter** key

Click on the **+ Add > Consumption** button

For **Name** enter **azelogic2**

Ensure that the **Subscription** and **Resource Group** are filled correctly

Click **Review + create** and then on **Create**

Once the deployment is done, click on **Go to resource**

Click on the **Blank Logic App** tile

Start typing **Azure Queues** in the search box

Below the box **Triggers** will start to appear. Click on the **When there are messages in a queue**

In the **Queue Name** select the queue. In our case it should be **outqueue**

Change the interval to **1 Minute**

Click on the **+ New step** button

Select the same mail provider used during the previous exercise

Then select the **Send an email** action

Fill the **To** section

In the **Subject** section enter **New message:**

And select **Message ID** from the pop up

In the **Body** section enter **Message says:**

And then select **Message Text** from the pop up

Click on **New step**

Select **Azure Queues**

In the actions list select **Delete message**

In the **Queue Name** select **outqueue**

Click in the **Message ID** field and from the pop up select the **Message ID** component

Click in the **Pop Receipt** field and from the pop up select the **Pop Receipt** component

Click on the **Save** button

Return to the **Overview**

After a while you should see that the application is working

Go to the mailbox and check for any new messages

Navigate to the queue and check if there are any messages left

Test the function app (first HTTP triggered function app) with a different parameter and check again the queue

Check again the mailbox. Everything seems to be working just fine

# A Reminder

Do not forget to stop and remove the resources created during this practice