# 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 searchfield and hit **Enter**

Once there, click on either **+ Add** 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**

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

In the **image name** 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. 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**

Click on **Next: Advanced**

Accept all default values

Click on **Review + create** button

Click on **Create**

##### Interact with 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 – **Events** 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 **ps**, **top**, **ls**, 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**

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**

Now, let’s execute a command, for example to see the list of processes:

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

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 **ps**, **top**, **ls**, 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**

#### 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

#### 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**

Edit the **index.html** file in the web folder and at the end add this:

<br />

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

<img src="file.png" />  
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 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

Click on the **+ Add** button

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

Click on **Create**

For **Registry name** enter **azecr2019**

**Enable** the **Admin user**

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

Click on **Create**

##### 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 azecr2019 --sku Basic --admin-enabled true**

##### Publish the image

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

Tag the image against the new registry:

**docker tag aze-image-2 azecr2019.azurecr.io/aze-image-2:v1**

Next, login the registry:

**az acr login --name azecr2019**

Then push the image to the registry

**docker push azecr2019.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 **azecr2019** 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 azecr2019 --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 azecr2019 --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. This time there is a slight complication, our image is in a private registry, so we must provide credentials

##### Azure Portal

Here the only difference is that during the creation process, you must select **Private** for **Image Type**

Of course, you must fill in the authentication details

The **Image registry password** can be taken from the **Access keys** section of the **Container Registry**

##### Azure CLI

Extended version of the container creation command would look like:

**az container create --resource-group RG-Containers --name aze-hello --image azecr2019.azurecr.io/aze-image-2:v1 --cpu 1 --memory 1 --registry-login-server azecr2019.azurecr.io --registry-username azecr2019 --registry-password "cSvxk50+Y7OLs4SXNqXZ2Xhaufvlhqys" --dns-name-label aze-hello --ports 80**

## 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

##### Resource group

Navigate to **resource groups** section

Create 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 **+ Add** 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**

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 **FTP**

Click on the **Dashboard** button

Open an FTP application. FileZilla will do the job just fine

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 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 **FTP** 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

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 --location westeurope --name azewebapp1**

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**

Or to narrow down the results:

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

##### 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 --location westeurope --name azewebapp1**

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

Go to the resource group created earlier

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

Click on **+ Add**

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-2019**

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.txt** 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 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

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

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

Click either on the **+ Add** 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 the **Runtime stack** is of no importance. Let’s select **PHP**

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

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

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

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**

Select **FTP**

Click on the **Dashboard** button

Open an FTP application. FileZilla will do the job just fine

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 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

##### Resource group

Navigate to **resource groups** section

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

Enter the resource group

#### Create Function App

Click on the **+ Add** button

Search for **Function App** and hit **Enter**

Click on the **+ Add** button

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

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

Select **.NET Core** 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 time triggered function

Select **azefunc** function application

Click on the **+ New Function** button

Click on **In-portal** and then on **Continue**

Select **Timer** and then on **Create**

Our function is immediately available

As you can see, it is a very simple one – it just logs a message periodically

Now, click on **Integrate** just below the function name

Here we can adjust the function. We can work in a graphical manner or switch to a code editor by clicking on **Advanced editor**. We can switch back by clicking on the **Standard editor** link

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 **Save**

Return to the function by clicking on its name

Click on the **Logs** item below the function’s code

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

#### Create HTTP triggered function

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

Select **Functions** and click on the **+ New function** button

Click on the **go to quickstart**

Select the **In-portal** tile and click on the **Continue** button

Select **Webhook + API** and click on the **Create** button

Once the function is created, 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**

#### Modify the app to use a Queue

Return to the function window and click on **Integrate**

Then click on **New Output**

Explore the options. Select the **Azure Queue Storage**

Scroll down and click on **Select**

You will see a notification saying that the extension is not installed. Click on the **Install** link

After a while the installation process will finish

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

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

Click on **Save**

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

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

There you should see the message

You can execute the function few more times

### Azure Logic Apps

##### Resource group

Return to the resource group created earlier or create a new one

#### Create Logic App #1

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

Then search for **Log Apps** and hit the **Enter** key

Click on the **+ Add** button

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

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

Click **Create**

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

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

Change the interval to **1 minute**

Click one **New step**

Type **send email**

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

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 finds any, it will notify us via email and then delete the queue message

Return to the resource group

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

Then search for **Log Apps** and hit the **Enter** key

Click on the **+ Add** button

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

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

Click **Create**

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

Click on **Blank Logic App**

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**

Enter a **Connection name**, for example **Demo**

Select the storage account used earlier with the function application

Click on the **Create** button

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

Change the interval to **1 Minute**

Click on **New step**

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 with a different parameter and check again the queue

Check again the mailbox

Everything seems to be working just fine