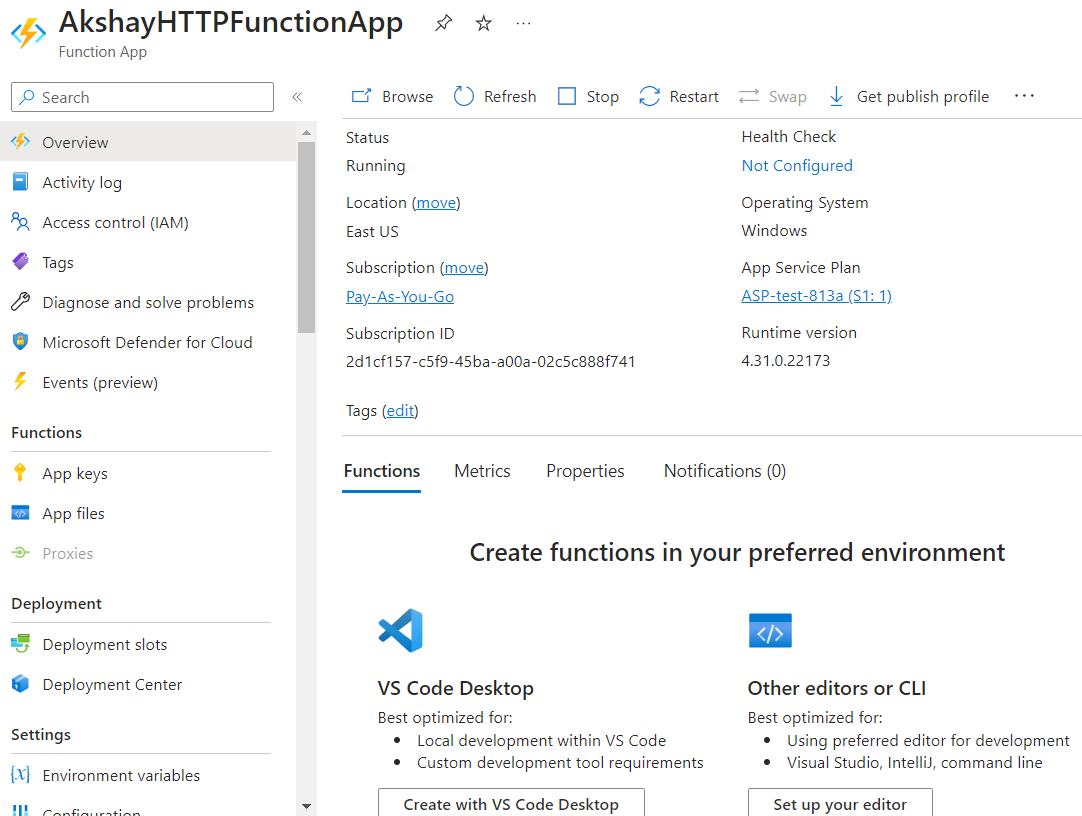
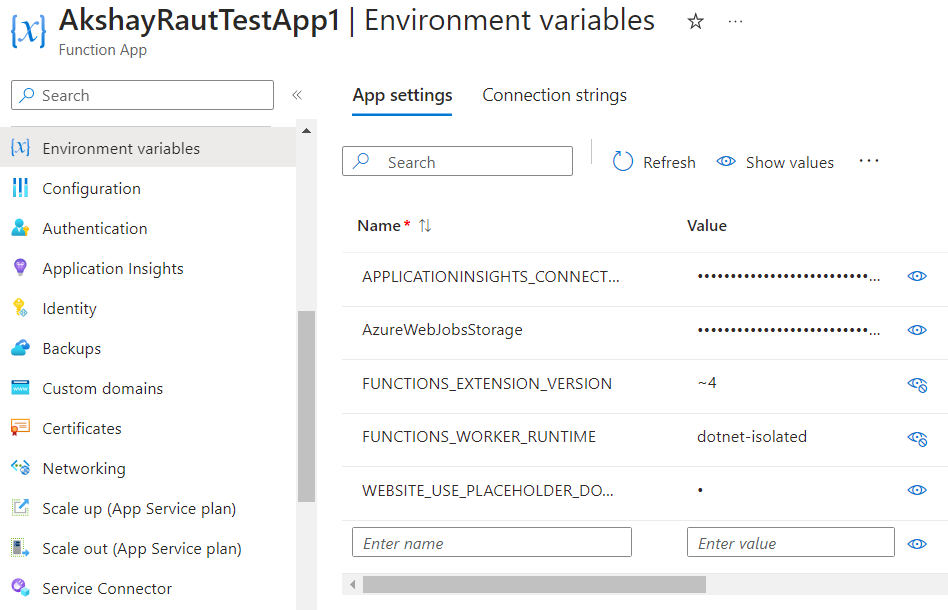
AZ 204 – Develop Solutions using Azure services

# Azure Compute services

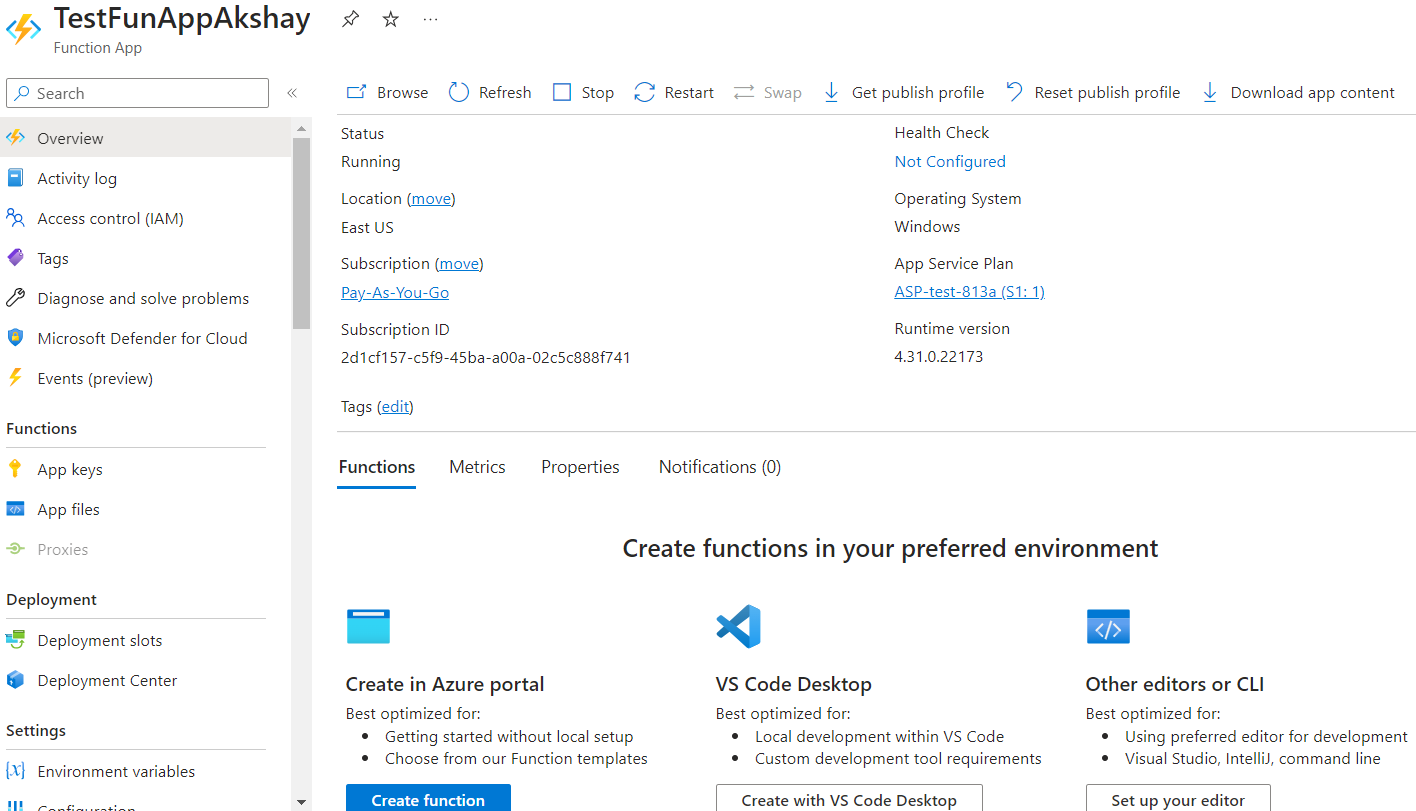
## Azure Functions



If you cannot see the option “Create in Azure Portal” in “Functions” tab in “Overview” then you need to change the Function Worker Runtime from “dotnet-isolated” to “dotnet”. This option is generally available for other tech stack.



After saving this change, you should see the “Create in Azure Portal”:



When you deploy your Function from Visual Studio or VS Code, it will replace existing Function that exists in your Azure Function app.

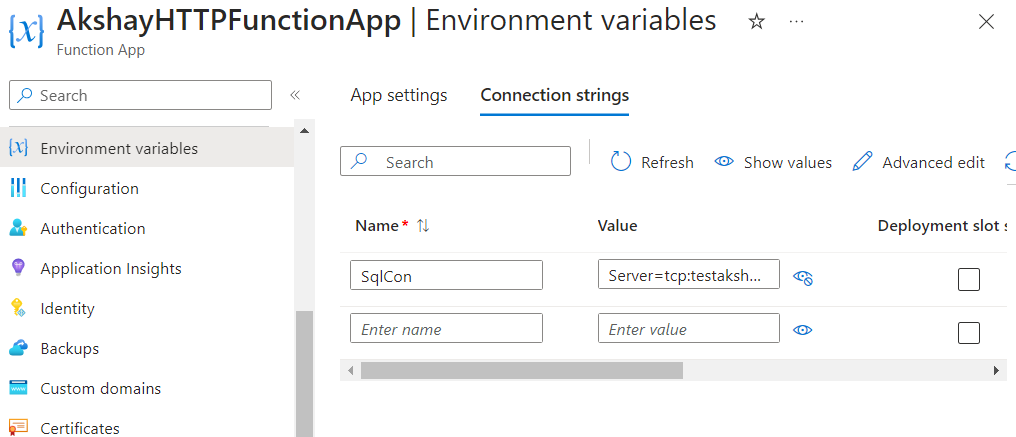
You cannot inject any additional parameters as dependency such as IConfiguration.

You can store configuration settings in “Environment Variables” page of Function App.

E.g. If you store sql connection string like this:

Key name: “SqlCon”

Value: <connection string>



Then you cannot use IConfiguration to get the connection string value.

Instead you need to use Environment class.

Environment.GetEnvironmentVariable("SQLAZURECONNSTR\_SqlCon");

Other supported prefixes can be found here for different types of SQL providers.

<https://learn.microsoft.com/en-us/aspnet/core/fundamentals/configuration/?view=aspnetcore-8.0&tabs=basicconfiguration#connection-string-prefixes>

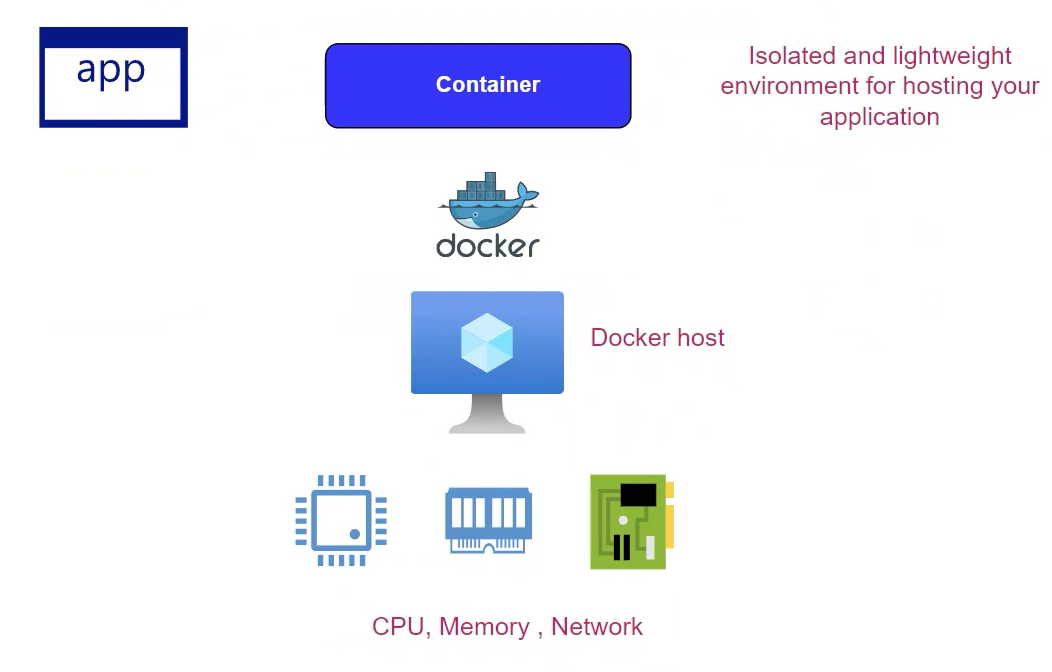
Following languages are supported by Azure Function app.

<https://learn.microsoft.com/en-us/azure/azure-functions/supported-languages?tabs=isolated-process%2Cv4&pivots=programming-language-csharp#language-support-details>

## Azure Containers

Docker

It’s an open platform that is used for developing, shipping, and running applications. Docker has the ability to package and run an application in a loosely isolated environment called a container.



You first need to install Docker runtime. Docker runtime provides the environment to Docker container to interact with underlying machine.

When you install Docker runtime on a machine, it is called as Docker host. Then you can have Docker containers run on a Docker host.



The next thing you will create is a Docker image.

This is a read-only template with instructions that are required to create the Docker container. Container is a runnable instance of an image.

Steps to setup Docker on Linux VM.

1. Create a Linux VM on Azure.
2. Use PuTTY to connect to your VM.
3. Follow the installation steps for Docker. <https://docs.docker.com/engine/install/ubuntu/>

Next we will try to run nginx server on this Linux VM via Docker.

We need to run docker command to load the nginx image via Docker Hub which is a repository.

We can run this command to see the images available in docker in current machine.

**sudo** docker images

Visit this website <https://hub.docker.com/> to search for image for nginx.

From the nginx details page, you can take this command to run in the VM:

docker pull nginx

OR we can directly spin an instance of nginx container by this command:

**sudo** docker run **--**name appnginx -p 80**:**1234 **-**d nginx

With this command, your are asking Docker to pull the nginx image from Docker hub into the VM, create a container using that image, and then run it.

-p stands for Publishing ports thus giving access to the Docker container to the Docker host i.e. our VM.

In this command, we have mapped the port 80 of Linux VM to the port 1234 of the Docker container.

You can read more about Publishing ports here: <https://docs.docker.com/network/#published-ports>

This will first check if the nginx image is available locally within VM otherwise pull (download) it from Docker Hub.

After this command is successful, you can run “sudo docker images” to find nginx locally available.

Run this command to see the running containers

**sudo** docker ps

You should see your nginx container running on port 1234, which is accessible from port 80 of your Linux VM. You can go to the Overview of your VM on Azure, copy the IP address, and browse to it in a new tab. But the browser will not be able to connect to the nginx server because we have not enabled port 80 publicly on VM.

### How to containerize a .NET application.

We could generate image of our .NET application from our own computer but in this exercise, we will create image directly within Linux VM.

Create a Publish profile for your .NET application using Visual Studio that publishes to a folder location in your computer, then publish it.

Copy the published folder to Linux VM using WinSCP.

Create a Dockerfile using VS Code and add instructions.

We have to tell Docker what should be the base image in Docker hub that will be used to run my .net application. We need to tell Docker to how to build that image that can have ability to run our application.

.Net or Java apps need their runtime to run the application. That runtime needs to be present within our container.

There are images already present for this.

FROM mcr.microsoft.com/dotnet/aspnet:6.0

WORKDIR /app

COPY . .

EXPOSE 80

ENTRYPOINT ["dotnet", "AZ204AzureWebApp"]

#### FROM command

It pulls an image.

“mcr.microsoft.com/dotnet/aspnet” is the registry location where aspnet images are kept and 6.0 is the version of runtime, hence version of the image that we need to pull.

#### WORKDIR

/app is the working directory for building the image.

#### COPY

It will copy everything from the Published folder on Linux VM to this image.

The first parameter of COPY command is the source path and second is the destination path.

. as first parameter means it will copy everything from source directory.

. as second parameter means current working directory inside the container i.e. /app directory in container. If WORKDIR is not specified before this COPY instruction, the default working directory depends on the base image specified by the FROM instruction. For many Linux-based images, this would typically be /.

#### Absolute paths:

You can specify an absolute path starting with a **/** to copy files to a specific location in the container's filesystem. For example:

COPY . /app/

#### Relative paths:

If you specify a relative path (without a leading **/**), it will be relative to the **WORKDIR** directory. For instance, if you set **WORKDIR /app** earlier in your Dockerfile, then:

COPY . mydir/

#### Paths with environment variables:

Some Dockerfiles use environment variables (set with **ENV**) in the destination path. For example:

ENV APP\_HOME=/home/myapp

COPY . $APP\_HOME/

Remember, if the destination directory does not exist, Docker will automatically create it (including any necessary parent directories) as part of the COPY operation. Also, when specifying paths, be mindful of permissions and ownership, especially if your application requires specific users or groups to access the copied files.

#### EXPOSE

It will expose specified port.

#### ENTRYPOINT

With this command, we can tell Docker to run dotnet program and which app to run with it.

We will be using this Dockerfile from Linux VM, so we will have to transfer it via WinSCP to the Published folder.

Go to the PuTTY terminal and use cd to navigate to your published folder.

Now we can run docker build command to create the image for our app using contents inside published folder.

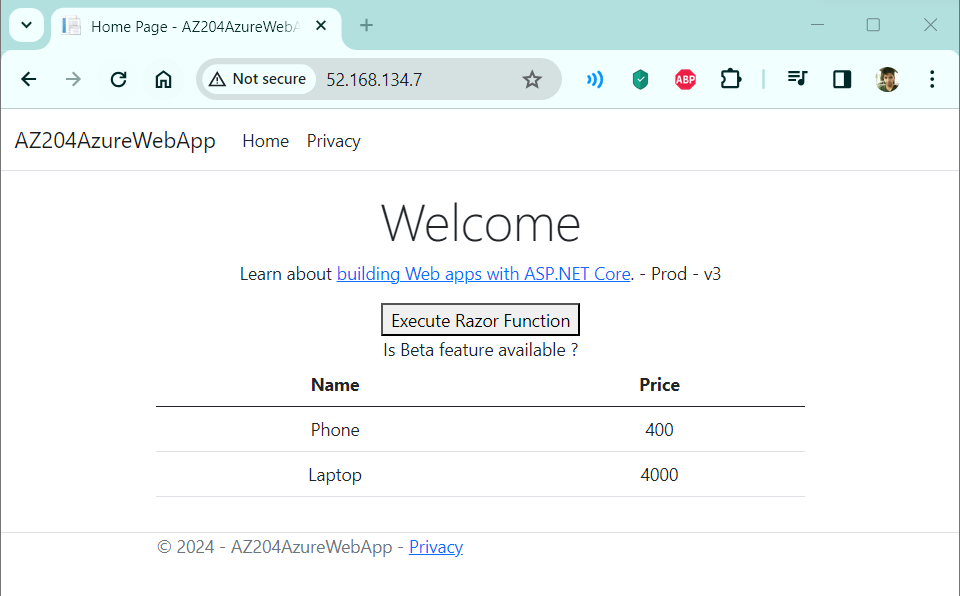
**sudo** docker build -t az204app **.**

Note: After this command, when you run command “sudo docker images”, you might expect to see the asp.net core image as well however, it doesn’t but your app can still run.

and next we run the container using this image:

**sudo** docker run **--**name az204dockapp -p 80**:**80 -d az204app

Output:



In case you encounter issue of containers not showing up, you can try to delete the image being used by the container and it will throw an error saying a container with <id> is using that image.

Then you can stop and remove that container using commands:

sudo docker stop <id>

sudo docker rm <id>