



Using Payara Platform with Docker on Microsoft® Azure

The Payara® Platform - Production-Ready,
Cloud Native and Aggressively Compatible.

Contents

Microsoft Azure Platform	1
Azure Portal	2
Get Started with Docker on Microsoft Azure	3
Prepare Docker Image to Run Your Payara Micro Application as a Docker Container on Azure	4
Create Docker Image	5
The Container Entrypoint	5
Deploy Application through Azure Portal	6
Docker Registry	6
Create and Start Container	9
Deploy Application through Azure CLI	11
Create Container Registry	11
Create and Start Container	12
Persist State on Volumes	12
Create File Share	13
Assign Volume to Container	13
Access Azure Files	15
Multiple Containers	16
Create Container Group	16
Using a Virtual Network	17
Docker on Azure Platform	18

Docker is an open source tool used to create, deploy and manage small portable containers. Containers are similar to virtual machines (VM), but where VMs run an entire operating system inside, the host containers only package the required components. Because of this, containers are extremely lightweight; they start-up in a fraction of the time of a regular VM, and waste very little extra resources on top of the main process being run inside them. They are used primarily as a lightweight way to assure that the program will run the same regardless of the host platform. Docker can also manage virtual networking between containers, as well as health checks to make sure each container is running properly.

The Payara Platform provides several Docker container images that can be used as-is to run your applications on Payara Server or Payara Micro. Or, you can create your own Docker images based on the provided Payara Docker container images.

Microsoft Azure Platform

The Microsoft Azure Platform is a very versatile environment. There are several options to run your application on the platform, including:

- **IaaS: Infrastructure as a Service:** It provides only a base infrastructure (Virtual machine, Software Define Network, Storage attached). The end-user has to configure and manage the platform and environment, and deploy applications on it.
- **PaaS: Platform as a Service:** It provides a platform allowing the end-user to develop, run, and manage applications without the complexity of building and maintaining the infrastructure.
- **CaaS: Container as a Service:** A form of container-based virtualization in which container engines, orchestration and the underlying compute resources are delivered to users as a service from a cloud provider.

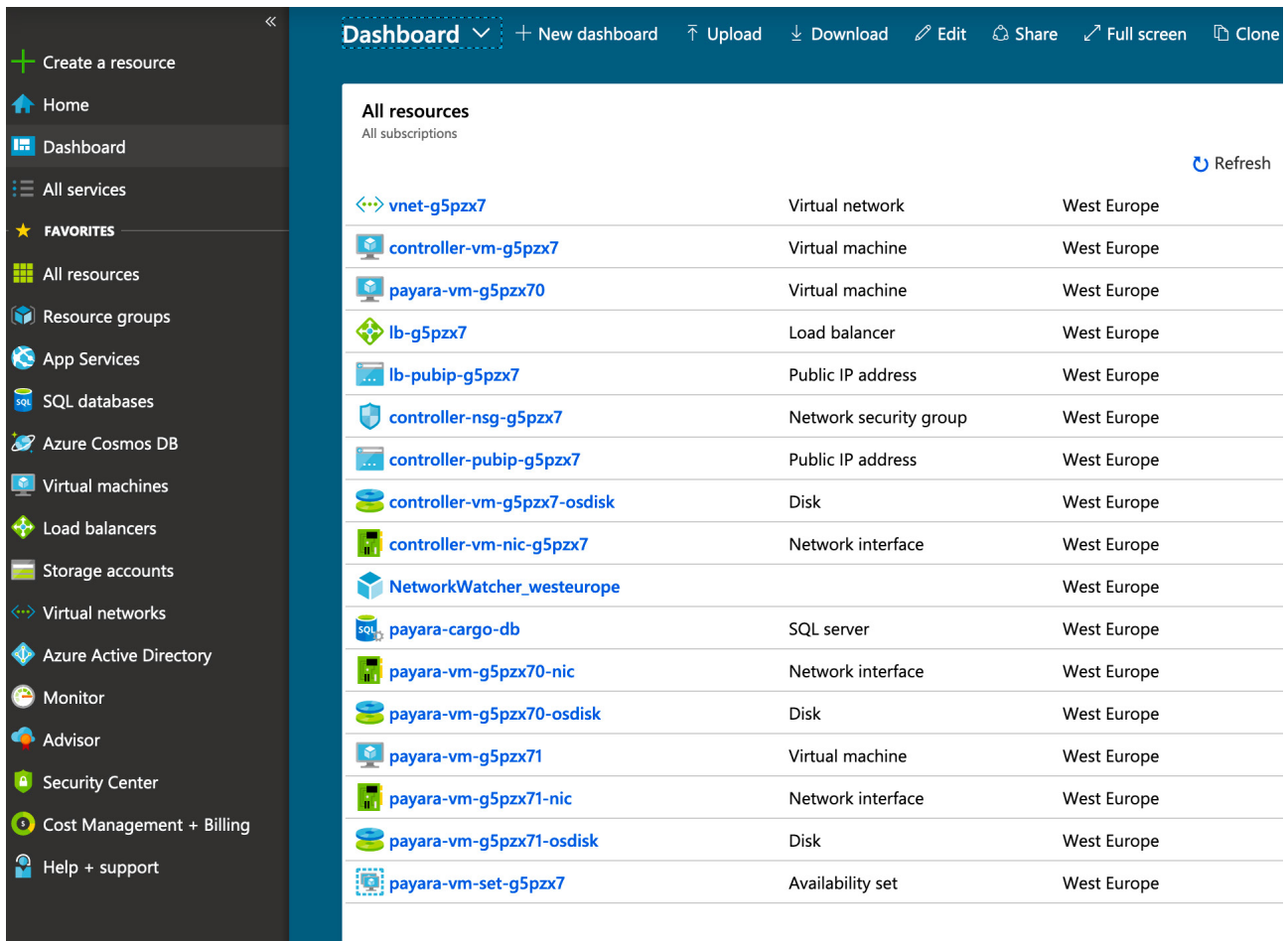
In this user guide, we focus a bit more on the Container as a Service option where the Microsoft Azure Platform provides all the capabilities for running your applications through the usage of a Docker Container in the cloud. Azure is ideal for creating traditional Payara Platform domain deployments where existing applications are lifted and shifted to the cloud.

With the Azure platform, you can maintain your CAAS with a graphic UI, the Azure Portal, or using a command-line tool.

Azure Portal

It's quick and easy to get started on the Azure Platform through the Azure Portal. It is a web-based, unified console that provides an alternative to command-line tools. With the Azure portal, you can manage your Azure subscription using a graphical user interface. You can build, manage, and monitor everything from simple web apps to complex cloud deployments, create custom dashboards for an organized view of resources, and configure accessibility options for the best experience.

To deploy the Payara Platform using the CAAS functionality, you create a Docker Registry and a Docker Container with a set of easy-to-follow screens and wizards. You don't need to know all the commands to set up a complete environment. Also, it gives you an overview of what is available in terms of configuration options, for example, or features.



Resource Name	Resource Type	Location
vnet-g5pzx7	Virtual network	West Europe
controller-vm-g5pzx7	Virtual machine	West Europe
payara-vm-g5pzx70	Virtual machine	West Europe
lb-g5pzx7	Load balancer	West Europe
lb-pubip-g5pzx7	Public IP address	West Europe
controller-nsg-g5pzx7	Network security group	West Europe
controller-pubip-g5pzx7	Public IP address	West Europe
controller-vm-g5pzx7-osdisk	Disk	West Europe
controller-vm-nic-g5pzx7	Network interface	West Europe
NetworkWatcher_westeurope		West Europe
payara-cargo-db	SQL server	West Europe
payara-vm-g5pzx70-nic	Network interface	West Europe
payara-vm-g5pzx70-osdisk	Disk	West Europe
payara-vm-g5pzx71	Virtual machine	West Europe
payara-vm-g5pzx71-nic	Network interface	West Europe
payara-vm-g5pzx71-osdisk	Disk	West Europe
payara-vm-set-g5pzx7	Availability set	West Europe

Working with a UI can be error-prone if you need to repeat your work when setting up a production environment after the configuration of a test environment. A scripted way of working is preferred to install your application environment using the Payara Platform. With the Azure Command Line tool, you can perform any action from the command prompt. Since you are in a scripting environment, it can also be automated.

Get Started with Docker on Microsoft Azure

Here's some terminology we'll be using:

Image	An image is a snapshot of how a container should look before it starts up. It will define which programs are installed, what the startup program will be, and which ports will be exposed.
Container	A container is an image at runtime. It will be running the process and the configuration defined by the image, although the configuration may differ from the image if any new commands have been run inside the container since startup.
DockerHub	DockerHub is a central repository where images can be uploaded, comparable to what Maven Central is for Maven artifacts. When pulling an image remotely, it will be pulled from DockerHub if no other repository is specified.
Repository	A repository in the context of Docker is a collection of Docker images. Different images in the repository are labelled using tags.
Tag	A tag distinguishes multiple images within a repository from one another. Often these tags correspond with specific versions. Because of this, if no tag is specified when running an image, the 'latest' tag is assumed.
Entrypoint	The Entrypoint command is run when the container starts. Containers will exit as soon as the entrypoint process terminates.
Resource Group	A container that holds related resources for an Azure solution like IaaS or Azure Containers is a resource group. The resource group includes those resources that you want to manage as a group. These resources are very versatile and include Virtual Machines, Registries, Networks, and IP address.
Subscription	You can create several subscriptions within your Azure account, each containing multiple resource groups. You mainly define some Subscription to have a separation in the billing and management of the resources.

The Docker support on Microsoft Azure is a bit different than when you would run the Docker system on your local laptop or on-premise Server. You do not use the Docker Client to send commands to the Docker Daemon on the host. With the Container As A Service solution of Azure, these Docker concepts are hidden within the platform. You still can run a Docker Container by pointing the system to a Docker Image, but you do not have the flexibility and all the functionality which the Docker client gives you.

The advantage of how the Azure Platform works is that you can get your custom image running quickly and easily, but it requires custom Azure functionality for some other concepts. Instead of using the standard Docker functionality, you need to use the Azure Files system to store data persistently and use the Virtual Networks to group some containers so that they can access each other endpoints directly.

Prepare Docker Image to Run Your Payara Micro Application as a Docker Container on Azure

In the next sections, we describe how you can get your application using Payara Micro running as a Docker Container on the Azure Platform using the Azure Portal and the Azure CLI. Since the solution is based on Docker Containers and Images, you are not limited to Payara Micro as a solution. You can use also Payara Server in a very similar way, but we choose Payara Micro here as it is a natural choice for cloud deployments.

But first, we prepare a Docker Image with our application. This step is always required, regardless of the deployment option, graphical or command line you choose. If you want more information on Docker itself, you can have a look at our guide “[Using Payara Micro with Docker](#)”.

Follow along with our demo video:



Create Docker Image

When you have developed your application using Eclipse MicroProfile or Java EE Web Profile, for example, you have a WAR file available at some point. You can also turn this WAR file into an executable JAR file and turn that into a Docker Image. We use the possibility to use this WAR file as is with Payara Micro and create a Docker Image for it.

Payara provides a Docker image for Payara Micro which is tuned for production usage but is also very useful in development. It's pre-configured in a way that makes it easy to start and use without a custom Dockerfile. You can also build your images using the Payara Micro image as a base to remove several required steps from your Dockerfile.

The Payara Micro Docker image can be found on DockerHub: [payara/micro](https://hub.docker.com/r/payara/micro)

Since we need a specific Docker Image containing the application, we can build it with the following Dockerfile.

Dockerfile

```
FROM payara/micro:5.192
COPY application.war $DEPLOY_DIR
```

Running the container produced from this image will also deploy the application on startup. The image will need rebuilding when the application changes.

The Container Entrypoint

The Payara Micro Docker image runs Java as PID 1, with an entry point of the Payara Micro jar. By default, the CMD arguments to the entry point are the deployment directory. This can be overridden using the CMD instruction in the Dockerfile, but the deployment directory would need to be specified again. For example:

Dockerfile

```
FROM payara/micro:5.192
COPY application.war $DEPLOY_DIR
CMD ["--nocluster", "--deploymentDir", "/opt/payara/deployments"]
```

This will run Payara Micro without starting Hazelcast® in cluster mode. See the [Payara Micro documentation](#) for a list of all possible arguments.

Building the images from the Dockerfile can be done using the Docker build command

commandline

```
docker build -t testmicro .
```

The tag associated with a Docker Image needs to have specified format for the Azure Docker Registry. So have a look in the following sections as you probably want to give it the appropriate tag immediately when you build it.

Deploy Application through Azure Portal

Let us go over the different steps to deploy an application using Payara Micro on the Azure Platform using the graphical UI.

Docker Registry

The Docker Image needs to be available in a Registry in order to be used by the Azure Platform. This can be the DockerHub repository, but you can also create a private Repository on the Azure platform which is preferred for your own applications.

You can create a registry through the portal but connecting to this registry and pushing images to it is easier using the Azure CLI.

Create container registry

☐ ☐

*

Registry name

payaraTest

✓

.azurecr.io

*

Subscription

Pay-As-You-Go

▼

*

Resource group

(New) myResourceGroup

▼

Create new

*

Location

West Europe

▼

*

Admin user

i

Enable

Disable

*

SKU

i

Basic

▼

When using the portal, look for a new resource called “Container Registry” or ACR. You need to specify the following values:

- *Registry name*, this name will become part of the URL on which your private Docker Registry is hosted. This needs to be unique across the world, so maybe your first choice is no longer available.
- *Resource Group*, is a logical grouping of resources. It makes it easy for you to find a specific resource or can be used to remove all resources in go (like all the resources you create when you follow the examples in this guide)
- *SKU* defines the size of the registry. For most cases, the Basic option will suffice.

Now that we have a Docker Registry available for our Docker images, we are ready to push to the Azure Infrastructure.

Since we have created a private Repository some authentication is required to access it. We can do this directly with the Docker CLI, but the Azure CLI has a very useful command for this, based on the credentials we have specified for this CLI.

commandLine

```
az acr login --name payaraTest
```

Now our Docker CLI is configured to send the appropriate authentication and we can prepare the Image and send it over to the Registry.

If you already have an image available, you can give it the correct tag so that it can be pushed. Otherwise, you can build it and supply the expected tag name.

The tag name needs to be of the following format:

```
<myRegistry>.azurecr.io/<name>:<version>
```

And of course, that version is optional as you probably know, but it is a good practice to always define it.

For example, when you already have a Docker Image available locally, called *testmicro* the following command defines it with the expected tag name:

commandline

```
docker tag testmicro payaratest.azurecr.io/testmicro:v1
```

Or you can specify the tag name directly when you build the Docker Image with a command similar to:

commandline

```
docker build -t payaratest.azurecr.io/testmicro:v1 .
```

And the last step we need to perform when we push the Image, is to include all the layers which it needs. So depending on the Image, it can take some time:

commandline

```
docker push payaratest.azurecr.io/testmicro:v1
```

Since we have executed that ‘azr acr login’ command earlier, Docker knows exactly the location where the image needs to be pushed.

Create and Start Container

Within the Portal, we need to create another Resource which has the Name “Container Instance” this time.

When selecting this resource, we get a wizard to enter the required information.

On the first screen, we specify our image. Since we are using the Private Repository we have created, we specify the location of our image in this repository payaratest.azurecr.io/testmicro:v1.

Create container instance

Azure Container Instances

PROJECT DETAILS

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

* Subscription ⓘ	<div>Pay-As-You-Go</div>
* Resource group ⓘ	<div>(New) myResourceGroup</div>

[Create new](#)

CONTAINER DETAILS

* Container name ⓘ	<div>payara-micro</div>
* Region ⓘ	<div>(Europe) West Europe</div>
* Image type ⓘ	<div><input type="radio"/> Public <input checked="" type="radio"/> Private</div>
* Image name ⓘ	<div>payaratest.azurecr.io/testmicro:v1</div>
* Image registry login server ⓘ	<div>payaratest.azurecr.io</div>
* Image registry user name ⓘ	<div></div>
* Image registry password ⓘ	<div></div>
* OS type	<div><input checked="" type="radio"/> Linux <input type="radio"/> Windows</div>
* Size ⓘ	<div>1 vcpu, 1.5 GiB memory, 0 gpus</div>

[Change size](#)



On the second screen of the wizard, we need to indicate all things related to the exposure of our application through some network information.

[Basics](#) [Networking](#) [Advanced](#) [Tags](#) [Review + create](#)

You can configure networking settings for your container, such as ports and protocols as well as a DNS name label. If you choose not to include a public IP address, you will still be able to access your container and logs using the command line. [Learn more about Azure Container Instances networking](#)

Include public IP address ☒ Yes ☐ No

Ports ⓘ

PORTS	PORTS PROTOCOL	
80	TCP	
<input type="text" value="8080"/> ✓	<input type="text" value="TCP"/> ▼	
<input type="text"/>	<input type="text" value="TCP"/> ▼	

DNS name label ⓘ

✓

.westeurope.azurecontainer.io

Since our image is using Payara Micro which makes our application available on port 8080, we fill in this value.

At the moment, there is no possibility to define a port mapping. So in our case, the application will also be available on port 8080.

Through a post-boot command of Payara Micro, you can of course change this port value in case the default value is not suited for you.

We also indicate that we need a public IP address and the dns-name so that we can access the application.

When this Resource is created, our application is available at the indicated DNS name, for `http://example test-micro.westeurope.azurecontainer.io:8080/test`.

Deploy Application through Azure CLI

In this section, we perform the same steps as above, but now we will use the Azure CLI to start and deploy our application on the Azure Platform using commands only.

Create Container Registry

The creation of the Container Registry can be performed by issuing the following commands:

commandline

```
az group create --name myResourceGroup --location eastus
```

This command creates a Resource Group, a logical grouping of your Azure resources. It allows you to easily find a specific resource or remove all resources related to this demo if you have finished testing.

For a location near you, have a look at this list <https://azure.microsoft.com/en-us/global-infrastructure/locations/> and the output of:

commandline

```
az account list-locations
```

Now that we have a resource group created, or we can reuse one of the Resource Groups we have created earlier of course, we can create the Registry with the following command:

commandline

```
az acr create --resource-group myResourceGroup --name payaratest --sku Basic
```

The next step is to deploy the Docker Image to this registry. The image needs to have the proper tag attached to it and pushed. The commands used to do this, *azr acr login*, *docker tag* and *docker push* are described in more detail in the section of the Azure Portal.

Create and Start Container

You can issue the following command on your console which creates the Docker Container and defines the networking information:

commandline

```
az container create --resource-group myResourceGroup --name test-micro --image payaratest.azurecr.io/testmicro:v1 --dns-name-label test-micro --ports 8080
```

The parameters of the container create command are

- resource-group : The resource group in which the Container is created.
- name: The name we assign to the Container which can be used later on to identify it.
- image: the tag name of the Docker image the system needs to use to create the Container.
- dns-name-label: The part of the final URL our application will be available on.
- ports: The ports which need to be exposed from the Container.

After executing this command, our application, through the Docker Image, is available on <http://test-micro.westeurope.azurecontainer.io:8080/>.

Persist State on Volumes

Just as with any Docker Container, when the Container running on the Azure Platform stops, all information is lost. Starting it up again is based on the Docker Image and thus no data from previous runs are available. Most common is that data is stored in some kind of database. But other common usages is that some files are stored on persistent storage which survives the (Docker) Container restart. As mentioned before, we do not have access to the Docker Daemon thus we need an alternative for the Docker Volumes on the Azure Platform. The Azure Files feature can be used together with the CAAS functionality and thus used as persistent storage.

Just as with any other feature of the Azure Platform, one can use the graphical way, the Azure Portal, or the Command line interface.

In this example, we will store the Payara Server log files to an Azure Files instance so that we have easy and permanent access to the server log files.

Create File Share

The first step we need to do is to create a volume which can be shared on the Azure Platform, including the Container solution. Although we have a subscription and account for the platform, we need a specific storage account which can be created with the following command:

commandLine

```
az storage account create
--resource-group myResourceGroup
--name <<storageAccountName>>
--location eastus
--sku Standard_LRS
```

- resource-group: The name of the resource group which will be linked to the storage account.
- name: of the storage account. There are a couple of restrictions on this value. It should be between 3 and 24 characters in length and use numbers and lower-case letters only. But more importantly, it must be unique within all the names already created by all the other users of the platform. So your first choice may be no longer available.
- location: Physical location of the data center where the file will be stored. You should keep this close to the location of the Container to reduce latency.

The next step is the creation of the actual Volume which can be assigned to the other Azure Resources.

commandLine

```
az storage share create --name <<storageName>> --account-name
<<storageAccountName>>
```

- name: of the Volume which can be shared.
- account-name: Determines the location and the resource group in which this Azure Files resource is created.

Assign Volume to Container

The data for the Azure File instance needs to be supplied during the creation of the Container Instance. There seems no possibility to supply this kind of information when creating the Container using the Azure Portal. Additional parameters are available when using the 'az container create' statement.

But since the access to our Volume is restricted, as it should since not everyone should have access to the contents of it, we need to have some kind of token which grants us all the right to it. This can be retrieved using the following command (and will be further on referred to as <<storageKey>>):

commandLine

```
az storage account keys list --resource-group myResourceGroup --account-name
<<storageAccountName>> --query "[0].value" --output tsv
```

The 'keys list' sub-command returns all kind of information, but with the additional parameters, we can restrict the output to the token itself. This is very useful in case you automate a few things using scripts (which you always should do in these cases)

Now that everything is in place and we have all the required information at hand, we can launch our final command. In this example, it is the launch of Payara Server within a container where we store the server log file to the Volume.

commandLine

```
az container create
  --resource-group myResourceGroup
  --name payara-test
  --image payara/server-full:5.192
  --dns-name-label payara-test
  --ports 8080
# The following parameters are related to the Azure Files storage.
  --azure-file-volume-account-name <<storageAccountName>>
  --azure-file-volume-account-key <<storageKey>>
  --azure-file-volume-share-name <<storageName>>
  --azure-file-volume-mount-path /opt/payara/appserver/glassfish/domains/
production/logs
```

Of course, in a real-world situation, you should have created your own custom Docker Image, containing your application ready for serving the users requests.

Access Azure Files

You can access the contents of the Azure Files resource in several different ways.

1. The Azure Files Resource supports SMB which means you can mount the resource as a 'disk' to your own laptop/computer, as described in the [documentation for Windows, Linux, and Mac OS](#).
2. You can copy the file for the Resource to your computer/laptop using the command line

The list of files on the storage can be retrieved by the 'file list' sub-command:

commandLine

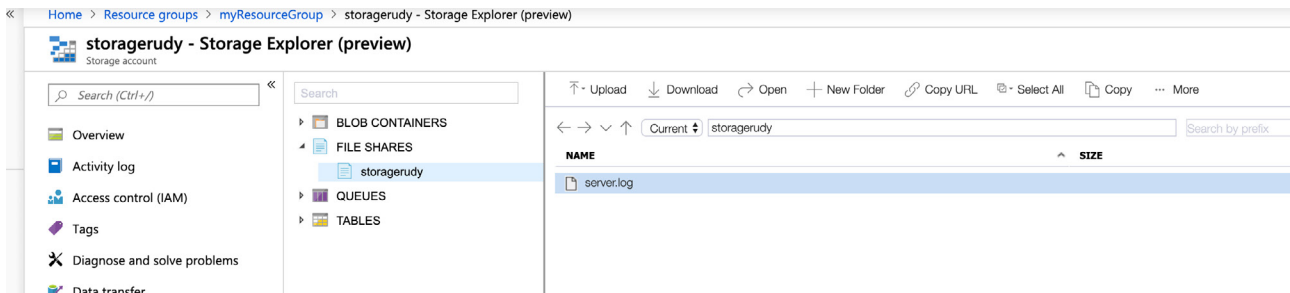
```
az storage file list
  --account-name <<storageAccountName>>
  --account-key <<storageKey>>
  --share-name <<storageName>>
  --path "myDirectory"
  --output table
```

Downloading a single to your computer/laptop is done by the following command:

commandLine

```
az storage file download
  --account-name <<storageAccountName>>
  --account-key <<storageKey>>
  --share-name <<storageName>>
  --path "server.log" \
  --dest "~/download/azure/server.log"
```

3. You can also use the Azure Storage Explorer which is integrated within the Azure Portal to have a look at the files and manipulate them.



Multiple Containers

There are various situations where you need multiple containers that need to work together. If we compare it with the plain Docker situation, it corresponds with the case where you put several containers in a custom Docker Network. As already mentioned, you cannot create such a custom Docker Network on the Azure Platform as you do not have direct access to the Docker system.

However, you have two options to achieve the same functionality on the Azure Platform but both solutions are rather complex, in comparison to the other interactions described in this user guide with the Azure Platform. Therefore, we will only describe the general idea of the solution here and refer you to the Azure documentation where you can find detailed descriptions on how you can perform it.

Create Container Group

Instead of creating just one container, you can also create multiple containers in one command and they can access each other very easily. However, it is not possible to just execute a simple command with some parameters as in the case when we created just one Container. You need to send multiple sets of the same data. To define this information, you can use JSON or YAML structure which follows the deployment template.

An example of a snippet of such a deployment template JSON:

Deployment Template JSON

```
{
  "$schema": "https://schema.management.azure.com/schemas/2015-01-01/
deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  ...
  "resources": [
    {
      "name": "myContainerGroup",
      "type": "Microsoft.ContainerInstance/containerGroups",
      "apiVersion": "2018-10-01",
      "location": "[resourceGroup().location]",
      "properties": {
        "containers": [
          {
            "name": "Payara Server DAS",
            "properties": {
              "image": "payara/server-full:5.192",
            ...

```

As you can see, within this structure you have the possibility to define multiple containers. The command to use this file, for example *azuredploy.json*, is as follows

commandLine

```
az group deployment create --resource-group myResourceGroup --template-file
azuredploy.json
```

You can read more about this option and details on this [Azure documentation page](#).

Using a Virtual Network

The Azure platform has a resource type Virtual Network. This resembles a classic network, but Virtual Machines and Containers assigned to the Virtual Network receives a private IP address. The usage of this private IP address ensures that the resources attached to it cannot be accessed from the outside but they can access each other and the internet.

You can also define subnets, just as with the physical networks to have a segmentation but also define some security rules.

The command to create the Virtual Network using the command line, you can do it also using the Azure Portal, is

commandLine

```
az network vnet create
  --name myVirtualNetwork
  --resource-group myResourceGroup
  --subnet-name default
```

However, the Azure documentation recommends that you specify the network ranges. We refer you to the online [documentation](#) how to can specify these values.

When the Virtual Network and Subnet are created, the parameters to define a certain Container are

commandLine

```
az container create
...
  --vnet <<virtualNetworkName>>
  --subnet <<subnetName>>
```

Note, however, the subnet can only contain Containers and no other Resources.

Docker on Azure Platform

We showed you in this guide how easy it is to deploy your Payara Micro-based application using Docker Image on the Azure Platform using a private Image Repository. Since the Azure platform has taken his approach for running Docker Images, you do not have access to the classic concepts as Volumes and Networks. You can also use Microsoft Azure's persistent storage to store information that needs to survive a Container restart, including Payara Server log files, and then deploy multiple containers using the Virtual Network Resource.

Together with all the other features and functionalities provided by the platform, it is an interesting solution for bringing your application to the cloud.

Microsoft® and Azure® are registered trademarks of Microsoft Corporation in the United States and/or other countries.

MicroProfile® is a registered trademark of the Eclipse Foundation.

Docker and the Docker logo are trademarks or registered trademarks of Docker, Inc. in the United States and/or other countries. Docker, Inc. and other parties may also have trademark rights in other terms used herein.



sales@payara.fish



+44 207 754 0481



www.payara.fish

Payara Services Ltd 2016 All Rights Reserved. Registered in England and Wales; Registration Number 09998946
Registered Office: Malvern Hills Science Park, Geraldine Road, Malvern, United Kingdom, WR14 3SZ