THOMAS MORE

Azure IaC

Kynan Van Looy
2CCS01
R0983516

2024 - 2025

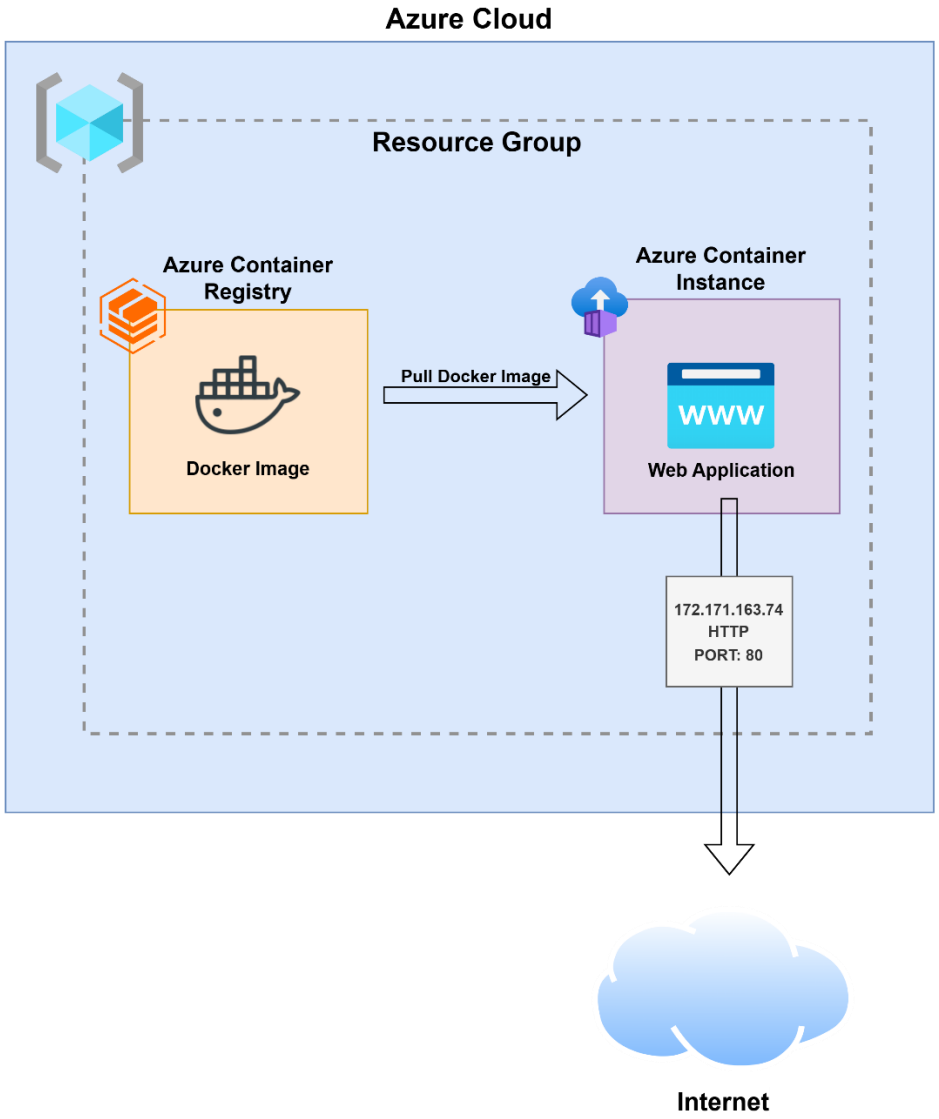
Table of contents

1. Introduction

This document is a step-by-step guide on how to deploy a CRUD application to Azure with the use of Infrastructure-as-Code (IaC). It will cover all the steps required to host the application, while keeping it easy to redeploy somewhere else if needed.

2. Structure Diagram

The diagram below shows the structure we are going to set up in the cloud. As you can see there is a resource group with both an Azure Container Registry (for storing the docker image) and an Azure Container Instance (to run the application in a container). The ACI will pull the image from the ACR and will then serve the application to the internet via port HTTP on port 80. The ACI has its own public IP address through which you can connect to the application.



3. Docker Image

First of all, we need an image for our application. For this, I used the code for the application from the provided Git repository. (https://github.com/gurkanakdeniz/example-flask-crud). Then I modified the Dockerfile in order to automatically deploy the application. The Dockerfile can be found below:

```
# Use an official Python runtime as a parent image
FROM python:3.9

# Set the working directory in the container
WORKDIR /app

# Copy the current directory contents into the container at /app
COPY . /app

# Install any needed packages specified in requirements.txt
RUN pip install --no-cache-dir -r requirements.txt

# Initialize the database
RUN python -c "from app import db; db.create_all()"

# Expose port 80 for the Flask app
EXPOSE 80

# Define environment variables for Flask
ENV FLASK_APP=app.py
ENV FLASK_RUN_HOST=0.0.0.0
ENV FLASK_RUN_PORT=80

# Run the application
CMD ["flask", "run"]
```

After that, I built the Docker image using following command:
“docker build . -t kvl-flask-crud-app”

This command builds the image and names it “kvl-flask-crud-app”. The screenshot below should be the result when executing this command.

```
PS C:\School\Schooljaar 2024-2025 - Semester 2\Cloud Platforms\Azure IAC\azure-iac\flask-crud> docker build . -t kvl-fla
sk-crud-app
[+] Building 1.7s (10/10) FINISHED                                                          docker:desktop-linux
 => [internal] load build definition from Dockerfile                                                    0.1s
 => => transferring dockerfile: 634B                                                                    0.0s
 => [internal] load metadata for docker.io/library/python:3.9                                           1.1s
 => [internal] load .dockerignore                                                                       0.0s
 => => transferring context: 2B                                                                         0.0s
 => [1/5] FROM docker.io/library/python:3.9@sha256:bc2e05bca883473050fc3b7c134c28ab822be73126ba1ce29517d9e8b7f370  0.0s
 => => resolve docker.io/library/python:3.9@sha256:bc2e05bca883473050fc3b7c134c28ab822be73126ba1ce29517d9e8b7f370  0.0s
 => [internal] load build context                                                                       0.0s
 => => transferring context: 2.66kB                                                                     0.0s
 => CACHED [2/5] WORKDIR /app                                                                           0.0s
 => CACHED [3/5] COPY . /app                                                                            0.0s
 => CACHED [4/5] RUN pip install --no-cache-dir -r requirements.txt                                     0.0s
 => CACHED [5/5] RUN python -c "from app import db; db.create_all()"                                    0.0s
 => exporting to image                                                                                  0.2s
 => => exporting layers                                                                                 0.0s
 => => exporting manifest sha256:801f31f4950d56b8f5b79ed0018b341b5eaffe1e4641f9c67971a73002e4a5f7       0.0s
 => => exporting config sha256:060f82cca7a5d8b7f7e50d8de36455d356e6a3f71fed611172e548fe352d78b5         0.0s
 => => exporting attestation manifest sha256:7011879862dfada7a1454b365150a1aad795f350ee91e95f7cdef1d6e7ed3402  0.1s
 => => exporting manifest list sha256:56794fdf7aa0045211d5496a26695203ea6d60ed75a02c3e14aba4368f2288dc  0.0s
 => => naming to docker.io/library/kvl-flask-crud-app:latest                                            0.0s
 => => unpacking to docker.io/library/kvl-flask-crud-app:latest                                         0.0s

What's next:
    View a summary of image vulnerabilities and recommendations → docker scout quickview
PS C:\School\Schooljaar 2024-2025 - Semester 2\Cloud Platforms\Azure IAC\azure-iac\flask-crud>
```

4. Resource Group

The next step is to create a resource group on Azure. This is a logical container which will hold all of our resources. We do this using following command:

```
PS C:\School\Schooljaar 2024-2025 - Semester 2\Cloud Platforms\Azure IAC\azure-iac\flask-crud> az group create --name KV
LResourceGroup --location eastus
{
  "id": "/subscriptions/665b2b07-157f-492c-a47f-ad002b94a17b/resourceGroups/KVLResourceGroup",
  "location": "eastus",
  "managedBy": null,
  "name": "KVLResourceGroup",
  "properties": {
    "provisioningState": "Succeeded"
  },
  "tags": null,
  "type": "Microsoft.Resources/resourceGroups"
}
```

This command creates a resource group with the name “KVLResourceGroup”, which will run on the servers in East US.

5. Azure Container Registry

Then, we must create a container registry. This registry will store our docker images, so that the container instance can later access them. We will do this with the use of Infrastrucure-as-Code, and in Azure, this means bicep files. The following bicep file will be used to create the Azure Container Registry inside the resource group we created earlier.

```
// Set the name of the ACR
param acrName string = 'kvlAcr'

// Get the location of the resource group
param location string = resourceGroup().location

// Create the ACR resource
resource acr 'Microsoft.ContainerRegistry/registries@2023-07-01' = {
  // Set the properties such as name, location and stock keeping unit
  name: acrName
  location: location
  sku: {
    name: 'Basic'
  }
}
```

Now that we have the configuration file for the ACR, we still must deploy it to Azure. We will do this with the use of Azure CLI. The following command is used to deploy this resource:

az deployment group create --resource-group KVLResourceGroup --template-file ../Azure/acr.bicep

```
PS C:\School\Schooljaar 2024-2025 - Semester 2\Cloud Platforms\Azure IAC\azure-iac\flask-crud> az deployment group creat
e --resource-group KVLResourceGroup --template-file ../Azure/acr.bicep
The configuration value of bicep.use_binary_from_path has been set to 'false'.
{
  "id": "/subscriptions/665b2b07-157f-492c-a47f-ad002b94a17b/resourceGroups/KVLResourceGroup/providers/Microsoft.Resourc
es/deployments/acr",
  "location": null,
  "name": "acr",
  "properties": {
    "correlationId": "a44173c7-b3ed-44fc-91ad-aa58b5942672",
    "debugSetting": null,
    "dependencies": [],
    "duration": null,
    "error": null,
    "mode": "Incremental",
    "onErrorDeployment": null,
    "outputResources": [
      {
        "id": "/subscriptions/665b2b07-157f-492c-a47f-ad002b94a17b/resourceGroups/KVLResourceGroup/providers/Microsoft.C
ontainerRegistry/registries/myacrdihamlg4kam2i",
        "resourceGroup": "KVLResourceGroup"
      }
    ],
    "outputs": {
      "acrLoginServer": {
        "type": "String",
        "value": "myacrdihamlg4kam2i.azurecr.io"
      }
    },
    "parameters": {
```

The next step is to push our Docker image to this registry. For this, we first have to login to the ACR login server. To do this, we first have to get some information from the ACR. With the commands below, you can put the name of the ACR and the name of the login server in a variable, so it's easier to reuse later.

```
PS C:\School\Schooljaar 2024-2025 - Semester 2\Cloud Platforms\Azure IAC\azure-iac\flask-crud> $ACR_NAME = (az acr list
--resource-group KVLResourceGroup --query "[0].name" --output tsv)
PS C:\School\Schooljaar 2024-2025 - Semester 2\Cloud Platforms\Azure IAC\azure-iac\flask-crud> $ACR_LOGIN_SERVER = (az a
cr list --resource-group KVLResourceGroup --query "[0].loginServer" --output tsv)
```

Next up, we have to login to the ACR. We do this with the following command:

```
PS C:\School\Schooljaar 2024-2025 - Semester 2\Cloud Platforms\Azure IAC\azure-iac\flask-crud> az acr login --name $ACR_
NAME
Uppercase characters are detected in the registry name. When using its server url in docker commands, to avoid authentic
ation errors, use all lowercase.
Login Succeeded
PS C:\School\Schooljaar 2024-2025 - Semester 2\Cloud Platforms\Azure IAC\azure-iac\flask-crud>
```

If everything is correct, this should be successful. Next up we will tag the image we built earlier with the login server of the ACR. This can be done with the following command:

```
PS C:\School\Schooljaar 2024-2025 - Semester 2\Cloud Platforms\Azure IAC\azure-iac\flask-crud> docker tag flask-crud-app
 $ACR_LOGIN_SERVER/flask-crud-app:v1
PS C:\School\Schooljaar 2024-2025 - Semester 2\Cloud Platforms\Azure IAC\azure-iac\flask-crud>
```

Lastly, we have to push the image to the registry. For this we use the docker push command, like visible in the screenshot below. If everything went well, the image should now be inside the registry one Azure.

```
PS C:\School\Schooljaar 2024-2025 - Semester 2\Cloud Platforms\Azure IAC\azure-iac\flask-crud> docker push $ACR_LOGIN_SE
RVER/flask-crud-app:v1
The push refers to repository [kvlacrdihamlg4kam2i.azurecr.io/flask-crud-app]
255774e0027b: Pushed
5b5d22f54067: Pushed
353e14e5cc47: Pushed
091eb8249475: Pushed
2d9631d9be96: Pushed
f299e0671245: Pushed
6e02a90e58ae: Pushed
5eb33f5a35e8: Pushed
7cd785773db4: Pushed
e8bee085e55f: Pushed
f4214d9fa5c8: Pushed
f6d72b00ae7c: Pushed
v1: digest: sha256:06c3b420d04db47b6dc70c0668bf4857de880e3b9533d957edfff7eb3209ddbf size: 856
PS C:\School\Schooljaar 2024-2025 - Semester 2\Cloud Platforms\Azure IAC\azure-iac\flask-crud>
```

6. ACR Token

After we have prepared our ACR, it's time to create and deploy our ACI, and run our application. For this, we will once again use a bicep file, but first we must do some preparation work. Since the ACR is private, the ACI can't access it to pull an image without permission. Therefore, we first must create and ACR Token, which the ACI can use to authenticate with the ACR.

First, we have to create a scope map. This groups the repository permissions we apply to a token. To this, we have to use the Azure CLI and execute the following command:

```
PS C:\School\Schooljaar 2024-2025 - Semester 2\Cloud Platforms\Azure IAC\azure-iac\flask-crud> az acr scope-map create --name kvlScopeMap  --registry kvlAcr   --repository kvl-flask-crud-app content/read
{
  "actions": [
    "repositories/kvl-flask-crud-app/content/read"
  ],
  "creationDate": "2025-03-30T13:55:18.794520+00:00",
  "description": null,
  "id": "/subscriptions/665b2b07-157f-492c-a47f-ad002b94a17b/resourceGroups/kvlresourcegroup/providers/Microsoft.ContainerRegistry/registries/kvlAcr/scopeMaps/kvlScopeMap",
  "name": "kvlScopeMap",
  "provisioningState": "Succeeded",
  "resourceGroup": "kvlresourcegroup",
  "systemData": {
    "createdAt": "2025-03-30T13:55:18.495543+00:00",
    "createdBy": "r0983516@student.thomasmore.be",
    "createdByType": "User",
    "lastModifiedAt": "2025-03-30T13:55:18.495543+00:00",
    "lastModifiedBy": "r0983516@student.thomasmore.be",
    "lastModifiedByType": "User"
  },
  "type": "Microsoft.ContainerRegistry/registries/scopeMaps",
  "typePropertiesType": "UserDefined"
}
PS C:\School\Schooljaar 2024-2025 - Semester 2\Cloud Platforms\Azure IAC\azure-iac\flask-crud>
```

This will create a scope map in the registry we created and apply the content/read permissions to the repository in which we put our image.

Next up, we will create our token with the following command:

```
PS C:\School\Schooljaar 2024-2025 - Semester 2\Cloud Platforms\Azure IAC\azure-iac\flask-crud> az acr token create --name kvlAcrToken --registry kvlAcr  --scope-map kvlScopeMap
```

This will create a token with then name kvlAcrToken, inside our ACR and linked to the scope map we made earlier. After this, we have to generate a password for this token, which we can later use for our ACI. We generate this with this command:

```
PS C:\School\Schooljaar 2024-2025 - Semester 2\Cloud Platforms\Azure IAC\azure-iac\flask-crud> az acr token credential generate --name kvlAcrToken --registry kvlAcr
```

This will create 2 passwords. Make sure to save these somewhere safe.

7. Azure Container Instance

Now that we have an ACR Token, we can start deploying our ACI. For this we will create another bicep file.

```
// Set the name of the login server
param acrLoginServer string = 'kvlacr.azurecr.io'

// Set the name of the ACR Token
param acrUsername string = 'kvlAcrToken'

// Set the name of ACR
param acrName string = 'kvlAcr'

// Set the name of container
param containerName string = 'kvl-flask-crud-app'

// Get the location of the resource group
```

```
param location string = resourceGroup().location

// Set the password of the Token, handled in a secure way (via cli)
@secure()
param acrPassword string

// Check if the ACR resource exists
resource acr 'Microsoft.ContainerRegistry/registries@2023-07-01' existing = {
  name: acrName
}

// Create the ACI resource
resource aci 'Microsoft.ContainerInstance/containerGroups@2022-09-01' = {
  // Set the name and location of the ACI
  name: containerName
  location: location
  properties: {
    containers: [
      {
        // Specify which image to use from the registry
        name: containerName
        properties: {
          image: '${acr.properties.loginServer}/kvl-flask-crud-app:v1'
          // Set the ports which will be opened and the protocol to use
          ports: [
            {
              protocol: 'TCP'
              port: 80
            }
          ]
          // Set the resources of the container
          resources: {
            requests: {
              cpu: 1
              memoryInGB: 2
            }
          }
        }
      }
    ]
    // Set the credentials to use to authenticate with the ACR
    imageRegistryCredentials: [
      {
        server: acrLoginServer
        username: acrUsername
        password: acrPassword
      }
```

```
    ]
    // Specify which operating system will be used
    osType: 'Linux'
    // Create a public ip adress on port 80
    ipAddress: {
      type: 'Public'
      dnsNameLabel: '${containerName}-dns'
      ports: [
        {
          protocol: 'TCP'
          port: 80
        }
      ]
    }
  }
}
```

This file will create an ACI resource and authenticate with the ACR to pull the image from the registry. It will then serve the image on port 80 with a public IP address.

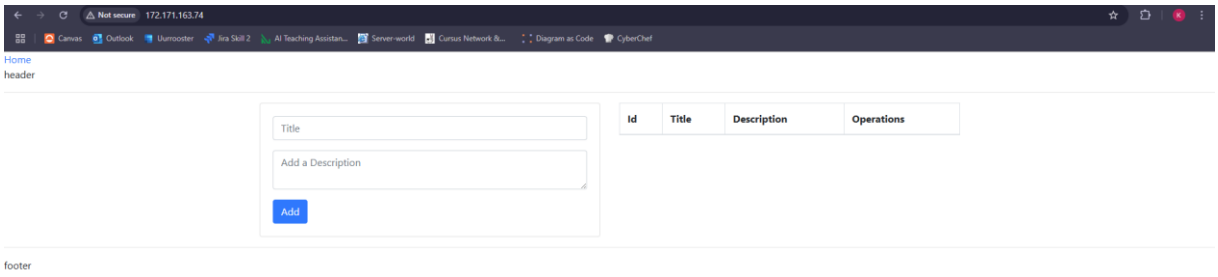
To deploy this resource, we will use the CLI again. We will deploy it with the password of the ACR Token.

```
PS C:\School\Schooljaar 2024-2025 - Semester 2\Cloud Platforms\Azure IAC\azure-iac\flask-crud> az deployment group creat
e --resource-group kvlResourceGroup --template-file ../Azure/aci.bicep --parameters acrPassword='XUprJV91iHUlmTjywCkX7/k
vqBXN7q8XojhmuE1bfV+ACRBqs6OG'
 \ Running ..
```

As you can see there is a new parameter with the value acrPassword. This will hold the password to authenticate with the ACR. If everything went well, your ACI should now be running the application on port 80.

8. Result

If you followed every step in this document, you should now be able to go to the public IP address of the ACI, and connect with the application.



This should be the result. Everything works and is running in the cloud.

9. Conclusion

You now have a working CRUD application running on Azure in the cloud. Small changes can be easily made and everything can be redeployed fast thanks to the IaC we used. This was a challenging task, but I managed to figure it out and get everything working.