

|  |
| --- |
| Azure DevOps Lab  **Microsoft Azure** |
| Module 8. Azure container services.  Home tasks |

## Task 1

1. Create azure Kubernetes service.

Create initial deployment ARM template that will be used for nested templates execution.

Create nested template that will create Azure Kubernetes Service cluster with 2 nodes and Azure Virtual Network.

1. Install and configure kubectl.

Install and configure kubectl that will be used for cluster management.

* 1. Install and configure azure cli:

<https://docs.microsoft.com/en-us/cli/azure/install-azure-cli?view=azure-cli-latest>

* 1. Login to azure portal:

[az login](https://docs.microsoft.com/en-us/cli/azure/authenticate-azure-cli?view=azure-cli-latest)

* 1. Install kubectl for Kubernetes cluster management:

az aks install-cli

* 1. To configure kubectl to connect to your Kubernetes cluster, use the az aks

get-credentials command. This command downloads credentials and configures the Kubernetes CLI to use them:

az aks get-credentials --resource-group myResourceGroup --name myAKSCluster

* 1. Test connections:

kubectl get nodes

1. Create azure container registry.

Create nested template that will create Azure Container Registry.

1. Create secret for container registry.

Login to azure portal using azure cli or PowerShell. Download your AKS credential config. Make sure that you can get access to cluster from your PC. Use the following **kubectl** command to create the Kubernetes secret. Replace <acr-login-server> with the fully qualified name of your Azure container registry (it's in the format "acrname.azurecr.io"). Replace <service-principal-ID> and <service-principal-password> with the values you obtained by running the previous script. Replace <email-address> with any well-formed email address.

kubectl create secret docker-registry acr-auth --docker-server <acr-login-server> --docker-username <service-principal-ID> --docker-password <service-principal-password> --docker-email <email-address>

## Task 2

1. Create an empty directory. Change directories (cd) into the new directory, create a file called Dockerfile, copy-and-paste the following content into that file, and save it. Take note of the comments that explain each statement in your new Dockerfile.

**Dockerfile**

# Use an official Python runtime as a parent image

FROM python:2.7-slim

# Set the working directory to /app

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 --trusted-host pypi.python.org -r requirements.txt

# Make port 80 available to the world outside this container

EXPOSE 80

# Define environment variable

ENV NAME World

# Run app.py when the container launches

CMD ["python", "app.py"]

1. **Create 2 files for python application:**

**Requirements.txt**

Flask

Redis

**App.py**

from flask import Flask

from redis import Redis, RedisError

import os

import socket

# Connect to Redis

redis = Redis(host="redis", db=0, socket\_connect\_timeout=2, socket\_timeout=2)

app = Flask(\_\_name\_\_)

@app.route("/")

def hello():

try:

visits = redis.incr("counter")

except RedisError:

visits = "<i>cannot connect to Redis, counter disabled</i>"

html = "<h3>Hello {name}!</h3>" \

"<b>Hostname:</b> {hostname}<br/>" \

"<b>Visits:</b> {visits}"

return html.format(name=os.getenv("NAME", "world"), hostname=socket.gethostname(), visits=visits)

if \_\_name\_\_ == "\_\_main\_\_":

app.run(host='0.0.0.0', port=80)

1. **Build application with docker file and push it to docker registry that was create on Part 2, step 3.**

Notes: read the article below to get more information about development process that based on docker:

<https://docs.docker.com/get-started/part2/>

## Task 3

1. **Deploy test application to Kubernetes.**

Create two files with content below:

**Deployment.yaml**

apiVersion: apps/v1

kind: Deployment

metadata:

name: applicationName

spec:

selector:

matchLabels:

app: applicationName

replicas: 1

template:

metadata:

labels:

app: applicationName

spec:

containers:

- name: applicationName

image: imageName

ports:

- containerPort: 80

**Service.yaml**

apiVersion: v1

kind: Service

metadata:

name: applicationName

spec:

selector:

app: applicationName

ports:

- protocol: "TCP"

port: 80

targetPort: 80

type: LoadBalancer

1. **Set application name and your docker image that was pushed on part 2. Deploy application using kubectl CLI to Azure aks cluster.**

As a result of the steps below, you should be able to open the service by IP.

1. **Connect to Azure Kubernetes UI dashboard.**

Kubernetes includes a web dashboard that can be used for basic management

operations.

This dashboard lets you view basic health status and metrics for your applications, create and deploy services, and edit existing applications. To start the Kubernetes dashboard, use the [az aks browse](https://docs.microsoft.com/cli/azure/aks#az-aks-browse) command. The following example opens the dashboard for the cluster named myAKSCluster in the resource group named myResourceGroup:

az aks browse --resource-group myResourceGroup --name myAKSCluster

Read more about Kubernetes:

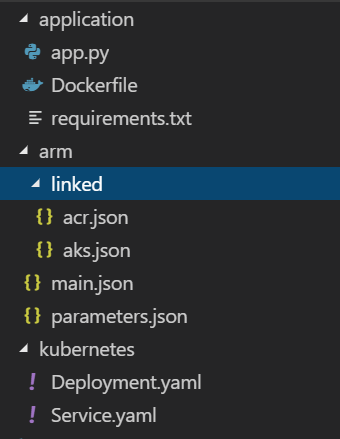
Service;

Deployment;

Pods.

## Hometasks requirements

1. All configuration is under git.
2. Directory layer:



1. Scripts:

ARM automation

Script for preparation according to Part 2.

Script for kubernetes deployment.

1. Documentation with all steps that need to be done to run application.

## Global requirements for homework.

1. One JSON file for describing all Azure resources is forbidden. **Please use linked templates**.

2. Linked templates **must be called from** **the initial deployment template** (main.json).

3. Main and parameters templates must be named as **main.json** and **parameters.json** accordingly.

4. Maximum **number of parameters** in Main.json is **5.**

5. Main.json and parameters.json must be **executed from local folders**. Using **-TemplateParameterUri** and **-TemplateUri** options in PS script are forbidden.

6. All artifacts (JSONs and PS files) must be stored in **Azure Storage Account**. Using any GitHubs or other public repos is forbidden.  
7. Create a **PowerShell** **deployment** with following functionality:

                a. Create resource group.

                b. Create storage account and container within for artifacts (For example: JSONs, PS file(s), ZIP files).

                c. Upload the linked templates and other task-related artifacts to the created Storage Account.

d. Execute main.json file for deploy Azure resources.

8. Each ARM json file must have at least **1 output**.

9. All homework **artifacts must be executable** (e.g. if Mentor starts your script execution and it fails - all homework artifacts will be sent back for fixing)

10. All **resources must be deleted** after homework completion.

11. Use the next **folder structure** for storing artifacts. **Subfolder** with resources JSONs must be named “**linked**”: