# Multi-service applications in the Cloud

## Objectives

In this lab assignment you will:

- Build and deploy a multi-service application with **Docker Compose**.
- Deploy a multi-service application on a **local Kubernetes cluster**.
- Deploy a multi-service application on a **Kubernetes cluster** in **Microsoft Azure**.

## Submission

- In order to submit your work, you need to answer all the questions that you **find here**.
- **Only one member** of the group must fill in the questionnaire.
- You can answer the open questions either in French or in English.
- Some questions require you to upload files.
- You can pause the test at any moment. Your answers will be stored; you'll find them when you resume the test.
- After answering the last question, you need to click on the button *Terminer le test* (*Finish attempt* if you use the interface in English) to submit the questionnaire. **The submission is final. After submitting, you cannot change your answers anymore.**
- **Deadline: 23 May 2024, 23h59**

## Context

We intend to build and deploy a Web application called *TripMeal* that let users share their favorite recipes. You can download the application here. The source code has been readapted from this GitHub repository.

The downloaded file is an archive. Extracting the archive will create a folder named `tripmeal_sujet`.

## 1 Bulding and deploying with Docker Compose

You're going to build and deploy the application using Docker Compose. In the folder `tripmeal_sujet` you should see a file named *docker-compose.yml*.

**Exercise**
**Exercise 1.1.** Answer on Edunao. What is the file *docker-compose.yml* used for?

Solution

The folder contains:
- A file `docker-compose.yml` that will contain the instructions to build and deploy the application.
- A subdirectory for each service composing the application. Each subdirectory contains the source code of the corresponding service, as well as a Dockerfile with the instructions to build an image for the service.

Look at the files and folders inside the folder `tripmeal_sujet`.

**Exercise**
**Exercise 1.2.** Answer on Edunao. How many services does the application *TripMeal* have?

Solution

The application consists of two services:
- web: It is a web application, developed with a combination of HTML, Python and Flask
- db: It is a relational database. From the Dockerfile, which is already given, we understand that the DBMS used is MySQL.

The `Dockerfile` in the directory `db` is already implemented. Open it and read the content to answer the following questions.

**Exercise**
**Exercise 1.3.** Answer on Edunao. Which DataBase Management System (DBMS) is used in the *TripMeal* application?

Solution

MySQL. You can see it from the Dockerfile of the db service.

**Exercise**
**Exercise 1.4.** Answer on Edunao. What is the version (tag) of the image used for the database management system of *TripMeal*? Find the documentation of the image and use it to answer this question.

Solution

The documentation is available at here. Since no tag is specified in the Dockerfile, the latest version is selected.

**Exercise**
**Exercise 1.5.** Where does Docker get the base image for the database management system from?

Solution

From the Docker official registry. In fact, in the Dockerfile we only give the name of the image and we don't specify any reference to another registry. Therefore, by default, Docker looks into its own registry.
Of course, if the image is already available locally, Docker does not look for it in the registry.

## 1.1 Dockerfile in folder `web`

If you open folder `web`, you'll see that the Dockerfile is empty.

**Activity**
Complete the `Dockerfile` in the directory `web`.
- You need a Python 3.7 environment to run the application.
- The application consists of several files and folders. You need to include all of them into a Docker image.
- Add the following instruction in your Dockerfile:

```
RUN chmod -x path_to_file_app
```

where `path_to_file_app` is the path to the file `app.py` **inside the image**. This will prevent an `Exec format error` from occurring.

Time to make sure that we can build an image from this Dockerfile and we can run a container from that image.

**Exercise**
**Exercise 1.6.**
- Build an image from the Dockerfile that you've just completed using the `docker build` command. You can give the image any name you want. For simplicity, I assume that you call it `web-test`.
- Run a container from this image by typing the following command. The command will only work if you type it from the directory where the file `tripmeal.env` is stored.

```
docker run --env-file tripmeal.env -p 3000:5000 web-test
```

- Open a Web browser and type the following URL: `localhost:3000`. You should see the interface of the application. If that's not the case, stop the container, and correct the Dockerfile.
- If you click on the links *New recipe*, *All recipes*, *Weekly menu* and *Login* in the application interface, you will get a connection error to the database server. This is normal and will be fixed later.

If the image passes the test, **upload the Dockerfile to Edunao.**

Solution

Different solutions exist, here is a proposition. It is important that we build a minimal image, so we choose the environment python:3-7-slim.

```
FROM python:3.7-slim
RUN mkdir -p /app & \
    mkdir -p /app/templates & \
    mkdir -p /app/static
RUN /usr/local/bin/python -m pip install --upgrade pip
COPY ./static /app/static/
COPY ./templates /app/templates/
COPY requirements.txt /app/
WORKDIR /app
RUN pip install -r requirements.txt
COPY app.py dbconnect.py /app/
ENTRYPOINT ["python","app.py"]
```

## 1.2 The environment variables

When testing the image of the service `web`, you may have noticed that we passed the file `tripmeal.env` as an argument to the command `docker run`. This file contains the definition of **environment variables** that are used in the application. Open this file and look at the environment variables to answer the following questions.

**Exercise**
**Exercise 1.7.** Answer on Edunao. What does the environment variable SERVER_PORT refer to?

Solution

SERVER_PORT refers to the port where the the web application will be waiting for incoming requests.

**Exercise**
**Exercise 1.8.** Answer on Edunao. Which of the following sentences are true?

a. When you execute TripMeal, SERVER_PORT will be opened on the network interface of the host computer (your computer).
b. When you execute TripMeal, SERVER_PORT will be opened on the network interface of a container.
c. In order to access the Web interface of TripMeal, you'll need to map SERVER_PORT to a port number p; p will be opened on the network interface of the host computer (your computer).
d. When you execute TripMeal, you'll be able to access the Web interface by typing the URL localhost:$SERVER_PORT in your browser.

Solution

b. and c. are true.

**Exercise**
**Exercise 1.9.** Answer on Edunao. When you write the file `docker-compose.yml`, you'll need to give the database service a name. Which of the environment variables tells you the name that you must use?

Solution

The variable that tells us the name of the database service is DATABASE_HOST. The web service connects to the database that is found at DATABASE_HOST. In our case, the host is a container. Evidently, Docker Compose uses the name of the service as the host name for the container.

## 1.3 Volumes and networks

The *TripMeal* application keeps user information and recipe information in a database. Therefore, you'll need to define a Docker **volume** to hold the data.

**Exercise**
**Exercise 1.10.** Answer on Edunao. Look at the documentation of the base image of the database management system of the application. Which directory inside the database image will you attach the volume to?

Solution

Looking at the documentation of the MySQL image, the path is /var/lib/mysql.

The containers composing the *TripMeal* application need to communicate through a Docker **network**.

**Exercise**
**Exercise 1.11.** Answer on Edunao. If you don't explicitly create any network, the containers will be able to communicate anyway. How do you explain it?
**Note.** You can answer this question after writing the Docker compose file and running the application.

Solution

By default, containers that are not explicitly connected to a network are automatically connected to the network called bridge, which is the default network in Docker. Therefore, the services of the application TripMeal can communicate through this network.

**Exercise**
**Exercise 1.12.** Answer on Edunao. Why is not creating a network for the application considered a bad idea, even if the application works?

Solution

All containers created on the host machine will be connected to a single network. As a result, all containers can communicate. If only one container is compromised, the attacker will have a shot at compromising all of them.

## 1.4 The compose file

You're finally ready to complete the file `docker-compose.yml`.

**Activity**
Write the file `docker-compose.yml`.
- Remember that you need to pass the **environment variables** to your application.
- As a documentation of Docker Compose you can use:
  - The examples that we've seen together.
  - The overview presented on the official Docker website.
  - You can also find the full specification of Compose here.
- Run the application and test it. You should be able to see the Web interface of the application, create an account, share a recipe and look at the list of shared recipes.
- If you experience problems, you might find the command `docker-compose logs` useful.

Solution

```
version: "3"

services:
    web:
        build: web
        image: quercinigia/trip-meal-web
        env_file: tripmeal.env
        networks:
            - tripmeal-network
        ports:
            - "5000:5000"
    db:
        build: database
        image: quercinigia/trip-meal-db
        env_file: tripmeal.env
        networks:
            - tripmeal-network
        volumes:
            - db-data:/var/lib/mysql

volumes:
    db-data:

networks:
    tripmeal-network:
```

If you successfully deployed the application, you can continue.

**Shut down the application!**
Shut down both the application before you move on.

**Exercise**
**Exercise 1.13.** Upload your file `docker-compose.yml` to Edunao.

**Exercise**
**Exercise 1.14.** Use Docker Compose to push all the images that you built in this section to your DockerHub registry.
**In the answer to this exercise write the link to the uploaded images.**

Solution

We can use DockerCompose to push the images. The command is:
`docker-compose push`

## 2 Local Kubernetes cluster

We intend to deploy the application *TripMeal* on a **local Kubernetes cluster** (either Docker Desktop or Minikube).

**Exercise**
**Exercise 2.1.** Answer on Edunao. For each service of the application *TripMeal*, specify the Kubernetes objects that you need to create and their types. Justify your answers.

Solution

For the service `web` we need two objects:
- A **deployment**, which is collection of identical pods, each pod being an instance of the service. Deployments are often used for stateless services, which is the case of the `web` service.
- A **LoadBalancer service**. We need it in order to expose the service so that a client can connect to it.

For the service `db` we need two objects:
- A **StatefulSet**, which is a sort of Deployment used for stateful services.
- A **ClusterIP service**, used to expose the service internally in the Kubernetes cluster. The database is only used the by service `web`, it doesn't need to be accessed from external clients.

**Help**
In order to write the specification of each object, you can refer to the examples that we discussed together in the second tutorial. You can also refer to the API documentation on the Kubernetes website.

**Activity**
For each Kubernetes object:

1. **Configure.** Write a Yaml configuration file. Remember that you need to pass the environment variables to the application.
2. **Create.** Create the object in Kubernetes with `kubectl`.
3. **Analyze.** Execute the command `kubectl get all` to confirm that all components run as intended.
4. **Test.** Play with the application to confirm that is works as intended.

**In case of problems.**
You can look at the logs of a pod with the following command: `kubectl logs NAME-OF-POD --previous`

Solution

The configuration of the Deployment associated to the `web` service is as follows:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web
spec:
  replicas: 1
  selector:
    matchLabels:
      app: tripmeal
      service: web
  template:
    metadata:
      labels:
        app: tripmeal
        service: web
    spec:
      containers:
      - image: quercinigia/trip-meal-web:latest
        name: web
        ports:
        - containerPort: 5000
          protocol: TCP
        env:
        - name: DATABASE_NAME
          value: "tripmealdb"
        - name: DATABASE_USER
          value: "root"
        - name: MYSQL_ROOT_PASSWORD
          value: "my-secret-pw"
        - name: DATABASE_HOST
          value: "db"
        - name: DATABASE_PORT
          value: "3306"
        - name: TRIPMEAL_KEY
          value: "my-secret-key"
        - name: SERVER_PORT
          value: "5000"
```

When we create the deployment with the command:
`kubectl create -f web-deployment.yml`
three objects are created in Kubernetes:

- A pod, that represents an instance of the service. Here we specified only one replica, hence we have only one pod.
- A Deployment, the object that we demanded.
- A ReplicaSet, the object that controls the collection of pods that are the instances of the service. The ReplicaSet is managed by the Deployment.

The configuration of the LoadBalancer service associated with the `web` service is as follows:

```
apiVersion: v1
kind: Service
metadata:
  name: web
spec:
  type: LoadBalancer
  ports:
  - port: 5000
    targetPort: 5000
    protocol: TCP
  selector:
    app: tripmeal
```

**Shut down the application!**
Shut down the application by removing all the Kubernetes objects.

**Exercise**
**Exercise 2.2.** Create a new file called `tripmeal.yml` that contains the definition of all the Kubernetes objects of the application, as we have seen in the conclusion of the tutorial 2.
**Upload this file to Edunao.**

# 3 Kubernetes cluster on Microsoft Azure

At this point you should have successfully deployed the application on your local Kubernetes cluster.

You're now going to create a Kubernetes cluster on **Microsoft Azure** and deploy *TripMeal* on that cluster.

**Warning**
If you haven't activated it yet:
- Connect to this webpage.
- Click on `Start free`.
- Sign in by using your CentraleSupélec credentials.
- Follow the instructions.

During this activity, you'll need to answer some questions that encourage you to gain a deeper knowledge of the Azure platform and better understand the commands that you type. **Feel free to read the documentation on the Azure platform in order to answer the questions.**

## 3.1 Login to Azure through CLI

**Warning**
If you get the message *command not found* when you type `az`, it means that you haven't installed the Azure CLI yet. You'll find more information on the Edunao course page.
Make sure you run version 2.25 or higher of the Azure CLI, by typing:
`az --version`
**If you use a VM with Multipass you'll need to install the Azure CLI with the following command:**
`curl -sL https://aka.ms/InstallAzureCLIDeb | sudo bash`
**If you use a VM either with Multipass or with VirtualBox you'll need to re-install kubectl:**
- Type the following command so as the command `kubectl` does not refer to `minikube kubectl` anymore:

`unalias kubectl`
- Type the following command to install a new version of `kubectl`:

`sudo az aks install-cli`

Whether you're on Windows, macOS or Linux, you need to **open a terminal**.

In order to log in to your Azure account, type the following command:

`az login`

A Web page will open in your default Web browser, where you can type your username and password (your CentraleSupélec credentials). After authenticating your Azure account, you can go back to the terminal, where you should see some information about your account, such as:

```
[
  {
    "cloudName": "AzureCloud",
    "homeTenantId": "<id>",
    "id": "<id>",
    "isDefault": true,
    "managedByTenants": [],
    "name": "Azure for Students",
    "state": "Enabled",
    "tenantId": "<id>",
    "user": {
      "name": "<email-address>",
      "type": "user"
    }
  }
]
```

## 3.2 Deploy and use Azure Container Registry

When we run TripMeal on a local Kubernetes cluster, we assumed that our computer could have a direct access to the Internet and, therefore, to the DockerHub registry, where we could pull the images of the TripMeal services. When we run a containerized application in production, we cannot make this assumption as the production servers have often no direct access to the Internet.

We need to place the images in a container registry that is in the same context as our production servers. Since we're going to run the application on Azure, we can use the **Azure Container Registry (ACR)**.

### 3.2.1 Registry creation

First, we need to create a **resource group**.

> **Exercise**
> **Exercise 3.1.** Answer on Edunao. What is a **resource group** in Azure and how is it useful?

Solution

> It is a logical container of resources that are managed together, typically because they belong to the same application. Resource groups are useful to manage with a single click the whole set of resources (e.g., the permissions). For instance, when we want to decommission the application, we can remove all the resources with just one command/one click.

The command to create a new resource group is the following (replace RES_GRP_NAME with a name of your choice).

```
az group create --name  RES_GRP_NAME --location francecentral
```

> **Warning**
> If `francecentral` does not work for your account, the previous command will normally suggest a possible location. Choose one that is near you.

You can verify that the resources are correctly created by looking in the Azure portal.

Next, we need to create a **container registry** with the following command (replace RES_GRP_NAME with the name of your resource group and REG_NAME with a name of your choice for the registry).

```
az acr create --resource-group RES_GRP_NAME --name REG_NAME --sku Basic
```

**Exercise**
**Exercise 3.2.** In the previous command, what does `sku` refer to? Feel free to look up on the Azure website to find the answer.

Solution

SKU stands for *Stock Keeping Unit*, it represents the different service levels of a product. Here we use a registry container with a *Basic* service level.

**Exercise**
**Exercise 3.3.** Answer on Edunao. Based on the specified `sku`, find on the Internet how much the container registry will cost (in euros) per day if the container registry is instantiated in the region `France Central`.

Solution

Each sku is associated with a price. Here students need to do some searching. If you look at this webpage we find the price of a basic container registry and the included storage. The idea here is to make students aware of the fact that the options that we select have a cost.

### 3.2.2 Registry login

After creating the registry, we can log into it with the following command:

```
az acr login --name REG_NAME
```

### 3.2.3 Image tagging

We're almost ready to push the images that compose the application *TripMeal* to the registry. In order to do that, we need to **tag** (i.e., rename) our images so that their names are preceded by the **login server name** of the registry.

In order to get the name of the **login server name** (that we denote here as `acrloginserver`) of your registry, you can type the following command:

```
az acr list --resource-group RES_GRP_NAME --query "[].{acrLoginServer:loginServer}"
--output table
```

Your `acrloginserver` will be something like `xxx.azurecr.io`.

**Activity**
Tag the images that correspond to the services of the application *TripMeal* so that their name is similar to: `acrloginserver/nameofimage:latest`. You can use the command `docker tag`.

Solution

> We can use the `docker tag` function. For instance, in my case I use the following two commands to rename my two images:
> `docker tag quercinigia/trip-meal-web:latest tripmealacr.azurecr.io/trip-meal-web:latest`
> `docker tag quercinigia/trip-meal-db:latest tripmealacr.azurecr.io/trip-meal-db:latest`

#### 3.2.4 Pushing the images

We can push the images with the following command (use your image name instead of `xxx.azurecr.io/imagename:latest`).

`docker push xxx.azurecr.io/imagename:latest`

Make sure to **type this command for each image** that you want to push.

> **Warning**
> It might take few minutes before the images are completely pushed to the registry.

Finally, verify that the images are actually in the registry with the following command:

`az acr repository list --name REG_NAME --output table`

Alternatively, you can connect to your Azure portal and visually browse your resources.

### 3.3 Deploy a Kubernetes cluster

We now deploy a **Kubernetes cluster on Azure**.

#### 3.3.1 Cluster creation

We create the cluster with the following command (replace CLUSTER_NAME with a name of your choice. As before, RES_GRP_NAME is the name of your resource group and REG_NAME is the name of the container registry).

```
az aks create \
    --resource-group RES_GRP_NAME \
    --name CLUSTER_NAME \
    --node-count 2 \
    --generate-ssh-keys \
    --attach-acr REG_NAME
```

**Warning**
If you get an error on your SSH key, then your key is not in the right format.
Here is how we suggest you to proceed.

1. Open your home directory in Visual Studio.
2. Locate the hidden folder `.ssh`.
3. You should have a file named `id_rsa.pub` under folder `.ssh`. Open it.
4. Copy the whole content of the file.
5. Create a new file under folder `.ssh`. Give it a name of your choice (for instance, `id_kube_rsa.pub`).
6. Paste the content into the new file, while making sure that you don't have any whitespace before the first character.
7. Save and close the file.
8. Type the command `az aks create` and replace the `--generate--ssh-keys` option with `--ssh-key-value ~/.ssh/id_kube_rsa.pub`.

The cluster will take a while to start. Time for a coffee! But before, answer the following question!

**Exercise**
**Exercise 3.4.** What is the meaning of the option `node-count` in the previous command?

Solution

The number of nodes in the Kubernetes cluster. Here we specify 2. Note that this refers to the number of worker nodes. In fact, the control plane is managed by Azure. It is still possible though to create a custom advanced configuration to control the number of master and etcd nodes.

### 3.3.2 Connect to the cluster

We can configure `kubectl` to connect to the newly created cluster. You need to type the following command:

```
az aks get-credentials --resource-group RES_GRP_NAME --name CLUSTER_NAME
```

Now, your Kubernetes cluster should be visible locally. To verify it, type the following command:

```
kubectl config get-contexts
```

Here *context* refers to the Kubernetes clusters that *kubectl* has access to. The Kubernetes cluster that you created on Azure should be visible in the output; an asterisk should appear in front of its name, indicating that it is the *current context* (that is the Kubernetes cluster being currently referenced by `kubectl`).

Type the following command:

```
kubectl get nodes
```

You should see the information about the nodes in the cluster.

## 3.4 Deploy the application

We're almost done! The images are in the registry, the Kubernetes cluster is up and running. The only missing piece of the puzzle is our application *TripMeal*.

First thing to do is to slightly modify the file `tripmeal.yml` that you created at the end of the previous section.

**Activity**
Look at the names of the images of each service in that file. How must these names change? Modify the file accordingly (**no need to upload it on Edunao**).

Solution

The existing names refer to the images on the DockerHub registry. We must change them so that they refer to the images on our Azure container registry.

It is the moment you've been waiting for! Deploy your application by typing the following command:

```
kubectl apply -f tripmeal.yml
```

Look at Kubernetes objects created after this command:

```
kubectl get all
```

Wait for all the components to be up and running. Get the external IP address of the web service and try to connect to the application, in the same way you did in the previous section.

**If you can play with the application like you did in your local deployment it means that you reached the conclusion of this assignment! Bravo!**

**Exercise**
**Exercise 3.5.** Submit a **video** like that the one that **you can see here**.
The video must **clearly** show that:
- You're connected to **your Azure portal** (Email address on the top right corner of the portal).
- All the passages that you see in the sample video: you need to show the **public IP address** of the application that you deployed, use that address to connect to your application and play with the application to show that it works correctly.

The next exercise is **optional**.

If you don't intend to do it, **make sure to read the conclusion of this document to take down the application and remove all resources!!**.

**Exercise**
**Exercise 3.6.** The title of this exercise is *Cerise sur le gâteau* (i.e., this is **optional**!).
At this point, you can connect to *TripMeal* by using an IP address. It would be nice if you could use a URL, like in the real websites. Can you find a way to assign a URL to your web service? **Describe your procedure.**
You need a bit of Googling here. . . .

Solution

We have to modify the definition of the web service in the file `tripmeal.yml` More precisely, here is the new definition:

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    service.beta.kubernetes.io/azure-dns-label-name: mytripmeal
  name: web
spec:
  type: LoadBalancer
  ports:
  - port: 5000
    targetPort: 5000
    protocol: TCP
  selector:
    app: tripmeal
    service: web
```

We added the field `metadata.annotations` and specify a DNS label. Then we need to restart the service. If everything goes well, the application is reachable at the following address:
`http://mytripmeal.francecentral.cloudapp.azure.com:5000/`
The URL is composed by the DNS label that we specified + the region where we deployed the cluster + cloudapp.azure.com

## Conclusion

Make sure you follow these instructions:

- Take down your application in Kubernetes by using the following command:

```
kubectl delete -f tripmeal.yml
```

- Change the context of the `kubectl` command so that it points back to a local Kubernetes cluster. Type the following command, where CONTEXT_NAME will be `docker-desktop` or `minikube`, depending on which local Kubernetes cluster you use.

```
kubectl config use-context CONTEXT_NAME
```

- **Destroy all the resources** linked to the application *TripMeal* on Microsoft Azure, otherwise you'll get billed even if you don't use them! You can destroy all the resources by simply deleting the resource group to which they belong. You can do it through the Azure portal or by typing the following command (replace RES_GRP_NAME with the name of the resource group that you intend to remove).

```
az group delete --name RES_GRP_NAME
```

- You can check the balance of your Azure credit here.
- You can stop Docker and Kubernetes if you don't need it anymore.