SAE3 – Déploiement – TD1

BUT INFO – SAE3

La société *HerbergIQ* vous propose deux technologies et hébergements associés à utiliser dans le cadre de vos SAE :

- **ASP.Net Core API, SQLite** : dans ce cas, il faudra déployer votre application sur les serveurs dockers mises à votre disposition. Le but de ce TD est de voir comment cela fonctionne.

- **PHP** : dans ce cas, il faudra déployer sur le serveur IIS/PHP/MySQL mis à votre disposition, comme vous l'avez fait en R3-01 par exemple. La liste des comptes a été mise à jour et est disponible sur le commun.

Exercice 1 – Création d'une nouvelle application API web ASP.Net Core

1) Lancer Visual Studio. Créer un projet C# « API web ASP.NET Core ».
2) Choisir un nom de projet (par exemple SAE3DepTD1) et un emplacement pour le sauvegarder.
3) Ensuite, choisir .Net 8.0, garder la configuration HTTPS, activer la prise en charge de conteneur, avec une cible conteneur OS Linux et un type de build Dockerfile.
4) Lorsque Visual Studio vous demande de démarrer Docker Desktop, répondez-lui « Oui ».
5) Vous pouvez tout de suite l'exécuter, cela démarrera un petit exemple en local. Ainsi, par l'intermédiaire de Swagger, vous pourrez interroger l'API.
    a. URL Swagger : http://localhost:32768/swagger/index.html ou https://localhost:32769/swagger/index.html (elle doit s'ouvrir automatiquement dans votre navigateur par défaut).
    b. URL sans Swagger (pour obtenir un JSON) : https://localhost:32769/WeatherForeCast par exemple.

⇨ *Vous possédez donc maintenant une application qui se déploie dans un conteneur pour s'exécuter en local sur votre machine. Nous allons passer (Exercice 2) à l'étape configuration de cette conteneurisation pour permettre son export et son exécution en dehors de votre machine.*

Exercice 2 – Configuration pour l'exportation du docker

Il nous faut indiquer à l'application les URLs et ports à utiliser (notamment pour l'HTTPS).

1) Dans l'explorateur de solutions, ouvrir le fichier « Properties/launchSettings.json ». Dans la section « Container / Docker », ajouter une 3ème variable d'environnement :

```
"environmentVariables": {
  "ASPNETCORE_HTTPS_PORTS": "8081",
  "ASPNETCORE_HTTP_PORTS": "8080",
  "ASPNETCORE_URLS": "https://+:8081;http://+:8080"
},
```

2) Ouvrir le fichier « Dockerfile » et ajouter la ligne dans le premier bloc :

```
FROM mcr.microsoft.com/dotnet/aspnet:8.0 AS base
USER app
WORKDIR /app
EXPOSE 8080
EXPOSE 8081
ENV ASPNETCORE_URLS="https://*:8081;http://*:8080"
```

⇨ *Cette ligne permet d'être en cohérence avec les URLs que l'on a indiqué dans la question 1 (pour accepter les connexions en HTTPS sur le port 8081 et les connexions en HTTP sur le port 8080).*

3) Dans le deuxième bloc, ajouter :

```
FROM mcr.microsoft.com/dotnet/sdk:8.0 AS build
ARG BUILD_CONFIGURATION=Release
RUN dotnet dev-certs https --trust
WORKDIR /src
…
```

⇨ *Cette ligne permet de valider l'utilisation du certificat auto-généré et auto-signé par Visual Studio quand on crée une application conteneurisable.*

4) Et dans le dernier bloc (attention, il n'y a que 2 lignes, ENV, et COPY (la ligne COPY est trop grande pour tenir sur une seule et même ligne)) :

```
FROM base AS final
WORKDIR /app
ENV ASPNETCORE_Kestrel__Certificates__Default__Path="/https/thecertif.pfx"
COPY --from=publish --chmod=555
/root/.dotnet/corefx/cryptography/x509stores/my/* /https/thecertif.pfx
COPY --from=publish /app/publish .
…
```

⇨ *La première ligne permet d'indiquer où sera, dans notre docker, le certificat à utiliser pour gérer l'HTTPS.*

⇨ *La deuxième ligne permet de copier dans le conteneur le certificat à utiliser. Remarque : cela est très bien pour une utilisation développement/test, ce que nous faisons ici dans le cadre de la SAE. Sur une plateforme de production, en entreprise donc, il faudrait plutôt passer par un proxy de type nginx et un certificat délivré par une autorité de certification pour assurer la connexion HTTPS.*

5) **Attention**, pour la suite (Exercice 3), il faut exécuter votre API en mode « **Release** » (parfois un Nettoyage de la solution (clic-droit sur la solution puis Nettoyer) s'impose avant que le mode Release fonctionne).

Exercice 3 – Déploiement du docker sur le serveur mis à votre disposition

1) Lancer Terminal.
2) Lancer votre Debian (ou Ubuntu) (flèche vers le bas dans la barre de titre). **Si ni Debian, ni Ubuntu sont installés**, il vous faudra installer une distribution Linux par la commande :

   wsl --install -d Debian
   (wsl --list --online permet d'obtenir la liste des distributions Linux disponibles).

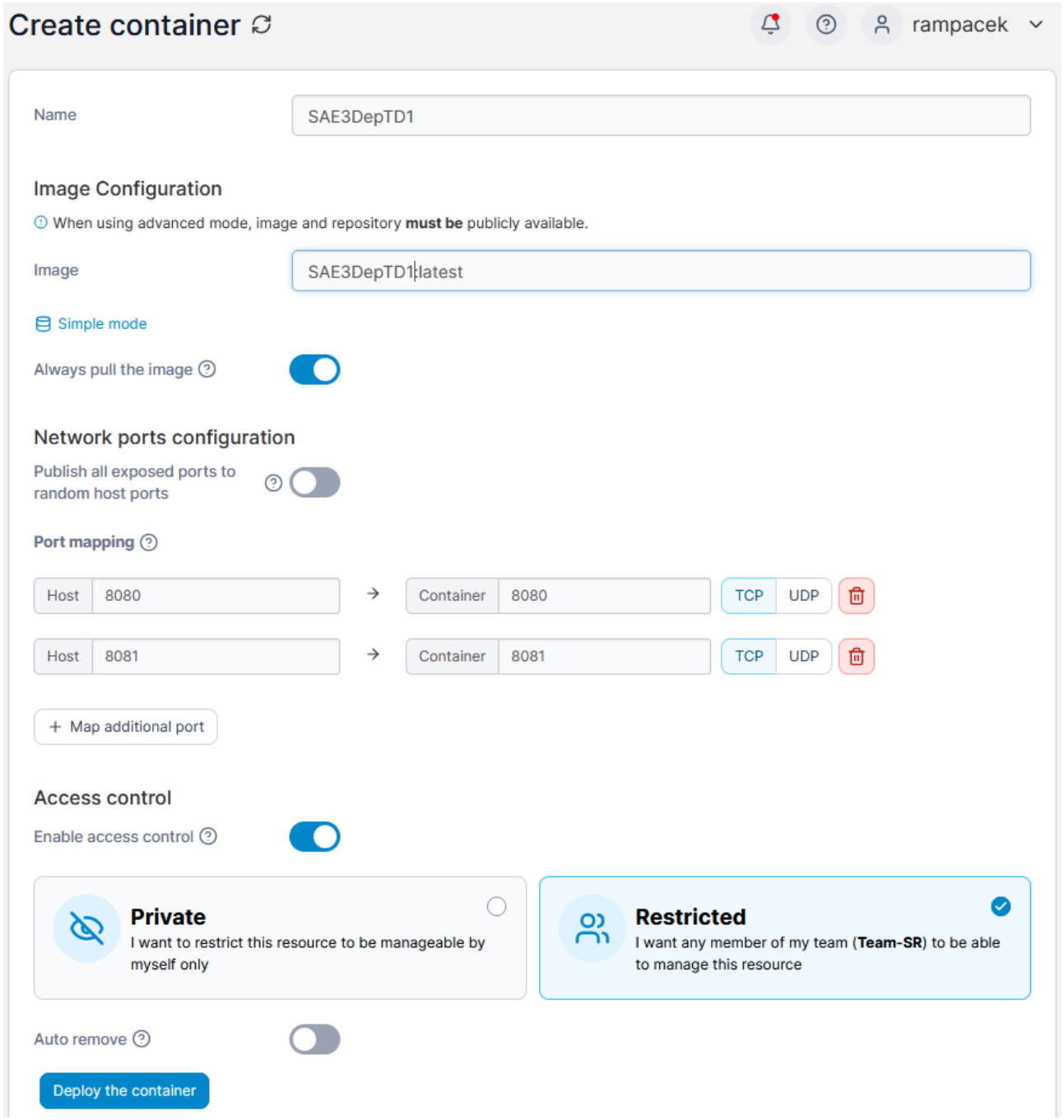
3) La commande « docker image list » vous montre la liste des conteneurs disponible sur votre WSL votre application à déployer est bien là (SAE3DepTD1).

   Remarque : si la commande « docker image list » retourne une erreur comme quoi la commande docker n'est pas disponible, alors il faut aller dans « Docker Desktop », puis Settings, puis Ressources, puis WSL Integration, et Activer Debian (ou la distribution que vous utilisez).

4) Sauvegarde l'image docker :

   docker save sae3deptd1:latest | gzip > sae3deptd1.tgz

5) Depuis votre interface Portainer : https://10.128.207.XXX:9443, après être identifié, sélectionner « Home » puis « IQ-SAE-DCK-SYGZ ».
6) Ensuite, dans « Images », cliquer sur « Import » puis « Select File ». Chercher le fichier SAE3DepTD1.tgz (dossier similaire à \\wsl.localhost\Debian\home\login). Et enfin, cliquer sur « Upload ».
7) Ensuite, dans « Containers », nous allons créer le conteneur (cf capture ci-dessous).
   a. Cliquer sur « Add container »
   b. Name : lui indiquer un nom, par exemple « sae3deptd1 »
   c. Cliquer sur « Advanced Mode » et remplir le champs « Image* » par :
      i. sae3deptd1:latest
   d. Cliquer sur « Map additional port »
      i. 8080 -> 8080 (ce sera pour l'HTTP)
      ii. 8081 -> 8081 (ce sera pour l'HTTPS)
   e. Choisir « Access Control » et « Restricted ».
   f. Puis cliquer sur « Deploy the Container ». Vous conteneur nouvellement créer doit normalement être affiché avec une icône « Running » verte.



8) Pour tester : https://10.128.207.XXX:8081/WeatherForecast -> vous devez obtenir un JSON similaire à l'Exercice 1 (noter que c'est normal que Swagger ne fonctionne pas ici).

⇨ *Vous êtes donc maintenant capable de déployer votre application (partie serveur) sur le serveur mis à votre disposition.*

Exercice 4 – Déployer une application existante utilisant SQLite

Nous allons prendre l'exemple MacIUT_API_Net8.7z sur le commun (application - ASP.Net Core API contenant une base de données SQLite, .Net8), mis à disposition par M. Simonet, et nous allons là conteneuriser.

1) Ouvrir la solution.
2) Dans l'explorateur de solution, sur le projet MacIUT_API, faire un clic-droit puis « Ajouter », puis « Prise en charge de Docker ».
3) Choisir « Linux » comme cible.
4) Ouvrir le fichier « Dockerfile », dans la dernière partie, ajouter la ligne indiquant de copier la base de données dans le conteneur :

```
…
COPY --from=publish /app/publish .
COPY --chmod=666 ["MacIUT_API/database.db", "."]
ENTRYPOINT ["dotnet", "MacIUT_API.dll"]
```

5) Ensuite, réaliser l'Exercice 2 et l'Exercice 3 en changeant :
   a. SAE3DepTD1 en MacIUT_API ;
   b. sae3deptd1 en maciutapi.

Remarque : *pour vérifier que la base de données SQLite est bien présente dans votre docker, lancer la console du docker*[1] *(lorsque celui-ci est exécuté), et afficher les fichiers dans le dossier /app. Il doit contenir database.db, et celui-ci ne doit pas être vide (s'il est vide, c'est que la commande de COPY si dessous a échoué).*

6) Tester votre API, après l'avoir déployée, par une URL du type : https://10.128.207.XXX:8081/Types/GetAll.

Exercice 5 – Client C#

Prendre l'exemple MacIUT_Client.7z du client C# sur le commun. Plutôt que de faire des requêtes API en localhost, nous allons devoir les faire vers le conteneur sur l'adresse IP 10.128.207.XXX.

Il faudra bien sûr continuer à utiliser HTTPS (de toute façon, .Net l'oblige depuis quelques temps et redirige les connexions HTTP directement vers HTTPS). Par contre, le client va vouloir vérifier

[1] Dans DockerDesktop : Containers/SAE3DepTD1/Exec. Taper bash. Puis ls -al /app.
Dans Portainer : Containers, puis, sur la ligne SAE4DepTD1, cliquer sur l'icône « >_ » de la colonne Quick Actions. Puis ls -al.

l'authenticité du certificat. Et le fait que nous utilisions des certificats autosignés va le faire générer une exception. Il faut donc gérer cela.

1) Plutôt que d'accepter l'ensemble des certificats, nous allons limiter au certificat autosigné que l'on utilise. Pour ce faire, nous allons récupérer la signature du certificat. Dans votre navigateur, rendez-vous sur la page de votre API : https://10.128.207.XXX:8081/Types/GetAll par exemple. Cliquer, à côté de l'URL, sur le cadenas, puis « connexion non sécurisée », puis « plus d'informations »[2]. Choisir ensuite « Afficher le certificat ». Dans la section « Empreintes numériques », récupérer la SHA-1.
2) Dans l'application MacIUT_Metier, ouvrir le fichier Data/DAO.cs.
3) Changer l'attribut adressAPI :

```
private string adressAPI = "https://10.128.207.XXX:8081/";
```

4) Dans le constructeur, ajouter la partie autorisation du certificat autosigné (en changeant la signature) :

```
public DAO()
{
    HttpClientHandler handler = new HttpClientHandler();
    handler.ServerCertificateCustomValidationCallback = (message, cert, chain,
sslPolicyErrors) => {
        if (cert.GetCertHashString() ==
"BEF549E4EB9E285B01A7D9F496DD181E9E266AF4") // empreinte SHA-1 (cf question 1)
        {
            return true;
        }
        return false;
    };
    this.client = new HttpClient(handler);
}
```

Exercice 6 – Client JS

Vous pouvez également écrire un client au format JavaScript.

1) Reprendre un exemple sur le commun ClientAPI_JS.7z.
2) Il faut faire une modification dans votre API, pour les problèmes de CORS (vérification de l'origine des requêtes). Dans le fichier *program.cs* de l'API, ajouter la ligne surligné ci-dessous :

```
app.UseHttpsRedirection();
app.UseCors(options => options.AllowAnyHeader().AllowAnyHeader().AllowAnyOrigin());
app.UseAuthorization();
```

3) Redéployer votre API.

[2] Ceci est valable pour Firefox, vous pourrez facilement adapter aux autres navigateurs.

4) Dans le client JavaScript, dans le fichier js/Data/dao.js, modifier la ligne :

```
this._adresseAPI = "https://localhost:7024";
```

en

```
this._adresseAPI = "https://10.128.207.XXX:8081";
```

5) Tester le bon fonctionnement du client JS en ouvrant le fichier index.html dans votre navigateur. Vous devez pouvoir faire une commande (double-clic que les items) et la valider.

Exercice 7 – Sur votre ordinateur personnel.

Si besoin, pour tester ce TD ou votre SAE sur votre ordinateur personnel, il vous faut donc installer :

- Le VPN de l'IUT (pour accéder à l'hébergement) -> https://iutdijon.u-bourgogne.fr/siav/category/acces-a-distance-vpn/ .
- Visual Studio 2022
- WSL2 -> https://iutdijon.u-bourgogne.fr/intra/info/softs/files/wsl2.pdf et un Linux[3] déployé dessus (Debian de préférence, mais cela fonctionne également avec Ubuntu).
- Docker Desktop (en indiquant la cible Linux pendant l'installation) -> https://docs.docker.com/desktop/install/windows-install/#what-to-know-before-you-install

Exercice 8 – Utiliser un volume pour stocker des fichiers

Nous allons maintenant éviter que les fichiers stockés dans le conteneur ne disparaissent à chaque déploiement d'une nouvelle version de celui-ci.

Pour ceci, nous allons créer des volumes. Un volume de données contient des fichiers, organisé à la mode Linux. Il faut donc le monter/attacher à un « point de montage »/dossier (par exemple « /data ») de votre conteneur.

Nous allons réaliser cette opération aussi bien sur votre PC de développement (pour tester) que sur le serveur de déploiement.

1) Cet exercice est à réaliser à la suite de l'Exercice 4.

2) Première étape, création et montage en local sur votre PC de développement.

   a. Dans DockerDesktop, allez dans volume, Create, puis indiquez le nom « MonPremierVolume ».

[3] Pour Installer Debian : « wsl --install -d Debian ». Pour obtenir la liste des distributions Linux disponibles « wsl --list –online ».

b. Dans WSL, après avoir déployé votre application au moins une fois (Exercice 3 et Exercice 4), taper la commande suivante :

    docker run --rm -it --entrypoint sh maciutapi

   Sur le prompt affiché, taper la commande « id ».

   Noter l'ID pour l'utilisateur « app » obtenu (1654 par exemple).

c. Maintenant que l'on connait l'ID de l'utilisateur qui doit aller écrire dans votre nouveau volume, on va aller lui donner les droits. Dans WSL taper la ligne de commande (changer 1654 ci besoin) :

    docker run --rm -v MonPremierVolume:/data alpine chown -R 1654:1654 /data

d. Ensuite, dans VisualStudio (suite de l'exercice 4), dans l'Explorateur de solution, double-cliquez sur le nom de la solution pour les propriétés de construction MSBuild de l'application.

e. Ajouter les lignes suivantes :

```
<PropertyGroup>
    <DockerfileRunArguments>-v MonPremierVolume:/data</DockerfileRunArguments>
</PropertyGroup>
```

f. Votre application peut maintenant écrire dans le dossier /data. Pour le vérifier, à la fin de la méthode ExecuteQuery du fichier SQLLiteConnector.cs, ajouter :

```
using (FileStream stream = File.Create("/data/ecriture.txt"))
{
    using (StreamWriter writer = new StreamWriter(stream))
    {
        writer.WriteLine(query);
    }
}
```

g. Tester en exécutant la requête API : https://localhost:32794/Types/GetAll .

h. Si vous n'obtenez pas d'erreur, c'est tout bon. Vous pouvez également aller voir le fichier en allant dans Docker Desktop, Containers, MacIUT_API, Exec. Et taper « cat /data/ecriture.txt ». Vous devez voir la dernière requête SQL exécutée par votre programme.

3) Maintenant, passons à la création et montage sur le serveur de Docker pour le déploiement.

a. Rendez-vous sur l'interface Portainer qui vous est attribué https://10.128.207.XXX:9443

b. Sélectionnez « Home » puis votre environnement (IQ-SAE-DCK-SXYZ).

c. Cliquez sur volume. Puis « Add volume ». Choisir un nom « MonPremierVolume » et donnez-lui des droits « Restricted ». Cliquez sur « Create the volume ».

d. Petit retour rapide sous Windows et votre WSL. Taper la ligne de commande : « docker save alpine:latest | gzip > alpine.tgz » pour exporter la micro image Linux que l'on a utilisé à l'étape précédente.

e. Dans Portainer, faire Images / Import / Select File et choisir le fichier alpine.tgz.

f. Dans Portainer, faire Containers / Create (à nouveau, on donne les droits à l'utilisation d'ID 1654).

   i. Name : fix-permissions

   ii. Image / Advanced Mode : alpine:latest

   iii. Restricted

   iv. Command / Override : chown -R 1654:1654 /data

   v. Volumes / Map additional volume

      1. Container : /data

      2. Volume : MonPremierVolume.

   vi. Deploy the container. Cela devrait exécuter ce nouveau container et la quitter aussitôt. Vous n'aurez même pas le temps de le voir s'exécuter.

g. Maintenant, lors de la création de votre conteneur MacIUT_API, il faut ajouter la même étape d'ajout du volume. Avant de cliquer sur « Deploy the Container » dans l'Exercice 3, il faut, tout en bas, cliquer sur « Volumes », puis « map additional volume ».

h. Dans « container », indiquer un point de montage (/data par exemple) et dans volume, sélectionner le volume que vous souhaitez utiliser (MonPremierVolume). Laisser bien « Writable ».

i. Enfin, vous pouvez cliquer sur « Deploy the Container ».

j. Et là, de la même façon, https://10.128.207.XXX:8081/Types/GetAll devrait s'exécuter normalement et écrire dans le fichier /data/ecriture.txt. Il faut cliquer sur l'icône « >_ » à côté du nom de votre container pour avoir le shell et aller vérifier.

4) Vous pouvez donc maintenant supprimer votre conteneur principal sans que cela efface votre volume (enfin, si vous **NE** cocher **PAS** la case « Automatically remove non-persistent volumes »).