Containers and cloud

Move database from Atlas Mongo DB to Azure Cosmos DB

- Activate Azure account
- Create an Azure Cosmos account
- Go to the Azure Cosmos DB account page.
- **Create Cosmos DB Database and Collections**.
- Once Database is created, click on **Connection String option** from the left panel. Copy **Primary Connection String**.

- Now in Asp.Net Project open the **appsettings.json** file and set the **connections string** to this file which we already copied from the Azure portal.

```
"ConnectionStrings":
  { "ServerName": "<Primary Connection String>",
    "DatabaseName": "<Database Name> },
```

- In each **Services file** in Project use this commands to reach the **Collection from Cosmos Db**:

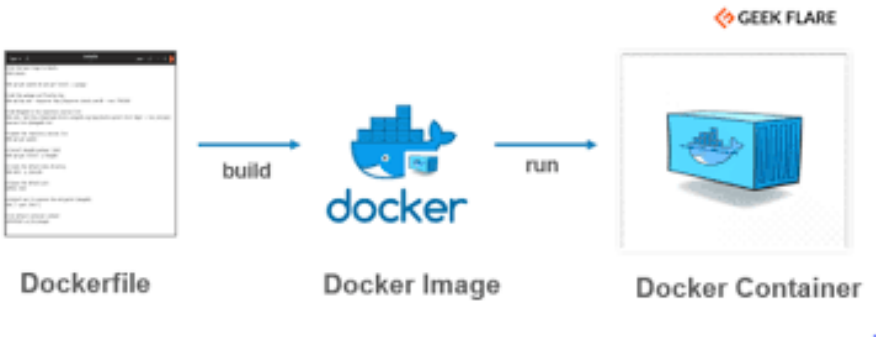
```
private readonly IMongoCollection<Apartments> apartments;
public ApartmentsService(IConfiguration config)
{     MongoClient client = new
                MongoClient(config.GetConnectionString("ServerName"));
      IMongoDatabase database =
        client.GetDatabase(config.GetConnectionString("DatabaseName"));
      apartments = database.GetCollection<Apartments>("Apartments"); }
```

- In each **Model** in Project add another **id for Cosmos DB**:

```
[BsonId]
[BsonRepresentation(BsonType.ObjectId)]
public string objectId { get; set; }
[Key]

public int id { get; set; }
```

Dockerizing the app



- In the Solution Explorer, right-click the App project, point to Add, and select **Docker Support**.
- Make sure that **Linux** is selected and click OK.

Differences between Windows and Linux

- Docker supports only certain versions of Windows (namely, Windows Server 2016 and Windows 10). In contrast, Docker can run on any type of modern Linux-based operating system.
- Even on Windows versions that are supported by Docker, Windows has stricter requirements regarding image compatibility. Read more about those here.
- Some Docker networking features for containers are not yet supported on Windows. They are detailed at the bottom of this page.

Visual Studio creates a Dockerfile for you, which defines how to create the container image for your web app project.

- **Multistage build in Docker File**

The multistage build feature helps make the process of building containers more efficient, and makes containers smaller by allowing them to contain only the bits that your app needs at run time. **Multistage build is used for .NET Core projects, not .NET Framework projects.**

- **Base Stage**:
  - FROM mcr.microsoft.com/dotnet/core/aspnet:2.2-stretch-slim AS base
  - WORKDIR /app
  - EXPOSE 80
  - EXPOSE 443

The lines in the Dockerfile begin with the **Debian image from Microsoft Container** Registry (mcr.microsoft.com) and create an intermediate image base that exposes ports 80 and 443, and sets the working directory to /app.

- **Build Stage**:
  - FROM mcr.microsoft.com/dotnet/core/sdk:2.2-stretch AS build
  - WORKDIR /src
  - COPY ["WebApplication43/WebApplication43.csproj", "WebApplication43/"]
  - RUN dotnet restore "WebApplication43/WebApplication43.csproj"
  - COPY . .
  - WORKDIR "/src/WebApplication43"
  - RUN dotnet build "WebApplication43.csproj" -c Release -o /app

You can see that the build stage starts from a **different original image from the registry** (**sdk rather than aspnet**), rather than continuing from base. The sdk image has all the build tools, and for that reason it's a lot bigger than the aspnet image, which only contains runtime components.

- **Final Stage**:
  - FROM build AS publish
  - RUN dotnet publish "WebApplication43.csproj" -c Release -o /app
  - FROM base AS final
  - WORKDIR /app
  - COPY --from=publish /app .
  - ENTRYPOINT ["dotnet", "WebApplication43.dll"]

The final stage starts again from **base**, and includes the COPY --from=publish to copy the published output to the final image. This process makes it possible for the final image to be a **lot smaller**, since it doesn't need to include all of the build tools that were in the sdk image.

- Now, project will run inside a **Docker container**, rather than in **IIS Express** by clicking the **Docker** button on the toolbar.

Publish to Azure App Service

Now we have created a container image, so we can publish it to Azure App Service Linux

- In the Solution Explorer, right-click the App project and select **Publish**.
- On the Pick a publish target dialog box, click **App Service Linux**, choose **Create New App Service** for Containers, and click **Publish**.
- Fill the forms and select **Create** to start creating the Azure resources.
- Once the wizard completes, it publishes my **container image** to **App Service Linux**.
- Then again select **publish** in publish confirmation page.
- Now in **Summary** I can click on the **URL** to view my new site in my web browser.

https://queuingapp.azurewebsites.net/