SSE Lab 10 - Microservices

Work in Pairs

A current trend in software systems design is to break large systems into sets of smaller services which collaborate to form a complete application. In this lab we will look at how to build and deploy separate small services that communicate with one another through APIs.

Last term we learnt how to consume APIs from services like Spotify, GitHub, Wikipedia etc to read data and use it in our applications (and sometimes to send data to them too). Here we will build one service that publishes a simple API, and a separate one that consumes data from it.

We'll build each service as a Docker image and deploy them in production using Azure Container Instances.

Part 1: Writing the API Service (Completing parts 1-3 will give 50% of the marks)

Create a very simple Flask (or FastAPI if you prefer) app that following provides a route where clients can request data about books, with the data returned in JSON format. Here is some sample data you can use to get started: https://gist.github.com/rchatley/d137a943e905303c5006a0a8fd85b55c - feel free to add more.

To convert a Python dictionary into JSON you can use `jsonify` which is part of Flask.

You should be able to run your Flask app locally and make a request in a browser to get a JSON response. If you like, you could install a browser plugin like JSONView to render the JSON more nicely.

Once you have got your API server running, we can deploy it to Azure. Let's see how to do that.

Part 2: Build

Package up your new app using Docker (as we did last week).

Remember to specify host 0.0.0.0 when starting Flask to allow connections from other machines.

***If you are using a Mac with Apple Silicon**, you will need to specify some extra flags to build a Docker image that runs on Intel/AMD hardware. The new Apple hardware is not compatible with the Intel/AMD hardware that Azure is running in its datacentres. This is a bit annoying, and can cause some mysterious problems, but it's not hard to work around. Use a Docker build command that specifies a target platform, e.g.*

```
$ docker buildx build --platform linux/amd64 -t my-image:latest .
```

Push the image to GHCR. Make it public.

Part 3: Deploy

Instead of creating a virtual machine in Azure ourselves, we'll use a managed service to run our containers. From the Azure portal webpage, find the *Container Instances* dashboard. Click *Create*.

Use your *Azure for Students* subscription. Either re-use a *Resource Group* from before, or create a new one.

Give your container a name (e.g. book-api-server).

Specify that it comes from another registry (as your image is in GHCR).
If you've made your image *public* in GHCR then you won't need any access tokens etc.

Specify your image name - it will probably be something like ghcr.io/your-user/your-image-name

In the *Networking* section:
Give a DNS name label - again this could be something like *book-api-server*
Add a row for the port that your new Flask service runs on (unless you chose 80). The protocol will be TCP.

Now you can Review and Create. Hopefully your app will deploy…

When it's done, you should be able to see it in a browser using its FQDN (fully-qualified domain name). That should look something like http://book-api-server.d6fyech3c6gcb4bn.uksouth.azurecontainer.io:5000/

Part 4: Writing the Client Service (Completing parts 4-5 will give 30% of the marks)

Now let's write another service to consume data from the first one. Create another, separate, Flask app.

Give it an endpoint that when you hit it, makes a request to your first service. Use the `requests` module (or something similar) to retrieve the data (we did this back in the APIs lab last term).

Extract just some of the data (for example, just the third book) and return that to the browser (you could do this as JSON as you did for the first service, as HTML, or just as plain text if you prefer).

Instead of just picking the third book, add a query parameter that allows you to filter the returned books by, for example, genre - so you can do a like query http://book-search.azure.com/books?genre=dystopian

Run your client locally to test it out.

Part 5: Build and Deploy

Build an image, push it to GHCR and deploy to ACI, as you did in parts 2 and 3.

To redeploy your application after making any changes, you should just be able to push a new version of the image to GHCR and then *restart* your container instances in ACI. You should not need to delete and re-create the instances, you just need to trigger Azure to pull a new version from the registry.

Potential Extensions (worth up to 20%):

You might not want to hard-code the URL of your books API in the source code of your client.
Consider specifying it as an Environment Variable. You can set Environment variables when you create the container configuration in ACI.

Sending all the books over the network and filtering them on the client might not be very efficient (especially if we expanded our library to thousands of books). Consider enriching your API to allow some parameters to be passed so that only the relevant data is returned from server to client.

Give your client an HTML front end with a form so the user can type in their search and view their results.

For the curious… (no extra marks for these)

Write a docker-compose file to run your two services together in your development environment.

Try re-writing one of your services in a different language.

What to submit:

Once you have completed the parts above, push the code for your two services to GitHub.

If you want to leave your app running for the markers to look at it should cost around $1/day in total from your Azure for Students budget.

Alternatively, if you want to delete the containers, take some screenshots instead to show what you did.

Submit a PDF file to Scientia containing the following:

- The names and DoC username of the people in your pair
- Links to the GitHub repos for your two services
- Links to GHCR for your two images
- Either links to your live applications, or screenshots showing your app deployed and running.
- A few notes detailing any extensions that you did.