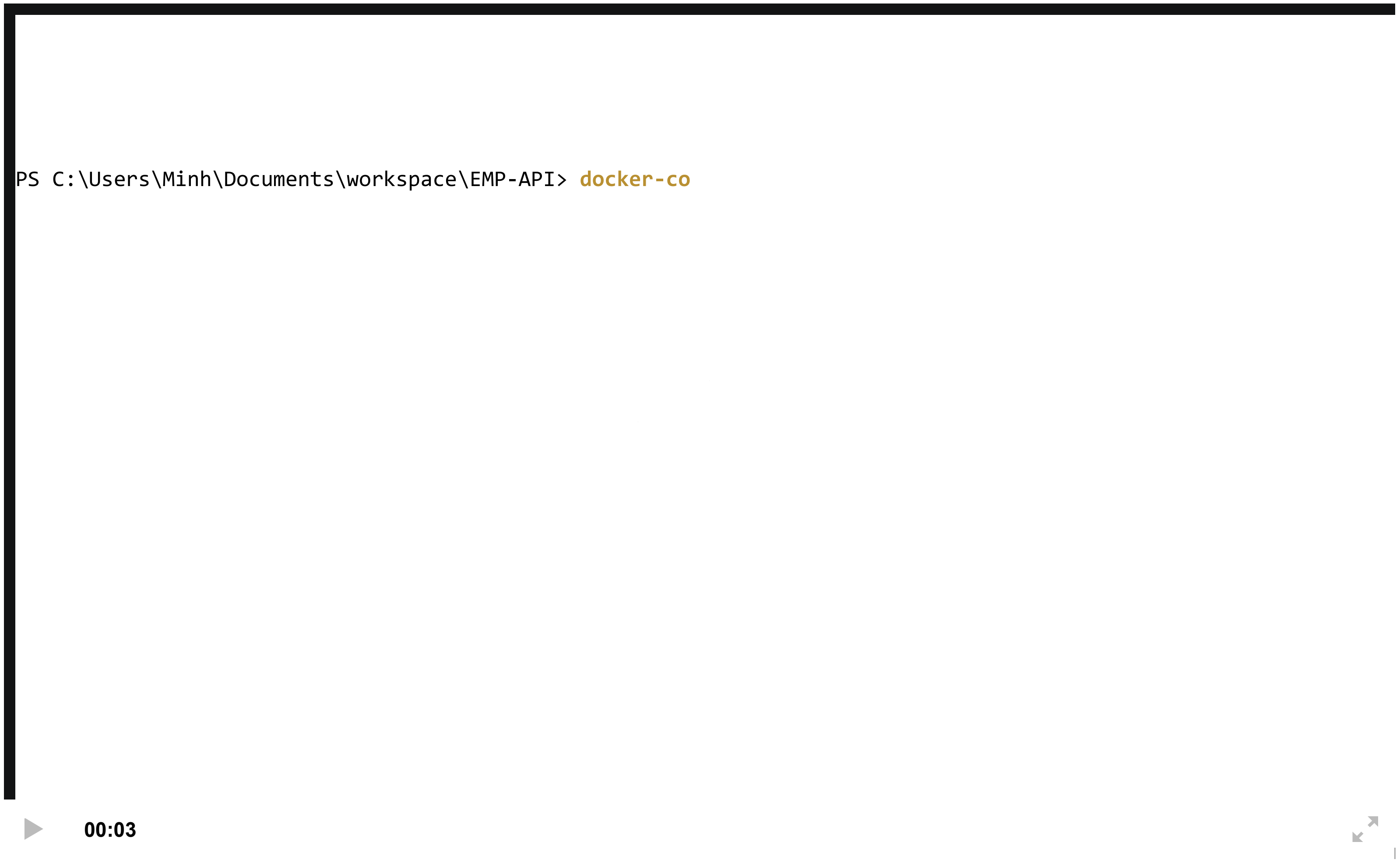
Demo dockerized development environment



```
PS C:\Users\Minh\Documents\workspace\EMP-API> docker-co
```

Table of Contents

From docker to docker-compose

Compose is a tool for defining and running multi-container Docker applications.
Using Compose is basically a three-step process:

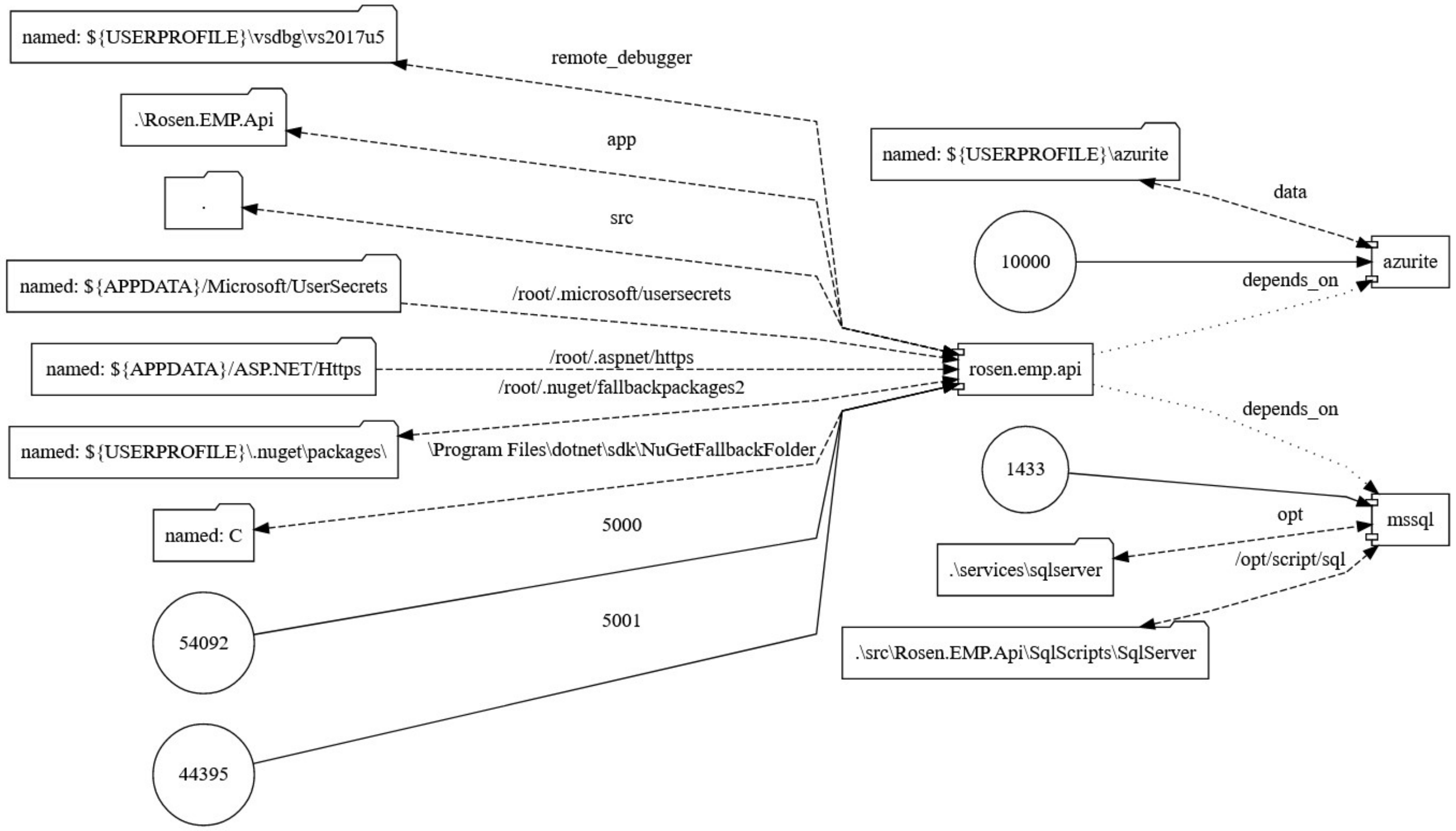
1. Define your app’s environment with a Dockerfile so it can be reproduced anywhere.

2. Define the services that make up your app in docker-compose.yml so they can be run together in an isolated environment.

3. Run docker-compose up and Compose starts and runs your entire app.

Overall architecture of the services

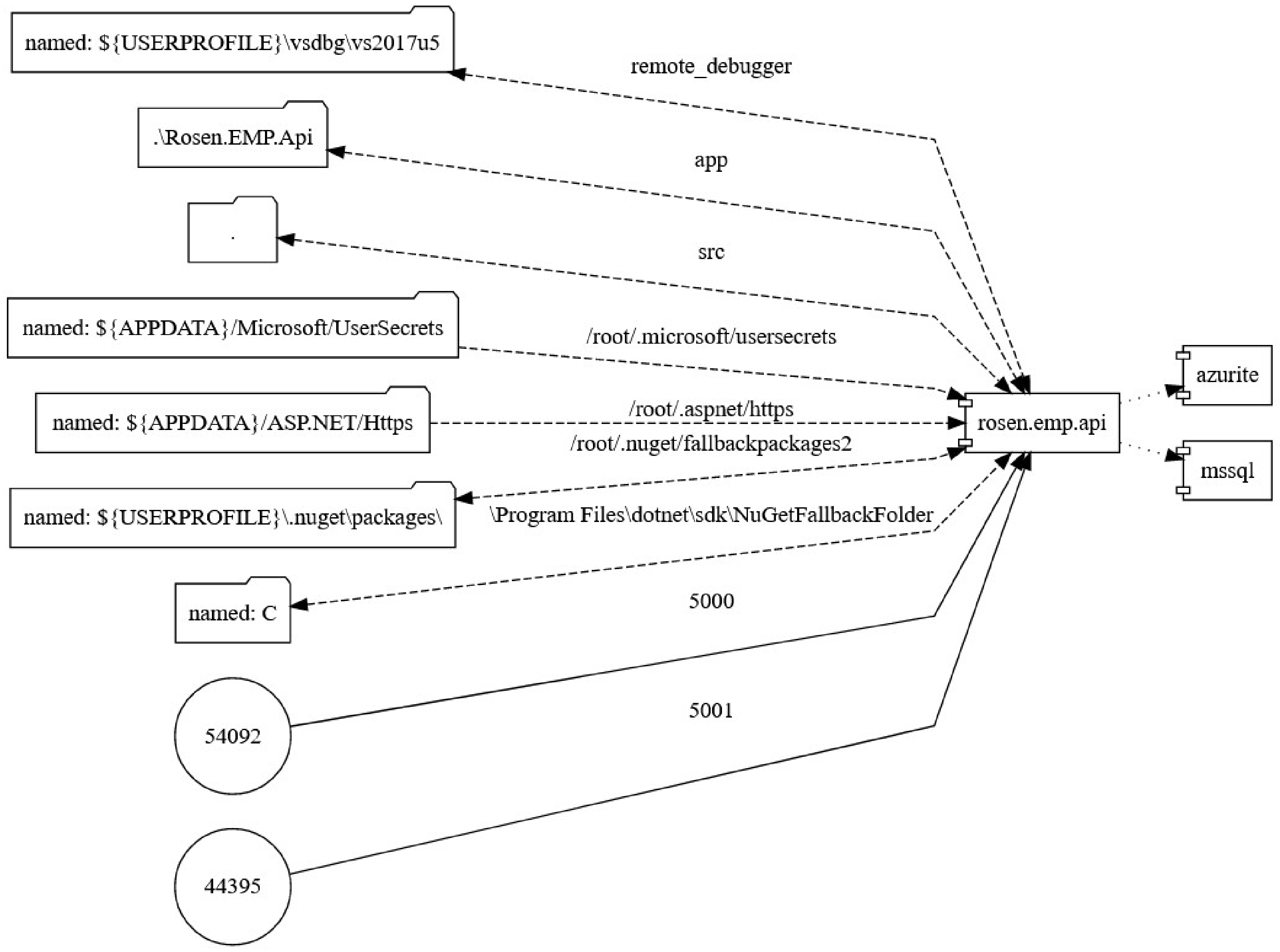
back



named: ${USERPROFILE}\vsdbg\vs2017u5
remote_debugger
.\Rosen.EMP.Api
app
.
src
named: ${APPDATA}/Microsoft/UserSecrets
/root/.microsoft/usersecrets
named: ${APPDATA}/ASP.NET/Https
/root/.aspnet/https
/root/.nuget/fallbackpackages2
named: ${USERPROFILE}\.nuget\packages\
\Program Files\dotnet\sdk\NuGetFallbackFolder
named: C
5000
54092
5001
44395
rosen.emp.api
named: ${USERPROFILE}\azurite
data
10000
azurite
depends_on
depends_on
1433
mssql
opt
.\services\sqlserver
/opt/script/sql
.\src\Rosen.EMP.Api\SqlScripts\SqlServer

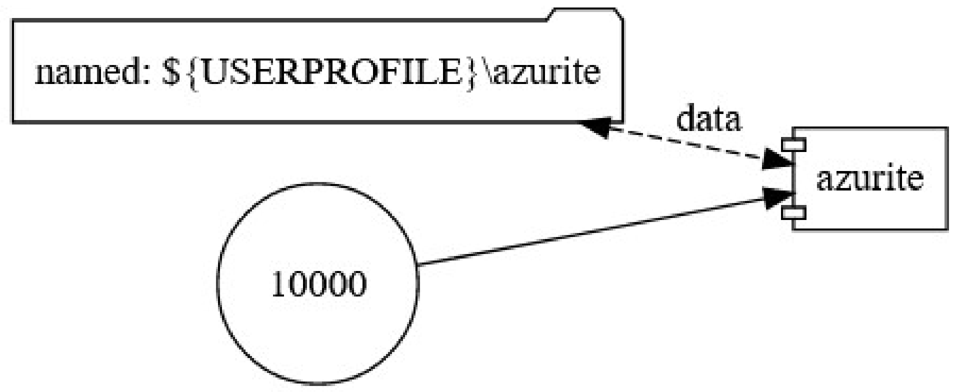
Rosen.EMP.API

back


named: ${USERPROFILE}\vsdbg\vs2017u5
remote_debugger
.\Rosen.EMP.Api
app
.
src
named: ${APPDATA}/Microsoft/UserSecrets
/root/.microsoft/usersecrets
named: ${APPDATA}/ASP.NET/Https
/root/.aspnet/https
/root/.nuget/fallbackpackages2
named: ${USERPROFILE}\.nuget\packages\
\Program Files\dotnet\sdk\NuGetFallbackFolder
named: C
5000
5001
54092
44395
rosen.emp.api
azurite
mssql

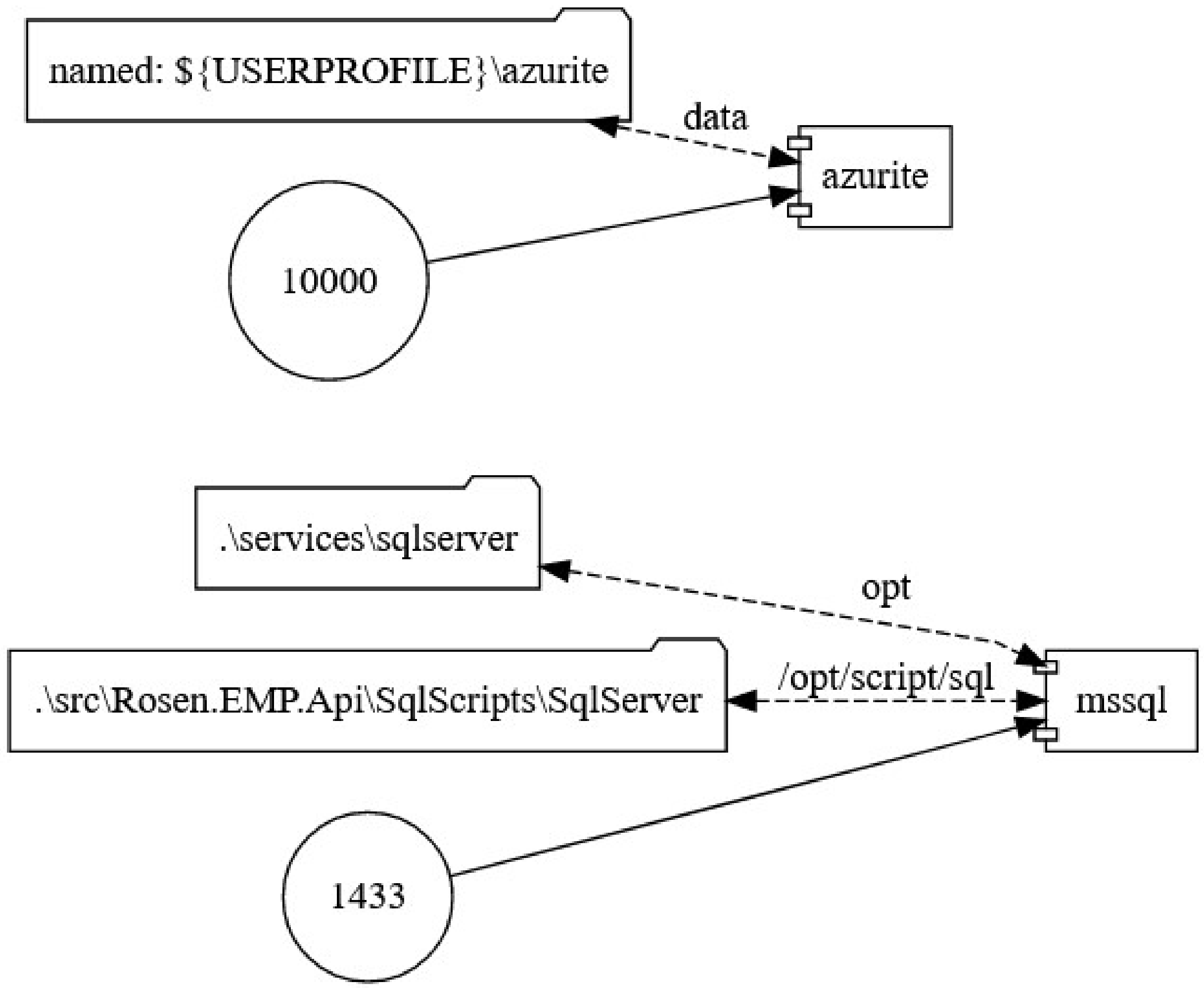
Azurite - Azure storage emulator

back

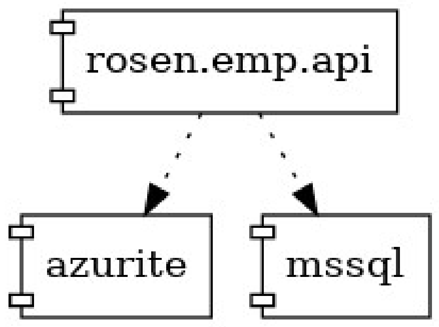


MSSQL - SQL database

back


named: ${USERPROFILE}\azurite
data
azurite
10000
.\services\sqlserver
opt
.\src\Rosen.EMP.Api\SqlScripts\SqlServer
/opt/script/sql
mssql
1433

Expressing connection between services



By default Compose sets up a single network for your app.

Each container for a service joins the default network and is both reachable by other containers on that network, and discoverable by them at a hostname identical to the container name.

Accessing other services

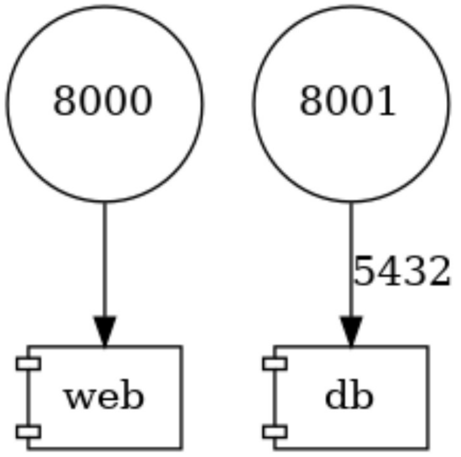
back

For example, suppose your app is in a directory called myapp, and your docker-compose.yml looks like this:

```
version: "3"
services:
  web:
    build: .
    ports:
      - "8000:8000"
  db:
    image: postgres
    ports:
      - "8001:5432"
```

Accessing other services

When you run docker-compose up, the following happens:



A network called myapp_default is created.

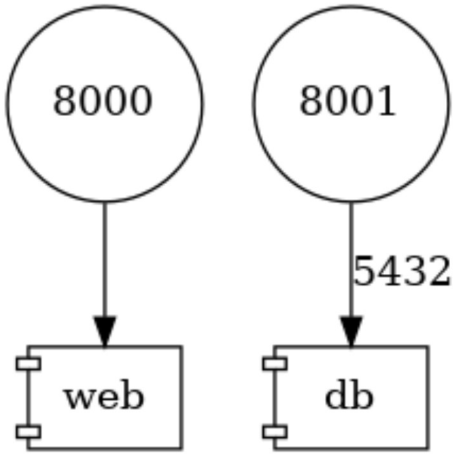
A container is created using web’s configuration.

It joins the network myapp_default under the name web.

A container is created using db’s configuration.

It joins the network myapp_default under the name db.

Accessing other services



Each container can now look up the hostname web or db and get back the appropriate container’s IP address.

web’s application code could connect to the URL postgres://db:5432 and start using the Postgres database.using the Postgres database.

Guideline for developer

- SQL connection string
- Azure storage connection string
- Create a https dev-cert
- Config Kestrel to use the created cert

SQL connection string:

mssql is the name of the MSSQL service's name defined in the docker-compose.yml

1433 is the port configured for the MSSQL service in the docker-compose.override.yml

```
"DbConnectionStringLocalDev.Docker": "Server=mssql,1433\\Catalog=EMPTEST;Database=EMPTE
```

Azure storage connection string:

azurite is the name of the Azurite service's name defined in the docker-compose.yml

10000 is the port configured for the Azurite service in the docker-compose.override.yml

```
    "SAConnectionStringDev": "BlobEndpoint=http://azurite:10000/devstoreaccount1;AccountNam
```

Create a dev-certs using dev-certs tool and use it for the API service's https

generate a dev-cert

```
dotnet dev-certs https --clean
dotnet dev-certs https -ep %APPDATA%\ASP.NET\Https\aspnetapp.pfx -p "password"
dotnet dev-certs https --trust
```

Create a dev-certs using dev-certs tool and use it for the API service's https

config the kestrel server to use the generated dev-cert

```
   "Kestrel": {
   "Endpoints": {
     "Http": {
       "Url": "http://0.0.0.0:5000"
     },
     "HttpsInlineCertFile": {
       "Url": "https://0.0.0.0:5001",
       "Certificate": {
         "Path": "/root/.aspnet/https/aspnetapp.pfx",
         "Password": "password"
       }
     }
   }
 }
```

DEMO Time!

QA Time!

The End