Continuous Integration (CI) and Continuous Delivery (CD) #1

Majid Babaei


REVIEW

Create a compose file

docker-compose.yml

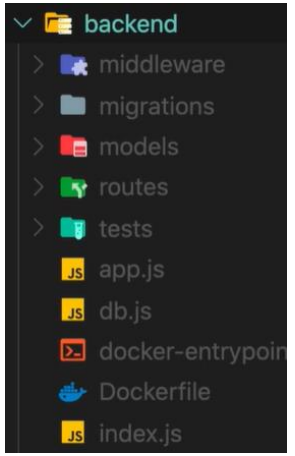
```yaml
version: "3.8"
services:
  web:
    build: ./frontend
    ports:
      - 3000:3000
  api:
    build: ./backend
    ports:
      - 3001:3001
    environment:
      - DB_URL="mongodb://db/movieList"
  db:
    image: mongo:4.0-xenial
    ports:
      - 27017:27017
    volumes:
      - movieList:data/db
volumes:
  movieList:
```

```
docker run -v $(pwd):/app
```

```
volumes:
  - ./backend:/app
```

```
sh: nodemon: not found
npm ERR! code ELIFECYCLE
npm ERR! syscall spawn
npm ERR! file sh
```

```
"devDependencies": {
  "jest": "^26.6.3",
  "migrate-mongo": "^8.1.4",
  "nodemon": "^2.0.7",
  "supertest": "^6.1.3"
},
```



How can we publish our application changes using docker-compose?

```
api_1  | [nodemon] starting `node index.js`
db_1   | 2021-03-17T22:42:11.000+0000 I SHA
api_1  | Server started on port 3001...
```

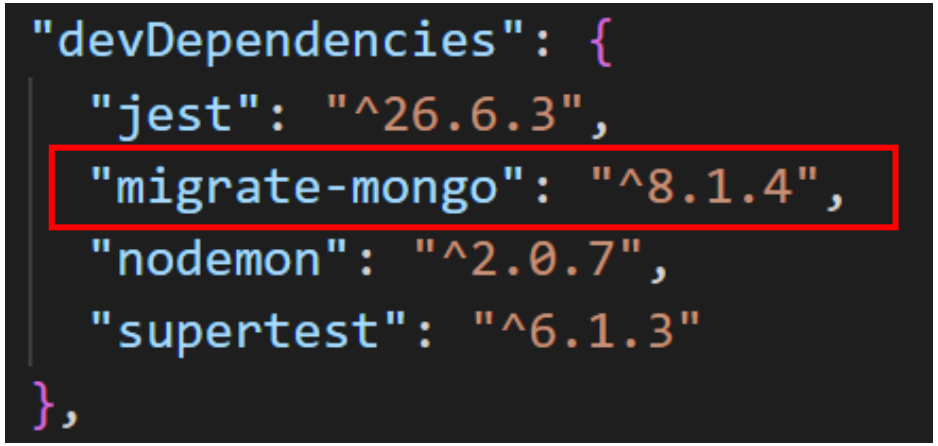
Database Migration

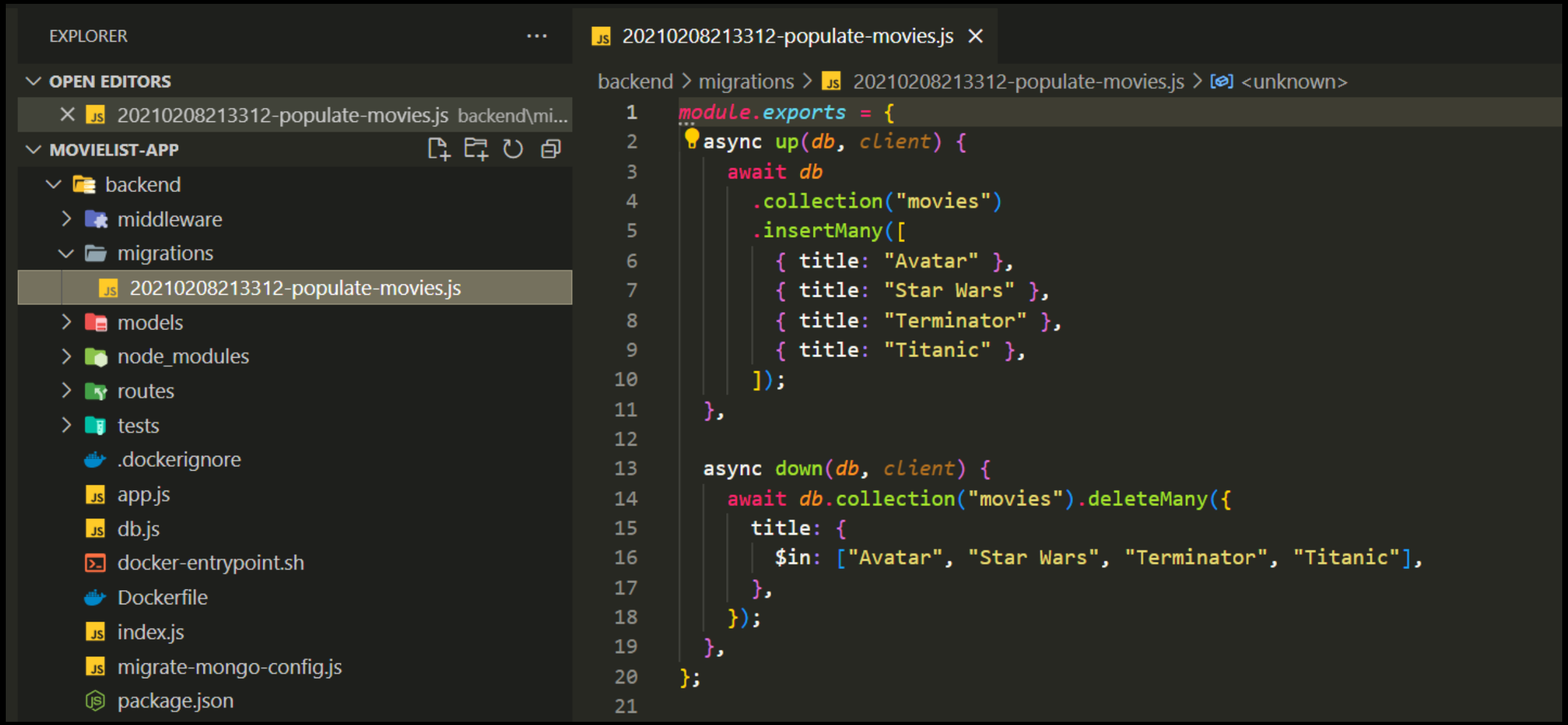
- Most of the time when we release our application, we want our database to be in a particular shape with some data. It is called **database migration**.

- In this project we will use a tool *migrate-mongo*

- Using this tool, you can create *db migration script*

- It is a basic JS file with two function:
  - Up: for upgrading the db
    - We have collections
    - We insert objects into collections
  - Down: for downgrading the db
    - We remove some objects




EXPLORER
OPEN EDITORS
20210208213312-populate-movies.js backend\mi...
MOVIELIST-APP
backend
middleware
migrations
20210208213312-populate-movies.js
models
node_modules
routes
tests
.dockerignore
app.js
db.js
docker-entrypoint.sh
Dockerfile
index.js
migrate-mongo-config.js
package.json
20210208213312-populate-movies.js
backend > migrations > 20210208213312-populate-movies.js > <unknown>
module.exports = {
async up(db, client) {
await db
.collection("movies")
.insertMany([
{ title: "Avatar" },
{ title: "Star Wars" },
{ title: "Terminator" },
{ title: "Titanic" },
]);
},
async down(db, client) {
await db.collection("movies").deleteMany({
title: {
$in: ["Avatar", "Star Wars", "Terminator", "Titanic"],
},
});
},
};

Database Migration

- To do this process manually, you can use terminal: migrate-mogo up
- Look into the migration directory and execute all migration scripts
- What if we execute this command multiple times?!
  - With the records on the changelog, it will avoid duplicate data in db
- We should run this command as part of our package development
- How can we execute this script as part of starting our application?
- We need to include it in our dockerfile!
- Since we have multiple dockerfiles,
  we should manage this in *docker-compose*

```
"scripts": {
  "db:up": "migrate-mongo up",
  "start": "nodemon --ignore './tests' index.js",
  "test": "jest --watchAll --colors"
},
```

Database Migration

- Recall, we have CMD in docker file to execute a default starting commands
- With *command* in docker-compose file we can overwrite what we defined in CMD
- Let's add the *command* in our docker-compose
- Do you see any issues with this script?
- It is possible that our database server is not ready at the time of executing migrate-mongo up command
- We need to wait until the db engine is ready, i.e., its port is start listening
- Starting a db engine takes several seconds!

```
api:
  build: ./backend
  ports:
    - 3001:3001
  environment:
    - DB_URL="mongodb://db/movieList"
  volumes:
    - ./backend:/app/backend
  command: migrate-mongo up && npm start
```

Database Migration

- We need to use a *waiting script*; you can find many on Internet!
- Search for *docker wait for container*
- There are a few tools out there that provide easy to use waiting logic, for simple tcp port checks:
  - wait-for-it
  - dockerize
- Download the scrip and use it in your app

```yaml
api:
  build: ./backend
  ports:
    - 3001:3001
  environment:
    - DB_URL="mongodb://db/movieList"
  volumes:
    - ./backend:/app/backend
  command: ./wait-for-it db:27017 && migrate-mongo up && npm start
```

- The command can be very long and difficult to interpret!

```yaml
command: ./docker-entrypoint.sh
```

- We can create a shell script file called *docer-entrypoint.sh* and use it here

backend > docker-entrypoint.sh

```
1   #!/bin/sh
2
3   echo "Waiting for MongoDB to start..."
4   ./wait-for db:27017
5
6   echo "Migrating the databse..."
7   npm run db:up
8
9   echo "Starting the server..."
10  npm start
```

- Let's create a clear stage for our app before we run it!

- First, we remove all containers

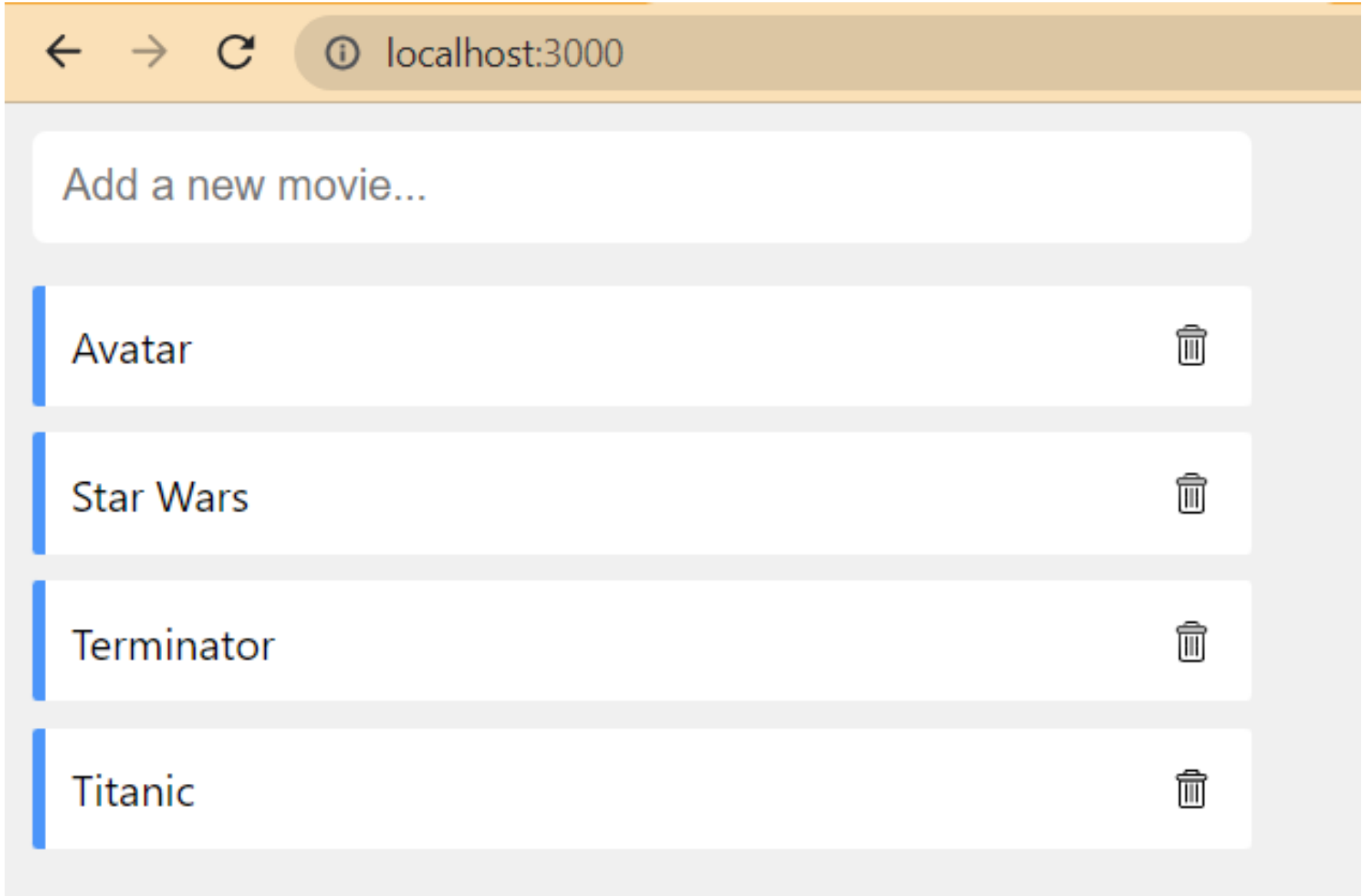
```
er\movieList-app\frontend> docker-compose down
[+] Running 4/4
 ✓ Container movielist-app-web-1  Removed
 ✓ Container movielist-app-api-1  Removed
 ✓ Container movielist-app-db-1   Removed
 ✓ Network movielist-app_default  Removed
```

- Then, remove the volume

```
er\movieList-app\frontend> docker volume ls
DRIVER    VOLUME NAME
local     0acf7e067ce39d3a22396065f921e60f877134eda924f25f5bb2501a866101c0
local     8751cf84768b4087dcd977ac872822ae65231513d0a7b8ee3486e9576ada874c
local     app-data
local     f0aef733f6f115302ff5d7a0b2b995c7df1dbd127d38814871d886fe5253437f
local     movielist-app_movieList
```

```
er\movieList-app\frontend> docker volume rm movielist-app_movieList
movielist-app_movieList
```

- Finally, run all the containers: docker-compose up


localhost:3000
Add a new movie...
Avatar
Star Wars
Terminator
Titanic

Deploy a multi-container Application on Microsoft Azure: Cloud Computing Services

Everything works fine locally, let's bring it down!

- Make sure that everything is ok on your local machine
- Stop all services [Ctrl + c]
- Remove containers [docker-compose down]
- Remove images [image rm -f $(docker image ls -aq)], because we need to rename them later!
- Remove the volume [docker volume rm movielist-app_movielist]

```
$ docker-compose down; docker image rm -f $(docker image ls -aq); docker rm -f $(docker container ls -aq); docker volume rm movielist-app_movielist
```

Defining production configuration

- The current compose file is great for development but not ideal for production!
- In production environment you don’t need the volume mapping we did to share our source code with the container.
- We don’t need a testing services because it is a waste of resources in production. Because you don’t change the code in production!
- We need to create a new compose file only for production!
- Ports should be changed to have a better access to the service in production. E.g., port 3000 is not what we want in production!

- The name of the images changed to be publishable on the Azure Registry
- Port 3000 changed to 80 to make frontend accessible via the default port

What will happen if any of these services goes down in production?

docker-compose.prod.yml

```yaml
version: "3.8"

services:
  web:
    image: regecse437.azurecr.io/movielist-app-frontend:v1
    build: ./frontend
    ports:
      - 80:3000

  api:
    image: regecse437.azurecr.io/movielist-app-backend:v1
    build: ./backend
    ports:
      - 3001:3001
    environment:
      DB_URL: mongodb://db/movielist
    command: ./docker-entrypoint.sh

  db:
    image: mongo:4.0-xenial
    ports:
      - 27017:27017
    volumes:
      - movielist:/data/db
volumes:
  movielist:
```

Docker Compose Restart Policies

- Restart policies are strategies we can use to restart Docker containers automatically and manage their lifecycles.
- Given that containers can fail unexpectedly, Docker has safeguards to prevent services from running into a restart loop.
- In case of a failure, restart policies don't take effect unless the container runs successfully for at least 10 seconds.
- We can also assume that manually stopping a container will make Docker restart the service automatically when a restart policy is provided.
- However, these policies are dismissed in this case to prevent containers from restarting after being stopped arbitrarily.

Docker Compose Restart Policies

- To use restart policies, Docker provides the following options:
  - *no*: Containers won't restart automatically.
  - *on-failure[:max-retries]*: Restart the container if it exits with a non-zero exit code, and provide a maximum number of attempts for the Docker daemon to restart the container.
  - *always*: Always restart the container if it stops.
  - *unless-stopped*: Always restart the container unless it was stopped arbitrarily, or by the Docker daemon.

```
docker run --restart always my-service
```

The previous example demonstrates how the restart flag configures the strategy to restart a single container automatically. However, *Docker Compose* allows us to configure restart policies to manage multiple containers by using the restart or restart_policy properties.

profiles

```
profiles: ["frontend", "debug"]
profiles:
  - frontend
  - debug
```

`profiles` defines a list of named profiles for the service to be enabled under. When not set, the service is *always* enabled. For the services that make up your core application you should omit `profiles` so they will always be started.

Valid profile names follow the regex format `[a-zA-Z0-9][a-zA-Z0-9_.-]+`.

See also *Using profiles with Compose* to learn more about profiles.

restart

`no` is the default restart policy, and it does not restart a container under any circumstance. When `always` is specified, the container always restarts. The `on-failure` policy restarts a container if the exit code indicates an on-failure error. `unless-stopped` always restarts a container, except when the container is stopped (manually or otherwise).

```
restart: "no"
restart: always
restart: on-failure
restart: unless-stopped
```

docker-compose.prod.yml

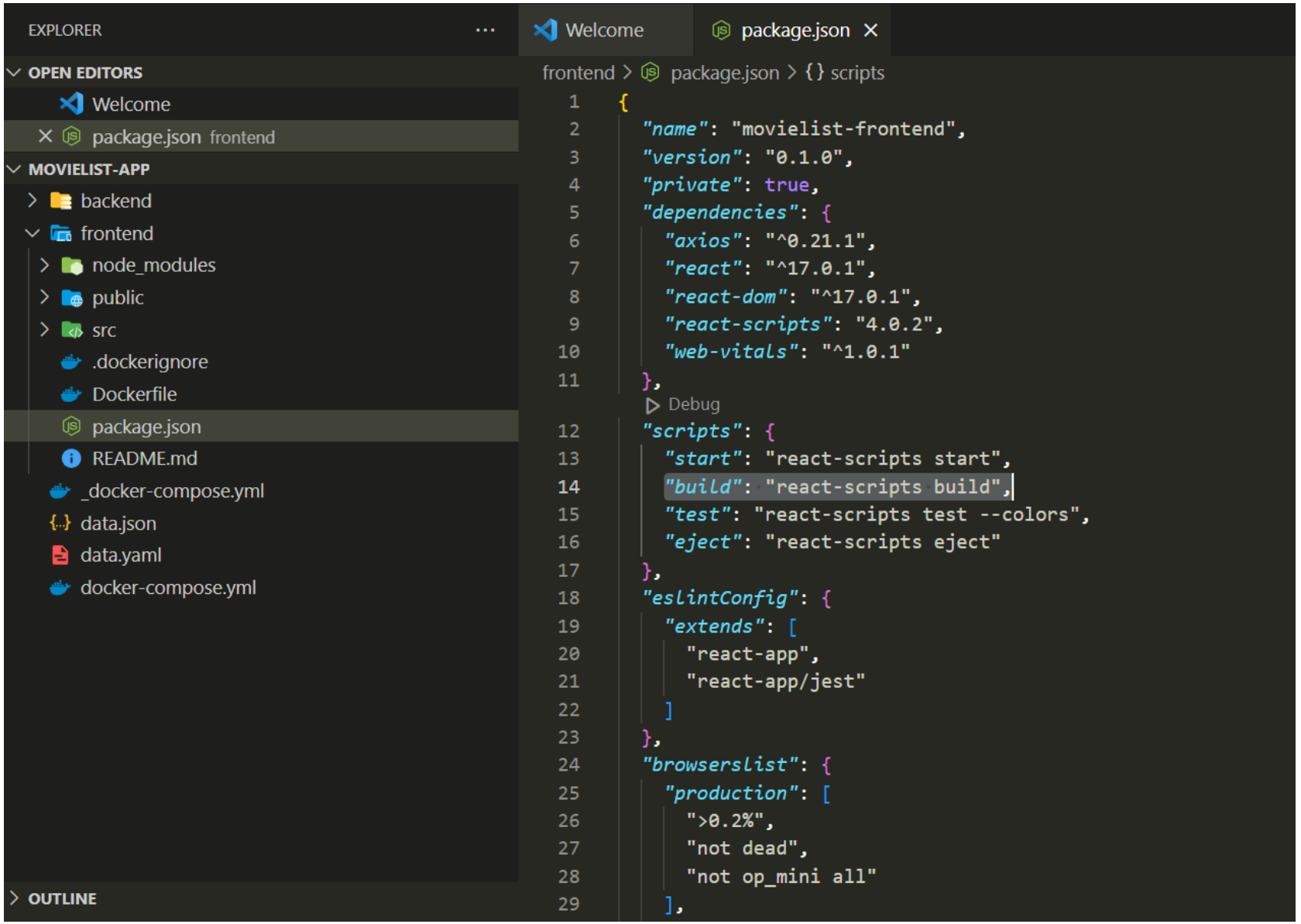
```yaml
version: "3.8"
services:
  web:
    image: regecse437.azurecr.io/movielist-app-frontend:v1
    build: ./frontend
    ports:
      - 80:3000
    restart: unless-stopped
  api:
    image: regecse437.azurecr.io/movielist-app-backend:v1
    build: ./backend
    ports:
      - 3001:3001
    environment:
      DB_URL: mongodb://db/movielist
    command: ./docker-entrypoint.sh
    restart: unless-stopped
  db:
    image: mongo:4.0-xenial
    ports:
      - 27017:27017
    volumes:
      - movielist:/data/db
    restart: unless-stopped
volumes:
  movielist:
```

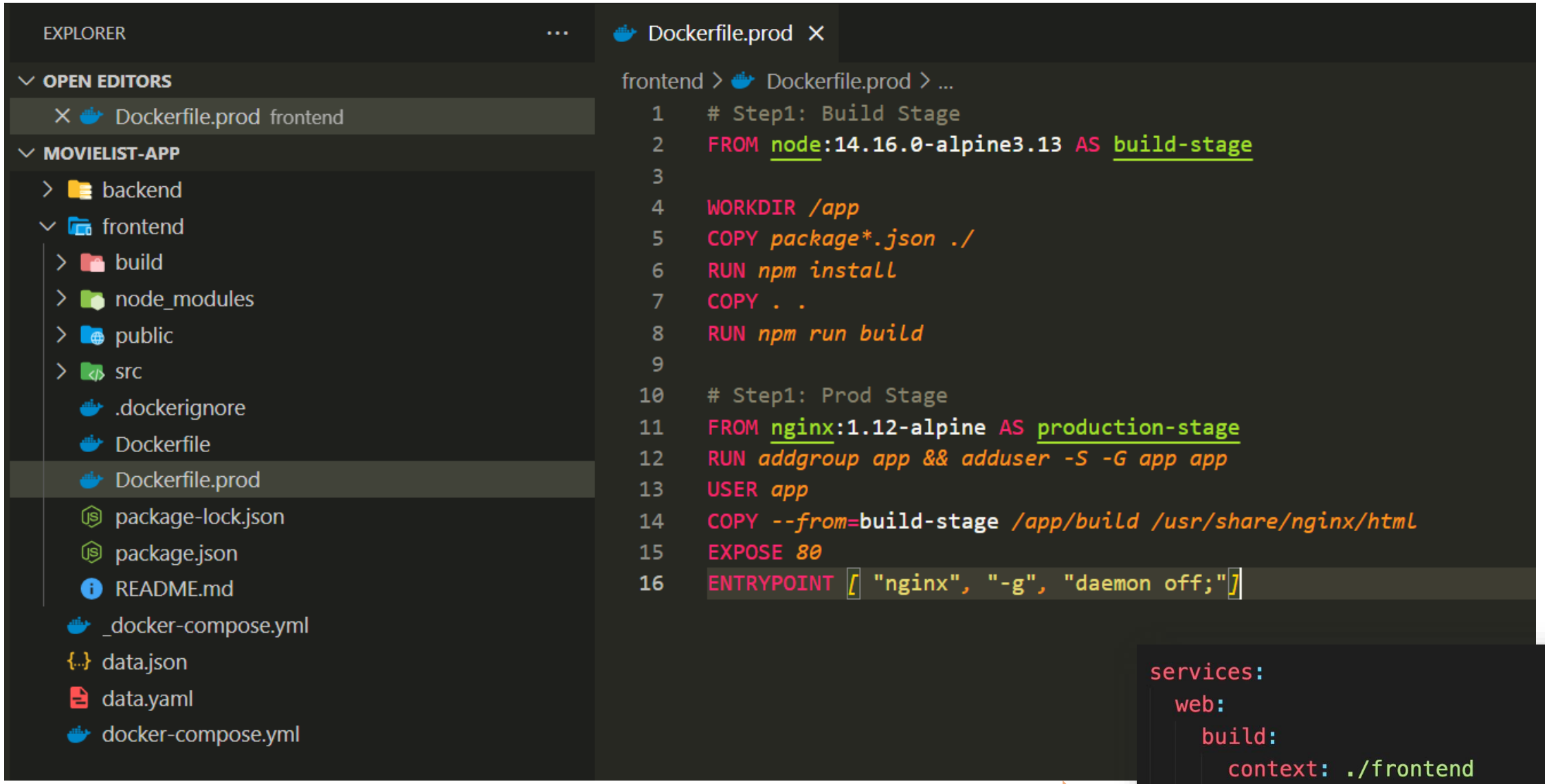
```
C:\Users\Toronto>docker images
REPOSITORY                  TAG       IMAGE ID       CREATED          SIZE
movielist-app-frontend      latest    befe655dda33   7 days ago       466MB
movielist-app-backend       latest    03663e74262f   7 days ago       252MB
```

We use *build* command to create optimized assets for production!

Go to the frontend dir
And run npm run build

This is going to create what we need in prod!





docker-compose.prod.yml

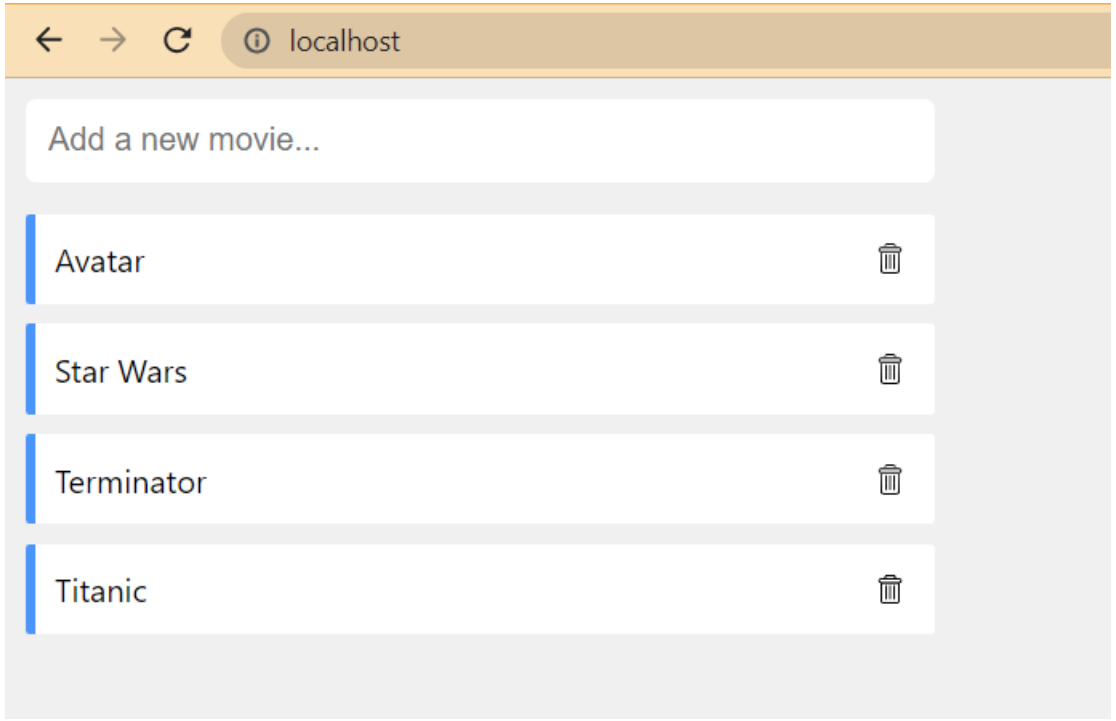
```yaml
services:
  web:
    build:
      context: ./frontend
      dockerfile: Dockerfile.prod
    ports:
      - 80:80
    restart: unless-stopped
```

frontend > Dockerfile > ...

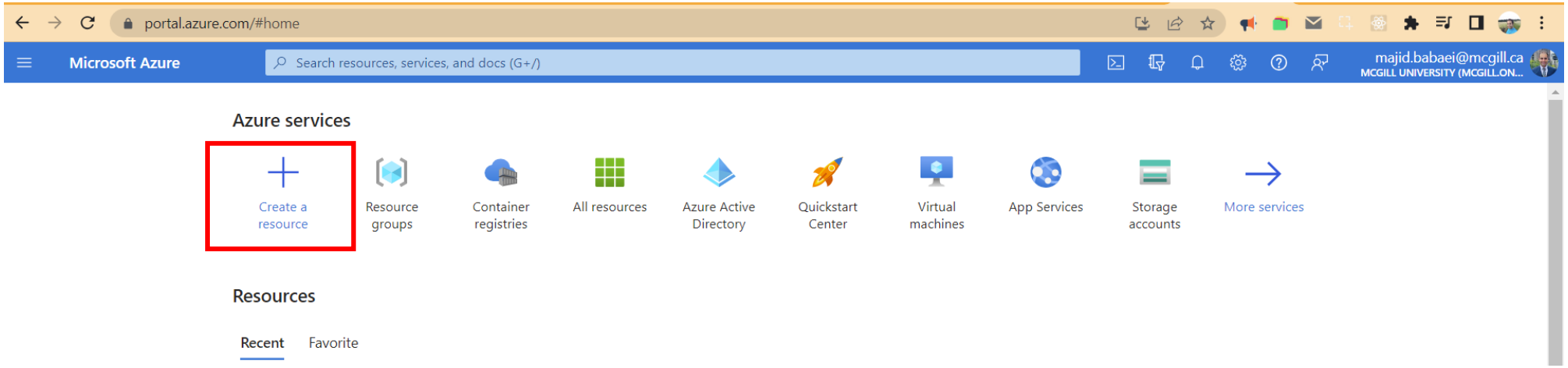
```dockerfile
FROM node:14.16.0-alpine3.13

RUN addgroup app && adduser -S -G app app
USER app

WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .

EXPOSE 3000

CMD ["npm", "start"]
```

Try the production configuration

- You can run the production file: docker-compose -f .\docker-compose.prod.yml up
- You will see the result on port 80



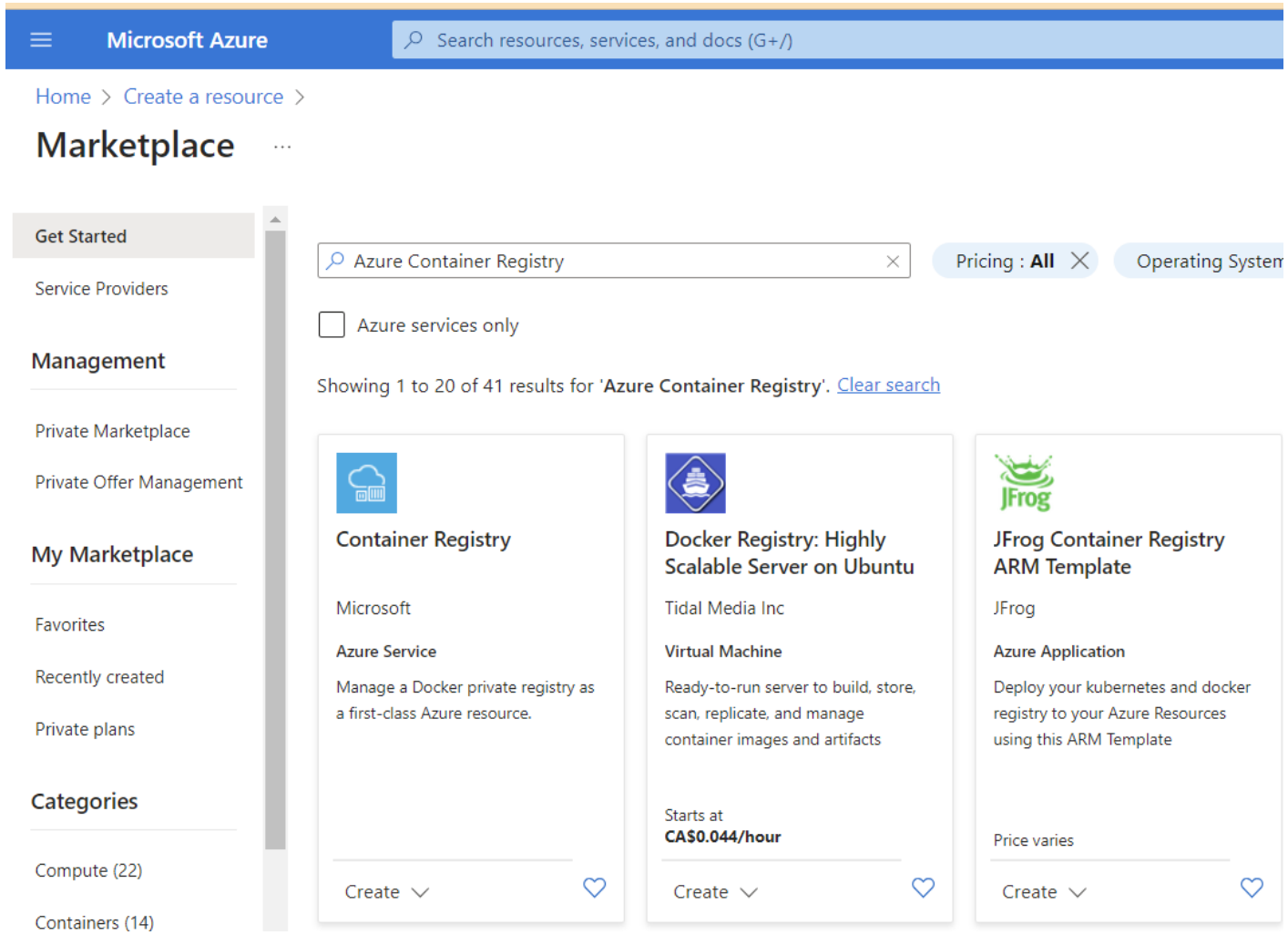
- Bring the App down! docker-compose down

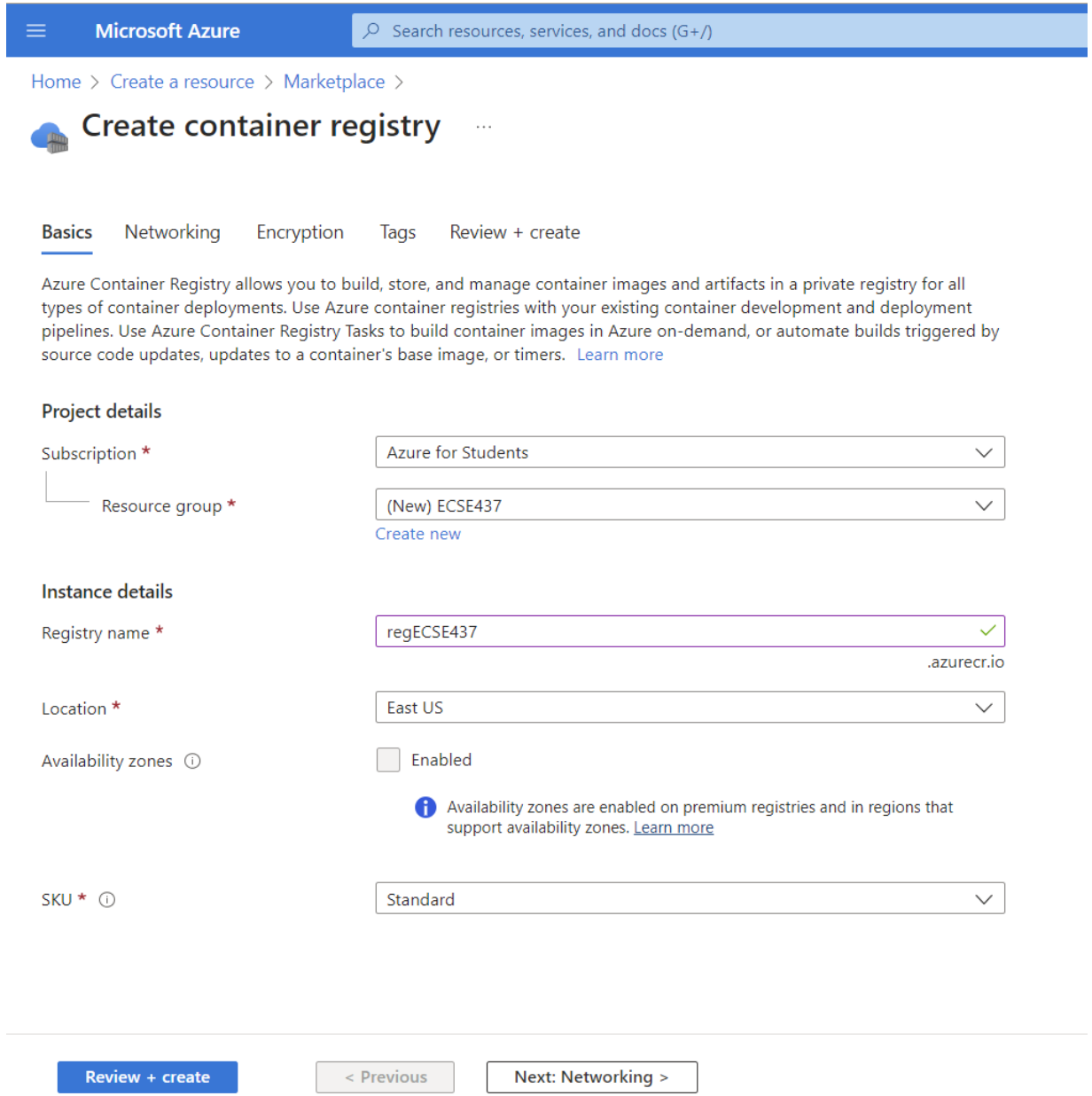


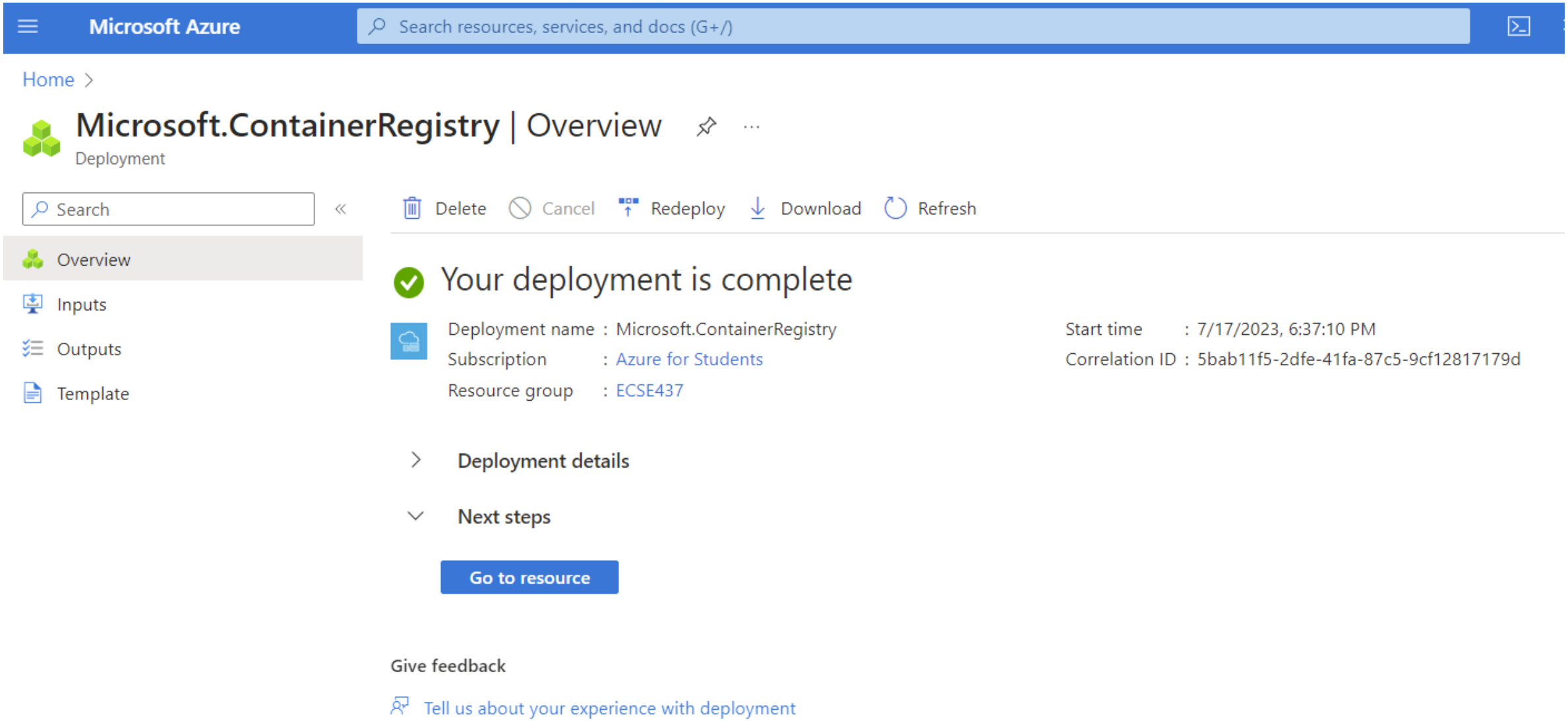
https://portal.azure.com

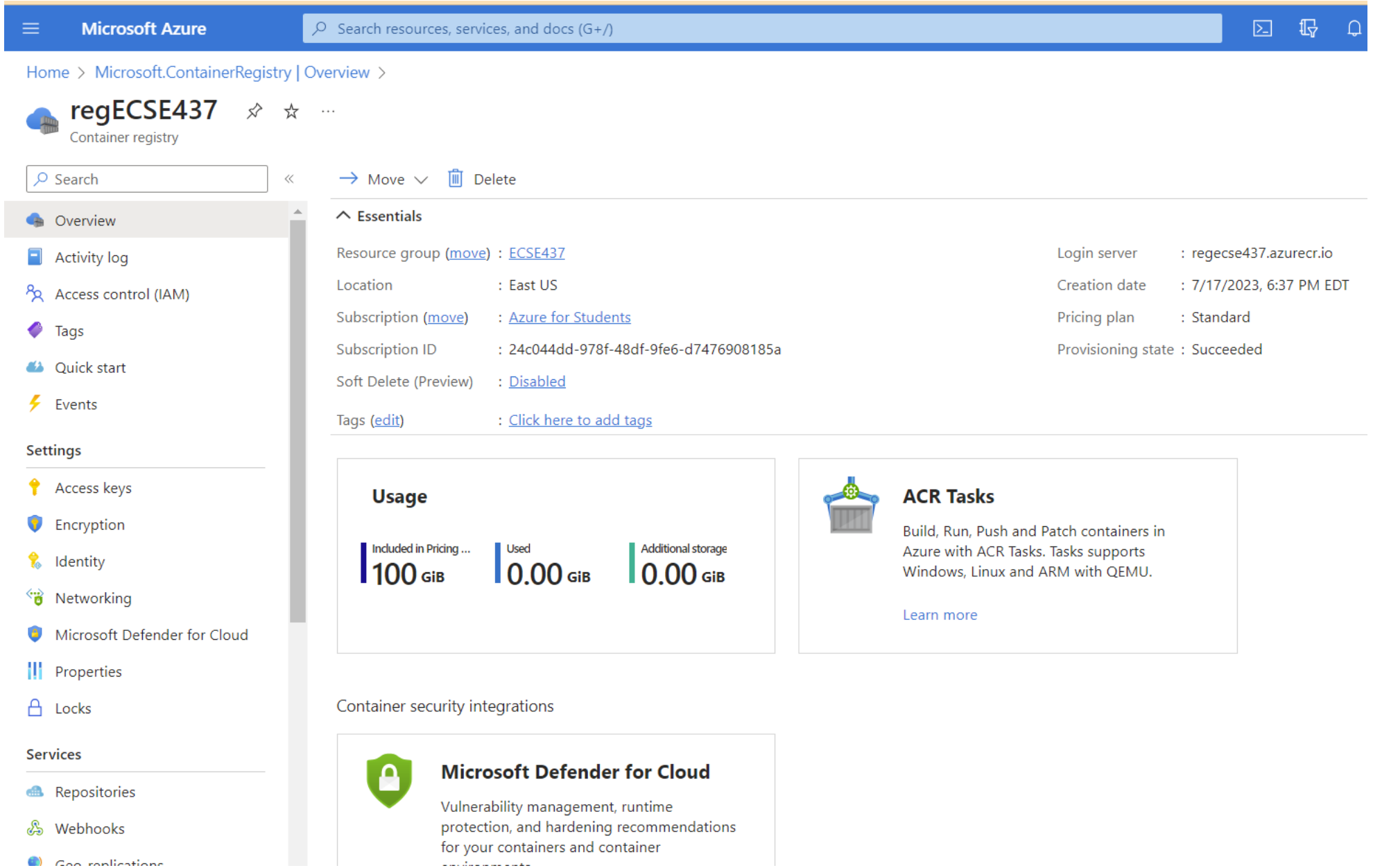
Create Azure Container Registry

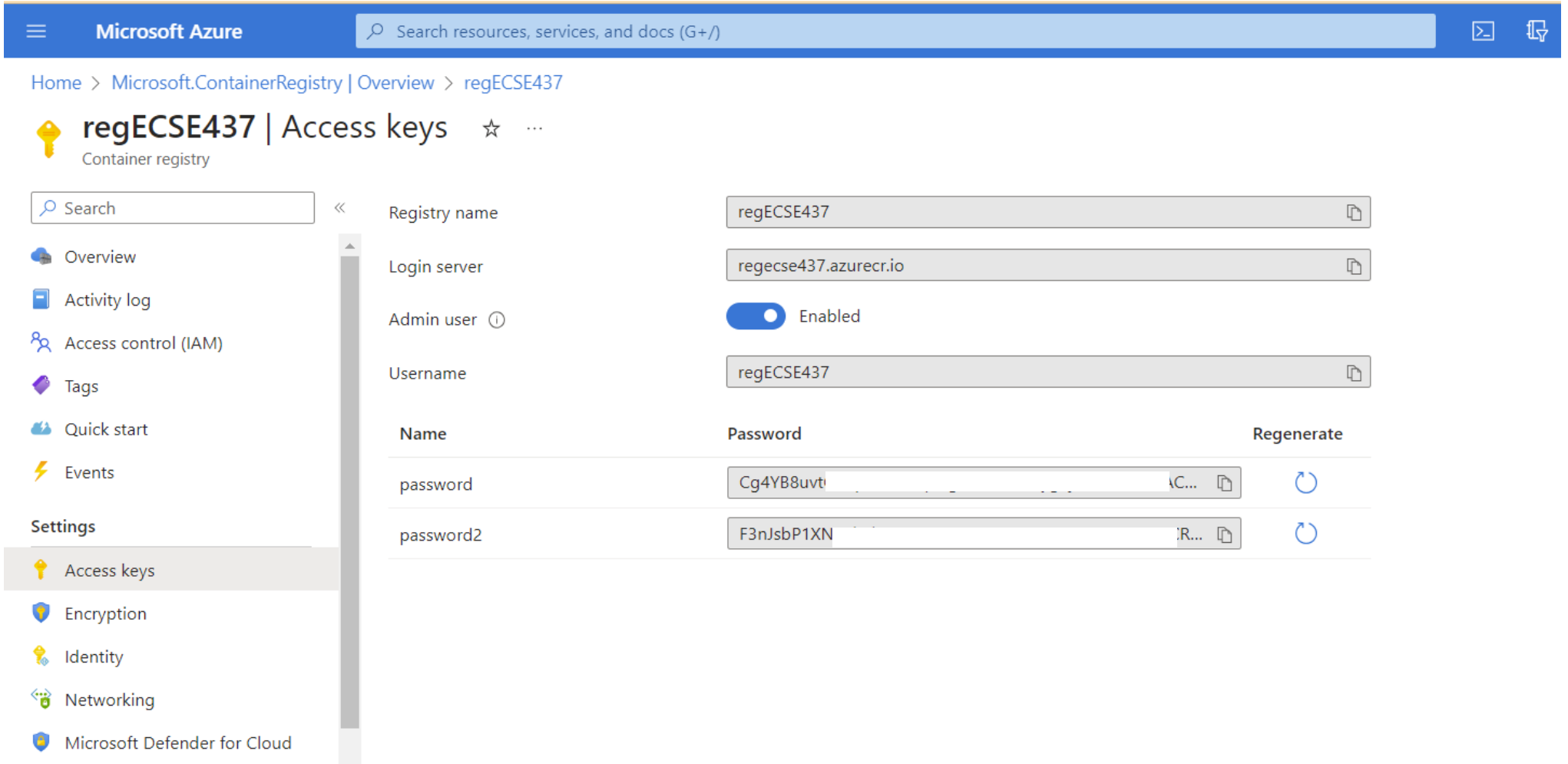
- We can build, store, secure, scan, replicate, and manage container images and artifacts.
- Go to this https://portal.azure.com/#create/hub
- Then search for: *Azure Container Registry*




Microsoft Azure
Search resources, services, and docs (G+/)
Home > Create a resource > Marketplace >
Create container registry
Basics
Networking
Encryption
Tags
Review + create
Azure Container Registry allows you to build, store, and manage container images and artifacts in a private registry for all types of container deployments. Use Azure container registries with your existing container development and deployment pipelines. Use Azure Container Registry Tasks to build container images in Azure on-demand, or automate builds triggered by source code updates, updates to a container's base image, or timers. Learn more
Project details
Subscription *
Azure for Students
Resource group *
(New) ECSE437
Create new
Instance details
Registry name *
regECSE437
.azurecr.io
Location *
East US
Availability zones
Enabled
Availability zones are enabled on premium registries and in regions that support availability zones. Learn more
SKU *
Standard
Review + create
< Previous
Next: Networking >


Microsoft Azure
Search resources, services, and docs (G+/)
Home
Microsoft.ContainerRegistry | Overview
Deployment
Search
Overview
Inputs
Outputs
Template
Delete
Cancel
Redeploy
Download
Refresh
Your deployment is complete
Deployment name : Microsoft.ContainerRegistry
Subscription : Azure for Students
Resource group : ECSE437
Start time : 7/17/2023, 6:37:10 PM
Correlation ID : 5bab11f5-2dfe-41fa-87c5-9cf12817179d
Deployment details
Next steps
Go to resource
Give feedback
Tell us about your experience with deployment


Microsoft Azure
Search resources, services, and docs (G+/)
Home > Microsoft.ContainerRegistry | Overview >
regECSE437
Container registry
Search
Overview
Activity log
Access control (IAM)
Tags
Quick start
Events
Settings
Access keys
Encryption
Identity
Networking
Microsoft Defender for Cloud
Properties
Locks
Services
Repositories
Webhooks
Move
Delete
Essentials
Resource group (move) : ECSE437
Location : East US
Subscription (move) : Azure for Students
Subscription ID : 24c044dd-978f-48df-9fe6-d7476908185a
Soft Delete (Preview) : Disabled
Tags (edit) : Click here to add tags
Login server : regecse437.azurecr.io
Creation date : 7/17/2023, 6:37 PM EDT
Pricing plan : Standard
Provisioning state : Succeeded
Usage
Included in Pricing ...
100 GiB
Used
0.00 GiB
Additional storage
0.00 GiB
ACR Tasks
Build, Run, Push and Patch containers in Azure with ACR Tasks. Tasks supports Windows, Linux and ARM with QEMU.
Learn more
Container security integrations
Microsoft Defender for Cloud
Vulnerability management, runtime protection, and hardening recommendations for your containers and container




```
PS C:\Windows\System32> docker login -u regECSE437 -p Cg4YB8u                                  TN regecse437.azurecr.io
WARNING! Using --password via the CLI is insecure. Use --password-stdin.
Login Succeeded
```

Push images to Azure Registry

- Run docker-compose push
  - This will push all the images we have built (i.e., frontend and backend)

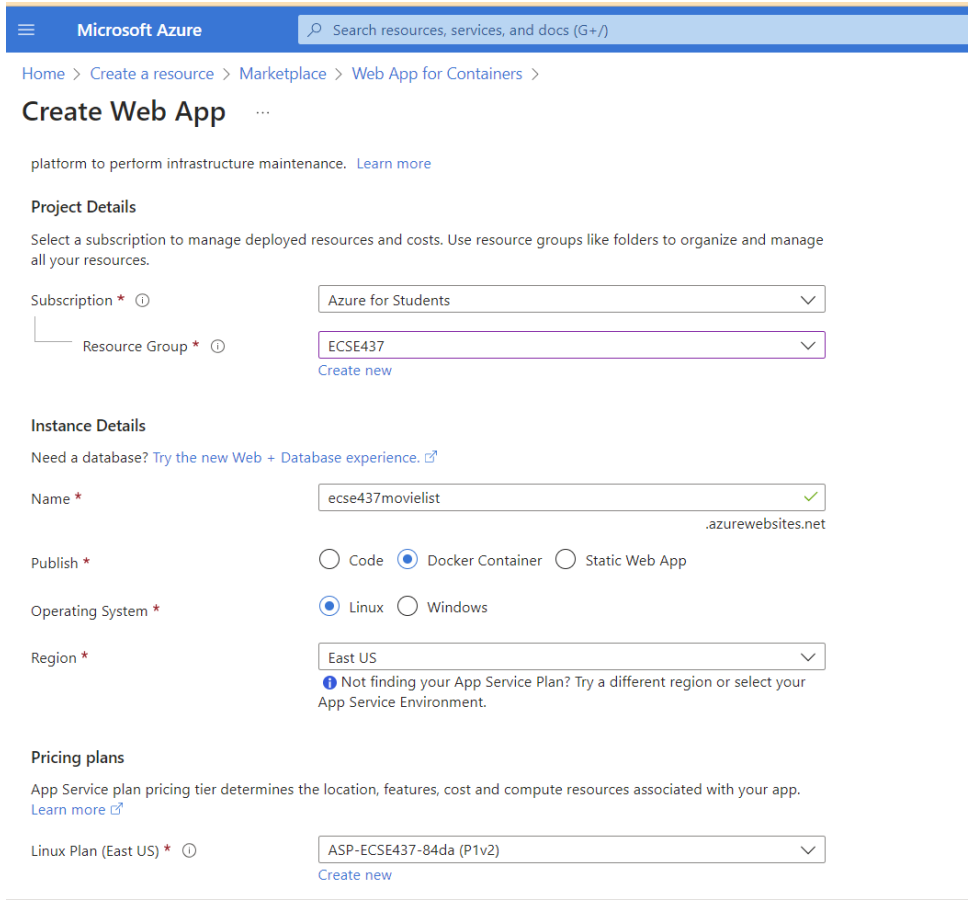
- But what about the db!

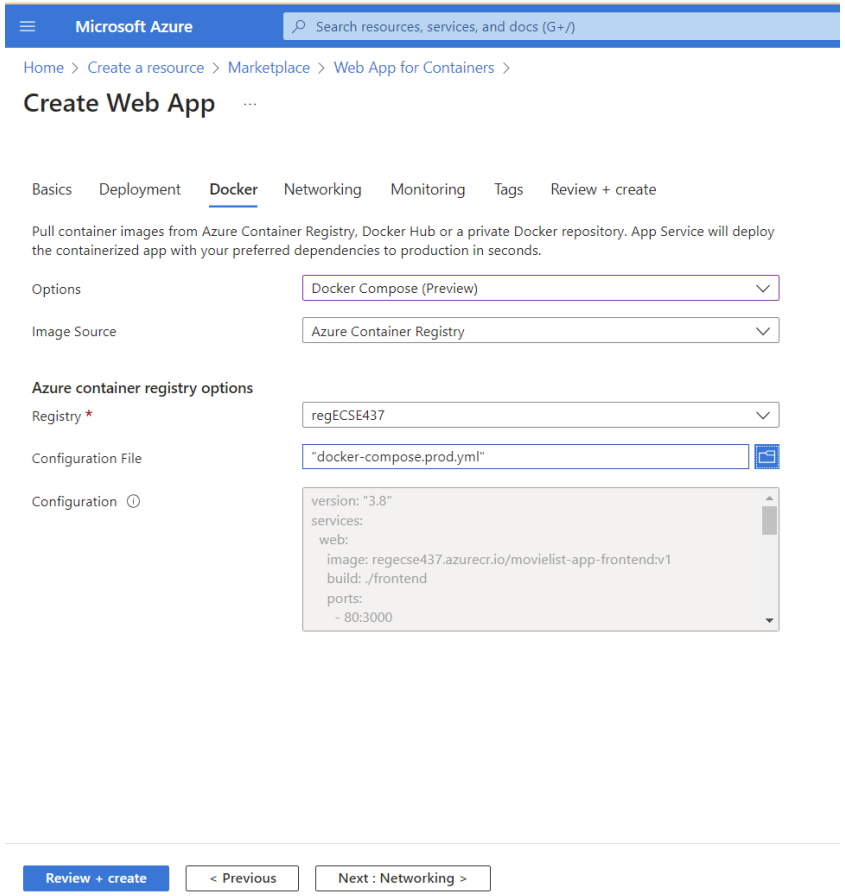
- Since we don't build mongo in our app and we just get it from docker hub we need to change its name and push it manually.

```
PS C:\Windows\System32> docker tag mongo:4.0-xenial regecse437.azurecr.io/mongo:v1
PS C:\Windows\System32> docker push regecse437.azurecr.io/mongo:v1
The push refers to repository [regecse437.azurecr.io/mongo]
```

Create a Web App in our portal

- To deploy our app, we need to create a Web App using the production docker compose file





Microsoft Azure | Search resources, services, and docs (G+/)

Home >

Microsoft.Web-WebApp-Portal-81ac31b9-9e94 | Overview

Deployment

Search

- Overview
- Inputs
- Outputs
- Template

Delete | Cancel | Redeploy | Download | Refresh

Your deployment is complete

Deployment name: Microsoft.Web-WebApp-Portal-81ac31b9-9e94
Subscription: Azure for Students
Resource group: ECSE437

Start time: 7/18/2023, 12:52:03 PM
Correlation ID: a63ef4f0-79e6-42ce-8500-2126afb2f93a

Deployment details

Next steps

Manage deployments for your app. Recommended

Protect your app with authentication. Recommended

Go to resource

Give feedback

Tell us about your experience with deployment