Assignment #2
HA and robust applications

Dr. Daniel Yellin

Due date

The assignment is due January 2, 2025 before 23:59

יש להגיש את המטלה עד 02/01/2025
עד שעה 23:59

Assignment #2

In this assignment you will have to:

1. Use **Docker Compose** to create an application that is built from 5 services:
   - 2 instances of the stocks service you did for assignment #1,
   - an additional service you will build: a *capital-gains* service,
   - a database service and a reverse-proxy service that you download from DockerHub
2. The stocks services must be **persistent** (data stored in a database)
   - In this presentation I use MongoDB, but you can use another DB if you want.
3. Use Docker Compose to **restart the stocks after a failure** (and process requests as if it never failed)
4. Use a **reverse-proxy** (NGINX) to route requests to the right server
5. Implement **load balancing** for the stocks service

Stocks Service

You run two different instances of the stocks service, stocks1 and stocks2.

- For instance, you may have two different stock brokerage accounts.
- You can assume that the same stock (same symbol) will not be in both portfolios.
- Each instance is independent and will listen at a different port on the host.
- The stocks servics must be able to resume to function after a failure
  - Data stored in the database will be available after a crash

Capital Gains service

The capital gains service provides information on your portfolio.

- It needs to communicate with the stocks services to provide the requested information.
- It is not a persistent service (it stores no state).

The capital-gains service has one resource:

/capital-gains

The only request supported on this resource is the GET request.

GET /capital-gains

If no query string is provided, the **GET /capital-gains** request returns the **total** capital gain of your stocks. It is a **float**.

1. The **capital gain** of **one** share of stock is:
cp_one = current stock value of one share – purchase price of one share

2. Hence the capital gain of a **given stock** in the portfolio is:

(cp_one) * (number of shares in portfolio)

3. The capital gain of the entire portfolio is the sum of the capital gain of all the stocks in the portfolio

Note that the capital gain may be negative. (That's called a capital loss)

GET /capital-gains query strings

Recall that a query string has the following form:

?x1=value1&x2=value2&…

Where each x=value can be one of the following:

1. **portfolio=stocks1**
2. **portfolio=stocks2**
3. **numsharesgt=<int>**
4. **numshareslt=<int>**

If you are given a query string in the GET request, then you only return the capital gains of those stocks that satisfy the query string.

Explanation of query strings for GET /capital-gains

1. **portfolio=stocks1**
   This means that the request returns only the capital gains of stocks in the first stock portfolio.
2. **portfolio=stocks2**
   This means that the request returns only the capital gains of stocks in the second stock portfolio.
3. **numsharesgt=<int>**
   This means that the capital gains returned only for stocks for which the number of shares in the portfolio are more than the given integer.
4. **numshareslt=<int>**
   This means that the capital gains returned only for stocks for which the number of shares in the portfolio are less than the given integer.

Examples

- GET /capital-gains?numsharesgt=10

Retrieves capital gains for stocks in your entire portfolio (both stocks1 and stocks2) for which number of shares is greater than 10.

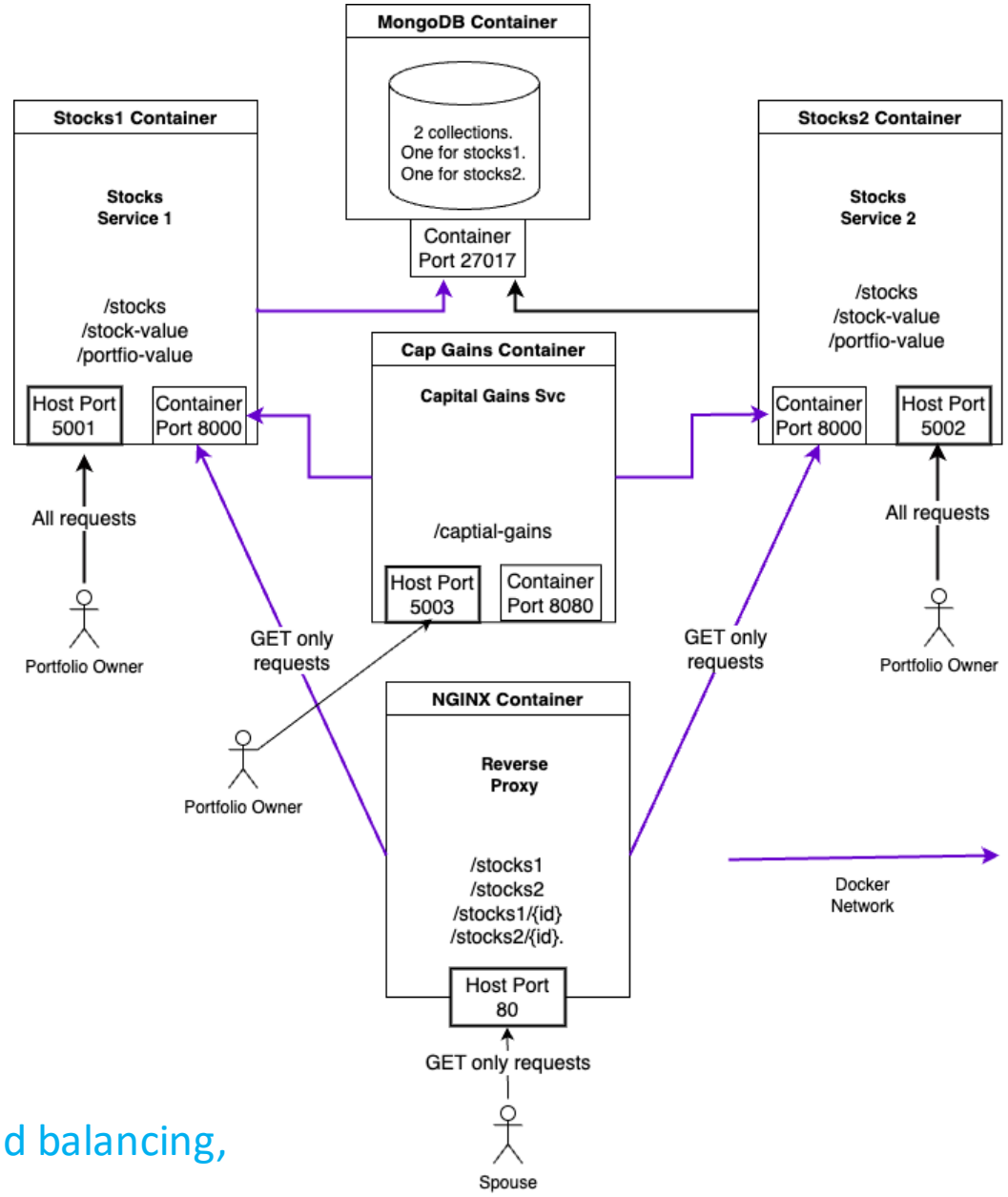
- GET /capital-gains?portfolio=stocks2

Retrieves capital gains for stocks in your stocks2 portfolio only

- GET /capital-gains?portfolio=stocks1&numsharesgt=10
  Retrieves capital gains for stocks in your stocks1 portfolio only for which number of shares you own is greater than 10.
- GET /capital-gains?portfolio=stocks2&numsharesgt=10&numshareslt=20
  Retrieves capital gains for stocks in your stocks2 portfolio only for which number of shares you own is greater than 10 but less than 20.

Architecture

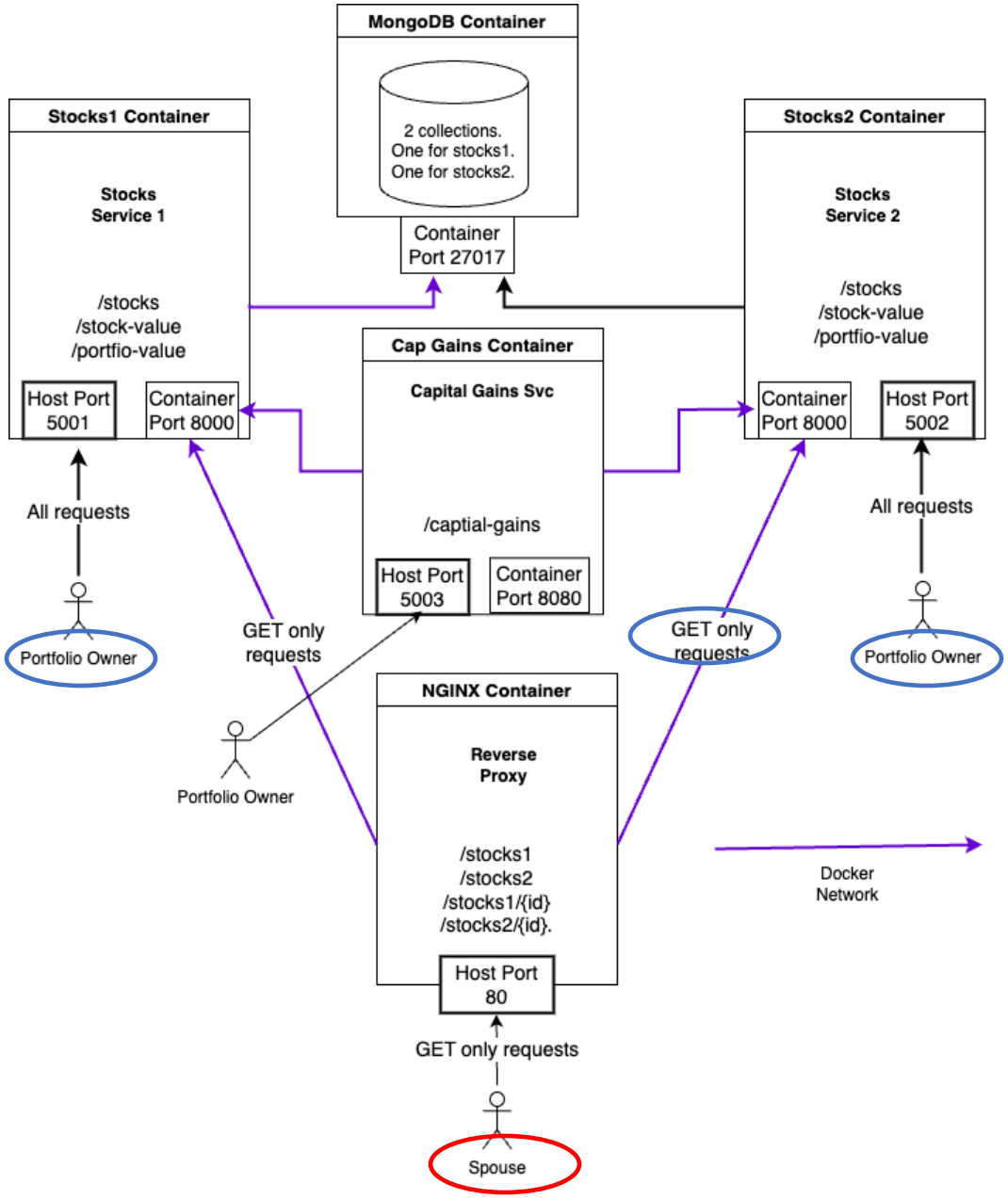
- Create a docker-compose.yml that implements this architecture
  - It must include instructions for restarting services when they fail
  - It must have the services responding to requests on the host ports listed
  - It must start the services in the appropriate order
- Implement persistence for the stocks1 and stocks2 microservices



This diagram does not show load balancing, which we illustrate later.

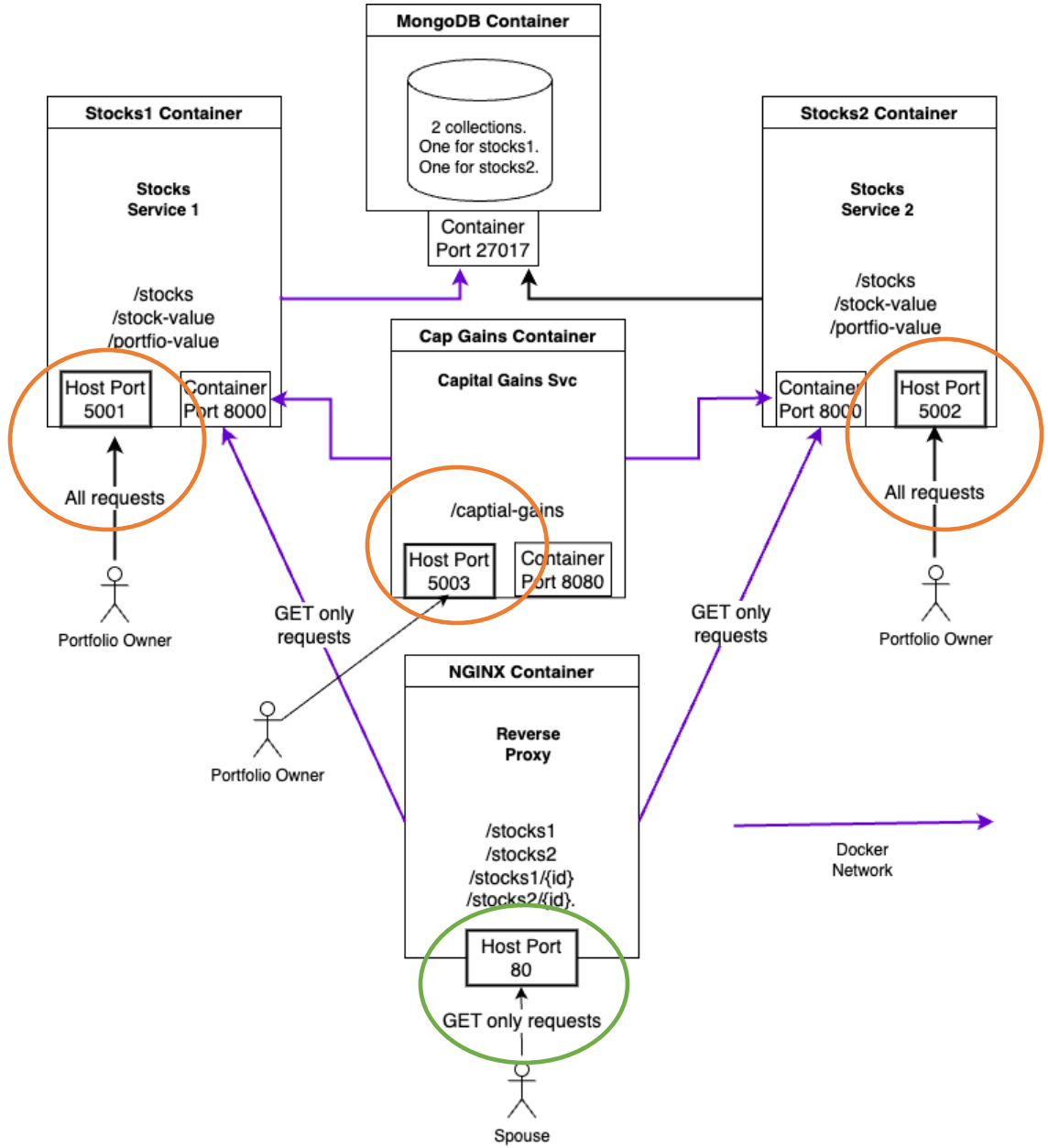
Access

- We have 2 *hypothetical* users accessing the application: the portfolio owner and her spouse.
  - These hypothetical users are just to motivate why we do not allow access to all capabilities via NGINX.
- The portfolio owner can directly issues requests to the stocks1, stocks2 and capital-gains services.
- The spouse can only access certain capabilities via the NGINX container.



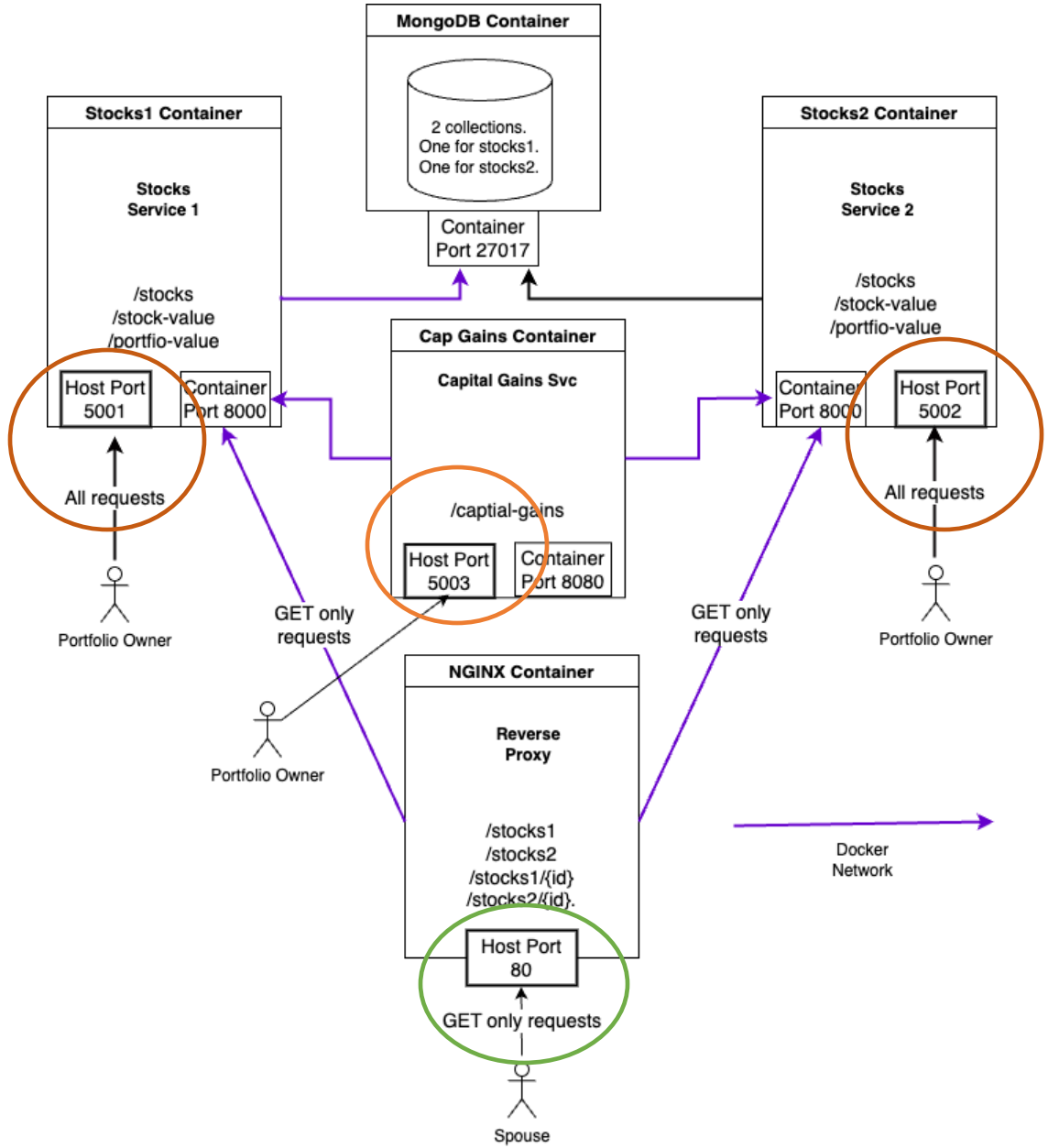
NGINX

- Configure the reverse proxy (NGINX) as in the diagram
- GET requests on the host port 80 for/stocks1 /stocks2, /stocks1/{id} or stocks2/{id} get forwarded as a request to /stocks or stocks/{id} resource of the stocks1 or stocks2 service respectively.
  - Note that NGINX will only accept requests for these resources, not for /stock-value/{id} or /portfolio-value.
- Requests on host ports 5001, 5002 and 5003 do **not** go through the reverse-proxy.
  - All requests (GET,POST,PUT…) on the stock service resources and on the capital gains service resources are permitted.



Container ports

- You must use the given **host** ports as in the diagram.
  - When we check your assignment, we will send requests to these ports.
- You can change the container ports to others if you would like.
  - We will not directly invoke requests on the container ports. Only your code will make requests on these ports. For example:
    - Your app talking to the DB.
    - The NGINX server when forwarding requests to other services.
    - The Capital Gains service invoking requests on the stock services.



How to invoke a container API from another container

Assume we have two services, A-svc and B-svc.

- A-svc provides a REST API for a resource “/my_resource”.
- The A-svc declares the port mapping “4017:8090”.
- Using the docker-compose.yml given here, how would the B-svc invoke this REST API; e.g., with a GET request?

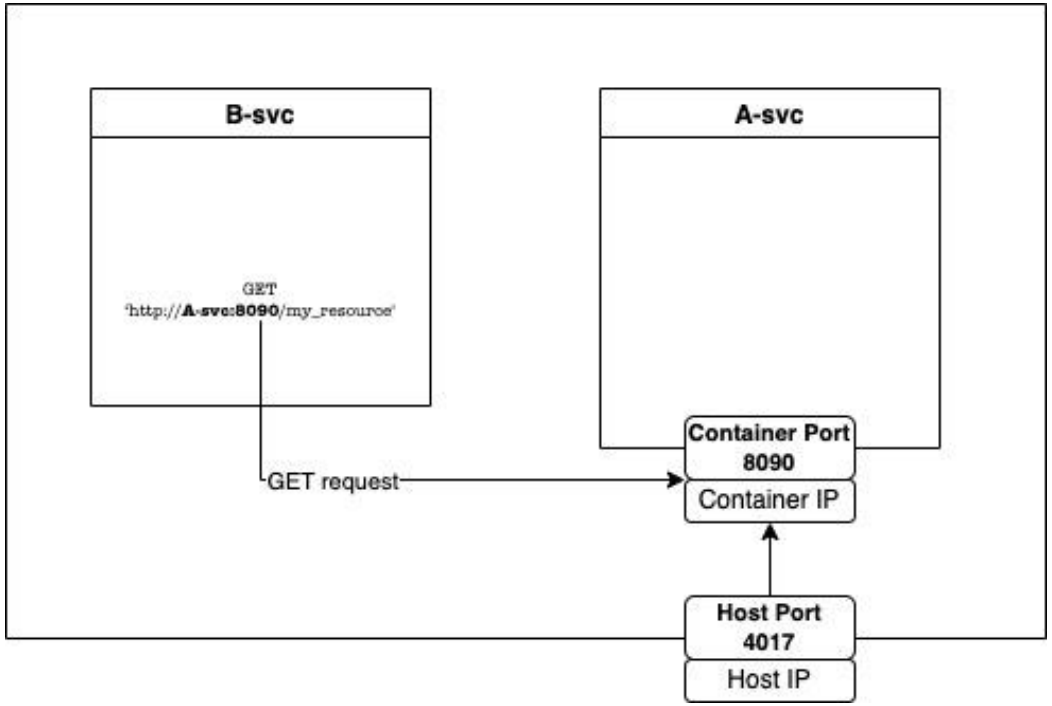
```
version: '3'  # version of compose format

services:
  A-svc:
    build: ./app
    …
    ports:
       - "4017:8090"
    expose:

       - 8090

B-svc:
    build: ./dir

    …
```

How to invoke a container API from another container (cont)

For the B-svc to invoke the API provided by the A-svc, it would issue the cmd:

```
GET 'http://A-svc:8090/my_resource'
```

- It needs to invoke the API from inside Docker, using the Docker network.
- **"A-svc"** is a symbolic name that refers to the Docker IP of the container running the A-svc service.
- Since this invocation is not going through the host, but directly to the container's IP inside Docker, the port must be the container port 8090.



How to invoke a container API from another container (cont)

IMPORTANT: you must use the expose command on the container IP (8090) to allow other container services to reach this port.

```
version: '3'  # version of compose format

services:
  A-svc:
    build: ./app
    ...
    ports:
      - "4017:8090"
    expose:
      - 8090

B-svc:
    build: ./dir

    ...
```

Docker Compose restart

Your stocks containers need to be able to automatically restart on failure. This is accomplished by using the Docker Compose restart instruction, as illustrated in class.

Testing Docker Compose restart

In order to test that your stock containers automatically restart on failure, your stock services need to implement the following /kill request.

```
@app.route('/kill', methods=['GET'])
def kill_container():
    os._exit(1)
```

Upon executing the `GET /kill` request, the stock service will execute os._exit(1). This command will kill the main process and cause the container to fail.

Load balancing

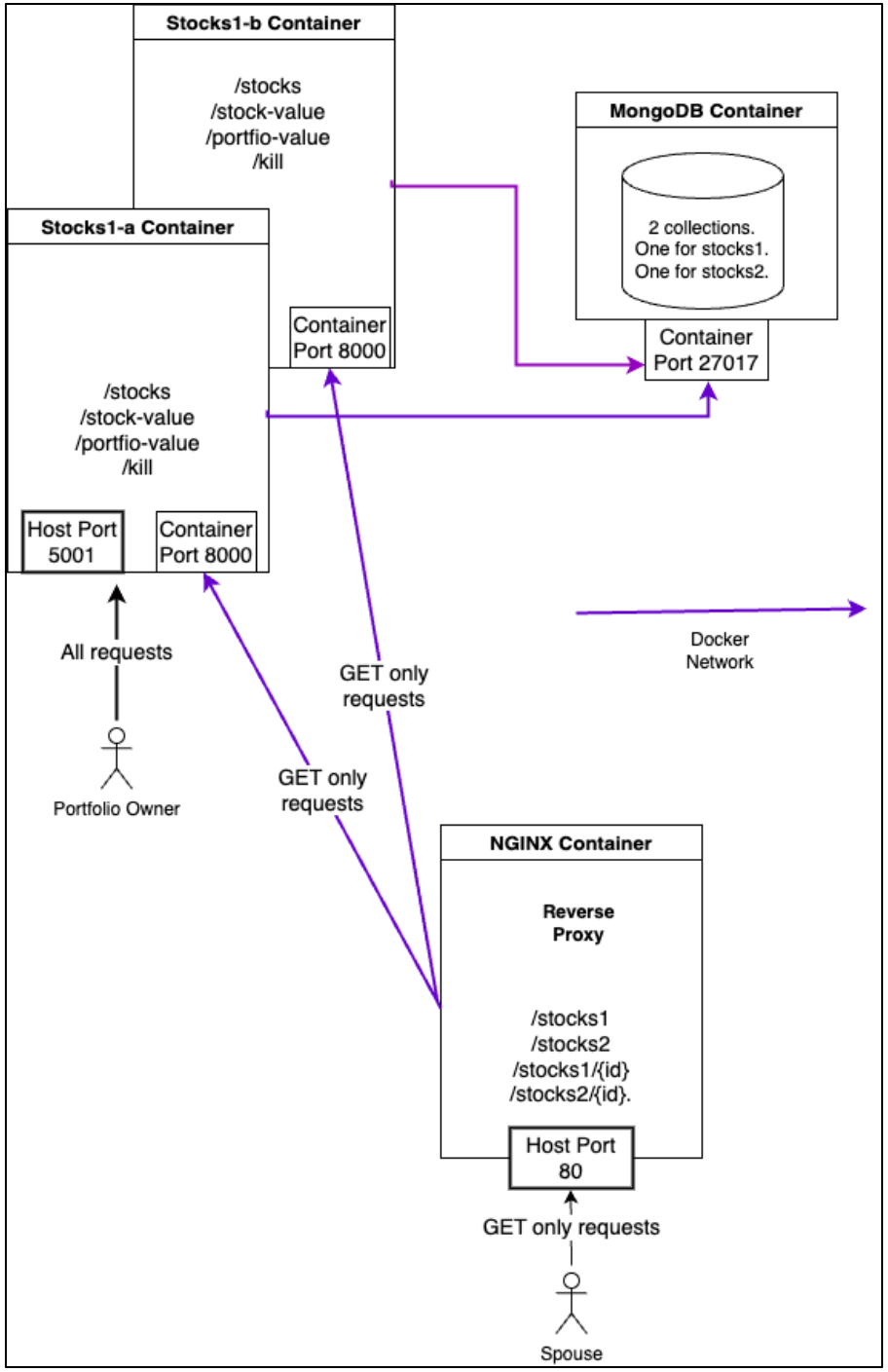
You only need to implement load balancing for stocks1 service.

- You should have **2 instances** of the **stocks1** service.
- You should use a weighted round robbin scheduling policy, as we discussed in class (see NGINX slides).
- For every 3 requests that the **first** instance of stocks1 service receives, the **second** instance of the stocks1 service should receive 1 request. Saying the same thing in different words: the first stocks1 service instance should receive 3 times more the number of requests than the second stocks1 service instance receives.
- Note that the load balancing is only for stocks1 service. All requests for stocks2 goes to the **single stocks2** instance.

stocks1 load balancing

The diagram on the right shows the two stocks1 services (containers), stocks1-a and stocks1-b.

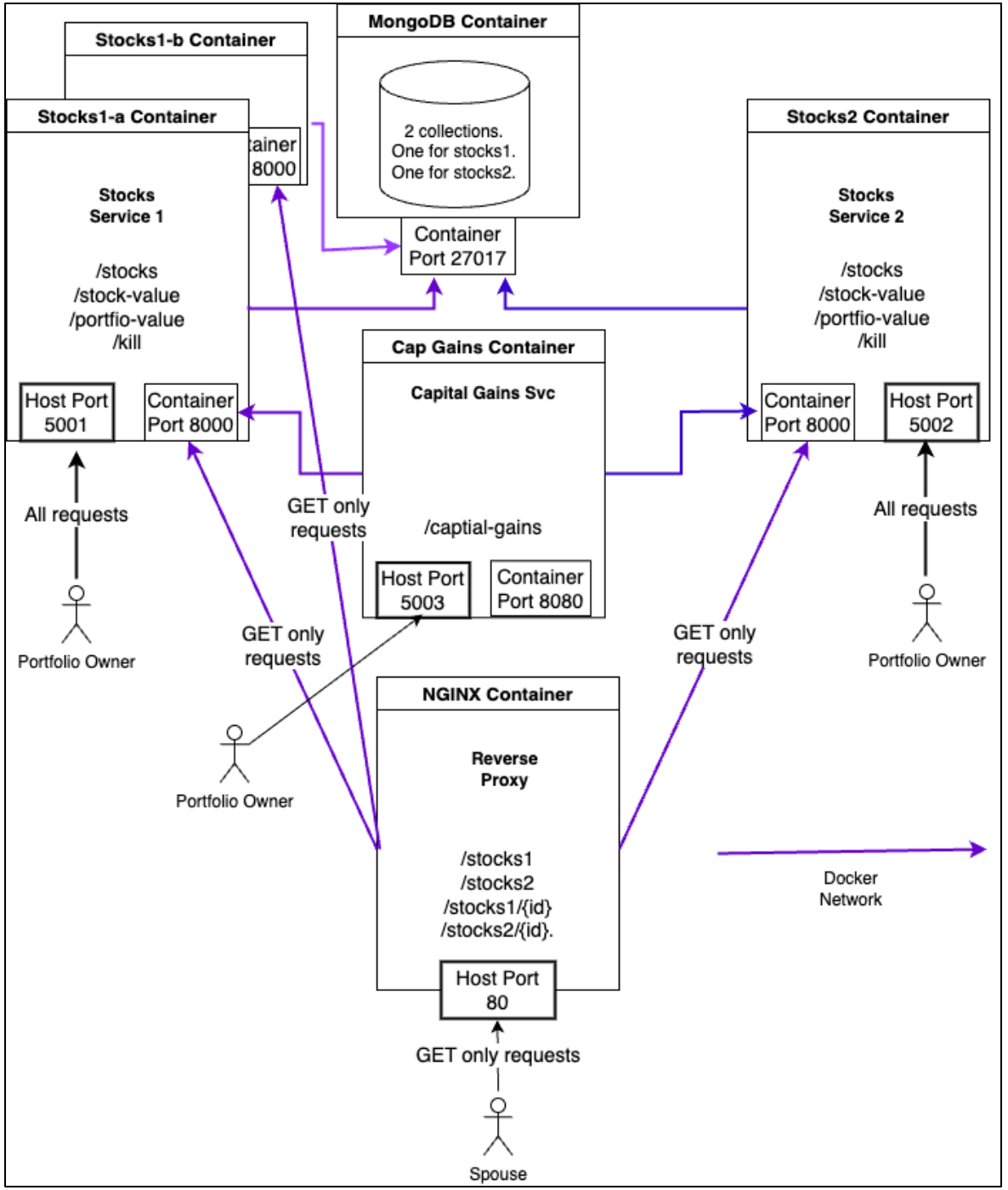
- All requests from the Portfolio owner go to the stocks1-a instance on **host** port 5001.
- GET requests for the stocks1 service from NGINX will go either to the stocks1-a or the stocks1-b instance, in a weighted round robbin fashion. These requests will be sent from NGINX over the Docker NW to the **container** ports of these instances.
  - As mentioned above, you can use whatever container ports you want. However, you must be consistent when you define them in the docker-compose.yml and the NGINX configuration file.
- Requests from the capital-gains service to the stocks1 service also goes to the stocks1-a instance.



Total architecture

To summarize:

- There are two instances of the stocks1 service. These are both the same service, storing and retrieving their data from the same MongoDB collection. Think of the stocks1 service as containing stock data for one brokerage account (MyBroker).
- There is one instance of the stocks2 service. This service is different from the stocks1 service. Think of the stocks2 service as containing stock data for a different brokerage account (MyBank). stocks2 stores and retrieves its data from a different MongoDB collection than the stocks1 service.



Submitting your assignment

If submitting work as a team, please **attach a document listing the team members**.

If team members change from one assignment to another, please indicate this again in the document; otherwise, the grade distribution will be based on the previous assignment.

Submission

- The code you write will need supply all the files required to run your application.
  - This includes code for your services, the Dockerfiles required for your services, the docker-compose.yml, the configuration file for NGINX and any other required files.
- You should submit all of your code in a zip file. If your docker-compose or Dockerfile assumes a specific file structure (e.g., it assumes that code on the host is in a subdirectory) then unzipping the zip file should preserve that directory structure.
- **You should test that your assignment works before submitting.** I recommend that you give your zip file to a friend to unzip and issue the Docker Compose build and run commands. I.e., make sure that it will work for the TA (not only on your computer) before submitting.

Grading of the assignment (1)

We will test your code by:

- Running your docker-compose.yml to build the images and containers, and then run the application.
- Issuing REST requests to the different services on the specified host ports and checking that the correct responses are returned.
  - Includes checking query strings work properly.
- Testing that your services automatically restart after failure.
- Testing the persistence of the stock services, even after they fail and are restarted.
- Testing that NGINX forwards requests properly and denies requests according to the given policies.
- Testing NGINX load balancing implements weighted round robin correctly.