Assignment: Improving our services

Description:
This is a team assignment. Coordination between teams will be required.

We will discuss this more in class -- this document alone doesn't convey the full details of the assignment.

The phases below should not drive the timing of your work on this assignment. They are simply times when you'll make versions available to other teams.

You are welcome/encouraged to communicate & exchange information with other teams on using Docker.

Phase 1:
Do not change your API at all. You may modify/improve your code. The purpose of this phase it to get an initial version of your service(s) into the hands of the other teams.

Phase 2:
1) Your team should create a new version of your API based on the following:
   a) Feedback to your team from me on the last assignment
   b) The general observations that I made in feedback to all teams in the last assignment
   c) Discussions with other teams on what is needed – we will discuss this more in class.
2) Your team should implement that API in new & improved code based on the following:
   a) Implementing your API changes
   b) Fully implementing all functionality required in the last assignment (if not already done)
   c) The general observations that I made in feedback to all teams in the last assignment
3) Some specific requirements at Phase 2 (& beyond):
   a) For each service, there can (and will) be more than one server running that services.
   b) Servers may be started & stopped without impacting the service as long as one server remains running (if a server is stopped while processing a query, it is okay if that query fails).
   c) Assume that more than one client may be working on a given ProblemID at the same time (any service).
   d) Assume that more than one simulated robot may be present in the same problem at the same time.

Phase 3:
Your team will create an easy build that allows me (and others) to run a Docker command to build & start up all of the services, including at least two instances of a server for each service. It should be as simple as building and running the Docker example Voting App. You will use the services (in the form of Docker images/files) provided by others in Phase 2.

What to turn in:

For each phase, you'll submit a link via Canvas. That link should allow me (and others in the class) to grab what you built and easily run it in Docker (similar to the Docker voting app example). If instructions are needed (and they probably are), just put them at the link. You should communicate the link to the other teams via Slack at the same time.

- Phase 1 & 2:
  - Something like that allows me (and any other team) to run your API service. You can do this in one image or in multiple images.
  - *Don't overwrite Phase 1 when you start working on Phase 2 – Phase 1 needs to remain accessible to the other teams.*
  - Here is what needs to be present:
    - You should host the interactive API documentation (e.g., generated by Swagger) so that someone can connect to it via a browser. The interactive API documentation should be able to contact your running server(s) – e.g., via the "try it out" button that sends a request and gets back a response.
    - Your running service that can be contacted at some port number.
    - If your service relies on one of the other services (i.e., you aren't the storage team), then you should have an additional service(s) that is a "mock" of the service(s) that you rely upon.
      - Other teams should be able to replace the "mock" service(s) with their own simply through a configuration.
      - Your service should still function on test cases when you are using this "mock" service.
      - You may **not** use the other team's services in what you turn in for phases 1 & 2.
    - You should have an image that can be accessed to run all of the tests for your service(s). Access could be via running a shell or by vnc'ing in. We'll discuss more in class.
      - For example, you can start a shell in a running container named "db" using the following:
        - docker exec –it db /bin/bash
      - If you want to vnc into a container, then that container has to be running a vnc service (easy to install & add).
    - Your automated tests in Phases 1 & 2 do **not** have to demonstrate use on more than one node/host. So you can, if you like, use just Docker Compose:
      - https://docs.docker.com/compose/overview/

- Phase 3

- Mostly just do what Phase 3 above says is needed. The purpose is to make a single "system" of containers.
  - You will need to have multiple instances of a service, so it is likely you will need to use Docker Swarm instead of only Docker Compose. You can find out more about Docker Swarm here:
    - https://docs.docker.com/engine/swarm/swarm-tutorial/

- You are to carry out performance testing on the system that you create. The performance tests, of course, should be run within containers. The performance tests should do two things:
  - Exercise one (or more) of the "public-facing" APIs (i.e., not the storage API) with "representative" workloads.
  - Focus on one (or more) aspects of performance of your team's API within the overall system.
- The basic idea is to find workloads that expose performance weaknesses in your implementation. The idea isn't to find errors (shouldn't be any). Further, the idea isn't to just make performance measurements along with some graphs. Or to conclude that it requires more CPU time to satisfy more requests.
  - Your tests should have clearly presented results when they are run by someone (e.g., me).
  - You should have a short PDF that explains your conclusions based on the tests.

For the storage service team: You will need to make use of the docker "volume" concept. It is used in the voting app example. You can find practical details on volumes at: https://docs.docker.com/engine/tutorials/dockervolumes/#data-volumes. In Phase 2 (and beyond), you'll need to use shared volumes so that you can run multiple servers within your service. Some useful info at:

- https://devops.profitbricks.com/tutorials/creating-a-mongodb-docker-container-with-an-attached-storage-volume/
- https://www.mongodb.com/containers-and-orchestration-explained