SER422 Spring 2018 ONLINE Lab 5 – Microservices Due: Wednesday, April 25 2018 at 11:59:00pm

No late passes will be accepted for this lab. The deadline is firm and there will be no grade appeals. Your software and configuration must work the first time we get it, we do not have time to go back and forth and deal with configuration errors. Please spend time on this, and plan to spend time! As a lab in Microservices and Containerization, this lab is as much about a stable deployment as it is about the code!

Overview:

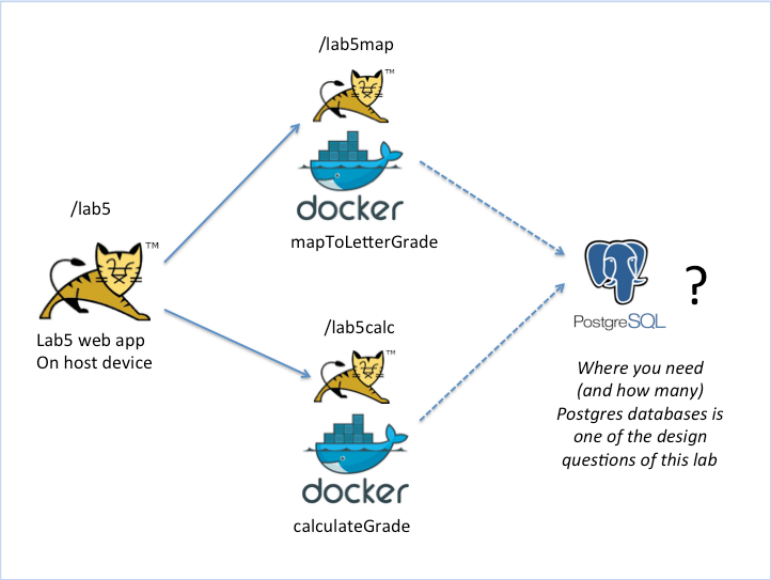
In this lab you will take an existing simple servlet app and refactor in-memory calls to Microservice calls. The example web application "lab5given" has a simple servlet that uses an in-memory class Lab5Service to get numeric grades and mapping to letter grades. The Lab5Service calculates numeric scores for a mock set of Students who are in various majors ("History", "Engineering", "English", "Nursing", or "Psychology") and various years of study ("Freshman", "Sophomore", "Junior", "Senior", or "Graduate"). The calculateGrade method takes an int indicating a year in school, OR a String subject, OR both and returns a double value that is the average score of that cohort. You should of course be familiar with the application as it is the same domain you used in Lab 4. The code has been modified only slightly, including the addition of a simple web form to input year and subject values. Lab5Service does the same function as Lab4Service, and still works against a relational database.

Your refactoring should result in the Microservices architecture shown in the figure below. Your web application should run on your host machine at context /lab5, but now the calculateGrade function should run in one Docker container at context /lab5calc, and the mapToLetterGradeFunction should run in a second Docker container at context /lab5map. The web application on your host should invoke each of these in turn to render the same result page that it currently does.

To get you started, I've created a Docker image that you may use, available on DockerHub at kgaryasu/ser422lab5given. This image runs the entire lab5given application inside a Docker container, including the Tomcat and Postgres components. Basically, it containerizes a monolith. I am asking you to split this up, but the Dockerfile is helpful in setting up your 2 containers.

Requirement, Constraints, and Evaluation Criteria:

Create a microservices version of the given web application by a) putting Lab5Service method calculateGrade into a containerized microservice, b) putting Lab5Service method mapToLetterGrade into a containerized microservice, and c) re-working the given web application to call the new microservices instead of using the in-memory Lab5Service.



1. Your architecture must resemble the figure. Your code and deployment should not rely on specific port numbers – specifically we should not have to recompile any code if the containers' ports are mapped to various port numbers of our choosing.
2. The application should be unchanged from a UI perspective (it is a simple app anyway), with the only exception that you are allowed to indicate errors if there are any problems calling the microservices.
3. Your two microservices do not need to know about each other in this scenario. You do not need to use more advanced Docker capabilities like docker networking or docker compose.
4. Your containers should be configured to run only what they need to run (are as lightweight as possible).

Your submission README.txt must be precise and complete; this lab is as much about how the software is built and deployed as it is about the code, and will be graded as such. Specifically,

1. Your README.txt must have complete step-by-step instructions on building and deploying you software to the host Tomcat and the microservices to the 2 containers.
2. If you create a new Dockerfile or modify the one given to you, you must indicate that and tell us anything we need to know to build your image (we presume we will have to run docker build).
3. The same applied to any shell scripts you reference (I give you one as a start), though usually these require no outside changes.
4. If there are any manual steps we have to do, please make sure they are exactly specified where and when we run them. For example, "Run 'ant dist' from directory XXX, and then do 'docker cp lab5-kgary.war container:/usr/local/tomcat/webapps'".
5. Images for each of your 2 microservices should be pushed to DockerHub. This means you will have to create a DockerHub account. In your README.txt you need to tell us the image locations of the form <account>/<image>[:<tag>], where <account> is your DockerHub account, <image> is the name of the image, which should be "lab5calc" and "lab5map" respectively, and optionally a tagname (we will assume we use the default :latest, so if that is fine you can leave the tag off).
6. Submit the source trees (again, source tree is source code, buid scripts, static resources – everything required to build your software into an executable package and nothing more than that) for the web application and both microservices. These should be distinct after refactoring.Name them lab5-<asurite>-app.zip, lab5-<asurite>-calc.zip and lab5-<asurite>-map.zip respectively.

Your solution will be graded on correctness, quality of design (of both code partitioning and docker configuration), ease of ability to setup, build, and run, and of course adherence to the requirements and constraints in this section.

HERE ARE SOME TIPS ON HOW TO GO ABOUT THIS. NOT REQUIRED BUT HEAVILY RECOMMENDED!

First steps: Ensure Docker is working

1. Watch the video overview of Dockerfiles and for lab 5 setup
2. Run image kgaryasu/ser422lab5given in Docker. You can do this by pulling the image first or by directly starting with a "docker container run" command. You will need to publish ports 8080 and 5432 from the container back to your host at whatever target ports you desire.
3. In your web browser on your host machine, go to *http://localhost<:port_you_mapped_to>/lab5given* and verify the web application works.

Please do this first so you know your initial environment is good to go

Next steps: Identify your Microservices

Here are some suggestions about design considerations and how you can go about this lab incrementally.

Design

1. Recognize that since you are making network calls (yes on your one machine but simulating a physical network), there is more risk that calls will fail and your design must reflect that.
2. You will not be passing object types around through in-memory calls, so what response format will your microservices give?
3. Most importantly, what are the boundaries of your microservices? What code and what internal components are required to create each service?

Incremental Process:

1. Decide on JSON request and response payloads. You basically are creating 2 mini-APIs for 2 methods in Lab5Service. You will have to marshal data on the wire, and the lightweight protocol of choice is JSON. Note that this specification does not lend itself to REST, so there are no constraints on the navigability or semantic metadata (or lack thereof). You just have to get information to the microservice, and read what information comes back.
    - I suggest a small console driver program or unit test to harden the format.
2. I suggest putting the containers aside until you get the microservices code organized according to your Design above. You should already have 2 running Tomcats on your workstation for this class. After refactoring the code, deploy 3 Tomcat web applications in those 2 Tomcats – the main servlet on http://localhost:8080/lab5given (as it is now), and the other 2 on port 8081. For example, http://localhost:8081/lab5calc and http://localhost:8081/lab5map. Doing this keeps Docker out of the equation so you can be sure your code works first. We can give you partial credit from that if you struggle with containerization.
    - Note that this step can be broken down incrementally as well.
        1. In Lab5Servlet, stub out the Lab5Service calls to return a mock of your JSON response payload. This will help you flesh out the format, and also consider things like error cases.
        2. Similarly, write the 2 endpoints and use a tool like Postman or ARC to send various requests with payloads and see that you get the responses you expect.
        3. In this way you can independently test each of the 3 parts of the figure before wiring them together.
3. Once you have it working without Docker, you can then port it into Docker. To do this you will need to do the following:
    - One of the 2 services you need to construct will follow the image I've given you. This is a hint about the partition of your code into services!
    - The other one will only require tomcat. You can work backwards from my Dockerfile or you can start with the official tomcat image and simply add the few steps needed to setup your application.
    - Note you can use the "docker cp" command to copy the warfiles built on your host into your containerized tomcats.
4. Once you have containerized these 2 services, then you can wire the Lab5Servlet to actually call these services instead of the ones on your host from #2 above. Done!

Extra Credit:

1. (20 points) In lab 4 we added a 3rd method on the internal service, *getSubjects()*. Add this behavior back in as a 3rd microservice *written on a different platform*. This can be Node.js, Python, Ruby, whatever (as long as it is not Java/Tomcat). Use the service to dynamically fill-in a drop-down selection box for the subject parameter (replace the textbox in the UI).

SUBMISSION:

- Your submission should be named lab5.<asurite>.zip for upload to Blackboard. In the zipfile should be 3 source tree zipfiles as described above under Requirements #6.
- If you do the EC, you should add to the zip a file lab5-ec.<asurite>.zip
- All of your microservices (the 2 required and 1 optional extra credit) must be on DockerHub by the due date. You must tell us where they are in your READMEs.
- If you decide to use Dockerfiles, perhaps augmented with scripts, make sure these are in your source tree zipfiles.
- You may submit as many times as you like, there is NO reason to to ask for a late submission!
- If you use jar to archive your files, please be sure to change the extension from .jar to .zip before submitting.
- Don't just copy over the entire set of library jarfiles from Lab 4, we don't need all that axis2 stuff, or jersey, or anything other than the postgres driver (in the given) and perhaps a JSON library.