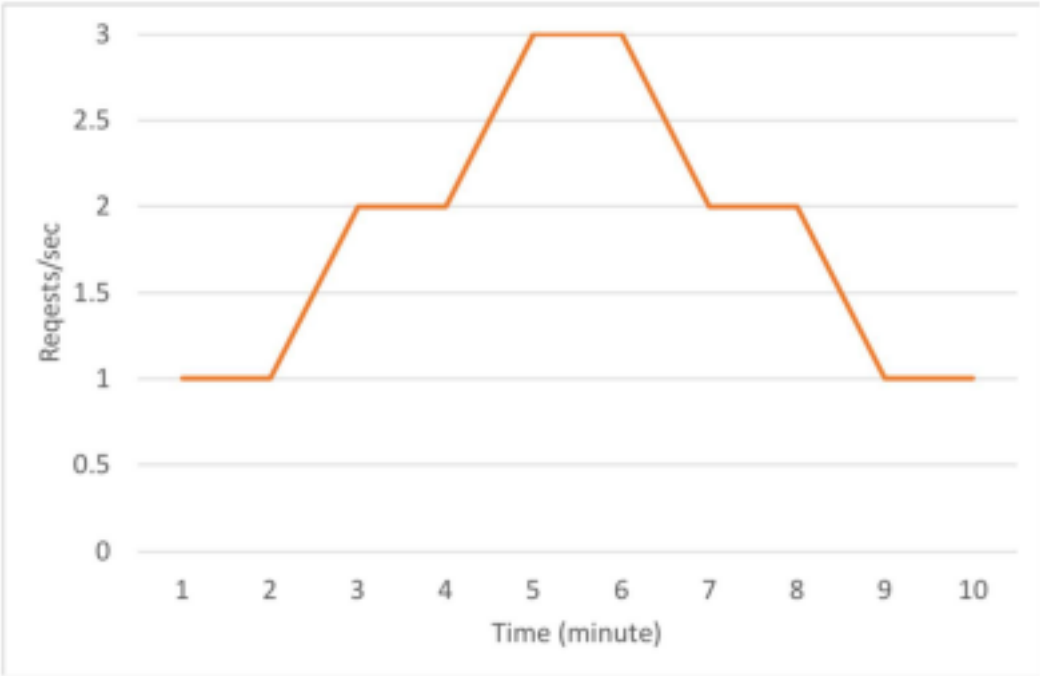
Auto-Scaling for Cloud Microservices

ECE 422 LEC B1 - Winter 2024
Reliable Secure Systems Design

Your goal in this project is to implement a reactive auto-scaling engine for a cloud microservice application. You deploy the application on the Cybera infrastructure using Docker microservices.

A starter kit is provided for this project: start-kit

The auto-scaler should scale out or in (i.e. horizontal scalability) the application according to the workload. To do so, the auto-scaler needs to monitor the response time of users' requests and check if the response times are in the acceptable range or not; acceptable range is indicated by an upper and lower threshold. If response times are longer than the upper bound, then it needs to scale out the application to maintain the reliability and performance. And if the response times are shorter than the lower threshold, your auto-scaler should scale in the application to optimize the operational cost on the cloud. This way you will have a self-adaptive application that optimizes the performance and cost at the same time. The best way to test your application is to apply a normal shape (aka bell shape) workload such as the following to your application.



Hints

1. This application comprises two microservices, namely web and the datastore (i.e. Redis). The web microservices is the bottleneck; so, your auto-scaler only needs to manage the web microservice.
2. To avoid oscillation (aka ping-pong effect), your auto-scaler is better to monitor response times for a period of time (e.g. 10 to 20 seconds) and then based on the average value decide what to do next.

The result of this project should be an auto-scalable application that reactively adjusts itself with respect to the workload. You can implement the auto-scaler in any language you want, such as Python, Java, or C++.

Design Requirements

1. A high-level architectural view of your application in which auto-scalability features have been shown.
2. A state diagram that shows the state, events and actions in the auto-scaler.
3. The pseudocode of the auto-scaling algorithm along with reasons behind the parameter settings in the algorithm; important parameters

may include lower and higher thresholds, the length of the monitoring interval, scaling policies, etc.

All design artifacts should be stored in a folder named “Design” in your project repository.

Final Report

Your final report should include all information related to your project, such as:

- Title and Author name (this is required for your report).
- Abstract/Introduction
- Technologies, methodologies, tools, etc. that you used for accomplishing the project and the reason you chose them.
- Design artifacts with explanation (i.e., whatever you have in your Design folder).
- Deployment instructions and user guides.
- Conclusion.
- References.

Your report should be a 6 to 10 pages document in PDF format and it must be located at the root of your project repository with the name of “Final Report”.

Submission

There are 2 phases to complete your project:

You need to host your project on a private GitHub project and add your TA (GitHub Id: zhijiewang22) as a project member. Your project should include source codes, design artifacts, the final report and anything else you deemed necessary. You may not modify anything in your project repository after the deadline, or you will be penalized according to the Syllabus Late Work policies.

Finally, you must demo your auto-scalable application for the TA within one week after the deadline. You should arrange your demo time with the TA (Zhijie Wang < zhijie.wang@ualberta.ca>). Each demo should not take more than 15 minutes.

Grading

Design 30%

Correct operations 70%