Fundamentals of System Security
(COMSM0122)

Lab Exercise: Designing a System that Scales to Millions of Users

Prerequisite:
(1) Visual Studio Code (VSCode)
(2) Node.js
(3) Some knowledge of JavaScript
(4) A GitHub account

Use cases

Knowledge of specific cloud architectures such as AWS or Azure is not required for this exercise. However, many of the principles you will learn and tasks performed can be applied more generally in any cloud computing ecosystem**.** In this exercise, you will design a system that incrementally scales from tens to *hundreds* to thousands and, subsequently, millions of user requests. At each point, you will ensure that your system architecture is robust and can withstand denial-of-service attacks. You will evaluate the effectiveness of your architecture at each level using a design configuration and load-balancing testbed (https://github.com/Inah-Dev/LoadBalancerTestbed).

We'll scope the problem to handle only the following use cases:

1. The user makes a read or write request to the system with each request carrying a data payload.
2. The server provides the user with a session service that processes and stores the request payload in a database.
3. The system needs to evolve from serving a small amount of users to millions of users
4. Discuss general scaling patterns as you evolve the design architecture to handle an increasing number of users and requests.
5. The system is cost-effective, highly available and can withstand denial-of-service attacks.

Constraints and assumptions

a) User traffic is not evenly distributed. The size of each request's payload therefore varies.

b) You have the option of selecting from one of three load-balancing algorithms. These are Round Robin, Least Connections and Consistent Hashing.

c) You may choose to scale vertically, horizontally, or both. However, you are limited by the following:
   - The number of user request sessions that a server can handle at any point in time is limited.
   - Each server can have multiple replicas up to a limit of 100
   - Server performance is not evenly distributed, with each varying by the processing time required to satisfy a request.

d) All servers satisfy a request by writing its payload to a datastore and responding to the user on a successful write. To achieve this function, servers are connected to one or more databases. The capacity of databases is not evenly distributed.

Setup: Load-Balancing Testbed

- Check that NodeJS is installed by running the command node on the terminal. You can download and install the latest version from https://nodejs.org/en
- Create a new project folder "LoadBalancingTB" on VSCode.
- To copy code files from GitHub - from the new project folder, run the command: git clone https://github.com/Inah-Dev/LoadBalancerTestbed.git

The testbed depends on "find-free-ports" package on npm for NodeJS HTTP server configuration. To install this package run npm install find-free-ports

Objective

You will use the above use case, constraints and assumptions in the following hands-on lab. The objective is to work in groups of 3-4 to transform the requirements above into a highly-available, resilient and secured system design architecture. You will achieve this objective by addressing the tasks listed below.

Tasks

1. Create a high-level design diagram of an architecture that scales for tens of users.

2. Generate an architectural configuration file containing a config object with the following variable settings:

```
config = {
   LoadBalancingAlgorithm: <value>,
   NoServers: <value>,
   Replicas: <value>,
   MaxNoSessionsPerServer: <value>,
   MinNoSessionsPerServer: <value>,
   MaxProcessingTime: <value>, //seconds
   MinProcessingTime: <value>, //seconds
   NoDatastores: <value>,
   MaxDatastoreSpace: <value>, //GB
   MinDatastoreSpace: <value>, //MB
   MaxSizeRequestPayload: <value>, //MB
   MinSizeRequestPayload: <value>, //MB
}
```

3. To do an initial manual setup of your configuration, run the command: node MainTerminal.js on your terminal. This will enable you to manually set your design configuration and to execute the test bed.
   To test the configuration, open a second terminal and run the command: curl http://localhost:8080/<request> - where <request> is the payload. The request is successful if the response is "Session ACCESS GRANTED" and fails if the response is "Session ACCESS DENIED". A session is denied when

either due to lack of space in the datastore or an unavailable server to handle the session.

4. Create a new JavaScript file “RequestSimulation.js” and implement a function that curls tens of server requests every 5, 10, 15, 20 and 30 seconds, as well as record the success and failure rates.

5. Evaluate the effectiveness of your configuration in Task (2) by discussing the failure/success rates results in Task (4)

6. Generate five more configurations in task (2) and repeat tasks (4) and (5). Compare your configurations by discussing the ability of each to satisfy highly-available, resilient requirements and the ability to withstand denial-of-service attacks.

7. Repeat all Tasks above for (a) *hundreds*, (b) thousands and, (c) millions of user requests.

Deliverables
Submission should be made electronically as a folder containing all deliverables via Blackboard for Workshop1. Your folder should be the name of your group]. The folder should contain the following:

1. A pdf document containing your answers to discussion Tasks. State the group name and its members including registration number at the top of the document. Also, state your group’s docker hub username and password.
2. Your project folder “LoadBalancingTB” which contains RequestSimulation.js files.

One member of a group may submit on behalf of the whole group. A reflective log is not compulsory as this is a formative assessment, but you are encouraged to submit one to help us keep a record of your learning.

Ensure one of the TAs or the course lecturer has reviewed your solutions before the end of lab session.

.