TicketOverflow

Software Architecture

Semester 1, 2023

Brae Webb

Summary

In this assignment, you will demonstrate your ability to *design*, *implement*, and *deploy* a web API that can process a high load, i.e. a scalable application. You are to deploy an API for booking concert tickets, generating printed tickets, and generating seating plans. Specially your application needs to support:

- Purchasing of tickets via API requests.
- Generation of printed (downloadable) tickets.
- Generation of seating plans for concerts.
- Providing access via a specified REST API, e.g. for use by front-end interfaces.
- Remaining responsive to the user while generating tickets and seating plans.

Your service will be deployed to AWS and will undergo automated correctness and load-testing to ensure it meets the requirements.

1 Introduction

Recent controversy has brought attention to the monopoly held by certain ticketing companies. In hopes of an eventual collapse of the current monopoly, we aim to develop TicketOverflow, an online concert ticket booking system.

Task For this assignment, you are working for TicketOverflow, a new competitor in the online ticket booking space. TicketOverflow uses a microservices based architecture to implement their new platform. The CEO saw on your resume that you took Software Architecture and has assigned you to design and implement the ticket purchasing service. This service must be scalable to cope with a large influx of bookings.

You also need to implement a mock of the users service. This service is being implemented by another team at TicketOverflow. You need to implement a mock to allow your ticket purchasing service to be tested. Your mock of this service will use a hard-coded list of users, which is available at https://github.com/csse6400/ticketoverflow-users.

Requirements As you may be aware, ticket booking platforms have intense peaks of traffic. At the scale that TicketOverflow hopes to operate, it would be inappropriate to run the servers required to meet demand at all times. Thus, our service must be elastic — able to scale up to meet demand and able to scale down to preserve costs.

Persistence is an important characteristic of a ticket booking platform. Customers naturally will be upset if they purchase a ticket and the system loses it because a server crashed. Upon receiving a ticket purchase request, the system must guarantee that the data has been saved to persistent storage before returning a success response. (Note: The course coordinator had the experience of selling 400 duplicate tickets for a concert, and having the duplicate ticket holders show up at the venue, all because the system failed to record that those tickets had already been sold.)

2 Interface

As you are operating in a microservices context, other service providers have been given an API specification for your service. They have been developing their services based on this specification so you must match it exactly.

The interface specification is available to all service owners online: https://csse6400.uqcloud.net/assessment/ticketoverflow

3 Implementation

The following constraints apply to the implementation of your assignment solution.

3.1 Hamilton

The tool to generate appropriately styled ticket and seating plan images has already been developed by the company's UX team. TicketOverflow's investors like the designs generated by this tool. Unfortunately, the tool developer has quit and the tool can run quite slowly. You will have to work around this bottleneck in the design and development of your parts of the system. (You are not allowed to reimplement or modify this tool.)

Your service must utilise the `hamilton` command line tool provided for this assignment. You may not make any modifications to this tool. The compiled binaries are available in the tool's GitHub repository: https://github.com/CSSE6400/hamilton

3.2 AWS Services

Please make note of the AWS services[1] that you can use in the AWS Learner Labs, and the limitations that are placed on the usage of these services. To view this page you need to be logged in to your AWS learner lab environment and have a lab open.

3.3 External Services

You may **not** use services or products from outside of the AWS Learner Labs environment. For example, you may not host instances of the `hamilton` command line tool on another cloud platform (e.g. Google Cloud).

You may **not** use services or products that run on AWS infrastructure external to your learner lab environment. For example, you may not deploy a third-party product like MongoDB Atlas on AWS and then use it from your service.

4 Submission

Your solution must committed to the GitHub repository specified in Section 4.1. Committing to this repository will be disabled after **16:00 (AEST) on 16 May 2023** and the contents of the repository at this time will be taken as your submission. The repository **must** contain everything required to successfully deploy your application. You must include all of the following in the repository:

[1] https://labs.vocareum.com/web/2460291/1564816.0/ASNLIB/public/docs/lang/en-us/README.html#services

- Your implementation of the service API, including the source code and a mechanism to build the service.[2]
- Terraform code that can provision your service in a fresh AWS Learner Lab environment.
- A `deploy.sh` script that can use your Terraform code to deploy your application. This script can perform other tasks as required.

When deploying your application to mark, we will follow reproducible steps, outlined below. You may re-create the process yourself.

1. Your Git repository will be checked out locally.
2. AWS credentials will be copied into your repository in the top-level directory, in a file called `credentials`.
3. The script `deploy.sh` in the top-level of the repository will be run.
4. The `deploy.sh` script **must** create a file named `api.txt` which contains the URL at which your API is deployed, e.g. `http://my-lb.com/`
5. We will run automated functionality and load-testing on the URL provided in the `api.txt` file.

Important Note: Ensure your service does not exceed the resource limits of AWS Learner Labs. For example, AWS will deactivate your account if more than 15 EC2 instances are running.

4.1 GitHub Repository

You will be provisioned with a private repo on GitHub for this assignment, via GitHub Classroom. You must click on the link below and associate your GitHub username with your UQ student ID in the Classroom.

`https://classroom.github.com/a/ZW_8y-7G`

Associating your GitHub username with another student's ID, or getting someone else to associate their GitHub username with your student ID, is academic misconduct[3].

If for some reason you have accidently associated your GitHub username with the wrong student ID, contact the course staff as soon as possible.

4.2 Tips

If something goes wrong Ideally, your infrastructure will be deployed successfully but this could go wrong for any number of unforeseen reasons. If something does go wrong during deployment, teaching staff will refer to your `README.md` file to attempt to resolve the issue where possible. Please ensure that your `README.md` documentation is of high quality to allow yourself the best chance for recovery.

Terraform plan/apply hanging If your `terraform plan` or `terraform apply` command hangs without any output, check your AWS credentials. Using credentials of an expired learner lab session will cause Terraform to hang.

[2] If you use external libraries, ensure that you pin the versions to avoid external changes breaking your application.

[3] `https://my.uq.edu.au/information-and-services/manage-my-program/student-integrity-and-conduct/academic-integrity-and-student-conduct`

Fresh AWS learner lab Your AWS learner lab can be reset using the reset button in the learner lab toolbar.

To ensure that you are not accidentally depending on anything specific to your learner lab environment, we recommend that you reset your lab prior to final submission. Note that resetting the lab can take a considerable amount of time, in the order of hours. You should do this at least 4 to 6 hours before the submission deadline. Please do not wait to the last minute.

Deploying with Docker In this course, you have been shown how to use Docker containers to deploy on EC2/ECS. You may refer to the practical worksheets for a description of how to deploy with containers [1].

Authenticating AWS resources from EC2 You may find it helpful to access other AWS resources, such as SQS from an EC2 instance. AWS offers libraries to handle this communication[4], however you will need to authenticate to access the resources.

You can give the EC2 instance the built-in IAM instance profile, `LabInstanceProfile` using Terraform[5]. This profile will provide your EC2 instance with access to all AWS resources on the account.

4.3 Fine Print

You can reproduce our process for deploying your application using our Docker image[6]:

```
» cat Dockerfile
FROM ubuntu:22.04

# Install terraform
RUN apt-get update \
    && apt-get install -y unzip wget \
    && rm -rf /var/lib/apt/lists/*
RUN wget https://releases.hashicorp.com/terraform/1.4.4/terraform_1.4.4_linux_amd64.zip \
    && unzip terraform_1.4.4_linux_amd64.zip -d /usr/local/bin \
    && rm -rf terraform_1.4.4_linux_amd64.zip \
    && chmod +x /usr/local/bin/terraform

# Install docker client
RUN apt-get update \
    && apt-get install -y docker.io \
    && rm -rf /var/lib/apt/lists/*

WORKDIR /workspace
CMD ["bash", "/workspace/deploy.sh"]
```

Our steps for deploying your infrastructure using this container are as follows. `$REPO` is the name of your repository, and `$CREDENTIALS` is the path where we will store our AWS credentials.

[4] https://aws.amazon.com/tools/

[5] https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/instance#iam_instance_profile

[6] https://ghcr.io/CSSE6400/csse6400-cloud-testing

```
$ git clone git@github.com:CSSE6400/$REPO
$ cp $CREDENTIALS $REPO
$ docker run -v /var/run/docker.sock:/var/run/docker.sock -v $(pwd)/$REPO:/workspace
    csse6400-cloud-testing
$ cat $REPO/api.txt # this will be used for load-testing
```

Note that the Docker socket of the host has been mounted. This enables running `docker` in the container. This has been tested on MacOSX and Linux but may require WSL2 on Windows.

5 Criteria

Your assignment submission will be assessed on its ability to support the specified use cases. Testing is divided into functionality testing and quality testing. Functionality testing is to ensure that your backend software and API meet the MVP requirements by satisfying the API specification without any excessive load. Quality testing is based upon several likely use case scenarios. The scenarios create different scaling requirements.

Partial marks are available for both types of tests, i.e. if some functionality is implemented you can receive marks for that functionality, or if your service can handle 80% of the load during quality testing you will receive marks for that.

5.1 Functionality

40% of the total marks for the assignment are for correctly implementing the API specification, irrespective of whether it is able to cope with high loads. A suite of automated API tests will assess the correctness of your implementation, via a sequence of API calls. Some tests from this suite will be made available before the assignment due date.

5.2 Quality Scenarios

The remaining 60% of the marks will be derived from how well your service handles various scenarios. These scenarios will require you to consider how your application performs under load. Examples of possible scenarios are described below. These are not descriptions of specific tests that will be run, rather they are examples of the types of tests that will be run.

Small concert As TicketOverflow is starting out, it will need to scale to the size of a small concert. Playhouse at QPAC is one of the first venues to sign up for the service. Their customers purchase tickets in advance over a longer period of time.

Pre-sale for Hamilton Hamilton is now showing at the Lyric Theatre at QPAC and the pre-sale tickets are released for a limited time. The tickets are in high demand, and there are a large number of customers trying to purchase tickets at the same time.

General sale for Hamilton The general sale for Hamilton has started, and there is a surge of customers trying to purchase tickets at the same time. This leads to a high traffic volume on the website, which puts a strain on TicketOverflow.

Seating plan launch The frontend team at TicketOverflow have developed a new ticket purchasing interface that shows users the seating plan before they purchase. A large concert is being used to trial this new feature. You need to ensure that the rendering of the seating plan does not interfere with the high volume of associated ticket purchases.

Evening shows at QPAC All QPAC venues are now supported in TicketOverflow. Each evening there are multiple concerts that run concurrently. TicketOverflow must be able to handle a steady stream of parallel purchases, ticket generation, and seating plan generation.

Priority tickets Tickets recently went on sale for a large concert in four months time. Some of the ticket purchasers are downloading their tickets in advance. Meanwhile an evening show for a smaller concert is starting in a few minutes. The evening show attendees should be able to download their tickets without being impacted by the later concert.

Copyright infringement Due to a copyright notice, "Elsa on Ice", has to be renamed to "Bob on Ice". Unfortunately, this is a last minute name change while many users are logging in to generate their tickets. Your system must ensure that they are not shown incorrect tickets and gracefully handles re-generation of tickets.

You are not required to queue up re-printing all the tickets that have already been printed. As indicated in the API specification for the update concert endpoint (`/concerts/id PUT`[7]), "any existing tickets that have been generated need to be removed." This means that, for the concert which is updated, all tickets that have a `PRINTED` status must have that status changed to `NOT_PRINTED`. You need to ensure that tickets with a `PENDING` status cannot print tickets with outdated information. It is the client's responsibility to determine that the ticket status has changed and to re-print the ticket. Your system has to manage the load.

Taylor Swift tour Taylor Swift has chosen TicketOverflow for the release of her Eras Tour. It is estimated that there will be about 2.8 million purchases during this time. Your service should do its best to remain responsive during the launch of the tour.

5.3 Marking

Functionality accounts for 40% of the marks for the assignment. This is split as 25% for correct implementation of the provided API, and 15% for correct generation of tickets and seating plans. The simple queries in the API are worth much less of the mark compared to the API operations that require processing of data.

Functionality marks are based on correct implementation of the functionality, which is primarily assessed by the automated functionality tests.

Persistence is a core functional requirement of the system. If your implementation does not save all new concert registrations and ticket purchases to persistent storage, your grade for the assignment will be capped at 4.

Your persistence mechanism must be robust, so that it can cope with catastrophic failure of the system. If all running instances of your services are terminated, the system must be able to restart and guarantee that it has not lost any data about concerts or ticket purchases for which it returned a success response to the caller. There will not be a test that explicitly kills all services and restarts the system. This will be assessed based on the services you use and how your implementation invokes those services. If you do not store data to a persistent data store, or you return a success response before the data has been saved, are the criteria that determine whether you have successfully implemented persistence.

[7] https://csse6400.uqcloud.net/assessment/ticketoverflow/#api-Concert-be

Scaling your application to deliver the quality scenarios accounts for the other 60% of the marks. The scenarios described in section 5.2 provide guidance as to the type of scalability issues your system is expected to handle. They are not literal descriptions of the exact loads that will be used. Tests related to scenarios that involve more complex behaviour will have higher weight than other tests.

The scenarios will also evaluate whether your service is being wasteful in resource usage. The amount of resources deployed in your AWS account will be monitored to ensure that your service implements a scaling up and scaling down procedure.

Scaling will be assessed via automated tests. A small subset of these will be released shortly before the due date. These tests may consume a ***significant*** portion of your AWS credit. You are advised to be prudent in how many times you execute these tests.

You will be given advanced warning of when your submission will be tested. The idea is that it will give you a window of opportunity to confirm that your system is running correctly in your account, before we run the automated tests. This will also give you the opportunity to resolve any production errors that occur during testing. For this reason, you should ensure that your service has appropriate observability so that you can identify issues and respond accordingly.

Please refer to the marking criteria at the end of this document.

6 Academic Integrity

As this is a higher-level course, you are expected to be familiar with the importance of academic integrity in general, and the details of UQ's rules. If you need a reminder, review the Academic Integrity Modules[8]. Submissions will be checked to ensure that the work submitted is not plagiarised.

This is an individual assignment. You may not discuss details of approaches to solve the problem with other students in the course. All code that you submit must be your own work. You may not directly copy code that you have found online to solve parts of the assignment. If you find ideas from online sources (e.g. Stack Overflow), you must cite and reference[9] these sources. Use the IEEE referencing style[10] for citations and references. Citations should be included in a comment at the location where the idea is used in your code. All references for citations must be included in a file called `refs.txt`. This file should be in the root directory of your project.

Uncited or unreferenced material will be treated as not being your own work. Significant amounts of cited material from other sources will be considered to be of no academic merit.

References

[1] E. Hughes and B. Webb, "Database & container deployment," vol. 5 of *CSSE6400 Practicals*, The University of Queensland, March 2023. https://csse6400.uqcloud.net/practicals/week05.pdf.

[8] https://web.library.uq.edu.au/library-services/it/learnuq-blackboard-help/academic-integrity-modules

[9] https://web.library.uq.edu.au/node/4221/2

[10] https://libraryguides.vu.edu.au/ieeereferencing/gettingstarted

Cloud Criteria

Criteria	Standard						
	Exceptional (7)	**Advanced (6)**	**Proficient (5)**	**Functional (4)**	**Developing (3)**	**Little Evidence (2)**	**No Evidence (1)**
Correct API 25%	Deployed API successfully meets all required specifications by passing the entire test suite.	Deployed API passes most of the test suite, including tests for at least two endpoints that update persistent data.	Deployed API passes a majority of the test suite, including most tests for at least one endpoint that updates persistent data.	Deployed API passes a majority of tests on simple endpoints that do not update persistent data.	Deployed API passes some of the tests on simple endpoints that do not update persistent data.	API cannot be deployed and only runs locally, passing some tests on simple endpoints that do not update persistent data	API passes few tests on simple endpoints that do not update persistent data.
Generated Content 15%	Deployed API can successfully generate a seating plan and printed ticket for all test cases.	Deployed API can successfully generate a seating plan and printed ticket for most test cases.	Deployed API can generate a seating plan and printed ticket but fails several test cases.	Deployed API can generate at least one seating plan and one printed ticket.	API can only generate a seating plan or a printed ticket but not both.	API appears to communicate with the hamilton program but not serve generated content.	No apparent communication with hamilton program.
Quality Scenarios 60%	Nearly all complex scenarios are handled well. Excessive resources have not been used when not required.	Most complex scenarios are handled well. Excessive resources have not been used when not required.	A few complex scenarios are handled, or excessive resources have been used when not required.	Simple scenarios are handled well. or excessive resources have been used when not required.	Some simple scenarios have been handled well.	Minimal simple scenarios are handled.	No scenarios are handled.