SpamOverflow — Software Architecture

Semester 1, 2024 — Evan Hughes

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 deloy an API for scanning and filtering spam/malicious emails. Specially your application needs to support:

- Scanning an email via an API request.
- Providing access via a specified REST API, e.g. for use by front-end interfaces.
- Remaining responsive to the user while scanning emails.

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

1 Introduction

....

Task For this assignment, you are working for SpamOverflow, a new competitor in the email security space. SpamOverflow 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 service. This service must be scalable to cope with a large influx of emails that occur over the day.

Requirements Some email filtering software can be in the flow of traffic or it can be used to scan emails that have already arrived. SpamOverflow has decided to implement a service that does not impede the flow of traffic and gets a API call at receipt of an email. The service then pulls the email from the users inbox as fast as it can to stop the user from seeing the email or clicking any links.

Mail providers like Microsoft and Google only send a single API request for each email received, so for optimal performance this service needs to be able to handle a large number of requests in a short period of time to not miss any emails.

Since these emails can be dangerous, the service must be able to report that it is bad or good in a timely manner. Though genuine emails that are incorrectly marked as dangerous should be returned to the user as quickly as possible.

Persistence is an important characteristic of the platform. Customers will want to do reports and analyse why emails were flagged and naturally will be upset if the system loses it because a server crashed. Upon receiving an email scan request, the system must guarantee that the data has been saved to persistent storage before returning a success response.

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/spamoverflow`

3 Implementation

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

3.1 SpamHammer

A collection of scanners have been provided to you by the company's security team. A suite of these tools are developed by Mr. Hughes which use common metrics to know if an email is bad but are not always accurate. One of these tools is developed by Dr. Richardson who is an AI and Linguistic expert and is 100 percent accurate but is very slow. Unfortunately, because the company wants to aim for a low false positive rate the CEO wants both accuracy and speed. You will have to work around this bottleneck in the design and development of your parts of the system.

Warning

You are not allowed to reimplement or modify this tool.

Your service must utilise the `spamhammer` 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/spamhammer`

This tool is not as magical as it sounds, in the API you will notice a field that must be included that makes it all the way to the scanner but is not part of the email text. This is a setting which decides what each scanner is going to return (either a 0 or 1) and contains the information if the email is actually good or bad. Demonstrations are provided in the repository to show how to use the tool to generate your own examples.

3.2 Similarity

Dr. Richardson has also provided some advice to help you with filtering through the emails and has suggested that you use a similarity metric to compare the emails to a database of known bad emails. The doctor explains it as "many of the emails we have seen with our first customers have the same content and structure it is just that the link or 'Dear <name>' is slightly different".

With this knowledge you have found a common method of getting the difference between documents called the Cosine Similarity which is explained in some videos available on YouTube.

- `https://www.youtube.com/watch?v=e9U0QAFbfLI`
- `https://www.youtube.com/watch?v=Dd16LVt5ct4`

Info

Dr. Richardson emphasises that the similarity metric is not a replacement for the scanners and is an optional way to improve the efficiency of the system.

3.3 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.

3.4 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 `spamhammer` 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.

You may **not** deploy machine learning or GPU backed services.

4 Submission

This assignment has three submissions.

1. March 19 – API Functionality
2. April 12 – Deployed to Cloud
3. May 3 – Scalable Application

All submissions are due at 15:00 on the specified date.

Your solution for each submission must committed to the GitHub repository specified in Section 4.1. Whatever is in the main branch of your repository at each submission deadline will be checked out and assessed. If you have an extension, then the main branch of your repository will be checked out based on your extended deadline. Extension requests **must** be made prior to the submission deadline via my.UQ[2]. If you intend to make a late submission, you **must** inform the course coordinator by email prior to the submission deadline. You **must** then inform the course coordinator by email of the date of your late submission. Late penalties will be applied, as described in the course profile.

Note: Experience has shown that the large majority of students who make a late submission, lose more marks from the late penalty than they gain from any improvements they make to their solution. We strongly encourage you to submit your work on-time.

Your repository **must** contain everything required to successfully deploy your application. The first submission must include the following in your repository:

- Docker container of your implementation of the service API, including the source code and a mechanism to build and run the service.[3]

We will run your Docker container to test the functionality of your API.

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

[2] https://my.uq.edu.au/

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

The second and third submissions must include all of the following in your repository:

- Your implementation of the service API, including the source code and a mechanism to build the service.[4]
- 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 second and third submissions 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 fifteen EC2 instances are running.

4.1 GitHub Repository

You will be provisioned with a private repository 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[5].

If for some reason you have accidentally 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 your deployment is automatically detected to have an issue then a redeployment will be attempted. If this fails then it will be up to the discretion of the course coordinator on how to proceed.

The assessment is split into multiple checkpoints to help ensure that you are on the right track and that your deployment functions.

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.

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

[5] `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 ECS. You may refer to the practical worksheets for a description of how to deploy with containers [1].

4.3 Fine Print

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

```
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.

```
$ 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.

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

5 Criteria

Your assignment submission will be assessed on its ability to support the specified use cases. Testing is divided into functionality testing, deployment 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. Deployment is to ensure that this MVP can then be hosted in the target cloud provider. 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 scenario during quality testing you will receive marks for that. Deployment is marked in its entirety.

5.1 API 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. The API test suite will be made available before the functionality due date.

5.2 Deployed to Cloud

25% of the total marks for the assignment are for deploying a correctly implemented service to AWS irrespective of whether it is able to cope with high loads. The deployment will be assessed by running a script that deploys your service to AWS that then runs a suite of automated API tests to assess the correctness of your implementation. The script will be provided to you before the deployment checkpoint due date with the API test suite.

5.3 Scalable Application

The remaining 35% 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.

Steady Stream Steady receipt of email messages at a rate of *N* per minute, evenly spread across all accounts. Approximately 20% of the messages are 'bad'.

Bad 'News' Stream Steady receipt of email messages at a rate of *N* per minute, evenly spread across all accounts. Approximately 80% of the messages are 'bad'.

Peaks and Troughs Periods of receipt of a high volume of email messages, followed by periods of low volume.

High Value Customer The Department of Defence (DoD) has adopted SpamOverflow. They are a high value customer and you must ensure that their requests are handled quickly. All other customers have a service level agreement (SLA) that guarantees a certain level of responsiveness. You cannot ignore their requests to only prioritise the DoD's requests.

Leaked Directory A bad actor has managed to get the email addresses of all employees of <Large Company>. They have sent a phishing email to all of the users advertising a pay raise with a link to a fake login page. The email is sent to all 10,000 employees at the same time.

Personalised Attack A bad actor has trained an AI model using social media profiles of targeted victims. They can generate personalised phishing email messages based on personal information. As the messages are personalised, they can be of greatly different lengths and contain different content. These messages can only be identified by Dr. Richardson's tool. They have sent these phishing messages to 2,000 users at the same time.

5.4 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 email scans 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 emails 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.

Deploying your service is worth 25% of the marks for the assignment. This is based on the successful deployment, using Terraform, of your service to AWS and the ability to access the service via the API. Your service must be fully functional while deployed so the functionality tests can be run which determines the marks for deployment.

Scaling your application to deliver the quality scenarios accounts for the other 35% of the marks. The scenarios described in section 5.3 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.

All stages of the assessment will be marked using automation and a subset of the tests will be released. 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. The tests for Functionality and Deployment will be released in their entirety for the checkpoints but the tests for Quality Scenarios will only be released at the Course Coordinators discretion or after the due date.

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[7]. 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 on-line to solve parts of the assignment. If you find ideas from on-line sources (e.g. Stack Overflow), you must cite and reference[8] these sources. Use the IEEE referencing style[9] 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.

You may use generative AI tools (e.g. copilot) to assist you in writing code to implement your solutions. You may also use generative AI tools to help you test your implementations. You **must** include, in the root directory of your project, a file `AI.md` that indicates the generative AI tools that you used, how you used them, and the extent of their use. (e.g. All code was written by providing copilot with class descriptions and then revising the generated code.)

Uncited or unreferenced material, or unacknowledged use of generative AI tools, will be treated as not being your own work. Significant amounts of cited or acknowledged work 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.

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

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

[9] 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.