SpamOverflow

Software Architecture

Semester 1, 2024

Evan Hughes, Brae Webb and Richard Thomas

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 to a specified REST API, e.g. for use by front-end interfaces and internal teams.
- Remaining responsive 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

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 malicious email filtering platform. The CEO saw on your resume that you are taking Software Architecture and has assigned you to design and implement a service. This service must be scalable to cope with the anticipated large influx of emails.

Requirements Email filtering software can filter email as it arrives or after. SpamOverflow will implement a service that does not impede the flow of traffic (i.e. does not prevent the email arriving). It will receive an API call when the mail server receives an email message. The service then pulls the email from the user's inbox as fast as it can to prevent the user from seeing the malicious email or clicking any links.

Commercial email providers send an API request for each email received. For optimal performance this service needs to be able to handle a large number of requests in a short period of time, so as 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 analyse why emails were flagged after the fact. Upon receiving an email scan request, and after filtering, 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

You have been provided with a command line tool called `spamhammer` that can be used to scan emails for malicious content. This tool is developed by Dr. Richardson who is an AI and linguistic expert. The tool has varying performance, roughly related to the length of the content. You will have to work around this bottleneck in the design and development of your parts of the system.

Your service **must** utilise the `spamhammer` command line tool provided for this assignment. The compiled binaries are available in the tool's GitHub repository: https://github.com/CSSE6400/spamhammer.

Warning

You are **not** allowed to reimplement or modify this tool.

This tool is not as magical as it sounds. For the purposes of this assignment, in the API specification you will notice the metadata->spamhammer field in the POST body. This is a setting which decides whether the email is malicious and how long processing should take. Demonstrations are provided in the repository to show how to use the tool to generate your own examples. You **must** not use this field to simplify the task of scanning the email.

3.2 Similarity

Dr. Richardson has also provided some advice to help you with filtering through the emails. She has suggested that you use a similarity metric to compare the body of emails to previously 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[1] which is explained in these videos.

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

For the purposes of the assignment you may build up a database of known bad emails during the operation of your API. You must only use the body of the email content to compare similarity.

Info

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

3.3 AWS Services

Please make note of the AWS services[2] that you can use in the AWS Learner Lab, 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.

[1] https://www.learndatasci.com/glossary/cosine-similarity/

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

3.4 External Services

You may **not** use services or products from outside of the AWS Learner Lab 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^{th} – API Functionality
2. April 12^{th} – Deployed to Cloud
3. May 3^{rd} – Scalable Application

All submissions are due at 15:00 on the specified date. Your solution for each submission must be committed and pushed to the GitHub repository specified in Section 4.3.

Each submission is to be tagged[3] to indicate which commit is to be marked. The tags that you **must** use are:

- **stage-1** for API Functionality, due on March 19^{th}
- **stage-2** for Deployed to Cloud, due on April 12^{th}
- **stage-3** for Scalable Application, due on May 3^{rd}

When marking a stage, we will checkout the commit you have tagged for that stage. This allows you to make a conscious decision, if you wish to make a late submission. We will mark the late submission that you have tagged for the stage. 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.

If you forget to tag your submission, we will checkout and mark the latest commit that you made to the main branch before the submission deadline. You should commit and push your work to your repository regularly. If a misconduct case is raised about your submission, a history of regular progress on the assignment through a series of commits could support your argument that the work was your own.

Extension requests **must** be made *prior* to the submission deadline via my.UQ[4].

[3] Atlassian has a good tutorial about Git tag, if you are not familiar with tagging. See: https://www.atlassian.com/git/tutorials/inspecting-a-repository/git-tag.

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

Your repository **must** contain everything required to successfully deploy your application.

4.1 API Functionality Submission

Your first submission **must** include the following in your repository:

- Docker container (Dockerfile) of your implementation of the service API, including the source code and a mechanism to build and run the service.[5]
- A `local.sh` script that can be used to build and run your service locally. Where your container will be launched with port 8080 being passed from the container to the testing environment and your service must be available at `http://localhost:8080/`.

We will run a suite of tests against your API on this endpoint.

4.2 Deployed to Cloud & Scalabiliity Submissions

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.[6]
- Terraform code that can provision your service in a fresh AWS environment.
- A `deploy.sh` script that can use your Terraform code to deploy your application. This script **must** be in the top-level of your repository. This script may 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-api.com/` or `http://123.456.789.012/`.
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.

4.3 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[7].

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

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

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

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

4.4 Tips

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.

Fresh AWS Learner Lab Your AWS Learner Lab can be reset using the reset button in the 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].

4.5 Fine Print

You can reproduce our process for deploying your application using our Docker image[8]:

```
   » 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/1.7.4/terraform_1.7.4_linux_amd64.
   zip \
    && unzip terraform_1.7.4_linux_amd64.zip -d /usr/local/bin \
    && rm -rf terraform_1.7.4_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 your AWS credentials.

[8] 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 Mac OSX 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, deployment and scalability testing, corresponding to the three submission stages of the assignment. 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.

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.

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 and then runs a suite of automated API tests to assess the correctness of your implementation.

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 M per minute, fairly evenly spread across all customers. Approximately 20% of the messages are malicious.

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

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 SpamHammer. They have sent these phishing messages to 2,000 users at the same time.

5.4 Marking

Persistence is a core functional requirement of the system. If your implementation does not save 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. Not saving data to a persistent data store, or returning a success response before the data has been saved, are the criteria that determine whether you have successfully implemented persistence.

Functionality of your service is worth 40% of the marks for the assignment. This is based on the successful implementation of the API specification given and the ability to use the given tool in your implementation.

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 successfully handle the usage 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 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 amount of tests to be released is at the Course Coordinator's discretion.

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

5.4.1 Grade Improvement

Improving your application's functionality in a later submission will be used to improve the grade you received for an earlier submission.

Stage 1 Improvement If your stage 1 submission (API Functionality) performs poorly, you may improve your grade for this stage in a later stage. This will occur if your deployed stage 2 or 3 submission passes more of the functionality tests than your stage 1 submission. This new result will be used to recalculate your grade for stage 1. (We will **not** rerun the local functionality tests in stages 2 or 3. We will **only** test your deployed application.)

Stage 2 Improvement Similarly, if your deployed stage 3 submission passes more of the functionality tests than your stage 2 submission, this result will be used to recalculate your grade for stage 2.

Late Penalties If an earlier submission was late, the same late penalty will be applied to your improved grade. (e.g. If your stage 1 submission was 3 hours late, and in stage 2 your submission's functionality improved from a grade of 4 to a grade of 6, your grade for stage 1 would be increased to a 5 after the one grade point per 24 hours penalty was applied.)

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[9]. Submissions will be checked to ensure that the work submitted is not plagiarised or of no academic merit.

This is an *individual* assignment. You may **not** discuss details of approaches to solve the problem with other students in the course. The Grade Improvement rule (section 5.4.1) means that you may **not** discuss details of any earlier submission stage with other students until **two weeks** *after* the final submission date of May 3^{rd} (i.e. **May 20**th).

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[10] these sources. Use the IEEE referencing style[11] 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 must be in the top-level of your repository.

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 top-level of your repository, a file called `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.

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

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

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

Cloud-Infrastructure Assignment Criteria

Criteria	Standard						
	Exceptional (7)	**Advanced (6)**	**Proficient (5)**	**Functional (4)**	**Developing (3)**	**Little Evidence (2)**	**No Evidence (1)**
API Functionality 40%	API passes at least 85% of the full test suite, including persistence.	API passes at least 75% of the full test suite, including persistence.	API passes at least 65% of the full test suite, including persistence.	API passes at least 50% of the full test suite.	API passes at least 40% of the full test suite.	API passes at least 20% of the full test suite.	API passes less than 20% of the full test suite.
Deployed to Cloud 25%	Deployed API passes at least 85% of the full test suite, including persistence.	Deployed API passes at least 75% of the full test suite, including persistence.	Deployed API passes at least 65% of the full test suite, including persistence.	Deployed API passes at least 50% of the full test suite.	Deployed API passes at least 40% of the full test suite.	Deployed API passes at least 20% of the full test suite.	Deployed API passes less than 20% of the full test suite.
Scalable Application 35%	Nearly all complex scenarios are handled well. Resources have been used efficiently (scale up & down).	Most complex scenarios are handled well. Resources have been used efficiently (scale up & down).	A few complex scenarios are handled, or resource usage is not efficient (scale down is poorly implemented).	Simple scenarios are handled well. or resource usage is not efficient (scale down is not implemented).	Some simple scenarios have been handled well.	Minimal simple scenarios are handled.	No scenarios are handled.