CoughOverflow

Software Architecture

Semester 1, 2025

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 deploy an API to analyse images of saliva samples to identify if a patient has COVID-19 or avian influenza (H5N1), which is commonly called bird flu. Specially your application needs to support:

- Analysing an image received via an API request.
- Providing access to a specified REST API, e.g. for use by front-end interfaces and other applications.
- Remaining responsive while analysing images.

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 CoughOverflow, a new UQ start-up. CoughOverflow uses machine learning techniques developed by QDHeC[1] to analyse images of saliva samples. The analysis is able to identify if an individual is infected with one of a few pathogens. The initial service focuses on identification of COVID-19 or H5N1, due to their current level of risk to the public.

Task CoughOverflow uses a microservices architecture to implement their analysis platform. The CTO saw on your resume that you are taking Software Architecture and has assigned you to design and implement the pathogen analysis service. This service must scale to cope with the anticipated large number of tests.

Requirements Automated identification of pathogens is an important service. Manual testing by lab technicians is labour intensive and time consuming. Automated tests free lab technicians for more important work, and provide faster responses to healthcare staff. This is critical in an epidemic or pandemic scenario, when tens or hundreds of thousands of tests need to be performed daily.

CoughOverflow's pathogen analysis service (PAS) needs to be designed to scale to match demand. Pathology labs will obtain saliva samples from patients. The labs will create images of the cells in the samples. These images will be sent to the PAS for analysis.

The algorithms used to analyse the images are computationally intensive. It is not possible to return a result immediately for a submitted image. Labs, or other healthcare providers, will need to query the PAS to obtain results at a later time. Results can be queried for a single test, or a batch of tests for a lab or patient.

As COVID-19 and H5N1 are potentially life threatening to some patients, the service must be able to provide test results in a timely manner. Early treatment and effective isolation practices can significantly reduce the impacts of these diseases, as well as reducing strain on healthcare resources.

[1] https://chsr.centre.uq.edu.au/research/queensland-digital-health-centre

Persistence is an important characteristic of the platform. Resubmitting analysis requests due to lost data would place unnecessary strain on pathology labs, at times when they may be under extreme pressure to deliver results. Upon receiving an analysis request, and after error checking, the PAS 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/coughoverflow`

3 Implementation

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

3.1 Analysis Engine

You have been provided with a command line tool called `engine` that must be used to analyse sample images. This tool was developed by AI and medical researchers at QDHeC[2]. The tool has varying performance, due to how clear the pathogen markers are in the cell sample images. You will have to work around this bottleneck in the design and development of the PAS.

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

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

The analysis engine requires pre-processing of cell sample images to highlight pathogen markers. This pre-processing is done by the pathology labs. You **must** use the provided sample images for testing purposes. If you try to generate your own images, they are likely to fail analysis or give false results.

3.2 AWS Services

Please make note of the AWS services[3] 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.

3.3 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 `engine` command line tool on another cloud platform (e.g. Google Cloud).

[2] `https://chsr.centre.uq.edu.au/research/queensland-digital-health-centre`
[3] `https://labs.vocareum.com/web/2460291/1564816.0/ASNLIB/public/docs/lang/en-us/README.html#services`

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. April 4^{th} – API Functionality
2. April 17^{th} – Deployed to Cloud
3. May 9^{th} – 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 in its own branch. You **must** use the branch names **exactly** as indicated below. Failure to use these branch names may result in your submission not being marked, and you obtaining a grade of 0 for the submission.

- **stage-1** for API Functionality, due on April 4^{th}
- **stage-2** for Deployed to Cloud, due on April 17^{th}
- **stage-3** for Scalable Application, due on May 9^{th}

When marking a stage, we will checkout the branch for that stage. Any code in the main branch or *any* other branch, will be ignored when marking. We will checkout the latest commit in the branch being marked. If the commit date and time is after the submission deadline, late penalties will be applied, unless you have an extension. Late penalties are 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.

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].

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

4.1 API Functionality Submission

Your first submission **must** include all of 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. This script **must** be in the root directory of your repository. The `local.sh` script must launch your container 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 at this URL.

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

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

4.2 Deployed to Cloud & Scalability 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 provisions your service in a fresh AWS environment.
- A `deploy.sh` script that uses your Terraform code to deploy your application. This script **must** be in the root directory of your repository. This script may perform other tasks as required by your implementation.

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 cloned locally. The submission branch will be checked out.
2. AWS credentials will be copied into your repository in the root directory, in a file called `credentials`.
3. The script `deploy.sh` in the **root directory** 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. If you use up your allocated budget in the Learner Lab, you will not be able to run any services.

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/whdIS1AE`

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.

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.

[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`

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, on the order of hours. You should do this at *least* 3 to 4 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].

```
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.

```
$ 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.

[8] 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, 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. Scalability testing is based upon several likely usage scenarios. The scenarios create different scaling requirements.

5.1 API Functionality

10%, from the 35% of the grade for the assignment, is 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

10%, from the 35% of the grade for the assignment, is 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

15%, from the 35% of the grade for the assignment, will be derived from how well your service handles various scenarios. These scenarios will require you to consider how your application performs under different load characteristics. 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.

Normal Circumstances Average number of analysis requests are submitted by a wide range of labs. These requests are distributed fairly evenly over the working day. Few of the analysis requests are urgent. Queries for results are also distributed fairly evenly over the day.

Curiosity Killed the Server Queensland Health creates a web portal allowing patients to view their analysis results. Tens of thousands of people all try to access their results in a short period of time.

Epidemic Early Stages There is a significant increase the number of analysis request submitted by a few labs. These labs also mark a much higher number of the requests as being urgent. Other labs are operating as per normal circumstances. There are many more queries for results, often being repeated for any pending analysis jobs. Health authorities make many requests for batches of results from labs or for patients.

Epidemic Peak Almost all labs submit a very large volume of analysis requests. Up to 30% of the requests may be given an urgent status. The service must still analyse non-urgent requests in a timely manner. If it does not, the risk is that labs will start to make all requests urgent. Due to the high volume of analysis requests, there will be a high volume of queries for results. Any results that are still pending analysis jobs, will be repeated. Health authorities make a moderate number of requests for batches of results from almost all labs. Labs start querying batches of results for individual patients.

Epidemic Tail Almost all labs submit a large volume of analysis requests. Up to 10% of the requests may be given an urgent status. Due to the high volume of analysis requests, there will be a high volume of queries for results. Most results that are still pending analysis jobs, will be repeated. Health authorities make a moderate number of requests for batches of results from almost all labs. Labs query batches of results for a moderate number of patients.

Epidemic Follow-up Some labs submit a large volume of analysis requests. Up to 5% of the requests may be given an urgent status. Other labs are back to operating as per normal circumstances. There will be a moderate volume of queries for results. Most results that are still pending analysis jobs, will be repeated. Health authorities make a large number of requests for batches of results from almost all labs. Labs query batches of results for a large number of patients.

Research Project A researcher wants to analyse the progression of how a disease spreads in the community. They will query all the results from all the labs to generate their database of infections in different areas. The system must remain responsive to analysis requests while processing the researcher's queries.

5.4 Marking

Persistence is a core functional requirement of your service. If your implementation does not save analysis requests, with their associated images, 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 your service. 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 analysis requests for which it returned a success response to the client. 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 **10%** from the 35% of the grade for the assignment. This is based on successful implementation of the given API specification and the ability to use the provided tool in your implementation.

Deploying your service is worth **10%** from the 35% of the grade 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 remaining **15%** of the grade for the assignment. 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.

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. As some students may have an extension and then decide to make a late submission, you may **not** discuss details of a submission stage with other students until **two weeks** *after* the original submission date. i.e. You may not discuss how to

- implement **functionality** until *after* **April 18**th;
- **deploy** the service to the cloud until *after* **May 1**st; and
- implement **scalability** until *after* **May 23**rd.

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 root directory 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 root directory 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 10%	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 10%	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 15%	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 implemented poorly).	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.