

# M7024E Laboratory 4: Programming Cloud Services - RESTful APIs

Ameer Hamza, Wania Khan

December 24, 2021

## Lab Report

### Questions

**1. What are microservices? Describe in detail the pros and cons of microservices architecture by giving examples.**

Microservices are an architectural approach that structures an application as collection of small independent services that communicates over APIs. These microservices are highly maintainable, independently deployable and testable which allow easy scaling and fast development of applications. [1] [2]

There are several pros and cons of microservices architecture [3] [4] and few of them are discussed below:

1. **Independence:** as we know that in microservices architecture, each service is developed and deployed independently which makes it easier to update any one component and not the entire application.
  - Let's take an example of e-commerce web application, we have different microservices such as payment section, shopping cart, advertisements etc. So adding a new feature or updating any existing payment method does not require the entire e-commerce application to come down in order to deploy that update.
2. **Scalability:** since all the services are separate, it is easy to scale any specific service at appropriate times instead of the whole application.
  - Considering the same e-commerce example as above, during specific times of a year these websites have to scale their shopping cart services due to high demand which doesn't require the scaling of entire application.
3. **Flexibility:** since all the communication happens over the network, there is more flexibility in how each endpoint is composed.
  - For example, people with different skill sets can contribute to a project without operating outside of their areas of expertise.
4. **Robust:** applications developed using microservices architecture are more robust as a whole. Large applications mostly remains unaffected by the failure of a single service as there is a loose coupling between the services.
  - Assume an example of food restaurant application where an online service of ordering is down but their contact service is open which helps customers to order food by contacting customer service representatives.
5. **Complex communication:** as above mention that everything is now an independent service, it is needed to handle requests carefully between modules. The developers may have to deal with complications and need to write extra code to avoid disruption.
  - For example; when remote calls experience latency.

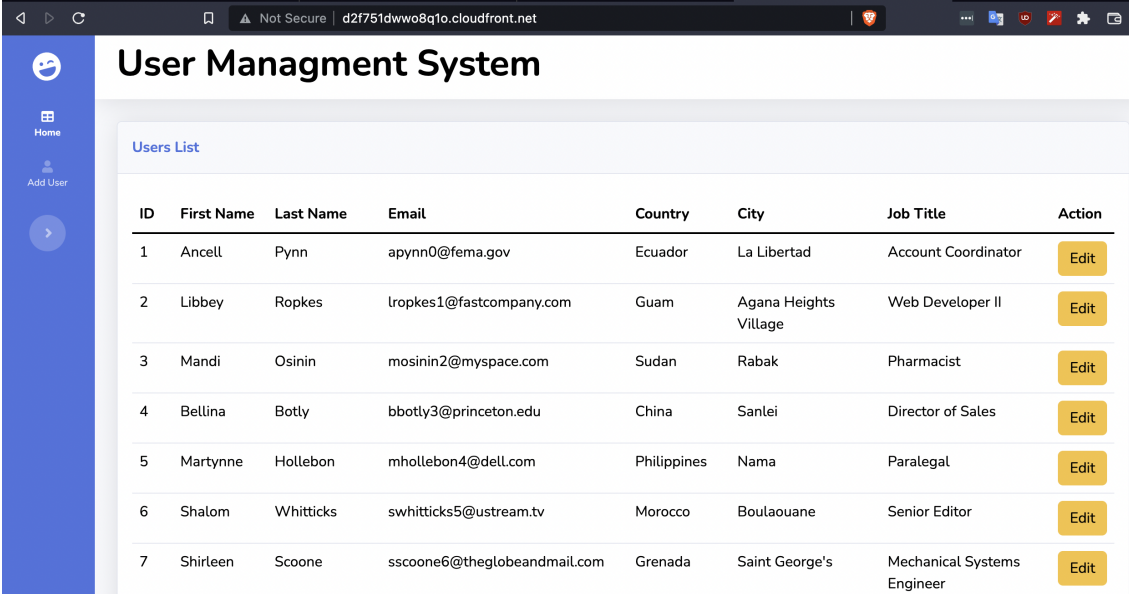
6. **Global Testing:** spinning up test environments is more involved with microservices-based applications due to increased number of nodes. As each dependent service has to be confirmed before testing can occur.
7. **Performance:** as it is known that communication happens over a network which is considerably slower than in memory. For many applications, it might not be a problem as network speeds improve, but this is something that should be taken into consideration.
8. **Refactoring:** refactoring an application across multiple services is harder than a monolith. For example, using different languages for different modules and components might cause issues when functionality from one service has to be ported over to another.

## Exercise

1. Develop a simple RESTful API for any application.
2. Use docker container for deployment of the service.
3. Use microservices as an architectural paradigm to design a simple service(s). Describe and document that service.
4. Create a RESTful API to implement your service.

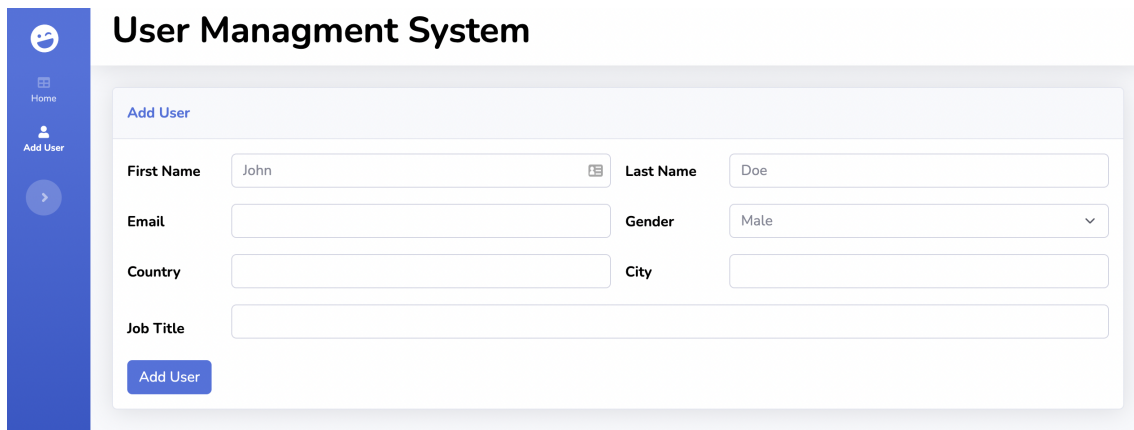
## Overview

For this task we developed a simple user management application which allows a user to list, add, modify and delete the details of the users. Application is built using bootstrap for front-end, flask and RESTful APIs for back-end and MySQL for the database.



ID	First Name	Last Name	Email	Country	City	Job Title	Action
1	Ancell	Pynn	apynn0@fema.gov	Ecuador	La Libertad	Account Coordinator	Edit
2	Libbey	Ropkes	lropkes1@fastcompany.com	Guam	Agana Heights Village	Web Developer II	Edit
3	Mandi	Osinin	mosinin2@myspace.com	Sudan	Rabak	Pharmacist	Edit
4	Bellina	Botly	bbotly3@princeton.edu	China	Sanlei	Director of Sales	Edit
5	Martynne	Hollebon	mhollebon4@dell.com	Philippines	Nama	Paralegal	Edit
6	Shalom	Whitticks	swhitticks5@ustream.tv	Morocco	Boulaouane	Senior Editor	Edit
7	Shirleen	Scoone	sscoone6@theglobeandmail.com	Grenada	Saint George's	Mechanical Systems Engineer	Edit

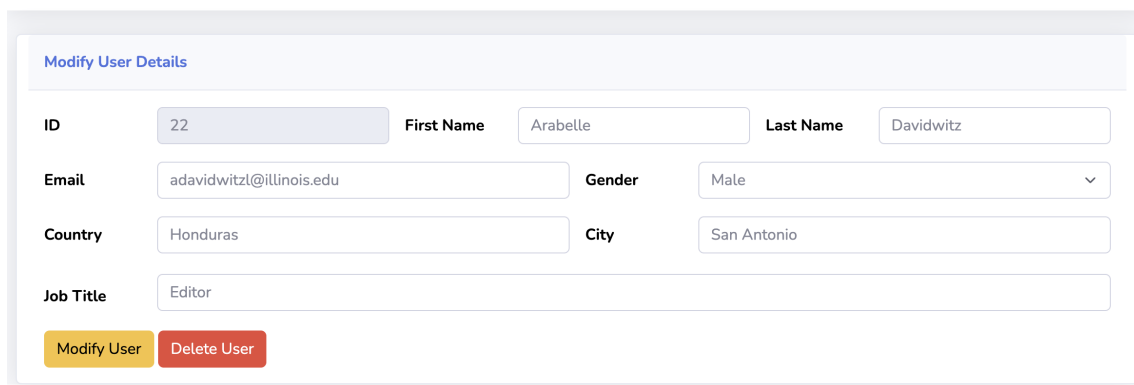
Figure 1: User Interface - Front Page



The image shows a web application titled "User Management System". On the left is a blue sidebar with a home icon, "Home", an "Add User" icon, "Add User", and a right arrow. The main content area has a light blue header "Add User". Below it is a form with fields for "First Name" (John), "Last Name" (Doe), "Email", "Gender" (Male), "Country", "City", and "Job Title". A blue "Add User" button is at the bottom left of the form.

Figure 2: User Interface - Add a User

## User Management System



The image shows a web application titled "User Management System". The main content area has a light blue header "Modify User Details". Below it is a form for editing user information. The fields are: "ID" (22), "First Name" (Arabelle), "Last Name" (Davidwitz), "Email" (adavidwitzl@illinois.edu), "Gender" (Male), "Country" (Honduras), "City" (San Antonio), and "Job Title" (Editor). At the bottom are two buttons: "Modify User" (yellow) and "Delete User" (red).

Figure 3: User Interface - Modify and Delete User

## RESTful API

Our application has the following endpoints:

1. `/api/v1/users` – GET →List all users
2. `/api/v1/users/<id>`– GET →List details of a particular user
3. `/api/v1/users` – POST →Create a new user
4. `/api/v1/users/<id>`– PUT →Modify details of a particular user
5. `/api/v1/users/<id>`– DELETE →Remove a user

```

# API Routes

# List all users
@app.route("/api/v1/users", methods=["GET"])
> def index(): ...

# List a single user
@app.route("/api/v1/users/<id>", methods=["GET"])
> def get_user_by_id(id): ...

# Add a new user
@app.route("/api/v1/users", methods=["POST"])
> def create_user(): ...

# Modify a user
@app.route("/api/v1/users/<id>", methods=["PUT"])
> def update_user_by_id(id): ...

# Delete a user
@app.route("/api/v1/users/<id>", methods=["DELETE"])
> def delete_user_by_id(id): ...

```

Figure 4: API Routes

## Architecture

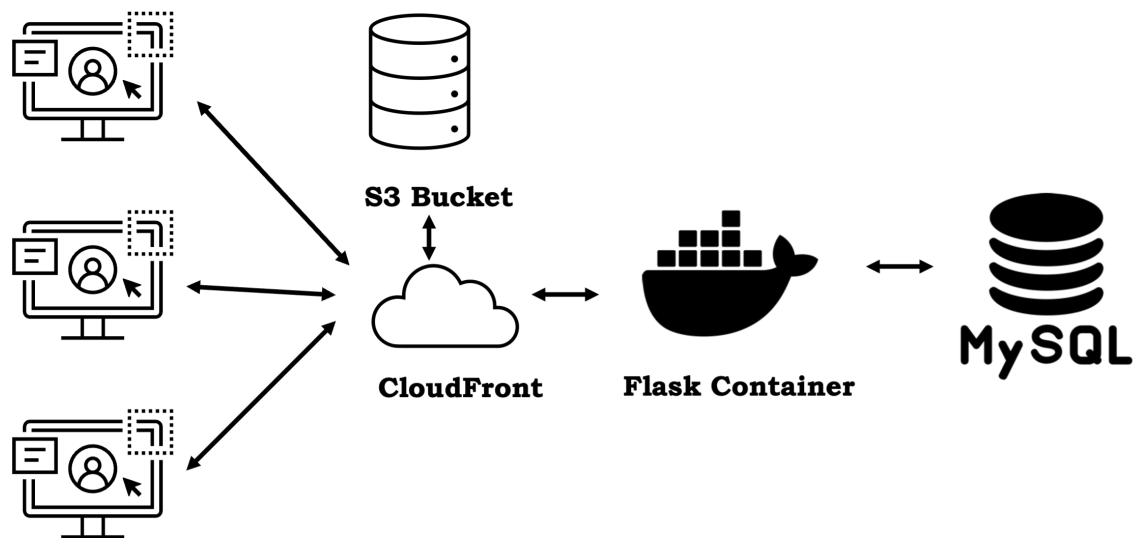


Figure 5: Architecture

There are three main components of our applications as show in figure 5:

## Front End

The front-end of the application consists of HTML, CSS and JavaScript file. Since these are static files, they are deployed on S3 bucket followed by CloudFront to handle use traffic dynamically. Static files communicate with back-end using Restful APIs.

### Details

Distribution domain name  
 d2f751dww08q1o.cloudfront.net

ARN  
 arn:aws:cloudfront::351880275390:distribution/E3ADTIPNXJJUSG

Last modified  
 Deploying

### Settings

Edit

Description  
Flask App Hamza

Alternate domain names  
-

Standard logging  
Off

Price class  
Use all edge locations (best performance)

Cookie logging  
Off

Supported HTTP versions  
HTTP/2, HTTP/1.1, HTTP/1.0

Default root object  
-

AWS WAF  
-

Figure 6: CloudFront

flask-app-1

Info

Objects

Properties

Permissions

Metrics

Management

Access Points

Objects (4)

Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 inventory](#) to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. [Learn more](#)

Copy S3 URI
 Copy URL
 Download
 Open
Delete
Actions

Create folder
Upload

☐ Show versions

< 1 >

<input type="checkbox"/>	Name ▲	Type ▼	Last modified ▼	Size ▼	Storage class ▼
<input type="checkbox"/>	<a href="#">add_user.html</a>	html	December 23, 2021, 18:12:28 (UTC+01:00)	8.4 KB	Standard
<input type="checkbox"/>	<a href="#">assets/</a>	Folder	-	-	-
<input type="checkbox"/>	<a href="#">index.html</a>	html	December 23, 2021, 18:12:28 (UTC+01:00)	4.4 KB	Standard
<input type="checkbox"/>	<a href="#">modify_user.html</a>	html	December 23, 2021, 18:12:28 (UTC+01:00)	8.4 KB	Standard

Figure 7: S3

## Back End / Docker Container

We have used Python Flask for the back-end and to make Restful API. Once the application is complete it is dockerized using docker file and the image is deployed on EC2. Ideally AWS ECS should be used as it gives a lot of flexibility for scaling the application. Since we didn't have the permission for ECS, docker was installed on EC2 and the image was deployed. Link to docker image: <https://hub.docker.com/r/ameerhamza360/flask>

```

1  # init a base image (Alpine is small Linux distro)
2  FROM python:3.9-alpine
3
4  # FROM python
5
6
7  RUN apk update && apk add gcc g++ make bash && rm -rf /var/cache/apk/*
8
9
10 # define the present working directory
11 WORKDIR /flask-img
12 # copy the contents into the working dir
13 ADD . /flask-img
14 # run pip to install the dependencies of the flask app
15 RUN pip install -r requirements.txt
16 # define the command to start the container
17
18 EXPOSE 5000
19
20
21 CMD ["python", "app.py"]

```

Figure 8: Docker File

## Database

MySQL is used for this application though any relational database can be used without any code change since ORM is used in the Flask App. Relational database is used since our application has a fixed structure though NoSQL database will work equally well. MySQL server is deployed in a different EC2 instance. Database is not deployed using docker container as it introduces unnecessary complexity without relative benefit. Ideally we wanted to use Amazon RDS as it allows scaling of the database very easily. However, we didn't have the permission to use that service.

<input type="checkbox"/>	Hamza-MySQL	i-0f44c1e925145a4df	Running	🔍	t3.micro	2/2 checks passed
<input type="checkbox"/>	Hamza-Docker	i-0ad2e0331161512c1	Running	🔍	t4g.nano	2/2 checks passed

Figure 9: EC2 Instances for Docker and MySQL

## References

- [1] “Microservices.” [Online]. Available: <https://aws.amazon.com/microservices/>
- [2] “Microservices architecture.” [Online]. Available: <https://microservices.io/>
- [3] “Pros and cons microservices architecture.” [Online]. Available: <https://www.spkaa.com/blog/top-4-pros-cons-microservices-architecture/>
- [4] “Pros and cons of microservices.” [Online]. Available: <https://cloudacademy.com/blog/microservices-architecture-challenge-advantage-drawback/>