Laboratory 3: Programming Cloud Services - Compute Services

M7024E Cloud Services - Assoc. Prof. Karan Mitra

Josue Becerra Rico
Mohammad Messbah Uddin

1. Objectives

The objective of this lab is to:

- Understand methodologies for developing Cloud application and services.
- Program a simple RESTful API.

2. Questions

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

Microservices are an architectural approach to build application made of independent services which can communicate via APIs. Traditionally the applications were build using a monolithic approach where all processes ran as a single service. Through microservices, the features of an application can be divided into small components which makes the application scalable, resilient and easily upgradable.

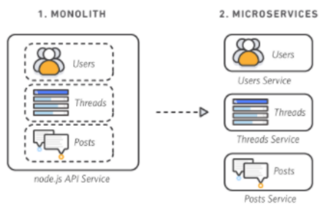


Figure 1: Monolithic service to Microservices (source: AWS)

Pros of using microservices:

- **Agility**: Microservices allow small teams to work on a particular service that they take the ownership on. This allows the team to work independently and fast which eventually shortens the development life cycle.
- **Flexible Scaling**: Each service can be independently scaled when required enabling teams to measure infrastructure needs, cost and maintain availability.
- **Easy Deployment**: Since each service is independent, integrating, updating and deploying a service is easy.
- **Technological Freedom**: Teams have the freedom to use the best available tool to build their service. This not only gives freedom to the developers, but also makes the application better.
- **Reusable Code**: Sometimes a code written for a certain function might be used as a building block for another feature. This saves time while adding new features.

- **Resilience**: The independence of the services makes the application failure-resistant. One service's failure could cause the whole system to fail in monolithic architecture but in microservices, one single service's failure doesn't cause the whole system to crash.

Cons of using microservices:

- **Communication**: An application may provide lots of microservices which need to communicate among themselves. Building this communication becomes very complex sometimes.
- **Testing**: Although unit testing is easier in microservices, integration testing is not. Developers can not test an entire system from individual machines.
- **Critical Interface Control**: Microservice based applications are dependent on APIs. A change in certain API may affect the application using that API if the change is not backward compatible.
- **High Upfront Cost**: To use microservices, sufficient hosting infrastructure and a higher number of developers are required who will work independently to build separate services. This increases the need of developer resources. Companies using monolithic architecture may need to reinvest for the change.

3. Exercises

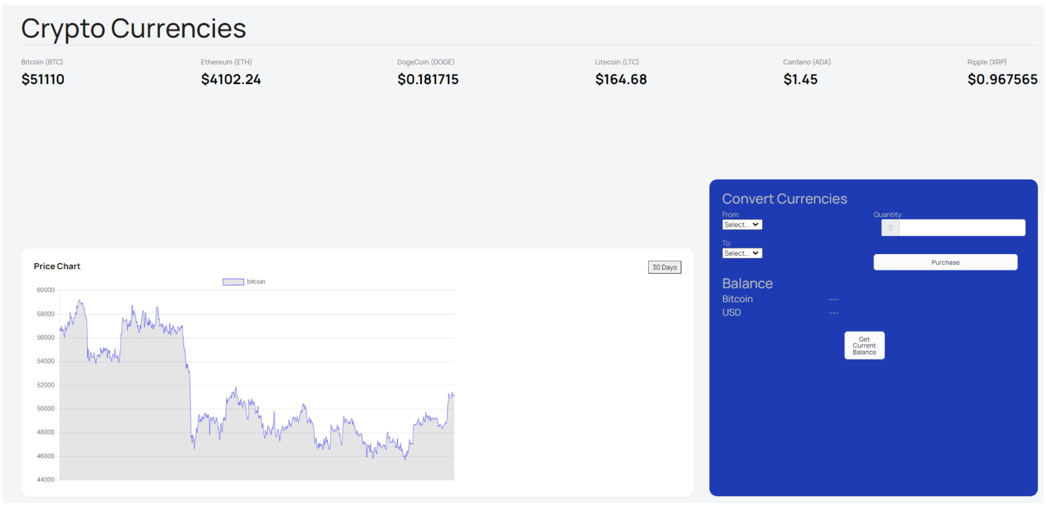
3.1. Tasks

- Use microservices as an architectural paradigm to design a simple service(s). Describe and document that service.
- Create a RESTful API to implement your service. Your RESTful API should implement at least two verbs, for example GET and POST. For GET, you should implement at least one path.

3.2. Our Solution

We have built an WebApp that can act as both trading simulator for Bitcoin and price tracking website for crypto currencies. The key features of our website is

- Realtime Crypto currency price tracking.
- Historical price checking through chart (1 Day, 7Days and 30 Days).
- Investing simulator in which user start with a certain amount of cash and can trade BTC to USD and vice versa with realtime data.


Crypto Currencies

Bitcoin (BTC)	Ethereum (ETH)	DogeCoin (DOGE)	Litecoin (LTC)	Cardano (ADA)	Ripple (XRP)
$51110	$4102.24	$0.181715	$164.68	$1.45	$0.967565

Convert Currencies

From: Select...

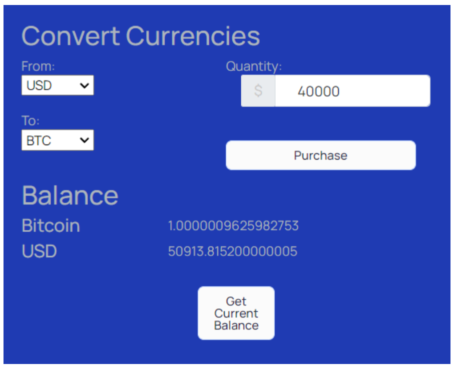
To: Select...

Quantity: $

Balance

Bitcoin	---
USD	---

Figure 2: Website



40000

Balance

Bitcoin 1.0000009625982753

USD 50913.815200000005

Figure 3: Balance and Currency conversion

3.2.1 Language and Framework

We used Python and Flask for the REST framework. Javascript(chart.js, ajax), HTML and CSS are used in the frontend. For database, we used mongoDB since it is well suited for microservice architecture providing flexible schema, automation, redundancy and scalability.
For deployment, Docker container is used. We also used nginx as reverse proxy to point the port 80 to the port where flask is running.

3.2.2 Architecture

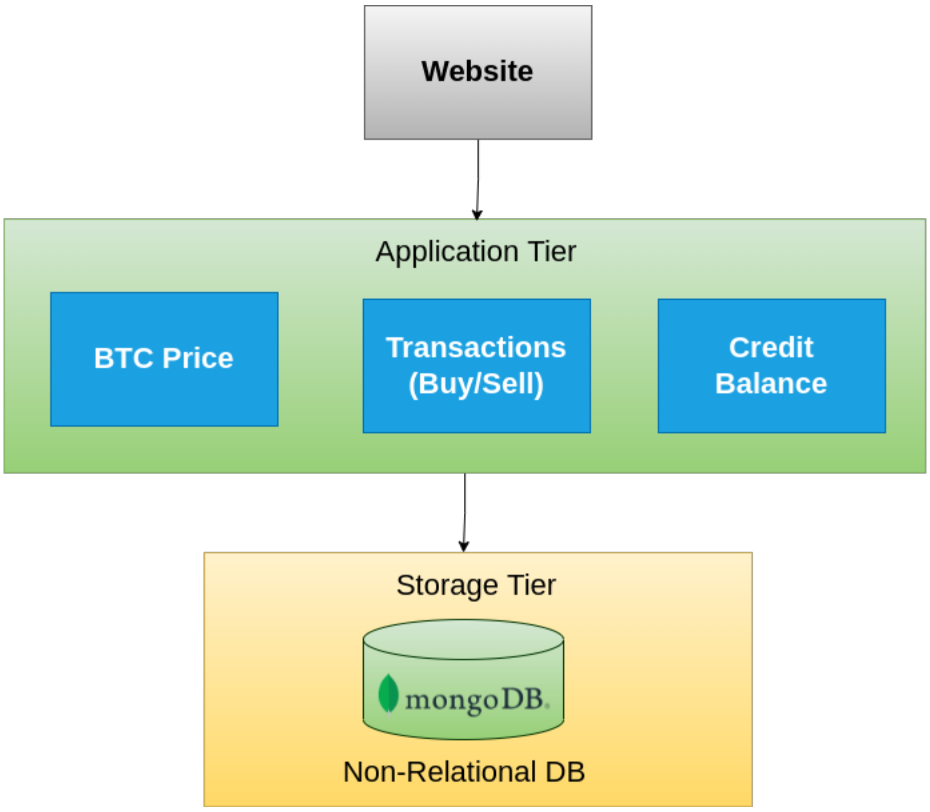


Figure 4: Cloud Desig Architecture.

The website (user frontend) gets the value of BTC through the API-1 that is calling CoinGekoAPI, then the website stores the transaction by posting in the API-2, which manages the non-relational database, the website can also get the latest transaction to display the current balance (Figure 5).

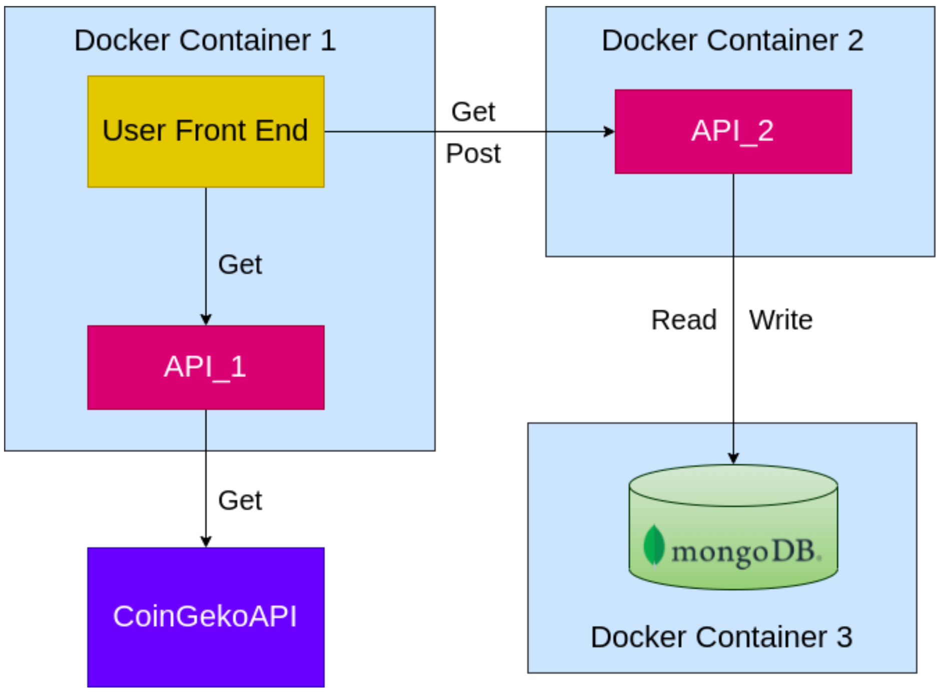


Figure 5: Architecture of Docker containers.

API_1 fetches crypto currency price using a GET request from coingecko.com (A popular website for crypto currency related data). The data is then processed to show price and chart in user Front End which is deployed in Docker container 1. Container 2 holds API_2 which is used to read and write data in the database. The database(mongoDB) is deployed in the 3rd container.

4. Conclusion and Future Works

Microservices allowed us to independently build parts of the application and easily merge them together. How microservices work and how advantageous building application using microservices are well understood through this lab. Our project includes everything expected to be done in this lab. We have implemented both GET and POST methods, used a database and also used external APIs along with the ones we built. In future we hope to add some extra features like logging in and keeping track of user's custom portfolio. Building a trading bot to trade the assets by itself using price prediction is also in our goal.

A. Website and APIs:

http://ec2-16-170-218-138.eu-north-1.compute.amazonaws.com/
http://ec2-16-170-218-138.eu-north-1.compute.amazonaws.com/get-price
http://16.170.218.138:5001/balance