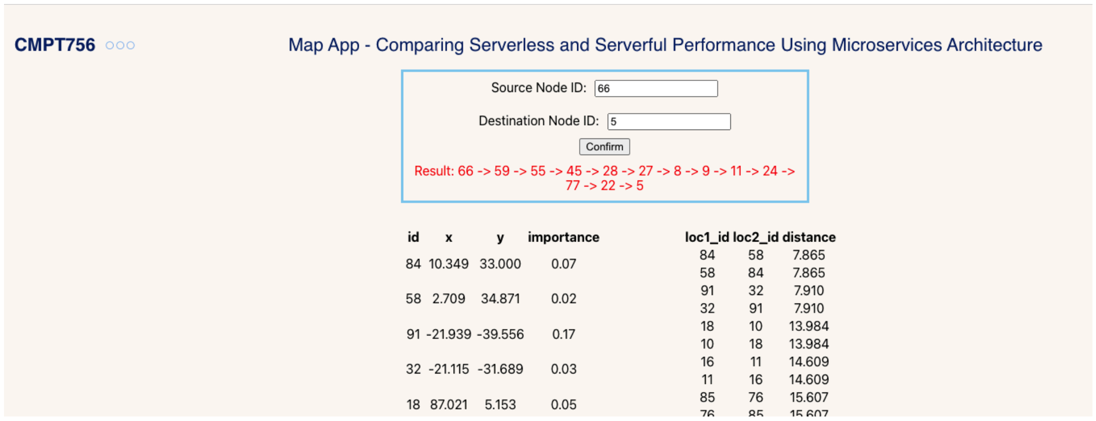
CMPT756 Project Outcomes

<u>Project Title:</u> (1) compare serverless and serverful performance for an application using a microservices architecture.

<u>System Design:</u>

In our system design, we created a dynamic map application that enables users to view and navigate within a virtual map. The term "dynamic" refers to the simulation of real-world changes in road conditions, such as roadblocks and congestion. The User-agent interacts with the system by issuing read map or navigation requests through a dedicated web frontend, whereas the Measurement agent issues map update requests.


CMPT756

Map App - Comparing Serverless and Serverful Performance Using Microservices Architecture

Source Node ID: 66

Destination Node ID: 5

Result: 66 -> 59 -> 55 -> 45 -> 28 -> 27 -> 8 -> 9 -> 11 -> 24 -> 77 -> 22 -> 5

id	x	y	importance
84	10.349	33.000	0.07
58	2.709	34.871	0.02
91	-21.939	-39.556	0.17
32	-21.115	-31.689	0.03
18	87.021	5.153	0.05

loc1_id	loc2_id	distance
84	58	7.865
58	84	7.865
91	32	7.910
32	91	7.910
18	10	13.984
10	18	13.984
16	11	14.609
11	16	14.609
85	76	15.607
76	85	15.607

Figure 1: Web frontend for users to enter “starting point” and “destination”

The following is a detailed description of the microservices.

1. *Navigation Service:* It handles navigation queries and returns the shortest node path upon user request. Navigation data is stored in the Navigation Store, which is optimized for navigation queries.
2. *Geographical Information System (GIS)*: This service manages the reads and writes of the most up-to-date map definition in the Map Definition Data Store.
3. *Map Update Service:* This service publishes the most up-to-date map definition to a single-partition AWS Kinesis Data Stream before other services receive the most up-to-date map definition. The map update process is asynchronous, meaning that the up-to-date map data is not immediately reflected, they are temporarily pushed to a stream which will process the updates later.

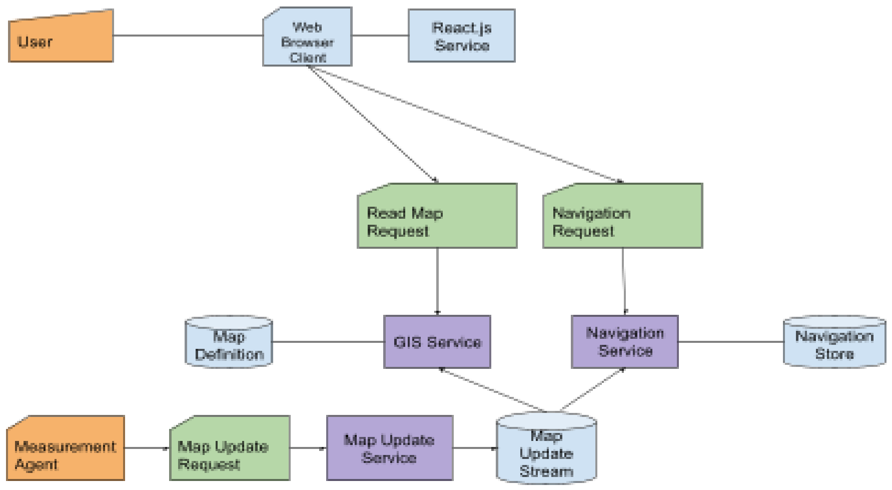


Figure 2: System Design for the Dynamic Map Application

Implementation:

Our map application is built using a combination of Python and JavaScript programming languages. The frontend is developed using React.js framework, while Flask.py serves as the backend API service framework. Our data is stored using PostgreSQL on RDS. In addition, we leveraged the protocol buffer technology to transfer map update messages through the Kinesis stream.

For the service deployment, we opted for AWS Elastic Container Service (ECS) due to its support for both AWS Fargate and Amazon EC2 launch types. This enables us to create serverless and serverful clusters using a unified graphical interface. Compared to other options we have tested, such as Elastic Kubernetes Service (EKS) and Elastic Beanstalk, we found ECS is more straightforward for small-scale deployment that does not require advanced container orchestration features.

In the serverful implementation, EC2 offers complete customization over instances and configurations, allowing us to tailor our infrastructure to our specific needs. We established provisions for EC2 with five t2.small machines to accommodate our serverful requirements. In addition, we also set aside an EC2 instance with the autoscaling feature enabled, which permits the number of instances to automatically adjust from one to five in response to fluctuations in workload demands. Compared to the fixed instances, this approach should provide more optimal resource utilization for handling fluctuating demand. For the purpose of our study, having this additional instance will help us gain a better understanding of the differences between serverless and serverful deployments.

In the serverless implementation, Fargate's automatic scaling capability enables on-demand provisioning and adjusts resources dynamically to handle varying read requests from map users. It abstracts the underlying infrastructure management, allowing us to focus solely on the application.

Results and Measurement & Analysis:

Our evaluation criteria involve conducting load testing to assess the performance impact of both serverful and serverless implementations. To achieve this, we utilized Apache JMeter as our testing tool to measure the response time of the system. We simulated 1,000 map users sending read map requests to the Geographical Information System (GIS) and observed their respective response times at one-minute intervals over a 15-minute duration.

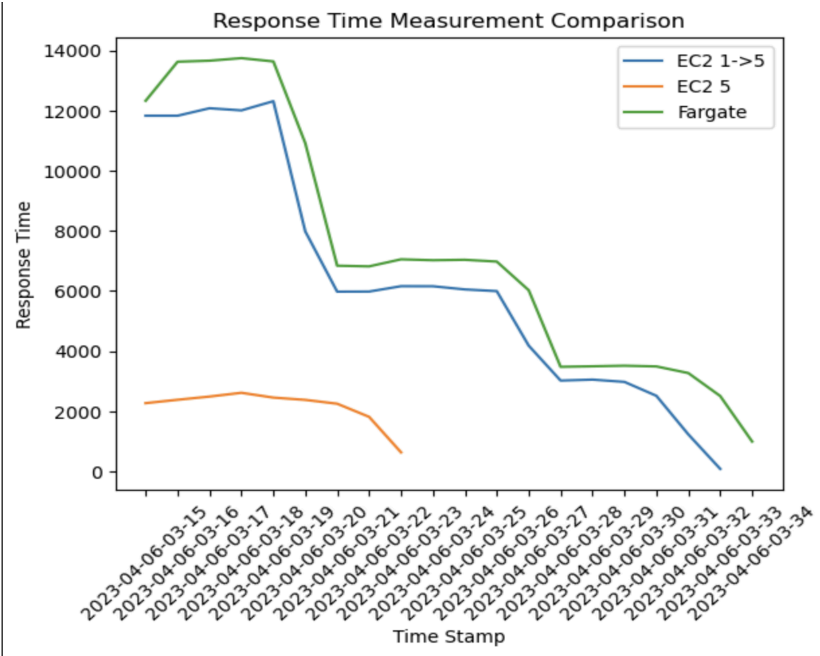


Figure 3: Response Time

The serverful EC2 implementation with fixed instances is capable of immediately handling the demand without the need for additional scaling time, resulting in lower response times compared to both Fargate and autoscaled EC2 instances. The waterfall patterns observed in Fargate and EC2 (with autoscaling from 1 to 5) indicate that both implementations require some time to scale in and adapt to the increasing user request demand. The autoscaling conditions for Fargate and autoscaled EC2 instances are identical, in which instances are scaled in when the average CPU utilization for each instance exceeds 50%. As a result, the response time for Fargate and autoscaled had a slower response time compared to fixed instances. We also observed that the response time for autoscaled EC2 instances is consistently faster than Fargate. This prompted us to investigate further what would happen when both implementations have scaled to the same number of 5 instances.

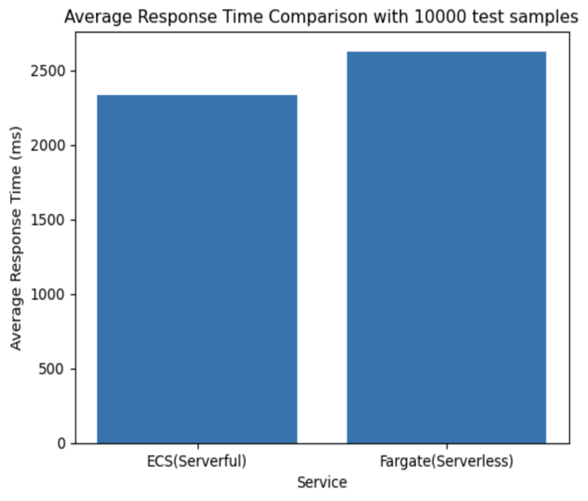


Figure 4: Response time for 5 instances in both implementations

The bar chart revealed that even when Fargate scaled to the same number of instances as EC2, it still exhibited slower response times, contrary to our initial expectation that they would be similar. We discovered that, in Fargate, tasks may be assigned to any available underlying host within the infrastructure. This allocation process is managed by AWS, which selects the most appropriate host at the time of deployment. It can also introduce network latency when the selected host might not be the closest or most optimal in terms of network connectivity, resulting in increased response times due to the additional distance between the Fargate task and the client or other services within the application. In contrast, EC2 instances are pre-allocated and can respond quickly to increasing demand.

In conclusion, for our dynamic map application, the choice of infrastructure depends on the anticipated workload and demand patterns. If the application is expected to experience small workloads with occasional bursts, Fargate's serverless approach would be a suitable option, as it offers easy management and can adapt to sudden changes in demand. However, if the application is expected to have consistent or larger workloads, EC2's serverful approach would be more appropriate, as it provides better response times and allows for more customization to optimize the infrastructure for the specific needs of the application. It is essential to evaluate the usage patterns of the map application to make an informed decision on the most suitable infrastructure solution.