Configuration Manual

MSc Research Project
MSc in Cloud Computing

Allen Joy
Student ID: 23202351

School of Computing
National College of Ireland

Supervisor: Shaguna Gupta

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Allen Joy
Student ID:	23202351
Programme:	MSc in Cloud Computing
Year:	2024
Module:	MSc Research Project
Supervisor:	Shaguna Gupta
Submission Due Date:	12/12/2024
Project Title:	Configuration Manual
Word Count:	943
Page Count:	10

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Allen Joy
Date:	11-12-2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	✓
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	✓
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	✓

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Allen Joy
23202351

1 Introduction

The document has the information regarding implementation of project and this document mainly focuses on the environment and the libraries used for the same. These libraries and software help the project to deliver its purpose to the best.

2 AWS

The project is deployed on AWS Cloud, which helps in managing and implementing the three microservices in an effective manner Services (n.d.).

3 Visual Studio Code

The project is developed on VS Code which is compatible with the project of microservices (Visual Studio Code, 2023).

4 Node.js

Node.js is a JavaScript runtime built into the Chrome V8 engine that allows developers to execute JavaScript on the server side *Introduction to Node.js* (n.d.). Due to its non-blocking, event-oriented processing, it is widely used to create scalable, fast and efficient web applications.

5 Summary Version Table

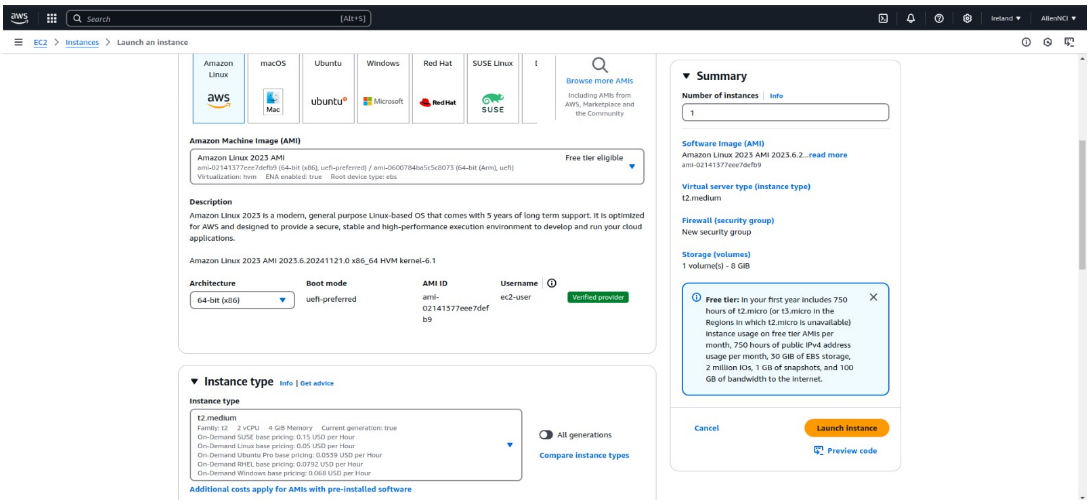
Software Name	**Version**	**Download Link**
Node.js	18	https://nodejs.org/en/blog/release/v18.20.2
Visual Studio Code	October 2023 (v1.84)	https://code.visualstudio.com/updates/v1.84

Table 1: Software Details

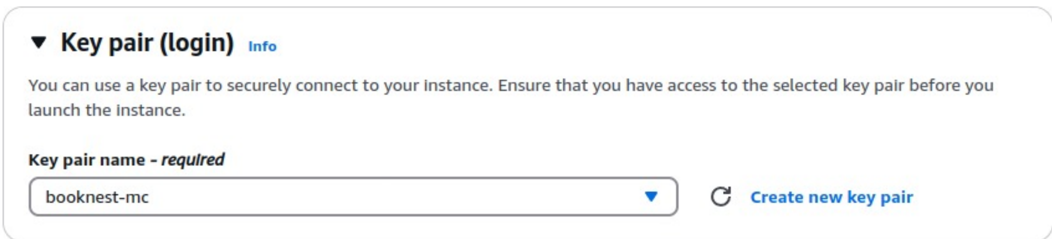
Library Name	Version	Purpose	Install Command
Express	4.17.1	To set up REST API	`npm install express`
CORS	2.8.5	To allow cross-origin requests	`npm install cors`
Day.js	1.11.13	For date management	`npm install dayjs`
Jaeger-Client	3.19.0	To track application traces	`npm install jaeger-client`
Prom-Client	15.1.3	To generate Prometheus metrics	`npm install prom-client`
Mongoose	8.5.2	Object Data Modeling (ODM) for MongoDB	`npm install mongoose`

Table 2: Library Details

6 Deployment in AWS



Log in to the AWS Management Console,then Go to the AWS Management Console and log in with your AWS credentials and Launch an Instance, then in the search bar, type ”EC2” and select the EC2 service, and Click on the ”Launch Instance” button. select the operating system that is Amazon Linux, then choose an Instance Type that is ”t2.medium” from the list shown in the instance types, which provides a good balance of CPU and memory for many general-purpose workloads.



To connect to an EC2 choose a Keypair.

Connect to the EC2 using SSH

```
$ ssh -i "booknest-mc.pem" ec2-user@ec2-34-244-188-242.eu-west-1.compute.amazonaws.com
```

Connect to the EC2 using SSH

```
[ec2-user@ip-172-31-38-223 ~]$ sudo yum install -y docker
```

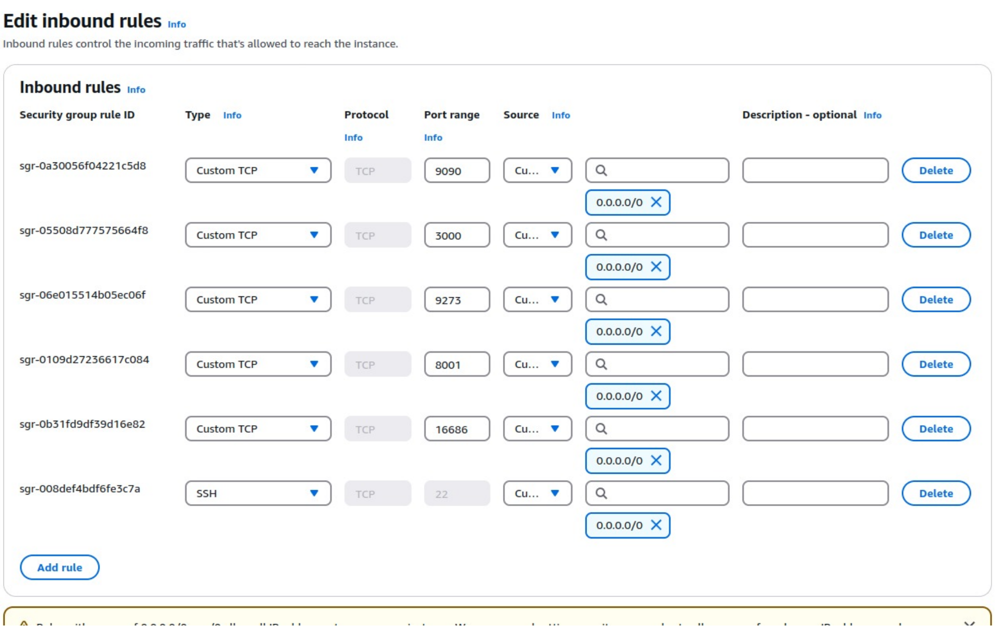
Install Docker

```
scp -i "booknest-mc.pem" -r ./booknest-mc ec2-user@ec2-34-244-188-242.eu-west-1.compute.amazonaws.com:~/
```

Copy the project folder inside the ec2 from the system.

```
[ec2-user@ip-172-31-38-223 ~]$ cd booknest-mc/
[ec2-user@ip-172-31-38-223 booknest-mc]$ docker-compose up -d
books is up-to-date
prometheus is up-to-date
grafana is up-to-date
jaeger is up-to-date
borrow is up-to-date
auth is up-to-date
otel-collector is up-to-date
telegraf is up-to-date
booknest-mc_nginx-proxy_1 is up-to-date
[ec2-user@ip-172-31-38-223 booknest-mc]$
```

Go to the copied folder and spin up the containers using docker-compose



Then go to the ec2's security groups and open these ports, 9090, 3000, 9273, 8001 and 16686 etc. so that our applications can be accessed from the browser.

7 Implementation of Tools

7.1 Prometheus:

Step 1 Prometheus Container Run

```yaml
prometheus:
  image: prom/prometheus
  container_name: prometheus
  ports:
    - "9090:9090"
  volumes:
    - ./prometheus-data:/prometheus
    - ./prometheus.yml:/etc/prometheus/prometheus.yml
  networks:
    - monitoring
```

Step 2 Setup prometheus configuration file. prometheus.yml, we write the endpoints from which to scrape metrices here in this file.

prometheus.yml

```yaml
global:
  scrape_interval: 15s

scrape_configs:
  - job_name: "auth"
    static_configs:
      - targets: ["auth:5000"]
  - job_name: "books"
    static_configs:
      - targets: ["books:5001"]
  - job_name: "borrow"
    static_configs:
      - targets: ["borrow:5002"]
  - job_name: "telegraf"
    static_configs:
      - targets: ["telegraf:9273"]
```

Step 3 Defining custom metrices.

```
const httpRequestDuration = new client.Histogram({
  name: "http_request_duration_seconds",
  help: "Duration of HTTP requests in seconds",
  labelNames: ["method", "route", "status_code"] as const,
});

const httpRequestCounter = new client.Counter({
  name: "http_requests_total",
  help: "Total number of HTTP requests",
  labelNames: ["method", "route", "status_code"] as const,
});

const activeUsersGauge = new client.Gauge({
  name: "active_users_total",
  help: "Total number of active users",
  labelNames: ["status"],
});

const httpResponseCounter = new client.Counter({
  name: "http_response_codes_total",
  help: "Total number of HTTP responses by status code",
  labelNames: ["status_code"],
});
```

7.2 Telegraf:

Step 1 Run telegraf as container. snippet from docker-compose file

```
telegraf:
  image: telegraf:latest
  container_name: telegraf
  ports:
    - "9273:9273"
  restart: always
  networks:
    - monitoring
  volumes:
    - ./telegraf.conf:/etc/telegraf/telegraf.conf:ro
  depends_on:
    - auth
    - books
    - borrow
  command: telegraf --config /etc/telegraf/telegraf.conf
```

Step 2 Create Telegraf.conf config file of Telegraf. Telegraf collects metrices from the exposed endpoitns and send to Prometheus

```
[[inputs.http]]
  urls = [
    "http://auth:5000/metrics",
    "http://books:5001/metrics",
    "http://borrow:5002/metrics"
  ]
  method = "GET"
  interval = "10s"

[[outputs.prometheus_client]]
  listen = ":9273"
```

7.3 Jaeger:

Step 1 Install packages

```json
"dependencies": {
  "@opentelemetry/api": "^1.9.0",
  "@opentelemetry/auto-instrumentations-node": "^0.53.0",
  "@opentelemetry/exporter-metrics-otlp-http": "^0.55.0",
  "@opentelemetry/exporter-metrics-otlp-proto": "^0.55.0",
  "@opentelemetry/exporter-trace-otlp-http": "^0.55.0",
  "@opentelemetry/resources": "^1.28.0",
  "@opentelemetry/sdk-metrics": "^1.28.0",
  "@opentelemetry/sdk-node": "^0.55.0",
  "@opentelemetry/sdk-trace-node": "^1.28.0",
  "@opentelemetry/semantic-conventions": "^1.28.0",
```

Step 2 Running Jaeger as container.

```yaml
jaeger:
  image: jaegertracing/all-in-one:latest
  container_name: jaeger
  ports:
    - "16686:16686"
    - "14268"
    - "14250"
  networks:
    - monitoring
  restart: always
```

Step 3 Using the tracer create span and using span inject custom traces to Jaeger.

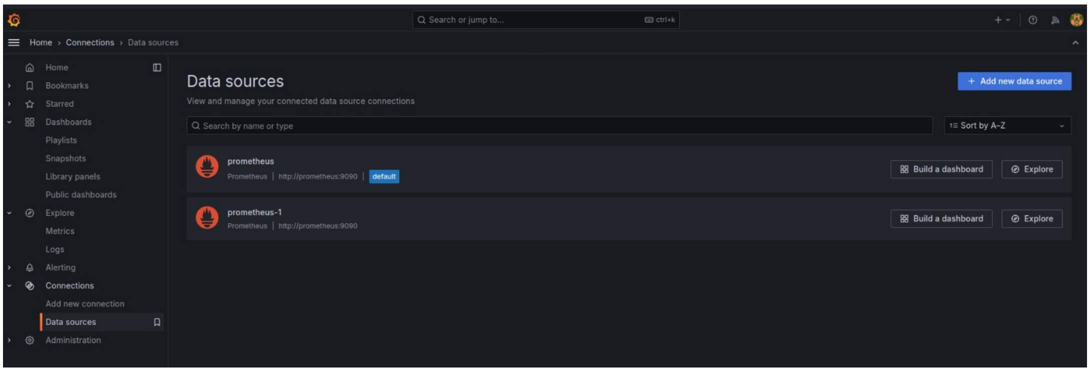
```typescript
export const signUpUser = async (req: Request, res: Response) => {
  const span = tracer.startSpan("signUpUser");
  try {
    const { fullName, userName, email, password } = req.body;
    const isExistingUser = await User.findOne({
      $or: [{ userName }, { email }],
    });

    if (isExistingUser && isExistingUser.email === email) {      Adarsh V S, last
      span.addEvent("Email already exists");
      return ResponseHelper.handleError(
        res,
        "Email already exists",
        undefined,
        StatusCodes.CONFLICT
      );
    }

    if (isExistingUser && isExistingUser.userName === userName) {
      span.addEvent("UserName already exists");
      return ResponseHelper.handleError(
        res,
        "UserName already exists",
        undefined,
        StatusCodes.CONFLICT
      );
    }
```

7.4 Grafana:

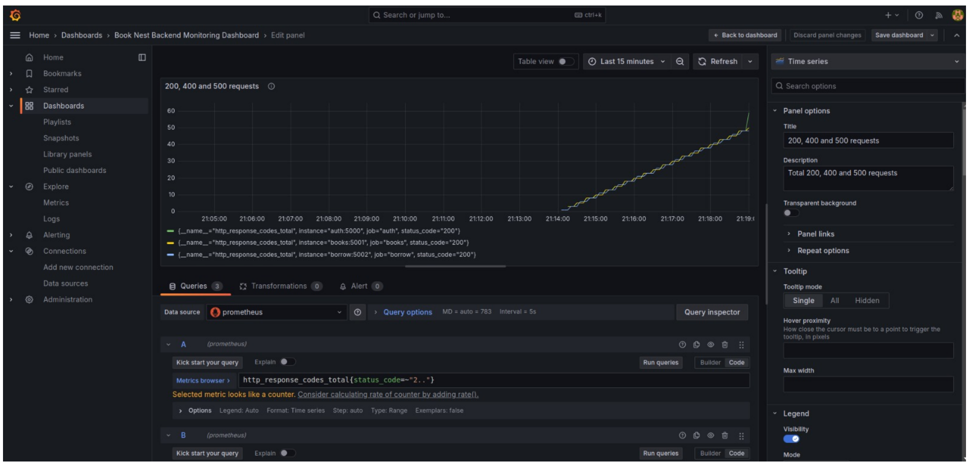
Step 1 Run grafana container

```yaml
grafana:
  image: grafana/grafana
  container_name: grafana
  ports:
    - "3000:3000"
  restart: always
  volumes:
    - grafana-data:/var/lib/grafana
    - ./provisioning:/etc/grafana/provisioning
  networks:
    - monitoring
```

Step 2 Go to datasources and add new datasource



Step 3 Type in the query and choose the visualization type to create the panel. then save dashboard



8 Evaluation

The evaluation part focuses on assessing the performance and resource utilization of lightweight monitoring tools deployed for a microservices architecture in cloud environmentsOyeniran et al. (2024). The tools used for monitoring the microservices were found very effective, and these findings are the results from the code implementation. The auth, books, and borrow services all metrics were scraped and stored in Prometheus server and Telegraf made sure that all scraping the metrics effectively and consistently. It visualize custom metrics such as HTTP request durations and active user counts in Grafana to understand how our system behaved under different loads. The dialog below may indicate a potential auth service bottleneck compared to other services; Jaeger's traces also indicated the transaction path in detail.

8.1 Overhead Analysis of Monitoring Tool

The analysis of CPU usage and memory consumption revealed that Prometheus and Telegraf are lightweight in terms of resource utilization which makes them suitable for resource-constrained environments. Due to the visualization capabilities of Grafana, and Jaeger, for its distributed tracing, demonstrated higher resource overhead, that is expected to give their functionalities.

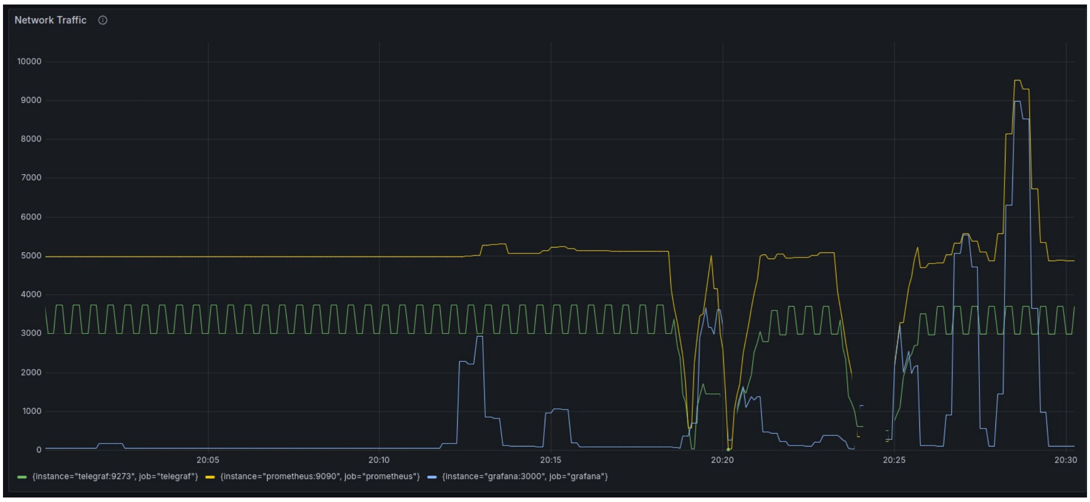


Figure 1: Network traffic of Monitoring Tools

8.2 Data Handling Capabilities in Real-Time

The scenarios which involves real-time data processing, the Prometheus exhibited low response times and consistent performance under dynamic workloads. With minimal data processing latency Telegraf also performed well,and attributed to its efficient push-based metrics collection mechanism. However, response times increased slightly during visualization updates with Grafana, while handling complex traces, Jaeger exhibited higher latency.

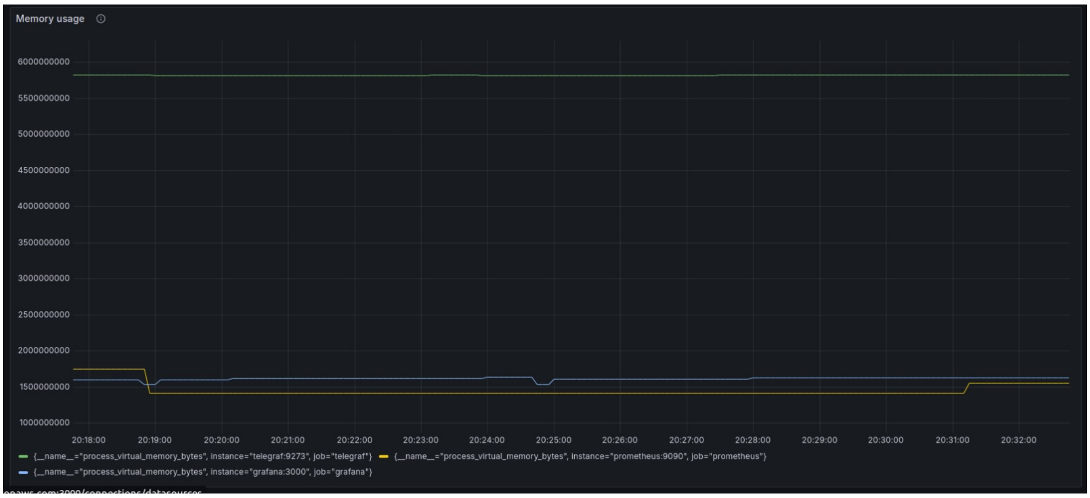


Figure 2: Memory usage of Monitoring Tools

8.3 Scalability of Monitoring Tools

As the number of monitored microservices increases, Prometheus and Telegraf are the best to demonstrate excellent scalability with consistent throughput and network utilization. The Grafana's performance was more variable, requiring optimization for scalability under heavy workloads.

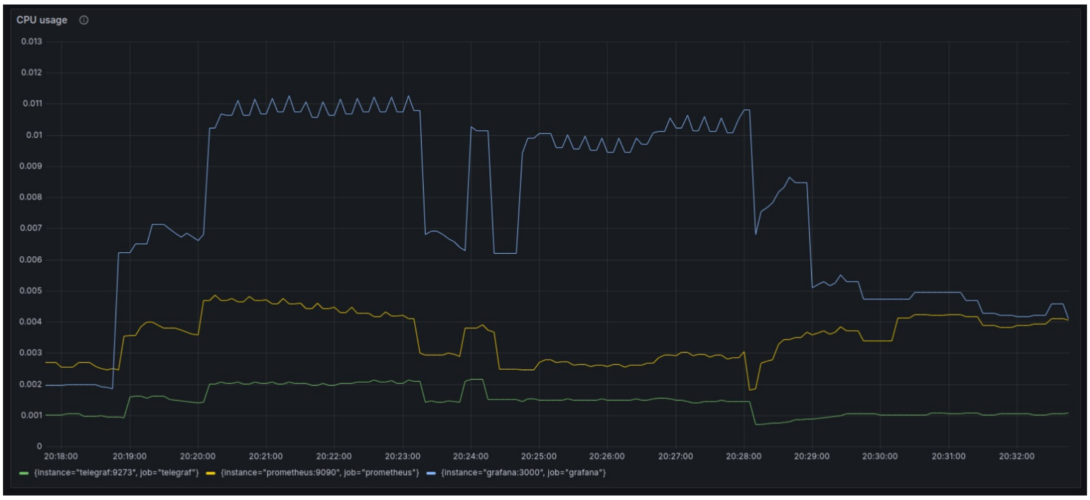


Figure 3: CPU usage of Monitoring Tools

8.4 Effectiveness of Distributed Tracing

The capabilities in distributed tracing were more effective in identifying bottlenecks and visualizing inter-service dependencies are in Jaeger *Getting Started* (2022). Instead of its higher resource utilization, its detailed traces provide mmore significant value for debugging and optimizing microservices performance.

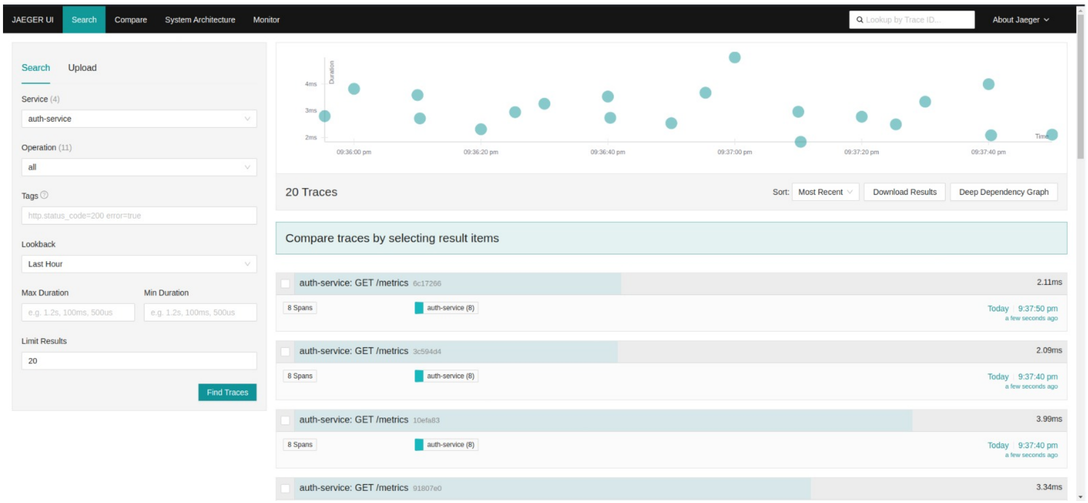


Figure 4: Jaegers Metrics for distributed tracing

References

Getting Started (2022).
URL: *https://www.jaegertracing.io/docs/1.64/getting-started/*

Introduction to Node.js (n.d.).
URL: *https://nodejs.org/en/learn/getting-started/introduction-to-nodejs*

Oyeniran, N. O. C., Okechukwu, A., Adams, N., Anthony, L. and Azubuko, F. (2024). Microservices architecture in cloud-native applications: Design patterns and scalability, *Computer Science IT Research Journal* **5**: 2107–2124.
URL: *https://www.researchgate.net/publication/383831564*$_M$*icroservices*$_a$*rchitecture*$_i$*n*$_c$*loud−native*$_a$*pplications*$_D$*esign*$_p$*atterns*$_a$*nd*$_s$*calability*

Services, A. W. (n.d.). What is amazon ec2? - amazon elastic compute cloud.
URL: *https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html*