FOURTH YEAR SEMESTER-2 MAJOR PROJECT

long-term internship Project Submitted

In partial fulfillment of the requirements for the award of the degree

Of

BACHELOR OF TECHNOLOGY

By

D. Vardhan

Roll No: R170060

Under the supervision of

P.Santosh Kumar

(Assistant Professor)

Department of Computer Science and Engineering

Rajiv Gandhi University of Knowledge Technologies, RK Valley Idupulapaya, Kadapa(Dist), Andhra Pradesh

Rajiv Gandhi University of Knowledge Technologies,RK Valley

Idupulapaya, Kadapa (Dist), Andhra Pradesh, 516330

CERTIFICATE

This is to certify that the project work titled " **Integration of tracing into cloud native application"** is a long internship submitted by D Vardhan (R170060) in the department of Computer Science and Engineering in partial fulfillment of requirements for the award of degree of Bachelor of Technology for the year 2022-2023 carried out the work under the supervision

Internal Guide	HEAD OF THE DEPARTMENT
P SANTOSH KUMAR	N SATYANANDRAM
(Assistant Professor)	**(Assistant Professor)**

Project Coordinator
M.MUNI BABU
(Assistant Professor)

RAJIV GANDHI UNIVERSITY OF KNOWLEDGE TECHNOLOGIES
(A.P. Government Act 18 of 2008)
RGUKT-RKValley, Kadapa Dist -516330

CERTIFICATE OF EXAMINATION

This is to certify that the work entitled, " **Integration of tracing into cloud native application**" is the bonafied work of D VARDHAN *(R170060)* Here by accord our approval of it as a study carried out and presented in a manner required for its acceptance Major of Bachelor of Technology for which it has been submitted. This approval does not necessarily endorse or accept every statement made, opinion expressed or conclusion drawn, as a recorded in this thesis. It only signifies the acceptance of this thesis for the purpose for which it has submitted.

P.Santosh Kumar	Examiner
Project Supervisor	Project Examiner
Dept. of CSE	Lecturer Dept. CSE
RGUKT IIIT RKValley	RGUKT IIIT RKValley

RAJIV GANDHI UNIVERSITY OFKNOWLEDGE TECHNOLOGIES

(A.P. Government Act 18 of 2008)
RGUKT-RKValley, Kadapa Dist –516330

DECLARATION

I am **D Vardhan(R170060)** hereby declare that the project report entitle,"**Integration of tracing into cloud native application"** done under the guidance of **P. Santosh Kumar** is submitted for minor project of **Bachelor of Technology** in **Computer Science and Engineering,** is an authentic record of our own work carried out under the supervision of

D. Vardhan, the Major Project january 2022 – April 2023 at RGUKT – RK Valley. We also declare that this project is a result of our own effort and has not been copied or imitated from any source. Citations from any websites are mentioned in the references.

The results embodied in this project report have not been submitted to any other university or institute for the award of any degree or diploma.

D . Vardhan (R170060)
Date: 03-05-2023
Place: RK Valley

ACKNOWLEDGEMENT

The satisfaction that accompanies the successful completion of any task would be incomplete without the mention of the people who made it possible and whose constant guidance and encouragement crown all the efforts success. We are extremely grateful to our respected Director, **Prof. K. Sandhya Rani** Mam for fostering an excellent academic climate in our institution. We also express my sincere gratitude to our respected Head of the Department **Mr.SatyaNandaram** Sir for his encouragement, overall guidance in viewing this project a good asset and effort in bringing out this project. We would like to convey thanks to my project guide **P.Santosh Kumar** Sir for his guidance, encouragement, co-operation and kindness during the entire duration of the course and academics.

My sincere thanks to all the members who helped me directly and indirectly for the completion of project work. I express my profound gratitude to all our friends and family members for their encouragement.

CONTENTS

Abstract

Security is a key factor and essesntial requirement for every application and software to maintain trust towards authentication that will be provided to users information and their access. Softwares that are being developed will be having security and quality issues, with that issues no software can’t make available to users. So, Here in Synopsys provides a static tool to analyze source code to identify application security and quality issues and this tool will be deployed on cloud with all resources and services that are required. Where cloud computing is technology used to organise and manage resources and services and provides platform for various computation, software access and data handling for betterment of availability for tool to users. Tool is developed using technologies like Docker, Kubernetes, Golang and Helm charts. In the world of microservices, most issues occur due to networking issues and the relations between the different microservices makes it a lot harder and tough to get to the root of an issue. To resolve these issues, we need to see which service sent what parameters to another service or a component and for this **integration of Jaeger Tracing into cloud-native application** was acted.

1.DOCKER – OVERVIEW

1.1 Why do we need docker ?

1. Compatibility/Dependency

Assume that there is an application that requires Web-Server (node.js), Database(MongoDB) and few other components. We will face lot of issues developing this application with all these components firstly, compatibility with the underlying operating system. We had to ensure that all these different services were compatible with the version of the operating system we were planning to use, there have been times when certain version of these services were not compatible with the OS and we had to go back and look for another OS that was compatible with all these different services. Secondly, we have to check the compatibility between these services and the libraries, dependencies on the OS. we may face issues where one service requires one version of a dependent library whereas another service required another version. Whenever architecture of our application changed over time we need to upgrade to newer versions of these components. So, If something changed we had to go through the same process of checking compatibility between these various components and underlying infrastructure.

2. Long setup time

Whenever, we have new developer we found it really difficult to setup a new environment. The new developers need to follow a large set of instructions and run hundreds of commands to finally setup their environment and they need to make sure they were using the right operating system, the right versions of each of the component. Moreover, developer need to set all these by himself/herself each time.

3.Different Dev/Test/Prod ENV

We will have different development, test and production environments and there might be situation where different developers use different OS and we couldn’t guarantee that the application that we were building would run the same way in different environments .

1.2 What can it do ?

- ➔ Docker can help us to modify or change these components without affecting other components and even modify underlying OS as required. (containerizing application)
- ➔ Docker can run each component in a separate container with it's own libraries andit's own dependencies, all on the same VM and OS. For these we need to build a docker configuration file once and all other developer can use that (usinf docker run command).

1.3 What is container ?

containers are completely isolated environments as in they can have their own processes or services, their own networking interfaces as similar to virtual machines but except they all share the same OS kernel.

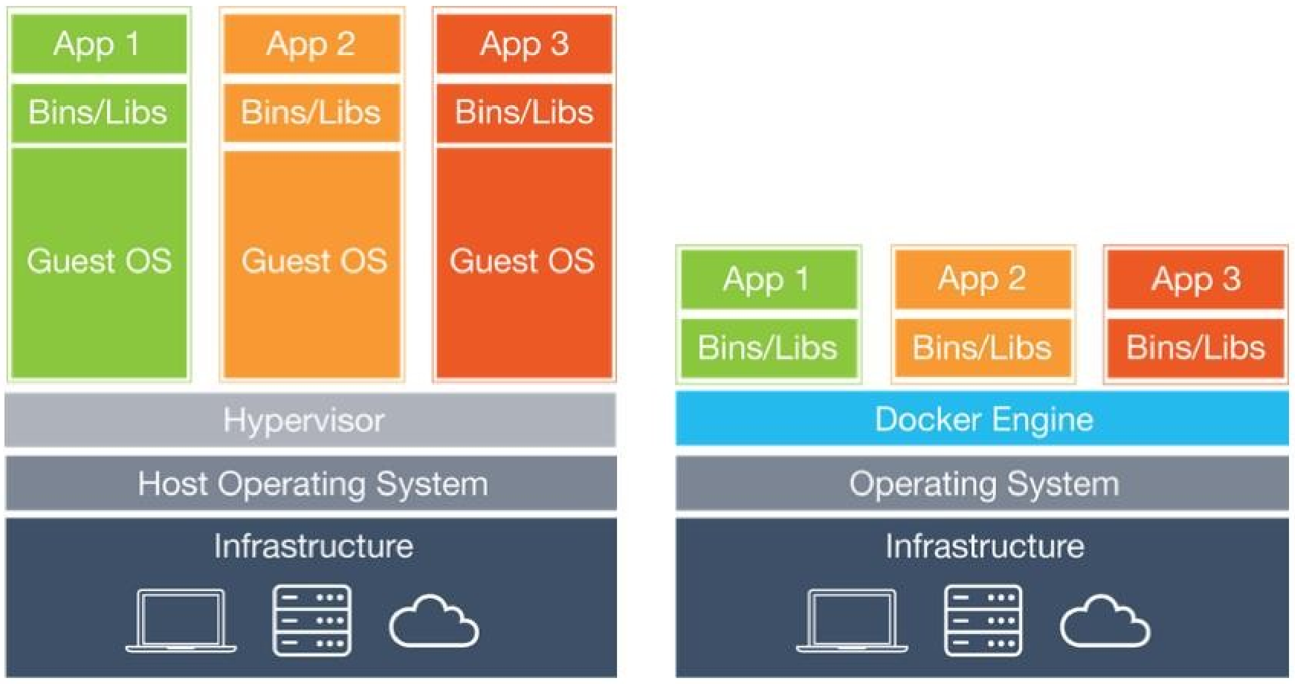


Fig - 1 : Container Architecture

1.4 DOCKER ARCHITECTURE :

1.<u>Docker Daemon</u>

Docker client uses commands and REST APIs to communicate with the Docker Daemon (Server). When a client runs any docker command on the docker client terminal, the client terminal sends these docker commands to the Docker daemon. Docker daemon receives these commands from the docker client in the form of command and REST API's request.

2.<u>Docker Host</u>

Docker Host is used to provide an environment to execute and run applications. It contains the docker daemon, images, containers, networks, and storage.

3. <u>Docker Registry</u>

Docker Registry manages and stores the Docker images.

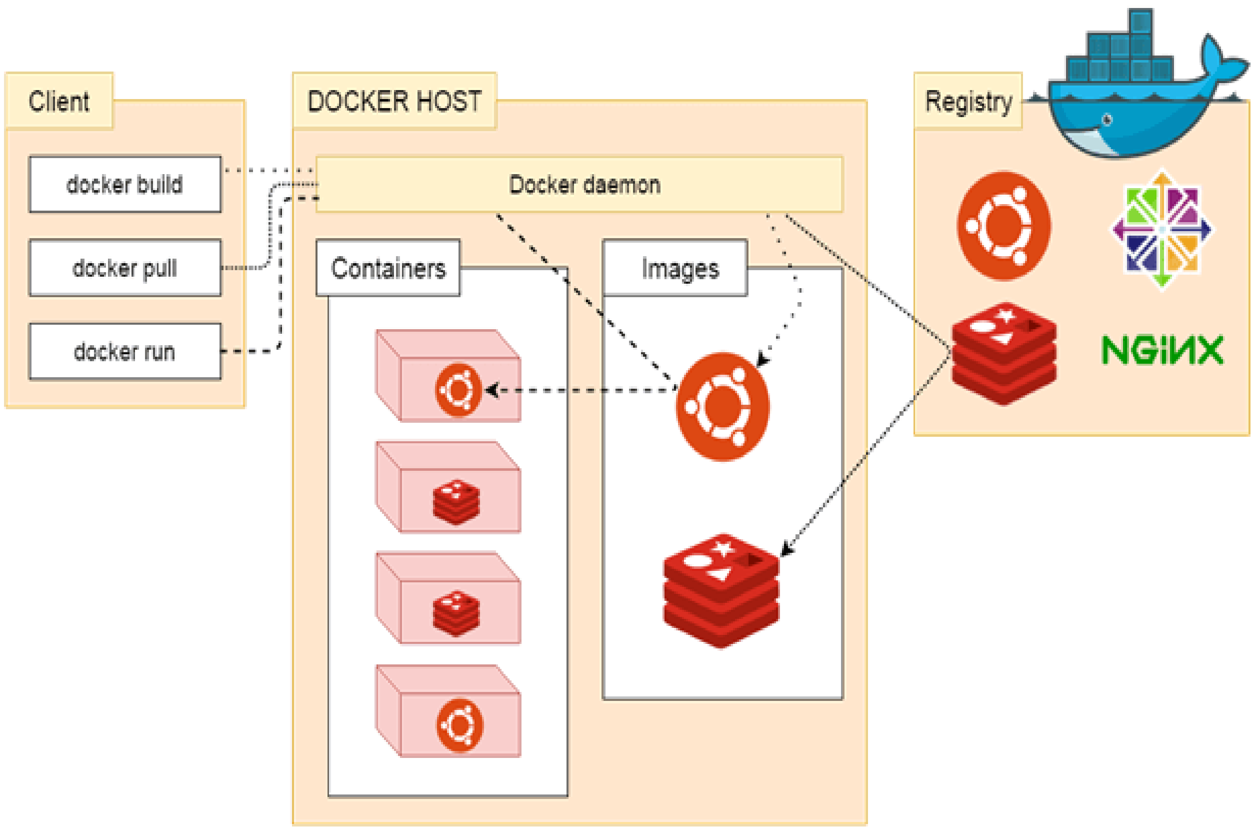


Fig - 2 : Docker Architecture

1.5 DOCKER COMMANDS :

- ➔ **docker pull -** This command is used to pull images from the docker repository.
  Usage: docker pull <image-name>

- ➔ **docker ps -** This command is used to list the running containers.

- ➔ **docker run -** This command is used to create a container from an image
  Usage: docker run -it -d <image-name>

- ➔ **docker stop**- This command stops a running container
  Usage: docker stop <container-id>

- ➔ **docker login -** This command is used to login to the docker hub repository

- ➔ **docker push -** This command is used to push an image to the docker hub repository
  Usage: docker push <image-name>

1.6 What does the Dockerfile contain ?

FROM – here defining the base image for the Dockerfile.
RUN - It is used to install the packages required for the parent image. We can execute shell commands also.
COPY - this copies files from a local source location to a destination in the Docker container
ENTRYPOINT - this instruction is used to set executables that will always run when the container is initiated To build Docker image we have to use the following command.
Command: docker build [options] <image-name>:<tag-name>

So, Docker is an open platform for developing, shipping, and running applications. Where Docker provides the ability to package and run an application in a loosely isolated environment called a container with libraries and dependencies required to run that code in any environment.

2. <u>KUBERNETES - OVERVIEW :</u>

2.1 Why do we need kubernetes ?

Containers are a good and easy way to bundle and run your applications. In a production environment, you need to manage the containers that run the applications and ensure that there is no downtime. For example, if a container goes down, another container needs to start.

Wouldn't it be easier if this behaviour was automated and handled by a system? That's how Kubernetes comes to the rescue! Kubernetes provides us an interface to run distributed systems smoothly. It takes care of scaling and failover for your application, provides deployment patterns, and more.

Few features that Kubernetes providing:

-->Load Balancing

-->Rollouts and Rollbacks

2.2 KUBERNETES ARCHITECTURE :

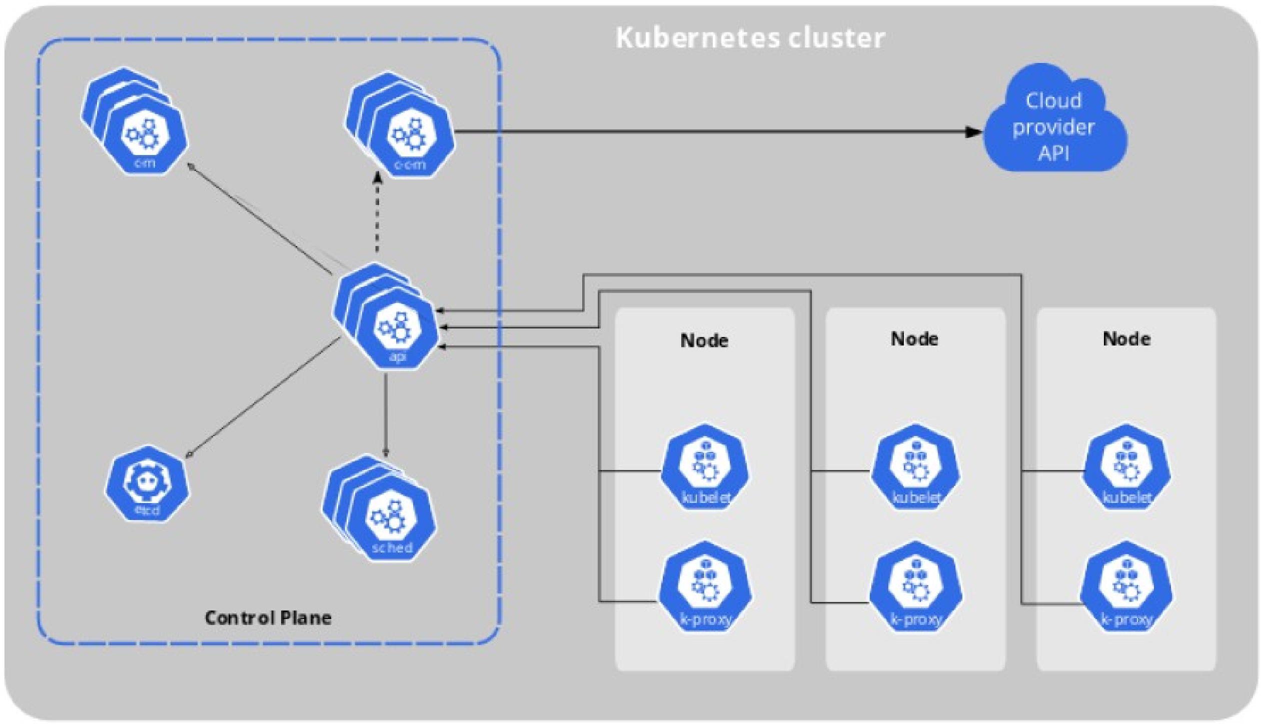


Fig - 3 : Kubernetes Architecture

Kubernetes is an architecture that offers a loosely coupled mechanism for service discovery across a cluster. A Kubernetes cluster has one or more control planes, and one or more compute nodes. Overall, the control plane is responsible for managing the overall cluster, exposing the application program interface (API), and for scheduling the initiation and shutdown of compute nodes based on a desired configuration. Each of the compute nodes runs a container runtime like Docker along with an agent, kubelet, which communicates with the control plane. Each node can be bare metal servers, or on-premises or cloud-based virtual machines (Vms).

2.3 Kubernetes basic components :

POD : Pods are the smallest deployable units of computing that you can create and manage in Kubernetes. Where it is a group of one or more containers, with shared storage and network resources, and a specification for how to run the containers.
Deployments : A Kubernetes Deployment tells Kubernetes how to create or modify instances of the pods that hold a containerized application. Deployments can help to efficiently scale the number of replica pods, enable the rollout of updated code in a controlled manner, or roll back to an earlier deployment version if necessary.
ReplicaSet : A ReplicaSet is a Kubernetes object that ensures there is always a stable set of running pods for a specific workload. The ReplicaSet configuration defines a number of identical pods required, and if a pod is evicted or fails, creates more pods to compensate for the loss.

2.4 Kubernetes basic kubectl commands:

- ➔ **Kubectl get** : This command is capable of fetching data on the cluster about Kubernetes resources.
- ➔ **kubectl describe** : Describes any particular resource in kubernetes.Shows details of resource or a group of resources.
- ➔ **kubectl apply** : It has the capability to configure a resource by file or stdin.
- ➔ **kubectl delete** : Deletes resources by file name, stdin, resource and names.
- ➔ k**ubectl exec** : This helps to execute a command in the container.
- ➔ **kubectl port-forward** : They are used to forward one or more local port to pods.

So , Kubernetes, also known as K8s and container orchestration technology , is an opensource system for automating deployment, scaling, and management of containerized applications. It groups containers that make up an application into logical units for easy management of services.

3. JAEGER TRACING – OVERVIEW

3.1 Why do we need jaeger tracing ?

In the world of microservices, most issues occur due to networking issues and the relations between the different microservices. A distributed application architecture (monolith application) makes it a lot harder and tough to get to the root of an issue. To resolve these issues, we need to see which service sent what parameters to another service or a component (DB, Web-Server, Cache-Service,Storage-Service and etc...) .

For this we can follow, Distributed tracing, also known as Distributed Request Tracing, is a technique for monitoring microservices-based applications from frontend devices to backend services and databases. Its purpose is to help developers identify performance issues by profiling and monitoring modern applications built using microservices and/or cloud-native applications.

You can collect data on each request using a distributed tracing tool, which will enable you present, analyse, and visualize the request in detail. You can observe each step that a request takes and how long each step takes using these graphic representations. Developers can look at this data to discover where the system is encountering bottlenecks and latencies and can figure out what's causing them that issues.

3.2 What is Jaeger Tracing?

Jaeger is an open-source distributed tracing platform created by Uber back in 2015. It visualizes the traces and spans data collected form services and components in Jaeger UI and for that we uses jaeger collector, jaeger agent and jaeger query.

1. Jaeger data model is compatible with OpenTracing — which is a specification that defines how the collected tracing data would look.

2. spans with OpenTelemetry and send them to Jaeger tracing for visualization.

3. Most importantly, Jaeger works with spans and traces.

4. A span represents a unit of work in an application(HTTP request, call to a DB, etc) and is Jaeger's most basic unit of work. A span must have an operation name, start time, and duration.

5. A trace is a collection/list of spans connected in a parent/child relationship.Traces specify how requests are propagated through our services and other components .

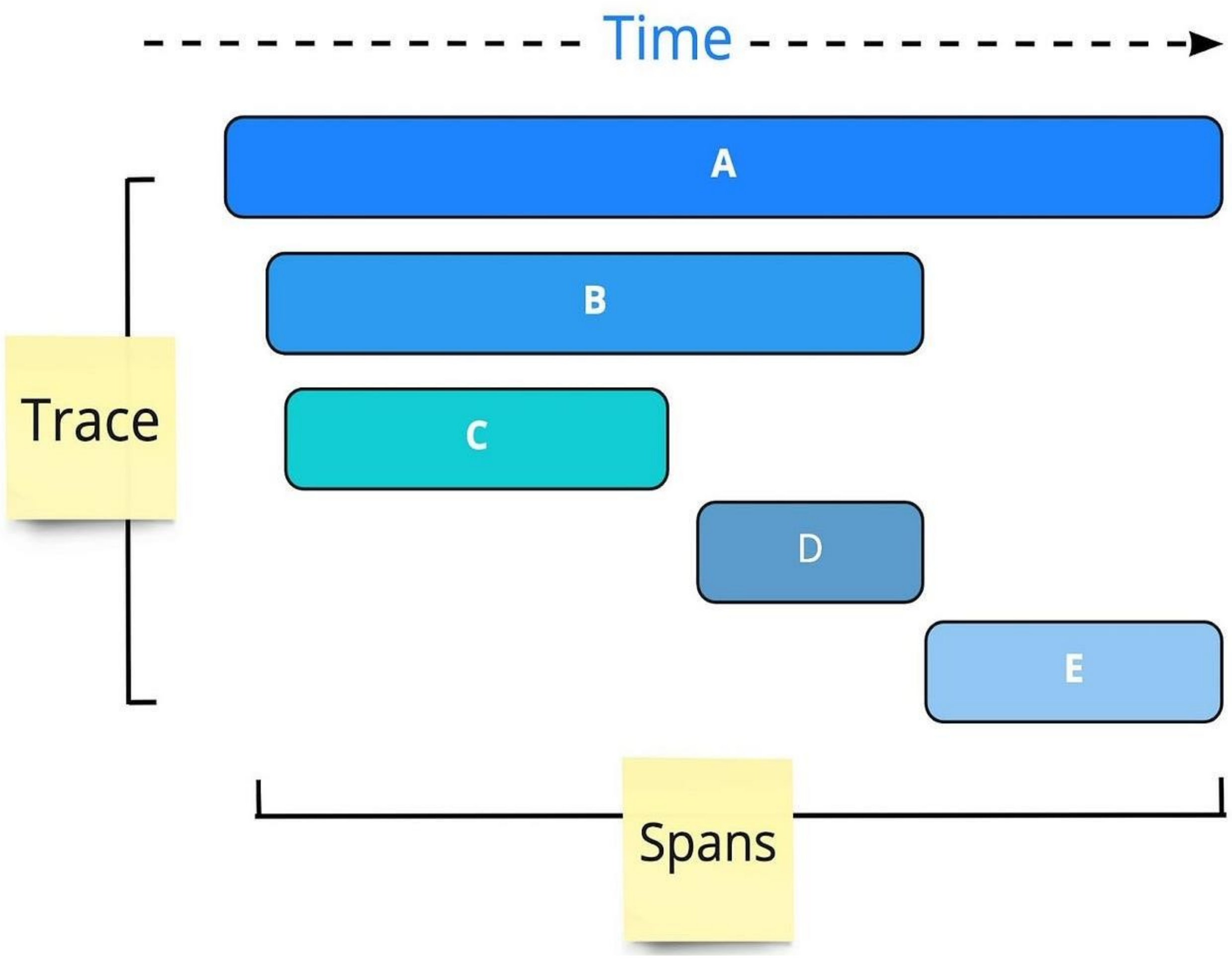


Fig - 4 : Traces and Spans

3.3 Jaeger Tracing Architecture :

Jaeger Agent : Jaeger agent is a network daemon that listens for spans received from the Jaeger client over UDP. It gathers batches of them and then sends them together to
the collector.

Jaeger Collector : The Jaeger collector is responsible for receiving traces from the Jaeger agent, performing validations and transformations, and saving them to the selected storage backends.

Storage Backends : Jaeger supports various storage backends to store the spans.Supported storage backends are In-Memory, Cassandra, Elasticsearch, and Badger (for single-instance collector deployments).

Jaeger Query : This is a service responsible for retrieving traces from the Jaeger storage backend and making them accessible for the Jaeger UI.

Jaeger UI : A React application that lets you visualize the traces and analyze them , that is useful for debugging application issues.

Ingester : The ingester is relevant only if we use Kafka as a buffer between the collector and the storage backend. It is responsible for receiving data from Kafka and ingesting it into the storage backend.

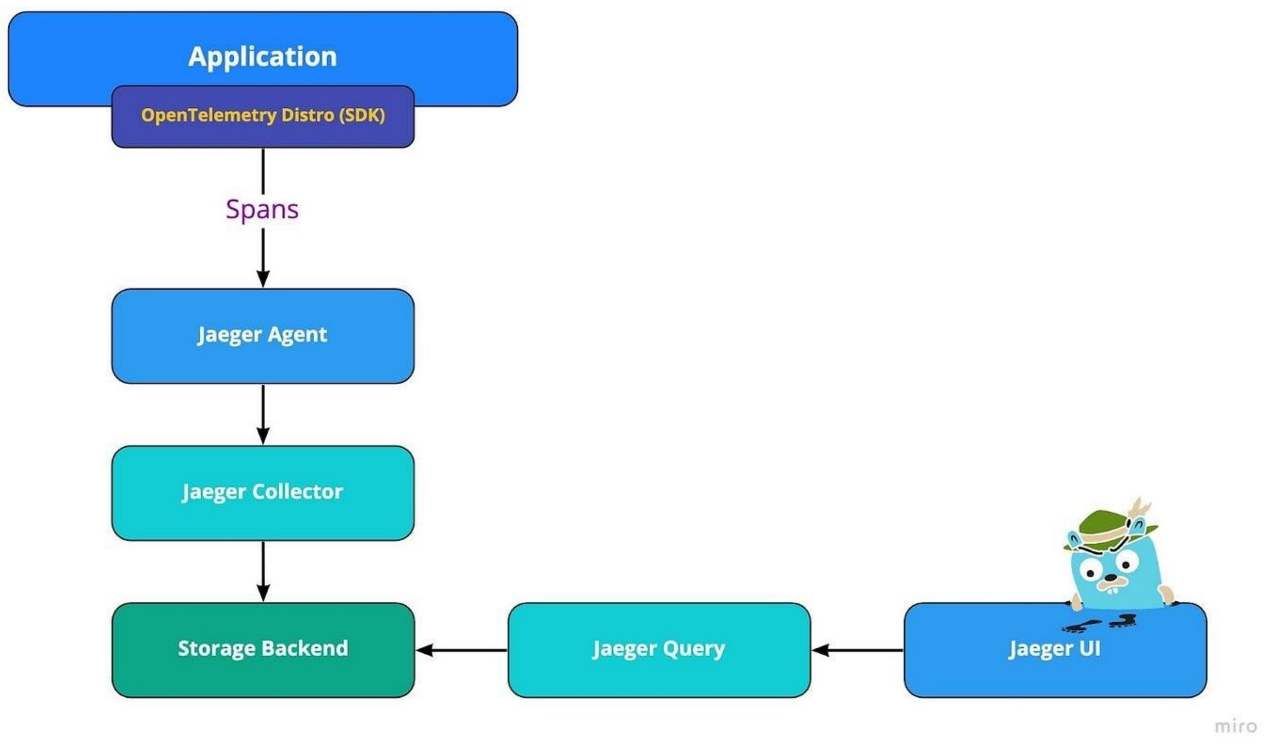


Fig - 5 : Jaeger Architecture

3.4 Running Jaeger locally using Docker :

Jaeger comes with a ready-to-use all-in-one Docker image that contains all the components necessary for Jaeger to run.

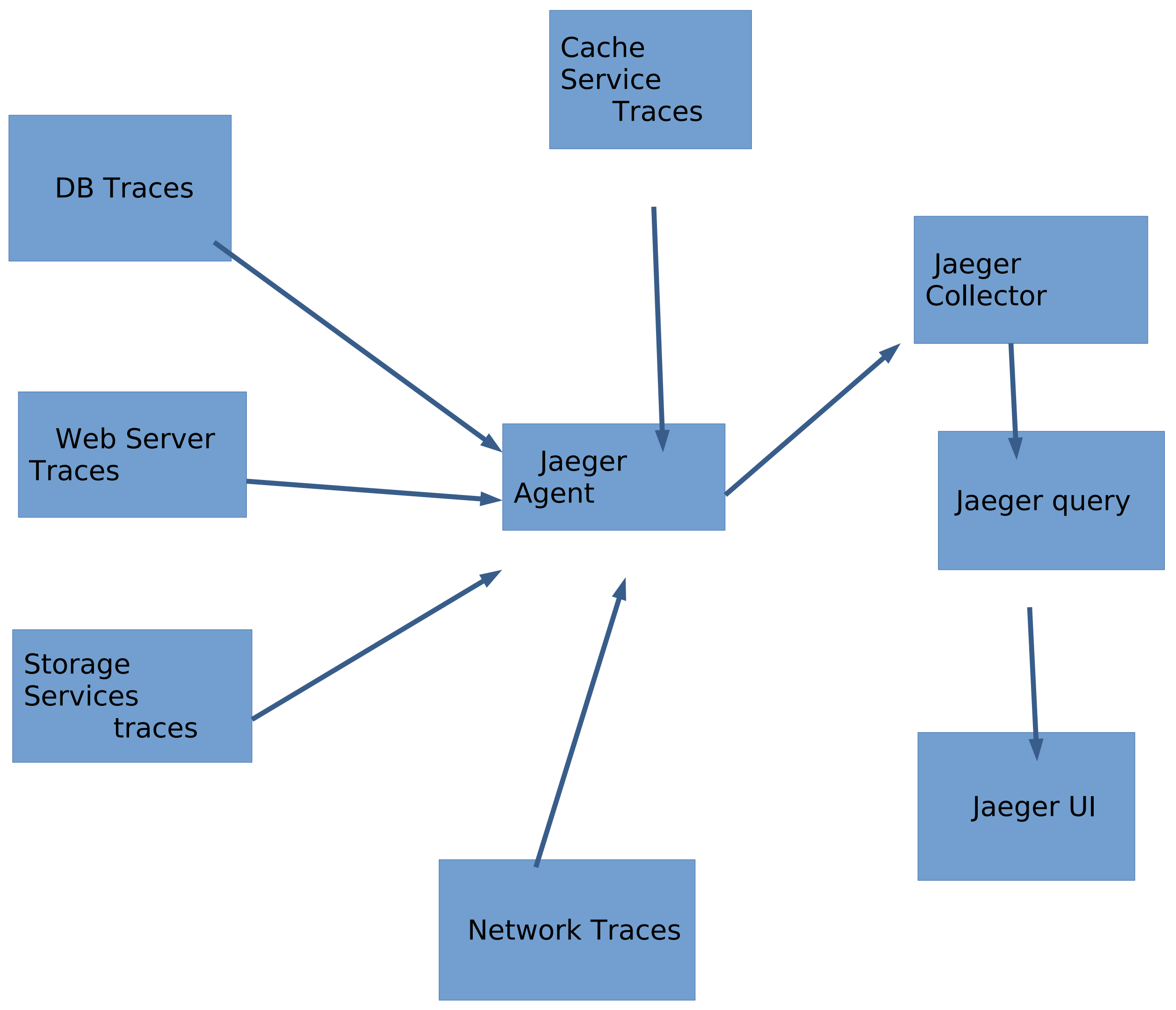
Docker Command :

```
docker run -p 16686:16686 jaegertracing/all-in-one:latest
```

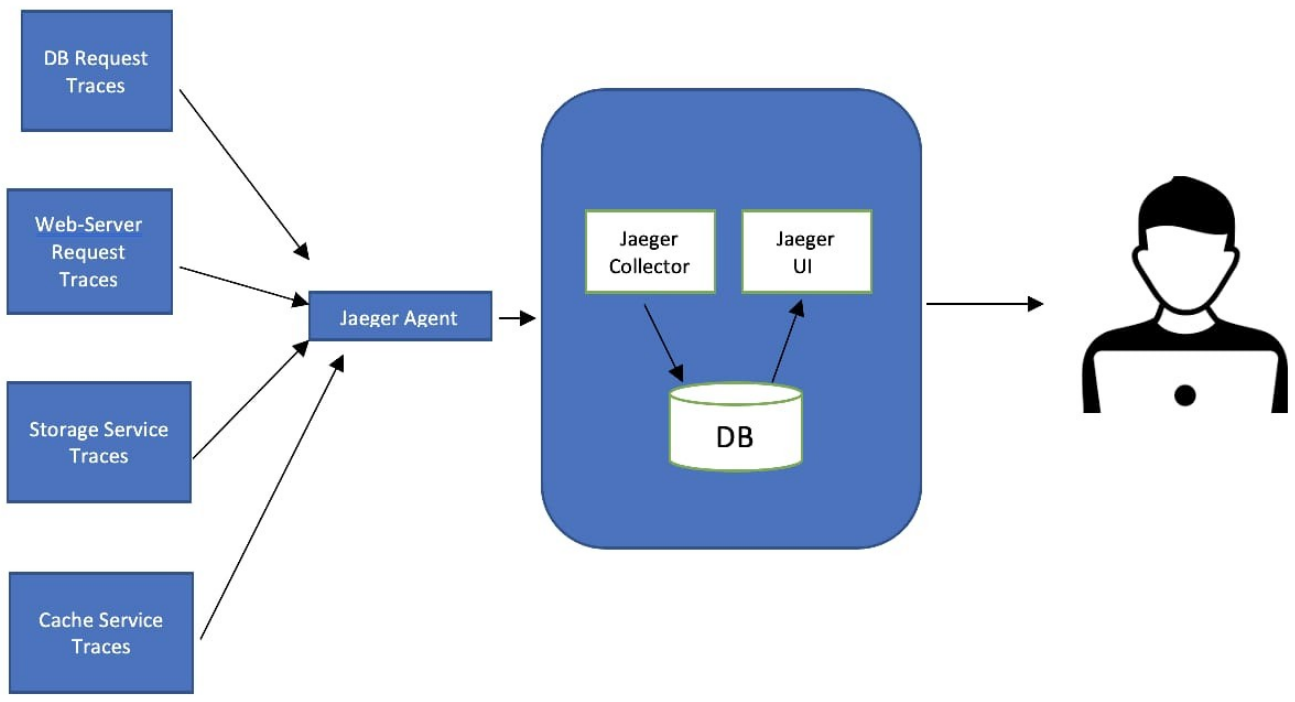
Then you can simply open the jaeger UI on http://localhost:16686/

4. System Design

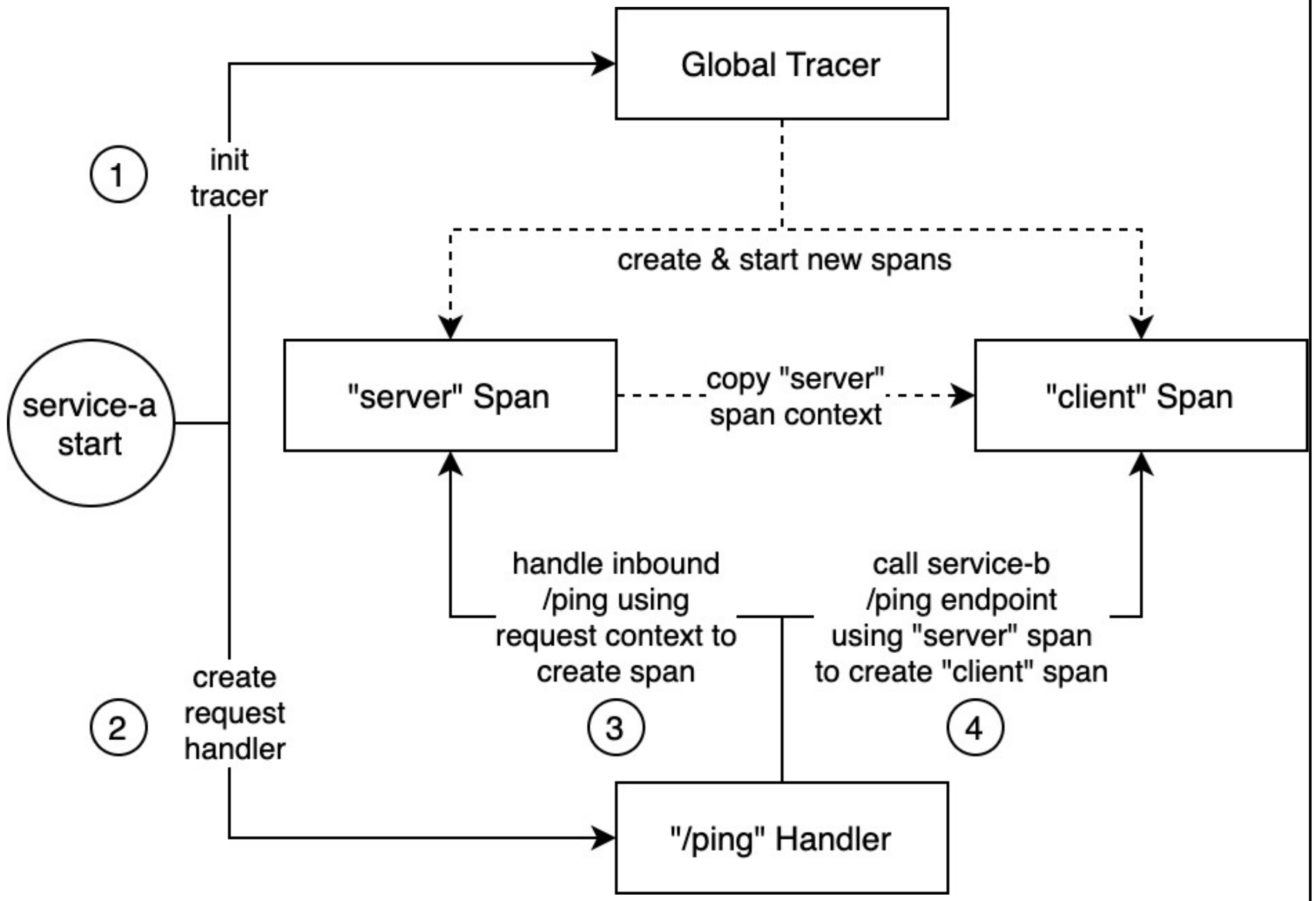
4.1 Data Flow Diagram



4.2 Use case Diagram



4.3 Flowchart


Global Tracer
1
init tracer
create & start new spans
service-a start
"server" Span
copy "server" span context
"client" Span
handle inbound /ping using request context to create span
call service-b /ping endpoint using "server" span to create "client" span
2
create request handler
3
4
"/ping" Handler

5. CODE

```
import sys
import time
import logging
import random
import collections.abc
import collections
collections.MutableMapping = collections.abc.MutableMapping
from jaeger_client import Config
from opentracing_instrumentation.request_context import get_current_span, span_in_context

def init_tracer(service):
    logging.getLogger('').handlers = []
    logging.basicConfig(format='%(message)s', level=logging.DEBUG)
    config = Config(
        config={
            'sampler': {
                'type': 'const',
                'param': 1,
            },
            'logging': True,
        },
        service_name=service,
    )
    return config.initialize_tracer()
def booking_mgr(movie):
    with tracer.start_span('booking') as span:
        span.set_tag('Movie', movie)
        with span_in_context(span):
            cinema_details = check_cinema(movie)
            showtime_details = check_showtime(cinema_details)
            book_show(showtime_details)
```

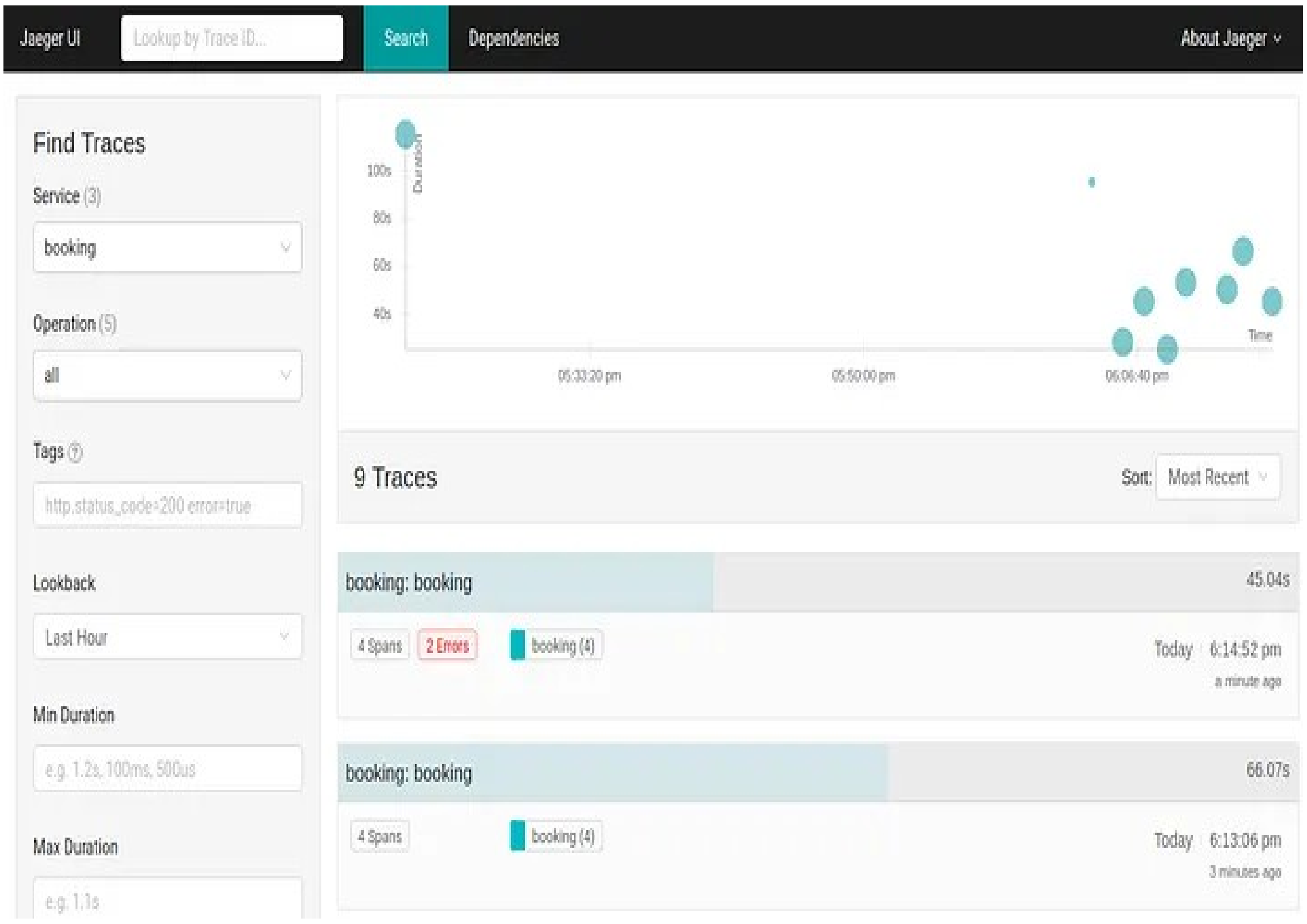
```
def check_cinema(movie):
     # creates a check_cinema span
    with tracer.start_span('CheckCinema', child_of=get_current_span()) as span:
        with span_in_context(span):
            num = random.randint(1,30)
            time.sleep(num)
            cinema_details = "Cinema Details"
            flags = ['false', 'true', 'false']
            random_flag = random.choice(flags)
            span.set_tag('error', random_flag)
            span.log_kv({'event': 'CheckCinema' , 'value': cinema_details })
            return cinema_details

def check_showtime( cinema_details ):
      # creates a check_showtime span
    with tracer.start_span('CheckShowtime', child_of=get_current_span()) as span:
        with span_in_context(span):
            num = random.randint(1,30)
            time.sleep(num)
            showtime_details = "Showtime Details"
            flags = ['false', 'true', 'false']
            random_flag = random.choice(flags)
            span.set_tag('error', random_flag)
            span.log_kv({'event': 'CheckCinema' , 'value': showtime_details })
            return showtime_details
```

```
def book_show(showtime_details):
    # creates a book_show span
    with tracer.start_span('BookShow',  child_of=get_current_span()) as span:
        with span_in_context(span):
            num = random.randint(1,30)
            time.sleep(num)
            Ticket_details = "Ticket Details"
            flags = ['false', 'true', 'false']
            random_flag = random.choice(flags)
             # error tag will be added in traces to filter traces with errors or without errors
            span.set_tag('error', random_flag)
            span.log_kv({'event': 'CheckCinema' , 'value': showtime_details })
            print(Ticket_details)

assert len(sys.argv) == 2
# starts the tracer
tracer = init_tracer('booking')
movie = sys.argv[1]
booking_mgr(movie)
# sleeps to finish the spans
time.sleep(2)
tracer.close()
```

6. Result



Find Traces

Service (3): booking

Operation (5): all

Tags: http.status_code=200 error=true

Lookback: Last Hour

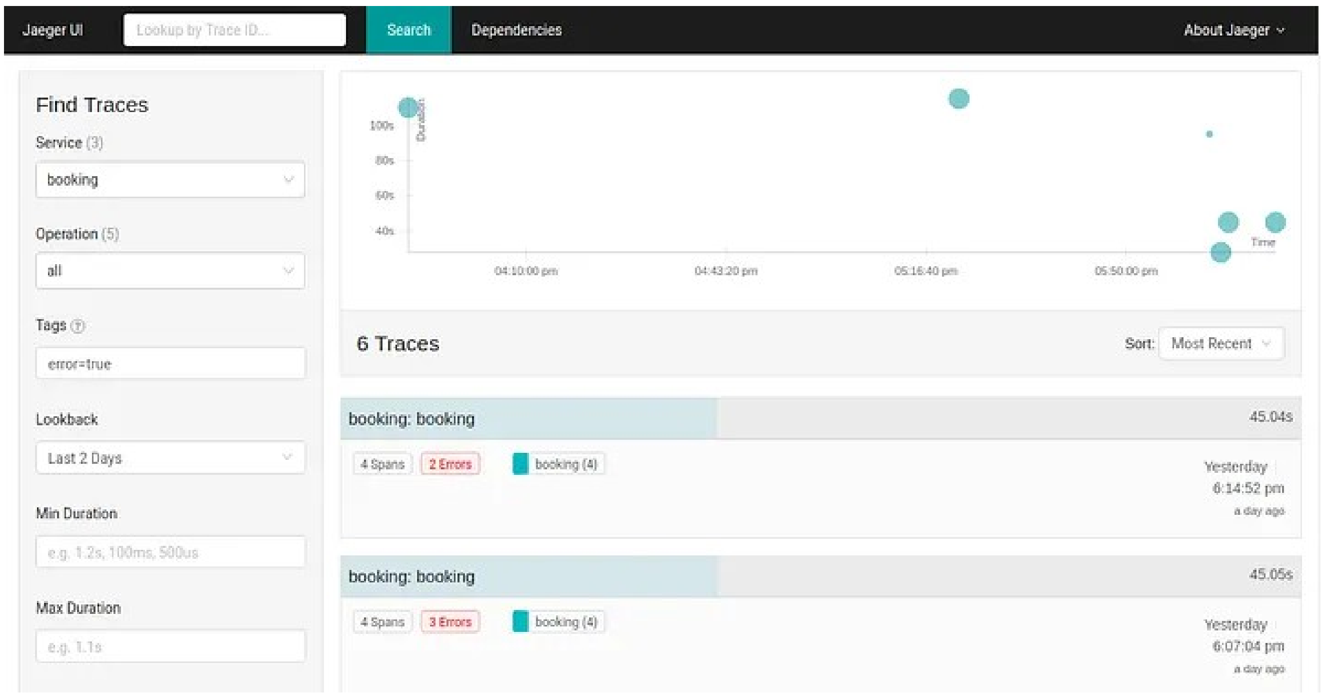
Min Duration: e.g. 1.2s, 100ms, 500us

Max Duration: e.g. 1.1s

9 Traces — Sort: Most Recent

booking: booking — 45.04s
4 Spans | 2 Errors | booking (4) — Today 6:14:52 pm, a minute ago

booking: booking — 66.07s
4 Spans | booking (4) — Today 6:13:06 pm, 3 minutes ago



Find Traces

Service (3): booking

Operation (5): all

Tags: error=true

Lookback: Last 2 Days

Min Duration: e.g. 1.2s, 100ms, 500us

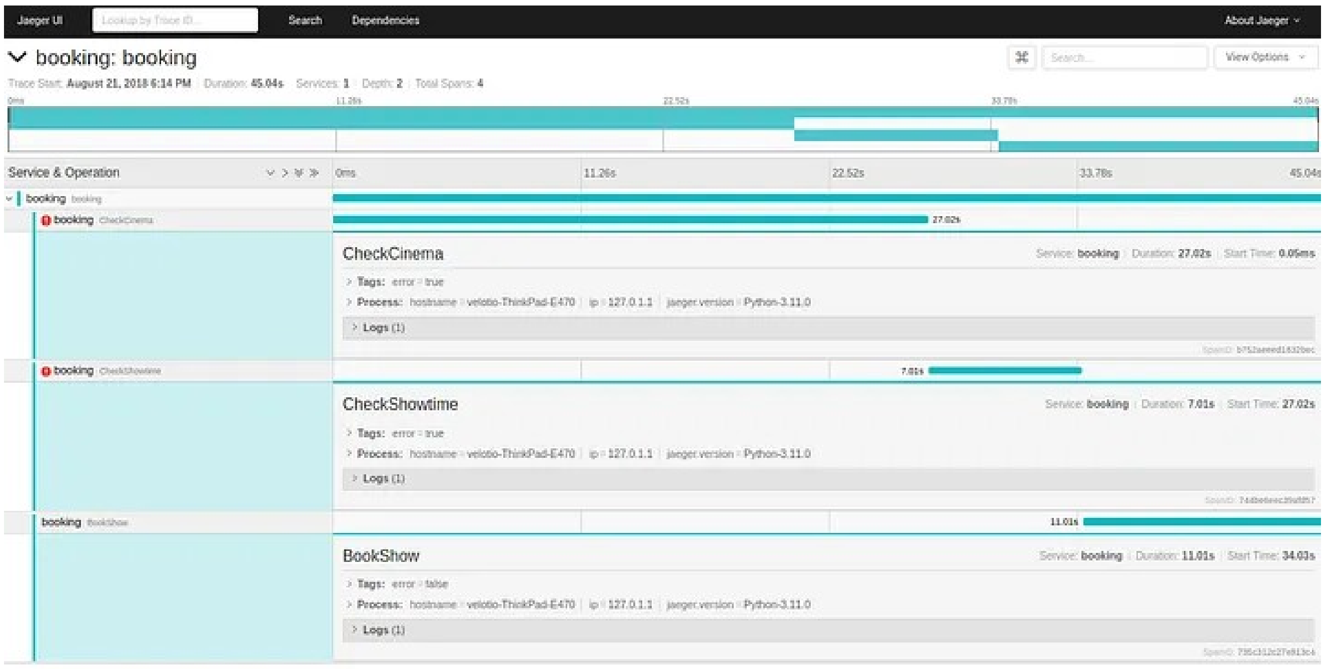
Max Duration: e.g. 1.1s

6 Traces

Sort: Most Recent

booking: booking 45.04s
4 Spans | 2 Errors | booking (4)
Yesterday 6:14:52 pm
a day ago

booking: booking 45.05s
4 Spans | 3 Errors | booking (4)
Yesterday 6:07:04 pm
a day ago



booking: booking

Trace Start: **August 21, 2018 6:14 PM** | Duration: **45.04s** | Services: **1** | Depth: **2** | Total Spans: **4**

booking booking

booking CheckCinema

CheckCinema — Service: **booking** | Duration: **27.02s** | Start Time: **0.05ms**

> **Tags:** error = true
> **Process:** hostname = velotio-ThinkPad-E470 | ip = 127.0.1.1 | jaeger.version = Python-3.11.0
> **Logs** (1)

SpanID: b752aeeed1832bec

booking CheckShowtime

CheckShowtime — Service: **booking** | Duration: **7.01s** | Start Time: **27.02s**

> **Tags:** error = true
> **Process:** hostname = velotio-ThinkPad-E470 | ip = 127.0.1.1 | jaeger.version = Python-3.11.0
> **Logs** (1)

SpanID: 74dbebeec39a8f57

booking BookShow

BookShow — Service: **booking** | Duration: **11.01s** | Start Time: **34.03s**

> **Tags:** error = false
> **Process:** hostname = velotio-ThinkPad-E470 | ip = 127.0.1.1 | jaeger.version = Python-3.11.0
> **Logs** (1)

SpanID: 795c312c27e813c4

7.Conclusion

In real life, applications are even more complex and with the increasing complexity of applications, monitoring the applications has been a hard and tough task. So, With the help of integrating Jaeger Tracing we can get a data that says : Time taken by each service, Latency between the services, Hierarchy of services, Errors or exceptions occured during execution of each service or component to debug and optimise application for developers.

8.**References**

[1] Docker documentation : https://docs.docker.chom/

[2] Jaeger tracing documentation : https://medium.com/velotio-perspectives/a-comprehensive-tutorial-to-implementing-opentracing-with-jaeger-a01752e1a8ce

[3] Kubernetes documentation : https://kubernetes.io/

[4] Video references : https://kodekloud.com/