INTERNSHIP REPORT

Summer internship program at Jio AI-CoE, Hyderabad

Sachin Rungta
Bits Pilani Hyderabad Campus

ACKNOWLEDGEMENT

Reliance Jio. Is considered one of the best and largest Telecom companies in India. The technology used here is considered most advanced. So Internship atJIO AICOE. has been a great experience of my life. I was exposed to new and advanced Techniques of Software Development which changed my view and thinking perspective of software development.
I would like to express my sincere thanks to the JIO Healthcare team and JIO Hr team for carrying out my 42 days Internship at such a prestigious organization.
I would like to pay my gratitude to Dr. Shailesh Kumar (Chief Data Scientist) for his guidance and support. I am extremely grateful and convey my deep sense of gratitude to Mr. Anjul Mishra & Mr. Ved Mulkalwar my project guides for their support and professional assistance and special thanks to Mr. Somesh Granti (HR) because of which I have the opportunity to understand the technical aspects and appreciate the rich professional culture more carefully.

INDEX

Introduction

Reliance Jio Infocomm Limited, Jio, is an Indian telecommunications company and subsidiary of Jio Platforms, headquartered in Mumbai, Maharashtra, India. It operates a national LTE network with coverage across all 22 telecom circles. It does not offer 2G or 3G service, and instead uses only voice over LTE to provide voice service on its 4G network.

Jio soft launched on 27 December 2015 with a beta for partners and employees, and became publicly available on 5 September 2016. As of 31 December 2019, it is the largest mobile network operator in India and the third largest mobile network operator in the world with over 387.5 million subscribers.

The company was registered in Ambawadi, Ahmedabad, Gujarat on 15 February 2007 as Reliance Jio Infocomm Limited. In June 2010, Reliance Industries (RIL) bought a 95% stake in Infotel Broadband Services Limited (IBSL) for ₹4,800 crore (US$670 million). Although unlisted, IBSL was the only company that won broadband spectrum in all 22 circles in India in the 4G auction that took place earlier that year. Later continuing as RIL's telecom subsidiary, Infotel Broadband Services Limited was renamed as Reliance Jio Infocomm Limited (RJIL) in January 2013.

In June 2015, Jio announced that it would start its operations throughout the country by the end of 2015. However, four months later in October, the company postponed the launch to the first quarter of the financial year 2016–2017.

Later, in July 2015, a PIL filed in the Supreme Court by an NGO called the Centre for Public Interest Litigation, through Prashant Bhushan, challenged the grant of a pan-India licence to Jio by the Government of India. The PIL also alleged that the firm was being allowed to provide voice telephony along with its 4G data service, by paying an additional fee of just ₹165.8 crore (US$23 million) which was arbitrary and unreasonable, and contributed to a loss of ₹2,284.2 crore (US$320 million) to the exchequer. The Indian Department of Telecommunications (DoT), however, explained that the rules for 3G and BWA spectrum didn't restrict BWA winners from providing voice telephony. As a result, the PIL was revoked, and the accusations were dismissed.

The 4G services were launched internally on 27 December 2015. The company commercially launched its 4G services on 5 September 2016. Within the first month, Jio announced that it had acquired 16 million subscribers. Jio crossed 50 million subscriber mark in 83 days since its launch, subsequently crossing 100 million subscribers on 22 February 2017. By October 2017 it had about 130 million subscribers.

Project-1 Introduction

Problem Statement:-

Setup a complete monitoring dashboard using grafana and prometheus to monitor the performance of Apache Kafka(messaging queue) with features of alerting when the system faces some problem.

What is Kafka ?

Apache Kafka® is *a distributed streaming platform*. What exactly does that mean?

A streaming platform has three key capabilities:

- Publish and subscribe to streams of records, similar to a message queue or enterprise messaging system.
- Store streams of records in a fault-tolerant durable way.
- Process streams of records as they occur.

Kafka is generally used for two broad classes of applications:

- Building real-time streaming data pipelines that reliably get data between systems or applications
- Building real-time streaming applications that transform or react to the streams of data

To understand how Kafka does these things, let's dive in and explore Kafka's capabilities from the bottom up.

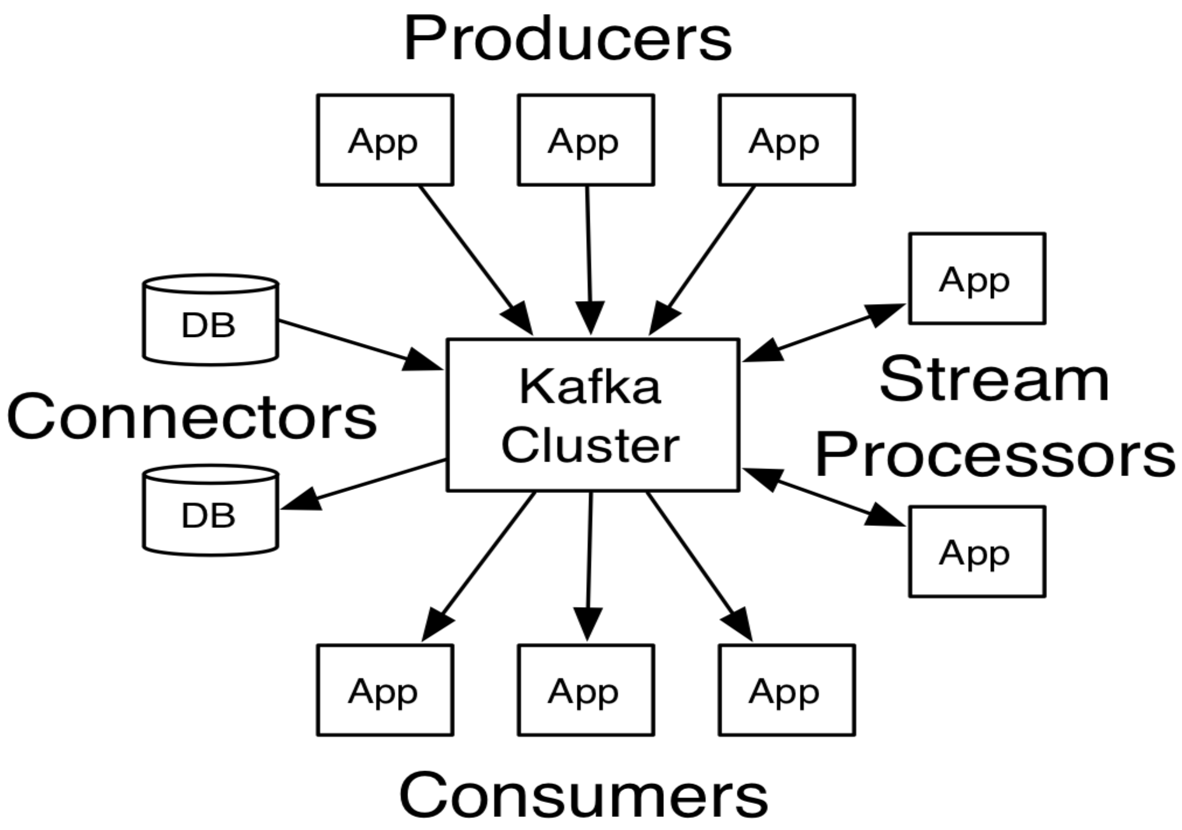
First a few concepts:

- Kafka is run as a cluster on one or more servers that can span multiple datacenters.

- The Kafka cluster stores streams of *records* in categories called *topics*.
- Each record consists of a key, a value, and a timestamp.

Kafka has five core APIs:

- The Producer API allows an application to publish a stream of records to one or more Kafka topics.
- The Consumer API allows an application to subscribe to one or more topics and process the stream of records produced to them.
- The Streams API allows an application to act as a *stream processor*, consuming an input stream from one or more topics and producing an output stream to one or more output topics, effectively transforming the input streams to output streams.
- The Connector API allows building and running reusable producers or consumers that connect Kafka topics to existing applications or data systems. For example, a connector to a relational database might capture every change to a table.
- The Admin API allows managing and inspecting topics, brokers and other Kafka objects.



In Kafka the communication between the clients and the servers is done with a simple, high-performance, language agnostic TCP protocol. This protocol is versioned and maintains backwards compatibility with older versions. We provide a Java client for Kafka, but clients are available in many languages.

What is Prometheus?

Prometheus is an open-source systems monitoring and alerting toolkit originally built at SoundCloud. Since its inception in 2012, many companies and organizations have adopted Prometheus, and the project has a very active developer and user community. It is now a standalone open source project and maintained independently of any company. To emphasize this, and to clarify the project's governance structure, Prometheus joined the Cloud Native Computing Foundation in 2016 as the second hosted project, after Kubernetes.

Features

Prometheus's main features are:

- a multi-dimensional data model with time series data identified by metric name and key/value pairs
- PromQL, a flexible query language to leverage this dimensionality
- no reliance on distributed storage; single server nodes are autonomous
- time series collection happens via a pull model over HTTP
- pushing time series is supported via an intermediary gateway
- targets are discovered via service discovery or static configuration
- multiple modes of graphing and dashboarding support

Components

The Prometheus ecosystem consists of multiple components, many of which are optional:

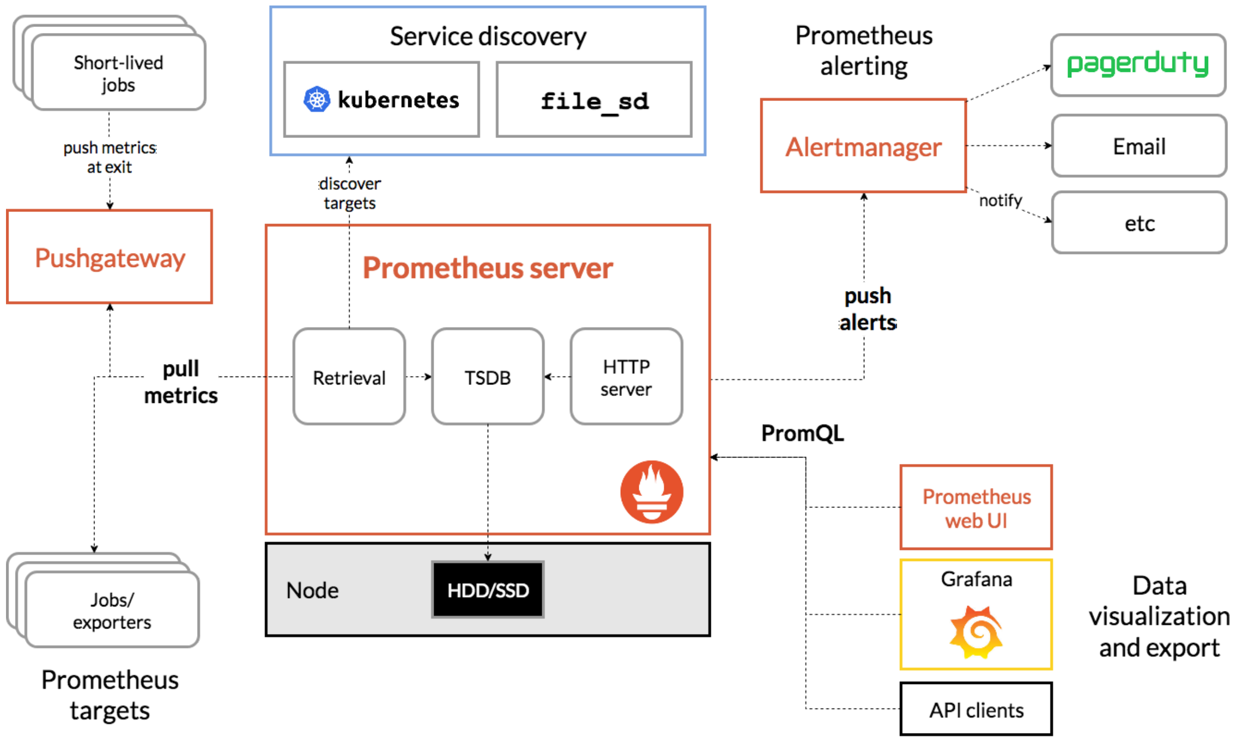
- the main Prometheus server which scrapes and stores time series data
- client libraries for instrumenting application code

- a push gateway for supporting short-lived jobs
- special-purpose exporters for services like HAProxy, StatsD, Graphite, etc.
- an alertmanager to handle alerts
- various support tools

Most Prometheus components are written in Go, making them easy to build and deploy as static binaries.

Architecture

This diagram illustrates the architecture of Prometheus and some of its ecosystem components:



Prometheus scrapes metrics from instrumented jobs, either directly or via an intermediary push gateway for short-lived jobs. It stores all scraped samples locally and runs rules over this data to either aggregate and record new time series from existing data or generate alerts. Grafana or other API consumers can be used to visualize the collected data.

What is Grafana?

Grafana is open source visualization and analytics software. It allows you to query, visualize, alert on, and explore your metrics no matter where they are stored. In plain English, it provides you with tools to turn your time-series database (TSDB) data into beautiful graphs and visualizations.

After creating a dashboard, there are many possible things you might do next. It all depends on your needs and your use case.

For example, if you want to view weather data and statistics about your smart home, then you might create a playlist. If you are the administrator for a corporation and are managing Grafana for multiple teams, then you might need to set up provisioning and authentication.

Project Implementation:

Github link for Complete Project setup is

https://github.com/sac6120/Docker-kafka

Setting up Kafka monitoring usingPrometheus and Grafana on Docker

Kafka is a distributed streaming platform that is used to publish and subscribe to streams of records. Kafka is used for fault tolerant storage. It replicates topic log partitions to multiple servers. Kafka is designed to allow your apps to process records as they occur.

Prerequisites

This tutorial assumes that you have a stable build of docker and docker-compose pre-installed. Installation guidelines can be found here.

Initializing your containers

In a suitable location, clone the set up repo. Switch to this directory and call docker-compose to start your containers

```
$ git clone https://github.com/sac6120/Docker-kafka.git
$ cd Docker-Kafka
$ docker-compose up -d
```

The -d flag starts the containers in background mode. At the point, you may call `docker-compose ps` to confirm that your 4 containers are up. Additionally, you may shut down your containers using `docker-compose stop CONTAINER_NAME`

Kafka set up

To similiate a kafka set up, we will set up a sample Producer and Consumer under a sample topic. Start by entering the kafka container using the docker command and creating a topic

```
$ sudo docker exec -it kafka /bin/sh
$ cd /opt/kafka
$ ./bin/kafka-topics.sh --create --zookeeper zookeeper:2181 --replication-factor 1 --partitions 1
--topic helloworld
```

You can see the list of all topics created using

```
$ bin/kafka-topics.sh --list --zookeeper zookeeper:2181
```

Now that we have a sample topic called helloworld, we will link a producer to this topic. Also, we'll add some data:

```
$ bin/kafka-console-producer.sh --broker-list localhost:9092 --topic helloworld
>this is the FIRST message
>this is the SECOND message
>this is the THIRD message
>this is the FOURTH message
>this is the FIFTH message
```

In a second terminal opened at the same location, we'll create a consumer on this topic.

```
$ sudo docker exec -it kafka /bin/sh
$ cd opt/kafka
$ bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic helloworld
--from-beginning
```

Creating a script for the producer that simulates a constant producer is left to the user as per need.

Prometheus set up

Opening Prometheus Dashboard shows a rudimentary view of all Kafka metrics. Confirm that Kafka data is being scraped by visiting targets, where the endpoint kafka should be up.

Adding Up metrics to Prometheus Dashboard

Metrics can be observed right from the Prometheus dashboard. Click on the graph option and enter the prometheus query desired.

Grafana set up

Log in

Login into Grafana using credentials 'admin' in both user and password fields. You may or may not choose to skip changing the password.

Add Data Source

Before setting up a dashboard, Prometheus must be added as a data source. Do so by clicking on the Add you first data source card and choosing Prometheus. Assign a suitable name for this and provide the source URL as `http://localhost:9090` and Access type of Browser (not Server). Confirm that everything is set upright with the Save & Test button and return to the Grafana dashboard.

Add basic dashboard

On the left side of the dashboard, hover over the Create tab and choose Import. Enter `721` under Import via Grafana and Load. Once again, assign a suitable name and choose source as the same Prometheus data source we set up above and import.

The scope of the graphs and refresh rate can be adjusted on the top right of the dashboard.

Adding metrics to Grafana Panel

Additional panels can be added by clicking the add panel option and entering the desired query. Grafana suggests possible queries based on the input provided and complete action with `shift + enter`.

Setting up email alerts

Setting up SMTP on your account to receive alert notifications

We will set up an SMTP server to send alerts to your gmail account from Grafana's dashboard. However, as a security feature, Google requires this to be activated and simply blocks it for accounts with two factor authentication.

If this meets your needs, enable SMTP access to your account and allow the use of less secure apps. Following this, install SMTP with:

```
$ sudo apt-get update
$ sudo apt-get install ssmtp
```

Edit the ssmtp.conf file to connect to gmail account:

```
$ vi /etc/ssmtp/ssmtp.conf
root=(emailID)
mailhub=smtp.gmail.com.587
FromLineOverride=YES
AuthUser=(emailID)
AuthPass=(passwd)
UseTLS=YES
UseSTARTTLS=YES
```

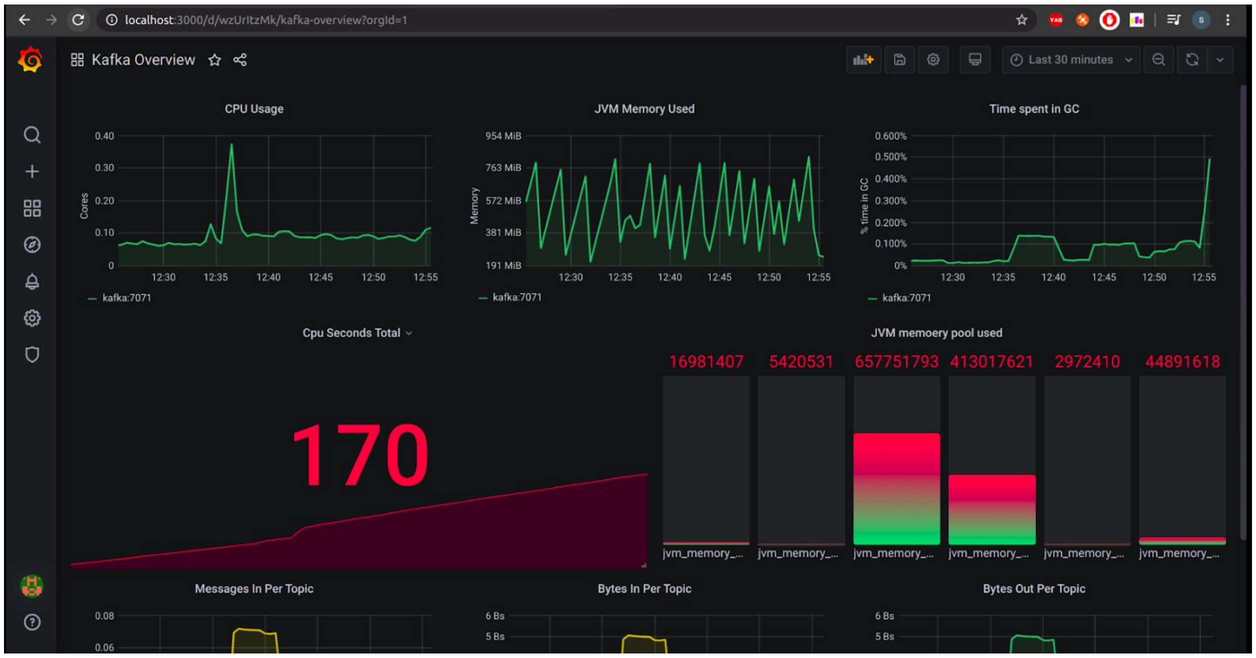
Run the command `$ echo “Email using CLI” | ssmtp (emailID).`

Adding up Alerts Rule to Grafana Panel

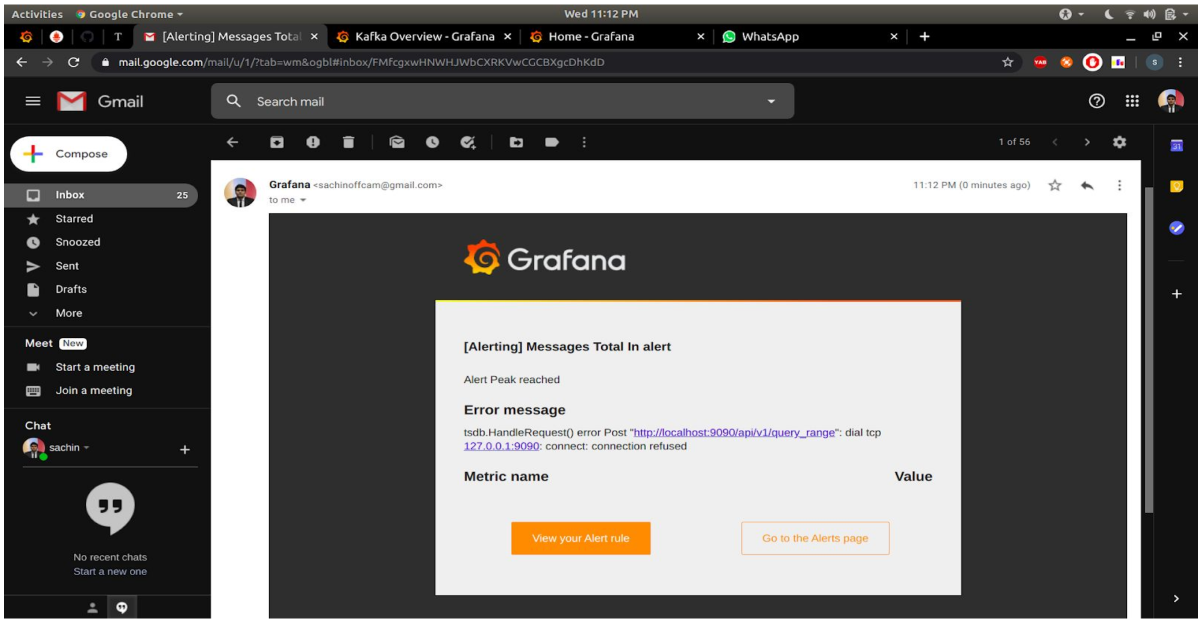
Locate and edit grafana.ini to enable sending emails using SMTP:

```
$ vi /etc/grafana/grafana.ini
Change SMTP section as shown below:
[smtp]
enabled=true
host=smtp.gmail.com:587
user=(emailID)
```

password=(passwd)
;cert_file =
;key_file =
skip_verify = true
from_address = (emailID)
#Dashboard Picture



#Alert Notifications Picture



Grafana <sachinoffcam@gmail.com>
to me
11:12 PM (0 minutes ago)

Grafana

[Alerting] Messages Total In alert

Alert Peak reached

Error message

tsdb.HandleRequest() error Post "http://localhost:9090/api/v1/query_range": dial tcp 127.0.0.1:9090: connect: connection refused

Metric name	Value

View your Alert rule

Go to the Alerts page

Project-2 Introduction

Problem Statement:-

Develop a Contact Tracing which is based on Dp3T protocol and has basic functionalities of Report Covid Positive User and Track Covid Status of a User without taking user data on server.

DP3T:

Decentralized Privacy-Preserving Proximity Tracing (DP-3T, stylized as dp^{3}t) is an open protocol developed in response to the COVID-19 pandemic to facilitate digital contact tracing of infected participants. The protocol, like competing protocol Pan-European Privacy-Preserving Proximity Tracing (PEPP-PT), uses Bluetooth Low Energy to track and log encounters with other users. The protocols differ in their reporting mechanism, with PEPP-PT requiring clients to upload contact logs to a central reporting server, whereas with DP-3T, the central reporting server never has access to contact logs nor is it responsible for processing and informing clients of contact. Because contact logs are never transmitted to third parties, it has major privacy benefits over the PEPP-PT approach, however this comes at the cost of requiring more computing power on the client side to process infection reports.

FastAPI:

FastAPI is a modern, fast (high-performance), web framework for building APIs with Python 3.6+ based on standard Python type hints.

The key features are:
Fast: Very high performance, on par with NodeJS and Go (thanks to Starlette and Pydantic). One of the fastest Python frameworks available.

- Fast to code: Increase the speed to develop features by about 200% to 300%.
- Fewer bugs: Reduce about 40% of human (developer) induced errors. *
- Intuitive: Great editor support. Completion everywhere. Less time debugging.
- Easy: Designed to be easy to use and learn. Less time reading docs.
- Short: Minimize code duplication. Multiple features from each parameter declaration. Fewer bugs.

- Robust: Get production-ready code. With automatic interactive documentation.
- Standards-based: Based on (and fully compatible with) the open standards for APIs: OpenAPI (previously known as Swagger) and JSON Schema.

Android Rooms:

Room is a persistence library, part of the Android Architecture Components. It makes it easier to work with SQLiteDatabase objects in your app, decreasing the amount of boilerplate code and verifying SQL queries at compile time.

The Room persistence library provides an abstraction layer over SQLite to allow for more robust database access while harnessing the full power of SQLite.

The library helps you create a cache of your app's data on a device that's running your app. This cache, which serves as your app's single source of truth, allows users to view a consistent copy of key information within your app, regardless of whether users have an internet connection.

Google Nearby messages API:

The Nearby Messages API is a publish-subscribe API that lets you pass small binary payloads between internet-connected Android and iOS devices. The devices don't have to be on the same network, but they do have to be connected to the Internet.

Nearby uses a combination of Bluetooth, Bluetooth Low Energy, Wi-Fi and near-ultrasonic audio to communicate a unique-in-time pairing code between devices. The server facilitates message exchange between devices that detect the same pairing code. When a device detects a pairing code from a nearby device, it sends the pairing code to the Nearby Messages server for validation, and to check whether there are any messages to deliver for the application's current set of subscriptions.

Nearby Messages is unauthenticated and does not require a Google Account.

The exact mechanism for data exchange may vary from release to release. The following sequence shows the events leading to message exchange:

1. A publishing app makes a request to associate a binary payload (the message) with a unique-in-time pairing code (token). The server makes a temporary association between the message payload and the token.
2. The publishing device uses a combination of Bluetooth, Bluetooth Low Energy,

Wi-Fi and an ultrasonic modem to make the token detectable by nearby devices. The publishing device also uses these technologies to scan for tokens from other devices.

3. A subscribing app associates its subscription with a token and uses a mix of the above technologies to send its token to the publisher, and to detect the publisher's token.
4. When either side detects the other's token, it reports it to the server.
5. The server facilitates message exchange between two devices when both are associated with a common token, and the API keys used by the calling apps are associated with the same project in the Google Developers Console.

Project Implementation:

Github Link:-

https://github.com/NealMenon/Contact-Tracer-App

Clone the Repository and Paste your Google Api in manifest file and install the App.

To generate Api Key follow Google Api Console steps.

****

End

****