MediTrack

KUBERNETES CLUSTER

CONTENTS

Table of Figures 2
1 Solution 3
1.1 Introduction 3
1.2 Solution architecture - Minikube 3
1.3 Solution architecture – EKS 4
2 Microservices 5
2.1 Appointment Scheduling Service 5
2.2 Notification Service 6
2.3 Patient Records Service 6
2.4 Analytics and Aggregation Service 7
3 Deployment 9
3.1 Deployment method 9
4 Minikube cluster 10
5 Mongodb 13
6 Redshift 14
7 Quicksight Dashboard 15
8 CI/CD 16
9 Security and ethics 18
9.1.1 Security Challenges 18
9.1.2 Ethics Challenges 18
10 Runbook 20
10.1 Installing Minikube 20
10.2 Starting Minikube 20
10.3 Enabling Minikube Add-ons 21
10.4 Setting Up Microservices 21
10.5 Connecting MongoDB 22
10.6 Connecting to Amazon Redshift 22
10.7 Setting Up QuickSight 22
10.8 Configuring GitHub Actions 23
10.9 Testing the Workflow 23
Reference 24

TABLE OF FIGURES

Figure 1: Minikube Solution Architecture Diagram 3
Figure 2: AWS EKS Solution Architecture Diagram 4
Figure 3: Microservices Architecture Diagram 5
Figure 4: Appointments Scheduling Microservice 6
Figure 5: Notification Microservice.................... 6
Figure 6: Patient Record Microservice 7
Figure 7: Analytics Microservice.................... 7
Figure 8: Redshift Analytics Microservice 8
Figure 9: Deployment Diagram 9
Figure 10: Ingress Configuration and Folder Structure 10
Figure 11: Minikube Pods and Services 11
Figure 12: Minikube Pod Health Dashboard.................... 11
Figure 13: Minikube Dashboard - Cron Jobs and Deployments 11
Figure 14: Minikube Dashboard Ingress.................... 12
Figure 15: MongoDB 13
Figure 16: AWS Redshift Queries.................... 14
Figure 17: AWS Redshift Query Results 14
Figure 18: AWS QuickSight Dashboard.................... 15
Figure 19: Data Sources for Quicksight 15
Figure 20: Github Actions Red-Green Workflow.................... 16
Figure 21: Github Action CI/CD.................... 16
Figure 22: Github Folder Structure 17
Figure 23: Minikube Version Check 20
Figure 24: Minikube Status Check.................... 20
Figure 25: Minikube Addon list.................... 21
Figure 26: MongoDB URL.................... 22
Figure 27: Redshift Python Driver in Requirements.txt.................... 22

1 SOLUTION

1.1 INTRODUCTION

HealthSync, developed for MediTrack, streamlines patient data management and appointment tracking, enabling healthcare providers to efficiently access and update health records.

1.2 SOLUTION ARCHITECTURE - MINIKUBE

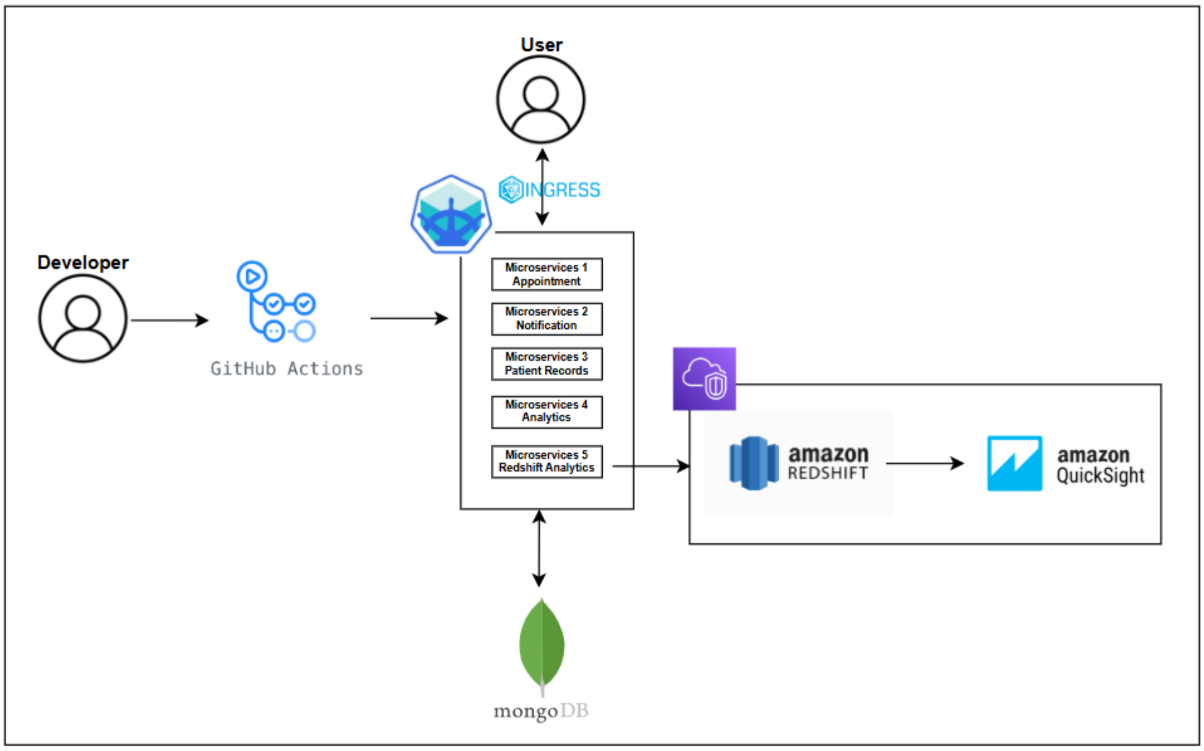

User
INGRESS
Developer
GitHub Actions
Microservices 1 Appointment
Microservices 2 Notification
Microservices 3 Patient Records
Microservices 4 Analytics
Microservices 5 Redshift Analytics
amazon REDSHIFT
amazon QuickSight
mongoDB

Figure 1: Minikube Solution Architecture Diagram

The solution is deployed on a local Kubernetes environment using Minikube. Minikube is configured to run on the local machine with Docker as the container runtime. Docker images are built directly on the Minikube virtual machine and deployed as Kubernetes pods. A Python Flask microservice handles incoming user requests and stores data in a locally hosted MongoDB instance. Another microservice consumes data from MongoDB, performs analytical processing, and sends the results to Amazon Redshift, which is hosted in an AWS VPC with appropriate security configurations. Visualizations of the analytical data are generated using Amazon QuickSight, which directly connects to the Redshift cluster. Continuous integration and deployment (CI/CD) are automated using GitHub Actions, which builds the Docker images, runs tests, and deploys updated services to the Minikube cluster.

1.3 SOLUTION ARCHITECTURE – EKS

The proposed solution migrates the entire infrastructure to AWS for improved scalability, security, and performance. The application is deployed in an Amazon VPC with both public and private subnets. The Kubernetes cluster is hosted on Amazon Elastic Kubernetes Service (EKS), which manages the orchestration of containerized microservices. Docker images are stored in Amazon Elastic Container Registry (ECR) and pulled by EKS during deployments. User requests are routed through an Internet Gateway to an Application Load Balancer (ALB), which directs traffic to the Flask microservice running within the EKS cluster. The Flask microservice interacts with MongoDB, which is either hosted on MongoDB Atlas or self-managed in a private subnet within the VPC. The analytics microservice processes data from MongoDB and pushes the results to Amazon Redshift, also hosted within the VPC. Amazon QuickSight connects to Redshift to generate visual dashboards and reports. CI/CD pipelines are implemented using GitHub Actions, automating the process of building Docker images, pushing them to ECR, and updating deployments in the EKS cluster.

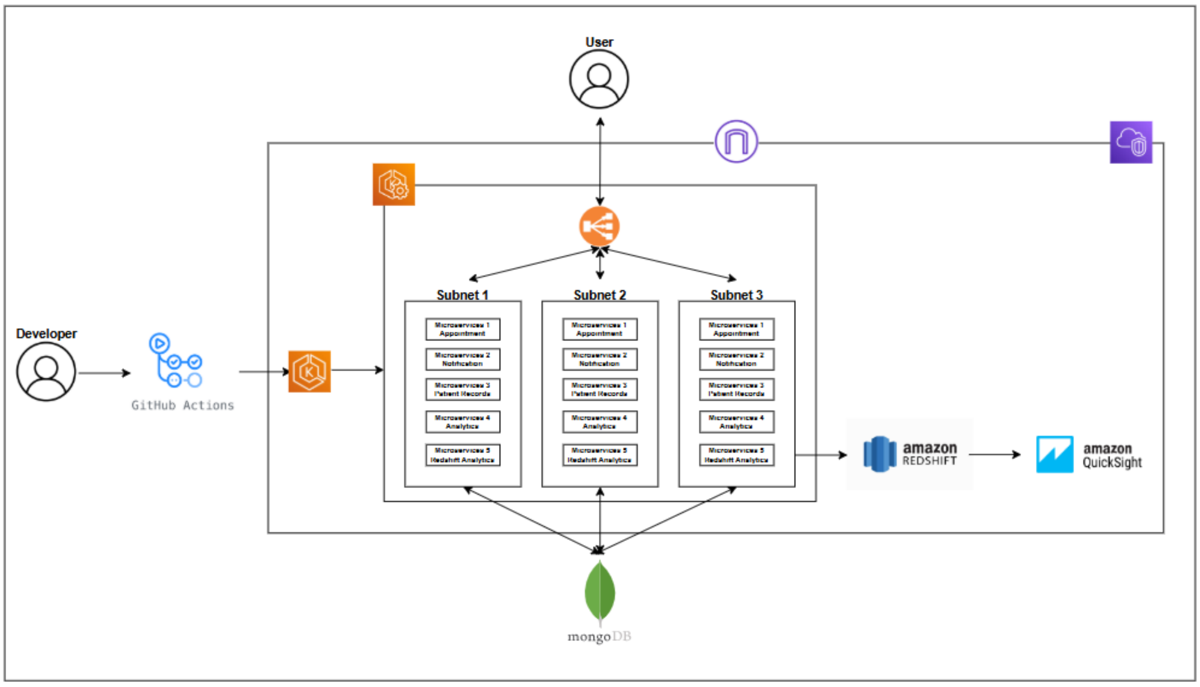

User
Developer
GitHub Actions
Subnet 1
Subnet 2
Subnet 3
Microservices 1 Appointment
Microservices 2 Notification
Microservices 3 Patient Records
Microservices 4 Analytics
Microservices 5 Redshift Analytics
amazon REDSHIFT
amazon QuickSight
mongoDB

Figure 2: AWS EKS Solution Architecture Diagram

2 MICROSERVICES

I have implemented 4 microservices to efficiently handle data in this solution as,

- Appointment Scheduling Service
- Notification Service
- Patient Records Service
- Analytics and Aggregation Service and Redshift Analytics Service

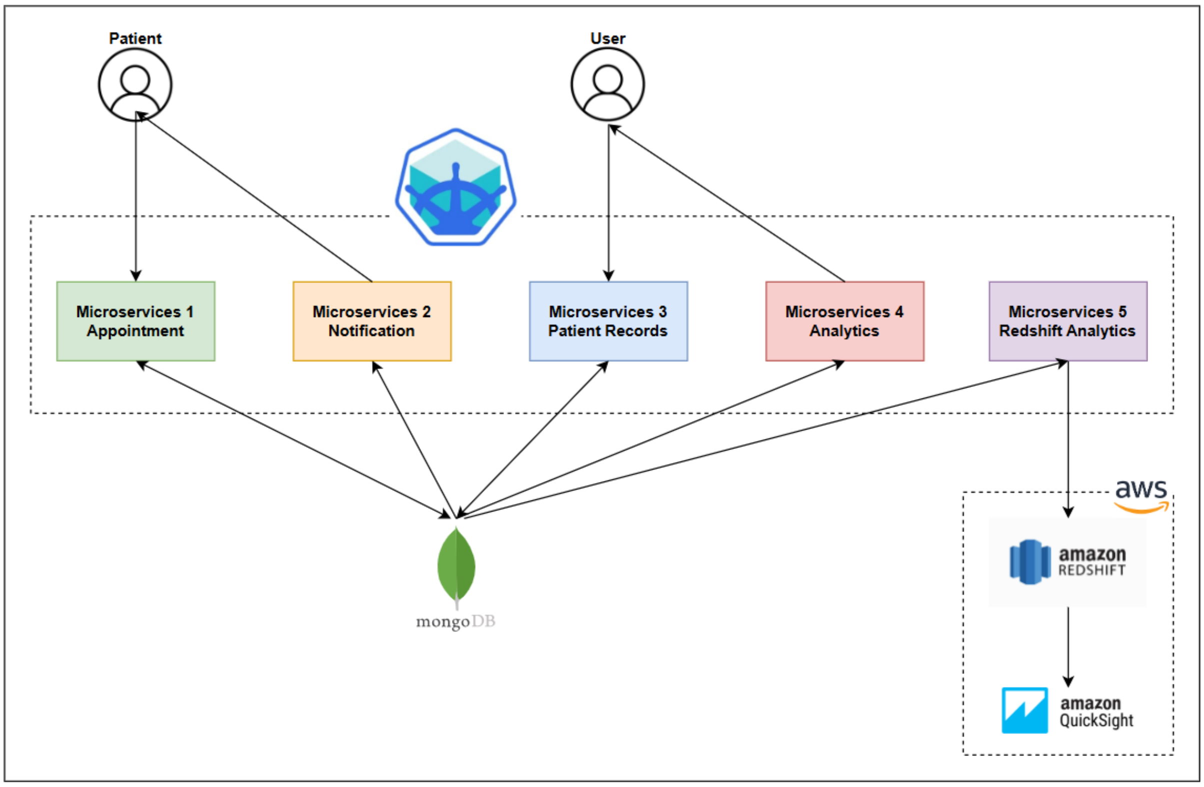


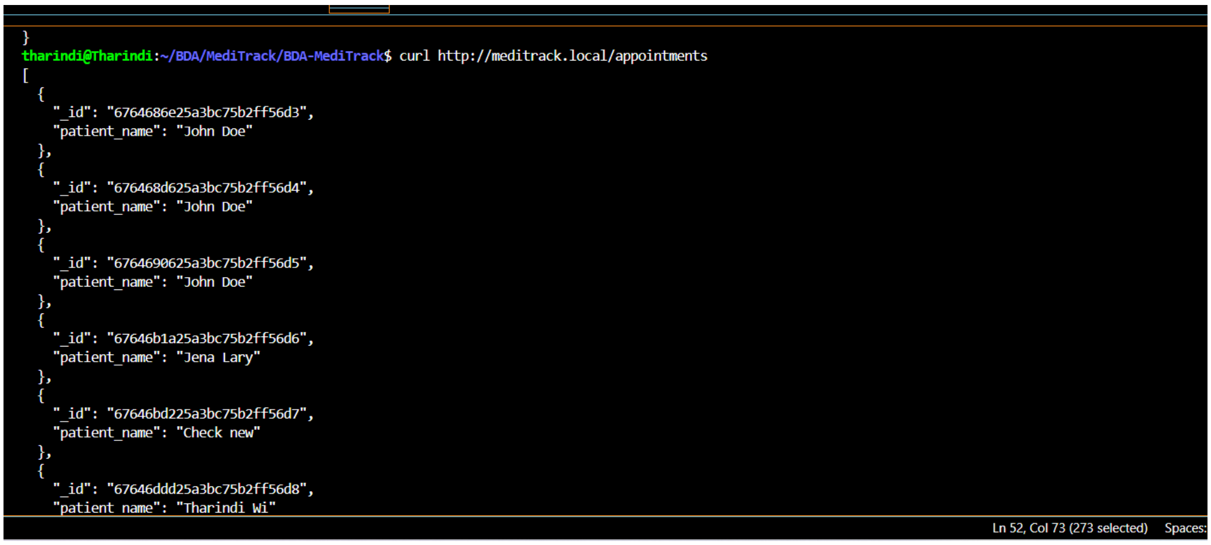
Figure 3: Microservices Architecture Diagram

2.1 APPOINTMENT SCHEDULING SERVICE

The Appointment Scheduling Service manages the core functionality of scheduling and maintaining appointments between patients and doctors. When a new appointment is scheduled, the service validates the incoming data for completeness and then saves it into the database. Upon successful insertion, it communicates with the Notification Service to send confirmation messages to patients, ensuring a seamless and connected experience. Key functionalities include,

- Adding Appointments
- Updating and Deleting Appointments
- Fetching Appointments

The service is designed to be robust and scalable, ensuring it meets the high demand of managing large volumes of appointment data efficiently.



```
}
tharindi@Tharindi:~/BDA/MediTrack/BDA-MediTrack$ curl http://meditrack.local/appointments
[
  {
    "_id": "6764686e25a3bc75b2ff56d3",
    "patient_name": "John Doe"
  },
  {
    "_id": "676468d625a3bc75b2ff56d4",
    "patient_name": "John Doe"
  },
  {
    "_id": "6764690625a3bc75b2ff56d5",
    "patient_name": "John Doe"
  },
  {
    "_id": "67646b1a25a3bc75b2ff56d6",
    "patient_name": "Jena Lary"
  },
  {
    "_id": "67646bd225a3bc75b2ff56d7",
    "patient_name": "Check new"
  },
  {
    "_id": "67646ddd25a3bc75b2ff56d8",
    "patient_name": "Tharindi Wi"
                                                                Ln 52, Col 73 (273 selected)    Spaces:
```

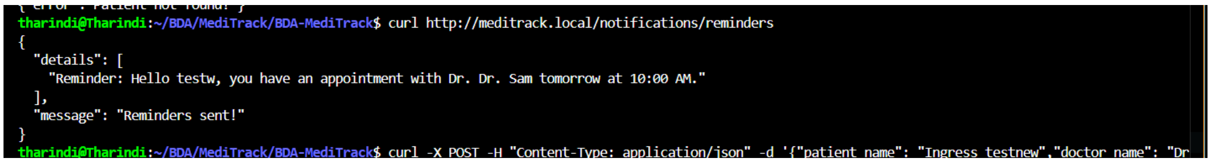
Figure 4: Appointments Scheduling Microservice

2.2 NOTIFICATION SERVICE

The Notification Service complements the Appointment Scheduling Service by handling all patient communication related to appointments. This includes sending notifications about newly scheduled appointments and reminders for upcoming ones. It also queries the Appointments collection directly to identify upcoming appointments and send timely reminders. Key functionalities include,

- Scheduled Notifications
- Reminders sending
- Error Handling

The Notification Service adds value by ensuring patients are always informed about their healthcare schedules, improving reliability and trust in the system.



```
{"error": "Patient not found!"}
tharindi@Tharindi:~/BDA/MediTrack/BDA-MediTrack$ curl http://meditrack.local/notifications/reminders
{
  "details": [
    "Reminder: Hello testw, you have an appointment with Dr. Dr. Sam tomorrow at 10:00 AM."
  ],
  "message": "Reminders sent!"
}
tharindi@Tharindi:~/BDA/MediTrack/BDA-MediTrack$ curl -X POST -H "Content-Type: application/json" -d '{"patient_name": "Ingress testnew","doctor_name": "Dr
```

Figure 5: Notification Microservice

2.3 PATIENT RECORDS SERVICE

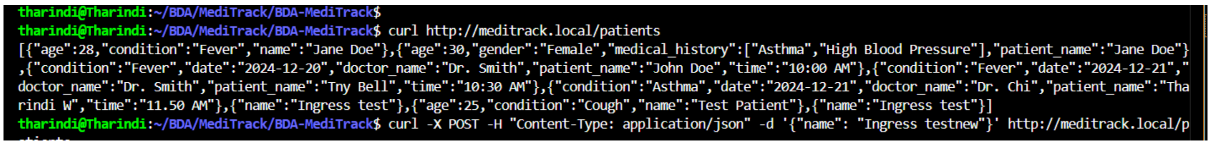
The Patient Records Service is responsible for managing patient-related data such as demographics, medical history, prescriptions, and lab results. It stores data in the Patients collection and provides an interface to add, update, or delete records.

This service is critical for maintaining up-to-date patient information, which can be referenced by other microservices to provide personalized care and insights. Key functionalities include,

- Adding New Patients

- Updating Patient Information
- Deleting Patient Records

This service ensures data integrity and acts as a single source of truth for patient information across the system.



```
tharindi@Tharindi:~/BDA/MediTrack/BDA-MediTrack$
tharindi@Tharindi:~/BDA/MediTrack/BDA-MediTrack$ curl http://meditrack.local/patients
[{"age":28,"condition":"Fever","name":"Jane Doe"},{"age":30,"gender":"Female","medical_history":["Asthma","High Blood Pressure"],"patient_name":"Jane Doe"}
,{"condition":"Fever","date":"2024-12-20","doctor_name":"Dr. Smith","patient_name":"John Doe","time":"10:00 AM"},{"condition":"Fever","date":"2024-12-21","
doctor_name":"Dr. Smith","patient_name":"Tny Bell","time":"10:30 AM"},{"condition":"Asthma","date":"2024-12-21","doctor_name":"Dr. Chi","patient_name":"Tha
rindi W","time":"11.50 AM"},{"name":"Ingress test"},{"age":25,"condition":"Cough","name":"Test Patient"},{"name":"Ingress test"}]
tharindi@Tharindi:~/BDA/MediTrack/BDA-MediTrack$ curl -X POST -H "Content-Type: application/json" -d '{"name": "Ingress testnew"}' http://meditrack.local/p
```

Figure 6: Patient Record Microservice

2.4 ANALYTICS AND AGGREGATION SERVICE

The Analytics and Aggregation Service provides insightful reporting and data analysis based on the information stored by other microservices. It performs scheduled jobs and uses MongoDB aggregation pipelines to derive actionable insights, which help improve healthcare operations and identify trends. Key functionalities include,

1. Number of Appointments per Doctor:

$$A_d = Count\ (appoinments\ per\ doctor\ )$$

2. Frequency of Appointments Over Time:

$$A_t = Count\ (appoinments\ per\ date\ )$$

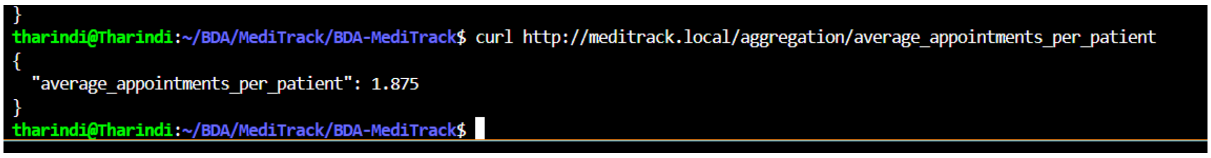
3. Common Symptoms and Conditions by Specialty:

   Groups medical conditions recorded during appointments by the specialty of the attending doctor.

4. Average Appointments per Patient:

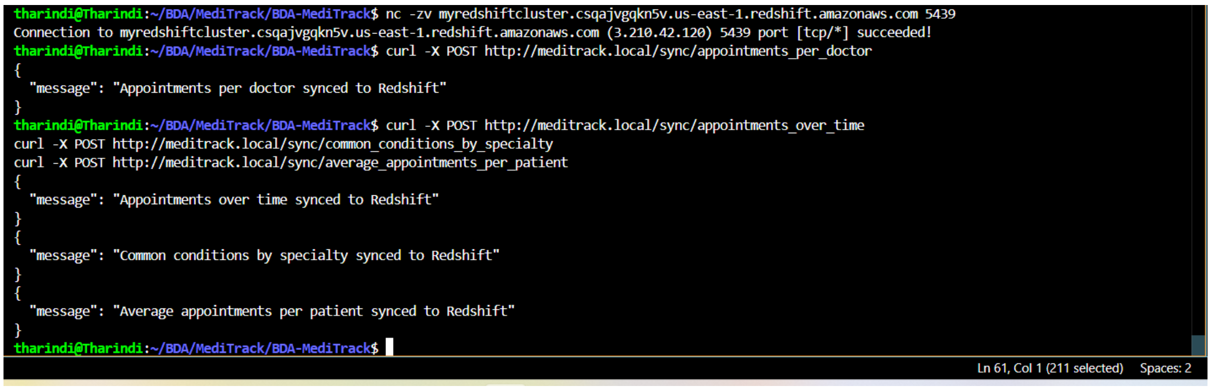
$$Avg_p = \frac{\sum appointments\ per\ patient}{total\ number\ of\ patients}$$

These analytical reports enable hospitals and clinics to make data-driven decisions, optimize their services, and enhance patient outcomes. Initially this was tested with analytics microservice then using a new microservice as redshift analytics same results were pushed to AWS Redshift.



```
}
tharindi@Tharindi:~/BDA/MediTrack/BDA-MediTrack$ curl http://meditrack.local/aggregation/average_appointments_per_patient
{
  "average_appointments_per_patient": 1.875
}
tharindi@Tharindi:~/BDA/MediTrack/BDA-MediTrack$
```

Figure 7: Analytics Microservice



```
tharindi@Tharindi:~/BDA/MediTrack/BDA-MediTrack$ nc -zv myredshiftcluster.csqajvgqkn5v.us-east-1.redshift.amazonaws.com 5439
Connection to myredshiftcluster.csqajvgqkn5v.us-east-1.redshift.amazonaws.com (3.210.42.120) 5439 port [tcp/*] succeeded!
tharindi@Tharindi:~/BDA/MediTrack/BDA-MediTrack$ curl -X POST http://meditrack.local/sync/appointments_per_doctor
{
  "message": "Appointments per doctor synced to Redshift"
}
tharindi@Tharindi:~/BDA/MediTrack/BDA-MediTrack$ curl -X POST http://meditrack.local/sync/appointments_over_time
curl -X POST http://meditrack.local/sync/common_conditions_by_specialty
curl -X POST http://meditrack.local/sync/average_appointments_per_patient
{
  "message": "Appointments over time synced to Redshift"
}
{
  "message": "Common conditions by specialty synced to Redshift"
}
{
  "message": "Average appointments per patient synced to Redshift"
}
tharindi@Tharindi:~/BDA/MediTrack/BDA-MediTrack$
```

Figure 8: Redshift Analytics Microservice

3 DEPLOYMENT

3.1 DEPLOYMENT METHOD

The deployment process begins with a CI/CD pipeline using GitHub Actions, which automates the build and deployment of Dockerized microservices on a Minikube cluster running locally. The pipeline initializes the Minikube environment, builds Docker images within the Minikube VM, applies Kubernetes manifests to deploy the microservices, and runs integration tests to verify functionality. The microservices include a Flask-based API for data storage in MongoDB Atlas and an analytics service scheduled via a Cron job to process data and push results to AWS Redshift daily. This workflow ensures consistent deployments and streamlined testing, with all updates managed through version control in GitHub.

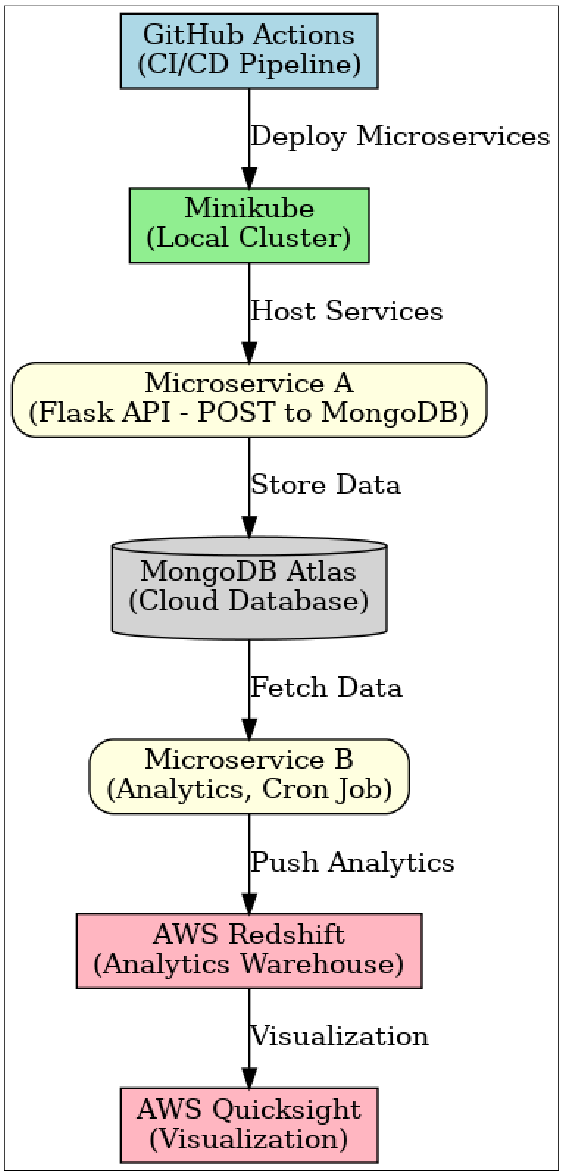
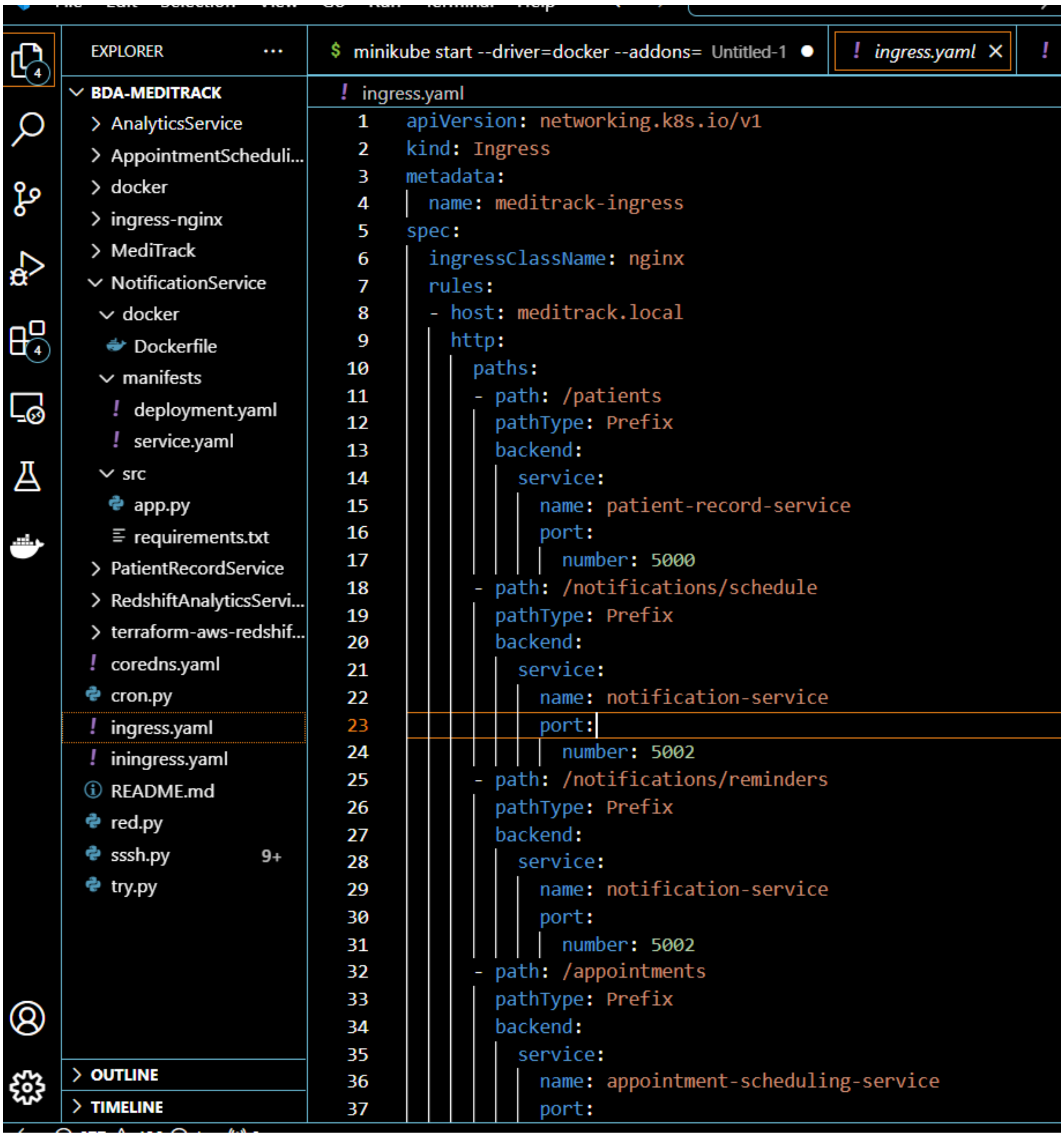

GitHub Actions (CI/CD Pipeline)
Deploy Microservices
Minikube (Local Cluster)
Host Services
Microservice A (Flask API - POST to MongoDB)
Store Data
MongoDB Atlas (Cloud Database)
Fetch Data
Microservice B (Analytics, Cron Job)
Push Analytics
AWS Redshift (Analytics Warehouse)
Visualization
AWS Quicksight (Visualization)

Figure 9: Deployment Diagram

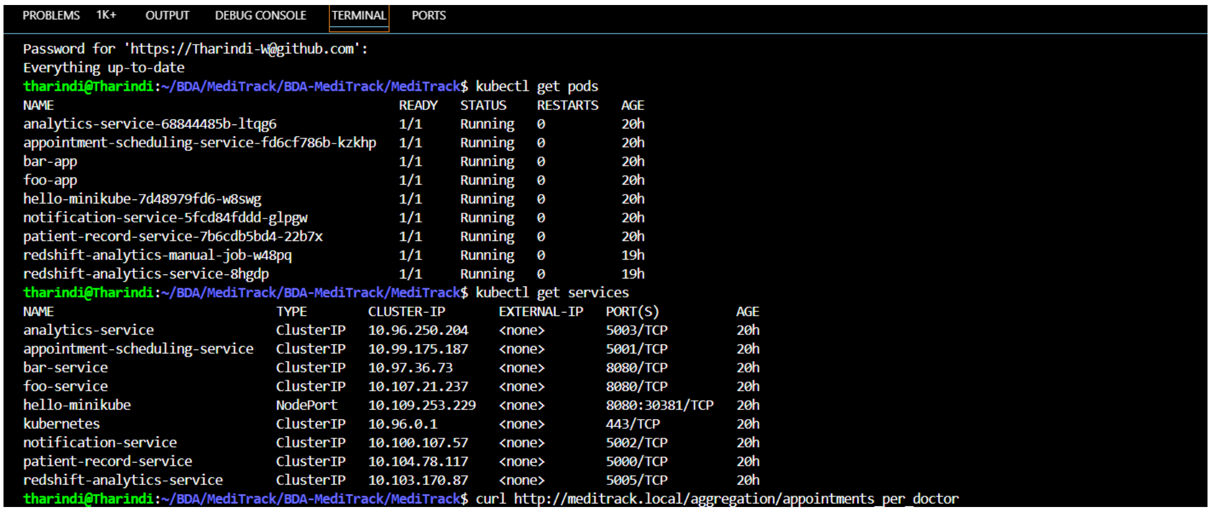
4 MINIKUBE CLUSTER

The Minikube cluster hosts all microservices in a controlled local Kubernetes environment, with each microservice running in its own pod. An NGINX ingress controller, configured through the provided ingress manifest, manages load balancing and routing based on predefined rules, ensuring seamless API access via the meditrack.local domain. This setup directs requests to appropriate backend services like patient records, notifications, appointment scheduling, analytics, and Redshift data synchronization, using their respective paths and ports. Observability is enhanced using the Minikube dashboard.



```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: meditrack-ingress
spec:
  ingressClassName: nginx
  rules:
  - host: meditrack.local
    http:
      paths:
      - path: /patients
        pathType: Prefix
        backend:
          service:
            name: patient-record-service
            port:
              number: 5000
      - path: /notifications/schedule
        pathType: Prefix
        backend:
          service:
            name: notification-service
            port:
              number: 5002
      - path: /notifications/reminders
        pathType: Prefix
        backend:
          service:
            name: notification-service
            port:
              number: 5002
      - path: /appointments
        pathType: Prefix
        backend:
          service:
            name: appointment-scheduling-service
            port:
```

Figure 10: Ingress Configuration and Folder Structure



```
Password for 'https://Tharindi-W@github.com':
Everything up-to-date
tharindi@Tharindi:~/BDA/MediTrack/BDA-MediTrack/MediTrack$ kubectl get pods
NAME                                               READY   STATUS    RESTARTS   AGE
analytics-service-68844485b-ltqg6                  1/1     Running   0          20h
appointment-scheduling-service-fd6cf786b-kzkhp     1/1     Running   0          20h
bar-app                                            1/1     Running   0          20h
foo-app                                            1/1     Running   0          20h
hello-minikube-7d48979fd6-w8swg                    1/1     Running   0          20h
notification-service-5fcd84fddd-glpgw              1/1     Running   0          20h
patient-record-service-7b6cdb5bd4-22b7x            1/1     Running   0          20h
redshift-analytics-manual-job-w48pq                1/1     Running   0          19h
redshift-analytics-service-8hgdp                   1/1     Running   0          19h
tharindi@Tharindi:~/BDA/MediTrack/BDA-MediTrack/MediTrack$ kubectl get services
NAME                             TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
analytics-service                ClusterIP   10.96.250.204    <none>        5003/TCP         20h
appointment-scheduling-service   ClusterIP   10.99.175.187    <none>        5001/TCP         20h
bar-service                      ClusterIP   10.97.36.73      <none>        8080/TCP         20h
foo-service                      ClusterIP   10.107.21.237    <none>        8080/TCP         20h
hello-minikube                   NodePort    10.109.253.229   <none>        8080:30381/TCP   20h
kubernetes                       ClusterIP   10.96.0.1        <none>        443/TCP          20h
notification-service             ClusterIP   10.100.107.57    <none>        5002/TCP         20h
patient-record-service           ClusterIP   10.104.78.117    <none>        5000/TCP         20h
redshift-analytics-service       ClusterIP   10.103.170.87    <none>        5005/TCP         20h
tharindi@Tharindi:~/BDA/MediTrack/BDA-MediTrack/MediTrack$ curl http://meditrack.local/aggregation/appointments_per_doctor
```

Figure 11: Minikube Pods and Services

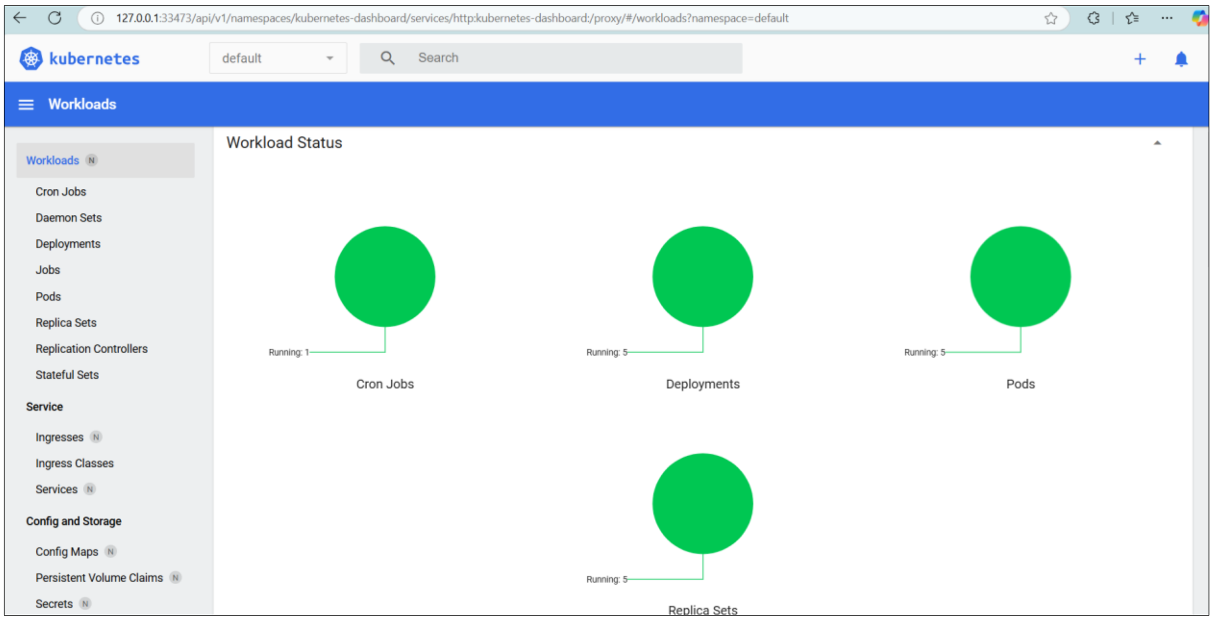
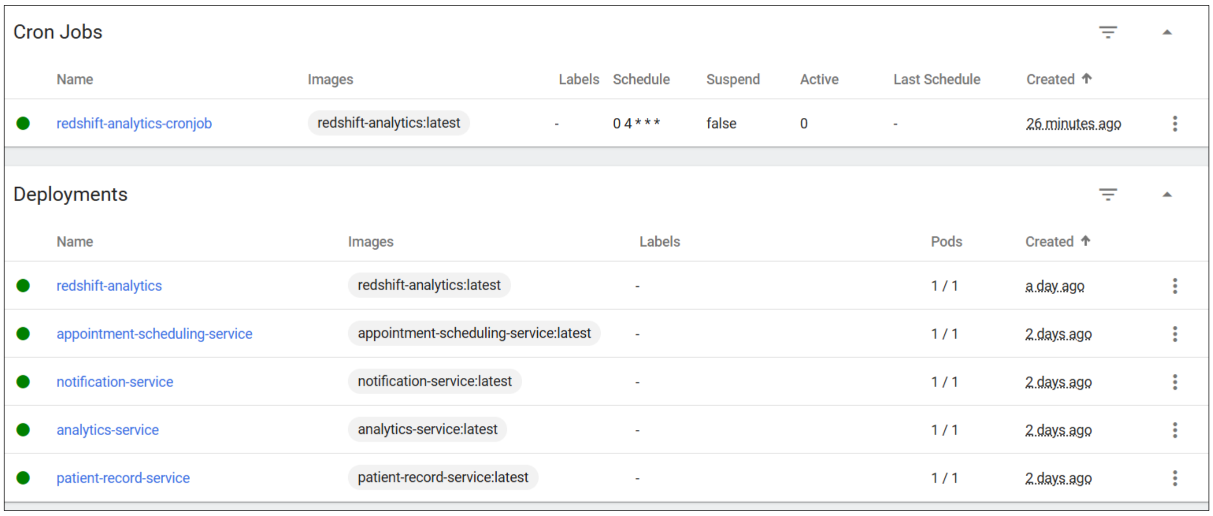


Figure 12: Minikube Pod Health Dashboard



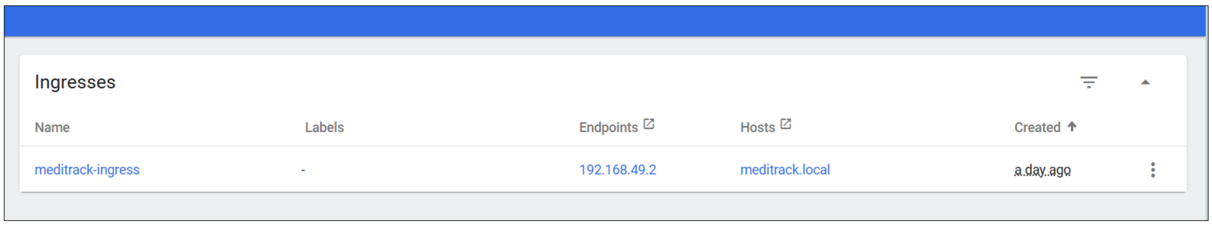
Cron Jobs

	Name	Images	Labels	Schedule	Suspend	Active	Last Schedule	Created ↑
●	redshift-analytics-cronjob	redshift-analytics:latest	-	0 4 * * *	false	0	-	26 minutes ago

Deployments

	Name	Images	Labels	Pods	Created ↑
●	redshift-analytics	redshift-analytics:latest	-	1 / 1	a day ago
●	appointment-scheduling-service	appointment-scheduling-service:latest	-	1 / 1	2 days ago
●	notification-service	notification-service:latest	-	1 / 1	2 days ago
●	analytics-service	analytics-service:latest	-	1 / 1	2 days ago
●	patient-record-service	patient-record-service:latest	-	1 / 1	2 days ago

Figure 13: Minikube Dashboard - Cron Jobs and Deployments



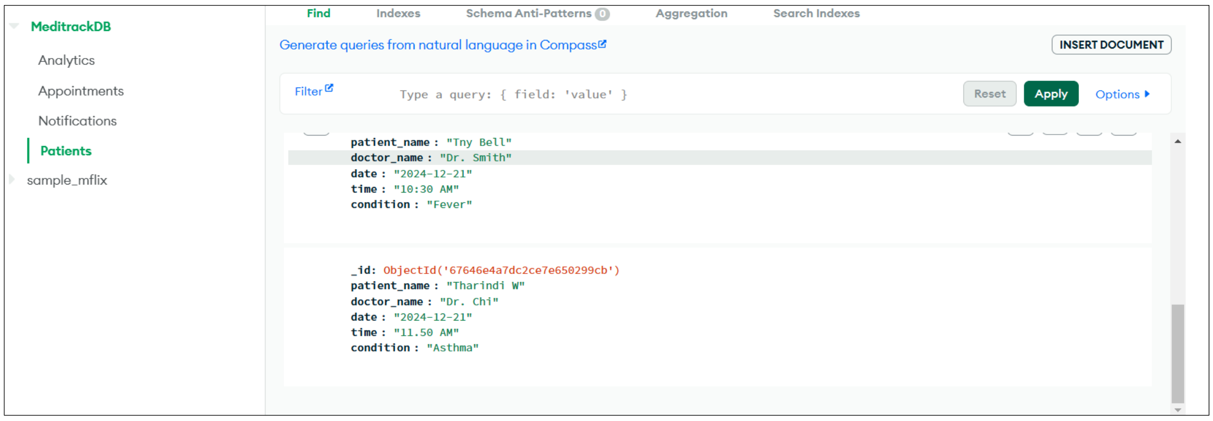
Ingresses

Name	Labels	Endpoints	Hosts	Created
meditrack-ingress	-	192.168.49.2	meditrack.local	a day ago

Figure 14: Minikube Dashboard Ingress

5 MONGODB

MongoDB Atlas, a managed NoSQL database, was chosen for its scalability, flexibility, and ability to handle semi-structured medical data efficiently. Its distributed architecture ensures high availability and performance. Flask-based microservices connect securely to the cluster using a connection string and the pymongo driver. This setup ensures seamless integration, secure communication, and efficient data management.

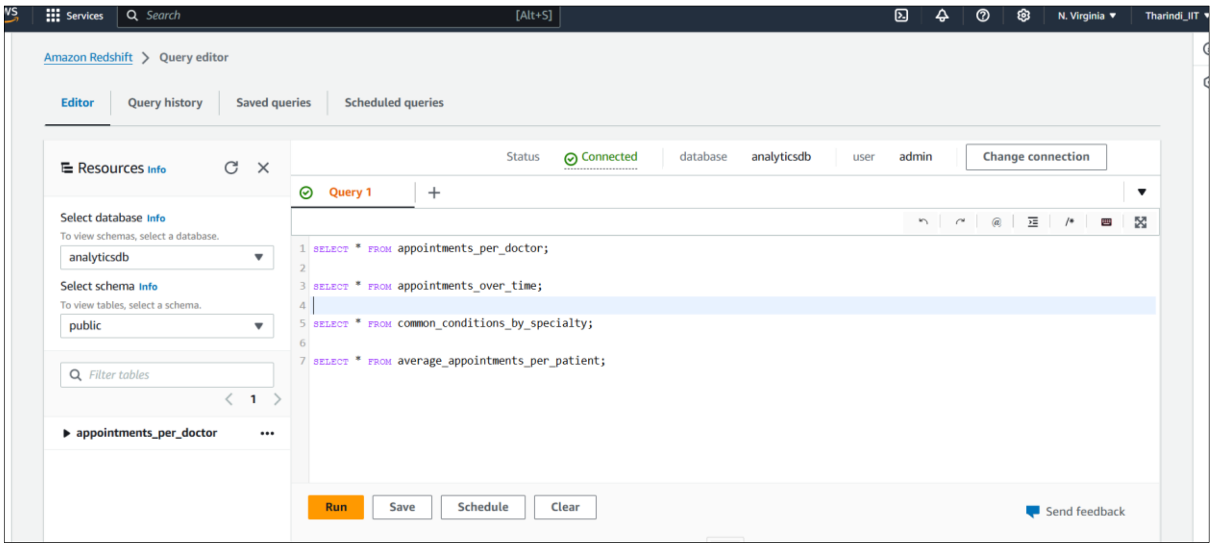


```
patient_name : "Tny Bell"
doctor_name : "Dr. Smith"
date : "2024-12-21"
time : "10:30 AM"
condition : "Fever"

_id: ObjectId('67646e4a7dc2ce7e650299cb')
patient_name : "Tharindi W"
doctor_name : "Dr. Chi"
date : "2024-12-21"
time : "11.50 AM"
condition : "Asthma"
```

Figure 15: MongoDB

6 REDSHIFT

AWS Redshift is deployed within a VPC, leveraging three subnets across different availability zones for high availability and fault tolerance. Custom route tables are configured for secure data flow within the VPC, while IAM roles enable granular access control for data ingestion and analytics tasks. Security groups restrict inbound and outbound traffic, ensuring only authorized services and Quicksight have access to the database. This setup provides a robust and secure data warehouse for processing analytics.



Select database: analyticsdb

Select schema: public

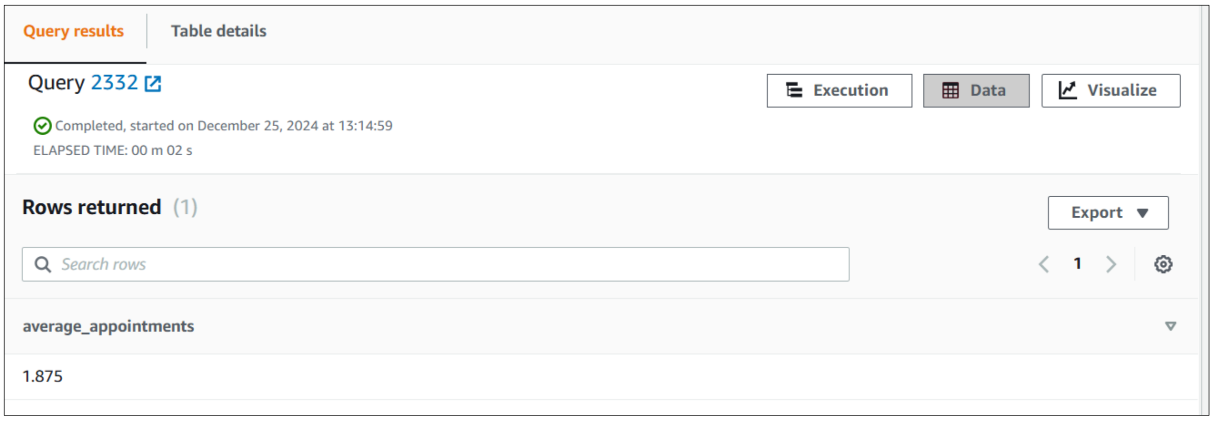
Status: Connected | database: analyticsdb | user: admin | Change connection

Query 1

```
SELECT * FROM appointments_per_doctor;

SELECT * FROM appointments_over_time;

SELECT * FROM common_conditions_by_specialty;

SELECT * FROM average_appointments_per_patient;
```

appointments_per_doctor

Figure 16: AWS Redshift Queries



Query 2332

Completed, started on December 25, 2024 at 13:14:59

ELAPSED TIME: 00 m 02 s

Rows returned (1)

average_appointments
1.875

Figure 17: AWS Redshift Query Results

7 QUICKSIGHT DASHBOARD

AWS QuickSight integrates with Redshift to deliver interactive visualizations and dashboards. QuickSight's user-friendly interface and rich visualization capabilities

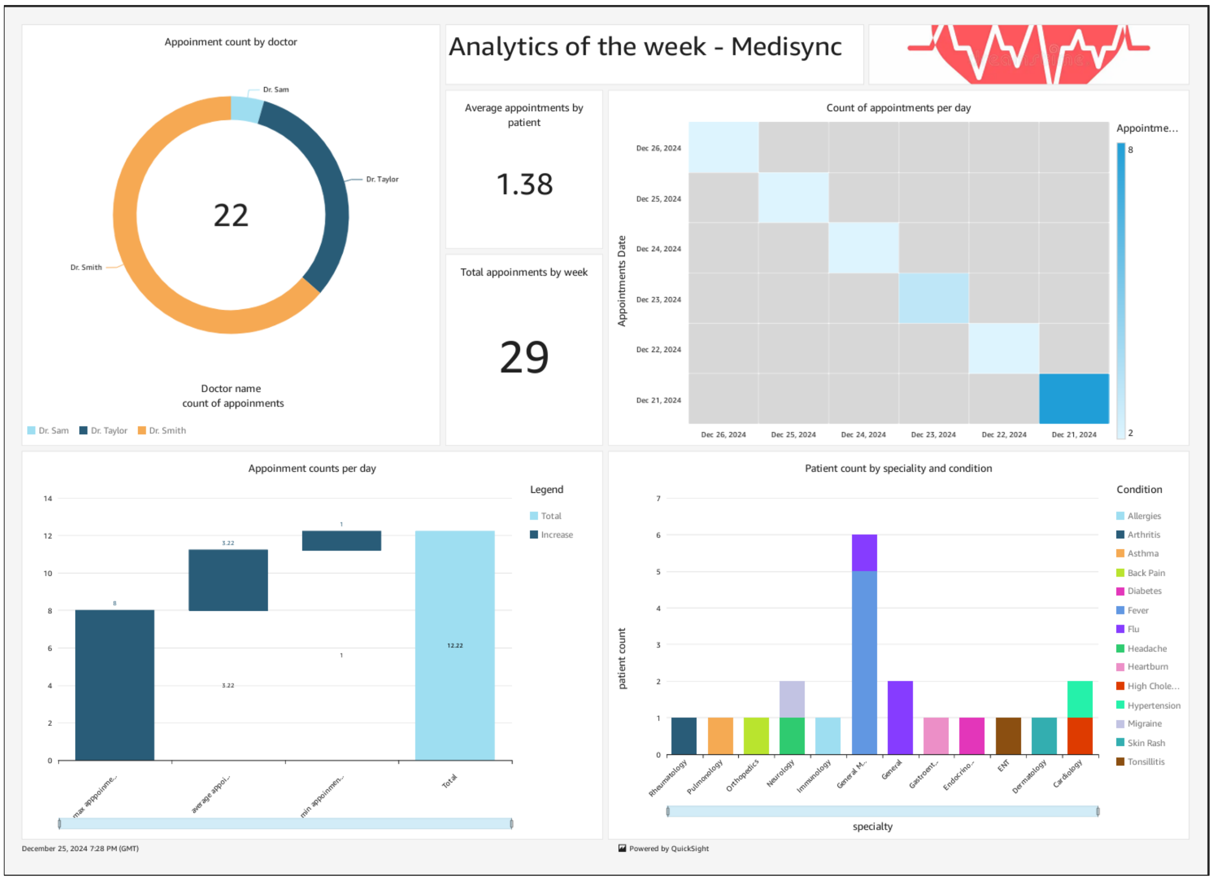
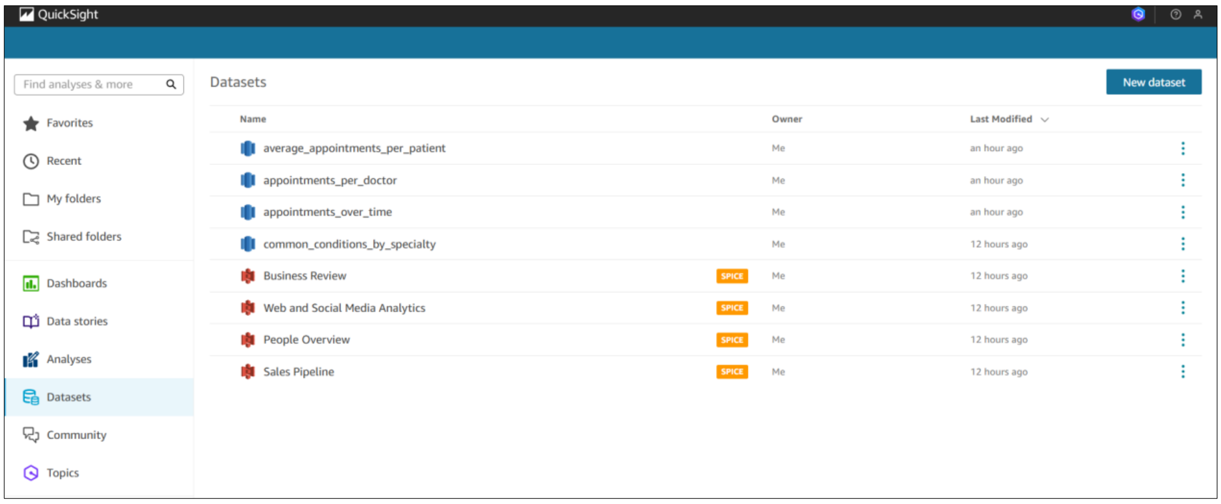


Figure 18: AWS QuickSight Dashboard



QuickSight

Find analyses & more

- Favorites
- Recent
- My folders
- Shared folders
- Dashboards
- Data stories
- Analyses
- Datasets
- Community
- Topics

Datasets

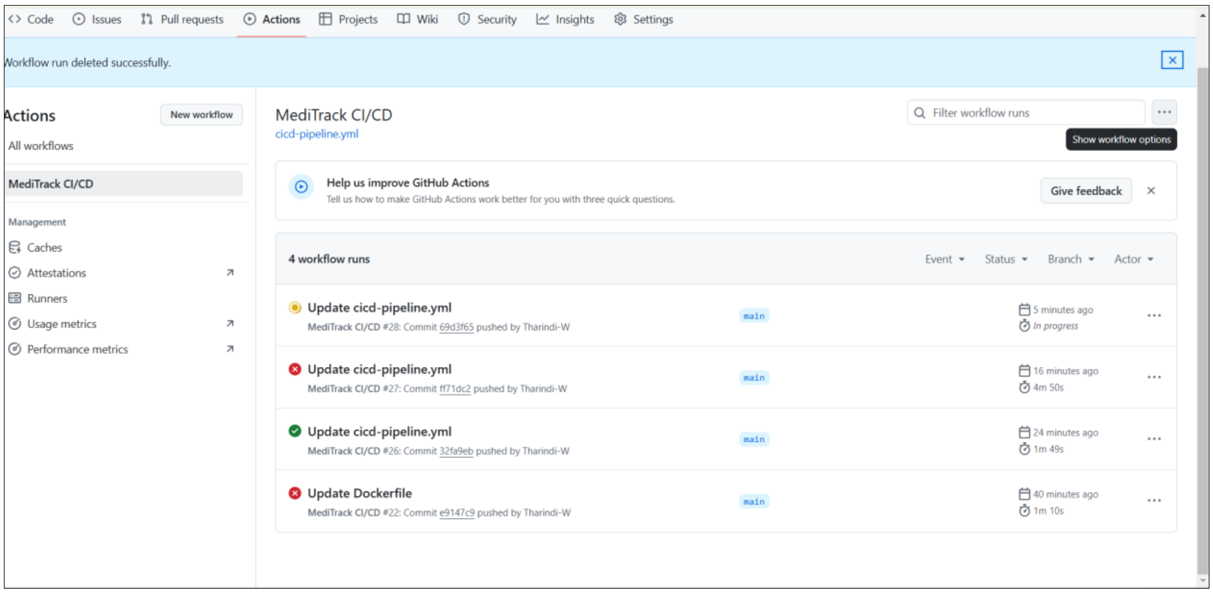
New dataset

Name		Owner	Last Modified
average_appointments_per_patient		Me	an hour ago
appointments_per_doctor		Me	an hour ago
appointments_over_time		Me	an hour ago
common_conditions_by_specialty		Me	12 hours ago
Business Review	SPICE	Me	12 hours ago
Web and Social Media Analytics	SPICE	Me	12 hours ago
People Overview	SPICE	Me	12 hours ago
Sales Pipeline	SPICE	Me	12 hours ago

Figure 19: Data Sources for Quicksight

8 CI/CD

The CI/CD pipeline, implemented using GitHub Actions, automates the deployment process for microservices on a Minikube cluster. It initializes Minikube, builds Docker images in the Minikube VM, applies Kubernetes manifests, and performs integration tests to ensure service functionality. This setup ensures seamless version control, automated updates, and reliable testing before production deployment.



Workflow run deleted successfully.

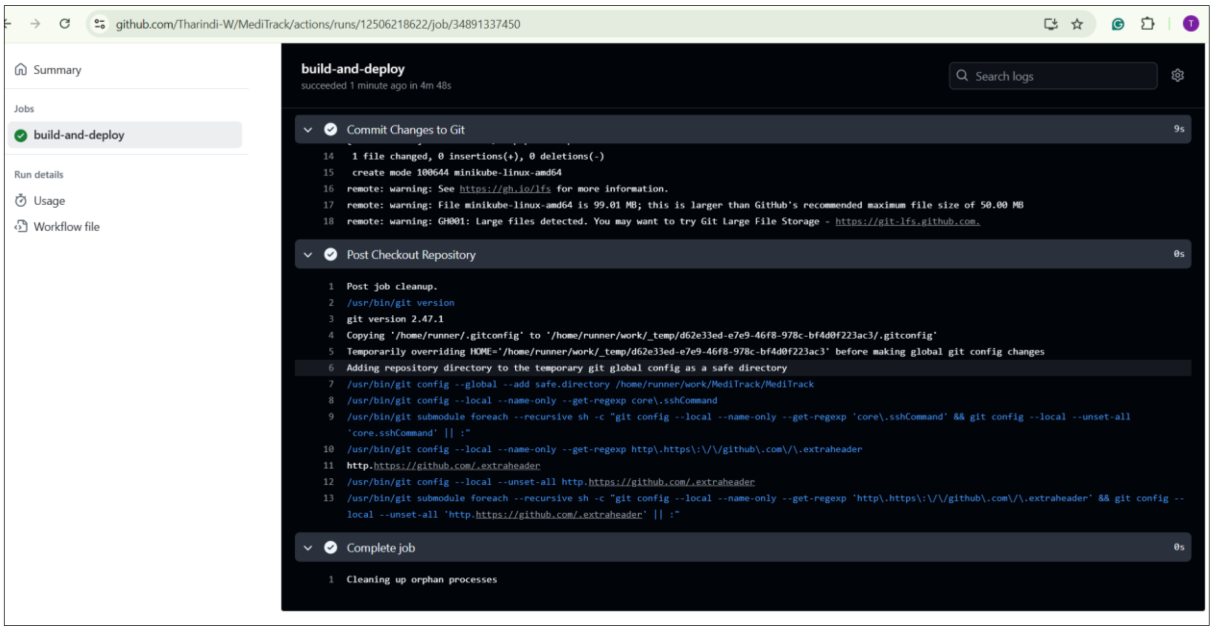
MediTrack CI/CD
cicd-pipeline.yml

Help us improve GitHub Actions
Tell us how to make GitHub Actions work better for you with three quick questions.

4 workflow runs

Run	Branch	Time
Update cicd-pipeline.yml — MediTrack CI/CD #28: Commit 69d3f65 pushed by Tharindi-W	main	5 minutes ago, In progress
Update cicd-pipeline.yml — MediTrack CI/CD #27: Commit ff71dc2 pushed by Tharindi-W	main	16 minutes ago, 4m 50s
Update cicd-pipeline.yml — MediTrack CI/CD #26: Commit 32fa9eb pushed by Tharindi-W	main	24 minutes ago, 1m 49s
Update Dockerfile — MediTrack CI/CD #22: Commit e9147c9 pushed by Tharindi-W	main	40 minutes ago, 1m 10s

Figure 20: Github Actions Red-Green Workflow

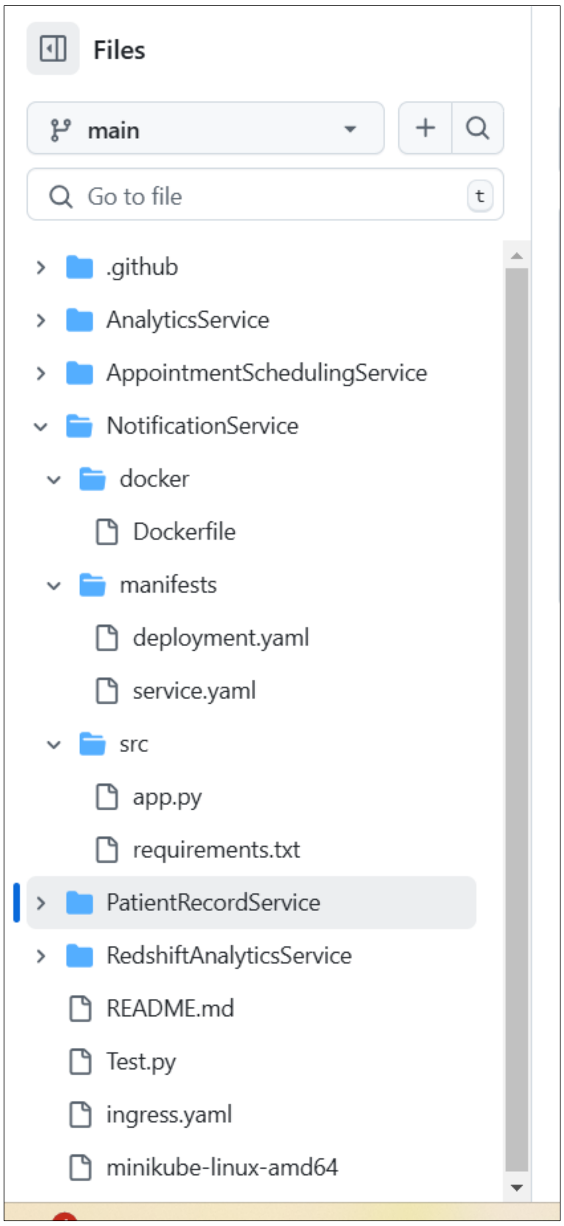


github.com/Tharindi-W/MediTrack/actions/runs/12506218622/job/34891337450

build-and-deploy
succeeded 1 minute ago in 4m 48s

```
Commit Changes to Git                                                      9s
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 minikube-linux-amd64
remote: warning: See https://gh.io/lfs for more information.
remote: warning: File minikube-linux-amd64 is 99.01 MB; this is larger than GitHub's recommended maximum file size of 50.00 MB
remote: warning: GH001: Large files detected. You may want to try Git Large File Storage - https://git-lfs.github.com.

Post Checkout Repository                                                   0s
Post job cleanup.
/usr/bin/git version
git version 2.47.1
Copying '/home/runner/.gitconfig' to '/home/runner/work/_temp/d62e33ed-e7e9-46f8-978c-bf4d0f223ac3/.gitconfig'
Temporarily overriding HOME='/home/runner/work/_temp/d62e33ed-e7e9-46f8-978c-bf4d0f223ac3' before making global git config changes
Adding repository directory to the temporary git global config as a safe directory
/usr/bin/git config --global --add safe.directory /home/runner/work/MediTrack/MediTrack
/usr/bin/git config --local --name-only --get-regexp core\.sshCommand
/usr/bin/git submodule foreach --recursive sh -c "git config --local --name-only --get-regexp 'core\.sshCommand' && git config --local --unset-all
'core.sshCommand' || :"
/usr/bin/git config --local --name-only --get-regexp http\.https\:\/\/github\.com\/\.extraheader
http.https://github.com/.extraheader
/usr/bin/git config --local --unset-all http.https://github.com/.extraheader
/usr/bin/git submodule foreach --recursive sh -c "git config --local --name-only --get-regexp 'http\.https\:\/\/github\.com\/\.extraheader' && git config --
local --unset-all 'http.https://github.com/.extraheader' || :"

Complete job                                                               0s
Cleaning up orphan processes
```

Figure 21: Github Action CI/CD



Files

main

Go to file

```
> .github
> AnalyticsService
> AppointmentSchedulingService
v NotificationService
  v docker
      Dockerfile
  v manifests
      deployment.yaml
      service.yaml
  v src
      app.py
      requirements.txt
> PatientRecordService
> RedshiftAnalyticsService
  README.md
  Test.py
  ingress.yaml
  minikube-linux-amd64
```

Figure 22: Github Folder Structure

9 SECURITY AND ETHICS

9.1.1 Security Challenges

1. Sensitive Data Protection (Patient Data),
   Storing and transmitting patient health data without encryption exposes sensitive personal health information (PHI) to potential breaches. Therefore, implementing end to end encryption for data both in transit and at rest, utilizing industry standard encryption protocols like TLS for data transmission and AES for data storage or exploring the use of AWS Key Management Service (KMS) to manage and rotate encryption keys securely would help overcome this issue.
2. Access Control and Authentication
   Exposed public repositories and improper handling of API keys and database credentials could lead to unauthorized access to the system. Implementing secure authentication mechanisms using OAuth or OpenID Connect for API access and microservice authentication. Storing sensitive information such as passwords, API keys, and database credentials in AWS Secrets Manager or similar secure services can be used. In the solution used GitHub Actions secrets for securely managing credentials in CI/CD pipelines.
3. Container Security
   Running containers in the Minikube Kubernetes cluster without proper security measures exposes the system to vulnerabilities such as unauthorized access and malicious attacks. Regularly scanning container images for vulnerabilities using tools like Clair or Anchore. Implementing Pod Security Policies (PSP) and Role Based Access Control (RBAC) in Kubernetes to restrict access can be used.
4. Public Repository Risks
   Exposing source code and configuration files in a public GitHub repository increases the risk of revealing sensitive information, such as API keys and database connection details. Avoided committing sensitive information by using .gitignore files and ensuring that all credentials are stored in environment variables or GitHub Actions secrets. Considering switching to private repositories for production code to better protect sensitive information can be used.

9.1.2 Ethics Challenges

1. Patient Consent and Data Transparency
   Collecting, storing, and processing patient data without clear consent can raise concerns regarding patient autonomy and privacy. Implementing clear and transparent consent mechanisms during patient registration, ensuring patients are aware of what data is being collected and how it will be used has to be concerned.
2. Data Privacy and Compliance
   Storing patient health data without ensuring compliance with relevant privacy regulations, such as HIPAA or GDPR, could lead to serious legal and ethical issues. Ensuring compliance with privacy regulations by implementing strong data access controls, audit logs, and user consent management systems. Adhering to data minimization practices by collecting only necessary patient information and implementing robust data protection measures must be implemented.

3. Fair Evaluation and Use of Analytics
   The use of analytics and productivity metrics may lead to biased or unfair evaluations of healthcare providers or patients. Ensuring that any analytics or metrics used within the platform are employed responsibly and focused on improving patient care and provider performance. Providing transparency into how metrics are calculated and used to avoid misuse for unfair comparisons or evaluations is the best.

10 RUNBOOK

10.1 INSTALLING MINIKUBE

1. **Install Docker** - Minikube requires Docker as a container runtime. Install Docker using the following commands,

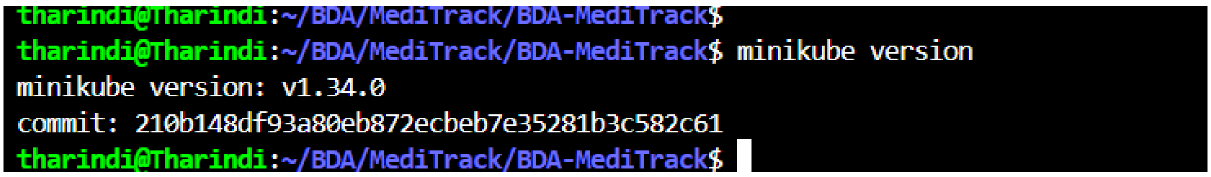
```
sudo apt update
sudo apt install docker.io -y
```

2. **Install Minikube** - Download and install Minikube from the official website or use the following commands,

```
curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64
sudo install minikube-linux-amd64 /usr/local/bin/minikube
```

3. **Verify Installation** - Run the command to confirm Minikube is installed,

```
minikube version
```



```
tharindi@Tharindi:~/BDA/MediTrack/BDA-MediTrack$
tharindi@Tharindi:~/BDA/MediTrack/BDA-MediTrack$ minikube version
minikube version: v1.34.0
commit: 210b148df93a80eb872ecbeb7e35281b3c582c61
tharindi@Tharindi:~/BDA/MediTrack/BDA-MediTrack$
```

Figure 23: Minikube Version Check

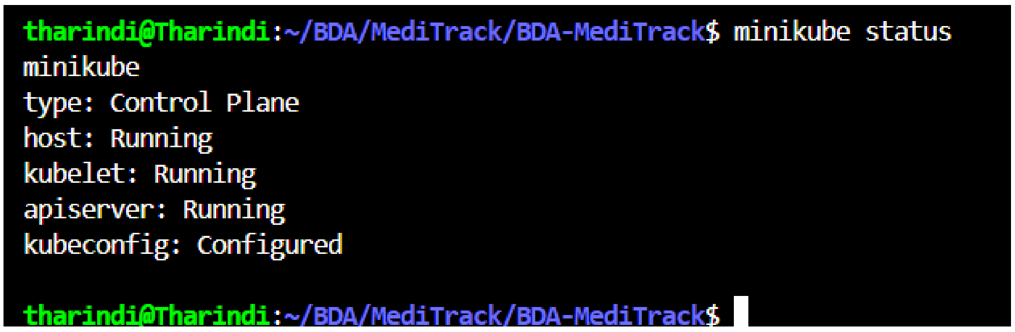
10.2 STARTING MINIKUBE

1. **Start Minikube** - Use the following command to start Minikube,

```
minikube start --driver=docker
```

2. **Verify Minikube Status** - Confirm Minikube is running,

```
minikube status
```



```
tharindi@Tharindi:~/BDA/MediTrack/BDA-MediTrack$ minikube status
minikube
type: Control Plane
host: Running
kubelet: Running
apiserver: Running
kubeconfig: Configured

tharindi@Tharindi:~/BDA/MediTrack/BDA-MediTrack$
```

Figure 24: Minikube Status Check

10.3 Enabling Minikube Add-ons

1. **Enable Ingress** - Minikube's ingress addon is required for managing traffic,

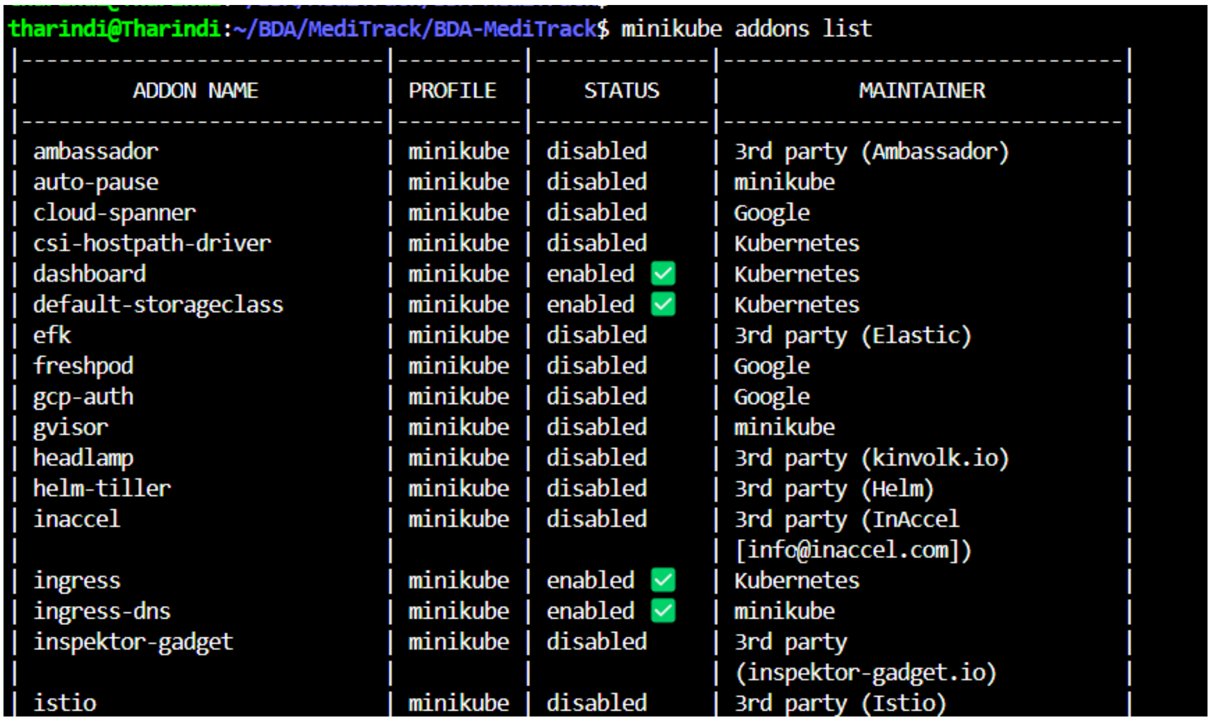
   ```
   minikube addons enable ingress
   ```

2. **Enable Dashboard -** To visually monitor Minikube resources,

   ```
   minikube addons enable dashboard
   ```

3. **Access the Dashboard** - Open the Kubernetes dashboard,

   ```
   minikube dashboard
   ```



```
tharindi@Tharindi:~/BDA/MediTrack/BDA-MediTrack$ minikube addons list
|-----------------------------|----------|------------|--------------------------------|
|         ADDON NAME          | PROFILE  |   STATUS   |           MAINTAINER           |
|-----------------------------|----------|------------|--------------------------------|
| ambassador                  | minikube | disabled   | 3rd party (Ambassador)         |
| auto-pause                  | minikube | disabled   | minikube                       |
| cloud-spanner               | minikube | disabled   | Google                         |
| csi-hostpath-driver         | minikube | disabled   | Kubernetes                     |
| dashboard                   | minikube | enabled ✅ | Kubernetes                     |
| default-storageclass        | minikube | enabled ✅ | Kubernetes                     |
| efk                         | minikube | disabled   | 3rd party (Elastic)            |
| freshpod                    | minikube | disabled   | Google                         |
| gcp-auth                    | minikube | disabled   | Google                         |
| gvisor                      | minikube | disabled   | minikube                       |
| headlamp                    | minikube | disabled   | 3rd party (kinvolk.io)         |
| helm-tiller                 | minikube | disabled   | 3rd party (Helm)               |
| inaccel                     | minikube | disabled   | 3rd party (InAccel             |
|                             |          |            | [info@inaccel.com])            |
| ingress                     | minikube | enabled ✅ | Kubernetes                     |
| ingress-dns                 | minikube | enabled ✅ | minikube                       |
| inspektor-gadget            | minikube | disabled   | 3rd party                      |
|                             |          |            | (inspektor-gadget.io)          |
| istio                       | minikube | disabled   | 3rd party (Istio)              |
```

Figure 25: Minikube Addon list

10.4 Setting Up Microservices

1. **Build Docker Images -** Create Docker images for your Flask and analytics microservices.

   ```
   docker build -t patient-record-service:latest -f docker/Dockerfile .
   ```

2. **Push Images to Minikube** - Transfer the images to Minikube's internal Docker registry,

   ```
   eval $(minikube docker-env)
   docker images
   ```

3. **Deploy Microservices**

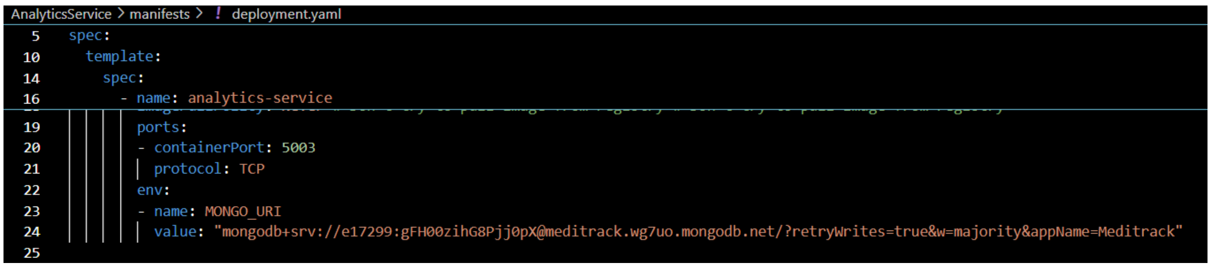
- Write Kubernetes manifests (YAML files) for Flask and analytics services.
- Apply the manifests,

```
kubectl apply -f analytics-service.yaml
```

Refer Figure 11

10.5 Connecting MongoDB

1. **Deploy MongoDB** – Use MongoDB Atlas and verify connectivity and networks
2. **Update Microservice Configuration**
   - Add the MongoDB connection URL to your Flask and analytics microservices' environment variables.
   - Redeploy the microservices with updated configurations.



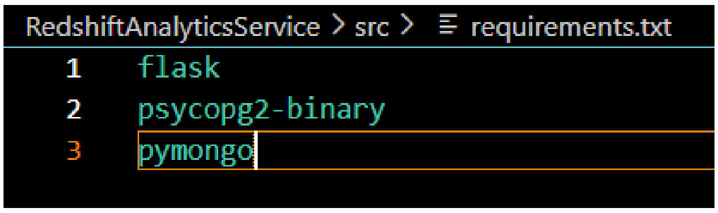
AnalyticsService > manifests > deployment.yaml

```yaml
spec:
  template:
    spec:
      - name: analytics-service
        ports:
        - containerPort: 5003
          protocol: TCP
        env:
        - name: MONGO_URI
          value: "mongodb+srv://e17299:gFH00zihG8Pjj0pX@meditrack.wg7uo.mongodb.net/?retryWrites=true&w=majority&appName=Meditrack"
```

Figure 26: MongoDB URL

10.6 Connecting to Amazon Redshift

1. **Configure Redshift**
   - Create a Redshift cluster in AWS and note its connection details (IAM roles, VPC, Subnets, Security groups, port, database name, etc.).
2. **Update Microservices**
   - Add the Redshift credentials to the analytics microservice configuration.
   - Install a Redshift Python driver (e.g., psycopg2) in the microservice using requiremts.txt



RedshiftAnalyticsService > src > requirements.txt

```
flask
psycopg2-binary
pymongo
```

Figure 27: Redshift Python Driver in Requirements.txt

10.7 Setting Up QuickSight

1. **Connect to Redshift**,
   - In the AWS QuickSight interface, add Redshift as a data source.
2. **Create Visualizations**,

- Use the connected dataset to build dashboards and reports.

Refer Figure 18 and Figure 19

10.8 Configuring GitHub Actions

1. **Set Up Workflow**
    - Add a .github/workflows/deploy.yml file with the following:

```
name: Deploy to Minikube
on:
  push:
    branches:
      - main
jobs:
  deploy:
    runs-on: ubuntu-latest
    steps:
    - name: Checkout code
      uses: actions/checkout@v3
    - name: Set up Docker
      uses: docker/setup-buildx-action@v2
    - name: Build and push Docker image
      run: |
        docker build -t flask-service:latest ./flask-service
        docker build -t analytics-service:latest ./analytics-service
    - name: Deploy to Minikube
      run: |
        kubectl apply -f flask-service.yaml
        kubectl apply -f analytics-service.yaml
```

10.9 Testing the Workflow

1. **Send Sample Requests**,
    - Use tools like curl or Postman to test the Flask microservice

```
curl -X POST http://<minikube_ip>:<node_port>
```

2. **Verify Data Flow**:
    - Check MongoDB for data storage.
    - Confirm analytics results in Redshift.

This runbook provides a step-by-step guide to deploying and testing a Minikube-based microservices solution, ensuring connectivity with MongoDB, Redshift, and QuickSight, with automation powered by GitHub Actions.

REFERENCE

https://dev.to/bravinsimiyu/building-and-deploying-a-flask-application-to-kubernetes-cluster-58e9

https://medium.com/@ikbenezer/how-to-create-an-amazon-elastic-kubernetes-service-cluster-using-terraform-c615eca68c54

https://minikube.sigs.k8s.io/docs/
https://github.blog/enterprise-software/ci-cd/build-ci-cd-pipeline-github-actions-four-steps/