Course- SEZG583 SCALABLE SERVICES

Assignment- Micro-services Development and Deployment

Group Details

Group No:

Group Member Names

Sl No	BITS ID	Name
1	2023MT03187	Gaurav Singhal
2	2023MT03146	Palan Bhavin Bharatbhai
3	2023MT03164	Deepak Khandelwal

Note: While All information is added in this document, Below Github link has all the code files, and outputs.

Code Link :

Description of Application

This application is to design a scalable online solution DOCTO which provides services like online appointments, medical records, Billing and payments for both doctors and patients.

Functional Requirement :

1. Patient Management
2. Appointment Management
3. Doctor Management
4. Payment Management
5. Alert Management

Non Functional Requirement

1. High Availability of critical services - Billing, Patient Data, Doctor Availability
2. Scalable Solution - Able to handle traffic and future growth
3. Data Redundancy

Part 1 - Design

Core Functional Use Cases:

1. As a patient I want register my self to DOCTO.
2. As a patient, I should be able to search doctors for a specialization.
3. As a patient, I want to book an appointment for a doctor.
4. As a Patient, I want to see my user profile.
5. As a patient, I want to see my medical history.
6. As a patient, I should be able to pay Doctor Fee.
7. As a patient, I should receive notifications about by appointments.
8. As a doctor, I want to register my self to DOCTO
9. As a doctor, I want to see my appointments for a given date.
10. As a doctor, I want to see my patient's medical history.
11. As a doctor, I want to cancel my appointments for a day.
12. As a doctor, I should have way to accept/reject a new appointment.
13. As a doctor, I should be able to setup my availability..
14. As a doctor, I should receive notifications about by appointments.

Identifying System Operations

There are 2 types of system Operations - Command and Query

Actor	UseCase	Command	Query	Description
Patient	Register	createPatient()		Creates a Patient record
Patient	Doctor Lookup		searchDoctor()	Doctor for a specialization(Pediatrics, Gynecologist)
Patient	Appointment	bookAppointment()		Books an appointment for a doctor and doctor is available
Patient	Profile		accessPatientInfo()	View Patient profile
Patient	My Medical History		fetchMedicalRecords()	
Patient	Payment	payFee()		Fees payment for Doctor appointment
Doctor	Doctor Registration	onboardDoctor()		Add a new doctor to the panel
Doctor	Appointment List		fetchAllAppointments()	Fetch appointments for a day.
Doctor	Particular Patient Medical History		fetchMedicalRecords()	Medical history of my patient
Doctor	Cancel Appointments	cancelAppointments()		Cancel all appointments for a day
Doctor	Accept Appointment	acceptAppointment()		Doctor Receives a new appointment request and can accept/Reject it.
Doctor	Set Availability	setupSlots()		Doctor can define all slots doctor provides for his/her availability.

* Above are **System operations**, there will be multiple internal operations called by other services. Where Services act as an Actor. Ex - While booking an appointment, It looks up doctors from Doctor Service, for a selected Doctor, appointment service will call the Doctor Service to check the available slots, then fetches the Patient Information from Patient Service and create an appointment in “PENDING CONFIRMATION” State.

Identifying Business Capability

An organizations business capabilities captures what an organization's business is.

Patient Management: Managing information about Patients.

Doctor Management: Managing Doctors profile, their available days and location details.

Appointment Fulfillment:

- Appointment Management - Enabling Patient to create and manage appointments.

- Doctor Availability Management - Managing the available slot for a doctor.

- Notification Management - Reminding Patient and Doctor about appointments. Sending notification to doctor to accept/reject an appointment.

- Prescription Management - After appointment managing the prescriptions for future reference.

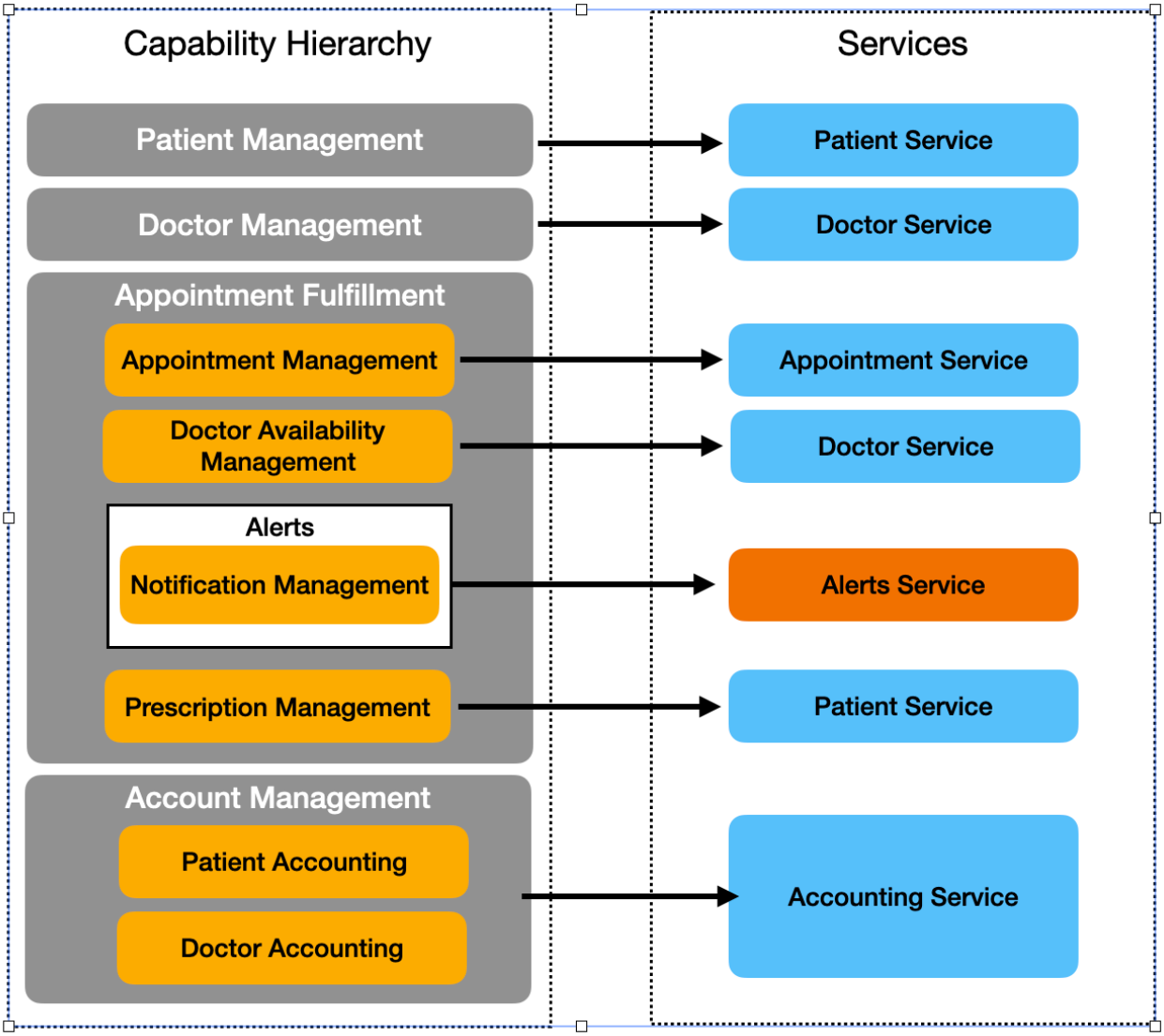
Accounting:

- Patient Accounting - Managing billing of consumers.

- Doctor accounting - Managing payments to doctors.

Business Capability to Service Mapping

A service needs to be defined for each capability of group of related capabilities.



System Operations to Services Assignment

Service	Operation	Description
Patient Service	createPatient(), accessPatientInfo() fetchMedicalRecords() verifyPatientDetail()	This service provides all operations related to a patient Management
Doctor Service	searchDoctor() onboardDoctor() setupSlots() acceptAppointment()	This Service provides all operation for Doctor Management.
Appointment Service	bookAppointment() fetchAllAppointments() cancelAppointments() updateAppointmentStatus()	All operations related to Appointment Management.
Alert Service	sendNotification()	This Service is internally used by other services.
Account Service	payFee()	Account management Service.

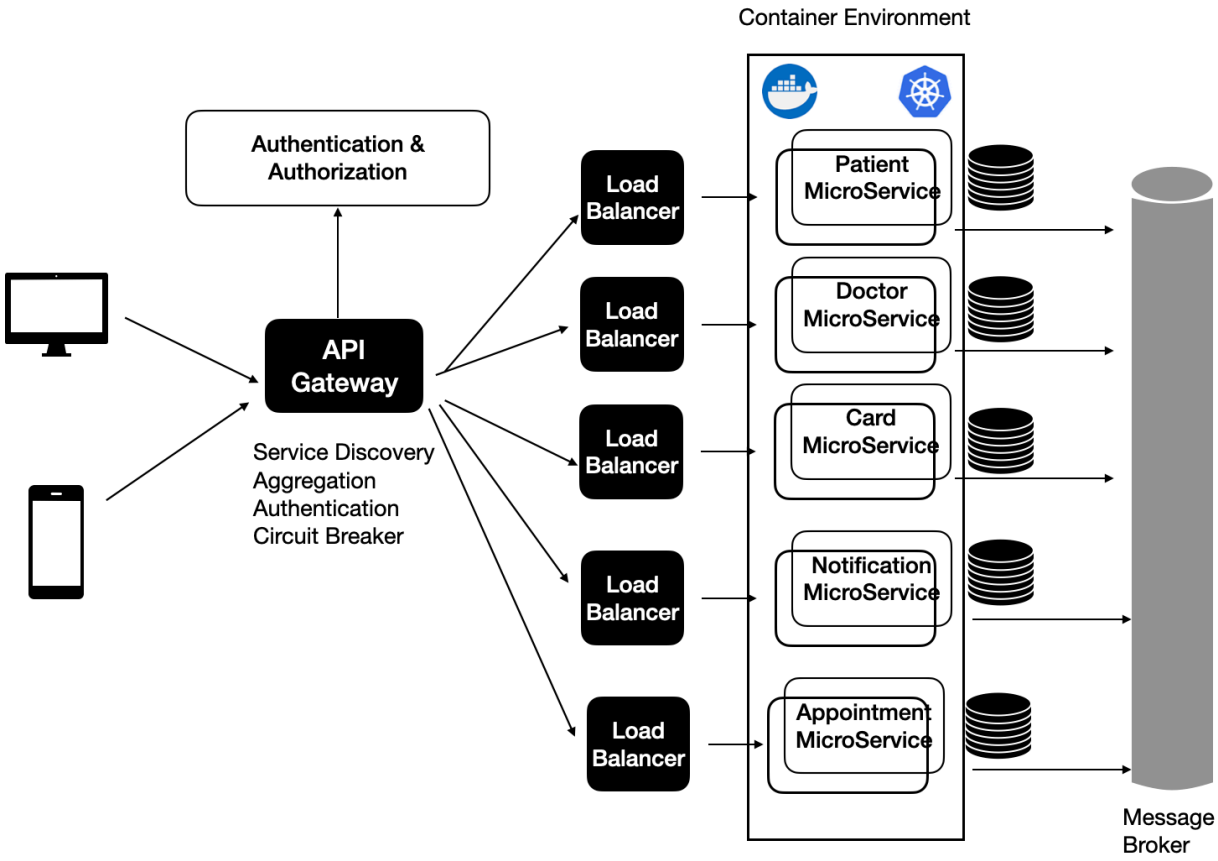
Identify Collaboration between Services

While many of the services, individually can handle the complete use case (Ex. Patient Registration) Other Services like Appointment Management would require to co-ordinate with multiple services.

Collaboration for create Appointment Service -

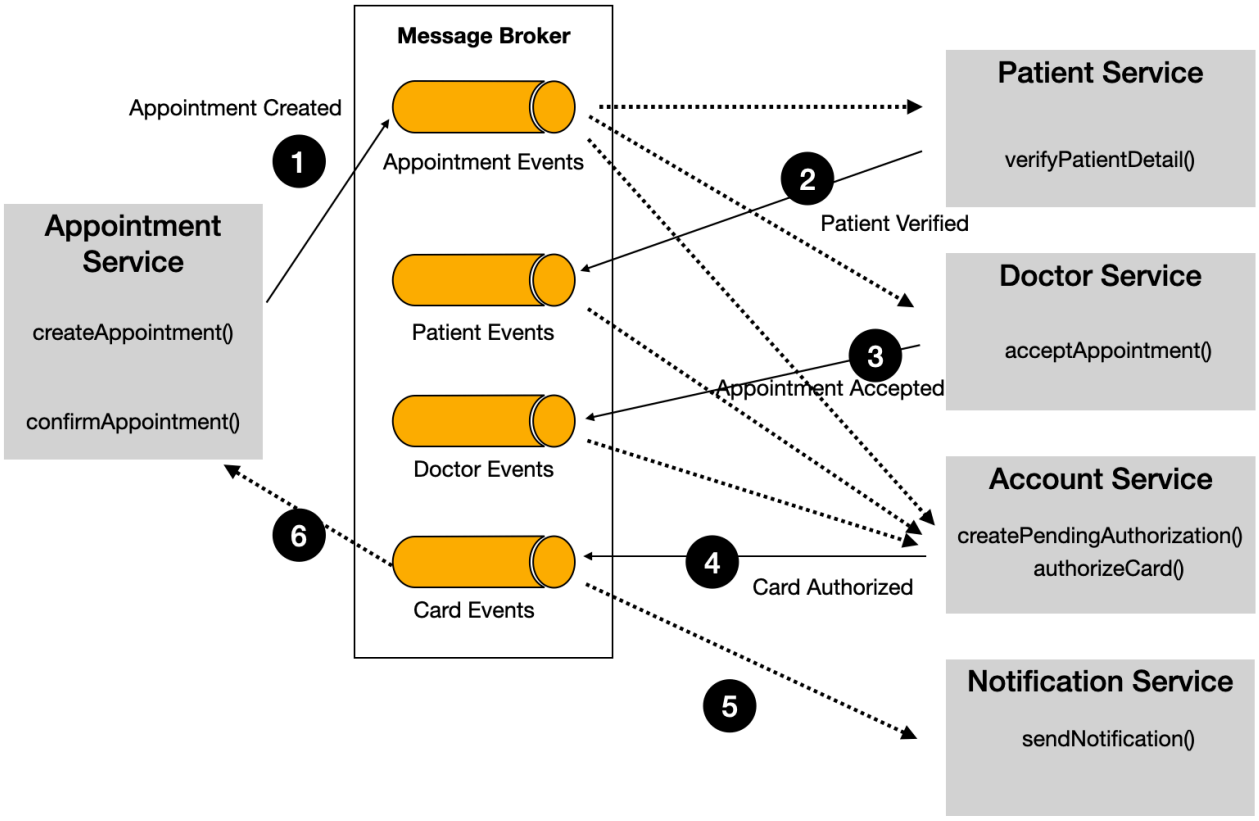
Service	Operations	Collaborators
Patient Service	VerifyPatientDetails()	NA
Appointment Service	createAppointment() acceptAppointment()	Patient Service - VerifyPatientDetails DoctorService - lookupDoctors() - verifySlots() Notification Service - sendNotification() Accounting Service - authorizeCard()
Doctor Service	verifySlot() lookupDoctors()	Appointment Service - acceptAppointment()
Accounting Service	authorizeCard()	NA
Notification Service	sendNotifcation()	NA

Scalable Architecture



1. All API calls from external client comes to API Gateway.
2. API Gateway does authentication, and call different services.
3. All Services has a Load Balancer to ensure High Availability.
4. All services follow Microservice architecture.
5. Services are deployed as docker container and managed by Kubernetes.
6. Each service has its own data base.
7. Based on the use case, services either call other services synchronously or use a Message Broker (Kafka, Rabbit MQ) to pass the message asynchronously.

Appointment Creation Flow



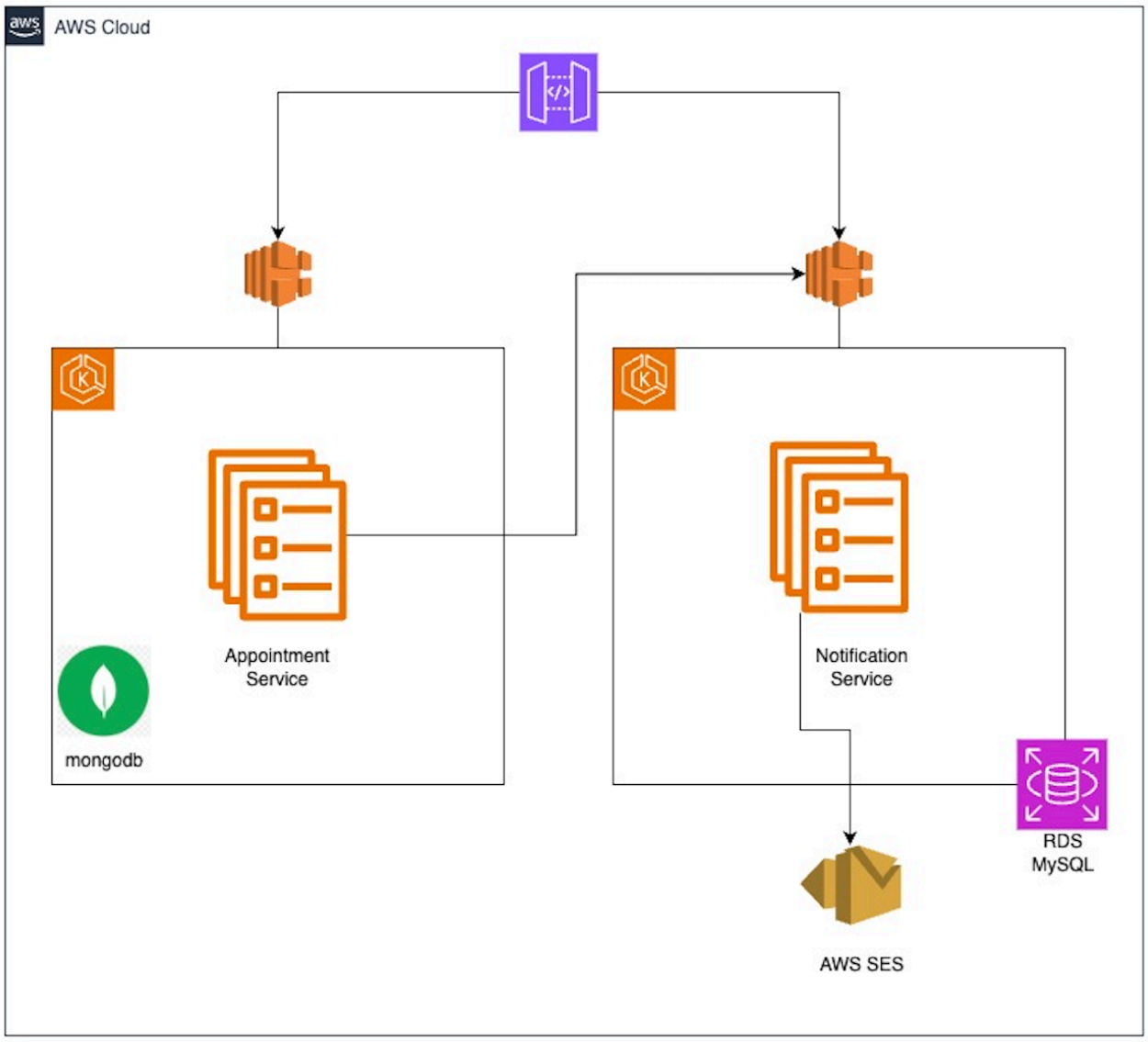
PART 2: IMPLEMENTATION

As part of Implementation, two services are implemented as detailed below.

1) Appointment Service

2) Notification Service.

High Level Architecture is as below:



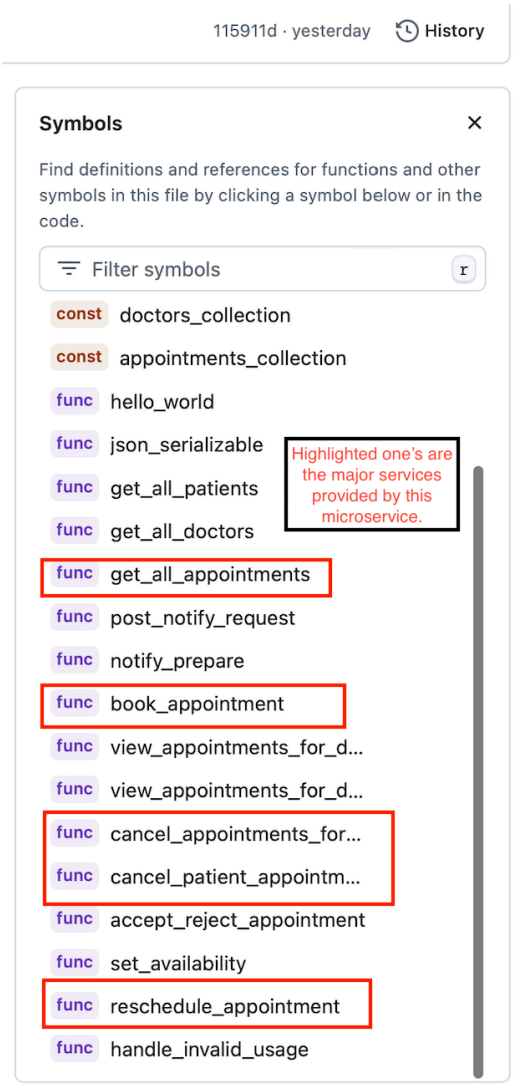
2.1 Appointment Service

Purpose of this micro-service is to provide simple appointment management service for patients to book, reschedule or cancel the Doctor's appointment. It also allows Doctors to cancel all the booked appointment on a given day.

Technology Stack used:

1. Python - Implementation of Appointment Management Service
2. Flask - Creates robust and scalable APIs
3. Mongo DB - Stores the Patient, Doctor and Appointment data

Note: As we have to implement only two of these services from our architecture, to show the complete end to end use-case we have few helper API's, that are also part of this micro-service itself like **get_all_doctors, get_all_patients etc**. Ideally they should have been implemented and exposed by patient and doctor micro service.



•This micro-service provides these major services (highlighted in left figure) to doctors and patients. Snapshot for each of these API;s invocation is added below.

•The micro service also interacts using API's with the Notification Service (another micro-service) to send a notification to the patient.

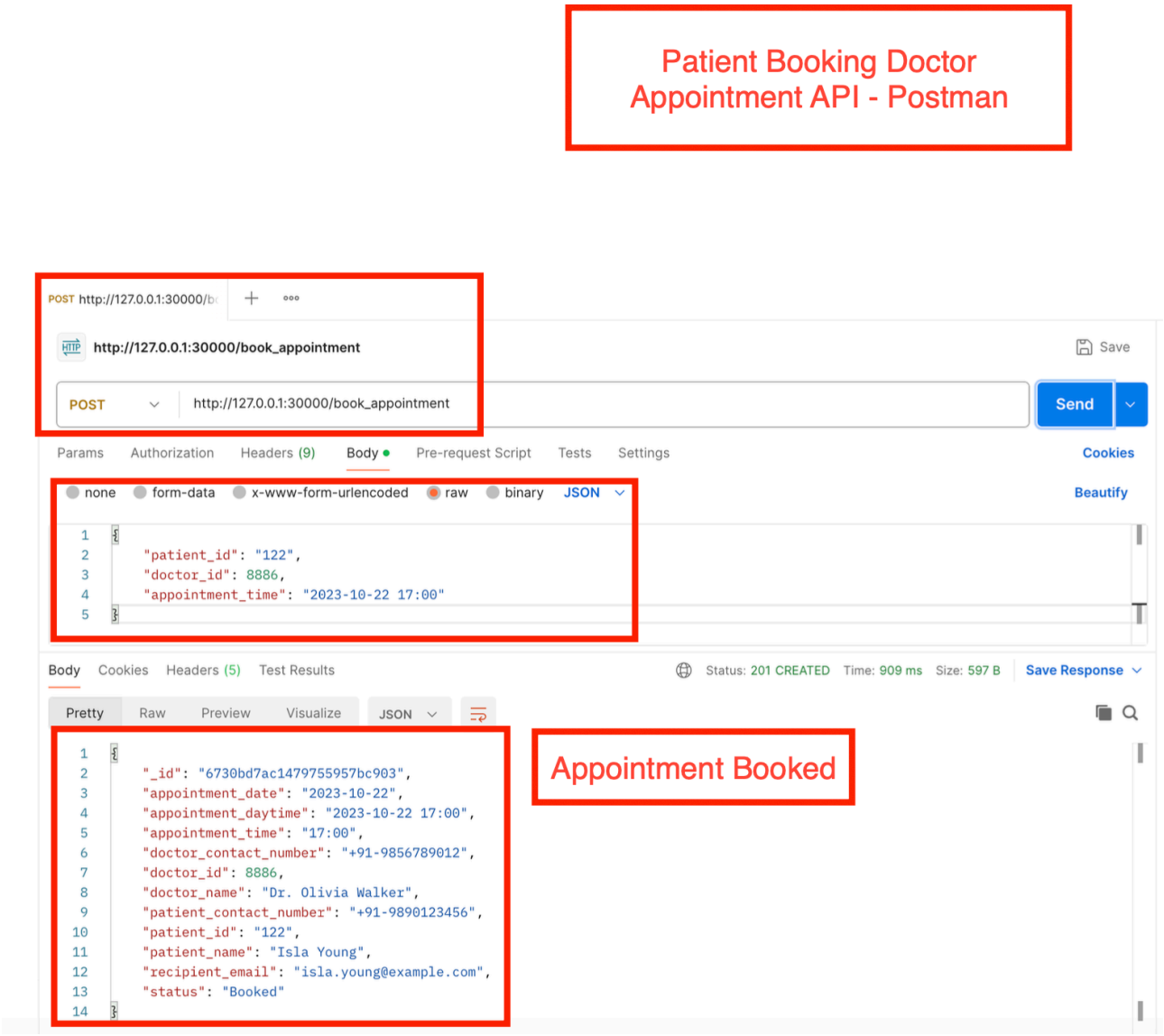
•All data is maintained inside a mongoDB database (appointment_system). Three collections are maintained inside the database. A) patients B) doctors C) appointments.

• Python based flask webApp framework is used to implement all these API endpoints (app.py)

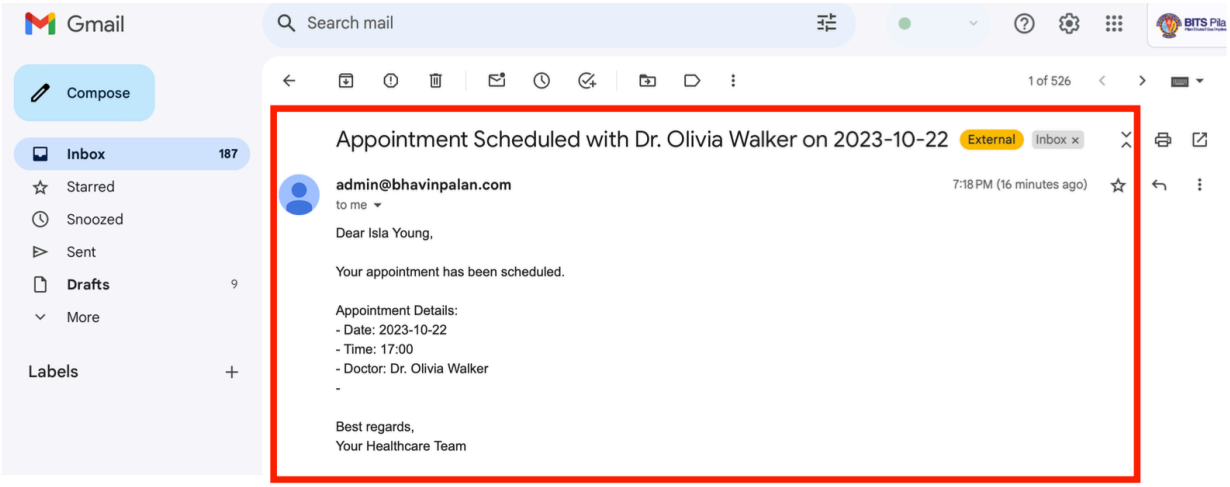
• pymongo Library is used to connect the application instance to mongodb instance.

•Github

A) API (/book_appointment) for patients to book an appointment with doctor. Once the appointment is booked successfully, the patient will receive a notification on his/her registered email**.**



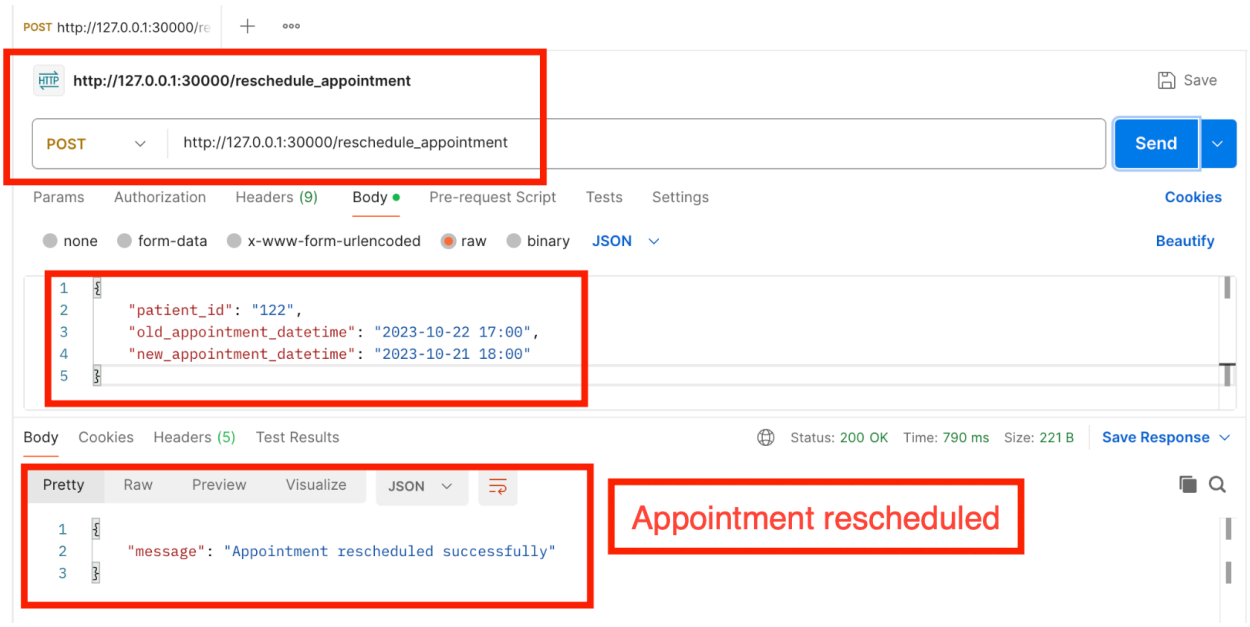
Appointment Booked Notification @Gmail via Notification Service



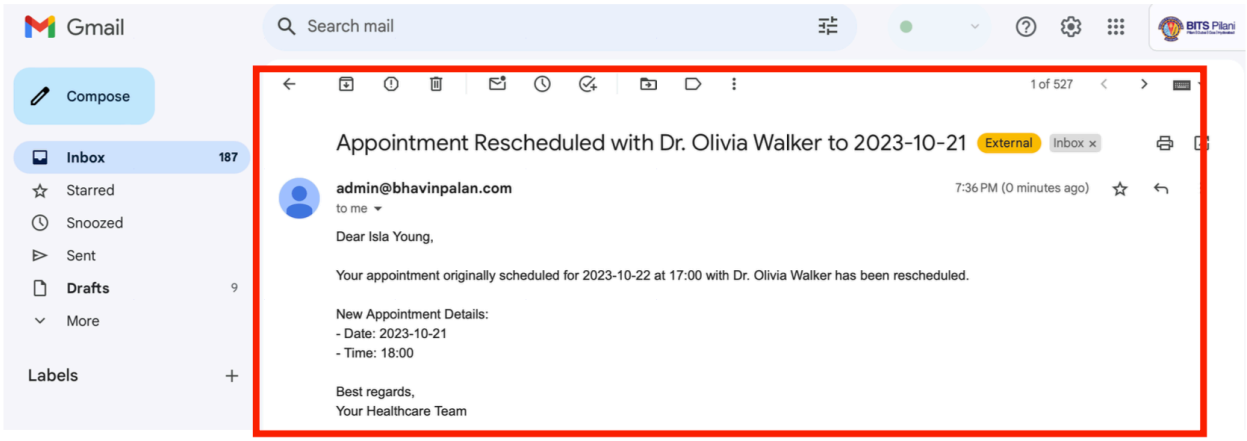
Appointment Scheduled Notification is received to patient email

B) API (/reschedule_appointment) for patients to reschedule a booked appointment with doctor. Once the appointment is rescheduled successfully, the patient will receive a reschedule notification on his/her registered email.

Patient Reschedule Doctor Appointment API - Postman

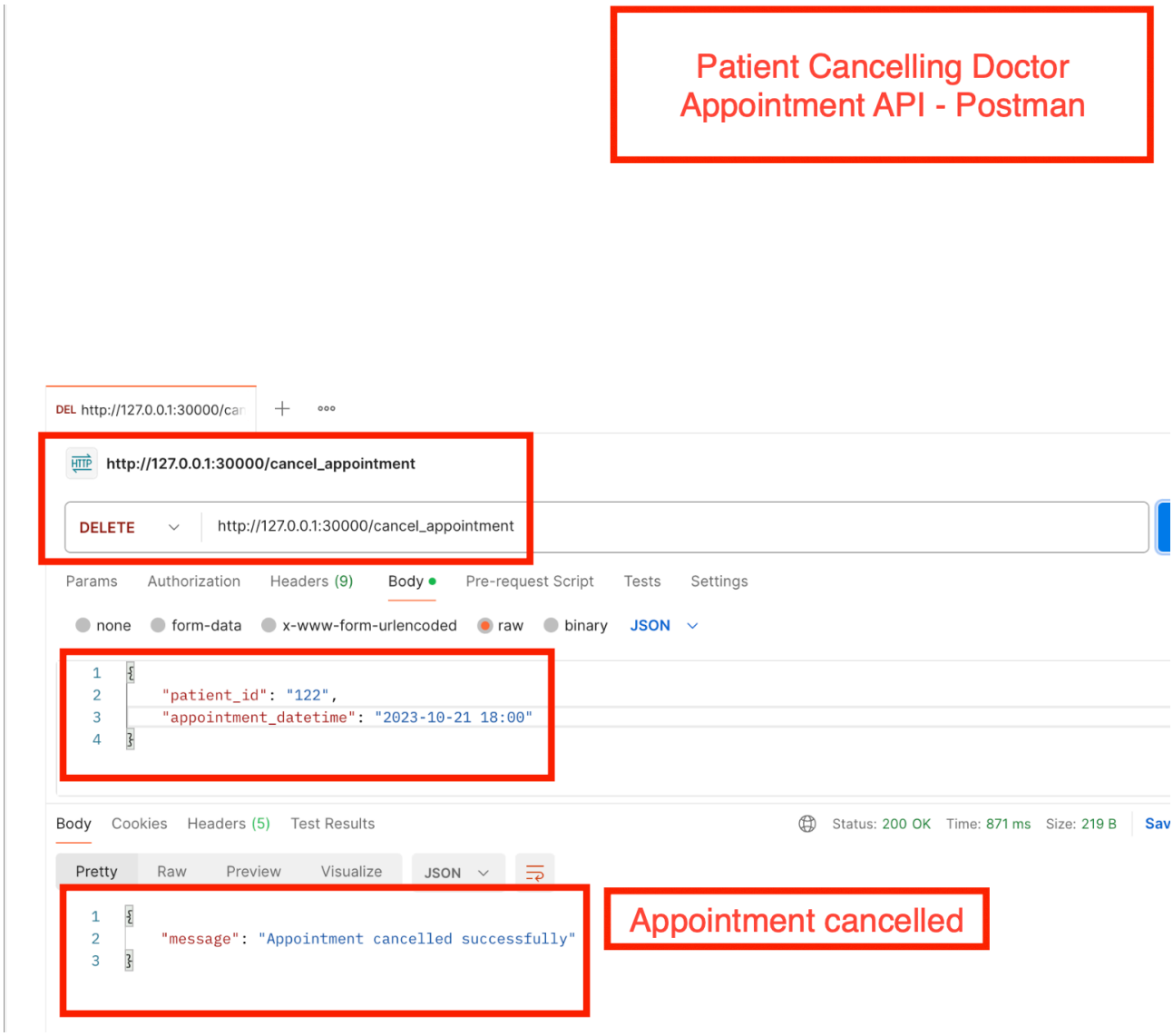


Appointment Rescheduled Notification @Gmail
via Notification Service

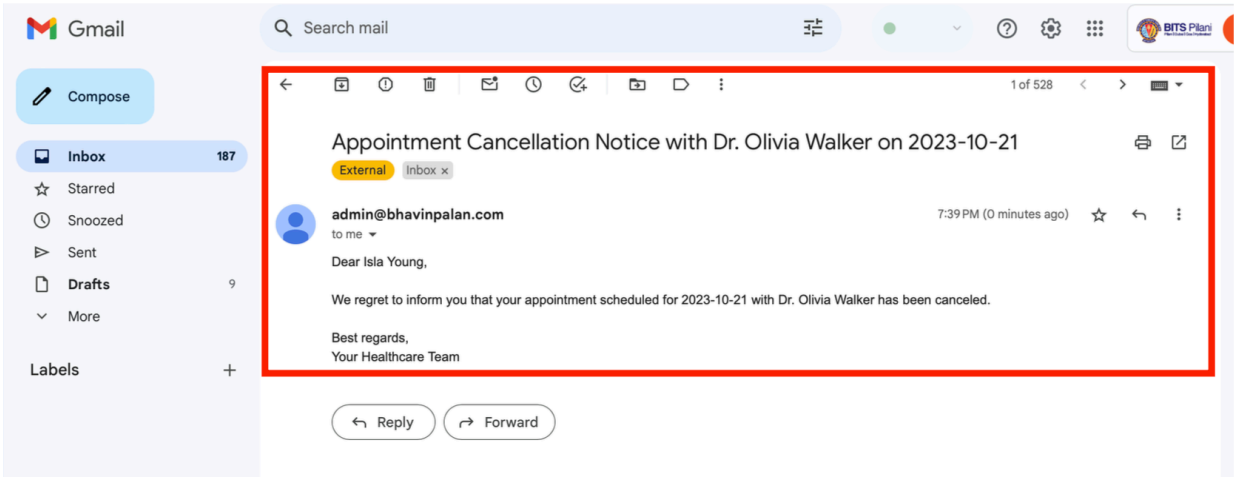


Appointment Re-scheduled Notification is received to patient email

C) API(/cancel_appointment) for patients to cancel an appointment with doctor. Once the appointment is cancelled successfully, the patient will receive a notification on his/her registered email**.**

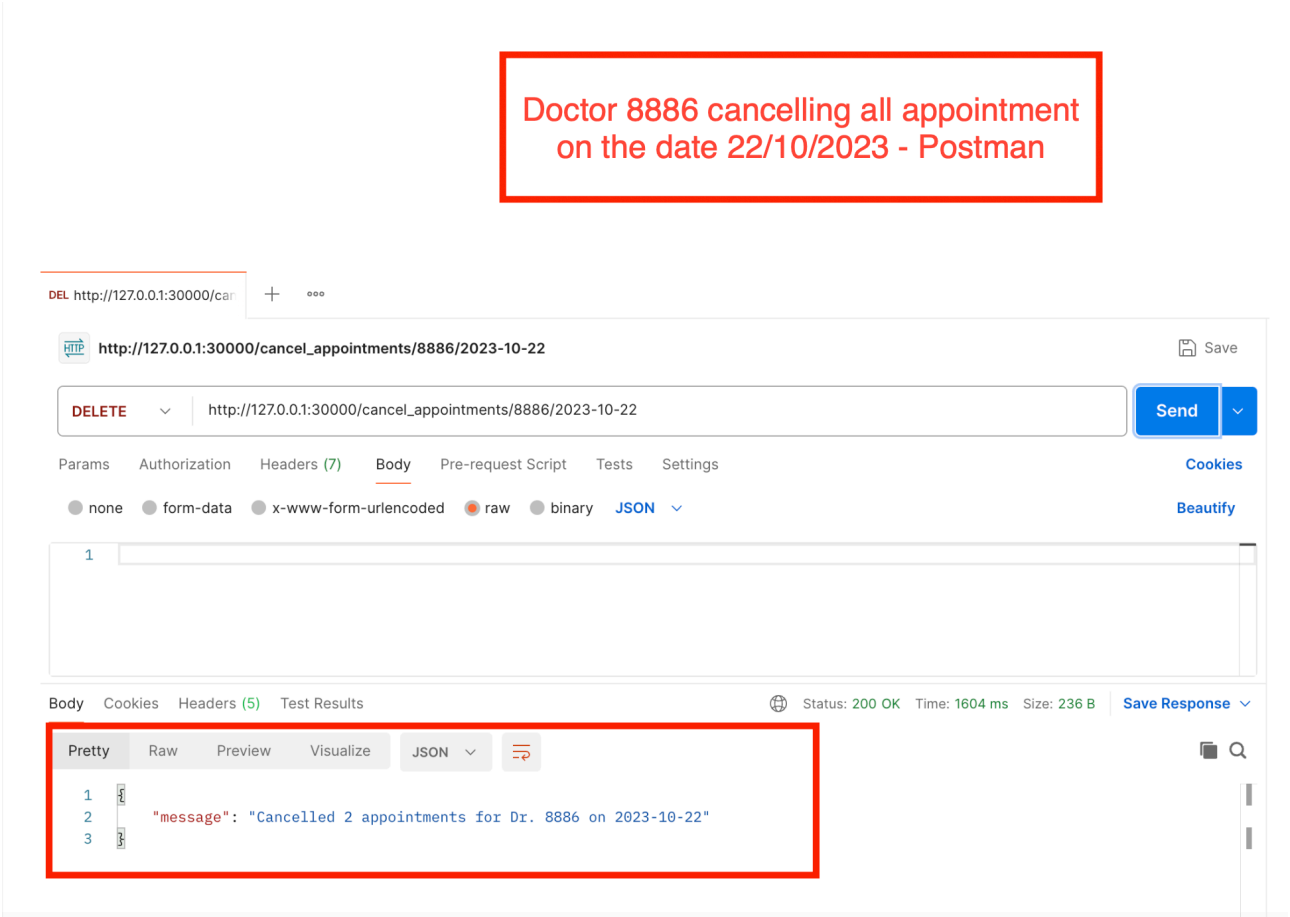


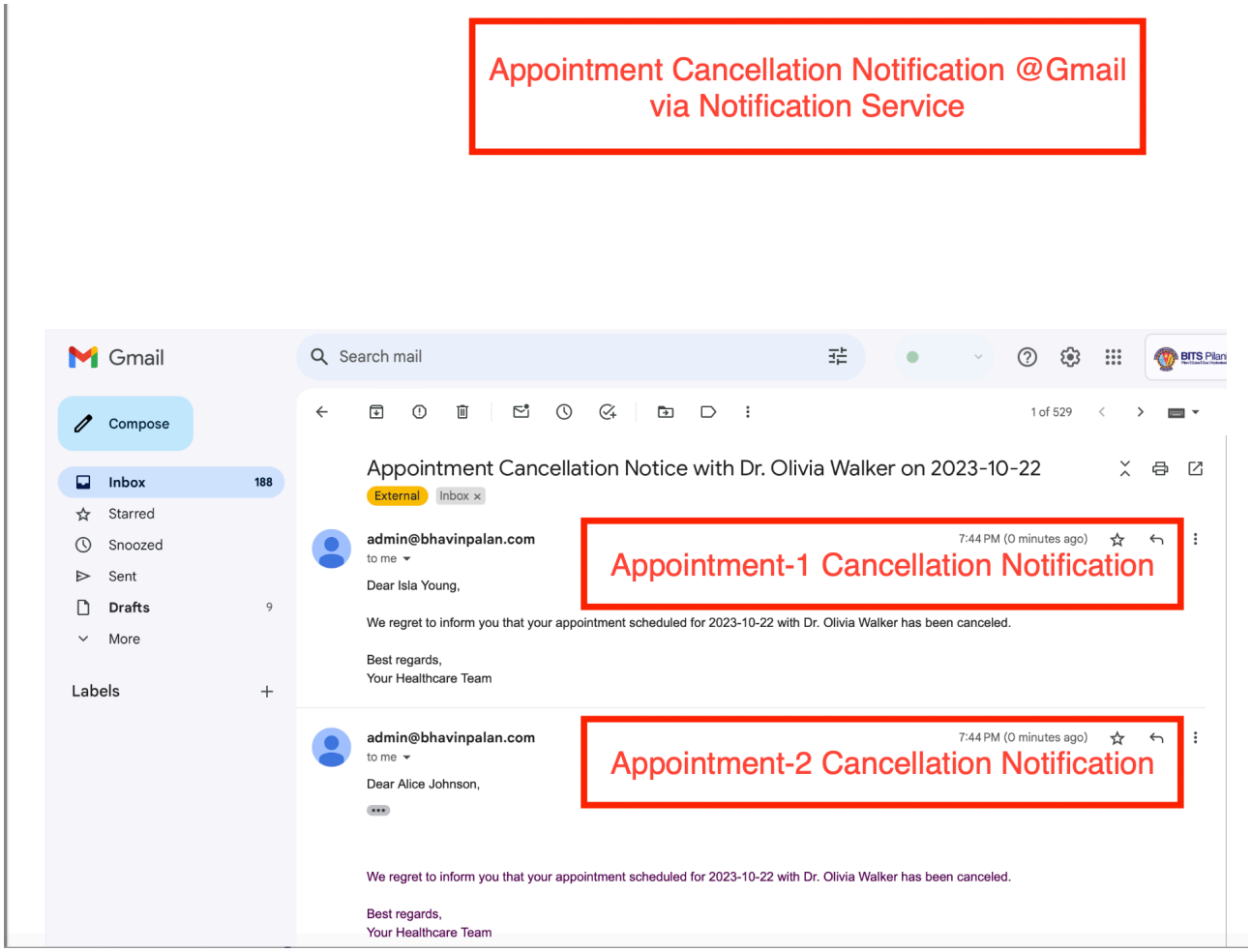
Appointment Cancellation Notification @Gmail via Notification Service



Appointment Cancellation Notification is received to patient email

D) API(/cancel_appointments/{doctor_id}/{date}) for doctor to cancel all appointments on a given day. Once the appointments are cancelled successfully, the respective patients will receive a notification on his/her registered email.





All appointments are Cancelled on the given date, and Notification is received to the respective patients email.

2.2 Notification Service

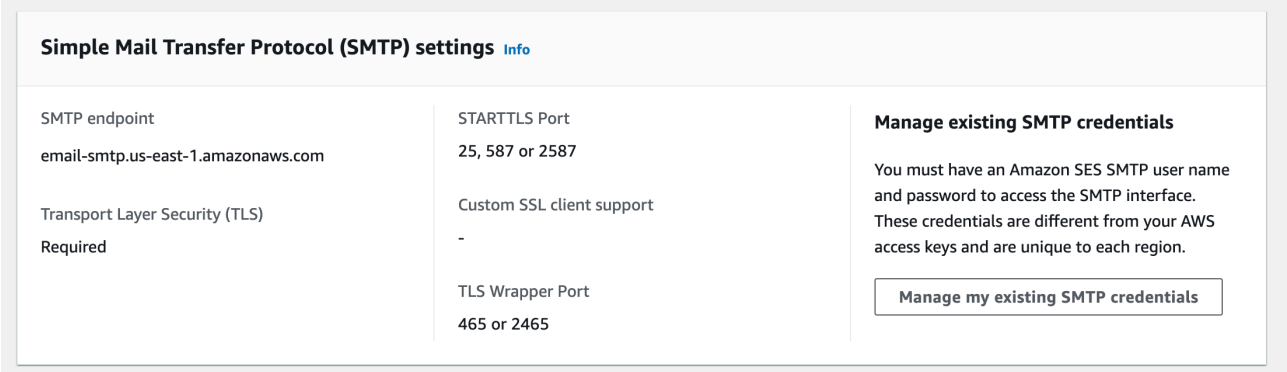
Purpose of this service is to trigger an email alert to the Patient based on appointment updates (New, Cancellation or Reschedules)

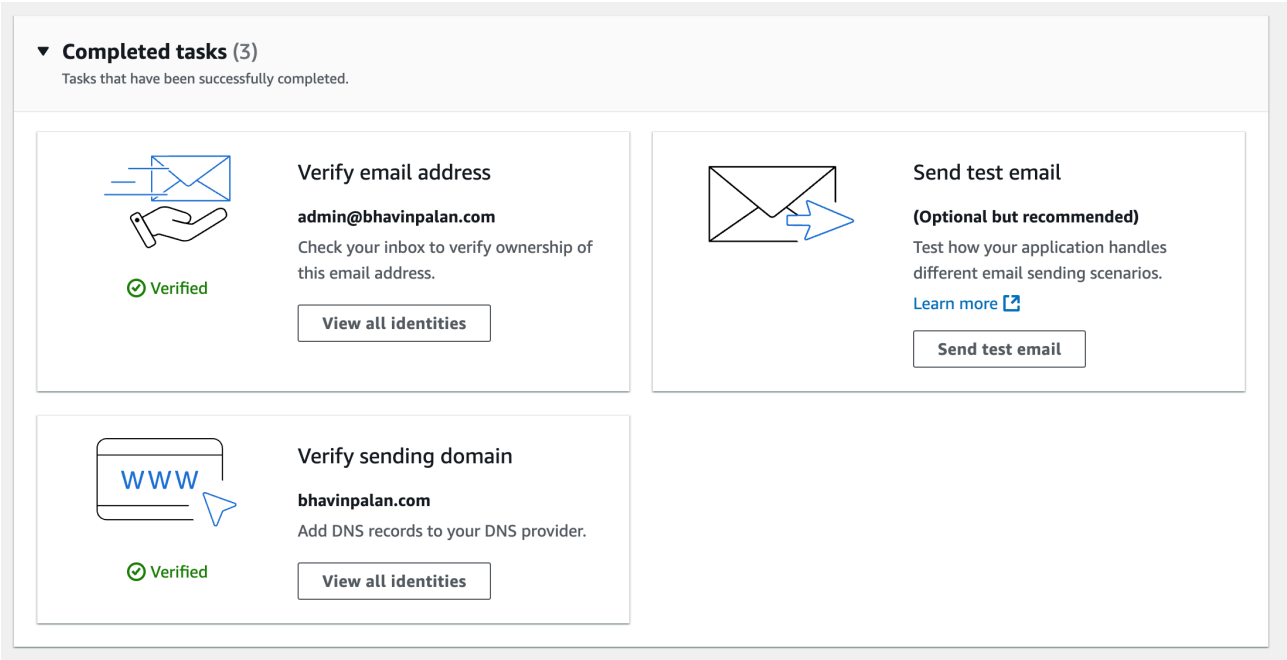
Technology Stack used:

1. AWS SES - For Email Notifications
2. AWS RDS - Template Store for different kind of emails
3. AWS EKS - Service Deployment with URL Endpoint to ELB to be called by Appointment Service.

AWS SES Implementation

It is a two step process where we first create an SES account and then verify the domain and email ownership. Once it is done, it is available for usage.



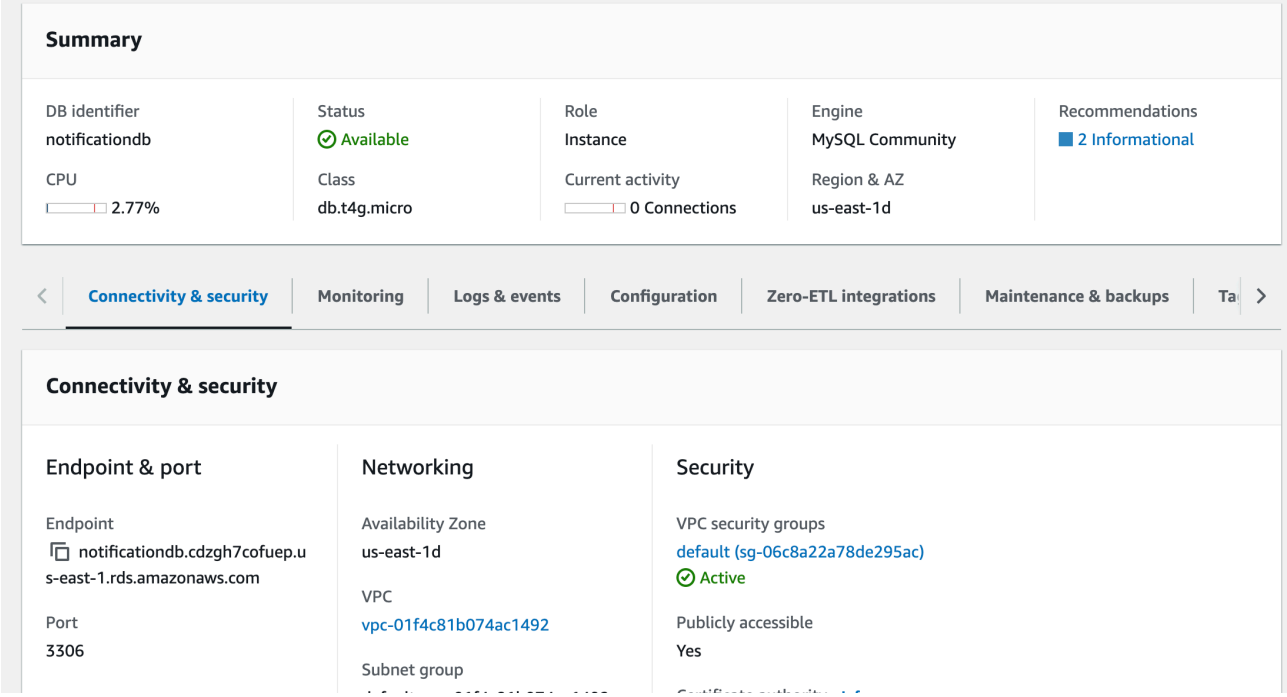


AWS RDS Implementation

Overall Process is to create an RDS database instance first and then maintain the Templates for each Appointment requirements.

1. Snapshot for RDS Instance

2. Database Update with Email Template Code



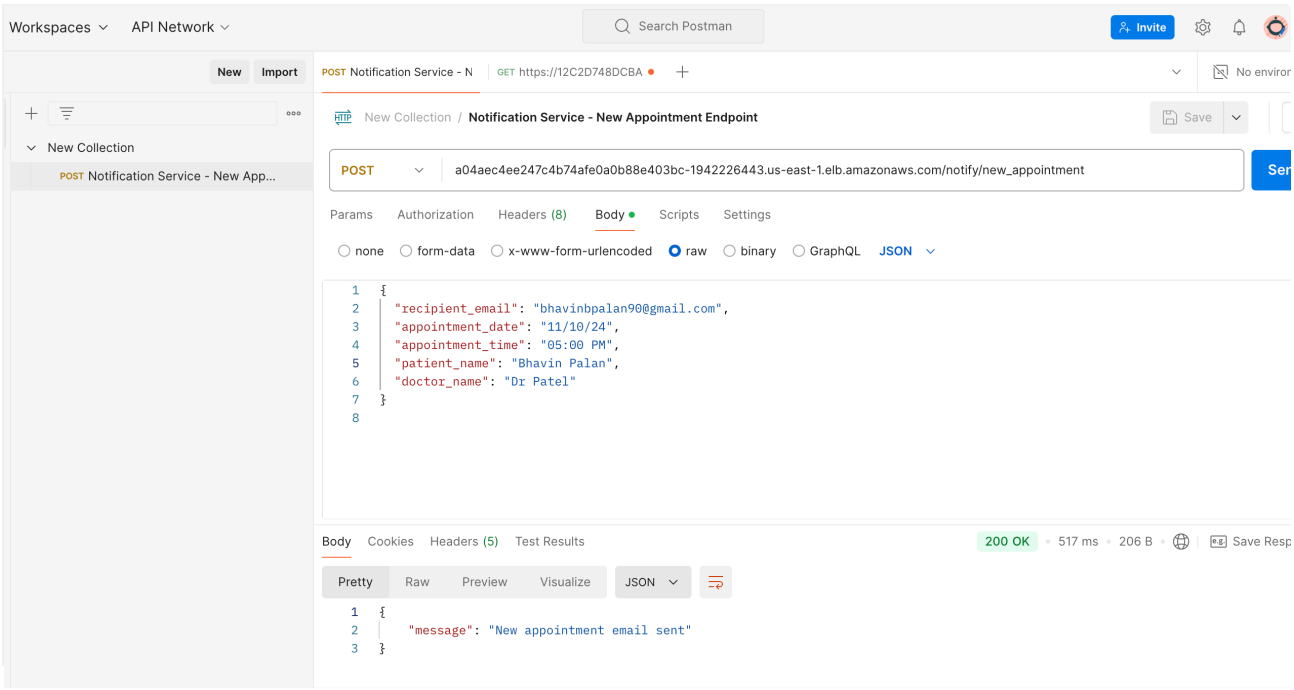
Data is updated in the table of RDS instance through python code where we are connecting to RDS instance, creating table in the database and Inserting templates with placeholders in it.

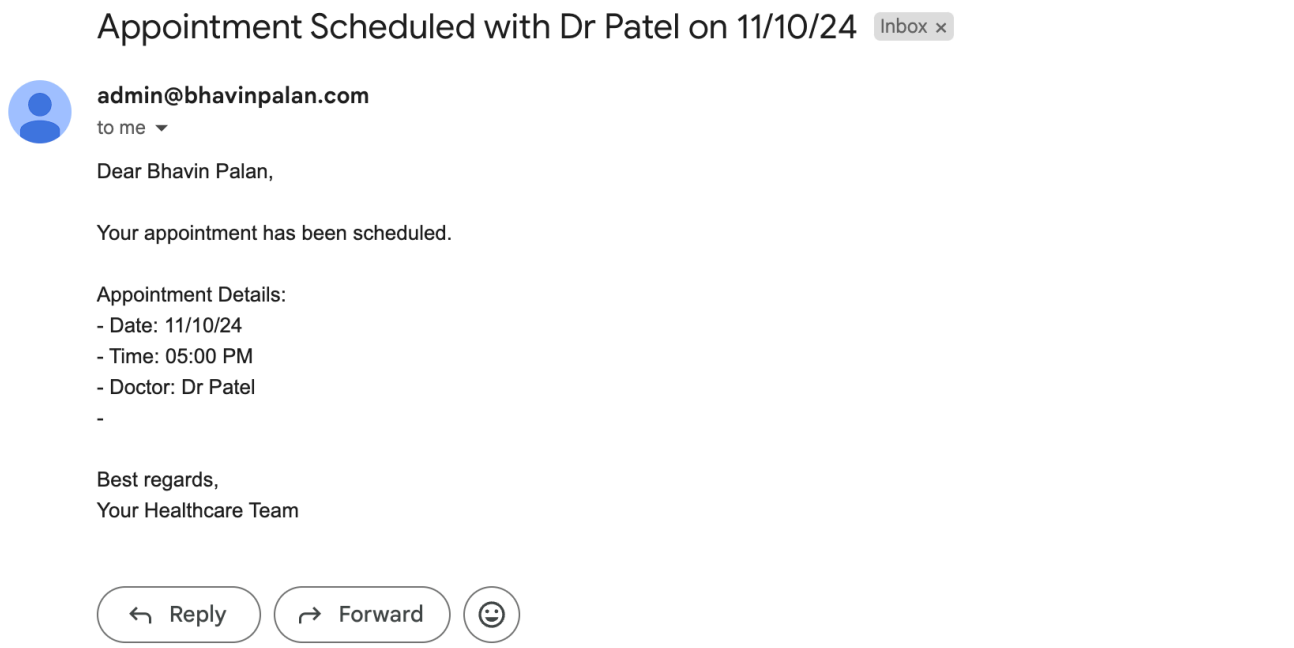
AWS EKS Implementation

Although the code is deployable in MiniKube, it is also deployed in AWS EKS Cluster for learning purpose.

Invoking of the Notification Service Example

Below is the sample Email Notification where new_appointment endpoint is called with a details of appointment and Email is sent successfully.


Workspaces
API Network
Search Postman
Invite
New
Import
POST Notification Service - N
GET https://12C2D748DCBA
No enviror
New Collection
POST Notification Service - New App...
New Collection / Notification Service - New Appointment Endpoint
Save
POST
a04aec4ee247c4b74afe0a0b88e403bc-1942226443.us-east-1.elb.amazonaws.com/notify/new_appointment
Params
Authorization
Headers (8)
Body
Scripts
Settings
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
{
"recipient_email": "bhavinbpalan90@gmail.com",
"appointment_date": "11/10/24",
"appointment_time": "05:00 PM",
"patient_name": "Bhavin Palan",
"doctor_name": "Dr Patel"
}
Body
Cookies
Headers (5)
Test Results
200 OK
517 ms
206 B
Save Resp
Pretty
Raw
Preview
Visualize
JSON
{
"message": "New appointment email sent"
}


Appointment Scheduled with Dr Patel on 11/10/24
Inbox ×
admin@bhavinpalan.com
to me
Dear Bhavin Palan,
Your appointment has been scheduled.
Appointment Details:
- Date: 11/10/24
- Time: 05:00 PM
- Doctor: Dr Patel
-
Best regards,
Your Healthcare Team
Reply
Forward

PART 3: DEPLOYMENT

3.1 Deploy each service on separate docker containers.

I. Created two deployment file 2 deploy the services in 2 separate containers.

Appointment Service with Mongo DB Container.

appointment-svc-deployment.yml

~/Library/Mobile Documents/com~apple.../.../.../.../Kubernetes/appointment-svc-deployment....

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: appointment-service
spec:
  replicas: 1
  selector:
    matchLabels:
      app: appointment-service
  template:
    metadata:
      labels:
        app: appointment-service
    spec:
      containers:
        # Main application container
        - name: appointment-service
          image: bits2023mt03164/appointment_management:v1
          ports:
            - containerPort: 6000

        # MongoDB container
        - name: mongodb
          image: mongo:latest  # Official MongoDB image
          ports:
            - containerPort: 27017  # MongoDB default port
          volumeMounts:
            - name: mongo-data
              mountPath: /data/db  # MongoDB data directory

      volumes:
        - name: mongo-data
          emptyDir: {}  # Use emptyDir volume for shared storage
---
apiVersion: v1
kind: Service
metadata:
  name: appointment-service
spec:
  type: NodePort
  selector:
    app: appointment-service
  ports:
    - port: 6000
      targetPort: 6000
      nodePort: 30000
---
```

Notification Service

notification-service-deployment.yml

~/Library/Mobile Documents/com~ap.../.../.../.../Kubernetes/notification-service-

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: notification-service
spec:
  replicas: 1
  selector:
    matchLabels:
      app: notification-service
  template:
    metadata:
      labels:
        app: notification-service
    spec:
      containers:
      - name: notification-service
        image: bhavinbpalan90/notificationservice
        ports:
        - containerPort: 8081
---
apiVersion: v1
kind: Service
metadata:
  name: notification-service
spec:
  type: NodePort
  selector:
    app: notification-service
  ports:
    - port: 80
      targetPort: 8081
      nodePort: 30002
```

3.2. Deploy your application to the Kubernetes cluster.

kubectl apply -f appointment-svc-deployment.yml

```
gauravsinghal@Gauravs-Air-1 Kubernetes % kubectl apply -f appointment-svc-deployment.yml
deployment.apps/appointment-service created
service/appointment-service created
service/mongodb created
```

kubectl apply -f notification-service-deployment.yml

```
gauravsinghal@Gauravs-Air-1 Kubernetes % kubectl apply -f notification-service-deployment.yml
deployment.apps/notification-service created
service/notification-service created
```

```
gauravsinghal@Gauravs-Air-1 Kubernetes % kubectl get all
NAME                                        READY   STATUS    RESTARTS   AGE
pod/appointment-service-56d49fcf4c-fgxjq    2/2     Running   0          112m
pod/notification-service-57f5556ff7-qgjw9   1/1     Running   0          113s

NAME                           TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)           AGE
service/appointment-service    NodePort    10.103.203.243   <none>        6000:30000/TCP    112m
service/kubernetes             ClusterIP   10.96.0.1        <none>        443/TCP           113m
service/mongodb                NodePort    10.111.53.216    <none>        27017:30001/TCP   112m
service/notification-service   NodePort    10.110.113.181   <none>        80:30002/TCP      113s

NAME                                   READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/appointment-service    1/1     1            1           112m
deployment.apps/notification-service   1/1     1            1           113s

NAME                                              DESIRED   CURRENT   READY   AGE
replicaset.apps/appointment-service-56d49fcf4c    1         1         1       112m
replicaset.apps/notification-service-57f5556ff7   1         1         1       113s
gauravsinghal@Gauravs-Air-1 Kubernetes %
```

Both Services running on separate pods

3.3. Run a minikube cluster on your local machine and deploy your application on the cluster

```
gauravsinghal@Gauravs-Air-1 Kubernetes % minikube start --driver=docker --ports=127.0.0.1:30000:30000
😄  minikube v1.34.0 on Darwin 14.4.1 (arm64)
✨  Using the docker driver based on user configuration
📌  Using Docker Desktop driver with root privileges
👍  Starting "minikube" primary control-plane node in "minikube" cluster
🚜  Pulling base image v0.0.45 ...
🔥  Creating docker container (CPUs=2, Memory=2200MB) ...
❗  Failing to connect to https://registry.k8s.io/ from inside the minikube container
💡  To pull new external images, you may need to configure a proxy: https://minikube.sigs.k8s.io/docs/reference/networking/proxy/
🐳  Preparing Kubernetes v1.31.0 on Docker 27.2.0 ...
    ▪ Generating certificates and keys ...
    ▪ Booting up control plane ...
    ▪ Configuring RBAC rules ...
🔗  Configuring bridge CNI (Container Networking Interface) ...
🔎  Verifying Kubernetes components...
    ▪ Using image gcr.io/k8s-minikube/storage-provisioner:v5
🌟  Enabled addons: default-storageclass, storage-provisioner
🏄  Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
```

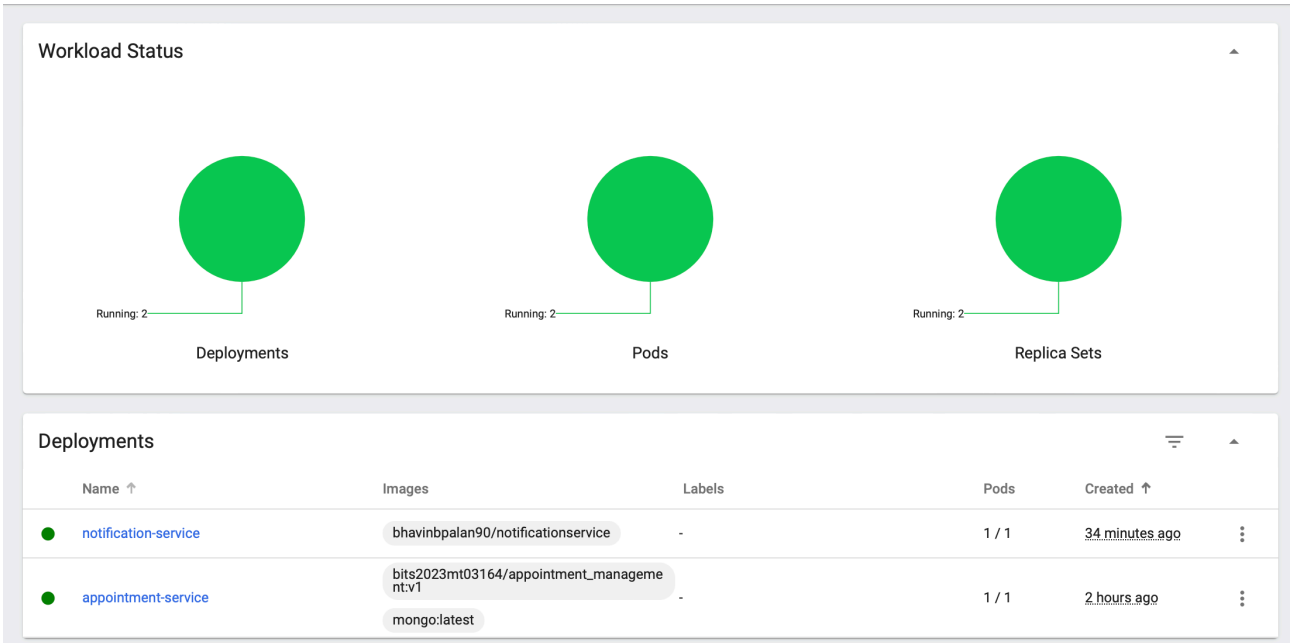
```
gauravsinghal@Gauravs-Air-1 Kubernetes % kubectl version
Client Version: v1.31.2
Kustomize Version: v5.4.2
Server Version: v1.31.0
gauravsinghal@Gauravs-Air-1 Kubernetes % minikube status
minikube
type: Control Plane
host: Running
kubelet: Running
apiserver: Running
kubeconfig: Configured

gauravsinghal@Gauravs-Air-1 Kubernetes % minikube service list
|-------------|----------------------|---------------|-----|
|  NAMESPACE  |         NAME         |  TARGET PORT  | URL |
|-------------|----------------------|---------------|-----|
| default     | appointment-service  |          6000 |     |
| default     | kubernetes           | No node port  |     |
| default     | mongodb              |         27017 |     |
| default     | notification-service |            80 |     |
| kube-system | kube-dns             | No node port  |     |
|-------------|----------------------|---------------|-----|
gauravsinghal@Gauravs-Air-1 Kubernetes %
```

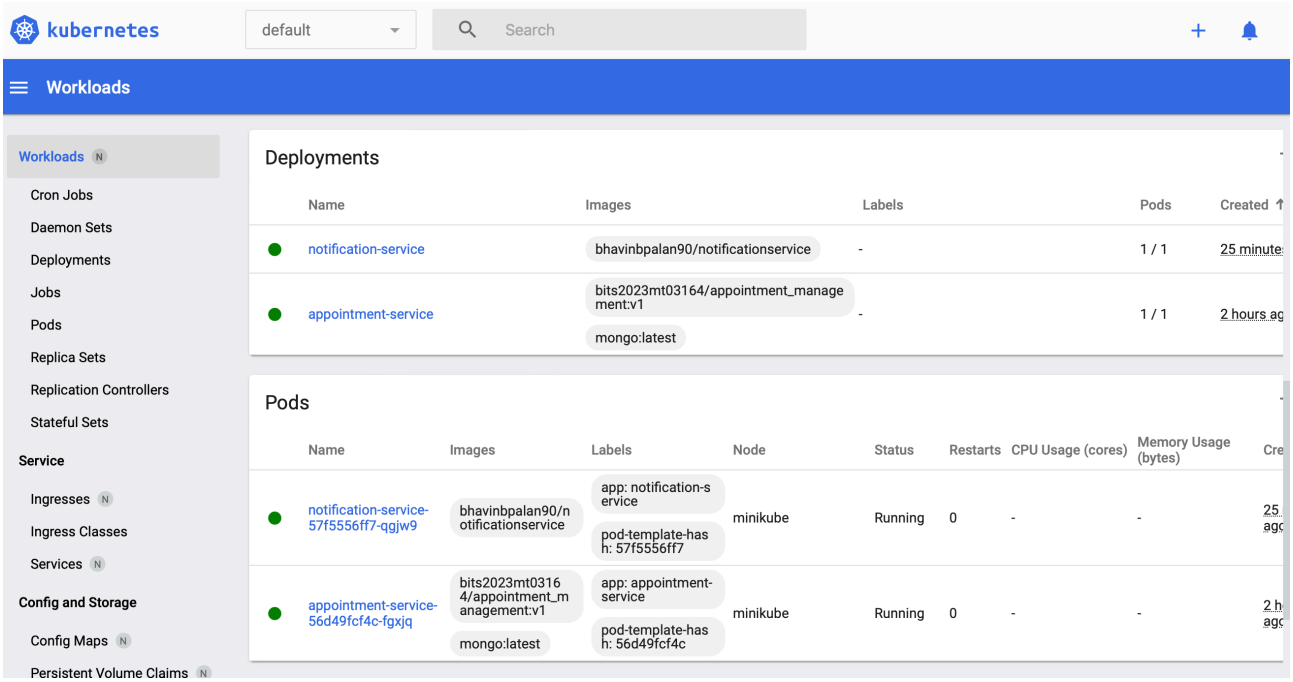
```
gauravsinghal@Gauravs-Air-1 Kubernetes % kubectl apply -f appointment-svc-deployment.yml
deployment.apps/appointment-service created
service/appointment-service created
service/mongodb created
```

```
gauravsinghal@Gauravs-Air-1 Kubernetes % kubectl apply -f notification-service-deployment.yml
deployment.apps/notification-service created
service/notification-service created
```

3.4. Access the Kubernetes dashboard and analyze the deployments made.

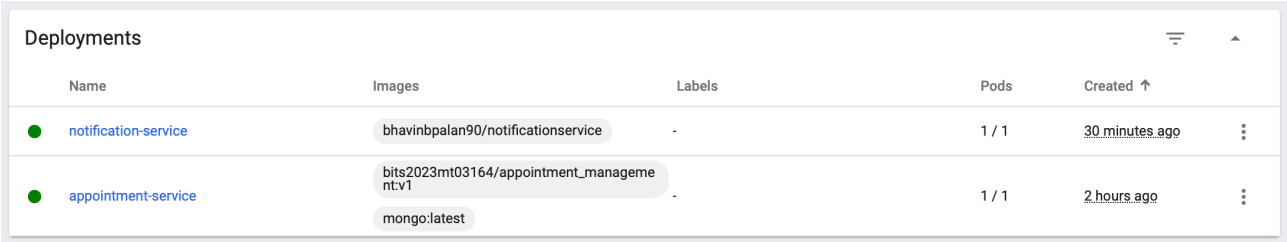
```
gauravsinghal@Gauravs-Air-1 Kubernetes % minikube dashboard
🔌  Enabling dashboard ...
    ▪ Using image docker.io/kubernetesui/dashboard:v2.7.0
    ▪ Using image docker.io/kubernetesui/metrics-scraper:v1.0.8
💡  Some dashboard features require the metrics-server addon. To enable all features please run:

        minikube addons enable metrics-server

🤔  Verifying dashboard health ...
🚀  Launching proxy ...
🤔  Verifying proxy health ...
```





Deployment Analysis :

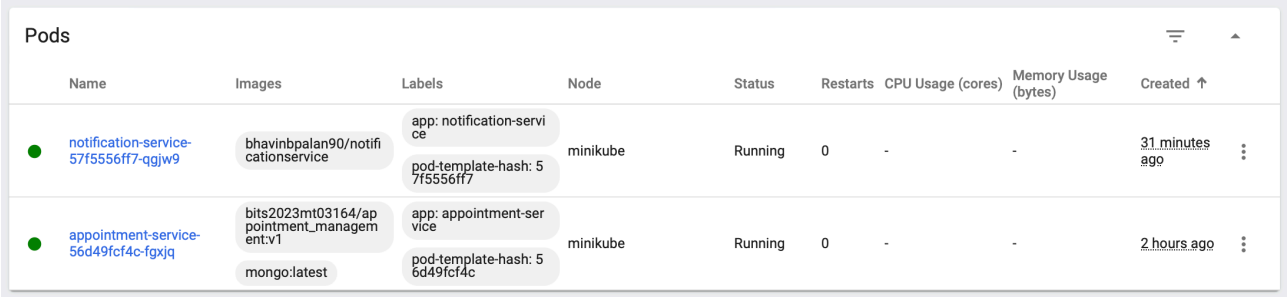
1. There are 2 deployments running. **notification-service** and **appointment-service**



Deployments

Name	Images	Labels	Pods	Created
notification-service	bhavinbpalan90/notificationservice	-	1 / 1	30 minutes ago
appointment-service	bits2023mt03164/appointment_management:v1 mongo:latest	-	1 / 1	2 hours ago

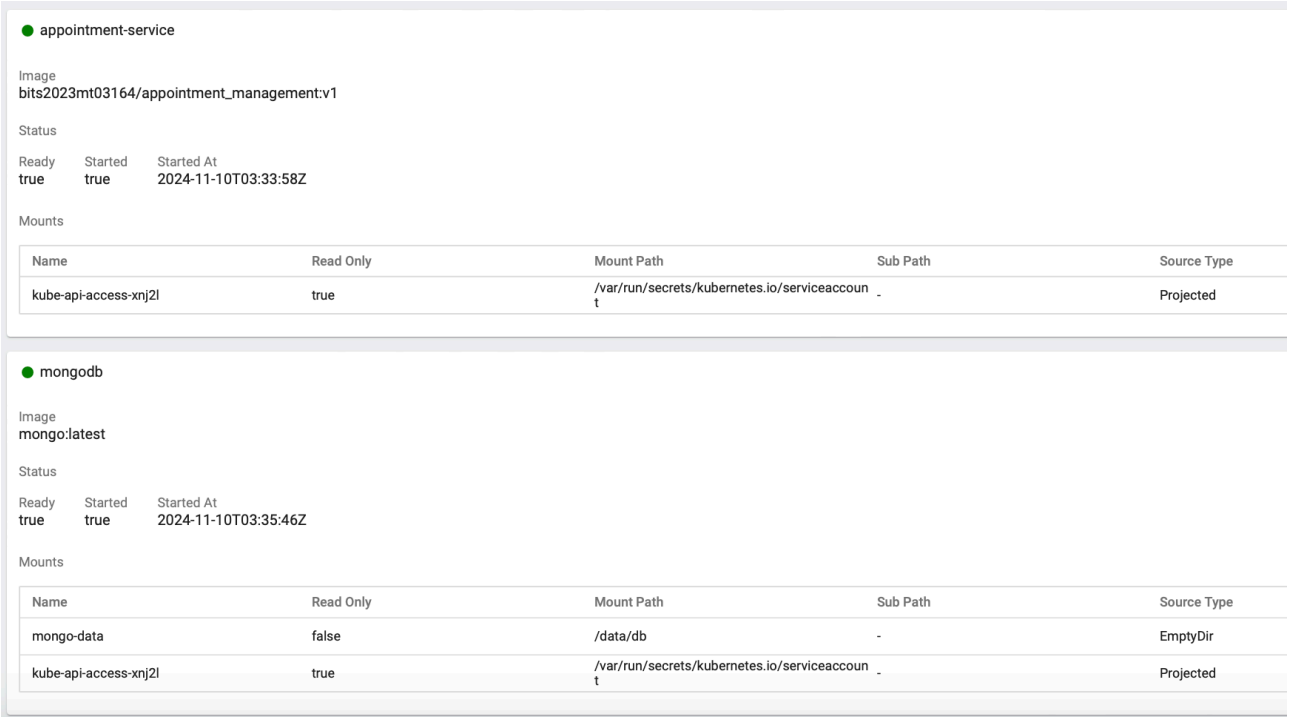
2. **There are 2 pods running, one each in each deployment**



Pods

Name	Images	Labels	Node	Status	Restarts	CPU Usage (cores)	Memory Usage (bytes)	Created
notification-service-57f5556ff7-qgjw9	bhavinbpalan90/notificationservice	app: notification-service pod-template-hash: 57f5556ff7	minikube	Running	0	-	-	31 minutes ago
appointment-service-56d49fcf4c-fgxjq	bits2023mt03164/appointment_management:v1 mongo:latest	app: appointment-service pod-template-hash: 56d49fcf4c	minikube	Running	0	-	-	2 hours ago

3. **Appointment-service has 2 containers running - one for app and another for mongo DB**



appointment-service

Image
bits2023mt03164/appointment_management:v1

Status

Ready	Started	Started At
true	true	2024-11-10T03:33:58Z

Mounts

Name	Read Only	Mount Path	Sub Path	Source Type
kube-api-access-xnj2l	true	/var/run/secrets/kubernetes.io/serviceaccount	-	Projected

mongodb

Image
mongo:latest

Status

Ready	Started	Started At
true	true	2024-11-10T03:35:46Z

Mounts

Name	Read Only	Mount Path	Sub Path	Source Type
mongo-data	false	/data/db	-	EmptyDir
kube-api-access-xnj2l	true	/var/run/secrets/kubernetes.io/serviceaccount	-	Projected

4. Notification-service pod has only one container