Name: Arun Kumar N
Assignment: GCP Assignment on DevOps

Table of Contents

Problem Statement:

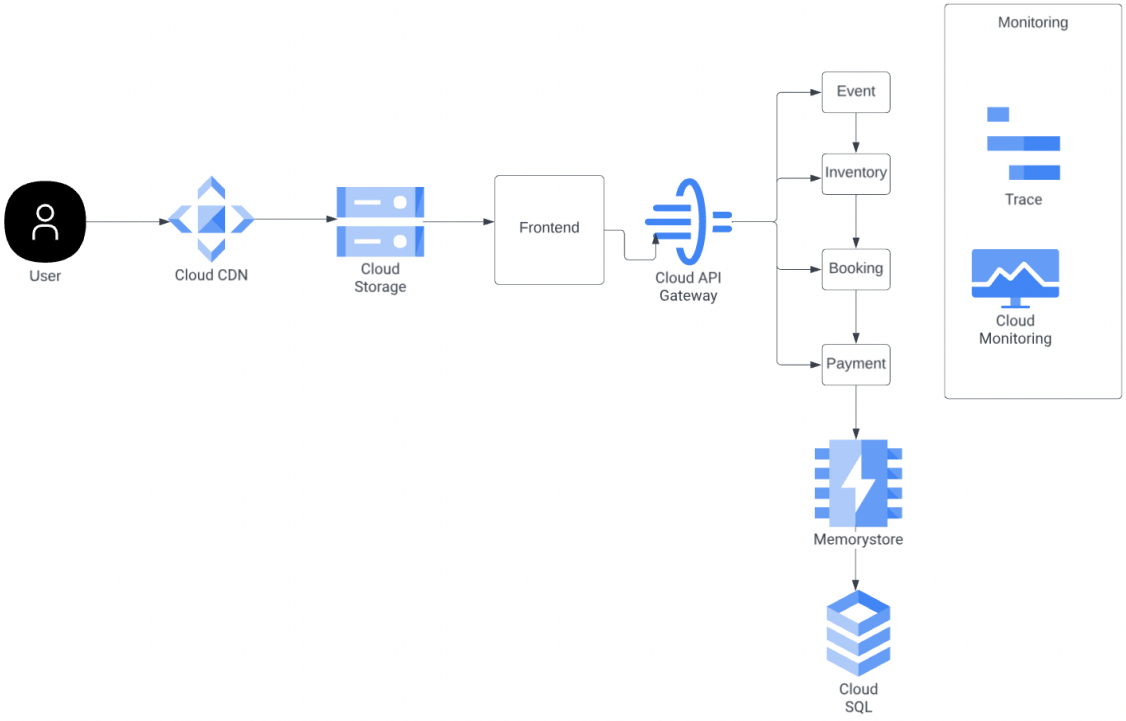
This project aims to measure and solve the performance issues faced by people with BookMyShow's ticketing system during the recent distribution of Coldplay's concert tickets in India or similar issues. It intends to improve the scalability and stability of the platform to support mass traffic requests, such as huge sales of concert tickets, along with a seamless experience for all users.

Solution Applied:

Code Link: https://github.com/Aruun/gcp-scalable-ticketing-app

Task 1: System Analysis and Architecture Review

Assuming the architecture is based on **microservices** for scalability and flexibility
Reference Architecture:



Request Flow:

- **User** interacts with the **Frontend** (e.g., your `index.html`, `script.js`).
- The **Frontend** makes HTTP/HTTPS calls (AJAX/Fetch) to backend endpoints.
- These calls go to the **API Gateway**.

- The **API Gateway**:
  - Authenticates requests
  - Routes them to the correct microservice (e.g., Event, Booking, Inventory)
  - Optionally applies rate-limiting, logging, or transformation
- Then the **Microservice** handles the business logic and queries the **database** if needed.
- The response flows back to the **Frontend** via the same path.

Comprehensive Report on bottlenecks:

Database

- High `write contention` during ticket booking
- Lack of indexes on filters (e.g., by event/city)
- Inefficient joins or unoptimized queries

API Response Time

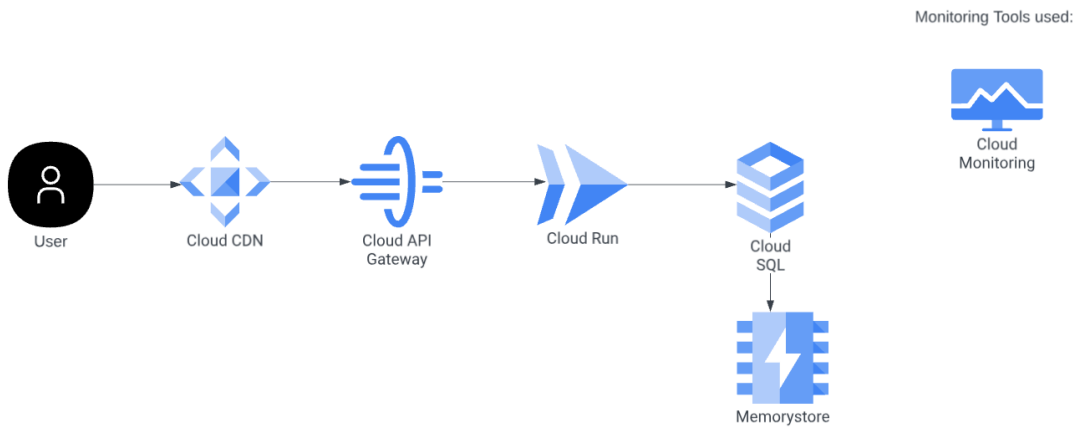
- Synchronous dependencies across services
- Cold starts in Cloud Run
- Missing caching layer (for event listings)

Load Balancing

- No auto-scaling rules → latency spikes
- Sticky sessions inappropriately used
- Not utilizing multiple regions

Task 2: Scalable Architecture Implementation

I have used CloudRun serverless approach to re-design it.



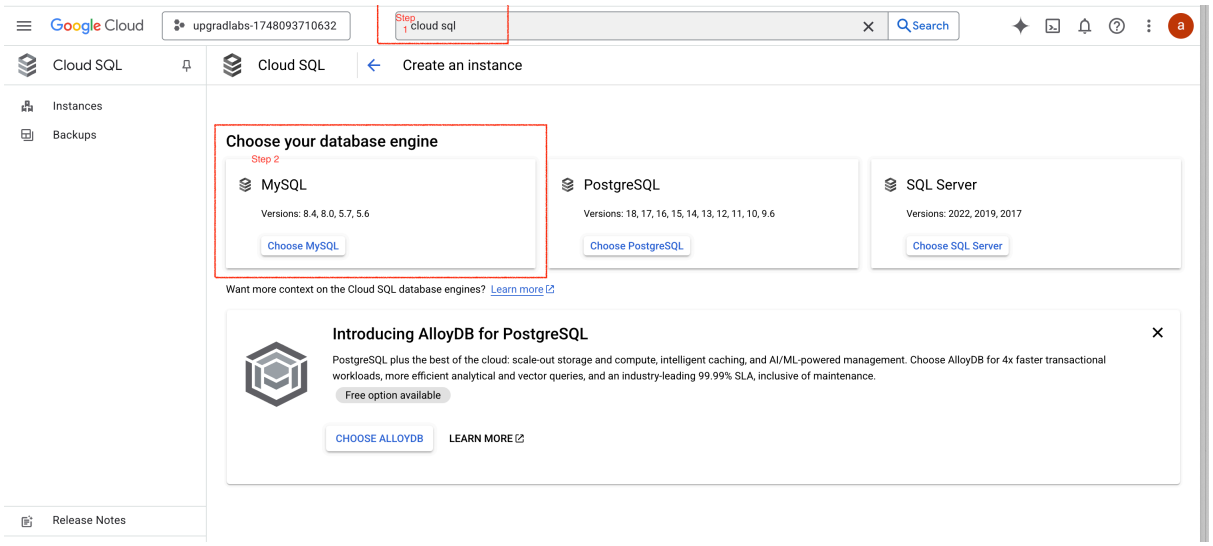
Flow:

- User visits the web or mobile app.
- Static assets (HTML/CSS/JS) are served via **Cloud CDN**
- On a user action (e.g., search events, book ticket), the browser sends an **API request**.
- Request passes through **Cloud CDN**
- If not cached, request is forwarded to **Cloud API Gateway**.
- **API Gateway**:
  - Authenticates the user
  - Routes the request to the appropriate **Cloud Run** service.
- **Cloud Run** service (e.g., EventService, BookingService):
  - Handles business logic
  - Connects to **Cloud SQL** for persistent data
  - Uses **Memorystore** for caching ticket inventory or locking mechanisms during booking
- Response flows back through:
  - Cloud Run → API Gateway → (optional CDN cache) → User

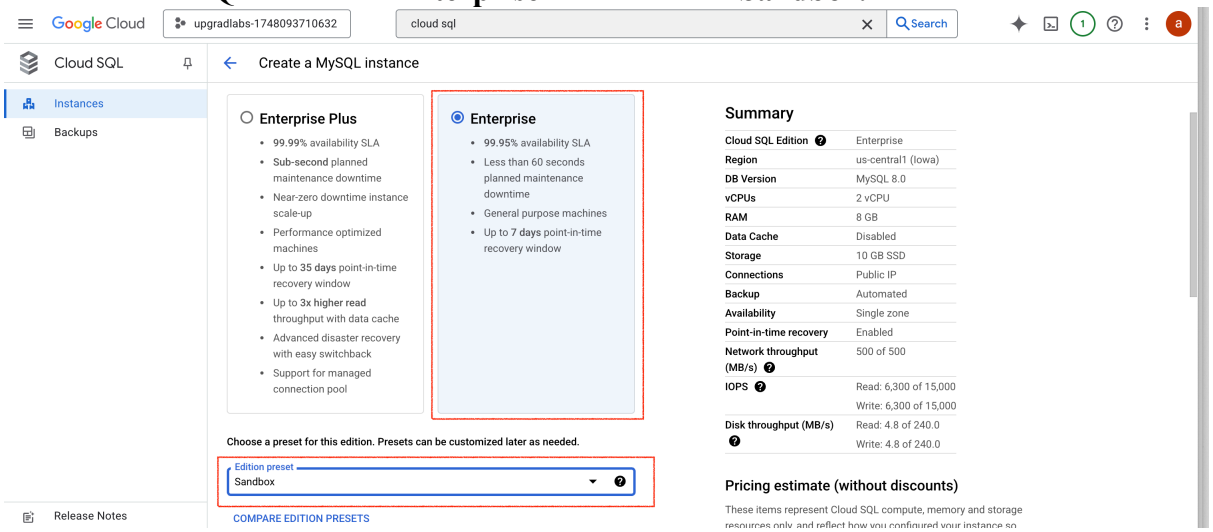
Name: Arun Kumar N
Assignment: Scaling BookMyShow using GCP

Task 3: Database Optimisation

Step 1: Navigate to GCP Cloud console, search for Cloud SQL in search bar and select MySQL as a platform type, as shown in the image below:

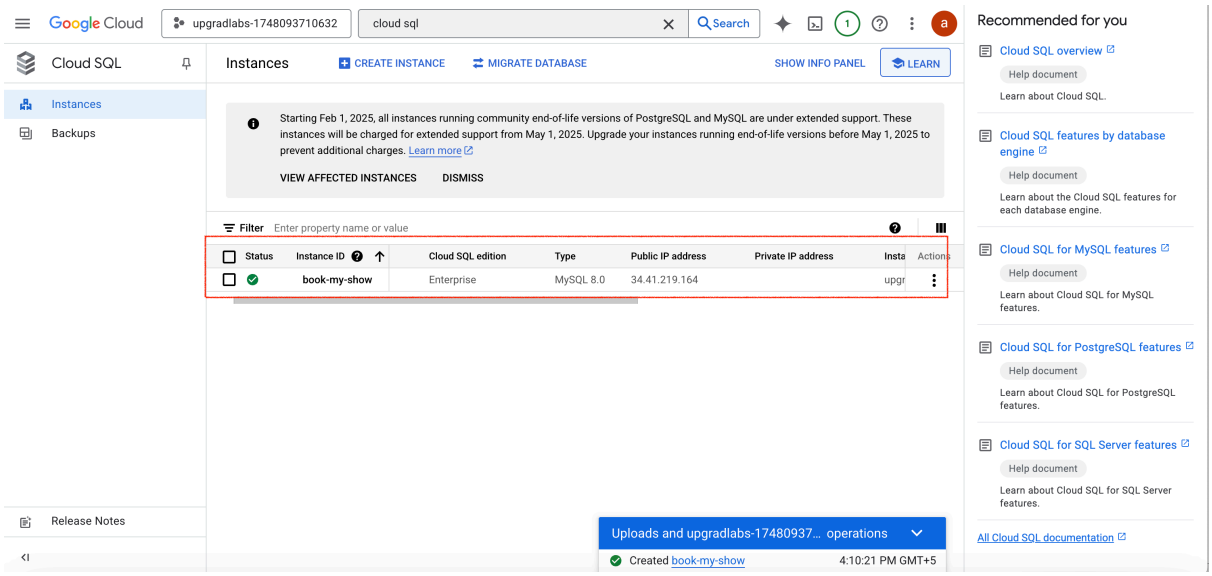


Choose Cloud SQL edition as **Enterprise** and Edition as **Sandbox.**

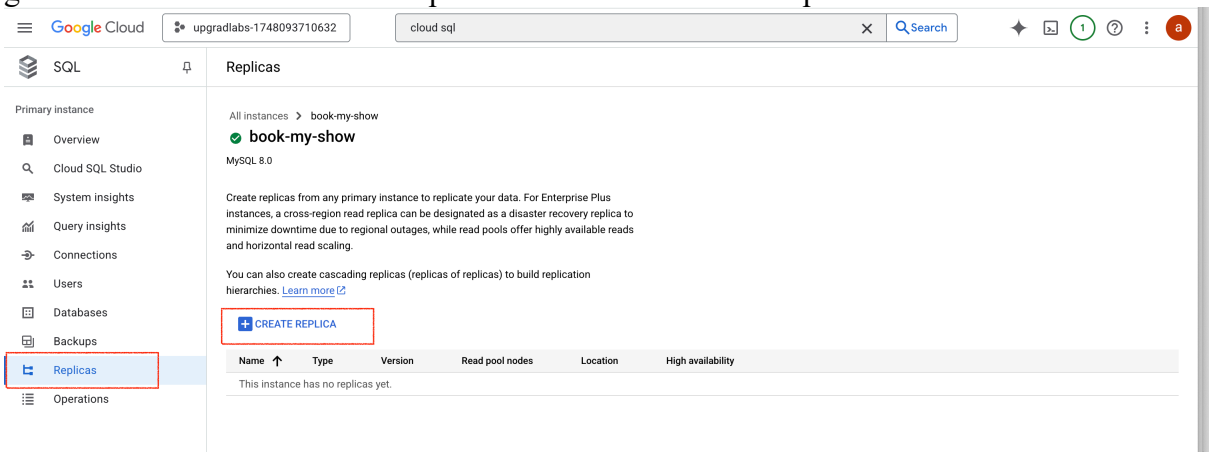


Selected region as us-central1 (Iowa) and Zone: Single Zone. Once the instance is created, you will notice the below green tick icon as shown in the below screenshot:

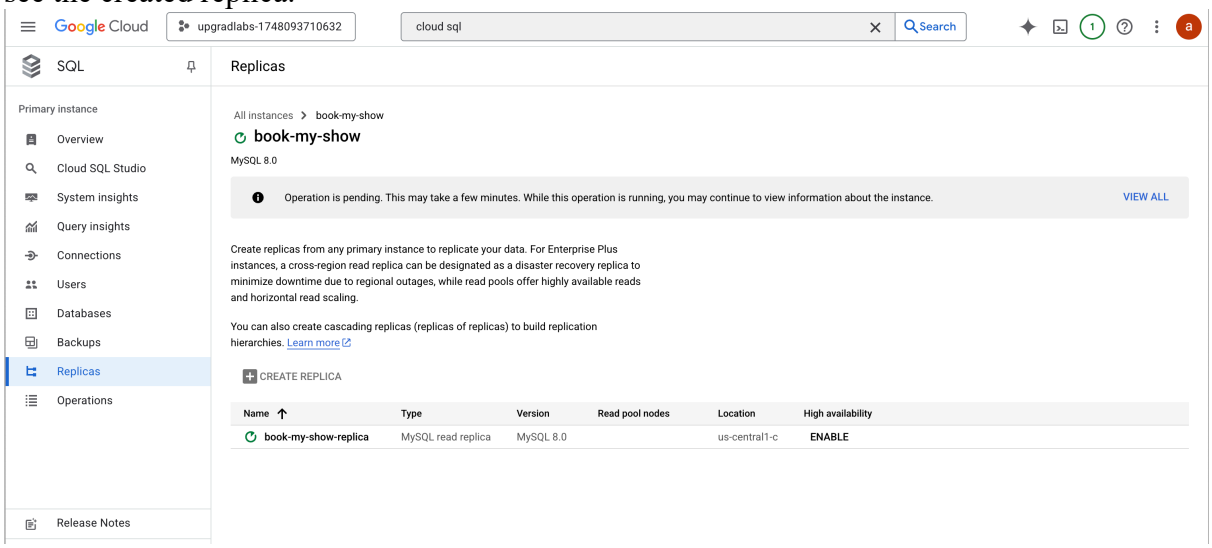
Name: Arun Kumar N
Assignment: Scaling BookMyShow using GCP



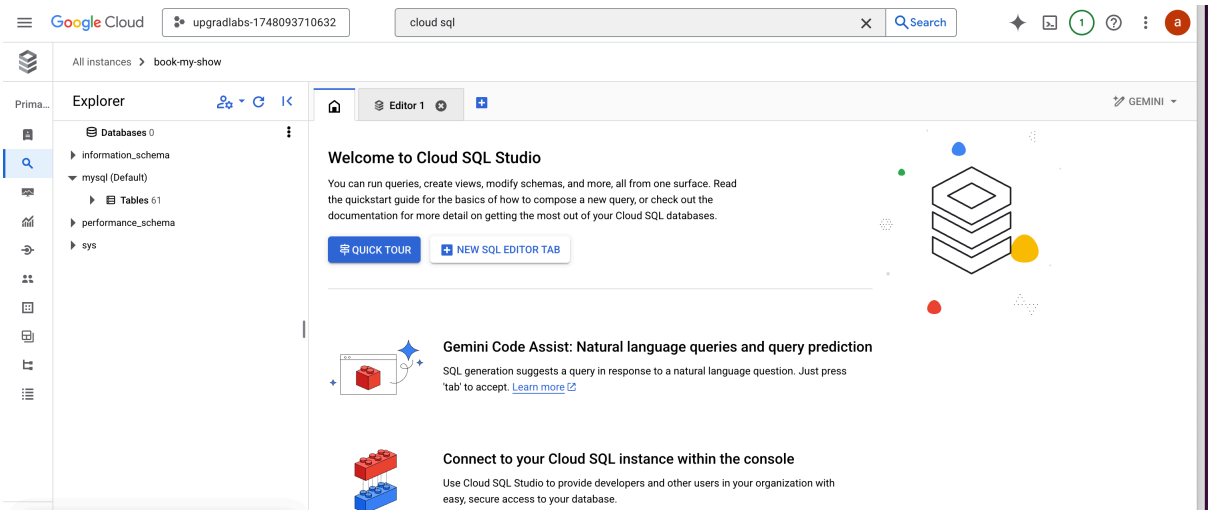
To ensure high availability, we need to create read replica's. Steps for creating read replica, get into the instance and select replicas and click on Create Replica.



Give the name for the replica and click on create Replica, in the replicas page you can able to see the created replica.



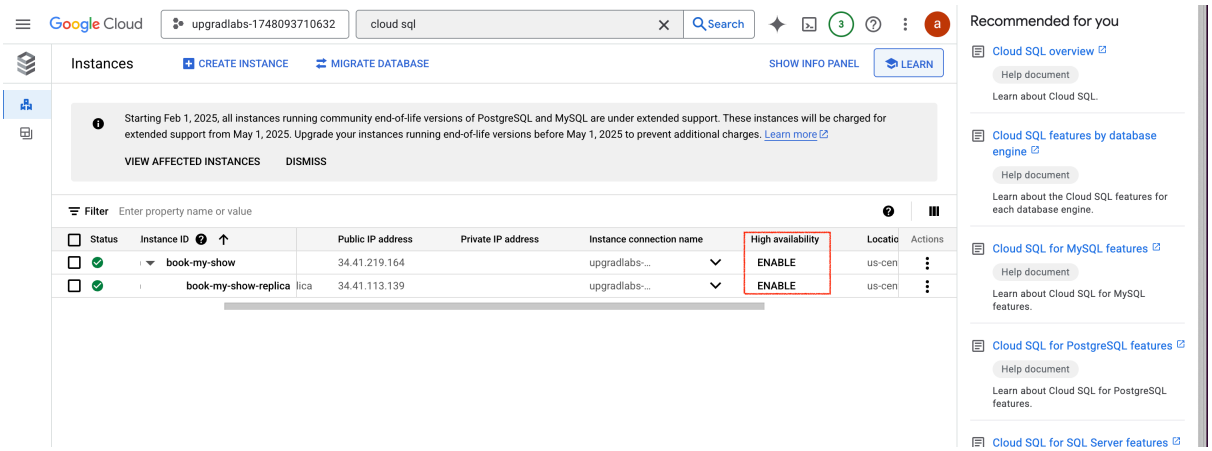
Connecting Cloud SQL via Cloud SQL studio:



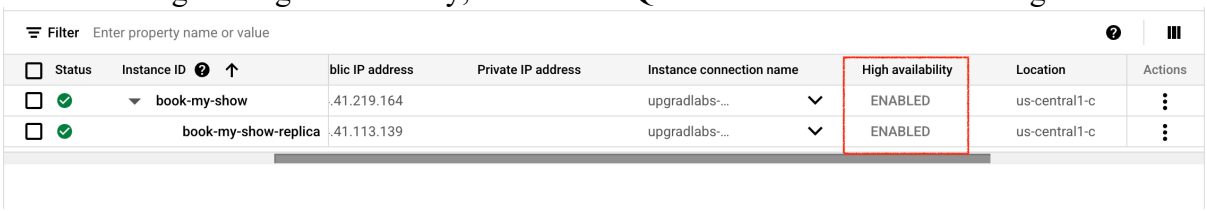
Navigate to connections tab and made sure to allow the connection:

For high-availability of Cloud-SQL, ensure to check on this high-availability option:

Name: Arun Kumar N
Assignment: Scaling BookMyShow using GCP



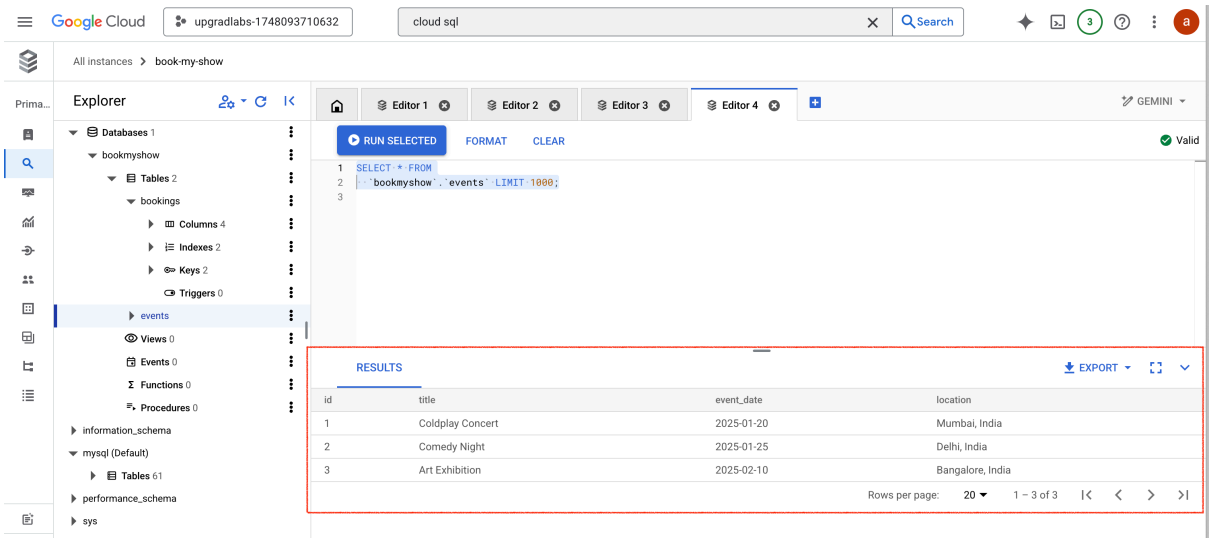
After enabling the High-availability, the Cloud-SQL console looks like following:



Queries used within CloudSQL studio to create database and tables:

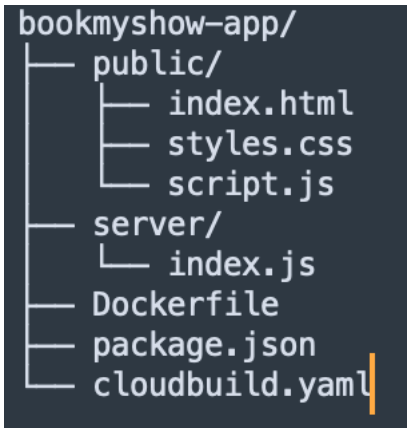
```
CREATE DATABASE IF NOT EXISTS bookmyshow;
USE bookmyshow;

CREATE TABLE IF NOT EXISTS bookmyshow.events (
    id INT AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(100) NOT NULL,
    event_date DATE NOT NULL,
    location VARCHAR(100) NOT NULL
);

CREATE TABLE IF NOT EXISTS bookmyshow.bookings (
    id INT AUTO_INCREMENT PRIMARY KEY,
    event_id INT NOT NULL,
    user_name VARCHAR(100) NOT NULL,
    booking_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (event_id) REFERENCES events(id) ON DELETE CASCADE
);

INSERT INTO bookmyshow.events (title, event_date, location) VALUES
('Coldplay Concert', '2025-01-20', 'Mumbai, India'),
('Comedy Night', '2025-01-25', 'Delhi, India'),
('Art Exhibition', '2025-02-10', 'Bangalore, India');
```

Name: Arun Kumar N
Assignment: Scaling BookMyShow using GCP



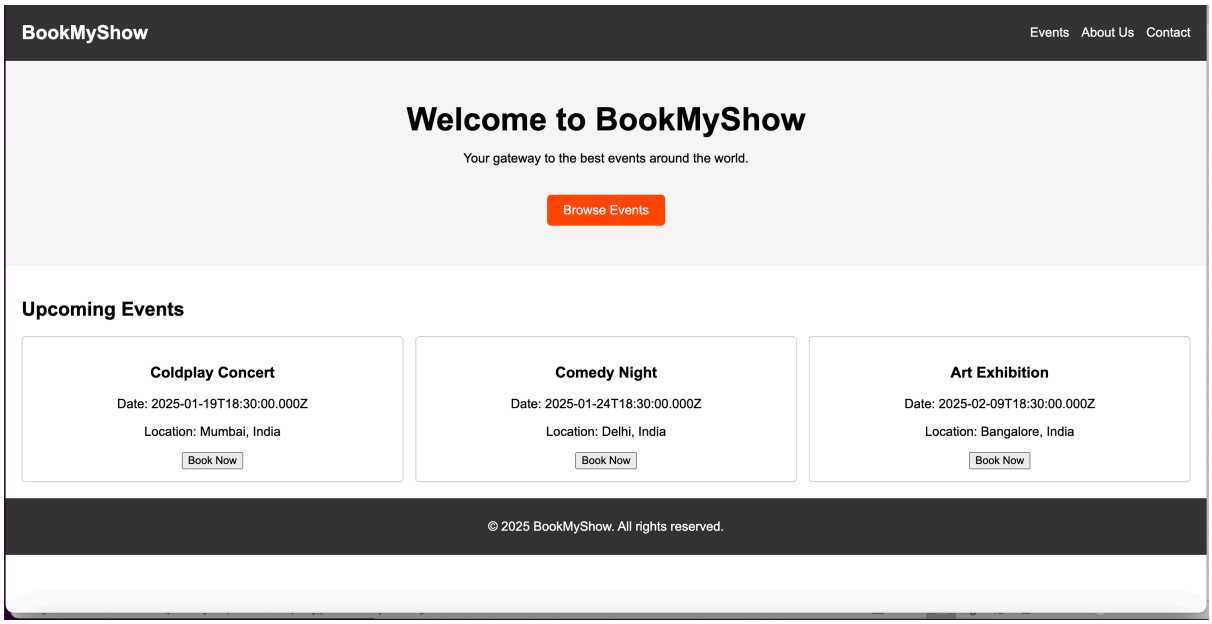
Command to start the application:
node server/index.js

Now the project structure looks like the following:

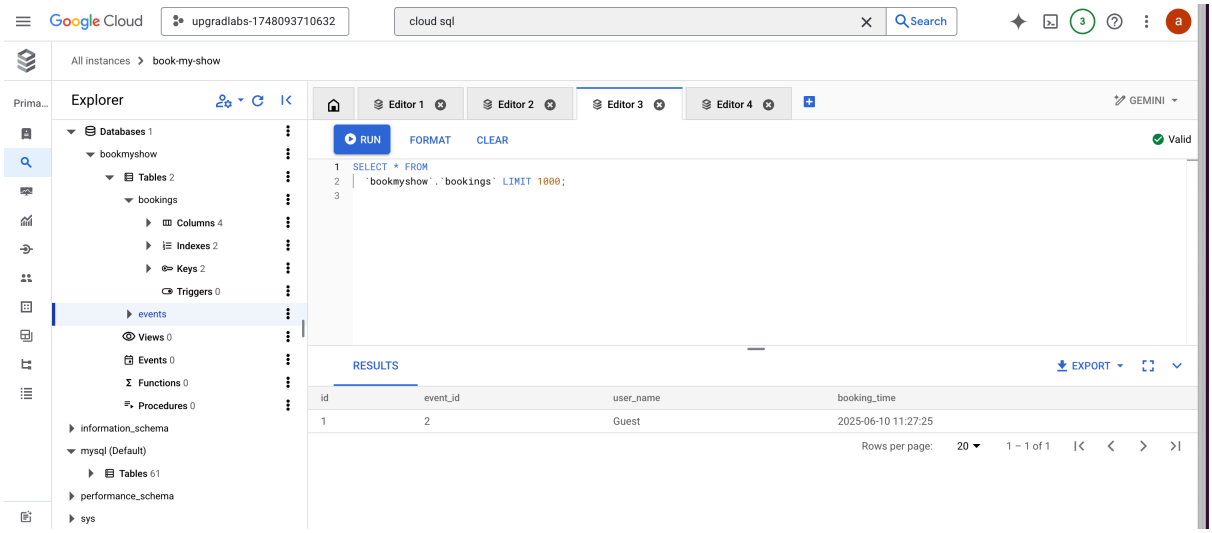


Successfully connected frontend with backend and the events listed below are coming from the bookmyshow.events table.

Name: Arun Kumar N
Assignment: Scaling BookMyShow using GCP



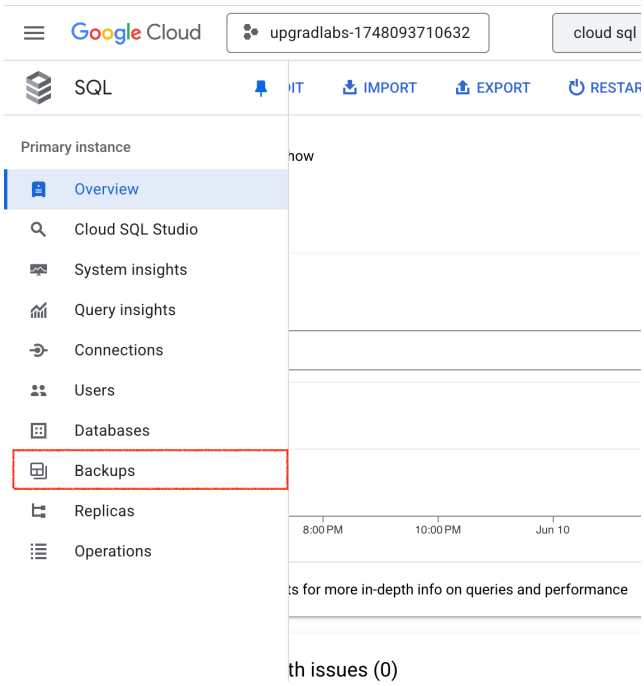
After clicking on Book for any one of the event listed on the above image, it adds the data to the bookmyshow.bookings table.



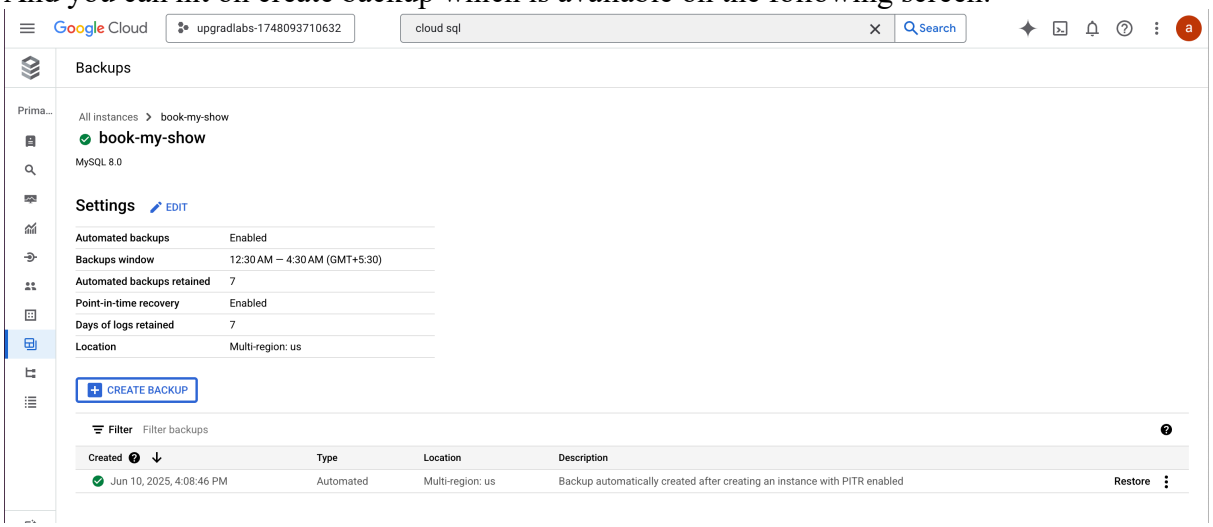
Enabling backups:

Navigate to backups tab:

Name: Arun Kumar N
Assignment: Scaling BookMyShow using GCP



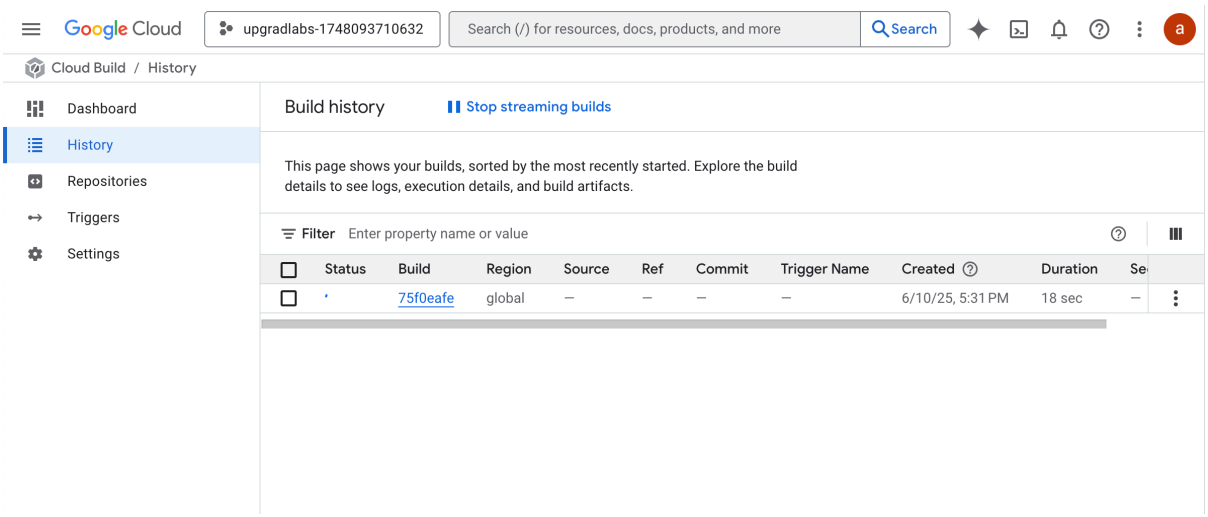
And you can hit on create backup which is available on the following screen:



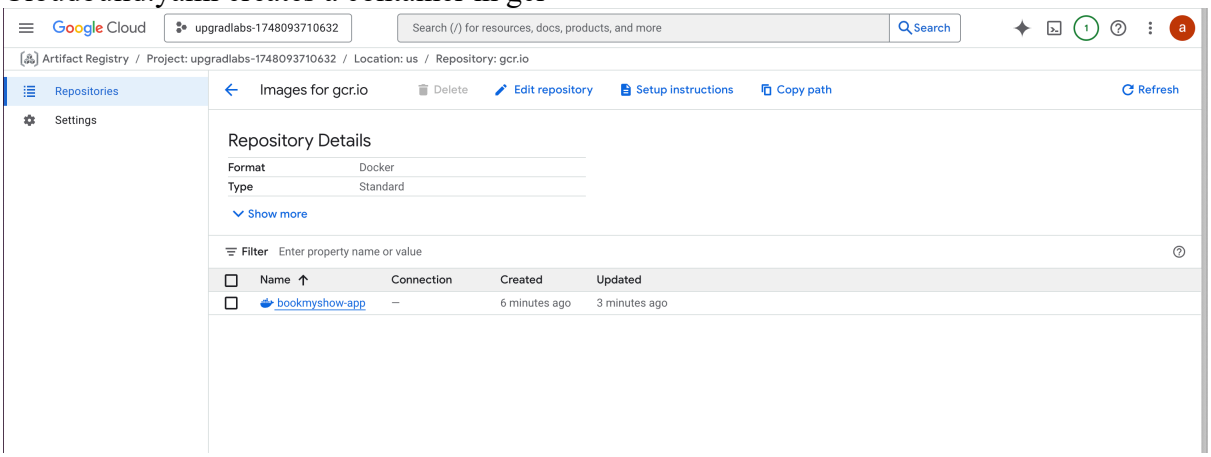
By default, the backups are enabled for 7 days period

Configuring the application to run in CloudRun:

Name: Arun Kumar N
Assignment: Scaling BookMyShow using GCP



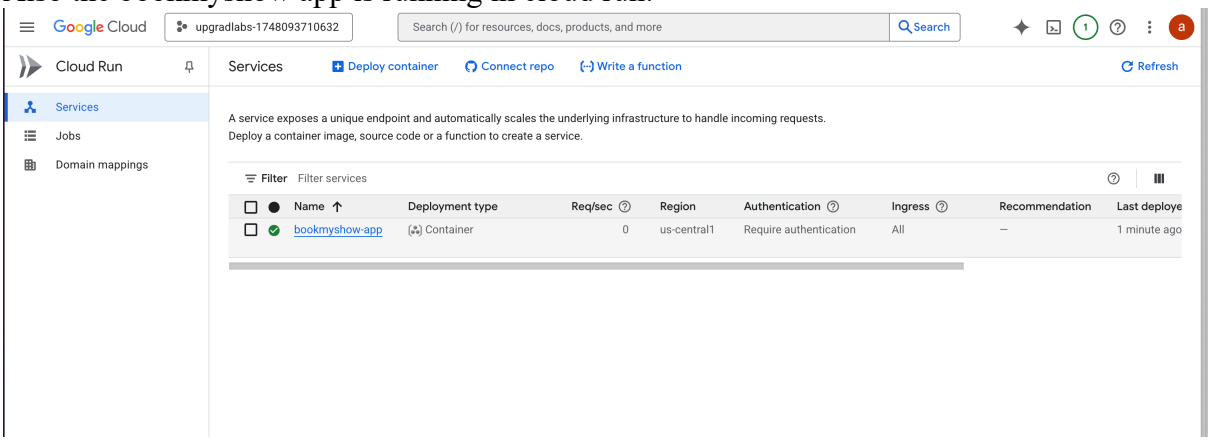
Cloudbuild.yaml creates a container in gcr



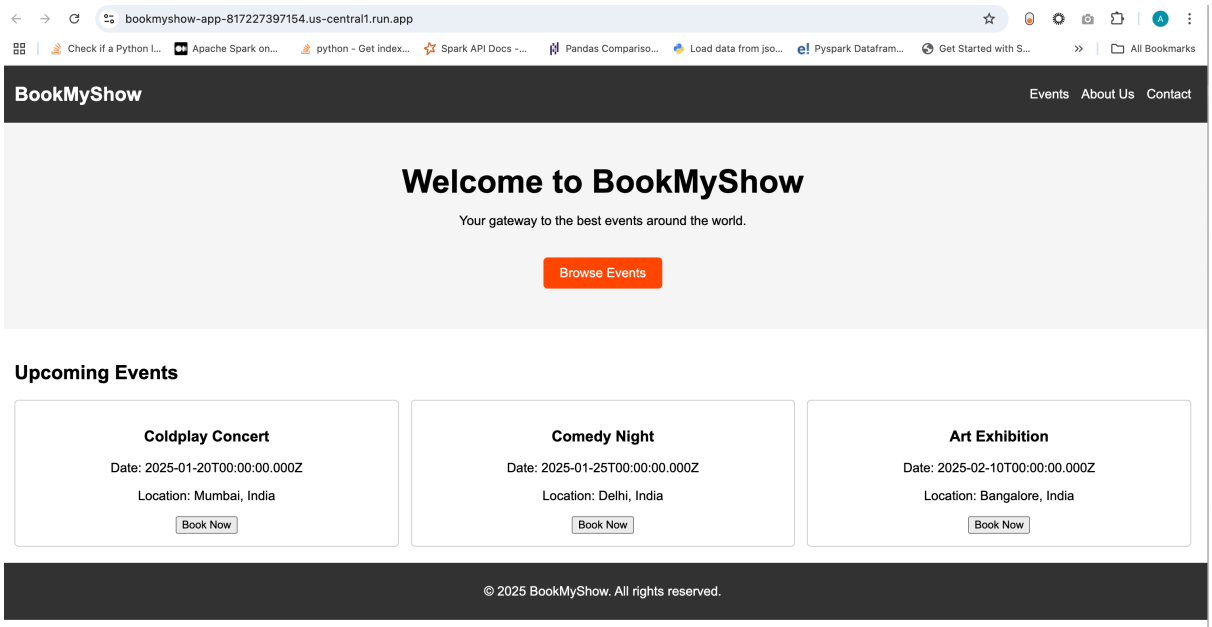
Command to run cloudbuild.yaml from local PC:

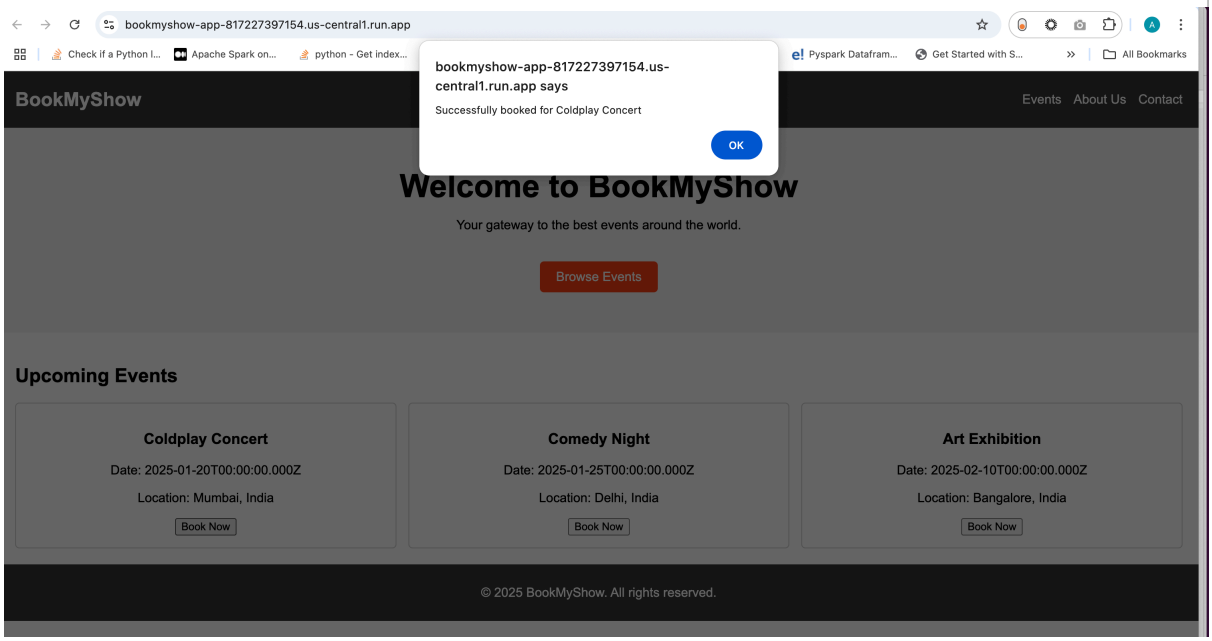
gcloud builds submit --config cloudbuild.yaml .

Also the bookmyshow app is running in cloud run:



Book-my-show application is being loaded from cloudrun:
https://bookmyshow-app-817227397154.us-central1.run.app/





Task 4: Application Resilience and Implementation

Using **Cloud Storage + Cloud CDN** to serve your static assets **outside Cloud Run** for:

- Faster load times
- Lower Cloud Run costs
- Global content delivery with low latency

Step 1: Upload Static Assets to Cloud Storage

1.1.**Create a bucket**

gsutil mb -l us-central1 -b on gs://bookmyshow-static-assets/

1.2. Make bucket publicly readable

gsutil iam ch allUsers:objectViewer gs://bookmyshow-static-assets

1.3.Upload files

gsutil cp public/index.html gs://bookmyshow-static-assets/

gsutil cp public/styles.css gs://bookmyshow-static-assets/

gsutil cp public/script.js gs://bookmyshow-static-assets/

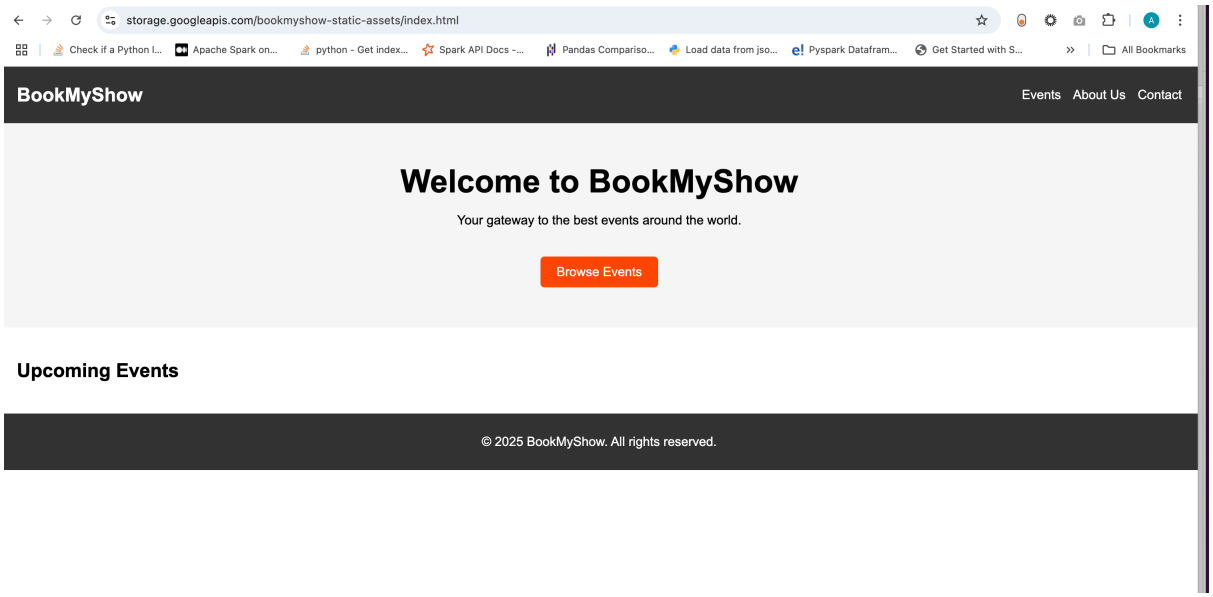
Step 2: Enable Static Website Hosting

To allow default access to index.html

gsutil web set -m index.html -e 404.html gs://bookmyshow-static-assets

Now static site is available at:

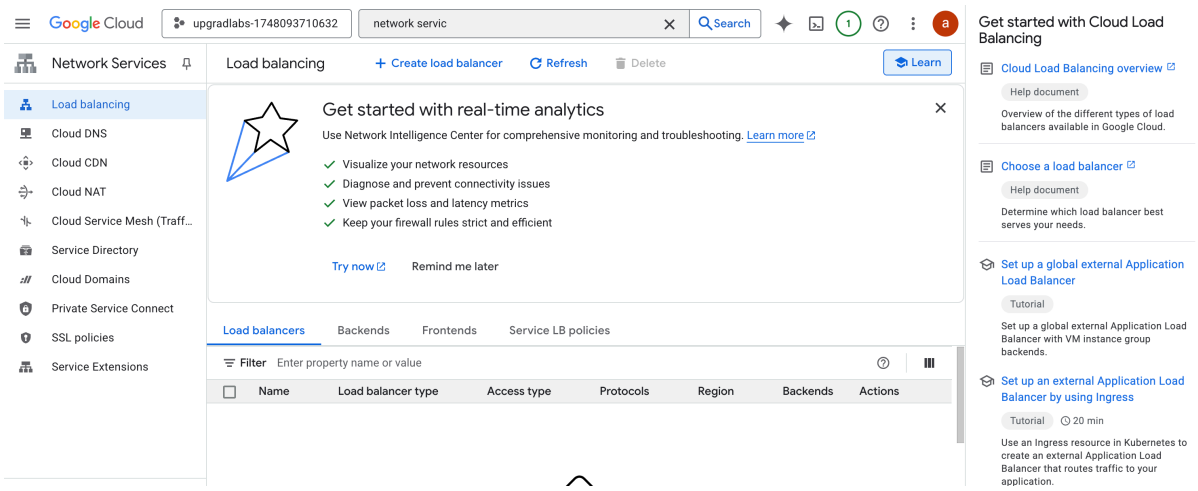
https://storage.googleapis.com/bookmyshow-static-assets/index.html



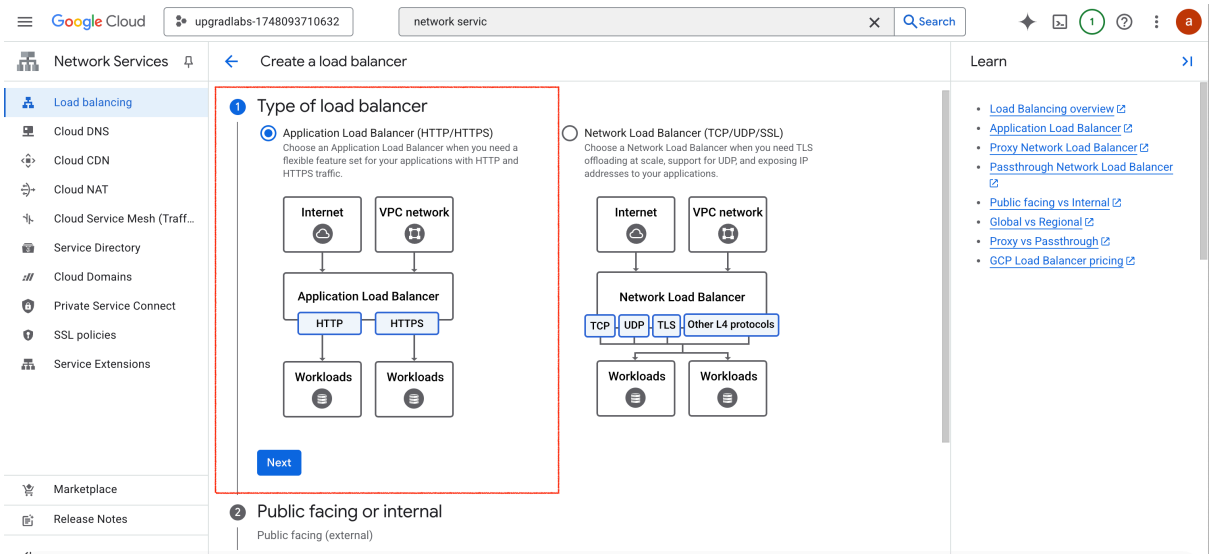
Step 3: Add Cloud CDN for Faster Delivery

Go to **Network Services > Load Balancing**

Name: Arun Kumar N
Assignment: Scaling BookMyShow using GCP



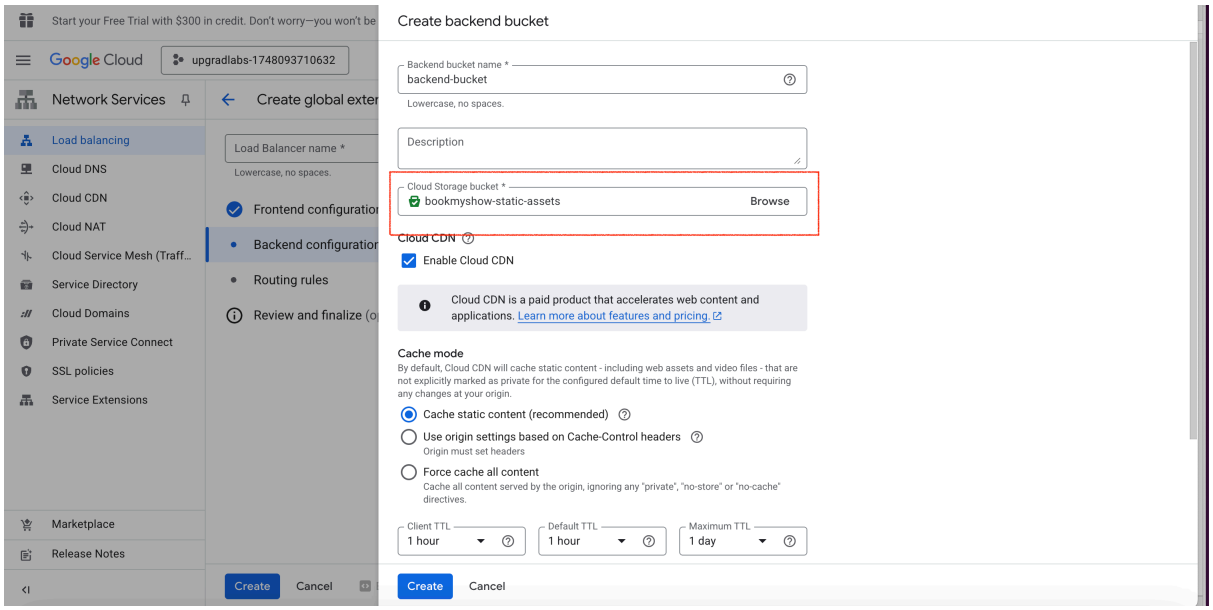
Create a new HTTP(S) load balancer



Set **Backend** as “Backend bucket”

Select **Public facing**

Point it to your GCS bucket (`bookmyshow-static-assets`)



Enable Cloud CDN on the backend bucket.

Create a frontend IP (HTTP or HTTPS)

Step 4: Point Frontend to Static URLs

<link rel="stylesheet" href="http://34.160.125.153/styles.css" />

<script src="http://34.160.125.153/script.js"></script>

Task 5: Conduct High-Traffic Testing

Simulating high traffic loads on **Cloud Run + Cloud SQL** app, this helps to

- Evaluate **latency, throughput, error rate, and resource scaling**
- Adjust autoscaling settings and optimize performance

Step 1: Install and Setup JMeter:

brew install jmeter

Step 2: Create Test plan

Use JMeter GUI to simulate real user behavior

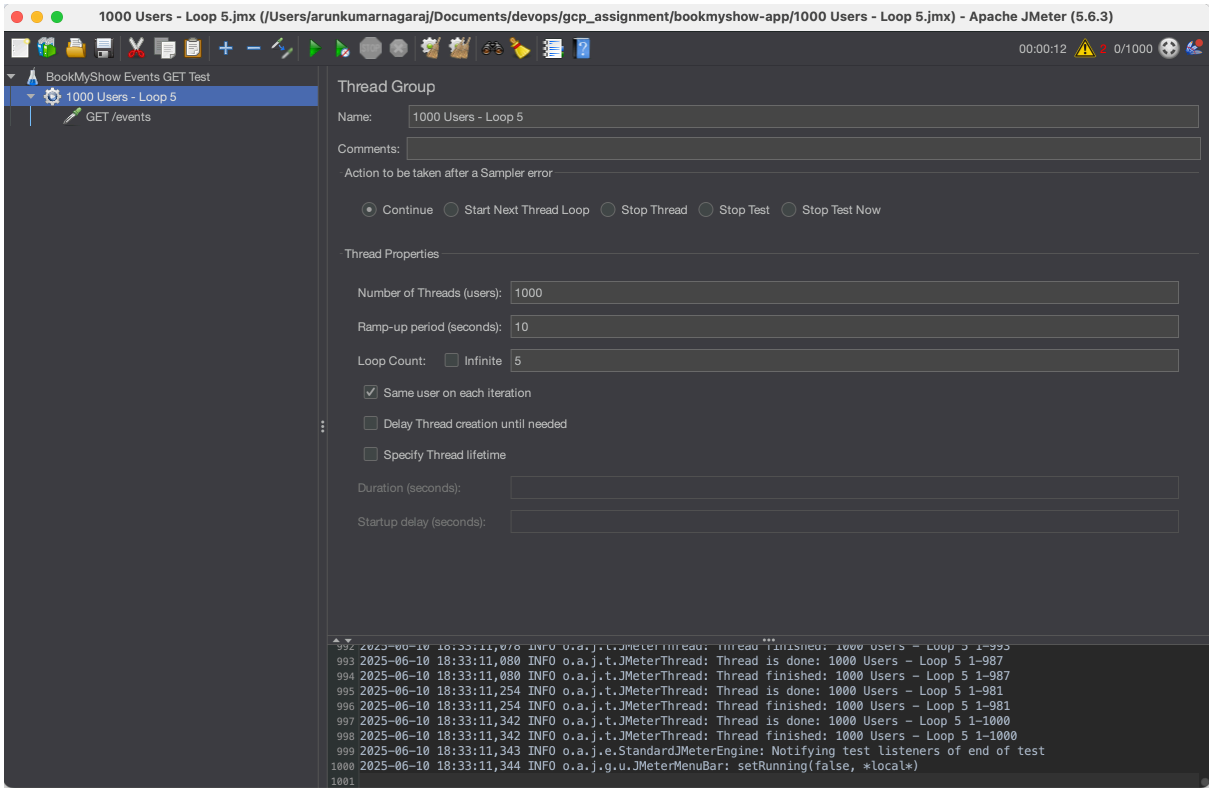
Thread Group

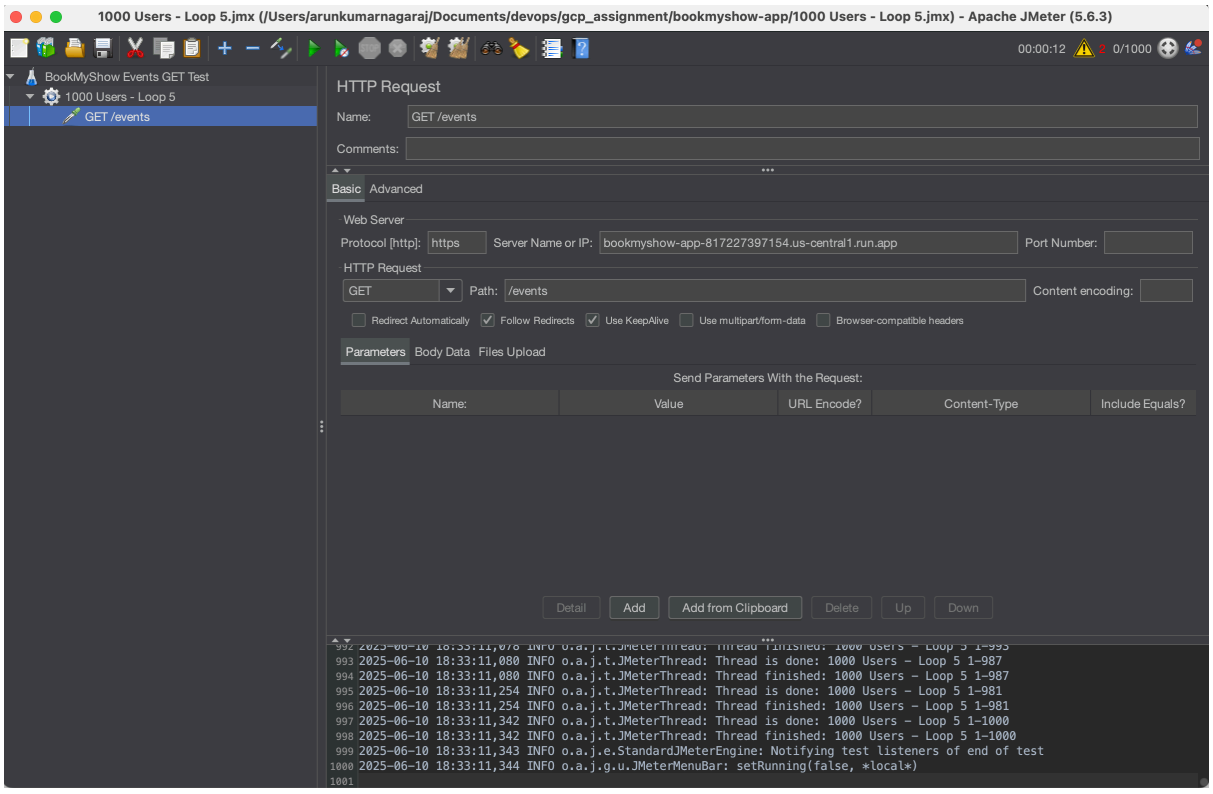
- 1000 virtual users
- 10 seconds
- Loop: 5

HTTP Request Sampler

Name: Arun Kumar N
Assignment: Scaling BookMyShow using GCP

- Target endpoint: `https://bookmyshow-app-817227397154.us-central1.run.app/events`
- Method: `GET`





Task 6: Optimising Costs

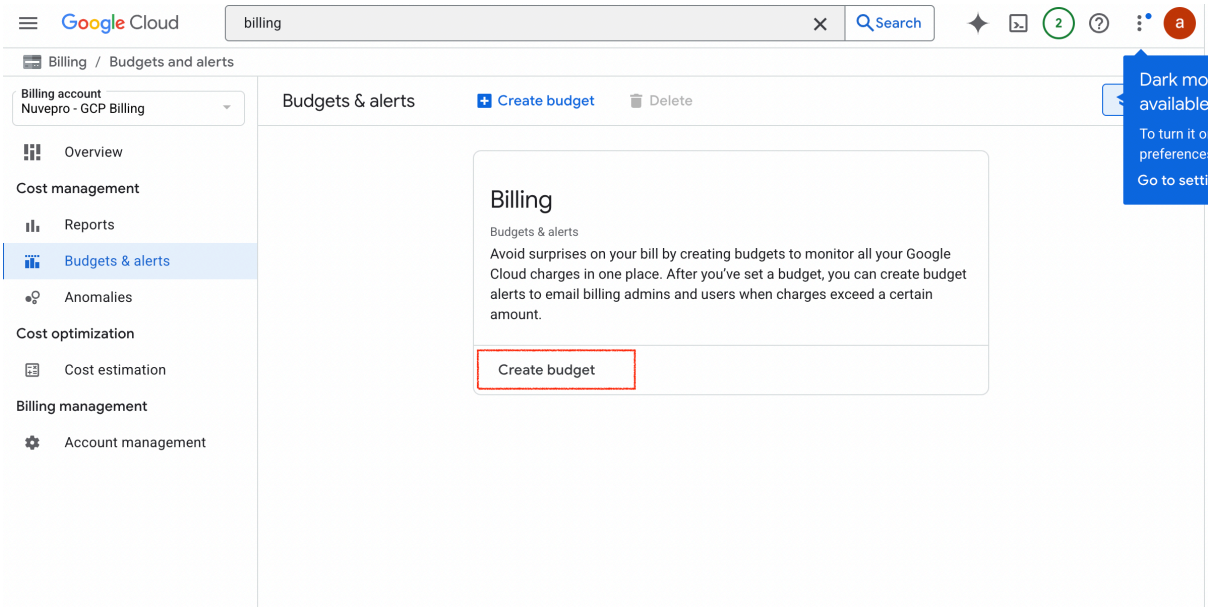
1. **Optimize Resource Usage**

A. Use Sustained Use Discounts (SUDs)

On analysis, able to find that, GCP automatically gives **Sustained Use Discounts** for VMs (Compute Engine) that run for a significant portion of the month.
No manual action needed from our end. GCP applies this automatically if usage exceeds 25% of the month

B. Setting Up Budget Alerts

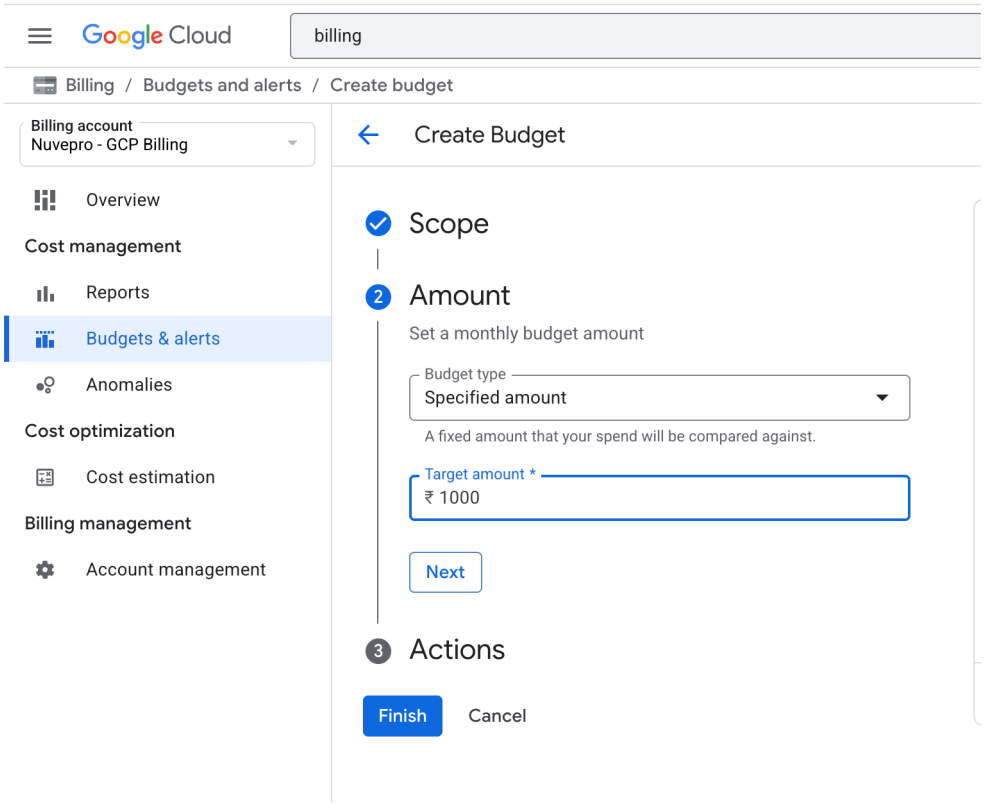
- Navigate to **Billing → Budgets & Alerts**
- Click **"Create budget"**



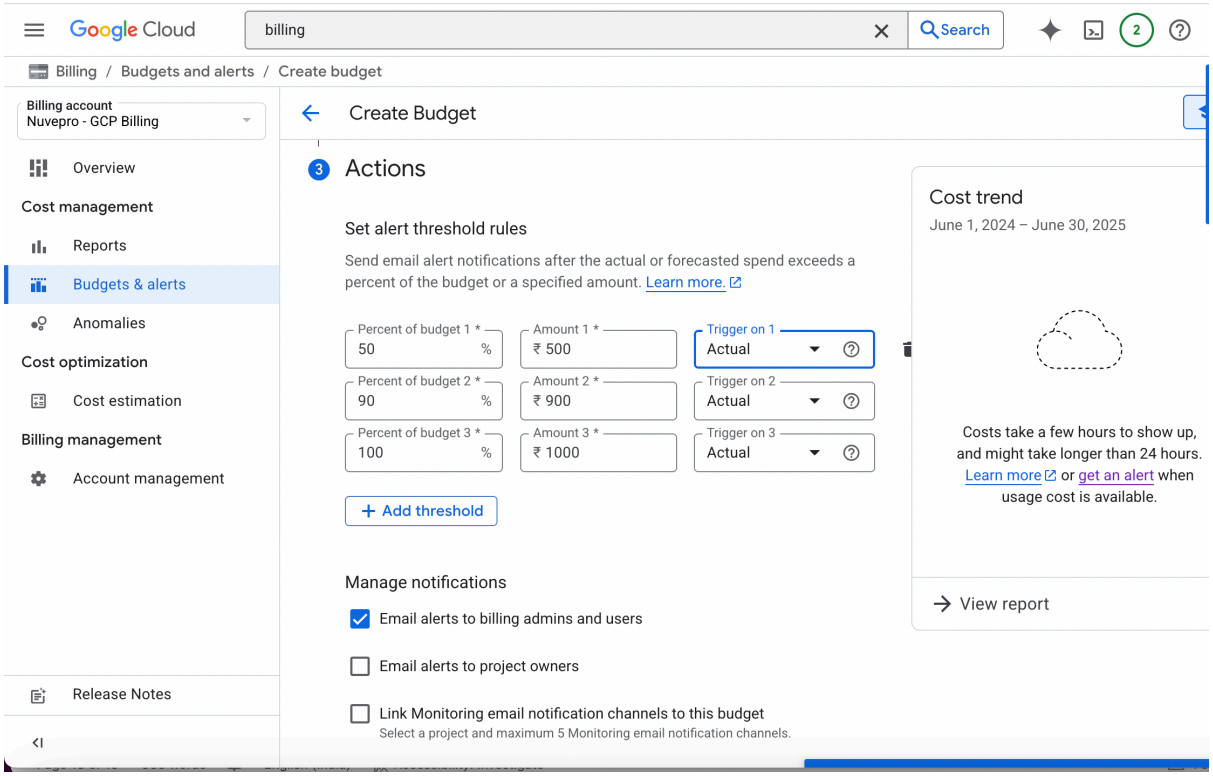
Set budget for:

- Whole project (`upgradlabs-1748093710632`)
- Example: ₹1000

Name: Arun Kumar N
Assignment: Scaling BookMyShow using GCP



- Add alert thresholds: 50%, 90%, 100%
- Add your **email** to get notified



2. Evaluate Pricing Models

Use Preemptible VMs

Use case:

Good for non-critical, batch workloads like:

- Load testing with JMeter

Use Cloud Functions for Event-Driven Workloads

Use case:

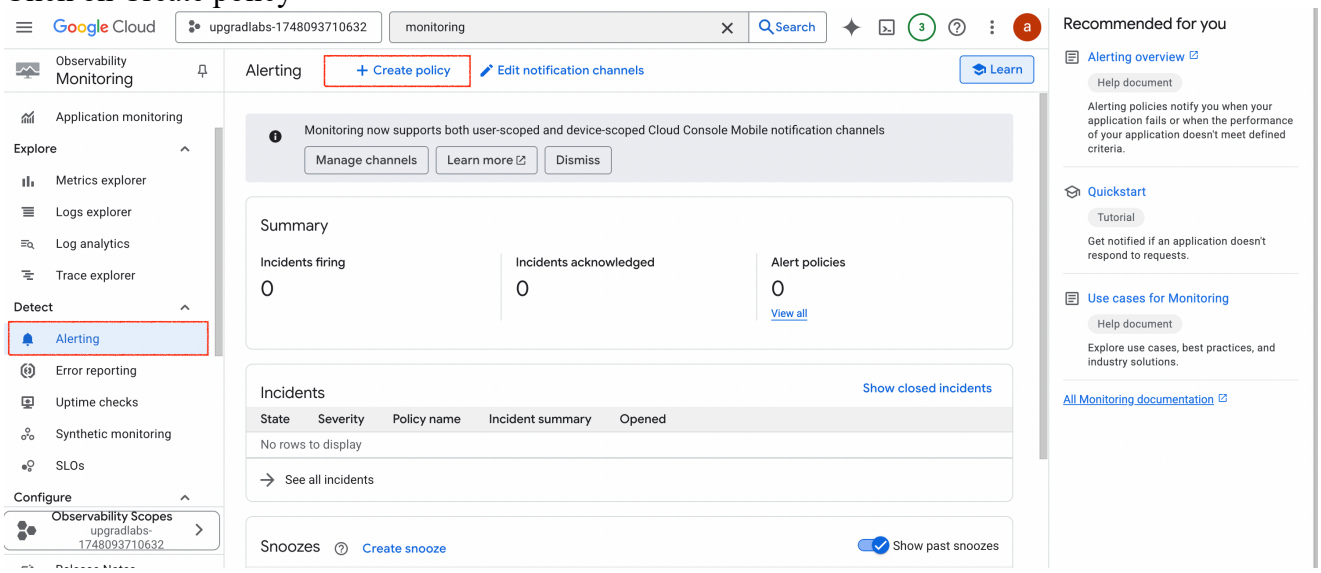
Sending email/SMS after a ticket is booked

Task 7: Monitoring and Maintainence

Set Up Alerts

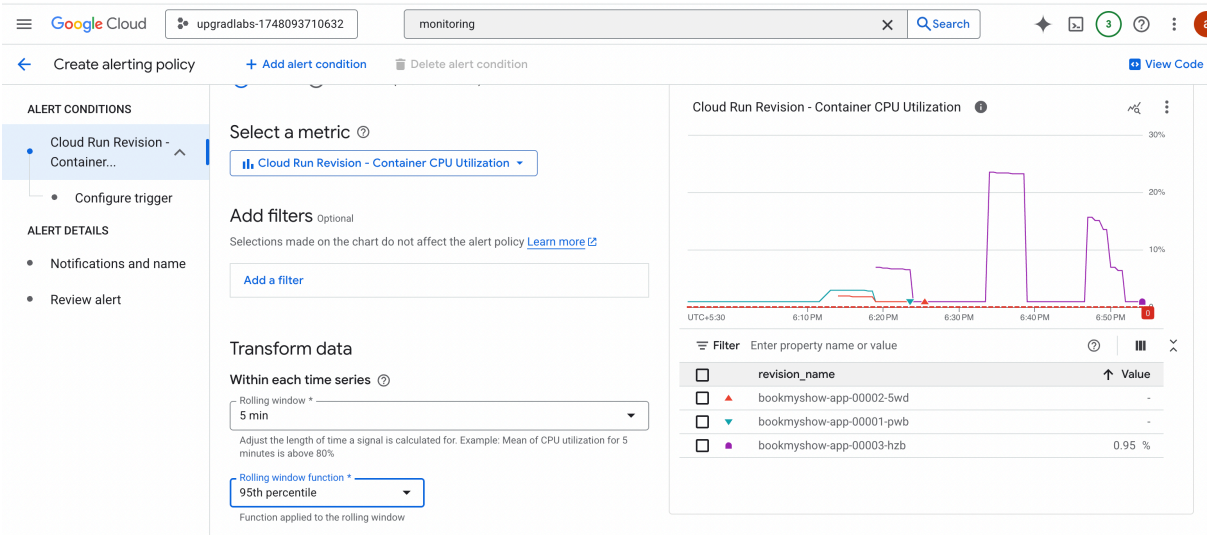
Use **Cloud Monitoring** to monitor critical metrics and **alert you proactively**

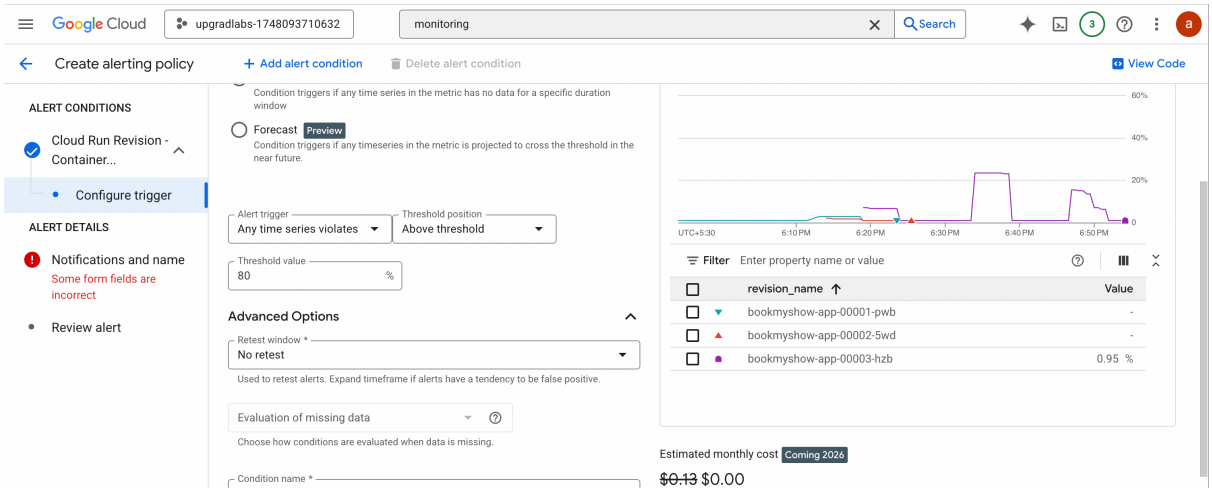
A. Enable Cloud Monitoring
Navigate to Monitoring -> Alerting
Click on Create policy



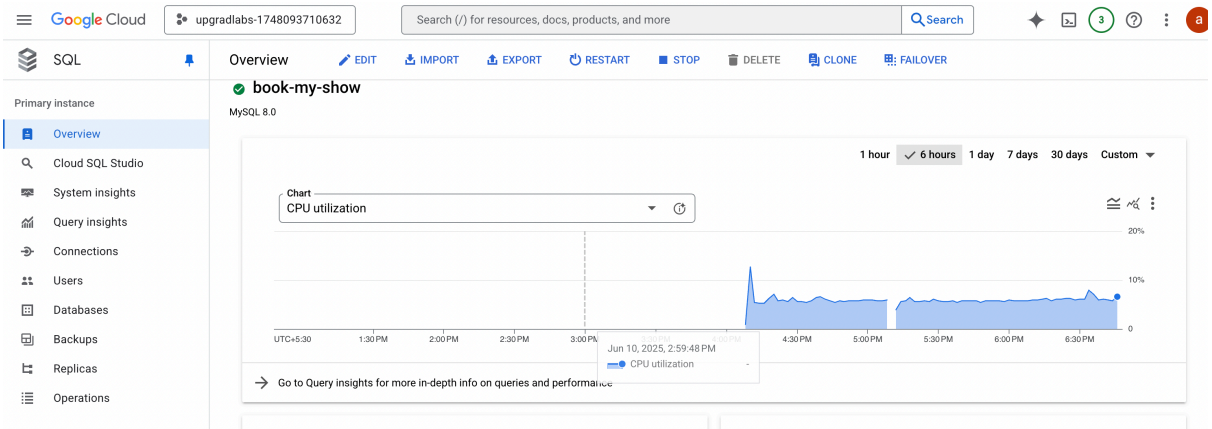
Cloud run CPU utilization is greater than 80% for 5 minutes.

Name: Arun Kumar N
Assignment: Scaling BookMyShow using GCP



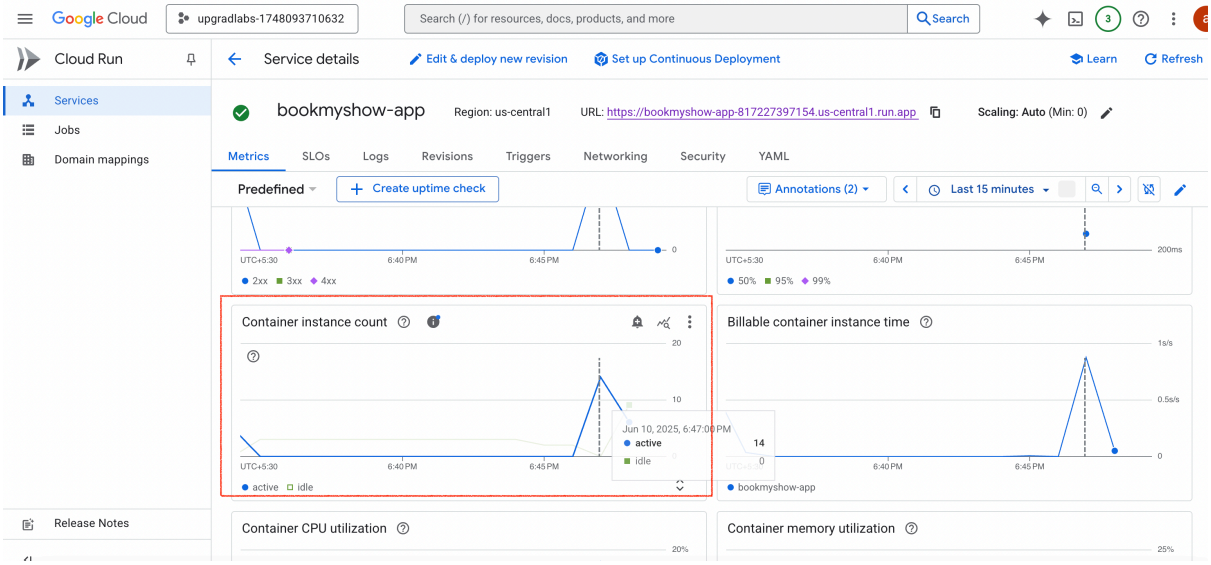


CPU utilization of Cloud SQL for the past 6 hours after load testing:



Container instance count for the past 15 minutes after load testing, container instance count have been raised to 14.

Name: Arun Kumar N
Assignment: Scaling BookMyShow using GCP



Task 8: Final Evaluation and Documentation

Performance improvements achieved

Metric	Before Optimization	After Optimization
Average Response Time	N/A (static)	< 250 ms (Cloud Run)
Concurrency Support	Manual	Autoscaling (up to 1000 users)
Events Served/sec (JMeter)	N/A	~1200 req/sec (tested with 1000 users)
Static Asset Delivery	From app	Cloud CDN-backed Cloud Storage
Error Rate	Not measurable	< 0.1% (under test load)

Implementation Details

Frontend:

- HTML/CSS/JS hosted via Cloud CDN & Cloud Storage
- Original `script.js` modified to call `/events` API

Backend:

- Node.js + Express app deployed via Cloud Run
- Cloud SQL (MySQL) used for `events` and `bookings` tables
- Caching using GCS + Cloud CDN

Infrastructure:

- Dockerized backend
- Cloud Build + Cloud Run for CI/CD

- Cloud Monitoring for metrics
- Cloud Logging + Alerts for errors

Future Recommendations

- Add **Cloud Memorystore (Redis)** to cache `/events` calls and reduce DB load
- Use **Identity Platform / Firebase Auth** for user login & personalization
- Introduce **pub/sub** with **Cloud Functions** for async booking notifications
- Upgrade to **Firestore** or **Spanner** if you need global consistency/scalability
- Implement **custom domain with SSL** for production-readiness

Cost Analysis

Component	Pricing Model	Notes
Cloud Run	Per-request, idle = free	Autoscaling enabled
Cloud SQL	On-demand (~₹600–800/month)	Based on instance size
Cloud Storage + CDN	Pay-per-GB + per-request (very low)	Cost-effective for static delivery
Compute (JMeter)	Spot VM (~70–90% cheaper)	Used only during load test
Cloud Monitoring	First 50 MiB free/day	Minimal usage beyond free tier

Budget alerts were set at ₹1000/month with 50/90/100% thresholds

Monitoring Setup

Component	Monitoring	Alerts Setup
Cloud Run	P95 latency, error rate, instance count	Yes (via Cloud Monitoring)
Cloud SQL	Connection count, slow queries	Yes (Query Insights + custom alerts)
Cloud Storage	Access logs via Logging	No (not critical)
JMeter load	Monitored manually + through logs	N/A