TrekTale

Share Your Journey, Inspire the World

A Cloud-Powered Platform for Travel Enthusiasts

Cloud Computing CISC-5550

Project Member:

Akash Kumar Yadav

Fordham University, Lincoln Center

113 W 60th Street, New York, NY 10023, USA

Instructor: Prof. Zhou Ji

May 5, 2025

http://35.188.97.99:5173/login

1. INTRODUCTION

TrekTale is a live website aimed at uniting travel enthusiasts from across the globe. It is a vibrant cyber space where travelers interact and share their own unique travel experience in the form of exciting stories and breathtaking photographs, building a worldwide community of exploration and cultural discovery lovers. Originally built using the MERN stack, TrekTale has been containerized and deployed on Google Cloud Platform (GCP) with Docker and Kubernetes, delivering a scalable, reliable, and accessible experience in uniting adventurers worldwide.

2. PROJECT OVERVIEW & KEY FEATURES

Getting started with TrekTale is simple: users can log in with their pre-existing credentials or create a new account easily. Users have access to all the functionality of TrekTale upon logging in. Users can then proceed to share their very own unique experience by posting new 'tales.' They do this by uploading the photograph taken from the excursion and, most importantl,y posting a written review beneath it explaining their experience, their feelings, and the highlights. They can even geotag where their trek took place, so other people can easily view stories of a specific location. It makes the site more worthwhile because not only do people share their trip, but the feelings and lessons they've learned along the way, which subsequently results in TrekTale's active community of travelers.

Key features include:

- User registration and login.
- Creation of "tales" with image uploads and rich text descriptions.
- Ability to geotag trek locations.
- A community platform for sharing and discovering travel experiences.

3. EXTENSIONS BEYOND HOMEWORK #4

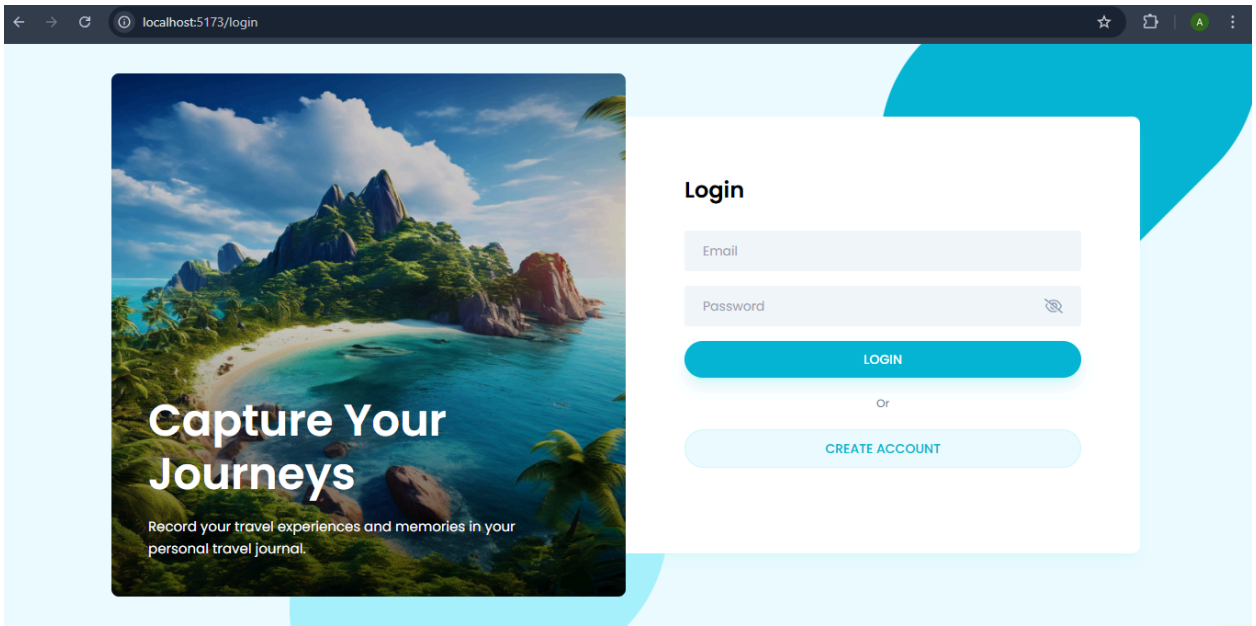
This project significantly builds upon the foundations laid in Homework #4 by incorporating several key enhancements and exploring more advanced cloud technologies:

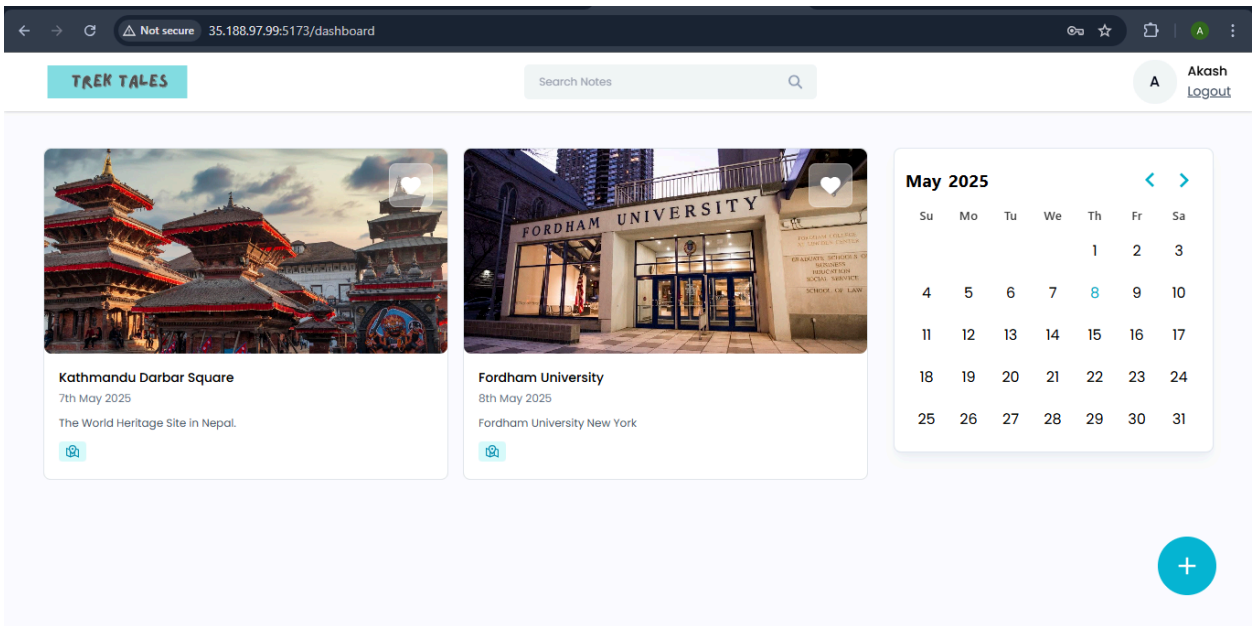
Migration to Cloud-Hosted Database (MongoDB Atlas): Homework #4 might have employed a local or simpler database setup. The project migrated the backend to make use of MongoDB Atlas, which is a cloud-hosted, fully managed NoSQL database service. This provides enhanced scalability, reliability, and data persistence suitable for a production setting.

Robust Configuration Management with Kubernetes ConfigMaps: Sensitive configurations, such as the MongoDB connection string, were initially managed less securely. This project implements Kubernetes ConfigMaps, populated from a .env file, to securely inject configuration into the backend pods. This is a best practice for managing application settings in Kubernetes.

Full Application Containerization and Orchestration: While Homework #4 introduced the notion of containerization, this project ensures that both backend and frontend services are fully containerized and orchestrated using Docker Compose for local development and Kubernetes (GKE) for deployment into production. This includes separate Dockerfiles for each service and extensive Kubernetes manifest files.

Enhanced Deployment Automation: A deploy.sh script was developed to streamline the deployment of Kubernetes resources, making the update process more efficient and repeatable.



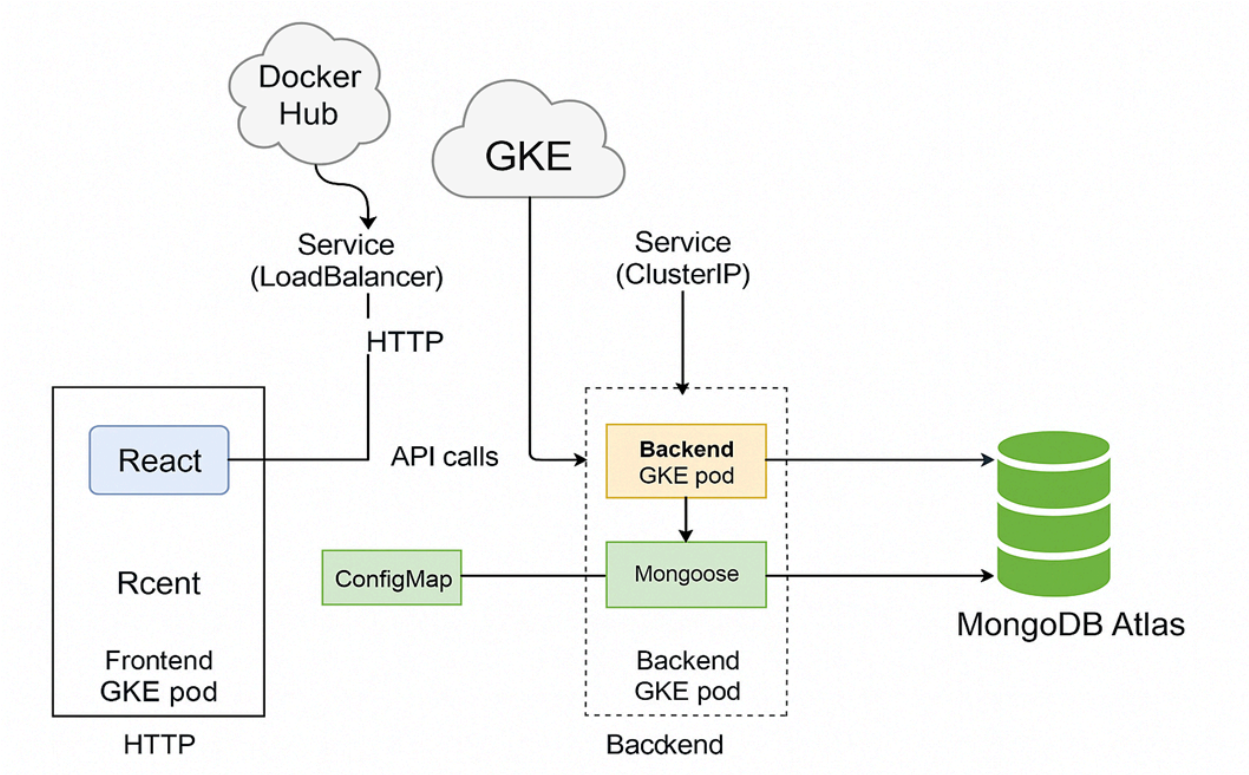


These extensions demonstrate a deeper dive into cloud-native architecture, production-readiness, and advanced deployment techniques.

4. TECHNOLOGY STACK

- **Frontend**: React + Vite
- **Backend**: Express + Node.js + Mongoose
- **Database**: MongoDB Atlas
- **Containerization**: Docker
- **Container Registry**: Docker Hub
- **Orchestration:**
  - Docker Compose (For multi-container local development)
  - Google Kubernetes Engine (GKE) (For production deployment, scaling, and management)
- **Cloud Platform and Infrastructure**: Google Cloud Platform (GCP)

5. ARCHITECTURE DIAGRAM



6. LOCAL DEVELOPMENT ENVIRONMENT

6.1. Project File Structure

The project is organized into separate frontend and backend folders, each with their own source code, Dockerfile, and package.json. There is a docker-compose.yml at the root that orchestrates the local development environment.

```yaml
# version: "3.9"

services:
  frontend:
    build:
      context: ./frontend
      dockerfile: Dockerfile
    ports:
      - "5173:5173"
    volumes:
      - ./frontend:/frontend
      - /frontend/node_modules
    environment:
      - NODE_ENV=development
      - CHOKIDAR_USEPOLLING=true
    networks:
      - app-network
    command: npm run dev

  backend:
    build:
      context: ./backend
      dockerfile: Dockerfile
    ports:
```

```
backend-1   | > nodemon index.js
backend-1   | > nodemon index.js
frontend-1  |   ➜  Local:   http://localhost:5173/
backend-1   | [nodemon] to restart at any time, enter `rs`
backend-1   | [nodemon] watching path(s): *.*
backend-1   | [nodemon] watching extensions: js,mjs,cjs,json
backend-1   | [nodemon] starting `node index.js`
```

6.2. Containerization with Docker

Separate Dockerfiles are used for containerizing the frontend and backend applications.

6.2.1. Backend Dockerfile:

This Dockerfile sets up the Node.js environment, installs dependencies, copies the backend source code, and specifies the command to start the server.

```dockerfile
# Backend Dockerfile
FROM node:18-alpine

WORKDIR /backend

COPY package*.json ./
RUN npm install

COPY . .

ENV NODE_ENV=development
EXPOSE 8000

CMD ["npm", "start"]
```

6.2.2. Frontend Dockerfile:

This Dockerfile sets up the Node.js environment for building the React/Vite application and then uses a lightweight server (like Nginx or Vite's preview server) to serve the built static assets.

```
Dockerfile 1 ×
TrekTale > frontend > Dockerfile > ...
# Frontend Dockerfile
FROM node:18-alpine

WORKDIR /frontend

COPY package*.json ./
RUN npm install

COPY . .
RUN npm run build

EXPOSE 5173
CMD ["npm", "run", "dev"]
```

6.2.3. Docker Compose

The **docker-compose.yml** file defines and runs the multi-container TrekTale application locally. It specifies the build contexts, port mappings, volumes for live reloading (in development), environment variables, and networks.

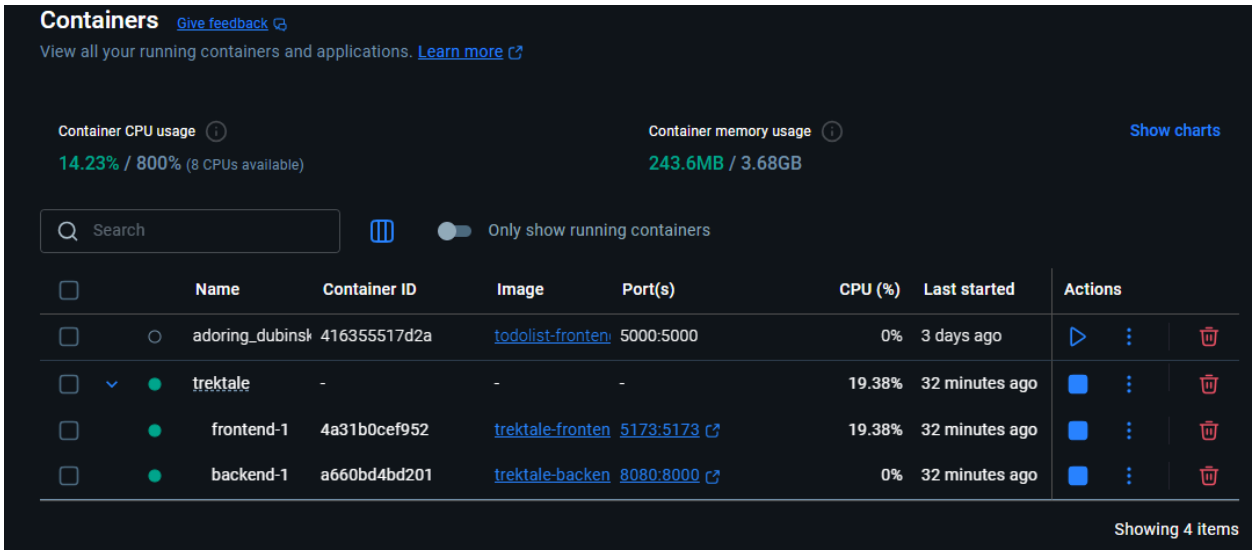
```
docker-compose.yml ×
TrekTale > docker-compose.yml > YAML > {} services
docker-compose.yml - The Compose specification establishes a standard for the definition of multi-container platform-agnostic applications (compose-spec.json) | ▷Run All Services
services:
  ▷Run Service
  frontend:
    build:
      context: ./frontend
      dockerfile: Dockerfile
    ports:
      - "5173:5173"
    volumes:
      - ./frontend:/frontend
      - /frontend/node_modules
    environment:
      - NODE_ENV=development
      - CHOKIDAR_USEPOLLING=true
    networks:
      - app-network
    command: npm run dev

  ▷Run Service
  backend:
    build:
      context: ./backend
      dockerfile: Dockerfile
    ports:
      - "8080:8000"
    volumes:
      - ./backend:/backend
    env_file:
      - ./backend/.env
    environment:
      - NODE_ENV=development
    networks:
      - app-network

networks:
  app-network:
    driver: bridge
```

6.3. Running Locally

The project was successfully run locally using **docker compose up --build**. Both frontend and backend containers were created and are functional.

```
C:\Users\Akash yadav\Downloads\TrekTale\TrekTale>docker compose up --build
Compose can now delegate builds to bake for better performance.
 To do so, set COMPOSE_BAKE=true.
[+] Building 1.5s (22/22) FINISHED                                                                                docker:desktop-linux
 => [frontend internal] load build definition from Dockerfile                                                                     0.1s
 => => transferring dockerfile: 217B                                                                                              0.0s
 => [backend internal] load build definition from Dockerfile                                                                      0.1s
 => => transferring dockerfile: 362B                                                                                              0.0s
 => [backend internal] load metadata for docker.io/library/node:18-alpine                                                         0.7s
 => [backend auth] library/node:pull token for registry-1.docker.io                                                               0.0s
 => [frontend internal] load .dockerignore                                                                                        0.1s
 => => transferring context: 77B                                                                                                  0.0s
 => [backend internal] load .dockerignore                                                                                         0.0s
 => => transferring context: 99B                                                                                                  0.0s
 => [frontend 1/5] FROM docker.io/library/node:18-alpine@sha256:8d6421d663b4c28fd3ebc498332f249011d118945588d0a35cb9bc4b8ca09d9e   0.0s
 => [backend internal] load build context                                                                                         0.1s
 => => transferring context: 551B                                                                                                 0.0s
 => [frontend internal] load build context                                                                                        0.1s
 => => transferring context: 2.39kB                                                                                               0.0s
 => CACHED [backend 2/5] WORKDIR /backend                                                                                         0.0s
 => CACHED [backend 3/5] COPY package*.json ./                                                                                    0.0s
 => CACHED [backend 4/5] RUN npm install                                                                                          0.0s
 => CACHED [backend 5/5] COPY . .                                                                                                 0.0s
 => CACHED [frontend 2/6] WORKDIR /frontend                                                                                       0.0s
 => CACHED [frontend 3/6] COPY package*.json ./                                                                                   0.0s
 => CACHED [frontend 4/6] RUN npm install                                                                                         0.0s
 => CACHED [frontend 5/6] COPY . .                                                                                                0.0s
 => CACHED [frontend 6/6] RUN npm run build                                                                                       0.0s
 => [frontend] exporting to image                                                                                                 0.1s
 => => exporting layers                                                                                                           0.0s
 => => writing image sha256:4b815d5d78744bcb44343bdb0a9531fe78236539affb69d468d22a1d4935ac47                                      0.0s
 => => naming to docker.io/library/trektale-frontend                                                                              0.0s
 => [backend] exporting to image                                                                                                  0.1s
 => => exporting layers                                                                                                           0.0s
 => => writing image sha256:c6abef5783314918dca0b6c1c239d5124cb7c7acc8e6ba6f2d7d65b8e46f1013                                      0.0s
 => => naming to docker.io/library/trektale-backend                                                                               0.0s
 => [frontend] resolving provenance for metadata file                                                                             0.1s
```

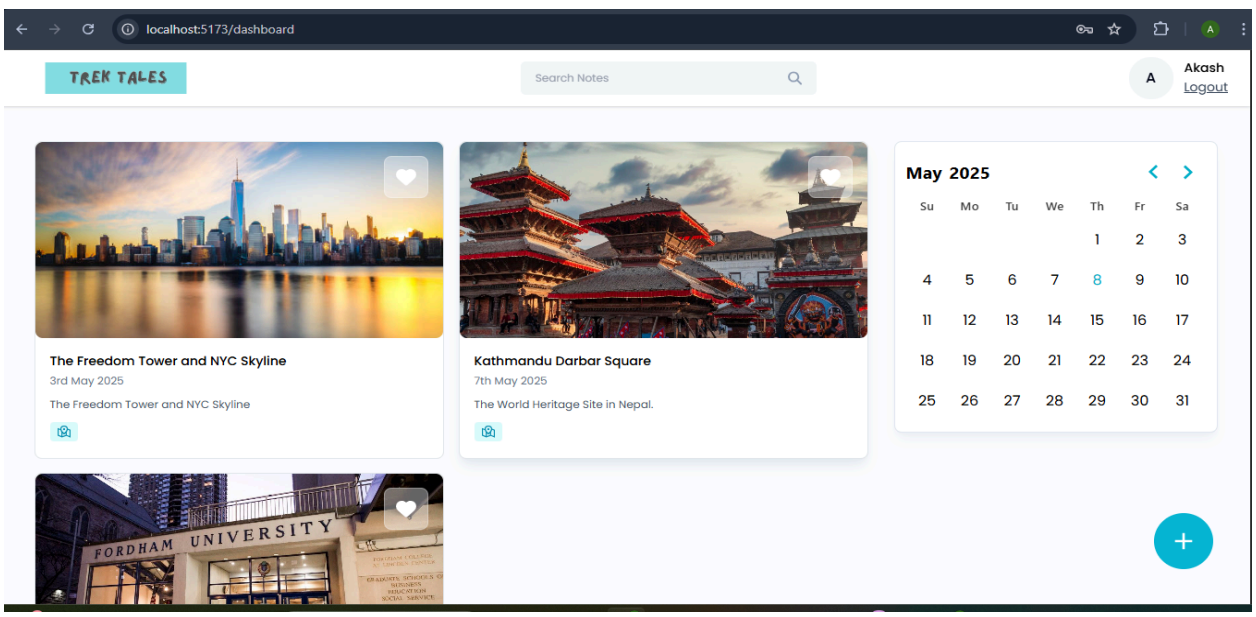
```
 => [backend] exporting to image
 => => exporting layers
 => => writing image sha256:c6abef5783314918dca0b6c1c239d5124cb7c7acc8e6ba6f2d7d65b8e46f1013
 => => naming to docker.io/library/trektale-backend
 => [frontend] resolving provenance for metadata file
 => [backend] resolving provenance for metadata file
[+] Running 4/4
 ✓ backend                        Built
 ✓ frontend                       Built
 ✓ Container trektale-frontend-1  Created
 ✓ Container trektale-backend-1   Created
Attaching to backend-1, frontend-1
frontend-1  |
frontend-1  | > trek-tale-app@0.0.0 dev
backend-1   |
frontend-1  | > vite
backend-1   | > travel-story-backend@1.0.0 start
frontend-1  |
backend-1   | > nodemon index.js
backend-1   | > nodemon index.js
frontend-1  |   ➜  Local:   http://localhost:5173/
backend-1   | [nodemon] to restart at any time, enter `rs`
backend-1   | [nodemon] watching path(s): *.*
backend-1   | [nodemon] watching extensions: js,mjs,cjs,json
backend-1   | [nodemon] starting `node index.js`

v View in Docker Desktop   o View Config   w Enable Watch
```



The application was tested locally:

Backend running on http://localhost:8000/

Frontend running on http://localhost:5173/login



7. PRODUCTION DEPLOYMENT ON GOOGLE CLOUD PLATFORM (GCP)

7.1. Prerequisites

- A Google Cloud Platform (GCP) account with billing enabled.
- gcloud CLI installed and authenticated.
- kubectl CLI installed.
- Docker Desktop is installed.

- A Docker Hub account for pushing images.

7.2. Building and Pushing Docker Images

Before deploying to GKE, production-ready Docker images for both frontend and backend were built and pushed to Docker Hub.

```
#!/bin/bash
docker build -t akash5468/trektale-backend:v1.6 . --no-cache
docker push akash5468/trektale-backend:v1.6
```

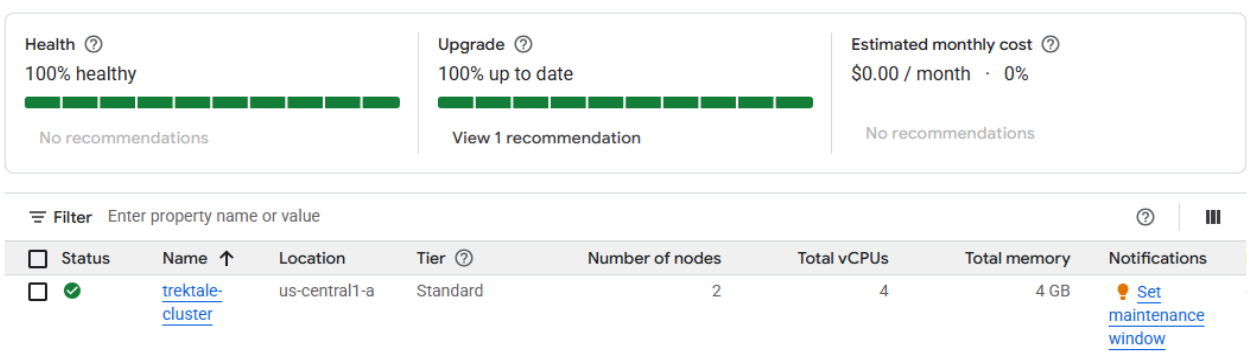
```
#!/bin/bash
docker build -t akash5468/trektale-frontend:v1.1 . --no-cache
docker push akash5468/trektale-frontend:v1.1
```

7.3. Google Kubernetes Engine (GKE) Setup

A GKE cluster named **trektale-cluster** was created using the following gcloud command:

```
C:\Users\Akash yadav\AppData\Local\Google\Cloud SDK>gcloud container clusters create trektale-cluster  --project=cisc5550-454815  --zone=us-central1
-a --num-nodes=2 --machine-type=e2-small
Note: The Kubelet readonly port (10255) is now deprecated. Please update your workloads to use the recommended alternatives. See https://cloud.googl
e.com/kubernetes-engine/docs/how-to/disable-kubelet-readonly-port for ways to check usage and for migration instructions.
Note: Your Pod address range (`--cluster-ipv4-cidr`) can accommodate at most 1008 node(s).
Creating cluster trektale-cluster in us-central1-a... Cluster is being health-checked (Kubernetes Control Plane is heal
thy)...done.
Created [https://container.googleapis.com/v1/projects/cisc5550-454815/zones/us-central1-a/clusters/trektale-cluster].
To inspect the contents of your cluster, go to: https://console.cloud.google.com/kubernetes/workload_/gcloud/us-central1-a/trektale-cluster?project=
cisc5550-454815
kubeconfig entry generated for trektale-cluster.
NAME             LOCATION       MASTER_VERSION      MASTER_IP      MACHINE_TYPE  NODE_VERSION        NUM_NODES  STATUS
trektale-cluster  us-central1-a  1.32.2-gke.1182003  34.134.146.77  e2-small      1.32.2-gke.1182003  2          RUNNING

C:\Users\Akash yadav\AppData\Local\Google\Cloud SDK>gcloud components install gke-gcloud-auth-plugin

All components are up to date.

C:\Users\Akash yadav\AppData\Local\Google\Cloud SDK>gcloud container clusters get-credentials trektale-cluster
Fetching cluster endpoint and auth data.
kubeconfig entry generated for trektale-cluster.

C:\Users\Akash yadav\AppData\Local\Google\Cloud SDK>gcloud container clusters get-credentials trektale-cluster --region=us-central1-a --project=cisc
5550-454815
Fetching cluster endpoint and auth data.
kubeconfig entry generated for trektale-cluster.
```

Cluster:



7.4. Kubernetes Manifest Files

Kubernetes manifest files (YAML) were created to define the desired state of the deployments, services, and configurations.

7.4.1. Backend Configuration (ConfigMap - configmap.yaml)

A ConfigMap named **trektale-config** was created from the backend's **.env** file to manage the MongoDB connection string and other environment variables securely.

```
# Ensure .env file is present in the current directory
kubectl delete configmap trektale-config --ignore-not-found
kubectl create configmap trektale-config --from-env-file=.env
```

7.4.2. Backend Deployment (backend-deployment.yaml)

This file defines the backend deployment, specifying the Docker image, replicas, ports, and how to use the ConfigMap.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: trektale-backend
  labels:
    app: trektale-backend
spec:
  replicas: 1
  selector:
    matchLabels:
      app: trektale-backend
  template:
    metadata:
      labels:
        app: trektale-backend
    spec:
      containers:
        - name: trektale-backend
          image: akash5468/trektale-backend:v1.6
          ports:
            - containerPort: 8000
          envFrom:
          - configMapRef:
              name: trektale-config
```

7.4.3. Backend Service (backend-service.yaml)

This file defines an internal service (ClusterIP) to expose the backend deployment within the GKE cluster. The frontend will use this service to communicate with the backend.

```yaml
apiVersion: v1
kind: Service
metadata:
  name: trektale-backend-service
spec:
  selector:
    app: trektale-backend
  ports:
    - port: 8000
      targetPort: 8000
  type: LoadBalancer
```

7.4.4. Frontend Deployment (frontend-deployment.yaml)

This file defines the frontend deployment.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: trektale-frontend
  labels:
    app: trektale-frontend
spec:
  replicas: 1
  selector:
    matchLabels:
      app: trektale-frontend
  template:
    metadata:
      labels:
        app: trektale-frontend
    spec:
      containers:
        - name: trektale-frontend
          image: akash5468/trektale-frontend:v1.1
          ports:
            - containerPort: 5173
```

```
          args: ["npm", "run", "dev", "--", "--host"]
```

7.4.5. Frontend Service (frontend-service.yaml)

This file defines an external service (LoadBalancer) to expose the frontend to the internet.

```yaml
apiVersion: v1
kind: Service
metadata:
  name: trektale-frontend-service
spec:
  selector:
    app: trektale-frontend
  ports:
    - port: 5173
      targetPort: 5173
  type: LoadBalancer
```

7.5. Deployment Script (deploy.sh)

For Backend:

```bash
#!/bin/bash
docker build -t akash5468/trektale-backend:v1.6 . --no-cache
docker push akash5468/trektale-backend:v1.6

# Ensure .env file is present in the current directory
kubectl delete configmap trektale-config --ignore-not-found
kubectl create configmap trektale-config --from-env-file=.env

kubectl apply -f backend-deployment.yaml
kubectl apply -f backend-service.yaml
kubectl get service trektale-backend-service
```

For Frontend:

```bash
#!/bin/bash
docker build -t akash5468/trektale-frontend:v1.1 . --no-cache
docker push akash5468/trektale-frontend:v1.1

kubectl apply -f frontend-deployment.yaml
kubectl apply -f frontend-service.yaml
kubectl get service trektale-frontend-service
```

```
Akash yadav@DESKTOP-FOPKH3J MINGW64 ~/Downloads/TrekTale/TrekTale/backend (main)
$ ./deploy.sh
[+] Building 8.0s (11/11) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 220B
 => [internal] load metadata for docker.io/library/node:18-alpine
 => [auth] library/node:pull token for registry-1.docker.io
 => [internal] load .dockerignore
 => => transferring context: 99B
 => [1/5] FROM docker.io/library/node:18-alpine@sha256:8d6421d663b4c28fd3ebc498332f249011d118945588d0a35cb9bc4b8ca09d9e
 => [internal] load build context
 => => transferring context: 65.71kB
 => CACHED [2/5] WORKDIR /backend
 => [3/5] COPY package*.json ./
 => [4/5] RUN npm install
 => [5/5] COPY . .
 => exporting to image
 => => exporting layers
 => => writing image sha256:a6a0dfc1a7413e7ffb8a56eb3d91d6654e08a774b11e85ad8f3a4268440a7a9d
 => => naming to docker.io/akash5468/trektale-backend:v1.5

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/tzknv7t5f5f7x5zibyy5fw0md
The push refers to repository [docker.io/akash5468/trektale-backend]
8569b5caec9d: Pushed
456b90feaedb: Pushed
55bd5800e21d: Pushed
ad6826c8618a: Layer already exists
82140d9a70a7: Layer already exists
f3b40b0cdb1c: Layer already exists
0b1f26057bd0: Layer already exists
08000c18d16d: Layer already exists
v1.5: digest: sha256:d202eb9206854e415621cd2c1ff7bec4d2da5e4ee71f9ce170ece4fdce512542 size: 1995
configmap "trektale-config" deleted
configmap/trektale-config created
deployment.apps/trektale-backend created
service/trektale-backend-service unchanged
NAME                        TYPE           CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
trektale-backend-service    LoadBalancer   34.118.229.135   34.10.82.81   8000:32252/TCP   13h
```

```
Akash yadav@DESKTOP-FOPKH3J MINGW64 ~/Downloads/TrekTale/TrekTale/frontend (main)
$ ./deploy.sh
[+] Building 53.6s (11/11) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 242B
 => [internal] load metadata for docker.io/library/node:18-alpine
 => [internal] load .dockerignore
 => => transferring context: 77B
 => [1/6] FROM docker.io/library/node:18-alpine@sha256:8d6421d663b4c28fd3ebc498332f249011d118945588d0a35cb9bc4b8ca09d9e
 => [internal] load build context
 => => transferring context: 2.60kB
 => CACHED [2/6] WORKDIR /frontend
 => [3/6] COPY package*.json ./
 => [4/6] RUN npm install
 => [5/6] COPY . .
 => [6/6] RUN npm run build
 => exporting to image
 => => exporting layers
 => => writing image sha256:5f054907d56db69f1419ea23e2146caf6821ad6e6e765e88f4fe849531968d9c
 => => naming to docker.io/akash5468/trektale-frontend:v1.1

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/ntrh589v79kkgwmgff5043a4l
The push refers to repository [docker.io/akash5468/trektale-frontend]
73736872624c: Pushed
ace0dc4cb27a: Pushed
f3eba23b7a92: Pushed
d482e92a38c2: Pushed
1199f186113f: Layer already exists
82140d9a70a7: Layer already exists
f3b40b0cdb1c: Layer already exists
0b1f26057bd0: Layer already exists
08000c18d16d: Layer already exists
v1.1: digest: sha256:898045336ebc066b6f32efa99c741d54c71aa920e669b393bc1925d23e76e9fd size: 2206
deployment.apps/trektale-frontend unchanged
service/trektale-frontend-service unchanged
NAME                         TYPE           CLUSTER-IP       EXTERNAL-IP    PORT(S)          AGE
trektale-frontend-service    LoadBalancer   34.118.235.174   35.188.97.99   5173:30711/TCP   13h
```

7.6. Deploying to GKE and Verifying Workloads

The **deploy.sh** script was executed to deploy all resources to the GKE cluster.

Workloads Refresh Deploy Create Job Delete Learn

Cluster Namespace Reset Save

Overview Observability Cost Optimization

Filter Is system object : False Filter workloads

Name	Status	Type	Pods	Namespace	Cluster
trektale-backend	OK	Deployment	1/1	default	trektale-cluster
trektale-frontend	OK	Deployment	1/1	default	trektale-cluster

The status of the pods was checked:

Backend Pod:

Managed pods

Revision	Name	Status	Restarts	Created on
2	trektale-backend-79b8977d88-nwgcd	Running	0	May 8, 2025, 3:33:46 PM

Exposing services

Name	Type	Endpoints
trektale-backend-service	Load balancer	34.10.82.81:8000

Frontend Pod:

Managed pods

Revision	Name	Status	Restarts	Created on
1	trektale-frontend-76f8b4d5cf-j5npq	Running	0	May 8, 2025, 9:26:19 AM

Exposing services

Name	Type	Endpoints
trektale-frontend-service	Load balancer	35.188.97.99:5173

7.7. Accessing the Deployed Application

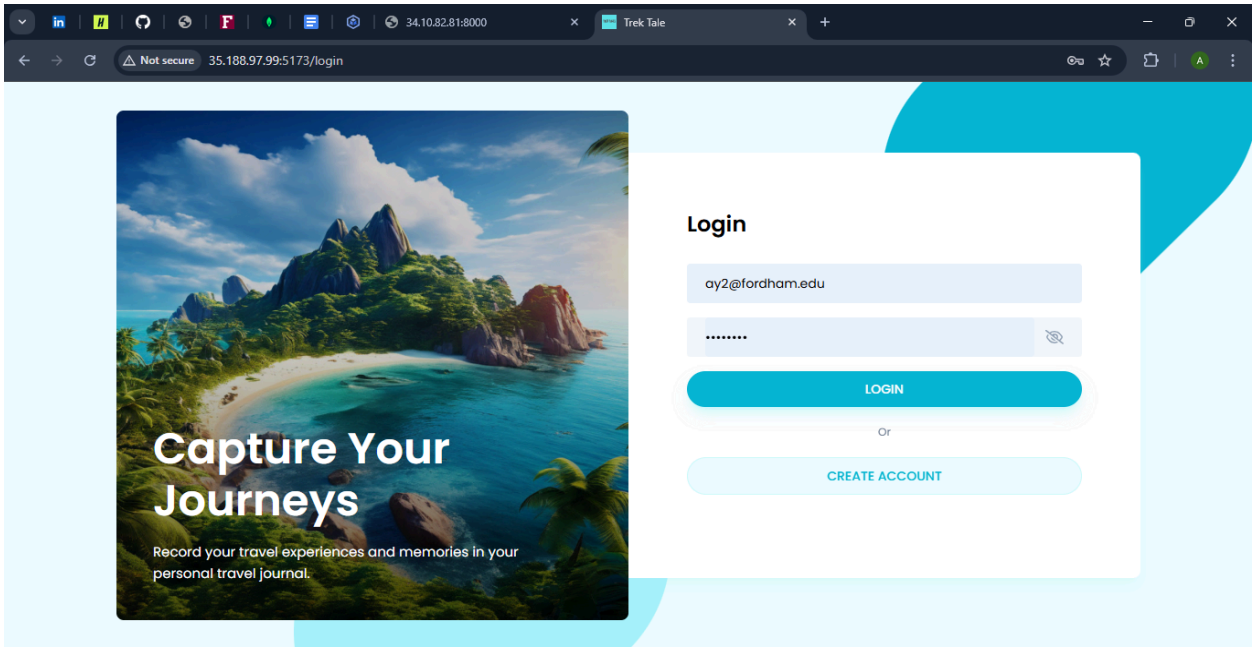
To access the deployed TrekTale application, the external IP address of the **trektale-frontend-service** (LoadBalancer) was obtained:

```
kubectl get service trektale-frontend-service
```

The application is then accessible at http://35.188.97.99:5173/login

The Backend is accessible at http://34.10.82.81:8000/

For direct login, you can use: Username = ram@gmail.com, Password = ram123



8. TROUBLESHOOTING AND RESOLVED ISSUES

Issue Faced:

While deploying the backend to GCP, it was getting deployed, but I was encountering an issue. Upon checking each pod properly, I analyzed that the **config.json** file (which initially held the MongoDB connection string) was not being included or correctly accessed during deployment. This file is crucial as it contains the MongoDB connection strings necessary for communication with the database.

Resolution:

The configuration issue was handled by:

1. Moving the MongoDB connection string and other sensitive/environment-specific configurations from config.json to an .env file in the backend's root directory.
2. Creating a Kubernetes ConfigMap named trektale-config directly from this .env file using the following kubectl commands:

```
3. kubectl delete configmap trektale-config --ignore-not-found
4. kubectl create configmap trektale-config --from-env-file=.env
```

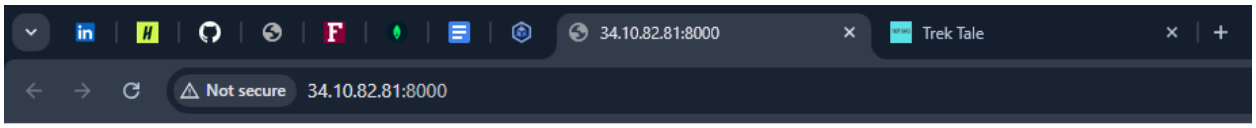
3. Updating the backend Kubernetes deployment (backend-deployment.yaml) to mount these environment variables from the trektale-config ConfigMap using envFrom.configMapRef.

```
C:\Users\Akash yadav\AppData\Local\Google\Cloud SDK>kubectl get svc
NAME                      TYPE           CLUSTER-IP       EXTERNAL-IP    PORT(S)          AGE
kubernetes                ClusterIP      34.118.224.1     <none>         443/TCP          14h
trektale-backend-service  LoadBalancer   34.118.229.135   34.10.82.81    8000:32252/TCP   13h
trektale-frontend-service LoadBalancer   34.118.235.174   35.188.97.99   5173:30711/TCP   13h
```

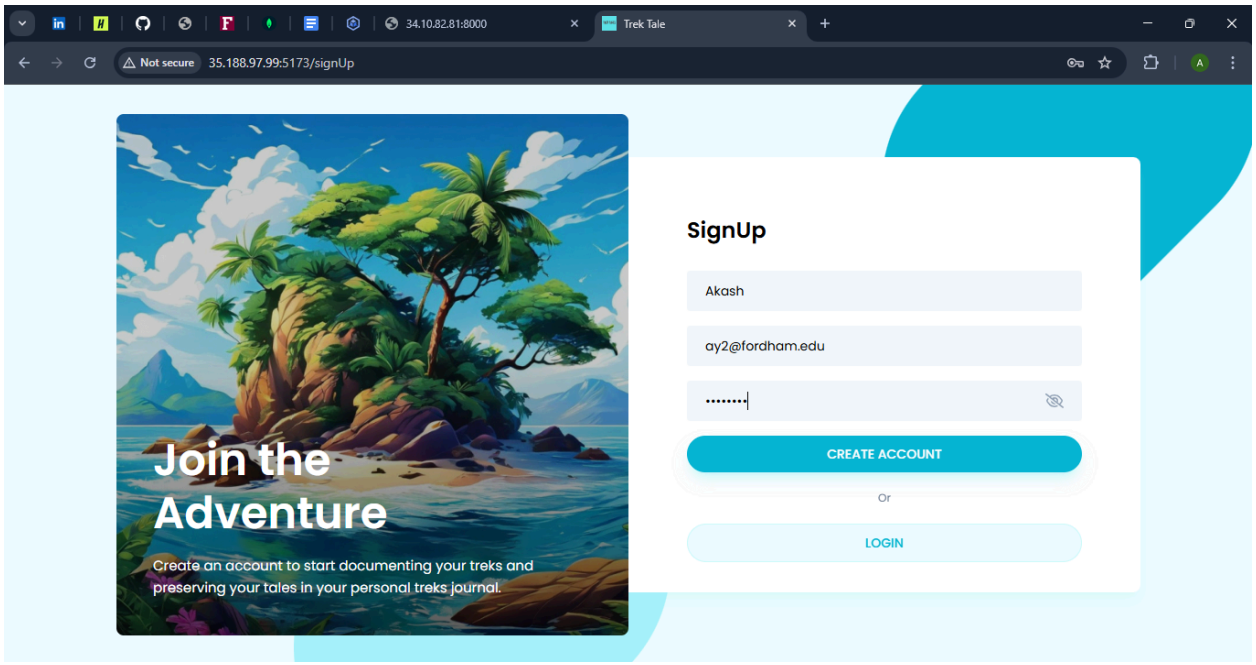
```
C:\Users\Akash yadav\AppData\Local\Google\Cloud SDK>kubectl logs trektale-backend-55b7448d8d-f8jzf

> travel-story-backend@1.0.0 start
> nodemon index.js

[nodemon] 3.1.10
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node index.js`
node:internal/modules/cjs/loader:1143
  throw err;
  ^

Error: Cannot find module './config.json'
Require stack:
- /backend/index.js
    at Module._resolveFilename (node:internal/modules/cjs/loader:1140:15)
    at Module._load (node:internal/modules/cjs/loader:981:27)
    at Module.require (node:internal/modules/cjs/loader:1231:19)
    at require (node:internal/modules/helpers:177:18)
    at Object.<anonymous> (/backend/index.js:3:16)
    at Module._compile (node:internal/modules/cjs/loader:1364:14)
    at Module._extensions..js (node:internal/modules/cjs/loader:1422:10)
    at Module.load (node:internal/modules/cjs/loader:1203:32)
    at Module._load (node:internal/modules/cjs/loader:1019:12)
    at Function.executeUserEntryPoint [as runMain] (node:internal/modules/run_main:128:12) {
  code: 'MODULE_NOT_FOUND',
  requireStack: [ '/backend/index.js' ]
}

Node.js v18.20.8
[nodemon] app crashed - waiting for file changes before starting...
```

This approach ensures that the backend pods receive the necessary configuration at runtime in a secure and Kubernetes-native way.

Finally, with this fix, both the Frontend and Backend were successfully deployed and functioning correctly on Google Cloud Platform.



TrekTale backend is running 🚀



9. CONCLUSION

This project successfully modernized the TrekTale application by transitioning it to a cloud-native architecture. Key achievements include:

- Splitting the application into individual frontend and backend services.
- Containerizing both services using Docker for reproducible environments and portability.
- Creating a solid local development workflow using Docker Compose.
- Deploy the application on Google Kubernetes Engine (GKE), utilizing its orchestration features for management and scaling.
- Migrating data persistence to a cloud-hosted solution (MongoDB Atlas).
- Practicing secure configuration management using ConfigMaps in Kubernetes.

The project provided valuable hands-on experience with a range of cloud technologies and demonstrated the benefits of a microservices-oriented, containerized deployment strategy.