Task 10.2HD – Building a Cloud-Native Application

Github link:
https://github.com/HaydenDuong/SIT323_Cloud_Native_Application_Development/tree/main/Student%20Task%20Manager

I. Tools:

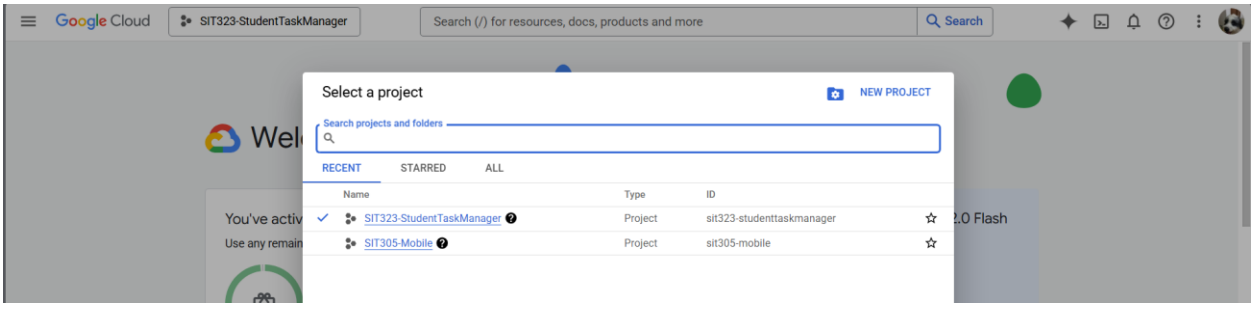
- Node.js
- Visual Studio Code
- Docker
- Kubernetes
- Kubectl
- Google Cloud Platform Services

II. Steps:

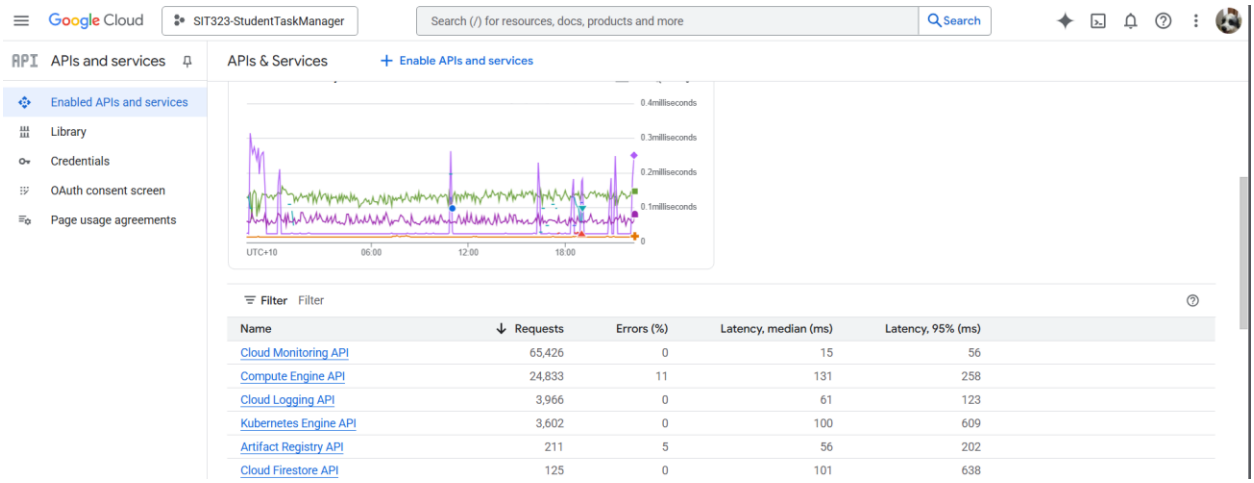
A/ Build a cloud-native web application using Node.js

B/ Set up a GCP account, create a project, and enable the revelant APIs

- For this part, I used by my own personal Gmail account to setup a project for this task and named it as SIT323-StudentTaskManager

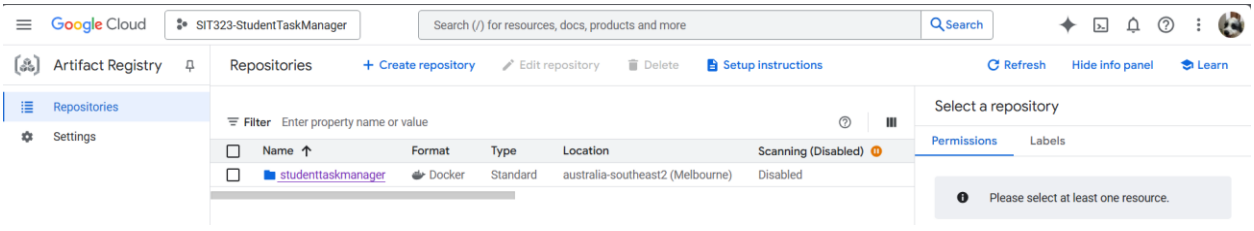


- For this task, I enabled the following APIs:
    - Artifact Registry – for pushing the Docker Image of my application to this repo because I learned that Google Container Registry (GCR) is deprecated and the read access to existing images is ending after May 20, 2025 (today). Thus, AR is a better choice.
    - Kubernetes Engine – for create a k8s cluster on GCP and deploying nodes.
    - Cloud Firestore – for storing data (task information) from my web application Student Task Manager.



C/ Use Artifact Registry (AR) to store the Docker image of my application

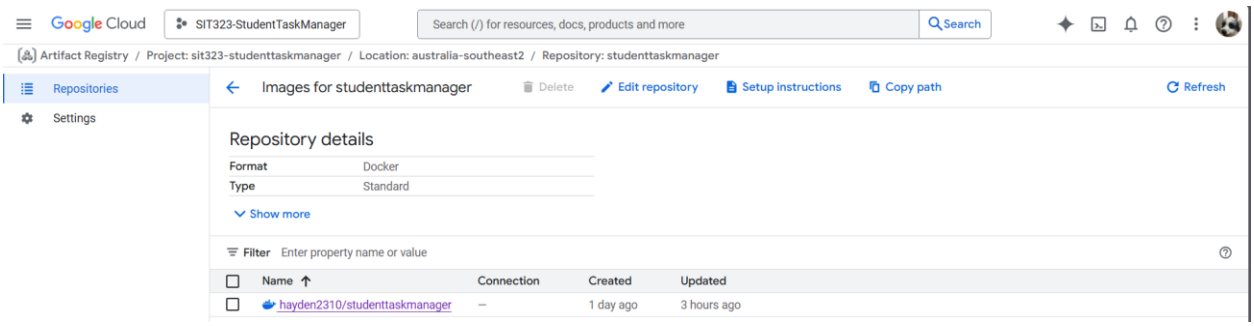
- Log in GCP via Google Cloud SDK through command: `gcloud auth login`
- Enter my personal Gmail account in the pop-up Google login page.
- Enter command: `gcloud config get-value project` – to make sure the SDK is "pointing" to the right project.
- (Optional) Enter command: `gcloud project lists` – to see all current projects exist in my Gmail account.
- If not, I will use command: `gcloud config set project <project_id>` to change to current unwanted project to the correct one.
- Then I created an AR repository with the following choices:
    - Name: "studenttaskmanager"
    - Format: "Docker"
    - Mode: "Standard"
    - Location Type: "Region"
    - Region: "australia-southeast2 (Melbourne)"
    - Leave other selections as their initial state and click "Create" button.

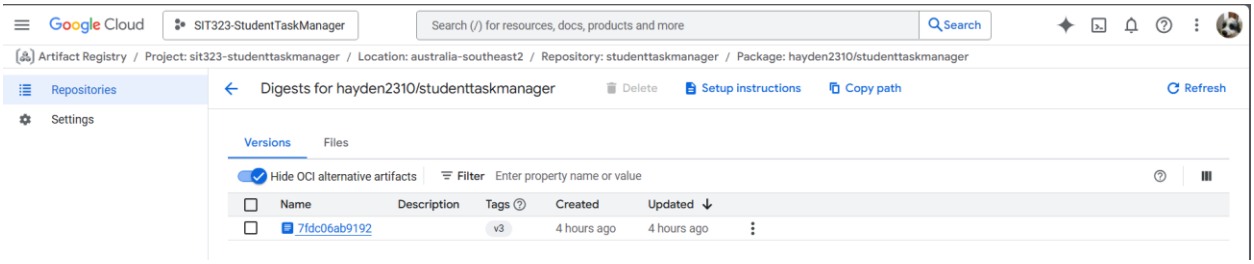


- Grant the permission of push and pull image from Docker Hub to this AR Repo by command: `gcloud auth config-docker <selected-region>-docker.pkg.dev`

- After created an Docker image for my application, I used the following command: `docker tag <image-name> <dockerhub-username>/<image-name>:<tag>` - to create a valid Docker image that can be pushed to Docker Hub.
- In the same manner, I created a new Docker image based on that Docker-valid image to push to AR repo through this command:

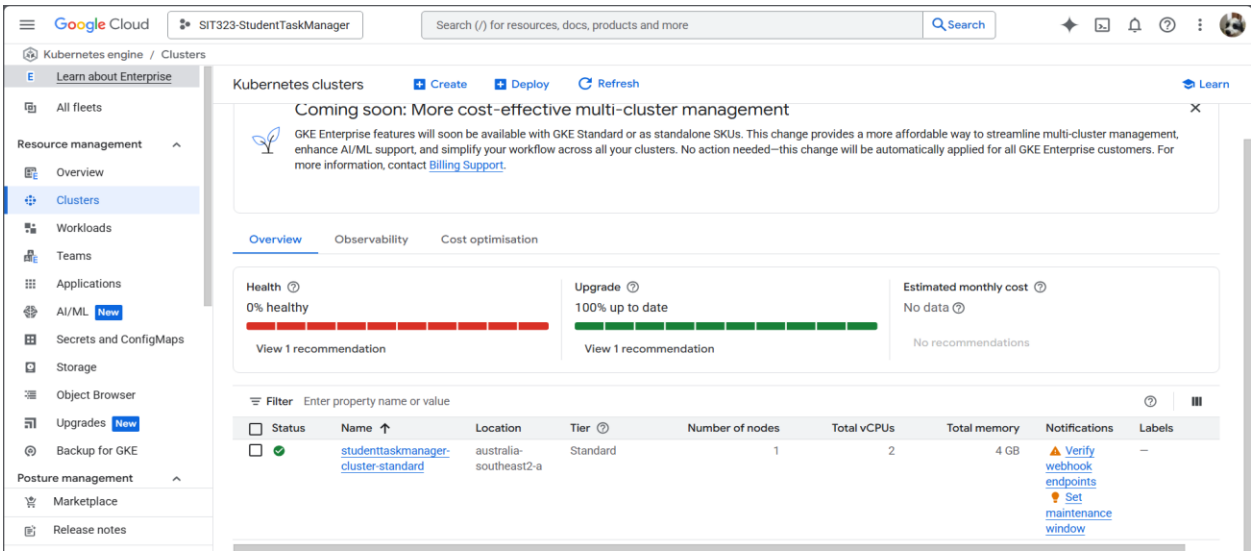
```
docker tag <image_name> <region_selected>-
docker.pkg.dev/<project_id>/<AR_Repo_name>/<image_name>:<tag>
```





D/ Create a Google Kubernetes Engine (GKE) cluster

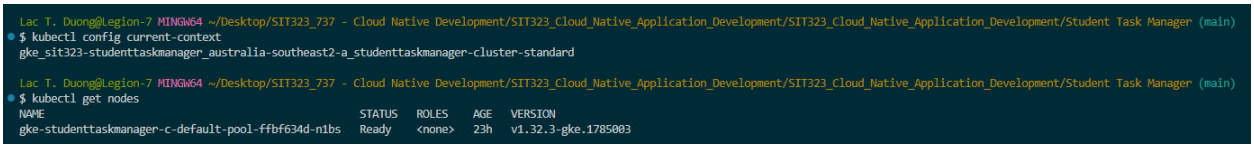
- I created a standard cluster with the following criteria:
  - Name: studenttaskmanager-cluster-standard
  - Location type: Zonal
  - Zone: australia-southeast2-a
  - In “default-pool”: Number of nodes = 1
  - In “Nodes”: Machine type = e2-medium (2 vCPU, 1 core, 4 GB memory)
    Boot disk size (GB) = 100



- I entered this command into Google Cloud SDK:

Gcloud container cluster get-credentials studenttaskmanager-cluster-standard –region australia-southeast2a –project sit323-studenttaskmanager

- In VSCode, I entered these two kubectl commands to check if my machine is correctly connected to the right cluster as well as checking the status of the node



- Because I specified for only 1 node and chose Zonal instead of Region, thus, only 1 node was generated.

E/ Make a deployment for the Docker Image in the generated Kubernetes cluster

- I created both the deployment.yaml and service.yaml to deploy my application to this K8s cluster, codes as follows:

Deployment.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: studenttaskmanager-deployment
  labels:
    app: studenttaskmanager
spec:
  replicas: 1
```

```
  selector:
    matchLabels:
      app: studenttaskmanager
  template:
    metadata:
      labels:
        app: studenttaskmanager
    spec:
      serviceAccountName: studenttaskmanager-ksa
      containers:
        - name: studenttaskmanager-container
          image: australia-southeast2-docker.pkg.dev/sit323-
studenttaskmanager/studenttaskmanager/hayden2310/studenttaskmanager:v3
          ports:
            - containerPort: 8080
          resources:
            requests:
              memory: "128Mi"
              cpu: "100m"
```

Service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: studenttaskmanager-service
  labels:
    app: studenttaskmanager
spec:
  selector:
    app: studenttaskmanager
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
  type: LoadBalancer
```

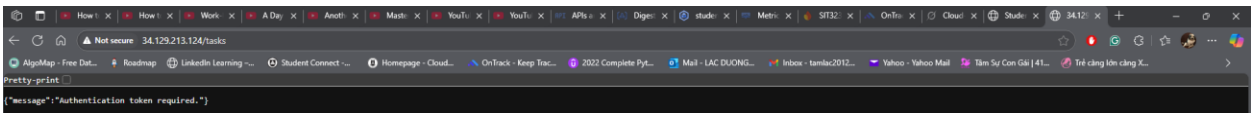
- Then I applied these two files with command: `kubectl apply -f <path/file-name>.yaml`
- Entered command: `kubectl get all`

```
$ kubectl get all
NAME                                                  READY   STATUS    RESTARTS   AGE
pod/studenttaskmanager-deployment-55f8fc9cc7-kr77d    1/1     Running   0          125m

NAME                                  TYPE           CLUSTER-IP       EXTERNAL-IP      PORT(S)        AGE
service/kubernetes                    ClusterIP      34.118.224.1     <none>           443/TCP        23h
service/studenttaskmanager-service    LoadBalancer   34.118.235.157   34.129.213.124   80:31948/TCP   23h

NAME                                             READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/studenttaskmanager-deployment    1/1     1            1           22h

NAME                                                        DESIRED   CURRENT   READY   AGE
replicaset.apps/studenttaskmanager-deployment-55f8fc9cc7    1         1         1       125m
```

- (Optional) check which node does my pod is running at through command: kubectl describe pod <pod-name>

```
$ kubectl describe pod studenttaskmanager-deployment-55f8fc9cc7-kr77d
Name:             studenttaskmanager-deployment-55f8fc9cc7-kr77d
Namespace:        default
Priority:         0
Service Account:  studenttaskmanager-ksa
Node:             gke-studenttaskmanager-c-default-pool-ffbf634d-n1bs/10.192.0.6
Start Time:       Tue, 20 May 2025 18:32:42 +1000
Labels:           app=studenttaskmanager
                  pod-template-hash=55f8fc9cc7
Annotations:      <none>
Status:           Running
IP:               10.40.0.14
IPs:
  IP:           10.40.0.14
Controlled By:  ReplicaSet/studenttaskmanager-deployment-55f8fc9cc7
Containers:
  studenttaskmanager-container:
    Container ID:   containerd://d47f4b9c727002d4cce840a3ac98e6d039c43a8beac3b07753a88bc29e8cc212
    Image:          australia-southeast2-docker.pkg.dev/sit323-studenttaskmanager/studenttaskmanager/hayden2310/studenttaskmanager:v3
    Image ID:       australia-southeast2-docker.pkg.dev/sit323-studenttaskmanager/studenttaskmanager/hayden2310/studenttaskmanager@sha256:7fdc06ab9192caf185d168bbb2cb3401e9e4a4265f6229b7a2f22680fbc6fe16
    Port:           8080/TCP
    Host Port:      0/TCP
    State:          Running
      Started:      Tue, 20 May 2025 18:32:51 +1000
    Ready:          True
    Restart Count:  0
    Requests:
    Container ID:   containerd://d47f4b9c727002d4cce840a3ac98e6d039c43a8beac3b07753a88bc29e8cc212
    Image:          australia-southeast2-docker.pkg.dev/sit323-studenttaskmanager/studenttaskmanager/hayden2310/studenttaskmanager:v3
    Image ID:       australia-southeast2-docker.pkg.dev/sit323-studenttaskmanager/studenttaskmanager/hayden2310/studenttaskmanager@sha256:7fdc06ab9192caf185d168bbb2cb3401e9e4a4265f6229b7a2f22680fbc6fe16
    Port:           8080/TCP
    Host Port:      0/TCP
    State:          Running
      Started:      Tue, 20 May 2025 18:32:51 +1000
    Ready:          True
    Restart Count:  0
    Requests:
    Image:          australia-southeast2-docker.pkg.dev/sit323-studenttaskmanager/studenttaskmanager/hayden2310/studenttaskmanager:v3
    Image ID:       australia-southeast2-docker.pkg.dev/sit323-studenttaskmanager/studenttaskmanager/hayden2310/studenttaskmanager@sha256:7fdc06ab9192caf185d168bbb2cb3401e9e4a4265f6229b7a2f22680fbc6fe16
    Port:           8080/TCP
    Host Port:      0/TCP
    State:          Running
      Started:      Tue, 20 May 2025 18:32:51 +1000
    Ready:          True
    Restart Count:  0
    Requests:
    Image ID:       australia-southeast2-docker.pkg.dev/sit323-studenttaskmanager/studenttaskmanager/hayden2310/studenttaskmanager@sha256:7fdc06ab9192caf185d168bbb2cb3401e9e4a4265f6229b7a2f22680fbc6fe16
    Port:           8080/TCP
    Host Port:      0/TCP
```

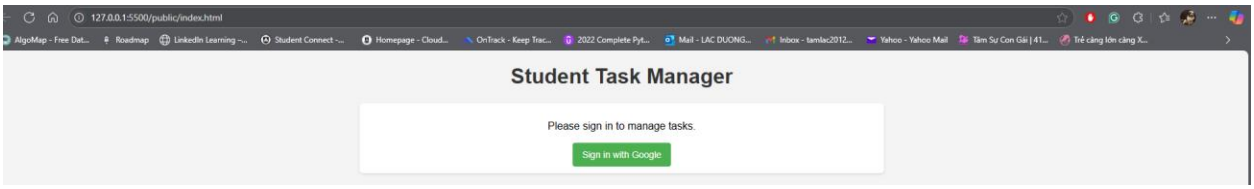
- Because my app had integrated with Oauth authentication, thus, access to the web application via LoadBalancer Service would yield this result:



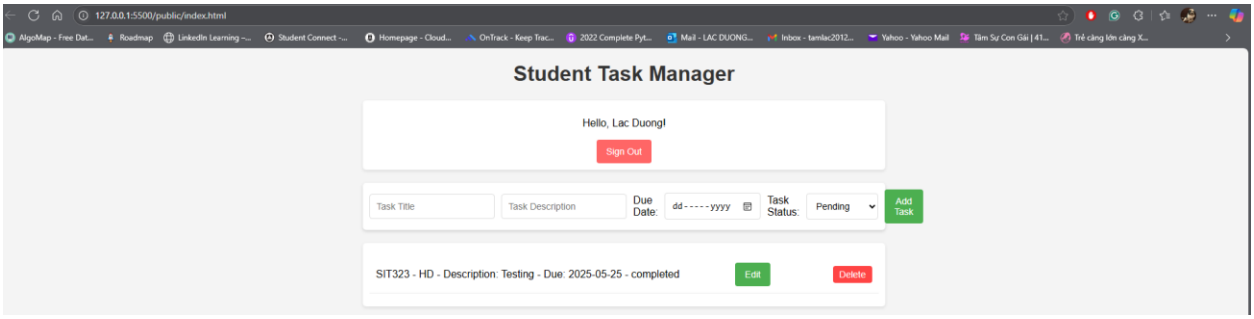
- This is because: the API is secured and expects a Firebase ID token, which is only provided by the frontend application after successful user authentication.
- The frontend (HTML/JavaScript) is currently served using a local development server (e.g., at http://127.0.0.1:5500). This local frontend then communicates with the

GKE-hosted backend API. The GKE deployment is not yet configured to serve the static frontend files, which is why the application is not directly accessible through the LoadBalancer's external IP address for viewing the user interface.

- Thus, I accessed my web application by right-click “index.html” and choose “Open with Live Server” which will directly me to the login page, where I must login via Gmail account, to access to the main / dashboard page



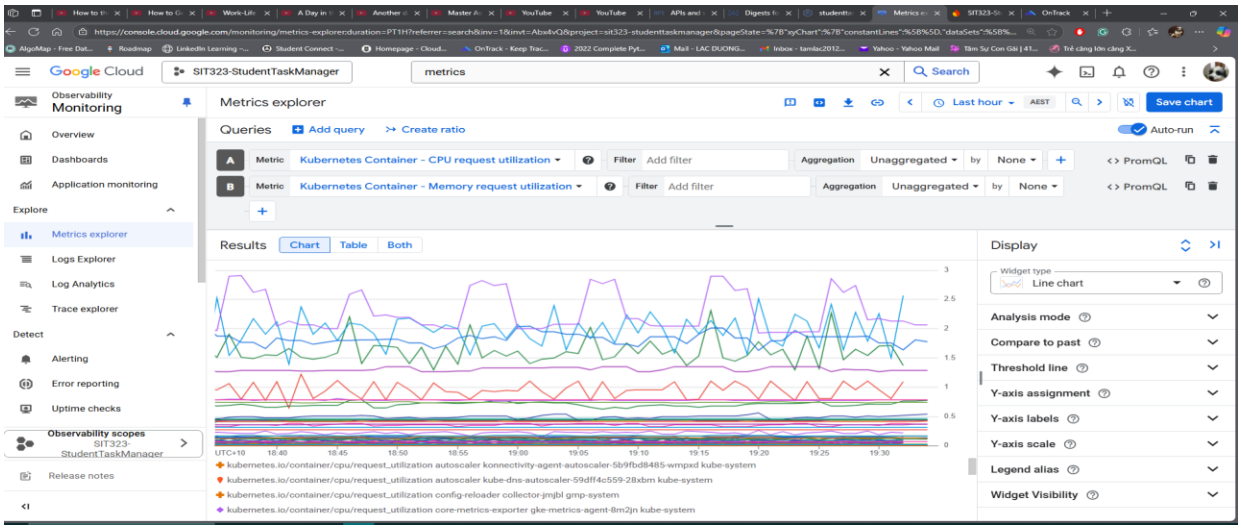
Picture 1 – Login page



Picture 2 – Main / Dashboard page

F/ Configure monitoring and logging for my application with GCP tools

- Navigated to Observability Monitoring on GCP.
- I created two queries (similar to task 10.1P) by entered these two:
  - kubernetes.io/container/cpu/request_utilization
  - kubernetes.io/container/memory/request_utilization
- Result:



Challenges / Issues:

GKE, K8s -Side

1. Pod Stuck in Pending State Due to Insufficient CPU Resources

Symptoms Observed:

- Upon applying the deployment.yaml using kubectl apply -f k8s/deployment.yaml, the command kubectl get pods consistently showed the application pod in a Pending state.
- Further investigation using kubectl describe pod <pod-name> revealed critical information in the Events section. Events with Type: Warning, Reason: FailedScheduling indicated that the Kubernetes scheduler was unable to assign the pod to any available node. The message associated with this event was specific: 0/1 nodes are available: 1 Insufficient cpu.
- An additional event, Type: Normal, Reason: NotTriggerScaleUp, with the message pod didn't trigger scale-up: 1 Insufficient cpu, suggested that even the cluster autoscaler did not deem it appropriate to add more nodes or larger nodes based on this single pod's request.

Diagnostic Steps:

1. Reviewed the resources.requests.cpu value in k8s/deployment.yaml. It was initially set to 250m (0.25 vCPU).

2. Examined the GKE node's capacity using kubectl get nodes and then kubectl describe node <node-name>. This showed that the node (an e2-small instance with 2 vCPUs total) had an Allocatable CPU capacity of 940m. While 250m is less than 940m, the Insufficient cpu error implied that either other system pods were consuming a portion of the allocatable CPU, or the requested amount, combined with potential CPU fragmentation, prevented scheduling.

Root Cause Analysis:

The primary cause was that the CPU resources requested by the pod (250m) exceeded the *effectively available and schedulable* CPU on the sole GKE node at the time of scheduling. GKE nodes reserve a portion of their total capacity for the Kubernetes control plane components (like kubelet, kube-proxy) and the underlying operating system. The remaining "allocatable" resources are available for user pods. The scheduler could not guarantee the 250m CPU requested amidst the existing load and reservations.

Solution Implemented:

The cpu resource request in the k8s/deployment.yaml file, under spec.template.spec.containers.resources.requests, was adjusted downwards.

- Initial change: Modified from 250m to 100m.
- Verification: The updated deployment.yaml was applied using kubectl apply -f k8s/deployment.yaml.
- Outcome: The Kubernetes scheduler successfully found sufficient resources for the 100m CPU request, and the pod transitioned from Pending to ContainerCreating and then eventually to Running (though it subsequently hit another issue). This confirmed that the resource request was the primary blocker for scheduling.

2. Pod Crashing with CrashLoopBackOff Due to Missing Service Account Key File

Symptoms Observed:

- Once the scheduling issue was resolved, kubectl get pods showed the pod transitioning to Running briefly, but then quickly changing to CrashLoopBackOff or Error. The RESTARTS count for the pod incremented repeatedly.

- The command kubectl logs <pod-name> was used to retrieve the application logs from the failing container. The logs clearly displayed a startup error: Error: Cannot find module './sit323-studenttaskmanager-43b726fc0e9a.json'. The stack trace indicated this error originated from the line in index.js where the Firebase Admin SDK was being initialized with this JSON key file.

Diagnostic Steps:

1. Reviewed index.js to confirm the require('./sit323-studenttaskmanager-43b726fc0e9a.json') statement.
2. Checked the Dockerfile to see how the application source code was copied (COPY . .).
3. Checked the .dockerignore file, which correctly listed sit323-studenttaskmanager-43b726fc0e9a.json. This confirmed the key file was intentionally and correctly excluded from the Docker image build process.

Root Cause Analysis:

The application was designed for local development to use a downloaded JSON service account key for Firebase authentication. However, for security and best practices in containerized environments (especially GKE), service account keys should not be bundled into Docker images. The .dockerignore ensured this, but the application code was not yet adapted to run in an environment (like GKE with Workload Identity) where credentials would be provided ambiently. The application crashed because it couldn't find the expected key file at runtime inside the container.

Solution Implemented:

A multi-step approach was taken to address this, preparing for Workload Identity:

1. **Code Modification (index.js):** The explicit loading of the JSON key file was removed. The Firebase Admin SDK initialization was changed to admin.initializeApp({ projectId: 'sit323-studenttaskmanager' });. This allows the SDK to automatically detect and use ambient credentials when available (e.g., from Workload Identity).
2. **New Docker Image (:v2):** A new version of the Docker image (student-task-manager:v2) was built using docker build -t student-task-manager:v2 . to include the updated index.js.
3. **Push to Artifact Registry:** The new :v2 image was tagged and pushed to the designated Google Artifact Registry repository.

4. **Deployment Update:** The k8s/deployment.yaml was modified to reference the new image tag (…/student-task-manager:v2).

- Outcome: After applying these changes, the pod no longer crashed due to the missing file. It reached a Running state, although subsequent Firestore operations would initially fail due to lack of authenticated identity, which was the next problem to solve.

3. API Calls Failing with 7 PERMISSION_DENIED: Request had insufficient authentication scopes

Symptoms Observed:

- With the pod running (after fixing the missing module error), attempts to access API endpoints that interacted with Firestore (e.g., GET /tasks via Postman) resulted in a 500 Internal Server Error from the application.
- The application logs, retrieved using kubectl logs <pod-name>, showed a clear error from the Firebase Admin SDK: Error: 7 PERMISSION_DENIED: Request had insufficient authentication scopes. The error details specified reason: 'ACCESS_TOKEN_SCOPE_INSUFFICIENT' and service: 'firestore.googleapis.com'.

Diagnostic Steps:

1. **Workload Identity Configuration Verification:**

- Confirmed a Kubernetes Service Account (KSA), student-task-manager-ksa, was created.
- Verified the Google Service Account (GSA), student-task-manager-dev@…, had the Cloud Datastore User role in IAM.
- Ensured the gcloud iam service-accounts add-iam-policy-binding command was correctly run to link the KSA to the GSA with the roles/iam.workloadIdentityUser role.
- Confirmed the KSA was annotated using kubectl annotate serviceaccount … iam.gke.io/gcp-service-account=YOUR_GSA_EMAIL.
- Checked that k8s/deployment.yaml specified serviceAccountName: student-task-manager-ksa for the pod template.

All these steps for Workload Identity setup appeared correct.

2. **IAM Permission vs. OAuth Scopes:** The error message specifically mentioned "scopes," not just general permissions. This indicated that while the GSA *had permission* to access Firestore, the access token being used by the application (obtained via Workload Identity acting as the GSA) was not minted with the *scope* required for the firestore.googleapis.com service.

Root Cause Analysis:

GKE nodes are provisioned with a set of default OAuth 2.0 scopes that their node service account can use when generating access tokens for Google Cloud APIs. If these default scopes are too restrictive and do not include https://www.googleapis.com/auth/datastore (or the broader https://www.googleapis.com/auth/cloud-platform), then even if Workload Identity correctly impersonates a GSA with the right IAM roles, the access tokens generated might still lack the required scope for certain services. This was the case: the original node pool was created without explicitly providing sufficient scopes.

Solution Implemented:
The GKE node pool (default-pool) was recreated with the necessary OAuth scopes:

1. **Set gcloud context:** Project, zone, and cluster were set as defaults for subsequent gcloud commands.
2. **Node Pool Deletion:** The existing default-pool was deleted using gcloud container node-pools delete default-pool --quiet. This temporarily removed the compute capacity for the application.
3. **Node Pool Recreation with Scopes:** A new default-pool was created with the same name but now including the --scopes "https://www.googleapis.com/auth/cloud-platform" flag. This flag ensures the node's service account can request tokens with all necessary scopes for Google Cloud services, including Firestore. The machine type was also upgraded from e2-small to e2-medium during this recreation for better resource availability.

Verification: After the new node pool became active, the studenttaskmanager-deployment automatically scheduled its pod onto the new node.

Outcome: API calls made to the application's external IP address via Postman (e.g., GET /tasks) now returned a 200 OK status with data successfully retrieved from Firestore. The "insufficient authentication scopes" error was resolved, confirming that the broader scopes on the node pool enabled Workload Identity to function correctly for Firestore access.

B/ Application-Side

1/ Firebase Auth Functions Not Available in script.js

Symptoms:

- When the page loaded or when "Sign in with Google" was clicked, the browser console showed errors like:
- Firebase onAuthStateChanged or auth is not available. Auth state will not be monitored.
- Firebase Auth functions not available. Check Firebase setup in index.html.
- An alert box appeared: "Authentication service is not ready. Please try again later."
- The UI displayed "Authentication service error. Check console."

Root Cause:

- The main application script (public/script.js) was attempting to use Firebase authentication functions (e.g., window.firebaseAuth, window.onAuthStateChanged) *before* the Firebase initialization script in public/index.html (which defines these on the window object) had a chance to execute and make them available. This was due to the relative order and loading/execution behavior of the <script> tags.

Resolution:

- The order of the script tags in public/index.html was adjusted.
- The <script type="module">...</script> block responsible for initializing Firebase and setting up window.firebaseAuth, etc., was moved to be **before** the <script src="script.js" defer></script> tag.
- This ensured that by the time script.js executed, the necessary Firebase functions and objects were already defined and accessible on the window object.

- A try-catch block was also added around the Firebase initialization in index.html for more robust error reporting if the core SDK itself failed to load or initialize.

2/ auth/unauthorized-domain Error on Google Sign-In

Symptoms:

- After fixing the script loading order, clicking "Sign in with Google" resulted in:
- A Firebase error popup/alert in the browser: Sign-in failed: Firebase: Error (auth/unauthorized-domain).
- A new browser popup window appearing briefly with a message like "The requested action is invalid" or similar, on a *.firebaseapp.com/__/auth/handler?... URL.
- Console errors indicating that the current domain was not authorized for OAuth operations.

Root Cause:

- Firebase Authentication, for security reasons, requires that the domain from which an authentication request originates (especially for OAuth providers like Google using popups/redirects) be explicitly whitelisted in the Firebase project settings.
- When running public/index.html directly from the local file system (e.g., file:///C:/...), the browser often treats the origin for network requests as 127.0.0.1 (or sometimes localhost, though 127.0.0.1 is more common for file:/// origins in this context). This domain was not in the "Authorized domains" list in the Firebase project.

Resolution:

- Navigated to the Firebase Console -> Project -> Authentication -> Settings tab.
- In the "Authorized domains" section, **127.0.0.1** was added to the list of authorized domains.
- After waiting a few minutes for the setting to propagate, a hard refresh of the application page allowed the Google Sign-in popup to proceed correctly without the auth/unauthorized-domain error.

3/ 500 Internal Server Error when Fetching Tasks After Login

Symptoms:

- After successfully logging in, the frontend UI showed "Error fetching tasks. See console."
- The browser console showed GET http://34.129.213.124/tasks 500 (Internal Server Error).
- Task creation *was* working, and tasks were being saved to Firestore with the correct userId.

Root Cause:

- The GKE pod logs for the backend application revealed the detailed error: Error: 9 FAILED_PRECONDITION: The query requires an index. You can create it here: [URL to create index].
- The backend's GET /tasks route was attempting to query the tasks collection in Firestore using db.collection('tasks').where('userId', '==', uid).orderBy('createdAt', 'desc').
- A Firestore query that filters on one field (userId) and orders by a *different* field (createdAt) requires a composite index to be executed efficiently. This index was missing.

Resolution:

- The link provided in the GKE pod log error message was used to navigate directly to the "Create Index" page in the Firebase console.
- The console pre-filled the necessary fields for the composite index:
- Collection: tasks
- Fields: userId (Ascending), createdAt (Descending)
- The index was created by clicking "Save".
- After waiting a few minutes for the index to build and become "Enabled", the application was retested (with a hard refresh on the frontend).
- The GET /tasks request then succeeded, and tasks were correctly fetched and displayed for the logged-in user. No backend code changes or redeployment were needed for this specific fix, as the issue was purely a database indexing requirement.