CMPE48A – Cloud Computing
FINAL REPORT

Students:
Hasancan Keleş and Barış Öksüz

Architecture Diagram

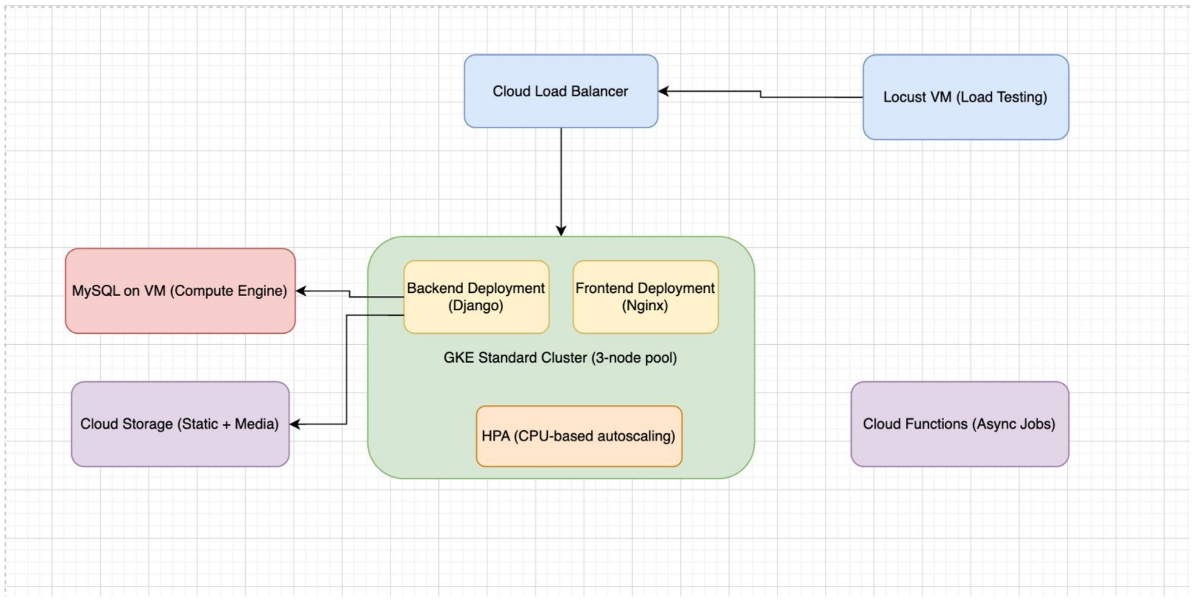

Cloud Load Balancer
Locust VM (Load Testing)
MySQL on VM (Compute Engine)
Backend Deployment (Django)
Frontend Deployment (Nginx)
GKE Standard Cluster (3-node pool)
Cloud Storage (Static + Media)
HPA (CPU-based autoscaling)
Cloud Functions (Async Jobs)

Figure 1: Cloud architecture: Load Balancer → GKE (frontend/backend) → MySQL VM, with buckets, Cloud Functions, Artifact Registry, and Locust VM.

1 System Architecture

Important Disclaimer (Experimental Node Pool)

Important: During performance evaluation, we temporarily enabled a larger GKE node pool (`bigger-pool`) to explore bottlenecks and scaling behavior. This was **not** part of the original proposal architecture, and it should be treated as an experimental configuration used only for load testing iterations. The original small pool (`nutrihub-pool`, $3 \times$ `e2-medium`) still exists but may be scaled down while experiments are running. To return to the proposal-aligned infrastructure (and the Terraform default configuration), the cluster should be reverted by **shrinking/disabling the experimental pool** and **restoring the original pool** to 3 nodes.

1.1 Components

- **Cloud Load Balancer (HTTP/S)**: Public entrypoint at `136.110.255.27`, terminates HTTP/S and forwards to GKE Ingress.
- **GKE Standard Cluster (3 x e2-medium)**: Two deployments in namespace `nutrihub`:
  - *Frontend* (Nginx) serving the React SPA.
  - *Backend* (Django + Gunicorn) exposing the API.

  Workload Identity: KSA `backend-sa` bound to GSA `nutrihub-backend`.
- **MySQL VM (e2-micro)**: Compute Engine instance with private IP for app data.
- **Cloud Storage Buckets**: *nutrihub-static-media* for static/media; *baris-media-dev* and function/source buckets (e.g., *term-project-480817-image-cache*) for uploads/cache artifacts.
- **Cloud Functions** (three services in `gcp-functions/`):

  - *login_email_sender*: Sends login/verification emails; HTTP or Pub/Sub trigger; can read assets from buckets.
  - *image_cache_subscriber* (image-cache-function): Subscribes to cache events (Pub/Sub), reads/writes cached objects (e.g., image cache bucket).
  - *badge_calculator*: Processes badge logic asynchronously (Pub/Sub), can call backend APIs or persist results.

- **Locust VM (e2-standard-2)**: Load generator with Locust UI on port 8089; targets the load balancer IP.
- **Artifact Registry**: Hosts backend/frontend images in `europe-west1-docker.pkg.dev/$PROJECT/nutrihub/`.
- **Infrastructure as Code**: Terraform configs (`infra/terraform/`) provision GKE, node pool, MySQL VM, buckets, static IP, Artifact Registry; Kubernetes manifests (`deploy/gke/k8s-manifests.yaml`) deploy frontend/backend, services, ingress, config/secrets.
- **Application Source**: Full codebase present (backend Django app, frontend React app, mobile React Native app), plus Locust script (`locustfile.py`) and Cloud Functions.

1.2 Interactions and Data Flow

1. **User Traffic**: Load Balancer → Ingress routes:
   - `/` → frontend Service → frontend Pod (Nginx serves SPA).
   - `/api/` → backend Service → backend Pod (Django/Gunicorn).
2. **Backend to Database**: Backend pods reach MySQL VM via private IP on port 3306.
3. **Static/Media**: Backend reads/writes `nutrihub-static-media`; frontend references assets via HTTPS URLs.
4. **Load Generation**: Locust VM sends HTTP(s) to the LB IP, exercising frontend and backend endpoints.
5. **Cloud Functions**: Pub/Sub or HTTP triggers for async tasks (email, cache warming, badge calc); may call backend APIs or read/write buckets.
6. **Observability**: GKE, load balancer, and MySQL VM metrics (CPU, memory, latency, error rates) in Cloud Monitoring.

1.3 Operational Notes

- Node pool fixed at 3 nodes (e2-medium); backend/frontend sized to fit this capacity.
- Images pulled from `europe-west1-docker.pkg.dev/$PROJECT/nutrihub/`.
- Ingress uses the reserved global static IP `nutrihub-ip`.
- Terraform defaults reflect the small pool; manifests and function code are in-repo for reproducible deployment.

2 Deployment Process (Step by Step)

1. **Provision infra (Terraform)**
   - `cd infra/terraform`
   - `terraform init`
   - `terraform apply -var "project_id=YOUR_PROJECT_ID"`
   - Defaults: $3 \times$ e2-medium GKE nodes, e2-micro MySQL VM, static IP, buckets, Artifact Registry.
2. **Get cluster credentials**
   - `gcloud container clusters get-credentials nutrihub --zone europe-west1-b --project YOUR_PROJECT_ID`
3. **Apply Kubernetes manifests**
   - Edit `deploy/gke/k8s-manifests.yaml`: set `MYSQL_HOST`, bucket names, secrets.
   - `kubectl apply -f deploy/gke/k8s-manifests.yaml`
4. **Build & push images to Artifact Registry**
   - `REGION=europe-west1; PROJ=YOUR_PROJECT_ID`
   - `docker build -t $REGION-docker.pkg.dev/$PROJ/nutrihub/backend:latest backend`
   - `docker push $REGION-docker.pkg.dev/$PROJ/nutrihub/backend:latest`
   - `docker build -t $REGION-docker.pkg.dev/$PROJ/nutrihub/frontend:latest frontend \`
   - `  --build-arg VITE_API_BASE_URL=http://136.110.255.27/api`
   - `docker push $REGION-docker.pkg.dev/$PROJ/nutrihub/frontend:latest`
5. **Roll out deployments**
   - `kubectl rollout restart deploy/backend deploy/frontend -n nutrihub`
6. **(Optional) Adjust backend resources for tests**
   - `kubectl set resources deploy/backend -n nutrihub \`
   - `  --requests=cpu=500m,memory=512Mi --limits=cpu=500m,memory=1Gi`
   - Adjust as needed for other iterations; HPA lives in the manifest.
7. **Run Locust load test**
   - `locust -f locustfile.py --host http://136.110.255.27`
   - Configure users/ramp in the UI; collect CSV/graphs for analysis.
8. **Deploy Cloud Functions**
   - From each function folder under `gcp-functions/`:
   - `gcloud functions deploy <name> --runtime=python311 --trigger-topic=<topic>` (or HTTP trigger)
   - Ensure env/secrets and bucket names align with the Terraform outputs.

3 Load Testing

Methodology Note (minor tuning not shown)

In addition to the three documented experiments below, we made several smaller configuration adjustments while troubleshooting and validating the system under load (e.g., iterating on backend CPU/memory *requests/limits*, adjusting HPA bounds, attempting node-pool resize/new pool creation under quota constraints, and fixing frontend API base URL configuration). We do not enumerate every minor tweak in this report because many of them were transient, not run as full controlled experiments with the same duration/parameters, and would add noise without improving the main performance narrative. Instead, we present the three runs where we captured complete artifacts (Locust metrics + graphs + Cloud Monitoring screenshots) and kept settings stable for the duration of each run, which supports clear bottleneck identification and comparative analysis.

3.1 Initial Trial (baseline, small pool)

Test setup

- **Scenario**: Locust UI (HTTP). Endpoints: `/`, `/api/foods/`, `/api/time?name=locust`, `/api/healthz/`. The external `/api/foods/random-meal/` endpoint was excluded to avoid third-party latency dominating results.
- **Load pattern**: 800 users, spawn rate 50 users/s, runtime 10 minutes.
- **Target**: GKE Ingress / Load Balancer at `http://136.110.255.27`.
- **Backend deployment (baseline)**: $3 \times$ e2-medium node pool (2 vCPU / 4 GiB each). Backend pods limited to ∼500m CPU each. HPA min=2, max=6, target CPU=60%. MySQL runs on an e2-micro VM.

Key metrics (Locust UI)

- **Throughput**: ∼100–150 req/s steady.
- **Latency**:
  - `/`: median ∼110 ms, p95 ∼110 ms.
  - `/api/foods/`: median ∼6.8 s, p95 ∼18 s, p99 ∼22 s.
  - `/api/healthz/` and `/api/time?name=locust`: similar to `/api/foods/` (median ∼6.5–6.6 s, p95 ∼18 s, p99 ∼22 s).
  - Aggregated: median ∼1.1 s, p95 ∼17 s, p99 ∼21 s.
- **Errors**: 2,352 of 73,844 requests failed.

Infra/resource observations (Cloud Monitoring)

- **Backend HPA**: scaled to max=6 and stayed pinned for the run (ScalingLimited=True).
- **Backend pod CPU**: ∼0.5 cores per pod (matches the 500m limit), flat-topped ⇒ CPU-limited/throttled. Pod memory ∼900 MiB, stable.
- **Nodes**: one node ∼1.0–1.2 cores (about 60% of e2-medium), others ∼0.5–0.7 cores ⇒ cluster had spare headroom, but pod CPU limits prevented using it.
- **Load balancer**: request rate ∼100–150 rps; latency rose to ∼2–3 s; 5xx spikes during peak.
- **MySQL VM (e2-micro)**: CPU ∼10–20%, memory ∼8.8 GiB steady, disk IO ∼1.5 MiB/s ⇒ DB not saturated.
- **Frontend API base fix**: frontend was rebuilt with `VITE_API_BASE_URL=http://136.110.255.27/api` to fix 405s caused by missing `/api/` prefix.

Interpretation (bottleneck)

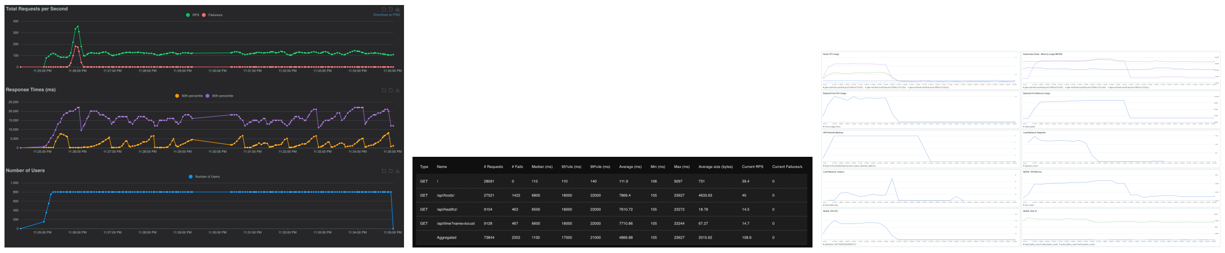
- The application tier is the bottleneck: each backend pod is capped at 500m CPU and the HPA maxed out at 6 replicas, constraining total backend compute.
- Nodes still had free CPU and MySQL stayed healthy, while latency and LB 5xx increased under load $\Rightarrow$ classic pod CPU saturation/throttling and insufficient replica ceiling.

Infra constraints observed

- Regional `IN_USE_ADDRESSES` quota was 4. Capacity experiments were constrained by quota, and we later created a separate larger pool (`bigger-pool`) while keeping the same Ingress IP.

Planned remediation test (iteration 2)

- **Objective**: relieve backend CPU throttling and demonstrate improved p95/p99 and fewer 5xx.
- **Planned backend resources**: requests 600m CPU / 900Mi; limits 1 vCPU / 1.5 GiB; 2 replicas.
- **HPA**: min=2, max=6, target CPU=60%.
- **Planned load**: 600 users, spawn rate 30 users/s, duration 5 minutes (same endpoints).
- **Success criteria**: lower p95/p99 on `/api/foods/`, reduced LB 5xx, backend pods not pinned at 100% of request; HPA scales within 2–4 as needed.



Type	Name	# Requests	# Fails	Median (ms)	95%ile (ms)	99%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
GET	/	28091	0	110	110	140	111.9	106	9297	731	39.4	0
GET	/api/foods/	27521	1422	6800	18000	22000	7869.4	105	23627	4633.63	40	0
GET	/api/healthz/	9104	463	6500	18000	22000	7610.72	105	23272	18.78	14.5	0
GET	/api/time?name=locust	9128	467	6600	18000	22000	7710.86	105	23244	67.27	14.7	0
	Aggregated	73844	2352	1100	17000	21000	4866.88	105	23627	2015.62	108.6	0

Figure 2: Initial trial: Locust charts, Locust stats, GCP monitoring.

3.2 First Iteration (Post–Node Pool Upgrade)

What changed

- Switched to a new, larger node pool (`bigger-pool`), keeping the same Ingress IP (`136.110.255.27`).
- Backend stayed at 500m CPU / 1Gi limits and 250m / 512Mi requests (2 replicas); HPA set to min=2, max=4, target CPU=60%.
- Frontend rebuilt with `VITE_API_BASE_URL=http://136.110.255.27/api` to fix missing `/api/` prefix and prevent 405 errors.

Test setup

- Locust UI, endpoints: `/`, `/api/foods/`, `/api/time?name=locust`, `/api/healthz/`.
- Load: ~800 users (steady), spawn rate as configured in UI, runtime ~5 minutes.
- Target: `http://136.110.255.27` (Ingress).

Key results (Locust)

- **Throughput**: peaked ∼450–500 req/s then flattened ∼120–150 req/s.
- **Latency**:
  - `/`: median ∼110 ms, p95 ∼120–150 ms.
  - `/api/foods/`: median ∼3.6 s, p95 ∼19 s, p99 ∼33 s.
  - `/api/healthz/`: median ∼3.3 s, p95 ∼19 s, p99 ∼31 s.
  - `/api/time?name=locust`: median ∼3.4 s, p95 ∼18 s, p99 ∼31 s.
  - Aggregated: median ∼390 ms, p95 ∼16 s, p99 ∼27 s.
- **Failures**: 0.

Infra/resource observations (Cloud Monitoring)

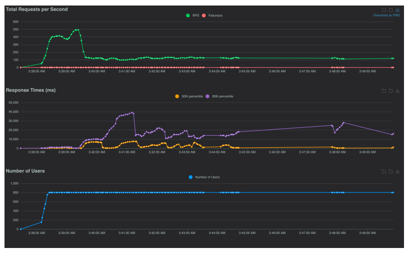
- Nodes (`bigger-pool`): CPU low overall; brief bump early then ∼10–20% CPU. Memory ample.
- HPA: scaled up to 4 replicas briefly, then back down; min=2, max=4 in effect.
- Backend pods: CPU briefly spiked, then low; memory ∼200–300 MiB. No sustained node CPU pressure.
- Load balancer: latency peaked then declined; request rate flattened ∼150 rps.
- MySQL VM: CPU modest (∼10–20%), memory stable (∼12–13 GiB), disk IO low.

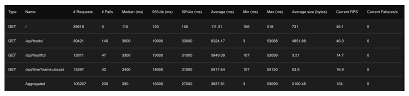
Interpretation

- Despite larger nodes and frontend fix, backend latency remained high (p95 up to ∼19–33 s) and RPS flattened because backend pods were still constrained at 500m CPU and only 2–4 replicas.
- DB and nodes were not saturated. The bottleneck remained in the backend tier (insufficient per-pod CPU and replica headroom), not in Locust or MySQL.

Next steps

- Apply higher backend pod resources: requests 1 vCPU / 1.5 GiB; limits 2 vCPU / 2 GiB; restart deployment.
- Increase HPA headroom: min=2, max=8, target CPU=60%.
- Re-run a push 10-minute test with higher load (1500 users, spawn rate 100/s) and compare p95/p99 and RPS.
- If latency persists after more CPU/replicas, inspect app/DB paths (slow queries, external calls), since node/VM headroom is not the limiter.





Type	Name	# Requests	# Fails	Median (ms)	95%ile (ms)	99%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
GET	/	39618	0	110	120	150	111.31	106	518	731	46.1	0
GET	/api/foods/	39421	140	3600	19000	33000	6224.17	3	53088	4851.88	46.3	0
GET	/api/healthz/	12871	47	3300	19000	31000	5849.09	107	53099	3.21	14.7	0
GET	/api/time?name=locust	13297	43	3400	18000	31000	5917.64	107	52120	53.9	16.9	0
	Aggregated	105207	230	390	16000	27000	3837.61	3	53099	2100.48	124	0

Figure 3: First iteration: Locust charts, Locust stats, GCP monitoring.

3.3 Second Iteration (post–resource bump, 15k-user push)

What changed

- Backend pod resources increased to requests 1 vCPU / 1.5 GiB, limits 2 vCPU / 2 GiB.
- HPA headroom raised to min=2, max=8, target CPU=60% (HPA requested up to 8).
- New larger node pool (`bigger-pool`) in place; frontend rebuilt with correct API base.

Test setup

- Locust UI, endpoints: `/`, `/api/foods/`, `/api/time?name=locust`, `/api/healthz/`.
- Load: ∼15,000 users (aggressive push), spawn ramp in UI, duration ∼5–10 minutes.
- Target: `http://136.110.255.27` (Ingress).

Key results (Locust)

- **Throughput**: burst into ∼700–900 rps; sustained ∼700–800 rps with high failures during the plateau.
- **Latency (latest run table)**:
  - `/`: median 7.2 s, p95 22 s, p99 50 s.
  - `/api/foods/`: median 7.5 s, p95 37 s, p99 74 s.
  - `/api/healthz/` / `/api/time`: median 7.5 s, p95 37 s, p99 72–77 s.
  - Aggregated: median 7.4 s, p95 33 s, p99 62 s.
- **Failures**: ∼112k failures (111,937) out of ∼209k requests; failures rose as soon as the plateau started.

Infra/resource observations (Cloud Monitoring & logs)

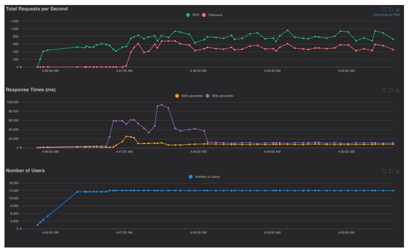
- HPA requested 7–8 pods, but cluster-autoscaler scale-up failed: `CPUS_ALL_REGIONS` quota exceeded and node-pool max=3. Only 4 pods remained Running; additional replicas stayed Pending or were deleted.
- Nodes (`bigger-pool`): CPU low (∼18% / 9%); memory ample.
- Backend pods: CPU modest (∼100–260m); numerous readiness/liveness probe timeouts during peak; Gunicorn worker timeouts (`WORKER TIMEOUT`, “no URI read”, SIGKILL).
- Load balancer latency climbed with failures; request rate flattened once errors surged.
- MySQL VM: CPU/memory/IO modest—DB not the bottleneck.

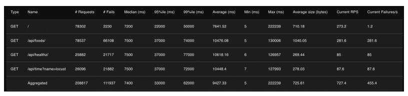
Interpretation

- At ∼15k users the backend could not keep up: p95 33–62 s and > 100,000 failures. HPA asked for up to 8 replicas, but autoscaler was blocked by quota (`CPUS_ALL_REGIONS`) and node-pool max=3, so extra pods could not land; existing pods then timed out (Gunicorn worker timeouts, probe failures).
- Nodes and DB were not saturated—capacity was capped by quota and app timeouts under overload.

Decision

- This was an experimental push run. Increasing capacity would require higher CPU quotas and/or larger nodes, which we are not pursuing.
- We will revert to the original smaller node pool plan (3 × e2-medium) for ongoing use and documentation.





Type	Name	# Requests	# Fails	Median (ms)	95%ile (ms)	99%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
GET	/	78302	2230	7200	22000	50000	7641.52	5	222239	710.18	273.2	1.2
GET	/api/foods/	78537	66108	7500	37000	74000	10476.08	5	130006	1040.05	281.6	281.6
GET	/api/healthz/	25882	21717	7500	37000	77000	10618.16	6	126957	269.44	85	85
GET	/api/time?name=locust	26096	21882	7500	37000	72000	10448.4	7	127993	278.03	87.6	87.6
	Aggregated	208817	111937	7400	33000	62000	9427.33	5	222239	725.61	727.4	455.4

Node CPU Usage
Backend Pod CPU Usage
HPA Desired Replicas
Load Balancer Latency
MySQL VM CPU

Figure 4: Second iteration: Locust charts, Locust stats, GCP monitoring.

3.4 Cross-Run Interpretation

- Backend CPU limits/replica ceilings dominated p95 and throughput in all runs; larger nodes alone did not help.
- Scaling up pod resources without available cluster quota led to failures (iteration 2): HPA asked for pods that could not schedule; timeouts followed.
- DB and nodes remained healthy across runs; bottleneck is backend capacity and timeouts.
- Decision: revert to the small, cost-bound node pool and avoid further scale-up unless quotas/costs change.

4 Cost Estimation (3 Months)

4.1 GKE Standard Cluster

- Node type: `e2-medium` (2 vCPU, 4 GiB RAM).
- Approx. \$34/month per node (if running continuously).
- 3-node pool (used only during development/testing).
- Optimized active usage assumption: cluster active 120 hours/month.
- Estimated monthly GKE compute:

$$120 \text{ h} \times 3 \times 0.047 \text{ USD/h} \approx 16.9 \text{ USD/month} \approx 17 \text{ USD/month}$$

4.2 MySQL VM

- Instance: `e2-micro`.
- Cost: ∼\$7–8 per month (∼\$21–24 over 3 months).

4.3 Load Balancer, Logs, and Network

- Cloud Load Balancer + logging: ∼\$10 per month (∼\$30 over 3 months).

4.4 Cloud Storage

- 20 GiB storage budget.
- Cost: ∼\$0.40 per month (∼\$1.20 over 3 months).

4.5 Cloud Functions

- Light usage stays within the free tier.
- Cost: \$0 (for this project usage pattern).

4.6 Locust VM (Load Testing)

- Instance: `e2-standard-2`.
- Usage: 20 hours total.
- Estimated cost:

$$0.083 \text{ USD/h} \times 20 \text{ h} \approx 1.66 \text{ USD} \approx 1.6 \text{ USD}$$

4.7 Total Estimated Cost (3 Months)

- GKE (optimized): ∼\$51
- MySQL VM: ∼\$21–24
- Load Balancer + logs: ∼\$30
- Cloud Storage: ∼\$1.2
- Cloud Functions: \$0
- Locust VM: ∼\$1.6
- Total: ∼\$70–80 (3 months), which fits within the Google Cloud Free Trial credit (\$300).