🚀Hospital API VM Setup & Usage Guide

This is a **complete guide** to set up, run, and test all hospital API Flask scripts (patients, appointments, lab results) on a GCP VM, following enterprise best practices.

---

1. GCP Firewall & Network

- **Create firewall rule** to allow required ports:
- TCP 22 (SSH)
- TCP 5001 (Patients API)
- TCP 5002 (Appointments API)
- TCP 5003 (Lab Results API)

Command:

```
gcloud compute firewall-rules create hospital-api-ingress \
  --direction=INGRESS \
  --priority=1000 \
  --network=default \
  --action=ALLOW \
  --rules=tcp:22,tcp:5001,tcp:5002,tcp:5003 \
  --source-ranges=0.0.0.0/0 \
  --target-tags=hospital-api-vm \
  --description="Allow SSH and API access for hospital APIs"
```

Tip: In production, restrict `--source-ranges` to your office/public IPs.

---

2. Service Account & IAM Roles

- **Create a dedicated service account:**

```
gcloud iam service-accounts create hospital-api-vm-sa \
  --description="Service Account for API VM" \
  --display-name="Hospital API VM Service Account"
```

- **Grant permissions:**

```
gcloud projects add-iam-policy-binding YOUR_PROJECT_ID \
  --member="serviceAccount:hospital-api-vm-
sa@YOUR_PROJECT_ID.iam.gserviceaccount.com" \
  --role="roles/secretmanager.secretAccessor"

gcloud projects add-iam-policy-binding YOUR_PROJECT_ID \
  --member="serviceAccount:hospital-api-vm-
```

```
sa@YOUR_PROJECT_ID.iam.gserviceaccount.com" \
  --role="roles/logging.logWriter"
```

Add `roles/pubsub.publisher` if your API/producer will write to Pub/Sub.

---

3. VM Creation

- **Create VM with the correct tag and service account:**

```
gcloud compute instances create hospital-api-vm \
  --zone=YOUR_ZONE \
  --machine-type=e2-medium \
  --image-family=ubuntu-2204-lts \
  --image-project=ubuntu-os-cloud \
  --tags=hospital-api-vm \
  --service-account=hospital-api-vm-
sa@YOUR_PROJECT_ID.iam.gserviceaccount.com \
  --scopes=https://www.googleapis.com/auth/cloud-platform
```

---

4. VM Bootstrapping (SSH into VM)

4A. (Recommended) Install Python 3.11.9 and Create a Virtual Environment

1. **Install Python 3.11.9 (if not already present):**

```
sudo apt update
sudo apt install -y wget build-essential zlib1g-dev libncurses5-dev libgdbm-
dev \
libnss3-dev libssl-dev libreadline-dev libffi-dev libsqlite3-dev curl libbz2-
dev

cd /tmp
wget https://www.python.org/ftp/python/3.11.9/Python-3.11.9.tgz
tar -xf Python-3.11.9.tgz
cd Python-3.11.9
./configure --enable-optimizations
make -j $(nproc)
sudo make altinstall  # installs as python3.11, keeps system python untouched
```

1. **Check Python version:**

```
python3.11 --version
# Output should be: Python 3.11.9
```

1. **Create and activate a virtual environment:**

```
cd <your-repo>
python3.11 -m venv venv
source venv/bin/activate
```

- You'll now see `(venv)` in your prompt.
- All `pip install` commands go inside the venv (safe, clean).
- **(Now) Install dependencies:**

```
pip install --upgrade pip
pip install -r requirements.txt
```

- To leave the env: `deactivate`

Best Practice:

- A virtual environment keeps all Python packages project-specific (never pollutes system Python).
- Prevents version conflicts and makes re-setup clean on any VM/server.

---

1. **Install Python, pip, and Git:**

```
sudo apt update
sudo apt install -y python3 python3-pip git
```

1. **Clone your GitHub repo:**

```
git clone https://github.com/<your-org>/<your-repo>.git
cd <your-repo>
```

1. **requirements.txt — All Needed Packages**

In your repo **root**, create or verify a file called `requirements.txt` with these contents:

```
flask
gunicorn
google-cloud-secret-manager
google-cloud-logging
google-cloud-pubsub
requests
python-json-logger
pydantic
typing-extensions
```

txt flask gunicorn google-cloud-secret-manager

If you use Cloud Logging:

google-cloud-logging

If you use Pub/Sub anywhere in producer or other services:

google-cloud-pubsub

Utility libraries (if used in shared_module):

requests

For advanced logging/monitoring (optional)

python-json-logger

```
- **Edit this file** to add/remove packages as your code evolves (e.g., add
`pytest` for testing, `pandas` if needed, etc).
- **Keep this file in the root of your repo** so `pip3 install -r
requirements.txt` works out of the box for anyone who clones your code.

4. **Install dependencies:**

```bash
pip3 install -r requirements.txt
```

---

5. API Key Management with Google Secret Manager (GSM)

1. **Generate strong random API keys:**

```
openssl rand -hex 32 > patients_api_key.txt
openssl rand -hex 32 > appointments_api_key.txt
openssl rand -hex 32 > lab_results_api_key.txt
```

NOTE: How the API Key `` File Works

- The command

  ```
  openssl rand -hex 32 > patients_api_key.txt
  ```
  ```

generates a random 32-byte (256-bit) hexadecimal string and writes it into a new file named `patients_api_key.txt` in your **current working directory**.

- This file's only purpose is to **hold your new API key just long enough to upload it to Google Secret Manager (GSM)** using the next gcloud command.
- You use this file **only for upload**:

```
gcloud secrets versions add patients_api_key --data-
file=patients_api_key.txt
```

- **After upload, you do NOT use this file in your application code**. All Flask apps will fetch the actual API key directly from GSM using the environment variable path.
- **Best practice:** Delete the `.txt` file after upload to GSM:

```
rm patients_api_key.txt
```

- **Never commit these .txt files to your repo**—they are for one-time secure upload only.

Summary: The `.txt` file is a temporary holder for a random key. It is never read by your Flask app and is not needed after you upload to GSM.

---

1. **Create secrets in GSM:**

```
gcloud secrets create patients_api_key --replication-policy="automatic"
gcloud secrets versions add patients_api_key --data-file=patients_api_key.txt

gcloud secrets create appointments_api_key --replication-policy="automatic"
gcloud secrets versions add appointments_api_key --data-
file=appointments_api_key.txt

gcloud secrets create lab_results_api_key --replication-policy="automatic"
gcloud secrets versions add lab_results_api_key --data-
file=lab_results_api_key.txt
```

---

6. Running the Flask APIs with Gunicorn

Step 1: Stop any old Gunicorn processes

```
pkill gunicorn
```

Step 2: Export API Key Secret IDs for Each API

Each API should use its own unique API key, stored securely in Google Secret Manager (GSM). We export **one environment variable per API**, each pointing to the correct GSM secret version:

```
# Patients API Key (for patients_api.py)
export PATIENTS_API_KEY_SECRET_ID="projects/gcp-de-vaarahi-b38/secrets/
patients_api_key/versions/latest"

# Appointments API Key (for appointments_api.py)
export APPOINTMENTS_API_KEY_SECRET_ID="projects/gcp-de-vaarahi-b38/secrets/
appointments_api_key/versions/latest"

# Lab Results API Key (for lab_results_api.py)
export LAB_RESULTS_API_KEY_SECRET_ID="projects/gcp-de-vaarahi-b38/secrets/
lab_results_api_key/versions/latest"
```

Explanation:

- Each export command sets an environment variable, visible to all processes started in that shell session.
- Each variable points to the *full resource path* of the API key secret in GSM (using the secret's name and `latest` version for automatic rotation support).
- When you start each API, the corresponding Python code will fetch its API key from GSM using the correct environment variable.

Step 3: Start Each API with Gunicorn (in background)

> **Tip:** Run these from your `api/` directory, after exporting the relevant secret ID variables above **in the same shell/tab** so each API process can read its variable.

```
# Patients API (port 5001)
gunicorn -w 2 -b 0.0.0.0:5001 patients_api:app &

# Appointments API (port 5002)
gunicorn -w 2 -b 0.0.0.0:5002 appointments_api:app &

# Lab Results API (port 5003)
gunicorn -w 2 -b 0.0.0.0:5003 lab_results_api:app &
```

- `-w 2` is number of worker processes; tune based on VM CPU.
- The environment variables set above will be inherited by the Gunicorn processes, allowing the API code to securely fetch the correct API key from GSM.

Example in Python (`` or similar):**

```
import os
from google.cloud import secretmanager

def get_api_key_from_gsm(env_var):
    secret_id = os.environ[env_var]
    client = secretmanager.SecretManagerServiceClient()
    response = client.access_secret_version(request={"name": secret_id})
```

```
    return response.payload.data.decode('UTF-8')

# For patients_api: key = get_api_key_from_gsm('PATIENTS_API_KEY_SECRET_ID')
# For appointments_api: key =
get_api_key_from_gsm('APPOINTMENTS_API_KEY_SECRET_ID')
# For lab_results_api: key =
get_api_key_from_gsm('LAB_RESULTS_API_KEY_SECRET_ID')
```

> This approach makes it very clear for every engineer or DevOps member which API key is used by each Flask service and how to rotate/manage them in GSM securely.

---

7. Testing with cURL

Patients API:

```
curl -H "x-api-key: <YOUR_PATIENTS_API_KEY>" http://<EXTERNAL_IP>:5001/
patients
```

Appointments API:

```
curl -H "x-api-key: <YOUR_APPOINTMENTS_API_KEY>" http://<EXTERNAL_IP>:5002/
appointments
```

Lab Results API:

```
curl -H "x-api-key: <YOUR_LAB_RESULTS_API_KEY>" http://<EXTERNAL_IP>:5003/
lab_results
```

Health Check for All:

```
curl -H "x-api-key: <YOUR_API_KEY>" http://<EXTERNAL_IP>:5001/health
curl -H "x-api-key: <YOUR_API_KEY>" http://<EXTERNAL_IP>:5002/health
curl -H "x-api-key: <YOUR_API_KEY>" http://<EXTERNAL_IP>:5003/health
```

---

8. Security & Production Tips

- **Restrict firewall** to trusted IPs in production.
- Use **HTTPS/reverse proxy (Nginx/Apache)** for security.
- Run APIs as **systemd services** for auto-restart.
- **Rotate API keys** in GSM as needed.
- **Never store API keys in source code or env files.**
- Use GCP IAM & GSM for all secrets.

9. API Directory Structure Example

```
hospital-streaming-pipeline/
├── README.md
├── api
│   ├── __init__.py
│   ├── __pycache__/
│   │   ├── appointments_api.cpython-310.pyc
│   │   ├── lab_results_api.cpython-310.pyc
│   │   └── patients_api.cpython-310.pyc
│   ├── appointments_api.py
│   ├── lab_results_api.py
│   ├── patients_api.py
│   └── shared_module
│       ├── __init__.py
│       ├── __pycache__/
│       │   ├── auth.cpython-310.pyc
│       │   └── utils.cpython-310.pyc
│       ├── auth.py
│       ├── logger.py
│       ├── schemas.py
│       └── utils.py
├── config
│   ├── dev_config.yaml
│   └── example.env
├── patients_api_key.txt         # (Temporary, should be deleted after GSM
upload)
└── requirements.txt
```

- ``: All Flask API scripts (patients, appointments, lab results) and shared modules.
- ``: Common utilities (auth, logging, schemas, utils).
- ``: Configuration files (non-secret YAML/env files).
- ``: All Python dependencies for setup.
- ``: Temporary secret file (should be deleted after GSM upload).

---

10. FAQ / Troubleshooting

- **Q:** My API isn't reachable?\ **A:** Check firewall rule, Gunicorn process, and correct port.
- **Q:** Permission denied when fetching API key? **A:** Make sure your VM's service account has `roles/secretmanager.secretAccessor`.
- **Q:** How do I check API logs? **A:** Use Cloud Logging, or check Gunicorn/stdout in your VM.

11. Producer VM, Pub/Sub, and End-to-End Data Flow

- **Typical Enterprise Pipeline:**\ `Client → Hospital API VM → Producer VM → Pub/Sub → (Dataflow, BigQuery, etc.)`

- **Producer VM:**
- Has `roles/pubsub.publisher` on the topic.
- Fetches data from your API endpoint using the API key.
- Publishes data to Pub/Sub for downstream processing.
- **Pub/Sub Setup Example:**

```
bash
```

CopyEdit

`gcloud pubsub topics create my-topic gcloud pubsub subscriptions create my-sub --topic=my-topic` - **Interview Tip:** Always describe the *full* data flow in enterprise interviews—API auth, separation of duties, and audit trail.

---

12. API Key Management: Single vs. Per-API Key Design

- **Single Secret:** One API key (e.g., `hospital-api-key`) for all endpoints—simplifies rotation but less granular.
- **Per-API Secret:** Separate GSM secret per endpoint—best for microservices or if you want to revoke keys independently.
- **Best Practice:** Choose based on security need and explain which is in use in your README.

---

13. Why Not Use Service Account JSON for API Clients?

- **API keys** are for stateless, client-accessible endpoints.
- **Service Account JSON** is for GCP-internal server-to-server auth (not for user or external client use).
- **Key Interview Q:** Never share SA JSON with API clients—rotate/secure API keys using GSM instead.

---

14. Gunicorn: Foreground, Daemon, and Production Patterns

- **Foreground (``):** Good for development/testing.
- **Daemon mode (``):** Safer for backgrounding.
- **Recommended:** Use `systemd` or `supervisor` for reliability (auto-restart on reboot/crash).

Sample systemd unit:

```
ini
```

CopyEdit

`[Unit] Description=Patients API Gunicorn Service After=network.target [Service] User=ubuntu WorkingDirectory=/path/to/api ExecStart=/path/to/venv/bin/gunicorn -w 2 -b 0.0.0.0:5001 patients_api:app [Install] WantedBy=multi-user.target`

15. Audit Columns and Data Compliance

ColumnTypeDescription		
created_ts	TIMESTAMP	When record was created
updated_ts	TIMESTAMP	When record last updated
created_by	STRING	Creator/service identifier
updated_by	STRING	Last updater identifier
is_active	BOOLEAN	Soft delete/logical active status

- **Always** use these for healthcare, financial, and any regulated pipelines.

---

16. Advanced Testing & Troubleshooting

- **Check Gunicorn status/ports:** `sudo netstat -tulnp | grep gunicorn`
- **Logs:** `journalctl -u <service-name>` (if using systemd)
- **Curl with debug:** `curl -v ...`
- **If API is not working:** Recheck firewall, Gunicorn, service account, and GSM permissions. Restart process after changes.

---

17. Containerization, CI/CD, and Cloud Readiness

- **Recommended for prod:** Use Docker for packaging, CI/CD (GitHub Actions, Cloud Build) for deployments.
- **Secrets/configs:** Pass via environment variables or GSM.
- **Cloud Run/Kubernetes:** Easily port your Gunicorn/Flask app as a containerized microservice.

---

18. Interview & Documentation Keywords

Enterprise API Security, Google Secret Manager (GSM), WSGI Hosting (Gunicorn), GCP IAM & Roles, Stateless Microservices, Pub/Sub Producer, Audit Columns, Data Compliance, CI/CD, Containerization.

This guide is production-proven. Copy into your README or share with your team to make setup a breeze!