AI-03939 OpenEMR Production Deployment Docker GCP

Owner	Ⓡ Rigel Arcayan
Tags	Project
Created time	June 15, 2026 12:20 AM

Reference Documents:
1. openemr/openemr - Docker Image
2. OpenEMR Project Wiki

Can OpenEMR Docker Deployment in Google Cloud Platform e2-highmem 8GB 60GB be run on real Medical Clinic in Hernando, Florida 34442 without issue? Provide recommendations if there are any.

Yes, an OpenEMR Docker deployment on a Google Cloud Platform (GCP) **e2-highmem-2** instance (which provides 2 vCPUs, 16 GB RAM by default—or an `e2-standard-2` / custom shape matching your **8 GB RAM and 60 GB SSD** target) can absolutely run a real, small-to-medium-sized medical clinic without issue.

OpenEMR is highly efficient. The base application and its database (typically MariaDB) require minimal compute overhead, meaning 8 GB of RAM is more than enough for a standard practice with a dozen concurrent users.

However, running a *live clinical production environment* in Florida requires addressing critical operational, compliance, and architectural gaps that go beyond basic hardware specs.

🛑 The "Gotchas": Infrastructure Vulnerabilities

While 8 GB RAM / 60 GB Storage is perfectly fine for compute, the **single-instance layout** has major single points of failure (SPOFs) that must be mitigated before onboarding patients.

1. Storage Capacity & IOPS

- **The Issue:** 60 GB is plenty for the Linux OS, Docker images, and the initial SQL database. However, it will fill up rapidly once the clinic begins scanning patient IDs, uploading insurance cards, storing PDFs, or saving medical imaging (DICOM/X-rays).
- **The Risk:** If the boot disk hits 100%, the MariaDB container will suffer table corruption, crashing the clinic's schedule mid-day.

2. High Availability & State

- **The Issue:** If you store patient data inside the local Docker volume container on that single instance, you have tightly coupled your compute layer with your data layer.
- **The Risk:** If the underlying GCP hardware fails or the VM corrupts during an OS upgrade, your entire clinic goes offline, and data loss is highly probable.

3. Regulatory Compliance (HIPAA)

- **The Issue:** Operating in Hernando, FL means strict adherence to **HIPAA (Health Insurance Portability and Accountability Act)** regulations. A default Docker Compose deployment exposes plain-text data.
- **The Risk:** Storing Unencrypted Protected Health Information (ePHI) at rest, or transmitting it over unencrypted HTTP channels, carries massive legal and financial penalties.

🛠️ Production-Ready Recommendations

To transform this baseline setup into a resilient, compliant, and rock-solid deployment, implement the following architectural enhancements:

Decouple Data from Compute

Do not rely on the local 60 GB boot disk for your critical database and documents.

- Move the MariaDB database out of Docker and use **Google Cloud SQL for MySQL/MariaDB**. It handles automated patching, replication, and point-in-time recovery seamlessly.

- Mount a secondary persistent directory or utilize an object storage gateway (like a secured Google Cloud Storage bucket) to handle document and image attachments, keeping your application stateless.

Implement Proper Secrets Management

- **Never** hardcode MariaDB root passwords, OpenEMR admin credentials, or API keys directly in your `docker-compose.yml` or local environment files.
- Utilize a dedicated tool like **GCP Secret Manager** or inject them securely at runtime using protected Docker Secrets.

Fortify Security & Encryption (End-to-End TLS)

To satisfy HIPAA technical safeguards:

- Ensure **Full Disk Encryption** is active on your GCP persistent disks (Google handles this by default, but verify customer-managed keys if your compliance officer requires it).
- Do not expose the OpenEMR container directly to the public internet. Put an **Nginx Reverse Proxy** or **GCP HTTP(S) Load Balancer** in front of it.
- Enforce **TLS 1.3** via Let's Encrypt certificates so that all transit data between the clinic's local workstations and GCP is fully encrypted.

Automated Backups & Disaster Recovery (DR)

- Set up an automated nightly cron job within the system architecture to dump the database and sync the OpenEMR `sites/default/documents` directory to an isolated, immutable Google Cloud Storage bucket.
- Enforce Object Lifecycle Management on that bucket to retain historical backups for at least 7 years (per medical record retention laws).
- **Test the restore process.** A backup is only as good as its verified restoration capability.

Network Access Control

- Do not leave the server open to the entire web. Use GCP Firewall Rules to restrict access to OpenEMR's web ports (443) exclusively to the **static public**

IP address of the physical clinic in Hernando.

- For remote clinician access (e.g., doctors charting from home), mandate a secure VPN tunnel (like WireGuard or Tailscale) into the VPC rather than exposing the login portal to the world.

Create a Step by Step Deployment in GCP. (I've already tested a Docker OpenEMR at GCP but you've made a valid and robust OpenEMR for reliability.) Just the high level task but drill down on the "Decouple Data Compute Section)

Here is the high-level, production-ready blueprint for deploying OpenEMR on Google Cloud Platform (GCP). This workflow transitions your setup from a single-instance proof of concept into a secure, decoupled, and resilient architecture suitable for a live medical clinic.

High-Level Deployment Workflow

```
[Clinic Location] ---> [GCP Cloud Armor / Firewall]
                                  |
                      [GCP Load Balancer / Nginx Proxy]
                                  |
                     [Compute: e2 Instance (Docker)]
                      /                           \
[State: Cloud SQL (MariaDB)]              [Files: Persistent Disk / GCS]
```

1. **Foundational Networking:** Create a custom VPC, configure firewall rules to restrict traffic exclusively to the clinic's static IP, and provision an external static IP address.
2. **Data Layer Provisioning (Detailed below):** Spin up a managed database instance and configure network attachment/disk sharing.
3. **Compute Setup:** Launch your `e2-highmem` or custom VM instance within the secure VPC subnet. Install Docker and Docker Compose.
4. **Secrets & Configuration:** Inject credentials securely via GCP Secret Manager or encrypted local `.env` files.
5. **Reverse Proxy & TLS:** Deploy Nginx Proxy Manager or an upstream GCP Load Balancer to terminate SSL/TLS 1.3 using Let's Encrypt certificates.
6. **Container Orchestration:** Run the OpenEMR container, explicitly bypassing its internal MySQL/MariaDB service initialization to target the external data layer.

7. **Backup Automation:** Establish automated snapshot schedules for both the cloud database and the document storage layers.

Deep Dive: Decoupling Data from Compute

In a standard proof-of-concept Docker deployment, the database engine and the uploaded files live on the same local file system as the application code. If the virtual machine crashes, corrupts, or is deleted, your data goes with it.

Decoupling splits your deployment into **Stateless Compute** (the OpenEMR Docker engine) and **Stateful Data** (the patient database and uploaded documents).

Step 1: Isolate the Database using Google Cloud SQL

Instead of running a MariaDB container locally inside Docker, you delegate database management to a specialized, automated service.

1. **Provision Cloud SQL:** Go to the GCP Console and create a **Cloud SQL for MySQL** instance (ensure the MySQL/MariaDB version matches the specific requirements of your targeted OpenEMR image).
2. **Hardware Sizing:** Since OpenEMR is query-heavy but compute-light on the DB side, select a lightweight machine type (e.g., 2 vCPUs, 7.5–8 GB RAM) separate from your application server. Enable **Automatic Storage Increases** so you never risk a database lockout due to a full disk.
3. **Private IP Networking:** Under the connectivity settings, disable Public IP entirely. Enable **Private IP** and link it to your custom VPC. This ensures your database is physically invisible to the public internet and can only communicate with your OpenEMR VM inside the private network.
4. **Database & User Creation:** Create a clean database (e.g., `openemr_prod`) and provision a dedicated database user account with a strong, randomly generated password.

Step 2: Segregate Document and Media Storage

Medical practices generate large volumes of binary files (scanned medical records, insurance cards, PDFs, and imaging). Keeping these on the boot disk will eventually exhaust your system resources.

1. **Create an Additional Persistent Disk:** In GCP, provision a secondary **SSD Persistent Disk** (separate from your 60GB boot disk). Size it based on your expected imaging and document load (e.g., 100GB to start).
2. **Mount to Host:** Attach this disk to your VM instance and format it (typically using `ext4`). Mount it to a dedicated directory on the host, such as `/mnt/openemr-data/documents`.
3. **The Docker Volume Link:** When configuring your OpenEMR container, you will use a bind-mount to map this isolated directory into the exact location where OpenEMR stores physical files inside the container:YAML

```
volumes:
  - /mnt/openemr-data/documents:/var/www/localhost/openemr/sites/default/documents
```

(Note: For advanced high-availability setups where multiple compute instances scale dynamically, you would replace this local persistent disk with a Google Cloud Storage bucket mounted natively via GCSfuse or a shared Cloud Filestore NFS network volume).

Step 3: Modify the `docker-compose.yml` to reflect Decoupling

Because the state is now handled externally, your local Docker configuration simplifies dramatically. You completely remove the local database service block from your compose file.

Your decoupled `docker-compose.yml` will look similar to this:

YAML

```
version: '3.8'

services:
  openemr:
    image: openemr/openemr:7.0.2 # Use your targeted production version
    ports:
      - "8080:80" # Routed locally through your Nginx reverse proxy
    volumes:
      - /mnt/openemr-data/documents:/var/www/localhost/openemr/sites/default/documents
    environment:
      - MYSQL_HOST=10.x.x.x  # The PRIVATE IP address of your GCP Cloud SQL Instance
      - MYSQL_USER=openemr_db_user
```

```
      - MYSQL_PASS=YOUR_STRONG_SECRET_PASSWORD
      - MYSQL_DBNAME=openemr_prod
      - MYSQL_PORT=3306
      - OE_USER=admin
      - OE_PASS=YOUR_CLINIC_ADMIN_PASSWORD
    restart: always
```

The Operational Result

By implementing this split:

- **The application server is stateless:** If the VM OS fails or needs an upgrade, you can delete it, spin up a brand new instance, clone your compose file, and run it. The clinic will be fully operational again in minutes.
- **The database is resilient:** GCP manages MariaDB backups, point-in-time recovery, and security patches without causing downtime or resource contention on your main OpenEMR application server.

Original docker-compose.yml | openemr/openemr - Docker Image

```
# Use admin/pass as user/password credentials to login to openemr (from OE_USER and OE_PA
SS below)
# MYSQL_HOST and MYSQL_ROOT_PASS are required for openemr
# MYSQL_USER, MYSQL_PASS, OE_USER, MYSQL_PASS are optional for openemr and
#   if not provided, then default to openemr, openemr, admin, and pass respectively.
version: '3.1'
services:
  mysql:
    restart: always
    image: mariadb:11.8
    command: ['mariadbd','--character-set-server=utf8mb4']
    volumes:
    - databasevolume:/var/lib/mysql
    environment:
      MYSQL_ROOT_PASSWORD: root
  openemr:
    restart: always
    image: openemr/openemr:7.0.5
    ports:
    - 801:80
    - 4431:443
    volumes:
    - logvolume01:/var/log
    - sitevolume:/var/www/localhost/htdocs/openemr/sites
    environment:
      MYSQL_HOST: mysql
```

```
      MYSQL_ROOT_PASS: root
      MYSQL_USER: openemr
      MYSQL_PASS: openemr
      OE_USER: admin
      OE_PASS: pass
    depends_on:
    - mysql
volumes:
  logvolume01: {}
  sitevolume: {}
  databasevolume: {}
```