

# AI-03921 GCP Multi Docker Deployment

Owner	Ⓜ Rigel Arcayan
Tags	Project
Created time	June 11, 2026 7:32 PM

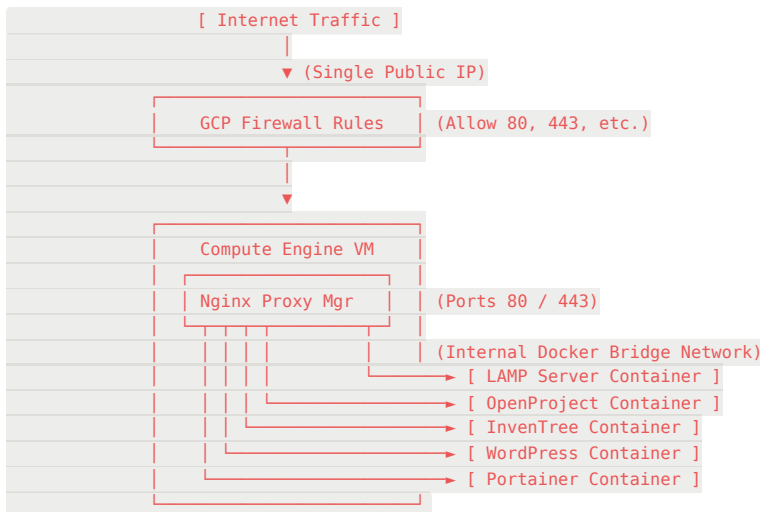
## Project goal

1. Design a Docker-based cloud architecture on Google Cloud Platform (GCP) that hosts the following services—WordPress, Portainer, Nginx Proxy Manager (NPM), InvenTree, OpenProject, and a LAMP-based web server—using a single public IP address.
2. Deliverable  
Provide detailed, step-by-step instructions and accompanying scripts to deploy, configure, and manage this multi-service Docker environment on GCP, from project setup to production-ready operation.

## High-Level Architecture Overview

To host multiple distinct web applications under a single public IP address, we will deploy a **Compute Engine** virtual machine on GCP.

The core of this architecture is **Nginx Proxy Manager (NPM)**. Operating as a reverse proxy, NPM listens on ports 80 and 443, intercepts incoming HTTP/HTTPS traffic, inspects the **Server Name (SNI)** in the request header (e.g., `wordpress.yourdomain.com`), and routes the traffic to the appropriate isolated Docker container over an internal Docker virtual bridge network.



## Step 1: Google Cloud Platform Setup

### 1.1 Create a Static External IP Address

Before launching the VM, reserve a static IP so your DNS records do not break if the instance reboots.

## Bash

```
gcloud compute addresses create multi-service-static-ip \  
  --region=us-central1 \  
  --network-tier=PREMIUM
```

*To find your reserved IP:*

## Bash

```
gcloud compute addresses describe multi-service-static-ip --region=us-central1 --format="value(address)"
```

## 1.2 Configure GCP Firewall Rules

We need to allow standard web traffic, temporary setup access to Portainer/NPM dashboards, and SSH.

## Bash

```
# Allow HTTP traffic  
gcloud compute firewall-rules create allow-http \  
  --allow=tcp:80 \  
  --target-tags=multi-service-host \  
  --description="Allow HTTP traffic"  
  
# Allow HTTPS traffic  
gcloud compute firewall-rules create allow-https \  
  --allow=tcp:443 \  
  --target-tags=multi-service-host \  
  --description="Allow HTTPS traffic"  
  
# Optional: Temporary direct access to NPM Admin Console (81) and Portainer (9443)  
# Best practice: Close these once your domain names are routed through the proxy.  
gcloud compute firewall-rules create allow-setup-dashboards \  
  --allow=tcp:81,tcp:9443 \  
  --target-tags=multi-service-host \  
  --description="Temporary setup access for NPM and Portainer"
```

## 1.3 Provision the Compute Engine Instance

Given the resource requirements of running multiple large stacks (especially OpenProject and individual databases), a minimum configuration of **e2-standard-4** (4 vCPUs, 16 GB RAM) is recommended.

## Bash

```
gcloud compute instances create docker-service-host \  
  --zone=us-central1-a \  
  --machine-type=e2-standard-4 \  
  --subnet=default \  
  --network-tier=PREMIUM \  
  --address=multi-service-static-ip \  
  --tags=multi-service-host \  
  --image-family=ubuntu-2204-lts \  
  --image-project=ubuntu-os-cloud \  
  --
```

```
--boot-disk-size=100GB \  
--boot-disk-type=pd-ssd
```

## Step 2: Host Environment Initialization

SSH into your new instance:

Bash

```
gcloud compute ssh docker-service-host --zone=us-central1-a
```

Execute the following script to update the system, install the official Docker Engine, and set up Docker Compose.

Bash

```
#!/bin/bash  
set -e  
  
# Update package database  
sudo apt-get update -y  
sudo apt-get upgrade -y  
  
# Install prerequisites  
sudo apt-get install -y ca-certificates curl gnupg lsb-release  
  
# Add Docker's official GPG key  
sudo mkdir -p /etc/apt/keyrings  
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg  
  
# Set up the stable repository  
echo \  
"deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/ubuntu \  
$(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null  
  
# Install Docker Engine & Compose plugin  
sudo apt-get update -y  
sudo apt-get install -y docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin  
  
# Verify installation  
sudo docker --version  
sudo docker compose version  
  
# Create application directory structure  
sudo mkdir -p /opt/docker-grid  
sudo chown -R $USER:$USER /opt/docker-grid  
cd /opt/docker-grid  
mkdir -p npm portainer wordpress inventree openproject lamp
```

## Step 3: Unified Docker Compose Deployment

We will deploy all applications using a single `docker-compose.yml` file located in `/opt/docker-grid/`. They will share a common user-defined bridge network called `proxy-tier`. This allows Nginx Proxy Manager to

resolve each application container internally using its container name as the hostname.

Create the file: `nano /opt/docker-grid/docker-compose.yml` and paste the following content:

YAML

```
version: '3.8'

networks:
  proxy-tier:
    name: proxy-tier
    driver: bridge

volumes:
  npm_data:
  npm_letsencrypt:
  portainer_data:
  wordpress_data:
  wordpress_db_data:
  inventree_data:
  inventree_db_data:
  openproject_pg_data:
  openproject_assets:
  lamp_web_data:
  lamp_db_data:

services:
# -----
# 1. Edge Reverse Proxy: Nginx Proxy Manager
# -----
  nginx-proxy-manager:
    image: 'jc21/nginx-proxy-manager:latest'
    container_name: nginx-proxy-manager
    restart: unless-stopped
    ports:
      - '80:80'
      - '443:443'
      - '81:81' # Admin Console port
    volumes:
      - npm_data:/data
      - npm_letsencrypt:/etc/letsencrypt
    networks:
      - proxy-tier

# -----
# 2. Container Management: Portainer CE
# -----
  portainer:
    image: 'portainer/portainer-ce:latest'
    container_name: portainer
    restart: unless-stopped
    security_opt:
      - no-new-privileges:true
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
      - portainer_data:/data
    ports:
      - '9443:9443' # Direct HTTPS port if needed during setup
    networks:
      - proxy-tier

# -----
# 3. CMS: WordPress & MySQL Stack
# -----
```

```

wordpress-db:
  image: mysql:8.0
  container_name: wordpress-db
  restart: unless-stopped
  environment:
    MYSQL_ROOT_PASSWORD: supersecret_wp_root_pass
    MYSQL_DATABASE: wordpress
    MYSQL_USER: wp_user
    MYSQL_PASSWORD: wp_secure_password
  volumes:
    - wordpress_db_data:/var/lib/mysql
  networks:
    - proxy-tier

wordpress:
  image: wordpress:latest
  container_name: wordpress
  restart: unless-stopped
  depends_on:
    - wordpress-db
  environment:
    WORDPRESS_DB_HOST: wordpress-db:3306
    WORDPRESS_DB_USER: wp_user
    WORDPRESS_DB_PASSWORD: wp_secure_password
    WORDPRESS_DB_NAME: wordpress
  volumes:
    - wordpress_data:/var/www/html
  networks:
    - proxy-tier

# -----
# 4. Inventory Management: InvenTree Stack
# -----
inventree-db:
  image: postgres:15
  container_name: inventree-db
  restart: unless-stopped
  environment:
    POSTGRES_DB: inventree
    POSTGRES_USER: inventree
    POSTGRES_PASSWORD: inventree_secure_password
  volumes:
    - inventree_db_data:/var/lib/postgresql/data
  networks:
    - proxy-tier

inventree:
  image: inventree/inventree:latest
  container_name: inventree
  restart: unless-stopped
  depends_on:
    - inventree-db
  environment:
    INVENTREE_DB_ENGINE: postgresql
    INVENTREE_DB_HOST: inventree-db
    INVENTREE_DB_PORT: 5432
    INVENTREE_DB_NAME: inventree
    INVENTREE_DB_USER: inventree
    INVENTREE_DB_PASSWORD: inventree_secure_password
    INVENTREE_SECRET_KEY: generated_random_secret_string_here
  volumes:
    - inventree_data:/home/inventree/data
  networks:
    - proxy-tier

# -----
# 5. Project Management: OpenProject

```

```

# -----
openproject:
  image: openproject/community:13
  container_name: openproject
  restart: unless-stopped
  environment:
    OPENPROJECT_SECRET_KEY_BASE: another_super_secure_random_key_base
    OPENPROJECT_HOST_NAME: openproject.yourdomain.com
    OPENPROJECT_HTTPS: "true"
  volumes:
    - openproject_pg_data:/var/openproject/pgdata
    - openproject_assets:/var/openproject/assets
  networks:
    - proxy-tier

# -----
# 6. Legacy Custom Development: LAMP Server Stack
# -----
lamp-db:
  image: mariadb:10.11
  container_name: lamp-db
  restart: unless-stopped
  environment:
    MARIADB_ROOT_PASSWORD: lamp_root_password
    MARIADB_DATABASE: custom_db
  volumes:
    - lamp_db_data:/var/lib/mysql
  networks:
    - proxy-tier

lamp-web:
  image: matray/php-apache:latest
  container_name: lamp-web
  restart: unless-stopped
  depends_on:
    - lamp-db
  volumes:
    - lamp_web_data:/var/www/html
  networks:
    - proxy-tier

```

## Launch the Environment

From `/opt/docker-grid/`, run the compose stack in detached mode:

Bash

```
sudo docker compose up -d
```

Verify all containers are up and green:

Bash

```
sudo docker compose ps
```

## Step 4: Routing & DNS Configuration

To make each service accessible under a clean URL, you must map your domains/subdomains to your single GCP static public IP address within your DNS registrar account (e.g., Cloudflare, Namecheap, Google Domains).

Create **A Records** pointing to your static public IP address as follows:

Host / Subdomain	Type	Destination
<code>npm.yourdomain.com</code>	A	<code>YOUR_GCP_STATIC_IP</code>
<code>portainer.yourdomain.com</code>	A	<code>YOUR_GCP_STATIC_IP</code>
<code>wordpress.yourdomain.com</code>	A	<code>YOUR_GCP_STATIC_IP</code>
<code>inventree.yourdomain.com</code>	A	<code>YOUR_GCP_STATIC_IP</code>
<code>openproject.yourdomain.com</code>	A	<code>YOUR_GCP_STATIC_IP</code>
<code>lamp.yourdomain.com</code>	A	<code>YOUR_GCP_STATIC_IP</code>

## Step 5: Reverse Proxy Configuration (Nginx Proxy Manager)

1. Open your web browser and navigate to `http://YOUR_GCP_STATIC_IP:81`.
2. Log in using the default NPM administrative credentials:
  - **Email:** `admin@example.com`
  - **Password:** `changingthis`
3. Complete the mandatory profile setup by specifying your personal administrator email address and a robust password.

### Adding Proxy Hosts

For each application, navigate to **Hosts** → **Proxy Hosts** → **Add Proxy Host** within the NPM control panel. Fill out the **Details** and **SSL** configurations precisely as outlined below:

App Subdomain	Scheme	Forward Hostname / IP	Forward Port	Essential Options to Enable	SSL Tab Selection
<code>npm.yourdomain.com</code>	<code>http</code>	<code>nginx-proxy-manager</code>	<code>81</code>	Block Common Exploits, WebSockets Support	Request a new SSL Certificate (Force SSL, HTTP/2)
<code>portainer.yourdomain.com</code>	<code>https</code>	<code>portainer</code>	<code>9443</code>	Block Common Exploits, WebSockets Support	Request a new SSL Certificate (Force SSL, HTTP/2)
<code>wordpress.yourdomain.com</code>	<code>http</code>	<code>wordpress</code>	<code>80</code>	Block Common Exploits	Request a new SSL Certificate (Force SSL, HTTP/2)
<code>inventree.yourdomain.com</code>	<code>http</code>	<code>inventree</code>	<code>8000</code>	Block Common Exploits	Request a new SSL Certificate (Force SSL, HTTP/2)

<code>openproject.yourdomain.com</code>	<code>http</code>	<code>openproject</code>	<code>80</code>	Block Common Exploits, WebSockets Support	Request a new SSL Certificate (Force SSL, HTTP/2)
<code>lamp.yourdomain.com</code>	<code>http</code>	<code>lamp-web</code>	<code>80</code>	Block Common Exploits	Request a new SSL Certificate (Force SSL, HTTP/2)

**Note on Host Resolution:** Because NPM and your apps occupy the exact same `proxy-tier` network layer, NPM maps directly to the container names (e.g., `wordpress`, `inventree`) without requiring host machine ports to be mapped or exposed publicly.

## Step 6: Post-Deployment Verification & Security Hardening

### 1. Close Setup Ports in GCP Firewall

Once your domains work securely via HTTPS (e.g., `https://npm.yourdomain.com`), remove the temporary direct access ports from your firewall rules to secure your environment.

Bash

```
gcloud compute firewall-rules delete allow-setup-dashboards --quiet
```

### 2. Verify Container Health and Auto-Restart Action

To test the environment's resilience, simulate a full system failure by rebooting your virtual machine engine:

Bash

```
sudo reboot
```

Wait 60 seconds, re-log into your server via SSH, and run:

Bash

```
sudo docker ps --format "table {{.Names}}\t{{.Status}}\t{{.Ports}}"
```

All containers should display an `Up` status, proving the container state logic handles system power events cleanly.

**Do a comprehensive of the entire GCP for 2 vCPU, 16 GB RAM, then apply CUD to show my cost per month.**

To model a cloud environment requiring **2 vCPUs and 16 GB of RAM**, we have a specific technical constraint: standard pre-configured Google Compute Engine shapes (like the `e2-standard-2`) utilize a \$1:4\$

ratio of vCPU to RAM, giving you 2 vCPUs and 8 GB of RAM.

To achieve a strict **2 vCPU / 16 GB RAM** split, we must look at two distinct provisioning paths on GCP:

1. **The Custom Machine Route:** Provisioning a custom E2 instance tailored exactly to 2 vCPUs and 16 GB RAM.
2. **The High-Memory Standard Route ( `e2-highmem-2` ):** Utilizing a standard pre-configured shape that natively pairs 2 vCPUs with 16 GB of RAM (\$1:8\$ ratio).

The following analysis details the cost structures for both configurations in the baseline **us-central1 (Iowa)** region, assuming \$24/7\$ continuous operation (730 hours/month), and applies both **1-Year** and **3-Year Committed Use Discounts (CUDs)**.

### Option 1: E2 Custom Instance (2 vCPUs, 16 GB RAM)

GCP allows you to explicitly define independent core and memory counts within the E2 cost-optimized family.

- **On-Demand Custom Pricing Rates:**
  - vCPU: \$0.022898 per vCPU/hour
  - RAM: \$0.003067 per GB/hour
- **Monthly Calculations (730 Hours):**
  - vCPUs:  $\$2 \times \$0.022898 \times 730 = \$33.43$
  - RAM:  $\$16 \times \$0.003067 \times 730 = \$35.82$

### Applied Committed Use Discounts (Resource-Based CUD)

Resource-based commitments for the E2 family yield a **37% discount** for a 1-year term and a **55% discount** for a 3-year term.

Commitment Term	Discount	Core Cost / mo	RAM Cost / mo	Total Compute Cost / month
On-Demand (Baseline)	0%	\$33.43	\$35.82	<b>\$69.25</b>
1-Year CUD	37%	\$21.06	\$22.57	<b>\$43.63</b>
3-Year CUD	55%	\$15.04	\$16.12	<b>\$31.16</b>

### Option 2: Pre-configured `e2-highmem-2` (2 vCPUs, 16 GB RAM)

If you prefer a standard, pre-packaged catalog item instead of typing in custom parameters, the `e2-highmem-2` matches your resource criteria exactly. Because it is a bundled standard shape, Google prices it slightly more aggressively than an exact custom-built equivalent.

### Applied Committed Use Discounts (Bundled Shape)

Commitment Term	Discount	Hourly Rate	Total Compute Cost / month
On-Demand (Baseline)	0%	\$0.090400	<b>\$65.99</b>
1-Year CUD	37%	\$0.056952	<b>\$41.57</b>
3-Year CUD	55%	\$0.040680	<b>\$29.70</b>

## The Reality Check: Total Cost of Ownership (TCO)

Compute resources do not live in a vacuum. To turn this instance into a production-ready operational asset, we must layer on baseline storage.

Assuming a lean, standard architecture block consisting of a **50 GB Balanced Persistent Disk (pd-balanced)** for system boot/logs and a fixed **Static External IP address**:

Infrastructure Component	Unit Price	Total Added Cost / month
50 GB Balanced PD Storage	\$0.10 per GB / month	\$5.00
Static Public IP Address (In-Use)	\$0.004 per hour	\$2.92
<b>Total Mandatory Base Add-ons</b>	—	<b>+\$7.92</b>

## Final Monthly Budget Projections ( **e2-highmem-2** + Storage + IP)

- **On-Demand Total:** \$65.99 + \$7.92 = **\$73.91 / month**
- **1-Year CUD Total:** \$41.57 + \$7.92 = **\$49.49 / month**
- **3-Year CUD Total:** \$29.70 + \$7.92 = **\$37.62 / month**

**Optimization Takeaway:** Selecting the pre-packaged **e2-highmem-2** over a custom machine definition trims roughly \$3.26 off the baseline on-demand monthly cost. Committing to a 3-Year CUD drives the total operational cost of this 2 vCPU / 16 GB RAM node down to under **\$38 a month**, including storage and its public interface address.

### Do I need to pay upfront? Final Monthly Budget Projections (e2-highmem-2 + Storage + IP)

**On-Demand Total:** \$65.99 + \$7.92 = **\$73.91 / month**

**1-Year CUD Total:** \$41.57 + \$7.92 = **\$49.49 / month**

**3-Year CUD Total:** \$29.70 + \$7.92 = **\$37.62 / month**

**No, you do not need to pay anything upfront.** Google Cloud Platform (GCP) structures its Committed Use Discounts (CUDs) as a **monthly recurring charge**, not an upfront lump-sum payment.

Here is exactly how the billing mechanics work for the three brackets:

### 1. On-Demand Total (\$73.91 / month)

- **How you pay:** Standard pay-as-you-go post-paid billing.
- **Mechanics:** Google tracks your exact usage down to the second. At the end of your monthly billing cycle, you are invoiced for the hours your instance, storage, and IP address were active. You can delete the VM at any time to stop the charges.

### 2. 1-Year or 3-Year CUD Totals (\$49.49 or \$37.62 / month)

- **How you pay:** Monthly installments over the life of the contract.
- **Mechanics:** When you sign up for a 1-year or 3-year CUD, you are contractually agreeing to pay for those 2 vCPUs and 16 GB of RAM for every single hour of that term. Google breaks this contract down into **predictable monthly line items** on your invoice.
- **The Catch:** There is no upfront premium, but the commitment is **non-cancelable**. Even if you stop or delete your **e2-highmem-2** container host intermediate through the contract, you will continue to see the discounted compute fee on your bill every month until the 1 or 3 years lapse.

## Summary Breakdown

The infrastructure storage fee (\$5.00) and the static IP reservation (\$2.92) are always billed on-demand monthly regardless of the plan. The only element changing is the compute charge, which changes from a flexible hourly rate to a fixed monthly fee:

- **Upfront cash required today:** \$0.00
- **First bill (approx. 30 days from now):** You will pay your chosen monthly tier price (\$73.91, \$49.49, or \$37.62) based on the contract length you selected in the console.

### Is this GCP 2 vCPUs and 16 GB of RAM economical compared to running my Dell Poweredge R720 server at home?

To determine if moving to GCP is truly economical compared to your Dell PowerEdge R720, we have to run a classic **Total Cost of Ownership (TCO)** math model.

An enterprise server at home doesn't charge you a flat subscription, but it extracts a continuous tax via **electricity, cooling, and hardware depreciation**.

Here is how the economics stack up when comparing a 3-Year GCP commitment against running an R720 in a home lab.

## 1. The Home Server Baseline: Dell PowerEdge R720

An R720 (typically equipped with dual Intel Xeon E5-2600 series processors) is a powerhouse, but it is architecturally older and highly inefficient by modern cloud data center standards.

- **Idle Power Draw:** Even under minimal load hosting just your Docker stacks, an R720 with dual power supplies, a few modules of DDR3 RAM, and a hard drive array idles at roughly **100 to 140 Watts**.
- **The Power Math (at an average US rate of \$0.16 per kWh):**  
$$\text{\$120} \times \text{24} \times \text{30.5} = \text{87.84} \text{ kWh per month} \text{\$}$$
$$\text{\$87.84} \times \text{\$0.16} = \text{\$14.05} \text{ per month in raw electricity} \text{\$}$$
- **The Hidden "AC Tax":** Every watt a server consumes is converted entirely into heat. Your home air conditioning has to work to remove that heat. In warmer climates or during summer months, this effectively adds a 30% to 50% premium onto the server's baseline power cost just for cooling (approx. **+\$4.00 to \$7.00/mo**).

**Estimated Bare Minimum Cost to Keep the R720 On: \$15.00 to \$21.00 / month**

## 2. Head-to-Head Comparison

Cost Factor	GCP Cloud (e2-highmem-2 3-Yr CUD)	Home Dell PowerEdge R720
Monthly Operating Cost	\$37.62 / mo (Fixed)	\$15.00 - \$21.00 / mo (Variable Power)
Upstream Internet Cost	Included	Uses home bandwidth (Symmetric fiber preferred, data caps may apply)
Public IP & DNS	Clean Static IP included	Dynamic DNS or extra ISP fee for Static IP
Hardware Failure Risk	0% (Google replaces dead nodes instantly)	High (Drive failures, power supply failures—you pay for replacements)

<b>Power/Network Redundancy</b>	\$99.99\%\$ Uptime (Enterprise generators + dual fiber)	Consumes home UPS battery; drops if home internet drops
<b>Available Resource Pool</b>	Strict <b>2 vCPUs / 16 GB RAM</b>	Massive (Typically 12 to 24 physical cores / 64GB+ RAM available)

### 3. The Financial Verdict

#### When the Home R720 Wins on Economics

If your Docker ecosystem expands beyond the 6 services you listed, or if your applications begin processing heavy workloads, the 2 vCPU / 16 GB limit on GCP will force you to scale up to a larger instance tier, doubling your cloud cost. The R720 already has a massive sunken hardware capacity. If you need 8 cores and 64 GB of RAM tomorrow, **the R720 costs the exact same \$18/month to run**, while GCP would skyrocket to over \$100+/month.

#### When GCP Wins on Economics

If your goal is strict "**Set it and forget it**" **high availability**, GCP wins. For the extra \$15 to \$20 a month over your electric bill, the cloud completely eliminates the "Catch-22" of home hosting:

- You don't have to configure complex reverse proxy configurations to punch holes through a residential ISP firewall.
- You get a pristine, dedicated static IP address that won't get blacklisted by corporate firewalls or mail servers.
- Your services stay online even if your home loses power or suffers a local internet outage.

#### Summary

If you only ever intend to run those specific 6 services and value **absolute mobility, zero noise, and zero hardware maintenance**, paying the \$37.62/month premium for GCP is highly economical. If you enjoy raw hardware overhead and want the freedom to spin up 10 more containers next month without opening your wallet, keep the R720 spinning.

**I have GCP 2 vCPUs and 2 GB of RAM, will I lose my work (docker apps) if I upgrade to GCP 2 vCPUs and 16 GB of RAM? Provide guidance.**

**No, you will not lose your work, your Docker containers, or any of your configuration data.** On Google Cloud Platform, the core operating system and all your data live on a separate component called a **Persistent Disk (PD)**. The virtual machine type (vCPU and RAM) is just a compute profile attached to that disk. Upgrading is the cloud equivalent of taking a hard drive out of an old computer and plugging it into a brand-new, higher-spec machine.

However, because changing the machine type requires a virtual hardware modification, **the VM must be stopped during the process**. This means your applications will experience a few minutes of downtime.

Here is the precise, step-by-step guidance to execute this migration seamlessly without losing a single line of code.

### Step-by-Step Upgrade Guide

#### Phase 1: Graceful Docker Shutdown (Recommended)

While GCP safely handles system shutdowns, it is always a best practice to gracefully stop your Docker containers first to ensure database engines (like MySQL or PostgreSQL) write all cached data to the disk.

1. SSH into your current 2 vCPU / 2 GB instance.
2. Navigate to your project directory and halt the containers: Bash

```
cd /opt/docker-grid
sudo docker compose down
```

3. Close the SSH connection.

## Phase 2: Upgrading the Machine Type in GCP

You can complete this phase via the Google Cloud Console UI or straight from your terminal using the `gcloud` CLI.

### Method A: Using the GCP Console (Web Web Browser)

1. Go to the **Compute Engine** → **VM Instances** page.
2. Click on the checkbox next to your instance name and click the **Stop** button at the top of the screen. Wait for the green checkmark to change to a gray stopped icon.
3. Click on the **Instance Name** to open its details page, then click **Edit**.
4. Scroll down to the **Machine Configuration** section.
5. Change the **Machine Type** from your current custom configuration to **Standard** or **High-Memory**, and select `e2-highmem-2` (which natively maps to 2 vCPUs and 16 GB of RAM).
6. Scroll to the very bottom of the page and click **Save**.
7. Click **Start / Resume** at the top of the page to power the machine back up.

### Method B: Using the `gcloud` CLI

If you prefer using the terminal command line, run these three commands sequentially:

Bash

```
# 1. Stop the instance
gcloud compute instances stop docker-service-host --zone=us-central1-a

# 2. Alter the machine type to e2-highmem-2
gcloud compute instances set-machine-type docker-service-host \
  --zone=us-central1-a \
  --machine-type=e2-highmem-2

# 3. Restart the instance
gcloud compute instances start docker-service-host --zone=us-central1-a
```

## Phase 3: Post-Upgrade Verification

Once the instance boots up (usually takes 30 to 45 seconds):

1. SSH back into the newly upgraded machine.
2. Confirm that the operating system recognizes the full 16 GB of RAM: Bash

```
free -h
```

*You should see roughly `15Gi` available under the total memory column.*

3. Navigate back to your configuration folder: Bash

```
cd /opt/docker-grid
```

4. Relaunch your entire multi-service stack: Bash

```
sudo docker compose up -d
```

Because your original configuration utilized `restart: unless-stopped` flags inside the `docker-compose.yml` file, the containers may even auto-start on boot, but running `up -d` ensures everything syncs cleanly with the newly provisioned hardware boundaries. Your Nginx Proxy Manager, WordPress, and database components will instantly adapt to the expanded memory limits.