AI-04029 Google Cloud Journey

Owner	Ⓡ Rigel Arcayan
Tags	Project
Created time	July 14, 2026 9:14 AM

I am new to Google Cloud Platform and on Free Trial. I setup my Ubuntu Dockerlab with NPM, Portainer, Mariadb, Postgres, pgadmin, phpmyadmin, OpenEMR, OpenProject, Docmost, Zammad. I've configured GCP proxy, open some ports, SSH to GCP from VSCode. Is it time for me to move to Terraform training? Or there is another path I have to take before Terraform? Explain and create a High level and low level roadmap. I'm always careful in my learning of new tech not to jump over basic underlying knowledge and skills. I'm on a tight budget and willing to spend >$40 dollars per month for GCP. Will using cloud sql breaks my bank? Explain to me how to execute my GCP training without overspending per month. I am retired and all I'm doing is just a hobby and learning (nothing else). Also, I don't want to run a Dell Poweredge R720 in my condo and configure all the network and tailscale, GCP is my answer to my new found toy. Enlighten me on my desire to master GCP with the applications I mentioned.

You have built a solid foundation. Setting up a modular stack (NPM, Portainer, databases, and heavy-duty applications like OpenEMR and OpenProject) on an Ubuntu VM, configuring a GCP proxy, mapping ports, and tying it into VSCode via SSH is an excellent first milestone.

Is it time for Terraform? Yes and no.

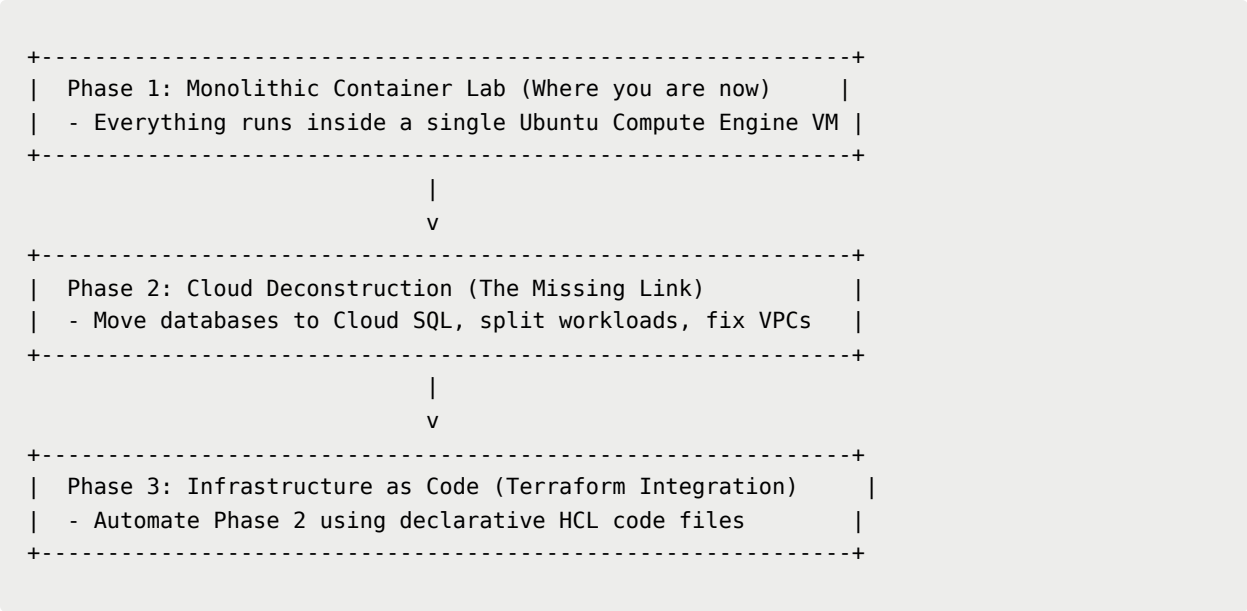
While you have the command-line confidence and container knowledge to grasp Terraform, jumping straight into it right now presents a major hurdle: **you don't have enough cloud-native resources to manage yet.** Right now, your entire ecosystem lives inside a single Virtual Machine (VM). Terraform's true power is

managing *cloud* infrastructure (networks, managed databases, firewalls, and storage buckets), not configuring the software inside a single Ubuntu OS.

If you use Terraform immediately, you'll just write code to spin up that exact same single VM, which skips the actual structural design patterns you need to master.

Before automating the cloud with code, you should learn to architect it manually. Here is the strategic roadmap to transition you from a single-server setup to an automated cloud infrastructure specialist.

High-Level Roadmap: The Cloud Progression



Low-Level Roadmap: Step-by-Step Implementation

Phase 1: Harden & Observe Your Current Stack

Before changing the architecture, treat your current lab like a production environment.

- **Version Control:** Commit your entire setup (Docker Compose files, reverse proxy configurations, and scripts) to a private GitHub repository. **Never** commit your database root passwords or API keys; use environment variables (`.env`).

- **Centralized Logging:** Stop using `docker logs` manually. Enable Google Cloud Logging on your VM instance so you can monitor your stack directly from the GCP Console.

Phase 2: Cloud Deconstruction (Prerequisites for Terraform)

To understand what Terraform is automating, you need to break your single-server setup into native cloud components using the web console first.

- **Isolate the Databases:** Pull MariaDB and Postgres out of Docker. Manually provision a **GCP Cloud SQL** instance. Configure your Docker containers on the VM to communicate securely with Cloud SQL via the Cloud SQL Auth Proxy.
- **Master Networking (VPC):** Delete your generic default network. Create a custom Virtual Private Cloud (VPC) with a public subnet (for your reverse proxy/NPM) and a private subnet (for your heavy applications and databases). Learn how to write explicit ingress/egress firewall rules.
- **Object Storage Integration:** Configure apps like Docmost or Zammad to use **Google Cloud Storage (GCS) buckets** for attachments and file uploads instead of local directories.

Phase 3: The Terraform Transition

Once you understand how VPCs, Compute Engines, Cloud SQL, and Firewalls manually plug together, you are ready for Terraform.

- **Learn HCL Syntax:** Get familiar with HashiCorp Configuration Language (variables, resource blocks, outputs, and providers).
- **State Management:** Learn how Terraform keeps track of your infrastructure. Move your local `terraform.tfstate` file to a secure, remote GCS bucket to learn team-style execution.
- **The "One-Click" Lab:** Write a Terraform configuration that provisions your entire custom VPC, firewall rules, GCS buckets, and Compute Engine VM from scratch with a single `terraform apply` command.

> ⚠️ **Free Trial Budget Warning:** Be highly vigilant when moving to Phase 2. Google Cloud SQL and multiple custom IP addresses can exhaust your $300 Free Trial credits much faster than a single e2-micro VM. When practicing

> Phase 2 manually, or Phase 3 with Terraform, always remember to tear down or stop resources when you aren't actively studying them.

You exactly answered my question and concern, I'm always careful in my learning of new tech not to jump over basic underlying knowledge and skills. I'm on a tight budget and willing to spend >$40 dollars per month for GCP. Will using cloud sql breaks my bank? Explain to me how to execute my GCP training without overspending per month. I am retired and all I'm doing is just a hobby and learning (nothing else). Also, I don't want to run a Dell Poweredge R720 in my condo and configure all the network and tailscale, GCP is my answer to my new found toy. Enlighten me on my desire to master GCP with the applications I mentioned.

It sounds like you have found the perfect sandbox. Escaping the noise, heat, and complex physical networking of a rack-mounted server like a Dell PowerEdge R720 in a condo is one of the best reasons to embrace public cloud. You get all the infrastructure toys, but none of the hardware headaches.

To your main concern: **Will Cloud SQL break your bank?**

If left running 24/7, a standard production-grade Cloud SQL instance absolutely will wipe out a $40/month budget. However, because you are running a personal learning lab, you can absolutely split your database out to Cloud SQL without spending a fortune—**provided you learn how to play by cloud rules.**

💰 The Cloud SQL Cost Breakdown

Cloud SQL pricing is unbundled. You pay for the instance size (compute/RAM), the provisioned disk storage, and the public IP address if it sits idle.

If you use the smallest possible settings for your learning exercises:

- **Machine Type:** `db-f1-micro` (Shared CPU, 0.6 GB RAM) costs roughly **$0.015 per hour** (~$11/month if kept on 24/7).
- **Storage:** 10 GB of SSD storage at roughly **$0.17 per GB/month** costs about **$1.70/month**.
- **The Hidden Trap (Idle IP Charge):** GCP charges roughly **$0.01 to $0.013 per hour** (~$7 to $9.50/month) for a public IPv4 address assigned to a database that isn't actively handling traffic or is stopped.

> 📉 **The Constant 24/7 Math:** One micro instance running constantly will hover around **$13 to $20/month** depending on your exact configuration and region. If you provision *both* a Postgres instance (for Outline/Docmost) and a MariaDB/MySQL instance (for OpenEMR), running them 24/7 puts you at $30–$40 immediately, eating your entire budget before accounting for your Ubuntu VM.

🛡️ How to Execute Training Without Overspending

As a hobbyist, your primary cost-saving superpower is simple: **Turn things off when you aren't sitting at your desk.** Cloud billing is down to the second. If an instance only runs while you are actively learning, it costs next to nothing.

1. The Scripted Sandbox Strategy (The "Learn & Burn" Method)

Instead of treating your cloud lab like a permanent data center, treat it like a sketchpad.

- **The Workflow:** Keep your Docker Compose files and custom data safe on your local computer or a GitHub repository. Use your VM and Cloud SQL solely to test configuration patterns.
- **The Action:** When you finish a 3-hour study session, don't just stop the database—**delete the Cloud SQL instance entirely**. Because you're only using minor testing databases with minimal dummy data, spinning them back up takes less than 5 minutes. Deleting the instance destroys the storage and frees the IP address, dropping your active cost to exactly **$0.00** the moment you log off.

2. Move to a Single Shared Cloud Database

Applications like Zammad, OpenProject, and OpenEMR each expect their own database. But you don't need a separate Cloud SQL machine for each one.

- Provision **one** single `db-f1-micro` Postgres instance.
- Log into it and create separate database schemas/names inside that single instance (e.g., `db_openproject`, `db_zammad`). This keeps your architecture decoupled at the application layer while keeping your GCP bill anchored to a single tiny asset.

3. Establish Billing Defenses Immediately

Before deploying another container, set up automated alarms in your billing dashboard:

- Go to **Billing > Budgets & Alerts**.
- Create a budget set strictly to **$40**.
- Configure thresholds to trigger emails to you at 50%, 80%, and 100% of your budget. You can even write simple Cloud Functions later that automatically shut off all running VMs if your budget hits 90%.

🚀 The Joy of Mastering This Stack

The specific applications you've chosen are fantastic choices for mastering GCP because they demand different infrastructure layers:

Application	The Infrastructure Muscle It Flexes
Nginx Proxy Manager	Master public vs. private subnets, SSL certificate automation, and VPC firewalls.
OpenProject / Zammad	Heavy enterprise Ruby/Java stacks. Teaches you how to decouple standard file storage by using a **Google Cloud Storage (GCS) bucket** instead of filling up your local VM's hard drive.
OpenEMR	A highly structured, secure app. Perfect for testing secure internal networking—making sure the web container can see the internet, but the database is deeply buried inside a private subnet with zero outside exposure.

By moving step-by-step—manually pulling the databases out of your Ubuntu VM into Cloud SQL, mapping the file attachments to Cloud Storage buckets, and then using Terraform to automate that layout—you are interacting with the cloud exactly how modern engineers do. You get all the analytical satisfaction of solving a complex puzzle, right from your desk, completely under your financial control.

Master Prompt: Building GCP Cloud Engineering Skills Through a Personal DockerLab (Beginner → Advanced Roadmap)

I am a beginner in Google Cloud Platform (GCP) and currently using the GCP Free Trial. My goal is not to become a professional cloud engineer immediately,

but to master cloud infrastructure concepts through hands-on experimentation, self-hosted applications, DevOps practices, and continuous learning.

My Current GCP Environment

I have already deployed an Ubuntu-based DockerLab in GCP and successfully configured:

Ubuntu Linux server
Docker Engine
Docker Compose
NGINX Proxy Manager (NPM)
Portainer
MariaDB / MySQL
PostgreSQL
pgAdmin
phpMyAdmin
OpenEMR
OpenProject
Docmost
Zammad

I have also completed some initial GCP tasks:

Created a GCP VM instance
Configured firewall rules
Opened required ports
Configured GCP networking basics
Connected to GCP using SSH through VSCode
Exposed self-hosted applications through NGINX Proxy Manager
Began understanding cloud-hosted infrastructure

I am building this as a personal cloud laboratory, not a commercial production environment.

My Main Question

Am I ready to start learning Terraform Infrastructure as Code (IaC)?

Or should I first strengthen other foundational GCP, Linux, networking, security, Docker, and DevOps concepts before moving into Terraform?

I want to avoid a common mistake in technology learning: jumping into advanced tools without understanding the underlying principles.

Please evaluate my current skill level and recommend the correct learning sequence.

Learning Philosophy

My learning style is:

Deep understanding over rushing
Hands-on labs over passive video watching
Understanding "why" before memorizing "how"
Building real systems instead of toy examples
Learning the fundamentals behind every abstraction

My goal is to eventually understand:

Cloud architecture
Infrastructure automation
DevOps workflows
Container orchestration
Database administration
Security principles
Reliable application hosting
Create a Complete GCP Mastery Roadmap

Please create:

Part 1 — Current Skill Assessment

Evaluate where I am today:

Beginner
Intermediate
Advanced beginner
Ready for Terraform?

Explain:

What skills I already demonstrated
What gaps remain
What mistakes beginners commonly make at this stage
Part 2 — High-Level Learning Roadmap

Create a roadmap from my current level to advanced GCP mastery.

Organize it into phases:

Example:

Phase 1 — Cloud Fundamentals

(GCP foundation)

Topics:

Projects
Billing
IAM
Regions/zones
VPC networking
Firewall rules
Compute Engine
Persistent disks
Snapshots
Cloud Storage
Phase 2 — Linux and Infrastructure Foundation

Topics:

Ubuntu administration
System services
SSH security
Users/groups
Permissions
Logs
Monitoring
Backup strategy
Phase 3 — Docker and Container Mastery

Topics:

Images
Containers
Volumes
Networks

Docker Compose
Container security
Registry concepts
Phase 4 — Infrastructure as Code

Terraform:

Why Terraform exists
Terraform architecture
Providers
Resources
Variables
State files
Modules
Remote state
Best practices
Phase 5 — Cloud-Native Architecture

Topics:

Kubernetes
GKE
Managed databases
Load balancing
Cloud networking
Observability
Phase 6 — Production-Level DevOps

Topics:

CI/CD
GitHub Actions
Secrets management
Monitoring
Disaster recovery
Security hardening
Part 3 — Low-Level Hands-On Roadmap

Create a detailed step-by-step lab progression.

For example:

Lab 1:
Create GCP VM manually

Lab 2:
Secure Ubuntu

Lab 3:
Deploy Docker applications

Lab 4:
Create Terraform configuration for the VM

Lab 5:
Rebuild entire DockerLab using Terraform

Lab 6:
Move databases to managed services

Lab 7:
Implement backup and disaster recovery

Include:

Objective
Skills learned
Tools used
Expected outcome
Difficulty level
Part 4 — Should I Use Cloud SQL?

Analyze my situation:

I currently run:

PostgreSQL Docker container
MariaDB Docker container

Should I migrate to:

Cloud SQL PostgreSQL?
Cloud SQL MySQL?
Keep databases inside Docker?

Explain:

Technical tradeoffs

Compare:

Option Advantages Disadvantages
Self-hosted Docker DB
Cloud SQL
Hybrid approach

Consider:

Cost
Learning value
Maintenance
Backup
Security
Real-world cloud skills
Part 5 — GCP Cost Optimization Strategy

My budget:

I am retired
This is a hobby and learning project
I do not need enterprise production availability
I want to stay around $40/month or less if possible

Create a realistic monthly budget plan.

Include:

VM sizing recommendations
Disk sizing
When to stop/start instances
Free tier opportunities
Services to avoid initially
Services worth paying for

Explain:

Would Cloud SQL realistically fit my budget?

Example:

"Spend $10/month here, $15/month there..."

Part 6 — Terraform Timing Decision

Give me a direct recommendation:

Should I:

A. Start Terraform now?

B. Wait 1–3 months?

C. Master another technology first?

Explain the reasoning.

If Terraform is appropriate now, create my first Terraform project:

Example:

terraform/
├── main.tf
├── variables.tf
├── outputs.tf
├── terraform.tfvars

Explain each file.

Part 7 — My Long-Term Goal

My vision:

I do not want to run a large Dell PowerEdge R720 server in my condo anymore.

I want GCP to become my personal "infinite learning playground."

My goal is to master:

GCP
Linux
Docker
Terraform
PostgreSQL
MySQL/MariaDB
DevOps
Cloud architecture
Self-hosted applications

Using real applications:

OpenEMR
OpenProject
Docmost
Zammad

Explain how these applications can become my practical cloud engineering training environment.

Final Deliverable

Create:

A GCP Beginner → Master roadmap
A Terraform readiness assessment
A 12-month learning plan
A monthly GCP cost-control strategy
A hands-on lab sequence
Recommended books, courses, and documentation
Common mistakes to avoid

Approach this as if you are my senior cloud architect mentor guiding a retired technology professional who wants deep mastery through curiosity, experimentation, and disciplined learning.

Welcome to the cloud, my friend. Honestly, looking at your setup, you have built an incredibly solid foundation. Many engineers entering the cloud space spend months looking at presentation slides without ever touching a configuration file; you have already wrestled with container networking, reverse proxies, and database persistence. That practical friction is where real learning happens.

Let's approach this as a design review between peers. You are not trying to pass a multiple-choice certification exam; you want to truly understand the mechanics beneath the abstractions.

Here is your comprehensive architectural roadmap to transition from manual deployment to advanced cloud engineering.

Part 1: Current Skill Assessment

Where You Stand Today

You are firmly in the **Advanced Beginner transitioning to Intermediate** stage of cloud infrastructure.

```
[ Beginner ] ──> [ Advanced Beginner ] ──> [ Intermediate ] ──> [ Advanced ]
                          ▲
                          |
                  (You are here)
```

By manually orchestrating a multi-container environment (OpenEMR, OpenProject, Zammad) behind NGINX Proxy Manager (NPM), you have demonstrated a strong grasp of service connectivity, data persistence (volumes), and port mapping.

Are You Ready for Terraform?

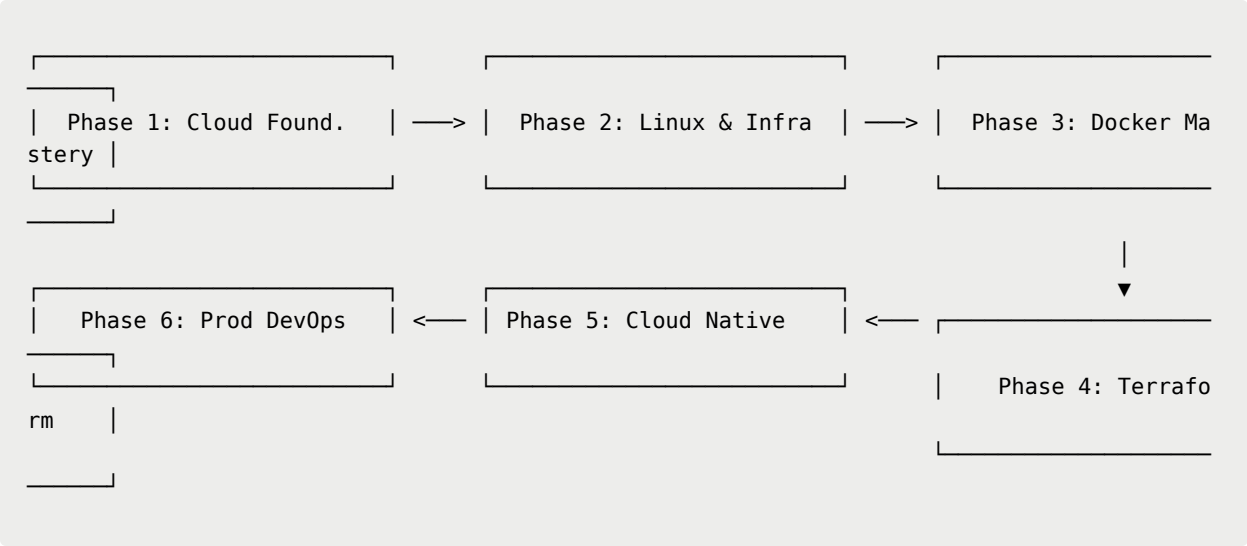
Yes, absolutely. You have met the absolute prerequisite for Infrastructure as Code (IaC): **you know how to build the infrastructure manually first.** If you don't know what a Virtual Private Cloud (VPC), a firewall rule, or a Compute Engine instance looks like in the GCP Console, writing Terraform is just guessing syntax. Because you have suffered through clicking the buttons and configuring the OS, you will instantly appreciate *why* Terraform exists.

Skill Gaps & Common Mistakes to Avoid At This Stage

- **The "ClickOps" Dependency:** The moment you modify a manually created infrastructure component via the GCP Console after writing Terraform for it, you introduce "configuration drift." You must learn to treat the console as a read-only dashboard.
- **Hardcoded Secrets:** Right now, your Docker Compose files likely contain plaintext database passwords. Moving to the cloud requires an immediate shift to environment variables and secret management.
- **Assuming Cloud Immutability:** In a home lab, a server stays up for years. In the cloud, instances are ephemeral. If your VM dies tonight, can you restore your entire stack in ten minutes using only your code and a backup? If not, that is your primary architectural gap.

Part 2: High-Level Learning Roadmap

This 12-month framework transitions you from a manual systems administrator to a cloud-native architect, spending roughly 1.5 to 2 months per phase.



Phase 1: Cloud Fundamentals (GCP Foundation)

- **Core Concepts:** Resource Hierarchy (Organization $\rightarrow$ Folder $\rightarrow$ Project), Billing Accounts, IAM Roles vs. Permissions (Principle of Least Privilege).
- **Networking:** Custom VPC creation, subnets, implicit firewall rules vs. user-defined rules, Cloud NAT (keeping VMs private while allowing internet outbound).
- **Compute & Storage:** Machine types, Persistent Disk types (pd-standard vs. pd-balanced), snapshot schedules, and Cloud Storage bucket lifecycle policies.

Phase 2: Linux and Infrastructure Foundation

- **Administration:** Mastering `systemd` (creating custom service files), log rotation (`logrotate`), and standard Linux directory structures (`/var/log`, `/etc`, `/opt`).
- **Security & Monitoring:** Hardening SSH (`sshd_config`, disabling password auth), utilizing standard tools (`journalctl`, `htop`, `iotop`), and understanding user/group permissions (`chmod`/`chown` deep dive).

Phase 3: Docker and Container Mastery

- **Under the Hood:** How namespaces and cgroups isolate processes.
- **Networking:** Deep dive into Docker bridge networks, overlay networks, and embedded DNS resolution.
- **Optimization:** Multi-stage builds, writing minimal `Dockerfile` configs using Alpine/Distroless bases, and scanning images for vulnerabilities.

Phase 4: Infrastructure as Code (Terraform)

- **Architecture:** The core loop (Write $\rightarrow$ Plan $\rightarrow$ Apply), declarative vs. imperative state management.
- **State Management:** The purpose of `terraform.tfstate`, concurrency locking with Cloud Storage, and how to import existing resources.
- **Modularity:** Designing reusable code blocks for networks and compute instances.

Phase 5: Cloud-Native Architecture

- **Managed Services:** Shifting from self-hosted containers to cloud native abstractions: Cloud Run (Serverless containers), HTTP(S) Load Balancing, and Cloud SQL.
- **Kubernetes (GKE):** Understanding the control plane, pods, deployments, services, and persistent volume claims.

Phase 6: Production-Level DevOps

- **Automation:** GitHub Actions pipelines to lint Terraform code and build/push Docker images to Google Artifact Registry.
- **Observability:** Setting up Prometheus and Grafana or using GCP Cloud Logging and Monitoring agents to build operational dashboards.

Part 3: Low-Level Hands-On Roadmap

Execute these labs sequentially. Do not move to the next until the current one works flawlessly.

Lab 1: The Pristine Manual VPC & VM

- **Objective:** Build a secure networking foundation manually before introducing automation.
- **Tools Used:** GCP Console.
- **Method:** Create a custom VPC (do not use the default one). Create a private subnet. Add a firewall rule allowing *only* your specific home public IP address to access port 22. Deploy an `e2-micro` or `e2-small` Ubuntu VM inside this subnet.
- **Expected Outcome:** Successful SSH connection without exposing the instance to the wider public internet.
- **Difficulty:** Beginner.

Lab 2: Hardening & Monitoring the Node

- **Objective:** Transform a stock cloud image into a resilient host server.
- **Tools Used:** Linux CLI, `systemd`, `fail2ban`.
- **Method:** Edit `/etc/ssh/sshd_config` to disable root login and password authentication. Install `fail2ban`. Configure a cron job or systemd timer to log resource utilization (`df`, `free`, `uptime`) to a central file daily.
- **Expected Outcome:** A secure OS layer that automatically drops malicious authentication attempts.
- **Difficulty:** Advanced Beginner.

Lab 3: The Automated Git-Driven App Deployment

- **Objective:** Separate application code from server state.
- **Tools Used:** Git, Docker Compose, NGINX Proxy Manager.
- **Method:** Organize your application configurations (OpenProject, Docmost, etc.) into an upstream Git repository. Clone this repo onto the VM. Utilize environment files (`.env`) to store database credentials outside the main compose file.

- **Expected Outcome:** Running `docker compose up -d` completely spins up your application array without manual container interventions.
- **Difficulty:** Advanced Beginner.

Lab 4: The Terraform Mirror

- **Objective:** Write code that mirrors the exact manual setup from Lab 1.
- **Tools Used:** Terraform, GCP Provider.
- **Method:** Write a Terraform configuration defining your VPC, subnet, firewall rules, and a single VM instance. Run `terraform plan` to ensure it targets your specific GCP subscription correctly. **Do not apply it yet.**
- **Expected Outcome:** A verified execution plan matching your live infrastructure topology.
- **Difficulty:** Intermediate.

Lab 5: Destruction and Rebirth (The Stateless Test)

- **Objective:** Prove your infrastructure is completely defined by code.
- **Tools Used:** Terraform, Docker Compose.
- **Method:** Manually delete your entire GCP project or infrastructure components. Run `terraform apply` to rebuild the underlying hardware, then pull your Git repo from Lab 3 to restore your applications.
- **Expected Outcome:** The entire stack is completely rebuilt from absolute scratch inside 15 minutes.
- **Difficulty:** Intermediate.

Lab 6: The Stateful Decoupling (Database Extraction)

- **Objective:** Isolate processing logic from persistent data storage.
- **Tools Used:** Terraform, PostgreSQL/MariaDB dump utilities.
- **Method:** Use Terraform to provision a private Cloud SQL instance or a second isolated DB host VM. Export data from your existing self-hosted Docker

containers, import it into the new database endpoint, and update your application compose configurations to point to the new target.

- **Expected Outcome:** Your applications function perfectly while their native database containers are permanently spun down.
- **Difficulty:** Advanced.

Lab 7: Automated Disaster Recovery Execution

- **Objective:** Establish operational peace of mind without manual oversight.
- **Tools Used:** Restic or Duplicity, Google Cloud Storage (GCS).
- **Method:** Write a shell script that dumps all application persistent volumes, compresses them, encrypts them, and pushes them directly to a GCS bucket using `gsutil`. Automate this execution with a Linux systemd timer.
- **Expected Outcome:** Daily, encrypted backups resting safely in durable object storage with automatic data lifecycle policies.
- **Difficulty:** Advanced.

Part 4: Should I Use Cloud SQL?

Decoupling databases from compute nodes is a fundamental cloud design principle, but your context requires evaluating specific trade-offs.

Technical Trade-offs Matrix

Strategy	Advantages	Disadvantages	Best For
Self-Hosted Docker DB	• Absolutely free (uses existing VM compute) • Zero network latency overhead • Simple local loopback backups	• Consumes high memory/CPU • Blurs app and data boundaries • Single point of host failure	Budget-focused labs and lightweight app testbeds.
Managed Cloud SQL	• Fully automated backups & updates • High Availability (HA) failover options	• **Extremely expensive** for a hobby budget	Enterprise production environments with zero downtime targets.

	• Offloads host RAM usage	• Requires intricate private VPC peering	
Hybrid Approach *(Dedicated DB VM)*	• Isolate processing from data • Highly accurate real-world architecture simulator • Predictable pricing	• Manual maintenance required • Requires configuring secure internal network links	Deeply learning database administration without the massive cloud premium.

Architectural Recommendation

For your specific goal and budget: Keep your databases inside Docker on your main instance, or deploy a second small, dedicated VM for your databases via Terraform.

Managed Cloud SQL carries a high cost baseline that will rapidly consume your monthly budget. Running databases inside Docker or on a separate standard VM gives you excellent training in database optimization, connectivity mapping, and manual backup configuration—skills that are highly valuable and often overlooked when relying entirely on managed platform abstractions.

Part 5: GCP Cost Optimization Strategy

To maintain a strict baseline of **under $40/month** as a hobby learning platform without enterprise availability demands, you must scale your hardware intelligently.

Recommended Monthly Budget Allocation

```
[ Total Budget: $40.00 ]
 ├── Compute: e2-standard-2 VM (2 vCPU, 8 GB RAM)  ─── $25.00 (Assuming ~50% uptime / sch
eduling)
 ├── Storage: 50 GB Balanced Persistent Disk       ─── $ 5.00
 ├── Backups: 100 GB Cloud Storage (Nearline)       ─── $ 1.50
 └── Network: Egress & Static Public IP Allocation  ─── $ 3.50
 ──────────────────────────────────────────────────────────
 Remaining Buffer for Experimentation:                $ 5.00
```

Strategic Recommendations for Cost Control

- **Instance Sizing:** Heavy enterprise applications like Zammad, OpenProject, and OpenEMR running concurrently will easily saturate standard shared-core `e2-micro` or `e2-medium` instances. Use an `e2-standard-2` **(2 vCPUs, 8 GB RAM)**.
- **The Lifesaver Strategy (Instance Scheduling):** An `e2-standard-2` costs roughly $48/month if run continuously 24/7. However, if you configure a GCP Instance Schedule to automatically turn the VM **ON at 07:00** and **OFF at 22:00**, you slash your compute cost by nearly 40%, bringing it down to roughly **$28/month**.
- **Storage Tiering:** Avoid standard SSD options (`pd-ssd`). Use `pd-balanced` for your primary boot disk—it offers a sweet spot of performance per dollar. Route backups to **Nearline** or **Coldline** Cloud Storage buckets.
- **Services to Absolute Avoid Initially:**
  - *Cloud SQL:* Baselines at ~$35+/month alone.
  - *GKE (Standard Cluster Control Plane):* Costs roughly $73/month flat fee just to keep the master nodes alive (though the GKE Autopilot tier offers a free cluster per billing account, pod costs accumulate quickly).
  - *NAT Gateways:* Cloud NAT charges an hourly fee for the gateway itself plus processing fees. Use a public IP with strict firewall rules instead to keep data routing free.

Part 6: Terraform Timing Decision

The Verdict: Option A (Start Terraform Now)

You have hit the sweet spot. You understand the components, you know how they talk to each other, and you are ready to abstract them. Delaying it will only cause you to build more manual patterns that you will eventually have to tear down anyway.

Your Initial Blueprint Project

Create a local directory named `gcp-dockerlab` and structure it with these four baseline files:

```
gcp-dockerlab/
 ├── main.tf
 ├── variables.tf
 ├── outputs.tf
 └── terraform.tfvars
```

`main.tf`

This file defines the actual infrastructure components you want GCP to create.

Terraform

```
terraform {
  required_version = ">= 1.5.0"
  required_providers {
    google = {
      source  = "hashicorp/google"
      version = "~> 5.0"
    }
  }
}

provider "google" {
  project = var.project_id
  region  = var.region
  zone    = var.zone
}

# Create a custom learning VPC
resource "google_compute_network" "custom_vpc" {
  name                    = "learning-vpc"
  auto_create_subnetworks = false
}

# Create a secure subnet
resource "google_compute_subnetwork" "lab_subnet" {
  name          = "lab-subnet"
  ip_cidr_range = "10.0.1.0/24"
  region        = var.region
  network       = google_compute_network.custom_vpc.id
}

# Secure entry point rule
resource "google_compute_firewall" "allow_ssh" {
  name    = "allow-ssh-to-lab"
  network = google_compute_network.custom_vpc.name
```

```
  allow {
    protocol = "tcp"
    ports    = ["22", "80", "443"]
  }

  source_ranges = [var.my_home_ip]
}

# The Compute Node
resource "google_compute_instance" "docker_host" {
  name         = "dockerlab-host"
  machine_type = var.machine_type
  zone         = var.zone

  boot_disk {
    initialize_params {
      image = "ubuntu-os-cloud/ubuntu-2204-lts"
      size  = 50
      type  = "pd-balanced"
    }
  }

  network_interface {
    subnetwork = google_compute_subnetwork.lab_subnet.id
    access_config {
      // Ephemeral public IP assignment for external routing
    }
  }
}
```

`variables.tf`

This file declares your configuration parameters, acting as the inputs for your infrastructure definitions.

Terraform

```
variable "project_id" {
  type        = string
  description = "The specific GCP Project ID target."
}

variable "region" {
  type    = string
  default = "us-central1"
}
```

```
variable "zone" {
  type    = string
  default = "us-central1-a"
}

variable "machine_type" {
  type    = string
  default = "e2-standard-2"
}

variable "my_home_ip" {
  type        = string
  description = "Your home network's external IP address in CIDR format (e.g., 1.2.3.4/3
2)."
}
```

`terraform.tfvars`

This file contains the actual values matching your private environment variables. **Never check this file into public source control.**

Terraform

```
project_id = "your-actual-gcp-project-id-here"
my_home_ip = "198.51.100.45/32" # Replace with your actual public IP
region     = "us-central1"
zone       = "us-central1-a"
```

`outputs.tf`

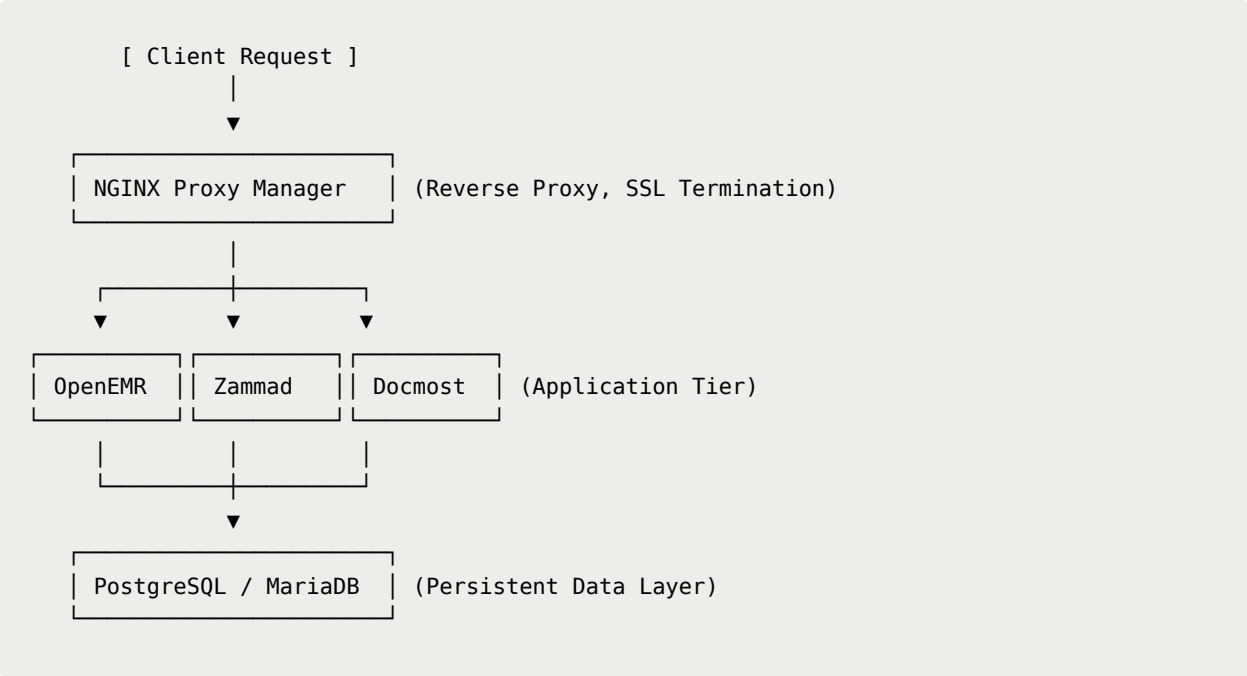
This file exposes post-deployment details, like the server's public IP address, for easy reference.

Terraform

```
output "instance_public_ip" {
  value       = google_compute_instance.docker_host.network_interface[0].access_config
[0].nat_ip
  description = "The external public IP address assigned to the host node."
}
```

Part 7: Your Long-Term Learning Framework

Your stack of real-world applications isn't just a collection of useful tools—it is a highly practical infrastructure training suite.



By maintaining this portfolio, you are gaining hands-on experience with core cloud architecture challenges:

- **OpenEMR:** Teaches you strict compliance-minded architecture patterns, forcing you to think carefully about database access control and encryption at rest.
- **OpenProject & Zammad:** These are memory-intensive, multi-process applications utilizing background workers (like Redis or Sidekiq). Running them gives you great practical experience managing system resource constraints and configuring cross-container communication.
- **Docmost:** Serves as the perfect documentation home base for your environment. Use it to map out your architecture updates, log network diagrams, and document post-mortem insights whenever a container component encounters a real-world failure.

You don't need a loud, power-hungry rack server running in your living space anymore. You have built a clean, scalable enterprise laboratory right in the cloud.

Senior Mentor Recommended Resources

Core Books

1. *Terraform Up & Running* by Yevgeniy Brikman. The definitive field manual for structuring clean, production-grade IaC code.
2. *The Phoenix Project* by Gene Kim, Kevin Behr, and George Spafford. A fantastic high-level look at the core operational principles that drive DevOps workflows.
3. *Unix and Linux System Administration Handbook* by Evi Nemeth et al. The absolute standard guide for understanding the underlying mechanics of modern Linux systems.

Primary Documentation Targets

- **GCP Architecture Framework:** bookmark the official Google Cloud documentation on security, cost optimization, and operational excellence.
- **HashiCorp Learn:** The official quickstart guides for Terraform provide an excellent foundation for mastering fundamental state management patterns.

ChatGPT

GCP Beginner → Master Roadmap Through Your Personal DockerLab

Senior Cloud Architect Assessment & Learning Plan

Executive Summary

Based on what you have already built, you are **not a pure beginner anymore**.

A true GCP beginner usually has only:

- Created a VM
- Followed a tutorial
- Installed a package or two

- Logged into the console

You have already crossed several important thresholds:

✅ Created cloud infrastructure

✅ Configured networking and firewall rules

✅ Managed Linux server

✅ Installed Docker ecosystem

✅ Created multi-container application stacks

✅ Exposed applications through reverse proxy

✅ Connected remotely through VSCode SSH

✅ Operated databases

✅ Deployed realistic business applications

Your current level:

Current Skill Level: Advanced Beginner → Early Intermediate Cloud Engineer

You are exactly at the point where **Terraform becomes valuable**.

However, the goal should not be:

> "Learn Terraform because Terraform is popular."

The goal should be:

> "Use Terraform to recreate and automate infrastructure you already understand manually."

That is the correct learning sequence.

Part 1 — Current Skill Assessment

Your Current Capability Map

Area	Current Level	Evidence
GCP Console	Beginner+	VM creation, firewall, networking
Linux	Intermediate beginner	Ubuntu server administration
Docker	Intermediate beginner	Multiple production-like containers
Docker Compose	Intermediate	Multi-app stacks
Networking	Beginner+	Ports, NPM, DNS exposure
Reverse Proxy	Beginner+	NGINX Proxy Manager
Databases	Beginner+	PostgreSQL/MariaDB deployment
DevOps Thinking	Beginner+	Infrastructure experimentation
Infrastructure as Code	Beginner	Not started
Kubernetes	Beginner	Not started
CI/CD	Beginner	Not started
Cloud Security	Beginner	Needs strengthening

What You Already Demonstrated

1. Infrastructure Thinking

You are no longer thinking:

> "How do I install OpenEMR?"

You are thinking:

> "How do I design a reusable healthcare application platform?"

That is the beginning of cloud engineering.

2. Application Stack Understanding

Your stack resembles a small business environment:

```
                Internet
                   |
```

```
              GCP External IP
                    |
             NGINX Proxy Manager
                    |
  ---------------------------------
  |        |          |            |
OpenEMR OpenProject Docmost Zammad

                    |
  ---------------------------------
  |                   |
PostgreSQL          MariaDB
```

This is not a toy environment.

You are learning:

- networking
- application architecture
- databases
- security boundaries
- persistence
- availability concepts

Remaining Gaps Before Advanced Cloud Engineering

Gap 1 — Networking Fundamentals

You need stronger understanding of:

- VPC
- Subnets
- Private IP ranges
- Routing
- NAT

- Load balancers
- DNS
- SSL/TLS
- VPN

Gap 2 — Security Model

You need deeper understanding of:

- IAM
- Service accounts
- Least privilege
- SSH keys
- Firewall strategy
- Secrets management
- Encryption

Gap 3 — Observability

Production systems require:

- Logs
- Metrics
- Alerts
- Performance monitoring

Gap 4 — Reproducibility

Currently:

> "I built this server."

Cloud engineering becomes:

> "I can destroy everything and rebuild it automatically."

This is where Terraform enters.

Part 2 — High-Level GCP Mastery Roadmap

Phase 1 — GCP Foundation (Month 1)

Objective

Understand cloud building blocks.

Learn:

GCP Core

- Projects
- Billing accounts
- Regions
- Zones
- Compute Engine
- Persistent disks
- Snapshots
- Images
- Cloud Storage

Hands-on:

Create:

```
Project
 |
 VM Instance
 |
 Disk
```

```
|
Snapshot
|
Backup
```

Phase 2 — Linux Infrastructure Mastery (Month 1-2)

Your Ubuntu knowledge should become administrator-level.

Master:

Users

```
adduser
usermod
groups
sudo
```

Permissions

```
chmod
chown
ACL
```

Services

```
systemctl
journalctl
```

Security

- SSH hardening
- Disable root login

- Fail2ban
- Firewall
- Updates

Phase 3 — Docker Mastery (Month 2-3)

You are already here.

Deepen:

Docker Architecture

Understand:

```
Docker Client

     |
Docker Engine

     |
Container Runtime

     |
Linux Kernel
```

Master:

- Images
- Layers
- Networks
- Volumes
- Compose
- Environment variables
- Secrets
- Registry

Phase 4 — Terraform Infrastructure as Code (Month 3)

This is where you are ready.

Terraform should become your automation layer.

Your progression:

Manual:

```
GCP Console
      |
      |
Create VM
```

Terraform:

```
main.tf

      |
      |
terraform apply

      |
      |
GCP creates VM
```

Phase 5 — Cloud Native Architecture (Month 5-8)

Learn:

Kubernetes

Start with:

- Pods

- Services
- Deployments
- ConfigMaps
- Secrets

Then:

GKE:

Google Kubernetes Engine

Managed Services

Learn:

- Cloud SQL
- Cloud Storage
- Secret Manager
- Cloud Monitoring

Phase 6 — Production DevOps (Month 8-12)

Master:

CI/CD

Example:

```
GitHub

 |
 |
GitHub Actions

 |
 |
Docker Build
```

```
 |
 |
Deploy GCP
```

Disaster Recovery

Learn:

- Backup
- Restore
- Replication
- RPO
- RTO

Part 3 — Hands-On Lab Progression

Lab 1 — Document Your Current DockerLab

Difficulty: ⭐

Objective:

Create architecture documentation.

Deliver:

```
DockerLab Architecture Diagram

Network Diagram

Application Inventory
```

Skills:

- Architecture thinking

Lab 2 — Harden Ubuntu

Difficulty: ⭐⭐

Tasks:

- Create admin user
- Disable root SSH
- Configure firewall
- Install updates
- Enable monitoring

Outcome:

Production-style Linux server.

Lab 3 — Docker Backup Strategy

Difficulty: ⭐⭐

Backup:

```
docker-compose.yml

.env files

volumes

database dumps
```

Outcome:

Destroy and rebuild confidence.

Lab 4 — First Terraform Project

Difficulty: ⭐⭐⭐

Create:

```
terraform/

├── main.tf
├── variables.tf
├── outputs.tf
└── terraform.tfvars
```

Example:

main.tf

Defines resources:

```
resource "google_compute_instance" "dockerlab" {

}
```

---

variables.tf

Reusable settings:

```
variable "region" {

}
```

---

outputs.tf

Shows results:

```
VM IP address
```

---

terraform.tfvars

Your values:

```
region="us-central1"
```

Lab 5 — Rebuild DockerLab Automatically

Difficulty: ⭐⭐⭐⭐

Goal:

One command:

```
terraform apply
```

Creates:

```
GCP VM

Docker

NPM

Portainer

Databases

Applications
```

This is your first true cloud engineering milestone.

Lab 6 — Cloud SQL Experiment

Difficulty:

⭐⭐⭐⭐

Move one database:

PostgreSQL → Cloud SQL

Learn:

- Managed databases
- Backups
- Connections
- IAM authentication

Lab 7 — Disaster Recovery Drill

Difficulty:

⭐⭐⭐⭐⭐

Destroy:

```
VM
Docker
Applications
```

Rebuild from:

- Terraform
- backups
- compose files

Part 4 — Should You Use Cloud SQL?

Option Comparison

Option	Advantages	Disadvantages
Docker PostgreSQL	Cheap, maximum learning, full control	You manage everything
Cloud SQL	Real enterprise experience, backups, HA	Expensive
Hybrid	Best learning path	More complexity

My Recommendation For You

Do NOT immediately migrate everything.

Your Docker databases are extremely valuable learning tools.

Keep:

```
PostgreSQL Docker
MariaDB Docker
```

Learn:

- replication
- backup
- tuning
- security

Then add:

```
Cloud SQL PostgreSQL
```

as a comparison lab.

Why PostgreSQL First?

Your applications:

- OpenEMR
- OpenProject
- Docmost

fit naturally with PostgreSQL learning.

Priority:

1. PostgreSQL

2. MariaDB/MySQL
3. Cloud SQL

Part 5 — GCP Cost Strategy

Your target:

<$40/month

Very achievable.

Recommended setup:

Item	Monthly
e2-medium VM	~$25
50GB persistent disk	~$5
Static IP	~$3
Backups/storage	~$5
Miscellaneous	~$2

Total:

≈ $35-$40/month

Avoid Initially

Avoid:

❌ Cloud SQL full time

❌ Kubernetes clusters running 24/7

❌ Load balancers

❌ Large disks

❌ GPU instances

Use:

Stop VM When Not Learning

Example:

Learning:

Saturday/Sunday

Stop:

Monday-Friday

You can reduce cost significantly.

Cloud SQL Reality

Cloud SQL usually will not fit comfortably into your budget.

Expect roughly:

$40-$100+/month depending on configuration.

Use it temporarily for labs.

Part 6 — Terraform Timing Decision

Recommendation:

Choose A: Start Terraform Now

But with one rule:

Do NOT learn Terraform abstractly.

Do this:

```
Manual GCP

↓

Understand resource

↓
```

```
Terraform recreate

↓

Destroy

↓

Rebuild
```

You are at the perfect transition point.

Your First Terraform Project

Goal:

Create your DockerLab VM.

Structure:

```
terraform-dockerlab/

├── main.tf
├── variables.tf
├── outputs.tf
├── terraform.tfvars
└── provider.tf
```

Learning sequence:

Week 1:

Create VM

Week 2:

Create firewall

Week 3:

Create network

Week 4:

Automate Docker installation

Week 5:

Deploy applications

Part 7 — Your DockerLab as a Cloud Engineering Academy

Your applications are excellent training platforms.

OpenEMR

Learn:

- healthcare systems
- databases
- backups
- security

OpenProject

Learn:

- project management
- workflows
- collaboration systems

Docmost

Learn:

- documentation platforms
- knowledge management

Zammad

Learn:

- ticketing systems
- support operations

Your environment resembles a small SaaS company:

```
                Users

                  |
           Load Balancer

                  |

          Reverse Proxy

                  |

       ---------------------

       Apps        Databases

       ---------------------

              Storage

              Backup
```

12-Month Learning Plan

Months 1-2

Master:

- GCP basics
- Linux
- Networking
- Security

Months 3-4

Master:

- Terraform
- Git
- Infrastructure automation

Months 5-6

Master:

- PostgreSQL administration
- MySQL administration
- Backup/recovery

Months 7-8

Learn:

- Cloud SQL
- Kubernetes basics
- Containers

Months 9-10

Learn:

- CI/CD
- Monitoring
- Logging

Months 11-12

Build:

"Production-like Medical Office Cloud Platform"

Using:

- OpenEMR
- automation
- backups
- Terraform
- security

Recommended Books

GCP

Google Cloud Platform in Action

Architecting Google Cloud Platform

Terraform

Terraform: Up & Running

Linux

How Linux Works

Docker

Docker Deep Dive

PostgreSQL

PostgreSQL: Up and Running

Common Beginner Mistakes To Avoid

Mistake 1

Learning tools without foundations.

Example:

"Kubernetes first."

Wrong.

Mistake 2

Building but never destroying.

Cloud engineers learn:

Create → Break → Recover

Mistake 3

Ignoring security because "it is only a lab."

Bad habits transfer.

Mistake 4

Using managed services too early.

You learn less.

Final Architect Recommendation

Your next 90 days:

Month 1

Do not add more applications.

Improve:

- Linux
- networking

- security
- backups

Month 2

Start Terraform.

Convert:

```
GCP VM
Firewall
Disk
Network
```

into code.

Month 3

Rebuild your DockerLab automatically.

Your milestone:

> "I can destroy my entire GCP environment and recreate it from code."

When you achieve that, you have crossed from **cloud user** into **cloud engineer mindset**.

Your DockerLab is already an excellent foundation. The next evolution is not more software installation — it is **automation, repeatability, security, and architecture thinkin**