AI-04025 DevOps Sandbox Foundational Plan

Owner	Ⓡ Rigel Arcayan
Tags	Project
Created time	July 13, 2026 6:57 AM

I want to learn both MySQL and PostgreSQL for DevOps, I already have a Mariadb and PostgreSQL db running on docker GCP and can access via pgadmin and PHPmysql , tell me which one to learn first or I can flip flop my learning and hands on on both. List 22 bkm to become a master db admin. What are the best books to learn GCP, MySQL, Postgres for mastering DevOps in 2026 and beyond.

Gemini

The Learning Order: Focus vs. Flip-Flop

Since you already have containerized environments running on GCP, **do not flip-flop day-to-day.**

Flip-flopping between MySQL (MariaDB) and PostgreSQL early on leads to syntax confusion and blurs the underlying structural differences. They handle concurrency, MVCC (Multi-Version Concurrency Control), connection overhead, and storage engines completely differently.

Learn PostgreSQL First, Then Bridge to MySQL

From a DevOps and infrastructure engineering perspective, start with **PostgreSQL**.

- **Why Postgres First?** Postgres follows a strict "process-per-connection" architecture, requiring an understanding of connection pooling (like PgBouncer), memory allocations per work process (work_mem), and aggressive write-ahead logging (WAL) management. Its strictness forces you to learn rigorous DBA habits.

- **The Transition to MySQL:** Once you grasp Postgres, transitioning to MySQL (or MariaDB) is straightforward. You will shift your mental model from a process-based architecture to MySQLâ€™s thread-per-connection model and focus heavily on storage engines (primarily InnoDB), table locking semantics, and thread pooling.

22 Best Known Methods (BKMs) for Master Database Administration

To manage, automate, and scale relational engines in cloud environments, prioritize these 22 industry-standard practices:

Architecture & Engine Tuning

1. **Match Connection Models to Resources:** Use connection poolers (PgBouncer for Postgres; ProxySQL or thread pools for MySQL) to prevent memory exhaustion from runaway application threads.
2. **Size Memory Allocations Judiciously:** Balance global buffers (MySQLâ€™s innodb_buffer_pool_size at ~70-80% of system RAM) against per-query operational memory (Postgres's work_mem or MySQL's sort_buffer_size) to avoid OOM (Out of Memory) kills.
3. **Optimize Disk I/O Real Estate:** Isolate write-ahead logs (WAL for Postgres, Redo logs for InnoDB) onto separate high-IOPS storage volumes from the main data directories to eliminate write bottlenecks.
4. **Master MVCC & Garbage Collection Mechanics:** Tune autovacuum profiles in Postgres to avoid table bloating and transaction ID wraparound; monitor the InnoDB purge thread in MySQL to manage undo log history.

Infrastructure as Code & Continuous Delivery

1. **Decouple Migrations via Schema Evolution Tools:** Treat database schemas as code. Use tools like Flyway, Liquibase, or Atlas integrated into CI/CD pipelines rather than executing raw manual DDL.
2. **Enforce Zero-Downtime Migrations:** Write backward-compatible DDL changes (e.g., adding nullable columns first, expanding data types in stages) to support blue-green application deployments.

3. **Declarative State via IaC:** Provision database infrastructure (GCP Cloud SQL instances, IAM bindings, networking, backups) exclusively using declarative tools like Terraform or OpenTofu.
4. **Immutability in Container Templates:** Standardize your local and testing topologies using custom Dockerfiles or Compose files that match production major versions exactly.

High Availability, Replication & Backups

1. **Automate Failover Topologies:** Implement orchestrators (Patroni for Postgres; Orchestrator or InnoDB Cluster for MySQL) to eliminate split-brain scenarios during automated master-replica promotions.
2. **Design for Read-Scale and Isolation:** Offload heavy analytical, reporting, or read-only operational traffic to asynchronous read replicas to protect the primary write instance.
3. **Validate Backups Automatically:** A backup is only as good as its restore execution. Automate a daily pipeline that provisions a transient instance, restores a snapshot, checks consistency, and tears it back down.
4. **Implement Point-In-Time Recovery (PITR):** Archive continuous WAL/binlog streams to secure object storage (like Google Cloud Storage) to allow precise restoration up to the second of a failure.

Security, Compliance & Governance

1. **Bind to Private Networks Only:** Keep databases off the public internet. Restrict all cloud instances to internal VPC networks, leveraging Cloud SQL Auth Proxy or internal peering for administrative access.
2. **Practice Principle of Least Privilege:** Ban applications from connecting as root or superuser. Assign granular privileges (SELECT, INSERT, UPDATE) isolated strictly to relevant database schemas.
3. **Decouple Secret Lifecycle Management:** Eliminate hardcoded credentials in application configurations. Inject rotating database credentials dynamically via tools like GCP Secret Manager or HashiCorp Vault.
4. **Encrypt at Rest and In Transit:** Enforce TLSv1.3 for all database connection vectors and verify underlying storage blocks utilize customer-managed

encryption keys (CMEK).

Observability, Performance & Operations

1. **Profile Execution Plans Early:** Treat query analysis (EXPLAIN ANALYZE) as a standard debugging procedure. Look for sequential disk scans, missing indexes, and bad join ordering before code leaves staging.
2. **Monitor the Golden Signals:** Build dashboards targeting engine-specific metrics: connection saturation, replication lag, slow queries, transaction volume, and disk IOPS capacity.
3. **Instrument Query Performance Insights:** Enable extensions like Postgres's pg_stat_statements or MySQL's Performance Schema to systematically identify top database resource consumers.
4. **Establish Active Connection Limits:** Implement proactive query timeouts (statement_timeout in Postgres, max_execution_time in MySQL) to abort long-running unoptimized transactions before they cause resource starvation.
5. **Proactively Prevent Connection Leaks:** Audit applications to ensure connections are consistently closed or returned to the pool, preventing dormant transactions from holding locks.
6. **Practice Chaos Engineering:** Run targeted, scheduled game days in non-production environmentsâ€”simulating unexpected failovers, network splits, or sudden disk capacity exhaustion to validate system alerts.

Best Books for Mastering DevOps & Databases in 2026

The following curated engineering literature bridges the gap between raw database administration, cloud architecture, and modern DevOps practices:

1. PostgreSQL Mastery

- **Mastering PostgreSQL 17 (or latest edition)** by *Hans-JÃ¼rgen SchÃ¶nig*
    - **Why:** This is the definitive manual for understanding production-scale Postgres deployment, performance optimization, indexing strategies, and advanced security configurations.

- **PostgreSQL 14 Internals** by *Egor Rogov* (Available via Postgres Professional)
    - **Why:** Essential reading to understand how things work under the hood. It provides an unmatched structural deep-dive into isolation levels, vacuum mechanics, page layouts, and the execution engine.

2. MySQL / MariaDB Mastery

- **High Performance MySQL (4th Edition)** by *Silvia Botros & Jeremy Tinley*
    - **Why:** The gold-standard industry guide for running MySQL at scale. It thoroughly covers everything from hardware tuning and InnoDB architecture choices to data sharding strategies and replication.
- **Efficient MySQL Performance** by *Daniel Nichter*
    - **Why:** Focuses directly on the intersection of application performance and database health, highlighting quick optimization wins, metric analysis, and index structures.

3. Google Cloud Platform & DevOps Architecture

- **Site Reliability Engineering: How Google Runs Production Systems** by *Niall Richard Murphy, Betsy Beyer, Chris Jones, & Jennifer Petoff*
    - **Why:** The fundamental text that defined modern SRE culture. It provides the architectural blueprint for telemetry, error budgets, automation, and system resilience.
- **Terraform: Up & Running (3rd Edition)** by *Yevgeniy Brikman*
    - **Why:** Unlocks the best practices for structuring infrastructure as code, which is critical for reliably automating stateful database instances, networks, and storage buckets in GCP.
- **Designing Data-Intensive Applications** by *Martin Kleppmann*
    - **Why:** While not a specific how-to guide, this remains a foundational book for any DevOps professional or DBA. It breaks down the underlying theories of storage engines, replication strategies, transactions, and distributed data systems.

Best MySQL book for DevOps in 2026

For a DevOps or Site Reliability Engineering (SRE) perspective on managing MySQL in 2026, the traditional recommendation to focus purely on granular internals has shifted. The focus is now on modern reliability metrics, observability, cloud-native environments, and maintaining performance at scale.

The standout choices for DevOps professionals are detailed below:

1. The Definitive Top Pick: *High Performance MySQL: Proven Strategies for Operating at Scale (4th Edition)*

- **Authors:** Silvia Botros and Jeremy Tinley
- **Why itâ€™s perfect for DevOps:** This edition was entirely rewritten from scratch to match a modern, cloud-centric, reliability engineering mindset. Instead of focusing heavily on squeezing single-server hardware optimizations, it handles MySQL as a critical component of a larger automated infrastructure system.
- **Key DevOps Topics Covered:**
  - Operating MySQL in both cloud-managed environments (like AWS RDS/Aurora or Google Cloud SQL) and self-hosted containerized topologies.
  - Setting up Service Level Objectives (SLOs) and Service Level Indicators (SLIs) specifically for database performance.
  - Deep dives into modern observability, tracking metrics, and integrating telemetry into reliability engineering workflows.
  - High availability, automated failovers, replication strategies, and full-stack architecture.

2. Best for Query Optimization & Telemetry: *Efficient MySQL Performance: Best Practices and Techniques*

- **Author:** Daniel Nichter
- **Why itâ€™s perfect for DevOps:** While *High Performance MySQL* gives a wide view of architecture and reliability, this book acts as a tight, highly practical reference (around 350 pages) for data infrastructure. It focuses strictly on how a database engine interacts with applications under heavy load.

- **Key DevOps Topics Covered:**
    - How to correctly measure database performance using query metrics rather than guessing.
    - Demystifying EXPLAIN outputs to trace real bottlenecks instead of throwing more compute or memory at a slow database instance.
    - Deep technical insight into indexing strategies and how the InnoDB engine behaves in production traffic scenarios.

3. Best for the Everyday Reference Shelf: *MySQL Cookbook (4th Edition)*

- **Authors:** Sveta Smirnova and Alkin Tezuysal
- **Why itâ€™s perfect for DevOps:** Sometimes you donâ€™t need an architectural philosophy; you need an immediate syntax template or operational workaround. This massive 900+ page cookbook acts as an ideal troubleshooting reference.
- **Key DevOps Topics Covered:**
    - Step-by-step configuration scripts for tasks like security hardening and replication setups.
    - Real-world diagnostic recipes for performance debugging, managing JSON data types, and handling administrative user permissions under strict access controls.

Summary Recommendation

If you need to design, deploy, and monitor scalable MySQL architectures while integrating them smoothly with automated deployment and observability pipelines, start with **Botros & Tinley's *High Performance MySQL (4th Edition)***. Pair it with **Daniel Nichter's *Efficient MySQL Performance*** for a deeper tactical understanding of profiling actual query bottlenecks.

Best PostgreSql book for DevOps in 2026

For a DevOps or Site Reliability Engineering (SRE) professional working with PostgreSQL, the ideal book needs to bridge the gap between application behavior and underlying database infrastructure. The goal is managing high availability, understanding MVCC (Multi-Version Concurrency Control) overhead, configuring connection pooling, and diagnosing heavy I/O bottlenecks.

The top PostgreSQL books that fit a DevOps, platform engineering, or SRE perspective focus on these exact areas:

1. The Definitive Core Pick: *Mastering PostgreSQL 17 (6th Edition)*

- **Author:** Hans-JÃ¼rgen SchÃ¶nig
- **Why itâ€™s perfect for DevOps:** Revised to cover the modern capabilities of PostgreSQL 17, this is the most comprehensive guide to running the engine at a massive scale. It moves beyond standard database administration to focus on advanced performance infrastructure, optimization, and scaling.
- **Key DevOps Topics Covered:**
  - **Advanced Partitioning & Sharding:** Designing massive, high-throughput time-series or multi-tenant architectures.
  - **Backup, Recovery, & Replication:** Setting up robust, zero-downtime streaming replication and configuring failovers.
  - **Deep Performance Profiling:** Interpreting complex execution plans and tracking query telemetry to pinpoint resource starvation.
  - **Security Hardening:** Implementing enterprise-grade access control and encryption methods.

2. Best for Tactical Application Reliability: *PostgreSQL Mistakes and How to Avoid Them*

- **Author:** Jimmy Angelakos
- **Why itâ€™s perfect for DevOps:** In a DevOps framework, systemic database outages often stem from a misalignment between application tier behavior (like ORM patterns) and the storage engine. This practical field guide targets exactly how to prevent production landmines before they happen.
- **Key DevOps Topics Covered:**
  - **Concurrency & Locking:** Recognizing hidden blocking operations and deadlocks caused by concurrent application deployments.
  - **Vacuum and MVCC Bloat:** Understanding how the database cleans up dead rows and how to tune autovacuum parameters to prevent severe performance degradation under continuous write traffic.

    - **Connection Handling:** Why unpooled application connections cripple database memory, and how to properly design scaling thresholds.

3. Best for the Operations Toolbelt: *PostgreSQL 16 Administration Cookbook*

- **Authors:** Gianni Ciolli, Boriss Mejias, Jimmy Angelakos, et al.
- **Why itâ€™s perfect for DevOps:** When a production incident triggers a PagerDuty alert, you rarely want a deep architectural history lessonâ€”you need immediate, validated diagnostic commands. This text features nearly 200 operational recipes.
- **Key DevOps Topics Covered:**
    - **Telemetry & Monitoring:** Interrogating internal views like pg_stat_statements and pg_stat_activity to diagnose running systems.
    - **Disaster Recovery:** Practical scripts and checklists for Point-In-Time Recovery (PITR) and testing backup integrity.
    - **Cloud & Container Integration:** Patterns that translate directly into deploying and configuring self-hosted database clusters or tuning local database runtimes.

4. Special Mention for Infrastructure Engineers: *The Art of PostgreSQL (2nd Edition)*

- **Author:** Dimitri Fontaine
- **Why itâ€™s perfect for DevOps:** If your primary role involves collaborating closely with software development squads to clean up messy microservice data pipelines, this book is essential. It focuses heavily on shifting business logic out of memory-heavy application loops and into highly performant, set-based SQL architecture natively inside the database layer.

Summary Recommendation

- Start with **SchÃ¶nig's *Mastering PostgreSQL 17*** if you need a structural blueprint for building, replication, and scaling the actual physical compute clusters.
- Pick up **Angelakos's *PostgreSQL Mistakes and How to Avoid Them*** if you are primarily focused on system reliability, preventing application-driven incidents,

and tuning out runtime degradation like table bloat.

I want to learn both MySQL and PostgreSQL for DevOps, I already have a Mariadb and PostgreSQL db running on docker GCP and can access via pgadmin and PHPmysql , tell me which one to learn first or I can flip flop my learning and hands on on both. List 22 bkm to become a master db admin.

Which One to Learn First?

Do not flip-flop.

Database engines rely on completely different architectural behaviors under the hood. If you jump back and forth while still mastering the basics, you will conflate how they handle concurrency, locking, and memory allocation. This leads to configuration mistakes in production.

Learn PostgreSQL first.

Since you already have a functional self-hosted lab running Docker on GCP, starting with PostgreSQL gives you a steeper but far more rewarding learning curve for modern DevOps:

- **True to Spec:** PostgreSQL strictly adheres to SQL standards, making it easier to downshift to MySQL/MariaDB syntax later than to adapt up.
- **Complex Architecture:** It forces you to understand explicit DevOps pain points early on, such as **MVCC (Multi-Version Concurrency Control) bloat**, connection scaling limits, and aggressive write-ahead logging (WAL).
- **MariaDB vs. MySQL Nuance:** Since your environment runs MariaDB, learning "MySQL" comes with structural deviations (like MariaDB's divergent thread pooling and storage engines). Concurrently juggling those nuances alongside PostgreSQL will create unnecessary friction.

Focus on PostgreSQL until you can confidently trace a query bottleneck and manage a replication pipeline, then apply that structural mindset to your MariaDB instance.

22 Best Known Methods (BKMs) to Become a Master Database Administrator

To transition from managing a sandbox environment to orchestrating enterprise-grade data layers at scale, master these 22 core engineering principles:

I. Core Architecture & Resource Management

1. **Master the Storage Engine Under the Hood:** Understand how data hits the disk. For MySQL/MariaDB, master the InnoDB buffer pool allocation. For PostgreSQL, understand the interplay between shared_buffers and the OS page cache.
2. **Tame the MVCC and Bloat Engine:** In PostgreSQL, master how VACUUM marks dead rows and tune autovacuum parameters aggressively before write traffic scales. In MariaDB, monitor the Undo Log and Purge threads.
3. **Decouple Connections with a Pooler:** Never allow application servers to connect directly to a core database instance at scale. Implement **PgBouncer** or **Odyssey** for Postgres, and use proxy layers or built-in thread pools for MariaDB to prevent memory starvation.
4. **Isolate the Write-Ahead Log (WAL / Redo Log):** Protect your transactional log throughput. Ensure your WAL or Redo configurations reside on high-IOPS storage tiers with sequential write prioritization to prevent checkpoint flushing bottlenecks.

II. Observability & Performance Engineering

1. **Enforce Global Telemetry via Query Insights:** Always enable pg_stat_statements (Postgres) or the Performance Schema (MariaDB). Never guess what is slowâ€”identify performance issues by tracking cumulative execution time and total I/O impact.
2. **Master Execution Plan Diagnosis:** Treat EXPLAIN (ANALYZE, BUFFERS) in Postgres and EXPLAIN EXTENDED in MySQL as your primary tools. Look for sequential disk scans, bad cardinality estimates, and hidden implicit type casting.
3. **Build an Index-Pruning Pipeline:** Unused indexes waste disk space and degrade write performance. Periodically audit your schema to drop unused or duplicate indexes, and use partial or covering indexes to optimize hot query paths.
4. **Establish Baseline System SLIs:** Track foundational infrastructure metrics: disk I/O utilization, CPU wait states, replication lag (in bytes/seconds), and available connection capacity.

III. High Availability & Scalability Architecture

1. **Enforce Single-Primary, Multi-Replica Topologies:** Build clear horizontal read scalability. Route heavy analytical queries or read-intensive API traffic to read replicas, preserving the primary node exclusively for state mutations.
2. **Automate Failover with Consensus Tools:** Avoid manual intervention during infrastructure drops. Implement split-brain protected orchestration systems like **Patroni** (with etcd/Consul) for Postgres, or **Orchestrator** / Galera Cluster mechanics for MariaDB.
3. **Design a Connection-Routing Proxy Layer:** Abstract topology changes away from the application tier using intelligent routing proxies like **HAProxy**, **ProxySQL** (for MySQL), or **Pgcat** (for Postgres) to handle transparent failovers.
4. **Implement Proactive Data Partitioning:** Prevent individual tables from growing into unmanageable multi-terabyte block structures. Use declarative range or list partitioning based on time or tenant IDs to keep indexes nimble and allow fast data drops.

IV. Data Protection & Disaster Recovery

1. **Implement Continuous WAL/Binlog Archiving:** Standard physical backups are not enough. Use tools like **pgBackRest** or **Wal-G** (Postgres) and Percona XtraBackup (MariaDB) to stream transaction logs to object storage (like Google Cloud Storage) for Point-In-Time Recovery (PITR).
2. **Enforce the Backup Verification Rule:** An untested backup is a failed backup. Build automated, scheduled pipeline environments that pull production backups, restore them onto isolated scratch instances, and run integrity checks (amcheck or checksum verifications).
3. **De-risk Schema Migrations:** Never allow blocking schema definitions (ALTER TABLE) to lock production databases. Use non-blocking migration patterns, online schema change tools (like gh-ost or pt-online-schema-change), or explicit lock timeouts.

V. Security, Governance & Compliance

1. **Enforce Least-Privilege Role Access:** Applications must never connect using superuser accounts (postgres or root). Restrict application users strictly to

isolated Data Definition/Manipulation (DDL/DML) privileges on explicit namespaces.

2. **Isolate Secrets Architecture:** Strip all cleartext credentials out of configuration files and environment definitions. Pull database credentials dynamically at runtime using secure secret managers (like HashiCorp Vault or GCP Secret Manager) combined with short-lived tokens.
3. **Audit the Perimeter:** Hard-bind your database engines exclusively to internal private overlay networks (VPCs) or specific Docker bridge structures. Never expose a database port directly to the public internet.

VI. Cloud-Native & Infrastructure as Code (IaC)

1. **Standardize Configurations via Declarative Code:** Treat your database parameter configurations (postgresql.conf or my.cnf) as immutable infrastructure. Manage changes inside a Git repository and deploy them using tools like Ansible or Terraform.
2. **Isolate Persistent Storage Lifecycles:** When running databases inside ephemeral environments like Docker or Kubernetes, ensure your data directories map cleanly to isolated, high-performance persistent block volumes that survive container destruction.
3. **Know When to Delegate to Cloud-Managed Services:** As an enterprise DevOps professional, know when to trade low-level infrastructure tuning for operational simplicity. Evaluate when to offload cluster management to platforms like Cloud SQL or AlloyDB so you can focus on performance optimization.
4. **Design Data Retries for Transient Drops:** Networks drop and clusters fail over. Ensure your application architecture implements elegant connection retry loops using exponential backoff to recover smoothly during database failover windows.

My learning objectives is to be able to confidently and become a Master level DevOps in docker GCP OpenEMR, OpenProject, Inventree and Zammad, NPM, Docmost, pgadmin, phpmyadmin , explain to me how to get a sandbox dockerlab running and maintaining. I have all the applications running in my dockerlab but I have zero structure in terms of lesson plans and infrastructure. I

have plenty of time in my hands, a GCP, books to read about the topic and money to spend. Guide me to become the best DevOps guy at 60 yo (retired) and beyond.

Retirement is the perfect time to build a master-level engineering sandbox. With your background, you donâ€™t need a high-level overview; you need a rigorous, production-grade framework that treats your self-hosted application stack as an enterprise-scale infrastructure project.

Right now, having all these apps (OpenEMR, Zammad, OpenProject, InvenTree, Docmost, NPM) running without a structured framework is what we call "ClickOps"â€”everything works, but itâ€™s fragile, manually configured, and undocumented. To move to a **Master-Level DevOps** mindset, we need to transition your lab into an automated, version-controlled, highly observable system.

Here is your multi-phase roadmap and infrastructure blueprint to master database and application operations on Google Cloud Platform (GCP).

Part 1: The Infrastructure Blueprint (Fixing the Foundation)

Before diving into lesson plans, your GCP Docker lab needs to be restructured using the exact tools enterprise SREs (Site Reliability Engineers) use.

1. Version Control & GitOps (The Single Source of Truth)

Stop configuring things via Portainer or direct SSH edits.

- Create a private **GitHub** or **GitLab** repository named gcp-infra-sandbox.
- Every single Docker Compose file, Nginx Proxy Manager configuration, and database environment variable must live in this repo.
- If a virtual machine (VM) dies, you should be able to clone this repository onto a fresh GCP Compute Engine instance, run a single command, and bring the entire stack back online in under 10 minutes.

2. Network Isolation (The Security Perimeter)

Do not let your applications talk to each other unless they absolutely have to. In your master Docker Compose layout, create isolated bridge networks:

- npm-tier: A network exclusively for Nginx Proxy Manager and the frontend ports of your web apps (OpenEMR, Zammad, Docmost, etc.).
- db-tier: An internal backend network where PostgreSQL and MariaDB sit. Only the application backends can touch this network. pgAdmin and phpMyAdmin should also sit here, locked behind strict authentication.

Part 2: The 5-Phase Master-Level DevOps Curriculum

With plenty of time and a dedicated budget, treat each of these phases as a 3-to-4-week deep dive. Do not rush to the next phase until the current one is entirely automated via code.

Phase 1: Infrastructure as Code (IaC) & Configuration Management

Objective: Eliminate the GCP console. You should build and destroy your lab using code.

- **The Task:** Learn **Terraform**. Write a script that provisions your GCP VPC (Virtual Private Cloud), sets up firewall rules (blocking everything except ports 80, 443, and your explicit WireGuard/VPN administrative port), and spins up your Compute Engine VM.
- **The Integration:** Use **Ansible** to automatically connect to that new VM, install Docker, configure user permissions, and pull down your Git repository.
- **The Definition of Done:** You can run terraform destroy followed by terraform apply, and your entire cloud network rebuilds itself flawlessly.

Phase 2: Database Hardening & Performance Optimization

Objective: Move from "it runs" to "it's optimized." Your stack uses both PostgreSQL (Zammad, OpenProject, Docmost) and MariaDB/MySQL (OpenEMR, InvenTree).

- **The Task:** Implement the Best Known Methods (BKMs) for storage layout. Split your GCP boot disk from your database storage. Attach an independent, high-

IOPS persistent SSD volume exclusively mounted to /var/lib/docker/volumes/ for database persistence.

- **The Integration:** Use pg_stat_statements on your Postgres containers. Force an artificial load on OpenProject or Zammad, then use pgAdmin to extract the EXPLAIN ANALYZE query execution plans. Tune the postgresql.conf parameters (like shared_buffers and work_mem) specifically for your VM's RAM profile.

Phase 3: Advanced Traffic Routing & Edge Security

Objective: Master Nginx Proxy Manager (NPM) and transport layer security.

- **The Task:** Stop using wildcard SSL setups blindly. Configure individual Let's Encrip certificates with strict HTTP Strict Transport Security (HSTS) headers enabled inside NPM.
- **The Integration:** Implement an **OAuth2 Proxy** or Cloudflare Tunnel architecture in front of administrative tools like pgAdmin and phpMyAdmin. No oneâ€"not even youâ€"should be able to see the login page of your database managers without first passing through an identity verification layer (like a hardware token or single sign-on).

Phase 4: Production-Grade Observability (The SRE Holy Grail)

Objective: You cannot manage what you do not measure. If Zammad drops a connection, you need to know why instantly.

- **The Task:** Spin up a dedicated monitoring stack alongside your apps using **Prometheus** and **Grafana**.
- **The Integration:** Deploy cadvisor (to monitor Docker container CPU/Memory), postgres_exporter, and mysqld_exporter. Build a unified Grafana dashboard that visualizes:
  - Database connection pool saturation.
  - Disk write latencies on your GCP persistent volumes.
  - Nginx response status codes (tracking 5xx errors across OpenEMR or Zammad).

Phase 5: State Preservation & Disaster Recovery (The Final Exam)

Objective: Achieve absolute data survival.

- **The Task:** Write a production backup pipeline. For PostgreSQL, integrate **pgBackRest** or **WAL-G** into your Docker setup. For MariaDB, use a non-blocking logical backup tool.
- **The Integration:** Automate a script that takes these backups, encrypts them, and ships them to a cold-storage **GCP Cloud Storage Bucket** with a strict lifecycle policy (retaining 30 days of data).
- **The Chaos Test:** Intentionally run docker compose down -v (wiping out all local database volumes) on your live environment. Time yourself on how long it takes to pull the backup from the cloud storage bucket, run a Point-In-Time Recovery (PITR), and bring your apps back online with zero data loss.

Part 3: The Day-to-Day Operational Rhythm

To build muscle memory like an enterprise operations team, run your sandbox using a structured weekly schedule:

Day	Focus Area	Operational Task
Monday	**Telemetry Audit**	Open your Grafana dashboard. Analyze memory consumption trends from the previous week. Check if OpenEMR or Zammad is leaking memory inside their respective container runtimes.
Wednesday	**Security & Patching**	Do not manually update containers. Update your Git repository's image tags, run them through a local staging check, and push the updates via your Ansible playbooks.
Friday	**Disaster Drills**	Execute a random failure test. Turn off your primary database container mid-query and verify how gracefully the application tier handles the transient drop and reconnects once the database recovers.

By treating your applications not just as isolated software packages, but as interconnected services governed by code, infrastructure tracking, and strict security limits, you will surpass typical enterprise-level DevOps competencies. You have the environment, the time, and the cloud capabilitiesâ€”now shift everything into code.

ChatGPT

Given your current setup (Docker on GCP, MariaDB running, PostgreSQL running, pgAdmin, phpMyAdmin, and your goal of **DevOps mastery**), you are already past the “install a database” stage. Your next level is not learning SQL syntax — it is learning **database operations: reliability, automation, performance, security, backup, recovery, scaling, and observability**.

For your situation, I would **not learn MySQL and PostgreSQL sequentially**. Learn them in parallel, but with a primary focus.

My recommendation:

Primary database: PostgreSQL (70%)

Secondary database: MySQL/MariaDB (30%)

Why PostgreSQL first:

- PostgreSQL is the dominant choice for modern cloud-native applications.
- It is more feature-rich: JSONB, extensions, advanced indexing, window functions, GIS, analytics.
- Many DevOps tools and SaaS platforms use PostgreSQL internally.
- Kubernetes operators, cloud managed databases, and modern architectures heavily favor PostgreSQL.
- It teaches deeper database concepts.

Why still learn MySQL/MariaDB:

- Huge installed base in enterprise.
- Many legacy applications use MySQL.
- WordPress, Magento, LAMP stacks, and many web apps depend on it.
- Understanding MySQL makes you a more versatile DBA.

Think of PostgreSQL as learning a **precision machine tool**, and MySQL as learning a **mass-produced industrial machine**. A master DBA should know both.

Recommended Learning Strategy: The 80/20 DevOps DBA Path

Phase 1 (Month 1-2): Database Foundations

Use both databases.

Build the same application twice:

Example:

Medical Practice App

PostgreSQL Version

|

|

MariaDB Version

Create:

- patients table
- appointments table
- billing table
- inventory table
- audit table

Learn:

- database design
- normalization
- relationships
- indexes
- constraints
- transactions

Your existing OpenEMR + InvenTree projects are perfect laboratories.

22 BKM (Best Known Methods) to Become a Master Database Administrator

1. Master SQL before tools

A DBA who depends on GUI tools is limited.

Master:

- SELECT
- JOIN
- GROUP BY
- WINDOW FUNCTIONS
- CTE
- TRANSACTIONS
- LOCKING

2. Understand database internals

Know:

PostgreSQL:

Client

|

Postmaster

|

Backend Process

|

Shared Buffers

|

WAL

|

Data Files

MySQL:

Client

|

Server Thread

|

InnoDB Buffer Pool

|

Redo Logs

|

Tablespace

3. Learn data modeling

Master:

- normalization
- denormalization
- star schema
- OLTP vs OLAP

4. Become an indexing expert

Understand:

PostgreSQL:

- B-tree
- Hash
- GIN

- GiST
- BRIN

MySQL:

- B-tree
- covering indexes
- composite indexes

5. Master EXPLAIN plans

A senior DBA lives inside:

PostgreSQL:

EXPLAIN ANALYZE

MySQL:

EXPLAIN FORMAT=JSON

6. Learn transactions deeply

Understand:

ACID:

- Atomicity
- Consistency
- Isolation
- Durability

7. Master backup and recovery

A DBA is measured by:

“Can you restore after disaster?”

Learn:

Postgres:

- pg_dump
- pg_restore
- WAL archiving
- PITR

MySQL:

- mysqldump
- mysqlpump
- binary logs

8. Build a disaster recovery lab

Create:

Production DB

|

|

Backup Server

|

|

Restore Testing

A backup never tested is not a backup.

9. Learn replication

PostgreSQL:

- streaming replication
- logical replication

MySQL:

- primary/replica
- GTID replication

10. Learn database security

Master:

- users
- roles
- privileges
- encryption
- SSL connections
- auditing

11. Learn Linux deeply

DBAs are Linux engineers.

Master:

- filesystem
- permissions
- systemd
- networking
- processes
- logs

12. Learn Docker database operations

You already started.

Master:

- persistent volumes

- container backups
- environment variables
- secrets
- health checks

13. Learn Kubernetes databases

Future DBA skill:

- StatefulSets
- Persistent Volumes
- Operators

Examples:

Postgres:

CloudNativePG PostgreSQL Operator

14. Learn Infrastructure as Code

Master:

- Terraform
- Ansible

Example:

Terraform

|

Creates GCP VM

|

Ansible

|

Installs PostgreSQL

15. Learn GCP database services

Master:

Google Cloud SQL

Learn:

- Cloud SQL PostgreSQL
- Cloud SQL MySQL
- backups
- replicas
- IAM integration

16. Learn monitoring

Master:

- Prometheus
- Grafana
- database metrics

Watch:

- CPU
- memory
- connections
- slow queries
- locks

17. Learn logging

Master:

Postgres:

- pg_stat_activity

- log_min_duration_statement

MySQL:

- slow query log

18. Learn performance tuning

Understand:

Postgres:

shared_buffers

work_mem

maintenance_work_mem

MySQL:

innodb_buffer_pool_size

query_cache

19. Learn connection management

Modern apps create too many connections.

Learn:

Postgres:

- PgBouncer

MySQL:

- ProxySQL

20. Learn automation scripting

Master:

Python

and:

- psycopg
- mysql connector

Automate:

- backups
- reports
- health checks

21. Learn database migrations

Master:

- Flyway
- Liquibase
- Alembic

Understand:

Application v1

Database v1

Application v2

Database v2

22. Think like a Site Reliability Engineer

A master DBA thinks:

“What happens at 3AM when this breaks?”

Master:

- RTO
- RPO
- HA
- disaster recovery

- incident response

Best Books for 2026+ DevOps DBA Mastery

PostgreSQL

1.

PostgreSQL: Up and Running

PostgreSQL: Up and Running

Best first PostgreSQL book.

Learn:

- SQL
- administration
- extensions
- performance

2.

PostgreSQL 16 Administration Cookbook

PostgreSQL 16 Administration Cookbook

Best for:

- DBA tasks
- production environments
- troubleshooting

3.

The Internals of PostgreSQL

Free online resource.

Master:

- storage engine
- vacuum
- WAL
- MVCC

MySQL / MariaDB

4.

High Performance MySQL

High Performance MySQL

The classic MySQL DBA book.

Learn:

- optimization
- indexing
- replication

5.

MySQL Cookbook

MySQL Cookbook

Excellent reference.

GCP / Cloud

6.

Google Cloud Platform in Action

Google Cloud Platform in Action

7.

Google Cloud Certified Professional Cloud Architect Study Guide

Google Cloud Certified Professional Cloud Architect Study Guide

Even if you don’t take the exam, it teaches architecture.

DevOps Books

8.

The DevOps Handbook

The DevOps Handbook

Core DevOps philosophy.

9.

Site Reliability Engineering

Site Reliability Engineering

Free from Google.

Learn:

- reliability
- monitoring
- automation

10.

Kubernetes Up & Running

Kubernetes: Up and Running

Your Personal Lab Roadmap (Perfect for Your GCP DockerLab)

I would evolve your environment like this:

GCP VM

|

Docker

|

+----------------+

| |

PostgreSQL MariaDB

|

|

OpenEMR WordPress

|

|

Backups

|

|

Google Cloud Storage

|

|

Monitoring

|

Grafana

Then add:

Month 1:

- CRUD databases
- SQL mastery
- schema design

Month 2:

- backup/restore
- replication

Month 3:

- performance tuning

Month 4:

- Terraform + Ansible

Month 5:

- Kubernetes databases

Month 6:

- production-grade HA

Given your existing **OpenEMR, InvenTree, Docker, GCP, PostgreSQL, MariaDB, and medical workflow automation projects**, you are actually building a realistic **mini enterprise DevOps environment**, not just a learning lab. The fastest path is to use those applications as your “production systems” and practice DBA operations against them.