AIDockerLab Essential Apps

Owner	Ⓡ Rigel Arcayan
Tags	Project
Created time	June 19, 2026 7:58 AM

I’m testing docker in GCP, e2 highmem, 8gb, 60gb space, I have npm, OpenEMR, OpenProject,Inventree, Mariadb, Postgres, list 10 essential/more good to have for learning in GCP and docker.

With an e2-highmem-2 instance (2 vCPUs, 16 GB RAM) paired with 60 GB of storage, you have an excellent baseline. While the memory is plentiful, your primary constraint for learning will be **disk space (60 GB)** and **vCPU cycles** when multiple heavy databases (MariaDB, Postgres) and enterprise apps (OpenEMR, OpenProject) run concurrently.
To round out your deployment, learn the GCP-to-Docker boundaries, and keep things highly efficient, here are 10 essential and "good-to-have" containerized tools split by practical utility.

I. Infrastructure Management & Visibility (The Core Layer)

Since you are running multiple microservices, you need ways to inspect, manage, and route traffic without raw CLI overhead.

1. Portainer CE or Dockge

- **What it does:** Web-based GUI Command Center for Docker.
- **Why it's essential:** Managing half a dozen stacks via SSH gets tedious. Portainer lets you look at real-time logs, container health, and resource usage metrics via a sleek browser UI. If you prefer a simpler, lighter, pure Docker Compose-focused YAML editor, look at **Dockge**.
- **Resource footprint:** Tiny (~50MB RAM).

2. Homepage or Dashy

- **What it does:** A lightweight, self-hosted applications dashboard.
- **Why it's good to have:** Instead of memorizing ports (:8080, :3000, :8000), a single dashboard acts as your entry portal. Homepage integrates directly with Docker sockets to show container status (up/down) and real-time API stats right on the landing page buttons.
- **Resource footprint:** Negligible.

3. Dozzle

- **What it does:** Real-time, minimalist log viewer for Docker containers.
- **Why it's essential:** When learning integration between complex systems like OpenEMR or InvenTree and their backend databases, watching stdout logs concurrently is crucial. Dozzle streams them to a clean interface instantly without needing to attachment-spam docker logs -f.
- **Resource footprint:** ~10-20MB RAM.

II. Security, Routing, & Network Boundary

Running apps on a cloud provider means protecting your public interface and handling SSL effortlessly.

4. Nginx Proxy Manager (NPM)

- **What it does:** Reverse proxy with a built-in GUI wrapper for Let's Encrypt SSL certificates.
- **Why it's essential:** *Note: You mentioned you have "npm" in your list—if you meant Node Package Manager, you absolutely need this instead. If you already meant Nginx Proxy Manager, consider this your confirmation.* It lets you expose OpenEMR or OpenProject via clean subdomains with automated HTTPS, handling the proxy layers effortlessly.
- **Resource footprint:** Very low.

5. Watchtower

- **What it does:** Automated container image updater.
- **Why it's essential:** It monitors your running containers, watches for updated image tags on Docker Hub, and gracefully restarts the container with the new image while preserving your data volumes. Excellent for maintaining hands-off setups.
- **Resource footprint:** Minimal (it wakes up on a cron schedule, runs, and idles).

6. Cloudflare Tunnel (cloudflared)

- **What it does:** Securely connects your local Docker environment to the Cloudflare network without opening firewall ports.
- **Why it's essential for GCP:** Instead of editing GCP VPC firewall rules to open public ports (like 80 or 443), you run a cloudflared container. It establishes an outbound connection, letting you map domains safely to your internal Docker network. It completely shields your VM's public IP from the open internet.
- **Resource footprint:** Low (~30MB RAM).

III. Systems Monitoring & Automation

Moving past basic deployments means learning how infrastructure stays online and handles automated workflows.

7. Uptime Kuma

- **What it does:** A self-hosted uptime monitoring tool with a gorgeous UI.
- **Why it's essential:** It tracks HTTP status codes, ping response times, and database TCP ports across your container stack. It can ping your OpenEMR or MariaDB instance every 60 seconds and fire alerts (via Discord, Telegram, or email) if an app crashes or runs out of memory.
- **Resource footprint:** Low (~50MB RAM).

8. n8n

- **What it does:** A fair-code node-based workflow automation tool (an open-source Zapier alternative).

- **Why it's great for learning:** Since you have data-heavy apps like InvenTree (inventory) and OpenProject (tasks), you can use n8n to connect them. For example: "If a new task is created in OpenProject, generate a log entry or check an inventory threshold in InvenTree." It teaches you webhook integration and API management inside container networks.
- **Resource footprint:** Moderate (~150MB-300MB RAM depending on active executions).

IV. Data Protection & Utilities

9. Duplicati or Restic (via Docker)

- **What it does:** Encrypted, deduplicated cloud backup software.
- **Why it's non-negotiable:** Your 60GB disk contains relational databases (Postgres, MariaDB). If your GCP instance undergoes a transient failure or you accidentally corrupt a volume during a Docker test, you need automated backups of your mounted volumes sent straight to an object storage bucket (like Google Cloud Storage or Backblaze B2).
- **Resource footprint:** Low during idle; spikes slightly during active compression/upload windows.

10. Gitea or Forgejo

- **What it does:** An ultra-lightweight, self-hosted Git service (a lightweight alternative to GitHub/GitLab).
- **Why it's great for learning:** Don't just launch containers imperatively via terminal. Store all your docker-compose.yml configurations, database initialization scripts, and custom environment variable configurations inside a local Git repository. This lets you practice clean GitOps workflows entirely within your own cloud boundary.
- **Resource footprint:** Remarkably light (~100MB RAM), unlike heavy alternatives like GitLab which would completely saturate your VM.

💡 Resource Optimization Strategy for Your VM:

Because your 60GB disk space can vanish quickly under heavy container log generation and database write-ahead logs (WAL), implement these two standard configurations immediately:

1. **Log Rotation:** Prevent Docker from eating up your disk space by enforcing global log limits in your /etc/docker/daemon.json:

```
{
  "log-driver": "json-file",
  "log-opts": {
    "max-size": "10m",
    "max-file": "3"
  }
}
```

1. **Named Volumes:** Always map database folders (/var/lib/postgresql/data and /var/lib/mysql) out to named persistent Docker volumes or bind mounts on the host so your state survives any container updates or stack modifications.