

AI-03086 Dockerlab to NativeDb

👤 Owner	© Rigel Arcayan
☰ Tags	Project
🕒 Created time	@October 17, 2025 2:16 PM

In my dockerlab, can i install mariadb and postgresdb bare metal then when I run docker apps, makes use the native mariadb and postgres. I just want a full control of my data, is this a normal route or there is a better way for data safety.

Absolutely, Rigel — running MariaDB and PostgreSQL natively on bare metal while having your Dockerized apps connect to them is a valid and often preferred approach for those who prioritize data sovereignty, performance, and granular control. Here's a breakdown of the pros, trade-offs, and best practices:

✅ Why Bare Metal DB + Docker Apps Can Be Smart

- Full Control Over Data:• You manage backups, replication, and storage locations directly.
 - No risk of container auto-removal wiping your data.
- Performance:• Native DBs often outperform containerized ones due to direct disk I/O and memory access.
 - Easier to fine-tune system-level parameters (e.g., `sysctl`, `ulimits`, caching).
- Security & Isolation:• You can harden the DB server independently (e.g., AppArmor, SELinux, firewall rules).
 - Docker apps can be restricted to only access DB ports via internal networking.
- Legacy Gifting & Protocol Clarity:• Your zipped documentation can clearly separate DB protocols from app containers.
 - Easier to mentor others on troubleshooting without Docker abstractions.

⚠️ Trade-offs to Consider

Aspect	Bare Metal DB	Dockerized DB
Portability	Less portable; tied to host	Highly portable
Backup Strategy	Manual or scripted	Can use Docker volumes/snapshots
Scaling	Requires external orchestration	Easier with Docker Swarm/K8s
Complexity	More moving parts	Simpler all-in-one stack

Best Practices for Your Setup

- Networking:
 - Use Docker's `bridge` or `host` mode to allow containers to reach native DBs.
 - Example: Set DB host in your app's config to `host.docker.internal` (on Linux, use actual IP or hostname).
 - User & Permissions:
 - Create dedicated DB users for each app with scoped privileges.
 - Avoid root or superuser access from containers.
 - Backups & Monitoring:
 - Use cron jobs or systemd timers for regular dumps (`pg_dump` , `mysqldump`).
 - Monitor with tools like `pgAdmin` , `phpMyAdmin` , or Prometheus exporters.
 - Documentation for Legacy Gifting:
 - Archive your DB setup as a separate protocol: install steps, config files, backup scripts.
 - Include a DockerLab README that explains how containers connect to native DBs.
-

Alternative Hybrid Route (if curious)

You could also run DBs in Docker but mount external volumes (e.g., `/mnt/data/postgres`) and disable auto-removal, giving you near-native control while retaining portability. But for your legacy-grade goals, bare metal DBs with Dockerized apps is a robust and mentor-friendly route.

Would you like a sample Docker Compose snippet that connects to a native PostgreSQL or MariaDB instance? Or a zipped protocol outline for gifting this setup?

Yes, use `docker-compose-app.yml` = inventree, openproject, osticket use `docker-compose-core.yml` = portainer, NPM Nginx, pgadmin and phpadmin

Perfect, Rigel. Here's a legacy-grade setup that lets your Dockerized apps (Inventree, OpenProject, OSTicket) connect to bare metal MariaDB and PostgreSQL, while keeping your core services (Portainer, NGINX Proxy Manager, pgAdmin, phpMyAdmin) modular and mentor-friendly.

Architecture Overview

[Bare Metal DBs]

- ├─ MariaDB (for OSTicket, phpMyAdmin)
- └─ PostgreSQL (for OpenProject, Inventree, pgAdmin)

[docker-compose-core.yml]

- ├─ Portainer
- ├─ NGINX Proxy Manager
- ├─ phpMyAdmin (connects to native MariaDB)
- └─ pgAdmin (connects to native PostgreSQL)

[docker-compose-app.yml]

- ├─ Inventree (PostgreSQL)
 - ├─ OpenProject (PostgreSQL)
 - └─ OSTicket (MariaDB)
-

Sample Compose Snippets

`docker-compose-core.yml`

```
version: '3.8'
services:
  portainer:
    image: portainer/portainer-ce
    ports:
      - "9000:9000"
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
      - portainer_data:/data

  npm:
    image: jc21/nginx-proxy-manager:latest
    ports:
      - "80:80"
```

```
- "81:81"
- "443:443"
volumes:
- npm_data:/data
- npm_letsencrypt:/etc/letsencrypt

phpmyadmin:
image: phpmyadmin/phpmyadmin
environment:
PMA_HOST: host.docker.internal
PMA_PORT: 3306
ports:
- "8080:80"

pgadmin:
image: dpage/pgadmin4
environment:
PGADMIN_DEFAULT_EMAIL: admin@local.com
PGADMIN_DEFAULT_PASSWORD: strongpassword
ports:
- "5050:80"
volumes:
- pgadmin_data:/var/lib/pgadmin

volumes:
portainer_data:
npm_data:
npm_letsencrypt:
pgadmin_data:
```

`docker-compose-app.yml`

```
version: '3.8'
services:
inventree:
image: inventree/inventree:latest
environment:
DB_HOST: host.docker.internal
DB_PORT: 5432
```

DB_NAME: inventree

DB_USER: inventree_user

DB_PASSWORD: securepass

ports:

- "8001:8000"

openproject:

image: openproject/community:latest

environment:

DATABASE_URL:

postgres://openproject_user:securepass@host.docker.internal:5432/openproject

ports:

- "8081:80"

osticket:

image: osticket/osticket:latest

environment:

MYSQL_HOST: host.docker.internal

MYSQL_PORT: 3306

MYSQL_DATABASE: osticket

MYSQL_USER: osticket_user

MYSQL_PASSWORD: securepass

ports:

- "8082:80"

Data Safety Protocols

- Backups:• Use `mysqldump` and `pg_dump` with cron/systemd timers.
 - Archive to `/mnt/legacy_backups/` with timestamped folders.
 - Monitoring:• Use pgAdmin and phpMyAdmin for visual checks.
 - Optionally add Prometheus exporters for DB metrics.
 - Firewall & Access Control:• Restrict DB ports to localhost or Docker bridge IPs.
 - Use strong DB user passwords and scoped privileges.
 - Legacy Gifting:• Zip your `docker-compose-*.yaml`, DB setup scripts, and backup protocols.
 - Include a README with connection diagrams and mentoring notes.
-

Would you like me to help you generate a zipped protocol folder structure or a markdown README template for gifting this setup to mentees? Or perhaps a Codex vignette that explains this architecture in mythic metaphor?

Explain the networking portion of native mariadb and postgres talking to docker network. Is this still a bridgemode?

Great question, Rigel — and it's one that touches the heart of Docker's networking model and your legacy-grade control ethos. Let's break it down clearly:

Native DB ↔ Docker Networking: How It Works

When you run MariaDB and PostgreSQL natively on bare metal, and Docker containers need to connect to them, you're bridging two worlds:

- Native DBs: Bound to host IP (e.g., `127.0.0.1` or `192.168.x.x`)
- Docker containers: By default, live in an isolated bridge network

So yes — Docker is still using bridge mode, but you need to expose the native DBs to the Docker bridge network.

Key Networking Concepts

Component	Description
-----------	-------------

Bridge Mode	Default Docker network. Containers get internal IPs (e.g., 172.18.0.x) and NAT to host.
-------------	---

<code>host.docker.internal</code>	Special DNS name (on Linux, may need workaround) that containers use to reach host services.
-----------------------------------	--

Native DB Binding	DBs must bind to 0.0.0.0 or host IP, not just localhost, to be reachable from containers.
-------------------	---

Firewall Rules	Ensure ports 3306 (MariaDB) and 5432 (PostgreSQL) are open to Docker bridge subnet.
----------------	---

How to Make It Work

1. Configure Native DBs to Accept External Connections

MariaDB (`/etc/mysql/my.cnf` or `/etc/my.cnf`)

```
[mysqld]
```

```
bind-address = 0.0.0.0
```

PostgreSQL (`postgresql.conf`)

```
listen_addresses = '*'
```

And in `pg_hba.conf`, allow connections from Docker subnet:

```
host all all 172.18.0.0/16 md5
```

1. Use Correct Hostname in Docker Compose

In your Docker containers, use:

```
DB_HOST: host.docker.internal
```

⚠ On Linux, `host.docker.internal` may not work out-of-the-box. You can:

- Use the host's IP (e.g., `192.168.1.100`)
 - Or create a custom Docker network and attach containers to it with `-add-host`
-

1. Verify Connectivity

From inside a container:

```
ping host.docker.internal
```

```
telnet host.docker.internal 3306
```

```
telnet host.docker.internal 5432
```

🕒 Optional: Use `host` Network Mode (Advanced)

If you want containers to share the host's network stack directly:

```
network_mode: "host"
```

But this disables Docker's internal DNS and isolation — not ideal for modular setups like yours.

🧬 Legacy Protocol Tip

In your zipped documentation, include:

- A diagram showing Docker bridge ↔ native DB
 - Sample `pg_hba.conf` and `my.cnf` edits
 - A troubleshooting vignette: "Why my container can't reach my DB"
 - A mentoring note: "Bridge mode is a metaphor — containers crossing into native territory"
-

Would you like me to sketch a sample network diagram or generate a markdown README for this networking protocol?

I got sidetracked, back to my dockerlab to natedb topic, my native ip is 192.168.1.3 and the docker app is in 172.0.0.x, in the openproject yml, the db port is mentioned but not the host ip, how does openproject know which ip to talk to? Nginx or there is a .conf somewhere. Enlighten me.

Ah, Rigel — this is a classic case of implicit networking assumptions in Dockerized apps. Let's dememmer this together:

Why OpenProject Can Talk to Native DB Without Explicit IP

If your `docker-compose.yml` for OpenProject includes:

environment:

```
DATABASE_URL: postgres://user:pass@host.docker.internal:5432/dbname
```

Then OpenProject knows to reach the host machine (your bare metal DB at `192.168.1.3`) via the special DNS name `host.docker.internal`.

But if the IP isn't specified and `host.docker.internal` isn't used, then:

Possibilities:

1. Default to Docker Network DNS:• If you had a container named `postgres`, Docker would resolve `postgres` to its internal IP (`172.x.x.x`).
 - But since your DB is native, this won't work unless you manually alias it.
2. Hardcoded in `.env` or `.conf`:• OpenProject might be reading from a `.env` file or config volume.
 - Check for `DATABASE_URL`, `DB_HOST`, or similar in:• `/etc/openproject/configuration.yml`
 - Mounted volumes or secrets
3. NGINX Proxy Manager (NPM):• NGINX only proxies HTTP traffic, not DB connections.
 - So it's not involved in PostgreSQL connectivity.
4. Fallback to `localhost` (Fails):• If OpenProject tries `localhost:5432`, it'll look inside its own container — and fail unless PostgreSQL is containerized.


How to Fix or Confirm

 Explicitly Set Host IP in Compose

environment:

DATABASE_URL: postgres://user:pass@192.168.1.3:5432/dbname

Or use `host.docker.internal` if supported.

 Test from Inside Container

```
docker exec -it openproject_container bash
```

```
apt update && apt install -y netcat
```

```
nc -zv 192.168.1.3 5432
```

If it connects, you're golden.

 Legacy Protocol Tip

In your zipped DockerLab archive, include:

- A Networking README: explains bridge mode, IP mapping, and `host.docker.internal`
 - A Connectivity Test Script: `ping`, `nc`, or `telnet` to DB ports
 - A Codex Vignette: "The Invisible IP: When Apps Know Without Being Told"

Would you like help generating that README or scripting a connectivity test for mentees?

Have you noticed my question? I'm learning concepts?

Do you think you can create me the following:

1. Readme-core.md information about how to dockerlab to native db, include yml with portainer, pgadmin, phpmyadmin, npm nginx
2. Readme-app.mp for openproject, OpenEMR, osTicket, inventree, wikijs

Here you go, Rigel — two legacy-grade README files crafted for your DockerLab architecture. These are formatted in Markdown and ready to be zipped, gifted, or archived into your Codex.

 `README-core.md`

DockerLab Core Services — Native DB Integration

Overview

This stack provides core infrastructure tools for managing Docker containers and accessing native databases. It connects to bare metal MariaDB and PostgreSQL running on host IP `192.168.1.3`.

Services

- **Portainer** — Docker management UI
- **NGINX Proxy Manager** — Reverse proxy + SSL
- **phpMyAdmin** — Connects to native MariaDB
- **pgAdmin** — Connects to native PostgreSQL

Networking Notes

- Docker uses bridge mode (`172.x.x.x`)
- Native DBs must bind to `0.0.0.0` and allow connections from Docker subnet
- Use `192.168.1.3` or `host.docker.internal` in container configs

`docker-compose-core.yml`

```
version: '3.8'
services:
  portainer:
    image: portainer/portainer-ce
    ports:
      - "9000:9000"
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
      - portainer_data:/data
```

npm:
image: jc21/nginx-proxy-manager:latest
ports:
- "80:80"
- "81:81"
- "443:443"
volumes:
- npm_data:/data
- npm_letsencrypt:/etc/letsencrypt

phpmyadmin:
image: phpmyadmin/phpmyadmin
environment:
PMA_HOST: 192.168.1.3
PMA_PORT: 3306
ports:
- "8080:80"

pgadmin:
image: dpage/pgadmin4
environment:
PGADMIN_DEFAULT_EMAIL: admin@local.com
PGADMIN_DEFAULT_PASSWORD: strongpassword
ports:
- "5050:80"
volumes:
- pgadmin_data:/var/lib/pgadmin

volumes:
portainer_data:
npm_data:
npm_letsencrypt:
pgadmin_data:

 Security & Maintenance

- Restrict DB access to Docker subnet
- Use strong DB credentials
- Backup native DBs via cron + `mysqldump` / `pg_dump`


Legacy Notes

- Archive `.yaml` + this README in zipped folder
- Include DB setup scripts and backup routines
- Codex vignette: "Remembering the Bridge: Native DB Meets Docker"


 `README-app.md`

```markdown

# DockerLab Application Stack — Native DB Integration

##  Overview

This stack contains application containers that connect to native MariaDB and PostgreSQL running on host IP `192.168.1.3`.

##  Apps & DB Mapping

| App         | DB Type    | DB Host          | Notes               |
|-------------|------------|------------------|---------------------|
| OpenProject | PostgreSQL | 192.168.1.3:5432 | Uses `DATABASE_URL` |
| OpenEMR     | MariaDB    | 192.168.1.3:3306 | Uses `MYSQL_HOST`   |
| OSTicket    | MariaDB    | 192.168.1.3:3306 | Uses `MYSQL_HOST`   |
| Inventree   | PostgreSQL | 192.168.1.3:5432 | Uses `DB_HOST`      |
| Wiki.js     | PostgreSQL | 192.168.1.3:5432 | Uses `DB_HOST`      |

```
🖋️ docker-compose-app.yml
```

```
``yaml
```

```
version: '3.8'
```

```
services:
```

```
 openproject:
```

```
 image: openproject/community:latest
```

```
 environment:
```

```
 DATABASE_URL: postgres://openproject_user:securepass@192.168.1.3:543
```

```
2/openproject
```

```
 ports:
```

```
 - "8081:80"
```

```
 openemr:
```

```
 image: openemr/openemr:latest
```

```
 environment:
```

```
 MYSQL_HOST: 192.168.1.3
```

```
 MYSQL_PORT: 3306
```

```
 MYSQL_DATABASE: openemr
```

```
 MYSQL_USER: openemr_user
```

```
 MYSQL_PASSWORD: securepass
```

```
 ports:
```

```
 - "8083:80"
```

```
 osticket:
```

```
 image: osticket/osticket:latest
```

```
 environment:
```

```
 MYSQL_HOST: 192.168.1.3
```

```
 MYSQL_PORT: 3306
```

```
 MYSQL_DATABASE: osticket
```

```
 MYSQL_USER: osticket_user
```

```
 MYSQL_PASSWORD: securepass
```

```
 ports:
```

```
 - "8082:80"
```

inventree:

image: inventree/inventree:latest

environment:

DB\_HOST: 192.168.1.3

DB\_PORT: 5432

DB\_NAME: inventree

DB\_USER: inventree\_user

DB\_PASSWORD: securepass

ports:

- "8001:8000"

wikijs:

image: requarks/wiki:2

environment:

DB\_TYPE: postgres

DB\_HOST: 192.168.1.3

DB\_PORT: 5432

DB\_NAME: wikijs

DB\_USER: wikijs\_user

DB\_PASS: securepass

ports:

- "8084:3000"

### Connectivity Test

From inside any container:

```
nc -zv 192.168.1.3 3306
```

```
nc -zv 192.168.1.3 5432
```

### Legacy Notes

- Archive `.yaml` + this README in zipped folder
- Include DB setup scripts and backup routines

- Codex vignette: "Demembering the Bridge: Native DB Meets Docker"

---

These are now ready to be zipped and gifted. If you'd like, I can help you build a folder structure like:

```
DockerLab/ |— core/ | |— docker-compose-core.yml | |— README-core.md |— apps/ | |— docker-compose-app.yml | |— README-app.md |— protocols/ |— db-backup.sh |— Codex-vignette-demember.md
```

Just say the word and I'll help you populate it. Ingat, and 73 for now!

Now another request, repeat my request of core and app yml readme but add mariadb and Postgres in docker. I want to understand the difference between the two deployments so that I can mitigate and plans for backup and redundancy. What I'm doing currently is using hyper-v snapshot and then commit and then manual copy to another drive. I'm learning all these because I'm 59 and my son is 29 MD, if I design something for him, it has to be easy to debug and recover. I hope you understand.

I absolutely understand, Rigel — and I deeply admire your mission. You're not just building a DockerLab; you're crafting a legacy-grade system that your son can debug, recover, and extend with clarity and confidence. Let's walk through this with that goal in mind.

---

### What You Asked For

You want two updated README files:

1. [README-core.md](#) — includes Portainer, NGINX Proxy Manager, phpMyAdmin, pgAdmin, plus MariaDB and PostgreSQL running in Docker.
2. [README-app.md](#) — includes OpenProject, OpenEMR, OSTicket, Inventree, Wiki.js, all configured to connect to the Dockerized MariaDB and PostgreSQL.

You also want to understand the difference between native vs Dockerized DBs, especially for backup, recovery, and resilience — so your son can easily maintain it.

---

 [README-core.md](#) (with Dockerized MariaDB & PostgreSQL)

# DockerLab Core Services — Dockerized DB Deployment

## Overview

This stack provides core infrastructure tools for managing Docker containers and databases. Unlike the native DB setup, MariaDB and PostgreSQL are now containerized for portability and simplified deployment.

## Services

- **Portainer** — Docker management UI
- **NGINX Proxy Manager** — Reverse proxy + SSL
- **MariaDB** — Containerized SQL backend
- **PostgreSQL** — Containerized SQL backend
- **phpMyAdmin** — Connects to MariaDB container
- **pgAdmin** — Connects to PostgreSQL container

## `docker-compose-core.yml`

```
version: '3.8'
services:
```

mariadb:

image: mariadb:latest

environment:

MYSQL\_ROOT\_PASSWORD: rootpass

MYSQL\_DATABASE: coredb

volumes:

- mariadb\_data:/var/lib/mysql

ports:

- "3306:3306"

postgres:

image: postgres:latest

environment:

POSTGRES\_PASSWORD: rootpass

POSTGRES\_DB: coredb

volumes:

- postgres\_data:/var/lib/postgresql/data

ports:

- "5432:5432"

phpmyadmin:

image: phpmyadmin/phpmyadmin

environment:

PMA\_HOST: mariadb

ports:

- "8080:80"

pgadmin:

image: dpage/pgadmin4

environment:

PGADMIN\_DEFAULT\_EMAIL: admin@local.com

PGADMIN\_DEFAULT\_PASSWORD: strongpassword

ports:

- "5050:80"

volumes:

- pgadmin\_data:/var/lib/pgadmin

portainer:

image: portainer/portainer-ce

ports:

- "9000:9000"

volumes:

- /var/run/docker.sock:/var/run/docker.sock

- portainer\_data:/data

npm:

image: jc21/nginx-proxy-manager:latest

ports:

- "80:80"

- "81:81"

- "443:443"

volumes:

- npm\_data:/data

- npm\_letsencrypt:/etc/letsencrypt

volumes:

mariadb\_data:

postgres\_data:

pgadmin\_data:

portainer\_data:

npm\_data:

npm\_letsencrypt:

## Backup & Recovery

- Use `docker exec` + `mysqldump` or `pg\_dump` to export data
- Schedule backups with cron or systemd timers
- Store backups in `/mnt/backup/` and sync to external drive

## Legacy Notes

- Archive `.yaml` + this README in zipped folder
- Include backup scripts and test restore instructions
- Codex vignette: "Containers Within Containers: The DB Shift"

---

## 📄 `README-app.md` (Apps using Dockerized DBs)

```markdown

DockerLab Application Stack — Dockerized DB Integration

🏠 Overview

This stack contains application containers that connect to Dockerized MariaDB and PostgreSQL containers defined in `docker-compose-core.yaml`.

🗺️ Apps & DB Mapping

| App | DB Type | DB Host | Notes |
|-------------|------------|----------|---------------------|
| OpenProject | PostgreSQL | postgres | Uses `DATABASE_URL` |
| OpenEMR | MariaDB | mariadb | Uses `MYSQL_HOST` |
| OSTicket | MariaDB | mariadb | Uses `MYSQL_HOST` |
| Inventree | PostgreSQL | postgres | Uses `DB_HOST` |
| Wiki.js | PostgreSQL | postgres | Uses `DB_HOST` |

📄 docker-compose-app.yaml

```yaml

version: '3.8'

services:

  openproject:

    image: openproject/community:latest

```
environment:
 DATABASE_URL: postgres://openproject_user:securepass@postgres:543
2/openproject
depends_on:
 - postgres
ports:
 - "8081:80"
```

```
openemr:
image: openemr/openemr:latest
environment:
 MYSQL_HOST: mariadb
 MYSQL_PORT: 3306
 MYSQL_DATABASE: openemr
 MYSQL_USER: openemr_user
 MYSQL_PASSWORD: securepass
depends_on:
 - mariadb
ports:
 - "8083:80"
```

```
osticket:
image: osticket/osticket:latest
environment:
 MYSQL_HOST: mariadb
 MYSQL_PORT: 3306
 MYSQL_DATABASE: osticket
 MYSQL_USER: osticket_user
 MYSQL_PASSWORD: securepass
depends_on:
 - mariadb
ports:
 - "8082:80"
```

```
inventree:
image: inventree/inventree:latest
```

```
environment:
 DB_HOST: postgres
 DB_PORT: 5432
 DB_NAME: inventree
 DB_USER: inventree_user
 DB_PASSWORD: securepass
depends_on:
 - postgres
ports:
 - "8001:8000"
```

```
wikijs:
 image: requarks/wiki:2
 environment:
 DB_TYPE: postgres
 DB_HOST: postgres
 DB_PORT: 5432
 DB_NAME: wikijs
 DB_USER: wikijs_user
 DB_PASS: securepass
 depends_on:
 - postgres
 ports:
 - "8084:3000"
```

### Connectivity Test

```
docker exec -it openproject bash
apt update && apt install -y netcat
nc -zv postgres 5432
```

### Legacy Notes

- Archive `.yaml` + this README in zipped folder

- Include DB user creation scripts and restore walkthrough
- Codex vignette: "Dockerized Dependents: When Apps Talk to Containers"

---

### ## 🔄 Native vs Dockerized DBs: Comparison for Legacy Planning

| Feature                  | Native DB (Bare Metal)        | Dockerized DB (Container)          |
|--------------------------|-------------------------------|------------------------------------|
| <b>Performance</b>       | Direct I/O, best performance  | Slight overhead, still fast        |
| <b>Portability</b>       | Tied to host OS               | Easily moved, versioned            |
| <b>Backup Strategy</b>   | OS-level tools, cron, scripts | `docker exec` + volume snapshots   |
| <b>Redundancy</b>        | Manual replication            | Can use Docker Swarm/K8s           |
| <b>Debugging</b>         | Familiar system tools         | Container logs, `docker exec`      |
| <b>Risk of Data Loss</b> | Safer if managed well         | Higher if volumes not persisted    |
| <b>Recovery</b>          | Snapshot + restore            | Volume restore + container rebuild |
| <b>Mentoring Ease</b>    | Familiar to sysadmins         | Easier for devs, portable          |

---

### ## 🌈 Legacy Protocol Suggestion

Since you're snapshotting Hyper-V and manually copying, consider this hybrid:

- **Daily container volume backup** (via ``docker run --rm`` + ``tar``)
- **Weekly Hyper-V snapshot** (as full rollback)
- **Monthly zipped archive** of:
  - ``docker-compose-*.yml``
  - ``README-*.md``
  - DB dumps
  - Restore script

This way, your