

Docker 容器化 的企业级能耗系统运维

本方案重点解决你最关心的三个核心问题：

- 1. **数据安全性：** 容器删了数据还在。
- 2. **版本迭代：** 数据库加字段怎么自动生效，不需人工介入。
- 3. **灾难恢复：** 如何自动备份，以及炸机后如何恢复。

一、 运维目录结构设计 (Standard Directory Layout)

首先，规范服务器上的目录结构。**所有数据和配置必须挂载到宿主机**，严禁散落在容器内部。

```
/opt/ems-platform
├─ docker-compose.yml      # 核心编排文件
├─ .env                    # 环境变量（密码、密钥）
├─ /configs                # 配置文件挂载
│   └─ nginx/conf.d/default.conf
│       └─ emqx/emqx.conf
├─ /data                   # 【核心】数据持久化目录（自动生成）
│   └─ postgres/          # PG 数据
│   └─ taos/              # TDengine 数据
│       └─ redis/         # Redis 数据
├─ /backups                # 【核心】自动备份落地目录
│   └─ postgres/
│       └─ tdengine/
├─ /db-migrations          # 数据库升级脚本（SQL文件）
│   └─ 000001_init.up.sql
│       └─ 000002_add_field.up.sql
└─ /scripts                # 运维脚本
    └─ backup_tdengine.sh
```

二、 数据库变更方案 (Schema Migration)

场景： 业务迭代，需要在 device 表新增字段 last_maintenance_date 。

原则： **禁止**进入数据库手动执行 ALTER TABLE ，必须代码化管理。

1. 选型

使用 **Golang-Migrate**。它能保证数据库结构的版本与代码版本严格一致。

2. workflow

1. **开发阶段**：开发者在代码库生成 `000002_add_field.up.sql`。
2. **部署阶段**：Go 服务启动时，会自动检测并执行这个 SQL。

3. 代码集成 (main.go)

在你的 Go 后端入口加入自动迁移逻辑：

```
import (  
    "github.com/golang-migrate/migrate/v4"  
    _ "github.com/golang-migrate/migrate/v4/database/postgres"  
    _ "github.com/golang-migrate/migrate/v4/source/file"  
)  
  
func initSchema(dbUrl string) {  
    // 指向容器内的 SQL 文件目录  
    m, err := migrate.New("file:///app/migrations", dbUrl)  
    if err != nil {  
        log.Fatal("迁移初始化失败:", err)  
    }  
    // 自动执行 Up 操作，追平最新版本  
    if err := m.Up(); err != nil && err != migrate.ErrNoChange {  
        log.Fatal("数据库升级失败:", err)  
    }  
    log.Println("数据库结构已升级到最新版本")  
}
```

4. Dockerfile 调整

确保 SQL 文件被打入镜像：

```
# ... build stage ...  
COPY ./db/migrations /app/migrations  
CMD ["/ems-server"]
```

三、 自动化备份方案 (Auto Backup)

1. PostgreSQL 自动备份 (Sidecar 模式)

在 `docker-compose.yml` 中增加一个专用容器，它是 PG 的“僚机”，负责每天半夜把数据打包出来。

```
# 数据库服务
postgres:
  image: postgres:15
  volumes:
    - ./data/postgres:/var/lib/postgresql/data # 数据持久化
  # ...

# 备份服务 (Sidecar)
pg-backup:
  image: prodrigestivill/postgres-backup-local
  restart: always
  environment:
    - POSTGRES_HOST=postgres
    - POSTGRES_DB=ems_db
    - POSTGRES_USER=ems_user
    - POSTGRES_PASSWORD=${DB_PASSWORD}
    - SCHEDULE=@daily # 每天 00:00 备份
    - BACKUP_KEEP_DAYS=7 # 只保留最近 7 天
    - BACKUP_DIR=/backups
  volumes:
    - ./backups/postgres:/backups # 映射到宿主机
  depends_on:
    - postgres
```

2. TDengine 自动备份 (Host Script 模式)

由于 TDengine 数据量极大，且官方暂无 Sidecar 镜像，建议使用宿主机 Crontab 调度。

****脚本：** `/opt/ems-platform/scripts/backup_td.sh`******

```
#!/bin/bash
# 设置变量
CONTAINER_NAME="ems-tdengine"
BACKUP_ROOT="/opt/ems-platform/backups/tdengine"
DATE=$(date +%Y%m%d_%H%M)
TARGET_DIR="$BACKUP_ROOT/$DATE"

mkdir -p $TARGET_DIR

# 1. 调用容器内的 taosdump 工具导出元数据和数据
echo "Starting TDengine backup..."
docker exec $CONTAINER_NAME taosdump -o /tmp/dump_out -D power_db

# 2. 将备份文件从容器复制到宿主机
docker cp $CONTAINER_NAME:/tmp/dump_out/. $TARGET_DIR/

# 3. 清理容器内临时文件
docker exec $CONTAINER_NAME rm -rf /tmp/dump_out

# 4. 压缩 (时序数据通常很大, 建议压缩)
cd $BACKUP_ROOT
tar -czf $DATE.tar.gz $DATE
rm -rf $DATE

# 5. 清理超过 30 天的备份
find $BACKUP_ROOT -name "*.tar.gz" -mtime +30 -delete

echo "Backup completed: $TARGET_DIR"
```

加入宿主机 Crontab:

```
# 每天凌晨 2 点执行
0 2 * * * /bin/bash /opt/ems-platform/scripts/backup_td.sh >> /var/log/ems_backup.log 2>&1
```

四、灾难恢复演练 (Disaster Recovery)

假设服务器硬盘损坏, 或者有人误删库, 如何通过备份文件恢复?

1. 恢复 PostgreSQL

1. 停止后端服务，防止写入

```
docker stop ems-server
```

2. 获取备份文件（例如昨天的备份）

```
BACKUP_FILE="./backups/postgres/ems_db-20231027.sql.gz"
```

3. 强行恢复（Drop 现有库并重建）

```
gunzip -c $BACKUP_FILE | docker exec -i ems-postgres psql -U ems_user -d ems_db
```

2. 恢复 TDengine

1. 解压备份

```
tar -xzf 20231027_0200.tar.gz
```

```
cd 20231027_0200
```

2. 拷贝进容器

```
docker cp . ems-tdengine:/tmp/restore_data
```

3. 执行恢复

```
docker exec ems-tdengine taosdump -i /tmp/restore_data
```

五、完整的 Docker Compose 配置

这是集成了上述所有理念的最终配置。

version: '3.8'

services:

业务服务

app-server:

image: my-ems/server:latest

restart: always

env_file: .env

volumes:

- ./configs/app.yaml:/app/config.yaml
- ./logs:/app/logs

depends_on:

- postgres
- tdengine
- redis

nginx:

image: nginx:alpine

ports: ["80:80", "443:443"]

volumes:

- ./configs/nginx:/etc/nginx/conf.d
- ./html:/usr/share/nginx/html
- ./certs:/etc/nginx/certs

数据服务（带持久化）

postgres:

image: postgres:15

container_name: ems-postgres

restart: always

environment:

POSTGRES_DB: ems_db

POSTGRES_USER: ems_user

POSTGRES_PASSWORD: \${DB_PASSWORD}

volumes:

- ./data/postgres:/var/lib/postgresql/data # 数据持久化
- /etc/localtime:/etc/localtime:ro

tdengine:

image: tdengine/tdengine:3.0

```
container_name: ems-tdengine
ports: ["6030:6030"]
volumes:
  - ./data/taos/data:/var/lib/taos # 数据持久化
  - ./data/taos/log:/var/log/taos
  - /etc/localtime:/etc/localtime:ro

redis:
  image: redis:7
  volumes:
    - ./data/redis:/data # 数据持久化

# -----
# 运维服务 (Sidecar)
# -----

pg-backup:
  image: prodrigestivill/postgres-backup-local
  restart: always
  environment:
    - POSTGRES_HOST=postgres
    - POSTGRES_DB=ems_db
    - POSTGRES_USER=ems_user
    - POSTGRES_PASSWORD=${DB_PASSWORD}
    - SCHEDULE=@daily
    - BACKUP_KEEP_DAYS=7
    - BACKUP_DIR=/backups
  volumes:
    - ./backups/postgres:/backups
  depends_on:
    - postgres
```

六、日常运维命令手册 (Cheat Sheet)

1. 发布新版本 (含数据库变更):

这一步会自动拉取新镜像, 重建容器, 启动时 Go 程序会自动执行 SQL 迁移

```
docker-compose up -d --build app-server
```

2. 查看实时日志:

```
docker-compose logs -f --tail=100 app-server
```

3. 检查数据库状态:

```
# 进 PG
```

```
docker exec -it ems-postgres psql -U ems_user -d ems_db
```

```
# 进 TDengine
```

```
docker exec -it ems-tdengine taos
```