Manual de Deploy de Aplicações na Plataforma DEISI

Documento de apoio para alunos e docentes

Criar repositório · Dockerfile · Workflow · Variáveis de ambiente · Base de dados · Migrations

URL final esperado

https://nome-da-app.apps.deisi.ulusofona.pt

Objetivo deste manual

Este documento foi escrito para uma pessoa que nunca usou a plataforma. Serve para qualquer tecnologia, desde que a aplicação consiga correr dentro de um container Docker.

1. Ideia geral

A plataforma permite colocar uma aplicação online a partir de um repositório GitHub. O utilizador prepara o código, o Dockerfile e um ficheiro de workflow. Depois faz push para a branch main. A pipeline central trata do build, da publicação da imagem e do deploy no cluster.

> **Importante**
> A plataforma é genérica. Não é específica para Django, Flask, Node.js, Java, PHP ou frontend. A única regra é: a aplicação tem de conseguir correr dentro de um container Docker e escutar na porta indicada no workflow.

O que a plataforma faz automaticamente

- Faz checkout do repositório da aplicação.
- Constrói a imagem Docker a partir do Dockerfile.
- Publica a imagem no GitHub Container Registry (GHCR).
- Cria ou atualiza os Secrets necessários no Kubernetes.
- Faz deploy da aplicação no cluster Kubernetes através do Helm chart central.
- Cria o Ingress e disponibiliza a aplicação por HTTPS no domínio institucional.
- Opcionalmente, injeta variáveis de ambiente, cria uma base de dados PostgreSQL e corre migrations.

Fluxo resumido

```
Repositório GitHub
  -> GitHub Actions
    -> Docker build
      -> GHCR
        -> Kubernetes + Helm
          -> https://nome-da-app.apps.deisi.ulusofona.pt
```

2. O que é preciso preparar

Antes de configurar o deploy, confirmar que o repositório tem estes elementos:

Requisito	O que significa
Repositório na organização deisi-org	O nome do repositório é usado para gerar o nome da aplicação e o URL final.
Branch main	O workflow deste manual corre quando existe push para a branch main.
Código da aplicação	Pode ser backend, frontend, API, monolito, site estático, etc.
Dockerfile funcional	Tem de construir a imagem e arrancar a aplicação dentro do container.
Porta conhecida	A app tem de escutar numa porta interna, por exemplo 3000, 8000, 8080 ou 80.
Secrets fora do código	Passwords, tokens e chaves devem ficar em GitHub Secrets, não no repositório.

Estrutura mínima

```
minha-app/
|-- .github/
|   `-- workflows/
|       `-- deploy.yml
|-- Dockerfile
`-- código-da-aplicação
```

Nome do repositório

Se o repositório se chamar caudais, o URL esperado será https://caudais.apps.deisi.ulusofona.pt. Convém usar nomes simples, em minúsculas e com hifens se necessário.

3. Criar o ficheiro de deploy

No repositório da aplicação, criar o ficheiro:

```
.github/workflows/deploy.yml
```

Para uma aplicação simples, sem base de dados e sem variáveis secretas, usar:

```yaml
name: Deploy

on:
  push:
    branches: [main]

permissions:
  contents: read
  packages: write

jobs:
  deploy:
    uses: deisi-org/platform-orchestrator/.github/workflows/deploy.yml@main
    with:
      app_port: "3000"
    secrets: inherit
```

Valor a alterar

Na maior parte dos casos, o único valor obrigatório a alterar é app_port. Esse valor tem de ser a porta onde a aplicação corre dentro do container Docker.

Regra prática

Se a aplicação arranca com 0.0.0.0:3000, usar app_port: "3000". Se arranca com 0.0.0.0:8000, usar app_port: "8000".

4. Escolher a porta correta

A porta costuma aparecer no Dockerfile, no comando de arranque da aplicação ou no ficheiro de configuração do framework.

Tecnologia	Porta comum	Observação
Django / Python	8000	Depende do gunicorn, uvicorn ou manage.py runserver usado no Dockerfile.
Flask	5000 ou 8000	Pode mudar se a app definir outra porta.
FastAPI	8000	Normalmente com uvicorn.
Node.js / Express	3000	Porta muito comum em apps Node.
React / Vite	3000	Se estiver a ser servido por Node. Se for site estático com Nginx, normalmente 80.
Spring Boot	8080	Porta padrão em muitos projetos Java.
Nginx / site estático	80	Quando o container usa Nginx.

Exemplo de Dockerfile Node

```
EXPOSE 3000
CMD ["npm", "start"]
```

Neste caso, usar:

```
app_port: "3000"
```

Erro comum

Dentro do container, a aplicação deve escutar em 0.0.0.0 e não apenas em localhost. Se escutar apenas em localhost, o Kubernetes pode não conseguir encaminhar tráfego para a app.

5. Variáveis de ambiente e secrets

Se a aplicação precisa de chaves, tokens, passwords ou configurações sensíveis, estes valores não devem ficar escritos no código nem no deploy.yml. Devem ser colocados num GitHub Secret chamado APP_RUNTIME_ENV.

Como criar o secret

- Abrir o repositório da aplicação no GitHub.
- Ir a Settings -> Secrets and variables -> Actions.
- Criar um novo Repository secret chamado APP_RUNTIME_ENV.
- No valor do secret, escrever as variáveis no formato de ficheiro .env.

Exemplo de APP_RUNTIME_ENV

```
SECRET_KEY=valor-secreto
DEBUG=false
API_TOKEN=token-privado
EMAIL_HOST=smtp.exemplo.pt
EMAIL_PORT=587
```

A pipeline transforma este GitHub Secret num Kubernetes Secret e injeta-o automaticamente no pod da aplicação. No código, a app deve ler estas variáveis a partir do ambiente.

Exemplos no código

```
// Node.js
const token = process.env.API_TOKEN;

# Python
import os
token = os.environ.get("API_TOKEN")
```

O que fica visível

O conteúdo do APP_RUNTIME_ENV não é escrito nos logs da pipeline. A pipeline pode indicar que o secret foi criado/atualizado, mas os valores sensíveis não devem ser impressos.

6. Aplicações com base de dados PostgreSQL

Se a aplicação precisa de uma base de dados PostgreSQL, basta adicionar database_enabled: true ao workflow. A plataforma cria automaticamente uma database, um utilizador, uma password e um Kubernetes Secret para essa aplicação.

Workflow com base de dados

```
name: Deploy

on:
  push:
    branches: [main]

permissions:
  contents: read
  packages: write

jobs:
  deploy:
    uses: deisi-org/platform-orchestrator/.github/workflows/deploy.yml@main
    with:
      app_port: "3000"
      database_enabled: true
    secrets: inherit
```

O que é criado automaticamente

```
App/repo: minha-app

Database: minha_app_db
User:     minha_app_user
Secret:   minha-app-db-secret
```

A base de dados é criada no sistema central de gestão de bases de dados PostgreSQL do cluster. Não é criada uma instância PostgreSQL por aplicação. As credenciais ficam guardadas num Kubernetes Secret, no namespace das aplicações, e são injetadas no pod da aplicação.

```
PostgreSQL central
  -> database minha_app_db
  -> database outra_app_db
  -> database projeto_final_db

Pod da aplicação
  -> recebe DATABASE_URL, DB_HOST, DB_NAME, DB_USER, DB_PASSWORD
```

7. Como a aplicação usa a base de dados

A aplicação não deve ter a password escrita no código. Deve ler as credenciais a partir das variáveis de ambiente que a plataforma injeta automaticamente.

Variáveis disponibilizadas no pod

Variável	Para que serve
DATABASE_URL	String completa de ligação PostgreSQL. É a opção mais simples para muitas frameworks.
DB_HOST	Host interno do PostgreSQL: postgres.databases.svc.cluster.local
DB_PORT	Porta 5432
DB_NAME	Nome da base de dados da app
DB_USER	Utilizador da base de dados da app
DB_PASSWORD	Password guardada em Kubernetes Secret
POSTGRES_*	Aliases equivalentes para apps que esperam nomes POSTGRES_DB, POSTGRES_USER, etc.

Exemplos

```
// Node.js
const databaseUrl = process.env.DATABASE_URL;

# Python
import os
database_url = os.environ.get("DATABASE_URL")
```

Password da base de dados

A password não aparece no deploy.yml nem nos logs da pipeline. Fica guardada num Kubernetes Secret e é disponibilizada apenas como variável de ambiente dentro do pod da aplicação.

Informação apresentada nas Actions

```
Database requested for app: minha-app
Database name: minha_app_db
Database user: minha_app_user
Database secret: minha-app-db-secret
Database provisioned successfully.
```

A pipeline mostra o nome da database, o user e o nome do Secret para ajudar no diagnóstico. A password não deve ser mostrada.

8. Migrations da base de dados

Se a aplicação precisa de criar tabelas ou atualizar o schema da base de dados, deve indicar um comando de migrations. Esse comando é executado antes do deploy da aplicação, num Job temporário dentro do Kubernetes.

Workflow com base de dados e migrations

```
jobs:
  deploy:
    uses: deisi-org/platform-orchestrator/.github/workflows/deploy.yml@main
    with:
      app_port: "3000"
      database_enabled: true
      migrations_command: "comando das migrations"
    secrets: inherit
```

Exemplos de comandos

Tecnologia	Exemplo de migrations_command
Django	python manage.py migrate --noinput
Flask / script próprio	python migrate.py
Node.js / script próprio	node migrate.js
Prisma	npx prisma migrate deploy
Alembic	alembic upgrade head
Laravel	php artisan migrate --force
Flyway	flyway migrate

Job de migrations

Quando existe migrations_command, a plataforma cria um pod temporário apenas para correr esse comando. Esse pod não é a base de dados. A base de dados continua no PostgreSQL central.

9. Exemplos completos de workflows

Aplicação sem base de dados

```
jobs:
  deploy:
    uses: deisi-org/platform-orchestrator/.github/workflows/deploy.yml@main
    with:
      app_port: "3000"
    secrets: inherit
```

Aplicação com PostgreSQL, sem migrations

```
jobs:
  deploy:
    uses: deisi-org/platform-orchestrator/.github/workflows/deploy.yml@main
    with:
      app_port: "3000"
      database_enabled: true
    secrets: inherit
```

Aplicação Django com PostgreSQL

```
jobs:
  deploy:
    uses: deisi-org/platform-orchestrator/.github/workflows/deploy.yml@main
    with:
      app_port: "8000"
      database_enabled: true
      migrations_command: "python manage.py migrate --noinput"
    secrets: inherit
```

Aplicação Node.js com PostgreSQL

```
jobs:
  deploy:
    uses: deisi-org/platform-orchestrator/.github/workflows/deploy.yml@main
    with:
      app_port: "3000"
      database_enabled: true
      migrations_command: "node migrate.js"
    secrets: inherit
```

10. Fazer deploy e acompanhar nas Actions

Depois de adicionar o Dockerfile e o deploy.yml, fazer commit e push para a branch main.

```
git add .
git commit -m "Add deploy workflow"
git push
```

Depois do push, abrir o repositório no GitHub e ir a:

```
Actions -> Deploy
```

Se a pipeline ficar verde, o deploy foi concluído. No final dos logs, deve aparecer o URL da aplicação.

O que se pode ver nos logs

Informação	Exemplo	Observação
Nome da app	APP_NAME=minha-app	Gerado a partir do nome do repositório.
URL final	https://minha-app.apps.deisi.ulusofona.pt	Endereço para abrir no browser.
Imagem Docker	ghcr.io/deisi-org/minha-app:	Imagem criada pela pipeline.
Runtime env	APP_RUNTIME_ENV secret was not provided / secret created	Não mostra os valores do secret.
Base de dados	Database name, user, secret	Mostra dados de diagnóstico, não mostra password.
Migrations	Migrations command provided	Mostra o comando executado.

Se a pipeline falhar

Abrir o passo vermelho nas Actions e ler o erro. Os problemas mais comuns são Dockerfile inválido, porta errada, migration com erro ou variável de ambiente em falta.

11. Problemas comuns

Sintoma	Causa comum	Como resolver
A pipeline não arranca	O push não foi para main ou o ficheiro não está em .github/workflows/deploy.yml	Confirmar branch main e caminho do ficheiro.
Docker build falha	Dockerfile incompleto ou dependências em falta	Testar docker build localmente e corrigir o Dockerfile.
Deploy fica verde mas site não abre	Porta errada em app_port ou app a escutar em localhost	Confirmar porta no Dockerfile/CMD e usar 0.0.0.0.
Migrations falham	Comando errado, dependências em falta ou app ainda configurada para SQLite	Ver logs do Job de migrations e corrigir configuração da BD.
App não encontra variáveis	APP_RUNTIME_ENV não foi criado ou variável tem outro nome	Criar/atualizar GitHub Secret APP_RUNTIME_ENV.
App não liga a BD	Código não lê DATABASE_URL/DB_* ou depende de localhost	Alterar código para ler variáveis de ambiente.

Nota sobre acesso externo a BD

As aplicações ligam-se à base de dados internamente, sem port-forward. O port-forward só é necessário para acesso manual externo, por exemplo a partir de DBeaver. A base de dados não fica exposta publicamente por segurança.

12. Checklist final

OK	Confirmação
	O repositório está na organização deisi-org.
	A branch principal chama-se main.
	Existe um Dockerfile funcional.
	A aplicação escuta em 0.0.0.0 e na porta correta.
	Existe o ficheiro .github/workflows/deploy.yml.
	O valor app_port corresponde à porta usada dentro do container.
	Segredos foram colocados em APP_RUNTIME_ENV e não no código.
	Se houver PostgreSQL, foi usado database_enabled: true.
	Se forem necessárias migrations, foi definido migrations_command.
	Foi feito push para main.
	A Action Deploy ficou verde.
	O URL final abriu no browser.

Resumo

Criar repositório, adicionar código, garantir Dockerfile funcional, criar deploy.yml, indicar a porta correta, configurar secrets se necessário, ativar base de dados se necessário, fazer push e abrir o URL gerado.