

Lab 1 Ambiente de desenvolvimento em equipa (maven, git, containers)

Objetivos de aprendizagem

- Utilizar um *framework* para a construção de projetos com gestão explícita de dependências.
- Aplicar práticas de gestão cooperativa do código fonte com o git.
- Utilizar tecnologia de containers para facilitar a preparação de ambientes de execução (e.g.: Docker).

Recursos e referências

- Livros gratuitos com exploração detalhada dos conceitos: “[Maven by Example](#)”; “[Pro Git](#)”
- Vídeo: [introdução ao Docker](#).

Entrega

Este guião (Lab 1) cobre duas semanas.

Cada aluno deve entregar, no Moodle, evidência da realização do guião. Para isso, deve preparar um ficheiro zip (único) por guião, com uma pasta por cada secção do guião (lab1.1, lab1.2,...). Na raiz, deve existir um ficheiro “[readme.md](#)” (ou [readme.html](#)) com a identificação do aluno e as explicações/respostas pedidas ao longo do exercício.

Use o **readme.md como um bloco de notas**; neste ficheiro, adicione notas para o ajudar a estudar mais tarde, definições, ..., e, claro, as respostas às perguntas colocados ao longo do exercício.

Os itens obrigatórios para constar da entrega estão assinalados no guião com 📌.

1.1 Gestão da *build* com *maven*

A construção um projeto de código pode incluir [várias etapas](#) (também chamadas “objetivos”) tais como a obtenção de dependências (componentes/bibliotecas externos), compilação do código fonte, produção de binários (*artifacts*), atualização da documentação, instalação no servidor, etc.

Em projetos de médio e grande porte, estas tarefas são coordenadas por uma ferramenta de construção (***build tool***); em Java, as *build tools* mais usadas são o [Maven](#) e o [Gradle](#).

O [Maven](#) é muito popular entre projetos profissionais em Java.

Preparar o ambiente para usar Maven

Os projetos configurados para usar o Maven podem ser abertos nos principais IDE, que reconhecem as configurações do Maven. No entanto, vamos instalar e configurar o Maven para estar disponível também na linha de comandos ([CLI](#)).

Todo o **ciclo da *build* pode ser gerido a partir da linha de comandos**, o que torna o Maven uma ferramenta flexível para integrar com ambientes de desenvolvimento (IDE) e em ferramentas de integração contínua (sem terminal/GUI interativo).

- [Instalar o Maven](#) no ambiente de desenvolvimento. (Requer o JDK; sugere-se a utilização do [JDK 11 LTS](#))

Nota: a variável de ambiente “**JAVA_HOME**” deve existir e indicar a pasta **raiz do JDK** (não o JRE).

Pode verificar a instalação do Maven (deve obter a indicação da instalação do Maven e do JDK):

```
$ mvn --version
Apache Maven 3.6.0 (97c98ec64a1fdfee7767ce5fffb20918da4f719f3; 2018-10-
24T19:41:47+01:00)
Maven home: /mnt/c/opt/apache-maven-36
Java version: 11.0.8, vendor: Ubuntu, runtime: /usr/lib/jvm/java-11-openjdk-
amd64
```

- b) Familiarize-se com os comandos principais executando o [guia introdutório: “Maven in 5 minutes”](#).
Adapte o “*groupId*” e “*artifactId*” para um exemplo mais adequado/personalizado, tendo em consideração as [regras convencionadas](#).

📄 No seu “bloco de notas (ficheiro “*readme*” da aula), explique o que é um *archetype*, *groupId* e *artifactId*.

Preparar um novo projeto Maven (aplicação para consulta da meteorologia)

Vamos agora criar uma pequena aplicação Java para invocar a [API da previsão meteorológica](#) disponível no [IPMA](#).

- c) Comece por perceber a estrutura do pedido e das respostas da API (ver “[Previsão Meteorológica Diária até 5 dias agregada por Local](#)”). Para isso, pode utilizar o *browser* (ou o comando *curl*). Por exemplo, para pedir a previsão dos próximos 5 dias para a cidade de Aveiro (que tem o código interno 1010500):

```
$ curl http://api.ipma.pt/open-
data/forecast/meteorology/cities/daily/1010500.json | json_pp
```

- d) Crie um projeto (MyWeatherRadar) para uma aplicação Java, baseado em Maven.
Pode fazê-lo:
- na linha de comandos ([archetype:generate](#)), ou
- num IDE com suporte para o Maven (e.g.: [VS Code](#), [IntelliJ IDEA Ultimate](#)¹). Certifique-se que o seu IDE tem as extensões necessárias.
Em todos os projetos Maven, deve indicar o “*groupId*”, “*artifactId*”; a “*versão*” inicial de um projeto deve ser “1.0-SNAPSHOT”.
- e) Verifique o conteúdo do ficheiro POM.xml e a [estrutura de pastas que foi criada](#).
- f) Configure alguns metadados do projeto (no POM), tais como: [a composição da equipa de desenvolvimento](#) e as propriedades que definem [versão do compilador](#) a usar (*compiler plugin properties*, *versão=11*).

```
<properties>
  <maven.compiler.source>11</maven.compiler.source>
  <maven.compiler.target>11</maven.compiler.target>
</properties>
```

Configurar dependências

A aplicação pretendida precisa de abrir uma ligação HTTP, criar um pedido GET, obter o JSON com a resposta, processar o conteúdo da resposta e usar o resultado.

Há vários passos envolvidos que podemos implementar de raiz, ou, mais naturalmente, recorrendo a bibliotecas externas credíveis. O uso dessas bibliotecas (ou *artifacts*) dá origem a [dependências](#) e o Maven ajuda nessa gestão.

- g) Neste exercício, vamos usar dependências para as bibliotecas [Retrofit](#) (fazer um pedido a uma API REST, em Java) e [Gson](#) (fazer o *parsing* do JSON para classes). Comece por declarar estas

¹ Está incluído na [licença académica](#).

dependência no POM.xml do projeto (ver passo [#2, aqui](#)). Em geral, deve usar as versões mais recentes de cada dependência.



No POM, declaramos as dependências diretas do nosso projeto ([retrofit](#), [converter-gson](#)); estes *artifacts*, por sua vez, terão outras dependências associadas.

- h) Numa versão inicial da solução, o corpo principal (*main*) pode ter uma estrutura [semelhante a esta](#).

Nota: utilize a [implementação indicativa das classes](#) `IpmaService`, `IpmaCityForecast` e `CityForecast` disponíveis, necessárias para a execução do projeto.

Assegure-se que o projeto está sem erros e experimente executá-lo.

Note que, tratando-se de um projeto configurado com o Maven, pode ser aberto e executado, sem alterações, em diferentes IDE e também na linha de comandos:

```
$ mvn clean package
$ mvn exec:java -Dexec.mainClass="weather.WeatherStarter"
```

- i) Altere a implementação para receber o código da cidade como parâmetro pela linha de comandos e imprima a informação da previsão de forma mais amigável e completa.

Note que na execução (`mvn exec:java`) pode também indicar argumentos (`exec.args`).

Entregar o projeto de código (completo), depois de fazer “mvn clean”.

No ficheiro “readme” da aula, responder às questões: o que é um “maven goal”? quais **os principais** “maven goals” e a respetiva sequência de invocação?

- j) **[Exercício opcional]** confirme que as classes têm documentação adequada (Javadoc) na descrição da classe e métodos (adicione a documentação em falta).

[Gere um site com a documentação](#) do projeto e explore a documentação produzida num browser (`./target/apidocs/index.html`)

```
$ mvn site
```

Para a maior parte das instalações, o objetivo anterior vai falhar. Trata-se de um conflito entre as versões dos *plugins* do Maven assumidas por omissão e requerida.

Resolva o problema [especificando a versão correta do plug-in](#).

Certifique-se que [inclui ainda a geração do javadoc na fase de reporting](#). (*Generate Javadocs As Part Of Project Reports*)

Verifique o relatório (site) gerado e o JavaDoc.

1.2 Gestão do código fonte

Certifique-se que tem a instalação do Git disponível na linha de comandos no seu ambiente de desenvolvimento.

```
$ git config --list
```

Se não obtiver um output com as definições do *git*, incluindo o seu nome e email, então [complete/configure a instalação do git](#).

Os exercícios baseiam-se na linha de comandos. A utilização de clientes gráficos é opcional (e.g.: [SmartGit](#), [SourceTree](#), [GitKraken](#)). Para além disso, os IDE têm suporte integrado para executar comandos *git*.

- a) [Facultativo] Para uma a iniciação ao *git*, faça o exercício proposto na [documentação do Bitbucket](#).
Caso já esteja familiarizado com o *git*, pode avançar para os próximos passos.
Nota: apesar do tutorial usar o Bitbucket, não há nenhuma preferência por este ambiente em detrimento de outros.

Fluxo de trabalho em Git

Os passos seguintes supõem uma equipa (2+ alunos); o “grupo” pode ser formado de forma espontânea e informal.

- b) Crie um repositório (remoto) que servirá para a equipa partilhar o código. Existem, para isso, vários recursos gratuitos disponíveis: GitHub, GitLab, BitBucket,...
- Tome nota do URL para acesso ao repositório.
- c) Tomando como ponto de partida o projeto já existente das alíneas anteriores, vamos partilhá-lo no repositório.
- Comece por adicionar a informação sobre as exclusões, isto é, indicar os ficheiros/pastas que só têm interesse local e não devem ser passados para o repositório comum (e.g.: a pasta `./target`, que tem as versões compiladas, não deve ir para o repositório).
- Para isso, [crie um ficheiro `.gitignore` na raiz do projeto](#) adequando aos projetos Maven.
- d) Adicione suporte ao projeto para trabalhar com *git* e, de seguida e publique as alterações para o repositório remoto. **Apenas um** dos membros da equipa deve partilhar o projeto.

Exemplo indicativo (tem de adaptar para o seu caso):

```
$ cd project_folder
$ git init                # initialize a local git repo in this folder
$ git remote add origin git@gitlab.com:ico\_gl/ies-labs.git #adapt the url
$ git add .               # mark all existing changes in this root to be committed
$ git commit -m "Initial commit" #create the commit snapshot locally
$ git push -u origin master #uploads the local commit to the shared repo
```

- e) Depois, o(s) outro(s) membro(s) deve obter uma cópia de trabalho (i.e., *clone*).
- f) Considere aqui que a equipa vai criar duas novas funcionalidades (ou *features*) no projeto.
- Certifique-se que adota um modelo de trabalho (*workflow*) cooperativo e distribuído, tal como o [feature-branch workflow](#).
- Neste caso, cada *feature* pode ficar entregue a um membro da equipa:

- (1) Adicione suporte para *logging* para ficheiro, i.e., a aplicação deve **escrever um log** com o registo das operações que vão sendo feitas, bem como de eventuais problemas que tenham ocorrido. O log pode ser direcionado para a consola ou para ficheiro (ou para ambos).
- Recorra a uma biblioteca adequada de *logging* para Java [como o Log4j 2](#).

Note que no exemplo do *link* existe um [ficheiro de configuração](#) (`log4j2.xml`) que deve ser colocado na pasta de recursos do projeto maven, seguindo a [estrutura convencional](#).

- (2) Adicione o suporte necessário para receber (como argumento da linha de comandos) o **nome da cidade** que se pretende consultar, em vez do código, e apresenta a previsão para os 5 dias seguintes.

Sugestão: apesar de poder [consultar programaticamente o código associado ao nome da cidade](#) usando a API do IPMA, para este efeito, basta manter um dicionário (hashmap) local e estático (nome da cidade, código do IPMA).

- 📄 Entregar o log das operações de git realizadas, que pode ser obtido com:
\$ git log --graph --oneline --decorate > lab1-2.log
- 📄 Entregar o projeto, com as alterações.

1.3 Introdução ao Docker

A utilização de *containers* (juntamente com ficheiros que contém as instruções para preparar um ambiente de execução de uma solução) [facilita a vida](#) do *developer* (é mais fácil preparar e partilhar um ambiente) e da produção (“operations”).
Iremos recorrer várias vezes a *containers* na disciplina.

a) Começamos por [instalar o Docker Engine – Community](#).

[Windows] O ambiente do Docker Desktop permite correr Docker em Windows. Há várias opções de configuração disponíveis; se instalar o Windows Linux Subsystem v2, [pode integrar o “Docker do Windows” e o “Docker da distribuição Linux”](#).

[Linux] Certifique-se que usa o [Docker com um utilizador não-root](#) (sem precisar de sudo).

b) Faça o passo 1 do tutorial do Docker [“Orientation and Setup”](#).

- 📄 Recorde que deve usar o ficheiro “readme.md” (da entrega) como um bloco de notas. Certifique-se que inclui uma síntese (“cheat sheet”) dos comandos docker usados e para que servem.

c) Faça o passo 2 do tutorial do Docker [“Build and run you image”](#).

d) Embora toda a gestão dos *containers* possa ser feita na linha de comandos, por vezes é mais fácil usar um ambiente gráfico. O *portainer* é uma aplicação web que permite aceder, no browser, ao ambiente de gestão do Docker.

[Instalar o portainer](#) (ver secção “Portainer running within Docker”) a partir da imagem partilhada no Docker Hub.

Depois de instalado, aceda a <http://localhost:9000> (na primeira vez, defina as credenciais de administração); quando perguntado, escolha gerir *containers* Docker (e não Kubernetes).


e) Utilize o Docker para criar uma imagem do servidor PostgreSQL ([“Dockerize PostgreSQL”](#)).


Certifique-se que faz as configurações necessárias (Dockerfile) para inicializar o servidor e deixar o serviço a reiniciar automaticamente (serviço). A área de dados (da base de dados) deve persistir entre *reboots* (a base de dados não deve ser volátil).

Comprove que consegue aceder ao servidor PostgreSQL através de uma aplicação cliente.
(Nota: verifique o porto em que o *container* publica o serviço).

f) Frequentemente, o ambiente alvo é composto por vários serviços e, nesse caso, é útil configurar vários *containers* tratados em conjunto. Para isso, o ambiente do Docker disponibiliza uma ferramenta chamada “Docker composer”.

Complete os [exercícios tutoriais](#) propostos na documentação. No exemplo, há dois serviços (web server com Flask e uma base de dados com Redis) a usados/geridos em conjunto.

 Resultado do comando: `$ docker container ls -all`

 Qual a relevância de configurar “*volumes*” quando se pretende preparar um *container* para servir uma base de dados?