

Pokročilé programování na platformě Java:
Správa, tvorba a nasazení Java projektu:
Maven, Jenkins

Osnova

- **Projekt**

- Co obsahuje/ jak jej dělit
- Životní cyklus projektu
- Nasazení / instalace
- Nástroj pro správu projektu - Maven
- Nástroj pro automatizované nasazení - Jenkins

Co obsahuje projekt

- Kód
- Knihovny (závislosti)
- Konfiguraci
- Testy
- Ostatní
 - Dokumentace
 - Testovací sady dat
 - Statické soubory

Kód projektu - dělení

- Nedělit - jednoúčelové skripty
- Dělení do jmenných prostorů
- Dělení do podprojektů/modulů
- Dělení na úrovni služeb (několik propojených projektů na úrovni API)
 - Servisně orientované architektury - SOA
 - Mikro služby (VMS, Docker)
- Dělení na úrovni skupin služeb
 - Google Kubernetes (Pods)

Nijak nedělit kód

- Jednoúčelové programy (Perl, Python, Golang, Java)
 - Periodická úloha/skript (smaž staré soubory)
- Jednorázové načtení/uložení dat
- Testování konceptu
 - Online (např. <http://www.tutorialspoint.com/codingground.htm>)
 - Offline
 - Pro skriptovací jazyky např. vlastní skript
 - „Náčrtek“ (IDEA) – na cvičení
 - Možnost rychle si vyzkoušet kód
 - Bez nutnosti kompilovat celý program

Jmenné prostory

- Na souborovém systému zpravidla jako složky
- Komunikační prostředek
 - `cz.tul.crypt`
 - `cz.tul.utils`
 - `cz.tul.coordinator`
 - `cz.tul.coordinator.impl`
- Pojmenování vznikne „obrácením“ URL schématu
`utils.fm.tul.cz` -> `cz.tul.fm.utils`

Jmenné prostory výhody

- Řešení kolize pojmenování
 - `java.io.File`
 - `cz.tul.fm.File`
- Dělení na logické komponenty
 - `cz.tul.utils`
 - `cz.tul.utils.db`
- Konvence nad konfigurací
 - Automatické vyhledání dle očekávaného umístění (např. PHP)
 - `IndexController` -> `/controllers/index`
- Kontrola nad přístupem (`package private`)
 - Jasná definice externího API -> jednodušší změny uvnitř prostoru

Jmenné prostory nevýhody

- Prakticky žádné
 - Pokud použijeme zdravý rozum
 - `cz.tul.db.mongodb.MyLittlePony` ?
- Standard pro všechny větší projekty

Dělení na moduly

- Rozdělení podle hlavních částí projektu
- Počet
 - Jeden
 - Cvičení
 - [MySQL Java konektor](#)
 - Dva
 - Cvičení
 - Obvyklé např. pro webové aplikace
 - Appserver (webové api)
 - Core (jádro aplikace)
 - Jednotky
 - Mikroslužby
 - [Logback](#)
 - Desítky
 - Mikroslužby
 - [Spring](#)

Dělení na moduly

- Kdy už je to modul
 - Má (reálný !) přínos samostatně
 - Pro uživatele (Spring – Core, Data, Integration)
 - Pro kolegy (utility – [Apache Commons](#))
 - Více jazyků (polygot projekt)
 - Java API
 - Python API
 - Když se tým dohodne 😊
- Pravidlo package << modul
- Verzování
 - Společný repozitář
 - Per modul

Moduly - výhody

- Logické oddělení
 - Správa
 - Verzování
 - Závislostí
 - Výstup (release management)
 - Vývoj
 - Modul per tým
 - Vlastní rychlost vývoje
 - Kompilace
 - Testování

Moduly - nevýhody

- Závislosti
 - Před kompilací A potřebuji B, to zas potřebuje C, D, Z
- Nekompatibilita
 - A funguje s B verze 0.3.0 a 0.3.9 ale nefunguje s 0.3.1-8
- O starost navíc
 - Pokud nepotřebujeme – nevidíme jasný přínos
 - Pain to Gain
 - Pro většinu menších projektů stačí 1 modul

Čím více zvětšujeme nezávislost
jednotlivých komponent, tím více vzniká
i problémů s jejich následnou
integrací a údržbou

Vždy je dobré stanovit určitou úroveň
a pevně ji v týmu dodržovat

Dělení na služby

- Služba >> modul >> package
- Služba
 - Ucelená funkcionality poskytující aplikační rozhraní
 - Obvykle ve formě API (REST, RPC, SOAP)
- Předpověď počasí
 - Vyhledání místa (město -> ID)
 - Aktuální data
 - Dlouhodobá data, agregace
 - UI

Služby - výhody

- Dělení služby na úrovní týmů
- Škálovatelné – např. více instancí služby
- Závislosti jen na úrovni API
- Umožňuje aktualizaci za běhu (při duplikovaných službách)
 - Bez krátkodobých nedostupností (HA služby)

Služby - nevýhody

- Výsledkem je distribuovaný systém
 - Problematika na zvláštní předmět
- Jak se o sobě služby dozví
 - Registr služeb - Service discovery
 - Apache Zookeeper, Consul
- Latence (namísto volání metody - HTTP požadavek)
- Zabezpečení, rozložení zátěže, výpadky sítě ...

Hledání ideálního místa

- Obecně postupujeme směrem k většímu oddělení
- Při dobře vytvořené architektuře lze i během vývoje měnit úroveň rozdělení
- Neřešme případ 100 000 uživatelů, když nemáme ani jednoho
- Běžný server zvládne 10 000+ požadavků za sekundu
- <http://www.techempower.com/benchmarks>

Závislosti projektu - knihovny

- Potřeba centralizované správy
 - Dostupnost
 - Verzování (0.2.3beta, 20151013-0602, 0.2M1)
 - Kompatibilita (0.2.x – 0.3.7)
 - Distribuce
 - Strom závislostí
- Dependency hell
 - Lib A
 - logging 1.2
 - Lib B
 - logging 0.6

Závislosti projektu - prostředí

- Verze Java
- Operační systém
- Databáze
- Ostatní aplikace
- Řešení
 - Virtuální stroje (VirtualBox, VmWare)
 - Kloudové infrastruktury (AWS, Azure, RackSpace, Bluemix)
 - Kontejnerizace (Docker, Racket)
 - Neměnná infrastruktura popsaná kódem (Ansible, Puppet, Chef ...)
- Budeme se v tomto předmětu zabývat pouze okrajově
 - Vagrant + Ansible + VirtualBox - na cvičení 5

Konfigurace projektu

- Umístění
 - Do kódu
 - Externě (soubory, DB, systémové proměnné, ...)
- Formát
 - Obecně podporovaný
 - XML, JSON, YAML (superset JSONu - Ansible)
- Profily
 - Testování
 - Produkce
 - Vývoj

Dobré praktiky

- Veškerá nastavení by mělo být konfigurovatelné
 - Aktivní profil
 - Adresy, porty
 - Timeouty spojení
 - Logování, ...
- Možnost přepsat nastavení pomocí proměnných prostředí
 - I např. umístění konfiguračního souboru
- Dělení nastavení do jmenných prostorů (db.mysql.port)
- **Konzistence**
 - `app.db.connection.read_timeout=10 // seconds`
 - `app.db.connection.writeTimeout=10000 // millis`

Testování

- Automatické ověření funkcionality
- Dokumentace funkcionality kódu
- Principy
 - Každý opravený bug -> test (Zabránění znovu zanesení)
 - Úprava kódu -> spuštění testů (Kontrola, že vše stále funguje)
- Funkční kód
 - Naprogramovaný, **zdokumentovaný a otestovaný**
- TDD – Test Driven Development

Typy testů

- Jednotkové (Unit)
 - Testujeme konkrétní funkcionalitu/metodu
- Integrační
 - Testuje integraci s externím systémem (Google API, DB)
- End 2 End
 - Testuje celkové chování/propojení komponent
- UI test
 - Testuje chování frontendu (JS, HTML, CSS)
 - Kliknu na tlačítko a objeví se reklama
- A další ... zaměříme se především na jednotkové a integrační

Sestavení

- Pořadí kompilace
 - A -> B -> C
- Volitelné kroky
 - Spuštění testů
 - Generování dokumentace
 - Spuštění skriptů
- Výstup a formát kompilace
 - JAR, WAR, TAR DEB
- Sdílení výstupu
 - Repozitář

Automatizace správy projektu

- Ručně
 - Běžné před 40 lety
 - Shell, Make - 1977
- IDE
- Nástroje pro automatizaci správy projektu
- CI server (průběžná integrace)
- Kontejnerizace

Nástroje pro JVM

- Apache Ant (XML)
 - První řešení (2000)
- Apache Ivy (XML)
 - Správa závislostí
 - Používán pro Ant a SBT
- Apache Maven (XML)
 - Řešení problémů Ant
 - Konvence před konfigurací
- Gradle (Groovy)
 - Řešení problémů Maven
- Simple Build Tool – SBT (Scala)

Základy práce s nástrojem Maven

Maven

- Open-source systém pro správu životního cyklu a závislostí projektu
- Klíčové rysy:
 - ***Project object model (POM)***
 - Popis projektu v XML
 - ***Konvence nad konfigurací***
 - Minimalistická konfigurace projektu využívající co nejvíce defaultní nastavení
 - Šablony projektů (archetypes)
 - ***Management závislostí***
 - Je možné definovat závislosti projektu na dalších projektech (knihovnách)
 - Maven je může také kompilovat
 - ***Repositáře***
 - Závislosti mohou být nahrávány z lokálního soub. systému, internetu, veřejných repositářů nebo centrálního Maven repositáře
 - ***Rozšiřitelnost pomocí plug-inů (modulů)***
 - Jádro Mavenu není rozsáhlé, většina funkcionalit je řešena pomocí modulů (lze dopisovat vlastní)
 - Např. kompilace Java kódu, testování...

Maven zdroje

- Maven
 - <http://maven.apache.org>
- Online kniha
 - <https://books.sonatype.com/mvnref-book/reference/index.html>

Project Object Model (POM)

- Soubor definující projekt (pom.xml)
- Obsahuje
 - Popisnou část
 - Identifikátory projektu = název, skupina, verze
 - Definici závislostí
 - Proměnné
 - Pluginy
 - ...
- Maven podporuje při práci s POM dědičnost
 - Vlastnosti daného POM mohou být odvozeny od jeho rodiče (PARENT POM)
 - Efektivní POM – výsledek kombinace POM

Super POM

- Defaultní POM pro danou verzi Mavenu
- Ze Super POM jsou na pozadí děděna některá (defaultní) nastavení
 - Závislosti, plug-in listy, nastavení pluginů, repositáře ...
 - POM projektu může být o tyto definice menší
- Příklady defaultní hodnot:
 - Repositories: <http://repo.maven.apache.org/maven2>
 - Packaging type: JAR
- Ukázka
<https://maven.apache.org/ref/3.0.4/maven-model-builder/super-pom.html>

Dědičnost projektů

- Dědičnost díky objektovému modelu neplatí jen pro Super POM
- Lez ji využít i pro správu více projektů najednou
- Projekt může dědit od svého rodiče
 - např. závislosti, konfiguraci pluginů, ...
- Dědění od nějakého projektu se zajistí přidáním tagu parent
 - obsahuje odkaz na daný projekt + volitelně cestu

```
<parent>  
  <groupId>com.mycompany.app</groupId>  
  <artifactId>my-parent</artifactId>  
  <version>2.0</version>  
  <relativePath>../my-parent</relativePath>  
</parent>
```


Modulární projekty – agregace projektů

- Maven umožňuje vytvářet i projekty s více moduly
 - Každý modul je ve skutečnosti v podstatě samostatný projekt
- V rámci hlavního POM se definuje seznam modulů
 - POM každého modulu je implicitně hledán v odpovídajícím podadresáři
- Dílčí modul může sdílet s hlavním závislosti, atributy, ...
 - Viz hlavní POM ukázkových aplikací pro přednášky z PPJ
- Agregaci modulů lze kombinovat s dědičností

```
<modules>  
  <module>my-project</module>  
  <module>another-project</module>  
</modules>
```

Minimální POM

- `modelVersion` – verze POM použitá pro popis
- `groupId` – ID skupiny, do které projekt spadá
- `artifactId` – ID samotného projektu

- `version` – verze projektu, používá se konvence:

- `<major version>.<minor version>.<incremental version>-<qualifier>`

- kvalifikátory

- SNAPSHOT verze směřující k RELEASE (během vývoje, např. každou noc)
- RELEASE verze pro nasazení

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.mycompany.app</groupId>
  <artifactId>my-app</artifactId>
  <version>1.0.2-SNAPSHOT</version>
</project>
```

Maven artefakt (artifact)

- Maven artefakt je výsledek sestavení projektu umístěný v repozitáři
 - Jde např. výsledný JAR nebo WAR soubor
- Artefakt je jednoznačně identifikovatelný jménem ve tvaru
 - `<groupId>:<artifactId>:<version>`
 - Každá závislost projektu (= artefakt) je definováno tímto identifikátorem
 - tj. pomocí tagů `<groupId>`, `<artifactId>` a `<version>`
- Každý plug-in je také Artefakt
 - Např. plug-in pro kompilaci – viz. definice Super POMu
- Projekt -> kompilace (JAR, WAR) -> artefakt

Závislost = artefakt

- Reference na artefakt v repozitáři
- Specifikace verze
 - 4.1 – **Preferovaná** verze
 - [3.8,4.0) - 3.8+ (včetně) a méně než 4.0
 - [4.1] – **Požadovaná – striktní** verze
- Transitivní závislosti
 - Artefakt A závisí na B a to závisí na C -> C je tranzitivní závislost pro A

```
<properties>
    <junit.version>4.12</junit.version>
</properties>

<dependencies>
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>${junit.version}</version>
    </dependency>
</dependencies>
```

Konflikty závislostí

- Project A - mylib:1.0
- Project B - mylib:2.0
- Knihovna nedostupná v repozitáři nebo nevyhovující licence
- Řešení
 - Automatické - nearest win
 - A -> B -> C -> D 2.0 a A -> E -> D 1.0, pak se použije D 1.0
 - Sjednocení na společnou verzi
 - Přímá definice závislosti (v *dependencyManagement* sekci)
 - Popř. definice v *dependencies* sekci (bude mít přednost = úroveň 1)
 - Exkluze závislosti = vynechání závislosti
 - Lze například vynechat nižší verzi knihovny

Centralizované řešení závislostí – možnost 1

- Parent dependency management

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.sonatype.mavenbook</groupId>
  <artifactId>a-parent</artifactId>
  <version>1.0.0</version>
  ...
  <dependencyManagement>
    <dependencies>
      <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>5.1.2</version>
        <scope>runtime</scope>
      </dependency>
      ...
    </dependencies>
  </dependencyManagement>
</project>
```

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.sonatype.mavenbook</groupId>
    <artifactId>a-parent</artifactId>
    <version>1.0.0</version>
  </parent>
  <artifactId>project-a</artifactId>
  ...
  <dependencies>
    <dependency>
      <groupId>mysql</groupId>
      <artifactId>mysql-connector-java</artifactId>
    </dependency>
  </dependencies>
</project>
```

- Pouze jeden parent
 - Verze se uvádí pouze v parent projektu

Centralizované řešení závislostí – možnost 2

- POM import

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.sonatype.mavenbook</groupId>
  <artifactId>a-parent</artifactId>
  <version>1.0.0</version>
  ...
  <dependencyManagement>
    <dependencies>
      <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>5.1.2</version>
        <scope>runtime</scope>
      </dependency>
      ...
    </dependencies>
  </dependencyManagement>
```

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.sonatype.mavenbook</groupId>
      <artifactId>a-parent</artifactId>
      <version>1.0.0</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
```

Rozsah platnosti závislostí (scope)

- Compile
 - Defaultní (definovaná, pokud není žádná jiná)
 - Závislost se propaguje se i do všech projektů, které na daném projektu závisí
 - Závislost je dostupná na všech classpath projektu a ve výsledku kompilace (JAR, WAR)
- Provided
 - Závislost je poskytnuta cílovým běhovým prostředím (např. web. container u web. aplikace)
 - Nepropaguje se
 - Je dostupná na CP compile a test, nikoli ve výsledku kompilace
- Runtime
 - Závislost není nutná pro kompilace, ale pouze pro běh (test)
 - Je dostupná na CP runtime a test
- Test
- System (provided pro nativní závislosti, nedoporučuje se)
- Import (pouze pro import definice)

Repositáře závislostí

- Lokální – staženy všechny odkazované závislosti
- Vzdálené
 - maven central - <http://repo.maven.apache.org/maven2/>
 - Definované v super POM
- Artefakty uložené ve stromové struktuře podle namespace
- Search <http://mvnrepository.com/>

Proces sestavení (build)

- Ústřední a hlavní proces celého Mavenu
- Je definován v rámci životních cyklů (build lifecycles)
 - Default (Build) – sestavení projektu
 - Clean – vyčištění projektu
 - Site – generování dokumentace
- Každý cyklus je definován jako pevná sekvence fází $F1...fN$
 - Každou fázi lze spustit samostatně a ručně
 - Pak jsou ale vykonány všechny fáze, které jsou v daném cyklu nadefinovány jako předcházející
 - viz práce s nástrojem Maven dále

Hlavní fáze cyklu default (build)

- Validate
 - Validace dostupnosti všech potřebných dat a informací
- Compile
 - Zkompilování zdrojových kódů
- Test
 - Otestování zkompilevaného kódu (bez nutnosti zabalení nebo umístění do repositáře)
- Package
 - Zabalení zkompilevaného kódu k distribuci (JAR, WAR)
- Integration-test
 - Nasazení balíčku do prostředí pro otestování (pokud je to nutné)
- Install
 - Umístění balíčku do lokálního repositáře (může být použit jako závislost v dalších projektech)
- Deploy
 - Umístění balíčku do vzdáleného repositáře (pro další sdílení)
- **Zvolání fáze deploy (mvn deploy) má za následek také spuštění všech předchozích fází.**
- Ostatní cykly mají fází výrazně méně (např. clean má pouze pre-clean, clean a post clean)
- <https://books.sonatype.com/mvnref-book/reference/lifecycle-sect-structure.html>

Práce s nástrojem Maven pomocí IDE

- Při vytváření projektu se zadá projekt typu Maven
 - Lze vybrat příslušnou šablonu (archetype)
 - Projekt má pak příslušnou strukturu – viz dále
 - pom.xml je umístěn v kořenovém adresáři projektu
 - Z kontextového menu aplikace lze explicitně volat např. jednotlivé fáze či životní cykly (např. clean)
 - IDE dále využívá Maven automaticky na pozadí
 - např. pokud projekt kompilujeme

Práce s nástrojem Maven pomocí přík. řádky

- Po instalaci lze volat jednotlivé příkazy:
 - `mvn --help`
 - `mvn --version`
 - V kořenovém adresáři projektu (obsahuje `pom.xml`) lze pracovat s daným projektem:
 - `mvn clean`
 - Spustí životní cyklus `clean` (postupně všechny jeho fáze)
 - `mvn clean deploy`
 - Spustí životní cyklus `clean` (postupně všechny jeho fáze) a poté všechny fáze životního cyklu `default (build)` až do `deploy` (včetně)
 - `mvn clean install`
 - Typický povel = vyčištění + ... + kompilace + umístění výsledku do repozitáře

Goals (cíle)

- Životní cykly jsou prováděny jako sekvence fází
- Každá fáze sama je prováděna jako sekvence operací
 - Každá operace má svůj cíl (goal) a je prováděna pomocí příslušného pluginu
 - Operace vedoucí k danému cíli může být prováděna v rámci více fází
- Jednotlivé operace (cíle) lze volat z příkazové řádky:
 - `mvn clean dependency:copy-dependencies package`
 - Proveďte se
 - 1) životní cyklus clean
 - 2) operace `dependency:copy-dependencies` (pomocí pluginu `dependency`)
 - 3) fáze `package` a všechny předchozí fáze z cyklu `build`

Plugins

- Realizují operace (cíle), z kterých se skládají fáze životních cyklů
 - Maven je ve skutečnosti framework spouštějící jednotlivé pluginy !
- Existují tři hlavní typy pluginů
 - Pluginy odpovídající nějaké fázi některého životního procesu
 - tvoří jádro Mavenu - plugin clean, plugin compiler, ...
 - Pluginy zajišťující vytvoření cílového balíčku podle šablony (archetype) projektu
 - jar, war, ...
 - Pluginy zajišťují reportování
 - Javadoc,

Struktura Maven projektu

/src/main/java

cz.tul.ppj.MyApp

/src/main/resources

app.properties

/src/test/java

cz.tul.ppj.MyAppTest

/src/test/resources

app-test.properties

<https://maven.apache.org/guides/introduction/introduction-to-the-standard-directory-layout.html>

Šablony (archetypes)

- Umožňují
 - Standardizovat (definici) popis projektů určitého typu
 - Usnadňují vytváření projektů stejného typu

maven-archetype-j2ee-simple	An archetype to generate a simplified sample J2EE application.
maven-archetype-webapp	An archetype to generate a sample Maven Webapp project
maven-archetype-plugin	An archetype to generate a sample Maven plugin.

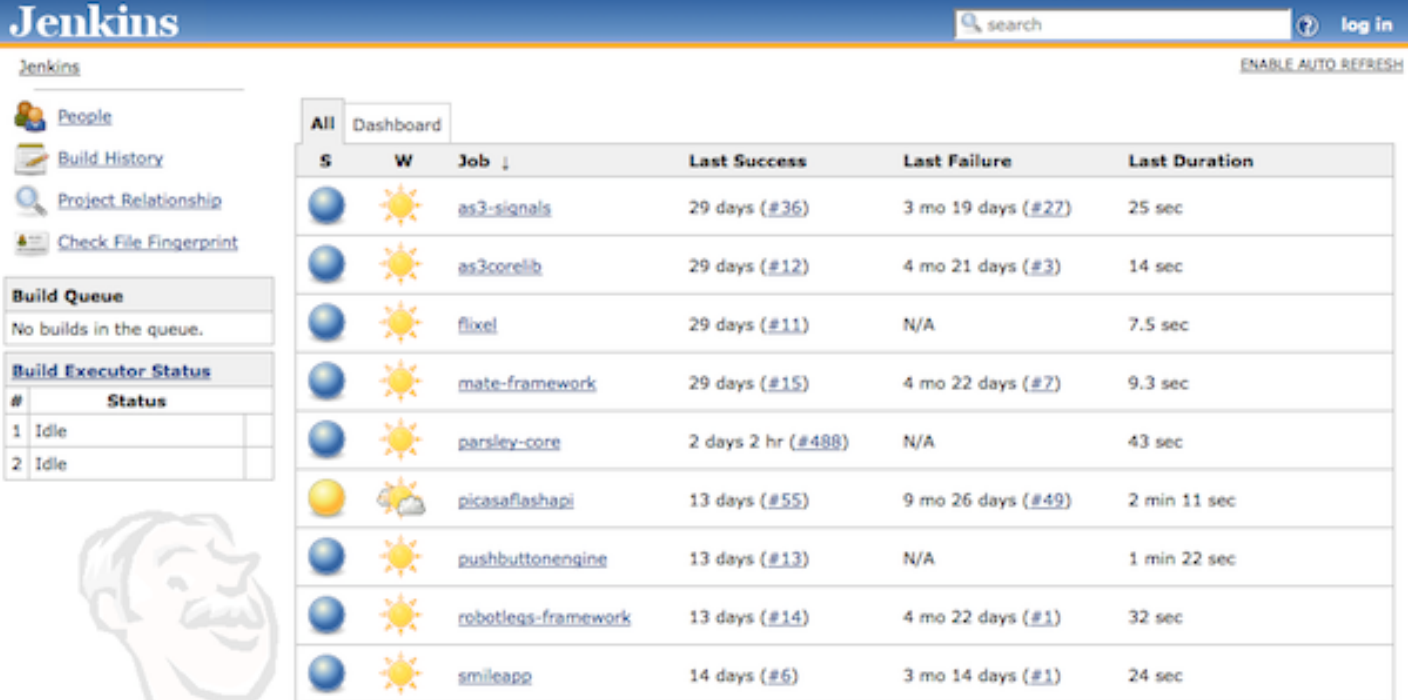
CI Server

Fáze integrace

- Průběžná integrace (CI)
 - Minimalizace doby od napsání kódu k jeho otestování a zpřístupnění v repu
- Průběžné dodání (CDel) – základem je CI
 - Rozšíření o prostředí (integrační testy)
 - Prostředí
 - Code Review
 - User Acceptance Testing (UAT)
 - Staging (před produkcí)
- Průběžná instalace (Cdep) – základem je Cdel
 - Umožňuje dodání přímo do produkce

CI Server – Jenkins (https://jenkins-ci.org)

- Kroky
 - Notifikace
 - Stažení verze z repositáře
 - Spuštění Maven cíle
 - Práce s výsledkem
- Spravuje
 - Závislosti mezi projekty
 - Výstup z konzole
 - Historii sestavení



The image shows the Jenkins dashboard interface. On the left, there is a sidebar with navigation links: People, Build History, Project Relationship, and Check File Fingerprint. Below these are sections for 'Build Queue' (showing 'No builds in the queue.') and 'Build Executor Status' (showing two executors in 'Idle' status). The main area displays a table of jobs. The table has columns for status (S for success, W for warning), job name, last success time, last failure time, and last duration. A faint watermark of a person's face is visible in the bottom left corner of the dashboard area.

S	W	Job	Last Success	Last Failure	Last Duration
●	☀	as3-signals	29 days (#36)	3 mo 19 days (#27)	25 sec
●	☀	as3corelib	29 days (#12)	4 mo 21 days (#3)	14 sec
●	☀	fixel	29 days (#11)	N/A	7.5 sec
●	☀	mate-framework	29 days (#15)	4 mo 22 days (#7)	9.3 sec
●	☀	parsley-core	2 days 2 hr (#488)	N/A	43 sec
●	☀	picasafashapi	13 days (#55)	9 mo 26 days (#49)	2 min 11 sec
●	☀	pushbuttonengine	13 days (#13)	N/A	1 min 22 sec
●	☀	robotlegs-framework	13 days (#14)	4 mo 22 days (#1)	32 sec
●	☀	smileapp	14 days (#6)	3 mo 14 days (#1)	24 sec