VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

INTEGRACE SERVERU UNDERTOW SE SYSTÉMEM JENKINS CI

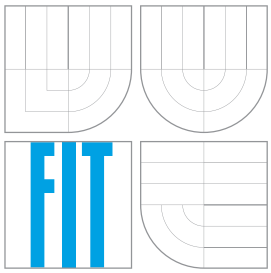
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. JAKUB BARTEČEK

BRNO 2014

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

INTEGRACE SERVERU UNDERTOW SE SYSTÉMEM JENKINS CI

INTEGRATION OF JENKINS CI WITH UNDERTOW

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. JAKUB BARTEČEK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. PETR MÜLLER

BRNO 2014

Abstrakt

Tato diplomová práce se zabývá nahrazením servlet kontejneru v systému Jenkins CI serverem Undertow. V práci jsou popsány obecné informace o programech, které se této problematiky týkají a je analyzován současný stav servlet kontejneru v Jenkins CI. Výstupem této práce je vytvořený nový servlet kontejner pro Jenkins CI. Obě varianty Jenkins CI byly testovány z hlediska výkonu a nová varianta prokázala lepší výsledky.

Abstract

This master's thesis deals with replacement of the servlet container in Jenkins CI with server Undertow. In the thesis, general information about programs, which are related to this topic, are described and current state of the servlet container is analyzed. The result is newly created servlet container for Jenkins CI. Both versions of the Jenkins CI were performance tested and the new version gave better results.

Klíčová slova

Jenkins, Undertow, servlet kontejner, integrace, kontinuální integrace, Winstone, Jetty, Java

Keywords

Jenkins, Undertow, servlet container, integration, continuous integration, Winstone, Jetty, Java

Citace

Jakub Barteček: Integrace serveru Undertow se systémem Jenkins CI, diplomová práce, Brno, FIT VUT v Brně, 2014

Integrace serveru Undertow se systémem Jenkins CI

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana inženýra Petra Müllera. Další informace mi poskytl pan doktor Vojtěch Juránek, který je zaměstnancem firmy Red Hat.

.....
Jakub Barteček
27. května 2014

Poděkování

Děkuji panu inženýru Petru Müllerovi za vedení mé diplomové práce a panu doktoru Vojtěchu Juránkovi za odborné konzultace týkající se zpravovávané problematiky.

Obsah

Kapitola 1

Úvod

Tato diplomová práce se zabývá vylepšením serveru Jenkins CI, který je v praxi využíván pro potřeby kontinuální integrace softwaru. Vylepšení se týká především webového serveru a servlet kontejneru, který je v Jenkins CI integrován. V současném stavu tyto funkce vykonává kombinace serverů Winstone a Jetty.

Server Winstone je již neudržovaný a zastaralý nástroj, proto byl z velké části nahrazen serverem Jetty, který potřebnou funkcionalitu poskytuje. Server Jetty je poměrně komplexní projekt a nabízí mnoho funkcí, ale na druhou stranu jeho rozsah neumožňuje dosažení maximální rychlosti.

Nedávno vznikl nový webový server Undertow, který si klade za cíl být co nejjednodušší a nejrychlejší a mohl by být potenciálně přínosný pro Jenkins CI. Vzhledem k tomu, že je tento server nový a je sponzorován firmou Red Hat, lze předpokládat, že jeho vývoj bude nadále pokračovat a nebude zastarávat.

Cílem této práce je nahradit server Jetty a případně i server Winstone serverem Undertow a integrovat jej se serverem Jenkins CI. Při integraci je kladen důraz na snahu zachovat zpětnou kompatibilitu s původním řešením.

V rámci této diplomové práce jsou nejprve rozebírány potřebné informace týkající se jednotlivých nástrojů a plánované integrace. Poté je detailně analyzována architektura Jenkins CI a způsob jeho integrace se servery Winstone a Jetty. Je diskutována varianta nahrazení pouze serveru Jetty a varianta nahrazení obou serverů pomocí Undertow. Z provedené analýzy byla zvolena jedna varianta, která byla vybrána pro integraci.

Následně je popsán způsob provedení navržené integrace serveru Undertow do Jenkins CI a poslední část této práce je věnována zhodnocení výkonu původní a upravené verze serveru Jenkins CI. Výkon obou variant je porovnáván ve třech zvolených testovaných případech. Na základě získaných informací o výkonnosti a výsledku celé integrace je v závěru této práce zhodnocen její přínos pro komunitu projektu Jenkins CI.

Diplomová práce navazuje na předcházející semestrální projekt, v rámci něhož byla provedena studie všech souvisejících nástrojů (kapitola 2), analýza architektury Jenkins CI a návrh způsobu integrace (kapitola 3). Nicméně uvedené kapitoly byly následně upravovány především z důvodu změn, které nastaly v průběhu vývoje serveru Undertow.

Kapitola 2

Jenkins CI a související nástroje

Tato kapitola se zaměřuje na teoretické základy práce, které je nutné nebo vhodné znát pro pochopení zpracovávané problematiky. Je zde detailněji popsán systém Jenkins CI (kapitola 2.1) ke kterému se tato práce přímo váže. S ním je spojeno seznámení se servery Winstone (kapitola 2.5) a Jetty (kapitola 2.4), jejichž kombinace je současně v Jenkins CI integrována. Po popisu těchto serverů je rozebírán průběh jejich využití v servlet kontejneru Jenkins CI a průběh jeho vývoje (kapitola 2.6). V kapitole 2.2 jsou vysvětleny často zde používané pojmy *servlet kontejner* a webový server.

Větší důraz je dále věnován serveru Undertow (kapitola 2.7), který byl vybrán jako nový webový server pro Jenkins CI. V tomto případě je provedena hlubší studie tohoto nástroje, aby na jejím základě bylo možné pochopit a provést samotnou integraci s Jenkins CI, která je jádrem této práce.

Uvedené informace mají spíše informativní charakter, aby poskytly ucelený úvod do zkoumané problematiky. Jsou zaměřeny především na informace týkající se samotné integrace. Pro případné získání detailnějších informací jsou uvedeny patřičné zdroje, kde je lze nalézt.

2.1 Jenkins CI

Jenkins CI je komunitní open source nástroj pro kontinuální integraci softwaru, který je vyvíjen pod svobodnou licencí MIT[1] [2]. Je velmi populární a využíván malými i velkými firmami jako je například firma Red Hat, kde tento program běží na stovkách serverů. Původní název tohoto projektu je Hudson[2]. Když se jeho vývoje ujala firma Oracle, tak se projekt rozštěpil a vznikla jeho komunitní verze, kterou je projekt Jenkins CI. Přesto se v některých částech tohoto projektu stále objevuje název Hudson, ale jedná se pouze o pozůstatek z původního projektu.

Zkratka CI je z anglického spojení *continuous integration*, což lze do češtiny přeložit jako kontinuální nebo průběžná integrace. Krátké seznámení s touto metodologií je v následující kapitole.

Informace v této kapitole byly čerpány především z knihy [18], kde lze nalézt další informace o serveru Jenkins CI, a také z webové stránky projektu [21].

[1] Text licence MIT: `http://opensource.org/licenses/MIT`

[2] Webové stránky projektu Hudson: `http://hudson-ci.org/`

2.1.1 Kontinuální integrace a využití Jenkins CI

V minulosti byla integrace programu do výsledného produktu velmi náročným procesem a často ztraceným časem. S vydáním každé verze programu se musel postup probíhající před vydáním produktu opakovat a pro tým vývojářů to prakticky znamenalo zdržení. Pokud se v tomto procesu odhalil nějaký problém (což bylo běžné), tak jeho řešení bylo z důvodu nedostatku času a jeho pozdního objevení mnohem problematičtější než kdyby byl tento problém odhalen dříve.

Kontinuální integrace je moderní přístup k vývoji softwaru, který mění způsob přemýšlení nad celým procesem vývoje a snaží se předcházet problémům popsaným výše a především ušetřit čas. V tomto přístupu je využíván nějaký kvalitní nástroj, který automatizovaně provádí specifikované kroky provázející integraci softwaru a jeho vydání.

Jedním z nástrojů poskytujících podporu pro kontinuální integraci při vývoji softwaru je server Jenkins CI.

Základními možnostmi, které umožňuje Jenkins CI nakonfigurovat, jsou:

- Spouštění integračních a jednotkových testů v přesně definovaném čase (např. v noci, kdy jsou servery méně vytížené)
- Spouštění integračních a jednotkových testů při změně ve verzovacím systému. Jenkins CI dokáže zaznamenat změnu v repozitáři, stáhnout si změny a spustit testování
- Shromažďování a vyhodnocování metrik vývoje softwaru
- Spouštění akceptačních testů
- Informování e-mailem o testech, které skončily chybou
- Automatické nahrávání nové verze produktu na server

Uživatelé mohou kdykoliv přidat využití libovolné funkcionality systému a neopakovat stále stejné kroky.

2.1.2 Způsoby použití aplikace

Celý program je napsán v jazyce Java a je tedy plně přenositelný mezi platformami. Architektura je navržena tak, aby byla lehce rozšiřitelná pomocí tzv. *pluginů*, kterých je pro něj vytvořené velké množství.

Jenkins CI je určen pro běh na serveru a je dostupný přes webové rozhraní (může být samozřejmě také spuštěn na libovolném osobním počítači). Komunikuje zejména pomocí protokolu HTTP nebo HTTPS, které jsou založeny na modelu *požadavek-odpověď* (angl. *request-response*). Pro svou činnost potřebuje servlet kontejner ve kterém samotná aplikace poběží. Tento pojem je blíže objasněn v kapitole 2.2.

Existují dvě možnosti jak spustit server Jenkins CI:

1. Může běžet na libovolném *Java EE* aplikačním serveru [1] jako jsou například servery JBoss[3] anebo GlassFish[4]. Jenkins CI se standardně nasadí na server (dle zvyklostí

[3] Více informací o serveru JBoss: `http://www.jboss.org/jbossas/`
[4] Více informací o serveru GlassFish: `http://www.oracle.com/technetwork/middleware/glassfish/`

konkrétního serveru) a poté je s ním možné pracovat. V tomto případě veškerou nízkoúrovňovou komunikaci pomocí síťových protokolů i práci servlet kontejneru zajišťuje aplikační server.

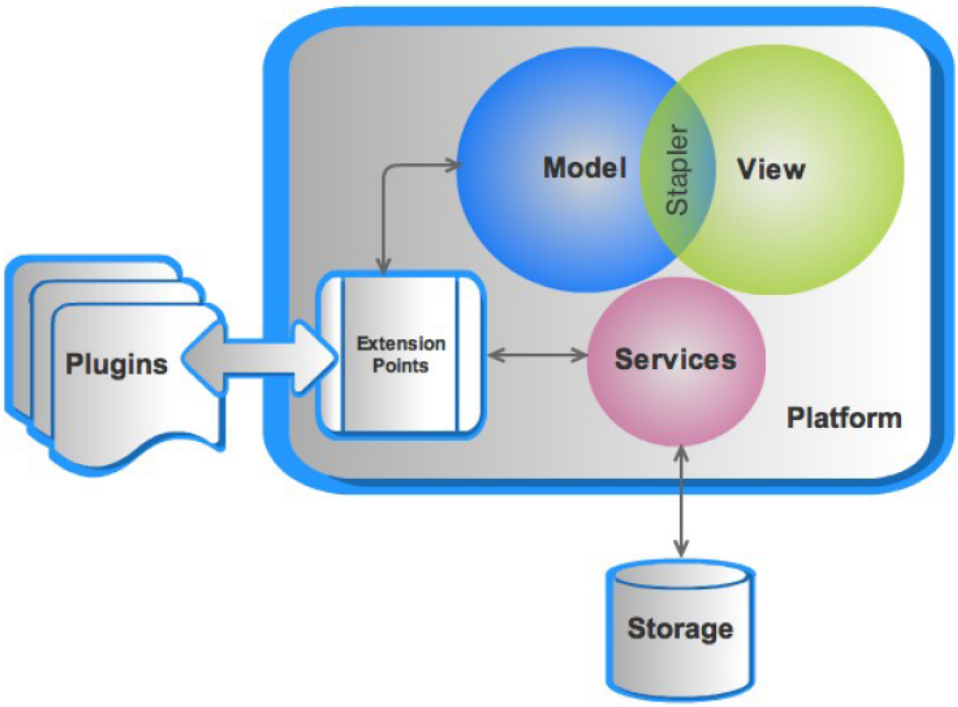
2. Pokud nechceme nebo nemůžeme spouštět Jenkins CI na aplikačním serveru, tak jej lze spustit přímo z vytvořeného `.war` archivu (překladem se zabývá kapitola 2.1.4). V tomto případě se o práci servlet kontejneru i webového serveru komunikujícího jedním ze standardních síťových protokolů stará kombinace nástrojů Winstone a Jetty, které jsou přímo integrovány do serveru Jenkins CI.

   Nahrazení těchto dvou nástrojů (nebo pouze serveru Jetty) a zajištění vykonávání této činnosti je hlavním cílem této práce. Záměrem je tedy nahradit server Jetty a případně i server Winstone pomocí zvoleného nového serveru Undertow. Studie těchto nástrojů a jejich rozbor je předmětem následujících kapitol.

Pro tuto práci je důležitý druhý způsob používání Jenkins CI a proto další informace se budou přímo vázat k němu.

2.1.3 Architektura serveru

Architektura systému Jenkins CI je poměrně komplikovaná a pro tuto práci není nutné ji detailně celou znát. Bude popsána na vysoké úrovni abstrakce a zaměří se pouze na komponenty, které se přímo týkají této práce a budou dále v textu odkazovány nebo blíže rozebírány. Přehled architektury je na obrázku 2.1.



Obrázek 2.1: Přehled architektury serveru Jenkins CI [15]

Základem architektury je část *Model*, což jsou objekty, které obsahují stav a data aplikace. Každý model je přímo navázán na konkrétní URL adresu s tím, že kořenovým modelem (dostupný pod URL „/“) je pevně daná instance s názvem `Hudson`. Data z objektů modelu jsou následně zobrazovány ve webovém rozhraní aplikace. Pro toto zobrazování je využita techonologie Jelly.

Velmi podstatnou součástí aplikace je prvek *Stapler*. Tento objekt provádí konkrétní propojování požadavků dle zadané URL s patřičnými objekty z modelu a spouštění vykonávání jejich metod. Stapler je jediným servletem, který je v aplikaci Jenkins CI zaveden do servlet kontejneru při spuštění aplikace (pojmy servlet a servlet kontejner jsou vysvětleny v kapitole 2.2.2). Povědomí o jeho činnosti je potřebné, protože s ním bude v této práci dále pracováno. Průběh vybírání patřičných metod pomocí této komponenty je přesně definován a lze jej najít v uživatelské příručce[5], ale není potřebné jej v této práci rozebírat.

Architektura serveru je přizpůsobena tak, aby byla snadno rozšiřitelná pomocí rozšiřujících modulů (angl. *plugins*). Popis technologie Jelly i tvorba rozšiřujících modulů je nad rámec této práce a lze tyto informace najít v odkazované literatuře.

Informace v této kapitole byly čerpány z tohoto dokumentu [15] a z webových stránek projektu Stapler [20].

2.1.4 Základy práce s Jenkins CI

Pro přeložení a spuštění aplikace je potřeba pracovat z příkazové řádky nebo provést instalaci nějakým dávkovým souborem (skriptem). Aplikaci je možné stáhnout připravenou přímo ze stránek projektu [21], ale pro tuto práci je potřeba pracovat s aplikací ze zdrojových souborů. Aktuální verze aplikace je dostupná na serveru GitHub[6].

Po stažení zdrojových souborů je potřeba provést překlad aplikace. Pro automatizovaný překlad se používá nástroj Maven, který je krátce popsán v kapitole 2.3.

Překlad aplikace bez spuštění jednotkových testů i integračních testů lze provést tímto příkazem:

```
mvn clean install -pl war -am -DskipTests
```

Po úspěšném překladu aplikace vznikne v adresáři `./war/target/` archiv `jenkins.war`, který obsahuje celou přeloženou webovou aplikaci včetně nástrojů na kterých závisí. Z tohoto archivu je možné aplikaci přímo spustit pomocí příkazu:

```
java -jar jenkins.war
```

Chování aplikace lze upravit nastavením různých parametrů při spuštění programu, které lze vypsat pomocí přidání parametru `--help`.

Pro práci se spuštěnou instancí aplikace se používá především webové rozhraní. Standardně aplikace komunikuje pomocí protokolu *HTTP* na portu 8080. Je tedy možné se na lokálním počítači připojit do aplikace zadáním URL adresy `http://localhost:8080` do webového prohlížeče.

Pokud se aplikaci povede spustit, tak je již možné libovolně pracovat s aplikací pouze z webového prohlížeče. Detaily práce s aplikací Jenkins CI lze nalézt v této knize [18], ale tyto informace jsou již nad rámec této publikace.

[5] Způsob zpracovávání požadavků komponentou Stapler: `http://stapler.kohsuke.org/reference.html`

[6] Adresa aktuální verze Jekins CI: `www.github.com/jenkinsci`

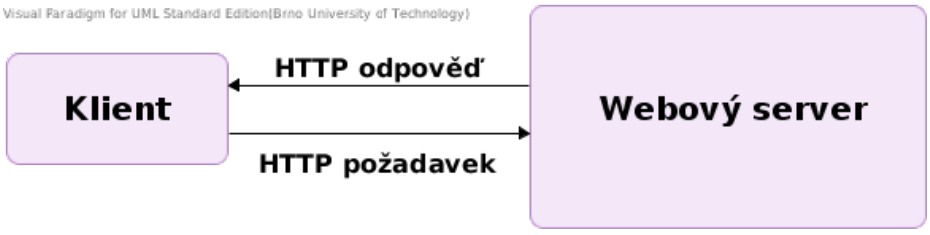
2.2 Webový server a servlet kontejner

V této kapitole jsou vysvětleny pojmy webový server a servlet kontejner, které se na mnoha místech této práce objevují. Porozumění těmto pojmům je důležité, protože bez jejich znalosti by následující kapitoly byly obtížněji pochopitelné. Informace zde uvedené byly čerpány z článku [19].

2.2.1 Webový server

Webový server je program, který zprostředkovává komunikaci přes síť s klienty, kteří se k němu připojí a požadují po něm nějaká data. Tato komunikace běžně probíhá pomocí protokolu HTTP nebo jeho šifrované verze HTTPS, které jsou založeny na modelu *požadavek-odpověď* (angl. *request-response*). Model této komunikace je zachycen na obrázku 2.2.

Typickým způsobem komunikace webového serveru je, že klient pomocí URL adresy specifikuje požadavek na nějaká data a server mu v odpovědi tato data pošle. Pokud by neexistoval za webovým serverem nějaký další program, tak by webový server vždy odpovídal na stejný požadavek stále stejnou odpovědí.



Obrázek 2.2: Model komunikace webového serveru s klientem

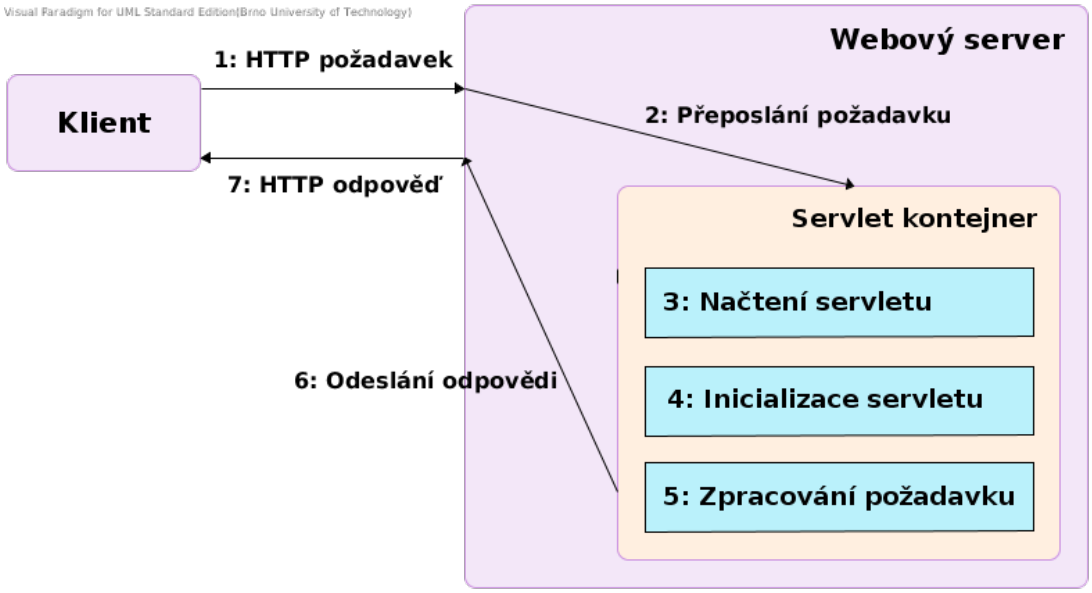
2.2.2 Servlet kontejner

Jelikož dostávat stále stejná statická data při stejném požadavku není dostatečná funkcionalita serveru, tak existují způsoby jak zajistit dynamickou práci s daty na serveru a tudíž i poskytovat měnící se odpovědi na stejný dotaz. Jedním ze způsobů, jak této funkce docílit, je využití tzv. *servletů*, které jsou navrženy pro programovací jazyk Java.

Servlet je standardní program (konkrétně jedna třída) napsaný v jazyce Java, který implementuje rozhraní `javax.servlet`. Implementace tohoto rozhraní ho zavazuje k definování několika metod, ale jinak se jedná o běžnou třídu, z které jsou při zpracovávání požadavků vytvářeny její instance. Po obdržení nějakého požadavku servlet provede zpracování vstupních dat a následně odešle patřičnou odpověď.

Základní myšlenkou servlet kontejneru je umožnit dynamicky vytvářet odpovědi (často webové stránky) pomocí vykonávání servletů. Samotný servlet kontejner je program, který poskytuje běhové prostředí pro vykonávání servletů. Zajišťuje jejich vytváření, vykonávání a odstraňování. Dále se také podílí na zpracovávání příchozích požadavků, které jsou mu předávány od webového serveru.

Popsaný způsob fungování servlet kontejneru je znázorněn na obrázku 2.3.



Obrázek 2.3: Ukázka způsobu činnosti servlet kontejneru a jeho spolupráce s webovým serverem

2.2.3 Struktura archivu webové aplikace

Kompletní popis architektury webových aplikací v jazyce Java by byl velmi rozsáhlý. Tato kapitola se zaměřuje pouze na strukturu archivu webové aplikace `.war` pro aplikace běžící v servlet kontejneru. Popis je zaměřen pouze na části, které jsou použity v Jenkins CI a jsou podstatné pro samotnou integraci serveru Undertow do Jenkins CI.

Archiv `.war` je standardní archiv zabalený metodou *ZIP* a základ jeho struktury je následující:

Příklad 2.2.1. Zjednodušená struktura archivu `.war`

```
webapp.war
    |-- WEB-INF/
        |-- lib/
            |-- *.jar
        |-- classes/
            |-- *.class
        |-- web.xml
    |-- <ostatní>
```

Pro konfiguraci aplikace je nejdůležitější adresář WEB-INF a především tyto jeho položky:

- Adresář **lib** obsahuje libovolné uživatelské knihovny, které jsou ve webové aplikaci využívány. Knihovny jsou ve standardním archivu `.jar`. Servlet kontejner je zodpovědný za zavádění těchto knihoven, aby je bylo možné v aplikaci následně využívat.
- Adresář **classes** obsahuje přeložené třídy, které se podílejí na činnosti aplikace. Servlet kontejner je opět zodpovědný za jejich zavedení.

- Soubor **web.xml** obsahuje nejpodstatnější konfiguraci webové aplikace pro servlet kontejner. Na základě tohoto souboru servlet kontejner provádí nastavení celé aplikace. Jsou zde popsány veškeré servlety, které aplikace obsahuje, pravidla pro zabezpečení aplikace, filtry (angl. *filter*, které mohou upravovat příchozí požadavky a spousta dalších konfigurací.

Kromě základních výše popsaných položek, může archiv obsahovat libovolné jiné soubory pro běh aplikace jako jsou HTML soubory, kaskádové styly (CSS), obrázky a jiné. Servlet kontejner musí spravovat tyto zdroje a při příchozích požadavcích je načítat. Pro odkazování těchto souborů je jako kořenový adresář považován samotný archiv aplikace, takže cesta k souboru web.xml je následující: `/WEB-INF/web.xml`.

2.3 Nástroj Maven

Pro vývoj a práci se serverem Jenkins CI v podobě zdrojových kódů je zapotřebí nástroj Maven. Jeho využití je také nutné pro provedení integrace serveru Undertow do Jenkins CI. V této kapitole budou o něm poskytnuty pouze základní informace. Pro případné bližší seznámení se s tímto nástrojem lze další informace získat z webových stránek projektu [22], ze kterých byly čerpány informace v této kapitole.

Maven je nástroj, který provádí činnosti spojené s vytváření spustitelných aplikací ze zdrojových kódů a je zaměřen na aplikace vytvářené v jazyce Java. Jeho hlavním cílem je umožnit uživatelům v co nejkratším čase provádět běžné a opakující se úkony při překladu aplikace. Především umožňuje provádění překladu aplikací, spuštění jejich automatizovaných jednotkových a integračních testů, uložení spustitelné podoby aplikace do lokálního repozitáře a případně i nahrání vytvořené aplikace na nějaký server.

Je dodáván jako aplikace, která se spouští z příkazové řádky příkazem `mvn`. Veškeré informace o projektu a definice činností, které má Maven provést, se specifikují v XML souboru `pom.xml`. Mezi nejdůležitější patří definice výsledné podoby aplikace a způsob jejího překladu, informace o závislostech projektu na jiných knihovnách a definice serverů, ze kterých má stahovat potřebné knihovny pro běh aplikace. Po spuštění aplikace jsou tedy lokalizovány knihovny, na kterých projekt závisí, a uloženy v lokálním repozitáři aniž by se o tuto činnost uživatel musel dále starat.

Zmíněný lokální repozitář neslouží pouze ke stahování knihoven ze serveru, ale také pro ukládání lokálně vyvíjených aplikací. Pokud je aplikace vyvíjena lokálně a skládá se z více nezávislých modulů, tak je potřeba postupně tyto moduly přidávat do lokálního repozitáře, aby je při sestavování výsledného programu Maven mohl najít (na serveru je totiž nenajde). Pro přidání modulu či aplikace do lokálního repozitáře slouží parametr `install`.

Na příkladu 2.3.1 je ukázka části souboru `pom.xml`, který je vytvořen pro Jenkins CI:

Příklad 2.3.1. Ukázka souboru `pom.xml`

```
...
<repositories>
  <repository>
    <id>repo.jenkins-ci.org</id>
    <url>http://repo.jenkins-ci.org/public/</url>
...
<dependency>
  <groupId>org.mockito</groupId>
  <artifactId>mockito-core</artifactId>
  <version>1.8.5</version>
</dependency>
...
```

V části `repository` je definována cesta k serveru, z kterého má Maven stahovat potřebné soubory a v těle značky `dependency` je pomocí hodnot *groupId, artifactId* a *version* jednoznačně definována potřebná knihovna. Na základě těchto informací poté Maven při překladu stáhne požadovanou knihovnu a při distribuci aplikace nemusí být tato knihovna k programu přibalena.

2.4 Server Jetty

Jednou z komponent, které jsou aktuálně integrovány do serveru Jenkins CI, je webový server Jetty. Tento server je *open source* projektem vyvíjeným pod licencemi Eclipse[7] a Apache[8]. Je využíván velkým množstvím nástrojů jako jsou například *Eclipse IDE*[9] nebo *Google AppEngine*[10].

Jetty je webový server a poskytuje funkcionalitu servlet kontejneru dle specifikace verze 3.0. Návrh serveru umožňuje samostatné spouštění aplikací i jeho integraci do jiné aplikace. Kromě těchto základních možností obsahuje řadu souvisejících technologií, jako jsou SPDY, webové sokety[11](angl. *websocket*) a další. Informace v této kapitole byly čerpány z webových stránek serveru Jetty [23], kde lze nalézt další informace o zmíněných technologiích a o tomto serveru.

Aktuální využití serveru Jetty v aplikaci Jenkins CI je podrobněji rozebráno v kapitole 2.6 a jeho srovnáním se serverem Undertow se zabývá kapitola 2.7.3.

2.5 Servlet kontejner Winstone

Další komponentou serveru Jenkins CI je servlet kontejner Winstone. Winstone je velmi jednoduchý a poskytuje funkcionalitu servlet kontejneru aniž by byl zatížen velkým množstvím požadavků, které jsou ve specifikaci jazyka Java EE. Nikdy neposkytoval veškeré služby servlet kontejneru, které specifikace jazyka definuje. V jeho názvu je zahrnutý pouze pojem servlet kontejner, ale tato aplikace vykonává i služby webového serveru odpovídající popisu v kapitole 2.2.

[7] Licence Eclipse je dostupná na adrese: `http://www.eclipse.org/legal/epl-v10.html`
[8] Licence Apache je dostupná na adrese: `http://www.apache.org/licenses/LICENSE-2.0.html`
[9] Webové stránky projektu Eclipse IDE: `http://www.eclipse.org/`
[10] Webové stránky platformy Google AppEngine: `https://developers.google.com/appengine/`
[11] Oficiální stránky specifikace webových soketů: `http://www.websocket.org/`

Hlavními cíli projektu bylo poskytovat funkcionalitu servlet kontejneru pouze pro jednu aplikaci, což je opačný přístup než u běžných aplikačních serverů jako jsou Glasfish, JBoss a jiné. Díky jeho omezeným službám je jeho velikost velmi malá a umožňuje jednoduchou integraci s cílovou aplikací.

Uvedené informace byly čerpány z oficiální stránky projektu Winstone [4].

2.5.1 Nedostatky servlet kontejneru Winstone

Původní myšlenka jednoduchého servlet kontejneru byla pro projekt Jenkins CI zajímavá, ale kontejner Winstone má několik zásadních nedostatků, kvůli kterým bylo časem jeho využití v Jenkins CI problematické [3].

Vývoj projektu Winstone již před delší dobou ustal a tím pádem nebyla poskytována žádná další podpora pro řešení a opravování objevených nedostatků a chyb. Značné množství bezpečnostních chyb objevených v projektu Jenkins CI bylo právě způsobeno tímto servlet kontejnerem.

Zpočátku možnosti servlet kontejneru postačovaly, ale časem bylo potřeba, aby byly přidávány nové funkcionality, které odpovídají aktuálním trendům a novým specifikacím. Příkladem mohou být nové specifikace servlet kontejneru nebo vývoj nových technologií jako webové sokety.

2.6 Vývoj servlet kontejneru v komunitě Jenkins CI

Po ukončení vývoje projektu Winstone se musela o jeho potřebné úpravy starat komunita Jenkins CI. Takováto práce je pro komunitu velmi zatěžující a naprosto neefektivní. Byly prováděny především nutné opravy bezpečnostních chyb v kontejneru, ale jinak aplikace dále degenerovala. Pro tento vývoj vznikl v projektu Jenkins CI nový repozitář, který vycházel z původní verze servlet kontejneru Winstone[12]. Z tohoto zdroje a z článku o integraci Jetty s kontejnerem Winstone [3] jsou čerpány uváděné informace.

Vývoj původního servlet kontejneru Winstone probíhal uvedeným způsobem až do verze *0.9.10-jenkins-47*. Následně byl tento způsob vývoje zastaven a do kontejneru Winstone byl integrován webový server a servlet kontejner Jetty. Tímto krokem vznikla interní verze servlet kontejneru *Winstone 2.0*. Velká část kódu kontejneru Winstone byla odstraněna a veškerá činnost webového serveru a servlet kontejneru je nyní vykonávána pomocí serveru Jetty. Z původního kontejneru Winstone zůstaly pomocné činnosti jako zpracování vstupních parametrů. Tato změna proběhla poměrně narychlo a nebyla detailně otestována. Obsahuje zřejmě ještě množství nepotřebného kódu a samotná implementace je dosti nepřehledná.

Těsně před integrací serveru Jetty s Jenkins CI vznikalo zadání tohoto diplomového projektu, které mělo za cíl výrazně zlepšit aktuální stav servlet kontejneru v projektu Jenkins CI. Během formulace zadání byl servlet kontejner v projektu přepracován a proto muselo být zadání upraveno. I po této změně byly stále důvody pro nahrazení stávajícího servlet kontejneru serverem Undertow. Aktuální situace již není tak kritická jako v předchozí verzi, ale server Undertow může přinést ještě další zlepšení.

[12] Repozitář, kde komunita Jenkins CI provádí úpravy servlet kontejneru Winstone: `https://github.com/jenkinsci/winstone`

2.7 Server Undertow

Undertow je webový server napsaný v jazyce Java, který vzniká za podpory firmy Red Hat a její sekce JBoss. Primárním účelem serveru Undertow je být výchozím webovým serverem v aplikačním serveru WildFly. Jeho první finální verze byla vydána teprve před nedávnem, takže se jedná o nově vytvořený server a jeho vývoj stále usilovně probíhá[13]. Pro stažení a využití tohoto serveru je nejjednodušší využít aplikaci Maven (kapitola 2.3) a nastavit patřičnou závislost[14].

Informace v této kapitole byly čerpány především z webových stránek projektu [24] a dokumentace serveru [9], ale jelikož je aplikace poměrně nová, tak zveřejněná dokumentace je poměrně nedostatečná. Některé informace z této kapitoly musely být čerpány z vygenerované projektové dokumentace a z komunikace s vývojáři projektu na chatu IRC.

2.7.1 Vlastnosti serveru

Server Undertow je zaměřen na to, aby byl plně integrovatelný do libovolných aplikací a byl co nejmenší a nejjednodušší. Samotný archiv s jádrem aplikace je menší než 1 MB a při běhu aplikace potřebuje méně než 4 MB dynamicky alokované paměti.

Je navržen takovým způsobem, aby při implementaci mohl uživatel využít jen část aplikace, kterou nutně potřebuje, a patřičně si ji upravit pro své vlastní potřeby. Tohoto přístupu je dosaženo kombinováním a řetězením obslužných funkcí (angl. *handler*), které server poskytuje. Díky tomuto přístupu je server velmi flexibilní a v jeho důsledku také patřičně rychlý, protože uživatele nebrzdí funkcionality serveru, které nutně nepotřebuje a nevyužívá.

Při komunikaci umožňuje server podporu jak pro asynchronní, tak pro synchronní komunikaci. Dalšími funcionalitami, které server poskytuje, jsou možnost integrace servlet kontejneru odpovídajícího specifikaci verze 3.1, využití plné podpory webových soketů (angl. *websockets*) nebo podpory technologie *HTTP upgrade*.

2.7.2 Architektura serveru

V této kapitole jsou podrobněji rozebrány základní principy, jak Undertow funguje, a je zjednodušeně popsána jeho architektura z pohledu uživatele. Další podrobnější informace o serveru lze nalézt ve webové dokumentaci projektu, která byla hlavním zdrojem této kapitoly [9]. Popis architektury serveru se přímo vztahuje k samotné integraci do Jenkins CI.

Webový server

Architektura serveru Undertow nevyužívá koncept jednoho velkého kontejneru, který se pomocí vysokoúrovňového rozhraní nastaví pro danou aplikaci a sám o sobě funguje. Naopak aplikace jsou sestavovány z množství tříd tzv. ***handlerů*** (viz níže), které spravují příchozí požadavky a utvářejí tak samotný webový server. Díky tomuto konceptu je možné využít jen ty funkcionality serveru, které jsou v aplikaci potřeba, a server není brzděn zbytečnými činnostmi, které nejsou pro konkrétní aplikaci potřebné.

Vstupním bodem aplikace jsou tzv. ***listenery*** (angl. *listener*), které naslouchají na určených síťových rozhraních a portech a zpracovávají příchozí požadavky. Jednotlivé *listenery* se liší dle protokolu, kterým komunikují. V Undertow jsou 3 základní typy *listenerů*

[13] Aktuální verzi serveru, lze najít na serveru GitHub: `https://github.com/undertow-io/undertow`

[14] Definice závilosti v aplikaci Maven pro stažení serveru Undertow: `http://undertow.io/downloads.html`

pro protokoly HTTP, HTTPS, a AJP. Také je vytvořena podpora protokolu SPDY, ale v aktuální verzi ještě není zahrnuta. Pro uživatele poskytují abstrakci nad samotnými síťovými protokoly, takže v serveru lze stejným způsobem zpracovávat požadavky přicházejícími v různých protokolech. Jediným samozřejmým omezením je, že nelze v jednom protokolu používat informace specifické pouze pro protokol jiný.

Pro umožnění této abstrakce každý *listener* provádí překlad požadavku do objektu třídy `HttpServerExchange`. V něm je uchováván aktuální stav požadavku a vytvářené odpovědi a při odesílání odpovědi *listener* z tohoto objektu vytvoří odpověď, která je poslána klientovi.

Tyto listenery jsou těsně navázané na knihovnu XNIO[15], která pro Undertow poskytuje abstrakci nad síťovou komunikací. Samotné listenery v Undertow je možné konfigurovat nastavením několika parametrů [10], ale lze také přímo konfigurovat síťové kanály knihovny XNIO.

Samotné jádro serveru tvoří již zmiňované *handlery*. ***Handler*** je třída, která definuje metodu `handleRequest` v níž je příchozí požadavek na serveru libovolně upraven a předán dalšímu *handleru* ke zpracování nebo je vytvořena odpověď a poslána zpět klientovi (tedy jsou vynechány následující *handlery*). *Handlery* jsou za sebou napojeny a tvoří řetěz (angl. *handler chain*). Je možné si nadefinovat libovolný vlastní *handler*, který pouze musí implementovat patřičné rozhraní s metodou `handleRequest` nebo je možné využít již předpřipravených tříd. V těchto předpřipravených třídách jsou běžně požadované funkce serveru jako úprava hlaviček příchozího požadavku, přesměrování požadavku apod. Pomocí jediného parametru (objekt třídy `HttpServerExchange`) metody `handleRequest` si *handlery* předávají aktuální stav požadavku a postupně vytvářené odpovědi.

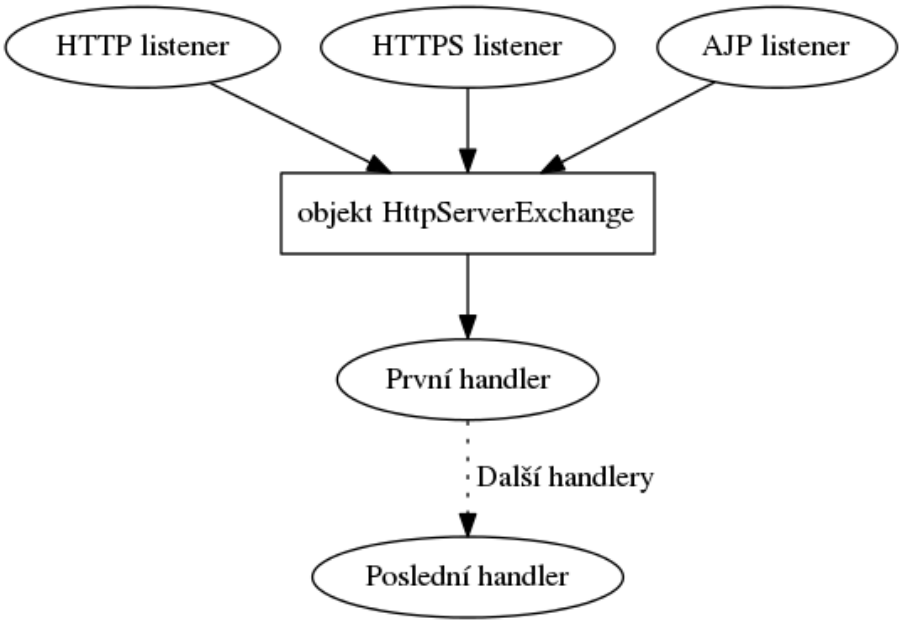
Popisované *handlery* a *listenery* jsou základem architektury serveru Undertow, která je znázorněna na obrázku 2.4. Na příkladu 2.7.1 je ukázáno vytvoření jednoduchého webového serveru.

Příklad 2.7.1. Vytvoření jednoduché instance serveru Undertow

```
Undertow server = Undertow.builder()
  .addHttpListener(8080, "localhost")
  .setHandler(new HttpHandler() {
    @Override
    public void handleRequest(final HttpServerExchange exchange)
        throws Exception {
      exchange.getResponseHeaders()
        .put(Headers.CONTENT_TYPE, "text/plain");
      exchange.getResponseSender().send("Hello World");
    }
  }).build();
server.start();
```

Třída `Undertow` reprezentuje instanci samotného webového serveru a je ji nutné definovat prakticky vždy. Metodou `addListener` je přidán výše zmiňovaný *listener*, který naslouchá na zvoleném síťovém rozhraní a portu (při zvolení IP adresy 0.0.0.0 naslouchá na všech rozhraních). Tento jednoduchý příklad obsahuje pouze jeden *handler* přidaný metodou `addHandler`, který provádí veškerou činnost serveru a tou je odpovídání na všechny

[15] Webové stránky projektu XNIO: `http://xnio.jboss.org/`



Obrázek 2.4: Zjednodušená architektura serveru Undertow

požadavky jednotným textem. Ve standardních aplikacích by následovalo volání dalšího *handleru*.

Servlet kontejner

Servlet kontejner obsažený v serveru Undertow je tvořen již implementovaným řetězcem *handlerů*, které poskytují potřebné funkcionality odpovídající specifikaci. Pro nastavení základní funkcionality kontejneru existuje vysokoúrovňové rozhraní, ale opět je možné jeho funkce upravit nebo rozšířit pomocí vlastních *handlerů*.

Při inicializaci servlet kontejneru se nastavují požadované funkce v entitě `DeploymentInfo` a až po jeho kompletním nastavení se provede vytvoření řetězce *handlerů*, což dává prostor k vnitřním optimalizacím. Základem konfigurace je potřeba nastavit prvky webové aplikace, které jsou definovány v souboru web.xml (viz. kapitola 2.2.3). Pro nastavení většiny možných prvků tohoto souboru je definováno několik statických metod ve třídě `io.undertow.servlets.Servlets`, které pomáhají servlet kontejner patřičně inicializovat.

Nastavení jednoduchého servlet kontejneru je demonstrováno na příkladu 2.7.2.

Příklad 2.7.2. Konfigurace jednoduchého servlet kontejneru

```
DeploymentInfo servletBuilder = Servlets.deployment()
    .setContextPath("/myapp")
    .setDeploymentName("test")
    .addServlets(
        Servlets.servlet("MessageServlet", MessageServlet.class));

DeploymentManager manager = Servlets.defaultContainer()
    .addDeployment(servletBuilder);
manager.deploy();
```

```
HttpHandler httpHandler = manager.start();
```

V první části ukázky je vytvořená instance `DeploymentInfo`, která bude obsahovat jednoduchou konfiguraci servlet kontejneru s jediným servletem. Následně proběhne přidání konfigurace do výchozího kontejneru a pomocí volání metody `deploy` se vytvoří celý řetězec *handlerů*, které budou poskytovat požadovanou funkčnost.

Celý proces je zakončen voláním metody `start`, kdy proběhnou poslední nastavení a obdržíme vstupní *handler* celého servlet kontejneru. Poté jej lze nastavit jako vstupní *handler* celé aplikace (viz metoda `addHandler` v příkladu 2.7.1), ale také mu může předcházet několik jiných handlerů, což je typičtější případ.

Pokročilá konfigurace servlet kontejneru

Potřebných či možných nastavení servlet kontejneru je celá řada. Z těch nejdůležitějších stojí za zmínku správa zdrojů (angl. *resource management*), nastavení autentizace a zabezpečení servlet kontejneru a úprava činnosti kontejneru pomocí vložení vlastních *handlerů*.

Pro správu zdrojů je nutné vytvořit instanci rozhraní `ResourceManager`, jinak kontejner při každém požadavku na zdroj žádný nenajde (zjištěno ze zdrojových kódů). Je možné vytvořit si vlastní implementaci pro správu zdrojů nebo použít jednu z implementací, které Undertow obsahuje:

- `FileResourceManager` – spravuje zdroje z určené složky souborového systému
- `ClassPathResourceManager` – spravuje soubory, které se nacházejí na *classpath* kontejneru (cesta ke knihovnám určená při spuštění programu)
- `CachingResourceManager` – zastřešuje jinou implementaci správce zdrojů a pro urychlení pracuje jako vyrovnávací paměť (angl. cache)

Servlet kontejner může poskytovat dva základní typy zabezpečení. První možností je poskytovat autentizaci v případech, které uživatel deklaruje v souboru *web.xml*. Při tomto způsobu kontejner kontroluje příchozí požadavky a v případě potřeby požádá uživatele o autentizaci. Pro správnou funkčnost tohoto způsobu zabezpečení je potřeba vytvořit objekt `LoginConfig` [13] (odpovídající nastavení části *login-config* ve *web.xml*) a vytvořit vlastní implementaci rozhraní `IdentityManager`. Tato instance na základě požadavku rozhoduje, zda uživatel byl rozpoznán (a případně jej blíže identifikuje) nebo zamítne požadavek na autentizaci.

Druhou možností je definovat libovolný počet instancí rozhraní `AuthenticationMechanism`. Každý příchozí požadavek následně prochází každou instancí, jejíž základním činností je určit, zda požadavek má mít umožněn přístup do kontejneru nebo ne. Bližší informace lze nalézt v dokumentaci z níž byly čerpány informace v této kapitole [11], [12].

Upravit činnost servlet kontejneru je navíc možné tak, že se do něj vloží vlastní instance *handlerů* a ty libovolně upravují příchozí požadavky. Samotné vložení není možné provést přímo, protože servlet kontejner se sestavuje automaticky na základě definovaných vlastností. Je tedy nutné zaobalit handler do instance rozhraní `HandlerWrapper`, který toto vložení umožní. Tato instance umožní následné vložení *handleru* do řetězce *handlerů* v kontejneru. Je možné jej vložit před vykonáním všech *handlerů* serveru, po nastavení kontextu požadavku nebo po vykonání všech handlerů před spuštěním kódu uvnitř uživatelského servletu [11].

2.7.3 Srovnání serverů Undertow a Jetty

Stávající servlet kontejner v Jenkins CI je sice tvořen kombinací servlet kontejneru Winstone a serveru Jetty, ale veškerou časově náročnou práci při běhu aplikace vykonává server Jetty.

Budeme tedy srovnávat servery Undertow a Jetty, jejich výhody a nevýhody vzhledem k využití v systému Jenkins CI:

- **Rychlost:** Server Undertow je sice nový, ale už přesto existují testy v kterých se ukázal být výkonnější než server Jetty. V tomto testování[16] byl v počtu zpracovaných požadavků za jednotku času server Undertow i 3,5 krát rychlejší než server Jetty . Při porovnávání doby odezvy serveru dosáhl server Undertow až třetinového času než server Jetty.

  Naopak při některých jiných konfiguracích byl rychlejší server Jetty a proto nelze definitivně určit, který server je rychlejší a podstatný je také způsob využití serveru. Dalším důležitým faktem je, že v Jenkins CI není využita nejnovější verze serveru Jetty (verze 9), ale jeho nižší verze 8, protože Jenkins CI aktuálně využívá starší verzi jazyka Java (verzi Java 6).

  Lze tedy usuzovat, že server Undertow má potenciál být výkonnějším. V rámci této práce bylo provedeno vlastní základní testování obou těch serverů a to ve vztahu přímo k systému Jenkins CI. Tímto testováním se zabývá kapitola 5.

- **Spolehlivost:** Dalším důležitým aspektem je spolehlivost daného serveru. Server Jetty má za sebou již dlouhou historii a je integrován ve velkém množství různých aplikací[17], což mu dodává velkou důvěryhodnost a lze očekávat, že bude pracovat velmi spolehlivě.

  U serveru Undertow byla dokončena první finální verze teprve nedávno, takže lze předpokládat, že může obsahovat ještě drobné nedostatky, které budou časem opravovány. Jelikož tento server je integrován v novém aplikačním serveru WildFly 8, tak lze předpokládat, že postupem času bude také velmi spolehlivý. Nicméně v tomto aspektu je server Jetty zřejmě aktuálně lepší.

- **Konfigurovatelnost:** Server Undertow je velmi flexibilní a způsob jeho návrhu, který byl popsán výše, umožňuje provádět mnoho různých úpravy své činnosti dle potřeby a využít pouze ty části, které potřebujeme. Tato filozofie je velmi blízká filozofii projektu Jenkins CI.

  Server Jetty je oproti tomu robustnější a rozsáhlejší, ale neumožňuje tak velké přizpůsobování potřebám uživatele jako například využití jen několika malých částí jeho funkčnosti či jejich kombinování.

- **Snadnost použití:** Server Jetty má již dlouhou historii a je v něm vše pečlivě dokumentováno a to jeho využití pro uživatele činí poměrně snadným. Podstatnou výhodou je, že server Jetty poskytuje velké množství vysokoúrovňových rozhraní pro různé konfigurace serveru.

  Jelikož je server Undertow velmi mladým projektem a není tolik ustálený, tak postrádá některé věci, které jsou u serveru Jetty samozřejmostí. Dokumentace k serveru Undertow je poměrně strohá, místy ne úplně přesná a stále se vyvíjí. Jeho velká flexibilita

[16] Porovnání rychlosti, odezvy a celkové výkonnosti serverů Undertow, Jetty a jiných: `http://www.techempower.com/benchmarks/#section=data-r8&hw=ec2&test=plaintext`

[17] Aplikace, které využívají server Jetty: `http://www.eclipse.org/jetty/powered/`

při konfiguraci s sebou nese i jisté nevýhody a tím je právě složitější způsob použití. Další nepříjemností je, že není dostupné tolik propracované rozhraní pro konfiguraci jako u serveru Jetty, což lze opět přičítat délce trvání projektu.

Je zjevné, že využití serveru Jetty je podstatně jednodušší než je tomu u serveru Undertow. Nicméně tato skutečnost je jen překážkou pro provedení integrace, ale následně nemusí činit další problémy a nijak neovlivňuje funkčnost výsledné aplikace.

Kapitola 3

Analýza současného stavu servlet kontejneru a návrh integrace

Tato kapitola se zabývá důkladnější analýzou a zkoumáním architektury aplikace Jenkins CI z pohledu jejího vestavěného kontejneru a jeho možných úprav (kapitola 3.1). Jsou konkretizovány jednotlivé činnosti, které současný servlet kontejner provádí a které musí nová implementace také poskytovat (kapitola 3.2). Poslední část této kapitoly diskutuje možné varianty integrace serveru Undertow do Jenkins CI, jejich výhody a nevýhody (kapitola 3.4). Jako výstup této analýzy je zvolen způsob integrace, jehož implementací se zabývá následující kapitola 4.

Jelikož k této problematice je minimum oficiální zdrojů, které by danou problematiku blíže popisovaly, tak převážná část zde uváděných informací byla čerpána přímo ze zdrojových kódů aplikace a komponent Jenkins CI. Na jejich základě byla prováděná analýza současného stavu servlet kontejneru v aplikaci a jeho architektury.

3.1 Aktuální stav architektury servlet kontejneru v Jenkins CI

V následujících dvou kapitolách je blíže analyzována architektura Jenkins CI z pohledu servlet kontejneru. Tato analýza je velmi důležitá pro pochopení návazností jednotlivých komponent, které musely být v rámci integrace upravovány.

Jak již bylo uvedeno v kapitole 2.6, současný servlet kontejner v Jenkins CI se skládá z nástojů Winstone a Jetty, které byly také popisovány v předchozích kapitolách. Velká část servlet kontejneru Winstone byla odstraněna a zůstala pouze část, která provádí zpracování vstupních parametrů po spuštění aplikace, a různé menší části vykonávající pomocné činnosti jako je rozbalování archivu apod. Činnost webového serveru, servlet kontejneru a věci s tím spojené vykonává server Jetty.

3.1.1 Architektura Jenkins CI z pohledu servlet kontejneru

Na vysoké úrovni pohledu lze architekturu serveru Jenkins CI rozdělit do tří částí:

- **Jádro aplikace**, do kterého patří nejnutnější základní komponenty systému a části, které jsou pro Jenkins CI specifické a nezbytné pro jeho běh. Vývoj těchto komponent je hlavní činností v rámci celého projektu Jenkins CI.
- **Přídavné moduly** aplikace, které jsou do ní dynamicky přidávány jako archivy Java programů `.jar`. Většina těchto součástí je nutná pro standardní běh aplikace

a aplikace je s nimi běžně dodávána (teoreticky lze aplikaci spustit např. bez servlet kontejneru pomocí aplikačního serveru, ale toto je spíše ojedinělý případ).

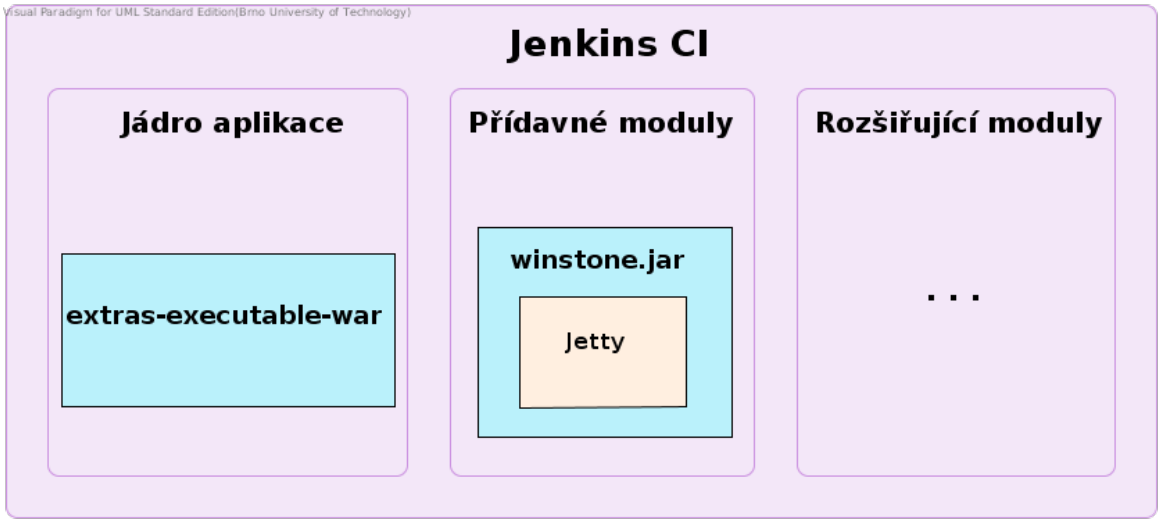
Komponenty z této kategorie pocházejí typicky z externích projektů. Spadá zde také servlet kontejner, který je aktuálně do aplikace přidáván v archivu s názvem `winstone.jar`.

- **Rozšiřující moduly** (angl. plugins) jsou samostatné menší programy, které nějakým způsobem přidávají funkcionalitu serveru Jenkins CI. Typicky je jich dodáváno jen velmi málo s distribucí aplikace a uživatel si je může stáhnout nebo si nějaký vlastní modul vytvořit.

  Samotná aplikace se dodává jen s těmi nejnutnějšími součástmi a ponechává na uživatelích, které další funkcionality si do systému doinstalují. V současné době již existují stovky takových modulů, které jsou volně ke stažení.

V následujícím textu bude při popisu interních součástí zdrojových kódů použita notace ve tvaru `Název_třídy::Název_metody`.

Architektura Jenkins CI z pohledu servlet kontejneru je znázorněna na obrázku 3.1. Jsou zde zachyceny především komponenty, které přímo souvisejí s jeho činností.



Obrázek 3.1: Přehled architektury systému Jenkins CI z pohledu vestavěného servlet kontejneru

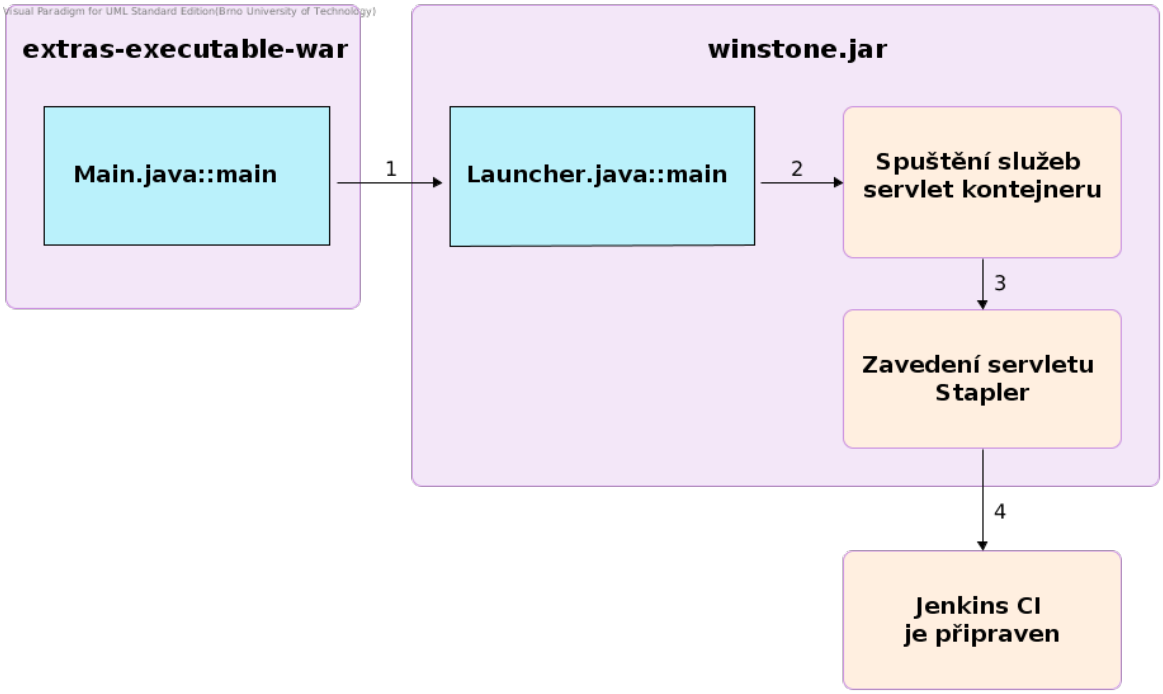
Archiv `winstone.jar` obsahuje servlet kontejner pro Jenkins CI. Název je nyní sice mírně matoucí (způsoben předchozí implementací), ale aktuálně jsou součástí tohoto archivu komponenty Winstone a Jetty.

Velmi podstatnou součástí aplikace z pohledu servlet kontejneru je komponenta `extras-executable-war`. Tato součást aplikace je sice malá, ale obsahuje metodu `main`, kterou se aplikace spouští (pokud není spuštěna v aplikačním serveru). Hlavním úkolem této komponenty je právě spustit servlet kontejner a tím spustit i celou aplikaci. Tento proces spuštění aplikace je zobrazen na obrázku 3.2.

Nejprve je spuštěna vstupní metoda celé aplikace `Main.java::main` z komponenty `extras-executable-war`. Po počáteční inicializaci je předáno řízení aplikace metodě

`Launcher.java::main` z archivu `winstone.jar`, která provede nastartování a zavedení servlet kontejneru. Následně je provedena inicializace a spuštění jediného servletu aplikace Jenkins CI a tím je servlet Stapler (bližší informace o tomto servletu jsou v kapitole 2.1.3). Pokud se podaří úspěšně spustit tento servlet, tak je spuštění celé aplikace Jenkins CI z pohledu servlet kontejneru úspěšně provedené a aplikace běží.

Zjištění, které komponenty má servlet kontejner při svém startu zavést, se nachází v souboru *web.xml*, což je standardní konfigurační soubor pro webové aplikace v jazyce Java EE. Může se zde nacházet také například konfigurace uživatelských účtů pro přístup k aplikaci nebo specifikování různých druhů přístupových práv.



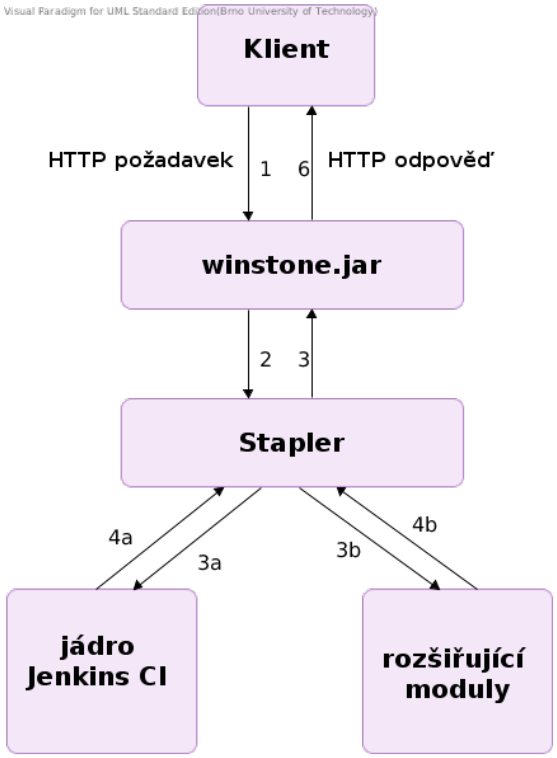
Obrázek 3.2: Průběh spouštění aplikace Jenkins CI při spouštění pomocí vestavěného servlet kontejneru

3.1.2 Průběh komunikace prostřednictvím servlet kontejneru

Po úspěšném zavedení a spuštění aplikace provádí servlet kontejner zpracovávání příchozích požadavků a předává je aplikaci Jenkins CI. Typický zjednodušený průběh komunikace servlet kontejneru s aplikací je znázorněn na obrázku 3.3.

Při komunikaci je příchozí HTTP požadavek zpracován pomocí servlet kontejneru a následně předán patřičnému servletu (dle nastavení pravidel pro směřování požadavků). Jelikož je v aplikaci pouze jediný servlet Stapler, tak je požadavek předán jemu. Tato komponenta následně provede netriviálním způsobem rozhodnutí, které části aplikace předat požadavek k vykonání nebo zda náleží nějakému rozšiřujícímu modulu. Následně stejnou cestou probíhá zaslání HTTP odpovědi zpět klientovi.

Byla zde zachycena pouze jedna z možných činností servlet kontejneru a tou je zpracovávání komunikace protokolem HTTP. Další činnosti kontejneru budou rozebírány později.



Obrázek 3.3: Průběh komunikace aplikace Jekins CI prostřednictvím vestavěného servlet kontejneru

3.2 Zpětná kompatibilita servlet kontejner v Jenkins CI

V této kapitole jsou blíže rozebírány a analyzovány jednotlivé funkce servlet kontejneru v Jenkins CI. Analýza jeho funkčnosti je zaměřena především na návrh nového servlet kontejneru implementovaného pomocí serveru Undertow. Z této kapitoly vyplývají hlavní požadavky, které musí nová implementace řešit.

Stávající servlet kontejner je v Jenkins CI již dlouhou dobu a mnoho uživatelů a komponent systému přímo používá jeho specifické parametry, které pocházejí z kontejneru Winstone [3]. Nová implementace tudíž musí zachovat naprosto stejný formát parametrů, aby se výměnou servlet kontejneru nestala část aplikace nebo rozšiřujících modulů nefunkční. Také je potřeba zachovat výchozí hodnoty jednotlivých parametrů.

3.2.1 Funkcionality servlet kontejneru

Při spouštění aplikace je před předáním řízení aplikace servlet kontejneru část parametrů zpracována komponentou `extras-executable-war` (blíže rozebrána v kapitole 3.1) a zbylé parametry jsou přímo předány servlet kontejneru, který podle nich nastaví svou činnost. Jednotlivé parametry zde nejsou rozebírány, ale následující analýza se zaměřuje především na klíčové body funkčnosti servlet kontejneru.

Pro zachování zpětné kompatibility musí servlet kontejner v aplikaci Jenkins CI poskytovat především následující funkce:

- Provádět inicializaci a spuštění webové aplikace, která je zabalená v archivu `.war` nebo je již rozbalená v nějakém konkrétním adresáři na disku.
- Poskytovat možnost komunikace nešifrovaným protokolem HTTP, který je výchozím komunikačním protokolem v serveru Jenkins CI. Aplikace musí naslouchat na všech síťových rozhraních na portu 8080.
- Umožnit komunikaci šifrovaným síťovým protokolem HTTPS. V Jenkins CI jsou dostupné 3 způsoby inicializace šifrované komunikace, které se liší poskytnutými informacemi pro zabezpečení pomocí SSL:
  - Zadání certifikátu ve formátu PEM a zároveň zadání privátního klíče serveru
  - Zadání souboru, který obsahuje úložiště certifikátů (v prostředí aplikací v jazyce Java tzv. *KeyStore*) a zároveň zadání přístupového hesla k tomuto souboru
  - Nezadání žádných bezpečnostních údajů. V tom případě si servlet kontejner musí certifikát sám vygenerovat a pomocí něj inicializovat komunikaci. Je důležité dodat, že tento způsob není pro běžné využití bezpečný a je vhodné jej použít jen pro účely testování.
- Komunikovat protokolem AJP13. Tento protokol se využívá při architektuře serveru, kdy je HTTP server umístěn až za tzv. *proxy* serverem. Typickým příkladem takového *proxy* serveru je program Apache[1]. V tomto případě je pro optimalizaci komunikace mezi HTTP serverem a *proxy* severem využit protokol AJP13 [7].
- Podporovat zabezpečení aplikace. Musí umožnit přihlašování uživatelů, správu přístupových práv a rolí. Tato problematika je blíže rozebírána v následující kapitole 3.2.2.
- Činnost servlet kontejneru může být ovlivňována pomocí komunikace přes síťové rozhraní zasláním zprávy na speciální port. Při obdržení zprávy na tomto portu je potřeba umožnit ukončení nebo restartování aplikace. Pro ukončení aplikace servlet kontejner očekává zprávu, která bude obsahovat pouze ASCII hodnotu číslice 0. Restartování aplikace provede při obdržení zprávy s ASCII hodnotou číslice 4. Jakékoliv jiné zprávy než v uvedeném formátu mohou být ignorovány.
- Zaznamenávat informace o přístupech k aplikaci. V současném servlet kontejneru je možné zaznamenávat údaje o IP adrese, jménu uživatele (pro přístup do aplikace), času, cílové URL adrese, hodnotě kódu HTTP odpovědi a o hodnotách hlaviček *Referer* a *User-Agent*, které jsou v protokolu HTTP. Při spuštění si uživatel může zvolit, které z údajů si přeje zaznamenávat a s každým přístup poté servlet kontejner uchovává patřičné informace v souboru na disku.

Po integraci serveru Jetty do servlet kontejneru byla přidána ještě možnost komunikace protokolem SPDY. SPDY je nový síťový protokol aplikační vrstvy, který je navržen pro přenos obsahu webových stránek v oblasti internetu. Jeho hlavní cílem je snížit odezvu stránek a zdokonalit komunikaci, pro kterou již protokol HTTP není ideální [16]. Jeho využití v budoucnu může přinést zrychlení aplikace, ale tento protokol ještě není v Jenkins CI více využíván a tudíž jeho podpora v servlet kontejneru není pro činnost Jenkins CI zásadní [3].

[1] Webové stránky projektu Apache: `http://httpd.apache.org/`

3.2.2 Zabezpečení v Jenkins CI

Zabezpečení serveru Jenkins CI z pohledu autentizace a autorizace je poměrně komplikované, ale pouze část se týká servlet kontejneru a vyžaduje jeho činnost.

Autorizace a autentizace je v Jenkins CI založena na tzv. filtrech (angl. *filters*) (prvky webové aplikace upravující příchozí požadavky než je začne zpracovávat servlet) a na dalších metodikách, jejich většina není pro činnost servlet kontejneru podstatná [14].

Kontrolu přístupu k jednotlivým prvkům aplikace provádí Jenkins CI samostatně, ale pro provádění autentizace využívá několik externích zdrojů. Pro nastavení autentizace poskytuje několik možností:

- Delegovat autentizaci na servlet kontejner
- Využít vlastní databázi uživatelů definovanou v Jenkins CI
- Autentizovat uživatele pomocí LDAP serveru
- Delegovat autentizaci na databázi uživatelů v operačním systému UNIX. Uživatelé se mohou přihlásit zadáním jejich přístupových práv do operačního systému.

Pro podporu autorizace a autorizace musí servlet kontejner zajistit správné zavedení aplikace a nastavení všech položek souboru *web.xml*, které se týkají bezpečnosti. Jedná se tedy především o následující položky [17]:

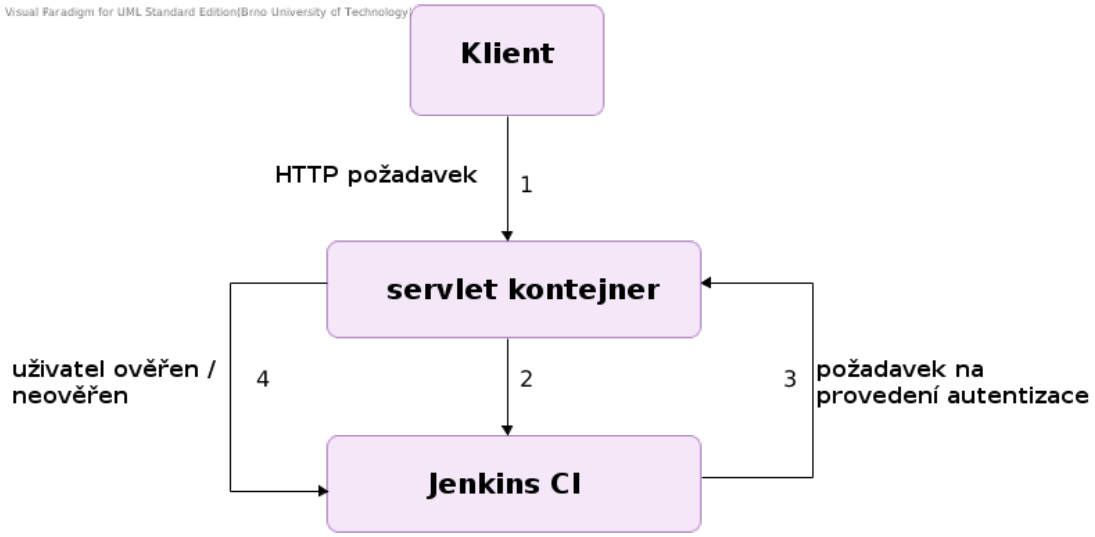
- *security-role* umožňuje specifikovat různé typy rolí pro uživatele v systému, které je jim možné přidělit
- *security-constraint* definuje části aplikace mají chráněný přístup a role, které k nim mohou přistupovat. V Jenkins CI je tato kontrola prováděná interně a je specifikován omezený přístup pouze k přihlašovacímu formuláři.
- *login-config* umožňuje určit způsob jakým se má po uživateli vyžadovat zadání přístupových údajů, pokud přistupuje k chráněnému zdroji, a na jaké stránky má být přesměrován v případě úspěchu nebo neúspěchu pokusu o přihlášení

Autentizace pomocí servlet kontejneru

Jedním z výše uvedených způsobů autentizace v Jenkins CI je využít servlet kontejneru pro autentizaci. Princip tohoto způsobu autentizace v Jenkins CI je znázorněn na obrázku 3.4. Tento způsob funguje následovně. Příchozí požadavek je předán servlet kontejnerem do Jenkins CI k normálnímu zpracování. Pokud Jenkins CI zjistí, že uživatel není autentizován, tak pošle požadavek servlet kontejneru pro ověření uživatele. Servlet kontejner provede ověření a vrátí buď identitu ověřeného uživatele nebo jeho ověření zamítne (z důvodu neplatných přihlašovacích údajů).

Při použití tohoto způsobu autentizace je nutné správně nakonfigurovat servlet kontejneru před spuštěním.

1. Definovat dostupné bezpečnostní role pro systém v souboru *web.xml*
2. Specifikovat uživatele systému s jejich hesly a s libovolným výčtem rolí, které jsou definovány v prvním kroku



Obrázek 3.4: Průběh autentizace uživatele pomocí servlet kontejneru

V aktuálním servlet kontejneru jsou předdefinovány dvě třídy, které provádějí autentizaci, a je možné si zvolit jednu z nich nebo si dodefinovat třídu vlastní. Dostupné třídy poskytují tyto dva způsoby přidávání uživatelů [5]:

- Přidávání uživatelů pomocí parametrů při spuštění příkazové řádky při spuštění servlet kontejneru
- Definování uživatelů v XML souboru s implicitním jménem *users.xml* nebo lze specifikovat umístění tohoto souboru pomocí parametru. Formát položky pro zadání uživatele je uveden v příkladu 3.2.1

Příklad 3.2.1. Formát položky XML souboru pro zadávání uživatelů v aktuálním servlet kontejneru

```
<user username="joe" password="doe" roles="user,admin" />
```

3.3 Zjištěné problémy

Z analyzování možností integrace vyplývá jeden problém. Při integraci serveru Jetty do Jenkins CI byla přidána možnost využít protokol SPDY, jehož implementace není v současné době v serveru Undertow dostupná. Nicméně tato možnost je zavedena jen krátce a zřejmě ještě není využívána žádnými komponentami, takže by neměl být problém, kdyby nová implementace neobsahovala tuto volbu.

Je důležité poznamenat, že základ implementace tohoto protokolu byl v serveru Undertow již proveden, ale ještě není začleněn do aktuální verze. Takže z dlouhodobého hlediska bude možné tento problém odstranit.

3.4 Návrh způsobu integrace serveru Undertow

Na základě analýzy, která proběhla v předchozích kapitolách, je v této kapitole zvolen způsob integrace serveru Undertow do systému Jenkins CI. Obě navržené možné varianty

integrace jsou porovnány z různých hledisek a je zvolen způsob, kterým bude následně implementace provedena.

3.4.1 Varianty integrace

Pro integraci serveru Undertow do systému Jenkins CI jsou možné tyto dva přístupy:

1. Nahrazení pouze serveru Jetty v servlet kontejneru a ponechání zbytku implementace kontejneru Winstone, což by představovalo pouze úpravy části stávajícího kódu.
2. Nahrazení jak serveru Jetty tak kontejneru Winstone. Tento přístup v podstatě znamená provedení celé implementace servlet kontejneru pro Jenkins CI úplně znovu.

Obě varianty integrace mají své klady a zápory. Srovnáme je z hlediska výkonu, proveditelnosti a zpětné kompatibility implementace, abychom mohli následně zvolit vhodnější variantu:

- **Srovnání výkonu:** V první variantě integrace je ponechána stále poměrně stará implementace kontejneru Winstone, která zřejmě stále ještě obsahuje nepotřebné součásti. Samotný způsob inicializace není optimalizován pro potřeby serveru Undertow, takže by mohlo docházet ke zbytečnému zpomalování aplikace. Nová implementace může být přímo optimalizována pro potřeby serveru Undertow a poskytovat lepší výkon i kratší dobu spuštění aplikace.
- **Srovnání proveditelnosti:** První varianta představuje provedení úprav pouze v částech, kde je přímo integrován server Jetty, zatímco v druhé variantě je potřeba provést znovu celou implementaci integrace servlet kontejneru.

  V tomto případě je úprava stávajícího kódu aplikace zřejmě snazším přístupem a také umožňuje provedení rychlejší implementace, protože není nutné návrhovat celý modul, ale pouze jeho části, a je možné využít některé stávající části aplikace jako zpracovávání parametrů, rozbalování archivů, apod.
- **Srovnání zpětné kompatibility:** Dosažení zpětné kompatibility u první varianty je snazší, jelikož jsou viditelná místa, která je potřeba znovu implementovat a nezanedbat. Nicméně druhá varianta nemá žádné faktické nevýhody, které by bránily dosažení stejné úrovně zpětné kompatibility jako u první varianty.
- **Další hlediska:** Jelikož dlouhodobý vývoj servlet kontejneru v Jenkins CI je na okraji zájmu a jsou prováděny především nejnutnější úpravy, tak celý kód poněkud zdegeneroval a je obtížně čitelný. Toto je u *open source* projektu velká závada a může bránit dalším přispěvovatelům k provádění potřebných změn.

  Z tohoto pohledu by nová implementace mohla přinést mnohem čitelnější způsob řešení a zbavit kód zbytečností z původní realizace servlet kontejneru.

3.4.2 Zvolení způsobu integrace

Při výběru varianty integrace serveru Undertow do Jenkins CI byly zváženy výhody a nevýhody, které byly popsány v předchozí kapitole. Při nahrazení pouze serveru Jetty by provedení integrace zřejmě probíhalo podstatně snadněji než ve variantě druhé, ale z kvalitativního pohledu se tato varianta jeví jako méně vhodná. Hlavními negativy jsou především potenciálně nižší výkonnost a také špatná čitelnost zdrojových kódů.

Při implementaci budou tedy nahrazeny obě komponenty stávajícího servlet kontejneru a bude tudíž provedena celá implementace znovu s využitím serveru Undertow. Tato varianta by měla přinést lepší čitelnost zdrojového kódu a poskytnout lepší výkonnost celé aplikace než druhá varianta.

Architekturu spouštění a integrace servlet kontejneru v aplikaci není potřeba měnit a při integraci bude zachován stávající stav tak, který byl popsán při analýze architektury (kapitola 3.1) Zvolený způsob integrace vyžaduje vytvoření úplně nového projektu a jeho začlenění do stávající verze serveru Jenkins CI. Nový servlet kontejner musí být vyvíjen jako samostatný modul, který bude zabalen v archivu `.jar`, aby jej bylo možné následně vložit jako externí modul do Jenkins CI. Pro spuštění nového servlet kontejneru je potřeba upravit modul `extras-executable-war` (viz kapitola 3.1), který je vstupním bodem aplikace a mimo jiné provádí spuštění servlet kontejneru.

Hlavním cílem tohoto projektu je zachovat kompatibilitu s aktuální verzí servlet kontejneru a tudíž je potřeba implementovat všechny podstatné funkcionality tak, jak byly popsány v kapitole 3.2.

Kapitola 4

Implementace

Tato kapitola popisuje způsob, jakým byla provedena samotná integrace serveru Undertow do Jenkins CI. Jednotlivé kroky zde popisované se přímo vážou k informacím, které byly uvedeny v předchozích kapitolách.

4.1 Základní informace

Integrace serveru Undertow do Jenkins CI zahrnovala vytvoření nového projektu, který jsem pojmenoval *Undertow4Jenkins*. Jak vyplývá z předchozích kapitol, tak program byl vytvořen v jazyce Java a jeho základ tvoří server Undertow. Z důvodu kompatibility se serverem Jenkins CI musela být využita starší verze jazyka a to verze 6. Pro správu a překlad projektu bylo nutné použít program Maven.

Projekt Jenkins CI je umístěn na serveru GitHub v několika oddělených repozitářích, které využívají verzovací program Git. Aby vývoj nového servlet kontejneru bylo možné následně předvést komunitě Jenkins CI, tak byl tento projekt také verzován na serveru GitHub[1] a je uveřejněn pod licencí *Apache License 2.0*[2], takže je *open source* projektem.

Na počátku integrace bylo nutné vytvořit kopie (tzv. *fork*) dvou repozitářů stávající implementace serveru Jenkins CI. Jedná se o repozitář s celým serverem[3] a také jeho modul `extras-executable-war`[4]. Vývoj nového servlet kontejneru probíhal na verzi `1.543` systému Jenkins CI, což byla nejnovější verze programu v době, kdy započala práce na tomto diplomovém projektu.

Jelikož hlavním cílem projektu je dosáhnout zpětné kompatibility s původní verzí servlet kontejneru, tak v případě nejasného přístupu k implementaci některých komponent byly studovány zdrojové kódy původní implementace a nové servlet kontejner jim byl přizpůsobován. Tohoto přístupu bylo využito například při volbě šifrovacích protokolů pro nastavení komunikaci pomocí protokolu HTTPS nebo nastavení zaznamenávání příchozích požadavků.

[1] Server GitHub je dostupný na adrese: `https://github.com/`

[2] Text licence *Apache license 2.0* je dostupný na: `http://www.apache.org/licenses/LICENSE-2.0.html`

[3] Adresa kopie repozitáře Jenkins CI: `https://github.com/jbartece/jenkins`

[4] Adresa kopie modulu `extras-executable-war`: `https://github.com/jbartece/extras-executable-war`

4.2 Rozbor způsobu implementace

Tato kapitola se více zaměřuje na způsob, jakým byla implementace provedena. Jsou rozebrány nejdůležitější části implementace výsledného servlet kontejneru, způsob integrace do Jenkins CI a celé jeho architektura. V této kapitole se předpokládá, že čtenář je základním způsobem seznámen se servery Undertow a Jenkins CI v rozsahu, který byl popsán v předchozích kapitolách.

4.2.1 Nahrazení původního servlet kontejneru

První krokem před započetím vývoje nového servlet kontejneru bylo zajištění vložení archivu `undertow4jenkins.jar` do výsledného archivu `.war`. Jelikož je Jenkins CI překládán programem Maven, tak byla změněna závislost v souboru `pom.xml` (viz kapitola 2.3 o programu Maven) z původního kontejneru *winstone* na nový projekt *Undertow4Jenkins*. Nastavení této konfigurace je uvedeno na příkladu 4.2.1. Jedná se o zvolenou identifikaci nového servlet kontejneru pro program Maven, který provede vložení archivu `.jar` do Jenkins CI ze svého lokálního repozitáře.

Příklad 4.2.1. Identifikace nového servlet kontejneru pro program Maven

```
<groupId>org.jenkins-ci</groupId>
<artifactId>undertow4jenkins</artifactId>
<version>0.1-SNAPSHOT</version>
<classifier>jar-with-dependencies</classifier>
```

Po úspěšném vložení archivu nové implementace byl upraven modul `extras-executable-war`. Většina jeho činnost byla ponechána, ale bylo nahrazeno spuštění původního servlet kontejneru jeho novou implementací. Po takto provedeném nastavení byla práce zaměřena přímo na vývoj nového servlet kontejneru.

4.2.2 Realizace základních funkcionalit

V této kapitole jsou popsány klíčové části vytvořené aplikace a způsob jakým byly implementovány. Většina těchto funkcionalit přímo koresponduje s uvedenými požadavky na zachování kompatibility s původní verzí, které jsou popsány v kapitole 3.2. Zde uváděné přístupy a komponenty serveru Undertow jsou blíže rozebrány v kapitole 2.7.

Proces inicializace servlet kontejneru probíhá v několika krocích, které na sebe navazují. V následujícím přehledu jsou jednotlivé činnosti seřazeny tak, jak se po sobě vykonávají. Část činností se provádí při každém spuštění servlet kontejneru, ale některé jsou volitelné a spouštějí se pouze pokud jsou při spuštění zadány určité parametry.

1. **Zpracování parametrů:** Jsou zkontrolovány zadané parametry aplikace a přítomnost povinných parametrů. Při spuštění aplikace musí být zadán archiv `.war` s aplikací nebo předána cesta ke složce, ve které je tento archiv extrahován. Parametry aplikace přesně odpovídají parametrům, které jsou dostupné v nápovědě serveru Jenkins CI.

   V servlet kontejneru je dostupné velké množství parametrů a proto je jejich zpracování implementováno s důrazem na obecnost a jednoduché přidávání nových parametrů. Pro přidání nového parametru stačí do třídy pro parametry přidat položku (tzv. *field*) a jeho datový typ (jsou podporovány datové typy `Boolean`, `String`, `Integer`) a jeho zpracování bude probíhat patřičným způsobem. K dosažení této obecnosti bylo

využito *Java Reflection API*. Jako podpora pro zpracování parametrů byla použita knihovna *Apache Commons CLI*[5]

2. **Příprava adresáře s webovou aplikací:** Webová aplikace je předávána servlet kontejneru ve formě archivu `.war` (parametr `warfile`). Tento archiv je následně extrahován do adresáře, který je specifikovaný jako další parametr servlet kontejneru (parametr `webroot`).

   Načítání aplikace bylo provedeno obecně a je ji možné spustit zadáním pouze adresáře, kde je archiv již extrahován. Tyto možnosti mohou posloužit např. při psaní testů.

3. **Připravení načítání tříd:** Jsou vyhledány knihovny a přeložené třídy aplikace ve standardních adresářích `/WEB-INF/lib` a `/WEB-INF/classes`, které jsou definovaným umístěním pro tyto zdroje (viz kapitola 2.2.3). Na základě dostupných knihoven a tříd je vytvořen instance třídy `URLClassLoader`, která provádí inicializaci a zavádění tříd v programu.

   Tento zavaděč je potřebný při vyhledávání tříd specifikovaných v souboru *web.xml* a jeho instance je předána serveru Undertow při inicializaci, aby bylo možné zavádět potřebné součásti aplikace za běhu.

4. **Analýza souboru web.xml:** Soubor *web.xml* je zpracován za pomoci XML parseru *StAX*, který je standardně dostupný v knihovně jazyka Java. Z analyzovaného souboru jsou načteny potřebné informace pro inicializaci Jenkins CI. Podpora všech možných konfigurací souboru *web.xml* je velmi rozsáhlá a proto byla implementována pouze podpora těch konfigurací, které jsou v systému Jenkins CI využité. Nicméně toto zpracování bylo vytvořeno obecně a je možné jej rozšířit o další konfigurace, což může být předmětem dalšího vývoje, ale není to nezbytné pro běh serveru Jenkins CI.

5. **Nastavení servlet kontejneru v Undertow:** V tomto kroku jsou již připraveny všechny pomocné informace pro nastavení servlet kontejneru a spuštění Jenkins CI. Nejprve jsou všechny načtené konfigurace ze souboru *web.xml* nastaveny v třídě `DeploymentInfo`, která reprezentuje data pro vytvoření servlet kontejneru. Při konfiguracích jsou načítány třídy z archivu aplikace pomocí zavaděče vytvořeného v předcházejících krocích.

   Jednou ze standardních konfigurací servlet kontejneru je definování typů souborů tzv. *MIME*. V původním servlet kontejneru bylo automaticky načítáno téměř tisíc těchto mapování a proto jsou tyto hodnoty načítány i v nové implementaci.

   Pro správu statických zdrojů Jenkins CI je vytvořena instance třídy `FileResourceManager` tak, aby poskytovala zdroje z adresáře s extrahovaným archivem aplikace. Tato třída byla navíc zaobalena ještě objektem třídy `CachingResourceManager`, která pracuje jako vyrovnávací paměť (angl. *cache*) a poskytuje lepší výkonnost při práci se zdroji.

6. **Nastavení správy uživatelských účtů (volitelné):** Základem autentizace uživatelů je třída, která implementuje rozhraní `IdentityManager`, a ta musí být předána Undertow před jeho spuštěním. V souladu s původní implementací servlet kontejneru byly vyvořeny dvě třídy pro autentizaci, které se liší pouze způsobem jak načítají data o uživatelích (uživatelské jméno, heslo a jeho role):

[5] Knihovna *Apache Commons CLI* je dostupná na: `http://commons.apache.org/proper/commons-cli/`

- `ArgumentsIdentityManager` – načítá informace o uživatelích z parametrů předáných při spuštění aplikace
- `FileIdentityManager` – načítá informace o uživatelích z XML souboru s definovaným formátem (viz kapitola 3.2.2)

Jelikož tyto třídy se liší pouze způsobem načítání uživatelů, tak byly ostatní činnosti potřebné pro autentizaci uživatelů extrahovány do abstraktní třídy `GenericIdentityManager`, kterou rozšiřují.

Do servlet kontejneru je možné přidat vlastní třídu pro autentizaci pomocí parametru, který definuje její plně kvalifikované jméno. Je pouze nutné, aby tato třída implementavala rozhraní `IdentityManager`.

Volba třídy pro autentizaci se provádí zadáním jejího plně kvalifikovaného jména, což je způsob, který musel být zachován v souladu s původní implementací servlet kontejneru. Zde ovšem vzniká problém, protože v původní implementaci byla jiná struktura balíků a názvy tříd. Z tohoto důvodu je v aplikaci kontrolováno, zda nebylo zadáno jméno třídy z původní implementace a je případně nahrazeno za třídu v současné implementaci. Stejný přístup byl použit při externím zadávání tříd v následujcím bodu.

7. **Nastavení zaznamenávání příchozích požadavků (volitelné):** Tato funkcionalita je řešena přidáním vlastního *handleru* na konec řetězce těsně před tím než je vyvolán samotný servlet. V servlet kontejneru je vytvořena jedna třída zajišťující tuto činnost (odpovídající implementaci v původním kontejneru), ale je opět možné přidat vlastní implementaci. Toto přidání je řešeno obdobně jako bylo popsáno v předchozím bodě, ale třída musí implementovat rozhraní `AccessLoggerHandler`.

8. **Konfigurace webového serveru:** Po inicializaci servlet kontejneru zbývá patřičně nastavit samotný webový server. Jsou incializovány konektory pro uživatelem zadané protokoly pro komunikaci (dostupnými protokoly jsou HTTP, HTTPS a AJP). Jsou implementovány tři způsoby nastavení šifrované komunikace pomocí SSL, jejichž implemetace je založena na implementaci v původním servlet kontejneru a odpovídá popisu v kapitole 3.2.1.

   Pokud uživatel specifikoval při spuštění požadavek (parametr `prefix`), aby URL adresa aplikace začínala nějakým daným řetězecem, tak je tento požadavek implementován přidáním *handleru* `RedirectHandler` jako vstupního *handleru* celé aplikace.

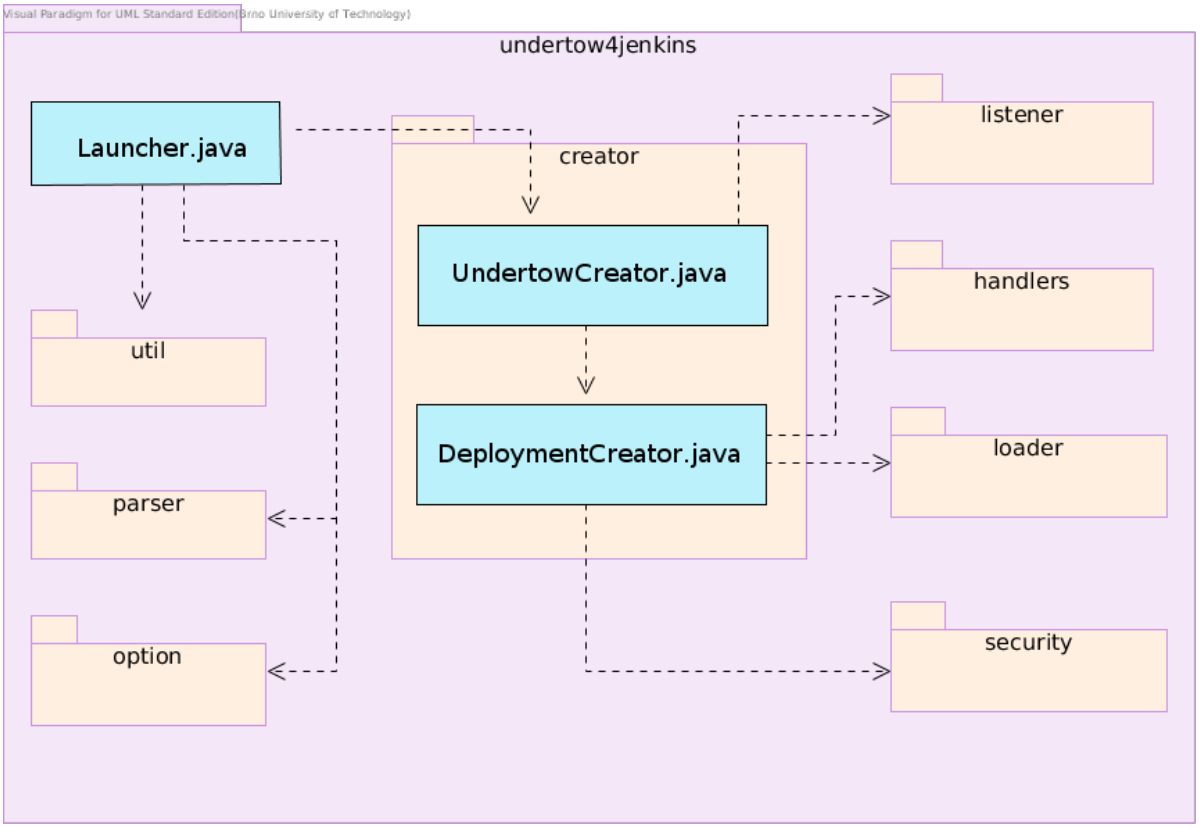
   Provedním těchto posledních konfigurací je webový server i servlet kontejner plně inicializován a následně je spuštěn.

9. **Vytvoření řídícího portu (volitelné):** V době kdy již prováhá spouštění webového serveru, tak je možné vytvořit rozhraní pro ukončení nebo restart aplikace. Je vytvořený standardní síťový *socket* a až do ukončení aplikace na něm naslouchá příchozím požadavkům (parametr `controlPort`). Protokol, který akceptuje je popsaný v kapitole 3.2.1.

4.2.3 Popis struktury programu

Tato kapitola se blíže zabývá vnitřní strukturou programu a jejím cílem je zjednodušit čtenáři orientaci ve zdrojových kódech aplikace. Popis všech tříd aplikace by byl příliš

rozsáhlý a neposkytoval by ucelený přehled o aplikaci. Budou tedy popsány pouze balíky, do kterých je aplikace rozdělena a tří nejdůležitější třídy, které jsou významné tím, že spojují jednotlivé části aplikace. Struktura aplikace je znázorněna na obrázku 4.1.



Obrázek 4.1: Diagram balíků, rozšířený o tři nejdůležitější třídy, který zachycuje architekturu servlet kontejneru

Hlavními třídami architektury programu jsou třídy:

- `Launcher` – vstupní bod aplikace obsahující metodu `main`. Zajišťuje kontrolu celého běhu servlet kontejneru.
- `UndertowCreator` – stará se o inicializaci webového serveru a konfiguraci servlet kontejneru deleguje na třídu `DeploymentCreator`.
- `DeploymentCreator` – zajišťuje inicializaci servlet kontejneru

V následujícím výčtu jsou stručně popsány balíky aplikace. Některé obsahují i větší množství tříd a proto je uvedeno pouze o jaké činnosti se třídy v daném balíku starají:

- `option` – zpracování parametrů a jejich reprezentace za běhu programu
- `parser` – načtení souboru *web.xml* a reprezetace načtených hodnot
- `creator` – vytvoření webového serveru a servlet kontejneru pomocí Undertow
- `listener` – vytvoření konektoru pro síťové protokoly HTTP, HTTPS a AJP

- `handlers` – vlastní vytvořené handlery pro běh aplikace
- `loader` – inicializace servlet kontejneru na základě načtených informací ze souboru *web.xml*
- `security` – zajištění podpory autentizace uživatelů
- `util` – zpracování webového archivu a načítání informací z konfiguračních souborů

4.3 Řešené problémy při implementaci

Při implementaci jsem se potýkal s nedostatečným komentováním zdrojových kódů serveru Undertow. Velká část serveru je naprosto nedostatečně komentovaná a největším problémem je, že není komentované ani veřejné rozraní API. Dokumentace projektu v průběhu implementace teprve postupně vznikala a navíc se zaobírá pouze popisem jednotlivých součástí, ale implementační detaily jsou rozebírány jen v některých částech. Z těchto důvodů bylo během implementace nutné velmi často zkoumat a analyzovat přímo zdrojové kódy serveru Undertow a někdy i napsané integrační testy, které umožňují lépe pochopit některé typy konfigurací (např. vytvoření podpory autentizace pomocí instance rozhraní `Identity Manager`).

Implementace některých částí aplikace byla podstatně komplikovanější oproti původní verzi servlet kontejneru a očekáváním na začátku projektu. Server Undertow neposkytuje některé standardní funkcionality pro integraci servlet kontejneru. Mezi nejvýznamnější patří nutnost zpracování souboru *web.xml* a nastavení dle něj servlet kontejneru nebo například nutnost provádět zavádění tříd servletů (tzv. *classloading*). V aplikaci Jetty lze provést načtení souboru *web.xml* pomocí jediného volání metody serveru.

4.4 Omezení aplikace

Veškerá podstatná funkčnost servlet kontejneru byla zachována a při testování nebyly odhaleny žádné chyby, ale některé nedůležité parametry nebyly implementovány. Aby nedocházelo k problémům s kompatibilitou (např. při spouštění staršími skripty), tak program zadání těchto parametrů akceptuje, ale pouze vypíše varování, že nejsou podporovány a dále je ignoruje.

Nepodporovanými parametry jsou:

- Parametr `spdy`, který umožňoval zapnutí podpory komunikace pomocí protokolu SPDY. Tento parametr nebyl implementován, jelikož v aktuální verzi Undertow není ještě podporován, ale zřejmě brzy bude přidán (viz kapitola 3.3.
- Skupina parametrů `httpDoHostnameLookups`, `httpsDoHostnameLookups`, `handlerCountStartup`, `logThrowingLineNo`, které zůstaly v kontejneru jako pozůstatek z původní implementace, ale po integraci serveru Jetty do servlet kontejneru se přestaly využívat.
- Parametry `webappsDir` a `hostsDir` byly v aktuálním servlet kontejneru implementovány a sloužily ke spuštění více webových aplikací v servlet kontejneru. Server Jenkins CI má omezení, že může současně běžet na jednom počítači pouze jeho jediná instance. Jelikož tento servlet kontejner je určen pouze pro Jenkins CI, tak jejich implementace nebyla smysluplná, protože by tyto parametry nemohly být využity pro jeho spuštění.

- Parametr `handlerCountMaxIdle` umožňoval určit maximální počet vláken servlet kontejneru, které aktuálně neprováděly žádnou činnost. V původní implementaci byla vlákna pro práci servlet kontejneru inicializována ručně instance rozhraní `ExecutorService` ze standardní knihovny. Jelikož tato možnost v serveru Undertow nebyla, tak byla správa vláken ponechána ve výchozím stavu, protože tento parametr nemá zásadní vliv na zachování zpětné kopatibility.

- Parametry `debug` a `logThrowingThread`, které pouze upravovaly vypisování průběhu běhu aplikace, ale nemají žádný vliv na jeho funkčnost. Jejich implementace může být předmětem dalšího vývoje.

Při implementaci dvou parametrů `httpKeepAliveTimeout`, `httpsKeepAliveTimeout`, které slouží k určení doby uchovávání aktivních spojení při komunikaci (hodnota *keep-alive*), nebylo možné jednoduchým způsobem nastavit chování serveru jako u původní verze. U původní verze bylo možné provést nastavení zvlášť pro protokol HTTP i HTTPS, ale při standardním využití serveru Undertow lze nastavit tuto hodnotu pouze pro celý server a tedy společnou pro oba protokoly.

Existuje možnost nevyužít veřejné API pro sestavení serveru a provést podstatně komplikovanější variantu jeho inicializace při které by bylo možné nastavit tyto hodnoty [8]. Tento přístup by byl překážkou při dalším vývoji servlet kontejneru, protože by bylo nutné tento kód upravovat s přicházejícími novými verzemi Undertow (např. při přidání podpory SPDY). Z těchto důvodů bylo zvoleno kompromisní řešení, kdy je možné zadat hodnoty těchto parametrů, ale jsou využity pro celý server. Pokud jsou zadány oba parametry, tak je jako prioritní zvolen parametr pro protokol HTTP. Tato úprava by neměla negativně ovlivnit chování serveru a pokud by se servlet kontejner ustálil, tak je možné provést zmíněné ruční sestavení serveru a poskytnout podporu obou těchto parametrů.

Kromě zde uvedených parametrů byly ostatní parametry původního servlet kontejneru zachovány.

4.5 Spuštění a testování

Nová implementace servlet kontejneru byla vyvíjena jako samostatný modul a její přeložení a uložení do lokálního Maven repozitáře provede následující příkaz (volaný z adresáře projektu Undertow4Jenkins):

```
mvn clean install -DskipTests
```

Po přeložení nového servlet kontejneru je následně nutné stejným příkazem přeložit upravenou verzi modulu `extras-executable-war`. Jakmile jsou moduly připravené v lokálním Maven repozitáři, tak již lze aplikaci Jenkins CI spustit standardním postupem, který je popsán v kapitole 2.1.4.

Při vývoji aplikace bylo vytvořeno 6 integračních testů, které testují základní funkčnosti vytvořeného servlet kontejneru. Nejedná se o žádnou kompletní sadu testů, ale sloužily pro otestování základní funkčnosti nezbytných komponent servlet kontejneru jako jsou komunikace pomocí HTTP, HTTPS, správa přístupových údajů, aj. Tyto testy byly inspirovány testy v původním servlet kontejneru. Pro jejich spuštění stačí při překladu vynechat poslední parametr `-DskipTests`.

Při testování výsledného programu bylo zváženo využití integračních testů celého serveru Jenkins CI. Bohužel tyto testy nevyužívají vestavěný servlet kontejner, takže jejich využití

pro prokázání funkčnosti aplikace nebylo možné. Kromě spouštění vlastních vytvořených testů byla aplikace opakovaně testována používáním jejich různých součástí v Jenkins CI a nebyly zjištěné žádné nefunkční prvky.

Kapitola 5

Srovnání výkonu původní a nové implementace

V této kapitole je porovnána z hlediska výkonu upravená varianta Jenkins CI s jeho původní implementací. Kapitoly 5.1 a 5.2 se zabývají samotným testování výkonností obou implementací. Výsledky testování jsou shrnuty v kapitole 5.3 a na závěr této části jsou diskutovány přínosy tohoto projektu pro komunitu Jenkins CI 5.4.

5.1 Podmínky pro testování

Testování probíhalo na dvou serverech, které jsou používány pro běh Jenkins CI ve firmě RedHat, ale v průběhu testování byly dané instance Jenkins CI vypnuté. Oba servery se nacházejí ve stejné laboratoři, což umožnilo minimalizovat ovlivnění výsledků zpožděním na síti. Testování nemohlo probhat pouze na jednom serveru, protože by se instance aplikace a testovací skripty navzájem ovlivňovaly.

V průběhu testování na jedno serveru běžela testovaná aplikace Jenkins CI a na druhém běžely testovací skripty, které zasílaly požadavky na server a měřily zpoždění odpovědí. Před každou iterací testování byl server vždy vypnut a znovu spuštěn, aby se jednotlivé iterace navzájem neovlivňovaly.

K testování aplikace byl využit framework *PerfCake 2.0*[1]. Tento framework je jednoduchým *open source* testovacím nástrojem ovládaným z příkazové řádky, který poskytuje podporu pro testování aplikací. Způsob testování aplikace se definuje pomocí souboru XML [6]. Pro potřeby této práce byly definovány tři testovací případy, jejichž popisem i vyhodnocením se zabývá následující kapitola.

Výsledky, které byly naměřené v rpůběhu testování, jsou uloženy na přiloženém DVD.

5.2 Testované případy

Byly formulovány tři scénáře pro otestování aplikace, jejichž cílem bylo testovat různorodé činnosti serveru. Jako protokol pro komunikaci se serverem byl zvolen protokol HTTP, jelikož je to výchozí protokol v Jenkins CI. Typy požadavků `DELETE` a `PUT` nejsou v aplikaci Jenkins CI používány (i mazání se děje požadavkem `POST`[2]), proto nebyly testovány tyto

[1] Stránky projektu *PerfCake*: `https://www.perfcake.org/`

[2] Mazání úlohy (angl. *job*) je popsáno v REST API aplikace, které se zobrazí po zadání adresy: `<adresa_serveru>/job/<nazev_ulohy>/API`

typy požadavků, ale pouze typy `GET` (2 scénáře) a `POST` (1 scénář).

Jednotlivé testované scénáře jsou popsány v následujících kapitolách. Jejich zápis pro framework PerfCake je umístěn v příloze C. Obě varianty aplikace byly vždy testovány za stejných podmínek a následně byly naměřené údaje analyzovány a vyhodnoceny.

5.2.1 Popis způsobu vyhodnocování výsledků

Při testování a vyhodnocení výsledků byly sledovány především průměrná odezva serveru na požadavky a doba za jakou se rychlost odpovědi ustálí pod určitou hranici, která sloužila jako pomocný ukazatel.

Doba odpovědi serveru na požadavek postupně rapidně klesá především z těchto důvodů:

- Na počátku běhu serveru je aplikace brzděna inicializacema svých jednotlivých částí
- Po prvním načtení dat z disku jsou data ukládána do vyrovnávacích pamětí
- V průběhu své činnosti je aplikace optimalizována interpretem jazyka Java, který dokáže znatelně ovlivnit její výkonnost

Samotný server Jenkins CI na počátku provádí asynchronně různé činnosti (např. zjišťování aktualizací), což může ovlivnit především počátek testování. Při testování byla tendence spustit testovací skript okamžitě po spuštění, ale jelikož neběžel na stejném serveru jako testovaná aplikace, tak nebylo možné dosáhnou stejného okamžiku.

Z těchto důvodů byla zkoumána především průměrná odezva serveru v dlouhém časovém úseku, kdy byly na server odeslány až desítky miliónů dotazů.

Druhým zkoumaným kritériem, byla doba, kdy dojde k ustálení doby odpovědi serveru (v tabulkách dále označován jako *Stabilní stav*). Tento stav byl definován jako okamžik, kdy je v aktuálním stavu pokles průměrné doby odezvy serveru menší než $0,5\%$ (definováno ve vzorci 5.1) oproti třem předcházejícím naměřeným hodnotám (období 3 sekund).

$$\frac{\mid predchozi\_hodnota - aktualni\_hodnota \mid}{predchozi\_hodnota} < 0,005 \tag{5.1}$$

Při analýze naměřených hodnot byla počítána směrodatná odchylka hodnot, která je definováná v rovnici 5.2 a aritmetický průměr souboru hodnot definovaný v rovnici 5.3.

$$s = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \overline{x})^2} \tag{5.2}$$

$$\overline{x} = \frac{1}{N}\sum_{i=1}^{N} x_i \tag{5.3}$$

V každém scénáři byly porovnány průměrné doby odezvy původní a nové varianty serveru Jenkins CI a bylo spočítáno zrychlení aplikace (případně zpomalení) pomocí vzorce 5.4.

$$\left(\frac{Prumerna\_rychlost\_puvodni\_implementace}{Prumerna\_rychlost\_nove\_implementace} - 1\right) * 100 \quad [\%] \tag{5.4}$$

5.2.2 Zobrazení hlavní stránky

V tomto scénáři byly pomocí metody `GET` posílány požadavky na zobrazení hlavní stránky serveru Jenkins CI. Testovací skript posílal požadavky pomocí **padesáti** vláken po dobu **osmi** minut. Pro každou variantu serveru bylo uskutečněno **10 běhů**, jejichž zpracované výsledky jsou zobrazeny v tabulce 5.1.

Běh	**Původní verze**		**Nová verze**	
	Aritmetický průměr odezvy [ms]	**Stabilní stav** [s]	**Aritmetický průměr odezvy** [s]	**Stabilní stav** [s]
1	28.797	124	26.595	119
2	29.754	151	26.577	116
3	29.764	147	27.921	113
4	31.513	159	26.202	122
5	29.652	104	26.809	119
6	30.989	112	26.509	166
7	29.668	100	26.453	159
8	29.839	92	27.343	114
9	29.323	163	26.528	119
10	29.839	143	26.786	119

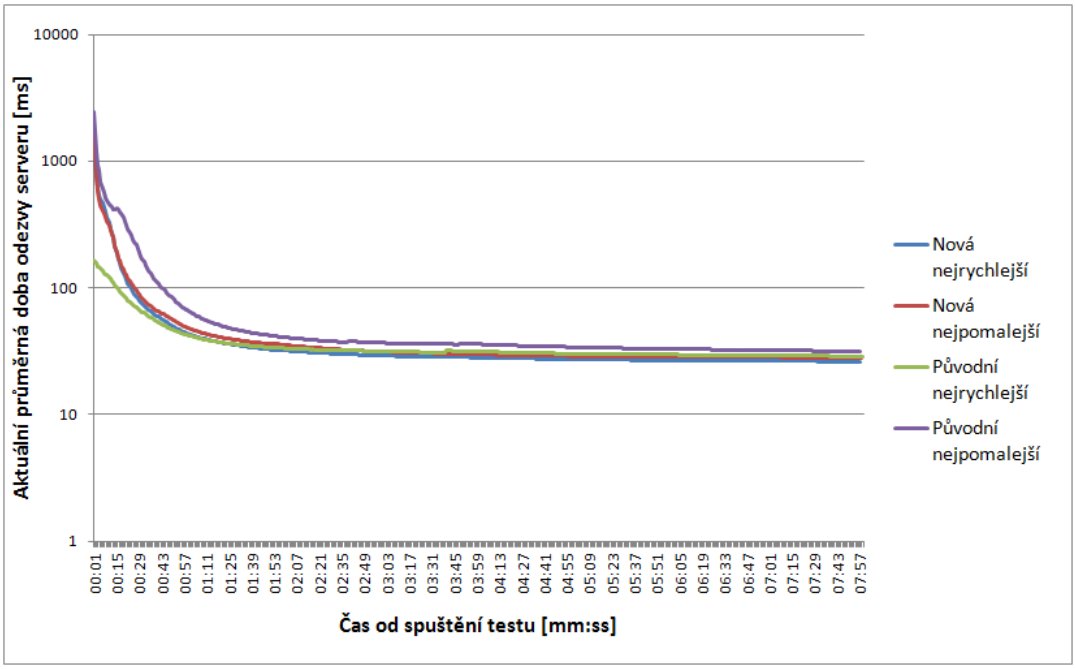
Tabulka 5.1: Tabulka zobrazující aritmetický průměr doby odezvy serverů a čas, za který se změna průměrné doby odezvy ustálí (definováno v kapitole 5.2.1)

	Doba odezvy		**Stabilní stav**	
	Aritmetický průměr [ms]	**Směrodatná odchylka** [ms]	**Aritmetický průměr** [s]	**Směrodatná odchylka** [s]
Původní verze	29.914	0.741	129.5	24.913
Nová verze	26.772	0.477	126.6	18.195
Rozdíl	**3.141**	**0.265**	**2.9**	**6.718**

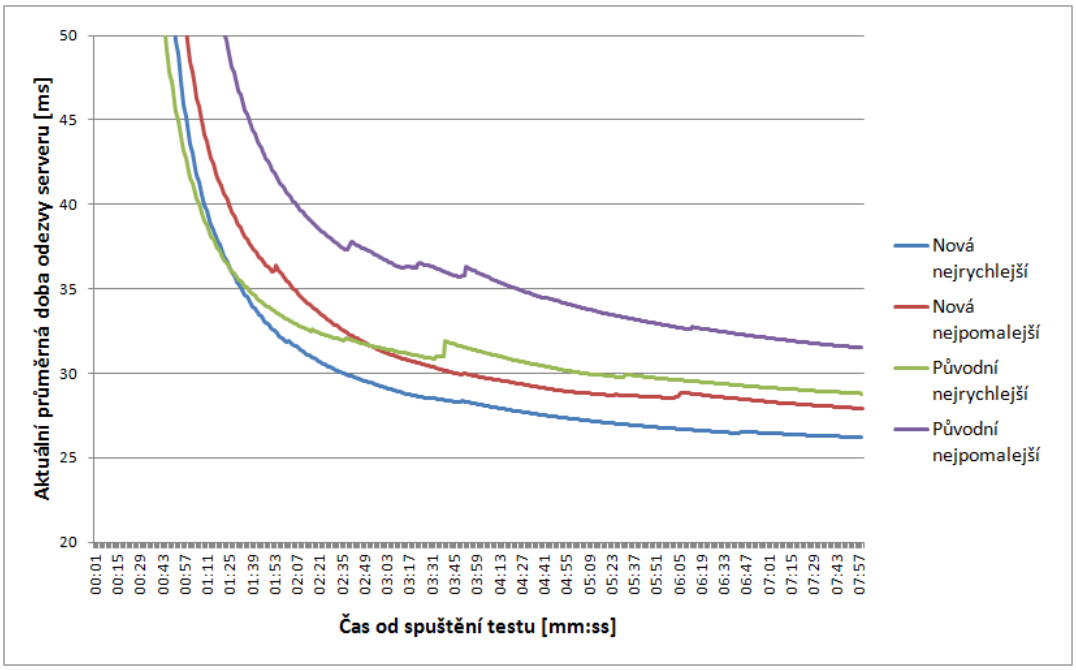
Tabulka 5.2: Tabulka shrnující výsledky měřeného experimentu, kdy byla zobrazována hlavní stránka Jenkins CI. Naměřené hodnoty jsou zobrazeny v tabulce 5.1

Průběh průměrné doby odezvy serverů je zobrazen v grafu 5.1 a v grafu 5.2 je zachycen detail pozdější fáze testu, kdy se už průměrná doba odezvy téměř neměnila. Z důvodu lepší čitelnosti grafů jsou vyneseny pouze ty experimenty, kdy byly naměřené nejlepší a nejhorší hodnoty pro danou variantu serveru. Pro vyhodnocení byly z naměřených dat vypočítány hodnoty aritmetického průměru a směrodatné odchylky, které shrnují naměřená data v experimentech (tabulka 5.2).

Na počátku testů byla odezva serveru vždy vysoká, ale v průběhu docházelo k rapidnímu poklesu. Tento pokles a stabilní stav nastával u nové varianty serveru mírně dříve, ale ze směrodatné odchylky vidíme, že jeho okamžik se příliš nelišil. Hlavním zjištěním testu je pokles průměrné doby odezvy serveru Jenkins CI o asi 3.141 milisekund, což představuje zrychlení celé aplikace v průměru o **11.73%** (dle vztahu pro výpočet 5.4). Z pohledu stability jednotlivých spuštění měla nová varianta Jenkins CI výrazně sníženou směrodatnou odchylku průměrné doby odezvy, což naznačuje konstantnější výkonnost celé aplikace.



Obrázek 5.1: Průběh naměřených hodnot při testu získávání hlavní stránky projektu. Graf zobrazuje nejlepší a nejhorší naměřené případ pro každý server. Na svislé ose je použito logaritmické měřítko.



Obrázek 5.2: Průběh naměřených hodnot při testu získávání hlavní stránky projektu. Graf zobrazuje nejlepší a nejhorší naměřené případ pro každý server.

Tento fakt můžeme pozorovat i na grafu 5.2, kde je křivka běhů nové implementace také stabilnější.

Z pohledu zrychlení je důležité zdůraznit, že se jedná o zrychlení celé aplikace, takže zrychlení samotného servlet kontejneru bude oproti původní variantě ještě vyšší než naměřených 11.73%.

5.2.3 Získání konfigurace úlohy

V tomto testu byly pomocí metody `GET` posílány požadavky na získání konfigurace jedné vytvořené úlohy na serveru (tzv. *freestyle job*). Testovací skript posílal požadavky pomocí **padesáti** vláken po dobu **osmi** minut. Pro každou variantu serveru bylo uskutečněno **10 běhů**.

Výsledky tohoto testu jsou zobrazeny v grafech 5.3 a 5.4. Z naměřených dat vypočítány hodnoty aritmetického průměru a směrodatné odchylky, které shrnují naměřená data v experimentech (tabulka 5.4). +Jelikož na počátku testů nedocházelo vždy k poklesům hodnot okamžitě zřejmě z důvodu náročnosti operace, tak pro určení stabilního stavu (vzorec 5.1) byla přidána podmínka, že průměrná hodnota doby odezvy musí být nižší než 4s.

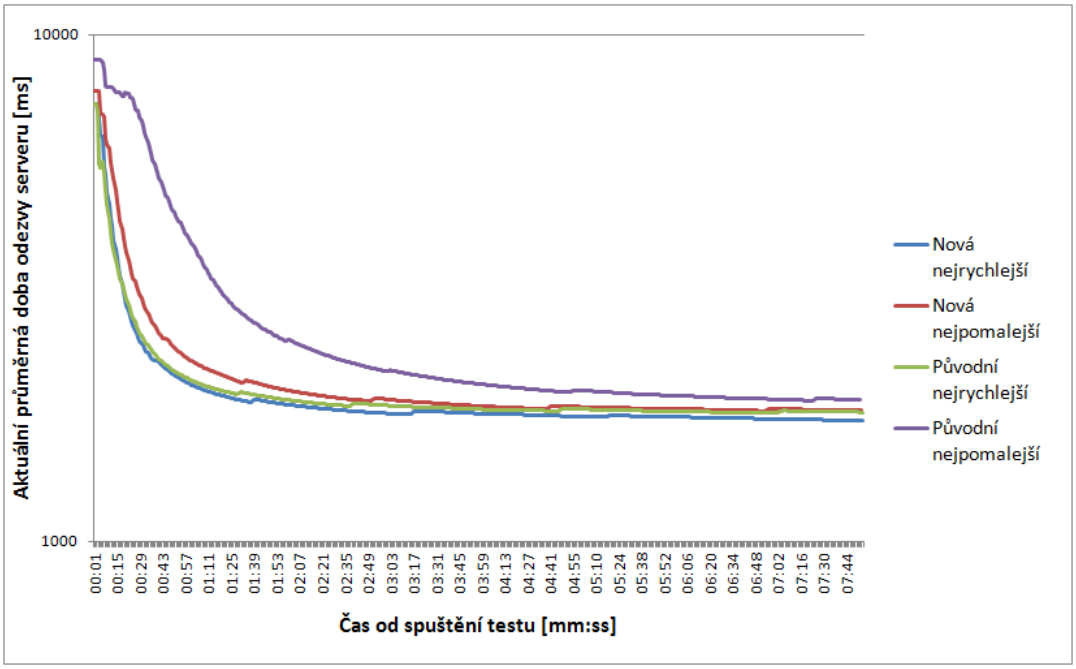
Běh	**Původní verze**		**Nová verze**	
	Aritmetický průměr odezvy [ms]	**Stabilní stav** [s]	**Aritmetický průměr odezvy** [s]	**Stabilní stav** [s]
1	1834.752	112	1780.747	102
2	1821.016	100	1753.001	96
3	1797.174	89	1763.485	93
4	1800.974	88	1734.458	92
5	1833.579	104	1756.754	101
6	1833.173	105	1758.102	100
7	1876.958	109	1771.095	85
8	1898.825	128	1763.257	100
9	1838.782	108	1810.809	100
10	1811.317	104	1789.687	104

Tabulka 5.3: Tabulka zobrazující výsledky jednotlivých běhů experimentu, kdy byla získávána konfigurace úlohy.

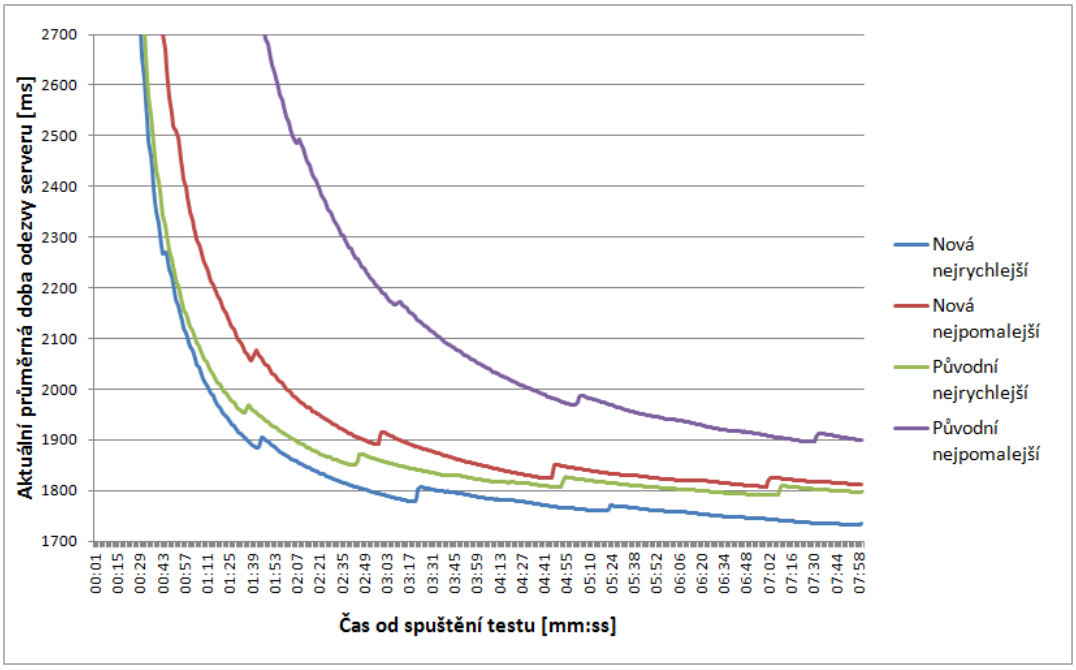
	Doba odezvy		**Stabilní stav**	
	Aritmetický průměr [ms]	**Směrodatná odchylka** [ms]	**Aritmetický průměr** [s]	**Směrodatná odchylka** [s]
Původní verze	1834.655	30.328	104.700	10.836
Nová verze	1768.139	20.229	97.300	5.496
Rozdíl	66.516	10.098	7.400	5.339

Tabulka 5.4: Tabulka shrnující výsledky měřeného experimentu, kdy byla získávána konfigurace úlohy.

V průběhu tohoto testu byl opět zaznamenán velký pokles doby odezvy na začátků experimentů. Z grafů vidíme, že k ustálení stavu v nové variantě docházelo v nejlepším případě v obdobné době. Naopak z grafův i hodnot směrodatné odchylky vidíme, že v nejhorší variantě byla doba ustálení u původní implementace Jenkins CI horší.



Obrázek 5.3: Průběh naměřených hodnot při testu získávání konfigurace úlohy ze serveru. Graf zobrazuje nejlepší a nejhorší naměřené případ pro každý server. Na svislé ose je použito logaritmické měřítko.



Obrázek 5.4: Průběh naměřených hodnot při testu získávání konfigurace úlohy ze serveru. Graf zobrazuje nejlepší a nejhorší naměřené případ pro každý server.

Z pohledu naměřených hodnot průměrné doby odezvy byl zaznamenán pokles u nové varianty asi o 66.516 milisekund, což činí zrychlení o **3.76%**. Na grafu 5.4 lze vidět, že

v obou variantách docházelo k mírným záchvěvům v průběhu testů. Tyto stavy byly zřejmě způsobeny tím, že požadavků na server docházelo velké množství a požadovaný úkon je výpočetně náročný. Nicméně z pohledu rozptylu naměřených hodnot mezi jednotlivými běhy poskytovala nová varianta Jenkins CI lepší hodnoty, což svědčí o větší stabilitě.

5.2.4 Vytvoření nové úlohy

V tomto testu byly pomocí metody `POST` posílány požadavky na vytvoření nové úlohy na serveru (tzv. *freestyle job*). Tento požadavek byl prováděn pomocí *REST API* serveru Jenkins CI. Testovací skript posílal požadavky pomocí **dvou** vláken po dobu **jedné** minuty. Pro každou variantu serveru bylo uskutečněno **10 běhů**.

Jelikož v průběhu testování tohoto scénáře na serveru vznikalo velmi velké množství dat v Jenkins CI (řádově stovky tisíc), tak musela být stanovena nižší doba testování i počet serverů, které posílají požadavky. Navíc před každým spuštěním byla smazána domovská složka aplikace. Pokud by smazání neproběhlo, tak by mohly být ovlivňovány pozdější běhy testování narůstajícím množstvím dat.

Výsledky tohoto testu jsou zobrazeny v grafech 5.5 a 5.6. V tabulce 5.5 jsou uvedené vypočítané statistické hodnoty z naměřených dat.

	Doba odezvy		**Stabilní stav**	
	Aritmetický průměr [ms]	**Směrodatná odchylka** [ms]	**Aritmetický průměr** [s]	**Směrodatná odchylka** [s]
Původní verze	19.644	0.381	10.900	2.914
Nová verze	19.665	0.282	15.900	1.814
Rozdíl	−0.021	0.099	−5.000	1.100

Tabulka 5.5: Tabulka shrnující výsledky měřeného experimentu, kdy byla vytvářena nová úloha.

Na počátku experimentů došlo opět k výraznému poklesu doby odezvy, ale oproti zbylým dvou případům se rychleji do stabilního stavu dostávala původní implementace Jenkins CI. Tento trend vidíme jak na grafu 5.6, tak z průměrných hodnot stabilního stavu.

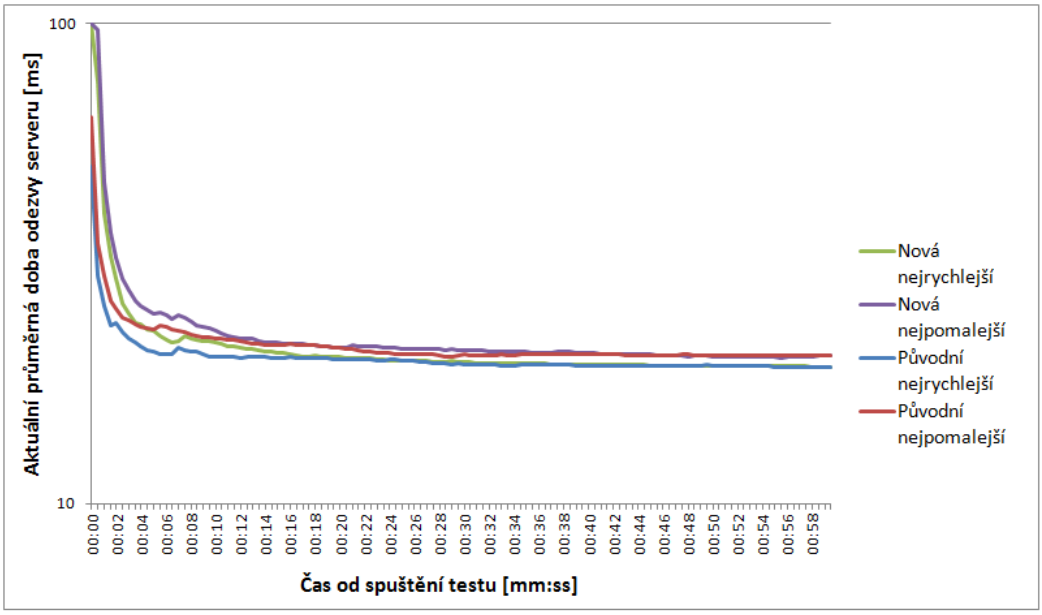
V tomto testovaném případě se nepotvrdilo zrychlení nové implementace Jenkins CI, které bylo naměřeno v předchozích testovaných případech. Bylo naměřeno mírné zpomalení o -0.021 milisekund, což je vyjádřeno procenty **0.11%**. Tato hodnota je velmi nízká a můžeme tedy prohlásit, že v tomto experimentu byly z pohledu doby odezvy obě varianty rovnocené. Podobnost krajních hodnoty průměrné doba odezvy je také vidět na grafu 5.6.

5.3 Zhodnocení výsledků testování

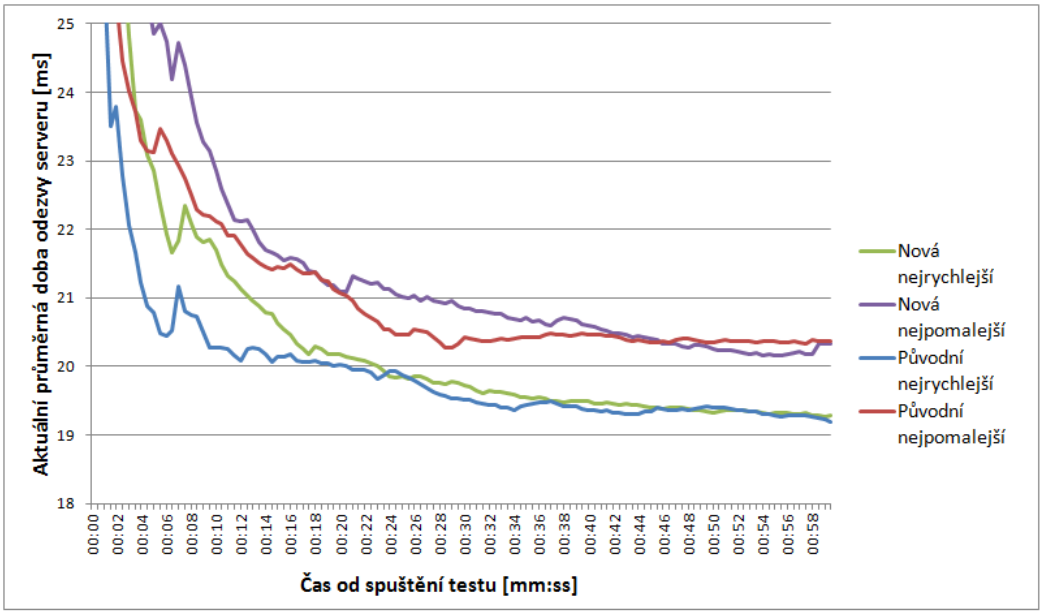
Ve třech testovacích případech byly otestovány původní implementace Jenkins CI a implementace s novým vestavěným servlet kontejnerem.

V prvních dvou případech dosáhla nová implementace lepších výsledků výkonnosti z pohledu doby odezvy aplikace. V prvním případě zrychlení činilo téměř 12% a ve druhém případě téměř 4%. V těchto dvou případech u nové varianty docházelo ke stabilnějším výsledkům mezi měřeními. O trochu dříve docházelo také k ustálení odpovědí serveru, ale zlepšení nebylo výrazné.

V posledním testovaném případě nebylo prokázáno zrychlení, ale mírné zpomalení o 0.1%. Nicméně tato hodnota je velmi nízká a lze prohlásit, že byly obě implementace v tomto tes-



Obrázek 5.5: Průběh naměřených hodnot při testu vytváření nové úlohy na serveru. Graf zobrazuje nejlepší a nejhorší naměřené případ pro každý server. Na svislé ose je použito logaritmické měřítko.



Obrázek 5.6: Průběh naměřených hodnot při testu vytváření nové úlohy na serveru. Graf zobrazuje nejlepší a nejhorší naměřené případ pro každý server.

tovaném případě stejně rychlé. Z pohledu rychlosti ustálení serveru byla v tomto případě mírně lepší původní implementace Jenkins CI.

V provedených měřeních byla tedy implementace Jenkins CI s novým servlet kontejnerem rychlejší než původní verze. Průměrné zrychlení aplikace činilo asi 5.1% (počítáno

jako aritmetický průměr ze tří hodnot zrychlení). Nová implementace také poskytovala stabilnější hodnoty mezi jednotlivými měřeními.

Rychlost servlet kontejneru ovlivňuje všechny testované činnosti, ale podstatná část zpoždění je také ovlivněna samotnou implementací webové aplikace. Tedy je zřejmé, že zrychlení samotného servlet kontejneru bylo v daných případech vyšší než je výsledek těchto měření pro celou aplikaci.

5.4 Zhodnocení přínosu pro komunitu Jenkins CI

Cíl vytvořit servlet kontejneru založený na serveru Undertow, který bude funkčně kompatibiltní se stávajícím servlet kontejnerem, se podařilo naplnit. Tento vytvořený servlet kontejner bude v následujících dnech představen komunitě Jenkins CI.

Vytvořená nová implementace servlet kontejneru může pomoci nahradit zastaralou implementaci stávajícího servlet kontejneru. Kód původní implementace již značně degeneroval a je obtížně čitelný, což by tato nová verze servlet kontejneru pomohla vyřešit.

Hlavním možným přínosem tohoto servlet kontejneru je jeho vyšší rychlost a výkonnost, která se v testech prokázala, ale určitě by bylo vhodné ještě pokračovat v dalším testování.

Pro svůj běh Jenkins CI využívá verzi jazyka Java 6 a ještě donedávna udržoval kompatibilitu s verzí Java 5. Komunita velmi dbá na udržování zpětné kompatibility a nedá se tedy předpokládat, že v brzké době proběhne změna využívané verze jazyka. Problémem stávajícího servlet kontejneru je, že nejnovější verze serveru Jetty odpovídá verzi Java 7. Z tohoto důvodu je aktuálně v Jenkins CI používána starší verze serveru Jetty a nebude možné přistoupit k jeho nahrazení verzí novější. Naopak server Undertow a servlet kontejner vytvořený v rámci této práce je kompatibilní s verzí Java 6 a může být do budoucna dále vylepšován, což je jistá výhoda oproti stávajícímu stavu.

Pokud by se komunita rozhodla, že chce tento servlet kontejner začlenit do Jenkins CI, tak by bylo nutné provést ještě komplexnější testování nové implementace servlet kontejneru, ale nebyly zjištěny žádné problémy které by začlenění bránily.

Kapitola 6

Závěr

V této práci proběhlo seznámení se s integračním serverem Jenkins CI, se servery Jetty a Winstone, které jsou součástí jeho servlet kontejneru, a také se serverem Undertow, jehož integrace do Jenkins CI je hlavní náplní této práce. Architektura aplikace Jenkins CI byla důkladně analyzována se zaměřením na integrovaný servlet kontejner.

Po seznámení se s podmínkami pro integraci byly zkoumány možnosti provedení samotné integrace serveru Undertow do Jenkins CI. První možností bylo nahrazení pouze komponenty Jetty, která vykonává většinu práce v aktuálním kontejneru, zatímco druhou variantou bylo nahrazení obou součástí. Po důkladném zvážení různých aspektů integrace byla zvolena varianta nahrazení obou komponent současného servlet kontejneru, a tudíž vytvoření zcela nové implementace.

Dle návrhu byla vytvořena nová implementace servlet kontejneru pro Jenkins CI. Při implementaci byl kladen důraz na zpětnou kompatibilitu vytvářeného řešení s původní verzí. Povedlo se implementovat všechny podstatné funkce původní verze servlet kontejneru. Upravená aplikace byla testována a zkoumána a nebyly zjištěny žádné problémy při jejím běhu.

Výkon původní a upravené verze Jenkins CI byl otestován ve třech definovaných případech a bylo naměřeno průměrné zrychlení celé aplikace o 5,1%. Jelikož tato hodnota udává zrychlení celé aplikace, je zřejmé, že zrychlení samotného servlet kontejneru bylo ještě větší. Kromě vyšší rychlosti dosahovala nová implemetace také stabilnějšího výkonu, což je pro práci serveru podstatná informace.

Výsledek této práce má potenciál být dlouhodobě rychlejším a spolehlivějším řešením servlet kontejneru pro Jenkins CI než současná implementace a bude prezentován komunitě projektu. Jistou výhodou je také skutečnost, že implementace byla provedena nanovo a byl odstraněn přebytečný kód ze zastaralé původní varianty.

Stanovených cílů bylo v rámci této diplomové práce dosaženo. Předmětem pro následující vývoj projektu je přidání podpory komunikace pomocí protokolu SPDY, která může být v blízké době dodělána po skončení beta testování nové verze serveru Undertow.

Literatura

[1] Kawaguchi, K.: Containers – Jenkins – Jenkins Wiki [online]. https://wiki.jenkins-ci.org/display/JENKINS/Containers, 2011-02-03 [cit. 2014-01-06].

[2] Kawaguchi, K.: Governanace Document – Jenkins – Jenkins Wiki [online]. https://wiki.jenkins-ci.org/display/JENKINS/Governance+Document, 2012-03-21 [cit. 2014-01-09].

[3] Kawaguchi, K.: Winstone is now powered by Jetty [online]. https://groups.google.com/forum/#!topic/jenkinsci-dev/R1FhPki9z4c, 2013-10-04 [cit. 2014-01-11].

[4] Knowles, R.: Winstone servlet container [online]. http://winstone.sourceforge.net/, cit. 2014-01-09.

[5] Knowles, R.: Winstone servlet container: Using Authentication Realms [online]. http://winstone.sourceforge.net/#authRealms, cit. 2014-01-12.

[6] Macík, P.: PerfCake 2.0: User Guide [online]. https://www.perfcake.org/docs/perfcake-user-guide.pdf, [cit. 2014-05-24].

[7] Milstein, D.: The Apache Tomcat Connector – AJP Protocol Reference [online]. http://tomcat.apache.org/connectors-doc/ajp/ajpv13a.html, cit. 2014-01-13.

[8] Bootstraping Undertow [online]. http://undertow.io/documentation/core/bootstrapping.html, [cit. 2014-05-21].

[9] Dokumentace serveru Undertow [online]. http://undertow.io/documentation/index.html, [cit. 2014-05-16].

[10] Undertow listeners [online]. http://undertow.io/documentation/core/listeners.html, [cit. 2014-05-17].

[11] Deployment [online]. http://undertow.io/documentation/servlet/deployment.html, [cit. 2014-05-18].

[12] Security [online]. http://undertow.io/documentation/core/security.html, [cit. 2014-05-18].

[13] Servlet security [online]. http://undertow.io/documentation/servlet/security.html, [cit. 2014-05-18].

[14] Prakash, W.: Hudson Security Architecture [online].
http://hudson-ci.org/docs/HudsonArch-Security.pdf, 2010 [cit. 2014-01-11].

[15] Prakash, W.: Hudson Web Architecture [online].
http://hudson-ci.org/docs/HudsonArch-Web.pdf, 2010 [cit. 2014-01-11].

[16] SPDY: An experimental protocol for a faster web [online].
http://www.chromium.org/spdy/spdy-whitepaper, cit. 2014-05-20.

[17] Web.xml Deployment Descriptor Elements [online].
http://docs.oracle.com/cd/E13222_01/wls/docs100/webapp/web_xml.html,
cit. 2014-05-20.

[18] Smart, J. F.: *Jenkins: The Definite Guide*. O'Reily Media, Inc., 2011, ISBN 978-1-449-30535-2.

[19] Wang, R.: What is a Servlet Container? [online].
http://java.dzone.com/articles/what-servlet-container, 2013-01-05
[cit. 2014-01-02].

[20] What is Stapler? [online]. http://stapler.kohsuke.org/what-is.html, 2013-11-15
[cit. 2014-01-13].

[21] Webové stránky projektu Jenkins CI [online]. http://jenkins-ci.org/.

[22] What is Maven? [online]. http://maven.apache.org/what-is-maven.html,
2014-05-08 [cit. 2014-05-15].

[23] Jetty – Servlet Engine and Http Server [online]. http://www.eclipse.org/jetty/,
2014 [cit. 2014-01-02].

[24] Webové stránky projektu Undertow [online]. http://undertow.io, [cit. 2014-05-16].

Příloha A

Obsah DVD

Přiložené DVD má tuto strukturu:

- /doc – vygenerovaná projektová dokumentace pro projekt *Undertow4Jenkins*
- /src/undertow4jenkins – zdrojové kódy vytvořeného servlet kontejneru
- /src/jenkins/ – zdrojové kódy Jenkins CI upravené pro potřeby integrace nového servlet kontejneru
- /war – upravená verze Jenkins CI v archivu `.war`
- /testResources – podklady, které byly využití při testování výkonnosti
- /testResults – výsledky naměřené při testování výkonnosti
- /thesis – text diplomové práce

Příloha B

Návod na přeložení a spuštění výsledné aplikace

Postup pro přeložení a spuštění aplikace se neliší od popisu v textu práce, ale pouze shrnuje uvedená fakta. Pro přeložení a spuštění Jenkins CI s novým servlet kontejnerem je potřeba učinit následující kroky:

1. TODO

Příloha C

Vytvořené scénáře pro testování

Pro potřeby testování byly vytvořeny 3 testovací scénáře pro framework PerfCake. Při tvorbě scénářů byly využity zdroje, které byly v minulosti vytvořené pro testování Jenkins CI[1]. Jedná se o zásuvný modul (angl. *plugin*) pro generování náhodných jmen a obsah zprávy pro vytvoření nové úlohy v Jenkins CI.

Pro některé URL adresy nebyl rozsah stránky dostatečný a proto byly rozděleny na dvě části, ale ve skutečnosti musí být spojeny. Vytvořené scénáře jsou také uložené na přiloženém DVD a jejich text je následující:

- Scénář pro testování zobrazení hlavní stránky:

```
<?xml version="1.0" encoding="utf-8"?>
<scenario xmlns="urn:perfcake:scenario:2.0">
  <generator class="DefaultMessageGenerator"
        threads="${perfcake.performance.thread.count:50}">
    <run type="time" value="${perfcake.performance.duration:480000}"/>
    <property name="threadQueueSize" value="50000"/>
  </generator>

  <sender class="HTTPSender">
    <property name="target"
        value="http://${server.url}:${server.port:8080}"/>
    <property name="method" value="GET"/>
  </sender>

  <reporting>
    <reporter class="ResponseTimeReporter">
      <destination class="CSVDestination">
        <period type="time" value="1000"/>
        <property name="path"
            value="perf-logs/getMainPage/response-time.csv"/>
      </destination>
    </reporter>
  </reporting>
</scenario>
```

[1] Adresa repozitáře projektu: `https://github.com/vjuranek/jenkins-perf-tests`

- Scénář pro testování vytvoření nové úlohy v Jenkins CI:

```
<?xml version="1.0" encoding="utf-8"?>
<scenario xmlns="urn:perfcake:scenario:2.0">
  <generator class="DefaultMessageGenerator"
        threads="${perfcake.performance.thread.count:2}">
    <run type="time" value="${perfcake.performance.duration:60000}"/>
    <property name="threadQueueSize" value="50000"/>
  </generator>

  <sender class="org.perfcake.plugins.jenkins_tools.RandomTargetSender">
    <property name="target"
        value="http://${server.url}:${server.port:8080}
            /createItem?name=$RANDOM"/>
    <property name="method" value="POST"/>
  </sender>

  <reporting>
    <reporter class="ResponseTimeReporter">
      <destination class="CSVDestination">
        <period type="time" value="500"/>
        <property name="path"
            value="perf-logs/createFreestyleJob/response-time.csv"/>
      </destination>
    </reporter>
  </reporting>

  <messages>
    <message uri="createFreestyle.xml"/>
  </messages>
</scenario>
```

- Scénář pro testování zobrazení konfigurace úlohy v Jenkins CI

```
<?xml version="1.0" encoding="utf-8"?>
<scenario xmlns="urn:perfcake:scenario:2.0">
  <generator class="DefaultMessageGenerator"
        threads="${perfcake.performance.thread.count:50}">
    <run type="time" value="${perfcake.performance.duration:480000}"/>
    <property name="threadQueueSize" value="50000"/>
  </generator>

  <sender class="HTTPSender">
    <property name="target"
        value="http://${server.url}:${server.port:8080}
            /job/testFreestyleJob/configure"/>
    <property name="method" value="GET"/>
  </sender>
```

```
  <reporting>
    <reporter class="ResponseTimeReporter">
      <destination class="CSVDestination">
        <period type="time" value="1000"/>
        <property name="path"
            value="perf-logs/getFreestyleJob/response-time.csv"/>
      </destination>
    </reporter>
  </reporting>
</scenario>
```