

# **Treinamento em Wildfly 16**

**Conceitos e abordagem prática**

**Zarb Solutions**

**Junho de 2019**

# Índice

|  |    |
|--|----|
| Conceitos e abordagem prática.....   | 1  |
| Introdução.....  | 4  |
| Java.....  | 4  |
| Java EE e Servidores de Aplicação.....   | 4  |
| JBoss EAP, JBoss AS e Wildfly.....   | 5  |
| JBoss AS.....  | 5  |
| JBoss EAP.....   | 6  |
| Wildfly.....   | 6  |
| Por que usar Wildfly?.....   | 6  |
| Infraestrutura do Curso.....   | 7  |
| Arquitetura e Componentes.....   | 7  |
| Configuração de nomes e endereços.....   | 7  |
| 1 Construção do ambiente JBoss e DevOps.....                                       | 7  |
| Ferramentas essenciais.....  | 7  |
| Criação de Usuários e configuração de permissões.....                              | 7  |
| Instalação de Java.....  | 8  |
| Instalação de Wildfly.....   | 9  |
| Instalando O WildFly como serviço em um Servidor Linux.....                        | 11 |
| Init.d.....  | 11 |
| Systemd.....   | 13 |
| Introdução ao Apache Maven 3 e o processo de build/empacotamento.....              | 14 |
| Instalando o Maven.....  | 16 |
| Descompactando.....  | 16 |
| Configurando.....  | 16 |
| Modos de Operação.....   | 17 |
| Modo Standalone.....   | 17 |
| Modo Domain.....   | 18 |
| Decidindo entre modo Standalone ou modo Domain.....                                | 18 |
| Opções de Gerenciamento.....   | 19 |
| Perfis.....  | 19 |
| Estrutura dos Diretórios.....  | 20 |
| Como está dividida a estrutura?.....   | 20 |
| Criando Usuário de Gerenciamento.....  | 22 |
| Dicas.....   | 25 |
| Criando usuário somente com um comando:.....                                       | 25 |
| Criando usuários com senhas fracas (não permitido através do script add-user)..... | 25 |
| Alterando a política de senhas.....  | 25 |
| Ferramentas de administração Wildfly.....  | 27 |
| Configurações Gerais.....  | 27 |
| Configurações modo Standalone.....   | 27 |
| Configurações modo Domínio.....  | 28 |
| Extension.....   | 28 |
| Subsystem.....   | 29 |
| Histórico de alterações.....   | 30 |
| Socket-bindings.....   | 30 |
| Como configurar.....   | 30 |
| Via Web Console.....   | 30 |
| Via CLI.....   | 33 |
| Definindo Port Offset em modo Domain.....  | 33 |
| Interfaces.....  | 33 |

|   |    |
|---|----|
| Deploy de aplicações.....   | 33 |
| Pacotes WAR, EAR, SAR.....  | 33 |
| Deploy em sistema de arquivos (Standalone).....                         | 34 |
| Deploy via plugin Maven.....  | 34 |
| Distribuindo sua aplicação no modo Domain.....                          | 35 |
| Serviço de Logging.....   | 35 |
| Administração JMS.....  | 36 |
| O que é JMS.....  | 36 |
| Configuração básica.....  | 37 |
| Transações e Persistência.....  | 38 |
| Utilizando driver JDBC como deployment.....                             | 39 |
| Instalando driver JDBC como módulo.....                                 | 39 |
| Tornando o Driver JDBC compatível com a versão 4.....                   | 40 |
| Segurança.....  | 40 |
| Introdução a JAAS.....  | 40 |
| SSL no wildfly.....   | 41 |
| Em ambiente de produção.....  | 42 |
| Subsistema Web.....   | 42 |
| WebSockets.....   | 42 |
| Alta Disponibilidade em Wildfly.....                                    | 43 |
| Configuração do Apache e NGInx.....                                     | 43 |
| Como escolher entre um e outro.....                                     | 43 |
| Balanceamento de carga entre servidores.....                            | 43 |
| Cluster e Domínio de servidores wildfly.....                            | 43 |
| Demonstração prática.....   | 43 |
| Monitoramento.....  | 43 |
| Profiling e Instrumentação.....   | 43 |
| Orientações gerais.....   | 43 |
| byteman.....  | 43 |
| jvisualvm, jstat.....   | 43 |
| Zorka, Zico.....  | 43 |
| Monitoramento.....  | 43 |
| Profiling.....  | 43 |
| 2 Otimização e Performance.....   | 44 |
| Cenários.....   | 44 |
| Orientações gerais, estratégias e dicas.....                            | 44 |
| Sequências de investigação e solução.....                               | 44 |
| Otimizações em infraestrutura Wildfly: threads, JVM, pools, perfis..... | 44 |
| Outros aspectos significantes para otimização.....                      | 44 |

# Introdução

## Java

**Java** é uma [linguagem de programação orientada a objetos](#) desenvolvida na [década de 90](#) por uma equipe de programadores chefiada por [James Gosling](#), na empresa [Sun Microsystems](#). Em 2008 o Java foi adquirido pela empresa [Oracle Corporation](#). Diferente das linguagens de programação modernas, que são [compiladas](#) para [código nativo](#), a linguagem Java é compilada para um [bytecode](#) que é interpretado por uma [máquina virtual](#) (Java Virtual Machine, mais conhecida pela sua abreviação JVM). A linguagem de programação Java é a linguagem convencional da [Plataforma Java](#), mas não é a sua única linguagem. [J2ME](#) Para programas e jogos de computador, celular, calculadoras, ou até mesmo o rádio do carro

## Java EE e Servidores de Aplicação

O **Java EE** (Java Enterprise Edition) consiste de uma série de especificações bem detalhadas, dando uma receita de como deve ser implementado um software que faz cada um desses serviços de infraestrutura.

Toda a especificação Java EE nos permite portabilidade de aplicação. Desenvolver um aplicativo sem se preocupar com a infra-estrutura ou o servidor de aplicação.

Atualmente estamos no Java EE 8 que compreende os seguintes componentes/especificações:

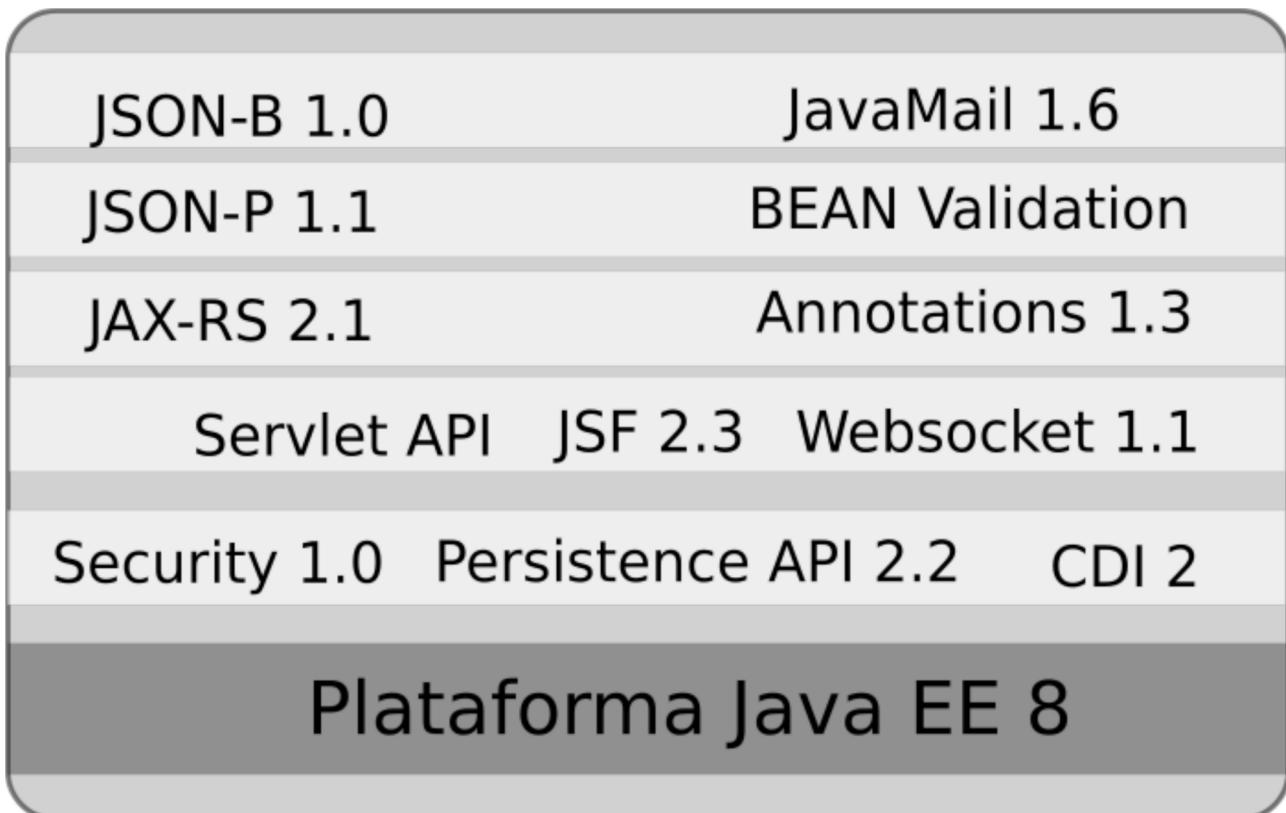
Java EE 8 continua melhorando e otimizando APIs e modelos de programação necessários para os dias atuais, além de adicionar features solicitadas pela comunidade de desenvolvimento. As principais melhorias:

- Java Servlet 4.0 API com suporte a HTTP/2
- suporte a Enhanced JSON incluindo uma nova API de JSON binding
- Novo REST Reactive Client API
- Eventos assíncronos CDI
- Nova Security API
- Suporte a eventos Server-Sent (Client & Server-side)
- Suporte às novidades de Java 8 (por exemplo Date & Time API, Streams API, Anotações )

Java EE 8 builds on Java EE 7. The following JSRs are new or updated in Java EE 8:

- [JSR 366](#) – Java EE 8 Platform
- [JSR 365](#) – Contexts and Dependency Injection (CDI) 2.0
- [JSR 367](#) – The Java API for JSON Binding (JSON-B) 1.0
- [JSR 369](#) – Java Servlet 4.0
- [JSR 370](#) – Java API for RESTful Web Services (JAX-RS) 2.1

- [JSR 372](#) – JavaServer Faces (JSF) 2.3
- [JSR 374](#) – Java API for JSON Processing (JSON-P)1.1
- [JSR 375](#) – Java EE Security API 1.0
- [JSR 380](#) – Bean Validation 2.0
- [JSR 250](#) – Common Annotations 1.3
- [JSR 338](#) – Java Persistence 2.2
- [JSR 356](#) – Java API for WebSocket 1.1
- [JSR 919](#) – JavaMail 1.6



*Figura 1: Plataforma Java Enterprise 8*

## **JBoss EAP, JBoss AS e Wildfly**

### **JBoss AS**

O JBoss AS (atualmente denominado WildFly) é um servidor de aplicações baseado em Java.

A grande vantagem de um servidor de aplicações é que os desenvolvedores podem se concentrar nas necessidades de negócio. Aspectos como conexões a bancos de dados, autenticação e gerenciamento de recursos são gerenciados pelo servidor de aplicações. Além disso, o padrão Java EE define padrões abertos que aceleram o desenvolvimento com uso de API padronizada e pensada para computação distribuída.

## **JBoss EAP**

É a versão comercial do JBoss sendo de desenvolvimento um pouco mais lento e voltado para ambientes empresariais que seguem padrões rígidos de estabilidade e segurança. Cada versão do JBoss EAP segue uma especificação JAVA EE específica e adiciona correções, melhorias de gerenciamento e alguns serviços solicitados por empresas.

## **Wildfly**

A partir do JBoss AS 7.1, nasceu o Wildfly 8 com funcionalidades melhoradas em ciclos menores de desenvolvimento e implementação de novas funcionalidades. O principal motivo da alteração ocorreu pelo fato da semelhança entre o *JBoss Application Server(AS)* e o *JBoss Enterprise Application Platform(EAP)* e também outros produtos do portfólio *JBoss* gerar um pouco de confusão entre eles.

Houve então uma votação na comunidade JBoss para decidir o novo nome do projeto da comunidade. Muitos nomes foram sugeridos (mais de 1500 nomes, de acordo com Mark Little, CTO da divisão de Middleware da Red Hat) e depois de uma pré-seleção cinco nomes foram selecionados para a votação da comunidade. Wildfly foi o nome mais votado entre vários outros, a troca de nomes foi oficialmente anunciada no *JUDCon Brazil* de 2013 em São Paulo e mais tarde Mark Little postou em seu blog o anúncio.

WildFly é líder neste segmento, oferecendo na sua versão da comunidade (open source) recursos avançados de clustering, segurança e integração. Esses recursos são ausentes em produtos concorrentes ou fornecidos apenas como complementos cobrados a parte. O JBoss AS ( WildFly) é utilizado pelo Banco Central, Caixa Econômica e Banco Votorantim.

## **Por que usar Wildfly?**

- **Prepare-se:** aderência a padrões abertos e código fonte garantem compatibilidade e minimizam riscos nos investimentos em TI, além de garantir estrategicamente o negócio.
- **Liberte-se:** o uso da versão Community permite independência total de fornecedor e escolha de todos os componentes de sua infraestrutura.
- **Atenda mais:** gerenciamento inteligente de conexões a bancos de dados e mecanismos avançados de aceleração de acesso a dados. Configuração e instalação de aplicações a quente minimizam indisponibilidades.
- **Agilize:** componentes prontos para segurança, acesso a dados, agendamento, enfileiramento, resiliência e outras funcionalidades que vão fazer a diferença entre protótipos e aplicações profissionais.
- **Simplifique:** aplicações dentro do **JBoss AS** podem chamar outras aplicações, em vez de integrações menos elegantes como acesso direto a banco de dados ou copiar e colar código para outras aplicações.

- **Cresça:** clusters de **JBoss AS** permitem escalar horizontalmente ou verticalmente para aproveitar toda a capacidade de hardware disponível e aumentar a capacidade de atendimento com alta disponibilidade.
- **Integre:** fácil integração com diretórios de rede, sistemas de gerência de identidade e mesmo biometria.
- **Acompanhe:** monitoramento da própria plataforma e uso e tempo de resposta de componentes da aplicação. O monitoramento de componentes da aplicação permite obter dados sobre comportamento dos usuários e indicadores de performance de negócios.

## Infraestrutura do Curso

Ambiente Linux Debian – Distribuição Ubuntu 19.04

O processo é semelhante em outras distribuições linux e também em servidores windows.

## Arquitetura e Componentes

Planejamento da arquitetura (servidor, cloud, cluster, domínio) de acordo com as necessidades.

Possibilidade de clientes em outras linguagens (PHP, Python, NodeJS) e dispositivos (Smartphones, Tablets, PWA)

## Configuração de nomes e endereços

## 1 Construção do ambiente JBoss e DevOps

### Ferramentas essenciais

Os requisitos básicos para se rodar o Wildfly são:

Arquitetura suportada pelo Java 64 bits

Sistemas operacionais modernos (Não tente rodar o Wildfly em computadores Macintosh ou 16 bits!)

É uma boa prática de mercado, a instalação de servidores de aplicação em ambiente Linux por questões de segurança, porém nada impede a instalação em ambiente Windows Server 2012 e superiores, bem como arquiteturas ARM como os famosos Raspberry PI 3.

### Criação de Usuários e configuração de permissões

Esse passo é importante principalmente em ambientes Linux. Segurança em primeiro lugar, evite ao máximo utilizar o usuário *root* para executar o WildFly porque desta forma estaremos protegendo o

servidor como um todo de forma que uma aplicação que permita execução de códigos arbitrários não execute nada no servidor com um usuário privilegiado.

Tome como uma boa prática de DevOps essa regra!

Neste exemplo, utilizarei um usuário chamado \_wildfly \_que será somente utilizado para executar o WildFly, para criar o usuário execute o seguinte comando:

```
# sudo groupadd wildfly  
# sudo useradd -r -g wildfly -d /opt/wildfly -s /sbin/nologin wildfly
```

## Instalação de Java

Para instalar o programa no Ubuntu e derivados e ainda poder receber automaticamente as futuras atualizações dele, execute o seguinte passo a passo:

Passo 1. Abra um terminal (use as teclas CTRL + ALT + T);

Passo 2. Se ainda não tiver, adicione o repositório do programa com este comando:

```
sudo add-apt-repository ppa:linuxuprising/java
```

Passo 3. Atualize o gerenciador de pacotes com o comando:

```
sudo apt-get update
```

Passo 4. Agora use o comando abaixo para instalar o programa;

```
sudo apt install oracle-java12-installer
```

### Definindo o Oracle Java 12 como padrão (ou não)

Este repositório traz consigo uma ferramenta distribuída na forma de um pacote muito útil para definir o Oracle Java 12 como a versão Java padrão do sistema.

Se você usa o Ubuntu, o pacote oracle-java12-set-default foi instalado como um pacote recomendado na instalação do pacote oracle-java12-installer e não é necessário fazer mais nada, exceto verificar a instalação.

```
java --version
```

Por outro lado, se você deseja instalar o Java 12, mas não o padrão. Então, você tem que remover o pacote oracle-java12-set-default.

```
sudo apt remove oracle-java12-set-default
```

## Instalação de Wildfly

Esta é realmente a primeira dúvida que temos ao realizar a instalação de qualquer aplicação em nossos servidores, com o WildFly não é diferente. Em poucas palavras, não existe um padrão definido que todos devem seguir, porém muitos administradores acabam escolhendo um determinado modelo a seguir para que todos os servidores possuam os mesmos diretórios/estrutura, isso que, facilita e muito a administração. Eu sempre costumo usar o diretório `/opt`, lembrando, isto não é uma regra.

Vamos agora descompactar o WildFly no diretório escolhido:

```
# tar -xzvf wildfly-16.0.0.Final.tar.gz --directory /opt
```

Ou se prefere utilizar o arquivo com extensão `.zip`:

```
unzip wildfly-16.0.0.Final.zip -d /opt
```

Para facilitar a administração do servidor de aplicação é interessante utilizar link simbólicos para facilitar uma atualização do WildFly, por exemplo, podemos criar um link simbólico do diretório `wildfly-16.0.0.Final` \_e chamá-lo somente de `wildfly`, e quando sugerir uma atualização de uma nova versão do WildFly basta somente atualizar o link simbólico, assim scripts de inicialização podem utilizar somente o diretório `/opt/wildfly`. Para efetuar esta configuração siga os passos abaixo:

```
# cd /opt
# ln -s wildfly-16.0.0.Final wildfly
```

Neste momento já temos o servidor WildFLy descompactado no diretório `/opt`:

```
# ll /opt
total 0
drwxr-xr-x. 10 root root 220 Jul 28 01:02 wildfly-16.0.0.Final
```

Note que as permissões de usuário e grupo estão configuradas para o usuário `root`, como boas práticas não iremos utilizar o usuário `root` para execução do WildFly, e sim o usuário criado anteriormente, altere as permissões com o seguinte comando:

```
# chown -R wildfly. /opt/wildfly-16.0.0.Final/
```

Agora podemos utilizar o usuário `wildfly` para executar o Servidor. Abra um shell utilizando este usuário e em seguida acesso o diretório `wildfly-16.0.0.Final`:

```
[root@wfly-server ~]$ cd /opt/wildfly-16.0.0.Final/
[root@wfly-server wildfly-16.0.0.Final]$ su -s "/bin/bash" -c "bin/standalone.sh" wildfly
```

Com os comandos executados acima estamos:

- logando com o usuário *wildfly*
- Acessando o *JBOSS\_HOME*
- Iniciando o Wildfly

```
JBoss Bootstrap Environment
JBoss_HOME: /opt/wildfly
JAVA: java

JAVA_OPTS: -server -Xms64m -Xmx512m -XX:MetaspaceSize=96M -XX:MaxMetaspaceSize=256M -Djava.net.preferIPv4Stack=true -Djboss.modules.system.pkgs=org.jboss.byteman -Djava.awt.headless=true --add-exports=java.base/sun.nio.ch=ALL-UNNAMED --add-exports=jdk.unsupported/sun.misc=ALL-UNNAMED --add-exports=jdk.unsupported/sun.reflect=ALL-UNNAMED --add-modules=java.se

=====
00:14:54,285 INFO [org.jboss.modules] (main) JBoss Modules version 1.9.0.Final
00:14:54,973 WARN [org.jboss.as.server] (main) WFLYSRV0266: Server home is set to '/opt/wildfly/standalone', but server real home is '/opt/wildfly-16.0.0.Final/standalone' - unpredictable results may occur.
00:14:54,980 INFO [org.jboss.msc] (main) JBoss MSC version 1.4.5.Final
00:14:54,993 INFO [org.jboss.threads] (main) JBoss Threads version 2.3.3.Final
00:14:55,116 INFO [org.jboss.as] (MSC service thread 1-2) WFLYSRV0049: WildFly Full 16.0.0.Final (WildFly Core 8.0.0.Final) starting
00:14:55,799 INFO [org.wildfly.security] (ServerService Thread Pool -- 22) ELY00001: WildFly Elytron version 1.8.0.Final
00:14:56,222 INFO [org.jboss.as.controller.management-deprecated] ((Controller Boot Thread) WFLYCTL0028: Attribute 'security-realm' in the resource at address '/core-service=management/management-interface=http-interface' is deprecated, and may be removed in a future version. See the attribute description in the output of the read-resource-description operation to learn more about the deprecation.
00:14:56,253 INFO [org.jboss.as.controller.management-deprecated] ((ServerService Thread Pool -- 35) WFLYCTL0028: Attribute 'security-realm' in the resource at address '/subsystem=undertow/server=default-server/https-listener=https' is deprecated, and may be removed in a future version. See the attribute description in the output of the read-resource-description operation to learn more about the deprecation.
00:14:56,311 INFO [org.jboss.as.server] (Controller Boot Thread) WFLYSRV0039: Creating http management service using socket-binding (management-http)
00:14:56,326 INFO [org.xnio] (MSC service thread 1-3) XNIO version 3.6.5.Final
00:14:56,334 INFO [org.xnio.nio] (MSC service thread 1-3) XNIO NIO Implementation Version 3.6.5.Final
00:14:56,400 INFO [org.jboss.remoting] (MSC service thread 1-1) JBoss Remoting version 5.0.8.Final
00:14:56,438 INFO [org.wildfly.extension.microprofile.config.smallrye_private] (ServerService Thread Pool -- 58) WFLYCONF0001: Activating WildFly MicroProfile Config subsystem
00:14:56,439 INFO [org.jboss.as.clustering.infinispan] (ServerService Thread Pool -- 49) WFLYCLINF0001: Activating Infinispan subsystem.
00:14:56,449 INFO [org.wildfly.extension.microprofile.opentracing] (ServerService Thread Pool -- 61) WFLYTRACEXT0001: Activating MicroProfile OpenTracing Subsystem
00:14:56,450 INFO [org.jboss.as.naming] (ServerService Thread Pool -- 62) WFLYNAM0001: Activating Naming Subsystem
00:14:56,431 INFO [org.jboss.as.jaxrs] (ServerService Thread Pool -- 51) WFLYRS0016: RESTEasy version 3.6.3.Final
00:14:56,431 INFO [org.jboss.as.connector] (MSC service thread 1-6) WFLYJCA0009: Starting JCA Subsystem (WildFly/IronJacamar 1.4.12.Final)
00:14:56,457 INFO [org.wildfly.extension.microprofile.metrics.smallrye] (ServerService Thread Pool -- 60) WFLYMETRICS0001: Activating Eclipse MicroProfile Metrics Subsystem
00:14:56,463 INFO [org.wildfly.extension.microprofile.health.smallrye] (ServerService Thread Pool -- 59) WFLYHEALTH0001: Activating Eclipse MicroProfile Health Subsystem
00:14:56,449 INFO [org.jboss.as.connector.subsystems.datasources] (ServerService Thread Pool -- 42) WFLYJCA0004: Deploying JDBC-compliant driver class org.h2.Driver (version 1.4)

00:14:57,002 INFO [org.jboss.as.server.deployment.scanner] (MSC service thread 1-2) WFLYDS0013: Started FileSystemDeploymentService for directory /opt/wildfly-16.0.0.Final/standalone/deployments
00:14:57,121 INFO [org.wildfly.extension.undertow] (MSC service thread 1-8) WFLYUT0006: Undertow HTTPS listener https listening on 127.0.0.1:8443
00:14:57,176 INFO [org.jboss.ws.common.management] (MSC service thread 1-1) JBossWS 5.2.4.Final (Apache CXF 3.2.7)
00:14:57,272 INFO [org.jboss.as.server] (Controller Boot Thread) WFLYSRV0212: Resuming server
00:14:57,274 INFO [org.jboss.as] (Controller Boot Thread) WFLYSRV0060: Http management interface listening on http://127.0.0.1:9990/management
00:14:57,275 INFO [org.jboss.as] (Controller Boot Thread) WFLYSRV0051: Admin console listening on http://127.0.0.1:9990
00:14:57,275 INFO [org.jboss.as] (Controller Boot Thread) WFLYSRV0025: WildFly Full 16.0.0.Final (WildFly Core 8.0.0.Final) started in 3656ms - Started 305 of 531 services (324 services are lazy, passive or on-demand)
```

Figura 2: Log de inicialização

Caso ocorra tudo bem durante a inicialização do WildFly você terá um log muito semelhante a este:

```
00:14:57,002 INFO [org.jboss.as.server.deployment.scanner] (MSC service thread 1-2) WFLYDS0013: Started FileSystemDeploymentService for directory /opt/wildfly-16.0.0.Final/standalone/deployments
00:14:57,121 INFO [org.wildfly.extension.undertow] (MSC service thread 1-8) WFLYUT0006: Undertow HTTPS listener https listening on 127.0.0.1:8443
00:14:57,176 INFO [org.jboss.ws.common.management] (MSC service thread 1-1) JBossWS 5.2.4.Final (Apache CXF 3.2.7)
00:14:57,272 INFO [org.jboss.as.server] (Controller Boot Thread) WFLYSRV0212: Resuming server
00:14:57,274 INFO [org.jboss.as] (Controller Boot Thread) WFLYSRV0060: Http management interface listening on http://127.0.0.1:9990/management
00:14:57,275 INFO [org.jboss.as] (Controller Boot Thread) WFLYSRV0051: Admin console listening on http://127.0.0.1:9990
00:14:57,275 INFO [org.jboss.as] (Controller Boot Thread) WFLYSRV0025: WildFly Full 16.0.0.Final (WildFly Core 8.0.0.Final) started in 3656ms - Started 305 of 531 services (324 services are lazy, passive or on-demand)
```

Figura 3: Linhas finais do log de inicialização

Abra o seu navegador e tente acessar o link <http://localhost:8080>. A seguinte tela será apresentada:



Figura 4: Tela inicial do JBoss após inicializar

Instalação testada, agora vamos deixar a configuração mais profissional. Até agora, vimos como instalar em uma máquina local para desenvolvimento e teste de aplicações. Vejamos como configurar como serviço a seguir.

## Instalando O WildFly como serviço em um Servidor Linux

Executar Servidores de Aplicação como serviço é muito comum nos dias de hoje, pela praticidade de gerenciamento e também por ser muito simples a restauração do serviço após o *boot* do Sistema Operacional.

No decorrer deste capítulo iremos descrever os passos necessários para configurar o WildFly Server como um serviço utilizando o **systemd** como gerenciador de serviços.

Nas versões anteriores do Wildfly, até as versão 9, os scripts estavam no diretório `$JBOSS_HOME/bin/init.d`, mas a partir da versão 10 os scripts foram movidos para o diretório `$JBOSS_HOME/docs/contrib/scripts`, lembrando que os scripts encontrados neste diretório são contribuições de usuários e estão disponíveis na distribuição do WildFly para serem utilizados como exemplo.

O a criação de um serviço com o **systemd** é muito simples quando comparado ao **init.d**. Para conhecer melhor o **systemd** acesse este [link](#).

Neste capítulo será exemplificado a instalação do serviço utilizando o **init.d** e o **systemd**.

### Init.d

Executar o WildFly como serviço utilizando o **init.d** é um tarefa simples, assim como utilizando o **systemd** que será abordado a seguir. Todos os scripts utilizando estão localizados no diretório `$JBOSS_HOME/docs/contrib/scripts/init.d/`. Dentro deste diretório podemos ver os seguintes arquivos:

- **wildfly-init-redhat.sh**: Arquivo a ser utilizado para distribuições Red Hat Enterprise Linux, como RHEL e Centos;
- **wildfly-init-debian.sh**: Arquivo a ser utilizado para distribuições Debian Linux, como o Debian e Ubuntu por exemplo;
- **wildfly.conf**: Por fim, o arquivo de configuração utilizado pelos dois arquivos init descritos acima.

O primeiro passo é copiar o o arquivo de script, de acordo com sua distribuição Linux, para o diretório `/etc/init.d`. Como utilizamos o Fedora, iremos executar o seguinte comando:

```
cp /opt/wildfly/docs/contrib/scripts/init.d/wildfly-init-redhat.sh /etc/init.d/wildfly
```

Agora precisamos copiar o arquivo de configuração **wildfly.conf** para onde o script espera que ele esteja, em `/etc/default` como podemos observar no script copiado anteriormente:

```
(...)
# Load JBoss AS init.d configuration.
if [ -z "$JBOSS_CONF" ]; then
    JBOSS_CONF="/etc/default/${WILDFLY_NAME}.conf"
fi
[ -r "$JBOSS_CONF" ] && . "$JBOSS_CONF"
(...)
```

Para copiar o arquivo, execute o seguinte comando:

```
mkdir -p /etc/default
cp /opt/wildfly/docs/contrib/scripts/init.d/wildfly.conf /etc/default
```

O próximo passo será editar o arquivo **wildfly.conf** previamente copiado com as seguintes informações:

```
#Location of Java
JAVA_HOME=/usr/java/jdk1.8.0_45

# Location of WildFly
JBOSS_HOME=/opt/wildfly/

# The username who should own the process.
JBOSS_USER=wildfly

# The mode WildFly should start, standalone or domain
JBOSS_MODE=standalone

# Configuration for standalone mode
JBOSS_CONFIG=standalone.xml
```

Note que é possível executar em modo domain, além de passar outros parâmetros, como:

```
## The amount of time to wait for startup
STARTUP_WAIT=60
```

```
## The amount of time to wait for shutdown
SHUTDOWN_WAIT=60
```

```
## Location to keep the console log
# JBOSS_CONSOLE_LOG="/var/log/wildfly/console.log"

## Additional args to include in startup
# JBOSS_OPTS="--admin-only -b 127.0.0.1"
```

Dando continuidade, o passo seguinte é instalar o wildfly como um serviço utilizando o comando **chkconfig**.

```
chkconfig --add wildfly
```

Em seguida, configurar o nível de inicialização onde o serviço será iniciado:

```
chkconfig --level 2345 wildfly on
```

Por fim vamos iniciar o serviço:

```
sudo service wildfly start
```

Caso tudo ocorra bem, a seguinte saída deve ser mostrada no console:

```
sudo service wildfly start
Starting wildfly (via systemctl): [ OK ]
```

Para encerrar o serviço, basta executar o comando:

```
sudo service wildfly stop
```

## Systemd

O arquivo de configuração do **systemd** disponibilizado na instalação do WildFly já está pronto para uso, sendo necessário realizar somente alguns passos antes de executá-lo como serviço.

Neste ponto, será necessário ter efetuado os passos descritos no tópico de instalação. Com o ambiente pré configurado, siga os passos abaixo:

```
# mkdir /etc/wildfly
# cp /opt/wildfly/docs/contrib/scripts/systemd/wildfly.conf /etc/wildfly/
# cp /opt/wildfly/docs/contrib/scripts/systemd/wildfly.service /etc/systemd/system/
# cp /opt/wildfly/docs/contrib/scripts/systemd/launch.sh /opt/wildfly/bin/
# chmod +x /opt/wildfly/bin/launch.sh
# chown wildfly. /opt/wildfly/bin/launch.sh
```

Como os scripts já estão pré configurados, depois de executar todos os passos acima, já estamos prontos para iniciar o wildfly.

Primeiramente vamos habilitá-lo para ser executado durante o boot do Sistema Operacional:

```
# systemctl enable wildfly  
# systemctl start wildfly
```

E para parar o serviço:

```
# systemctl stop wildfly
```

Para configurar o serviço do WildFly para usar o modo **Domain** ao invés do modo **Standalone** (padrão), edite o arquivo **wildfly.service** localizado em **/etc/wildfly** conforme o exemplo abaixo:

```
# The configuration you want to run  
WILDFLY_CONFIG=domain.xml  
  
# The mode you want to run  
WILDFLY_MODE=domain
```

Por padrão o endereço de bind vem configurado como **0.0.0.0**, não é uma boa prática deixá-lo com essa configuração, altere o endereço de bind para o endereço de IP do servidor em que o WildFly será executado. Para alterar o endereço de bind edite o mesmo arquivo citado acima da seguinte maneira, como exemplo será utilizado o ip **192.168.1.10**:

```
# The address to bind to  
WILDFLY_BIND=192.168.1.10
```

## Introdução ao Apache Maven 3 e o processo de build/empacotamento

Em seu cerne, **o Maven é uma ferramenta de gerenciamento e automação de construção (build) de projetos**. Entretanto, por fornecer diversas funcionalidades adicionais através do uso de plugins e estimular o emprego de melhores práticas de organização, desenvolvimento e manutenção de projetos, é muito mais do que apenas uma ferramenta auxiliar.

Um desenvolvedor que seja alocado em um projeto Java EE que **utilize o Maven corretamente** não terá que saber de imediato quais dependências (bibliotecas) o projeto necessita para compilar e executar, não precisará descobrir onde obtê-las e nem irá se preocupar em como realizar a construção do pacote do aplicativo. Com um comando simples, como mvn install, na raiz do código-fonte do projeto, instruirá o Maven a gerar o código extra necessário (cliente de um web service, por exemplo), validar e compilar o projeto, testá-lo através de seus testes unitários e gerar o pacote com o código compilado. Outras etapas poderiam incluir auditoria de qualidade de código, documentação, geração de estatísticas, entre diversas possibilidades.

**O Apache Maven é uma excelente ferramenta de apoio a qualquer equipe que trabalhe com projetos Java** (outras tecnologias também são suportadas), fornecendo aos desenvolvedores uma forma de automatizar e padronizar a construção e publicação de suas aplicações.

**O Apache Maven é uma ferramenta de automação e gerenciamento de projetos Java**, embora também possa ser utilizada com outras linguagens. Ela fornece às equipes de desenvolvimento uma forma padronizada de automação, construção e publicação de suas aplicações, agregando agilidade

e qualidade ao produto final. Por ser extremamente flexível, permite que sejam adicionados plugins a si, para estender suas funcionalidades nativas.

O processo de criação de um projeto [Java EE](#) em geral envolve a criação de um diretório principal com vários subdiretórios, a configuração de diversos arquivos XML, a obtenção (via cópia ou download) de bibliotecas para o projeto e, posteriormente, a execução dos testes unitários, a criação dos pacotes de publicação, a geração de documentação javadoc, entre outras etapas. Normalmente, até algum tempo atrás, cada projeto tinha sua própria estrutura, seu próprio jeito de gerar pacotes, de efetuar cada um destes passos. Projetos complexos, com vários módulos, ainda podem precisar que estes sejam compilados em determinada ordem, para que o pacote final seja criado.

Como se pode imaginar, facilmente isso poderia se tornar um pesadelo para os projetos em manutenção ou com um desenvolvimento mais extenso, em equipes que envolvam alguns desenvolvedores, porque as mudanças estruturais realizadas por um desenvolvedor têm que ser comunicadas a cada um dos demais colegas, para que o projeto continue a compilar no computador de cada um. Considere, por exemplo, que o desenvolvedor A precisou incluir uma biblioteca. Antes de colocar no controle de versão o código que a utiliza, ele precisaria avisar a cada um de seus colegas que adicionou algo novo no projeto, dizer de onde baixar o novo componente (ou passá-lo de computador em computador) e se assegurar que todos equalizaram seus ambientes de desenvolvimento, para que o código continue compilando nas máquinas de seus colegas. Tempos depois, o desenvolvedor B precisa efetuar uma manutenção no projeto e descobre que uma versão nova da biblioteca supracitada fornece uma funcionalidade nova, que facilitará o seu trabalho. Mais uma vez, o ciclo de aviso e cópia do novo componente começa, com perda de tempo da equipe e correndo o risco de algum computador ficar desatualizado e um desenvolvedor perder tempo tentando achar um erro acarretado pelo uso da biblioteca antiga.

Outra situação possível, em um projeto de vários módulos: um desenvolvedor modifica um dos módulos básicos do sistema e não avisa os demais colegas, ou estes não deram atenção ao e-mail de alerta. Como resultado o código ou para de compilar ou passa a ter comportamento instável, porque o módulo alterado não foi recompilado nos computadores do restante da equipe.

Agora, imagine os cenários acima em uma fábrica de software, em que o ritmo de trabalho é intenso e os prazos impõem pressão às equipes. Facilmente podemos prever que bibliotecas serão adicionadas sem que o restante do time seja avisado, acarretando impedimento de compilação do código mais recente do projeto. Pessoas modificariam módulos e se esqueceriam de avisar as demais para pegarem a versão mais recente de código e recompilar as alterações; mudanças no processo de geração de pacotes que não são comunicadas, e daí em diante.

Finalmente, se forem consideradas equipes distribuídas, com seus membros trabalhando em home office, a complexidade de propagar mudanças estruturais do projeto aumenta muito. Nesses casos, até mesmo iniciar o projeto podia trazer muitas dores de cabeça, até toda a equipe conseguir estabilizar o ambiente de desenvolvimento de cada participante.

Outra **característica do Maven** é estimular a adoção de boas práticas, porque uma das formas utilizadas por ele para reduzir o esforço de configuração do projeto é a utilização do conceito de *programação por convenção* (do inglês *convention over configuration*), em que a ferramenta assume que o seu usuário fará as coisas da forma como ela preconiza como ideais (estrutura de diretórios padrão, por exemplo), e o livra de ter que declarar algo que se repetirá em todo projeto. O incorporar as práticas aceitas pela comunidade Java como as mais indicadas para projetos Java EE,

o **Maven** acaba não só disseminando-as para novos desenvolvedores, como também as padroniza entre os projetos em que ele é empregado, permitindo que novatos se localizem muito mais rapidamente dentro de projetos novos. Obviamente, pode-se definir manualmente o que é assumido como padrão, ao preço do aumento na carga de trabalho para a configuração inicial do projeto.

## Instalando o Maven

A instalação do **Maven** é bastante simples.

O único pré-requisito para a instalação do Maven é ter o *Java Development Kit* (JDK) instalado, que pode ser obtido em [www.oracle.com/us/technologies/java/](http://www.oracle.com/us/technologies/java/) e que já foi previamente instalado.

Para realizar o download, clique [aqui](#) para visitar a página oficial. O software se encontra atualmente na versão 3.6.1, lançado neste mês de abril de 2019. Outra opção é abrir o terminal e executar o seguinte comando:

```
$ cd ~/Downloads/  
$ wget -c http://www-us.apache.org/dist/maven/maven-3/3.6.1/source/apache-maven-3.6.1-  
src.tar.gz
```

## Descompactando

Uma vez baixado o pacote, será necessário descompactá-lo. Para isso, navegue até o diretório onde se encontra o arquivo, nesse caso a pasta Download, e digite no terminal:

```
$ tar -zxvf apache-maven-3.6.1-src.tar.gz
```

**Lembre-se que para que o comando anterior seja bem sucedido, o programa tar deve estar instalado. O comando para instalação é:**

```
$ sudo apt install tar
```

Como forma de configuração mais elegante, é aconselhável encurtar o nome da pasta. Para isso faremos:

```
$ mv apache-maven-3.6.1-src.tar.gz maven
```

Agora precisamos mover a pasta para o diretório /opt, que na hierarquia do sistema de arquivo Linux, representa o lugar onde são guardados os programas que não fazem parte da instalação padrão do sistema. Com permissões de super usuário, faça:

```
sudo mv maven /opt/[simterm]
```

## Configurando

Por fim, temos que configurar a variável de ambiente PATH do sistema de forma que, ao digitarmos os comandos do Maven, o sistema encontre o arquivo de execução. Para que não seja exigido uma mesma configuração frequente, vamos salvar os comando em algum arquivo de inicialização do

sistema. acessando o arquivo /etc/profile devemos adicionar as seguintes linhas no final do documento:

```
$ sudo vim /etc/profile
```

#### Linha para adicionar:

```
export M3_HOME=/opt/maven  
export PATH=$M3_HOME/bin:$PATH
```

Após salvar o aquivo, basta encerrar a sessão para que a configuração seja salva. Para testar, execute o comando abaixo:

```
mvn -version
```

## Modos de Operação

O Wildfly tem dois modos de gerenciamento: *Standalone* e *Domain*. Cada modo permite que você gerencie seus servidores de modo diferente utilizando topologias diferentes.

### Modo Standalone

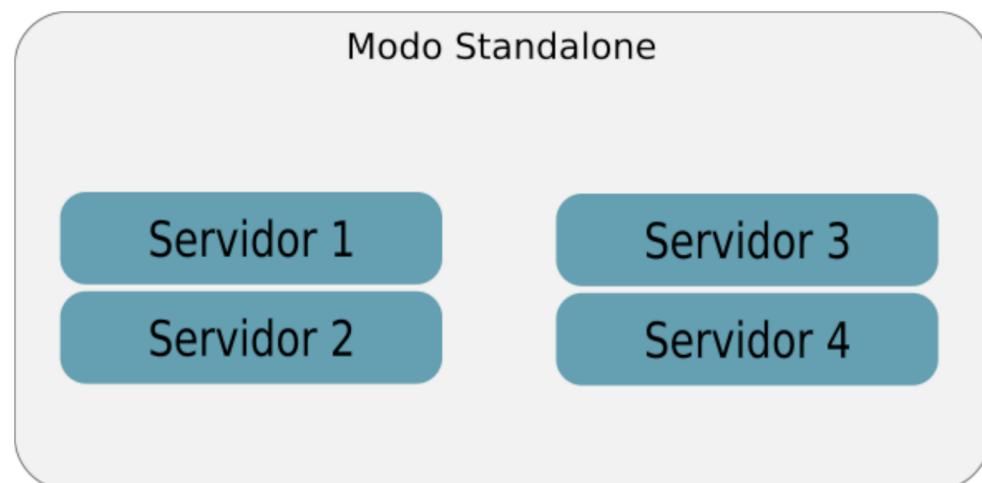


Figura 5: Diagrama visual do modo Standalone

O modo Standalone é o modo tradicional das versões anteriores.

Basicamente implica em ter uma instalação diferente (ou um diretório *standalone* diferente) para cada instância de Wildfly. Ou seja, para cada Wildfly

rodando no seu ambiente é necessário alterar seus próprios arquivo de configuração, suas próprias opções de execução para JVM, etc.

## Modo Domain

O modo Domain é o modo que foi introduzido no JBoss AS 7 onde é possível gerenciar um conjunto de instâncias Wildfly, agrupando-os e assim permitindo compartilhar configurações comuns entre eles. Além de compartilhar configurações, é possível também através de um único console de gerenciamento iniciar ou parar instâncias (ou grupos inteiros), verificar seu status e estatísticas de cada subsystem (falaremos sobre isso mais adiante), etc.

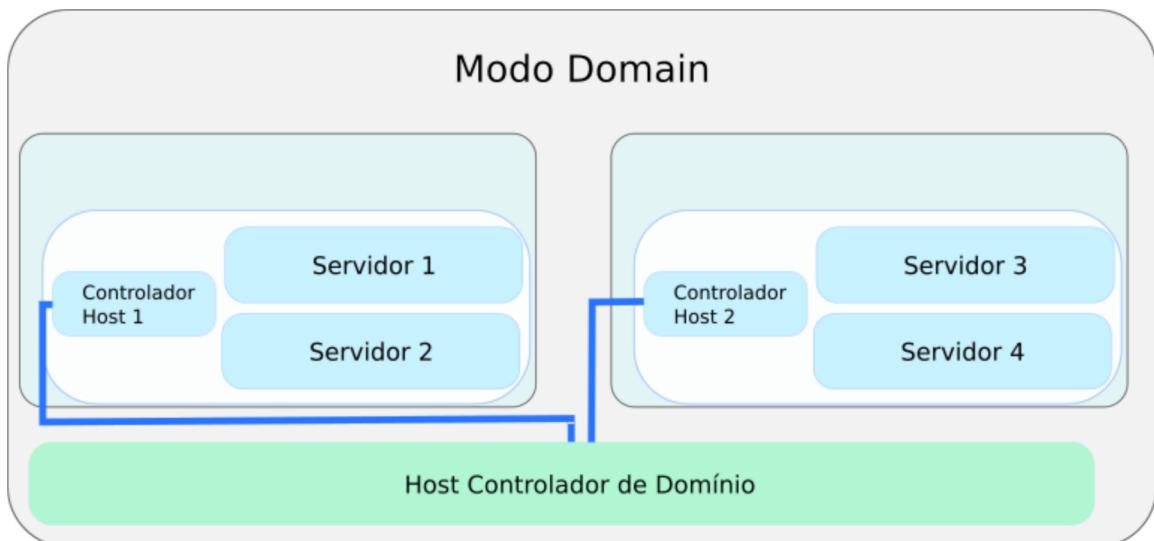


Figura 6: Diagrama ilustrativo do Modo Domain

## Decidindo entre modo Standalone ou modo Domain

Obviamente, ao analisar os dois modos de gerenciamento, vem aquela famosa pergunta: "E agora? Qual deles é melhor?". Com certeza a resposta sempre partirá do "Depende", mas para dar uma ajuda em escolher o melhor modo de gerenciamento é necessário ter em mente as seguintes perguntas:

- O número de instâncias é muito pequeno (algo em torno de 10) ou muito grande?
- As máquinas físicas que irão rodar as instâncias estão no mesmo Datacenter ou espalhados geograficamente?
- Eu quero ter um gerenciamento das instâncias individual ou centralizada?
- Eu quero ter um controle mais rígido sobre o deployment das aplicações no ambiente?

Com base nessas perguntas, você consegue decidir com bastante clareza e segurança qual o melhor modo a ser utilizado.

**IMPORTANTE:** os modos de gerenciamento em nenhum momento afetam Alta Disponibilidade, Performance ou alguma tecnologia Java EE. Eles apenas definem a melhor forma de gerenciamento das suas instâncias.

# Opções de Gerenciamento

## Perfis

No Java EE, conforme descrevemos na [Introdução](#), é dividido em dois perfis: Web e Full. Esses perfis (ou profiles) definem o quanto do Java EE será utilizado no desenvolvimento de suas aplicações. O Wildfly, pensando nesses perfis, criou arquivos de configuração para cada perfil Java EE bem como adicionou os componentes para *HA (High Availability, ou Alta disponibilidade)*, que são recursos que permitem o acesso às aplicações hospedadas no servidor mesmo em caso de falhas.

| Perfil     | Arquivo de configuração* | Utilização   |
|------------|--------------------------|--|
| Web        | standalone.xml           | Permite o uso do perfil Java EE Web                            |
| Full       | standalone-full.xml      | Permite o uso do perfil Java EE Full                           |
| Web c/ HA  | standalone-ha.xml        | Permite o uso do perfil Java EE Web com características de HA  |
| Full c/ HA | standalone-full-ha.xml   | Permite o uso do perfil Java EE Full com características de HA |

\* Apenas no modo standalone

## Rodando o Wildfly utilizando os profiles

Para rodar o Wildfly utilizando o profiles descritos na seção anterior, basta adicionar a opção `-c` à linha de comando passando o arquivo de configuração a ser utilizado. *Exemplo:* Se eu quiser utilizar o perfil Full com HA, basta eu rodar o seguinte comando para iniciar o Wildfly:

```
$ ./standalone.sh -c standalone-full-ha.xml
```

## E no modo Domain, como seleciono meu profile?

O modo domain, assim como explicado no tópico anterior ele é gerenciado por um host master e cada host slave possui o seu host controller. As configurações dos profiles são definidas no arquivo **domain.xml** do Domain Controller que são replicadas a todos os membros do Domínio.

Todos os arquivos de configuração do modo Domínio estão no diretório  
`$JBOSS_HOME/configuration/domain`

Os profiles são estruturados da mesma maneira que os profiles do modo standalone, porém com uma pequena diferença, repare na tabela abaixo:

| Perfil     | Nome do profile | Utilização   |
|------------|-----------------|--|
| Web        | default         | Permite o uso do perfil Java EE Web                            |
| Full       | full            | Permite o uso do perfil Java EE Full                           |
| Web c/ HA  | ha              | Permite o uso do perfil Java EE Web com características de HA  |
| Full c/ HA | full-ha         | Permite o uso do perfil Java EE Full com características de HA |

Veja abaixo um exemplo de uma definição de profile encontrada no arquivo **domain.xml**:

```
<profiles>
    <profile name="default">
        <subsystem xmlns="urn:jboss:domain:logging:3.0">
            ...
        </subsystem>
    <profile name="full">
        ...
    </profile>
</profiles>
```

```
</profiles>
```

A escolha do profile a ser utilizado por grupo de servidores é feita no também no arquivo **domain.xml** da seguinte maneira:

```
<server-groups>
    <server-group name="main-server-group" profile="full">
        <jvm name="default">
            <heap size="64m" max-size="512m"/>
        </jvm>
        <socket-binding-group ref="full-sockets"/>
    </server-group>
    <server-group name="other-server-group" profile="full-ha">
        <jvm name="default">
            <heap size="64m" max-size="512m"/>
        </jvm>
        <socket-binding-group ref="full-ha-sockets"/>
    </server-group>
</server-groups>
```

O treco a acima é encontrado no final do arquivo e note que em cada definição de grupo de servidores temos o parâmetro **profile** para cada um deles.

Para iniciar o WildFly no modo domínio basta executar o arquivo `$JBOSS_HOME/bin/domain.sh`

## Estrutura dos Diretórios

O WildFly manteve praticamente a mesma estrutura de diretórios do JBoss AS 7, enxuta e centralizada. Para aqueles que ainda não estão familiarizados com essa nova estrutura, a próxima seção descreverá em detalhes abaixo.

## Como está dividida a estrutura?

Bom, temos a seguinte estrutura de diretórios:

```
$JBOSS_HOME
├── appclient
├── bin
│   └── client
└── docs
    ├── contrib
    │   └── scripts
    │       ├── init.d
    │       ├── service
    │       └── systemd
    ├── examples
    ├── licenses
    └── schema
├── domain
│   ├── configuration
│   │   └── domain_xml_history
│   │       ├── current
│   │       └── snapshot
│   ├── data
│   └── servers
└── modules
└── standalone
```

```

    └── configuration
        └── standalone_xml_history
            ├── current
            └── snapshot
    └── data
    └── deployments
    └── lib
    └── log
    └── tmp
└── welcome-content

```

Veremos agora a função de cada diretório e o que é armazenado em cada um:

- *appclient* - Este diretório é utilizado para armazenar arquivos de deployments, configuração e também utilizado como área de escrita utilizados pelo **application client container**.
- *bin* - Todos os binários do servidor serão encontrados neste diretório, tais como standalone/domain.sh|bat, script para adição de usuários, etc.
  - *client* - Neste subdiretório está armazenados as bibliotecas necessárias para se conectar ao Wildfly utilizando JBoss CLI, EJB ou JMS.
- *docs* - Aqui você irá encontrar diversos tipos de arquivos como exemplos de configuração, exemplos se como executar o WildFLy como serviço, licenças e outros arquivos que lhe ajudarão a aprender mais sobre o WildFly.
  - *contrib* - Arquivos utilizados para executar o WildFly como serviço.
  - *scripts* - Scripts para configuração do WildFly como serviço.
    - *init.d* - Arquivos necessários para executar como serviço no Linux Linux baseado no gerenciador de serviços **init.d**.
    - *service* - Arquivos necessários para executar como serviço no Windows
    - *systemd* - Arquivos necessários para executar como serviço no Linux baseado no gerenciador de serviços **systemd**.
  - *examples* - Contém exemplos arquivos de configuração do modo standaloe pré configurados.
  - *licenses* - Contém todas as licenças sob o qual o código fonte do WildFLy está protegido.
  - *schema* - São os arquivos que definem as configurações/parâmetros aceitos nos *subsystems* presentes nos arquivos de configuração do Wildly, note que sua nomenclatura contém a versão do schema, assim como no arquivo de configuração contém a versão do schema que será utilizado para a configuração de determinado *subsystem*, Exemplo: Dado o seguinte *subsystem*:

```
<subsystem xmlns="urn:jboss:domain:logging:3.0">
```

Note que ele está utilizando a versão 3.0 do schema **logging**. Este mesmo padrão é válido para todos os outros *subsystems*.

- *domain* - Abriga toda a configuração do modo domain bem como os arquivos dos servidores gerenciados (veremos melhor sobre isso mais a frente) e arquivos de deployments, arquivos temporários e arquivos de logs.
  - *configuration* - Contém todos os arquivos de configuração do modo domínio.

- *domain\_xml\_history* - Contém todo o histórico dos arquivos de configuração.
  - *current* - Armazena a configuração corrente com sufixos v1, v2, vX.
  - *snapshot* - Armazena os snapshots, obtidos através do comando **CLI /host=HOSTNAME:take-snapshot**
- *servers* - Diretório utilizado para armazenar informações referentes aos servidores gerenciados controlados pelo domínio em questão. Dentro do diretório com o respectivo nome do servidor gerenciado existem também seus subdiretórios **data**, **tmp** e **log** cujo o objetivo é o mesmo do descrito no modo *standalone*.
- *data* - Diretório utilizado para persistir dados para que seja possível um restart sem perda de informação.
- *tmp* - Utilizado e gerado em Runtime, responsável por armazenar todos os arquivos temporários gerados durante a execução do servidor.
- *lib/ext* - Local utilizado para instalar bibliotecas utilizadas pelas aplicações através do mecanismo **Extension-List**.
- *modules* - O WildFly é baseado em um *classloader* modular (explicado em detalhes nos próximos tópicos), todos os módulos necessários para a execução do WildFly estão armazenados neste diretório, bem como os módulos customizados.
- *standalone* - Contém todos os arquivos necessários para a execução do WildFly no modo *standalone*.
  - *configuration* - Contém todo o histórico dos arquivos de configuração.
  - *standalone\_xml\_history* - Contém todo o histórico dos arquivos de configuração.
    - *current* - Armazena a configuração corrente com sufixos v1, v2, vX.
    - *snapshot* - Armazena os snapshots, obtidos através do comando **CLI :take-snapshot**
  - *data* - Diretório utilizado para persistir dados para que seja possível um restart sem perda de informação.
  - *deployments* - Existente somente no modo *standalone* é utilizado para realizar deployments utilizando o método **Deployment Scanner**.
  - *lib/ext* - Local utilizado para instalar bibliotecas utilizadas pelas aplicações através do mecanismo **Extension-List**.
  - *log* - Contém os logs do servidor em execução.
  - *tmp* - Diretório para escrita de arquivos temporários utilizados pelas aplicações em execução.
- *welcome-content* - É um diretório de uso interno do servidor que não deve ser modificado por usuários finais, sua alteração pode influenciar no funcionamento do WildFly bem como páginas de boas vindas e páginas de erros.

## Criando Usuário de Gerenciamento

Para conseguirmos utilizar a console de gerenciamento do WildFly é necessário criar um usuário de gerenciamento, também existe usuários de aplicação, segue uma pequena diferença entre eles:

- Management: Usuários utilizados para gerenciar o WildFly utilizando os seguintes métodos:
  - Console de gerenciamento

- JBoss CLI

Ambos estão disponíveis na porta 9990.

Neste tópico iremos abordar somente a criação de usuário de gerenciamento bem como acessar a console de gerenciamento.

Caso você tente acessar a console de gerenciamento sem ter antes, criado o usuário você será redirecionado para a página abaixo informando que é necessário executar o script *add-user.sh* (*add-user.bat* para Windows) para adicionar um usuário de gerenciamento utilizando o realm *ManagementRealm*:

- Acesse <http://localhost:9990>

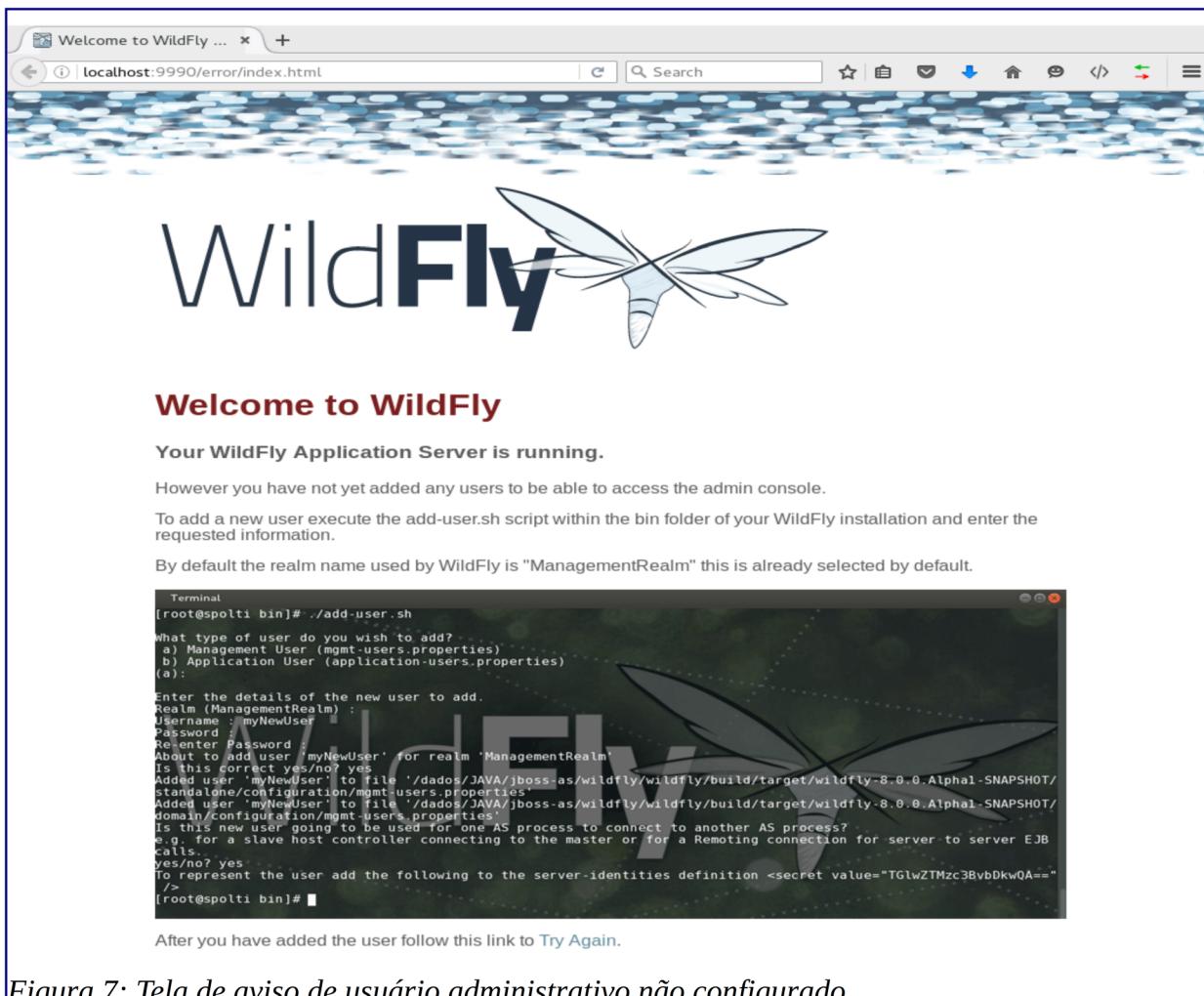


Figura 7: Tela de aviso de usuário administrativo não configurado

Para adicionar o usuário siga os passos a seguir:

- Execute o script *add-user.sh* que está localizado no diretório **\$JBOSS\_HOME/bin**

```
$ ./add-user.sh

What type of user do you wish to add?
a) Management User (mgmt-users.properties)
b) Application User (application-users.properties)
(a):
```

Por padrão o *ManagementRealm* é selecionado, apenas aperte *enter* para prosseguir.

O próximo passo será definir o nome do usuário, escolha um username e prossiga:

```
Enter the details of the new user to add.  
Using realm 'ManagementRealm' as discovered from the existing property files.  
Username : admin
```

Logo a seguir você será informado que o usuário *admin* já existe porém está desativado e irá lhe mostrar as opções disponíveis:

```
User 'admin' already exists and is disabled, would you like to...  
a) Update the existing user password and roles  
b) Enable the existing user  
c) Type a new username  
(a):
```

Você terá a opção de atualizar usuário existente e seus grupos, ativar o usuário existente ou digitar um novo username. Neste caso iremos somente atualizar a senha do usuário *admin*, por padrão esta opção já está selecionada, apenas tecle **enter**.

Agora defina a senha:

```
Password recommendations are listed below. To modify these restrictions edit the add-user.properties configuration file.  
- The password should be different from the username  
- The password should not be one of the following restricted values {root, admin, administrator}  
- The password should contain at least 8 characters, 1 alphabetic character(s), 1 digit(s), 1 non-alphanumeric symbol(s)  
Password :  
Re-enter Password :
```

O próximo passo é definir os grupos, no momento não será necessário definir nenhum, apenas prossiga:

```
What groups do you want this user to belong to? (Please enter a comma separated list, or leave blank for none)[ ]:  
Updated user 'admin' to file '/opt/wildfly-16.0.0.Final/standalone/configuration/mgmt-users.properties'  
Updated user 'admin' to file '/opt/wildfly-10.1.0.Final/domain/configuration/mgmt-users.properties'  
Updated user 'admin' with groups to file '/opt/wildfly-16.0.0.Final/standalone/configuration/mgmt-groups.properties'  
Updated user 'admin' with groups to file '/opt/wildfly-10.1.0.Final/domain/configuration/mgmt-groups.properties'
```

O script irá perguntá-lo se este usuário será utilizado para autenticação entre 2 servidores WildFly (Veremos com mais detalhes este processo na configuração do modo *Domain*). Neste caso será um usuário normal, digite **no** e tecle **enter**

```
Is this new user going to be used for one AS process to connect to another AS process?  
e.g. for a slave host controller connecting to the master or for a Remoting connection  
for server to server EJB calls.  
yes/no? no
```

Neste momento já estamos aptos a acessar a Console de Gerenciamento, tente acessá-la novamente e utilize as credenciais que criamos, caso esteja tudo certo você será redirecionado para a página principal:

## Dicas

### Criando usuário somente com um comando:

```
$WFLY_HOME/bin/add-user.sh -u admin2 -p teste@123 -s -e
```

### Criando usuários com senhas fracas (não permitido através do script add-user)

Este método não é recomendado, nunca use-o em produção.

```
echo -n "username:ManagementRealm:password" | openssl md5  
(stdin)= 8959126dd54df47f694cd762a51a1a6f
```

Com o **hash** em mãos edite o arquivo **\$WFLY\_HOME/standalone/configuration/mgmt-users.properties** e adicione o novo usuário seguido do hash, exemplo:

```
username=8959126dd54df47f694cd762a51a1a6f
```

Caso deseje criar um *application* user altere a Realm para *ApplicationRealm*.

### Alterando a política de senhas

Também é possível alterar a política de senhas utilizada pelo script de criação de usuários. No diretório **\*\*\$JBOSS\_HOME/bin \*\*** existe um arquivo chamado *add-user.properties* com o seguinte conteúdo:

```
#  
# Password restriction  
  
# Valid values: RELAX, WARN or REJECT  
# RELAX : Don't perform any strength checks on the password in both interactive and non-  
# interactive mode  
# WARN : Display a message about the strength of the password. Ask confirmation if the  
# password is weak in interactive mode  
# REJECT : Display a message about the strength of the password (if the password is weak,  
# the user is not created).  
# Ask confirmation if the password is weak in interactive mode  
password.restriction=WARN  
  
# Password minimum length  
password.restriction.minLength=8  
  
# Password must contains at least one alpha  
password.restriction.minAlpha=1  
  
# Password must contains at least one digit  
password.restriction.minDigit=1  
  
# Password must contains at least one symbol  
password.restriction.minSymbol=1  
  
# Password must not match the username. Valid values: TRUE or FALSE.  
password.restriction.mustNotMatchUsername=TRUE
```

Figura 8: Tela inicial do gerenciador

```
# Comma separated list of forbidden passwords (easily guessable)
password.restriction.forbiddenValue=root,admin,administrator

# Password strength. Valid values: VERY_WEAK, WEAK, MODERATE, MEDIUM, STRONG, VERY_STRONG
# or EXCEPTIONAL.
# If not present, it defaults to "MODERATE"
password.restriction.strength=MEDIUM

# Class of password strength checker.
# If not present, utility will revert to default implementation
password.restriction.checker=org.jboss.as.domain.management.security.password.simple.SimplePasswordStrengthChecker
```

Para alterar a política de senha conforme suas necessidades apenas edite a propriedade desejada neste arquivo e salve. As configurações realizadas já serão aplicadas na criação do próximo usuário.

# Ferramentas de administração Wildfly

Desde o JBoss AS 7, a configuração é centralizada o que, de certa forma facilita muito o gerenciamento do seu servidor ou parque de servidores.

A estrutura, quando comparada com o JBoss AS 7 não teve alterações, porém se você ainda não a conhece irá achar um pouco estranho a forma como está distribuída agora. Quando comparada com a versões mais antigas do JBoss, tem uma grande diferença, principalmente porque os arquivos de configuração eram diversos, em diferentes diretórios, componentes, etc. Se você está migrando do JBoss AS 6 ou versões anteriores, é de extrema importância que você leia este capítulo para ficar antenado com a estrutura dos arquivos de configuração do WildFly.

O intuito deste tópico será somente descrever os principais arquivos de configurações e sua aplicabilidade, será tratado em tópicos posteriores como configurar o WildFly.

Para facilitar assunto, o livro dividiremos em três partes:

- Configurações em geral;
- Configurações do modo standalone;
- Configurações do modo domínio.

## Configurações Gerais

Iremos começar com as configurações bem básicas que são as configurações realizadas dentro do diretório *bin*. Nele temos os seguintes arquivos de configuração:

- *add-user.properties* - Configuração da política de senha na criação de usuários
- *appclient.conf* - Arquivo que define as configurações do *appclient*, utilitário para execução de aplicações sem a necessidade de executar o WildFly
- *domain.conf* - Define as configurações utilizadas para iniciar o modo domínio.
- *jboss-cli-logging.properties* - Definições de log do JBoss CLI.
- *jboss-cli.xml* - Define as configurações que serão utilizadas pelo JBoss CLI, através do script *jboss-cli.sh* tais como *connection timeout* e *controller*.
- *standalone.conf* - Define as configurações utilizadas para iniciar o modo standalone.

*Obs:* O wildFly 10 já possui suporte ao *PowerShell*, note que já há vários scripts com o sufixo *ps1* no diretório *bin*.

## Configurações modo Standalone

Todos os arquivos de configuração do modo *Standalone* estão no diretório *\$JBOSS\_HOME/standalone/configuration*, neste diretório iremos encontrar os seguintes arquivos:

- *application-roles.properties*: Neste arquivo é configurado as permissões (Roles) dos usuários utilizados em aplicações.
- *application-users.properties*: Armazena usuários utilizados para autenticação realizada por aplicações.
- *logging.properties*: Arquivo que define as configurações de log do Wildfly, este arquivo é alterado de acordo com as definições de logging realizadas no subsystem *logging*.
- *mgmt-groups.properties*: Grupos dos usuários de gerenciamento, note os grupos são usados quando o *RBAC (Role Based Access Control)*, falaremos sobre isso nos próximos capítulos) ou Controle de Acesso Baseado em Roles.

- *mgmt-users.properties*: Define usuários e senha utilizados para autenticar usuários de gerenciamento.
- *standalone.xml*: Todas as definições e configurações do Wildfly são feitas através deste arquivo.

## Configurações modo Domínio

No modo *Domain* temos os mesmos arquivos de configuração com a exceção do *standalone.xml* e adição dos seguintes arquivos:

- *domain.xml*: Define todas as configurações do modo domínio. Em adição ao *domain.xml* temos mais 3 arquivos:
- *host-master.xml*: Este arquivo define as configuração da instância do WildFly que irá atuar somente como Controlador de Domínio.
- *host-slave.xml*: Um modo domínio distribuído tem seus host masters e slaves, este arquivo define as configurações do slaves.
- *host.xml*: Se por acaso desejar executar o WildFly no modo domínio em somente um servidor, é neste arquivo que estarão contidos todas as informações referentes ao host. Note que também é possível utilizá-lo como host master ou slave.
- *default-server-logging.properties*: Arquivo que define as configurações de log padrão dos servidores gerenciados.

Nos próximos tópicos, iremos conhecer como configurar cada componente Java EE do Wildfly e seus componentes de infraestrutura (como logging), mas antes de darmos início é necessário entendermos como eles foram desenvolvidos no Wildfly.

Diferente de versões anteriores, o Wildfly centralizou a configuração do servidor em um (no máximo dois em modo Domain) arquivo de configuração e que através dele é possível configurar qualquer coisa dentro do servidor. Na verdade, se considerarmos o que seria o Wildfly, ele é simplesmente uma espécie de Kernel do Wildfly (conhecido como *Wildfly Core*) onde é possível adicionar funcionalidades a ele (permitindo assim a extensibilidade na sua arquitetura interna). Para suas funcionalidades foi introduzido os conceitos de *extension* e *subsystem*, que são os blocos lógicos para adicionar/remover das funcionalidades do servidor e sua configuração. Mas o que significa esses dois conceitos? Veremos em detalhes a seguir.

## Extension

O Extension do Wildfly é a forma como o Wildfly Core pode reconhecer novas funcionalidades e assim agregar ao seu funcionamento principal. Basicamente, a função do extension é registrar uma funcionalidade para o Wildfly Core carregar em memória e também gerenciar o ciclo de vida dela. Por exemplo, as extensions listadas no perfil padrão (*standalone.xml*) são as seguintes:

- org.jboss.as.clustering.infinispan
- org.jboss.as.connector
- org.jboss.as.deployment-scanner
- org.jboss.as.ee
- org.jboss.as.ejb3
- org.jboss.as.jaxrs
- org.jboss.as.jdr
- org.jboss.as.jmx

- org.jboss.as.jpa
- org.jboss.as.jsf
- org.jboss.as.logging
- org.jboss.as.mail
- org.jboss.as.naming
- org.jboss.as.pojo
- org.jboss.as.remoting
- org.jboss.as.sar
- org.jboss.as.security
- org.jboss.as.transactions
- org.jboss.as.webservices
- org.jboss.as.weld
- org.wildfly.extension.batch.jberet
- org.wildfly.extension.bean-validation
- org.wildfly.extension.io
- org.wildfly.extension.request-controller
- org.wildfly.extension.security.manager
- org.wildfly.extension.undertow

## Subsystem

Subsystem nada mais é que a funcionalidade em si implementada e que é registrada no Wildfly Core pela Extension. Nela, é possível extender a funcionalidade através da configuração via xml. O Subsystem é onde realmente você terá o suporte a Servlets no Wildfly, EJB, Transações, Logging, etc. Como exemplo, abaixo temos a configuração do subsystem Logging:

```
<subsystem xmlns="urn:jboss:domain:logging:3.0">
    <console-handler name="CONSOLE">
        <level name="INFO"/>
        <formatter>
            <named-formatter name="COLOR-PATTERN"/>
        </formatter>
    </console-handler>
    <periodic-rotating-file-handler name="FILE" autoflush="true">
        <formatter>
            <named-formatter name="PATTERN"/>
        </formatter>
        <file relative-to="jboss.server.log.dir" path="server.log"/>
        <suffix value=".yyyy-MM-dd"/>
        <append value="true"/>
    </periodic-rotating-file-handler>
    <logger category="com.arjuna">
        <level name="WARN"/>
    </logger>
    <logger category="org.jboss.as.config">
        <level name="DEBUG"/>
    </logger>
    <logger category="sun.rmi">
        <level name="WARN"/>
    </logger>
    <root-logger>
        <level name="INFO"/>
        <handlers>
            <handler name="CONSOLE"/>
            <handler name="FILE"/>
        </handlers>
    </root-logger>
    <formatter name="PATTERN">
```

```

        <pattern-formatter pattern="%d{yyyy-MM-dd HH:mm:ss,SSS} %-5p [%c] (%t) %s
%e%n"/>
    </formatter>
    <formatter name="COLOR-PATTERN">
        <pattern-formatter pattern="%K{level}%d{HH:mm:ss,SSS} %-5p [%c] (%t) %s%e
%n"/>
    </formatter>
</subsystem>

```

## Histórico de alterações

Uma das maiores funcionalidades no Wildfly é o registro histórico de configurações. Dessa forma, se houver qualquer alteração nos arquivos de configuração (seja ela editando manualmente o arquivo, via JBoss CLI ou Console Web), o Wildfly irá criar um backup das alterações. Isso permite que no caso de alguma alteração afetar negativamente o ambiente é possível reverter a alteração apenas copiando a última versão alterada.

A estrutura de história está descrita no capítulo Diretórios. Para recuperar o arquivo, basta sobreescrever o atual com um dos snapshots que ficam dentro do diretório `snapshots` (o snapshot é o nome do arquivo de configuração seguido da data e hora de alteração).

## Socket-bindings

*Socket Bindings* define o conjunto de Sockets a serem utilizados pelos Subsystems do Wildfly. Nele, você pode definir um nome lógico para esse socket, a porta a ser utilizada, a qual Interface irá responder, se o tráfego é somente de saída, etc. Além disso, com a tag `<socket-binding-group>`, é possível definir um offset<sup>[^3]</sup> de portas para que instâncias diferentes rodando na mesma máquina física não tenham conflito de portas. Abaixo temos uma configuração de Socket Binding para o perfil Web:

```

<socket-binding-group name="standard-sockets" default-interface="public"
port-offset="${jboss.socket.binding.port-offset:0}">
    <socket-binding name="management-http" interface="management" port="$
{jboss.management.http.port:9990}">
        <socket-binding name="management-https" interface="management" port="$
{jboss.management.https.port:9993}">
            <socket-binding name="ajp" port="${
{jboss.ajp.port:8009}"/>
            <socket-binding name="http" port="${
{jboss.http.port:8080}"/>
            <socket-binding name="https" port="${
{jboss.https.port:8443}"/>
            <socket-binding name="txn-recovery-environment" port="4712"/>
            <socket-binding name="txn-status-manager" port="4713"/>
            <outbound-socket-binding name="mail-smtp">
                <remote-destination host="localhost" port="25"/>
            </outbound-socket-binding>
        </socket-binding>
    </socket-binding-group>

```

## Como configurar

Além da configuração manual no XML, é possível também alterar os Socket-Bindings de outras maneiras a seguir:

### Via Web Console

- Acesse <http://localhost:9990>
- Clique em Configuration

The screenshot shows the WildFly management console. The top navigation bar has tabs: Home, Deployments, Configuration (which is selected), Runtime, Access Control, and Patching. The left sidebar under the Configuration tab has links: Subsystems, Interfaces, Socket Binding (which is selected and highlighted in blue), Paths, and System Properties. The main content area is titled "Configuration" and describes it as the overall system configuration. It includes sections for "Common Configuration Tasks" (Configure subsystems, interfaces and socket bindings) and "Related Links" (Server Runtime). A "View" button is located at the bottom right of the sidebar.

Figura 9: Configuração de Socket Binding

Clique em **Socket Binding**

The screenshot shows the WildFly management console. The top navigation bar has tabs: Home, Deployments, Configuration (selected), Runtime, Access Control, and Patching. The left sidebar under the Configuration tab has links: Profiles, Interfaces, Socket Binding (selected and highlighted in blue), Paths, and System Properties. The main content area is titled "Socket Bindings" and describes socket bindings and socket binding groups allowing you to define network ports and their relationship to the networking interfaces required for a configuration profile. A "View" button is located at the bottom right of the sidebar.

Figura 10: Configuração de Socket Binding

- Clique em **View**

**WildFly**

« Back Configuration: Socket Binding

SOCKET BINDINGS

### Socket Binding Groups

Please choose an entry for specific settings.

| Name                  | Option                    |
|-----------------------|---------------------------|
| full-ha-sockets       | <a href="#">View &gt;</a> |
| full-sockets          | <a href="#">View &gt;</a> |
| ha-sockets            | <a href="#">View &gt;</a> |
| load-balancer-sockets | <a href="#">View &gt;</a> |
| standard-sockets      | <a href="#">View &gt;</a> |

Add Remove Clone

« < 1-5 of 5 > »

**Attributes**

|                    |                 |
|--------------------|-----------------|
| Name:              | full-ha-sockets |
| Default Interface: | public          |

Figura 11: Bindings disponíveis

- Aparecerá a listagem dos Socket Bindings disponíveis, onde é possível Adicionar/Remover/ Editar os Socket Bindings

**WildFly**

« Back Configuration: Socket Binding

SOCKET BINDINGS

### Socket Bindings: Group standard-sockets

A list of socket configurations. These configurations are referenced throughout the overall server/domain configuration.

| Name                     | Port                                   | MCast Port |
|--------------------------|--|------------|
| ajp                      | <code>\$(jboss.ajp.port:8009)</code>   |            |
| http                     | <code>\$(jboss.http.port:8080)</code>  |            |
| https                    | <code>\$(jboss.https.port:8443)</code> |            |
| txn-recovery-environment | 4712                                   |            |
| txn-status-manager       | 4713                                   |            |

Add Remove

Edit

Name: ajp

Interface:

Port: `$(jboss.ajp.port:8009)`

Fixed Port?: false

► Multicast

Need Help?

Figura 12: Configuração detalhada de binding

## Via CLI

```
/socket-binding-group=new-sockets:add(default-interface=public)  
  
/socket-binding-group=new-sockets/socket-binding=new-socket-binding:write-  
attribute(name=interface,value=unsecure)
```

## Definindo Port Offset em modo Domain

```
/host=master/server-config=server-two/:write-attribute(name=socket-binding-port-  
offset,value=250)
```

## Interfaces

Interfaces são denominações lógicas para interfaces de rede que irão se associar aos sockets (ver próxima seção) para expor algum serviço de rede para os Subsystems. As Interfaces podem associar os sockets para um IP em específico ou até mesmo para uma NIC[^2] da máquina física, permitindo então dedicar o tráfego de rede dos Subsystems do Wildfly passar por uma interface específica de rede. Abaixo temos um exemplo de Interfaces:

```
<interfaces>  
    <interface name="public">  
        <inet-address value="${jboss.bind.address:127.0.0.1}"/>  
    </interface>  
    <interface name="internal">  
        <nics name="eth1"/>  
    </interface>  
</interfaces>
```

## Deploy de aplicações

### Pacotes WAR, EAR, SAR

Em engenharia de software, um arquivo WAR (do inglês Web application ARchive) é um arquivo JAR usado para distribuir uma coleção de JavaServer Pages, Servlets Java, classes Java, arquivos XML, bibliotecas de tag, páginas web estáticas (arquivos HTML e relacionados) e outros recursos que, juntos, constituem uma aplicação web.

### Conteúdo e estrutura

Um arquivo WAR pode ser assinado digitalmente do mesmo modo que um arquivo JAR, afim de permitir que outros determinem de que fonte o código é proveniente. Há arquivos e diretórios especiais dentro de um arquivo WAR.

O diretório /WEB-INF no arquivo WAR contém um arquivo chamado web.xml que define a estrutura da aplicação web. Se a aplicação web está apenas servido arquivos JSP, o arquivo web.xml não é estritamente necessário. Se a aplicação web utiliza servlets, então o recipiente (container) de servlets usa o web.xml para determinar a qual servlet uma solicitação de URL será roteada. O web.xml também é usado para definir variáveis de contexto que podem ser referenciadas dentro dos servlets e é usada para definir dependências de ambiente que o implantador (deployer) espera configurar. Um exemplo disto é uma dependência de uma sessão de email usada para enviar email. O recipiente de servlet é responsável por fornecer este serviço.

Vantagens dos arquivos WAR:

- desenvolvimento, teste e implantação fáceis

- a versão da aplicação implantada é facilmente identificada
- todos os recipientes Java EE suportam arquivos .WAR

Uma desvantagem da implantação web utilizando arquivos WAR em ambientes muito dinâmicos é que alterações pequenas não podem ser feitas durante o tempo de execução. Qualquer alteração, seja qual for, requer regeração e reimplantação do arquivo WAR inteiro.

Um pacote EAR ou Enterprise Application aRchive é um empacotamento JRA que pode conter um ou vários pacotes WAR ou Jar (drivers JDBC) por exemplo. Esses pacotes internos são componentes que suportam as especificações Enterprise do Java EE como (EJB, JAX-RS, etc)

Na sua estrutura, possui uma pasta META-INF com um arquivo application.xml usado para configurações definidas no padrão Java EE.

O deploy funciona da mesma forma que os arquivos WAR. Para mais informações desse tópico, consulte o link:

<https://docs.oracle.com/javaee/7/tutorial/packaging.htm#gKJlQ4>

## Deploy em sistema de arquivos (Standalone)

Também é possível fazer deploy de pastas descompatadas. Para maiores detalhes, consulte o link:

<https://wildfly.org/news/2017/09/08/Exploded-deployments/>

## Deploy via plugin Maven

O plugin maven possui as tarefas padrão para [deploy](#), [undeploy](#) e [redeploy](#) de aplicações no WildFly, basta adicionar a entrada apropriada no arquivo POM:

```
<project>
  ...
  <build>
    ...
    <plugins>
      ...
      <plugin>
        <groupId>org.wildfly.plugins</groupId>
        <artifactId>wildfly-maven-plugin</artifactId>
        <version>2.0.1.Final</version>
      </plugin>
    ...
  </plugins>
  ...
</build>
...
</project>
```

O artefato é o nome do arquivo compilado e pronto para enviar ao servidor;

```
mvn wildfly:deploy
```

```
mvn wildfly:redeploy
mvn wildfly:undeploy
```

## Distribuindo sua aplicação no modo Domain

Semelhante ao passo anterior. Apenas acrescente o parâmetro server-groups

```
<project>
  ...
  <build>
    ...
    <plugins>
      ...
      <plugin>
        <groupId>org.wildfly.plugins</groupId>
        <artifactId>wildfly-maven-plugin</artifactId>
        <version>2.0.1.Final</version>
        <configuration>
          <server-groups>
            <server-group>main-server-group</server-group>
          </server-groups>
        </configuration>
      </plugin>
    ...
  </plugins>
  ...
</build>
...
</project>
```

## Serviço de Logging

Os arquivos de log se localizam em:

Em modo Standalone: WILDFLY\_HOME/standalone/server/log

- Em modo Domain:  
WILDFLY\_HOME/domain/servers/SERVIDOR/log/server.log

## Configurando Logging

- Em modo Standalone:  
WILDFLY\_HOME/standalone/configuration/logging.properties
- Em modo Domain:  
WILDFLY\_HOME/domain/servers/SERVIDOR/data/server.log

## Inspecionando erros em tempo de bootstrap

```
/core-service=management:read-boot-errors
```

## Configurando Logging por aplicação

```
<dependency>
  <groupId>org.jboss.logging</groupId>
  <artifactId>jboss-logging</artifactId>
  <version>3.3.0.Final-redhat-1</version>
  <scope>provided</scope>
</dependency>
import org.jboss.logging.Logger;

private static final Logger LOGGER = Logger.getLogger(MinhaClasse.class);
```

```

LOGGER.debug("debug");
LOGGER.info("info");
LOGGER.error("error");
LOGGER.trace("trace");
LOGGER.fatal("fatal");

```

- Para empacotamentos EAR, no diretório META-INF
- Para empacotamentos WAR, no diretório WEB-INF/classes

## Desabilitando Logging por aplicação via CLI

```
/subsystem=logging:write-attribute(name=use-deployment-logging-
config,value=false)
```

## Administração JMS

### O que é JMS

A mensageria é uma solução de integração de sistemas que permite a construção de aplicações distribuídas cujas partes não interagem entre si diretamente, mas trocam mensagens de forma assíncrona, através de um mediador central, com baixíssimo acoplamento.

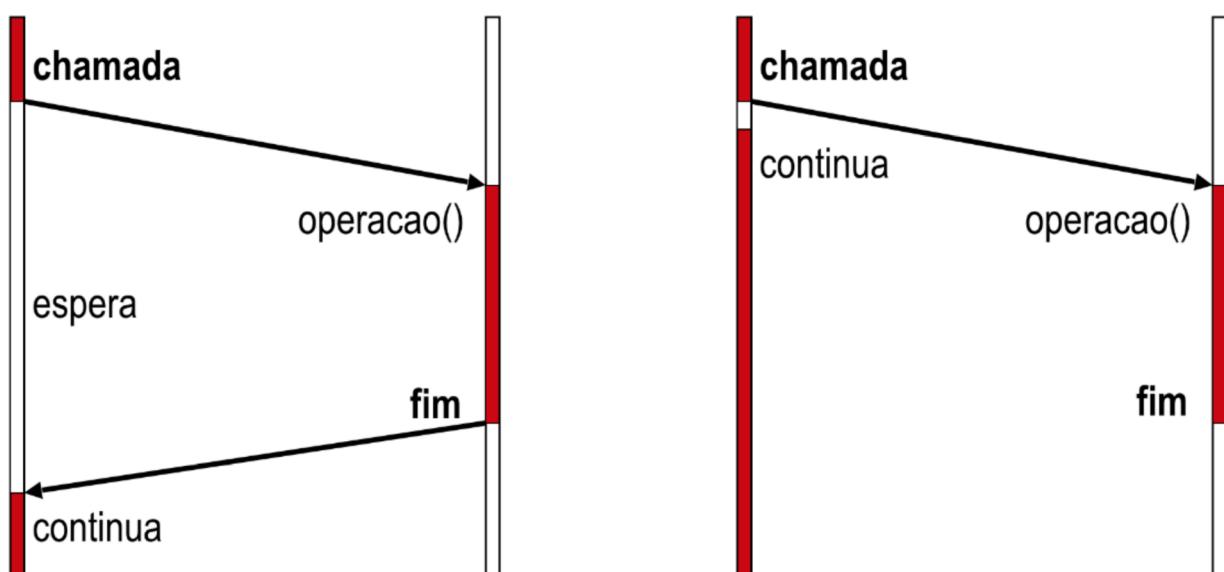


Figura 13: Comunicação síncrona (RMI/EJB) e assíncrona(Mensageria/JMS)

JMS é a API Java padrão para mensageria e que é implementada em todos os servidores que suportam Java EE (JBoss/WildFly, GlassFish, TomEE, etc.). JMS também pode ser usada em servidores que não oferecem outros serviços além da mensageria (como ActiveMQ). JMS consiste de uma coleção de interfaces uniformes para serviços de mensageria e permite escrever aplicações que irão rodar em diferentes provedores. A versão de JMS correspondente ao Java EE 8 é JMS 2.0, mas neste tutorial iremos mostrar também JMS 1.1, já que serviços populares como ActiveMQ ainda não suportam JMS 2.0. As interfaces JMS permitem que aplicações tenham acesso ao serviço de mensageria. Aplicações JMS são sempre clientes de mensageria. O serviço funciona como um

mediador que viabiliza a comunicação entre clientes produtores e consumidores. O ponto de comunicação é chamado de destino (Destination) e representa um canal de mensageria. Geralmente é implementado como uma fila de mensagens. Produtores e consumidores comunicam-se enviando e retirando mensagens do canal. A comunicação usando JMS suporta recebimento síncrono(usando um blocking method) ou assíncrono(usando um listener de eventos). As mensagens podem conter qualquer tipo de objeto. Aplicações JMS podem usar a infraestrutura de mensageria para transferir dados, texto, tipos primitivos, objetos serializados, XML, JSON, etc.

Primeira coisa que precisamos fazer é configurar o Wildfly. Se você utiliza o NetBeans como IDE aqui no blog já existe um pequeno tutorial de [como configurá-lo para desenvolvimento](#).

## Configuração básica

Basicamente o que precisamos fazer é dizer que ele deverá ser inicializado com a configuração do *standalone-full.xml* e do *standalone-full-ha.xml*, por default o Wildfly utiliza o *standalone.xml* que não traz a parte de JMS. Outra maneira de fazer é adicionar o subsistema de JMS no *standalone.xml*, mas isso da mais trabalho do que apenas trocar o arquivo de inicialização além de já conter todas as otimizações necessárias.

Veja um trecho da configuração do subsistema de mensageria do Wildfly 16:

```
<default-bindings context-service="java:jboss/ee/concurrency/context/default"
datasource="java:jboss/datasources/ExampleDS"
jms-connection-factory="java:jboss/DefaultJMSConnectionFactory" managed-executor-
service="java:jboss/ee/concurrency/executor/default" managed-scheduled-executor-
service="java:jboss/ee/concurrency/scheduler/default" managed-thread-factory="java:jboss/
ee/concurrency/factory/default"/>
    </subsystem>
    <subsystem xmlns="urn:jboss:domain:ee-security:1.0"/>
    <subsystem xmlns="urn:jboss:domain:ejb3:5.0">
        <session-bean>
            <stateless>
                <bean-instance-pool-ref pool-name="slsb-strict-max-pool"/>
            </stateless>
            <stateful default-access-timeout="5000" cache-ref="simple" passivation-
disabled-cache-ref="simple"/>
                <singleton default-access-timeout="5000"/>
        </session-bean>
        <mdb>
            <resource-adapter-ref resource-adapter-name="${ejb.resource-adapter-
name:activemq-ra.rar}"/>
                <bean-instance-pool-ref pool-name="mdb-strict-max-pool"/>
        </mdb>
        <pools>
            <bean-instance-pools>
                <strict-max-pool name="mdb-strict-max-pool" derive-size="from-cpu-
count" instance-acquisition-timeout="5" instance-acquisition-timeout-unit="MINUTES"/>
                    <strict-max-pool name="slsb-strict-max-pool" derive-size="from-
worker-pools" instance-acquisition-timeout="5" instance-acquisition-timeout-
unit="MINUTES"/>
            </bean-instance-pools>
        </pools>
        <caches>
            <cache name="simple"/>
            <cache name="distributable" passivation-store-ref="infinispan"-
aliases="passivating clustered"/>
        </caches>
        <passivation-stores>
            <passivation-store name="infinispan" cache-container="ejb" max-
size="10000"/>
        </passivation-stores>
        <async thread-pool-name="default"/>
    </subsystem>
```

```

        <timer-service thread-pool-name="default" default-data-store="default-file-
store">
            <data-stores>
                <file-data-store name="default-file-store" path="timer-service-data"
relative-to="jboss.server.data.dir"/>
            </data-stores>
        </timer-service>
        <remote connector-ref="http-remoting-connector" thread-pool-name="default">
            <channel-creation-options>
                <option name="READ_TIMEOUT" value="${prop.remoting-
connector.read.timeout:20}" type="xnio"/>
                <option name="MAX_OUTBOUND_MESSAGES" value="1234" type="remoting"/>
            </channel-creation-options>
        </remote>
        <thread-pools>
            <thread-pool name="default">
                <max-threads count="10"/>
                <keepalive-time time="100" unit="milliseconds"/>
            </thread-pool>
        </thread-pools>
    
```

## Transações e Persistência

Mais uma especificação Java EE das mais usadas. O Servidor de aplicação cuida de todo o ciclo de criação e manutenção das conexões de dados para que nos preocupemos apenas com a aplicação em si e a lógica de negócios. Veja o trecho do arquivo de configuração que usamos para configurar um DataSource e drivers JDBC:

```

<subsystem xmlns="urn:jboss:domain:datasources:4.0">
    <datasources>
        <datasource jndi-name="java:jboss/datasources/ExampleDS" pool-name="ExampleDS">
            <connection-url>jdbc:h2:mem:test;DB_CLOSE_DELAY=-1</connection-url>
            <driver>h2</driver>
            <pool>
                <min-pool-size>10</min-pool-size>
                <max-pool-size>20</max-pool-size>
                <prefill>true</prefill>
            </pool>
            <security>
                <user-name>sa</user-name>
                <password>sa</password>
            </security>
        </datasource>
        <xa-datasource jndi-name="java:jboss/datasources/ExampleXADS" pool-
name="ExampleXADS">
            <driver>h2</driver>
            <xa-datasource-property name="URL">jdbc:h2:mem:test</xa-datasource-property>
            <xa-pool>
                <min-pool-size>10</min-pool-size>
                <max-pool-size>20</max-pool-size>
                <prefill>true</prefill>
            </xa-pool>
            <security>
                <user-name>sa</user-name>
                <password>sa</password>
            </security>
        </xa-datasource>
        <drivers>
            <driver name="h2" module="com.h2database.h2">
                <xa-datasource-class>org.h2.jdbcx.JdbcDataSource</xa-datasource-class>
            </driver>
        </drivers>
    </datasources>

```

## Utilizando driver JDBC como deployment

- Baixe o Driver JDBC
- Se o Driver JDBC do fornecedor ainda não é compatível com a versão JDBC 4, veja na seção Tornando o Driver JDBC compatível com a versão 4 para converter o Driver.
- Faça o deploy do Driver JDBC:

```
$ WFLY_HOME/bin/jboss-cli.sh  
$ deploy driver-jdbc.jar
```

- Caso o deploy tenha sido feito com sucesso, você verá a seguinte mensagem no log do servidor:

```
WFLYJCA0018: Started Driver service with driver-name = jdbc-driver.jar_com.sua.classe.jdbc
```

## Instalando driver JDBC como módulo

- Baixe o Driver JDBC
- Se o Driver JDBC do fornecedor ainda não é compatível com a versão JDBC 4, veja na seção Tornando o Driver JDBC compatível com a versão 4 para converter o Driver.
- Execute o JBoss CLI

```
$ WFLY_HOME/bin/jboss-cli.sh
```

- No prompt do JBoss/Wildfly CLI, execute o seguinte comando:

```
module add --name=MODULE_NAME --resources=PATH_TO_JDBC_JAR --dependencies=DEPENDENCIES
```

Um exemplo para adicionar o Driver JDBC seria assim:

```
module add --name=com.mysql --resources=/caminho/do/mysql-connector-java-8.1.36-bin.jar  
--dependencies=javax.api,javax.transaction.api
```

- Ao executar o comando anterior, o Wildfly irá:
  - Criar um subdiretório WILDFLY\_HOME/modules/com/seu/modulo/jdbc/main
  - Copiar o JAR dentro desse diretório
  - Criar um arquivo *module.xml* contendo a configuração para o módulo (Veja a seção Adicionando um módulo customizado para maiores detalhes)
- Conecte-se à instância do Wildfly com o seguinte comando:

```
connect
```

- Registre o Driver JDBC, passando o nome do módulo recém-adicionado:

```
/subsystem=datasources/jdbc-driver=DRIVER_NAME:add(driver-name=DRIVER_NAME,driver-module-name=MODULE_NAME,driver-xa-datasource-class-name=XA_DATASOURCE_CLASS_NAME, driver-class-name=DRIVER_CLASS_NAME)
```

Um exemplo para adicionar o Driver JDBC do MySQL seria:

```
/subsystem=datasources/jdbc-driver=mysql:add(driver-name=mysql,driver-module-name=com.mysql.driver-xa-datasource-class-name=com.mysql.jdbc.jdbc2.optional.MysqlXADataSource, driver-class-name=com.mysql.jdbc.Driver)
```

## Tornando o Driver JDBC compatível com a versão 4

- Crie um diretório temporário para hospedar os arquivos.
- Crie dentro desse diretório um subdiretório *META-INF/services*
- Dentro do subdiretório *META-INF/services*, crie um arquivo chamado *java.sql.Driver*
- Abra o arquivo e escreva em uma única linha o nome completo da classe com o seu pacote(denominado Fully-Qualified Class Name). Exemplo para o Driver do MySQL;  
*com.mysql.jdbc.Driver*
- Utilize o comando *jar* do Java para incluir o novo arquivo no Driver JDBC:

```
$ jar \uf driver-jdbc.jar META-INF/services/java.sql.Driver
```

Veremos no curso como configurar a conexão.

## Segurança

### Introdução a JAAS

Aplicações que restringem o acesso de usuários ou processos a apenas alguns de seus recursos são muito comuns; implementar esse tipo de controle de segurança, no entanto, não é nada fácil - e fazê-lo de forma que a aplicação seja independente da tecnologia de segurança utilizada é ainda mais difícil. É exatamente isso que o JAAS (Java Authentication and Authorization Service) se propõe a resolver.

### Conceitos

Antes de descrever a API de autenticação e autorização de Java, precisamos apresentar alguns conceitos:

- Autenticação é o processo de identificação de um “usuário” (pessoa, processo etc.) em um sistema. Isto é feito comparando-se as credenciais passadas pelo usuário com as esperadas pelo sistema. O método mais comum de autenticação é o uso de senhas, mas pode-se utilizar várias outras técnicas, como biometria.
- Autorização é o processo de verificação dos direitos que um usuário possui para acessar/ manipular um determinado recurso do sistema. Um exemplo seria restringir o acesso de um cliente de um banco a apenas sua própria conta.

### Iniciando a autenticação

Ao iniciar o processo de autenticação com JAAS, a aplicação deve criar um objeto do tipo *javax.security.auth.login.LoginContext*, passando para ele um nome de configuração de módulos de login (que são implementações da interface *javax.security.auth.spi.LoginModule*). Essa configuração é definida em um arquivo-texto simples, externo à aplicação (veja detalhes adiante).

O LoginContext é responsável por armazenar as informações do usuário; o LoginModule, por obter essas informações.

Veja a Listagem 1. Note que, antes de instanciar o LoginContext, foi criado um javax.security.auth.Subject. Um Subject armazena as informações relacionadas a uma entidade colhidas no processo de autenticação, por exemplo: o nome, o login e a senha de um usuário. Para isso, o Subject usa um ou mais objetos java.security.Principal, representando identidades e, opcionalmente, credenciais públicas ou privadas.

Veremos na prática como configurar o módulo de JAAS do Wildfly. Mas segue o link com todos os detalhes na documentação oficial:

<https://docs.jboss.org/author/display/WFLY/Subsystem+configuration>

## SSL no wildfly

Esta configuração é apenas para habilitar o https em uma máquina de desenvolvimento com o Servidor de Aplicação Wildfly. Não fui a fundo na questão de segurança já que não sou eu que administro o servidor onde vai rodar a aplicação. Portanto não use isto em ambiente de produção.

A primeira coisa a fazer é entrar no diretório da instalação do Wildfly, na pasta **./standalone/configuration/** e executar o seguinte comando:

```
keytool -genkeypair -alias serverkey -keyalg RSA -keysize 2048 -validity 7360 -keystore localhost.keystore -keypass mypassword -storepass mypassword -dname "cn=Server Administrator,o=Acme,c=GB"
```

Edite o arquivo **./standalone/configuration/standalone.xml** e adicione o seguinte xml dentro de **security-realms**:

```
<security-realm name="UndertowRealm">
    <server-identities>
        <ssl>
            <keystore path="localhost.keystore" relative-to="jboss.server.config.dir"
keystore-password="mypassword"/>
        </ssl>
    </server-identities>
</security-realm>
```

1. Em seguida adicione a linha abaixo dentro de subsystem undertow -> server:

```
1
<https-listener name="https" socket-binding="https" security-realm="UndertowRealm"/>
```

Veja como ficou a entrada completa:

```
<subsystem xmlns="urn:jboss:domain:undertow:1.2">
    <buffer-cache name="default"/>
    <server name="default-server">
        <http-listener name="default" socket-binding="http"/>
        <https-listener name="https" socket-binding="https" security-
realm="UndertowRealm"/>
        <host name="default-host" alias="localhost">
            <location name="/" handler="welcome-content"/>
            <filter-ref name="server-header"/>
            <filter-ref name="x-powered-by-header"/>
        </host>
```

```

</server>
<servlet-container name="default">
    <jsp-config/>
    <websockets/>
</servlet-container>
<handlers>
    <file name="welcome-content" path="${jboss.home.dir}/welcome-content"/>
</handlers>
<filters>
    <response-header name="server-header" header-name="Server" header-value="WildFly/8"/>
    <response-header name="x-powered-by-header" header-name="X-Powered-By" header-value="Undertow/1"/>
</filters>
</subsystem>

```

Pronto, reinicie o servidor e acesse <https://localhost:8443>.

## Em ambiente de produção

A configuração acima, serve para ambiente de teste e desenvolvimento, para ambiente de produção, é necessário adquirir um certificado SSL comercial ou o gratuito mais famoso que oferece a mesma segurança dos pagos:

<http://www.letsencrypt.org>

## Subsistema Web

Nos primórdios do Wildfly, o subsistema web herdado do Jboss AS 7 foi substituído pelo Undertow. Ganhamos aumento de performance, suporte a websockets e tecnologias mais modernas. Recentemente nas últimas versões ganhamos suporte a HTTP 2.0 para suportar as necessidades de aplicações móveis e otimizar o tráfego de rede entre o cliente e o servidor de aplicações.

Na documentação oficial temos todos os detalhes desse subsistema:

<https://docs.jboss.org/author/display/WFLY/Undertow+subsystem+configuration>

## WebSockets

WebSocket é uma tecnologia que permite a comunicação bidirecional por canais full-duplex sobre um único soquete Transmission Control Protocol (TCP). Ele é projetado para ser executado em browsers e servidores web que suportem o HTML5,[1] mas pode ser usado por qualquer cliente ou servidor de aplicativos. A API WebSocket está sendo padronizada pelo W3C e o protocolo WebSocket está sendo padronizado pelo IETF.

Websocket foi desenvolvido para ser implementado em browsers web e servidores web, mas pode ser usado por qualquer cliente ou aplicação servidor. O protocolo Websocket é um protocolo independente baseado em TCP. Sua única relação com o HTTP é que seu handshake é interpretado por servidores HTTP como uma requisição de Upgrade.

Saiba mais em:

<http://websocket.org/>

Vamos atualizar um de nossos exemplos com uma funcionalidade onde o WebSockets seja importante.

## Alta Disponibilidade em Wildfly

Todo administrador de servidor e profissional de DevOps experiente já se deparou com o cenário de aumento de usuários e requisições no servidor e a diretoria querendo cortar custos. Como garantir um incremento de acessos de usuários de algumas centenas para milhares ou milhões e não ter um parque de servidores infinito? Wildfly te ajuda nisso de forma prática.

Lembrando que essas abordagens precisam ser planejadas e arquitetadas previamente durante a fase de especificação do seu projeto para garantir um melhor aproveitamento dos recursos de hardware e infra-estrutura de rede.

Vamos demonstrar algumas boas práticas para não deixar seus usuários insatisfeitos ou sem acesso:

- **Configuração do Apache e NGInx**
- **Como escolher entre um e outro**
- **Balanceamento de carga entre servidores**
- **Cluster e Domínio de servidores wildfly**

Usaremos como base de consulta a documentação da ultima versão do Wildfly:

[https://docs.wildfly.org/12/High\\_Availability\\_Guide.html](https://docs.wildfly.org/12/High_Availability_Guide.html)

## Demonstração prática

- **Monitoramento**
- **Profiling e Instrumentação**
- **Orientações gerais**
- **byteman**
- **jvisualvm, jstat**
- **Zorka, Zico**
- **Monitoramento**
- **Profiling**

## **2 Otimização e Performance**

- Cenários
- Orientações gerais, estratégias e dicas
- Sequências de investigação e solução
- Otimizações em infraestrutura Wildfly: threads, JVM, pools, perfis
- Outros aspectos significantes para otimização