Chapter No.1
"Installing WildFly"
In this package, you will find:

The author’s biography

A preview chapter from the book, Chapter no.1 "Installing WildFly"

A synopsis of the book’s content

Information on where to buy this book

About the Author

Christopher Ritchie is a Sun Certified Programmer with over 10 years of software experience. Having worked in both the UK and South Africa markets, he has worked on a variety of software applications, ranging from online gaming to telecoms and Internet banking. He has a keen interest in the WildFly application server and is an advocate of Java EE technologies.

He currently works as a technical lead at the company he cofounded, Sports Science Medicine Software, in South Africa. The company's core product is a research-based application that allows the profiling of soccer players through injury and exposure assessment.

Christopher was a technical reviewer for WildFly Performance Tuning, Packt Publishing. You can find him at www.chris-ritchie.com.

I would like to thank my wife, Samantha, for her unwavering support and patience, and my parents for their continued encouragement and support. I would also like to give special thanks to both my dad and Patrick Dzvoti for their help and advice throughout my professional career.

For More Information:

www.packtpub.com/networking-and-servers/wildfly-configuration-deployment-and-administration
WildFly is the new name for the community version of JBoss AS. WildFly is still the most popular Java Enterprise server out there. It is easy to use, has a clean management interface, a powerful command-line tool, a modular architecture, is light, and is lightning quick. If you need product support, it is very easy to move from WildFly to JBoss EAP Server and, importantly, the license and support costs won't break the bank.

This book gently introduces you to WildFly by looking at how to download and install the server. We then move on to look at configuring enterprise services and the various subsystems, and securing your server and applications. The topics become more advanced as the book progresses, so in the later part of the book, we look at high availability, which is achieved through clustering and load balancing.

Whether you are a Java developer who wishes to improve their knowledge of WildFly, or you are a server administrator who wants to gain a better understanding of the inner workings of WildFly, there is something in this book for everyone.

What This Book Covers

Chapter 1, Installing WildFly, introduces you to the WildFly server. You are taken through the installation of Java and WildFly and learn to start, stop, and restart the server. You will also discover the purpose of the various folders within the WildFly install directory and gain a basic understanding of the WildFly kernel. Lastly, you will learn to install the WildFly server adaptor in Eclipse.

Chapter 2, Configuring the Core WildFly Subsystems, provides a detailed description of the anatomy of the standalone configuration file. You will be introduced to the concepts of modules and subsystems, and then you will learn in detail how to configure two of the core subsystems, the logging subsystem, and the thread-pool subsystem.

Chapter 3, Configuring Enterprise Services, teaches you to configure enterprise services and components, such as transactions, connection pools, Enterprise JavaBeans, and JMS. You will also learn to install JDBC drivers and configure database connections.

Chapter 4, The Undertow Web Server, explains the architecture of Undertow and teaches you how to configure the servlet container. You will also learn how to serve static content and configure JSPs. In this chapter, we create a simple web application using JSF, EJB, and JPA and teach you to deploy the application using the Eclipse WildFly Server adaptor.

For More Information:
www.packtpub.com/networking-and-servers/wildfly-configuration-deployment-and-administration
Chapter 5, Configuring a WildFly Domain, teaches you how to manage a WildFly domain. It covers the domain and host controller configuration and outlines the differences between a server domain and multiple standalone server instances.

Chapter 6, Application Structure and Deployment, explains the structure of web and enterprise archives and how they are packaged. You will also learn in detail about the various ways to deploy your application to the WildFly server. We also explain how class loading works in WildFly.

Chapter 7, Using the Management Interfaces, introduces more advanced command-line interface commands, such as those for adding datasources and configuring JMS. We also provide a high-level overview of the web admin console.

Chapter 8, Clustering, provides detailed examples on clustering standalone and domain servers. You will also learn about JGroups and how to cluster the enterprise components, such as Messaging and Hibernate.

Chapter 9, Load-balancing Web Applications, explains the benefits of using the Apache web server with WildFly. You will also learn how to load-balance your web applications with mod_jk, mod_proxy, and mod_cluster.

Chapter 10, Securing WildFly, teaches you how to configure the security subsystem. We cover various login modules, such as database login and LDAP. We also look at securing enterprise components, such as Enterprise JavaBeans and web services. We then look at securing the management interfaces.

Chapter 11, WildFly, OpenShift, and Cloud Computing, discusses the advancement of cloud computing and the benefits it offers to your company. We see how OpenShift can be used to ease software development and aid rapid deployment of your applications to cloud servers.

Appendix, CLI References, provides a quick reference to some of the more commonly used commands in the CLI.

For More Information:
www.packtpub.com/networking-and-servers/wildfly-configuration-deployment-and-administration
Installing WildFly

The Java language has undergone many changes since its first release and will continue to adapt to meet the needs of the developer. Oracle, which acquired Sun in 2010, stated that its high-level Java strategy is to enhance and extend the reach of Java to new and emerging software development objectives; simplify, optimize, and integrate the Java platform into new deployment architectures; and invest in the Java developer community allowing for increased participation.

This has certainly been true in the Enterprise edition of Java, the main focus of which has been improved developer productivity, providing support for HTML5, and meeting enterprise demands. Out of all the Enterprise Java releases, Java EE 7 has been the most transparent and open to community participation. By allowing public feedback, the demands of the community can be realized and used to help shape Java EE 7 for the better, ultimately adding to the growth and success of Enterprise Java.

In addition, a large number of open source projects are used within the application server, such as Hibernate and Undertow. Integrating all these libraries does not come without a price because each library has evolved with complexity and requires more and more additional libraries to work.

As most IT experts agree, the challenge for today’s application servers is to combine a rich set of features requested by customers along with a lightweight and flexible container configuration.

What's new in WildFly 8?

WildFly 8 is the direct continuation to the JBoss AS project. The renaming of the community version of JBoss AS was done to reduce confusion between the open source JBoss server, the JBoss community, and the JBoss Enterprise Application Platform (JBoss EAP). WildFly 8 is free and open source, with support coming from the JBoss community, whereas JBoss EAP is a licensed product that comes with support from RedHat.

For More Information:
www.packtpub.com/networking-and-servers/wildfly-configuration-deployment-and-administration
The most notable updates in WildFly 8 from earlier versions are as follows:

- **Java EE7 certification**: WildFly is a fully compliant Java EE enterprise server, which means that it provides reference implementations for all **Java Specification Requests (JSRs)** that make up Java EE 7. JSRs are basically change requests for the Java language. For more information on how JSRs work, refer to https://www.jcp.org/en/jsr/overview.

- **Arrival of Undertow**: JBoss Web has been completely removed and replaced with Undertow. Undertow is a cutting-edge web server that supports non-blocking and blocking handlers, web sockets, and asynchronous servlets. It has been designed for scalability and maximum throughput. It is easy to use, easy to configure, and is highly customizable.

- **Port Reduction**: The number of open ports has been greatly reduced in WildFly. Only two ports are open: 8080 and 9990. This has been achieved by multiplexing protocols over HTTP using the HTTP upgrade feature of Undertow.

- **Security Manager**: You can now configure per-deployment security permissions.

- **Logging**: Several enhancements have been made to WildFly logging. You can now view logfiles via the management interface, define custom formatters, and configure logging per-deployment.

- **Clustering**: Clustering in WildFly is heavily refactored and includes many new features, including web sessions, single sign-on, and mod_cluster support for Undertow. There is also a new public clustering API and new @Stateful EJB caching implementation.

- **Command-line interface (CLI)**: You now have the ability to define an alias when connecting to a server, and the CLI GUI has additional functionality allowing you to explore any node in the tree.

In this chapter, we will cover the following topics:

- Installing the Java environment
- Installing WildFly 8
- Installing JBoss tools
- Exploring the application server filesystem
- Understanding the WildFly kernel
Chapter 1

Getting started with the application server

As far as hardware requirements are concerned, you should be aware that the server distribution, at the time of writing, requires about 150 MB of hard disk space and allocates a minimum of 64 MB and a maximum of 512 MB for a standalone server.

In order to get started, we are going to perform the following steps:

1. Download and install the Java Development Kit.
2. Download and install WildFly 8.
3. Download and install the Eclipse development environment. While we will use Eclipse in this book, you are free to use your IDE of choice.

At the end of this chapter, you will have all the required software installed and will be ready to start working with the application server.

Installing the Java environment

WildFly is written in Java; therefore it needs a Java Virtual Machine (JVM) in which to run, along with the standard edition Java libraries. So, before we can get started setting up or learning about WildFly, we first need to install the Java Development Kit (JDK).

To use WildFly, you will need at least Java SE 7 or above. Although there is no plan to use Java 8 language changes within the WildFly 8.x source code, WildFly is compiled against Java 8. It is recommended that you use the latest version of Java SE 8 to run WildFly.

So, let's move to the Oracle download page, http://www.oracle.com/technetwork/java/javase/downloads/index.html, which now hosts all JDK downloads, as shown in the following screenshot:

![Java SE Downloads](image)

For More Information:
www.packtpub.com/networking-and-servers/wildfly-configuration-deployment-and-administration
Installing WildFly

This will take you to the download page for the latest JDK. At the time of writing, this was Java 8 update 5. You will need to accept the license agreement before downloading the JDK. Choose to download the latest version of Java for your operating system. Have a look at the following screenshot:

<table>
<thead>
<tr>
<th>Product / File Description</th>
<th>File Size</th>
<th>Download</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux x86</td>
<td>133.58 MB</td>
<td>jdk-8u5-linux-i586.rpm</td>
</tr>
<tr>
<td>Linux x86</td>
<td>152.5 MB</td>
<td>jdk-8u5-linux-i586.tar.gz</td>
</tr>
<tr>
<td>Linux x64</td>
<td>133.87 MB</td>
<td>jdk-8u5-linux-x64.rpm</td>
</tr>
<tr>
<td>Linux x64</td>
<td>151.64 MB</td>
<td>jdk-8u5-linux-x64.tar.gz</td>
</tr>
<tr>
<td>Mac OS X x64</td>
<td>207.79 MB</td>
<td>jdk-8u5-macosx-x64.dmg</td>
</tr>
<tr>
<td>Solaris SPARC 64-bit (SVR4 package)</td>
<td>135.68 MB</td>
<td>jdk-8u5-solaris-sparcv9.tar.Z</td>
</tr>
<tr>
<td>Solaris SPARC 64-bit</td>
<td>95.54 MB</td>
<td>jdk-8u5-solaris-sparcv9.tar.gz</td>
</tr>
<tr>
<td>Solaris x86 (SVR4 package)</td>
<td>135.9 MB</td>
<td>jdk-8u5-solaris-x64.tar.Z</td>
</tr>
<tr>
<td>Solaris x64</td>
<td>93.19 MB</td>
<td>jdk-8u5-solaris-x64.tar.gz</td>
</tr>
<tr>
<td>Windows x86</td>
<td>151.71 MB</td>
<td>jdk-8u5-windows-i586.exe</td>
</tr>
<tr>
<td>Windows x64</td>
<td>155.18 MB</td>
<td>jdk-8u5-windows-x64.exe</td>
</tr>
</tbody>
</table>

The download will take a few minutes depending how fast your network is.

Installing Java on Linux

Installing Java on Linux is very straightforward. Once the download is complete, extract the tar.gz file to your chosen install location. This command extracts the archive to your current directory:

```
tar -xsvf jdk-8u5-linux-x64.tar.gz
```

Next, you need to add the path as an environment variable. This can be achieved by adding the following lines to your user profile script (the .profile file found in your home directory):

```
export JAVA_HOME=/installDir/jdk1.8.0_05
export PATH=$JAVA_HOME/bin:$PATH
```

For More Information:
www.packtpub.com/networking-and-servers/wildfly-configuration-deployment-and-administration
Installing Java on Windows

Windows users can simply run the executable (.exe) file to start the installation. The name of the installer varies depending on the operating system and your system architecture (32-bit or 64-bit); however, the steps will be the same—just the name will change. At the time of writing, the installer for the latest version of Java for 64-bit Windows is called `jdk-8u5-windows-x64.exe`.

When using Windows, you should stay away from installation paths that include empty spaces, such as `C:\Program Files`, as this leads to some issues when referencing the core libraries. An installation path such as `C:\Software\Java` or simply `C:\Java` is a better alternative.

When the installation is complete, you will need to update a couple of settings on the computer so that it will know where to find Java. The most important setting is `JAVA_HOME`, which is directly referenced by the WildFly startup script.

If you are running Windows XP/2000, follow these steps:

1. Right-click on My Computer, and select Properties from the context menu.
2. On the Advanced tab, click on the Environment Variables button.
3. Then, in the System variables box, click on New.
4. Name the new variable `JAVA_HOME`, and give a value of the path to your JDK installation; I recommend something like `C:\Java\jdk1.8.0_05`.

For More Information:

www.packtpub.com/networking-and-servers/wildfly-configuration-deployment-and-administration
Installing WildFly

Windows 7 tip

Because of increased security in Windows 7, standard users must have User Account Control (UAC) turned on to change the environment variables, and the change must be completed via user accounts. In the User Accounts window, under Tasks, select Change my environment variables. Use the New, Edit, or Delete button to amend environment variables.

5. Now it's time to modify the system's PATH variable. Double-click on the PATH system variable. In the box that pops up, navigate to the end of the Variable Value line, add a semicolon to the end, and then add the path to your JDK. This will be something like %JAVA_HOME%\bin.

Installing WildFly 8

The WildFly application server can be downloaded for free from the WildFly site, http://www.wildfly.org/downloads/. Have a look at the following screenshot:

You will notice that there is an option to download a minimalistic core distribution. This is aimed at developers who want to build their own application runtime using the WildFly 8 architecture.

Choose to download the full Java EE7 distribution. Like JBoss AS 7, WildFly does not come with an installer. It is simply a matter of extracting the compressed archive to a location of your choosing.

For More Information:

www.packtpub.com/networking-and-servers/wildfly-configuration-deployment-and-administration
Linux users can extract the file using the `tar` or `unzip` command (depending on the type of compressed file you downloaded):

```
tar -xzvf wildfly-8.1.0.Final.tar.gz
unzip wildfly-8.1.0.Final.zip
```

For those of you using Windows, you can use WinZip or WinRAR, taking care to choose a folder that does not contain empty spaces.

**Security warning**

Unix/Linux users should be aware that WildFly does not require root privileges, as none of the default ports used by WildFly are below the privileged port range of 1024. To reduce the risk of users gaining root privileges through WildFly, install and run WildFly as a non-root user.

## Starting WildFly

After installing WildFly, it is wise to perform a simple startup test to validate that there are no problems with your Java configuration. To test your installation, move to the `bin` directory of your WildFly install and issue the following command:

- For Linux/Unix users:
  ```
  $ ./standalone.sh
  ```

- For Windows users:
  ```
  > standalone.bat
  ```

The following screenshot shows a sample WildFly 8 startup console:

![WildFly Startup Console](image)

For More Information:

Installing WildFly

The preceding command starts up a WildFly standalone instance that's equivalent to starting the application server with the run.sh script used by releases prior to JBoss AS 7. The run.sh file remains in the WildFly bin directory but is merely a placeholder and will not start the application server.

Notice how fast the application server starts. This is due to the modular architecture of WildFly. Essential services are started concurrently on boot-up, and non-critical services are started only when needed, resulting in an exceptionally fast startup. Local caching means that the server will start even quicker second time round!

If you need to customize the startup properties of your application server, then you need to open and modify the standalone.conf file (or standalone.conf.bat for Windows users). This file contains the memory requirements of WildFly. The following is the Linux core section of it:

```bash
if [ "x$JAVA_OPTS" = "x" ]; then
  JAVA_OPTS="-Xms64m -Xmx512m -XX:MaxPermSize=256m -Djava.net.preferIPv4Stack=true"
  JAVA_OPTS="$JAVA_OPTS -Djboss.modules.system.pkgs=$JBOSS_MODULES_SYSTEM_PKGS -Djava.awt.headless=true"
fi
```

Java SE 8 users

PermGen has been replaced with Metaspace in Java 8. If you are using Java 8, then remove the -XX:MaxPermSize=256m property from the standalone.conf file, and replace it with -XX:MaxMetaspaceSize=256m. This will prevent VM warnings being printed to your WildFly logs on startup.

By default, the application server starts with a minimum heap space memory requirement of 64 MB and a maximum requirement of 512 MB. This will be just enough to get started; however, if you need to run a core Java EE application on it, you will likely require a minimum of 1 GB of heap space. More realistically, you will need 2 GB or more depending on your application type. Generally speaking, 32-bit machines cannot execute a process whose space exceeds 4 GB; however, on 64-bit machines, there's essentially no limit to process the size.

You can verify that the server is reachable from the network by simply pointing your browser to the application server's welcome page, which is reachable by default at the well-known address: http://localhost:8080. Have a look at the following screenshot:

For More Information:
www.packtpub.com/networking-and-servers/wildfly-configuration-deployment-and-administration
Connecting to the server with the command-line interface

If you have been using releases of the application server prior to JBoss AS 7, you might have heard about the `twiddle` command-line utility that queries the MBeans installed on the application server. This utility was replaced in JBoss AS 7 and is still used in WildFly. Its replacement is a more sophisticated interface named the **command-line interface (CLI)**, which can be found in the `JBOSS_HOME/bin` folder.

**References to JBOSS_HOME**

Although the community version of JBoss AS has been renamed to WildFly, you will see that the properties in the startup scripts continue to use the property, `JBOSS_HOME`, to reference the install directory of WildFly. For this reason, we will continue to use `JBOSS_HOME` when referring to the root install of WildFly.

For More Information:

www.packtpub.com/networking-and-servers/wildfly-configuration-deployment-and-administration
Installing WildFly

Just launch the `jboss-cli.sh` script (or `jboss-cli.bat` for Windows users), and you will be able to manage the application server via a shell interface, as shown in the following screenshot. Bear in mind that the server needs to be running in order to connect via the CLI.

Once you are in the shell session, if you are unsure of what commands can be issued, you can simply press the Tab button to display all possible commands. If your command is partly typed, and there is only one possible matching command, your command will be autocompleted. Those of you who use Linux will be used to this type of command-line assistance.

In the preceding screenshot, we have just connected to the server using the `connect` command, which, by default, uses the loopback server address and plugs into port number 9990.

The CLI is discussed in depth in Chapter 7, Using the Management Interfaces, which is all about the server-management interfaces. We will have an initial taste of its basic functionalities in the following sections, to get you accustomed to this powerful tool.

Stopping WildFly

Probably the easiest way to stop WildFly is to send an interrupt signal using Ctrl + C. This should be done in the same console window in which you issued the startup command, that is, where the server is running.

For More Information:

www.packtpub.com/networking-and-servers/wildfly-configuration-deployment-and-administration
However, if your WildFly process was launched in the background or is running on another machine (see in the following sections), then you can use the CLI interface to issue an immediate `shutdown` command as follows:

```
[disconnected /] connect
[standalone@localhost:9990 /] shutdown
[disconnected /]
```

### Locating the shutdown script

There is actually one more option to shut down the application server, which is pretty useful if you need to shut down the server from within a script. This option consists of passing the `--connect` option to the admin shell, thereby switching off the interactive mode as follows:

```
jboss-cli.sh --connect command=:shutdown       # Unix / Linux
jboss-cli.bat --connect command=:shutdown      # Windows
```

### Stopping WildFly on a remote machine

Shutting down an application server running on a remote machine is just a matter of connecting and providing the server's remote address to the CLI:

```
[disconnected /] connect 192.168.1.10

[192.168.1.10:9990 /] shutdown
```

Remotely accessing WildFly via the CLI requires authentication. Check out Chapter 10, Securing WildFly, for more information about it. It also requires that the management interface on the remote WildFly install is opened to allow remote connections. This is covered in detail in Chapter 7, Using the Management Interfaces.

### Restarting WildFly

The CLI contains a lot of useful commands. One of the most helpful options is the ability to reload all or part of the server configuration using the `reload` command.
Installing WildFly

When issued on the root node path of the server, WildFly reloads all the services configuration, as shown in the following command:

```
[disconnected /] connect

[standalone@localhost:9990 /] reload
```

Installing the Eclipse environment

Although the main focus of this book is the administration of the WildFly application server, we are also concerned with application packaging and deployment. For this reason, we will sometimes add examples that require a development environment to be installed on your machine.

The development environment used in this book is Eclipse. Eclipse is known by developers worldwide and contains a huge set of plugins, building on its core functionality. If you are comfortable with another IDE, then feel free to use it, but this book will demonstrate Eclipse only. At the time of writing this, only Eclipse and NetBeans have plugins for WildFly.


From this page, download the latest Enterprise edition. The compressed package contains all the Java EE plugins already installed and requires about 248 MB of disk space. Have a look at the following screenshot:

![Eclipse IDE for Java EE Developers](https://www.eclipse.org/downloads/)

If you are using Java 8, you should make sure you download Eclipse Luna (4.4) or the patched Version of 4.3.

Once you have downloaded Eclipse, unzip it to a folder of your choice. The extracted folder will be called eclipse. To start Eclipse, navigate to the eclipse folder and run:

```
$ ./eclipse
```

Windows users can simply double-click on the executable file contained in the eclipse folder (the one with the big, blue, round eclipse icon).

For More Information:

Installing JBoss tools

The next step is to install the WildFly 8 adapter, which is a part of the suite of plugins named JBoss tools. Installing new plugins in Eclipse is pretty simple; just perform the following steps:

1. From the menu, navigate to Help | Eclipse Marketplace.
2. Then, search for the plugin you want to install (in this case, type jboss tools).
3. Finally, click on Install as shown in the following screenshot:

   ![Eclipse Marketplace](image)

   Make sure you select the version of JBoss tools that matches your version of Eclipse, for example, Luna or Kepler. In this case, we are using Eclipse Luna, so I have selected the Luna version of JBoss tools. If you want to install just the WildFly adapter, select JBossAS Tools. Agree to the terms and click on OK.

   Restart Eclipse when prompted to do so.

You can now set up the WildFly server in Eclipse by performing the following steps:

1. Navigate to New | Server.
2. Expand the JBoss Community node.
3. Select the option, **WildFly 8**, as shown in the following screenshot:

![Server Selection Screenshot]

4. Make sure you select your installed Java 8 JRE.

5. Point the home directory to that of your WildFly root directory, as shown in the following screenshot:

![Runtime Configuration Screenshot]

For More Information:
www.packtpub.com/networking-and-servers/wildfly-configuration-deployment-and-administration
Exploring the application server filesystem

Now that we are done with the installation of all the necessary tools, we will concentrate on the application server structure. The first thing you’ll notice when you browse through the application server folders is that its filesystem is basically divided into two core parts: the dichotomy reflects the distinction between standalone servers and domain servers.

The concept of a domain server is not new in the market of application servers, however, it was only introduced in JBoss with AS 7 as a way to manage and coordinate a set of instances of the application server. An application server node which is not configured as part of a domain is qualified as a standalone server. A standalone server resembles, in practice, a single instance of the application server you used to see in releases of the application server prior to JBoss AS 7.

We will discuss the concept of domains in detail in Chapter 5, Configuring a WildFly Domain. For the time being, we will explore the different filesystem structures for both kinds of servers.

From a bird’s-eye perspective, we can see that the main filesystem is split in two: one section that is pertinent to domain servers and another that is relative to standalone servers. The following diagram depicts the tree of the application server:

For More Information:
www.packtpub.com/networking-and-servers/wildfly-configuration-deployment-and-administration
Installing WildFly

In the next section, we will dig deeper into the folder structure of the WildFly application server, dissecting its content and looking at what it is used for.

The bin folder

The bin folder is where you will find all your startup scripts, such as standalone.sh and domain.sh. In addition to the startup scripts, you can find standalone.conf, which can be used to customize WildFly's bootstrap process.

As you saw earlier, the bin folder also includes the jboss-cli.sh script (jboss-cli.bin for Windows users), which starts the interactive CLI. You will also find various other useful scripts, such as add-user.sh and vault.sh. This folder also contains the web services utility scripts (wsconsume.sh and wsprovide.sh) used to generate the web services definition language and the corresponding Java interfaces.

There are several subfolders within the bin directory. The service folder and the init.d folder contain programs that allow you to install WildFly as service on Windows and Linux, respectively.

The docs folder

The docs folder contains two subfolders, examples and schema. The schema folder contains all the .xsd schema definition files used by the configuration as schema.

The examples folder contains numerous configuration examples, from a minimalistic standalone example to an ec2 HA example (HA meaning high availability, and ec2 referring to Amazon Elastic Compute Cloud).

The domain folder

The next folder is the domain folder, which contains the domain structure split across a set of folders:

- The configuration folder contains all the configuration files:
  - The main configuration file is domain.xml, which contains all services that are used by the nodes of the domain. It also configures the socket-binding interfaces for all services.
  - Another key file for domains is host.xml, which is used to define the host controller (HC).

For More Information:
www.packtpub.com/networking-and-servers/wildfly-configuration-deployment-and-administration
The last file contained in the configuration folder is `logging.properties`, which is used to define the logging format of the bootstrap process for both the process controller (PC) and host controller.

• The `content` folder is used as a repository to store deployed modules.
• The `lib` folder hosts the subfolder `ext`, which is there to support Java SE/EE style extensions. Some of the application server deployers are able to scan this folder for additional libraries that are picked up by the local class loader. Nevertheless, this approach is not recommended and is maintained only for compliance with the language specifications. The `modules` folder should be used to install your libraries within WildFly.
• The `log` folder, as you might imagine, contains the logging output of the domain. The file, by default, is truncated every time the server is rebooted.
• The `servers` folder holds a set of subfolders for each server defined in the configuration file. The most useful directory contained beneath each server is the `log` folder, which is the location where single instances emit their log.
• The `data` folder is used by the application server to store its runtime data, such as transaction logging.
• Finally, the `tmp` folder is used to store temporary files written by the server.

The standalone folder

If you are running the application server in standalone mode, this is the part of the filesystem you will be interested in. Its structure is quite similar to the `domain` folder with the notable exception of a `deployment` folder. Let's proceed with order. Just below the `standalone` folder, you will find the following set of subdirectories:

• `configuration`
• `data`
• `deployments`
• `lib`
• `log`
• `tmp`
Installing WildFly

The content and use of these subdirectories is explained as follows:

- **The configuration** folder contains the application server configuration files. As a matter of fact, the application server ships with a set of different configuration files, each one using a different set of extensions. Launching the standalone startup script without passing in any parameters will, by default, use the standalone.xml configuration file.

  Besides standalone.xml, this folder contains the logging.properties file that configures the logging of the bootstrap process. The other files you will find here are mgmt-users.properties and mgmt-group.properties, which can be used to secure the management interfaces. Security is discussed in detail in Chapter 10, Securing WildFly.

- **The data** folder is used by the application server to store its runtime data, such as transaction logging.

- **The deployments** folder is the location in which users can place their deployment content (for example, WAR, EAR, JAR, and SAR files) to have it automatically deployed in the server runtime. Users, particularly those running production systems, are encouraged to use WildFly's management APIs to upload and deploy deployment content instead of relying on the deployment scanner subsystem that periodically scans this directory. See Chapter 6, Application Structure and Deployment, for more details.

- **The lib folder** hosts the subfolder ext, which is used to define extensions of the application server. The same considerations for the domain's lib path apply here.

- **The log folder** contains the logs emitted by the standalone instance of the application server. The default log file, named server.log, is, by default, truncated every time the server is rebooted. This can be configured within the standalone.xml file.

- **The tmp folder** is used to save temporary files written by WildFly.

### The welcome-content folder

The welcome-content folder contains the default page, which is loaded when you browse to the root of your application server (http://localhost:8080). In terms of web server configuration, this is the **Web root context**.

### The modules folder

Beneath the modules folder, you will find the application server's set of libraries, which are a part of the server distribution.
Historically, JBoss AS releases used to manage their set of libraries in different ways. Let's recap to bring about some order. Earlier, Release 4.x was used to define the core server libraries into the JBOSS_HOME/server libraries. Thereafter, each server definition had its specific library in the server/<servername>/lib folder.

This approach was pretty simple, however, it led to a useless proliferation of libraries that were replicated in the default/all server distribution.

Releases 5.x and 6.x had the concept of the common/lib folder, which was the main repository for all modules that were common to all server definitions. Each server distribution still contained a server/<servername>/lib path for the libraries that were specific to that server definition. Unchanged from the earlier release was the repository for core server modules comprised by JBOSS_HOME/server.

JBoss AS 7 followed a more modular approach improving over all the earlier approaches. This modular approach remains unchanged in WildFly. The server bootstrap library, jboss-modules.jar, can be found in the root of the application server. This single archive is all you need to bootstrap WildFly's application server kernel.

The main system modules are located in the system/layers/base folder under the modules folder. This has changed slightly in WildFly as, in JBoss AS 7, all modules were defined directly in the modules folder.

The following table outlines the diverse approaches used across different server releases:

<table>
<thead>
<tr>
<th>AS release</th>
<th>Bootstrap libraries</th>
<th>Server libraries</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.x</td>
<td>JBOSS_HOME/server</td>
<td>JBOSS_HOME/server/&lt;servername&gt;/lib</td>
</tr>
<tr>
<td>5.x and 6.x</td>
<td>JBOSS_HOME/server</td>
<td>JBOSS_HOME/common/lib and JBOSS_HOME/server/&lt;servername&gt;/lib</td>
</tr>
<tr>
<td>7.x and 8.x</td>
<td>JBOSS_HOME/jboss-modules.jar</td>
<td>JBOSS_HOME/modules</td>
</tr>
</tbody>
</table>

Listing all the modules will take up too much space, however, the module repository layout is often the same as the module name. For example, the org.jboss.as.ejb3 module can be found in the org/jboss/as.ejb3 subfolder of the modules folder. This approach to organizing the modules certainly makes sense, and if you are used to a maven repository layout structure, you will have no problem getting your head around it.

In the last section of this chapter, we will see how modules are actually loaded by the application server.

---

For More Information:
Understanding WildFly's kernel

WildFly's kernel was redesigned in JBoss AS 7. Understanding the details of the modular kernel will help you understand concepts introduced later in the book. The kernel is based on two main projects, as follows:

- **JBoss Modules**: This project handles class loading of resources in the container. You can think about JBoss modules as a thin bootstrap wrapper for executing an application in a modular environment.

- **Modular Service Container (MSC)**: This project provides a way to install, uninstall, and manage services used by a container. MSC further enables resource injection into services and dependency management between services.

The following diagram depicts the basic architecture of WildFly's server kernel:

![Diagram of WildFly's server kernel]

With this information, we can now progress to the loading of server modules.

For More Information:
www.packtpub.com/networking-and-servers/wildfly-configuration-deployment-and-administration
Loading application server modules

Learning more about JBoss modules is essential if you want to understand the server configuration discussed in the next few chapters. At its heart, a module is really just a wrapper for a JAR file but treated by the application container as a module. The reason for this is class loading and dependency management, as each module can be treated as a pluggable unit, as depicted by the next diagram. WildFly has two different types of modules; the only difference between them is the way they are packaged:

- Static modules
- Dynamic modules

Have a look at the following screenshot:

Using a static module is the simplest way to load a module, and it's used as the default module when starting up the application server. Static modules are defined within the JBOSS_HOME/modules/system/layers/base directory. Each module has a configuration file called module.xml. The following example shows the contents of the javax.batch.api module.xml file:

```xml
<module xmlns="urn:jboss:module:1.3" name="javax.batch.api">
  <resources>
    <resource-root path="jboss-batch-api_1.0_spec-1.0.0.Final.jar"/>
  </resources>
</dependencies>
```

For More Information:
www.packtpub.com/networking-and-servers/wildfly-configuration-deployment-and-administration
Installing WildFly

```xml
<module name="javax.api"/>
<module name="javax.enterprise.api"/>
</dependencies>
</module>
</dependencies>
</module>

As you can see, a module definition contains two main elements, the resources defined in the module (and their path) and the module's dependencies. In this example, the main resource is jboss-batch-api_1.0_spec-1.0.0.Final.jar, contained in the same folder as the module.xml file. It has dependencies on two other modules, javax.api and javax.enterprise.api.

A module which is defined with a main-class element is said to be executable. In other words, the module name can be listed on the command line, and the standard static main(String[]) method in the named module's main-class will be loaded and executed.

Creating custom static modules is useful should you have many applications deployed to your server, which rely on the same third-party libraries. This means that you do not have to deploy multiple applications with the same bundled libraries. The other benefit to creating custom static modules is that you can declare explicit dependencies on other static modules. Installing modules is covered in Chapter 3, Configuring Enterprise Services, in which we install a JDBC driver as a module.

The other way to approach the module repository is by using dynamic modules. This can be achieved in two ways, as follows:

- Firstly, we can add the module information, such as its dependencies, within the MANIFEST file within your JAR, for example, in the Main class mypackage/MyClass:
  Dependencies: org.jboss.logging

- The second way to do this is by adding the dependency to the jboss-deployment-structure.xml file, as shown in the following code:
  ```xml
  <jboss-deployment-structure>
  <deployment>
    <dependencies>
      <module name="org.jboss.logging" />
    </dependencies>
  </deployment>
  </jboss-deployment-structure>
  ```

We will cover this in more detail in Chapter 6, Application Structure and Deployment, in which we explain class loading.

For More Information:
www.packtpub.com/networking-and-servers/wildfly-configuration-deployment-and-administration
Summary
In this chapter, we outlined the latest features that come shipped with WildFly.

We have seen that WildFly is composed of modular architecture, and that the kernel of WildFly is made up of two separate projects: JBoss Modules and MSC.

This modular architecture results in an exceptionally light kernel that is able to load modules as required, resulting in a quicker startup time.

The physical structure of the application server reflects the dichotomy between standalone servers and domain servers, the former being a single node instance and the latter a set of managed resources controlled by a domain controller and a host controller.

In the next chapter, we will dig deeper into the details of how to configure the application server, focusing our attention on the standalone server configuration file (standalone.xml), which contains the configuration for both the core application server and the stack of enterprise services running on top of it.

For More Information:
www.packtpub.com/networking-and-servers/wildfly-configuration-deployment-and-administration
Where to buy this book


Free shipping to the US, UK, Europe and selected Asian countries. For more information, please read our shipping policy.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.