WildFly Domain and Standalone Modes

In this appendix, you will learn about the following:

- Domain mode and server groups
- Understanding the structure of domain.xml
- Understanding the structure of host.xml
- Understanding the structure of standalone.xml

Domain mode and server groups

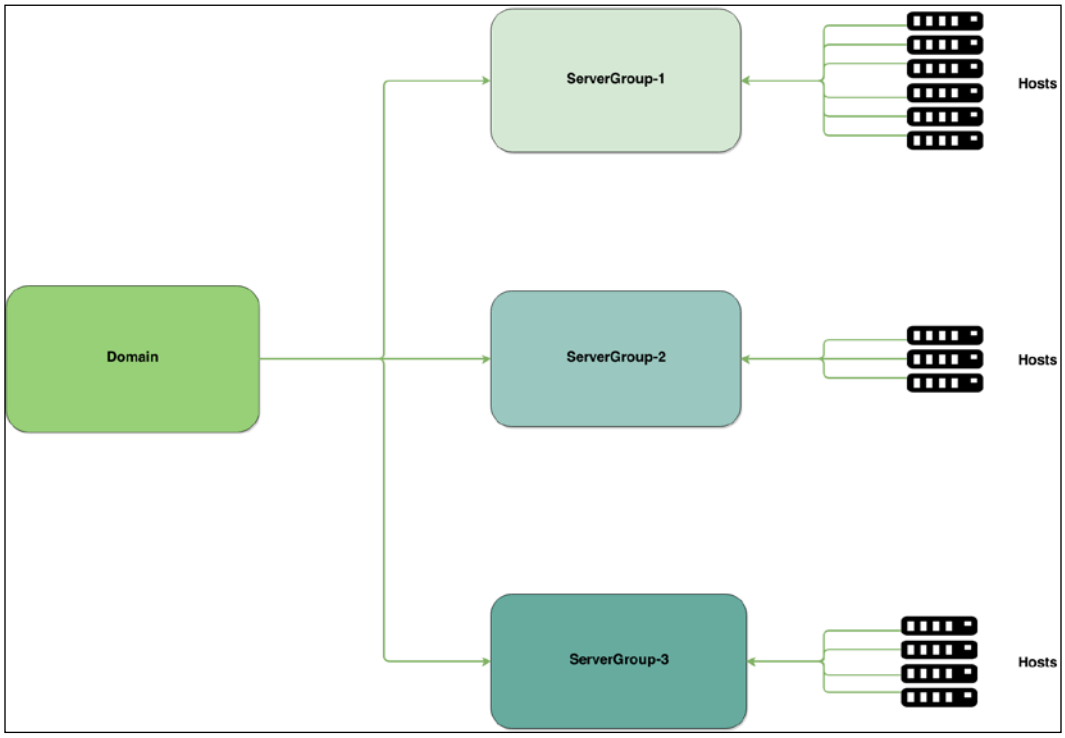
WildFly comes with two operational modes: standalone and domain. You can think of the standalone mode as a WildFly instance running on its own. Every WildFly standalone instance has its own management interface and configuration; thus, you can manage one instance at a time. The standalone mode is governed by the `standalone.xml` file.

On the other hand, WildFly instances running in the domain mode share the same main management interface called domain controller. Their management interfaces are still active but only available for the domain controller. Within the domain controller, you can issue commands on all running WildFly instances. The domain mode is governed by the `domain.xml` and `host.xml` files. The first one comprises of all the profiles and server groups, configured. The second one comprises of all servers that belong to the server groups defined in the `domain.xml`.

Without going any deeper, let's start with the three main concepts:

- Domain
- Server groups
- Hosts

Now let's summarize them in a diagram:



Domain, server groups, and hosts

That is, you define a list of server groups within your `domain.xml` file, and then within the `host.xml` file, you define which hosts (called server in the configuration file) belong to which server group.

Domain

The domain is a JVM process, actually called the **domain controller** (**DC**), and is the entry point to your WildFly management system. In this book, and even outside the book (articles, blogs, and so on), we will be referring to the DC as the master. It is responsible for pushing out configuration to all its slaves; they are the host controllers, as we will see shortly.

From a design point of view, the DC should be running in a separate server (virtual or physical), without any running WildFly instances. This is just to prevent having a machine doing too many things—we will go deep into it in *Chapter 3*, *Running WildFly in Domain Mode*.

Server group

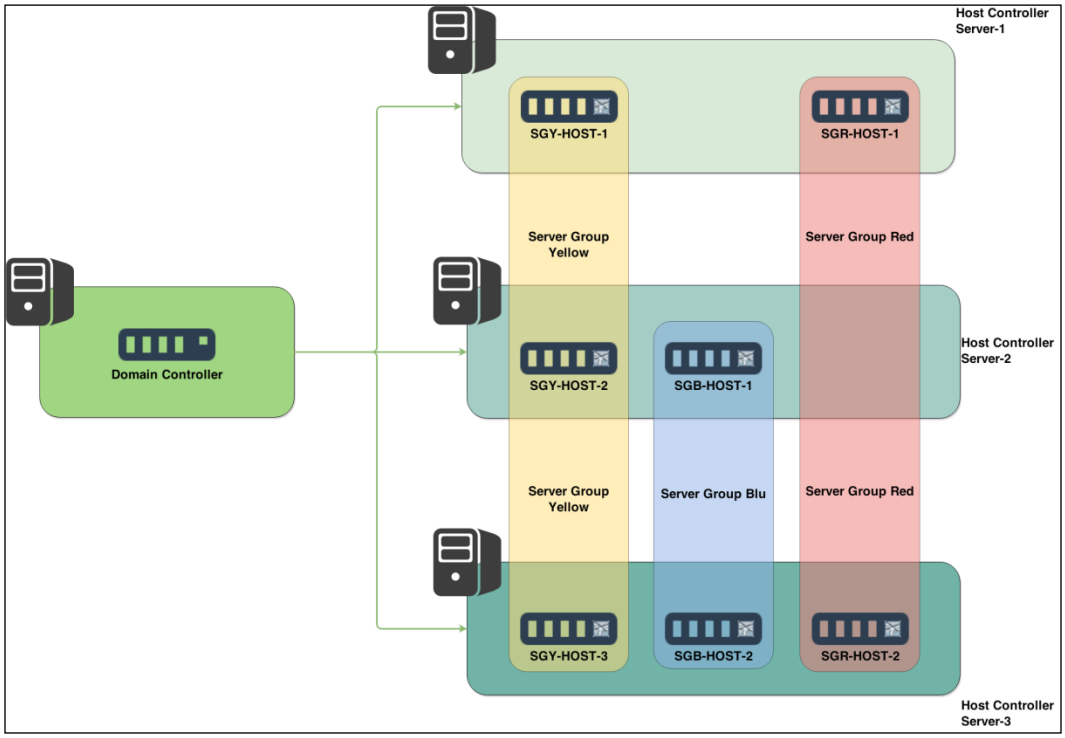
A **server group** (**SG**) is a logical representation of the WildFly configuration. It is composed of one or more server instances called hosts. Each host that belongs to an SG shares the same configuration from both the profile and the deployments point of view.

Host

The host is a JVM process, actually called the **host controller** (**HC**). It is the entry point to your WildFly management system, but specific for the hosts that belong to that host controller. We will be referring to the HC as the slave. The DC will communicate with the HC to push new configuration out to the relative hosts.

Both the domain controller and the host controller are responsible for remote management.

The previous diagram just explained the hierarchy by which you can manage your server groups and hosts. Instead, the following diagram tries to explain how you can really design and configure your WildFly instances in the domain mode.



An example of a WildFly server group

We can deduce the following from the preceding diagram:

- There is a server that acts as the domain controller and does not have any running host on it (the green one on the left)
- There are three distinct server groups named: **Server Group Yellow** (**SGY**), **Server Group Blu** (**SGB**), and **Server Group Red** (**SGR**)
- The domain controller communicates with three HCs: **Host Controller Server-1**, **Host Controller Server-2**, and **Host Controller Server-3**
    - **SGY** is composed of three hosts spanning three different machines
    - **SGB** is composed of two hosts spanning two different machines
    - **SGR** is composed of two hosts spanning two different machines

As you can see, all your server groups and hosts can be organized as you like, depending on your requirements and the available resources (that is, RAM, CPUs, and so on).

In the domain mode, you have two main configuration files:

- `domain.xml`: Here you describe profiles and the server groups
- `host.xml`: Here you define how server groups are composed, that is, the number of hosts

Understanding the structure of domain.xml

We are now going to look at the structure of `domain.xml` and understand it.

In *Chapter 2*, *Running WildFly in Standalone Mode*, which was dedicated to the standalone mode, we saw that WildFly installation comes with different `standalone.xml` files, each one relative to a profile: `default`, `ha`, `full`, and `full-ha`. If you skipped that chapter, we will recap the profiles in the following table:

Profile	Description
`default`	This is the `default` profile, which provides all technologies to run the Java EE 7 Web Profile Certified application. With this profile, you can use logging, DataSource, JNDI lookups, Servlet & JSP, mail services, webservices (SOAP and REST), JSF, JPA 2, EJB 3.2, CDI 1.1, and Batch.
`ha`	This is the same as default, except that it offers and provides high-availability (just HA balancing) and failover (clustering) capabilities to your application. It does this within WildFly; all you have to do is design and develop an application which will be running on multiple servers. We will talk more about it in the chapters dedicated to HA and clustering.
`full`	This is the same as default, except that it offers and provides messaging capabilities. In fact, it supports the JMS 2.0 Java EE 7 specification providing the HornetQ specification.
`full-ha`	This is the sum of all the previous profiles. It has HA and clustering capabilities plus messaging.

In the domain mode, we have exactly the same profiles, but all within the `domain.xml` file.

Open the `domain.xml` file into your preferred XML editor and you will notice that there are seven default main XML tags:

- `<extensions/>`
- `<system-properties/>`
- `<management/>`
- `<profiles/>`
- `<interfaces/>`
- `<socket-binding-groups/>`
- `<server-groups/>`

These tags can be viewed in the following screenshot:

```
<?xml version="1.0" ?>

<domain xmlns="urn:jboss:domain:3.0">

    <extensions> ...
    </extensions>

    <system-properties> ...
    </system-properties>

    <management> ...
    </management>

    <profiles>
        <profile name="default"> ...
        </profile>
        <profile name="ha"> ...
        </profile>
        <profile name="full"> ...
        </profile>
        <profile name="full-ha"> ...
        </profile>
    </profiles>

    <interfaces> ...
    </interfaces>

    <socket-binding-groups>
        <socket-binding-group name="standard-sockets" default-interface="public"> ...
        </socket-binding-group>
        <socket-binding-group name="ha-sockets" default-interface="public"> ...
        </socket-binding-group>
        <socket-binding-group name="full-sockets" default-interface="public"> ...
        </socket-binding-group>
        <socket-binding-group name="full-ha-sockets" default-interface="public"> ...
        </socket-binding-group>
    </socket-binding-groups>

    <server-groups>
        <server-group name="main-server-group" profile="full"> ...
        </server-group>
        <server-group name="other-server-group" profile="full-ha"> ...
        </server-group>
    </server-groups>

</domain>
```

Domain.xml structure

Let's look at each tag in detail.

<extensions/>

With regard to the structure of `standalone.xml`, the meaning of the tag and its single extension is the same. Each extension corresponds to a module.

A module is a logical unit composed of one or more Java archives and/or resources (that is, properties file). Each module can declare a dependency with another module. All WildFly modules are located in its installation directory at `modules/system/layers/base/`.

<system-properties/>

We've already talked about properties in *Chapter 2*, *Running WildFly in Standalone Mode*. However, in the domain mode, properties are treated a little differently. System properties defined in `domain.xml` are considered global, so they are inherited by each server group that is spread to each host. Furthermore, these properties can be overridden at SG or host level. The last one wins over all.

We will have a deeper insight into properties in a dedicated recipe later on in this chapter.

<management/>

This section in the `domain.xml` file is about defining a fine-grained access control for the WildFly Web Console. This feature is called **Role-based access control** (**RBAC**).

In short, you may create users and groups to map to the WildFly roles. Doing so, you will be able to create management users that can only deploy or view certain information about your server. Furthermore, you can integrate the Role-based access control feature with an LDAP or Active Directory systems to filter and map users. Security realms, audit log, and management interface section are defined in `host.xml`.

<profiles/>

This is a collection of profiles. In the standalone mode, you just have one instance running; thus, only one profile can be chosen. In the domain mode, you can have different instances running, sharing the same domain controller but providing different features, and thus, profiles.

```
<profiles>
    <profile name="default">...
    </profile>
    <profile name="ha">...
    </profile>
    <profile name="full">...
    </profile>
    <profile name="full-ha">...
    </profile>
</profiles>
```

Profiles in domain.xml

<interfaces/>

Interfaces refers to your server network interfaces. Typically, you will have to set the management interface used to interact with the Admin Console and the CLI, and the public interface where you want your WildFly instance to be available at. Following is the default configuration:

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

Default interfaces

You can set any interface by giving the IP, or by giving wildcards or physical names like this:

- `<any-address/>`
- `<any-ipv4-address/>`
- `<any-ipv6-address/>`
- `<nic name="eth0"/>`

<socket-binding-groups/>

This is a collection of socket-binding-group, each one dedicated to a profile. Same principle of the standalone mode holds true here: you configure the IPs and port numbers of the various services.

<server-groups/>

This is where you define your server groups. You can define one or more server group, each one with its own configuration, as follows:

```
<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>
```

Server groups in domain.xml

As you can see, to define a server group, you need to specify its name and the services it provides, that is, the profile. Also, you have to reference the socket-binding. Additionally, you can define the JVM settings.

Understanding the structure of host.xml

The `host.xml` file is where you design your server groups, defined in the `domain.xml` file. Other than that, there are other fine-grained configurations relative to management services, such as management (which includes security-realms, audit-log, and management-interfaces), interfaces, and the type and location of the domain controller.

Open the `host.xml` file in your preferred XML editor and you will notice that there are five default main XML tags:

- `<management/>`
- `<domain-controller/>`
- `<interfaces/>`
- `<jvms/>`
- `<servers/>`

You can see them in the following screenshot:

```
<?xml version='1.0' encoding='UTF-8'?>

<host name="master" xmlns="urn:jboss:domain:3.0">
    <extensions>
        <extension module="org.jboss.as.jmx"/>
    </extensions>

    <management>
        <security-realms> ...
        </security-realms>
        <audit-log> ...
        </audit-log>
        <management-interfaces> ...
        </management-interfaces>
    </management>

    <domain-controller>
        <local/>
        <remote protocol="remote" host="${jboss.domain.master.address}"
            port="${jboss.domain.master.port:9999}" security-realm="ManagementRealm"/>
    </domain-controller>

    <interfaces>
        <interface name="management">
            <inet-address value="${jboss.bind.address.management:127.0.0.1}"/>
        </interface>
        <interface name="public">
            <inet-address value="${jboss.bind.address:127.0.0.1}"/>
        </interface>
        <interface name="unsecure">
            <inet-address value="${jboss.bind.address.unsecure:127.0.0.1}"/>
        </interface>
    </interfaces>

    <jvms>
        <jvm name="default"> ...
        </jvm>
    </jvms>

    <servers>
        <server name="server-one" group="main-server-group">
        </server>
        <server name="server-two" group="main-server-group" auto-start="true">
            <socket-bindings port-offset="150"/>
        </server>
        <server name="server-three" group="other-server-group" auto-start="false">
            <socket-bindings port-offset="250"/>
        </server>
    </servers>

    <profile>
        <subsystem xmlns="urn:jboss:domain:jmx:1.3">
            <expose-resolved-model/>
            <expose-expression-model/>
            <remoting-connector/>
        </subsystem>
    </profile>
</host>
```

The host.xml file

Let's see each tag in detail.

<management/>

In the XML tag named `management` you define the security realms, audit log, management interfaces, and access control for your WildFly instance.

```
<management>
    <security-realms>
        <security-realm name="ManagementRealm">
            <authentication>
                <local default-user="$local" skip-group-loading="true"/>
                <properties path="mgmt-users.properties" relative-to="jboss.domain.config.dir"/>
            </authentication>
            <authorization map-groups-to-roles="false">
                <properties path="mgmt-groups.properties" relative-to="jboss.domain.config.dir"/>
            </authorization>
        </security-realm>
        <security-realm name="ApplicationRealm">
            <authentication>
                <local default-user="$local" allowed-users="*" skip-group-loading="true"/>
                <properties path="application-users.properties" relative-to="jboss.domain.config.dir"/>
            </authentication>
            <authorization>
                <properties path="application-roles.properties" relative-to="jboss.domain.config.dir"/>
            </authorization>
        </security-realm>
    </security-realms>
    <audit-log>
        <formatters>
            <json-formatter name="json-formatter"/>
        </formatters>
        <handlers>
            <file-handler name="host-file" formatter="json-formatter" relative-to="jboss.domain.data.dir" path="audit-log.log"/>
            <file-handler name="server-file" formatter="json-formatter" relative-to="jboss.server.data.dir" path="audit-log.log"/>
        </handlers>
        <logger log-boot="true" log-read-only="false" enabled="false">
            <handlers>
                <handler name="host-file"/>
            </handlers>
        </logger>
        <server-logger log-boot="true" log-read-only="false" enabled="false">
            <handlers>
                <handler name="server-file"/>
            </handlers>
        </server-logger>
    </audit-log>
    <management-interfaces>
        <native-interface security-realm="ManagementRealm">
            <socket interface="management" port="${jboss.management.native.port:9999}"/>
        </native-interface>
        <http-interface security-realm="ManagementRealm" http-upgrade-enabled="true">
            <socket interface="management" port="${jboss.management.http.port:9990}"/>
        </http-interface>
    </management-interfaces>
</management>
```

Management configuration

Security realms are about securing WildFly. In `ManagementRealm` you can define all those users that are able to manage the instance itself. Every realm should provide an authentication and authorization process; in this case, `ManagementRealm` makes use of two properties files:

- `mgmt-users.properties`: To bind the user and password
- `mgmt-users.properties`: To bind the user's roles

These files are located in the `jboss.domain.config.dir` directory, which means that they are in the configuration directory of the WildFly server, in the domain folder. You will find those files at the following location: `$WILDFLY_HOME/domain/configuration`.

The `ApplicationRealm` should be used only during the development phase to provide an easy mechanism for the protection of your applications. To better secure your application, you should define the security domain with a proper login-module inside the security subsystem of your profile in the `domain.xml`. We will talk about it later in the book.

Audit log, as the name suggests, is about auditing. Every operation that alters the WildFly configuration can be traced. By default, the audit is not enabled. To enable this feature, just change the logger's attribute name, enable, to the value of true.

A management interface, as the name suggests, is the place where you can configure your management interfaces, which are HTTP and native, used by the Web Console and the CLI respectively. By default, the HTTP interface is protected by `ManagementRealm`.

<domain-controller/>

This tag defines how to connect to `domain-controller`. At boot time, each `host-controller` tells the domain-controller who and where it is.

Within this tag, you can define a local `domain-controller`, so you have the DC and the `host-controller` running in the same machine. You can also define a remote DC by specifying the IP and the port number.

```
<domain-controller>
    <local/>
    <remote protocol="remote" host="${jboss.domain.master.address}"
        port="${jboss.domain.master.port:9999}" security-realm="ManagementRealm"/>
</domain-controller>
```

Domain controller definition in host.xml

The preceding image shows both `domain-controller` locators: local and remote. You will have to choose just one.

<interfaces/>

The `interfaces` tag refers to your server network interfaces. Typically, you will have to set the management interface used to interact with the Web Console and the CLI, and the public interface where you want your WildFly instance to be available at. The following screenshot depicts the default configuration:

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

Default interfaces

You can set any interface giving the IP address, or by setting wildcards or physical names such as this:

- `<any-address/>`
- `<any-ipv4-address/>`
- `<any-ipv6-address/>`
- `<nic name="eth0"/>`: Which is only available in Linux systems

<jvms/>

Here you can define a set of JVMs, each one tuned for its specific server group and/or host.

```
<jvms>
   <jvm name="default">
      <heap size="64m" max-size="256m"/>
      <permgen size="256m" max-size="256m"/>
        <jvm-options>
            <option value="-server"/>
        </jvm-options>
   </jvm>
</jvms>
```

JVM declaration in host.xml

<servers/>

This is a set of servers and hosts referencing the corresponding server group, declared in the `domain.xml` file.

```
<servers>
    <server name="server-one" group="main-server-group">
        <jvm name="default">
            <heap size="64m" max-size="256m"/>
        </jvm>
    </server>
    <server name="server-two" group="main-server-group" auto-start="true">
        <socket-bindings port-offset="150"/>
    </server>
    <server name="server-three" group="other-server-group" auto-start="false">
        <socket-bindings port-offset="250"/>
    </server>
</servers>
```

Servers declared in host.xml

All servers declared within this tag will run on the same machine, so you will have to pay attention to bind each service to its unique IP/Port pairs.

Understanding the structure of the standalone.xml

As mentioned in the first chapter, the standalone `configuration` folder comes with four pre-configured files, each representing its relative profile. Let's recap them all:

- `standalone.xml`: This is the `default` profile, which provides all technologies to run the Java EE 7 Web Profile Certified application. With this profile you can use logging, DataSource, JNDI lookups, Servlet & JSP, mail services, WebServices, RESTfull WebServices , JSF, JPA 2, EJB 3.2, CDI 1.1, and Batch.
- `standalone-ha.xml`: This is the same as default, except that it offers and provides high availability (HA—balancing) and failover (clustering) capabilities to your application. It does this within WildFly; all you have to do is design and develop an application which will be running on multiple servers. We will talk more about it in the chapters dedicated to HA and clustering.
- `standalone-full.xml`: This is the same as default, except that it offers and provides messaging capabilities. In fact, it supports the JMS 2.0 Java EE 7 specification providing the HornetQ implementation.
- `standalone-full-ha.xml`: This is the sum of all previous profiles. It has HA and clustering capabilities plus messaging.

Briefly, when you hear about the `ha` profile within WildFly, it means balancing and clustering. When you hear about `full` profile, it means messaging; this could be implemented by HornetQ, ActiveMQ, or any other JMS 2.0 implementation via resource adapters.

How do you choose the right profile?

- First and foremost, it depends on your design and application requirements. The default profile typically fits standard web application development, where high availability and failover characteristics are not required.
- If you are developing an application that provides JMS Queues and/or Topic features, you cannot leave aside the `full` profile.
- If you are running your application on multiple WildFly instances and you want consistency among them, you cannot prescind from the `ha` profile.

If you are running your application on multiple WildFly instances and you want consistency among them, and the applications you provide use JMS technology, then you need the `full-ha` profile.

Open the `standalone.xml` file in your preferred XML editor and you will notice that there are macro configuration XML tags inside the first one, which is `<server/>`; they are shown in the following screenshot:

```
<?xml version="1.0" ?>

<server xmlns="urn:jboss:domain:3.0">
    <extensions> ...
    </extensions>

    <management> ...
    </management>

    <profile> ...
    </profile>

    <interfaces> ...
    </interfaces>

    <socket-binding-group name="standard-sockets" default-interface="public"
        port-offset="${jboss.socket.binding.port-offset:0}"> ...
    </socket-binding-group>
</server>
```

Standalone.xml initial XML tags

Each of the tags is explained in detail in the upcoming subsections.

<extensions/>

Under the tag named `extensions`, all the extensions that provide a Java EE 7 specification for the current profile are present:

```
<extensions>
    <extension module="org.jboss.as.clustering.infinispan"/>
    <extension module="org.jboss.as.clustering.jgroups"/>
    <extension module="org.jboss.as.connector"/>
    <extension module="org.jboss.as.deployment-scanner"/>
    <extension module="org.jboss.as.ee"/>
    <extension module="org.jboss.as.ejb3"/>
    <extension module="org.jboss.as.jaxrs"/>
    <extension module="org.jboss.as.jdr"/>
    <extension module="org.jboss.as.jmx"/>
    <extension module="org.jboss.as.jpa"/>
    <extension module="org.jboss.as.jsf"/>
    <extension module="org.jboss.as.jsr77"/>
    <extension module="org.jboss.as.logging"/>
    <extension module="org.jboss.as.mail"/>
    <extension module="org.jboss.as.messaging"/>
    <extension module="org.jboss.as.modcluster"/>
    <extension module="org.jboss.as.naming"/>
    <extension module="org.jboss.as.pojo"/>
    <extension module="org.jboss.as.remoting"/>
    <extension module="org.jboss.as.sar"/>
    <extension module="org.jboss.as.security"/>
    <extension module="org.jboss.as.transactions"/>
    <extension module="org.jboss.as.webservices"/>
    <extension module="org.jboss.as.weld"/>
    <extension module="org.wildfly.extension.batch"/>
    <extension module="org.wildfly.extension.bean-validation"/>
    <extension module="org.wildfly.extension.io"/>
    <extension module="org.wildfly.extension.request-controller"/>
    <extension module="org.wildfly.extension.security.manager"/>
    <extension module="org.wildfly.extension.undertow"/>
    <extension module="org.wildfly.iiop-openjdk"/>
</extensions>
```

Extensions for default profile

Each extension corresponds to a module.

A module is a logical unit composed of one or more Java archives and/or resources (that is, properties file). Each module can declare a dependency with another module. All WildFly modules are located in its installation directory at `modules/system/layers/base/`.

For example, the module named `org.jboss.as.ejb3` will be located in the folder `modules/system/layers/base/org/jboss/as/ejb3`. The dot notation of the name corresponds to the relative file system path. Let's look at the following example:

```
[wildfly@foogaro WFC]$ cd wildfly
[wildfly@foogaro wildfly]$ cd modules/system/layers/base/
[wildfly@foogaro base]$ ls -la
total 56
drwxr-xr-x 26 wildfly cookbook 4096 Apr 12 09:54 .
drwxr-xr-x  4 wildfly cookbook 4096 Apr 12 09:54 ..
drwxr-xr-x  4 wildfly cookbook 4096 Apr 12 09:54 asm
drwxr-xr-x  4 wildfly cookbook 4096 Apr 12 09:54 ch
drwxr-xr-x 12 wildfly cookbook 4096 Apr 12 09:54 com
drwxr-xr-x  4 wildfly cookbook 4096 Apr 12 09:54 gnu
drwxr-xr-x  4 wildfly cookbook 4096 Apr 12 09:54 ibm
drwxr-xr-x  6 wildfly cookbook 4096 Apr 12 09:54 io
drwxr-xr-x  4 wildfly cookbook 4096 Apr 12 09:54 javaee
drwxr-xr-x 58 wildfly cookbook 4096 Apr 12 09:54 javax
drwxr-xr-x  4 wildfly cookbook 4096 Apr 12 09:54 net
drwxr-xr-x  4 wildfly cookbook 4096 Apr 12 09:54 nu
drwxr-xr-x 56 wildfly cookbook 4096 Apr 12 09:54 org
drwxr-xr-x  6 wildfly cookbook 4096 Apr 12 09:54 sun
[wildfly@foogaro base]$ cd org/jboss/as/ejb3/
[wildfly@foogaro ejb3]$ ls -la
total 12
drwxr-xr-x   4 wildfly cookbook 4096 Apr 12 09:54 .
drwxr-xr-x 118 wildfly cookbook 4096 Apr 12 09:54 ..
drwxr-xr-x   4 wildfly cookbook 4096 Apr 12 09:54 main
[wildfly@foogaro ejb3]$ cd main
[wildfly@foogaro main]$ ls -la
total 1584
drwxr-xr-x 4 wildfly cookbook    4096 Apr 12 09:54 .
drwxr-xr-x 4 wildfly cookbook    4096 Apr 12 09:54 ..
-rw-r--r-- 1 wildfly cookbook    4880 Mar 27 13:57 module.xml
drwxr-xr-x 2 wildfly cookbook    4096 Apr 12 09:54 timers
-rw-r--r-- 1 wildfly cookbook 1598862 Mar 27 13:57 wildfly-ejb3-9.0.0.Beta2.jar
[wildfly@foogaro main]$
```

Module's folder

Inside the `ejb` folder, you may have more then one directory, each one representing a particular version. The default version is named `main`; all the others should represent the indicating version number. In this case, on entering the `main` folder version, we will see the module itself composed of its configuration file `module.xml` and all its JAR and resources (yes, even folders).

Let's take a look at the configuration file:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
  ~ JBoss, Home of Professional Open Source.
  ~ Copyright (c) 2011, Red Hat, Inc., and individual contributors
  ~ as indicated by the @author tags. See the copyright.txt file in the
  ~ distribution for a full listing of individual contributors.
  ~
  ~ This is free software; you can redistribute it and/or modify it
  ~ under the terms of the GNU Lesser General Public License as
  ~ published by the Free Software Foundation; either version 2.1 of
  ~ the License, or (at your option) any later version.
  ~
  ~ This software is distributed in the hope that it will be useful,
  ~ but WITHOUT ANY WARRANTY; without even the implied warranty of
  ~ MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
  ~ Lesser General Public License for more details.
  ~
  ~ You should have received a copy of the GNU Lesser General Public
  ~ License along with this software; if not, write to the Free
  ~ Software Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA
  ~ 02110-1301 USA, or see the FSF site: http://www.fsf.org.
  -->
<module xmlns="urn:jboss:module:1.3" name="org.jboss.as.ejb3">
    <properties>
        <property name="jboss.api" value="private"/>
    </properties>

    <resources>
        <resource-root path="wildfly-ejb3-9.0.0.Beta2.jar"/>
        <resource-root path="timers" />
    </resources>

    <dependencies> ...
    </dependencies>
</module>
```

Example of module.xml

As you can see, in the configuration file you will specify its name, possible properties, its resources, and dependencies. Within the chapter dedicated to the development point of view, we will see how to specify module dependency in your application.

Let's go back to our `standalone.xml` file and follow the other elements that it is composed of.

<management/>

In the XML tag named `management`, you define security realms, audit log, management interfaces, and the access control for your WildFly instance.

```
<management>
    <security-realms>
        <security-realm name="ManagementRealm">
            <authentication>
                <local default-user="$local" skip-group-loading="true"/>
                <properties path="mgmt-users.properties" relative-to="jboss.server.config.dir"/>
            </authentication>
            <authorization map-groups-to-roles="false">
                <properties path="mgmt-groups.properties" relative-to="jboss.server.config.dir"/>
            </authorization>
        </security-realm>
        <security-realm name="ApplicationRealm">
            <authentication>
                <local default-user="$local" allowed-users="*" skip-group-loading="true"/>
                <properties path="application-users.properties" relative-to="jboss.server.config.dir"/>
            </authentication>
            <authorization>
                <properties path="application-roles.properties" relative-to="jboss.server.config.dir"/>
            </authorization>
        </security-realm>
    </security-realms>
    <audit-log>
        <formatters>
            <json-formatter name="json-formatter"/>
        </formatters>
        <handlers>
            <file-handler name="file" formatter="json-formatter" relative-to="jboss.server.data.dir" path="audit-log.log"/>
        </handlers>
        <logger log-boot="true" log-read-only="false" enabled="false">
            <handlers>
                <handler name="file"/>
            </handlers>
        </logger>
    </audit-log>
    <management-interfaces>
        <http-interface security-realm="ManagementRealm" http-upgrade-enabled="true">
            <socket-binding http="management-http"/>
        </http-interface>
    </management-interfaces>
    <access-control provider="simple">
        <role-mapping>
            <role name="SuperUser">
                <include>
                    <user name="$local"/>
                </include>
            </role>
        </role-mapping>
    </access-control>
</management>
```

Management for default profile

Security realms are about securing WildFly. In `ManagementRealm` you can define all users who are able to manage the instance itself. We have already seen this while adding the management user in our first chapter. Every realm should provide an authentication and authorization process; in this case `ManagementRealm` makes use of two properties files:

- One to bind the user and the password (`mgmt-users.properties`)
- One to bind the user's roles (`mgmt-users.properties`)

These files are located in `jboss.server.config.dir`. This means that they are in the configuration directory of the WildFly server which, in this case, is a standalone server. If you have followed all the installation steps as mentioned in the first chapter, you will find those files at the following location: `$WILDFLY_HOME/standalone/configuration`.

The `ApplicationRealm` realm should be used only while developing the applications to provide an easy mechanism for their protection. To better secure your application, you should define `security-domain` with a proper `login-module` inside the security subsystem of your `standalone.xml`. We will talk about it later in the book.

Audit log, as the name suggests, is about auditing. Every operation that alters the WildFly configuration can be traced. By default, the audit is not enabled. To enable this feature, just change the logger's attribute name `enable` to the value of `true`. We will have a recipe about it later in the book.

A management interface is the place where you can configure your management interfaces, which are HTTP and native, used by the Web Console and the CLI respectively. By default, the HTTP interface is protected by `ManagementRealm`.

Access control is a new feature that comes with WildFly 8. Basically, you may create users and groups to map to the WildFly roles. In doing so, you will be able to create management users that can only deploy or view certain information about your server. Furthermore, you can integrate the **Role-based access control** (**RBAC**) feature with an LDAP or Active Directory systems to filter and map the users.

<profile/>

This is actually where your entire standalone configuration resides. Inside it, you will find lots of subsystems, each corresponding to its relative extension. The following screenshot depicts all the subsystems that belong to the default profile configuration:

```
<profile>
    <subsystem xmlns="urn:jboss:domain:logging:3.0"> ...
    </subsystem>
    <subsystem xmlns="urn:jboss:domain:batch:1.0"> ...
    </subsystem>
    <subsystem xmlns="urn:jboss:domain:bean-validation:1.0"/>
    <subsystem xmlns="urn:jboss:domain:datasources:3.0"> ...
    </subsystem>
    <subsystem xmlns="urn:jboss:domain:deployment-scanner:2.0">
    </subsystem>
    <subsystem xmlns="urn:jboss:domain:ee:3.0"> ...
    </subsystem>
    <subsystem xmlns="urn:jboss:domain:ejb3:3.0"> ...
    </subsystem>
    <subsystem xmlns="urn:jboss:domain:io:1.1"> ...
    </subsystem>
    <subsystem xmlns="urn:jboss:domain:infinispan:3.0"> ...
    </subsystem>
    <subsystem xmlns="urn:jboss:domain:jaxrs:1.0"/>
    <subsystem xmlns="urn:jboss:domain:jca:3.0"> ...
    </subsystem>
    <subsystem xmlns="urn:jboss:domain:jdr:1.0"/>
    <subsystem xmlns="urn:jboss:domain:jmx:1.3"> ...
    </subsystem>
    <subsystem xmlns="urn:jboss:domain:jpa:1.1">
    </subsystem>
    <subsystem xmlns="urn:jboss:domain:jsf:1.0"/>
    <subsystem xmlns="urn:jboss:domain:mail:2.0"> ...
    </subsystem>
    <subsystem xmlns="urn:jboss:domain:naming:2.0">
    </subsystem>
    <subsystem xmlns="urn:jboss:domain:pojo:1.0"/>
    <subsystem xmlns="urn:jboss:domain:remoting:3.0"> ...
    </subsystem>
    <subsystem xmlns="urn:jboss:domain:resource-adapters:3.0"/>
    <subsystem xmlns="urn:jboss:domain:request-controller:1.0"/>
    <subsystem xmlns="urn:jboss:domain:sar:1.0"/>
    <subsystem xmlns="urn:jboss:domain:security-manager:1.0"> ...
    </subsystem>
    <subsystem xmlns="urn:jboss:domain:security:1.2"> ...
    </subsystem>
    <subsystem xmlns="urn:jboss:domain:transactions:3.0"> ...
    </subsystem>
    <subsystem xmlns="urn:jboss:domain:undertow:2.0"> ...
    </subsystem>
    <subsystem xmlns="urn:jboss:domain:webservices:2.0"> ...
    </subsystem>
    <subsystem xmlns="urn:jboss:domain:weld:2.0"/>
</profile>
```

Default profile subsystems

<interfaces/>

The `interfaces` tag refers to your server network interfaces. Typically, you will have to set the management interface used to interact with the Web Console and the CLI, and the public interface where you want your WildFly instance to be available. The following screenshot depicts the default configuration:

```
<interfaces>
    <interface name="management">
        <inet-address value="${jboss.bind.address.management:127.0.0.1}"/>
    </interface>
    <interface name="public">
        <inet-address value="${jboss.bind.address:127.0.0.1}"/>
    </interface>
    <!-- TODO - only show this if the jacorb subsystem is added  -->
    <interface name="unsecure">
        <!--
          ~  Used for IIOP sockets in the standard configuration.
          ~                    To secure JacORB you need to setup SSL
          -->
        <inet-address value="${jboss.bind.address.unsecure:127.0.0.1}"/>
    </interface>
</interfaces>
```

Default interfaces

You can set any interface giving the IP, or by setting wildcards or physical names such as this:

- `<any-address/>`
- `<any-ipv4-address/>`
- `<any-ipv6-address/>`
- `<nic name="eth0"/>`: This is only available on Linux systems

<socket-binding-group/>

This is the place where you define most of the ports used by the various interfaces and connectors. The following screenshot denotes the default configuration:

```
<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-group>
```

Default socket-binding-group configuration