
Servidores web y PaaS

José Luis Zamora Sánchez <<joseluiszamora.jlz@gmail.com>>

Table of Contents

1. Introducción a los servidores de aplicaciones y a WildFly	5
1.1. ¿Qué es un servidor de aplicaciones?	5
1.2. Evolución de Java Enterprise	7
1.3. Introducción a WildFly	8
Instalación de WildFly	9
Definición de dominio	9
Modos de trabajo de WildFly	9
Estructura de directorios de WildFly	11
Creación de un usuario administrador	11
1.4. Herramientas de administración de WildFly	12
Consola de administración Web	12
Command Line Interface (CLI)	13
Edición manual de los ficheros de configuración	16
1.5. Aplicaciones Java Enterprise	16
Empaquetado de aplicaciones	16
Despliegue de aplicaciones en WildFly	18
1.6. Arquitecturas basadas en microservicios	20
Ventajas de una arquitectura basada en Microservicios sobre una monolítica ...	20
Desventajas	21
Cuándo utilizar microservicios	21
Java EE Y los microservicios	22
Docker	24
1.7. Referencias	24
1.8. Ejercicios de Introducción a los servidores de aplicaciones y a WildFly	25
Desarrollo de un microservicio (0.3 Puntos)	25
Otras formas de despliegue (0.3 Puntos)	27
Virtualización con Docker (0.6 Puntos)	27
Entrega	31
2. Administración de recursos en WildFly	32
2.1. Fichero de configuración del Bootstrap	32
2.2. Estructura de los ficheros de configuración	32
2.3. Edición desde las herramientas de administración	35
2.4. Configuración de recursos	38
Configuración de Datasources	38
Creación del Datasource y los pool	40
Configuración del contenedor de EJB's	43
Descriptores de despliegue	45
Class Loading	45
Rendimiento	46
2.5. Referencias	47
2.6. Ejercicios de Administración de recursos en Wildfly	47
Librería compartida de Logs (0.6 puntos)	47
Camino a Eurovisión (1 punto)	48
3. Seguridad declarativa en WildFly (JAAS)	51
3.1. Implementación de la seguridad declarativa en WildFly	51

Security Domain y Login Modules	52
Activación de la auditoria de seguridad	54
3.2. Administración de usuarios y roless con RealmDirect	55
Realm Personalizado	55
Configuración de un login module basado en BD	56
Protección de las password	57
Integración con un servidor LDAP Externo	59
Login module LdapExtended	63
3.3. Role Based Access Control en interfaces de administración	64
3.4. Referencias	65
3.5. Ejercicios de seguridad declarativa en WildFly	65
Configuración, despliegue y alta de usuarios en WildFly (0.6 puntos)	66
Gestión de usuarios en Base de datos (0.3 puntos)	67
Seguridad integrada con LDAP corporativo (0.3 puntos)	67
Entrega	67
4. Certificados digitales y SSL	68
4.1. Seguridad en la capa de transporte	68
Certificados digitales	68
Introducción a SSL	68
One-Way y Two-Way SSL	69
Creación de una CA ficticia	69
4.2. Referencias	79
4.3. Ejercicios de certificados digitales y SSL	79
Encriptación de las comunicaciones One-Way SSL (0.4 puntos)	80
Encriptación de las comunicaciones con Two-Way SSL (0.4 puntos)	80
Autenticación de usuario a partir del certificado (0.4 puntos)	80
Entrega	81
5. Introducción a Cloud Computing y a OpenShift	82
5.1. ¿Qué es Cloud Computing?	82
Computación Pre-Cloud	82
5.2. ¿Qué nos aporta Cloud Computing?	82
5.3. Nuevos conceptos de Cloud Computing	83
5.4. Riesgos de Cloud	83
Qué opinan las empresas acerca de Cloud	84
5.5. Tipos de Cloud en función del servicio	85
5.6. Tipos de Cloud en función del tipo de acceso	88
5.7. Seguridad	89
5.8. Cómo elegir un proveedor de Cloud	89
5.9. Introducción a OpenShift	90
Lenguajes y runtimes soportados	91
Versiones de OpenShift	91
Modalidades de pago de OpenShift Online	92
Alternativas a OpenShift	93
5.10. Referencias	93
5.11. Ejercicios de Introducción a Cloud Computing	94
Despliegue en Cloud de una aplicación web (1 punto)	97
Importar el proyecto desde IntelliJ y modificar la aplicación (0.4 puntos)	98
Despliegue de un servicio de almacenamiento en Cloud (0.4 puntos)	99
Creación de un blog basado en WordPress (0.4 puntos)	100
Entrega	102
6. Plataforma como Servicio	103
6.1. Conceptos propios de OpenShift	103
6.2. Cómo funciona OpenShift	106

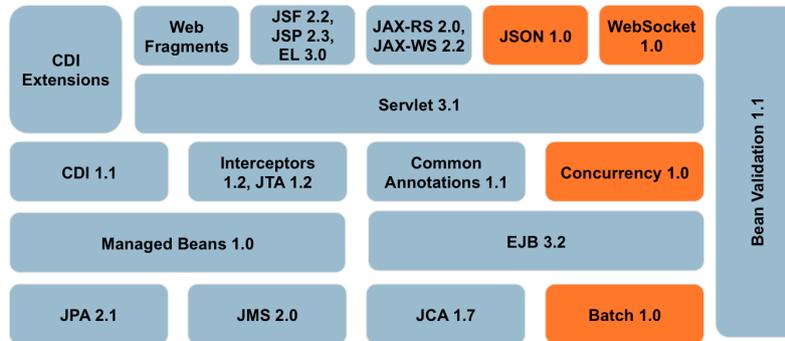
Creación de aplicaciones	108
Añadir Cartridges de base de datos	112
Port forwarding	113
Modificación y despliegue de las aplicaciones	114
Consulta de Logs	117
Borrado de aplicaciones	118
Gestión del dominio	118
6.3. Marketplace	119
6.4. Monitorización	119
Herramientas estándar de monitorización	119
Cartridges de terceros	121
6.5. Referencias	123
6.6. Ejercicios de Plataforma como servicio	123
Despliegue desde un repositorio externo (0.2 puntos)	123
Desarrollo mixto local/cloud (0.8 puntos)	124
Monitorización de una aplicación (0.2 puntos)	124
Entrega	124
7. Características IaaS+ de OpenShift	125
7.1. Introducción	125
7.2. Configuración avanzada	125
SSH	125
Variables de entorno	128
Uso de Markers	130
Action Hook Scripts	131
Transferencia de ficheros	132
Tareas planificadas (cron)	133
7.3. Almacenamiento en disco	134
Si ya nos hemos quedado sin espacio.	135
7.4. Copias de seguridad	135
Deployment History	136
Application Snapshots	137
7.5. Procedimientos	137
Depuración de aplicaciones	137
Generar un threaddump	139
Despliegue manual de las aplicaciones	139
Compilación de proyectos con dependencias propias	140
Despliegue de aplicaciones compiladas previamente.	140
7.6. Referencias	141
7.7. Ejercicios de Características IaaS+ de OpenShift	141
Instalación de Forge	141
Autoescuela	142
Despliegue básico de la aplicación (0.2 punto)	143
Configuración de la aplicación OpenShift (0.6 puntos)	144
Uso de una base de datos externa en Cloud (0.4 puntos)	144
Entrega	145
8. Integración continua y escalabilidad	146
8.1. Escalabilidad	146
Autoescalado en OpenShift	146
Definir una aplicación como escalable	148
Ajustar la configuración de escalado	149
Monitorización del balanceador	151
Configuración de HAProxy	151
8.2. DevOps	152

Introducción	152
DevOps y Cloud Computing	154
Integración continua (CI) con Jenkins	154
Como trabajar contra un repositorio de Bitbucket	159
Alternativas a Jenkins	163
Deployment pipeline	164
8.3. Referencias	165
8.4. Ejercicios de Integración continua y escalabilidad	166
Despliegue de una aplicación escalable (0.2 Puntos)	167
Prueba de Estrés (0.4 puntos)	167
Integración continua (0.2 puntos)	168
Pruebas con Arquillian (0.4 puntos)	169
Entrega	173

1. Introducción a los servidores de aplicaciones y a WildFly

1.1. ¿Qué es un servidor de aplicaciones?

Un servidor de aplicaciones es una plataforma que implementa los servicios definidos en la especificación de Java Enterprise.



Estos servicios van desde la ejecución de aplicaciones web sencillas, basadas en Servlets/JSP hasta aplicaciones distribuidas con soporte transaccional, basadas en Enterprise Java Beans (EJB) o servicios Web.

Especificación	Java EE 6	Java EE 7
Servlet	3.0	3.1
JavaServer Pages (JSP)	2.2	2.3
Unified Expression Language (EL)	2.2	3.0
Debugging Support for Other Languages (JSR-45)	1.0	1.0
JavaServer Pages Standard Tag Library (JSTL)	1.2	1.2
JavaServer Faces (JSF)	2.0	2.2
Java API for RESTful Web Services (JAX-RS)	n/a	2.0
Java API for WebSocket (WebSocket)	n/a	1.0
Java API for JSON Processing (JSON-P)	n/a	1.0
Common Annotations for the Java Platform (JSR-250)	1.1	1.2
Enterprise JavaBeans (EJB)	3.1 Lite	3.2 Lite
Java Transaction API (JTA)	1.1	1.2
Java Persistence API (JPA)	2.0	2.1
Bean Validation	1.0	1.1
Managed Beans	1.0	1.0
Interceptors	1.1	1.2
Contexts and Dependency Injection (CDI)	1.0	1.1



¿Se puede considerar Tomcat un servidor de aplicaciones?

Tomcat implementa un contenedor Web que en su versión 8 cumple las especificaciones Servlet 3.1, JSP 2.3, y EL 3.0 y WebSockets, pero no incluye un contenedor de EJB's o da soporte a otras APIs como JMS, por tanto no se puede considerar como un servidor de aplicaciones completo.

Sí que es cierto es que de la combinación de Tomcat con Spring, junto con otras librerías de terceros resulta en algo parecido a un servidor de aplicaciones. Es más, proyectos como Spring Boot o DropWizard permiten integrar dentro de un único fichero, tanto código de aplicación como los servicios que necesita para funcionar.

La principal ventaja de un servidor de aplicaciones es que integra todo los componentes necesarios para empezar a trabajar, permitiendo al programador centrarse en lo más importante que es el desarrollo de aplicaciones. Esto no quiere decir que no podamos utilizar las librerías que creamos oportunas, pero el producto ya nos ofrece una solución válida en conjunto.

Para un administrador de sistemas también es más sencillo pues al ser un producto estándar, puede trabajar sobre el servidor sin necesidad de conocer los entresijos de las aplicaciones que ejecutan.



¿Debemos entonces debemos utilizar un servidor de aplicaciones para todo?

No necesariamente, puesto que la administración y configuración de un Tomcat, Jetty u otros servidores es más sencilla, siempre y cuando nuestra aplicación no requiera de servicios más avanzados. Por otra parte si nuestro desarrollo es muy específico puede compensar el invertir tiempo y desarrollar una solución a medida con librerías y/o frameworks alternativos.

Lo que nos debe llevar a pensar en utilizar un servidor de aplicaciones es considerar que cuestiones como la **estandarización, la alta disponibilidad, la monitorización del rendimiento, transaccionalidad y seguridad** son importantes en nuestros desarrollos así como el poder centrarnos en resolver un problema abstrayéndonos de los servicios que vamos a utilizar.

La especificación Java EE 7 se publicó en mayo de 2013 pero no es hasta finales de 2015 cuando se puede considerar como instaurada. La propia Oracle certifica su producto comercial WebLogic en Octubre de 2015 y RedHat, a fecha de redacción de estos apuntes aún no ha publicado una versión comercial certificada.



Los más importantes son

- Comerciales: Oracle WebLogic, IBM WebSphere, Red Hat JBoss EAP.
- Open Source: Oracle GlassFish, RedHat WildFly.

¿Merece la pena pagar por un producto comercial? Hace unos años la respuesta a esta pregunta era más sencilla, pero hoy en día con la cantidad de información disponible y la madurez de los productos Open Source actuales es complicado el amortizar los costes de licencias, sobre todo en el caso de nuevos desarrollos basados en Java EE 6 o 7.

1.2. Evolución de Java Enterprise

Desde sus inicios, la especificación Java Enterprise ha intentado definir y estandarizar los servicios que una aplicación de negocio necesita, y a lo largo de sus primeras versiones fue añadiendo nuevas tecnologías, si bien el desarrollo de aplicaciones era bastante complejo.

Los primeros servidores de aplicaciones de algún modo emulaban el modelo de Mainframe ya conocido en aplicaciones de grandes empresas pero especializado en la ejecución de aplicaciones java y suponían un coste elevado en hardware y licencias.



Esta situación llevó a que surgieran iniciativas como Spring que ofrecían la misma idea de estandarización pero sustentada en contenedores web ligeros, gratuita y sustentada en una gran comunidad de desarrolladores.

Proyectos como Spring sirvieron para que, primero Sun y ahora Oracle, entendiesen que el desarrollo debía evolucionar hacia una concepción más sencilla donde el desarrollo fuese lo más cercano posible a la escritura de POJO's y los servidores de aplicaciones mucho más ligeros y optimizados. Y por qué no decirlo, se consiguió aceptando como estándar las mejores ideas de los competidores. Esto se consigue sobre todo a partir de la madurez en el mercado de Java EE 6.

1.3. Introducción a WildFly

WildFly es el nuevo nombre que recibe el servidor de aplicaciones JBoss, desarrollado por Red Hat. Frente a las versiones anteriores (JBoss AS 7.1 y EAP6) presenta las siguientes novedades:

- Compatible y optimizado para la JVM de Java SE 7. Esto se traduce en mejoras de rendimiento en Entrada/Salida, políticas de Garbage collector y concurrencia.
- Certificado para Java EE 7 Web y Full Profile.
- Roles definidos dentro de los usuarios administradores. Se puede delimitar con mayor detalle las tareas que puede realizar un usuario administrador.
- Nuevo Servidor Web. Tradicionalmente se utilizaba Tomcat, pero en esta versión se sustituye por Undertow, que proporciona un mayor rendimiento.
- Mejoras en las herramientas de administración (Consola, línea de comandos).
- Reducción del número de puertos de escucha a solo 2, puerto de trabajo y puerto de administración.

Si bien es interesante académicamente poder trabajar con la última especificación de Java EE, Red Hat no ofrece ninguna variante con soporte comercial, por tanto en el mundo de la

empresa es más recomendable continuar con una versión anterior ya sea comercial u Open Source.

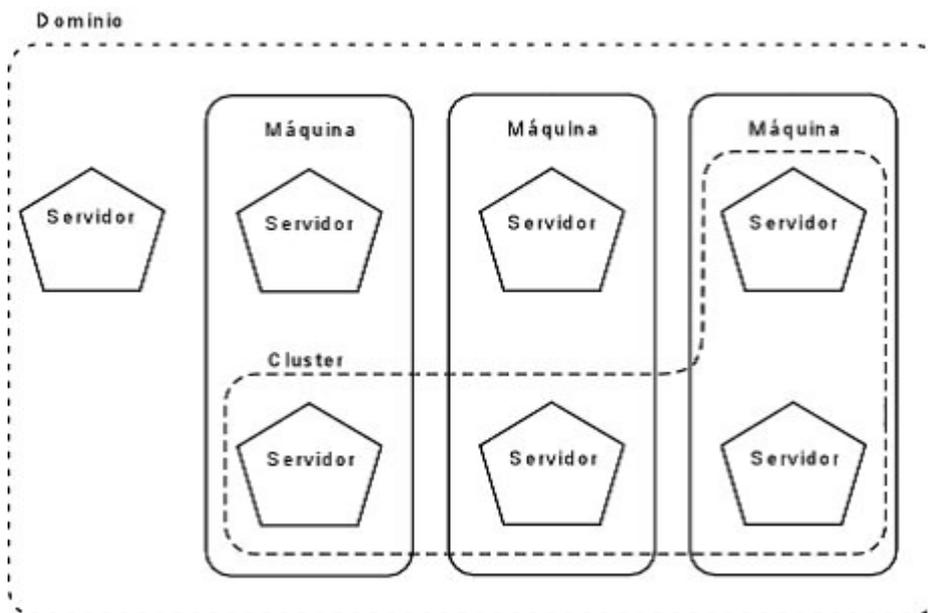
Instalación de WildFly

La instalación del servidor es muy sencilla. Basta con conectarse a <http://www.wildfly.org> y descargarse un fichero ZIP. Es recomendable establecer la variable de entorno JBOSS_HOME apuntando a la carpeta donde se vaya a descomprimir el servidor:

```
export JBOSS_HOME=/usr/share/wildfly-8.2.1
```

Definición de dominio

Un dominio define un conjunto de propiedades, recursos e instancias de servidores de aplicaciones. La definición de dominios permite flexibilizar y organizar la instalación de servidores de aplicaciones, ya que es posible asignar distintos dominios dentro de un mismo host a distintas organizaciones y administradores. Las aplicaciones y recursos instaladas en un dominio son independientes de los otros dominios. Para permitir que distintos dominios puedan estar en marcha al mismo tiempo, cada dominio utiliza distintos puertos de servicio y administración.



Modos de trabajo de WildFly

Standalone

Es el modo más sencillo y habitual. Permite ejecutar WildFly como un proceso independiente. Este modo de trabajo no es incompatible con una configuración HA, por lo que podemos tener configurados varios servidores independientes en clúster. Para iniciar WildFly en modo Standalone hay que ejecutar el script:

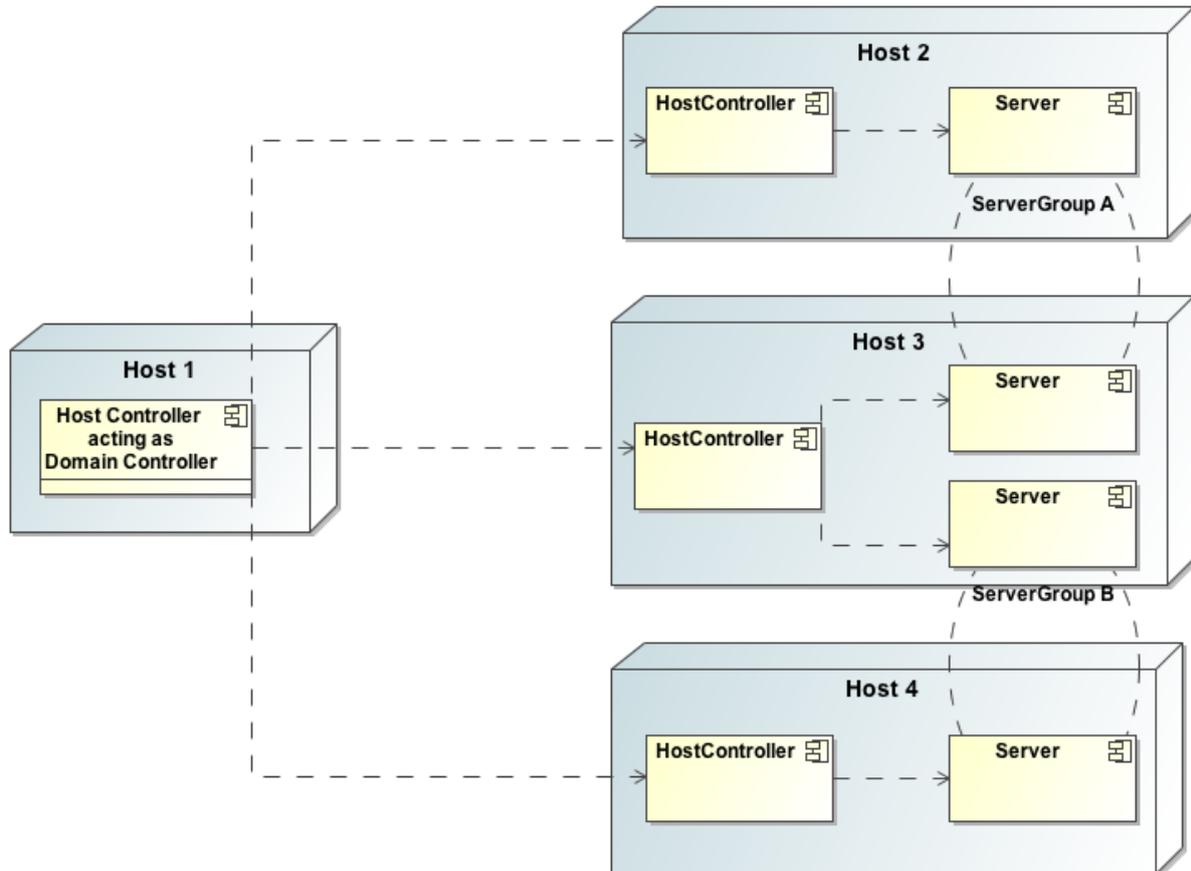
```
standalone.sh
```

Managed Domain

En este modo se permite iniciar uno o varios servidores de WildFly administrados de forma conjunta desde un único dominio. La forma de iniciar un dominio WildFly es a través del script:

```
domain.sh
```

Este script iniciará un proceso denominado Host Controller, que se encargará de iniciar los servidores administrados del dominio. El siguiente diagrama describe un dominio WildFly:



Los conceptos de host y servidor ya los conocemos, pero tenemos nuevos elementos:

Host Controller

Al ejecutar el script *domain.sh*, se inicia un proceso denominado Host Controller cuya única misión es la de iniciar/parar los distintos servidores que se hayan definido en la máquina. Para ello se comunica con el proceso *Domain Controller* que definiremos a continuación.

Domain Controller

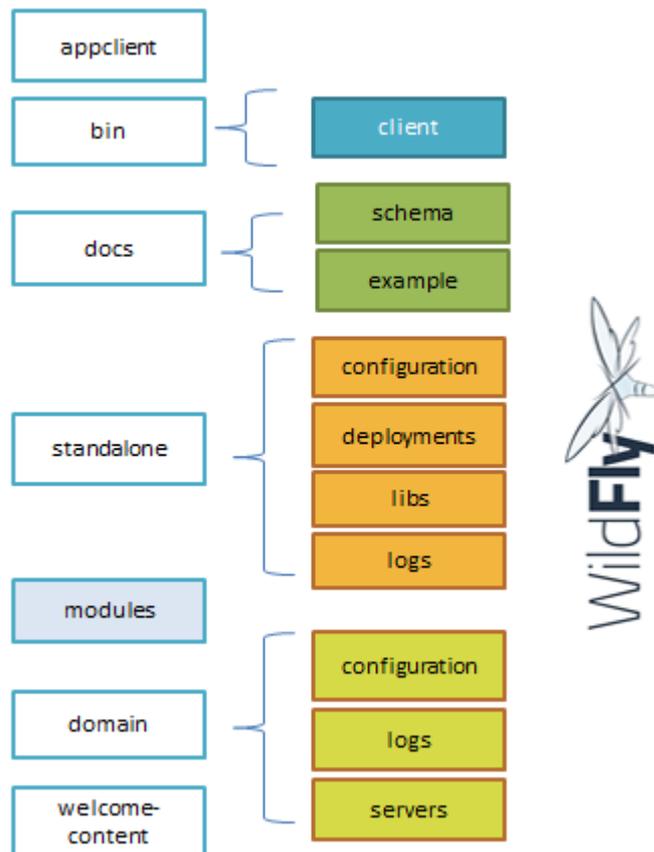
Dentro de un dominio debe existir un proceso Host Controller configurado como el Domain Controller, es decir como almacén central de la configuración del dominio. Este proceso será el encargado de gestionar las políticas de administración, de sincronizar dichas políticas con el resto de Host Controllers del dominio

Server Group

Un server group es un conjunto de instancias que deben ser administradas como si fueran una sola. En una configuración de dominio puede haber uno o varios grupos, y

cada servidor de aplicaciones debe ser miembro de un grupo, aún en el caso de que sólo exista un servidor en el dominio. Es responsabilidad del Domain Controller y de los Host Controllers que todos los servidores dentro de un grupo tengan una configuración coherente y los mismos despliegues.

Estructura de directorios de WildFly



Las carpetas más importantes:

- **appclient**: contiene ficheros de configuración y propios del contenedor de aplicaciones clientes.
- **bin**: contiene varios ficheros de configuración de arranque, jboss-client.jar para acceder crear una aplicación cliente Java.
- **domain**: Contiene los ficheros de configuración del modo *Managed Domain* y datos de ejecución del servidor en este modo.
- **standalone**: Similar la anterior, pero específica del modo *Standalone*.
- **welcome-content**: Contenido estático de la aplicación web por defecto.

Creación de un usuario administrador

Para poder comenzar a trabajar con WildFly es necesario crear un usuario administrador y para ello utilizaremos el script `add-user.sh`.

```

Terminal - expertojavajs@expertojavajs-VirtualBox: /usr/local/wildfly-8.1.0.Final/b - + x
Archivo Editar Ver Terminal Pestañas Ayuda
expertojavajs@expertojavajs-VirtualBox: /usr/local/wildfly-8.1.0.Final/bin$
expertojavajs@expertojavajs-VirtualBox: /usr/local/wildfly-8.1.0.Final/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): a

Enter the details of the new user to add.
Using realm 'ManagementRealm' as discovered from the existing property files.
Username : experto
Password recommendations are listed below. To modify these restrictions edit the
add-user.properties configuration file.
- 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)
- The password should be different from the username
Password :
JBAS015269: Password must have at least 8 characters!
Are you sure you want to use the password entered yes/no? y
Re-enter Password :
What groups do you want this user to belong to? (Please enter a comma separated list, or leave blank for none)[  ]:
About to add user 'experto' for realm 'ManagementRealm'
Is this correct yes/no? yes
Added user 'experto' to file '/usr/local/wildfly-8.1.0.Final/standalone/configuration/mgmt-users.properties'
Added user 'experto' to file '/usr/local/wildfly-8.1.0.Final/domain/configuration/mgmt-users.properties'
Added user 'experto' with groups  to file '/usr/local/wildfly-8.1.0.Final/standalone/configuration/mgmt-groups.properties'
Added user 'experto' with groups  to file '/usr/local/wildfly-8.1.0.Final/domain/configuration/mgmt-groups.properties'
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? █

```

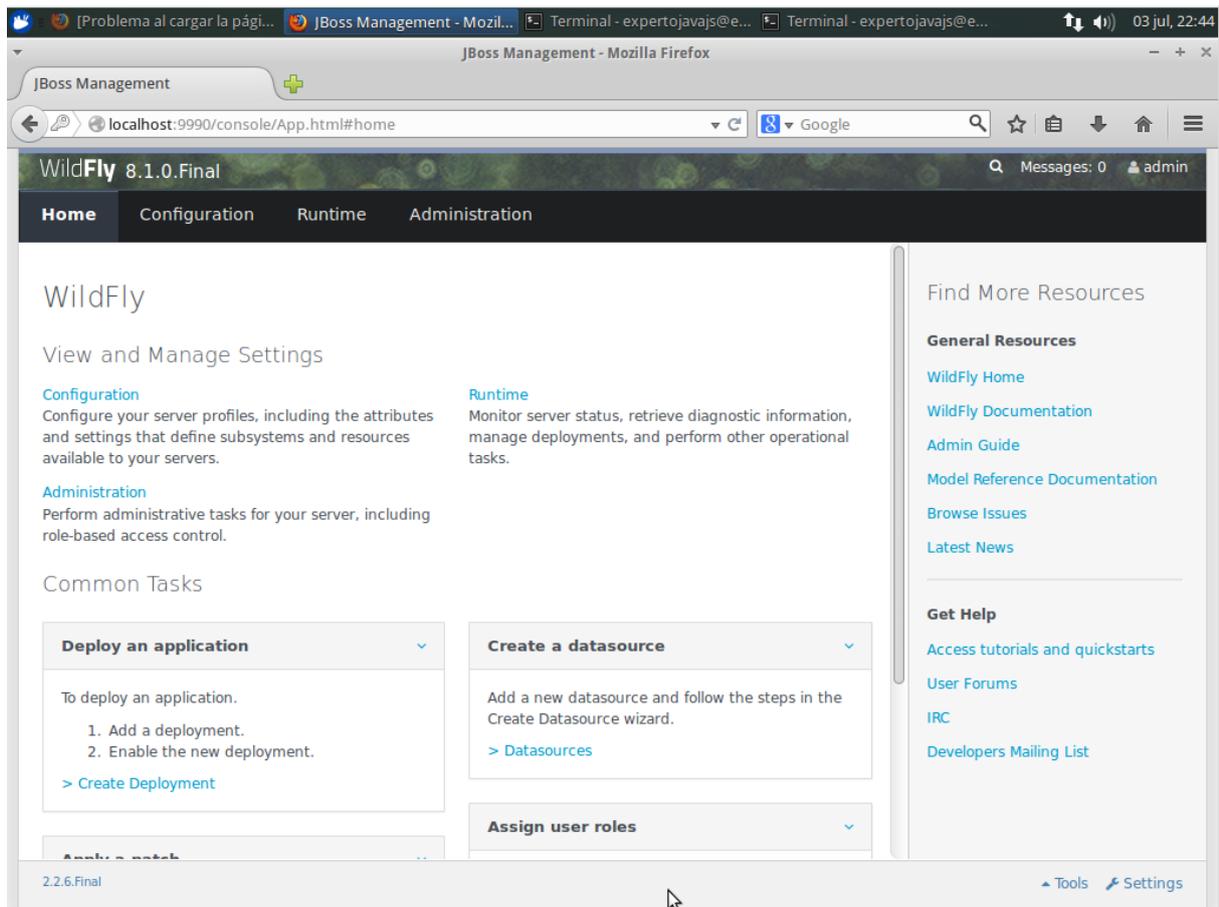
En este ejemplo, hemos creado el usuario *experto* como usuario administrador perteneciente al ManagementRealm. Es importante contestar SI a la última pregunta para indicar que el usuario se podrá conectar al domain controller. Se generará una password encriptada que se utilizará a la hora de configurar un dominio de servidores de aplicaciones.

1.4. Herramientas de administración de WildFly

Consola de administración Web

Mediante la consola de administración podemos acometer las tareas más básicas de una forma amigable. El puerto de administración habitual es el 9990, y para acceder a la consola basta con abrir la dirección:

<http://localhost:9990/App.html>



Comparada con la consola de administración de otros servidores como WebLogic o incluso GlassFish es mas rudimentaria, si bien ha mejorado notablemente con respecto a versiones anteriores.

Command Line Interface (CLI)

Se trata de una herramienta de administración desde una pantalla de terminal. A esta consola se accede ejecutando el script **jboss-cli.sh**

Desde esta consola se puede cambiar la configuración de un servidor local o remoto. Parámetros más importantes:

```

$ ./jboss-cli.sh [--help] [--version] [--connect] [--controller] [--
commands] [--user] [--password] [--file]
    
```

--help

Muestra un listado de comandos admitidos

--version

Muestra información de la versión de WildFly y de la máquina.

--connect

Comando utilizado para conectarnos al servidor e iniciar su administración

--controller

Host:puerto al que nos queremos conectar

--commands

Permite especificar una o varias instrucciones de administración.

--user/password

Credenciales para conectarse a un servidor, utilizadas normalmente al acceder a un servidor remoto.

--file

Permite ejecutar un script con múltiples instrucciones de administración.

Ejemplos de uso:

```
#Conexión a un servidor local (localhost:9990)
./jboss-cli.sh --connect

#Conexión a un host controller remoto
./jboss-cli.sh --connect --controller=192.168.10.1 --user=admin1234 --
password=pass1234
#Ejecutar una secuencia de comandos
./jboss-cli.sh --commands="connect,deploy prueba.jar"
#Ejecutar un script de administración
./jboss-cli.sh --file=myscript.cli
#Parar el servidor
./jboss-cli.sh --connect --command=shutdown
```

Ejemplo de script:

```
# Connect to Wildfly instance
connect

# Create Spring Batch Module

module add \
  --name=org.springframework.batch \
  --dependencies=javax.api,javaee.api \
  --resources=${wildfly.module.classpath}
```

En el caso de que tengamos algún problema con la configuración, que no permita operar normalmente al servidor, podemos recurrir al siguiente comando:

```
./standalone.sh --admin-only
```

Este comando permite arrancar el servidor en modo administración, bloqueado excepto para tareas administrativas. De esta forma podremos acceder a CLI y corregir la configuración. Una vez hecho esto, podemos salir del modo administración con el siguiente comando:

```
[standalone@localhost:9990/]reload --admin-only=false
```

Una vez estemos conectados a una instancia de WildFly podemos trabajar directamente sobre la configuración del servidor como si fuera un sistema de ficheros:

```

Terminal - expertojavajs@expertojavajs-VirtualBox: /usr/local/wildfly-8.1.0.Final/b - + x
Archivo  Editar  Ver  Terminal  Pestañas  Ayuda
[standalone@localhost:9990 /] exit

expertojavajs@expertojavajs-VirtualBox: /usr/local/wildfly-8.1.0.Final/bin$ ./jboss-cli.sh
You are disconnected at the moment. Type 'connect' to connect to the server or 'help' for the list of supported commands.
[disconnected /] connect
[standalone@localhost:9990 /] ls
core-service                management-minor-version=1
deployment                  name=expertojavajs-virtualbox
deployment-overlay          namespaces=[]
extension                   process-type=Server
interface                   product-name=undefined
path                       product-version=undefined
socket-binding-group        profile-name=undefined
subsystem                   release-codename=Kenny
system-property             release-version=8.1.0.Final
launch-type=STANDALONE      running-mode=NORMAL
management-major-version=2  schema-locations=[]
management-micro-version=0  server-state=running
[standalone@localhost:9990 /]

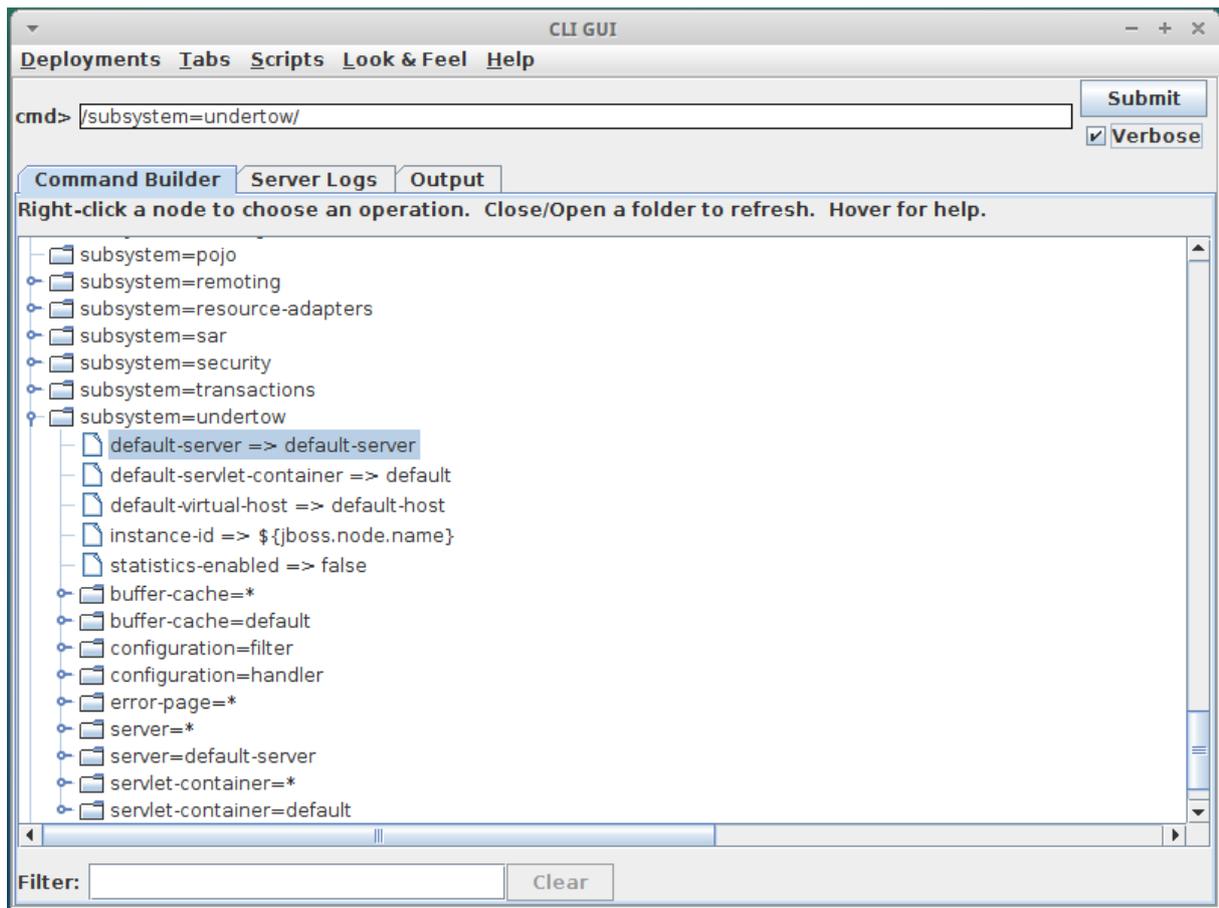
```

Por último, si queremos una interfaz más amigable, podemos ejecutar la consola en modo gráfico:

```

.....
./jboss-cli.sh --gui
.....

```



Edición manual de los ficheros de configuración

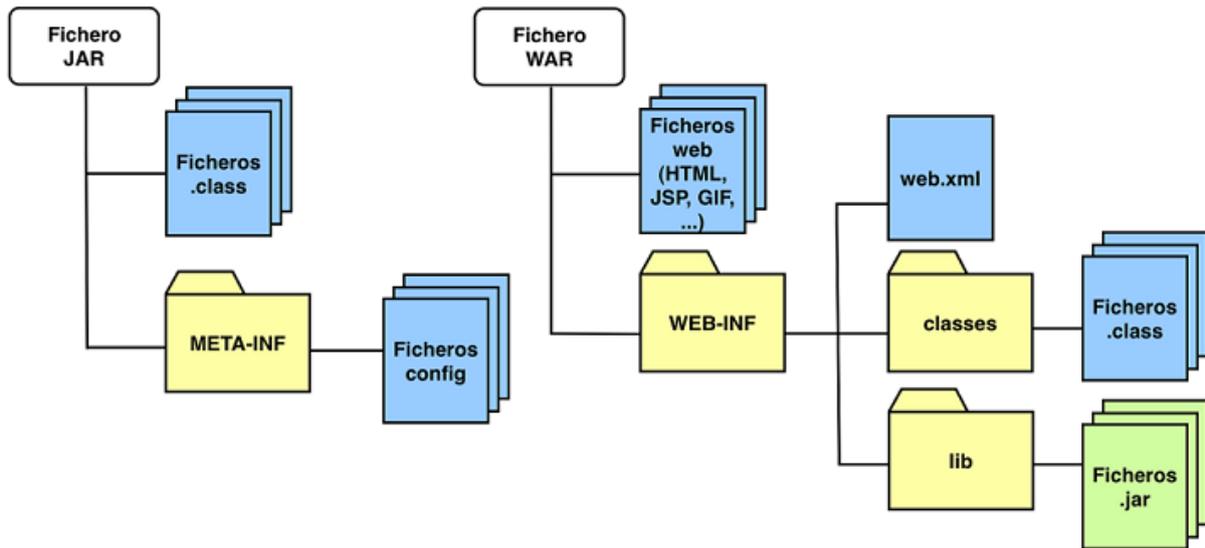
Toda la configuración del servidor persiste en ficheros XML que se pueden editar manualmente. En la siguiente sesión veremos con más detalle la forma en la que WildFly almacena esta configuración pero hay que tener en cuenta que lo recomendable hacer cambios en la configuración mediante las herramientas de administración, sobre todo en servidores de producción.

1.5. Aplicaciones Java Enterprise

Empaquetado de aplicaciones

Hasta ahora hemos utilizado dos tipos de ficheros para empaquetar aplicaciones Java: ficheros JAR y ficheros WAR. Además de estos tipos hay un tercero específico de las aplicaciones Java Enterprise: el EAR.

Los ficheros JAR empaquetan clases Java. Un fichero JAR contiene clases Java compiladas (ficheros .class) junto con un descriptor de la aplicación.



Cuando un fichero JAR se añade al classpath de una JVM (Máquina Virtual Java) las clases incluidas en él se ponen a disposición de cualquier aplicación Java que ejecute la JVM. De la misma forma, cuando un fichero JAR se define en el classpath del compilador Java, las clases incluidas en él pueden utilizarse en las clases que estamos compilando.

Una idea fundamental relacionada con el empaquetado de clases es que el compilador Java y la máquina virtual Java tienen distintos classpath, esto es, resuelven las dependencias utilizando distintos ficheros. Esto trasladado a una aplicación Java Enterprise permite que para compilar una aplicación estándar, únicamente necesitemos una dependencia:

```
<dependencies>
  <dependency>
    <groupId>javax</groupId>
    <artifactId>javaee-api</artifactId>
    <version>7.0</version>
    <scope>provided</scope>
  </dependency>
</dependencies>
```

El servidor de aplicaciones por su parte contendrá tanto la especificación como las librerías (jars) específicas que implementen la especificación.

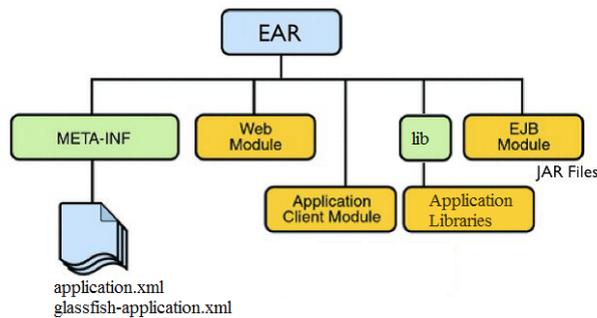
Las aplicaciones web empaquetadas en ficheros WAR son más complejas puesto que pueden contener recursos estáticos, clases compiladas y librerías Jar. Al desplegar una aplicación web en un servidor de aplicaciones, las librerías contenidas pasan a formar parte del classpath de la aplicación exclusivamente.

¿Qué ocurre si queremos utilizar una misma librería en distintas aplicaciones Web desplegadas en un servidor? Si las incluimos en cada aplicación y son pesadas, penalizará el arranque del servidor y elevará el consumo de recursos innecesariamente. Para evitarlo tenemos varias formas:

- Incluir la librería directamente en el classpath del servidor (¡No recomendado!)
- Despliegue del jar como *librería compartida*, en el caso concreto de WildFly sería declararla como módulo global del servidor en ficheros de configuración.

- Si son aplicaciones relacionadas empaquetarlas dentro de un fichero EAR.

Los ficheros EAR representan una aplicación empresarial formada por distintos módulos (aplicaciones web y ficheros JAR). Los ficheros JAR pueden ser librerías o *Enterprise JavaBeans* (EJB) usados por las aplicaciones web y estarán disponibles para cualquier aplicación contenida.



Otra característica de los EAR es que permite el control unificado (arranque, despliegue, parada...) del conjunto de las aplicaciones contenidas.

Físicamente, un fichero EAR es un fichero comprimido con el mismo formato que los ficheros JAR. Todos los comandos que se pueden utilizar con los ficheros JAR (jar -tvf mi-fichero.jar, etc.) sirven para los ficheros EAR. Los ficheros EAR llevan la extensión .ear. La estructura de un fichero EAR es sencilla, contiene un conjunto de ficheros WAR, un conjunto de ficheros JAR y el directorio *META-INF*, en el que se encuentra los distintos ficheros de configuración necesarios. Opcionalmente puede contener el descriptor de despliegue *application.xml* en el que se identifican los módulos que se incluyen en él.

Despliegue de aplicaciones en WildFly

WildFly nos ofrece tres posibilidades:

- Despliegue desde sistema de ficheros (sólo en modalidad standalone).

```
cp example.war /usr/local/wildfly-8.2.1.Final/standalone/deployments
```

Escáner de despliegue WildFly distingue entre aplicaciones empaquetadas (Jar,War,Ear) o aplicaciones descomprimidas (exploded). Por defecto el autodeploy está habilitado para las aplicaciones empaquetadas pero no para las exploded. En el caso de que el despliegue sea manual, se controla mediante ficheros auxiliares llamados *markers*. Estos ficheros también aparecen cuando falla un despliegue por autodeploy (fichero application.FAILED). Si lo eliminamos, el escáner intentará desplegar la aplicación de nuevo.

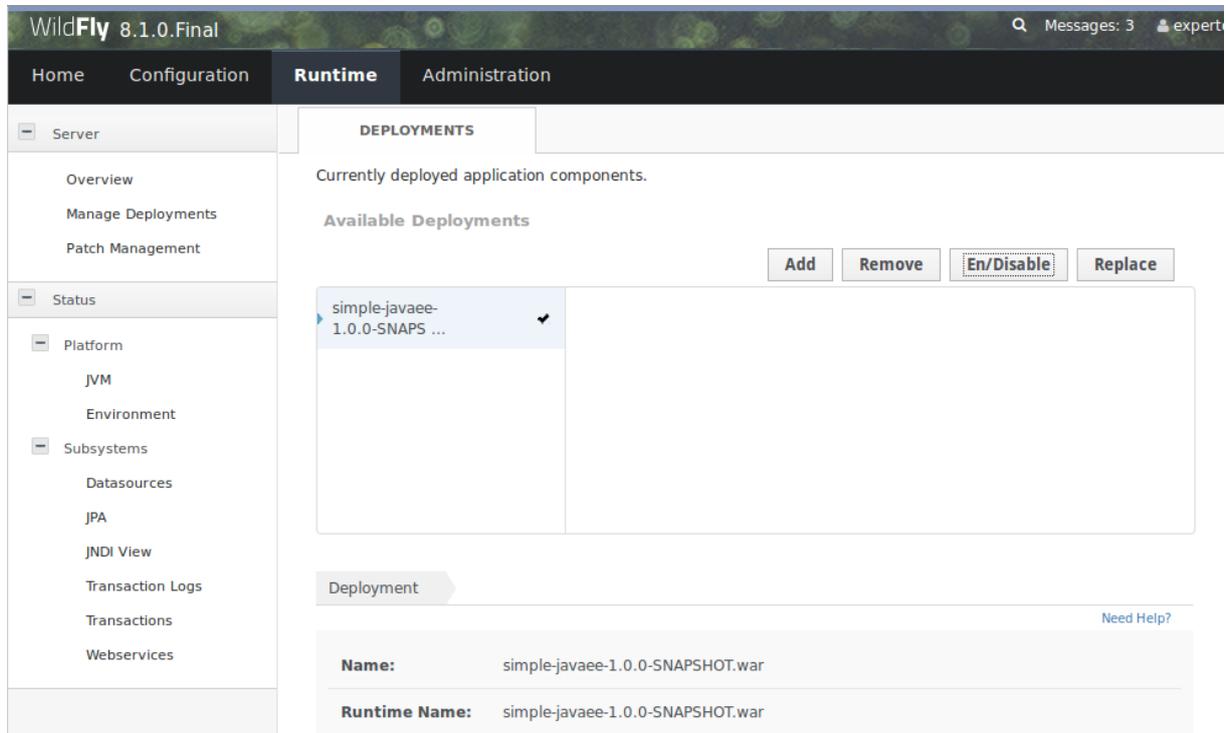
La configuración del escáner de despliegues se realiza sobre el fichero standalone.xml o a través de CLI:

```
<deployment-scanner scan-interval="5000" relative-to="jboss.server.base.dir"
  path="deployments" auto-deploy-zipped="true" auto-deploy-exploded="false"/>
```

Para más detalles podéis consultar la documentación de WildFly: <https://docs.jboss.org/author/display/WFLY8/Application+deployment>

- Mediante las herramientas de administración

```
[standalone@localhost:9990/]deploy example.war
```



- Uso de herramientas de terceros como por ejemplo wildfly-maven-plugin

```
<plugin>
  <groupId>org.wildfly.plugins</groupId>
  <artifactId>wildfly-maven-plugin</artifactId>
  <version>1.0.2.Final</version>
  <executions>
    <execution>
      <phase>install</phase>
      <goals>
        <goal>deploy</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

```
[standalone@localhost:9990/]mvn wildfly:deploy
[standalone@localhost:9990/]mvn wildfly:redploy
[standalone@localhost:9990/]mvn wildfly:undeploy
```



Ejecutar aplicación sin instalación previa de WildFly *wildfly:run*

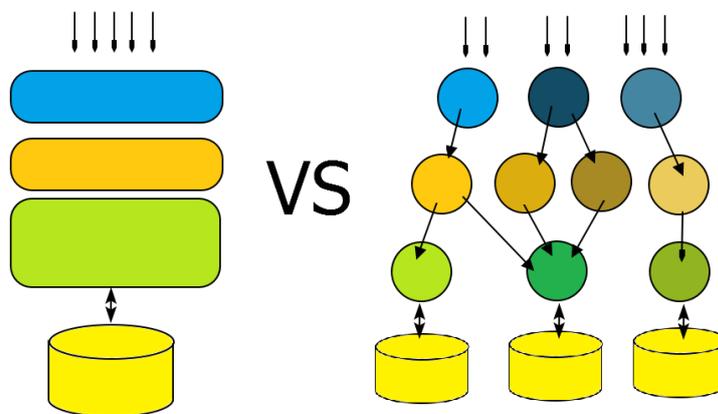
Es una función interesante del plugin de maven de cara a realizar pruebas o ejecutar la aplicación en una máquina donde no tenemos

configurada ninguna instancia de WildFly. Este comando Maven descarga automáticamente un servidor WildFly en modo standalone y despliega en el la aplicación compilada.

1.6. Arquitecturas basadas en microservicios

Una arquitectura basada en microservicios es una forma de entender las aplicaciones como un conjunto de pequeños servicios, cada uno ejecutándose como proceso independiente y comunicándose con el resto mediante mecanismos sencillos y "agnósticos", a menudo HTTP. Estos servicios se diseñan sobre funciones de negocio y siguen estrategias de de despliegue independientes. Aunque se trate de servicios independientes suele existir una pequeña lógica de administración común para estos servicios, que pueden estar contruidos usando distintos lenguajes de programación y diferentes tecnologías de almacenamiento.

Martin Fowler (padre del término microservicio)



Este tipo de arquitectura nace en contraposición a las grandes aplicaciones *Monolíticas*. En el mundo Java, una aplicación monolítica podría ser un EAR con una serie de módulos y librerías dependientes que se gestiona como una unidad. En si mismo esto no tiene porqué ser un problema (sigue siendo un modelo válido) hasta que la propia envergadura del proyecto o la dificultad para alcanzar requerimientos de escalabilidad o separación de responsabilidades evidencien dificultades de gestión. En cierta medida microservicios trata más de organización del trabajo que de tecnologías concretas.

También hay que destacar que el término de "micro" no se refiere al tamaño del servicio si no a la funcionalidad que desempeña. Para definir correctamente un microservicio hay que apoyarse en buenas prácticas de programación ya establecidas como la alta cohesión y bajo acoplamiento o el Principio de Responsabilidad Única (Un subsistema, módulo o incluso clase sólo debe tener un motivo para cambiar).

Se podría entender como un SOA 2.0 (Arquitectura Orientada a Servicios) o un SOA pragmático donde se sustituyen protocolos pesados como SOAP y elementos como ESB's donde se ejecutaba gran cantidad de lógica por servicios donde la inteligencia está en el propio servicio y no existe un control y persistencia centralizados.

Ventajas de una arquitectura basada en Microservicios sobre una monolítica

Facilidad para escalar el desarrollo

Permite dividir el trabajo entre equipos de trabajo reducidos, con conocimientos completos del stack de desarrollo y con una responsabilidad funcional acotada. (Two Pizza teams)

Aislamiento ante fallos

Si un servicio tiene un fallo, es factible corregir dicho servicio y desplegarlo individualmente sin afectar al resto de la aplicación.

Despliegues más rápidos

Poder desplegar por partes facilita el poder realizar despliegues a producción más frecuentes y con un riesgo menor.

Elimina la dependencia a largo plazo de una tecnología

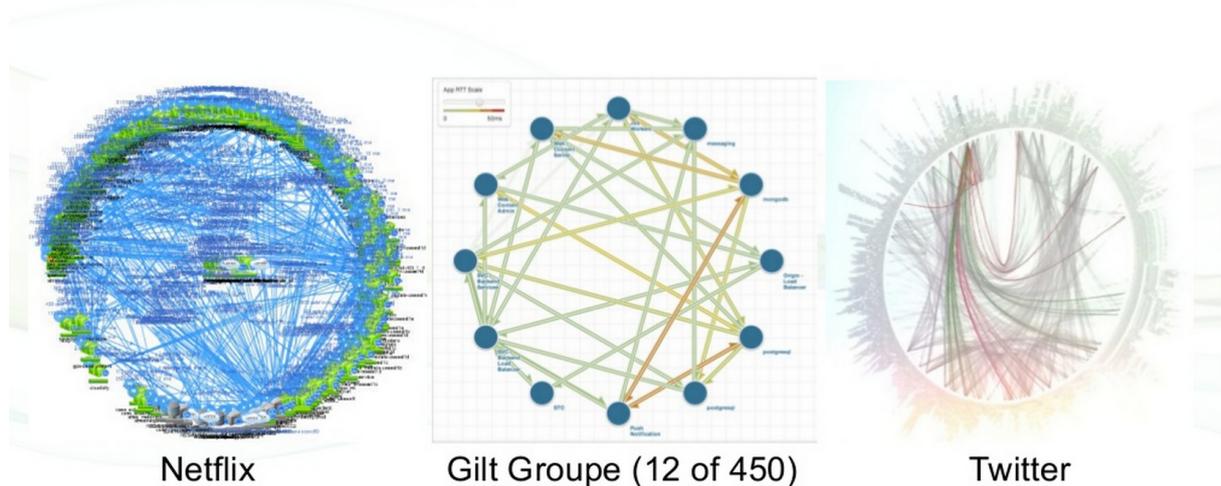
El utilizar mecanismo de comunicación agnósticos, permite que podamos utilizar la tecnología más adecuada para cada servicio, en contraposición al modelo de EAR donde se elige un único "stack" para toda la aplicación. Si en un momento decidimos cambiar de tecnología, podemos ir reemplazando paulatinamente los servicios afectados.

Desventajas

Complejidad de un sistema distribuido

Servicios independientes requieren una monitorización independiente, recursos de ejecución adicionales, herramientas de descubrimiento, gestión de logs... en definitiva costes y complejidad adicionales a las de un proyecto monolítico.

“Death Star” Architecture Diagrams



Requieren cultura DevOps

Las iniciativas DevOps permiten automatizar los procesos de validación y despliegue de los nuevos servicios y allanar las dificultades anteriores. Comentaremos con más detalle qué es DevOps en sesiones posteriores.

Falta de herramientas

Las herramientas tradicionales están orientados a desarrollos monolíticos y no tanto a proyectos complejos distribuidos.

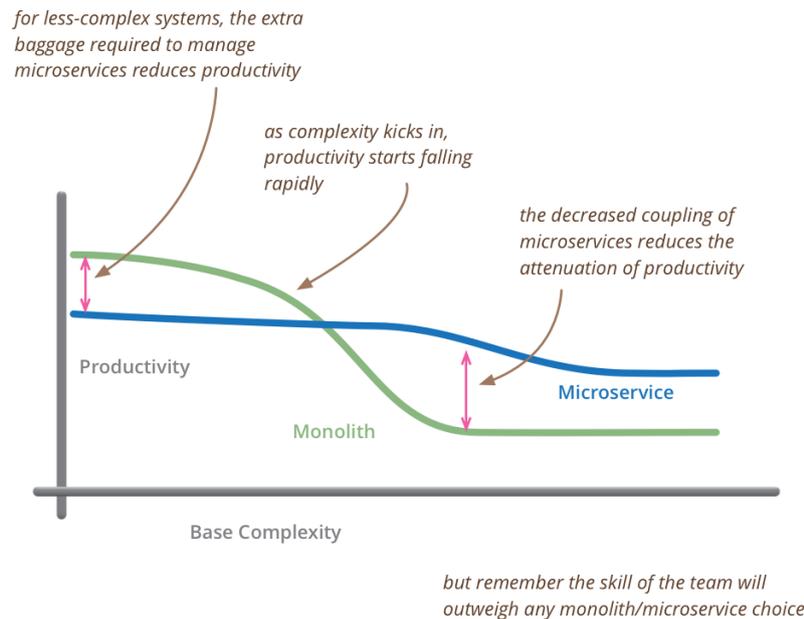
Comunicación y coordinación entre equipos

Si bien gestionar un equipo grande es complejo, el coordinar y compartir conocimiento entre equipos independientes también plantea retos.

Cuándo utilizar microservicios

Según Martin Fowler no se debería considerar una arquitectura basada en microservicios hasta tener un sistema monolítico que sea demasiado complejo de mantener. Nosotros

añadiríamos otro motivo: cuando las necesidades de escalabilidad del trabajo lo justifiquen o se necesite utilizar distintas tecnologías en distintas funciones de negocio.



Para un desarrollo ya existente, un enfoque muy aceptado es asegurar en primer lugar una que tenemos una aplicación monolítica correcta en términos de arquitectura:

- Separación de funciones.
- Alta cohesión y bajo acoplamiento utilizando APIs bien definidas.
- Sin redundancias.
- Diseño dirigido por el Dominio (DDD).
- Lo demás "sobra".

De forma progresiva, se puede ir identificando las distintas funciones de negocio y separarlas en servicios siguiendo el *principio de responsabilidad única* (SPR). Una vez definidos los servicios más básicos, las aplicaciones harán uso de ellos combinándolos según diferentes patrones (Ver referencias).

Una vez definidos los servicios básicos las aplicaciones harán uso de ellos combinándolos siguiendo distintos patrones:

Java EE Y los microservicios

El estado de evolución actual de la especificación Java EE permite afrontar el desarrollo de aplicaciones basados en nuevas arquitecturas. Hoy en día, es razonable utilizar un servidor de aplicaciones para desplegar un único servicio basado en Java EE sin incurrir en un consumo excesivo de recursos y beneficiándonos de años de estandarización y una amplia gama de servicios.

En cifras, usar un servidor de aplicaciones completo hoy en día puede suponer un sobrecoste de unos 30-50Mb de RAM sobre una aplicación java standalone , así como unos 100Mb de espacio en disco y el tiempo de arranque puede ser de 1-3 segundos de tiempo. En una aplicación empresarial con un número de servicios limitado esto no supone un problema. De hecho, resulta curioso ver cómo se ha ido reduciendo el consumo de recursos en la parte servidora y sin embargo hemos sobrepasado el consumo de las aplicaciones de escritorio pesadas en el lado del cliente (Vease Chrome, por ejemplo).

Sólo en el caso de aplicaciones altamente escalables y orientadas a internet (al nivel de Google, Netflix, etc.), merece la pena invertir esfuerzos en reducir al máximo las librerías necesarias para que un servicio funcione. Aún así puede ser más ventajoso aprovechar la modularidad de los servidores de aplicaciones que construir una solución desde cero.

Hay ponentes como Adam Bien que plantean un modelo minimalista de aplicación Java EE con las siguientes características:

1 servidor de aplicaciones, 1 dominio y 1 WAR

El concepto de servicio quedaría constituido por un servidor de aplicaciones que ofrezca todo el API Java EE (Full Profile) a una aplicación desplegada como WAR que implemente un servicio.

WARs ligeras

El tamaño de un fichero WAR y su complejidad interna influye negativamente en los tiempos de despliegue y arranque de una aplicación. Para conseguir mejores resultados Adam Bien propone los siguientes cambios sobre el modelo de desarrollo de aplicaciones Web:

- Utilizar preferentemente las librerías que proporciona el servidor de aplicaciones, en lugar de incluirlas en la aplicación. Esto ya puede suponer pasar de trabajar con un fichero de varios megabytes a unos pocos Kilobytes.
- Aplicaciones Autocontenidas: toda la lógica del servicio debe estar incluida dentro del War.
- Reducir o eliminar los módulos JAR internos. Ya hemos comentado que es habitual trabajar con aplicaciones WAR que contienen a su vez librerías JAR con clases diversas y EJB's. Este esquema ralentiza las compilaciones y despliegues y en realidad la aplicación siempre se compila o despliega como un todo, por lo que la división interna en librerías no aporta ventajas en muchos casos.

Estructura del código

- No utilizar interfaces. Únicamente los utilizaremos cuando sepamos que vamos a tener diferentes implementaciones para una misma funcionalidad, pero hasta entonces no complicar el código innecesariamente.
- Centrarnos en el código de negocio. Java EE permite que las aplicaciones estén compuestas básicamente por POJOS y permite abstraer la complejidad de los servicios. El trabajar con objetos pojo tiene el beneficio adicional de facilitar las pruebas unitarias
- Pocas anotaciones (Convención sobre declaración): Evitar declarar propiedades que ya tengan el mismo valor por defecto dentro de su especificación.

Ejemplo de microservicio *minimalista*:

```
.....  
@Path("cursos")  
public class CursosResource {  
  
    @GET  
    @Produces("application/json")  
    public JsonObject all(){  
        return Json.createObjectBuilder().add("cursos", "JavaEE,  
JavaScript, MongoDB").build();  
    }  
  
}
```

.....

Docker



Docker ha supuesto el salto a la fama del concepto de *contenerización* de servicios y viene a ser una mezcla de máquina virtual con control de versiones tipo Subversion. Permite desplegar todo el stack de una aplicación o servicio en una máquina con un simple comando *pull* como si fuera un único paquete. Se apoya los conceptos básicos de *imagen* y *contenedor*.

Una **imagen** se puede considerar como un sistema de archivos en el que podemos añadir lo que la aplicación necesite para funcionar, como una máquina virtual de Java o la instalación de un servidor de aplicaciones. Una característica fundamental es que está formado por capas, que funcionan como los *commit* de Git o SVN. Cada vez que se añade una capa, se superponen nuevos ficheros sobre la imagen inicial. En este caso es fundamental dejar en la última capa el despliegue de la aplicación para que cuando ésta se modifique, la imagen conserve todas sus capas y sustituya únicamente la última (que tendrá el tamaño del WAR desplegado). Aquí tenemos una importante mejor en el caso de las aplicaciones Java ligeras que proponemos, especialmente si la comparamos con las basadas en Java StandAlone (o FAT Jars).

Llamamos **contenedor** a la instancia en ejecución de una imagen. Apoyándose en los mecanismos de seguridad de Linux, el container se ejecuta en un espacio de memoria separado, con lo que es tarea del programador el definir la comunicación entre los procesos del container y el resto de la máquina (gestionando los puertos o montando directorios físicos fundamentalmente). Hay que tener siempre presente que cada vez que se inicia un contenedor, se parte de la imagen definida por lo que cualquier cambio que se realice sobre el sistema de ficheros propio del contenedor NO se almacenará en la imagen y por tanto se perderá.

Entender Docker es mucho más sencillo en la práctica, así que en los ejercicios veremos un pequeño ejemplo.

1.7. Referencias

- Información sobre Roles Java EE <http://docs.oracle.com/javaee/6/tutorial/doc/bnaca.html> .
- Wiki con los distintos servidores de aplicaciones y sus especificaciones http://en.wikipedia.org/wiki/Java_Platform,_Enterprise_Edition .
- Información sobre WildFly <http://wildfly.org/about/> .
- Guía de iniciación <https://docs.jboss.org/author/display/WFLY8/Getting+Started+Guide> .
- Guía completa de administración <https://docs.jboss.org/author/display/WFLY8/Admin+Guide#AdminGuide-Examplesinthisguide> .
- Despliegues desde sistema de ficheros <https://docs.jboss.org/author/display/WFLY8/Application+deployment>
- WildFly Maven Plugin <https://docs.jboss.org/wildfly/plugins/maven/latest/index.html>
- Java EE 7 Examples <https://github.com/javaee-samples/javaee7-samples>

- Spring Boot <http://projects.spring.io/spring-boot/>
- Dropwizard <http://www.dropwizard.io/>
- WildFly Swarm <http://wildfly.org/swarm/>
- Payara <http://www.payara.co.uk/about>
- WAS Liberty <https://developer.ibm.com/wasdev/>
- Desarrollo con WildFly y Docker <http://tools.jboss.org/blog/2015-03-03-docker-and-wildfly-2.html>
- Introducción a los microservicios <http://microservices.io/patterns/microservices.html>
- Patrones de diseño para microservicios <http://blog.arungupta.me/microservice-design-patterns/>
- Docker <https://www.docker.com/>
- Simulador online de Docker <http://dockersim.com/>
- Docker Maven plugin <https://github.com/spotify/docker-maven-plugin/tree/master>
- ECB Model <http://www.cs.sjsu.edu/~pearce/modules/patterns/enterprise/ecb/ecb.htm>

1.8. Ejercicios de Introducción a los servidores de aplicaciones y a WildFly

Al ser los primero ejercicios del módulo de PaaS debéis hacer un fork del repositorio *ejercicios_paas*. En el debéis almacenar todos los ejercicios del módulo Servidores Web y PaaS independientemente de que luego en IntelliJ se gestionen en proyectos separados. De hecho, en las sesiones OpenShift trabajaremos con repositorios Git independientes por aplicación.

Además de este repositorio, tenéis los ejemplos de cada sesión dentro de la carpeta *ejemplos* del mismo repositorio, por si queréis probarlos con más detalle.

Desarrollo de un microservicio (0.3 Puntos)

Partiendo de que ya habéis trabajado con WildFly en sesiones anteriores vamos a centrarnos en el desarrollo de una aplicación Java EE, de nombre *cursos*, que sirva para practicar los distintos tipos de despliegue que admite el servidor tanto como instalación local como virtualizada a partir de una imagen de Docker.

Para crear el proyecto utilizaremos Maven y un arquetipo elaborado por Adam Bien, que ofrece una aplicación web similar al arquetipo estándar *webapp-javaee7* pero con una única dependencia que permite referenciar todo el API de Java EE. Básicamente es un proyecto web minimalista con todo lo necesario para desarrollar.

GroupID	Artefacto	Versión
com.airhacks	javaee7-essentials-archetype	1.3

Pasos a seguir

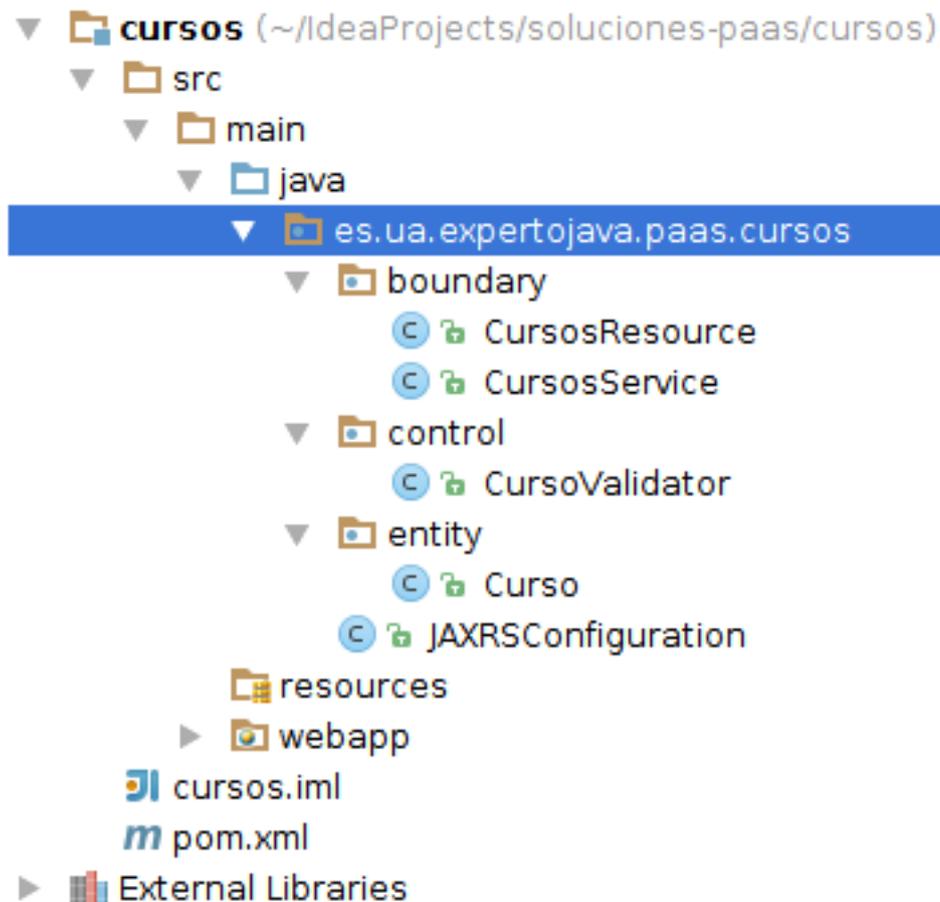
1.- Crearemos un nuevo módulo Maven dentro del proyecto de ejercicios a partir de este arquetipo y lo denominaremos *cursos*. Las propiedades completas del nuevo artefacto serán:

GroupID	Artefacto	Versión
org.expertojava.paas	cursos	1.0

Por último indicaremos que el nuevo módulo dentro de IntelliJ se llamará *cursos* así como el directorio dentro del proyecto principal.

2.- Renombraremos el paquete *com.airhacks* del proyecto generado a *es.ua.expertojava.paas.cursos* mediante la herramienta *Refactor* de IntelliJ. También eliminaremos el paquete *com* que queda vacío tras el proceso anterior.

3.- Dentro del repositorio de ejercicios hay una carpeta *sesion1* con subcarpetas de nombre: *boundary*, *control* y *entity*. Debéis copiar las tres carpetas a dentro de la carpeta *es/ua/expertojava/paas/cursos/* del proyecto. Al hacerlo IntelliJ automáticamente importará las clases Java contenidas en dichas carpetas. La estructura del proyecto debe quedar de la siguiente forma:



Lo que estamos haciendo es dividir una función de negocio "cursos" en tres paquetes distintos con diferentes cometidos:

Boundary

Contiene las clases que definen la interfaz entre la función de negocio y el exterior.

Control

Son las clases que definen la ejecución del servicio e interactúan con las otras dos capas.

Entity

Son las clases que gestionan elementos de información persistentes, de mayor o menor granularidad pero con significado propio.

Esta forma de trabajar se llama patrón Entity-Control-Boundary y es una variación del MVC orientado a servicios.

4.- Añadir la unidad de persistencia al proyecto. Para ello basta con copiar la carpeta META-INF del directorio *sesion1* a la ruta */src/main/resources* del proyecto.

5.- Crear la configuración de despliegue en IntelliJ para poder desplegar la aplicación en nuestra instancia de WildFly.



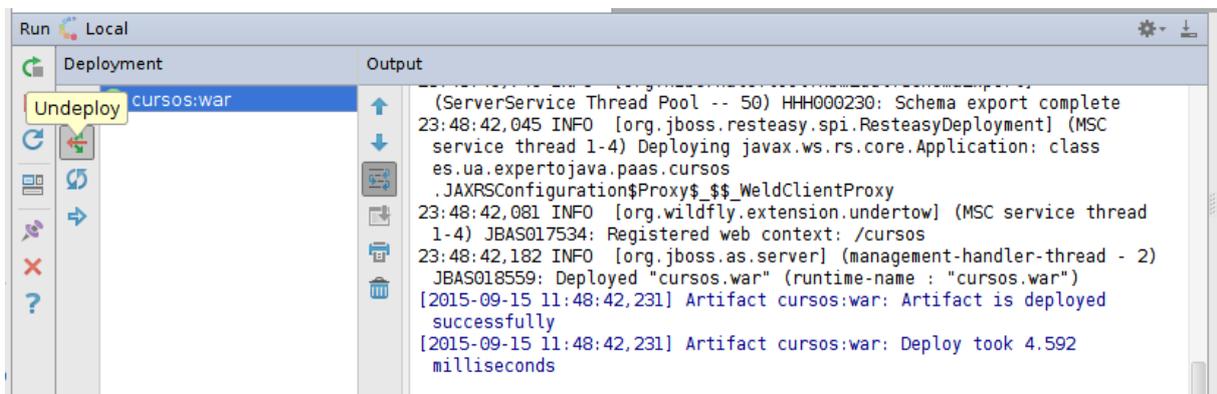
Si la configuración de ejecución os muestra un mensaje de error indicando que no está definida la versión del JDK a utilizar, ésta se puede establecer desde File → Project Structure → Project JDK. En nuestro caso trabajaremos con JDK 1.8.

6.- Una vez desplegada la aplicación comprobar que funciona correctamente mediante la herramienta *curl*:

```
#Dar de alta un curso
curl -H "Content-Type: application/json" -X POST -d '{"nombre":"Curso de prueba"}' http://localhost:8080/cursos/resources/cursos
```

```
#Obtener la lista de curso almacenada
curl -X GET http://localhost:8080/cursos/resources/cursos
```

7.- Realizar el undeploy desde IntelliJ



Otras formas de despliegue (0.3 Puntos)

1. Indicar en el fichero **soluciones.txt**, dentro de vuestro proyecto de *ejercicios_paaS*, los pasos a seguir para realizar las siguientes tareas:

1. Desplegar la aplicación mediante Autodeploy.
 2. Desplegar la aplicación mediante CLI.
2. Crear un script de CLI, de nombre **script.cli**, que liste las aplicaciones desplegadas, elimine la aplicación *cursos.war* y que pare el servidor.

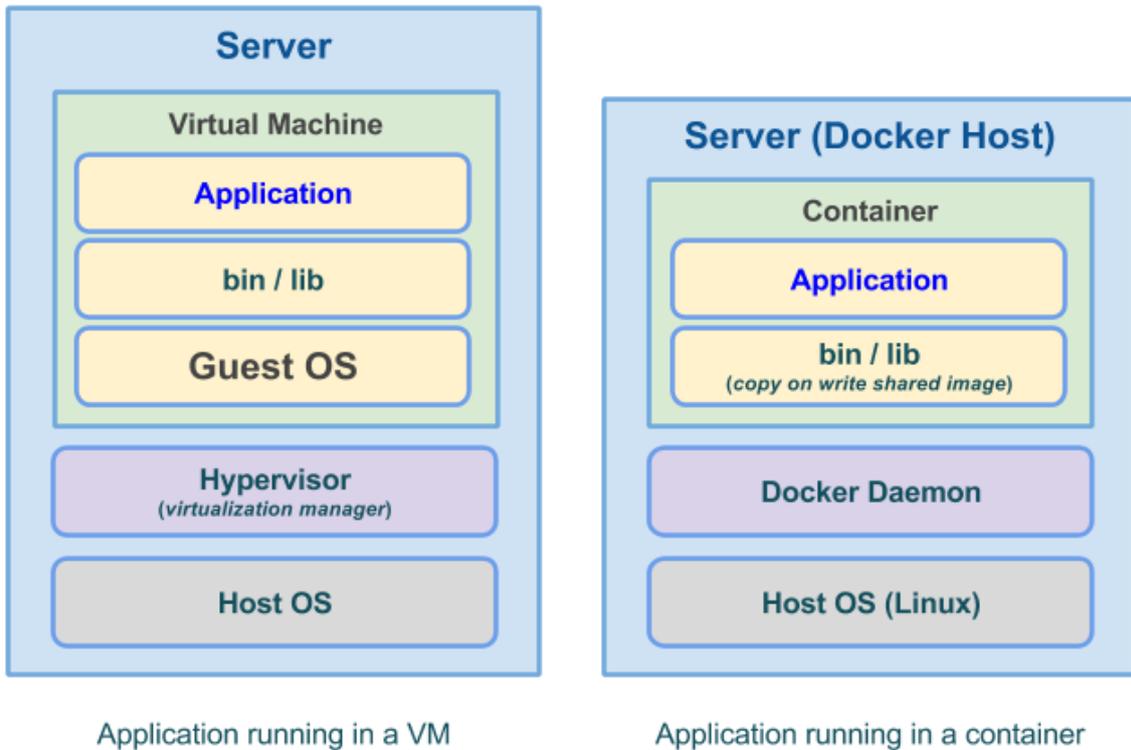
Virtualización con Docker (0.6 Puntos)



Nota

Este ejercicio será una demostración *guiada* para que os familiaricéis con Docker, y sentar las bases para quien quiera profundizar más en el tema.

La principal diferencia entre una aplicación desplegada en una máquina virtual con respecto a Docker es que el conjunto de capas sobre las que se apoya la aplicación es más reducido:



Los contenedores se ejecutan en el espacio de memoria del Sistema operativo Host (debidamente aislados) y no requieren disponer de un sistema de archivos tan complejo como el de un sistema operativo, si no que puede apoyarse en las herramientas específicas necesarias. El resultado es que trabajamos con imágenes más pequeñas y tiempos de arranque mucho más rápidos. También hay que recordar que las imágenes están compuestas por capas apiladas por lo que un cambio en la aplicación desplegada puede suponer sustituir una sola capa, si la imagen está bien estructurada (si se tiene que modificar una capa intermedia, se reemplazan todas las capas dependientes).



La composición en capas de una imagen se define paso a paso en un fichero denominado *Dockerfile*. Un ejemplo:

```
FROM jboss/wildfly:8.2.1.Final
MAINTAINER José Luis Zamora Sánchez joseluiszamora@jlz.gmail.com
```

```
EXPOSE 8080 9990
RUN /opt/jboss/wildfly/bin/add-user.sh expertojava expertojava --silent
COPY target/cursos.war /opt/jboss/wildfly/standalone/deployments/
CMD ["/opt/jboss/wildfly/bin/standalone.sh", "-b", "0.0.0.0", "-bmanagement", "0.0.0.0"]
```

Lo primero que se indica es la capa base que vamos a utilizar, y para ello podemos utilizar una imagen que ya tengamos instalada, o bien una imagen que nos descarguemos de un repositorio. El repositorio por defecto de Docker es **Docker-Hub**, con un planteamiento muy similar a Git Hub. En este repositorio es donde se suelen publicar las imágenes oficiales de todos los proyectos Open Source.

También debemos indicar quien será el responsable mantener esta imagen y cómo contactar con él.

La comunicación entre el contenedor y el sistema operativo host se realiza mediante conexiones de red, de forma muy similar a otras soluciones como VirtualBox, si bien la definición se realiza en dos partes:

- Desde el fichero Dockerfile donde se indica que puertos serán accesibles desde el exterior con el comando EXPOSE.
- Al iniciar el contenedor donde podemos mapear puertos del Host con puertos del contenedor.

A continuación veréis una serie de comandos que realizan operaciones sobre el sistema de archivos como RUN, COPY que realmente se ejecutan en el momento de generar la imagen, no al iniciarse. Es importante indicar que cada línea del fichero Dockerfile supone una capa, por lo que si agrupamos varios comandos en una línea, físicamente se agruparán en una capa. Si os fijáis, con el comando RUN estamos definiendo quien será el usuario administrador de la instancia de WildFly.

Esto es muy útil, por ejemplo, si copiamos un fichero ZIP con un servidor de aplicaciones, lo descomprimos y lo eliminamos en la misma fila. Si lo hacemos así, sólo se almacenarán las diferencias, es decir la nueva carpeta descomprimida del servidor. Si lo hacemos paso a paso, tendremos una capa intermedia con el ZIP completo, que tendremos que descargar para luego eliminarlo en capas posteriores.

Por último, con el comando CMD estamos indicando qué se tiene que ejecutar por defecto al iniciar un contenedor.

Ciente docker

Para trabajar con imágenes y contenedores, necesitamos una herramienta cliente, y en nuestro caso es el comando *docker*. En este enlace tenéis una "chuleta" con comandos útiles para trabajar con imágenes y con contenedores:

<https://github.com/javaee-samples/docker-java/blob/master/chapters/docker-commands.adoc>

Objetivos

Vamos a crear una imagen independiente que despliegue nuestro servicio, para ello seguiremos los siguientes pasos:

1. Abrir una cuenta gratuita en Docker Hub

La idea es que vuestra imagen quede publicada y que nosotros podamos validarla directamente a partir del repositorio, con lo que debéis acceder a la página de Docker Hub y registraros como usuarios. Posteriormente configuraréis vuestro cliente de Docker para que se conecte a Docker Hub utilizando vuestra cuenta:

```
.....
expertojava@expertojava:~/IdeaProjects/cursos$ docker login

Username: <vuestro usuario>
Password:
Email: <vuestro correo>
WARNING: login credentials saved in /home/expertojava/.docker/
config.json
Login Succeeded
.....
```

2. Simular cómo se actualizaría y desplegaría un servicio en producción:

```
.....
docker run -it --name=miservicio -p 8080:8080 -p 9990:9990 djbyte1977/
wildfly # Para crear un nuevo contenedor
docker pull djbyte1977/wildfly # Para actualizar la imagen
.....
```

Con estos comandos os descargaréis la última versión de la imagen wildfly de ejemplo que hemos preparado con la aplicación cursos.war ya desplegada y preparada para funcionar. También estamos mapeando los puertos 8080 y 9090 con sus equivalentes en el contenedor. El fichero Dockerfile utilizado para crear esta imagen es el que tenéis de ejemplo, por lo que si se modifica la aplicación cursos.war no tendreis que volver a descargar las capas anteriores (WildFly base + creación del usuario administrador).

El comando `run -it` crea un contenedo con el servidor de aplicaciones en modo interactivo, por lo que será análogo a lanzar un servidor directamente desde un CMD. Alternativamente sería posible lanzarlo en background con el comando:

```
.....
docker run -d --name=miservicio -p 8080:8080 -p 9990:9990 djbyte1977/
wildfly
.....
```

El comando `run` realmente crea y ejecuta un contenedor por lo que si se ha usado previamente y se ha especificado un nombre, en la siguiente ejecución nos dará un error indicando que ya existe un contenedor con ese nombre. Una vez ya se ha asignado un nombre al contenedor se puede ejecutar simplemente:

```
.....
docker start miservicio    #-i modo interactivo
.....
```

Para enumerar los contenedores existentes tenemos el comando:

```
.....
docker ps    # Lista los controles en ejecución
docker ps -a # Lista todos los controles
.....
```

Una vez hemos lanzado un contenedor en background, lo podemos gestionar mediante los comandos de docker específicos. Ejemplo:

```
# Conectarse a la salida estándar del contenedor
docker attach --sig-proxy=false miservicio

# Parar el contenedor
docker stop miservicio
# Matar el contenedor vía SIGKILL
docker kill miservicio
```

3. Empaquetar el WAR cursos en una imagen Docker

Lo primero que tenéis que hacer es crear un fichero Dockerfile con los mismos pasos indicados en el ejemplo y dejarlo en la carpeta donde tengáis vuestro proyecto:

```
expertojava@expertojava:~/IdeaProjects/cursos$ ls
cursos.iml  Dockerfile  pom.xml  README.md  src  target  WEB-INF
expertojava@expertojava:~/IdeaProjects/cursos$
```

Con la aplicación ya compilada, ejecutaréis el siguiente comando:

```
docker build -t <usuario>/wildfly .
```

Con este comando habéis creado una nueva imagen asociada a vuestro usuario, pero que únicamente existe en el repositorio local. Si queremos compartirla, la debemos subir a Docker Hub, para ello ejecutaréis:

```
docker push <usuario>/wildfly
```

4. Repetir el primer paso pero indicando la imagen que habéis creado y dándole un nombre distinto al del ejemplo. Si lo ejecutáis desde vuestra máquina no os descargaréis ninguna capa del repositorio, sin embargo si el servicio estuviera desplegado en otra máquina, al ejecutar un *pull* se descargaría la capa correspondiente al nuevo WAR.



Importante

La entrega final de este ejercicio constará del código fuente del proyecto, incluyendo el nuevo Dockerfile, y el nombre del repositorio asociado en Docker Hub para que podamos descargar y probar el servicio. El nombre del repositorio se indicará en el fichero **soluciones.txt**

Entrega

En esta sesión debéis entregar el código fuente del proyecto cursos, el fichero script.cli, y el fichero soluciones.txt actualizado con la información solicitada en los ejercicios. :huge-size: scaledwidth="100%" :large-size: scaledwidth="66%" :normal-size: scaledwidth="50%" :small-size: scaledwidth="33%" :tiny-size: scaledwidth="25%"

2. Administración de recursos en WildFly

En esta sesión vamos a centrarnos en la configuración básica de una instancia de WildFly. Aunque hemos hablado de modalidades de trabajo en Dominio, vamos a centrarnos en los modos standalone por sencillez y porque no nos limitan el acceso a características de alta disponibilidad como en otros servidores.

2.1. Fichero de configuración del Bootstrap

Hay una cuestión básica que es definir cuanta memoria puede utilizar la aplicación. La forma de definir el tamaño del Heap en Wildfly es bien especificarlo como parámetro del script de arranque, o lo que es más facil, editar el fichero standalone.conf

```
$JBOSS_HOME/bin/standalone.conf
```

```
JAVA_OPTS="-XX:MaxPermSize=256m -Xms64m -Xmx512m"
```



JDK 8

En la máquina virtual de la versión 8 desaparece el espacio de memoria denominado Permanent Generation y en su lugar se define uno nuevo denominado Metaspace. La forma de limitarlo es muy similar:

```
$JBOSS_HOME/bin/standalone.conf
```

```
JAVA_OPTS="-XX:MaxMetaspaceSize=256m -Xms64m -Xmx512m"
```

En este fichero también podemos especificar una JVM de Java concreta, así como otras propiedades como por ejemplo el número de descriptores de ficheros que podrá utilizar WildFly

2.2. Estructura de los ficheros de configuración

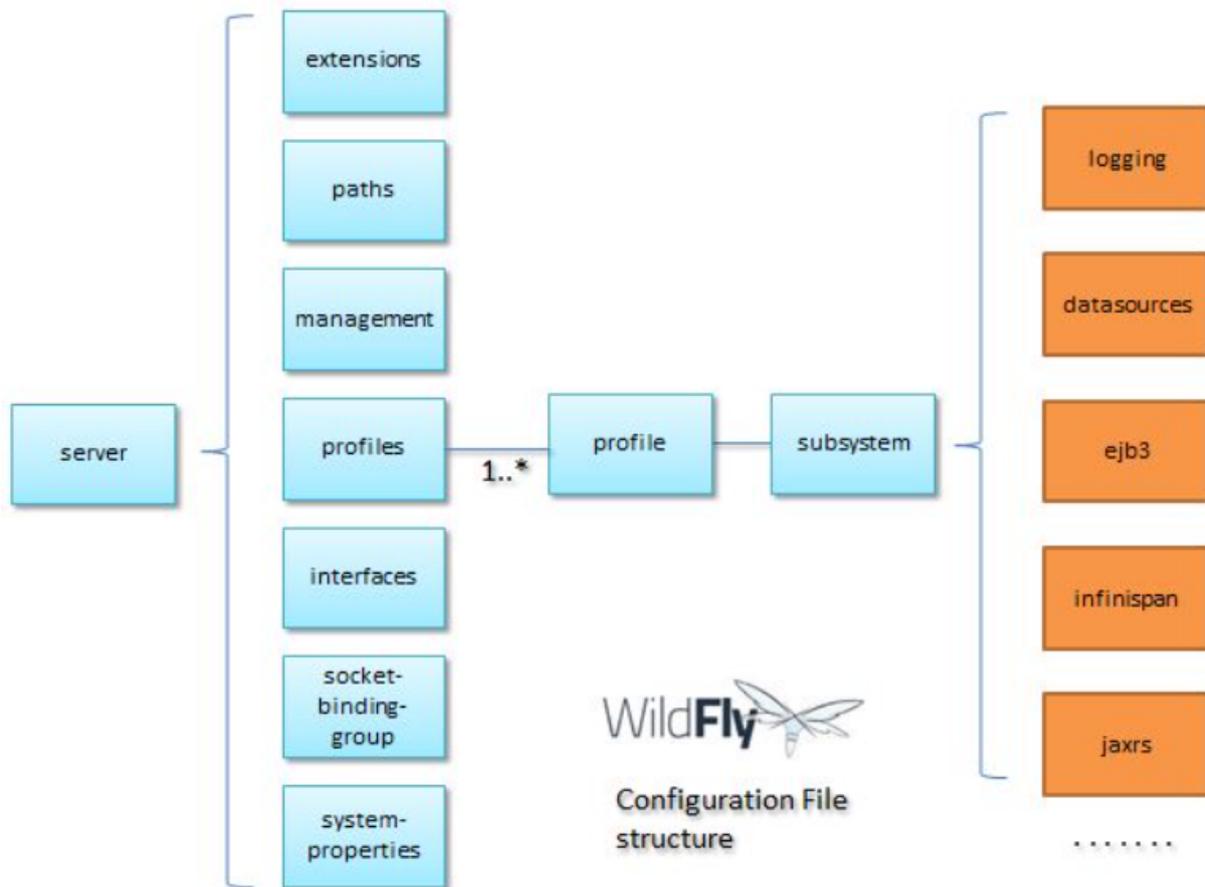
En la sesión anterior comentamos que las herramientas de administración modificación una configuración que físicamente se almacena en ficheros XML. Para cada modo de inicio (Standalone o Domain Managed) existen diferentes configuraciones de inicio, en el caso de operar como servidor independiente. Estos ficheros se encuentran dentro de la carpeta correspondiente a la modalidad (domain o standalone).

- standalone.xml Configuración por defecto, sin soporte JMS.
- standalone-full.xml Añade soporte JMS (HornetQ).
- standalone-ha.xml Soporte Clustering.
- standalone-full-ha.xml Soporte Clustering y JMS.

Se puede especificar un fichero de configuración específico, distinto de la configuración por defecto con el parámetro "-c"

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

Los ficheros de configuración presentan la siguiente estructura:



El elemento raíz es la etiqueta <server> y de ella cuelgan los siguientes elementos:

Profiles

Los perfiles son configuraciones de servidor y sus distintos subsistemas. En la modalidad standalone, Wildfly sólo permite un perfil por fichero de configuración, sin embargo en el modo domain, es posible definir varios perfiles para distintos tipos de servidores dentro de un dominio. Ejemplo:

```
<profile>
...
  <subsystem xmlns="urn:jboss:domain:deployment-scanner:2.0">
    <deployment-scanner path="deployments" relative-
to="jboss.server.base.dir" scan-interval="5000"/>
  </subsystem>
...
</profile>
```

Extensions

La mayor parte de las funcionalidades de WildFly se configuran como extensiones, que implementan especificaciones de Java EE.

```
<extensions>
```

```
[...]
<extension module="org.jboss.as.transactions"/>
<extension module="org.jboss.as.web" />
<extension module="org.jboss.as.webservices" />
<extension module="org.jboss.as.weld" />
</extensions>
```

Paths

Los paths son rutas a directorios o ficheros de la máquina a las que se les asocia un nombre lógico. Los paths pueden ser rutas absolutas o relativas a otro path, de modo que la configuración sea portable a otras máquinas:

```
<path name="log.dir" path="/home/wildfly/logs" /> <!-- Ruta absoluta -->
<file relative-to="log.dir" path="server.log"/> <!-- Ruta relativa a
log.dir -->
```

Interfaces

Una interfaz es un nombre lógico asociado a un nombre de máquina/dirección IP a través de la cual podemos acceder a nuestro servidor. Por defecto se definen tres interfaces:

- **management** La dirección que se utilizará para atender peticiones de administración.
- **public** Esta es la dirección de servicio que atenderá el acceso a las aplicaciones.
- **unsecure** Esta es la dirección que se utilizará para la invocación de objetos remotos IIOP.

```
<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>
```

Por defecto estas interfaces están vinculadas a propiedades del sistema, que se pueden incluso sobrescribir en el inicio del servidor:

```
./standalone.sh -Djboss.bind.address=192.168.1.100
```

Socket binding groups

Un socket binding es básicamente la definición de un punto de escucha y se compone de la interfaz y del puerto de red concreto que se va a utilizar.

```
<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-group/>
```

El parámetro port-offset se utiliza para desplegar en una misma máquina distintas instancias de un servidor y asignar a cada una de ellas puertos distintos. El valor del offset se suma a todos los puertos especificados. Una buena práctica es establecer el valor de offset como propiedad del sistema en el inicio, en lugar de indicarlo de forma estática en la configuración.

System properties

Estas propiedades de configuración del servidor, se pueden establecer de varias formas:

- En el fichero de configuración
-

```
<system-properties>  
  <property-name="propiedad" value="true" />  
</system-properties>
```

- En el script de arranque del servidor standalone.sh o domain.sh.
 - Por línea de comandos, con el parámetro -D
-

```
./standalone.sh -Dpropiedad=true
```

2.3. Edición desde las herramientas de administración

Como ya hemos comentado, la forma más segura de modificar la configuración, es a través de las herramientas de configuración. Uno de los beneficios que reporta utilizar las herramientas es que WildFly crea automáticamente copias de seguridad de la configuración, por si nos equivocamos en algún cambio.

```
Terminal - expertojavajs@expertojavajs-VirtualBox: /usr/local/wildfly-8.1.0.Final/si - + x
Archivo Editar Ver Terminal Pestañas Ayuda
expertojavajs@expertojavajs-VirtualBox:/usr/local/wildfly-8.1.0.Final/standalone
/configuration/standalone_xml_history$ ls
20140623-175326071 20140713-120945821 current
20140623-190923569 20140717-224855048 snapshot
20140624-105851687 20140717-225254519 standalone.boot.xml
20140703-192617079 20140717-225816580 standalone.initial.xml
20140710-205548084 20140719-125402649 standalone.last.xml
20140713-114919863 20140719-142428268
expertojavajs@expertojavajs-VirtualBox:/usr/local/wildfly-8.1.0.Final/standalone
/configuration/standalone_xml_history$
```

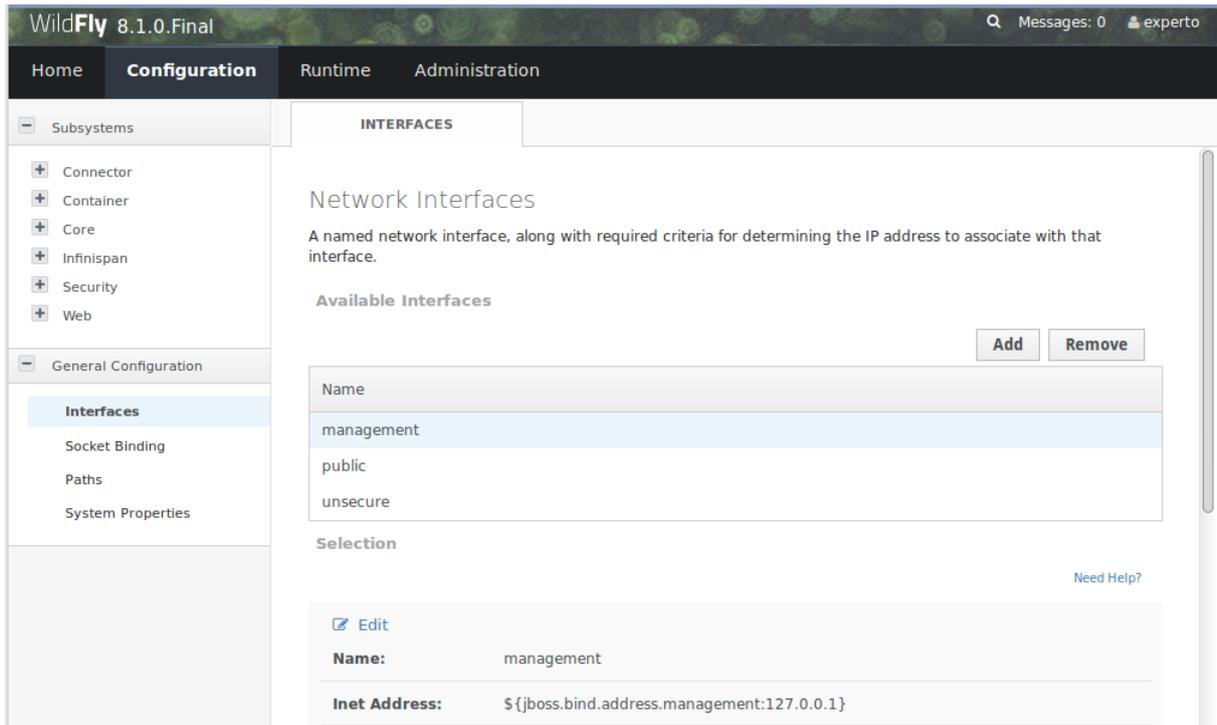
Sin embargo, en el caso concreto de JBoss, la edición de ficheros de configuración ha sido tradicionalmente la forma más efectiva de administración. Gracias a JBoss AS 7 y WildFly, las herramientas de administración comienzan a acercarse al nivel de otros productos.



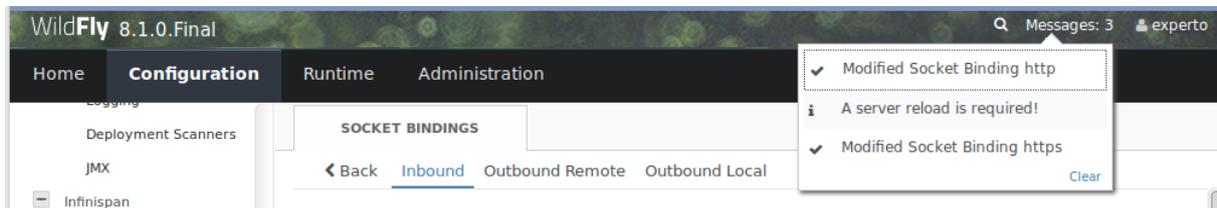
Descriptores de despliegue

Además de editar los ficheros de configuración del servidor, es posible codificar la mayor parte de los recursos en ficheros denominados descriptores de despliegue. Los descriptores de despliegue de WildFly complementan a los descriptores estándar de Java EE y permiten utilizar todas las características del servidor. Más información en: <https://docs.jboss.org/author/display/WFLY8/Deployment+Descriptors+used+In+WildFly>

La consola Web es una aplicación ligera que genera de forma automática vistas sobre la estructura de los ficheros de configuración, con lo que si conocemos la estructura del fichero, la representación Web nos resultará familiar:



Cuando modifiquemos algunas propiedades que requieran recargar la configuración, la aplicación nos avisará:



La recarga de la configuración la podemos realizar desde la pestaña *Runtime*, opción *Server*.

Desde CLI, los cambios se pueden realizar más rápidamente, gracias a la forma de navegar y la ayuda para autocompletar rutas:

```

Terminal - expertojavajs@expertojavajs-VirtualBox: /usr/local/wildfly-8.1.0.Final/b
Archivo  Editar  Ver  Terminal  Pestañas  Ayuda
ing=https:read-attribute(name=
port          fixed-port          bound-address      bound
interface     multicast-port     multicast-address
[standalone@localhost:9990 /] /socket-binding-group=standard-sockets/socket-bind
ing=https:read-attribute(name=
port          fixed-port          bound-address      bound
bound-port    client-mappings    name
interface     multicast-port     multicast-address
[standalone@localhost:9990 /] /socket-binding-group=standard-sockets/socket-bind
ing=https:read-attribute(name=port)
{
  "outcome" => "success",
  "result" => expression "${jboss.https.port:8443}"
}
[standalone@localhost:9990 /] /socket-binding-group=standard-sockets/socket-bind
ing=https:write-attribute(
name=  value=
[standalone@localhost:9990 /] /socket-binding-group=standard-sockets/socket-bind
ing=https:write-attribute(
name=  value=
[standalone@localhost:9990 /] /socket-binding-group=standard-sockets/socket-bind
ing=https:write-attribute(name=port,value=8444)
{"outcome" => "success"}
[standalone@localhost:9990 /] shutdown

```

Después de realizar cambios en la configuración, ésta se puede recargar mediante el comando *reload* o bien parando y arrancando de nuevo el servidor.

2.4. Configuración de recursos

La función más importante de un servidor de aplicaciones es la de proporcionar los recursos necesarios para que las aplicaciones puedan funcionar. A continuación repasaremos cómo configurar los recursos más habituales.

Configuración de Datasources

Para que las aplicaciones puedan tener acceso a base de datos previamente hay que definir un DataSource y un Pool de conexiones. El Pool de conexiones optimiza el acceso a base de datos creando un conjunto de conexiones antes de que las aplicaciones las lleguen a necesitar. El servidor de aplicaciones asigna estas conexiones dinámicamente a las aplicaciones y en cada asignación nos ahorramos el tiempo de conexión/desconexión a la base de datos.

Las propiedades más importantes que definen el comportamiento de un Pool de conexiones en Wildfly son:

Min Pool Size

El número mínimo de conexiones creadas que puede tener un pool.

Max Pool Size

El número máximo de conexiones creadas que puede tener un pool.

Strict Minimum

Si se activa, impide que el número de conexiones pueda bajar más allá del mínimo definido (por ejemplo si las sesiones se liberan por timeout).

Prefill Enabled

Si se activa, al iniciar un pool de conexiones se crean automáticamente el número de conexiones indicado como mínimo.

Flush Strategy

Especifica el modo en el que se liberan las conexiones de un pool ante un error.

Idle Timeout

Especifica el tiempo en minutos que se puede mantener una conexión sin uso, antes de ser liberada.

Para poder crear una conexión a base de datos en primer lugar hay que instalar el driver jdbc correspondiente. Esto se puede hacer de dos formas:

Instalación del driver como módulo

1. Crear una estructura de directorio dentro de JBOSS_HOME/modules. En el caso de MySQL debemos crear:

```
.....  
JBOSS_HOME/modules/system/layers/base/com/mysql/driver/main  
.....
```

2. Copiar el Jar dentro de la carpeta main
3. Crear un descriptor module.xml dentro de la carpeta main:

```
.....  
<module xmlns="urn:jboss:module:1.3" name="com.mysql.driver">  
  <resources>  
    <resource-root path="mysql-connector-java-5.1.33.jar" />  
  </resources>  
  <dependencies>  
    <module name="javax.api"/>  
    <module name="javax.transaction.api"/>  
  </dependencies>  
</module>  
.....
```

4. Declarar el driver en el fichero standalone.xml. Añadir al subsistema datasources el nuevo driver

```
.....  
<driver name="mysql" module="com.mysql.driver">  
  <driver-class>com.mysql.jdbc.Driver</driver-class>  
</driver>  
.....
```

Instalación del driver como despliegue

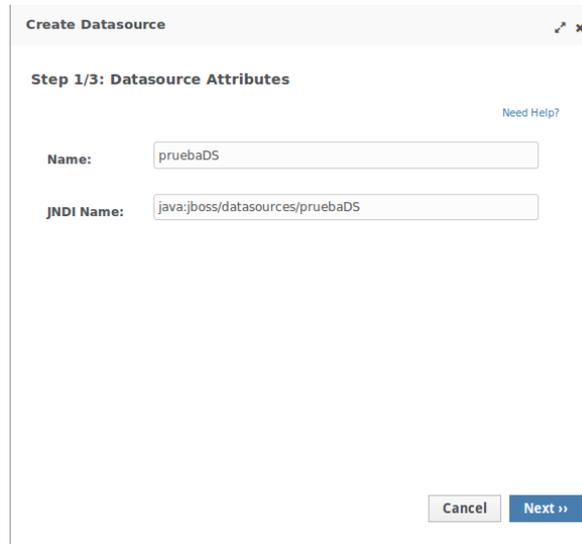
Si no tenemos permisos de administración, podemos simplemente copiar el fichero `.jar` con el driver de MySQL a la carpeta `deployments` del servidor. Al iniciar WildFly **debemos revisar** el nombre exacto del driver con el que se publica por si lo necesitamos referenciar en la creación de un `datasource`.

Descarga del driver NOTE: El jar lo podéis descargar desde la página de MySQL o bien desde aquí: <http://central.maven.org/maven2/mysql/mysql-connector-java/5.1.33/mysql-connector-java-5.1.33.jar>

Creación del Datasource y los pool

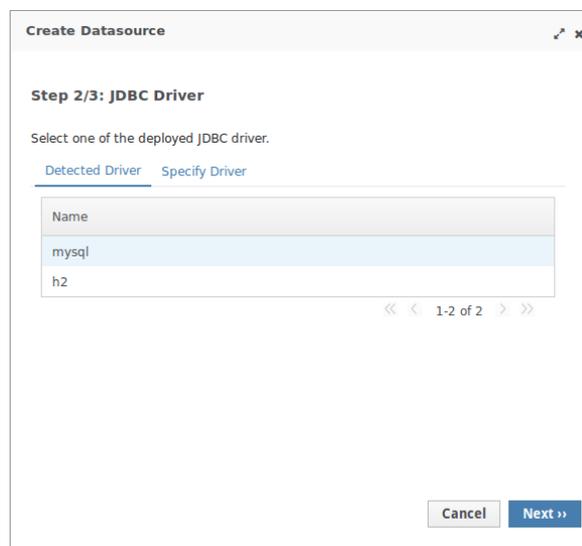
Nuevamente tenemos varias formas de configurar un Datasource, a continuación veremos los pasos para configurarlo desde la consola Web:

1. Definir el nombre del pool y el nombre JNDI del Datasource



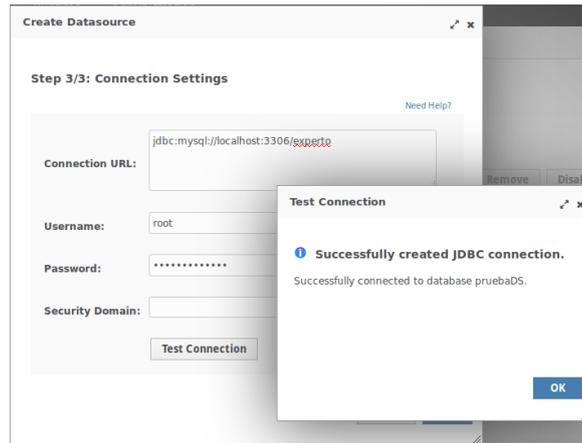
The screenshot shows a web-based configuration window titled "Create Datasource". It is at "Step 1/3: Datasource Attributes". There is a "Need Help?" link. The "Name" field contains "pruebaDS" and the "JNDI Name" field contains "java:boss/datasources/pruebaDS". At the bottom right, there are "Cancel" and "Next >>" buttons.

2. Seleccionar un driver de los instalados en el servidor:

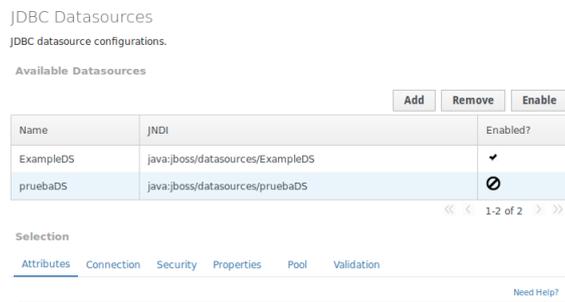


The screenshot shows the "Create Datasource" wizard at "Step 2/3: JDBC Driver". It instructs to "Select one of the deployed JDBC driver." There are two tabs: "Detected Driver" (active) and "Specify Driver". Under "Detected Driver", there is a list with "Name" as the header, containing "mysql" and "h2". The "mysql" entry is highlighted. Navigation arrows at the bottom of the list show "1-2 of 2". At the bottom right, there are "Cancel" and "Next >>" buttons.

3. Configurar la cadena de conexión y el usuario/password.



4. Activar el nuevo pool



¿Se puede desplegar un Datasource como un Deployment?

Si, se puede crear un fichero *-ds.xml con la configuración de uno o mas datasources. El fichero tiene la misma estructura que la del subsistema *datasources* del fichero *standalone.xml* y se puede copiar directamente a la carpeta *deployments*, o bien empaquetar dentro de una aplicación. Esta forma de crear datasources es recomendable en entornos de desarrollo, si bien, desde la consola web no es posible administrar este tipo de datasources.

```
<?xml version="1.0" encoding="UTF-8"?>
<datasources>
  <datasource jta="false" jndi-name="java:jboss/datasources/portalDS" pool-
name="portalDS" enabled="true" use-ccm="false">
    <connection-url>jdbc:mysql://localhost:3306/experto</
connection-url>
    <driver-class>com.mysql.jdbc.Driver</driver-class>
    <driver>mysql</driver>
    <security>
      <user-name>root</user-name>
      <password>expertojavajs</password>
    </security>
    <validation>
      <validate-on-match>>false</validate-on-match>
      <background-validation>>false</background-validation>
    </validation>
    <statement>
```

```

        <share-prepared-statements>false</share-prepared-
statements>
        </statement>
    </datasource>
</datasources>

```

Empaquetar Datasources en la aplicación:

Como hemos comentado el Datasource puede incluirse en las propias aplicaciones. Eso sí, la ubicación del fichero varía en función del tipo de aplicación:

Tipo Aplicación	Ubicación
Web App (.war)	WEB-INF
EJB (.jar)	META-INF
Enterprise (.ear)	META-INF (empaquetado principal)

Encriptación del password

WildFly tiene la particularidad de trabajar con passwords en plano en lo que respecta a Datasources. Si la configuración está centralizada en el fichero standalone.xml puede no ser un problema si está protegido para que no sea accesible. Sin embargo si adjuntamos un datasource a una aplicación, o lo copiamos en la carpeta *deployments* puede ser muy sencillo el averiguar el usuario y la clave de la base de datos. La forma de evitar esto es trabajar con claves encriptadas.

Los pasos son los siguientes:

Encriptar la password de la base de datos mediante la librería PicketBox

```

expertojavajs@expertojavajs-VirtualBox:/usr/local/
wildfly-8.2.1.Final/modules/system/layers/base/org/
picketbox/main$ java -classpath picketbox-4.0.21.Final.jar
  org.picketbox.datasource.security.SecureIdentityLoginModule expertojavajs
Encoded password: -46cea3eadea6ff81c9c59e773c1cfb95

```

Definir un *security-domain* asociado a las credenciales, editando el fichero *standalone.xml*: Para ello debemos definir un *security domain* en el servidor, editando el fichero *standalone.xml* y añadiendo la siguiente información dentro del subsistema `urn:jboss:domain:security`:

```

<security-domain name="ds-encrypted" cache-type="default">
  <authentication>
    <login-
module code="org.picketbox.datasource.security.SecureIdentityLoginModule"
  flag="required">
      <module-option name="username" value="root"/>
    <module-
option name="password" value="-46cea3eadea6ff81c9c59e773c1cfb95"/>
      <module-option name="managedConnectionFactoryName"
  value="jboss.jca:service=LocalTxCM,name=portalDS"/>

```

```

    </login-module>
  </authentication>
</security-domain>

```

Sustituir la información de usuario/password del datasource por la referencia al *security domain* :

```

<security>
  <security-domain>ds-encrypted</security-domain>
</security>

```

Configuración del contenedor de EJB's

Los EJB son componentes que implementan principalmente la lógica de negocio de una aplicación Java EE y el servidor de aplicaciones proporciona mecanismos para maximizar la capacidad de proceso de estos componentes.

Los EJB's se dividen en tres tipos:

Stateless Session Beans

Son objetos que implementan funciones de negocio pero no tienen un estado (valores en memoria).

Stateful Session Beans

Son objetos que implementan funciones de negocio pero además tienen un estado, que se conserva durante la vida de la sesión.

Message Driven Beans

Son objetos especializados en el proceso asíncrono de mensajes.

Los Session Beans se pueden instanciar desde cualquier componente de nuestra aplicación que se encuentre en el servidor (servlets, otros EJB's, etc.) pero también se pueden ejecutar desde procesos externos al servidor, mediante el protocolo *Remoting* (basado en RMI). Los objetos MDB son componentes especializados que sólo se ejecutan en respuesta a mensajes enviados asíncronamente a través de lo que se denomina tópico o cola de mensajes.

Pool de EJB's

Los EJB's son esencialmente POJO's a los que se les añade anotaciones específicas para integrarlos dentro del ciclo de vida del contenedor de EJB's. Así los objetos EJB se crean y se destruyen en función de la carga de trabajo. Al igual que ocurre con las conexiones a base de datos podemos definir el número de instancias de estos objetos de modo que podamos controlar los recursos destinados a ejecutar esta lógica de negocio. Esto es muy útil en momentos de picos de carga en los que un número excesivo de EJB's en ejecución pueden degradar notablemente el rendimiento global del servidor.

Este mecanismo es factible para los objetos que no tienen estado, como los Stateless Session Bean y los Message Driven Beans, pero los stateful son objetos equiparables a la sesión HTTP: contienen información de trabajo que no se puede copiar de una sesión a otra. En este caso hay mecanismos que agilizan la carga de la información, pero en ningún caso se puede crear un pool de objetos.



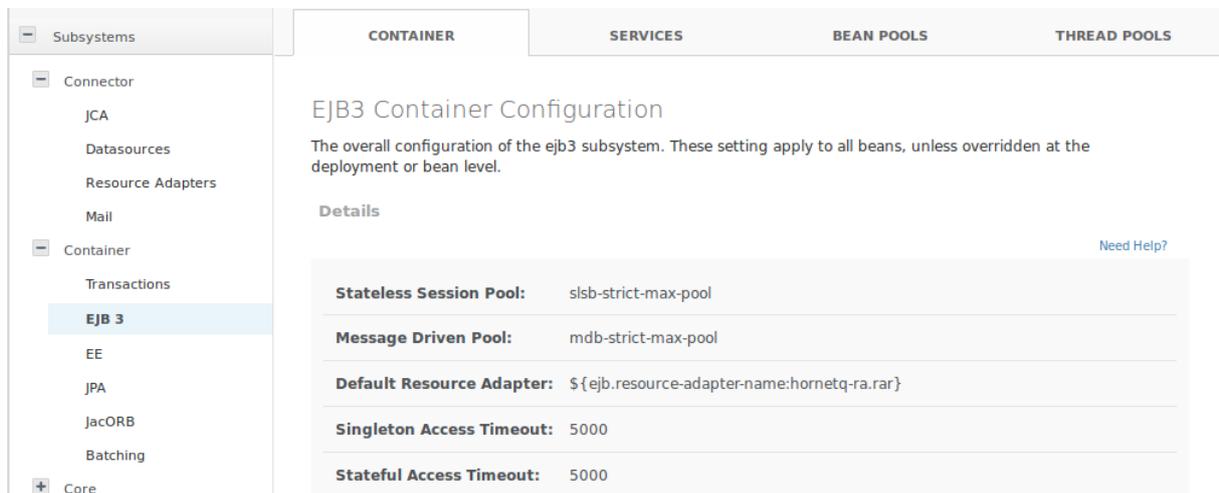
Pool de Objetos en WildFly

Antes de esta versión, existía una configuración de pool de objetos por defecto que aplicaba a todos los Stateless Session Bean. En WildFly

por defecto el pool de objetos está desactivado pues se entiende que hoy en día la mejora de rendimiento derivada del pooling no es tan evidente y el uso como contención de capacidad de proceso dependerá de cada aplicación, y por tanto es responsabilidad del desarrollador (o administrador) el parametrizar estos valores. En el caso de MDB's sigue estando activo un pool por defecto.

Para habilitar el pool por defecto *slsb-strict-max-pool* (límite de 20 instancias y timeout de acceso de 5 minutos) hay que editar el fichero de configuración añadiendo:

```
<subsystem xmlns="urn:jboss:domain:ejb3:2.0">
  <session-bean>
    <stateless> <!-- Añadir tag stateless-->
    <bean-instance-pool-ref pool-name="slsb-strict-max-pool"/>
  </stateless>
</subsystem>
```



Para definir un nuevo pool podemos hacerlo desde CLI o a mano editando el fichero de configuración.

- Desde CLI, el comando sería el siguiente:

```
[standalone@localhost:9990 /] /subsystem=ejb3/strict-max-bean-instance-pool=nuevo-pool:add(max-pool-size=5,timeout-unit=SECONDS,timeout=30)
```

Con este comando definiremos un nuevo pool de hasta 5 instancias

El resultado en el fichero standalone-full.xml es el siguiente:

```
<subsystem xmlns="urn:jboss:domain:ejb3:2.0">
  ....
  <pools>
    <bean-instance-pools>
      <strict-max-pool name="nuevo-pool" max-pool-size="5" instance-acquisition-timeout="30" instance-acquisition-timeout-unit="SECONDS"/>
    </bean-instance-pools>
  </pools>
  ....
</subsystem>
```

Para asociar un pool a un EJB concreto, basta añadir una anotación en la clase correspondiente indicando el nombre del pool:

```
@Stateless
@Pool (value="ranking-pool")
public class Ranking implements RankingLocal {
```

Descriptores de despliegue

Java Enterprise proporciona dos métodos para especificar aspectos de la configuración de una aplicación:

- Descriptores de despliegue (Ficheros XML).
- Anotaciones en código.

Sin embargo hay propiedades específicas de Wildfly (comportamiento de los EJB's del servidor Web Undertow, etc.) que no quedan cubiertas por el estándar Java EE y que pueden ser definidas desde la aplicación mediante ficheros de configuración propietarios. En la sección de referencias se encuentra un enlace a la lista de descriptores de WildFly, que a la fecha de este documento no estaba completa (se podría utilizar los propios de JBoss 7.1).

Class Loading

En este apartado trataremos brevemente el orden que sigue el servidor a la hora de localizar las clases que componen una aplicación. En la sesión anterior comentamos los distintos tipos de aplicaciones java y la forma de acceder a componentes compartidos. JBoss proporciona un mecanismo adicional: el uso de *módulos*.

Los módulos se dividen a su vez en

Módulos estáticos

se integran en las carpetas de módulos del servidor y pueden ser definidos como accesibles por todos los despliegues del servidor.

Módulos dinámicos

Se copian a la carpeta deployments y pueden ser reemplazados en caliente.

Si recordamos el caso del driver de MySQL veremos que hemos propuesto dos instalaciones, una como módulo estático y otra como módulo dinámico. Por defecto si añadimos un módulo dinámico, el nombre interno que tendrá el módulo en WildFly será:

```
deployment.[nombre del fichero]
```

La forma de especificar que una aplicación necesita acceder a recursos de un módulo, (estático o dinámico) es indicarlo en el fichero *MANIFEST.MF*

```
Dependencies: deployment.WebExample1.war
```

Por último, comentar el orden que sigue WildFly para resolver la ubicación de una clase Java. Ordenando de mayor prioridad a menor:

1. Dependencias del sistema: Son dependencias que el contenedor añade de forma automática a la aplicación, como por ejemplo el API de Java EE.
2. Dependencias de usuario: Son las que el desarrollador define a través del MANIFEST.MF, el Classpath para ficheros JAR o a través del fichero *jboss-deployment-structure.xml*.
3. Recursos locales. Son las clases que están ubicadas dentro del propio empaquetado de la aplicación, por ejemplo *WEB-INF/classes* o *WEB-INF/lib* de una aplicación WAR.
4. Dependencias compartidas. Son las dependencias compartidas entre varios componentes de una aplicación. Típicamente las librerías compartidas dentro de una aplicación EAR.

Rendimiento

Hablar de rendimiento en un servidor de aplicaciones nos permitiría llenar más de una sesión entera del curso, si bien, lo más importante es dimensionar los recursos del servidor (memoria, pool de objetos, conexiones a base de datos) en función de la carga de trabajo que va a soportar. Dado que gran parte de la programación que se verá en el curso gira en torno a JavaScript, comentar que hay una forma muy sencilla de agilizar los tiempos de carga de las aplicaciones y consiste en habilitar la compresión GZIP en las comunicaciones con el servidor. Descargar con compresión código JavaScript o texto estático en HTML puede reducir el tiempo de descarga hasta un 90%, a costa de un mayor consumo de CPU en el servidor.

La forma de habilitarlo en WildFly consiste en definir un filtro y según el tipo de recurso aplicar compresión o no (es contraproducente comprimir recursos estáticos ya comprimidos como imágenes o documentos PDF).

```
<subsystem xmlns="urn:jboss:domain:undertow:1.0">
  <buffer-caches>
    <buffer-cache name="default" buffer-size="1024" buffers-per-
region="1024" max-regions="10"/>
  </buffer-caches>
  <server name="default-server">
    <http-listener name="default" socket-binding="http"/>
    <host name="default-host" alias="localhost">
      <location name="/" handler="welcome-content" />
      <filter-ref name="gzipFilter" predicate="path-suffix['.css'] or
path-suffix['.js']" />
      <filter-ref name="server-header"/>
      <filter-ref name="x-powered-by-header"/>
    </host>
  </server>
  <servlet-container name="default" default-buffer-cache="default" stack-
trace-on-error="local-only">
    <jsp-config/>
  </servlet-container>
  <handlers>
    <file name="welcome-content" path="${jboss.home.dir}/welcome-
content" directory-listing="true"/>
  </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"/>
    <gzip name="gzipFilter"/>
  </filters>
</subsystem>
```

</subsystem>

Se añade la referencia al filtro *gzipFilter* en la lista de filtros y en el tag *<filter-ref name>* se define la expresión regular que determina si un elemento se debe comprimir o no.

2.5. Referencias

- WildFly Administration Guide <https://docs.jboss.org/author/display/WFLY8/Admin+Guide>
- Quickstart JMS <http://www.mastertheboss.com/jboss-jms/how-to-code-a-remote-jms-client-for-wildfly-8>
- Descriptores de despliegue estándar Java EE 7 <https://wikis.oracle.com/display/GlassFish/Java+EE+7+Maven+Coordinates>
- Descriptores de despliegue específicos de WildFly <https://docs.jboss.org/author/display/WFLY8/Deployment+Descriptors+used+In+WildFly>
- Configuración de los Pool de EJB en JBoss 7 <http://www.javacodegeeks.com/2011/11/jboss-as-7-ejb3-pools-configuration.html>
- Información sobre el ClassLoading en JBoss 7 <https://docs.jboss.org/author/display/AS71/Class+Loading+in+AS7>
- Best Practices for speeding up your Web Site <https://developer.yahoo.com/performance/rules.html>
- GZip Encoding in WildFly http://rumianom.pl/rumianom/entry/gzip_content_encoding_in_wildfly

2.6. Ejercicios de Administración de recursos en Wildfly



JMS y WildFly

Para que se pueda acceder a determinados recursos como las connection factory o colas del servidor desde un proceso externo, hay que utilizar un nombre JNDI adicional que comience por `java:jboss/exported`. El cliente `jboss-client` de WildFly 8.x no implementa JMS 2.0, si no que se corresponde con la especificación Java EE 6.0 La clase genérica `Destination` no está definida.

Ahora que ya tenemos una visión general sobre la configuración de recursos en WildFly, vamos a realizar una serie de ejercicios para poner en práctica lo que hemos aprendido.

Librería compartida de Logs (0.6 puntos)

Vuestro primer reto será crear una sencilla librería de log, que registre cualquier evento en una tabla de base de datos. Un evento tendrá los siguientes atributos: timestamp, un mensaje y un tipo: I:informativo, D:debug W:aviso E:error.

Debéis realizar las siguientes tareas:

- Iniciar WildFly en modo standalone en modo Full Profile Java EE (`standalone-full.xml`).
- Desplegar la librería `_apli-loglibrary-` como librería compartida, que contiene las clases necesarias para implementar la interfaz de log.
- Definir el datasource `java:jboss/datasources/appLogger` en un descriptor de despliegue con el password encriptado, que se debe incluir en la propia librería.

- Desplegar una sencilla aplicación Web con un único Servlet que registre en base de datos cada llamada que reciba. Debe enlazar externamente con la librería de logs.

Se os proporciona un script *eurovision.sql* con la tabla de logs y la base de datos para el siguiente ejercicio. También contaréis con los tres proyectos Maven que componen esta aplicación:

- Proyecto "Java Application" de nombre *apli-logLibrary* con la interfaz de Log predefinida.
- Proyecto "Web Application", *apli-AppLogger*, con el servlet de prueba: se invocará mediante la llamada a <http://localhost:8080/App/Test>
- Proyecto "Enterprise Application" *apli-Enterprise Logger* que incluirá únicamente al proyecto Web.

Debéis desplegar tanto la librería compartida como el EAR y comprobar que se registran en la tabla las llamadas al servlet. Los entregables serán los fuentes de todos los proyectos.



Pistas

Si queréis probar la librería sin desplegarla en WildFly debéis utilizar el denominado cliente completo de WildFly. Es un jar que se utiliza en clientes pesados que incluye todas las clases de WildFly necesarias para trabajar en remoto.

Para dar de alta el cliente de WildFly como una dependencia de Maven hay que hacer lo siguiente:

1. Acceder a la carpeta */usr/local/wildfly-8.2.1.Final/bin/client*.
2. Ejecutar:

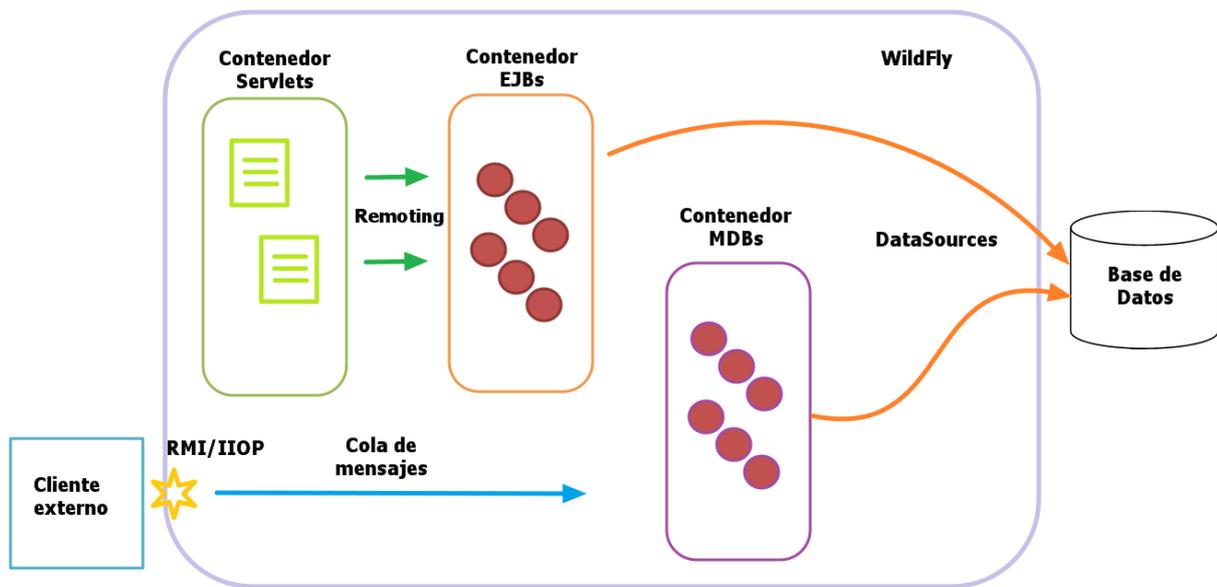
```
mvn install:install-file -Dfile=jboss-client.jar \  
-DgroupId=org.wildfly \  
-DartifactId=jboss-client \  
-Dversion=8.2.1 \  
-Dpackaging=jar \  
-DgeneratePom=true
```

Camino a Eurovisión (1 punto)

Tenemos la tarea de desplegar un sistema de votaciones que se utilizará para determinar la próxima canción candidata española a Eurovisión. Esta aplicación presenta una estructura de proyectos típica de las aplicaciones Java Enterprise de hace algunos años. Se trata de una aplicación EAR con distintos subproyectos para cada uno de los módulos que contiene y un pom principal que compila toda la solución. Esta basada en Java EE 5 aunque tiene referencias a las librerías actuales. En resumen, algo muy parecido a lo que os podéis encontrar en el mundo laboral. Los módulos son los siguientes:

- Una librería con un EJB que implementa la lógica de negocio.
- Una aplicación Web que implementa la presentación.
- Una aplicación empresarial EAR que engloba a todo el proyecto.
- Un cliente Java EE externo.

Gráficamente, la arquitectura de la aplicación es la siguiente:



Se utiliza una aplicación externa para simular una pasarela que recoge los votos emitidos mediante mensajes de móvil. La información que os interesará de dichos mensajes es el número de teléfono origen y el número de la canción votada. El formato del mensaje será el teléfono, de longitud variable, el símbolo "#" como separador y por último la palabra *EURO* más un número del 01 al 12, ejemplo: *654736534#EURO04*.

Partiréis de los proyectos Maven ya creados y un script SQL para la creación de la base de datos *eurovision*. Aunque no se estudia en detalle en esta sesión, dentro de la aplicación tenemos un ejemplo de procesamiento asíncrono mediante colas de mensajes y objetos MDB's. Para que esta parte funcione correctamente hay que crear un usuario de aplicación denominado **jmsuser**. Para que este nuevo usuario tenga permisos de lectura/escritura hay que asociarlo al grupo **guest**.

```

expertojavajs@expertojavajs-VirtualBox:/usr/local/wildfly-8.1.0.Final/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): b
Enter the details of the new user to add.
Using realm 'ApplicationRealm' as discovered from the existing property files.
Username : jmsuser
Password recommendations are listed below. To modify these restrictions edit the add-user.properties configuration file.
- 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)
- The password should be different from the username
Password :
JBAS015267: Password must have at least 1 non-alphanumeric symbol.
Are you sure you want to use the password entered yes/no? yes
Re-enter Password :
What groups do you want this user to belong to? (Please enter a comma separated list, or leave blank for none)[ ]: guest
About to add user 'jmsuser' for realm 'ApplicationRealm'
Is this correct yes/no? y
Added user 'jmsuser' to file '/usr/local/wildfly-8.1.0.Final/standalone/configuration/application-users.properties'
Added user 'jmsuser' to file '/usr/local/wildfly-8.1.0.Final/domain/configuration/application-users.properties'
Added user 'jmsuser' with groups guest to file '/usr/local/wildfly-8.1.0.Final/standalone/configuration/application-roles.properties'
Added user 'jmsuser' with groups guest to file '/usr/local/wildfly-8.1.0.Final/domain/configuration/application-roles.properties'
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? n
    
```

Para completar el ejercicio teneis que cubrir los siguientes objetivos:

1. Configurar la aplicación EAR para que utilice la librería compartida de logs del ejercicio anterior.

2. Ajustar la configuración del EJB Ranking que implementa la lógica de negocio añadiéndole un **límite máximo** de 5 objetos simultáneos para el EJB. Como esta configuración es específica de WildFly hay que añadir la siguiente dependencia al proyecto del EJB:

```
<dependency>
  <groupId>org.wildfly</groupId>
  <artifactId>wildfly-ejb3</artifactId>
  <version>8.2.1.Final</version>
  <scope>provided</scope>
</dependency>
```

3. La aplicación contiene un MDB que recibe los votos y los almacena en base de datos. Hay que revisar cómo está implementado y crear la conexión a base de datos que necesita desde la consola de WildFly.
4. Configurar el usuario JMS que utilizará el cliente Java *apli-EnvioSMS*. Se trata de una aplicación Java EE que se ejecuta fuera del servidor de aplicaciones pero utiliza los servicios que ofrece, concretamente una cola JMS, denominada *jms/entradaMensajes* a donde enviará los votos del
5. Desplegar la aplicación y acceder a la consulta de votaciones para ver las posibles canciones a las que se puede votar.
6. Utilizar la aplicación cliente para enviar varios votos y asegurarse de que se contabilizan correctamente.

Los entregables del ejercicio serán:

- El fichero de configuración standalone-full.xml
- El código fuente de todos los proyectos solicitados, *apli-EnvioSMS* y *Apli-CaminoAEurovision*.



Pistas

Para poder compilar la aplicación externa Java EE necesitaréis construir el cliente completo de WildFly, tal y como se explicaba en el ejercicio anterior. Estad muy pendientes del log de WildFly, que es donde se os dará una pista más clara de lo que ocurre en caso de problemas.

3. Seguridad declarativa en WildFly (JAAS)

Desde el punto de vista del programador, no hay grandes diferencias a la hora de desarrollar y desplegar una aplicación en WildFly, GlassFish u otro servidor de aplicaciones, pero es desde la perspectiva del administrador desde donde cada producto muestra una forma diferente de operar. Desde este punto de vista hemos querido cubrir los aspectos más generales del uso de WildFly. A partir de ahora nos centraremos en la seguridad.

En esta sesión hablaremos sobre seguridad en declarativa en WildFly y en la siguiente hablaremos de seguridad en la capa de transporte

3.1. Implementación de la seguridad declarativa en WildFly

A estas alturas ya tenemos una idea general sobre seguridad declarativa, pero como repaso vamos a revisar una serie de conceptos que enlazan lo que ya hemos visto con algunos elementos nuevos:

Usuarios, grupos y roles

Son los elementos básicos a la hora de definir la seguridad de una aplicación Web y que ya hemos utilizado a lo largo del curso. En el caso concreto de WildFly, grupo y rol se tratan como un mismo concepto, como veremos más adelante.

Políticas de seguridad

Son asociaciones entre recursos de WildFly y usuarios o roles. Es decir nos permiten definir exactamente quién puede acceder a un determinado recurso y qué operaciones puede hacer con él.

Dominios de seguridad (Security Domains)

Son las configuraciones de autenticación, autorización, mapeo de seguridad y auditoría propias del servidor de aplicaciones. Implementan la especificación JAAS (Java Authentication and Authorization Service) de seguridad declarativa.

Security Realms

Los Realm son los almacenes de usuarios, password y roles de las interfaces de administración del servidor (puntos de entrada) tanto desde el exterior como internamente en el propio servidor. Son componentes sobre los que se sustentan los security domain. De inicio WildFly cuenta con dos Security Realm:

- **ManagementRealm**: que sirve para securizar el acceso a las herramientas de administración (Autenticación)
- **ApplicationRealm**: que se utiliza para controlar el acceso a las aplicaciones desplegadas (Autenticación y Autorización).

La configuración del Realm de administración es la siguiente:

```
<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>
```

El Realm distingue entre accesos desde la máquina local y accesos remotos. Para los primeros no se requiere password, por lo que la herramienta CLI ejecutada en local se conecta directamente, como ya vimos en la primera sesión. En el caso de utilizar CLI contra una instancia remota o acceder a través de la Consola Web sí que se nos pedirá usuario/password.

Los ficheros indicados *mgmt-users.properties* y *mgmt-groups.properties* contendrán los usuarios y grupos a los que pertenecen.



El concepto de grupo sí existe para el ManagementRealm ya que se pueden definir grupos de usuarios y posteriormente asociarlos con roles de administración RBAC (que veremos al final de la sesión).

El Realm de aplicaciones tiene una configuración muy parecida:

```
<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>
```

También admite un usuario local de tipo invitado *\$local* y codifica la información de usuarios en ficheros locales. En este caso sí está habilitado el mapeo entre usuarios y roles y se define un fichero específico para ello.

Los Security Realm se pueden utilizar tanto en conexiones provenientes del exterior como en conexiones internas. El ejemplo típico de esto último es el subsistema *Remoting* utilizado para acceder al contenedor de EJB's, que por defecto está vinculado al Realm de aplicaciones:

```
<subsystem xmlns="urn:jboss:domain:remoting:2.0">
  <endpoint worker="default"/>
  <http-connector name="http-remoting-connector" connector-
ref="default" security-realm="ApplicationRealm"/>
</subsystem>
```

Security Domain y Login Modules

Los *Security Domain* definen las políticas de autenticación y autorización del servidor de aplicaciones. Un security domain configura uno o varios *login modules* que implementan estas políticas. La configuración por defecto es la siguiente:

```
<subsystem xmlns="urn:jboss:domain:security:1.2">
  <security-domains>
    <security-domain name="other" cache-type="default">
```

```

<authentication>
  <login-module code="Remoting" flag="optional">
    <module-option name="password-
stacking" value="useFirstPass"/>
  </login-module>
  <login-module code="RealmDirect" flag="required">
    <module-option name="password-
stacking" value="useFirstPass"/>
  </login-module>
</authentication>
</security-domain>
<security-domain name="jboss-web-policy" cache-
type="default">
  <authorization>
    <policy-module code="Delegating" flag="required"/>
  </authorization>
</security-domain>
<security-domain name="jboss-ejb-policy" cache-
type="default">
  <authorization>
    <policy-module code="Delegating" flag="required"/>
  </authorization>
</security-domain>
</security-domains>
</subsystem>

```

Además del Dominio *other* tenemos los dominios *jboss-ejb-policy* y *jboss-web-policy* que componen los módulos de autorización predefinidos, pero el security domain que nos interesa es el denominado *other*.

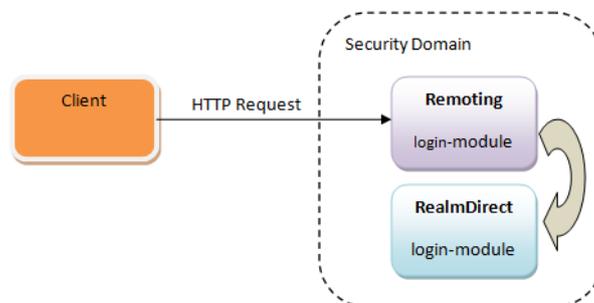
Other es el security domain por defecto en la instalación, preparado para trabajar con el *Application Realm* y en él podemos identificar dos login modules distintos:

Remoting

Utilizado internamente cuando las peticiones de autenticación se reciben a través del protocolo Remoting (habitualmente llamadas a un EJB).

RealmDirect

Utilizado en el resto de peticiones (por ejemplo, al acceder a una aplicación Web). Este módulo tiene la particularidad delegar en un Realm la tarea de autenticación, que por defecto será el *ApplicationRealm*.



El Realm que utilizará RealmDirect se puede especificar mediante el module-option *realm*.

Por defecto, los login modules se ejecutarán en el orden en el que se hayan definido. Adicionalmente existe un parámetro denominado *flag* en cada login module, que nos permitirá definir cómo tratar cada resultado parcial.

Puede tener los siguientes valores:

Required

El usuario debe validarse con éxito necesariamente. Independientemente del resultado intentará validarse con el resto de login modules.

Requisite

El usuario debe validarse con éxito necesariamente. Una vez validado, intentará validarse con el resto de login modules. Las siguientes validaciones podrán fallar pero el usuario seguirá autenticado (excepto si tienen el atributo de requerido). Si falla la validación, termina el proceso de autenticación.

Sufficient

El usuario intentará validarse, si falla lo intentará con el resto de login modules, pero si es autenticado, la validación termina en ese punto.

Optional

El usuario intentará validarse, aunque no necesariamente tiene porqué tener éxito. Si todos los login modules están configurados como optional, necesariamente tendrá que validarse con al menos uno de ellos.

El módulo RealmDirect tiene una única opción, el *password-stacking* que debe ser utilizado con el valor *useFirstPass* si se utilizan varios login modules.

En esta sesión trabajaremos con tres login modules:

- **RealmDirect** almacena usuarios y roles en ficheros. Estos ficheros son los definidos dentro del ApplicationRealm.
- **Database** almacena usuarios y roles en una base de datos relacional.
- **LDAP** almacena usuarios y roles en un servidor LDAP como OpenLDAP.



La lista completa de login modules la podéis encontrar en esta página: <https://docs.jboss.org/author/display/AS71/Security+subsystem+configuration>

Activación de la auditoria de seguridad

En la fase de construcción resulta muy útil habilitar la auditoría de seguridad antes de comenzar a trabajar con seguridad declarativa. El log normal de Wildfly no suele proporcionar información significativa cuando intentamos acceder a una aplicación securizada y por la causa que sea el servidor no considera válidas nuestras credenciales.

La forma de hacerlo en WildFly es un tanto enrevesada pues hay que añadir la siguiente información al subsistema de **logging**:

```
<subsystem xmlns="urn:jboss:domain:logging:2.0">
  ...
  <periodic-rotating-file-handler name="AUDIT" autoflush="true">
    <level name="TRACE"/>
    <formatter>
```

```

        <pattern-formatter pattern="%d{HH:mm:ss,SSS} %-5p [%c]
(%t) %s%E%n"/>
    </formatter>
    <file relative-to="jboss.server.log.dir" path="audit.log"/
>
    <suffix value=".yyyy-MM-dd"/>
    <append value="true"/>
</periodic-rotating-file-handler>
<logger category="org.jboss.security">
    <level name="TRACE"/>
    <handlers>
        <handler name="AUDIT"/>
    </handlers>
</logger>
...

```

Con este cambio crearemos un nuevo fichero de log, denominado *audit.log* en la carpeta de logs del servidor con toda la información de la negociación de la seguridad declarativa.

3.2. Administración de usuarios y roless con RealmDirect

La forma de dar de alta o editar usuarios ya la conocemos y es a través del script *add-user.sh* especificando el realm de Aplicaciones. Desde este script se permite asociar un usuario a uno o varios grupos (roles):

```

what groups do you want this user to belong to? (Please enter a comma
separated list, or leave blank for none)[ ]: usuarios

```



add-user.sh da de alta los usuarios indistintamente en la modalidad de standalone y domain.

Los usuarios se almacenarán con el siguiente formato:

```

username=HEX(MD5(username ':' realm ':' password))

```

Ejemplo:

```

experto=579f4faae456fd4b742f89e35fa935e2
admin=c22052286cd5d72239a90fe193737253

```

Los roles se especificarán como texto plano en el fichero *application-roles.properties*:

```

experto=Users
admin=Administrators

```

Realm Personalizado

Como comentamos, RealmDirect utiliza el ApplicationRealm como Realm por defecto a la hora de autenticar usuarios en ficheros. Es perfectamente posible crear un nuevo Realm con una

ubicación de ficheros distinta y especificar dicho Realm como el Realm a utilizar por Realm Direct en nuestro security domain. De esta forma podríamos tener separados los usuarios de nuestra aplicación, del común del servidor. La limitación es que el script add-user no da facilidades para trabajar con distintos Realm por lo que es necesario dar de alta manualmente los usuarios (encriptación(hash) de password incluida).

Configuración de un login module basado en BD

Vamos a agregar al dominio *other* el login module *Database*, apoyado en una base de datos externa siguiendo los siguientes pasos:

1. Crear una base de datos MySQL , denominada "seguridad" con las siguientes tablas

```

CREATE TABLE USERS (
    principal_id VARCHAR(64) primary key,
    password VARCHAR(64));
;

CREATE TABLE ROLES (
    principal_id VARCHAR(64) primary key,
    user_role VARCHAR(64),
;

ALTER TABLE ROLES
    ADD CONSTRAINT FK1_PRINCIPLES
    FOREIGN KEY ( principal_id)
    REFERENCES USERS (principal_id)
    ON DELETE CASCADE
;

```

2. Definir un usuario específico para la conexión a la BD seguridad: *usuSeg*, clave *usuSeg*. Esto lo podéis hacer desde la herramienta MySQL Workbench que tenéis en Linux.
3. Crear un origen de datos en WildFly, de nombre *java:jboss/datasources/seguridad*, que acceda a la BD que acabamos de crear.
4. Crear un nuevo security domain que utilice el login module *Database*:

```

# Crear un nuevo security domain con seguridad basada en base de datos
batch

connect

# Configure the security domain
/subsystem=security/security-domain=seguridad/:add(cache-type=default)
/subsystem=security/security-domain=seguridad/
authentication=classic:add(login-modules=[{"code"=>"Database",
"flag"=>"required", "module-options"=>[{"dsJndiName"=>"java:jboss/
datasources/seguridad"}, {"principalsQuery"=>"SELECT PASSWORD FROM USERS
WHERE principal_id = ?"}, {"rolesQuery"=>"SELECT user_role, 'Roles' FROM
ROLES where principal_id = ?"}]})

```

Run the batch commands

```
run-batch
```

```
# Reload the server configuration
:reload
```



Grupos

WildFly no introduce el concepto de grupos de usuarios pero la implementación del Login Module Database es muy flexible, por lo que se puede modelar una tabla de grupos, asociar usuarios a grupos y permitir asociar roles a los grupos. Después de esto, bastaría con modificar las queries de recuperación de roles.

Protección de las password

Nuevamente WildFly no nos da un sistema directo para proteger las contraseñas pero si nos permite especificar un mecanismo de encriptación/hash y encargarnos nosotros mismos de su gestión. Concretamente hay que añadir las siguientes opciones al login module:

```
<module-option name="password-stacking" value="useFirstPass"/>
<module-option name="hashAlgorithm" value="MD5"/>
<module-option name="hashEncoding" value="hex"/>
```

Con estas opciones se especifica el algoritmo para calcular el hash a partir de la contraseña, así como la forma de convertir la cadena binaria resultante en una representación en formato de texto (base64).

En la siguiente dirección tenéis una utilidad para calcular estos hash, es importante que el hash generado se introduzca en minúsculas porque la validación es case sensitive: <http://md5-hash-online.waraxe.us/>

Aplicación Web de Ejemplo

Vamos a construir una sencilla aplicación Web que aproveche la configuración de seguridad que hemos preparado hasta ahora. Esta aplicación Web contará con un único servlet que mostrará la lista de usuarios de la BD seguridad.

Para ello definiremos un servlet y lo protegeremos de tal modo que sólo los usuarios y administradores puedan acceder a él. Utilizaremos la autenticación Basic para simplificar.

Lo construiremos en los siguientes pasos:

1. Configurar el nuevo dominio "seguridad" en IntelliJ. Crear un nuevo proyecto Web, mediante Maven.
2. Crear un servlet que responda a la ruta /listado, y que devuelva la consulta:

```
@WebServlet("/protegido/ConsultaServlet")
@WebServletSecurity(@HttpConstraint(rolesAllowed =
    { "rol_usuario", "rol_administrador" }))
public class ConsultaServlet extends HttpServlet {

    @Resource(mappedName = "java:jboss/datasources/seguridad")
```

```

DataSource segDS;

protected void processRequest(HttpServletRequest request,
                               HttpServletResponse response)
    throws ServletException, IOException {

    String query = "SELECT * FROM USERS";
    Connection conn = null;
    Statement stmt;

    response.setContentType("text/html;charset=UTF-8");

    PrintWriter out = response.getWriter();
    try {

        conn = segDS.getConnection();
        stmt = conn.createStatement();

        out.println("<html>");
        out.println("<head>");
        out.println("<title>Servlet listado</title>");
        out.println("</head>");
        out.println("<body>");

        out.println("Lista de usuarios:<BR/>");

        ResultSet rs = stmt.executeQuery(query);
        while(rs.next())
        {

            out.println(rs.getString("principal_id")+"/"+rs.getString("password")+ "<br/>");
        }

        out.println("</body>");
        out.println("</html>");
    } catch (Exception e) {
        out.println(e.getMessage());
    }

    finally {
        try {
            out.close();
            if(conn!=null)
                conn.close();
        } catch (SQLException ex) {

            Logger.getLogger(ConsultaServlet.class.getName()).log(Level.SEVERE,
                null, ex);
        }
    }
}

```

-
3. Configurar los descriptores de despliegue. El archivo web.xml tendrá un aspecto parecido a este:
-

```
<session-config>
```

```

<session-timeout>30</session-timeout>
</session-config>
<welcome-file-list>
  <welcome-file>index.jsp</welcome-file>
</welcome-file-list>
<login-config>
  <auth-method>BASIC</auth-method>
  <realm-name>Mi dominio de seguridad</realm-name>
</login-config>
<security-role>
  <description />
  <role-name>rol_usuario</role-name>
</security-role>
<security-role>
  <description />
  <role-name>rol_administrador</role-name>
</security-role>
</web-app>

```



WildFly en su versión 8.10 da algunos problemas a la hora de combinar diferentes tipos de restricciones de integridad sobre un mismo recurso. Como recomendación intentad siempre que sea posible, no declarar más de una restricción sobre un mismo recurso de la aplicación.

4. Añadiremos el descriptor específico de WildFly *jboss-web.xml*, en la carpeta WEB-INF:

```

<jboss-web xmlns="http://www.jboss.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.jboss.com/xml/ns/javaee http://
www.jboss.org/schema/jbossas/jboss-web_8_0.xsd"
  version="8.0">

  <context-root>/intranet</context-root>
  <security-domain>
    seguridad
  </security-domain>
</jboss-web>

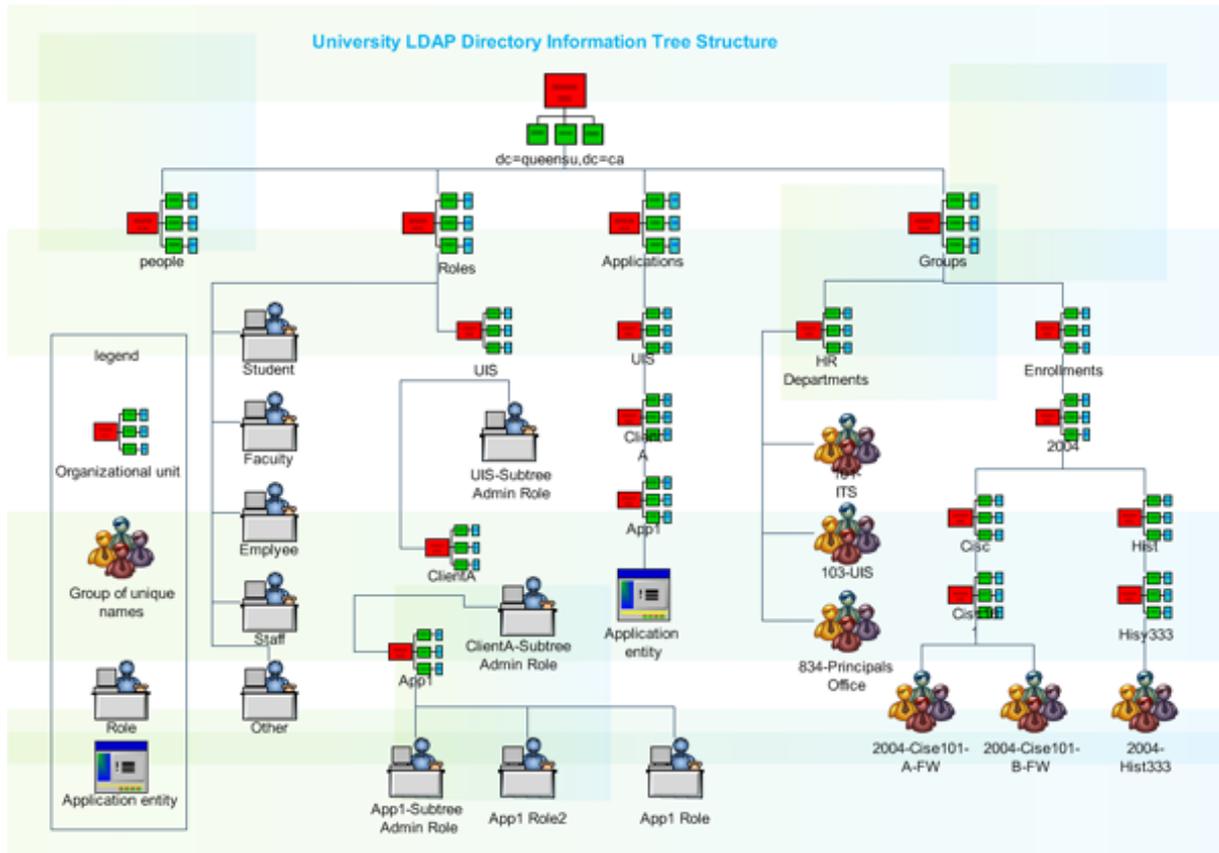
```

En el estamos indicando tanto el contexto raíz de la aplicación como el security domain contra el que vamos a autenticarnos.

Integración con un servidor LDAP Externo

LDAP ("Lightweight Directory Acces Protocol") es un protocolo de tipo cliente-servidor para acceder a un servicio de directorio.

Cada entrada en el directorio LDAP describe un objeto (por ejemplo: una persona, un recurso de red, una organización) y tiene un único identificador llamado Nombre Distinguido (DN, Distinguished Name). La entrada consiste de una colección de atributos (por ejemplo una persona podría tener apellido, organización, e-mail). Para encontrar las entradas hay que navegar a través del Árbol de Información de Directorio (DIT, Directory Information Tree). En la raíz del árbol se encuentra "el mundo", el cual esta subdividido en el siguiente nivel en países, y en el siguiente en organizaciones. Dentro de las organizaciones se almacenan información de gente, recursos, etc.



Es habitual en empresas de cierto tamaño el tener un servidor donde se almacena la información corporativa de directorio. Los servidores de aplicaciones suelen integrar componentes que permiten acceder a la información de usuarios y roles que reside en dicho servidor LDAP.

En nuestra máquina virtual, podemos simular este directorio corporativo instalando OpenLDAP y cargando un directorio de pruebas. Estos serán los pasos a seguir:

1. Instalar el servidor SLDAPd ejecutando los comandos:

```
sudo apt-get update
sudo apt-get -y install slapd ldap-utils
```

2. Configuración básica del directorio:

```
sudo dpkg-reconfigure slapd
```

Preguntas	Respuestas
Omit OpenLDAP server configuration	No
DNS domain name	ua.es
Organization name	jtech
Administrator password	expertojavajs
Database backend to use	HDB

Preguntas	Respuestas
Do you want the database to be purged	yes
Move old database	yes
Allow LDAPv2 protocol	no

3. Configurar el nuevo dominio LDAP mediante un script en lenguaje LDIF:

```
#grupo de usuarios
dn: ou=users,dc=ua,dc=es
objectClass: organizationalUnit
ou: users

#grupo de roles
dn: ou=roles,dc=ua,dc=es
objectClass: organizationalUnit
ou: roles

#alumno1
dn: uid=alu1,ou=users,dc=ua,dc=es
objectclass: top
objectclass: uidObject
objectClass: inetOrgPerson
objectclass: person
uid: alu1
cn: Cuenta de alu1
sn: alu1
userPassword: alu1
mail: alu1@ua.es

#alumno2
dn: uid=alu2,ou=users,dc=ua,dc=es
objectclass: top
objectclass: uidObject
objectClass: inetOrgPerson
objectclass: person
uid: alu2
cn: Cuenta de alu2
sn: alu2
userPassword: alu2
mail: alu2@ua.es

#profesor1
dn: uid=prof1,ou=users,dc=ua,dc=es
objectclass: top
objectclass: uidObject
objectClass: inetOrgPerson
objectclass: person
uid: prof1
cn: Cuenta de prof1
sn: prof1
userPassword: prof1
mail: prof1@ua.es

#definir rol_usuario y miembros
dn: cn=rol_usuario,ou=roles,dc=ua,dc=es
objectclass: top
```

```
objectclass: groupOfNames
cn: rol_usuario
description: grupo de alumnos
member: uid=alu1,ou=users,dc=ua,dc=es
member: uid=alu2,ou=users,dc=ua,dc=es

#definir rol_administrador y miembros
dn: cn=rol_administrador,ou=roles,dc=ua,dc=es
objectclass: top
objectclass: groupOfNames
cn: rol_administrador
description: grupo de profesores
member: uid=prof1,ou=users,dc=ua,dc=es
```

No vamos a entrar en detalle sobre cómo se configura un servidor LDAP, pero como podéis ver en la configuración, hay una clara jerarquía de objetos. Partiendo del dominio jtech, se crea un usuario administrador y dos unidades organizativas. Dentro de esas dos unidades organizativas (usuarios y grupos) crearemos a su vez un usuario (de login usuario) y un grupo denominado usuarios.

Si os fijáis, al final se indica que el usuario "usuario" pertenece al grupo "usuarios". Cada objeto que hemos comentado se localiza por su dn y en su nombre va arrastrando toda la estructura del directorio.

Para cargar esta configuración en nuestro LDAP, debéis ejecutar lo siguiente:

```
sudo ldapadd -x -D cn=admin,dc=ua,dc=es -W -f directorio.ldif
```

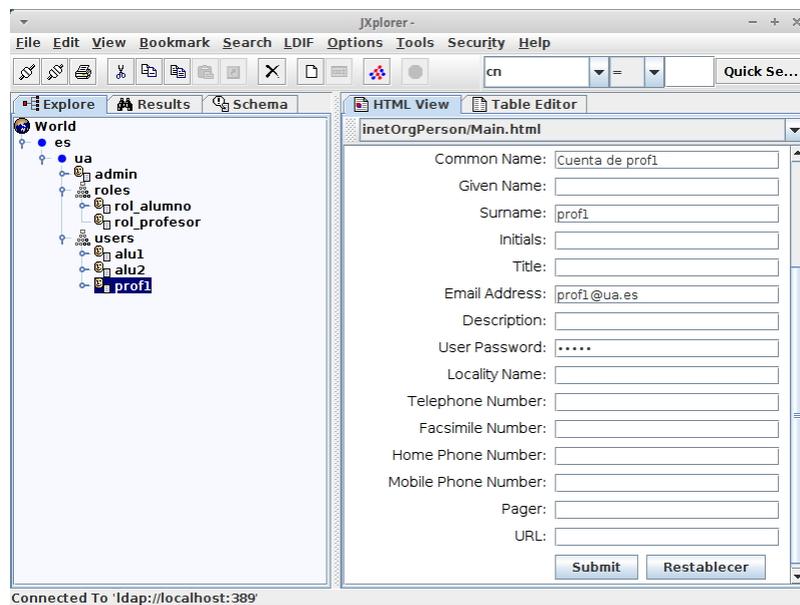
Para comprobar que todo ha ido bien, podéis consultar el directorio mediante el comando ldapsearch:

```
sudo ldapsearch -x -b 'dc=ua,dc=es' '(objectclass=*)'
```

(El asterisco es un comodín, si queréis podéis probar a sustituirlo por cualquier valor de "objectclass" que veáis en la configuración).

JXplorer

Otra opción para trabajar sobre el directorio es utilizar un cliente LDAP, como por ejemplo, JXplorer. Una vez el directorio tenga la configuración básica, podemos conectarnos a él, indicando el dn de la organización y el usuario/password del administrador. Se nos mostrará gráficamente la composición del directorio y se nos permitirá añadir/modificar/borrar elementos del mismo:



Login module LdapExtended

Una vez configurado nuestro directorio LDAP, debemos añadir y configurar el login module específico para Ldap en nuestro security domain:

```
<login-module code="LdapExtended" flag="required" >
  <module-
option name="java.naming.factory.initial" value="com.sun.jndi.LdapCtxFactory"/
>
  <module-option name="java.naming.provider.url" value="ldap://
localhost:389"/>
  <module-option name="java.naming.security.authentication" value="simple"/
>
  <module-option name="bindDN" value="cn=admin,dc=ua,dc=es"/>
  <module-option name="bindCredential" value="expertojavajs"/>
  <module-option name="baseCtxDN" value="ou=users,dc=ua,dc=es"/>
  <module-option name="baseFilter" value="(uid={0})" />

  <module-option name="rolesCtxDN" value="ou=roles,dc=ua,dc=es" />
  <module-option name="roleFilter" value="(member={1})" />
  <module-option name="roleAttributeID" value="cn" />
```

```
<module-option name="searchScope" value="ONELEVEL_SCOPE" />
  <module-option name="allowEmptyPasswords" value="true" />
</login-module>
```



Cuidado con los espacios en blanco después de los valores de cada propiedad que configuréis tanto en los ficheros LDIF como en los valores de las propiedades. Os ahorraréis bastantes quebraderos de cabeza...

3.3. Role Based Access Control en interfaces de administración

Por último, comentar que a partir de WildFly 8, se introduce también el control de autorización en tareas administrativas. Hasta ahora en JBoss cualquier usuario con acceso al ManagementRealm se consideraba super usuario y podía realizar cualquier tarea de administración. Desde esta nueva versión es posible establecer un control mas preciso de lo que un usuario puede hacer y lo que no, habilitando el control RBAC:

```
<management>

  <access-control provider="simple"> <!-- cambiar por rbac-->
    <role-mapping>
      <role name="SuperUser">
        <include>
          <user name="$local"/>
        <user alias="experto" name="experto" /> <!-- Añadir al menos un
usuario como Superusuario después del cambio -->
        </include>
      </role>
    </role-mapping>
  </access-control>

</management>
```

En la sección de roles se muestra un único rol (superusuario), pero tras habilitar el mecanismo de roles tenemos a disposición nuestra los siguientes roles:

Role	Permissions
Monitor	El más restrictivo. Solo permisos de consulta de la información en tiempo de ejecución. No permite la consulta de recursos, datos o información de log.
Operator	Los permisos de Monitor y además puede parar/arrancar instancias, activar/desactivar colas JMS y liberar conexiones de base de datos.
Maintainer	Los permisos del Operator y además puede modificar la configuración: desplegar nuevas aplicaciones y recursos.
Deployer	Los permisos de Maintainer pero restringidos al despliegue de aplicaciones.
Administrator	Los permisos de Maintainer pero además permite ver y modificar datos sensibles tales como sistemas de seguridad. No tiene acceso al sistema de auditoria de administración (audit logging system).

Role	Permissions
Auditor	Los permisos del monitor más la posibilidad de poder consultar/modificar el audit logging system).
Super User	Tiene todos los permisos, y es el equivalente al usuario administrador de versiones anteriores.

La administración de la asignación de roles a usuarios y/o grupos está integrada en la consola Web, en contraposición al alta de usuarios, que se sigue haciendo por script.

En el caso de querer asociar un grupo de usuarios (mgmt-groups.properties) a un rol de administración, se añadiría de la siguiente forma:

```
<role name="Deployer">
  <include>
    <group alias="group-lead-devs" name="lead-developers"/>
  </include>
</role>
```

3.4. Referencias

- WildFly Administration Guide <https://docs.jboss.org/author/display/WFLY8/Admin+Guide>
- OpenLDAP <http://www.openldap.org/doc/admin24/quickstart.html>
- Instalar LDAP en Ubuntu <http://www.openxarxes.com/instalar-y-configurar-ldap-en-ubuntu-lucid-10-04/>
- Security Realms <https://docs.jboss.org/author/display/AS71/Security+Realms>
- Security subsystem configuration <https://docs.jboss.org/author/display/AS71/Security+subsystem+configuration>
- Role Based Access Control <https://docs.jboss.org/author/display/WFLY9/RBAC>

3.5. Ejercicios de seguridad declarativa en WildFly

Nuestro siguiente proyecto será poner en marcha una sencilla intranet para nuestra organización donde se muestren las noticias más importantes del día.

Trabajaréis sobre el proyecto Maven *apli-intranet*. Esta aplicación web contempla dos tipos de acciones:

- Consulta de noticias del día, abierta a cualquier empleado (autenticado) de la empresa, mediante el servlet *ConsultaServlet*.
- Alta de nuevas noticias, únicamente habilitada para los usuarios administradores de la intranet con el servlet *NuevoMensajeServlet*.

El contexto raíz de la aplicación debe ser */intranet* y a este nivel se encuentra la página y el servlet que implementan la consulta. El JSP y Servlet de alta de noticias se encuentran en */intranet/admin* para que podamos establecer restricciones de seguridad diferenciadas.

La información relativa a nuestra aplicación se almacenará en la BD *seguridad* que hemos presentado en la teoría y que crearemos con el script incluido en las plantillas. Para acceder a la

base de datos, crearemos un origen de datos al que denominaremos *java:jboss/datasources/seguridad*.

La tabla donde almacenaremos las noticias tiene el siguiente esquema:

```
CREATE TABLE `noticias` (
  `fecha` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
  `noticia` varchar(1024) DEFAULT NULL
)
```

Para poner en marcha este proyecto definiremos la seguridad en tres fases separadas, en las que iremos sumando distintos orígenes de datos de información de autenticación (ficheros, base de datos, LDAP) para que trabajen de forma *conjunta*.

Configuración, despliegue y alta de usuarios en WildFly (0.6 puntos)

En este ejercicio configuraremos el modelo de seguridad general que queremos para nuestra aplicación. Para ello crearemos un nuevo security domain, al que llamaremos **seguridad-domain**. Funcionará de forma similar al security domain *Other* pero almacenando la información de usuarios y roles en ficheros específicos.

Será necesario crear un nuevo realm, de nombre **IntranetRealm** y que almacenará la información de usuarios en los siguientes ficheros:

- intranet-users.properties
- intranet-roles.properties

Una vez definido el modelo de seguridad general, definiremos los siguientes roles:

- *usuarios*, que contendrán al conjunto de usuarios de la empresa y tendrán el rol: *rol_usuario*.
- *administradores*, que serán aquellos usuarios que tengan permiso para dar de alta nuevas noticias en la aplicación, poseerán el rol: *rol_administrador*.

Como el desarrollo de la aplicación lo llevan a cabo dos programadores, daremos de alta cada uno de ellos con un rol distinto y los incluiremos como usuarios del IntranetRealm de WildFly:

- *programador1* (clave programador1), que tendrá asociado el rol usuario.
- *programador2* (clave programador2), asociado al rol administrador.

Por último, habrá que modificar el código fuente de la aplicación para que:

- Se utilice el nuevo security domain y especificar el contexto raíz */intranet*.
- Definir los roles autorizados para ejecutar los distintos servlets mediante anotaciones:
 - # */ConsultaServlet*: roles usuario y administrador:
 - # */admin/NuevoMensajeServlet*: sólo podrán ejecutarlo los usuarios administradores

Debéis comprobar que el usuario "programador1" sólo puede ver las noticias del día, mientras que el usuario "programador2" también puede añadir nuevas noticias a la base de datos.

Gestión de usuarios en Base de datos (0.3 puntos)

En nuestra empresa hay un número variable de empleados externos. Estos empleados no pertenecen a ningún directorio corporativo pero se mantienen en una base de datos. El objetivo de este ejercicio es añadir al security domain un login module que trabaje con usuarios y grupos mantenidos en la BD *seguridad*.



Debeis configurar login module como se explica en teoría, configurando el parámetro *flag* de los login module de modo que un usuario se reconozca como autenticado cuando sea válido para al menos un login module (basado en ficheros o en BD).

Para comprobar que todo está en orden, daréis de alta los siguientes usuarios:

- *consultor1* (clave consultor1), asociado al rol usuario.
- *consultor2* (clave consultor2), asociado al rol administrador.

Es importante almacenar las contraseñas codificadas por tanto hay que configurar el login module de modo que utilice MD5 en la validación. Asimismo tenéis que comprobar que los usuarios programador1 y programador2 siguen trabajando con normalidad!!

Seguridad integrada con LDAP corporativo (0.3 puntos)

En esta última fase, enlazaremos el directorio corporativo de la empresa con el security domain de WildFly, de modo que un usuario se definirá en el directorio, y en función del rol asociado, tendrá acceso parcial o completo a la aplicación.



Debéis seguir los pasos descritos en teoría y ejecutar el script LDIF que se adjuntan como plantilla. Este script creará automáticamente los siguientes usuarios:

- *alu1* (clave alu1) asociado al rol usuario.
- *alu2* (clave alu2) asociado al rol usuario.
- *prof1* (clave prof1) asociado al rol administrador.

Podéis utilizar la herramienta *JXplorer* para comprobar que los usuarios se han creado correctamente, y si tenéis problemas para autenticar algún usuario, recordad que tenéis que habilitar la auditoría de seguridad, para que os dé más detalle de lo que ocurre.

El resultado de las tres fases es un security domain que obtiene información sobre los usuarios de múltiples orígenes y permite establecer reglas de seguridad unificadas.

Entrega

En esta sesión debeis entregar el proyecto apli-intranet con las modificaciones requeridas y la configuración de WildFly *standalone.xml*:
`:huge-size: scaledwidth="100%" :large-size: scaledwidth="66%" :normal-size: scaledwidth="50%" :small-size: scaledwidth="33%" :tiny-size: scaledwidth="25%"`

4. Certificados digitales y SSL

Esta sesión está dedicada a entender en qué consiste la seguridad en la capa de transporte y cómo utilizar certificados digitales.

4.1. Seguridad en la capa de transporte

Certificados digitales

Junto con los passwords, el mecanismo de autenticación más común en aplicaciones enterprise son los certificados. Comúnmente se emplean para autenticar al servidor, pero también se pueden usar para el cliente, como se hace en la mayoría de organismos oficiales que ofrecen servicios a través de la red.

El uso de certificados viene derivado de la criptografía asimétrica en las que se trabaja con dos claves una privada y una pública. En el caso de una comunicación emisor/receptor podemos plantear dos supuestos:

1. El emisor encripta un mensaje mediante una clave pública y el receptor lo desencripta mediante una clave privada que sólo él conoce. Este tipo de comunicación garantiza la confidencialidad del mensaje pues sólo el receptor puede saber qué contiene realmente.
2. El emisor encripta un mensaje mediante una clave privada que sólo él conoce. El receptor en cambio lo desencripta con una clave pública que cualquier puede utilizar. Mediante este sistema garantizamos la identidad del emisor, y es la base de la **firma digital**.

Un certificado digital no es ni más ni menos que un pequeño documento en el que incluye información sobre la identidad de un individuo o empresa, y una clave pública asociada a este individuo. Por este motivo, también se les conoce como certificados de clave pública o certificados de identidad. Existen varios formatos para certificados digitales no obstante los más comunes se rigen por el estándar **X.509**.

Esta asociación debe ir firmada digitalmente por alguien de confianza que le otorgue validez, que son las denominadas **autoridades de certificación (CA)**. Hay autoridades de certificación privadas, compañías como Thawte o Verisign, y también públicas. En España tenemos la FNMT y en muchas comunidades autónomas también tienen sus propias CA's.

Estas entidades a su vez utilizan certificados digitales para acreditar su identidad y poder firmar digitalmente nuestros certificados, por tanto podríamos llegar a plantearnos ¿Quién firma los certificados de las autoridades certificadoras?

Para evitar una sucesión de certificados sin fin, existen los denominados certificados **autofirmados**, es decir firmados por el propio titular del certificado. Evidentemente estos certificados son un punto crítico en la seguridad del sistema y sólo deben obtenerse de fuentes y métodos absolutamente fiables. Normalmente vienen ya instalados en los sistemas (servidores de aplicaciones o navegadores) o se obtienen de forma presencial.

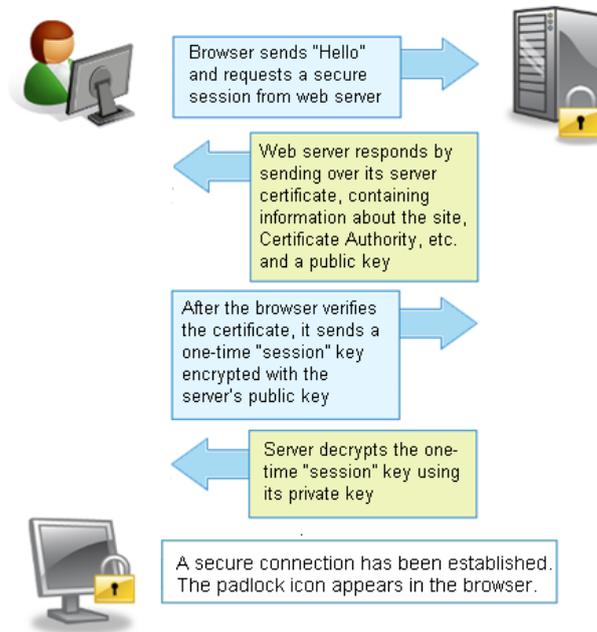
La secuencia anidada de certificados se denomina *cadena de confianza* (chain of trust). En dicha cadena, el certificado auto-firmado se denomina *certificado raíz* (root certificate).

Introducción a SSL

SSL es un protocolo es un protocolo desarrollado por Netscape, para permitir confidencialidad y autenticación en Internet. Funciona como una capa adicional con lo cual es posible

combinarlo con HTTP, FTP o TELNET que son protocolos que operan en la capa aplicación de TCP/IP.

Para establecer una comunicación segura, cliente y servidor entablan un diálogo denominado **handshake**:



Las fases son las siguientes:

1. Solicitud de SSL: El cliente solicita al servidor un recurso mediante conexión segura SSL. El servidor normalmente acepta este tipo de peticiones por un puerto distinto al habitual.
2. El servidor responde enviando un certificado digital para que el cliente pueda comprobar la identidad del servidor, el algoritmo de criptografía más potente de los que ambos puedan utilizar y una clave pública para encriptar la información que van a intercambiar.
3. El cliente comprueba si el certificado es de un sitio de confianza y si esto es así genera una clave privada de "sesión" aleatoria que envía encriptada al servidor con su clave pública.
4. Esta clave de sesión junto con el algoritmo de encriptación es lo que se va a utilizar en cada mensaje entre cliente y servidor. Para verificar los valores negociados, ambas partes se envían de nuevo la clave de sesión y verifican que coincidan. Si esto es así, la fase de handshake finaliza y se establece el canal de comunicación seguro.

One-Way y Two-Way SSL

En el ejemplo de handshake anterior, observamos que es el servidor el que en un momento dado debe acreditar su identidad mediante un certificado digital, que el cliente debe validar. Este tipo de negociación recibe el nombre de One-Way SSL.

Adicionalmente, puede exigir al cliente al cliente que se identifique enviando otro certificado digital, que el servidor deberá validar. Si el uno confía en el otro, se establecerá el canal de comunicación seguro. A esta modalidad de SSL se la denomina Two-Way SSL.

Creación de una CA ficticia

El primer paso para trabajar con certificados y SSL es verificar que tenemos instaladas las herramientas OpenSSL. Podéis probar a ejecutar `openssl` y si no se reconoce el comando instalarlo con:

```
sudo apt-get install openssl
```

Una vez instalado, hay que realizar un cambio en la configuración, que nos va a permitir firmar certificados:

```
/usr/lib/ssl/openssl.conf Línea 170  
basicConstraints=CA:TRUE # Antes FALSE
```

Utilizaremos *openssl* para generar certificados y firmar, y la herramienta *keytool* de java para importar estos certificados en almacenes de certificados en formato JKS.



keytool es una herramienta muy completa pero está pensada para trabajar con certificados autofirmados (que adicionalmente pueden ser enviados a una CA para que los firme). OpenSSL proporciona comandos para firmar certificados a partir de uno especificado como parámetro, a modo de CA y simular completamente lo que haría una autoridad de certificación.

Los almacenes son ficheros protegidos donde podemos almacenar uno o mas certificados digitales. Cada certificado digital estará asociado a un *alias* que se utilizará para recuperar el certificado del almacén. Es habitual tener un almacén para los certificados de identidad (aquellos que el servidor de aplicaciones utiliza para identificarse) y otro de confianza (certificados de cliente y/o CA's considerados como válidos).

1.- Para empezar, vamos a definir un certificado de CA ficticia que será quien de "credibilidad" a los certificados que vamos a crear.

```
openssl req -x509 -newkey rsa:2048 -keyout selfkey.pem -out selfcert.pem -  
days 365
```

Este comando nos pedirá la información que describirá a nuestra CA:

```

Terminal - expertojavajs@expertojavajs-VirtualBox: ~/cert
Archivo  Editar  Ver  Terminal  Pestañas  Ayuda
8 -keyout selfkey.pem -out selfcert.pem -days 365
Generating a 2048 bit RSA private key
.....+++
.....+++
writing new private key to 'selfkey.pem'
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:ES
State or Province Name (full name) [Some-State]:Spain
Locality Name (eg, city) []:Alicante
Organization Name (eg, company) [Internet Widgits Pty Ltd]:UA
Organizational Unit Name (eg, section) []:jtech
Common Name (e.g. server FQDN or YOUR name) []:CA
Email Address []:
expertojavajs@expertojavajs-VirtualBox:~/cert$

```

2.- Ahora importaremos este certificado en un almacén denominado trust.jks (certificados de confianza)

```

keytool -import -trustcacerts -alias trustself -keystore trust.jks -file
selfcert.pem -keyalg RSA -keypass secreto -storepass secreto

```

A continuación veremos como definir configurar WildFly en función del tipo de seguridad que queremos configurar.

Configuración del servidor para One-Way SSL

En esta modalidad de SSL, es el servidor el que debe identificarse al cliente y éste confiar en él (o no).

Pasos a seguir:

1.- Crear un certificado que identifique a nuestro servidor, para ello utilizaremos OpenSSL y los siguientes comandos:

```

openssl genrsa -out server.key 1024 #crear una clave nueva de certificado
openssl req -new -key server.key -out server.csr #crear una petición de
certificado.

```

Al ejecutar el segundo comando, la herramienta nos irá pidiendo datos para incluir en la petición de certificado:

```

Terminal - expertojavajs@expertojavajs-VirtualBox: ~/cert
Archivo  Editar  Ver  Terminal  Pestañas  Ayuda
.....+++++
e is 65537 (0x10001)
expertojavajs@expertojavajs-VirtualBox:~/cert$ openssl req -new -key server.key
-out server.csr
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:ES
State or Province Name (full name) [Some-State]:Spain
Locality Name (eg, city) []:Alicante
Organization Name (eg, company) [Internet Widgits Pty Ltd]:ua
Organizational Unit Name (eg, section) []:jtech
Common Name (e.g. server FQDN or YOUR name) []:localhost
Email Address []:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:secreto
An optional company name []:
expertojavajs@expertojavajs-VirtualBox:~/cert$

```

Si quisiéramos obtener un certificado aprobado por una CA real, deberíamos enviar el fichero server.csr a dicha autoridad (y pagar), y esta nos generará el certificado y nos dará las instrucciones necesarias para descargarlo.



Si alguien está interesado, algunas CA's como Verisign ofrecen certificados válidos durante 30 días a modo de prueba.

2.- Creación del certificado firmado con el certificado con nuestra CA:

```

openssl x509 -req -days 365 -in server.csr -CA selfcert.pem -CAkey
selfkey.pem -set_serial 01 -out server.crt

```



El serial number del certificado sirve para identificar los certificados firmados por una CA, y debe ser distinto para cada certificado. Por simplicidad, vamos a enumerarlos manualmente.

3.- Importar el certificado en nuestro almacén de identidad:

La herramienta keytool no permite importar la clave privada de los certificados en los almacenes a partir de un certificado y su clave, por lo que tenemos que recurrir a un formato intermedio:

```

# Convertir a PKCS12
openssl pkcs12 -export -name servercert -in server.crt -inkey server.key -
out keystore.p12
#finalmente importarlo en jks

```

```
keytool -importkeystore -destkeystore identity.jks -srckeystore
keystore.p12 -srcstoretype pkcs12 -alias servercert -keypass secreto -
storepass secreto
```

4.- Comprobar que los certificados están importados correctamente:

```
keytool -list -v -keystore trust.jks -storepass secreto
keytool -list -v -keystore identity.jks -storepass secreto
```



Hay muchas contraseñas involucradas en este proceso. Para no complicarnos demasiado utilizaremos la palabra "secreto" como contraseña en todos los casos.

5.- Copiar los almacenes a la carpeta *standalone/configuration*

6.- Definir el realm asociado a los certificados:

```
<security-realm name="SecureRealm">
  <server-identities>
    <ssl>
      <keystore path="identity.jks" relative-
to="jboss.server.config.dir" keystore-
password="secreto" alias="servercert" key-password="secreto"/>
    </ssl>
  </server-identities>
</security-realm>
```

7.- Para activar SSL hay que definir un *https-listener* dentro del subsistema *undertow* (el servidor web de WildFly):

```
<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="SecureRealm"/>
  </server>
</subsystem>
```

Hay dos elementos interesantes en esta línea:

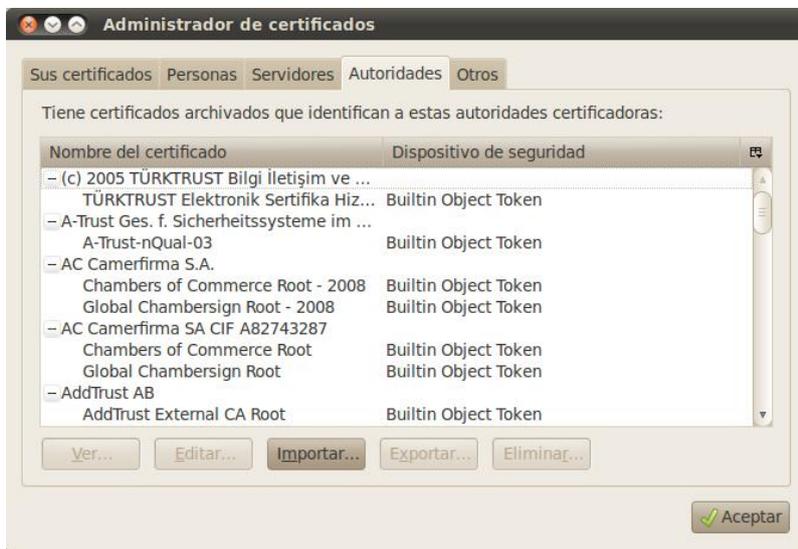
- El parámetro `socket-binding` está asociado a "https" pero aplica tanto a las peticiones estándar (8443) como a las de *management-https*, (9993).
- El parámetro `security-realm` asocia el canal a un Security Realm que será el que contenga al certificado de identidad del servidor.

Con esta configuración, estamos indicando que para las peticiones que lleguen por https nuestro servidor se identificará con el certificado asociado al alias *servercert*.

8.- Iniciar el servidor. Con esto ya hemos completado los pasos para implementar One Way SSL pero ahora nos queda hacer que el navegador reconozca el certificado *servercert* como válido.

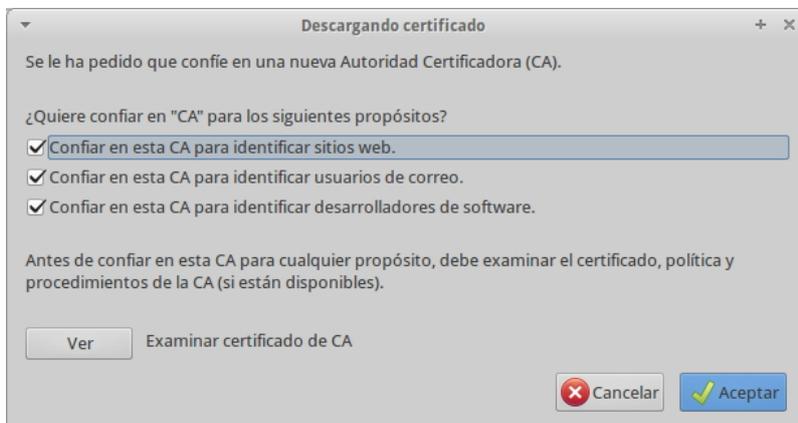


Para conseguir esto basta con importar el certificado de la CA en la lista de certificados de autoridades certificadoras del navegador. En Firefox iremos a *Editar#Preferencias#Avanzado#Cifrado#Ver certificados*:

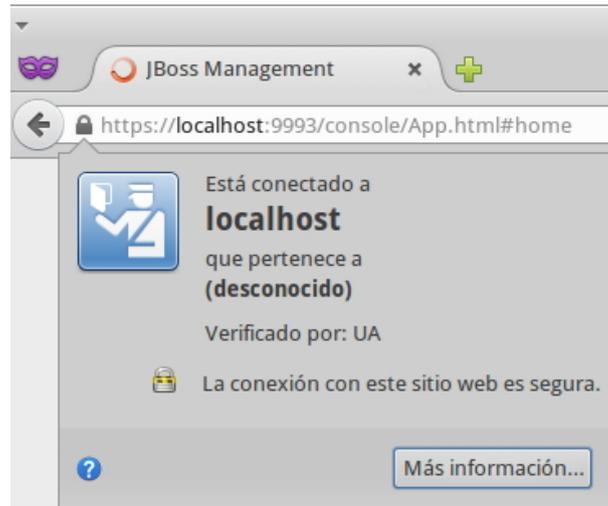


Pincharemos en importar y seleccionaremos el archivo **selfcert.pem**

Se nos pedirá definir para que se utilizará el certificado. Marcamos todas las opciones y aceptamos:



A partir de ahora, si accedemos a la consola, o a cualquier aplicación desplegada en el servidor veremos que se acepta el certificado que envía el servidor:



Cambios en las aplicaciones web

Si queremos que una serie de recursos de nuestra aplicación se muestren forzosamente mediante SSL, debemos añadir una restricción al archivo web.xml

```
<security-constraint>
<web-resource-collection>
<web-resource-name>Redireccion SSL</web-resource-name>
<url-pattern>/*</url-pattern>
</web-resource-collection>
<user-data-constraint>
<transport-guarantee>CONFIDENTIAL</transport-guarantee>
</user-data-constraint>
</security-constraint>
```

Securizar el acceso a la consola web de administración

Ahora que ya conocemos como utilizar SSL, podemos aplicar estos mismos cambios al realm de administración de modo que podamos forzar el acceso a la consola mediante una conexión segura. Los cambios son los siguientes:

1.- Añadir el certificado de servidor al ManagementRealm:

```
<server-identities>
<ssl>
<keystore path="identity.jks" relative-
to="jboss.server.config.dir" keystore-
password="secreto" alias="servercert" key-password="secreto"/>
</ssl>
</server-identities>
```

2.- Asociar el puerto seguro a la interfaz de administración:

```
<http-interface security-realm="ManagementRealm" http-upgrade-
enabled="true">
```

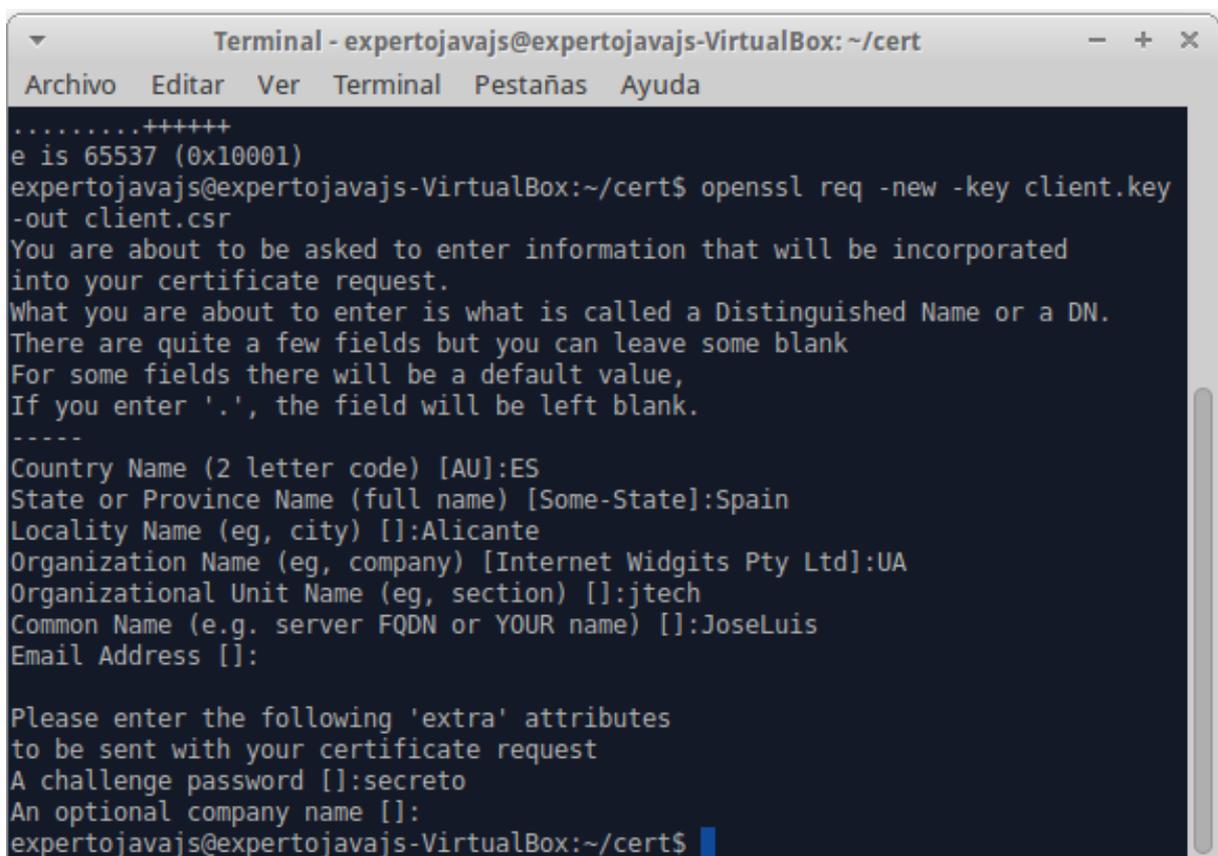
```
<socket-binding https="management-https"/> <!-- Sustituye a management-  
http -->  
</http-interface>
```

Configuración del servidor para Two-Way SSL

Mediante este sistema cliente y servidor se identifican y validan recíprocamente. El configurar esta modalidad de SSL es sencillo, partiendo de la configuración One-Way.

1. Crear un certificado de cliente, firmado por nuestra CA ficticia:

```
openssl genrsa -out client.key 1024  
openssl req -new -key client.key -out client.csr  
openssl x509 -req -days 365 -in client.csr -CA selfcert.pem -set_serial  
02 -CAkey selfkey.pem -out client.crt
```

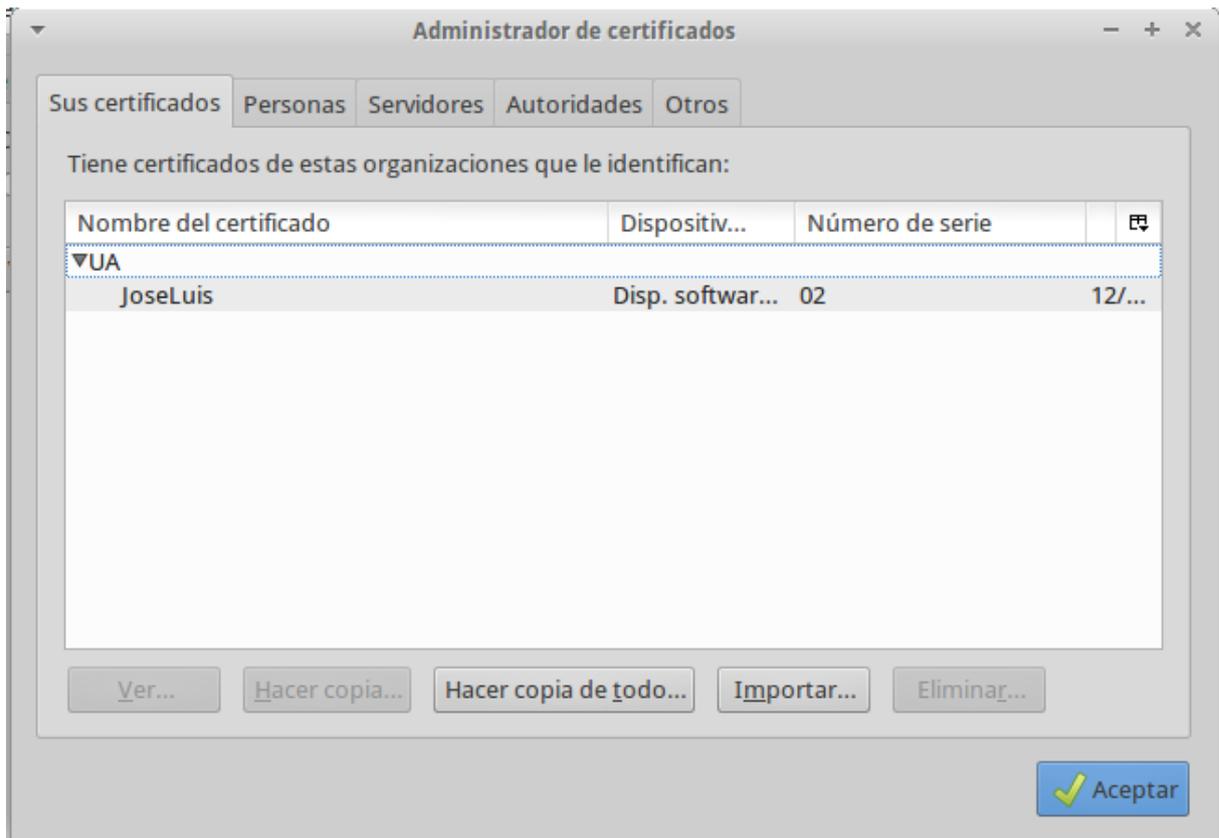


```
Terminal - expertojavajs@expertojavajs-VirtualBox: ~/cert  
Archivo Editar Ver Terminal Pestañas Ayuda  
.....++++++  
e is 65537 (0x10001)  
expertojavajs@expertojavajs-VirtualBox:~/cert$ openssl req -new -key client.key  
-out client.csr  
You are about to be asked to enter information that will be incorporated  
into your certificate request.  
What you are about to enter is what is called a Distinguished Name or a DN.  
There are quite a few fields but you can leave some blank  
For some fields there will be a default value,  
If you enter '.', the field will be left blank.  
-----  
Country Name (2 letter code) [AU]:ES  
State or Province Name (full name) [Some-State]:Spain  
Locality Name (eg, city) []:Alicante  
Organization Name (eg, company) [Internet Widgits Pty Ltd]:UA  
Organizational Unit Name (eg, section) []:jtech  
Common Name (e.g. server FQDN or YOUR name) []:JoseLuis  
Email Address []:  
  
Please enter the following 'extra' attributes  
to be sent with your certificate request  
A challenge password []:secreto  
An optional company name []:  
expertojavajs@expertojavajs-VirtualBox:~/cert$
```

2. Convertir al formato PKCS12 que incluye clave pública y privada y es reconocido por Firefox:

```
openssl pkcs12 -export -name clientcert -in client.crt -inkey  
client.key -out clientstore.p12
```

3. Ahora importaremos el certificado en el navegador. Recordad que ya tenemos el de la CA. Se debe importar el archivo .p12 en la pestaña de "sus certificados":



4. Pasamos a continuación a WildFly donde activaremos la función de autenticación en el Security Realm *SecureRealm* especificando el almacén de certificados de confianza.

```
<security-realm name="SecureRealm">
    ....
    <authentication>
        <truststore path="trust.jks" relative-
to="jboss.server.config.dir" keystore-password="secreto"/>
    </authentication>
</security-realm>
```

5. Por último, tenemos que indicar que queremos que el servidor "pida" al cliente que se identifique. Esto lo hacemos añadiendo un parámetro adicional al `_http-listener`:

```
<https-listener name="https" socket-binding="https" security-
realm="SecureRealm" verify-client="REQUESTED"/>
```

El parámetro `verify-client` tiene tres valores posibles:

NOT_REQUESTED

No solicitar certificado a cliente (valor por defecto)

REQUESTED

Se pide certificado de cliente, pero si no se presenta se permite continuar la conexión segura.

REQUIRED

Se requiere un certificado de cliente válido para poder establecer una conexión segura.

Autorización del cliente mediante certificados

A partir de un usuario identificado por su certificado de cliente, también podemos determinar si tiene un rol asociado y en consecuencia permitirle o no el acceso a determinados recursos. En la sesión anterior hemos aprendido a definir login modules específicos para autenticación y autorización de usuarios con respecto a la información de usuarios que teníamos en ficheros, base de datos o en un directorio LDAP. Ahora introduciremos un nuevo login module: *CertificateRoles*.

```
<security-domain name="seguridad-cert" cache-type="default">
  <authentication>
    <login-module code="CertificateRoles" flag="required">
      <module-option name="securityDomain" value="seguridad-cert"/>
      <module-
option name="verifier" value="org.jboss.security.auth.certs.AnyCertVerifier"/
>
      <module-option name="rolesProperties" value="file:/usr/local/wildfly/
standalone/configuration/roles.properties"/>
    </login-module>
  </authentication>
  <jsse keystore-password="secreto" keystore-url="file:/usr/
local/wildfly/standalone/configuration/trust.jks" truststore-
password="secreto" truststore-url="file:/usr/local/wildfly/standalone/
configuration/trust.jks"/>
</security-domain>
```

En este fragmento de XML definimos un nuevo dominio de seguridad *_seguridad-cert*, en el que introducimos nuevo login module. Los parámetros que son novedad para nosotros son:

- *rolesProperties*: Este login module requiere de un fichero externo en el que se indique para cada identidad asociada a un certificado (CN) a que rol o roles pertenece.
- *jsse*: Referencia al almacén de seguridad de certificados de confianza.

El fichero *rolesProperties* sigue el mismo formato que el fichero *application-roles.properties* sin embargo el usuario se debe identificar por el DN (distinguished name) completo del certificado, es decir con todos los datos que se nos pedía a la hora de generarlo. Además, tanto los espacios como los signos "=" se deben escapar con contrabarras. Como ejemplo:

```
CN\=JoseLuis,\ OU\=jtech,\ O\=UA,\ L\=Alicante,\ ST\=Spain,\ C
\=ES=rol_usuario
```

El último signo "=" no se escapa pues es el que separa el usuario de los roles.

Por último, también hay que realizar cambios en las aplicaciones, concretamente hay que cambiar el modo de autenticación de la aplicación web de BASIC a CLIENT-CERT en el fichero *web.xml*:

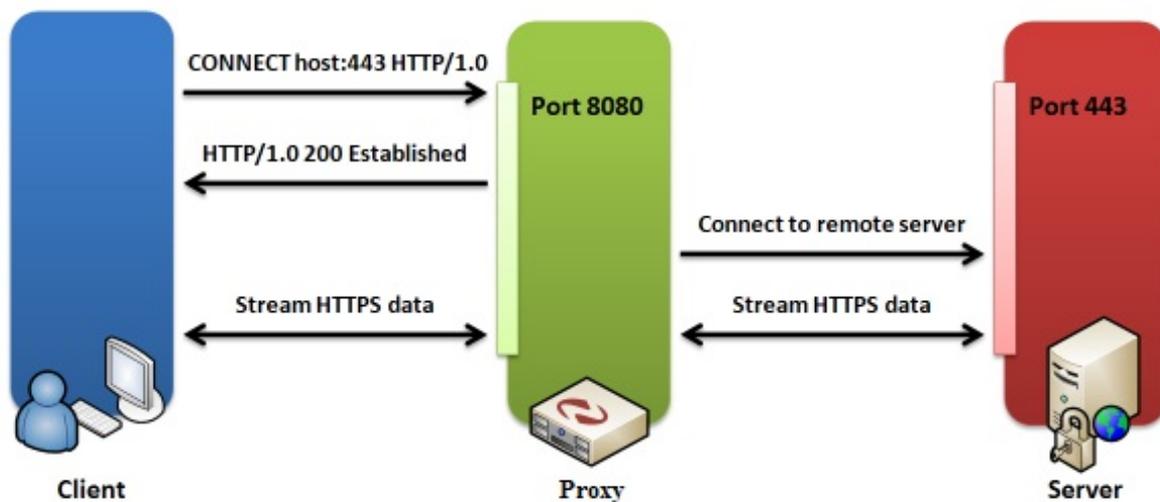
```
<login-config>
<auth-method>CLIENT-CERT</auth-method>
<!--<auth-method>BASIC</auth-method-->
<realm-name>mydomain</realm-name>
```

</login-config>

Comunicación SSL a través de proxies

Como ya sabemos, las conexiones SSL son conexiones punto a punto entre cliente y servidor y no admite elementos intermedios que puedan poner en riesgo la seguridad de las comunicaciones. Sin embargo, hay elementos como los denominados balanceadores de carga, que son capaces de encaminar una petición a distintos servidores en función de la carga de trabajo u otros parámetros.

En el caso de trabajar con este tipo de elementos, la conexión SSL se divide en dos tramos, una conexión segura entre el cliente y el "proxy", y otra entre el proxy y el servidor. Ambas conexiones se establecen de la misma forma que una conexión directa, es decir una parte hace de cliente y la otra de servidora, sin embargo el certificado del cliente no se utiliza para negociar la conexión con el servidor, si no el del propio proxy. Gráficamente:



Bajo este supuesto, podremos hacer llegar las credenciales del cliente al servidor, si utilizamos un proxy Apache en este elemento intermedio. Una de sus funciones es la de "inyectar" la información del certificado cliente en una cabecera especial, denominada `SSL_CLIENT_CERT`. Esta información también estará disponible para las aplicaciones, si estas consultan la cabecera en el parámetro request de la petición.

4.2. Referencias

- Login modules en WildFly <https://docs.jboss.org/author/display/WFLY8/Authentication+Modules>
- Securing Realms <https://docs.jboss.org/author/display/WFLY8/Security+Realms>
- SSL <https://docs.jboss.org/author/display/WFLY8/Admin+Guide#AdminGuide-%7B%7B%3Cssl%2F%3E%7D%7D>

4.3. Ejercicios de certificados digitales y SSL

Vamos a seguir trabajando con el dominio seguridad y la aplicación *apli-intranet* de la sesión anterior. Para poder realizar los ejercicios se debe haber completado, al menos, el primer ejercicio de la sesión anterior.

Encriptación de las comunicaciones One-Way SSL (0.4 puntos)

Queremos reforzar la seguridad de las comunicaciones con nuestro servidor, creando un canal seguro SSL en el que el servidor se identificará y el navegador del cliente deberá reconocer dicho certificado como válido.

Hay que crear los certificados de la CA, de servidor, importar la CA en el navegador y configurar el servidor para que utilice SSL.

Encriptación de las comunicaciones con Two-Way SSL (0.4 puntos)

Ahora lo que se pretende es que tanto servidor como cliente validen sus certificados respectivos. A los certificados ya creados en el ejercicio anterior se les une un certificado asociado a un *Usuario1* y que sea el certificado con el que el navegador se identifique para establecer la conexión.

Siguiendo los pasos descritos en la teoría, configurar el servidor para que se requieran certificados de cliente y se validen.



Debéis usar la misma contraseña para los almacenes que hemos visto en teoría para que el ejercicio pueda ser corregido.

Autenticación de usuario a partir del certificado (0.4 puntos)

En el ejercicio anterior generamos un certificado identificativo para el cliente mediante el cual se establece un canal seguro si el servidor lo acepta. Sin embargo este mecanismo convive con la validación BASIC mediante usuario/password, con lo que tenemos que seguir pidiendo credenciales al usuario para saber exactamente quién está trabajando desde el navegador y a qué grupo pertenece.

La propuesta de este ejercicio es hacer los cambios pertinentes en el dominio de seguridad y en la aplicación para que valide e identifique directamente al usuario cliente en base a su certificado sin comprobar nada más. Sería algo parecido a conectarnos a la web de la AEAT o a la Seguridad Social empleando un certificado digital que nos identifique.

Para completar el ejercicio teneis que:

1. Asociar el certificado del *usuario1*, del ejercicio anterior al rol de usuario.
2. Crear un nuevo certificado asociado a un *usuario2* y asociarlo al rol de administrador.
3. Configurar el servidor para que se utilice el login module de certificados.
4. Cambiar la seguridad de la aplicación intranet para que se valide en base a certificados.
5. Comprobar que si nos autenticamos con los certificados de usuario, cada uno se asocia al rol correspondiente.



Si añadís los tags para forzar el acceso SSL a la aplicación, podéis tener problemas a la hora de tratar los certificados de cliente, por la redirección que hace la aplicación apli-intranet, desde el JSP al servlet de consulta. Lo más sencillo, no utilizarlo.</note>



Especificar tanto en *web.xml* como en el descriptor *jboss-web.xml* el nombre del security domain que vayáis a utilizar y estad atentos al fichero de auditoría de seguridad (audit.log).

Entrega

Los entregables de esta sesión son el proyecto apli-intranet, los ficheros properties que defináis y el fichero *standalone.xml* con la configuración del servidor. Comentad en el código o en los descriptores los cambios que vayáis haciendo para poder comprobar que habéis completado los distintos ejercicios propuestos. :huge-size: scaledwidth="100%" :large-size: scaledwidth="66%" :normal-size: scaledwidth="50%" :small-size: scaledwidth="33%" :tiny-size: scaledwidth="25%"

5. Introducción a Cloud Computing y a OpenShift

5.1. ¿Qué es Cloud Computing?

Cloud Computing es una denominación comercial a un modelo de computación que permite utilizar recursos tecnológicos bajo demanda, ofrecidos por un proveedor. Este modelo se enfrenta al modelo tradicional de adquisición y administración de los recursos necesarios para proporcionar una cierta funcionalidad o servicio.

El uso de estas tecnologías no es algo nuevo, puesto que desde hace años estamos hablando de servicios de hosting o aplicaciones web. Incluso desde los años 70 existían las sesiones de Time Sharing en los grandes mainframes que nos permite acceder a los recursos de un gran ordenador desde un sencillo terminal.

Lo novedoso es que basándose en estas tecnologías se ha conseguido desarrollar un modelo de negocio que permite ofrecer recursos ajustados a las necesidades de los clientes.

Computación Pre-Cloud

Antes de las tecnologías Cloud, cualquier proyecto informático requería de una fuerte inversión previa tanto en hardware como en software.

En la parte hardware normalmente incluiremos ordenadores, infraestructura de red, almacenamiento de datos, etc. En la parte software normalmente tendremos el coste de las licencias de los sistemas operativos y de las aplicaciones que se vayan a utilizar. A este coste hay que sumarle los desarrollos a medida que se pudieran necesitar.

Ampliar hardware no es barato y el que adquiere se debe sobredimensionar por dos motivos:

- *Amortización de la inversión:* Debe utilizarse durante un mínimo de tiempo y permitir escalabilidad vertical u horizontal.
- *Picos de carga de trabajo:* El sistema debe poder hacer frente a la carga de trabajo que se pueda generar en los momentos de mayor actividad, así como ser lo suficientemente potente para que en caso de contingencia en un nodo, poder continuar dando servicio.

En resumen, para poder implantar un proyecto informático es necesaria una inversión económica considerable, tanto en software como en hardware, y éste último tiende a estar infrautilizado la mayor parte de su tiempo y su ampliación presenta limitaciones.

5.2. ¿Qué nos aporta Cloud Computing?

En contraposición, las principales ventajas que obtendremos al aplicar Cloud Computing son las siguientes:

Flexibilidad

La ventaja más importante de Cloud Computing tener a nuestra disposición los recursos que necesitamos, cuando los necesitamos y con una capacidad de adaptación muy rápida a cambios en los requerimientos.

Estandarización y reusabilidad

Los nuevos modelos de computación se basan en tecnologías estandarizadas, probadas por miles de clientes. El objetivo último es liberar al cliente de otras preocupaciones que no sean su proyecto

Eficiencia

En la práctica se consigue un aprovechamiento mucho mayor de los recursos contratados que mediante la adquisición tradicional. Los sistemas de Cloud Computing aportan métricas de calidad de servicio.

Acceso ubicuo

Mediante Cloud podemos acceder a los servicios contratados desde cualquier lugar, a cualquier hora y desde cualquier dispositivo que tenga acceso a Internet.

Otras ventajas propias de Cloud, con ciertos matices son las siguientes:

Seguridad

Aunque se suele ver más como un riesgo, realmente es una de las características más importantes sobre todo para pymes, pues los centros de datos de los proveedores de cloud deben implementar medidas de seguridad muy superiores a las habituales en la mayoría de las empresas. Hablaremos más adelante de Seguridad.

Ahorro de costes

Los proyectos implantados mediante tecnologías Cloud permiten un ahorro importante de dinero al comienzo de su utilización, y dado de que se paga por uso, a largo plazo también debería haber un ahorro importante frente a un enfoque tradicional.

La realidad es que hacer una previsión de costes a medio o largo plazo no resulta sencillo y a largo plazo puede no ser tan ventajoso. No obstante no hay que perder de vista el resto de ventajas que aporta, sobre todo en flexibilidad.

5.3. Nuevos conceptos de Cloud Computing

Cloud computing aporta nueva terminología, a caballo entre lo técnico y el marketing.

Multitenancy o tenencia múltiple

se refiere al uso compartido de un mismo recurso por parte de muchos usuarios, en contraposición al modelo de uso individual.

Resiliency o resiliencia

En el contexto de cloud computing, se refiere a la capacidad de un proveedor para mantener una calidad de servicio fijada, a pesar de los posibles fallos o ataques que pudiera sufrir.

5.4. Riesgos de Cloud

Por supuesto no todo es más fácil y bonito desde la llegada de Cloud, hay nuevas dificultades en el camino:

¿Cómo aplicar Cloud Computing?

Hay múltiples formas de entender la computación en la nube. Más adelante hablaremos de *Tipos de Cloud*.

Dependencias de terceros

Aparece un nuevo jugador que es el proveedor de servicios. No todos los proveedores ofrecen los mismos servicios ni con las mismas garantías.

Datos de carácter personal

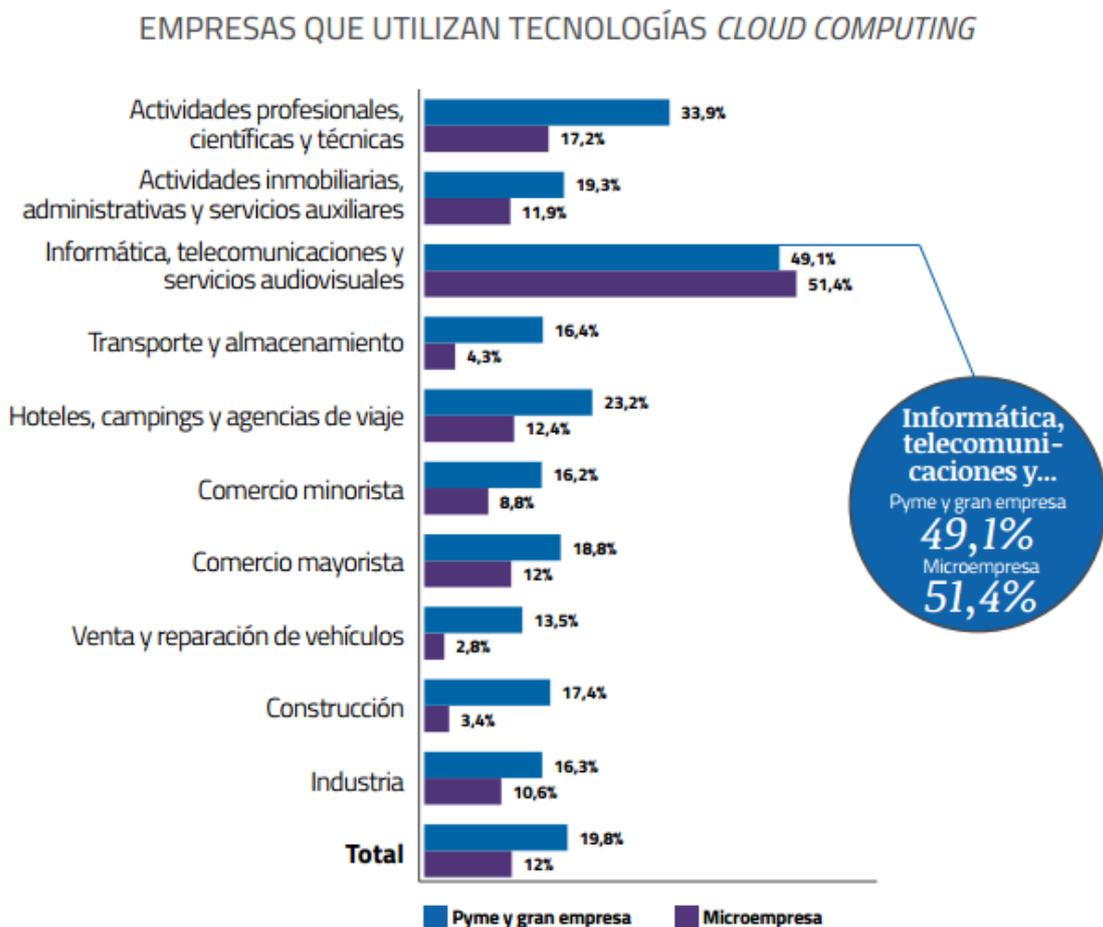
Al hilo del punto anterior, al trabajar en Cloud podemos necesitar transferir datos de carácter personal a un centro de datos externo a nuestra empresa e incluso en un país

extranjero. La Agencia Española de Protección de Datos dicta una serie de normas acerca de cómo se deben manejar este tipo de datos y puede condicionar la elección del proveedor.

Qué opinan las empresas acerca de Cloud

Según un informe del Observatorio Nacional de las Telecomunicaciones de Mayo de 2014, el grado de implantación de las tecnologías Cloud en la empresa española es el siguiente:

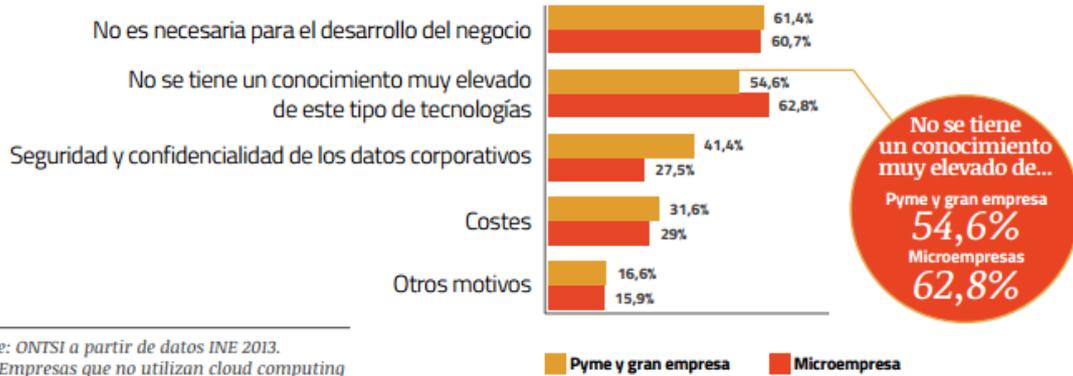
En dicho informe se puede obtener una visión más detallada por sectores. Concretamente en el sector comercial minorista se recogen los motivos que las empresas tienen para no usar tecnologías cloud:



Según este informe, 9 de cada 10 de las empresas que aplican tecnologías CC , utilizan soluciones de almacenamiento y 6 de cada 10, soluciones de backup.

7 / Sector comercio minorista

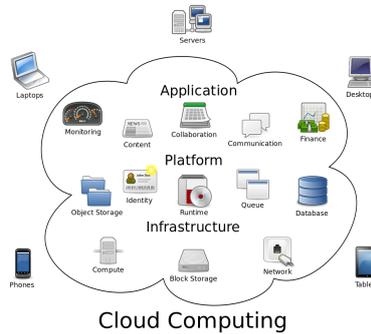
FIGURA 61: MOTIVOS PARA NO USAR SOLUCIONES CLOUD COMPUTING



Fuente: ONTSI a partir de datos INE 2013.
Base: Empresas que no utilizan cloud computing

5.5. Tipos de Cloud en función del servicio

Atendiendo a los servicios que proporcionan, y ordenados de mayor a menor abstracción, tenemos tres grandes tipos de Cloud:



SaaS (Software as a Service)

Dirigido fundamentalmente a *usuarios finales*. El proveedor ofrece el uso de aplicaciones completas que se ejecutan sobre la infraestructura del proveedor.

- El usuario no tiene que instalar ningún producto, únicamente necesita tener conectividad.
- El usuario no tiene ningún control sobre los aspectos técnicos de las aplicaciones a las que tiene acceso.



PaaS (Platform as a Service)

El proveedor proporciona a su cliente un conjunto de módulos estándar (como bases de datos, servidor de aplicaciones) sobre los que poder desplegar sus propias aplicaciones. El usuario habitual de estos servicios suele ser el de perfil *desarrollador*.

- El usuario tiene acceso a la configuración de los distintos módulos que contrata, así como a diversos parámetros que determinan la escalabilidad, seguridad, o auditoría del servicio PaaS
- Sin embargo el usuario se abstrae de las tareas de la configuración de las máquinas, mantenimiento de Sistemas operativos, o aspectos de comunicaciones.



IaaS (Infrastructure as a Service)

Se considera la capa inferior de los servicios Cloud, en la que el proveedor ofrece a sus clientes recursos hardware virtualizados, tales como servidores, sistemas de almacenamiento, enrutadores sobre los que el usuario tiene libertad para configurar como desee. El perfil de usuario habitual de esta tecnología es el de *desarrollador* junto con el *administrador de sistemas*.

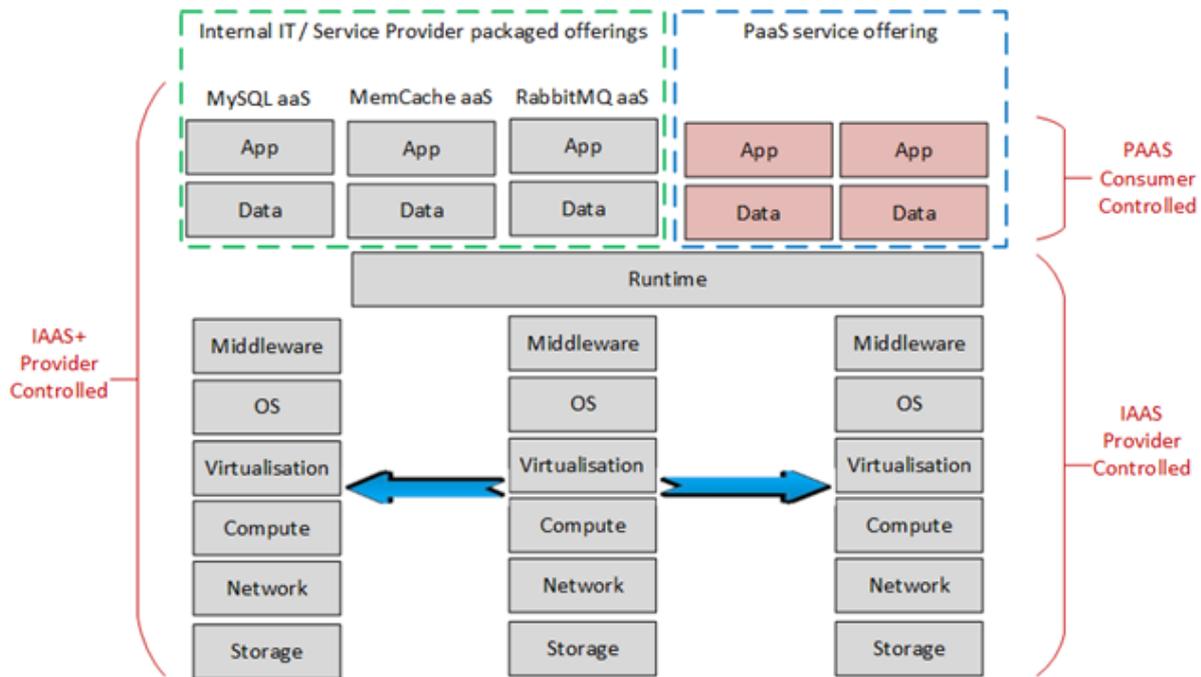
IaaS es el nivel más complejo de gestionar pero es el que otorga una libertad cercana al hardware propietario. Esto no quiere decir que tengamos que configurar todo a mano: en IaaS el proveedor proporciona seguridad, herramientas de clonado, paneles de control, monitorización etc.

Por debajo de estas capas estaría la capa de virtualización del proveedor y finalmente las máquinas físicas que dan el servicio real.



De la división tradicional de Cloud surgen numerosas variantes:

- *IaaS+*: Proporciona simultáneamente los recursos de PaaS (módulos software) y IaaS (recursos hardware virtualizados). En esta categoría podríamos situar a AWS Azure



- *dSaaS: Data Storage as a Service* Tipo de servicio que proporciona almacenamiento remoto y sistema de backup. Se sitúa al nivel de máquinas físicas
- *DCaaS: Data Center as a Service* Modelo de venta de servicios que pone a disposición de los clientes recursos de procesamiento de datos e infraestructura físicos. Orientado a grandes empresas que por motivos concretos no puedan expandir sus actuales CPD's
- *ItaaS: IT as a Service*. Modelo que disocia los departamentos de TI de las pequeñas y medianas empresas. Las empresas se podrán dirigira mercados donde podrán comparar y adquirir la solución que mejor soporte dé a su negocio.

5.6. Tipos de Cloud en función del tipo de acceso

Cloud pública

Donde los datos de distintos clientes se tratan conjuntamente en los sistemas del proveedor. Los costes son más económicos ya que se hace un uso muy eficiente de los recursos, pero el uso compartido puede mermar la confidencialidad de la información.

Cloud privada

Cuando por la naturaleza de los datos se requiere una seguridad adicional, el proveedor proporciona al cliente una infraestructura aislada del resto donde almacenar su información, y permite al cliente decidir quien puede tener acceso a la información. También se denomina cloud privada al modelo en el que el cliente utiliza su propia infraestructura y utiliza software de un proveedor para ejecutar una plataforma cloud dentro de sus instalaciones.

Cloud híbrida

Básicamente es una combinación de los dos modelos anteriores, estableciendo una comunicación VPN entre los mismos. Por ejemplo, una aplicación puede trabajar con datos de clientes que no sean de carácter personal, pero almacenar información de este tipo en la parte privada de modo que su confidencialidad sea máxima.

Cloud comunitaria

Infraestructuras compartidas entre varias organizaciones con principios similares.

5.7. Seguridad

Dato personal

Un dato personal es cualquier información concerniente a personas físicas identificadas o identificables.

La Ley Orgánica 15/1999 de 13 de diciembre de Protección de Datos de Carácter Personal (LOPD) regula los aspectos relativos al tratamiento de los datos personales y la libre circulación de los datos. La Agencia Española de Protección de Datos (AEPD) es el órgano de control que se encarga de garantizar el cumplimiento de esta normativa dentro del territorio español.

Si los datos con los que se va a trabajar en la nube pertenecen a esta categoría, la empresa que los trate debe cumplir con carácter previo con el conjunto de obligaciones previstas en la LOPD: la inscripción de ficheros, deberes relacionados con la información en la recogida, el consentimiento y la calidad de los datos, garantía de los llamados derechos ARCO (Acceso, Rectificación, Cancelación y Oposición) o la adopción de medidas de seguridad. Si los datos con los que se va a trabajar en la nube no son datos personales, se puede proceder sin que la LOPD señale impedimento alguno.

Transferencia internacional de datos personales

Es un tratamiento de datos que supone una transmisión de los mismos fuera del territorio del Espacio Económico Europeo (EEE), bien constituya una cesión o comunicación de datos, bien tenga por objeto la realización de un tratamiento de datos por cuenta del responsable del fichero establecido en territorio español.

Trabajar en la nube con datos de carácter personal, fuera del EEE tiene unos requisitos especiales:

- Notificación a la AEPD vía formulario online, si se trata de un país con un nivel de protección adecuado: Suiza, Argentina, Guernsey, Isla de Man, Jersey, Canadá (sólo empresas sujetas a la ley canadiense de protección de datos) y EE.UU. (sólo empresas adheridas a los principios de Safe harbour).
- Se deberá contar con la autorización del Director de la Agencia de Protección de Datos para el resto de destinos.

En Junio de 2014 Microsoft fue la primera empresa en conseguir una certificación por parte de la AEPD y la UE, que asegura que sus servicios en la nube ofrecen una buena protección (Office 365, Azure) y por tanto cuentan de antemano con la autorización para poder exportar datos a sus centros de datos.

En Octubre de 2015 el Tribunal de Justicia Europeo invalida los acuerdos de Safe Harbour a raíz del escándalo de Facebook y la vigilancia masiva por parte de la agencia norteamericana NSA. En primera instancia los acuerdos de Safe Harbour impedían la investigación de estos sucesos pero la nueva sentencia judicial permite que los países exijan las mismas garantías de seguridad que dicta las leyes europeas y pueden pronunciarse a favor o en contra de esta cesión de datos con total independencia.

5.8. Cómo elegir un proveedor de Cloud

Lo primero de todo es determinar qué modalidad de cloud se adapta mejor a nuestro proyecto informático (si aplica). Es posible que con pago por uso de una aplicación SaaS podamos cubrir nuestras necesidades, por lo que lo primero es explorar este tipo de posibilidades y evaluar coste/beneficio.

Una vez tenemos claro que necesitamos una solución a medida, debemos distinguir entre una solución más estandarizada, basada en PaaS, o una solución más flexible pero más compleja de administrar, basada en IaaS.

La decisión técnica entre PaaS puede no ser definitiva, pues también influirán los requerimientos del proyecto y por su puesto las tarifas de cada proveedor.

Factores que influyen en la elección de un proveedor de Cloud:

- Servicios ofertados: Es interesante que el proveedor escogido pueda ofrecer servicios que aunque en un primer momento no vamos a necesitar, podamos preveer que en un futuro sean de interés.
- Ubicación del *Centro de datos* donde se alojará la información: Tanto en términos de protección de datos como en rendimiento.
- Tipo de *Centro de datos* donde se alojará la información:
 - # De operador (propietarios) Son privados. Los operadores bonifican los costes pero dificultan los cambios.
 - # Neutrales (independientes) Accederemos a un pool de servicios. Mayor oferta de servicios y mayor facilidad para cambiar de proveedores.
- Calidad de las instalaciones:
 - # Tier 1 (sala refrigerada con recursos de extinción)
 - # Hasta tier 4 (Disponibilidad de al menos 99.995% del tiempo, personal 24x7, redundancia en almacenamiento, comunicaciones, refrigeración...)
- Medidas de seguridad: iguales o superiores a las necesarias para el proyecto.
- Políticas de importación/exportación de datos: que faciliten tanto la carga de información, como la extracción en el caso de que decidamos cambiar de proveedor.
- Calidad del servicio:
 - # SLA: acuerdos de nivel de servicio, con obligaciones y penalizaciones.
 - # SLO: objetivos de nivel de servicio, medida orientativa de la calidad de servicio que se va a obtener.
- Costes: siempre evaluados de forma conjunta con el resto de factores. Se deberá crear un escenario de uso, lo más completo posible para poder preveer mejor los costes.

Por último, destacar que la mayoría de los proveedores con centro de datos ubicados en España, ofrecen servicios de SaaS o IaaS.

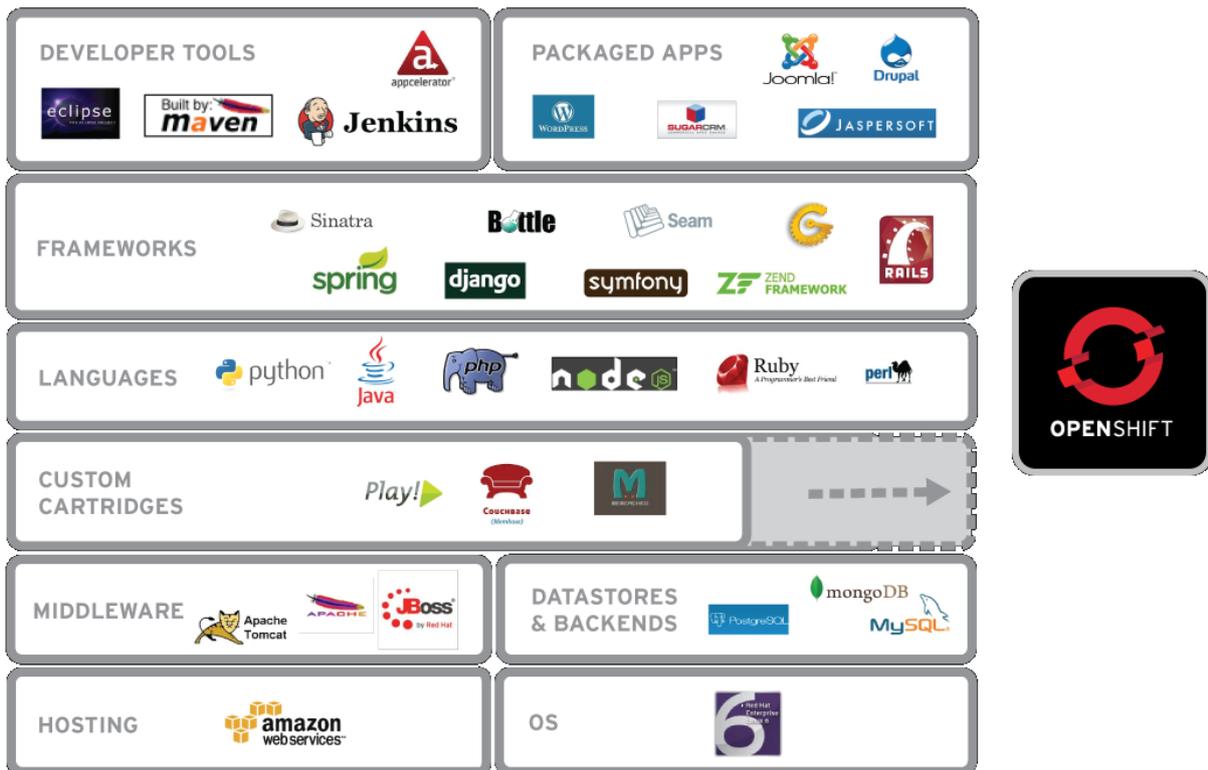
5.9. Introducción a OpenShift

A lo largo de estas sesiones vamos a utilizar PaaS como plataforma de trabajo por estar más enfocada a las necesidades del desarrollador, concretamente la solución PaaS de RedHat. Los motivos para elegir este producto para el curso son varios:

- Permite desplegar aplicaciones basadas en muy diversas tecnologías.
- Si no implementa algún servicio permite integrarse con terceros o construir componentes a medida.

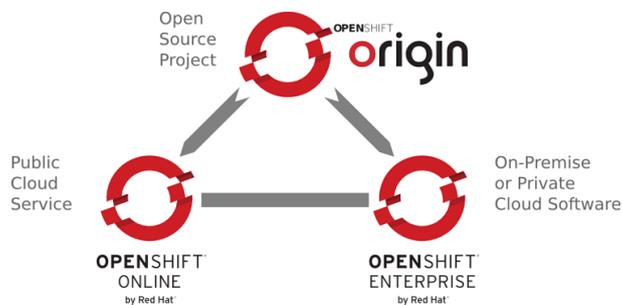
- Tiene una modalidad de trabajo gratuita (y que no requiere de tarjeta de crédito) lo suficientemente abierta para familiarizarse con la plataforma y desplegar aplicaciones de casi cualquier tipo.
- permite conocer la plataforma tanto desde el punto de vista del desarrollador como del administrador de sistemas. Podemos ejecutar nuestra propia nube de OpenShift localmente gracias a una máquina virtual o instalarla en nuestra infraestructura.
- Dentro de la plataforma PaaS, el software utilizado para ejecutar las aplicaciones es el mismo que se utiliza en las aplicaciones de escritorio (presenta características propias de IaaS).

Lenguajes y runtimes soportados



Versiones de OpenShift

A partir de la idea de ofrecer un servicio PaaS OpenShift se desarrolla en tres modalidades distintas:



OpenShift Origin

Es una solución PaaS gratuita, e código abierto y sujeta a licencia Apache2. Este proyecto es sobre el que se inician los nuevos desarrollos de la plataforma y por tanto es el que

tiene las funcionalidades más novedosas, pero también es la plataforma menos rodada. Si queremos trabajar con esta versión de OpenShift, debemos descargar el software y configurar nuestra propia infraestructura para ejecutarla. Si sólo queremos echar un vistazo, podemos descargar una imagen de una máquina ya configurada y ejecutarla con VMWare o VirtualBox.

OpenShift Online

Aproximadamente cada tres semanas se genera una nueva release de OpenShift, que Red Hat se encarga de configurar y desplegar sobre AWS (Amazon) y la oferta como nube pública tanto en modalidad gratuita como en modalidad de pago

OpenShift Enterprise

De forma trimestral se genera una versión en la que sólo se introducen las funcionalidades más maduras de OpenShift, y se ofrece como producto comercial con soporte completo por parte de Red Hat. En este caso, OpenShift Enterprise puede ejecutarse tanto sobre infraestructura de nuestra propia empresa como en modalidad de nube privada en Amazon o Rackspace.

Con estas opciones, es fácil que encontremos una configuración de cloud que se adapte a nuestras necesidades, contando con una ventaja adicional y es que una aplicación es portable entre estas tres versiones siempre y cuando estén disponible los mismos componentes en la plataforma destino.

En estas sesiones trabajaremos con OpenShift Online, dado que nos va a permitir centrarnos en la configuración y codificación de las aplicaciones, dejando las tareas de infraestructura al proveedor.



Nomenclatura versiones OpenShift

Las releases de OpenShift Origin se denominan Milestones, ya que son como puntos de control en un desarrollo que no acaba. Por su parte las versiones de OpenShift enterprise se rigen por el estándar Major version, Minor version, donde el número más significativo se incrementa cuando hay un cambio notable en la arquitectura de OpenShift. Hasta mediados de 2015 las versiones disponibles eran la M4 para Origin y la 2.2 para Enterprise, sin embargo a partir de ese momento se publica una nueva versión del software sustentada en los proyectos de Docker, Kubernetes y Atomic y se unifica la nomenclatura para las versiones Origin y Enterprise: OpenShift 3.

Modalidades de pago de OpenShift Online

OpenShift Online, nos ofrece tres modalidades distintas de pago

 FREE No credit card required Great for testing and development Up to 3 small gears Upgrade whenever you like GET STARTED	 BRONZE From \$0 a month Great for teams and larger projects Up to 16 gears of any size Custom SSL & upgradable storage UPGRADE TO BRONZE	 SILVER From \$20 a month Great for production apps All the features of Bronze plus Red Hat professional support UPGRADE TO SILVER
---	---	--

Gratuito

Nos permite trabajar con hasta tres servidores virtuales de tamaño pequeño y 1Gb de espacio en disco.

Bronze

Podemos trabajar con hasta 16 servidores, tres gratuitos de tamaño pequeño y los siguientes de cualquier tamaño. A partir del cuarto se generan costes según el uso.

Silver

Ofrece las mismas características que la modalidad bronze, pero además pagamos el soporte técnico por parte de Red Hat.

Como ya hemos dicho, para proyectos pequeños podemos trabajar con la modalidad gratuita pero con las siguientes limitaciones:

- Número reducido de servidores, y siempre de tamaño pequeño, lo que limita el software y la carga de trabajo que pueden gestionar.
- Sólo se puede crear un único dominio en el que desplegar las aplicaciones..
- Transcurridas 24h sin actividad, las aplicaciones se "serializan" a disco para liberar recursos, con lo que el siguiente acceso puede ralentizarse notablemente.
- No es posible trabajar con certificados digitales propios (como ocurre si queremos utilizar nuestro propio dominio), tan sólo se puede utilizar uno genérico de OpenShift.
- Espacio en disco limitado a 1 Gb.

Alternativas a OpenShift

Si estamos decididos a utilizar PaaS para un proyecto comercial deberíamos conocer qué otras opciones de PaaS tenemos y si alguna de ellas se ajusta mejor a nuestras necesidades. Las principales soluciones PaaS actuales son:

- *AWS Elastic Beanstalk* de Amazon, con soporte a aplicaciones Java, .NET, PHP, Node.js, Python, Ruby y Docker <http://aws.amazon.com/es/elasticbeanstalk/>
- *Heroku* , con soporte a Ruby, Node.js, Python, Java y PHP <https://www.heroku.com/features>
- *Cloud Foundry* de Pivotal, es una solución muy similar a OpenShift en cuando a planteamiento combinado Comercial/OpenSource. <http://cloudfoundry.org/>
- *BlueMix* de IBM, que es una implementación comercial de Cloud Foundry. <https://ace.ng.bluemix.net/>
- *Azure* de Microsoft, que incluye tanto IaaS como PaaS y soporta el despliegue de máquinas Linux. <http://azure.microsoft.com/en-us/documentation/articles/fundamentals-introduction-to-azure/>
- *Google App Engine* , la plataforma de PaaS más veterana (2008). Admite Python, Java , PHP y Go. <https://cloud.google.com/appengine>

5.10. Referencias

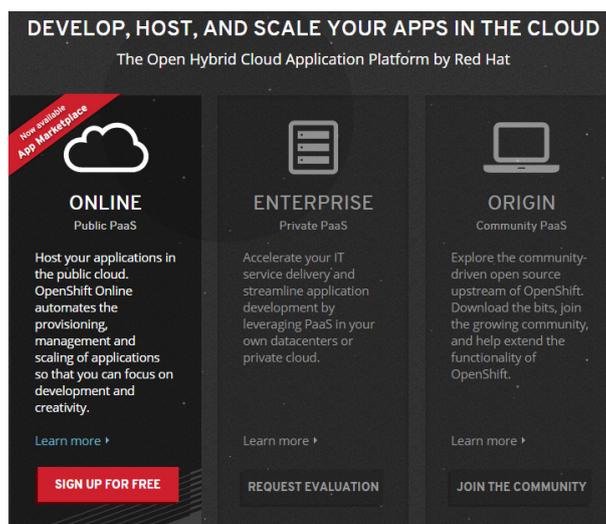
- Informe Ontsi Mayo 2014 http://www.ontsi.red.es/ontsi/sites/default/files/e-pyme_13_analisis_sectorial_de_implantacion_de_las_tic_en_la_pyme_espanola.pdf

- Proveedores de Cloud Computing en España <http://www.eurocloudspain.org/>
- Cómo elegir una plataforma PaaS <http://eugenedvorkin.com/choosing-a-cloud-platform-for-java-application/>
- Legislación Cloud http://www.agpd.es/portalwebAGPD/canaldocumentacion/publicaciones/common/Guias/GUIA_Cloud.pdf
- The NIST Definition of Cloud Computing <http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf>
- Información sobre la Protección de datos <http://cuidatusdatos.com/infoprotecciondedatos.html>
- Transferencia internacional de datos <https://www.ngncloud.com/blog/2013/10/lop-d-y-la-transferencia-internacional-de-datos-en-soluciones-cloud/>
- WordPress <http://es.wikipedia.org/wiki/WordPress>
- OpenStack: IaaS open source <https://www.openstack.org/>
- WordPress on OpenShift <http://code.tutsplus.com/tutorials/running-wordpress-on-openshift-an-introduction—cms-20058>
- Galería de aplicaciones desplegadas sobre OpenShift <https://www.openshift.com/application-gallery>

5.11. Ejercicios de Introducción a Cloud Computing

En esta sesión vamos a comenzar a trabajar con OpenShift Online, un producto de tipo nube pública de Red Hat. En las sesiones posteriores entraremos más en profundidad en sus características, pero ahora nos dedicaremos a configurar nuestra máquina virtual para trabajar con OpenShift y a realizar nuestros primeros despliegues en cloud.

Para trabajar con OpenShift hay que registrarse previamente accediendo a su página principal: <https://www.openshift.com/>



Haremos click en la versión ONLINE y rellenaremos el formulario de inscripción. Todos los accesos a OpenShift que hagamos desde la universidad se verán desde fuera como

provenientes de una misma dirección IP. Para evitar problemas de uso indebido de la plataforma Cloud, nos recomiendan que utilicemos la cuenta de correo del campus virtual en el formulario de registro:



A diferencia de otros proveedores, OpenShift no obliga a introducir los datos de una tarjeta de crédito para comenzar a trabajar, y la modalidad gratuita es suficiente para familiarizarse con el producto.

Una vez registrados se nos ofrece la posibilidad de crear nuestra primera aplicación en cloud, desde la consola web, o desde la línea de comandos, mediante una herramienta llamada *rhc*.

Por lo pronto instalaremos la herramienta en nuestra máquina virtual, con el siguiente comando:

```
$sudo apt-get update
$sudo apt-get install ruby
$sudo gem install rhc
```



Nota para Ubuntu 14.04

La versión integrada de OpenSSL es relativamente antigua con lo que cada vez que ejecutemos el comando *rhc* nos mostrará un aviso indicando que no se está utilizando los certificados de seguridad más potentes: *RSA 1024 bit CA certificates are loaded due to old openssl compatibility*

A continuación configuraremos la herramienta *rhc* para trabajar contra OpenShift Online:

```
rhc setup [--clean]
```

```
expertojava@expertojava-VirtualBox: ~
Archivo Edición Pestañas Ayuda
This wizard will help you upload your SSH keys, set your application namespace,
and check that other programs like Git are properly installed.

If you have your own OpenShift server, you can specify it now. Just hit enter to
use the server for OpenShift Online: openshift.redhat.com.
Enter the server hostname: |openshift.redhat.com|

You can add more servers later using 'rhc server'.

Login to openshift.redhat.com: martinez@psudl.com
Password: *****

OpenShift can create and store a token on disk which allows to you to access the
server without using your password. The key is stored in your home directory and
should be kept secret. You can delete the key at any time by running 'rhc
logout'.
Generate a token now? (yes|no) yes
Generating an authorization token for this client ... lasts about 1 month

Saving configuration to /home/expertojava/.openshift/express.conf ... done

No SSH keys were found. We will generate a pair of keys for you.

Created: /home/expertojava/.ssh/id_rsa.pub

Your public SSH key must be uploaded to the OpenShift server to access code.
Upload now? (yes|no)
yes

Since you do not have any keys associated with your OpenShift account, your new
key will be uploaded as the 'default' key.

Uploading key 'default' ... done

Checking for git ... found git version 1.9.1

Checking common problems .. done

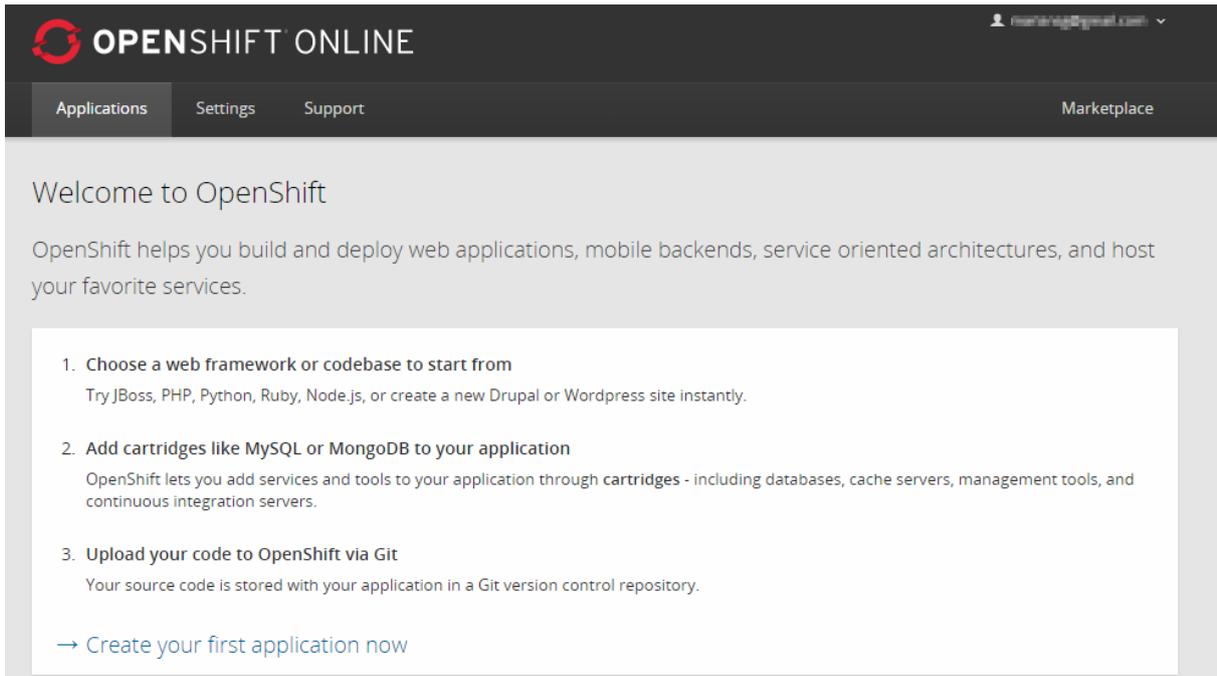
Checking for a domain ... jlzamora

Checking for applications ... found 1

hellocloud http://hellocloud-jlzamora.rhcloud.com/

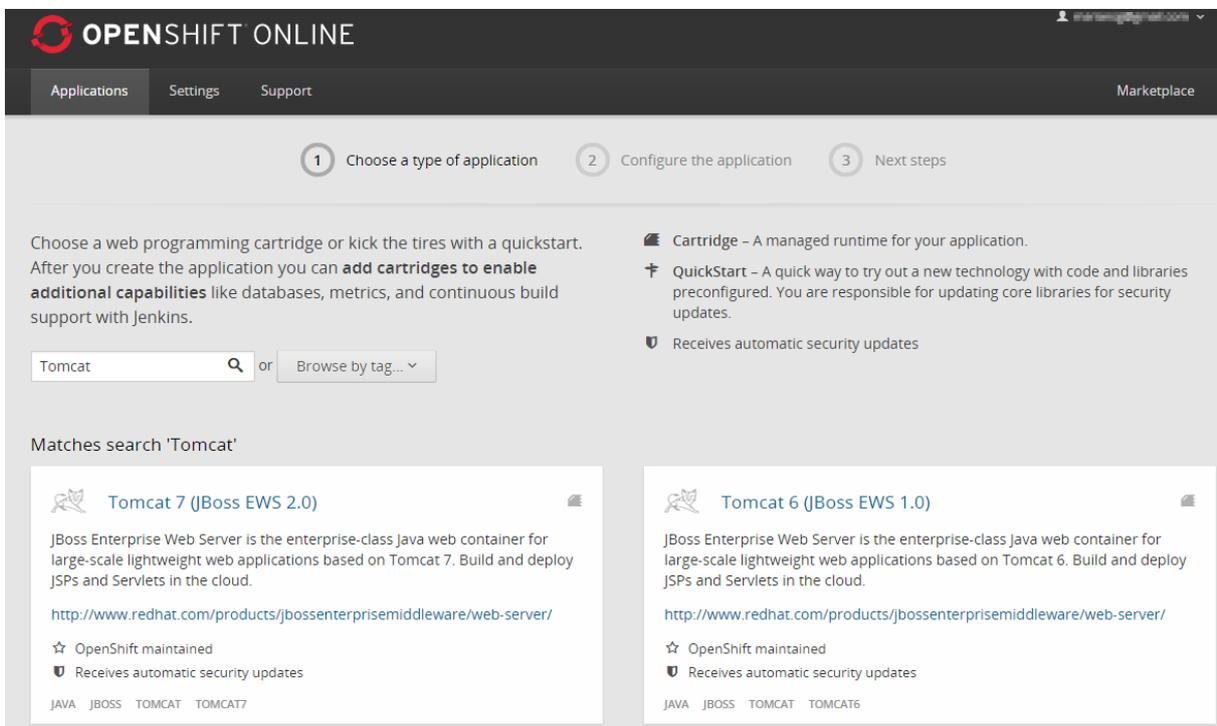
You are using 1 of 3 total gears
The following gear sizes are available to you: small
```

Mediante esta herramienta podremos realizar casi cualquier tarea contra OpenShift, pero en esta sesión trabajaremos desde la consola web:



Despliegue en Cloud de una aplicación web (1 punto)

Vamos a comenzar por crear una aplicación web, que desplegaremos sobre un servidor Tomcat. Este sería uno de los despliegues más sencillos que podemos hacer en OpenShift. Lo haremos desde la consola web, haciendo click en la pestaña *applications* donde seleccionaremos la opción de Tomcat 7:



En la siguiente pantalla, se nos pide información adicional. Nos interesa en este caso asignar un nombre a nuestra aplicación, a la que llamaremos *hellocloud* y un namespace, que agrupará a todas las aplicaciones que vayamos a desplegar en OpenShift. Para esta aplicación, y a partir de ahora utilizaremos **nuestro login** del curso.

Public URL	<input type="text" value="http://"/>	<input type="text" value="hellocloud"/>	-	<input type="text" value="jlzamora"/>	<input type="text" value=".rhcloud.com"/>
------------	--------------------------------------	---	---	---------------------------------------	---

Because this is your first application, you need to provide a domain under which your applications will be grouped

Sin más, pulsaremos en el botón de crear aplicación, y tras unos segundos de espera, habremos creado y desplegado nuestra aplicación web en la nube de RedHat. Lo primero que se nos muestra es información de cómo podemos acceder al código fuente de la aplicación para comenzar a trabajar:

Your application has been created. [Continue to the application overview page.](#)

Making code changes

Install the Git client for your operating system, and from your command line run

```
git clone ssh://5463b908ecb8d47a4490079d@hellocloud-jlzamora.rhcloud.com
~/git/hellocloud.git/
cd hellocloud/
```

This will create a folder with the source code of your application. After making a change, **add**, **commit**, and **push** your changes.

```
git add .
git commit -m 'My changes'
git push
```

When you push changes the OpenShift server will report back its status on deploying your code. The server will run any of your configured [deploy hooks](#) and then restart the application.

Manage your app

The console is convenient, but if you need deeper control try our other client tools

Command-Line

All of the capabilities of OpenShift are exposed through our command line tool, **rhc**. [Follow these steps to install the client](#) on Linux, Mac OS X, or Windows.

After installing the RHC read more on [how to manage your application from the command line](#) in our User Guide.

JBoss Developer Studio

The JBoss Developer Studio is a full featured IDE with OpenShift integration built in. It gives you the ability to create, edit and deploy applications without having to leave the IDE. Links to download, install and use the JBoss Developer Studio for Linux, Mac OS X, or Windows can be found on the [JBoss Developer Studio tools page](#).

Para acceder a ella, basta con abrir en un navegador la url: <http://<aplicacion>-<namespace>.rhcloud.com/>

Importar el proyecto desde IntelliJ y modificar la aplicación (0.4 puntos)

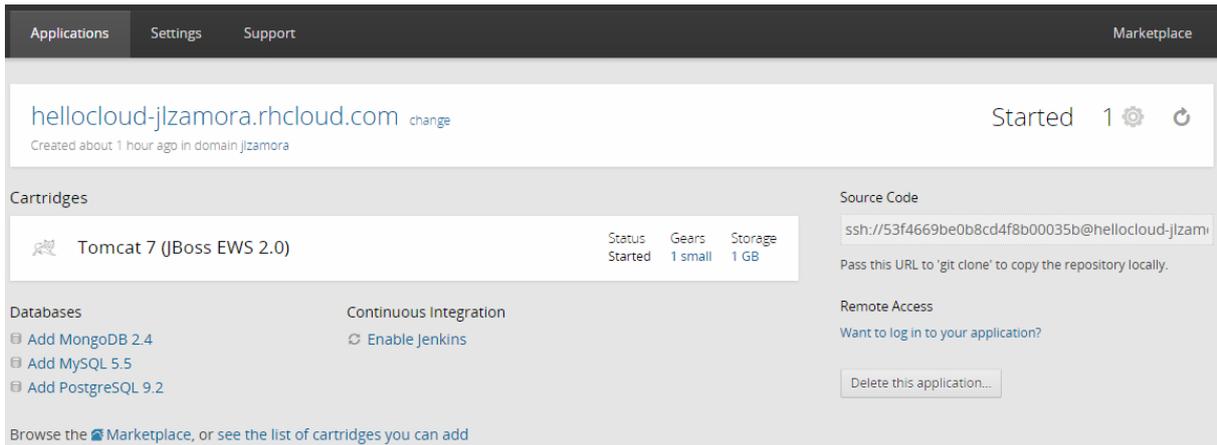
Si ejecutamos la aplicación web, veremos que se abre una aplicación plantilla que nos vuelve a dar información sobre cómo podemos modificar el código de la aplicación. Para cada aplicación en OpenShift, se crea un repositorio Git con las fuentes y lo que tenemos que hacer es clonar dicho repositorio:

```
git clone <git_url> <directory_to_create>

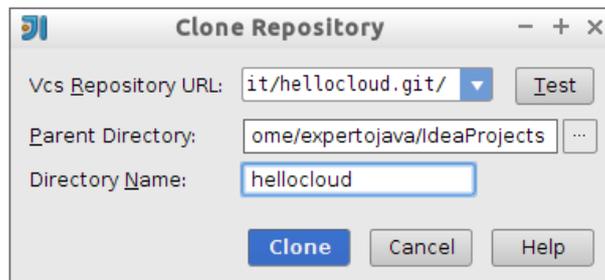
# Within your project directory
# Commit your changes and push to OpenShift

$ git commit -a -m 'Some commit message'
$ git push
```

Para saber la URL del repositorio podemos volver a la consola web, y dentro de *Applications* hacer click sobre la nueva aplicación:



A continuación ya podremos abrir el proyecto desde nuestro IDE y comenzar a trabajar. **Alternativamente** podemos utilizar la función de IntelliJ de abrir un proyecto desde un repositorio de fuentes, mediante el comando *Checkout from Version Control*.

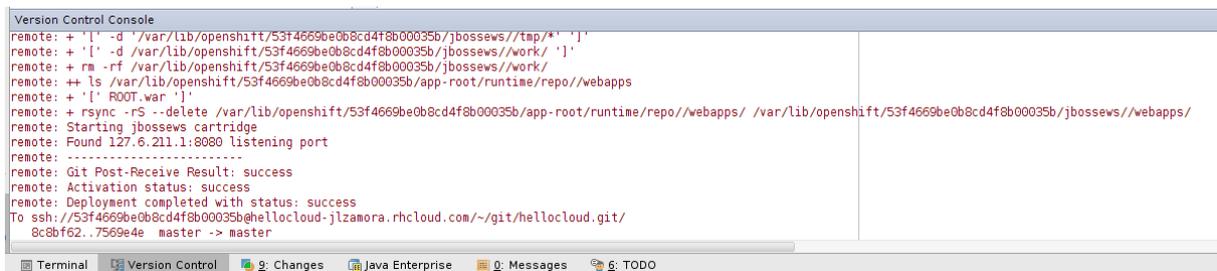


Ahora, una vez tenemos ya el código fuente en el IDE, debéis crear un sencillo formulario en la página inicial que nos pida un nombre de usuario y llame a un servlet, denominado *HelloServlet*. Este servlet atenderá a las peticiones que lleguen al contexto */hello* y simplemente nos mostrará un mensaje de bienvenida incluyendo el usuario que nos ha enviado el formulario. Lo crearemos en el paquete *org.expertojava.paas.servlets*.



Para crear el servlet lo más sencillo es crearlo directamente desde el menú de IntelliJ y anotarlo con `@WebServlet("contexto del servlet")`

Una vez tengamos el código modificado, debéis hacer un *commit* de los cambios y posteriormente un *Push* al repositorio de OpenShift. Si observáis en la ventana *Version Control* de IntelliJ podréis ver como el comando *Push* de Git ya encapsula todos los comandos necesarios para subir el código al servidor, compilar y redespigar la aplicación:



Despliegue de un servicio de almacenamiento en Cloud (0.4 puntos)

OpenShift aporta plantillas (Quickstarts) para proyectos muy diversos. Para empezar vamos a utilizar la plantilla de OwnCloud para crear una aplicación OpenShift con el mismo nombre.

OwnCloud es un software de almacenamiento en la nube en la línea de los servicios de DropBox o Google Drive aunque más rudimentario. Dispone además de herramientas cliente de sincronización para cualquier sistema operativo, y también permite montar directamente una unidad con su contenido mediante el protocolo WebDav.

Creación de un blog basado en WordPress (0.4 puntos)

Por último, vamos a utilizar OpenShift para crear un blog sobre Cloud Computing utilizando WordPress. Aunque WordPress ya ofrece la creación de blogs dentro de su dominio, estos se pueden gestionar de una forma muy limitada. La ventaja de utilizar Openshift es que nos va a facilitar toda la puesta en marcha del blog en cuanto a cuestiones técnicas, pero mantendremos las mismas posibilidades de personalización que una instalación en un servidor propio.



WordPress es una de las herramientas más utilizadas para la creación de blogs y cuenta con un amplio catálogo de plugins.

En este ejercicio hay que crear un blog sobre tecnologías Cloud (aunque si tenéis especial interés en otro tema, adelante), personalizar la apariencia del blog y añadir al menos un Post. Para ello debéis crear en OpenShift una nueva aplicación denominada *blog* mediante el Quickstart WordPress 4 e inicializarla.

Una vez hecho esto, debéis clonar el repositorio Git y crear un proyecto IntelliJ a partir de estos fuentes. Como veremos más adelante, la forma de aplicar cambios es siempre desde el repositorio Git al Gear y en este caso estamos cambiando la configuración directamente sobre el gear, con lo que nos encontraremos con que el repositorio no contendrá ninguna imagen o plugin que hayamos instalado. Queda fuera del alcance de esta sesión el conocer a fondo WordPress pero sí comentar que debéis utilizar la herramienta de *export* para extraer la información de los post y configuración básica del foro.

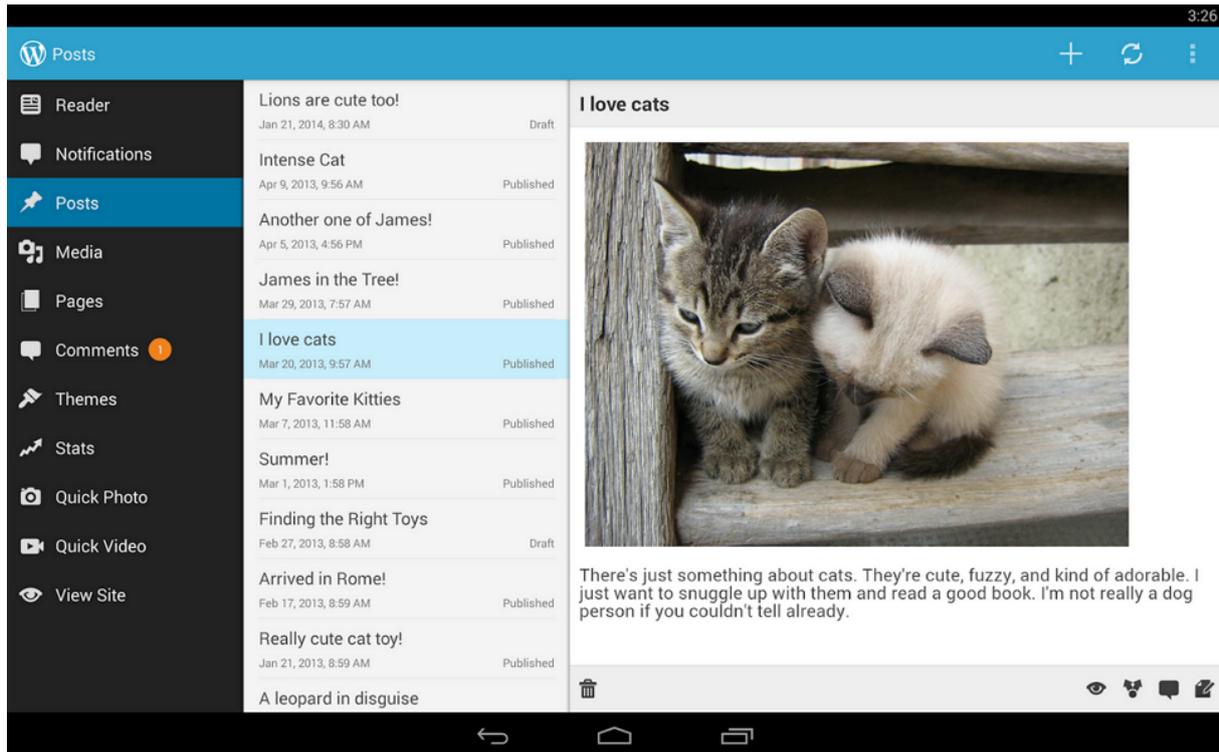


En el apartado de bibliografía tenéis un enlace en el que se explica cómo desplegar WordPress en OpenShift. No es necesario seguir todos esos pasos para completar el ejercicio pero sí os puede servir para profundizar en el tema si os interesa.

Una vez creado el blog, escribid algún post a modo de inauguración.



Existe una aplicación de móvil para WordPress que podéis configurar para administrar y escribir en vuestro blog desde cualquier lugar



Entrega

En esta sesión debeis entregar las copias de los tres repositorios, excluyendo la carpeta .git para no ocupar demasiado espacio, así como una exportación XML del contenido de vuestro blog.

6. Plataforma como Servicio

6.1. Conceptos propios de OpenShift

A nivel de infraestructura, OpenShift se apoya en dos unidades funcionales básicas: el *broker* y los *Node Servers*

Broker

Este elemento es el punto de entrada para todas las tareas de administración. Es el encargado de administrar los login, la resolución DNS, el estado de las aplicaciones y la orquestación general. Los desarrolladores no trabajan directamente contra este elemento, si no que se invoca indirectamente a través de las herramientas de administración, que veremos a continuación. El broker utiliza un frameworks de orquestación de procesos en paralelo, llamado *MCollective* para enviar instrucciones a los node servers.

Node Servers

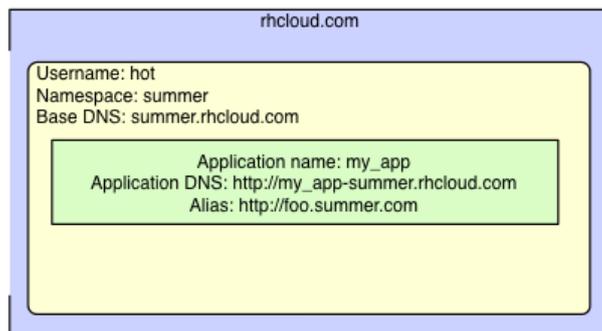
Son las máquinas host que ejecutan tanto los Cartridges como los Gear por tanto serán los encargados de alojar y ejecutar nuestras aplicaciones. Los Node Server también utilizan un cliente MCollective para recibir las instrucciones del broker.



La comunicación entre los clientes MCollective del broker y los node server se realiza asíncronamente, mediante un sistema de cola de mensajes.

Application

Será la aplicación propiamente dicha que despleguemos en la nube. En principio OpenShift está orientado hacia aplicaciones Web (en distintos lenguajes) por lo que los puertos a través de los cuales se podrá acceder a ellas son los típicos de HTTP (80), HTTPS(443) y SSH (22). Adicionalmente se ofrece soporte beta a conectividad basada en WebSockets, a través de los puertos 8000 y 8443.



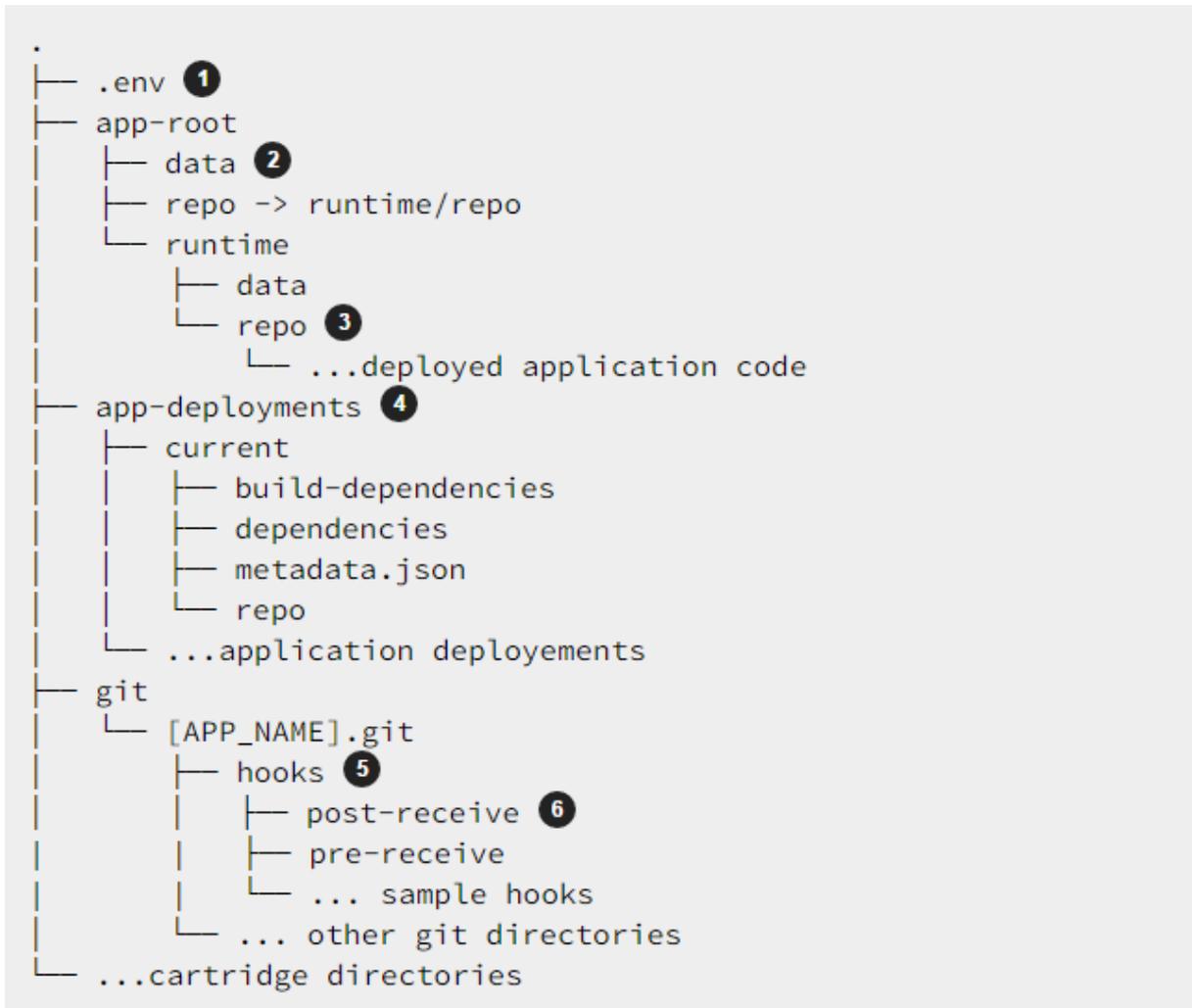
Las direcciones de las aplicaciones seguirán el siguiente patrón:

https://[APPNAME] - [DOMAIN] . rhcloud . com

Gear

Un gear es una partición de un Node server (CPU, memoria y almacenamiento), que proporciona los recursos necesarios para ejecutar las aplicaciones. Se dividen en tres categorías: small (512Mb RAM), medium (1Gb) y large (2Gb). Para pequeñas aplicaciones suele ser suficiente el gear más limitado, sin embargo hay configuraciones concretas que

recomiendan al menos 1Gb de RAM, como en el caso de querer trabajar con JBoss EAP. Cada Gear tendra una estructura de directorios como la siguiente:

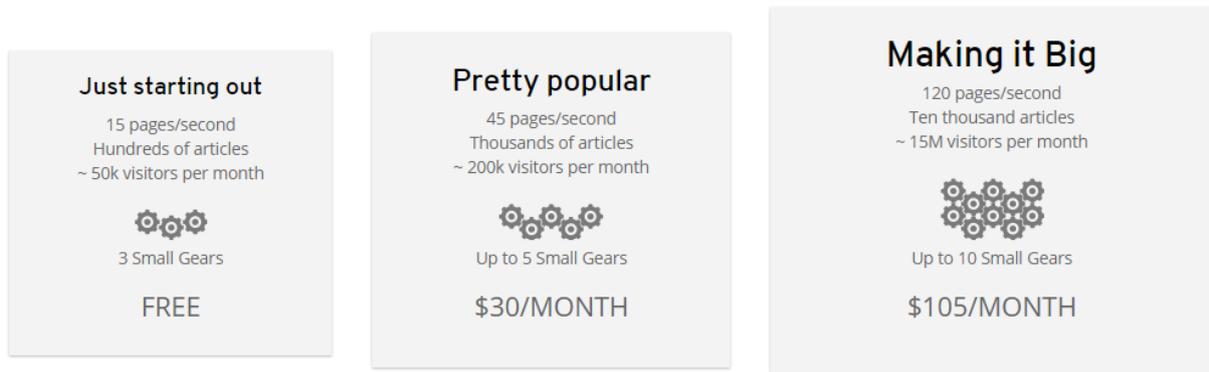


- ❶ Almacenamiento de variables de entorno
- ❷ Directorio de datos persistentes `$OPENSHIFT_DATA_DIR`
- ❸ Directorio del repositorio `$OPENSHIFT_REPO_DIR`
- ❹ Directorio de despliegues de aplicaciones
- ❺ Directorio de hooks (sólo lectura)
- ❻ El post-receive hook invoca la compilación y despliegue de las aplicaciones



Si nuestra aplicación supera el límite de memoria impuesto al gear, OpenShift matará automáticamente el proceso y lo relanzará.

Técnicamente, para aislar los procesos del Gear se utiliza Security-Enhanced Linux (SELinux) y para establecer las cuotas de memoria, cpu y disco se utiliza Control Groups (Cgroups). No se indica explícitamente la CPU y de E/S permitida para cada gear, pero Red Hat muestra estas cifras de ejemplo:



Recientemente se ha anunciado una nueva configuración, denominada *Small.highcpu* que mantiene el tamaño máximo de memoria pero duplica la capacidad de proceso. También es posible contratar máquinas dedicadas privadas.

Cartridge

Son los recursos que necesitan las aplicaciones para poder ejecutarse dentro de un gear. La idea es *paquetizar* los distintos elementos (servidor de aplicaciones, base de datos, proxy, etc.) en cajas negras y permitir que el desarrollador pueda "insertar" aquellos *cartuchos* que considere necesarios dentro del gear. En definitiva una forma de abstraer la gestión de la infraestructura y es lo que lo diferencia de un modelo IaaS, si bien como veremos luego tendremos cierta libertad de hacer cambios en estos cartuchos. Hay dos categorías de cartridges:

- **Standalone:** Son los runtime de los lenguajes o bien servidores de aplicaciones y los principales responsables de poder desplegar una aplicación. Ejemplo: JBoss, Tomcat, Python o Node.js.
- **Embedded:** Proporcionan funcionalidades extendidas como almacenamiento en base de datos o ejecución de tareas planificadas (Cron), pero por sí mismos no son suficientes para ejecutar una aplicación.

Application Scaling

Es una funcionalidad de escalado automática que permite adecuar la capacidad de proceso a la carga de trabajo actual. Si está activada, OpenShift monitoriza el tráfico de entrada y la carga de trabajo de los gear actuales y puede añadir nuevos gears para atender un pico de trabajo y liberarlos cuando ya no sean necesarios. Mas adelante hablaremos de esta funcionalidad con más detalle.

Proxy Ports

Permiten a los Gear exponer servicios que estén en ejecución. Cada Gear permite exponer hasta 5 servicios distintos.

Herramientas de administración

OpenShift proporciona tres herramientas distintas:

- **RHC** una herramienta de administración en línea de comandos desarrollada en Ruby y portable a todas las plataformas. Veremos que al contrario de Jboss-cli, rhc si puede considerarse ser la herramienta principal de trabajo de administración.
- **API REST** Gran parte de la funcionalidad de RHC se expone también como un API Rest integrable en otras aplicaciones.
- **Consola de administración Web** que nos ofrece un entorno más amigable pero más limitada que RHC.

- **JBoss Tools** Se trata de una serie de plugins para Eclipse que integran en el IDE tanto el trabajo con JBoss/WildFly como el despliegue en cloud. Hay una versión específica de Eclipse llamada *JBoss Developer Studio* que ya lleva integrados estos plugins.

SSH

OpenShift permite el acceso remoto a los gear desde nuestras máquinas locales mediante una conexión segura SSH. Como parte del proceso de configuración de la herramienta RHC se generan dos elementos de seguridad:

- Un par de claves, pública y privada, asociadas a nuestra máquina. Sería algo parecido a la autenticación basada en certificados que estudiamos en la sesión 4 pero en este caso lo que se hace es subir a OpenShift la clave pública, y en el handshake de la conexión, el equipo envía un mensaje encriptado con la clave privada y OpenShift lo desencripta con la clave pública. Estos ficheros se almacenan en la carpeta `./ssh` del directorio home en los ficheros `id_rsa` e `id_rsa.pub`.
- Un token de autorización que será el que nos identifique en cada conexión evitando el tener que estar introduciendo usuario y clave para autenticarnos. Los token se almacenan en la carpeta `./openshift` del directorio HOME.

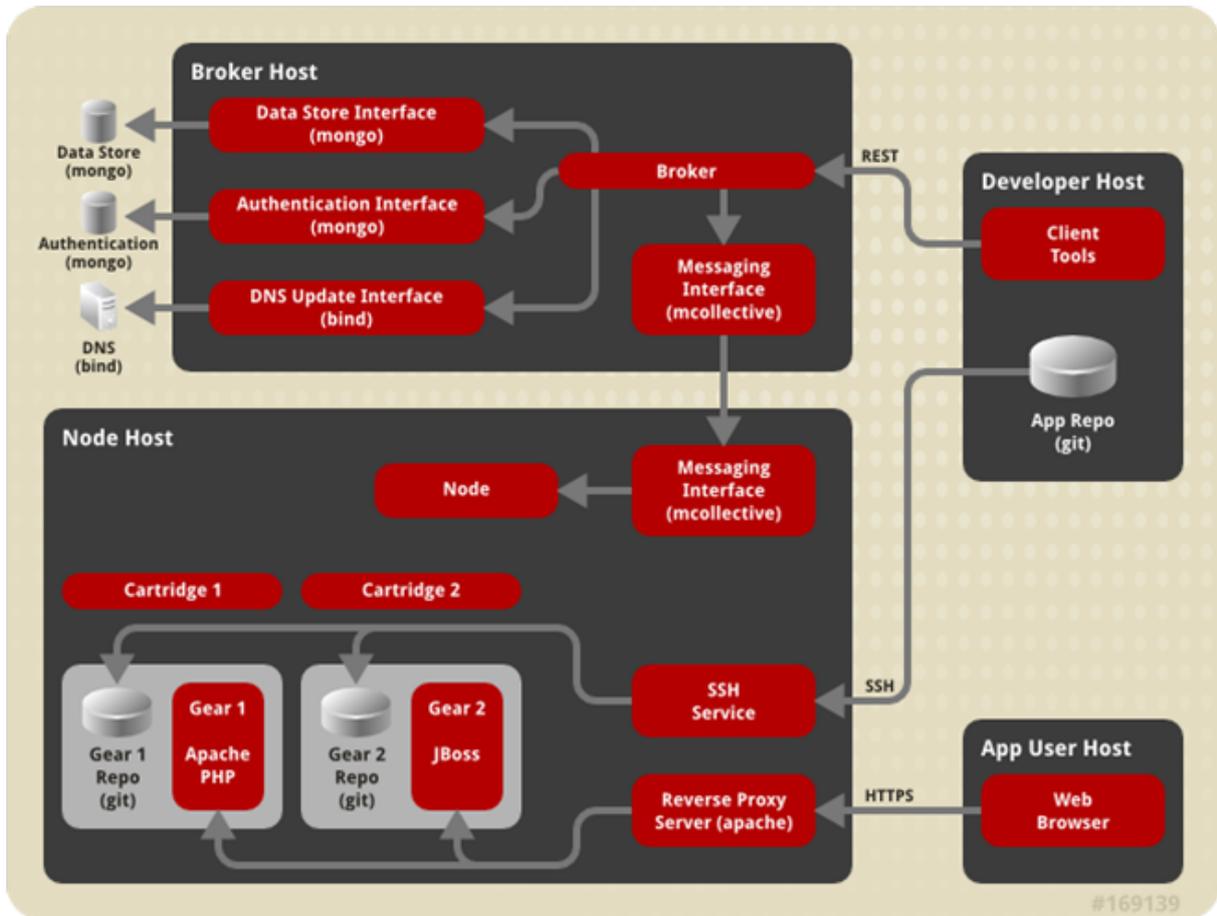
Dentro de una sesión SSH podremos acceder a los log de un gear, hacer cambios en la configuración e interactuar con los servicios desplegados pero no podremos realizar acciones que requieran ser superusuario. Mediante SSH también podremos activar una funcionalidad denominada *port-forwarding* que permite mapear puertos de nuestra máquina local con puertos del gear permitiendo realizar tareas de administración desde nuestra máquina local de forma transparente. Lo veremos con más detalle en la siguiente sesión.

Git

Git es la herramienta de control de versiones que utiliza OpenShift. Toda aplicación desplegada en OpenShift tiene un repositorio Git asociado que se crea automáticamente al dar de alta la aplicación. Este repositorio contendrá tanto los fuentes de la aplicación como los elementos de configuración y plugins asociados. OpenShift tiene la particularidad de que cuando hacemos *push* a este repositorio, no sólo subimos los ficheros sino que aplicamos cambios de configuración, recompilamos la aplicación y finalmente la redespelgamos, todo ello de forma automática.

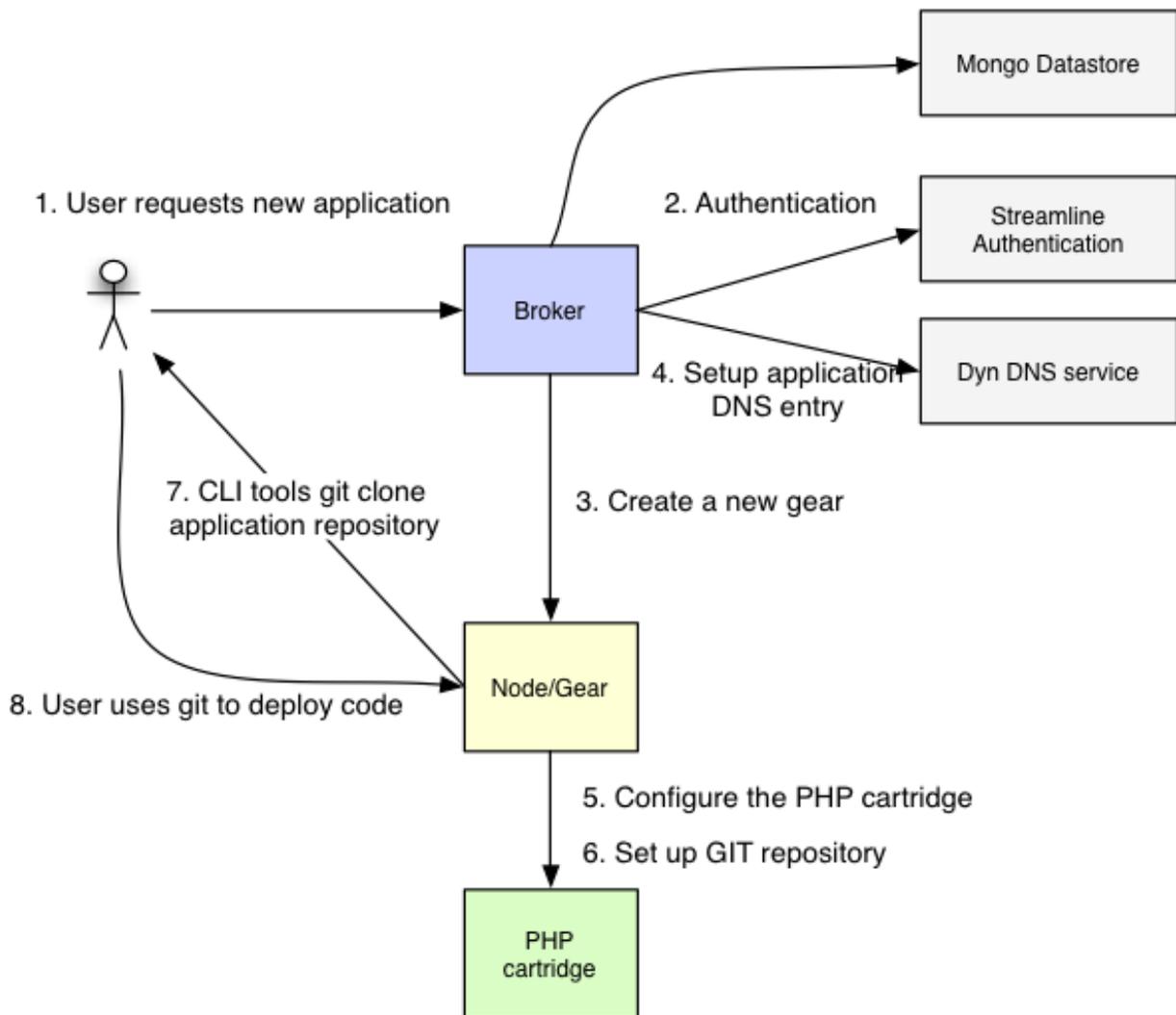
6.2. Cómo funciona OpenShift

Esta sería la representación de la arquitectura de OpenShift, en base a los elementos que ya conocemos:



Cuando un cliente realiza una petición HTTP a una aplicación, en primer lugar se ubica el node server en el que corre la aplicación, mediante la resolución DNS. Cada node server tiene un servidor apache con el módulo de proxy habilitado, y es quien encamina la petición al gear correspondiente, en función del nombre de la aplicación..

El siguiente gráfico describe el proceso de creación de una nueva aplicación:



Creación de aplicaciones

Vamos a crear una aplicación web mediante la herramienta *rhc*. Si ejecutamos *rhc* sin parámetros, se mostrará la ayuda y un listado con todas las opciones disponibles. A continuación enumeraremos las más comunes:

Listado de cartridges:

```
.....
rhc cartridge list
.....
```

```

expertojava@expertojava-VirtualBox: ~
Archivo Edición Pestañas Ayuda
jbossas-7          JBoss Application Server 7          web
jboss-dv-6.0.0 (!) JBoss Data Virtualization 6         web
jbosseap-6 (*)    JBoss Enterprise Application Platform 6 web
jenkins-1         Jenkins Server                       web
nodejs-0.10       Node.js 0.10                        web
perl-5.10         Perl 5.10                           web
php-5.3           PHP 5.3                             web
php-5.4           PHP 5.4                             web
zend-6.1          PHP 5.4 with Zend Server 6.1       web
python-2.6        Python 2.6                          web
python-2.7        Python 2.7                          web
python-3.3        Python 3.3                          web
ruby-1.8          Ruby 1.8                             web
ruby-1.9          Ruby 1.9                             web
ruby-2.0          Ruby 2.0                             web
jbosssews-1.0     Tomcat 6 (JBoss EWS 1.0)           web
jbosssews-2.0     Tomcat 7 (JBoss EWS 2.0)           web
jboss-vertx-2.1 (!) Vert.x 2.1                         web
diy-0.1           Do-It-Yourself 0.1                 web
l0gen-mms-agent-0.1 l0gen Mongo Monitoring Service Agent addon
cron-1.4          Cron 1.4                            addon
jenkins-client-1 Jenkins Client                       addon
mongodb-2.4       MongoDB 2.4                         addon
:...skipping...

```

La opción `cartridge` indica que vamos a ejecutar comandos relacionados con `cartridges` y `list` indica es la instrucción para que se muestren los `cartridges` disponibles. Una particularidad es que en la consola `web` muestra `cartridges` adicionales desarrollados por la comunidad y para los que Red Hat no ofrece soporte mientras que en la herramienta `rhc` no aparecen.



Es conveniente actualizar periódicamente las herramientas `rhc`, con el comando `gem update rhc`

Para crear un aplicación debemos tener claro en qué lenguaje vamos a trabajar y una idea de que `cartridges` vamos a necesitar. Con esta información ejecutaremos el siguiente comando:

Crear una aplicación

```
rhc app-create [nombre_app] [cartridge ...]
```

Ejemplo:

```
rhc app create miapp python-2.7
```

```

expertojava@expertojava-VirtualBox: ~/IdeaProjects
Archivo Edición Pestañas Ayuda
expertojava@expertojava-VirtualBox:~/IdeaProjects$ rhc app create miapp python-2.7
Application Options
-----
Domain:      jlzamora
Cartridges:  python-2.7
Gear Size:   default
Scaling:     no

Creating application 'miapp' ... done

Waiting for your DNS name to be available ... done

Clonar en «miapp»...
The authenticity of host 'miapp-jlzamora.rhcloud.com (50.16.41.21)' can't be established.
RSA key fingerprint is cf:ee:77:cb:0e:fc:02:d7:72:7e:ae:80:c0:90:88:a7.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'miapp-jlzamora.rhcloud.com,50.16.41.21' (RSA) to the list of known hosts.

Your application 'miapp' is now available.

URL:          http://miapp-jlzamora.rhcloud.com/
SSH to:       53ff9dca5973cab39f000664@miapp-jlzamora.rhcloud.com
Git remote:   ssh://53ff9dca5973cab39f000664@miapp-jlzamora.rhcloud.com/~/.git/miapp.git/
Cloned to:    /home/expertojava/IdeaProjects/miapp

Run 'rhc show-app miapp' for more details about your app.
expertojava@expertojava-VirtualBox:~/IdeaProjects$

```

Con esta instrucción crearemos un nuevo gear, un repositorio Git, un servidor Apache con el módulo `_mod_wsgi_` y una aplicación plantilla en Python desplegada. También se ejecutará un comando `git clone` y nos descargaremos el código fuente en una carpeta con el mismo nombre que la aplicación. ¡Fácil y rápido!

Este comando tiene parámetros adicionales que pueden ser de utilidad:

-n, --namespace NAME

Permite especificar el dominio de la aplicación.

-g, --gear-size SIZE

Permite especificar el tamaño del gear. Si no se especifica se utiliza el tamaño por defecto (small)

-s, --[no-]scaling

Habilita el Application Scaling

--from-app NAME

Permite crear una aplicación tomando otra como plantilla.

--from-code URL

Permite especificar un repositorio Git como origen del código fuente de la aplicación, en lugar de la plantilla.

--region

Zona geográfica donde se creará el gear: (aws-eu-west-1, aws-us-east-1)



Ubicación geográfica

Desde septiembre de 2014 es posible especificar la zona en la que queremos que se cree nuestra aplicación. Hasta esa fecha únicamente se

trabajaba sobre infraestructura de Amazon ubicada en Estados Unidos y a partir de ahora se puede trabajar sobre AWS en Irlanda. Hay que tener en cuenta que OpenShift sólo permite especificar la región si tenemos un plan de pago y vamos a utilizar un gear superior a *small*.

Si queremos crear una aplicación basada en un tipo de servidor que no esté en la lista de cartridges que muestra OpenShift tenemos dos opciones:

- Utilizar el cartridge *Do-It-Yourself* y configurar a mano un gear como si se tratase de una plataforma IaaS.
- Recurrir a los cartridges desarrollados por la comunidad de usuarios de OpenShift. De hecho, a la fecha de redacción de esta sesión WildFly 8 no tiene aún un cartridge soportado directamente por Red Hat.

Hay que tener claro a la hora de elegir un cartridge, el tipo de soporte que tendrá. La inmensa mayoría de cartridges mantenidos por OpenShift tienen soporte automático de parches de seguridad, con lo que podemos despreocuparnos de este aspecto. Si escogemos un cartridge desarrollado por terceros, debemos estar pendientes de aplicar estas actualizaciones. En un entorno de producción, sólo se deberían utilizar cartridges mantenidos por OpenShift, o en todo caso, generados por nosotros con el método genérico de Do-It-Yourself.

Para añadir un cartridge desarrollado por terceros se puede utilizar el siguiente comando

```
rhc app create <APP> <URL>

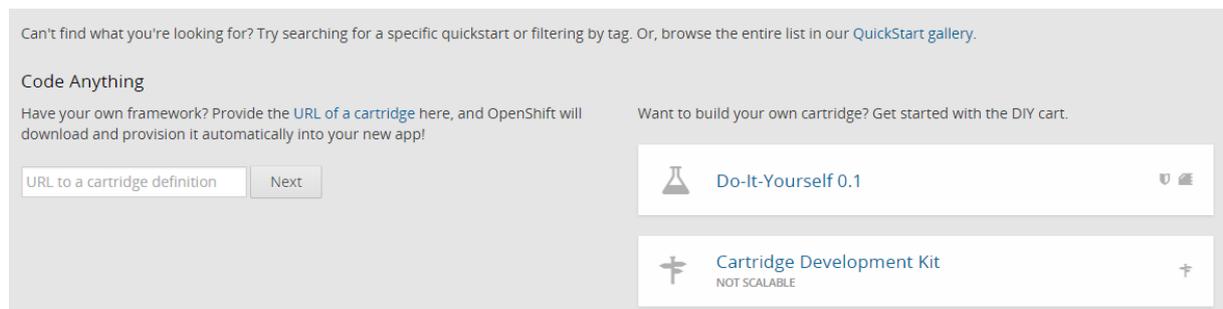
rhc wildfly8 create https://cartreflect-claytondev.rhcloud.com/reflect?
github=OpenShift-cartridges/OpenShift-wildfly-cartridge
```

El comando creará una aplicación llamada *wildfly8* y descargará el cartridge de la dirección indicada. Concretamente la URL tiene que apuntar a un fichero denominado *manifest.yml* que contiene la definición del cartridge.



Ya es posible crear una aplicación basada en WildFly sin necesidad de utilizar un cartridge externo a OpenShift. El nuevo cartridge se llama *jboss-wildfly-8*

Desde la consola web, el proceso de creación es muy similar, basta con elegir un tipo de aplicación y rellenar los datos que se solicitan. Alternativamente es posible indicar la URL de un cartridge descargable para crear una aplicación diferente a las enumeradas:



Otra característica propia de la consola web es que podemos crear una aplicación a partir de un *Quickstart*. Los quickstart tienen la apariencia de un cartridge normal, si bien son

combinaciones de cartridges ya predefinidas para desplegar un tipo de aplicación, además de una configuración de inicio para dicha aplicación. OpenShift tiene bastantes quickstarts con lo que merece la pena buscar uno que se adapte a nuestro proyecto.

Una vez creada la aplicación, normalmente se muestra por pantalla las claves de administración del nuevo servidor. No será habitual tener que acceder a la consola web, dado que la filosofía de OpenShift es desplegar y configurar la aplicación desde el repositorio Git, puede ser muy útil tener estas claves para hacer alguna prueba o un cambio puntual en una instancia que no queramos reiniciar.

```

✓ WildFly 8 administrator added. Please make note of these credentials:

Username: adminkDI2662
Password: _YBq@pmlk3ed

run 'rhc port-forward wildfly8 to access the web admin area on port 9990.
    
```

Añadir Cartridges de base de datos

Ya conocemos los cartridge que permiten crear aplicaciones pero ahora veremos qué cartridges podemos añadir que proporcionen funciones adicionales como almacenamiento de datos, ejecución de tareas programadas o monitorización.

La forma más sencilla de crear estos módulos es simplemente indicarlos en el momento de crear la aplicación.

```

rhc app create [nombre_app] [cartridge1] ... [cartridgen]
    
```

Aunque es perfectamente posible añadirlos o quitarlos a posteriori, con el comando:

```

rhc cartridge-add [cartridge] --app [nombre_app]
    
```

En el siguiente ejemplo, añadiremos una instancia de MySQL:

```

expertojava@expertojava-VirtualBox:~$ rhc cartridge-add mysql-5.5 --app wildfly8
Adding mysql-5.5 to application 'wildfly8' ... done

mysql-5.5 (MySQL 5.5)
-----
Gears:          Located with wildfly-wildfly-8
Connection URL: mysql://$OPENSHIFT_MYSQL_DB_HOST:$OPENSHIFT_MYSQL_DB_PORT/
Database Name:  wildfly8
Password:       HCpl2zXgS8Xh
Username:       adminGHuTbLH

MySQL 5.5 database added. Please make note of these credentials:

    Root User: adminGHuTbLH
    Root Password: HCpl2zXgS8Xh
    Database Name: wildfly8

Connection URL: mysql://$OPENSHIFT_MYSQL_DB_HOST:$OPENSHIFT_MYSQL_DB_PORT/

You can manage your new MySQL database by also embedding phpmyadmin.
The phpmyadmin username and password will be the same as the MySQL credentials above.
expertojava@expertojava-VirtualBox:~$
    
```

Si estamos trabajando desde la web, lo habitual es crear la aplicación y al seleccionarla, OpenShift nos ofrecerá directamente la posibilidad de añadir estos módulos.



Hay determinados cartridges que sólo se pueden incluir tras instalar uno específico, como por ejemplo, el módulo de administración de BD phpMyAdmin, que sólo se puede instalar si ya hemos incorporado una base de datos MySQL al proyecto.

Al añadir un nuevo cartridge de base de datos, tanto desde la consola web como mediante `rhc` se nos informa del usuario y password de administración de dicha base de datos. En este caso, no es necesario apuntarnos esta información porque siempre la podremos recuperar desde l/a consola web. Lo que sí que es muy recomendable es **reiniciar la aplicación** para que el gear cargue las variables de entorno necesarias para poder acceder a la nueva base de datos:



Creación de DataSources

Si examinamos la configuración del servidor, dentro del repositorio GIT:

```
.openshift/config/standalone.xml
```

Veremos que por defecto trae configurados tres datasources, uno para una base de datos en memoria, otro para MySQL y otro para PostgreSQL. Si hemos añadido una BD MySQL y hemos reiniciado la aplicación como se indicaba en el paso anterior, WildFly automáticamente activará su DataSource:

DATASOURCES		XA DATASOURCES	
		Add	Remove
		Disable	
Name	JNDI	Enabled?	
ExampleDS	java:jboss/datasources/ExampleDS	✓	
MySQLDS	java:jboss/datasources/MySQLDS	✓	
PostgreSQLDS	java:jboss/datasources/PostgreSQLDS	⊘	

[«](#) [<](#) 1-3 of 3 [>](#) [»](#)

Port forwarding

Repasando lo aprendido hasta ahora, hemos creado una aplicación Java, hemos añadido una base de datos y la hemos desplegado en un servidor. También sabemos que podemos modificar la configuración del servidor de aplicaciones y otros elementos haciendo cambios en los ficheros correspondientes del repositorio Git. Por ejemplo para definir un DataSource podríamos editar el fichero de configuración, creando uno nuevo a partir de los Datasources de ejemplo.

Pero ahora, al disponer de una base de datos necesitaremos trabajar directamente contra el servidor de base de datos, como por ejemplo para crear la BD de trabajo y sus datos. OpenShift permite hacer esto de varias formas, pero a continuación veremos un mecanismo que permite gestionar la configuración de los servicios desplegados en cloud, como si fueran servicios locales en la máquina del desarrollador.

El principal obstáculo que nos encontramos es que los puertos de administración **no** son visibles desde el exterior del servidor con lo que necesitamos algo que nos permita llegar a través de los puertos que sí son accesibles (https,ssh).

OpenShift proporciona para ello el comando **port-forward**. Se ejecuta especificando una aplicación:

```
rhc port-forward <app>
```

```
Terminal - expertojavajs@expertojavajs-VirtualBox: ~
Archivo  Editar  Ver  Terminal  Pestañas  Ayuda
expertojavajs@expertojavajs-VirtualBox:~$ rhc port-forward wildfly8
Checking available ports ... done
Forwarding ports ...
Address already in use - bind(2) while forwarding port 8080. Trying local port 8081
Address already in use - bind(2) while forwarding port 3306. Trying local port 3307

To connect to a service running on OpenShift, use the Local address

Service Local                OpenShift
-----
httpd   127.0.0.1:8080 => 127.9.73.131:8080
java    127.0.0.1:3528 => 127.9.73.129:3528
java    127.0.0.1:8081 => 127.9.73.129:8080
java    127.0.0.1:9990 => 127.9.73.129:9990
mysql   127.0.0.1:3307 => 127.9.73.130:3306

Press CTRL-C to terminate port forwarding
```

Se trata de una utilidad que busca puertos libres en nuestro equipo de desarrollo y enlaza con los puertos administrativos de los servicios que se ejecutan en un gear, todo esto a través de un canal ssh. Mientras el comando esté en ejecución podremos administrar los servicios del gear configurando las herramientas de administración para que se conecten al puerto correspondiente de *localhost*.

Modificación y despliegue de las aplicaciones

La estructura de un repositorio de una aplicación desplegada en OpenShift varía según la tecnología. En el caso de JBossAS 7 / WildFly es la siguiente:

<code>deployments/</code>	Location for built WARs (details below)
<code>src/</code>	Example Maven source structure
<code>pom.xml</code>	Example Maven build file
<code>.openshift/</code>	Location for OpenShift specific files
<code>config/</code>	location for configuration files such as <code>standalone.xml</code>
<code>action_hooks/</code>	See the Action Hooks documentation
<code>markers/</code>	See the Markers section below

<https://developers.OpenShift.com/en/jbossas-overview.html>

Si hemos creado el proyecto con `rhc` ya tendremos el proyecto en nuestro ordenador, pero si sólo hemos creado la aplicación desde la consola web, debemos descargarnos una copia del repositorio Git de OpenShift, podemos hacerlo de varias formas:

- Ejecutar `git clone` desde un terminal con la url que se muestra en las propiedades de la aplicación:

```
git clone ssh://5408af5a4382ece19c00020f@wildfly8-jlzamora.rhcloud.com/~/  
git/wildfly8.git/
```

- Utilizar el comando `git-clone` de `rhc`

```
rhc git-clone <app>
```



`rhc` añade información adicional al repositorio de modo que los siguientes comandos `rhc` que lancemos desde dentro de el automáticamente identifiquen la aplicación. Si hemos creado el repositorio con otras herramientas, será necesario especificar manualmente el nombre de la aplicación.

- Desde IntelliJ, con la opción *Check Out from Version Control*



Si la instancia está en estado *idle* por inactividad, es necesario acceder a la aplicación web para que se inicie porque si no el clonado del repositorio también fallará.

Una vez importado en IntelliJ podemos editar el fichero `pom.xml` del proyecto. En él, veremos una característica propia de OpenShift:

```
<profiles>  
  <profile>  
    <!-- When built in openshift the 'openshift' profile will be used  
    when invoking mvn. -->  
    <!-- Use this profile for any OpenShift specific customization your  
    app will need. -->  
    <!-- By default that is to put the resulting archive into the  
    'deployments' folder. -->
```

```

<!-- http://maven.apache.org/guides/mini/guide-building-for-
different-environments.html -->
<id>openshift</id>
<build>
  <finalName>wildfly8</finalName>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-war-plugin</artifactId>
      <version>2.4</version>
      <configuration>
        <failOnMissingWebXml>>false</failOnMissingWebXml>
        <outputDirectory>deployments</outputDirectory>
        <warName>ROOT</warName>
      </configuration>
    </plugin>
  </plugins>
</build>
</profile>
</profiles>

```

OpenShift se apoya en el sistema de perfiles de Maven para modificar las características del proyecto generado según estemos trabajando en nuestro equipo o estemos desplegando en la nube. La forma habitual de desplegar aplicaciones web en JBossAS7/WildFly consiste en copiar una aplicación empaquetada WAR/EAR a la carpeta `deployments` **del repositorio GIT**, con nombre de fichero `ROOT`.



Si el War/Ear tiene nombre `ROOT`, se podrá acceder a la aplicación con la URL <http://app-domain.rhcloud.com/mywebsite>. Si tuviera otro nombre, se accederá por el nombre específico: http://app_namespace.rhcloud.com/nombre_app.



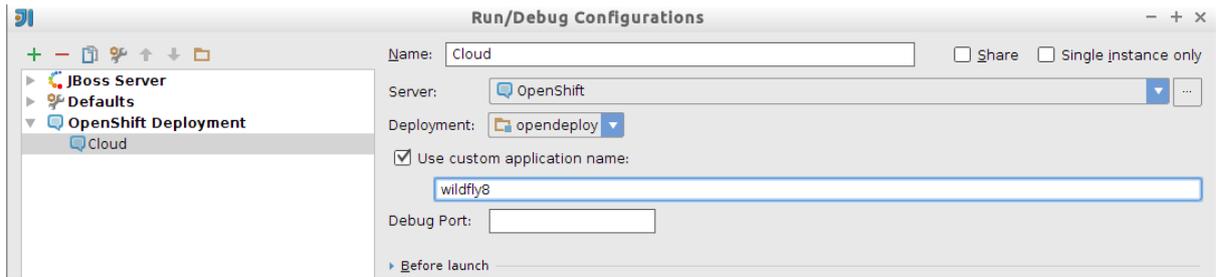
Los goals por defecto que indica OpenShift a la hora de construir una aplicación son “`clean package -Popenshift -DskipTests`”.

Una vez tenemos el proyecto importado podemos hacer los cambios que consideremos sobre él, manteniendo el perfil `OpenShift` y utilizar una instancia local de `WildFly` para hacer las pruebas que consideremos convenientes. Podemos proteger los cambios haciendo **commit** con la herramienta `git`, pero será cuando hagamos **push** cuando subamos los cambios al repositorio de `OpenShift` y desencadenemos la compilación y despliegue del proyecto en `cloud`.

Alternativamente en `IntelliJ` se puede definir la configuración de despliegue *OpenShift Deployment* de modo que para el IDE sea análogo a desplegar en cualquier otro servidor de aplicaciones. De hecho, podemos tener dos configuraciones, una *local* donde hacer nuestras pruebas y otra configuración, *cloud* para desplegar en `OpenShift`.

Una limitación de `IntelliJ 13.1` es que no detecta si una aplicación requiere `Java EE 7` y presupone que vamos a trabajar con `Java EE 6`. Por este motivo cuando desplaguemos una nueva aplicación en `cloud`, dicho plugin intentará crear un gear basado en `JbossAS7` y desplegar en él nuestra aplicación. La forma de salvar este comportamiento es la siguiente:

1. Crear una aplicación `Wildfly8` desde la consola web.
2. En `IntelliJ` desde la opción *Run#Edit configurations* crear un nuevo "OpenShift Deployment":



Es muy importante marcar *use custom application name* porque ahí vamos a indicar el nombre de la aplicación que hemos creado en el punto anterior. IntelliJ si detecta que ya existe el gear en lugar de crearlo trabajará contra su repositorio Git. El plugin se encargará de eliminar el contenido antiguo del repositorio y subir los ficheros específicos para hacer el despliegue de la nueva aplicación.

```
remote: Deploying WildFly
remote: Starting wildfly cart
remote: Found 127.7.4.129:8080 listening port
remote: Found 127.7.4.129:9990 listening port
remote: /var/lib/openshift/541433124382ec236900033e/wildfly/standalone/deployments /var/lib/openshift/541433124382ec236900033e/wildfly
remote: /var/lib/openshift/541433124382ec236900033e/wildfly
remote: CLIENT_MESSAGE: Artifacts deployed: ./ROOT.war
remote: -----
remote: Git Post-Receive Result: success
remote: Activation status: success
remote: Deployment completed with status: success
To ssh://541433124382ec236900033e@wildfly8-jlzamora.rhcloud.com:~/git/wildfly8.git/
85662d6..8ab1b6e master -> master
Application is available at http://wildfly8-jlzamora.rhcloud.com/
'opencloud' has been deployed successfully.
```

3. Añadir el profile "openshift" al fichero pom.xml, especificando como finalname el nombre de la aplicación (gear)
4. Ejecutar la aplicación seleccionando la configuración de IntelliJ que hemos creado y pulsando el botón "Play".



Si elimináis un gear y lo volvéis a crear con el mismo nombre, no os va a funcionar el despliegue desde IntelliJ porque las claves del gear cambian. Es necesario "parar" y "arrancar" OpenShift desde la vista de *Application Servers* para que se reenvíe esta información.

Consulta de Logs

Todas las aplicaciones suelen generar información de depuración para poder verificar su correcto funcionamiento, o tener información adicional acerca de un error que se esté produciendo. Para poder consultar el log de una aplicación existe un comando específico de RHC:

```
rhc tail <APP>
```

Con este comando se ejecutará en remoto el comando unix tail, y se visualizarán los logs de todos los cartridges definidos en el gear. En el caso de tener un servidor de aplicaciones y una base de datos, se devolvería la información de ambos. Admite los siguientes parámetros:

-f files

Permite especificar una o varias carpetas o ficheros a mostrará

-o 'opciones'

Permite pasar parámetros al comando tail, como por ejemplo el número de líneas a devolver

```
rhc tail wildfly8 -o '-n 100'
```

Además de consultar los logs existe un comando específico para limpiar los ficheros temporales y de logs de un gear tanto de git como de los servicios en ejecución:

```
rhc app tidy <APP>
```

Borrado de aplicaciones

Si queremos eliminar una aplicación de OpenShift, basta con acceder a la consola web, seleccionar la aplicación y pulsar en el botón *Delete this application...* Alternativamente, por línea de comandos:

```
rhc app delete <APP>
```



Si borramos una aplicación estamos liberando un gear con lo que destruiremos la aplicación desplegada, la configuración del servidor y el repositorio de código fuente con lo que esta opción siempre hay que usarla con precaución.

Gestión del dominio

Al comenzar a trabajar con OpenShift es necesario crear un dominio en el que comenzaremos a crear las aplicaciones. Si desde la consola web pulsamos en el nombre de un dominio, podemos acceder a la siguiente pantalla administrativa:

Desde esta pantalla podemos dar permisos sobre nuestro dominio, a otros usuarios de OpenShift y en distintas modalidades. Si por el contrario es a nuestro usuario al que se le otorgan permisos de acceso sobre otro dominio, veremos todos nuestros dominios más los autorizados desde la vista *Applications* junto con las aplicaciones que contengan.

Note: Los accesos en modo Edit o Administrator permiten modificar aspectos de las aplicaciones y acceder a sus repositorios de código fuente.

6.3. Marketplace

El Marketplace de OpenShift es un espacio de reciente creación donde se ofrece la posibilidad de que otras empresas publiquen herramientas o servicios integrables con la plataforma PaaS de Red Hat. El número de productos disponible hasta el momento es escaso, si bien los que hay son bastante interesantes y merece la pena revisarlos al completo. La inmensa mayoría de estos servicios se ofrecen en modalidad gratuita con una serie de limitaciones, de modo que nos sirvan para evaluarlos y si realmente estamos interesados en ellos, los contratemos.

Para acceder al Marketplace basta con acceder a la consola web de OpenShift y pulsar en el enlace de la parte superior derecha denominado *Addons* o acceder a la dirección:

<https://marketplace.openshift.com/openshift>

Los servicios se muestran clasificados por su tipología. Si nos interesa uno de ellos, en primer lugar hay que "comprarlo" y posteriormente añadirlo a las aplicaciones que queramos. Puede parecer similar a lo que ya conocemos de los cartridges pero la realidad es que cada producto se integra de una forma particular y escasamente documentada.

Por ejemplo: si queremos trabajar con una base de datos externa podemos contratar el servicio de ClearDB que nos proporciona una instancia MySQL de 10 Mb alojada en sus servidores. Una vez contratada, pulsaremos en el botón "Add to Apps" y seleccionaremos la aplicación o aplicaciones que queremos que utilicen este servicio. En el caso concreto de ClearDB se crean unas variables de entorno con los valores adecuados para crear un DataSource que apunte a la BD en la nube, pero es posible que algún servicio incorpore elementos adicionales a los gear. Para más información es conveniente revisar la documentación del Proveedor de servicios.

6.4. Monitorización

OpenShift hasta el momento no proporciona un cuadro de mando al estilo de otras plataformas como AWS, donde poder comprobar la carga de cada servidor, consumo de recursos o establecer alertas de monitorización. Debemos acudir a cartridges específicos o bien dado que trabajamos con componentes estándar, acudir a sus propias interfaces de monitorización.

Herramientas estándar de monitorización

snoop.jsp

Más que herramienta, se consideraría una página con información de monitorización que OpenShift incluye en sus plantillas de proyectos. En dicha página se muestra información del consumo de memoria de la JVM que proporciona el contenedor de servlets. Por defecto se trata de una página de acceso público. La forma usual de acceder es:

<http://<app>-<domain>.rhcloud.com/snoop.jsp>

port-forwarding y JMX

Ya sabemos que mediante el comando port-forward podemos acceder a servicios de OpenShift como si fueran servicios que se ejecutan en local. Gracias a esto, podemos acceder al puerto de administración de WildFly o cualquier otro servidor que se ejecute en un gear, desde nuestra máquina local y utilizar cualquier herramienta de monitorización basada en JMX para monitorizar el funcionamiento de la JVM y acceder a los MBeans del servidor.



Un MBean es un objeto de java muy similar a los componentes EJB, pero que cumplen la especificación del API JMX de monitorización. Un MBean puede representar un dispositivo, una aplicación desplegada en el servidor o cualquier recurso gestionado por el servidor de aplicaciones. Para cada uno de ellos, los MBean exponen atributos con su estado de sólo lectura, valores de configuración de lectura/escritura y/o métodos que permiten realizar tareas de administración.

Como ejemplo utilizaremos JVisualVm que es una herramienta perteneciente al JDK.

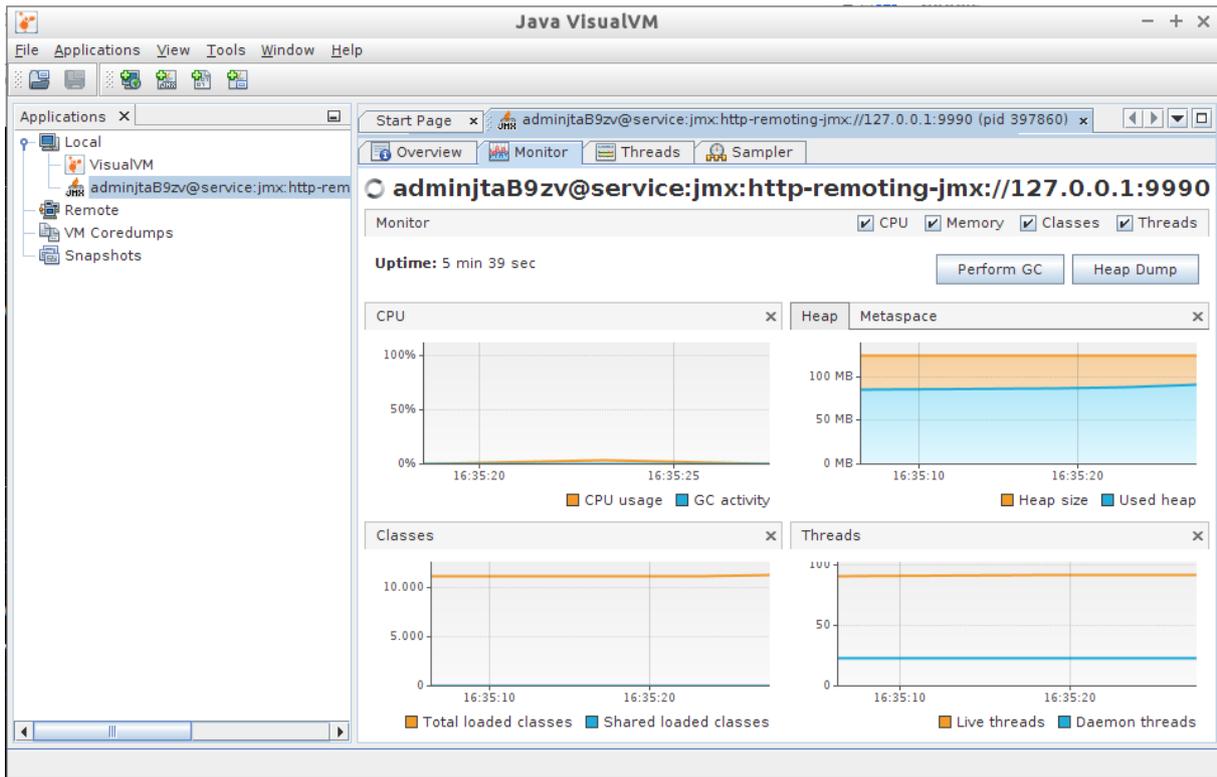
La ejecutaremos con el siguiente comando:

```
$ jvisualvm --cp:a /usr/local/wildfly/bin/client/jboss-cli-client.jar
```

Una vez iniciada la herramienta, añadiremos una nueva conexión JMX especificando la conexión al servidor a través de la IP local y el puerto redirigido al puerto de administración de WildFly, así como el usuario/password de administración de WildFly:



Una vez establecida la conexión ya podemos acceder a la información que se recoge en tiempo real del servidor:



En VisualVM es necesario añadir un plugin para poder acceder a la información de MBeans, desde la configuración de la propia herramienta.

Otra posibilidad es utilizar la herramienta cliente de WildFly conectándonos desde nuestra máquina. Dado que el mecanismo de port-forward intenta utilizar los mismos puertos en la máquina local y gear, bastaría con ejecutar CLI y especificar usuario y password de administración para empezar a trabajar.

En las referencias hay un enlace donde se explican algunos ejemplos de consultas que podemos hacer desde CLI.

Cartridges de terceros

Otra opción es buscar un módulo desarrollado por terceros que implemente funcionalidades de monitorización del gear, ya sea gratuito o de pago.

Como ejemplo de herramienta gratuita tenemos Monit. Monit presenta un panel muy sencillo en el que se muestra el estado de los servicios en ejecución del gear y el estado de las cuotas de memoria y almacenamiento.

Monit Service Manager					
Monit is running on ex-std-node459.prod.rhcloud.com with <i>uptime, 7m</i> and monitoring:					
System	Status	Load	CPU	Memory	Swap
ex-std-node459.prod.rhcloud.com	Running	[0.71] [0.62] [0.54]	6.4%us, 5.0%sy, 0.4%wa	72.4% [11086216 kB]	4.5% [2409620 kB]
Program	Status	Last started			Exit value
MEMORY_QUOTA	Status ok	Sun, 05 Oct 2014 16:05:42			68
STORAGE_QUOTA	Status ok	Sun, 05 Oct 2014 16:05:42			58
Host	Status				Protocol(s)
wildfly8-jizamora.rhcloud.com	Online with all services				[HTTP] at port 80

El módulo es OpenSource y su código fuente e instrucciones de instalación lo podéis encontrar en este enlace:

<https://raw.githubusercontent.com/openshift-cartridges/openshift-origin-cartridge-monit/master/metadata/manifest.yml>

Sus funciones por defecto son:

- Cuadro resumen del estado del gear, se accede a través de la dirección <https://<app>-<domain>.rhcloud.com/monit-status> y autentica mediante usuario/password.
- Alertas por mail cuando se alcance un 80% de ocupación de memoria o disco y acciones preventivas cuando se llegue al 90% (reiniciar la aplicación en el caso de un elevado consumo de memoria, y lanzar un comando "tidy" en el caso de que nos estemos quedando sin espacio.
- Alertas cuando haya caídas en algún servicio y si la caída dura más de 90 segundos, se intenta reiniciar la aplicación principal.

Estas reglas se pueden modificar y ampliar modificando el fichero `.monitrc`



La herramienta no está totalmente adaptada a OpenShift y lo podemos ver si intentamos pulsar en los enlaces de la pantalla de monitorización. Todos ellos están referidos al context-root del servidor cuando realmente en el context-root está nuestra aplicación. La herramienta funciona bien pero para acceder a las pantallas relacionadas hay que editar el link y añadir el contexto raíz de Monit, `monit-status`.

Monitorización de base de datos

Específicamente para tareas administrativas y de monitorización de bases de datos tenemos los siguientes cartridges:

- phpMyAdmin para MySQL.
- RockMongo, MongoDB Monitoring Service para MongoDB.

Estos módulos se pueden instalar en los gear que ya dispongan de una instancia del gestor de base de datos desplegada.



Figure 1. phpMyAdmin

6.5. Referencias

- OpenShift System Architecture Guide http://OpenShift.github.io/documentation/oo_system_architecture_guide.html
- OpenShift Quickstarts <https://www.OpenShift.com/quickstarts>
- Community Cartridges for OpenShift <https://github.com/OpenShift-cartridges>
- Portal de desarrolladores de OpenShift <https://developers.OpenShift.com/>
- Libro Getting started with OpenShift <https://www.OpenShift.com/promotions/ebook>
- Ejemplos: <https://github.com/codemiller/getting-started-with-OpenShift>
- API REST de administración https://access.redhat.com/documentation/en-US/OpenShift_Online/2.0/html/REST_API_Guide/index.html
- Monitorización de JBoss EAP/WildFly mediante CLI <http://blog.akquinet.de/2014/09/15/monitoring-the-jboss-eap-wildfly-application-server-with-the-command-line-interface-cli>
- Twitter de OpenShift Operations https://twitter.com/openshift_ops

6.6. Ejercicios de Plataforma como servicio

Vamos a comenzar a trabajar con OpenShift como plataforma sobre la que desplegar nuestras primeras aplicaciones. Como en la sesión anterior utilizamos los tres gear que tenemos disponibles para trabajar, debéis eliminar uno de ellos. Utilizad el comando `rhc` adecuado para borrar la aplicación `_OwnCloud_` de la sesión anterior.

Despliegue desde un repositorio externo (0.2 puntos)

Vamos a practicar cómo crear una aplicación OpenShift a partir de un código fuente situado en un repositorio externo. Tenéis que crear una nueva aplicación OpenShift de nombre *ejemplo* indicando de dónde se tienen que obtener los fuentes de la aplicación. En este caso trabajaréis con un repositorio en GitHub

<https://github.com/Djbyte1977/ejemplo-OpenShift.git>

Se pide:

1. Crear la aplicación *ejemplo* desde la consola Web de OpenShift.
2. Utilizar la herramienta RHC y crearla desde el terminal utilizando una sola instrucción.

Desarrollo mixto local/cloud (0.8 puntos)

Vamos a trabajar sobre la aplicación *InsultApp*. Esta aplicación es una versión de la aplicación con el mismo nombre publicada en el libro *Getting Started with OpenShift*, que podéis encontrar en las referencias. Se trata de un generador de "insultos" no demasiado ofensivos, que funciona en base a una base de datos de palabras. En las plantillas podéis encontrar el código fuente del proyecto en Java, así un fichero *import.sql* con la creación de las tablas que la aplicación necesita para funcionar.

Se pide:

1. Compilar y desplegar el proyecto en una instancia local de WildFly. Se ha de crear una base de datos, de nombre *InsultApp* y acceder a ella mediante un *DataSource* con el nombre indicado en la unidad de persistencia.
2. Crear un nuevo perfil de Maven en el proyecto para OpenShift.
3. Crear una nueva aplicación en OpenShift, de nombre *InsultApp*, integrar el código de *InsultApp* "local" y hacer los cambios necesarios para desplegar la aplicación en cloud.



Anotad el usuario y password de administración, lo necesitaréis para el último ejercicio.

Monitorización de una aplicación (0.2 puntos)

Como paso final, queremos añadir las herramientas de monitorización necesarias para asegurarnos de que el servidor de la aplicación *InsultApp* está funcionando correctamente. Se pide:

1. Monitorizar la JVM del servidor con *JVisualVM*
2. Añadir el cartridge *Monit* a la aplicación.

Entrega

En esta sesión hay que entregar un fichero TXT con el comando solicitado para el primer ejercicio, un breve resumen de los comandos ejecutados para el ejercicio de Monitorización, y una copia de los repositorios de las aplicaciones *ejemplo* y *InsultApp* :huge-size: scaledwidth="100%" :large-size: scaledwidth="66%" :normal-size: scaledwidth="50%" :small-size: scaledwidth="33%" :tiny-size: scaledwidth="25%"

7. Características IaaS+ de OpenShift

7.1. Introducción

En la sesión anterior adquirimos los conocimientos básicos para crear aplicaciones y desplegarlas en cloud de forma análoga a otros proveedores de PaaS. En esta sesión continuaremos aprendiendo características de Openshift que necesitaremos conocer si vamos a trabajar en un proyecto real y se acercan a la forma de trabajo de una plataforma IaaS+.

7.2. Configuración avanzada

SSH

OpenShift permite el acceso remoto a nuestros gear mediante una sesión SSH del mismo modo que accederíamos a una máquina física. Una vez estamos conectados a un gear podremos modificar su configuración, ejecutar comandos de administración o consultar los ficheros de log. Desde la sesión remota tenemos un control razonablemente completo de la parte de máquina que se nos asigna, pero sin embargo no podremos realizar cambios que requieran ser administrador.

Para poder iniciar una sesión remota tenemos dos posibilidades:

- Utilizar un cliente ssh, al que debemos llamar con una dirección específica que podemos obtener de la propia información de la aplicación, bien desde la consola web, o bien mediante rhc:

```
Remote Access
Want to log in to your application?
The command below will open a Secure Shell (SSH) session to
your application on most operating systems. See our SSH help
page for information about connecting with Windows, Mac, and
Linux computers.

ssh 541467525973cacb2c000e52@wildfly8-jlzamora
```

- Utilizar rhc como intermediario entre el cliente ssh y OpenShift. En este caso basta con ejecutar:

```
$ rhc ssh <app>
```



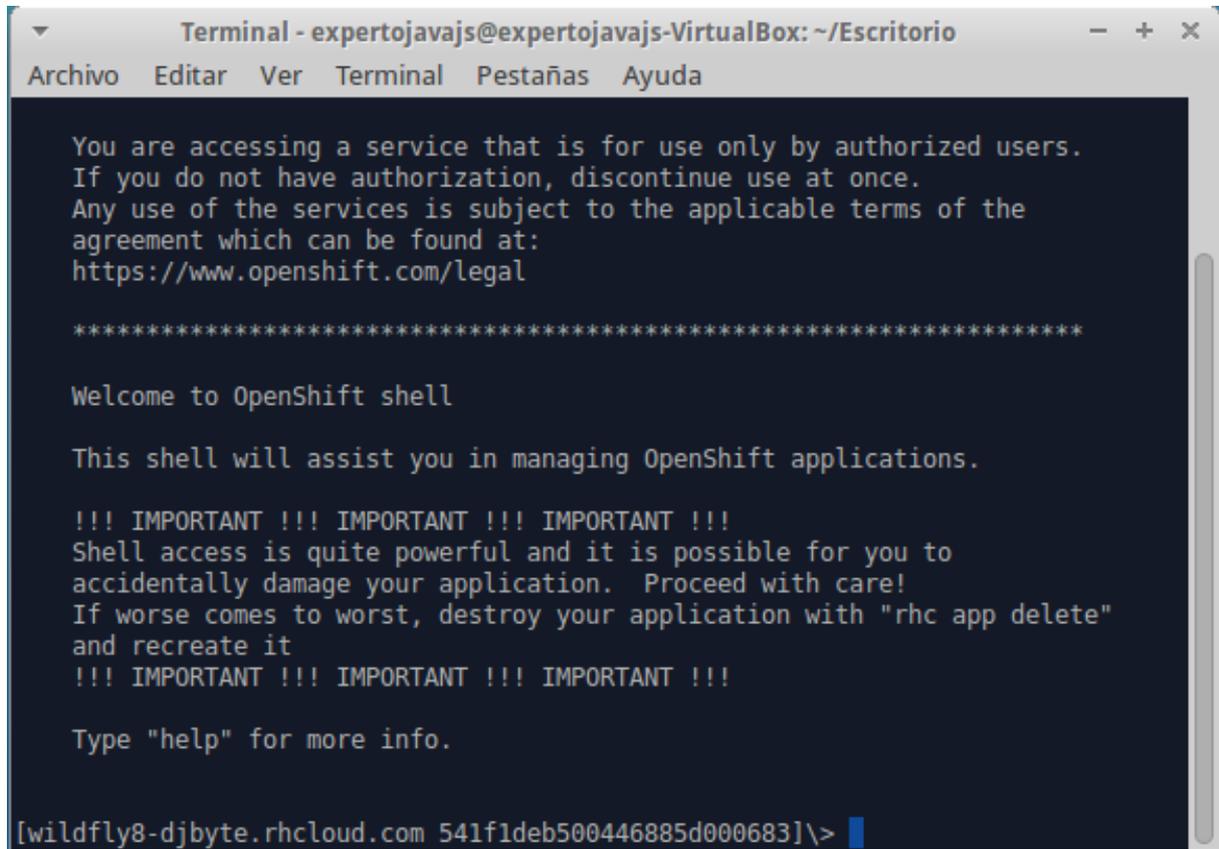
Conexión SSH desde Windows con Putty

Windows no incorpora ningún cliente de ssh de serie así que hay que recurrir a una aplicación externa. Una de las más extendidas es Putty pero para que emplea una codificación distinta para las claves con lo que es necesario convertir la clave privada generada por openshift, del formato OpenSSH al formato propio de Putty. Esto se puede hacer con la herramienta puttygen que incluye. Una vez convertida, hay que especificar la nueva clave privada en la propiedad *Auth* de la opción

ssh. Más información en: <https://developers.openshift.com/en/managing-remote-connection.html>

Cambios en la configuración de la aplicación.

Una vez iniciemos la sesión en un gear, veremos un mensaje de bienvenida similar a este:



```
Terminal - expertojavajs@expertojavajs-VirtualBox: ~/Escritorio
Archivo  Editar  Ver  Terminal  Pestañas  Ayuda

You are accessing a service that is for use only by authorized users.
If you do not have authorization, discontinue use at once.
Any use of the services is subject to the applicable terms of the
agreement which can be found at:
https://www.openshift.com/legal

*****

Welcome to OpenShift shell

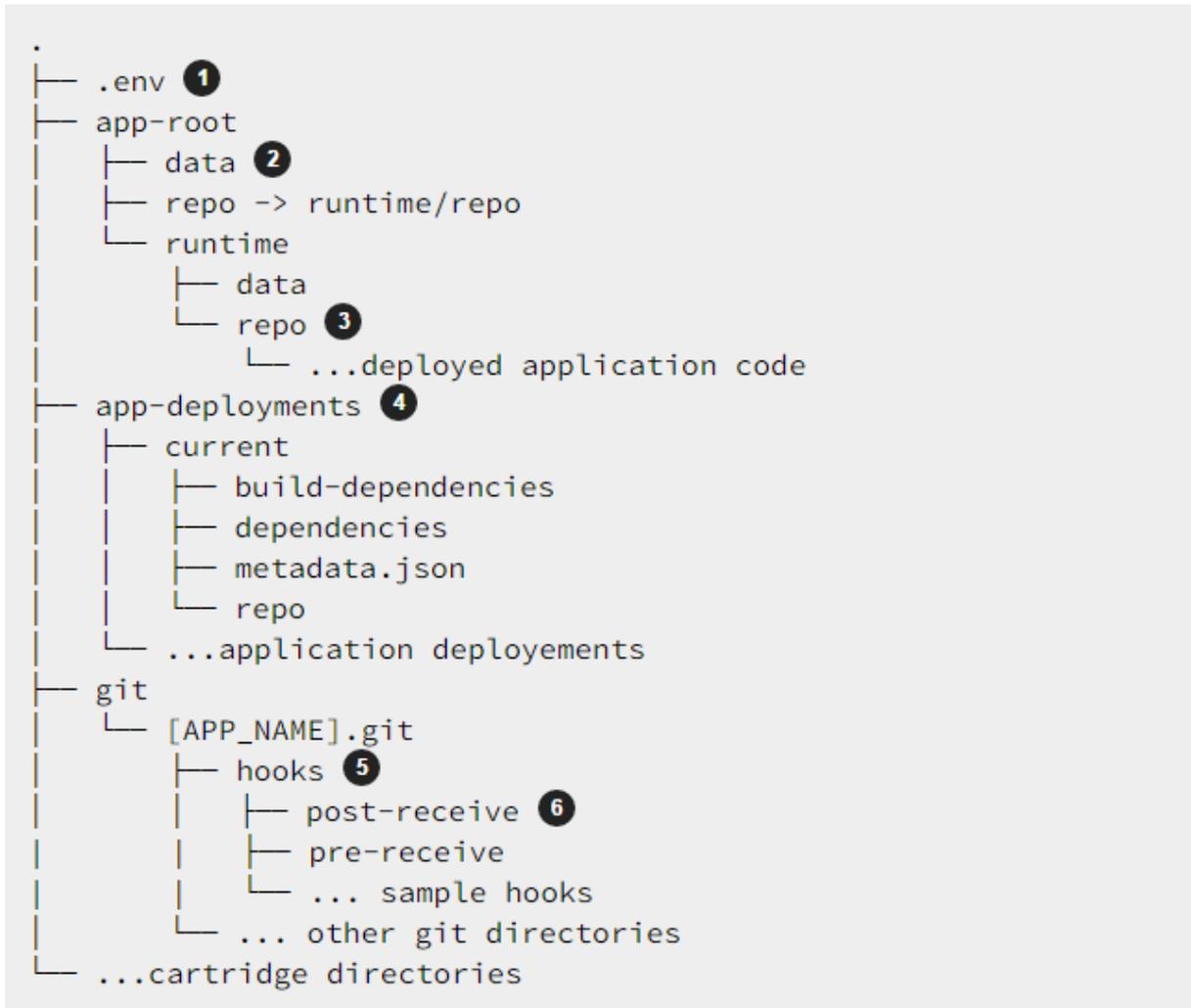
This shell will assist you in managing OpenShift applications.

!!! IMPORTANT !!! IMPORTANT !!! IMPORTANT !!!
Shell access is quite powerful and it is possible for you to
accidentally damage your application.  Proceed with care!
If worse comes to worst, destroy your application with "rhc app delete"
and recreate it
!!! IMPORTANT !!! IMPORTANT !!!

Type "help" for more info.

[wildfly8-djbyte.rhcloud.com 541f1deb500446885d000683]\>
```

Desde la sesión podemos utilizar los comandos habituales de Unix y navegar por la estructura de directorios, que ya anticipamos en la sesión anterior:



- ❶ Almacenamiento de variables de entorno
- ❷ Directorio de datos persistentes \$OPENSIFT_DATA_DIR
- ❸ Directorio del repositorio \$OPENSIFT_REPO_DIR
- ❹ Directorio de despliegues de aplicaciones
- ❺ Directorio de hooks (sólo lectura)
- ❻ El post-receive hook invoca la compilación y despliegue de las aplicaciones

Además de estas carpetas, normalmente nos encontraremos una carpeta adicional por cada cartridge instalado, en la cual encontraremos su configuración. Desde esta sesión podremos modificar ficheros de configuración reiniciar la aplicación, cargar datos en un SGBD, crear scripts o cualquier otra tarea que requiera trabajo "directo" con la máquina. Podemos utilizar la mayoría de los comandos de unix, más una serie de comandos propios de OpenShift:

Comando	Descripción
help	muestra un cuadro resumen con los comandos básicos de OpenShift
gear	permite controlar la aplicación (iniciarla, pararla, reiniciarla, etc.)

tail_all	similar al tail de Unix pero vuelca todos los logs de los servicios
quota	muestra el espacio utilizado y los límites definidos para el gear
mysql	lanza el cliente de MySQL si tenemos desplegado el cartridge
psql	cliente de PostgreSQL
mongo	cliente de MongoDB

Como norma general, modificaremos los ficheros de configuración directamente sobre el gear para realizar pruebas, para realizar cambios "en caliente" sobre la aplicación o bien cuando no exista reflejo de la configuración de un cartridge dentro del repositorio Git de la aplicación. Los cambios de configuración de servidores de aplicaciones habitualmente se harán a través del repositorio Git, mientras que los cambios de configuración de base de datos se harán directamente sobre el gear, o utilizando el mecanismo de *port-forward*.



Recordad que si hacemos cambios directamente en el gear y posteriormente modificamos la configuración desde el repositorio Git, perderemos los cambios realizados.

Variables de entorno

OpenShift permite configurar gears con los elementos que consideremos necesarios para ejecutar nuestra aplicación. Si utilizamos una base de datos, deberemos acceder a ella a través de un puerto determinado y con unas credenciales autogeneradas. Lo mismo ocurre con un servidor de aplicaciones o cualquier otro cartridge que hayamos instalado. Sería complicado tener que gestionar manualmente toda esta información de configuración desde la aplicación, configuración que probablemente cambie si desplegamos la aplicación en un gear distinto, o tras un mantenimiento realizado por Red Hat.

En OpenShift se propone como solución desacoplar esta configuración de las aplicaciones mediante *variables de entorno*. Si desde una sesión ssh, ejecutamos el comando *env* podremos ver la lista de variables de entorno *predefinidas*.

```

expertojava@expertojava-VirtualBox: ~
Archivo Edición Pestañas Ayuda
OPENSIFT_WILDFLY_JDK7=/etc/alternatives/java_sdk_1.7.0
OPENSIFT_WILDFLY_PASSWORD=uAgdSEyhFQis
HISTFILE=/var/lib/openshift/541467525973cacb2c000e52/app-root/data/.bash_history
OPENSIFT_CARTRIDGE_SDK_RUBY=/usr/lib/openshift/cartridge_sdk/ruby/sdk.rb
_=/bin/env
[wildfly8-jlzamora.rhcloud.com 541467525973cacb2c000e52]\> env | grep "WILDFLY"
OPENSIFT_WILDFLY_LOG_DIR=/var/lib/openshift/541467525973cacb2c000e52/wildfly/l
ogs/
OPENSIFT_WILDFLY_IP=127.8.232.129
OPENSIFT_WILDFLY_CLUSTER_PORT=7600
OPENSIFT_WILDFLY_VERSION=8
OPENSIFT_WILDFLY_USERNAME=adminYChLtbA
OPENSIFT_WILDFLY_HTTP_PORT=8080
OPENSIFT_WILDFLY_DIR=/var/lib/openshift/541467525973cacb2c000e52/wildfly/
OPENSIFT_WILDFLY_CLUSTER=
OPENSIFT_WILDFLY_PATH_ELEMENT=/var/lib/openshift/541467525973cacb2c000e52/wildf
ly/usr/lib/jvm/jdk1.8.0_05/bin:/etc/alternatives/maven-3.0/bin:/var/lib/openshif
t/541467525973cacb2c000e52/wildfly/bin/tools
OPENSIFT_WILDFLY_IDENT=wildfly:wildfly:8:0.0.5
OPENSIFT_WILDFLY_JDK8=/var/lib/openshift/541467525973cacb2c000e52/wildfly/usr/l
ib/jvm/jdk1.8.0_05
OPENSIFT_WILDFLY_MANAGEMENT_HTTP_PORT=9990
OPENSIFT_WILDFLY_JDK7=/etc/alternatives/java_sdk_1.7.0
OPENSIFT_WILDFLY_PASSWORD=uAgdSEyhFQis
    
```

Variable Name	Example	Purpose
OPENSIFT_APP_DNS	appname-namespace.rhcloud.com	The fully-qualified domain name for your application
OPENSIFT_APP_NAME	appname	Your application's name
OPENSIFT_APP_UUID	0123456789abcdef0 123456789abcdef	The UUID your application runs as (32 hex characters)
OPENSIFT__IP	127.0.250.1	The IP address your application will listen on
OPENSIFT__PORT	8080	The port your application will receive requests from
OPENSIFT_DATA_DIR	\$OPENSIFT_HOMEDIR/app-root/data/	A persistent directory for your data
OPENSIFT__LOG_DIR	\$OPENSIFT_HOMEDIR//logs/	Where your cartridge-specific logs are stored.
OPENSIFT__DB_LOG_DIR	\$OPENSIFT_HOMEDIR//log/	Where your database-specific logs are stored.

OPENSIFT_REPO_DIR	\$OPENSIFT_HOMEDIR/app-root/runtime/repo/	Repository containing the currently deployed version (of the application)
OPENSIFT_TMP_DIR	/tmp/	A temporary directory you can use. Even though the path is just /tmp, the magic of SELinux protects your data from other users

Adicionalmente cada cartridge que instalemos informa un conjunto de variables distinto por lo que es importante consultar la documentación de cada uno de estos componentes para saber como utilizarlos desde las aplicaciones.

Crear/modificar variables

El conjunto de variables de entorno predefinidas puede ser ampliado con las variables que consideremos necesario. Para gestionar las variables de entorno tenemos un comando específico *rhc env*:

```
$ rhc env set VARIABLE1=valor1 ... VARIABLEN=valorn -a <app> //
  Establecer variables de entorno
$ rhc env list -a <app> // muestra todas las variables de entorno
  definidas por el usuario
$ rhc env unset VARIABLE1=valor1 ... VARIABLEN=valorn -a <app> // Eliminar
  variables de usuario
$ rhc env show VARIABLE1... VARIABLEN -a <app> // Mostrar el valor de las
  variables especificadas.
```



La mayoría de las variables predefinidas están protegidas para preservar la integridad del gear, por tanto si intentamos modificarlas con este comando se nos mostrará un mensaje de error.

Uso de Markers

En la primera sesión de WildFly hablamos de unos ficheros llamados markers que se utilizaban para controlar el despliegue manual de las aplicaciones. El contenido de estos ficheros no es lo importante, si no su nombre y su ubicación. En Openshift se utiliza el mismo concepto para cambiar aspectos de la configuración.

El procedimiento habitual para crear estos marcadores es crearlos dentro del repositorio Git, y subirlos al servidor mediante *push*. La ubicación de estos ficheros dentro del repositorio es *./openshift/markers*.

Hot Deploy

Por defecto, tras realizar un push de cambios al repositorio, se reinicia automáticamente el servidor de aplicaciones. Si queremos redespargar sin reiniciar el servidor, o bien utilizamos un lenguaje interpretado, podemos desactivar el reinicio automático añadiendo el marcador *hot_deploy*:

```
$ touch .openshift/markers/hot_deploy
$ git add .openshift/markers/hot_deploy
$ git commit -m "Changing application to hot deploy"
$ git push origin
```

Cuando hagamos el push del repositorio, se aplicará el cambio en la configuración del servidor.

Otros markers de interés

disable_auto_scaling	Desactiva la creación/destrucción de gears en función de la carga de trabajo
force_clean_build	Fuerza a eliminar todos los artefactos previamente construidos
skip_maven_build	Desactiva la compilación Maven al hacer un push al repositorio
enable_jpda	Activa la depuración remota del servidor de aplicaciones
java7	Fuerza a utilizar el JRE de Java 7 (por defecto Java 6)
java8	Fuerza a utilizar el JRE de Java 8 (por defecto Java 6)

Action Hook Scripts

Los cartridge de Openshift *standalone*, implementan un ciclo de vida con las fases más importantes entre la compilación de la aplicación y su despliegue, de forma similar a herramientas como Maven. Al igual que maven, también se admite la ejecución de acciones definidas por el desarrollador al pasar por estas fases.

Para saber exactamente que fases están definidas es necesario consultar la documentación el cartridge concreto sin embargo son habituales las siguientes fases:

- Build action hooks: pre_build, build, deploy, post_deploy.
- Control action hooks: pre_start, post_start, pre_stop, post_stop, pre_reload, post_reload, pre_restart, post_restart, pre_tidy, post_tidy.

Los scripts de control, admiten especificar un cartridge concreto dentro del gear, por si, por ejemplo, queremos realizar acciones específicas en el arranque/parada de una base de datos.

```
pre_start_{cartridge}
```

Para codificar acciones asociadas a fases debemos crear un script dentro del repositorio git, en la carpeta **.openshift/action_hooks**. El script debe tener exactamente el mismo nombre de la fase y puede estar escrito en cualquier lenguaje de script que admita Openshift (shellscript, Python, PHP, Ruby...), no estando limitado a los cartridge concretos que hayamos instalado en el gear. Para que se pueda ejecutar el script es necesario darle permisos de ejecución.

Como ejemplo, para descargar un fichero tras compilar la aplicación:

1. Creamos el script *.openshift/action_hooks/build*

```
#!/bin/bash
curl -o $OPENSIFT_DATA_DIR/cartel.png http://web.ua.es/es/expertojava/
imagenes/cartel300.png
```

2. Subimos el script al repositorio

```
$ git add .openshift
$ git commit -m "Adding a build hook"
$ git push
```

Tras hacer el push, la aplicación se reiniciará y ejecutará la descarga del fichero tras completar la compilación.



Cuando hacemos *push* a un repositorio, se muestra por la salida estándar tanto la subida de código como el posterior reinicio del servidor y despliegue de la aplicación. En definitiva todo el ciclo de construcción se muestra a través del comando *git*. Los scripts de compilación o control forman parte de este ciclo y salvo que redirijamos su salida a fichero, también aparecerá su salida estándar.



Si trabajamos en Windows, los permisos de ejecución no se trasladan al repositorio remoto al hacer push. Para corregir este problema habría que ejecutar el comando *git update-index --chmod= +x .openshift/action_hooks/** y luego hacer de nuevo push.

Transferencia de ficheros

scp

Si desplegamos una aplicación por primera vez, podemos encontrarnos con la necesidad de cargar datos iniciales en la BD o simplemente crear el modelo de datos mediante un script. En el caso de que nuestra aplicación no sea escalable, esto es sencillo con lo que conocemos: Podemos subir el script al repositorio Git, y desde una sesión SSH ejecutarlo localmente. Otra opción sería utilizar el mecanismo de port-forwarding y lanzar el script desde la máquina de desarrollo.

Una alternativa es utilizar el comando *scp* (secure copy)

```
$ scp <origen> <destino>
$ scp miscript.sql 541f1deb500446885d000683@wildfly8-djbyte.rhcloud.com:~/
app-root/data
```

Este comando permite tanto copiar nuevos ficheros al gear, como descargarnos ficheros a local, variando las rutas de origen y destino. La ruta destino la obtenemos de los datos de conexión ssh que muestra la consola, o que se pueden obtener con el comando *rhc app show <app>*. A esta dirección se le añaden dos puntos y la ruta destino dentro de la máquina remota.

rsync

Es un comando similar a *scp* pero que permite la sincronización de ficheros entre origen y destino de forma incremental. Esto puede ser de utilidad para ficheros de log o ficheros de

tipo backup, y en general para ficheros que cambien con frecuencia pero siempre de forma incremental.

La sintaxis básica del comando sería:

```
$ rsync <origen> <destino>
$ rsync -avz -e ssh 542ab14750044656c70000a0@wildfly8-
jlzamora.rhcloud.com:~/app-root/data/prueba.png .
```

Este ejemplo descargaría del gear un fichero a la carpeta actual. Los parámetros `-avz` se corresponden con `archive` (preservar atributos de ficheros) `verbose` (mostrar información detallada del proceso de sincronización) y `compress` (utilizar compresión en la descarga de diferencias).

Si ejecutamos este comando un par de veces y el fichero origen no cambia, veremos como en el segundo caso la información que se transmite se reduce notablemente.

Tareas planificadas (cron)

OpenShift aporta un cartridge específico para la utilidad de linux `cron` que permite ejecutar tareas planificadas. Para trabajar con tareas planificadas primero debemos añadir este cartridge a la aplicación que deseemos.

```
$ rhc cartridge add cron
```

La forma de trabajar con cron es parecida a la que hemos visto para los Action Hook scripts. Dentro del repositorio git , en la carpeta `.openshift/cron/` nos encontraremos la siguiente estructura de directorios:

```
daily
hourly
minutely
monthly
weekly
```

Cada carpeta se corresponde con una periodicidad de ejecución y podremos añadir los jobs (scripts) que consideremos dentro de ella. A diferencia de los hook scripts, estos pueden tener el nombre que queramos pero si se trata de un shell script se debe indicarlo en la primera línea con la cabecera estándar:

```
#!/bin/bash
```

Los ficheros de los jobs también tendrán que tener el atributo de ejecución para que cron los tenga en cuenta.

Por último, como sólo trabajamos con periodicidades es posible que para algunos casos resulte limitada. Si necesitamos una planificación más exacta podemos añadir código adicional al script que trabaje con la hora del sistema. Por ejemplo, si este script se ubica en la carpeta `minutely`, únicamente se ejecutará en el minuto 12 de la hora actual:

```
#!/bin/bash

minute=$(date +%M)
if [ $minute != 12 ]; then
    exit
fi
# rest of the script
```

7.3. Almacenamiento en disco

A efectos del desarrollador, el disco del gear no difiere demasiado de una instalación local de linux en la que usamos un usuario sin permisos de administración. De toda la estructura de directorios únicamente vamos a tener permisos de escritura en dos sitios:

- La carpeta del gear (OPENSIFT_HOMEDIR).
- La carpeta temporal */tmp*

La carpeta temporal, a diferencia de una máquina linux física es una carpeta es una carpeta propia de cada usuario, es decir que no hay peligro de que otros usuarios que ejecuten sus aplicaciones en la misma máquina que nosotros, tengan acceso a ficheros generados por nosotros. Esta carpeta sólo se puede utilizar en momento puntuales y sabiendo de que no hay garantías que lo que allí almacenemos se vaya a conservar. De hecho, esta carpeta se inicializa en cada reinicio del gear.

Otra ubicación importante es la carpeta de datos de OpenShift, asociada a la variable de entorno OPENSIFT_DATA_DIR (\$OPENSIFT_HOMEDIR/app-root/data) donde si podremos guardar nuestros datos o configuración con garantías de perdurabilidad.

La carpeta git es un caso especial, si bien se podría escribir dentro de ella, todo lo que copiemos se perderá en el siguiente *push* al repositorio que hagamos.



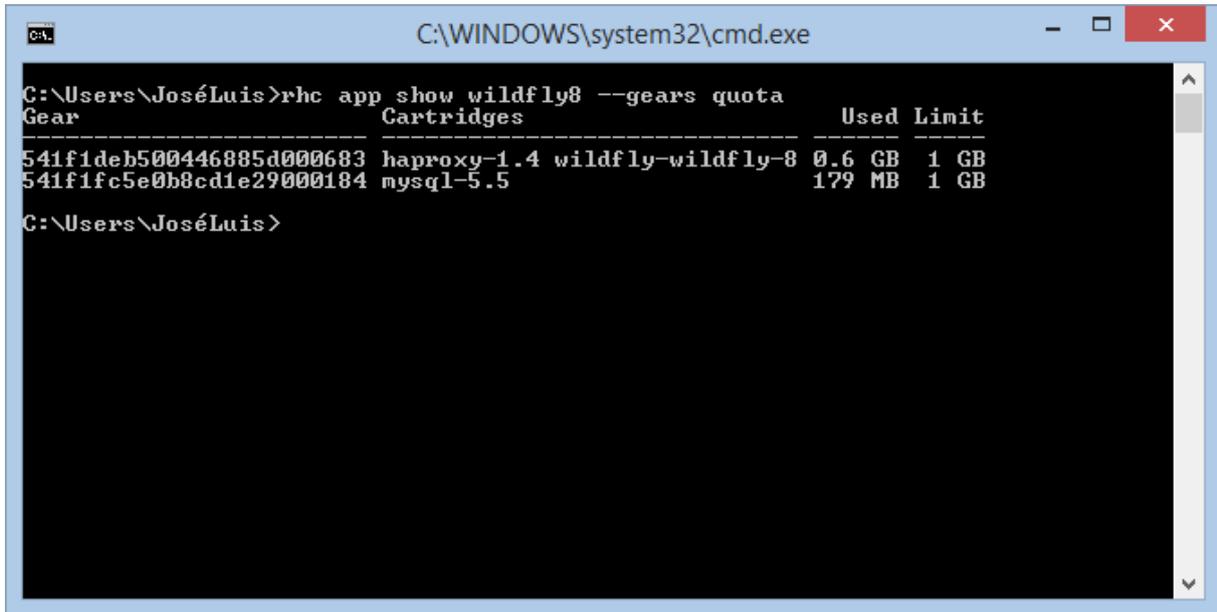
Una limitación que presenta OpenShift es que el directorio de datos es propio de cada gear y no se comparte entre los distintos gears que pueden estar ejecutando una aplicación escalable. Tampoco existe un mecanismo definido para sincronizar estos directorios de forma automática, por lo que el método recomendado para compartir información entre los gears es almacenarla en base de datos, o bien recurrir a un proveedor externo de almacenamiento, como por ejemplo Amazon S3.

En función del tipo de gear nos vamos a encontrar con una limitación de espacio de 1Gb de espacio o 6Gb. Básicamente se tiene en cuenta:

- El directorio */data* del gear.
- La carpeta temporal */tmp*.
- El repositorio Git.
- Los ficheros de log de las aplicaciones y servidores.
- Los ficheros de datos de las bases de datos.

Si nos quedamos sin espacio en un gear, todos los servicios comenzarán a fallar, así que es importante tener siempre controlado el espacio disponible. Ya sabemos una forma de obtener el espacio utilizado, y es mediante el comando *quota* desde una sesión SSH. Rhc nos ofrece un comando que nos dará una visión más global, mostrando el espacio utilizado y la cuota para cada uno de los gears sobre los que se ejecuta nuestra aplicación:

```
rhc app show <app> --gears quota
```



```
C:\WINDOWS\system32\cmd.exe
C:\Users\JoséLuis>rhc app show wildfly8 --gears quota
Gear
-----
541f1deb500446885d000683 haproxy-1.4 wildfly-wildfly-8 0.6 GB 1 GB
541f1fc5e0b8cd1e29000184 mysql-5.5                179 MB 1 GB
C:\Users\JoséLuis>
```



Cuando se alcance un 90% de ocupación de disco, Openshift comenzará a mostrarnos avisos tanto al iniciar una sesión SSH como al hacer un *push* al repositorio Git.

```
remote: error: unable to create temporary file: Disk quota exceeded
```

Si ya nos hemos quedado sin espacio.

Si hemos alcanzado el límite de espacio en un gear podemos resolver el problema de varias formas:

- Comando `tidy` de `rhc`, que elimina los ficheros de log, borra temporales y reorganiza el repositorio git.
- Si estamos en un plan de pago podemos ampliar el espacio en disco con el siguiente comando:

```
$ rhc cartridge storage <cartridge> -a <app> --set capacidad(GB)
```

```
$ rhc cartridge storage php-5 -a racer --set 5gb
```



Todos los gear comienzan con un 1Gb de datos y se pueden ampliar hasta un máximo de 30Gb

7.4. Copias de seguridad

Con lo aprendido hasta ahora, es perfectamente posible definir un proceso que de forma periódica, realice una copia de seguridad de los datos de nuestra aplicación. Deberíamos

crear un job que se ejecute con la periodicidad que creamos conveniente y que ejecute las siguientes tareas:

1. Detener la aplicación
2. Exportar los datos de la BD a fichero.
3. Comprimir y transferir la información de BD a otra ubicación (recomendable por seguridad y porque el espacio en el gear es limitado).

Como ejemplo, se podría subir la información a DropBox o cualquier otro almacenamiento en Cloud. En este enlace se muestra un ejemplo: <https://github.com/andreafabrizi/Dropbox-Uploader>

4. Comprimir y transferir la información de los log de aplicación.
5. Limpiar manualmente temporales y logs (desde la sesión ssh no está disponible el comando tidy).
 - a. Ejecutar `git gc` en la carpeta del repositorio Git `.git/<app>/tmp_`
 - b. Eliminar temporales en `/tmp`, `OPENSIFT_LOG_DIR` y `OPENSIFT_TMP_DIR`

A continuación veremos las herramientas de backup que ofrece OpenShift.

6. Iniciar de nuevo la aplicación.

Deployment History

En ocasiones, al desplegar una nueva versión de nuestra aplicación nos podemos encontrar con que no está funcionando como debería. El estudiar qué ha ocurrido, subir código nuevo al repositorio Git y desplegar puede resultar demasiado lento y mientras tanto la aplicación no está dando servicio o puede estar funcionando de forma incorrecta.

Por defecto no está activado pero OpenShift pide mantener un histórico de despliegues y darnos el mecanismo para desplegar una versión previa a la problemática. Para activar este mecanismo de histórico de despliegues hay que ejecutar el siguiente comando:

```
.....  
rhc app-configure --keep-deployments <num>  
.....
```

La información se almacenará en el gear, en la carpeta `app-deployments` clasificada por timestamp y por "id" de despliegue:

```
.....  
[wildfly8-jlzamora.rhcloud.com app-deployments]\> ls  
2014-10-03_11-32-48.438 2014-10-03_11-51-38.310 by-id current  
[wildfly8-jlzamora.rhcloud.com app-deployments]\> ls by-id  
88410fa9 b901caed  
[wildfly8-jlzamora.rhcloud.com app-deployments]\>  
.....
```

Desde la máquina de desarrollo podemos listar las versiones almacenadas con el comando `deployment list`:

```
.....  
# rhc deployment list <app>  
  
$ rhc deployment list wildfly8  
  
5:33 PM, deployment b901caed  
.....
```

5:52 PM, deployment 88410fa9

Cuando tengamos claro qué versión de la aplicación queremos desplegar, la seleccionaremos con el comando *deployment activate*:

```
# rhc deployment activate <id despliegue> -app <app>
$ rhc deployment activate b901caed -app wildfly8
```

```
Activating deployment 'b901caed' on application wildfly8 ...
```

Uno de los pros de esta funcionalidad es que de forma rápida podemos revertir el estado de nuestra aplicación a un punto anterior. La principal limitación es que sólo afecta al código de aplicación y sus dependencias, no incluye la configuración del servidor de aplicaciones, la definición de la base de datos ni su contenido.

Application Snapshots

Las snapshots son copias de seguridad de la aplicación al completo, código, base de datos y cualquier información que en los cartridge instalados se haya catalogado como exportable. El resultado es un fichero tar.gz con toda la información, que puede ser utilizado para restaurar la aplicación en otro momento.

Tanto la creación del snapshot como la restauración son procesos que necesariamente requieren parar la aplicación temporalmente por lo que hay que escoger el momento adecuado para este tipo de backup.

El comando para generar un snapshot es el siguiente:

```
& rhc snapshot-save <app>
Pulling down a snapshot of application 'wildfly8' to wildfly8.tar.gz ...
done
```

Los snapshot se almacenan en la máquina de desarrollo y salvo que especifiquemos otro nombre para el fichero, se sigue la nomenclatura <app>.tar.gz y se sobrescribe en sucesivas ejecuciones.

Si queremos restaurar, el comando a utilizar es *snapshot-restore*

```
& rhc snapshot-restore <app>
```

7.5. Procedimientos

Depuración de aplicaciones

En casos excepcionales es posible iniciar un proceso de depuración en remoto de una aplicación desplegada en OpenShift para ello tenemos que realizar los siguientes cambios:

1. Añadir el marker *enable_jpda* a la aplicación que queremos depurar. Salvo que tengamos habilitado el *hot deploy* el servidor se reiniciará y arrancará en modo depuración.
2. Ejecutar el comando *port-forward* de *rhc*. Deberá mostrarse una redirección del puerto remoto 8787 a nuestra máquina.

```

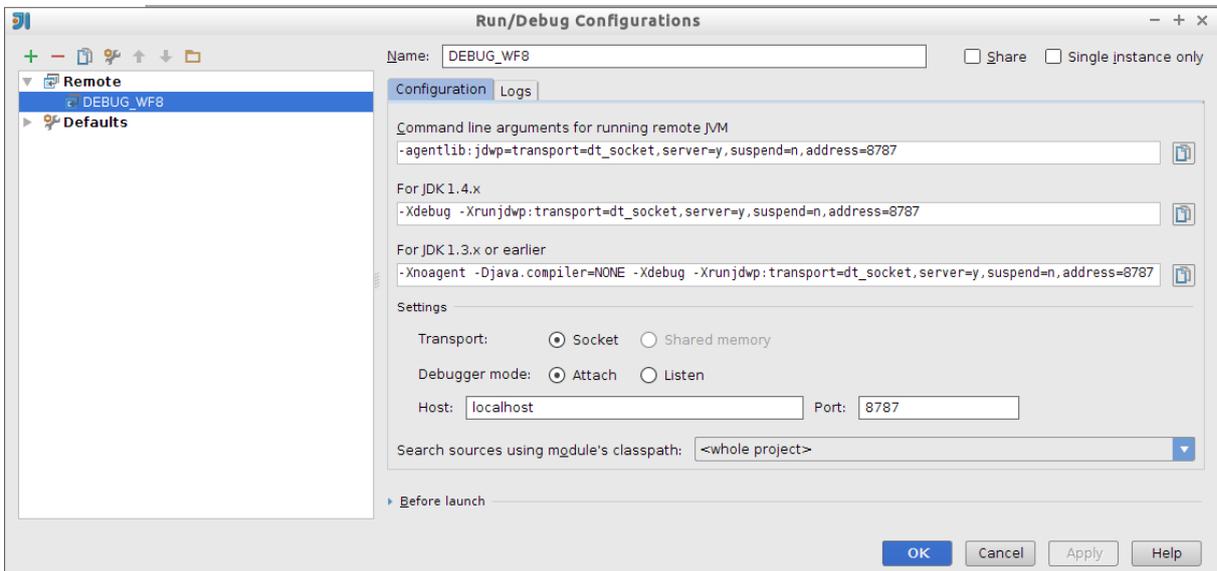
expertojava@expertojava-VirtualBox: ~/kk/wildfly8
Archivo Edición Pestañas Ayuda
remote: /var/lib/openshift/542ab14750044656c70000a0/wildfly
remote: CLIENT_MESSAGE: Artifacts deployed: ./ROOT.war
remote: -----
remote: Git Post-Receive Result: success
remote: Activation status: success
remote: Deployment completed with status: success
To ssh://542ab14750044656c70000a0@wildfly8-jlzamora.rhcloud.com/~/.git/wildfly8.git/
 23b2a48..8aefa85 master -> master
expertojava@expertojava-VirtualBox:~/kk/wildfly8$ rhc port-forward
Checking available ports ... done
Forwarding ports ...

To connect to a service running on OpenShift, use the Local address

Service Local          OpenShift
-----
java  127.0.0.1:3528 => 127.10.118.1:3528
java  127.0.0.1:8080 => 127.10.118.1:8080
java  127.0.0.1:8787 => 127.10.118.1:8787
java  127.0.0.1:9990 => 127.10.118.1:9990

Press CTRL-C to terminate port forwarding
    
```

3. Abrir el proyecto maven de la aplicación en IntelliJ.
4. Crear una nueva configuración de depuración en IntelliJ en modo remoto.



Es importante seleccionar como sistema de transporte el tipo **Socket** y el puerto local al que se redirige el puerto remoto de depuración. También se indica en el último desplegable dónde localizar el código fuente del proyecto que se va a depurar.

Si ejecutamos la configuración de depuración, nos conectaremos directamente al proceso remoto en modo debug, y podremos depurar la aplicación como si se ejecutase localmente, aunque el precio a pagar es que la depuración se ejecutará con mucha mayor lentitud.

Generar un threaddump

Un threaddump o volcado de hilos es un procedimiento que permite recoger en un fichero, el estado de todos los hilos en ejecución de una aplicación, y si están en ejecución, el código que se está ejecutando. Obtener varios threaddump en un intervalo de tiempo puede darnos información útil para identificar problemas de bloqueos o lentitud de procesamiento.

La forma de obtener este volcado es mediante un comando de rhc:

```
$ rhc threaddump -a <app>
```



El cartridge de Wildfly a fecha de estos apuntes no está redirigiendo correctamente la salida del threaddump al fichero temporal que espera el comando threaddump. En su lugar, hay que consultar el server.log directamente, o mediante el comando *tail* de rhc.

Despliegue manual de las aplicaciones

Aunque bien podría haberse implementado como markers, hay parte de la configuración de una aplicación que se resuelve mediante el comando *rhc app-configure*. Si lo ejecutamos sin parámetros nos mostrará la configuración actual de la aplicación:

```
wildfly8 @ http://wildfly8-jlzamora.rhcloud.com/ (uuid:  
542ab14750044656c70000a0)
```

```
Deployment:      auto (on git push)  
Keep Deployments: 1  
Deployment Type: git  
Deployment Branch: master
```

Pero también admite los siguientes parámetros:

```
$ rhc app-configure <app> [parámetros...]
```

--[no-]auto-deploy

Activar/desactivar la compilación y despliegue automático ante un push al repositorio. Por defecto TRUE

--keep-deployments INTEGER

Controla la profundidad del histórico de despliegues de una aplicación. Por defecto 1.

--deployment-branch BRANCH

Permite especificar la rama del repositorio git desde el cual queremos compilar y desplegar la aplicación. Por defecto master.

--deployment-type git|binary

Permite especificar si queremos desplegar un binario o una aplicación compilada a partir del repositorio Git. Por defecto git.

Si desactivamos la función de *auto-deploy* podemos desacoplar la subida de cambios al repositorio del despliegue y gestionar los despliegues de forma manual, mediante el comando *app-deploy*

```
[source,console]
----
$ rhc app-deploy <ref> --app <app>
----
```

Donde ref es una etiqueta, rama o commit específico del repositorio Git (por defecto utilizad master)

Admite los siguientes parámetros que se **superponen** a la configuración especificada vía markers:

--[no-]hot-deploy

Desplegar (o no) sin reiniciar el servidor.

--[no-]force-clean-build

Recompila todo el proyecto (o no) antes de desplegar, eliminando cualquier artefacto temporal previamente generado.

Compilación de proyectos con dependencias propias

En algunos proyectos es necesario referenciar a módulos precompilados de los cuales no tenemos el código fuente o no queremos incluirlo en el repositorio de nuestra aplicación. Estos módulos tampoco están disponibles en ningún repositorio público y si disponemos de un repositorio privado, es muy probable que no lo tengamos configurado para ser accesible desde internet. La solución a este problema pasaría por incorporar el artefacto al repositorio maven local mediante una acción personalizada previa a la compilación. Los pasos a realizar serían los siguientes:

1. Añadir el jar/war a la raíz del repositorio git.
2. Crear un script asociado a la fase *prebuild* que ejecute un "maven install":

```
#Ejemplo para ficheros jar

mvn install:install-file -Dfile=$OPENSIFT_REPO_DIR/fichero.jar
  -DgroupId=<grupo>-DartifactId=<artefacto> -Dversion=<versión> -
  Dpackaging=jar
```

3. Referenciar la dependencia en el pom de la aplicación como una dependencia normal, el scope debe ser *compile* para que se incluya en el empaquetado final de la aplicación.

Despliegue de aplicaciones compiladas previamente.

Como ya sabemos, la forma normal de desplegar aplicaciones en OpenShift pasa por subir código fuente a un repositorio Git interno a OpenShift, pero también se da soporte al despliegue de aplicaciones previamente compiladas aunque para ello hay que realizar una serie de cambios en el repositorio. Los pasos son los siguientes:

1. Crear una nueva aplicación y clonar su repositorio en nuestra máquina.

2. Dentro del repositorio, eliminar el código fuente completamente, con el comando:

```
& git rm -r src/ pom.xml
```

3. Copiar el war/ear a la carpeta deployments y hacer *commit* y *push*.

4. Limpiar el repositorio git y logs con el comando *rhc app tidy*

Tal y como comentamos en la sesión anterior si el war tiene como nombre ROOT, la URL de acceso será la URL indicada para nuestra aplicación. Si por el contrario tiene otro nombre, la aplicación se encontrará en la URL inicial más el nombre de la aplicación como contexto raíz.

7.6. Referencias

- WildFly Cartridge <https://developers.openshift.com/en/wildfly-overview.html>
- Variables de entorno <https://developers.openshift.com/en/managing-environment-variables.html>
- Depuración en remoto http://planet.jboss.org/post/debugging_and_browsing_openshift_applications_in_eclipse
- Action hooks http://openshift.github.io/documentation/oo_user_guide.html#action-hooks
- Openshift v3 <https://www.openshift.com/blogs/openshift-v3-platform-combines-docker-kubernetes-atomic-and-more>

7.7. Ejercicios de Características IaaS+ de OpenShift

Vamos a realizar una serie de ejercicios en torno a un aplicación que podrían acercarnos a una situación real en la que se desarrolla y despliega una aplicación en una plataforma cloud. En primer lugar vamos a crear la aplicación desde cero apoyándonos en una herramienta de generación de código, de nombre *JBoss Forge*.

Forge funciona como un shell con comandos especializados en la generación de código según las funcionalidades que nuestra aplicación va a necesitar. Por ejemplo, si vamos a trabajar con acceso a base de datos JPA sólo tenemos que declararlo y definir posteriormente nuestras entidades.

Instalación de Forge

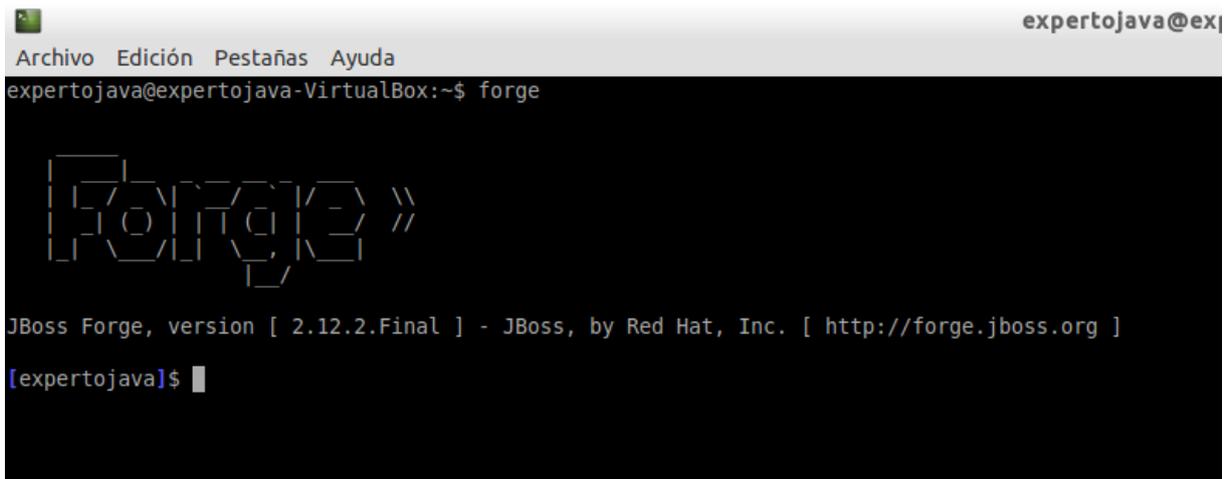
Para instalar la última versión basta con ejecutar el siguiente comando:

```
curl http://forge.jboss.org/sh | sh
```



Con este procedimiento se descarga siempre la última versión disponible, que a la fecha de redacción de estos apuntes es la 3.0 Alpha. En la carpeta de ejercicios de la sesión07 tenéis un script alternativo *forge.sh* que descarga la última versión estable, y es la que se debe utilizar.

Una vez instalado, basta con ejecutar *forge* para iniciar la herramienta:



```

expertojava@expertojava-VirtualBox:~$ forge

JBoss Forge, version [ 2.12.2.Final ] - JBoss, by Red Hat, Inc. [ http://forge.jboss.org ]

[expertojava]$

```

De forma similar a la herramienta CLI de WildFly, podemos movernos por la estructura del proyecto mediante el comando `cd` y podemos visualizar ficheros o ejecutar acciones sobre ellos.

Dado que acabamos de instalar la aplicación, añadiremos un par de plugins mediante los comandos:

```
addon-install-from-git --url https://github.com/forge/addon-arquillian.git
--coordinate org.arquillian.forge:arquillian-addon
```

```
addon-install-from-git --url https://github.com/forge/angularjs-addon.git
--coordinate org.jboss.forge.addon:angularjs
```

Autoescuela

Utilizando Forge, vamos a crear una aplicación sencilla de para la gestión de las clases prácticas de una autoescuela (muy simplificada). Definiremos tres entidades distintas:

- Alumno
- Profesor. Un profesor puede dar clase a varios alumnos, mientras que un alumno únicamente recibe clase de un profesor.
- Vehículo. En este caso un profesor puede utilizar varios vehículos y un vehículo puede ser utilizado por varios profesores, en distintas clases de prácticas.

Mediante Forge vamos a crear nuestra aplicación para gestionar las prácticas. Ejecutaremos los siguientes comandos:

```
project-new --named autoescuela --topLevelPackage org.expertojava.paas
```

```
jpa-setup
servlet-setup
cdi-setup
rest-setup
scaffold-setup
constraint-setup
```

```
jpa-new-entity --named Alumno
jpa-new-entity --named Profesor
```

```

jpa-new-entity --named Vehiculo

jpa-new-field --named modelo --type String --length 100
jpa-new-field --named matricula --type String --length 7
constraint-add --onProperty modelo --constraint NotNull
constraint-add --onProperty matricula --constraint NotNull
constraint-add --onProperty matricula --constraint Size --max 7

cd ..
cd Profesor.java
jpa-new-field --named nombre --type String --length 100
jpa-new-field --named vehiculos --type
  org.expertojava.paas.model.Vehiculo --relationshipType Many-to-Many;
jpa-new-field --named alumnos --type org.expertojava.paas.model.Alumno --
  relationshipType One-to-Many;
constraint-add --onProperty nombre --constraint NotNull

cd ..
cd Alumno.java

jpa-new-field --named nombre --type String --length 100
jpa-new-field --named direccion --type String --length 300
jpa-new-field --named telefono --type String --length 9
constraint-add --onProperty nombre --constraint NotNull
constraint-add --onProperty telefono --constraint Size --max 9

scaffold-generate --provider Faces --overwrite --targets
  org.expertojava.paas.model.*
rest-generate-endpoints-from-entities --targets
  org.expertojava.paas.model.*

build

```

Con estas instrucciones habremos creado un nuevo proyecto *autoescuela*, definido las entidades que intervienen con sus atributos y generado tanto un API REST para su mantenimiento como las páginas JSF necesarias para implementar un CRUD de los datos. Por defecto esta aplicación utilizará el Datasource *java:jboss/datasources/ExampleDS* que viene definido por defecto en WildFly y que trabaja con una base de datos en memoria, H2.



Por defecto genera un ID automático para todas las entidades. Para hacerlo transparente, la herramienta de scaffolding de JSF oculta automáticamente los campos que llevan la anotación de Generated. Además de generar una interfaz en JSF, también es posible generar el Scaffolding AngularJS.



Forge no es una herramienta "neutra" en el sentido de que el proyecto Maven resultante tiene dependencias específicas de JBoss. Podéis sustituirlas por la especificación Java EE 7.0 como se explicaba en las primeras sesiones. Todos los servidores de aplicaciones tienen integrada una base de datos y un datasource por defecto. Podéis eliminar directamente la etiqueta `<jta-data-source>` de la unidad de persistencia para utilizarlo.

Despliegue básico de la aplicación (0.2 punto)

Se pide:

1. Desplegar la aplicación en el servidor WildFly local de vuestra máquina virtual.
2. Añadir el perfil correspondiente para openshift y desplegarla en una aplicación OpenShift, de nombre , *autoescuela*.

Configuración de la aplicación OpenShift (0.6 puntos)

En este ejercicio debéis ajustar la configuración de la aplicación en OpenShift a unos requerimientos:

- Queremos poder aplicar cambios en el menor tiempo posible, esto es evitar reiniciar WildFly en cada cambio que hagamos en la aplicación.
- Por seguridad queremos mantener hasta 3 versiones de la aplicación por si en alguna modificación nos hemos equivocado y queremos deshacer los cambios.
- Sustituir la BD utilizada por defecto H2, por una base de datos PostgreSQL. Debéis crear un Datasource llamado *autoLocal* que acceda a la base de datos y configurar la unidad de persistencia para que lo utilice.

Para acceder a PostgreSQL hay que modificar la configuración de la unidad de persistencia:

```
<property name="hibernate.dialect" value="org.hibernate.dialect.PostgreSQLDialect"/>
<property name="hibernate.hbm2ddl.auto" value="update"/>
```

- Configurar un sistema de copias de seguridad, queremos que de forma diaria a las 01:00 AM se haga un backup del contenido de la base de datos.

El script para esta tarea podría ser el siguiente:

```
#!/bin/bash
if [ `date +%H:%M` == "18:57" ]
then
    DATE=`date +"%Y-%m-%d" `
    FILE="$OPENSIFT_APP_NAME-$DATE.sql.gz"
    INIT_PATH=$OPENSIFT_DATA_DIR/$FILE
    BACKUP_DIR=$OPENSIFT_DATA_DIR/sqlbackup
    if [ ! -d "$BACKUP_DIR" ]; then
        mkdir $BACKUP_DIR
    fi
    pg_dump $OPENSIFT_APP_NAME | gzip > $INIT_PATH
    mv $INIT_PATH $BACKUP_DIR/$FILE
fi
```

- Por ultimo, queremos mantener una copia sincronizada de los backups que hay en OpenShift con lo que debéis definir una instrucción que permita traerse los cambios que se detecten sobre la carpeta sqlbackup. incluidla en un fichero llamado *soluciones.txt*.

Uso de una base de datos externa en Cloud (0.4 puntos)

Por último, vamos a probar a sustituir la base de datos PostgreSQL integrada en el Gear por una base de datos externa proporcionada por otro proveedor. Para ello, debéis acceder al

MarketPlace de OpenShift y crear una cuenta en ElephantSQL. El acceso gratuito proporciona una base de datos de 20Mb con hasta 7 conexiones concurrentes. Debéis crear un nuevo datasource en WildFly, de nombre *autoRemoto*, que apunte a esta base de datos.

Cuando os déis de alta en ElephantSQL, veréis que el proveedor os facilita una conexión con el siguiente formato:

```
postgres://
sbdivvym:DBRXn9a3ewb8evi4qaFcRy5y_LpF9o1h@babar.elephantsql.com:5432/
sbdivvym
```

Si utilizásemos el cliente de PostgreSQL podríamos utilizar esta cadena de conexión tal cual está definida. Sin embargo para crear un Datasource debemos descomponerla en sus partes básicas:

```
postgres://[usuario]:[password]@[hostname:[puerto]//[instancia BD]
```

Por último y muy importante, la configuración del datasource la tenéis que hacer utilizando variables de entorno. Es decir, la configuración del datasource no estará definida directamente en WildFly sino que referenciará a variables que vais a tener que definir. Como sugerencias de nombres:

```
ELEPHANTSQL_USER
ELEPHANTSQL_PASSWORD
ELEPHANTSQL_HOSTNAME
ELEPHANTSQL_PORT
ELEPHANTSQL_DBINSTANCE
```

Entrega

En esta sesión hay que presentar una copia del repositorio *autoescuela* eliminando la configuración de git como ya hemos hecho en sesiones anteriores y un fichero llamado *soluciones.txt* con el comando de descarga solicitado en el segundo ejercicio.

8. Integración continua y escalabilidad

8.1. Escalabilidad

Ya sabemos que OpenShift nos permite trabajar con diferentes tipos de gear con diferentes tamaños de memoria, y recientemente capacidad de proceso (small high cpu), pero la forma natural de aumentar la capacidad de proceso de las aplicaciones en PaaS y en OpenShift es el llamado escalado horizontal, o lo que es lo mismo, que nuestra aplicación se ejecute en múltiples servidores simultáneamente.



Figure 2. Escalabilidad vertical contra escalabilidad horizontal

Autoescalado en OpenShift

Primero de todo aclarar que el que una aplicación sea escalable no implica que el autoescalado esté activado pero es que OpenShift, por defecto gestiona automáticamente la escalabilidad de las aplicaciones, y lo hace de la siguiente forma:

1. Introduce un elemento en el gear de la aplicación denominado HAProxy y su función es la de balanceador de carga por software. Este componente analiza el tráfico web entrante y lo reenvía al gear de la aplicación.
2. Por defecto OpenShift asigna un máximo de 16 conexiones por gear. Cuando HAProxy detecta que se alcanza un 90% de ocupación de estas conexiones de forma sostenida, dispara un evento para la creación de forma automática de un nuevo gear que pueda ayudar a tratar la carga de trabajo creciente.
3. Mediante la herramienta *rsync*, OpenShift copia la configuración, el código y los ficheros empaquetados de la aplicación a un nuevo gear y arranca una nueva instancia del servidor de aplicaciones.
4. Si HAProxy detecta una bajada del número de conexiones por debajo del 50% del máximo, durante un tiempo predefinido, desencadena la eliminación de los gear replicados hasta que se alcance una situación estable o bien se alcance el mínimo de gears activos configurados.

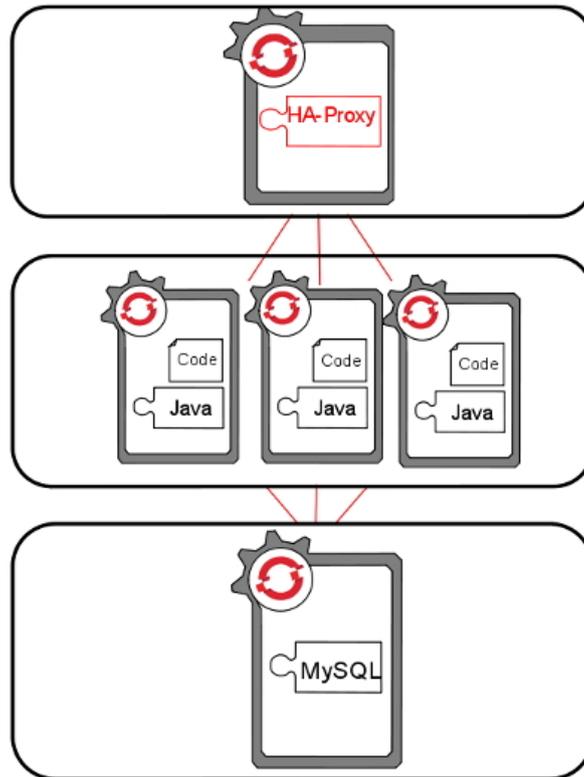


Figure 3. Aplicación escalable

Como vemos en la imagen, las aplicaciones escalables sitúan el backend en un gear independiente a las aplicaciones, en contraposición con las aplicaciones que hemos visto hasta ahora, en las que middleware y backend comparten gear. Esto es necesario para que todos los gear replicados puedan acceder a la base de datos concurrentemente. OpenShift elige la ubicación de la base de datos de forma automática, en función del tipo de aplicación.

HAProxy por defecto utiliza el algoritmo *leastconn* para repartir las conexiones entre los gear activos, y su forma de trabajar es asignar una nueva conexión al gear que tenga menos conexiones activas en ese momento. Este comportamiento se puede modificar, como veremos más adelante.



¿Y si hago cambios en el código de una aplicación escalable?

si hacemos un cambio en el código de una aplicación y lo subimos con un comando *push* de git, iniciaremos la compilación de la aplicación en el gear primario, y una vez compilada que se sincronicen los cambios entre todos los gears activos con el comando *rsync*.

Motivos para definir las aplicaciones como escalables por defecto

- Mayor rendimiento al tener cada servidor y cada BD en un gear dedicado.
- Aunque a priori no sea necesario, el configurar una aplicación como escalable permite que llegado el momento se pueda incrementar la capacidad de proceso.
- Permite autoescalado y escalado manual, en el caso de que sepamos que va a producirse un aumento importante de la carga de trabajo.
- **No** existe un comando que convierta una aplicación no escalable en escalable. Sólo se puede configurar como escalable en el momento de crear la aplicación.

Otras ventajas

- Alta disponibilidad. Si una instancia de servidor falla, HAProxy repartirá la carga de trabajo entre el resto de instancias que estén operativas. No está documentado que haya ningún proceso automático que reinicie las instancias en caso de caída.
- Replicación de la sesión: Si trabajamos con JBoss/Wildfly podemos beneficiarnos de la replica de la información de sesión entre los gears de la aplicación. Esto permite que si un servidor cae, la información de sesión se pueda recuperar de otra instancia. Para ello se utiliza un mecanismo de caché distribuida llamado Infinispan. Para habilitar este mecanismo hay que hacer un pequeño cambio en el fichero web.xml de las aplicaciones:

```
<web-app>
  <distributable/>
</web-app>
```

- El uso de una caché distribuida permite compartir información entre nodos leyendo y escribiendo elementos con los métodos *put/get*.

```
@Resource(lookup="java:jboss/infinispan/container/cluster")
private CacheContainer container;

// A partir de este punto se pueden utilizar los métodos get() y put()
para acceder y almacenar elementos en la caché.
```

Definir una aplicación como escalable

Como ya hemos comentado, sólo es posible configurar una aplicación como escalable en el momento de su creación. Como siempre, tenemos dos opciones:

Desde la consola web:

Public URL -

OpenShift will automatically register this domain name for your application. You can add your own domain name later.

Source Code

We'll create a Git code repository in the cloud, and populate it with a set of reasonable defaults. If you provide a Git URL, your application will start with an exact copy of the code and configuration provided in this Git repository.

Gears

Gears are the application containers running your code. For most applications, the small gear size provides plenty of resources. If you require more resources, select a larger gear size here. You can also [upgrade your plan](#) to get access to more gear sizes.

Cartridges **PHP 5.4**

Applications are composed of cartridges - each of which exposes a service or capability to your code. All applications must have a web cartridge.

Scaling

requests to your web gear. If you allow your load balancer and allocate more gears to handle traffic as you need it.

+1 ⚙

Desde un terminal con la herramienta rhc:

```
.....
rhc rhc app create <app> <cartridges>... -s
.....
```

Ajustar la configuración de escalado

Por defecto las aplicaciones utilizan un gear para la aplicación, más otro adicional si existe una base de datos. Para ajustar el número máximo y mínimo de gears debemos entrar en la configuración de nuestra aplicación y hacer click sobre los valores del apartado scales:

wildfly8-djbyte.rhcloud.com [change](#)
Created 1 day ago in domain djbyte and the aws-us-east-1 region

Cartridges

Icon	Cartridge Name	Scales	Status	Gears	Storage
	WildFly Application Server 8.1.0.Final	1 - 3	Started	1 small	1 GB
	Web Load Balancer				
	MySQL 5.5		Started	1 small	1 GB

Database: wildfly8 User: admin3z3F5yw Password: show

Scale wildfly8

Updated scale settings for cartridge 'WildFly Application Server 8.1.0.Final'

WildFly 8 administrator added. Please make note of these credentials: Username: admini95i4jy Password: TIPX-DzZj5q run 'rhc port-forward wildfly8 to access the web admin area on port 9990.

WildFly Application Server 8.1.0.Final using 2

OpenShift is configured to scale this cartridge with the web proxy HAProxy. OpenShift monitors the incoming web traffic to your application and automatically adds or removes copies of your cartridge (each running on their own gears) to serve requests as needed.

Control the number of gears OpenShift will use for your cartridge:

Minimum and Maximum small gears

Each scaled gear is created the same way - the normal post, pre, and deploy hooks are executed. Each cartridge will have its own copy of runtime data, so be sure to use a database if you need to share data across your web cartridges.

For status information about traffic to your application, see the HAProxy status page:
<http://wildfly8-djbyte.rhcloud.com/haproxy-status/>

MySQL 5.5
This cartridge does not scale.

Web Load Balancer
This cartridge assists other cartridges in scaling, but does not itself scale.

Alternativamente podemos utilizar el comando rhc donde habrá que especificar el cartridge concreto a escalar, la aplicación y el rango de valores.

```
rhc cartridge-scale <cartridge a configurar> -a <your app> --min <minimum> --max <maximum>
```



Si especificamos como máximo -1, estamos indicando que nuestro máximo será el número máximo de gears que nos permita nuestra cuenta.

Una vez creada y en ejecución podemos ajustar manualmente el número de gears con los comandos:

```

rhc app scale-up <app>
rhc app scale-down <app>
    
```

Por último, si queremos desactivar el autoescalado, y que seamos nosotros los únicos que gestionemos el número de gears debemos hacerlo a través de nuestra copia del repositorio Git. Debemos entrar en la ruta

```

cd <app>/openshift/markers
touch disable_auto_scaling
    
```

Con esto crearemos un fichero marcador, de nombre *disable_auto_scaling*, que debemos subir al repositorio con un commit y un push.

Monitorización del balanceador

HAProxy ofrece la posibilidad de consultar datos acumulados sobre la carga de trabajo que está gestionando nuestra aplicación. Podemos acceder a dicha información accediendo a la URL: <http://<app>-<domain>.rhcloud.com/haproxy-status>

> General process information

pid = 487457 (process #1, nbproc = 1)
 uptime = 0d 0h00m38s
 system limits: memmax = unlimited; ulimit-n = 528
 maxsock = 528; maxconn = 258; maxpipes = 0
 current conns = 1; current pipes = 0/0
 Running tasks: 1/2

Legend:
 active UP (green), active UP, going down (yellow), active DOWN, going up (orange), active or backup DOWN (red), backup UP (blue), backup UP, going down (purple), backup DOWN, going up (pink), not checked (grey), active or backup DOWN for maintenance (MAINT) (brown)
 Note: UP with load-balancing disabled is reported as "NOLB".

Display option:
 • Hide 'DOWN' servers
 • Refresh now
 • CSV export

External resources:
 • Primary site
 • Updates (v1.4)
 • Online manual

stats

	Queue			Session rate			Sessions				Bytes		Denied		Errors		Warnings		Server											
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle	
Frontend	1	1	-	1	1	128	2		189	978	0	0	0	0				OPEN									
Backend	0	0		0	0		0	0	128	0	0	189	978	0	0		0	0	0	0	38s UP		0	0	0			0		

express

	Queue			Session rate			Sessions				Bytes		Denied		Errors		Warnings		Server											
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle	
Frontend				0	0	-	0	0	128	0		0	0	0	0	0	0				OPEN									
local-gear	0	0	-	0	0	-	0	0	-	0	0	0	0	0	0	0	0	0	0	0	38s UP	L7OK/200 in 1ms	1	Y	-	0	0	0s	-	
Backend	0	0		0	0		0	0	128	0	0	0	0	0	0		0	0	0	0	38s UP		1	1	0			0		

De esta página nos interesa la segunda tabla, concretamente la fila local gear donde se muestra la información relativa al gear donde se ejecuta nuestra aplicación. En situaciones donde se requieran gear adicionales, aparecerán nuevas entradas en dicha tabla.

> General process information

pid = 483712 (process #1, nbproc = 1)
 uptime = 0d 0h01m40s
 system limits: memmax = unlimited; ulimit-n = 527
 maxsock = 527; maxconn = 258; maxpipes = 0
 current conns = 1; current pipes = 0/0
 Running tasks: 1/3

Legend:
 active UP (green), active UP, going down (yellow), active DOWN, going up (orange), active or backup DOWN (red), backup UP (blue), backup UP, going down (purple), backup DOWN, going up (pink), not checked (grey), active or backup DOWN for maintenance (MAINT) (brown)
 Note: UP with load-balancing disabled is reported as "NOLB".

Display option:
 • Hide 'DOWN' servers
 • Refresh now
 • CSV export

External resources:
 • Primary site
 • Updates (v1.4)
 • Online manual

stats

	Queue			Session rate			Sessions				Bytes		Denied		Errors		Warnings		Server											
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle	
Frontend	1	1	-	1	1	128	2		189	1117	0	0	0	0				OPEN									
Backend	0	0		0	0		0	0	128	0	0	189	1117	0	0		0	0	0	0	1m40s UP		0	0	0			0		

express

	Queue			Session rate			Sessions				Bytes		Denied		Errors		Warnings		Server											
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle	
Frontend				0	0	-	0	0	128	0		0	0	0	0	0	0				OPEN									
gear-5421d9c54382ecc36800037c-djbyte	0	0	-	0	0	-	0	0	-	0	0	0	0	0	0	0	0	0	0	0	1m40s UP	L7OK/200 in 4ms	1	Y	-	0	0	0s	-	
local-gear	0	0	-	0	0	-	0	0	-	0	0	0	0	0	0	0	0	0	0	0	1m40s UP	L7OK/200 in 1ms	1	Y	-	0	0	0s	-	
Backend	0	0		0	0		0	0	128	0	0	0	0	0	0		0	0	0	0	1m40s UP		2	2	0			0		

Cada gear se presentará en un color en función del estado en el que se encuentre.

Configuración de HAProxy

HAProxy mantiene su configuración en un fichero haproxy.cfg que se ha de modificar directamente en el gear principal de la aplicación. Para ello hay que iniciar una sesión ssh y acceder a la carpeta de configuración haproxy/conf:

```
$ rhc ssh <app>
$ cd haproxy/conf
$ vi haproxy.cfg
```

En este fichero podemos modificar entre otras cosas el algoritmo de reparto de carga. Si realizamos algún cambio en la configuración tendremos que reiniciar el cartridge correspondiente al balanceador.

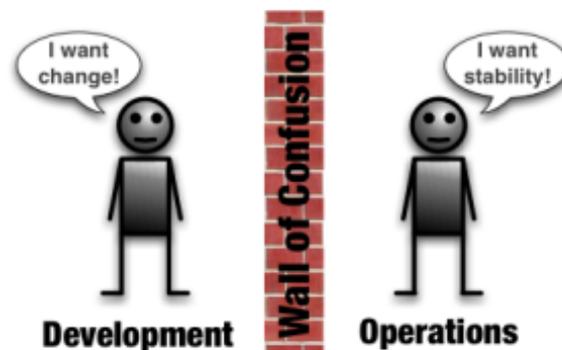
```
$ rhc cartridge-restart --cartridge haproxy -a <app>
```

8.2. DevOps

Introducción

El uso de metodologías ágiles en el desarrollo de aplicaciones es algo relativamente habitual en la actualidad. Estas metodologías favorecen las entregas parciales al usuario final y permite que éste haga cambios en los requerimientos, en función de sus necesidades. El problema lo tenemos cuando en lugar de facilitar un producto instalable, resulta que debemos proporcionar la infraestructura necesaria para hacerlo funcionar, como es el caso de grandes empresas. Y es que tradicionalmente desarrollo y sistemas suelen ser dos bandos con intereses contrapuestos:

- *Desarrollo*: quiero construir y publicar lo que me pide el negocio con agilidad.
- *Explotación*: me miden por el coste y disponibilidad. No quiero riesgos.



Como ejemplo, en un banco podemos encontrarnos con que desde negocio se presiona para poner en marcha un nuevo servicio a clientes. Como respuesta, un equipo de desarrollo comienza a trabajar en el nuevo proyecto. Internamente pueden utilizar la metodología en cascada o incluso metodologías ágiles para proporcionar al usuario interno, distintas entregas del proyecto pero si hablamos de desplegar una aplicación en producción entonces es cuando comienzan las fricciones:

- *Desarrollo*: Perciben demasiada burocracia, falta de permisos en producción (monitorización, incidencias), lentitud en los trasposos.
- *Explotación*: Consideran que les falta información acerca de las aplicaciones que luego tienen que soportar, procedimientos de despliegue mal documentados o con errores, incertidumbre.

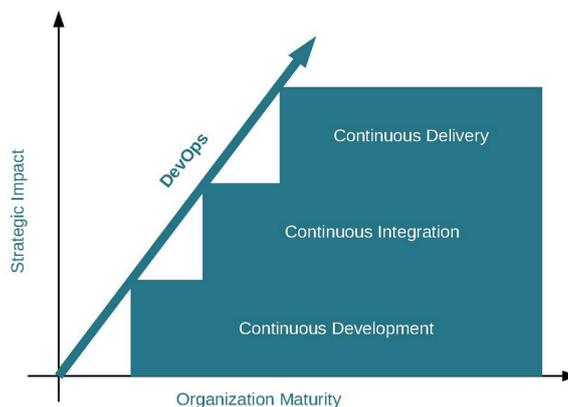
DevOps es un propósito de reconducir esa situación planteando la participación efectiva de los administradores de sistemas en el proceso de desarrollo de aplicaciones, utilizando las mismas metodologías ágiles que los desarrolladores.

5 BEST PRACTICES FOR DEVOPS

1. Continuous delivery.
2. Configuration management.
3. Continuous integration.
4. Automated testing.
5. Deployment pipelines.

Dentro de una organización se podría identificar la madurez de implantación de DevOps mediante el siguiente gráfico:

STRATEGIC IMPACT OF AGILE METHODS



El escalón más básico consiste en la adopción de metodologías ágiles no sólo para la construcción del proyecto sino también para la definición de la infraestructura necesaria para su funcionamiento (infraestructura como script). El siguiente nivel sería la automatización de la integración de cambios en infraestructura y aplicación, y por último el despliegue continuo. La entrega continua implica que dejen de existir las versiones como tal de las aplicaciones y que las nuevas funcionalidades que se desarrollen pasen por un exhaustivo proceso automático de validación y despliegue en producción.

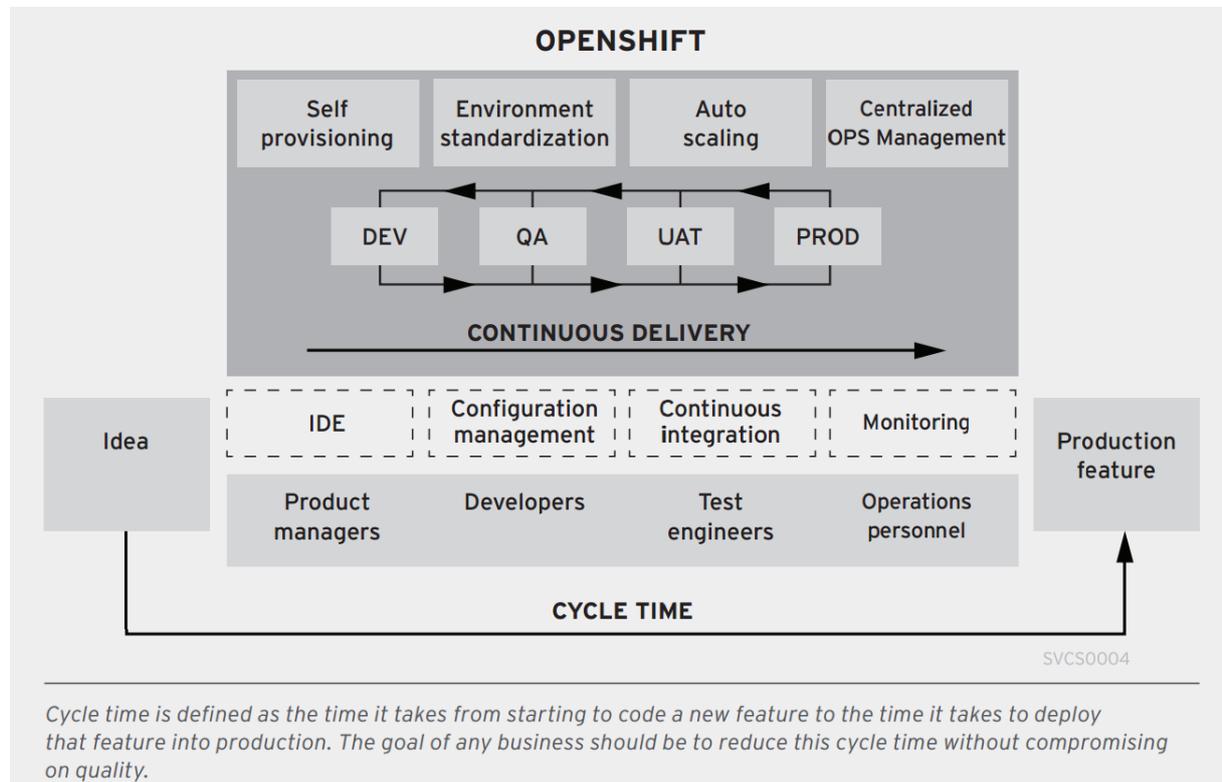
Note: Continuous Delivery es un concepto más genérico que Continuous Deployment. El primero hace referencia a todos los aspectos que permiten hacer despliegues por funcionalidad mientras que el segundo se refiere en sí al proceso técnico de ante un cambio en la aplicación, iniciar el despliegue en producción.

En resumen DevOps pretende mejorar la productividad potenciando el trabajo colaborativo dentre desarrolladores y técnicos de sistemas permitiendo que:

- Desarrollo se concentre en la construcción de las aplicaciones.
- Operaciones se concentre en la creación de entornos estandarizados.

DevOps y Cloud Computing

La gran ventaja de las grandes plataformas IaaS y PaaS frente a la informática tradicional es que resuelven gran parte del trabajo de estandarización de entornos y de procesos. Las prácticas que hemos comentado forman parte de los recursos que la propia arquitectura ofrece al administrador de sistemas con lo que simplifica su utilización:



En el caso concreto de OpenShift, ya sabemos que tanto la creación de gears como cualquier aspecto de configuración es susceptible de automatizarse mediante script y que en su concepción más básica se utiliza el despliegue continuo. A continuación veremos como aplicar también integración continua, pruebas automatizadas y cómo poder definir una build pipeline.



Las herramientas de integración continua, automatización de pruebas, metodologías por si mismas justificarían varias sesiones con lo que hablaremos de los aspecto más importantes y nos centraremos en su implementación en OpenShift.

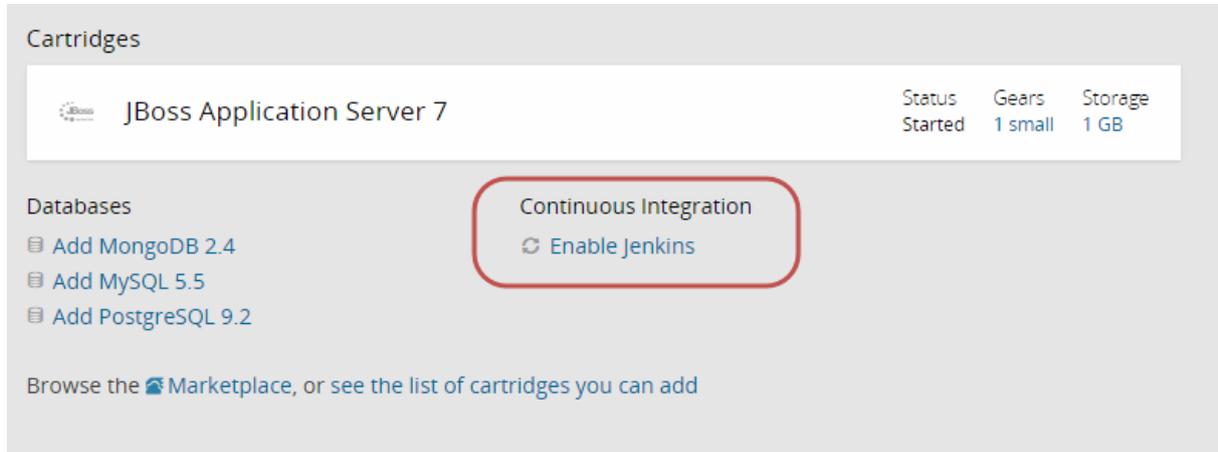
Integración continua (CI) con Jenkins

Entendemos por integración continua un sistema que ejecuta, ante un cambio en una aplicación, la compilación y ejecución automática de un conjunto de pruebas. Si la aplicación compila correctamente y supera las pruebas, la aplicación se considera apta para el despliegue (habitualmente en un entorno desarrollo o integración), pero por el contrario, si no se supera, se deja constancia del error y no se altera la versión de la aplicación que actualmente está ejecutándose en el siguiente entorno de trabajo.

Para aplicar integración continua en un proyecto, necesitaremos herramientas ya conocidas como control de versiones (Git, SVN, etc.) , compilación (desde javac hasta Maven, Gradle) y ejecución de pruebas, pero también necesitaremos una aplicación denominada servidor de

CI. Por defecto OpenShift utiliza Jenkins, aunque no nos impide que instalemos cualquier otro producto.

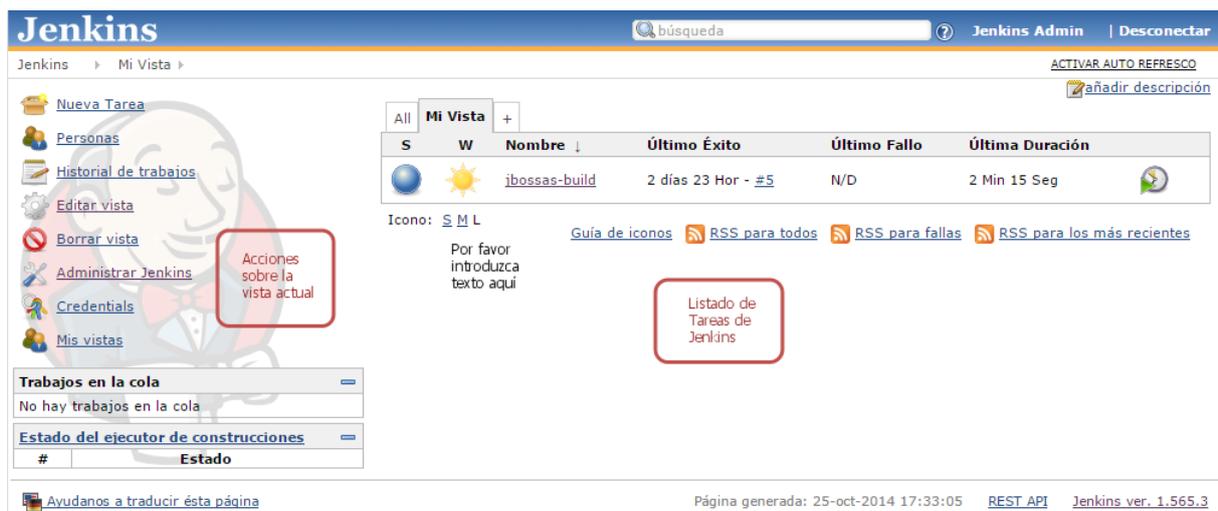
El uso de Jenkins está muy integrado en OpenShift en tanto desde la consola web basta con decir que indicamos que queremos utilizar integración continua para una de nuestras aplicaciones, para que el sistema configure los servicios necesarios para comenzar a trabajar con Jenkins.



Al activar Jenkins se creará un nuevo gear específico para el servidor de CI y se configurará una tarea específica para compilar la aplicación a través de Jenkins. Al crear el servidor Jenkins se nos informa de una URL para acceder a su configuración y un usuario/password para poder identificarnos como administrador.

Introducción a Jenkins

En el caso de Jenkins podemos entenderlo como una herramienta de ejecución y monitorización de jobs, en un sentido más amplio que la compilación y despliegue de aplicaciones. Si accedemos a la consola de Jenkins, nos encontraremos una pantalla similar a la siguiente:



En la parte principal de la pantalla veremos una tabla en la que aparece una única entrada con el mismo nombre que nuestra aplicación y una serie de métricas, inicialmente sin datos. Lo primero que hay que entender es que para Jenkins, cada aplicación que se construye

se denomina **tarea** y cada vez que se active dicha tarea se genera una **compilación**. Si instalamos Jenkins en nuestra propia infraestructura debemos configurar entre otras cosas:

- Rutas de las distintas herramientas que intervienen en la construcción de los proyectos.
- Seguridad básica de Jenkins.
- Las tareas necesarias para compilar.

Con OpenShift tenemos la ventaja de que estas tres cosas ya están preconfiguradas y listas para funcionar desde el primer momento. Si pulsamos en el icono de "Programar una ejecución", activaremos manualmente la tarea y comenzaremos a construir la aplicación. En la parte izquierda de la pantalla se mostrará una barra de progreso donde podemos ver el avance de la construcción y si pulsamos en *Console Output* podremos ver exactamente la salida por consola del script de compilación.



Una de las características más importantes de Jenkins es que almacena toda la información relativa a la construcción de una aplicación (compilación, pruebas, despliegue), no sólo la muestra en tiempo real.

Si dentro de una tarea pulsamos en *Configurar* se nos muestra un formulario donde se puede parametrizar casi cualquier aspecto de la construcción de la aplicación. Entre otros aspectos:

Definición y descripción de la tarea

Datos identificativos del proyecto.

Builder a utilizar

Al activarse una tarea se crea dinámicamente un nuevo gear que se encarga específicamente de la construcción de la aplicación.

Disparadores de ejecuciones

Podemos configurar si queremos que nuestra tarea se active mediante un script, al finalizar otras tareas, periódicamente o cuando se produzca un cambio en un repositorio.

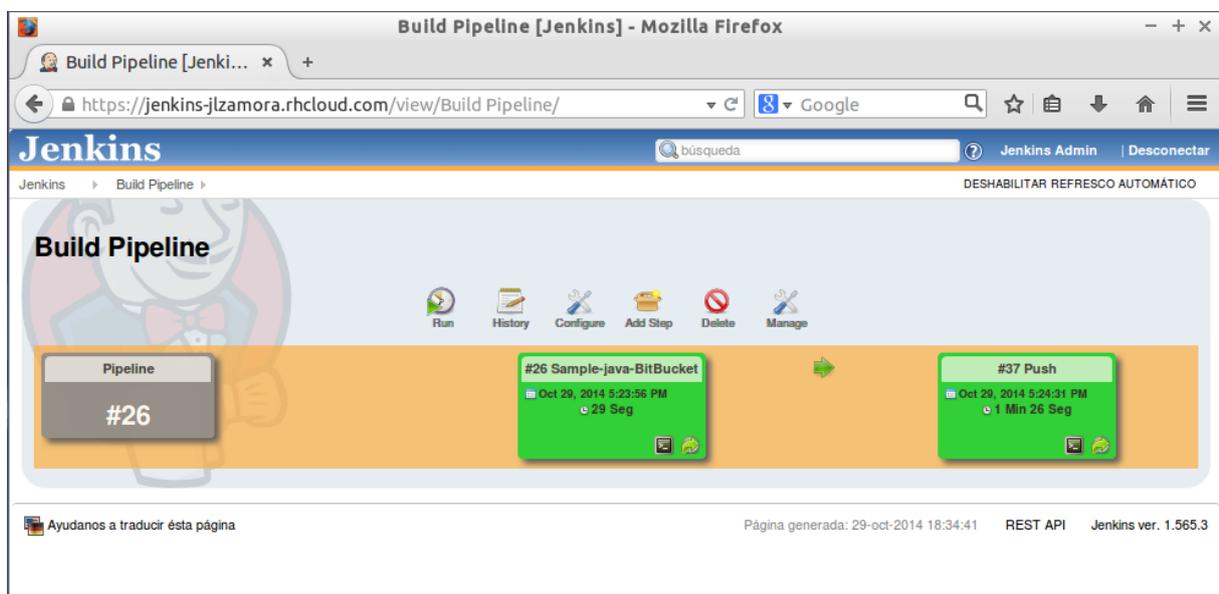
Ejecución

Es un area donde se muestra el sh que se va a ejecutar y se nos permite su total edición. Post Ejecución: Podemos definir comandos adicionales que se ejecuten al finalizar correctamente una tarea.

Una vez se haya ejecutado una tarea, veremos que en la pantalla principal aparecen dos iconos asociados a la tarea. El primero de ellos indica si la última compilación ha sido OK o KO, y el segundo muestra una tendencia, en forma de icono de "previsión meteorológica". Si todo va bien, se muestra un icono soleado, pero si se van produciendo errores de compilación, el icono va cambiando hasta mostrar una "tormenta".

Por último, comentar que Jenkins se puede extender mediante plugins y existe un conjunto muy numeroso de ellos por lo que es un servidor de CI que se puede adaptar a multitud de proyectos. Existen los plugin de tipo *publisher* que permiten mostrar información extendida sobre las tareas ejecutadas de forma visual. Es el caso del plugi de *Cobertura* que muestra gráficamente el grado de cobertura de las pruebas unitarias definidas para una aplicación.

Otro plugin interesante es el *Pipeline plugin* que permite representar un conjunto de tareas interrelacionadas de forma visual. Concretamente se suele utilizar en para representar los Deployment Pipeline de los procesos de Continuous Delivery.



Uso de Jenkins en OpenShift

Por defecto Jenkins se configura utilizando un Gear para el propio servidor y otro Gear "esclavo" para cada trabajo de compilación. Esto puede ser una limitación muy importante en el caso de trabajar con el plan gratuito pero veremos formas de paliar este problema. La ventaja de utilizar un Gear específico para compilar es que no estamos parando la aplicación en ningún momento, ni consumiendo parte de sus recursos en la construcción de la aplicación, y si se produce un fallo en el proceso no llega a afectar a la aplicación en ejecución.

Los eventos que se suceden cuando un usuario hace un cambio sobre una aplicación que trabaja con CI en OpenShift son los siguientes:

1. El desarrollador hace *push* de un cambio al repositorio de la aplicación.
2. Jenkins recibe una notificación de que se ha hecho un cambio en el repositorio de fuentes.
3. A continuación Jenkins crea un gear esclavo para construir la aplicación. Desde ese momento el nuevo gear se puede ver tanto con el comando *rhc domain show* como accediendo a la consola web. El nombre de la aplicación será el nombre original añadiéndole "bldr" (builder). Este mecanismo tiene una limitación y es que los 28 primeros

caracteres de las aplicaciones deben ser únicos o existe riesgo de que en lugar de crear un builder nuevo, se comparta entre las aplicaciones coincidentes.

4. Jenkins ejecuta la tarea de compilación sobre el gear.
 - a. Se descarga el código fuente del repositorio git y mediante el comando *rsync* se copian las librerías asociadas al proyecto que existieran en la aplicación.
 - b. Se ejecuta el script *ci_build.sh* que inicia el proceso de compilación en la máquina builder.
 - i. Se ejecutan los script *.openshift/action_hooks/pre_build* definidos para la aplicación antes de la construcción.
 - ii. Se construye la aplicación ejecutando todos los pasos de construcción definidos en la tarea de Jenkins.
 - iii. Se ejecutan los script *.openshift/action_hooks/build* definidos para la aplicación después de la construcción.
 - c. Ejecución de pasos adicionales post compilación que el desarrollador pueda haber definido.
5. Una vez construida la aplicación con éxito, Jenkins solicita la parada de la aplicación en ejecución.
6. Mediante *rsync* se copian los nuevos artefactos compilados al gear de la aplicación.
7. Jenkins inicia de nuevo el gear.
8. Se archiva toda la información del proceso de build en Jenkins.

Pasados 15 minutos sin actividad el builder se destruye automáticamente aunque la información de los procesos de build se mantienen en el gear principal de Jenkins.

Además de poder acceder a la consola de las build para analizar el proceso de construcción de una aplicación, en el caso de que algo no funcione como debe, tenemos a nuestra disposición el log de Jenkins, el cual podemos consultar mediante el comando habitual de OpenShift:

```
rhc tail jenkins
```

Por su parte, si la aplicación no se despliega correctamente podemos consultar el log del gear de la aplicación.

```
rhc tail <app>
```

Por último, mencionar que la construcción de la aplicación se realiza con los parámetros por defecto de OpenShift (*mvn -e clean package -Popenshift -DskipTests*), por lo que por defecto no se ejecutarán ningún caso de prueba de JUnit u otras herramientas que queramos utilizar. Para poder habilitarlas, dentro del código de la build hay una sección con un comentario:

```
# Run tests here
```

A continuación podríamos añadir la ejecución de pruebas unitarias y cobertura del código:

```
mvn test cobertura:cobertura
```

Si invocamos el comando cobertura necesitaremos el plugin de Cobertura para Jenkins para poder integrar la información de cobertura en la información de la build.

Como trabajar contra un repositorio de Bitbucket

Ya conocemos cómo OpenShift es capaz de instalar Jenkins y crear una tarea de construcción y despliegue por nosotros. No obstante, nuestras necesidades pueden ser algo más complejas o podemos necesitar ejecutar pasos adicionales en el proceso de construcción. Para ello Jenkins nos permite editar las tareas o bien crear una nueva tarea desde cero.

Por ejemplo, podemos querer trabajar con un repositorio en Bitbucket como repositorio principal, en lugar del asociado al gear. La modificación aparentemente es sencilla pues bastaría con edita la información que se muestra en la sección de "Configuración del origen del código fuente":



Pero si lo hacéis os encontraréis con errores de acceso y de permisos al escribir en rutas del gear.

Tipo de credenciales

Lo primero es comentar las tres formas de autenticarse más habituales contra repositorios:

Público (sin credenciales)

Los repositorios de GitHub gratuitos son todos públicos, pero como ya sabéis en BitBucket se pueden tener públicos o privados

Usuario/password

Donde Jenkins utilizará nuestras claves personales del repositorio. ¿Que ocurre si tenemos que cambiar de password?

Claves RSA

Mediante el sistema de clave pública/clave privada con lo que lo que se identificará será la propia instancia de Jenkins ante el repositorio de BitBucket o GitHub.

Para poder utilizar credenciales de tipo usuario/password es necesario instalar un componente adicional, el *plugin de Bitbucket para Jenkins*, pero el método por claves RSA ni siquiera os funcionará con este plugin pues la política de permisos de OpenShift sobre los gear impide que tengamos acceso a la carpeta `.ssh` donde se guardan las credenciales de usuario y al fichero `known_hosts` donde se almacenan las máquinas a las que nos hemos conectado. Para poder acceder a otros repositorios con claves RSA, se puede utilizar un catridge alternativo que ubica esta información en `$OPENSIFT_DATA_DIR`.

Vamos a proponer una solución que contempla ambos enfoques pero requiere una configuración adicional.

Jenkins "Alternativo"

En lugar de dejar a OpenShift la responsabilidad de crear una instancia de Jenkins, la crearemos nosotros en primer lugar, pero utilizando un cartridge de terceros:

```
rhc create-app jenkins jenkins-1 "https://cartreflect-
claytondev.rhcloud.com/reflect?github=majecek/openshift-community-git-ssh
```

Este cartridge tiene la particularidad de trasladar a la carpeta `$OPENSIFT_DATA_DIR/git-ssh` toda la información relativa a claves RSA con lo que podremos trabajar con libertad. La estrategia es definir una variable de entorno llamada `GIT_SSH` que permite especificar un script de tratamiento de credenciales, que entre otras cosas puede especificar la ubicación de las credenciales.



Dado que en gear sólo tenemos permisos para trabajar en carpetas específicas Jenkins está configurado para obtener la identificación del usuario de la carpeta `$OPENSIFT_DATA_DIR` en lugar de la propia del usuario como ocurriría en una instalación local. Esto se hace configurando la variable de entorno `GIT_SSH` para que apunte a un shellscrip que lea de otra ubicación las claves RSA. En el caso de Openshift se encuentra en `/usr/libexec/openshift/cartridges/jenkins/bin/git_ssh_wrapper.sh`. Este mismo mecanismo se utiliza en una instalación local de Jenkins para ejecutar tareas de Jenkins como distintos usuarios.

Si queremos trabajar con claves RSA, todavía tenemos pendiente el crearlas, así que accederemos a la carpeta `OPENSIFT_DATA_DIR/git-ssh` y ejecutaremos:

```
ssh-keygen
```

Deberemos especificar en qué ubicación guardar la clave privada. Especificaremos un nombre de fichero con su ruta completa:

```
/var/lib/openshift/547ae964e0b8cddb49000092/app-root/data/git-ssh/id_rsa
```

The key fingerprint is:

```
7d:f1:25:07:da:a7:e3:b0:c5:e0:e7:82:27:dc:2d:33
```

```
547ae964e0b8cddb49000092@ex-std-node44.prod.rhcloud.com
```

The key's randomart image is:

```
+--[ RSA 2048 ]-----+
|           .           |
|            o .        |
|           + o +       |
|          . . = *      |
|         S . + B       |
|          . + O .      |
|           + E +       |
|            o =        |
|-----+-----+
+-----+-----+
```

Por último, tenemos que avisar a BitBucket de que vamos a descargar código mediante Jenkins. para ello debemos añadir la clave RSA pública asociada a Jenkins dentro de las **Deployment key** de Bitbucket.

Settings

GENERAL

[Repository details](#)

[Access management](#)

[Branch management](#)

[Username aliases](#)

Deployment keys

[Transfer repository](#)

[Delete repository](#)

Deployment keys

Use deployment keys to gain read-only access to this repository. [Learn more about using SSH keys.](#)

Add key

Key	Added
MiJenkins	3 days ago Edit ✕
NuevoJenkins	27 minutes ago Edit ✕

Ahora ya estamos en condiciones de autenticarnos en repositorios privados tanto por usuario/ password como por claves RSA.

BitBucket Plugin

Gracias a este plugin es posible tener un mayor control de cómo se comunica Jenkins con BitBucket, pero no sólo sirve a la hora de acreditarnos, sino que nos permite que ante un cambio en Bitbucket se dispare la ejecución de una tarea Jenkins.

Si editamos una tarea de Jenkins tras instalar el plugin nos aparecerá una nueva forma de "disparar" la ejecución de la tarea.

Disparadores de ejecuciones

- Lanzar ejecuciones remotas (ejem: desde 'scripts')
- Build after other projects are built
- Build when a change is pushed to BitBucket
- Consultar repositorio (SCM)
- Ejecutar periódicamente

Por la parte de Bitbucket hay que añadir un nuevo elemento que avise a Jenkins cuando hay un cambio en el repositorio. Debemos añadir un **Hook** y lo más sencillo, es especificar uno de tipo POST, es decir una llamada http directa al servidor Jenkins:

Hooks

Hooks allow you to extend what Bitbucket does when the repository changes (for example, new code is pushed or a pull request is merged).

POST

Add hook

POST
https://jenkins-djbyte.rhcloud.com:443/bitbucket-hook
[Edit](#) ✕

La URL a indicar es la URL de Jenkins añadiendo la ruta */Bitbucket-hook/*

Por último tenemos que dar de alta las credenciales concretas que vamos a utilizar para autenticarnos. Para ello, en la configuración de la tarea, en la sección del repositorio de código fuente, aparece un desplegable que nos permite especificar qué credenciales queremos utilizar

para identificarnos. Como ya sabemos, estas credenciales pueden ser un usuario/password, o especificar la ruta a la clave privada RSA, o directamente "pegar" dicha clave en el formulario de configuración.

Scope 

Username

Description

Private Key Enter directly

From a file on Jenkins master

File

From the Jenkins master ~/.ssh

También tenemos a nivel del menú principal de Jenkins, una nueva opción "Credentials" para gestionar estas credenciales.

Por último, comentar que a la fecha de la redacción de estos apuntes no funciona correctamente la descarga de código del repositorio de Bitbucket, en el sentido de que el comando de compilación *gear build* no sabe exactamente qué rama hay que compilar. La solución pasa por forzar a utilizar una rama concreta. En nuestro ejemplo *master*. Hay que forzar a descargar el contenido a la rama local master:

Additional Behaviours

- Create a tag for every build 
- Wipe out repository & force clone 
- Check out to specific local branch 

Branch name

Y luego antes de compilar, especificar sobre qué rama vamos a trabajar. Hay que editar el código de la build:

```
# Build/update libs and run user pre_build and build

GIT_BRANCH=master # <-- Forzar a compilar sobre la rama master
gear build

# Run tests here
```



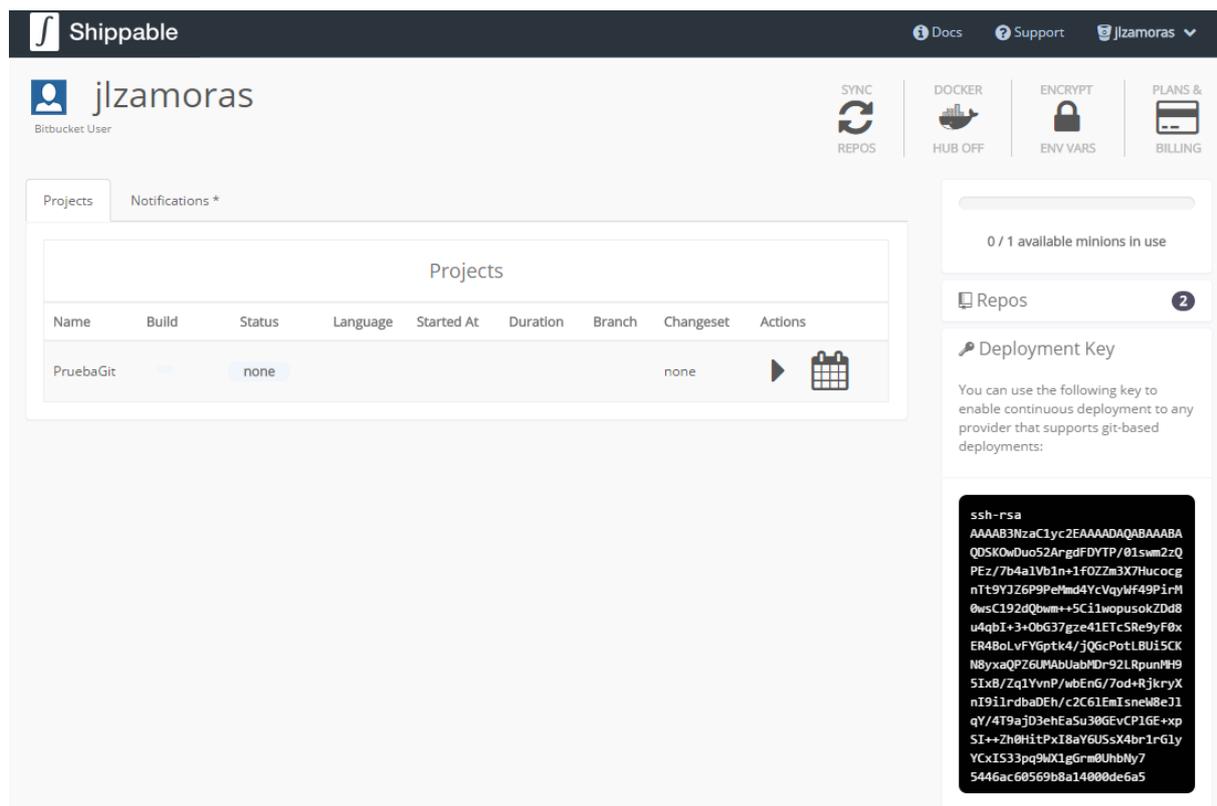
La misma tarea que estamos presentando para trabajar contra Bitbucket la podríamos utilizar para compilar proyectos alojados en GitHub. Bastaría

con instalar el plugin asociado a GitHub, configurar el repositorio remoto con la clave pública de Jenkins y cambiar la URL de Hook por: `https://[url jenkins]/github-webhook/` con content type `application/x-www-form-urlencoded`. En GitHub no se observa el problema de las ramas descrito para el plugin de Bitbucket.

Alternativas a Jenkins

Una de las principales ventajas de utilizar Jenkins integrado en OpenShift es que de forma automática tenemos un sistema que implementa el despliegue continuo. Sin embargo si utilizamos el plan gratuito podemos encontrarnos con sólo nos deja un único gear para poder desplegar nuestras aplicaciones. Una medida que puede paliar el problema es configurar el propio gear de Jenkins como un builder, de modo que tanto la ejecución del servidor como los procesos de compilación correrían en el mismo gear. Esto puede ser factible para proyectos pequeños pero si nos excedemos de los límites de almacenamiento o consumo de memoria, podemos tener errores inesperados.

La otra alternativa es utilizar un servidor de CI de otro proveedor en Cloud. Travis, Shippable y CodeShip son soluciones que funcionan con OpenShift. La estrategia con estos servidores es clonar desde un repositorio principal el código fuente, aplicar las validaciones que consideremos oportunas, y si todo es correcto, hacer un *push* al repositorio del gear en el que queremos desplegar.



En el caso concreto de Shippable, nos podemos identificar mediante Oauth Con las cuentas de Bitbucket o GitHub e inmediatamente acceder a nuestros proyectos, pero antes de poder desplegar hacen falta dos cosas:

1. Añadir la clave pública de Shippable dentro de las claves públicas admitidas de OpenShift, en la pestaña Settings:

Public Keys

OpenShift uses a public key to securely encrypt the connection between your local machine and your application and to authorize you to upload code. [Learn more about SSH keys.](#)

Key name	Type	Contents	
default	ssh-rsa	AAAB3Nza..Wdf2lIVt	Delete
j1msgSobremesa	ssh-rsa	AAAB3Nza..BL8lhB3T	Delete
id_rsa.pub	ssh-rsa	AAAB3Nza..VI4TAGfj	Delete
Shippable	ssh-rsa	AAAB3Nza..m0UhbNy7	Delete

Add a new key...

2. Definir un script de construcción y despliegue. En el caso de Shippable, se denomina *shippable.yml* y se encuentra en la carpeta raíz del proyecto:

```
language: java

jdk:
  - openjdk7

env:
  global:
    - JBOSS_HOME=/tmp/jboss-as-7.1.0.Final
    - JBOSS_SERVER_LOCATION=http://download.jboss.org/jbossas/7.1/jboss-
as-7.1.0.Final/jboss-as-7.1.0.Final.tar.gz
    - OPENSIFT_REPO=ssh://54566a5a5973ca89760000cf@jbossas-
djbyte.rhcloud.com/~/.git/jbossas.git/

before_install:
  - if [ ! -e $JBOSS_HOME ]; then curl -s $JBOSS_SERVER_LOCATION | tar zx
  -C /tmp; fi
  - git remote -v | grep ^openshift || git remote add openshift
  $OPENSIFT_REPO

before_script:
  - mkdir -p shippable/testresults
  - mkdir -p shippable/codecoverage

script:
  - mvn clean cobertura:cobertura
  - mvn test -Parq-jbossas-managed

after_success:
  - git push -f openshift $BRANCH:master
```

Deployment pipeline

Con los elementos que conocemos ya podríamos definir una estrategia de construcción y despliegue para una aplicación en cloud. Esta estrategia podrá ser más o menos compleja en función del tipo de aplicación y el número de partes que intervengan en el proceso. Además del uso que hemos visto de herramientas de CI como Jenkins o Shippable, donde lo que hacemos es básicamente mover código fuente o desplegar aplicaciones ya compiladas en gears, también podemos utilizar el propio api REST de OpenShift para crear, desplegar y destruir gears como parte del ciclo de construcción de una aplicación. En las referencias se cita

un enlace donde se explica el plugin *OpenShift Deployer* que utilizando el api REST permite integrar dentro del proceso de despliegue estas tareas.

También se incluye en las referencias algunos enlaces con *best practices* sobre cómo aprovechar OpenShift para implementar un deployment pipeline. En todas ellas lo habitual suele ser definir varios dominios, uno por cada entorno de trabajo que vayamos a definir. Una posible división entre entornos sería:

Desarrollo

Entorno al que los programadores tienen acceso completo y pueden subir sus cambios desarrollados en local cuando consideran que con correctos y se pueden integrar con la aplicación.

Test

De forma diaria, se ejecuta una tarea de Jenkins que trae las modificaciones de código realizadas en Desarrollo, recompila la aplicación y ejecuta todas las pruebas automatizadas. También este entorno sirve para que el equipo de testing pueda realizar sus pruebas. Sobre este entorno se podrían realizar pruebas de aceptación por parte de usuarios, o bien separarlas en diferentes entornos.

Preproducción o Stage

Cuando el equipo de testing da el ok a estas pruebas, manualmente se traspasan los cambios de Test a Preproducción. Todo el código de preproducción se considera código validado y preparado para pasar a producción. Sin embargo en este entorno el equipo de Operaciones puede realizar las últimas pruebas de cara a su puesta en producción.

Producción

Cuando se considere que la nueva versión puede publicarse, alguien desde operaciones iniciará el proceso de traspaso desde preproducción hasta producción manualmente, quedando la nueva aplicación disponible a los clientes.

En entornos estáticos tradicionales, se suele tener un servidor por entorno, lo cual puede condicionar las pruebas que se realicen sobre una aplicación ya que en proyectos grandes se pueden estar probando distintas *features* simultáneamente y ser incompatibles dichas pruebas. O también es habitual que se quieran promocionar determinadas funcionalidades pero no otras hasta una fecha pactada con el cliente. Gracias a las herramientas (Git, Jenkins, OpenShift) podemos crear servidores de pruebas temporales para ramas específicas de un proyecto, que se validen de forma independiente, y que cuando llegue el momento previo a su despliegue en producción, se integren en Preproducción.

8.3. Referencias

- Application Scaling <https://www.OpenShift.com/developers/scaling>
- Scaling in action <https://www.openshift.com/blogs/scaling-in-action-on-openshift>
- Autoescalado en OpenShift <https://blog.openshift.com/how-to-host-your-java-ee-application-with-auto-scaling/>
- Puppet [http://es.wikipedia.org/wiki/Puppet_\(software\)](http://es.wikipedia.org/wiki/Puppet_(software))
- Continuous delivery http://www.xebia.in/assets/files/whitepaper_xebia_continuous_delivery.pdf
- Continuous delivery using pipelines <http://www.methodsandtools.com/archive/archive.php?id=121>
- Openshift development pipeline <https://blog.openshift.com/elevating-your-development-pipeline-to-the-cloud-with-openshift/>

- Administering Jenkins <https://wiki.jenkins-ci.org/display/JENKINS/Administering+Jenkins>
- Cómo subar Sonar en OpenShift <http://majecek.wordpress.com/2013/12/06/how-to-run-sonarqube-4-0-on-openshift/>
- Desplegar en OpenShift desde un jenkins externo: <https://blog.openshift.com/deploy-openshift-external-jenkins-instance/>
- Estrategias de Release Management: <https://blog.openshift.com/release-management-in-the-cloud/>
- Development pipeline en cloud: <https://blog.openshift.com/elevating-your-development-pipeline-to-the-cloud-with-openshift/>
- Ticket Monster <http://www.jboss.org/ticket-monster/>
- Blog de Arun Gupta <http://blog.arungupta.me>

8.4. Ejercicios de Integración continua y escalabilidad

En esta sesión vamos a trabajar con la aplicación *Ticket Monster*. Esta aplicación es una plataforma de venta de entradas por internet y Red Hat la publica con fines educativos y como demostración de distintas tecnologías. Para más información en la sección de referencias hay un enlace con documentación acerca de esta aplicación.

TICKETMONSTER

About Events Venues Bookings Monitor Administration

TicketMonster.
A JBoss Example.

TicketMonster is an online ticketing demo application that gets you started with JBoss technologies, and helps you learn and evaluate them.

[Buy tickets now](#)

A jdfv showcase
TicketMonster's architecture uses the best technologies that make up the **JBoss Developer Framework**. Learn how easy it is to harness the power of **Java EE** with **HTML5 + REST**.

Mobile-ready
When you interact with TicketMonster on a **mobile** device, you will get a mobile-specific interface. Learn how easy it is to write Java EE applications that can accommodate desktop and mobile clients.

Cloud-ready
A deployment of TicketMonster runs continuously **in the cloud** on the **OpenShift** platform. Learn how easy it is to write Java EE applications that can deploy both locally and in the cloud.

Fork it on github
TicketMonster is available on **GitHub**. Within a click, you can get the source code, build it and run it!

Learn with it
TicketMonster is accompanied by a **tutorial** that walks your through the application architecture. Read it to learn how the application is built Or even

Customize it
TicketMonster an example that you can follow, but you can also use the predefined application structure to implement your own use cases and test your

Por lo pronto vamos a descargarnos el código de GitHub correspondiente a la rama de código dirigida a WildFly. Concretamente ejecutaremos:

```
git clone -b WildFly https://github.com/jboss-developer/ticket-monster.git
```

Una vez descargado lo compilaremos con los siguientes parámetros:

```
cd ticket-monster/demo
mvn clean package -Ppostgresql-openshift
```

La compilación generará un fichero *ticket-monster.war*.

Despliegue de una aplicación escalable (0.2 Puntos)

Para este ejercicio se pide:

- Crear una aplicación OpenShift de nombre *ticketmonster* que sea escalable y que utilice una base de datos *PostgreSQL*.
- Desplegar el war compilado como aplicación.
- Verificar que la aplicación funciona correctamente.

Prueba de Estrés (0.4 puntos)

En este ejercicio vamos a comprobar cómo funciona el mecanismo de autoescalado simulando una carga de trabajo elevada. Para ello, lo primero es instalar una herramienta de pruebas de estrés: *ApacheBench*.

```
$ sudo apt-get install apache2-utils
```

La aplicación Ticket monster, tiene una sección de administración generada mediante JBoss Forge y de la cual también expone una interfaz REST para manipular los datos de la aplicación. Vamos a lanzar una consulta de un servicio REST de forma masiva, con el fin de generar un número de sesiones concurrentes lo suficientemente elevado como para que OpenShift dispare la activación de un segundo gear.



Recordad que al configurar una aplicación como escalable ya consumimos 2 gear, así que como mucho, vamos a poder incorporar un tercero.

La forma de lanzar una prueba de carga contra un servicio REST de consulta (GET) sería la siguiente:

```
$ ab -n 50000 -c 50 -s 100 http://ticketmonster-jlzamora.rhcloud.com/rest/sections
```

Donde "n" es el número de peticiones a realizar, "c" es el número de peticiones concurrentes y "s" es el tiempo máximo de espera por petición.

Si nos conectamos mediante SSH al gear de la aplicación, podemos consultar el log del balanceador en la ruta `$OPENSIFT_LOG_DIR/haproxy_ctld.log` donde podremos ver los eventos de creación/destrucción de gears en función de la carga de trabajo. Adicionalmente en la consulta web de HAProxy también podemos ver el número de gears que tenemos en funcionamiento y su carga de trabajo.

Para este ejercicio se pide:

1. Ejecutar la prueba de estrés y verificar que pasado un tiempo se crea automáticamente un gear adicional.
2. En cuanto aparezca el nuevo gear, finalizar la prueba de carga mediante *CTRL+C* al comando de ApacheBench. Guardar los datos estadísticos de rendimiento que ha generado la herramienta en un fichero llamado **1Gear.txt**.
3. Comprobar que pasado un tiempo, se elimina automáticamente el segundo gear.
4. Obtener una copia del fichero **haproxy_ctld.log** como entregable.
5. Forzar a que la aplicación ticketmonster trabaje siempre con dos gears.
6. Repetir la prueba de carga hasta un número de ejecuciones similar a la realizada en el paso 2, y almacenar las estadísticas de ApacheBench en un fichero llamado **2Gear.txt**.



Si os interesa realizar pruebas de estrés, además de ApacheBench podéis utilizar JMeter para realizar pruebas de estrés. Se trata de una herramienta mucho más completa pero algo más complicada de sacar partido. Como SaaS, tenéis la herramienta BlazeMeter, que permite generar simulaciones de carga de trabajo de forma sencilla pero también permite ejecutar pruebas de estrés de JMeter. La pega es que el acceso gratuito permite generar un volúmen de peticiones limitado.

Integración continua (0.2 puntos)

Esta vez vamos a trabajar sobre una aplicación Java EE 7 de ejemplo muy sencilla, pero sobre la que aplicaremos los distintos conceptos que hemos aprendido sobre integración continua y despliegue continuo. En este primer ejercicio se pide crear una nueva aplicación OpenShift denominada *bigbang* y desplegar sobre ella la aplicación de ejemplo, alojada en GitHub. Para ello eliminaremos los fuentes por defecto y haremos un `_pull` del siguiente repositorio de GitHub:

```
git rm -rf src/ pom.xml
git commit . -m "Eliminar aplicación de ejemplo "

git remote add javaee7 https://github.com/Djbyte1977/bigbang.git
git pull javaee7 master

git push origin master
```

Una vez desplegada la aplicación, comprobad que funciona correctamente. La aplicación expone un servicio REST del que podéis obtener una lista de personas o el nombre de una persona concreta preguntando por su id:

```
http://[URL aplicación Openshift]/resources/persons
http://[URL aplicación Openshift]/resources/persons/id
```

Lo siguiente será habilitar integración continua para el proyecto y modificar la clase *PersonDatabase*:

```
public void init() {
    persons = Arrays.asList(
```

```
new Person("Penny"),
new Person("Leonard"),
new Person("Sheldon"),
new Person("Amy"),
new Person("Howard"),
new Person("Bernadette"),
new Person("Raj"),
new Person("Priya"),

// Añadir una persona a la lista
new Person("Stu"));

}
```

Por último, hacer un *push* del cambio y verificar que la aplicación se compila primero en Jenkins antes de desplegarse. Ahora, cada vez que hagamos un push, la aplicación se compilará previamente, y si se produce algún fallo, el despliegue no se realizará, por tanto tenemos un nivel adicional de tolerancia ante fallos.



Si ejecutamos la compilación por primera vez, es posible que se produzca algún error de comunicaciones entre nuestra consola y el gear Builder. Normalmente esto es debido a que son instancias gratuitas y que el primer arranque es más costoso. Si lo intentáis por segunda vez os debe funcionar.

Pruebas con Arquillian (0.4 puntos)

El siguiente paso es incluir una serie de pruebas que nos garanticen no sólo que el código compila sino que el funcionamiento de la aplicación será el esperado. Para ello incorporaremos a la aplicación un par de casos de prueba implementados con *Arquillian*. Esta herramienta permite diseñar pruebas unitarias y pruebas integradas que se pueden ejecutar directamente sobre un servidor de aplicaciones, permitiendo realizar pruebas en condiciones muy cercanas a las del funcionamiento real que tendrá la aplicación.

Para dotar al proyecto de estas capacidades utilizaremos Forge dentro de la carpeta del proyecto. Ejecutaremos:

```
$ forge

[bigbang]$ addon-install-from-git --url https://github.com/forge/addon-
arquillian.git

//Como Framework de pruebas elegid JUnit y elegir el contenedor que
queráis utilizar.
[bigbang]$ arquillian-setup

[bigbang]$ arquillian-container-setup
```

Forge no sólo se utiliza para crear nuevos proyectos sino que permite modificar proyectos ya existentes. En este caso hemos añadido un plugin a Forge para poder trabajar con Arquillian y hemos añadido a nuestra aplicación la configuración necesaria para utilizar distintos contenedores. Cada contenedor está especializado en interactuar con un tipo de servidor de aplicaciones, en distintas configuraciones. En el caso de WildFly tenemos:

arquillian-wildfly-remote

Que trabaja contra un servidor WildFly que se esté ejecutando en una máquina, por defecto en localhost:8080

arquillian-wildfly-managed

En este caso Arquillian se encarga de parar y arrancar el servidor y sólo debemos indicarle en qué carpeta se encuentra, bien a través de la variable de entorno `JBOSS_HOME` o bien en el fichero de configuración de arquillian (`arquillian.xml`).



Por defecto estos plugins siempre van a trabajar contra la última versión de WildFly con lo que si la aplicación se ejecutará en una versión distinta, habría que modificarla en el fichero `pom.xml`

Para trabajar con OpenShift la aproximación es diferente pues en principio no se puede iniciar un proceso servidor dentro de un gear fuera de los puertos autorizados. Por tanto no es sencillo el utilizar uno de los modos anteriores pues la instancia de WildFly no tendrá permisos para abrir los puertos que necesita para funcionar.

Por fortuna existe un contenedor específico para OpenShift, **arquillian-openshift-express**, que permite utilizar instancias de JBoss/WildFly ejecutadas dentro de un gear.

Como hemos seleccionado la opción de `openshift-express`, ya tenemos nuestro proyecto preparado con los perfiles adecuados para poder utilizarlo. Faltaría modificar el fichero `arquillian.xml` dentro de la carpeta de `test/resources` indicando sobre qué gear vamos a lanzar las pruebas. En nuestro caso, como sólo contamos con tres gear disponibles vamos a utilizar el mismo gear que el despliegue final.



En condiciones de "producción" necesitaremos 4 gear: 2 para Jenkins, uno para el servidor de Test y otro para el servidor de producción.

Añadiremos la configuración del gear sobre el que vamos a probar:

```
<!-- Datos de ejemplo, cambiar por los vuestros-->
<container qualifier="arquillian-openshift-express">
  <configuration>
    <property name="namespace">jlzamora</property>
    <property name="application">bigbang</property>
    <property name="libraDomain">rhcloud.com</property>
    <property name="sshUserName">5479d3685973ca8081000033</
property>
    <property name="login">jl_msg@hotmail.com</property>
    <property name="deploymentTimeoutInSeconds">1000</property>
    <property name="disableStrictHostChecking">>true</property>
  </configuration>
</container>
```



En lugar de indicar valores constantes, si vamos a ejecutar las pruebas desde jenkins podemos utilizar la sintaxis: `${env.NOMBRE_VARIABLE}` para leer el valor de una variable de entorno.

También es necesario hacer algunos cambios manualmente sobre el `pom.xml` por ¿fallo? de la versión 2.12 de Forge:

- Añadir el repositorio de JBoss para poder descargar las últimas versiones de los plugin:

```

<repository>
  <id>jboss-public-repository-group</id>
  <name>JBoss Public Repository Group</name>
  <url>http://repository.jboss.org/nexus/content/groups/
public/</url>
  <layout>default</layout>
  <releases>
    <enabled>>true</enabled>
    <updatePolicy>never</updatePolicy>
  </releases>
  <snapshots>
    <enabled>>true</enabled>
    <updatePolicy>never</updatePolicy>
  </snapshots>
</repository>

```

- Añadir dependencias para crear los clientes REST utilizados en las pruebas:

```

<dependency>
  <groupId>org.jboss.resteasy</groupId>
  <artifactId>resteasy-client</artifactId>
  <version>3.0.10.Final</version> <!--Sustituir por la versión que
utilice WildFly -->
  <scope>test</scope>
</dependency>
<dependency>
  <groupId>org.jboss.resteasy</groupId>
  <artifactId>resteasy-jaxb-provider</artifactId>
  <version>3.0.10.Final</version> <!--Sustituir por la versión que
utilice WildFly -->
  <scope>test</scope>
</dependency>

```

Además de incluir la configuración de Arquillian debéis incluir también algún caso de pruebas. En las plantillas de la sesión 8 encontraréis la clase *PersonTest.java* que define dos casos concretos para validar el servicio REST:

Esta es la parte concreta de los casos de prueba:

```

@Test
public void testGetAll() {
  Person[] persons = target.request().get(Person[].class);
  assertEquals(8, persons.length);

  assertEquals("Penny", persons[0].getName());
  assertEquals("Leonard", persons[1].getName());
  assertEquals("Sheldon", persons[2].getName());
  assertEquals("Amy", persons[3].getName());
  assertEquals("Howard", persons[4].getName());
  assertEquals("Bernadette", persons[5].getName());
  assertEquals("Raj", persons[6].getName());
  assertEquals("Priya", persons[7].getName());
}

```

```

/**
 * Test of get method, of class Person.
 */
@Test
public void testGetOne() {
    WebTarget target2 = target.path("{id}");

    Person response =
target2.resolveTemplate("id", 0).request().get(Person.class);
    assertEquals("Penny", response.getName());

    response =
target2.resolveTemplate("id", 1).request().get(Person.class);
    assertEquals("Leonard", response.getName());
}

```

En este ejercicio se pide:

1. Ejecutar las pruebas utilizando los dos tipos de contenedores de Wildfly en local. Documentar en el fichero *soluciones.txt* los pasos realizados. Deben ejecutarse correctamente las pruebas pero alguna de ellas detectará un error en el código. Revisar los logs y corregirlo.
2. Desplegar la aplicación en OpenShift.
3. Desde la máquina local probar a ejecutar las pruebas contra la aplicación bigbang de OpenShift:

```
mvn test -Parquillian-openshift-express
```

La salida de esta ejecución se debe incluir en el fichero *soluciones.txt*.

Posibles problemas

A pesar de tener los pasos detallados, a la hora de ejecutar estos ejercicios os podéis encontrar (entre otros) los siguientes problemas:

- Lentitud: Los gear gratuitos de OpenShift son muy lentos y tienen una memoria limitada. Las pruebas con Arquillian son pesadas y en estas condiciones tardan bastante en ejecutarse. Dado que los gear se ejecutan en la costa este de EEUU, tenemos el problema añadido de que por la noche son todavía más lentos.
- Desconexiones de los builder (excepción `hudson.remoting.RequestAbortedException: java.io.IOException: Unexpected termination of the channel`). Al lanzar tareas en Jenkins es posible que el gear builder se desconecte del servidor de Jenkins. Se puede solucionar accediendo a la configuración de los nodos, pulsar sobre el builder y a continuación pinchar en el botón "Lanzar agente esclavo" que se muestra para reconectar, cuando hemos perdido la conexión.
- Errores de ejecución en las pruebas: Las pruebas pueden mostrar un error 500 y en este caso habría que ir al log del servidor para obtener más información. Es posible que si no hemos incluido correctamente las dependencias de Resteasy (la implementación REST de JBoss/Wildfly) las pruebas fallen al no poder crear el cliente REST.

- Parar a mitad las pruebas con Arquillian. Si interrumpís alguna prueba, podéis hacer un pull del repositorio, borrar los ficheros temporales de la carpeta *deployments* y volver a hacer un push al servidor para dejarlo en la situación original.
- Para más información podéis consultar el enlace <http://blog.arungupta.me/2014/11/deployment-pipeline-javase7-wildfly-arquillian-openshift-jenkins-techtip56/> pero si probáis el ejemplo que se expone tal cual está, no va a funcionar en WildFly 8.2. Es necesario modificar la clase *PersonDatabase* y cambiar la anotación `@Singleton` por `@ApplicationScope` para que el servidor pueda inyectar correctamente el bean.

Entrega

En esta sesión hay que presentar una copia del repositorio *bigbang* eliminando la configuración de git como ya hemos hecho en sesiones anteriores y una serie de ficheros: * Los ficheros *1gear.txt* y *2gear.txt* del segundo ejercicio. * El Log del balanceador HAProxy: *haproxy_ctld.log*. * Recoger en el fichero de *soluciones.txt* los pasos necesarios para lanzar las pruebas de arquillian y el log de compilación de las pruebas contra OpenShift.