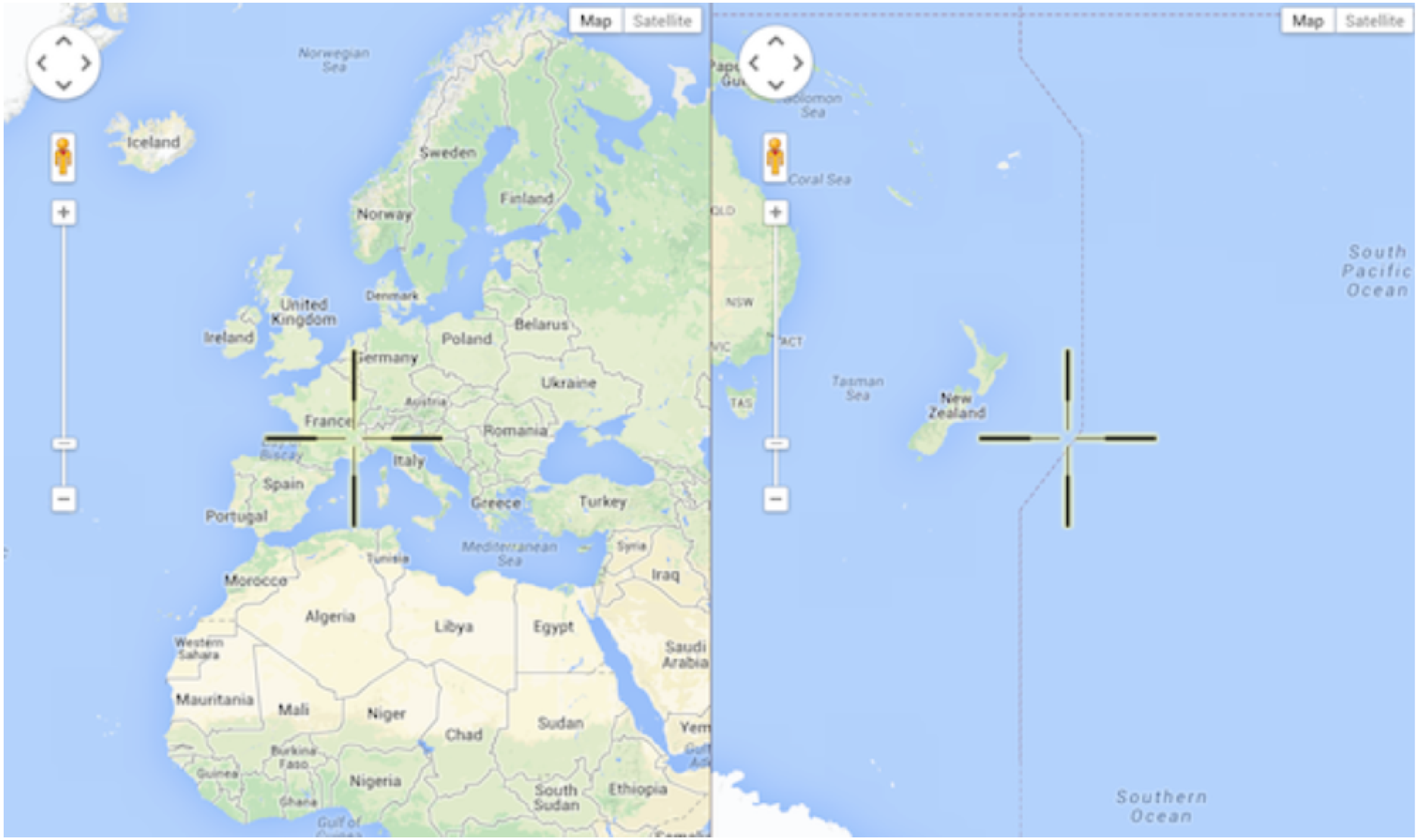
What's New in WildFly 8 ?

Jeff Mesnil · Red Hat · http://jmesnil.net/

Jeff Mesnil

- Senior software developer, Red Hat Inc.
- WildFly core developer
- Amateur Photographer
- Lives in France


Map
Satellite
Norwegian Sea
Iceland
Sweden
Finland
Norway
United Kingdom
Ireland
Denmark
Belarus
Poland
Germany
Ukraine
France
Austria
Romania
Spain
Portugal
Italy
Greece
Turkey
Mediterranean Sea
Syria
Iraq
Tunisia
Morocco
Algeria
Libya
Egypt
Saudi Arabia
Western Sahara
Mauritania
Mali
Niger
Chad
Sudan
Burkina Faso
Guinea
Nigeria
Ghana
South Sudan
Ethiopia
Gulf of Guinea
Map
Satellite
Coral Sea
NSW
ACT
VIC
TAS
Tasman Sea
New Zealand
South Pacific Ocean
Southern Ocean

zeroturnaround.com

The Great Java Application Server Debate with Tomcat, JBoss, GlassFish, Jetty and Liberty Profile

May 21, 2013 Simon Maple 27 comments

Tweet 171 Like 196 +1 105 Share 60 DZone 11 0

Part V – ...And The Best Application Server Award Goes To...

In case you were wondering, we did finally decide that one application server among those tested proved to win over the others...JBoss wins the award!

JBoss AS 7 (aka WildFly) **WINS THE AWARD!**

> *If we had to pick a winner, it would be JBoss. The only application server in the group whose score never dropped below a 4*
>
> — zeroturnaround.com

> *JBoss consistently performs very well in each category which is why it also shines in the developer profiles exercise*
>
> — zeroturnaround.com


WildFly
The new and improved JBoss Application Server!

`Objective:`

Understand the new features of WildFly 8 and revise some of the features carry forward from AS 7.x.

What is WildFly 8 ?

What is WildFly 8 ?

- Previously called “JBoss Application Server”

What is WildFly 8 ?

- **Upstream for Red Hat JBoss Enterprise Application Platform (JBoss EAP)**

What is WildFly 8 ?

- **Fast, lightweight, manageable**

What is WildFly 8 ?

- Developer friendly

What is WildFly 8 ?

- Supports Java EE standards and beyond

What is WildFly 8 ?

- Open source

WildFly 8 Main Features

WildFly 8 Main Features

- **Java EE7 support**

WildFly 8 Main Features

- High performance web server Undertow

WildFly 8 Main Features

- **Reduced port usage**

WildFly 8 Main Features

- **Role based access control**

WildFly 8 Main Features

- Auditing

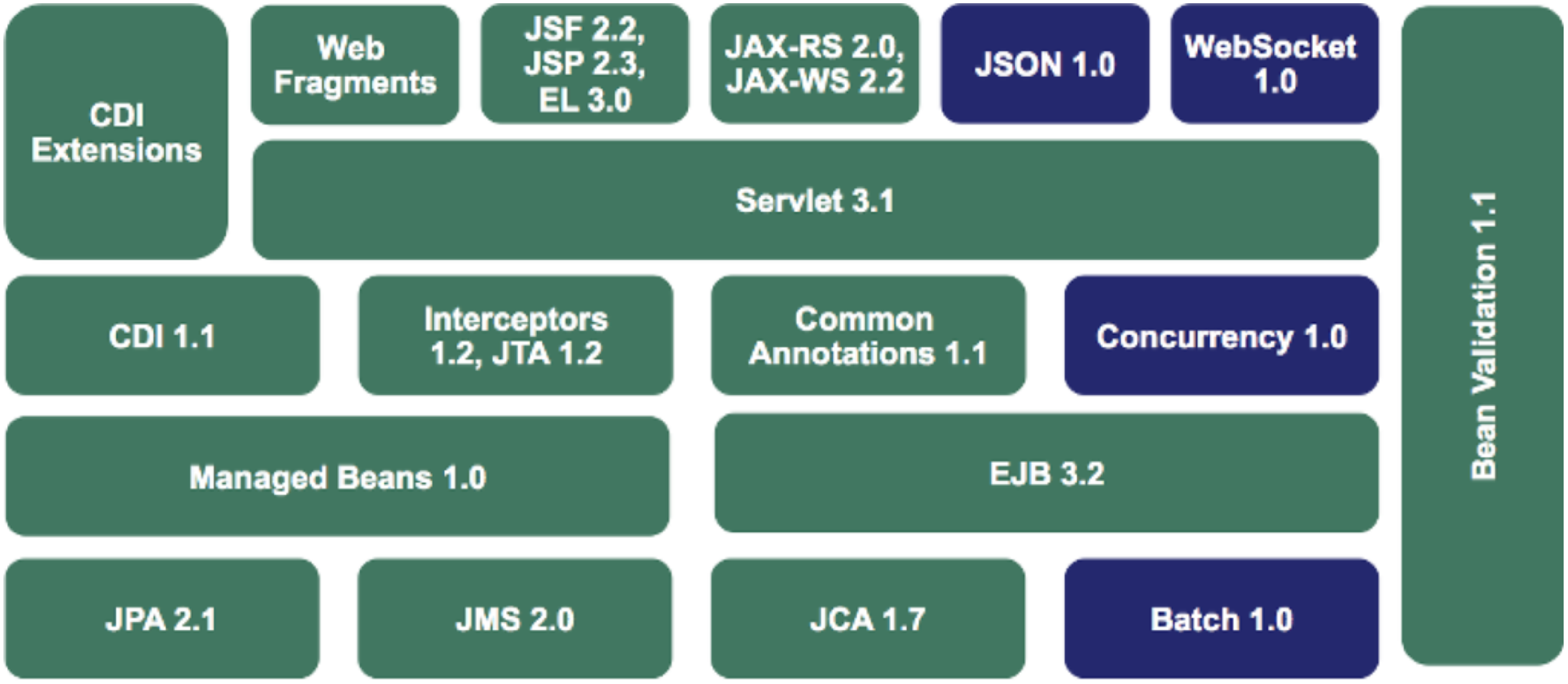
WildFly 8 Main Features

- Automated patching

WildFly 8 Main Features

- Minimalistic "core" distribution

WildFly: Java EE 7



ChatServer.java

```
@ServerEndpoint("/chat") (1)
public class ChatEndpoint {
    @OnMessage (2)
    public void message(String message,
                        Session client) (3)
                  throws IOException, EncodeException {
        for (Session peer : client.getOpenSessions()) {
            peer.getBasicRemote().sendText(message);
        }
    }
}
```

(1) Creates a WebSocket endpoint, defines the listening URL

(2) Marks the method that receives incoming WebSocket message

(3) Payload of the WebSocket message

job.xml

```
<job id="myJob" xmlns="http://xmlns.jcp.org/xml/ns/javaee" version="1.0">
    <step id="myStep" >
        <chunk item-count="3"> (1)
            <reader ref="myItemReader"/> (2)
            <processor ref="myItemProcessor"/> (3)
            <writer ref="myItemWriter"/> (4)
        </chunk>
    </step>
</job>
```

(1) Item-oriented processing, number of items in chunk

(2) Item reader for chunk processing

(3) Item processor for chunk processing

(4) Item writer for chunk processing

CreateJson.java

```
JsonObject jsonObject = Json.createObjectBuilder() (1)
                .add("apple", "red") (2)
                .add("banana", "yellow")
                .build(); (3)
StringWriter w = new StringWriter();
JsonWriter writer = Json.createWriter(w); (4)
writer.write(jsonObject);
```

(1) Creates a JSON object builder

(2) Adds a name/value pair to the JSON object

(3) Returns the JSON object associated with this builder

(4) Writes the JSON object to the writer

RunMyTask.java

```
public class MyTask implements Runnable {  (1)

    @Override
    public void run() {
        . . .
    }
}
```

```
@Resource(name = "DefaultManagedExecutorService")  (2)
ManagedExecutorService defaultExecutor;

executor.submit(new MyTask());  (3)
```

(1) `Runnable` or `Callable` tasks can be submitted

(2) `ManagedExecutor` is injected, default resource provided

(3) Submit the task

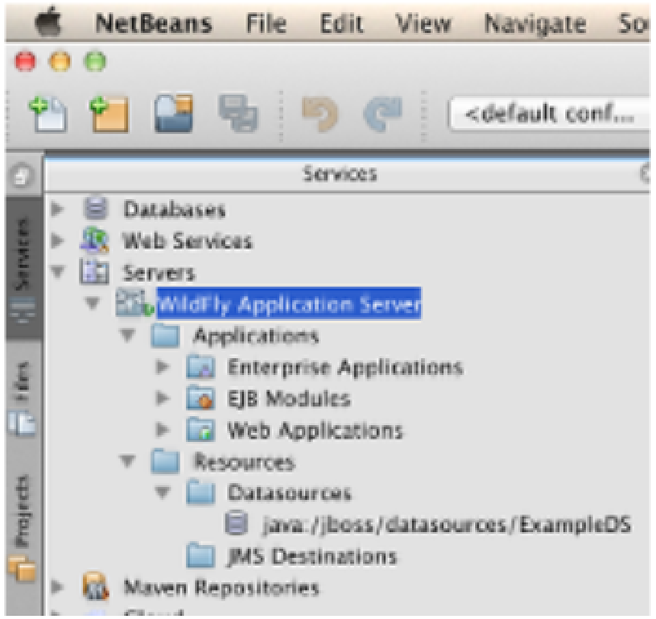
RunClient.java

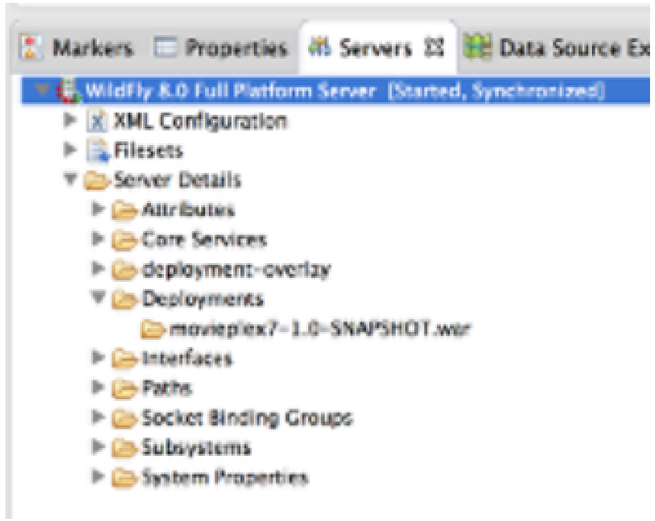
```
Client client = ClientBuilder.newClient(); (1)
WebTarget target = client.target("..."); (2)
target.register(Person.class);
Person p = target
            .path("{id}") (3)
            .resolveTemplate("id", "1")
            .request(MediaType.APPLICATION_XML) (4)
            .get(Person.class); (5)
```

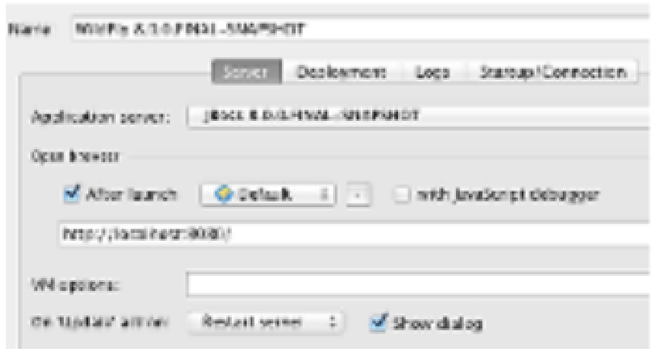
1. `ClientBuilder` is the entry point
2. Build a new web resource target, specifies the path
3. Sub resource URI
4. Define the accepted response media types
5. Call HTTP GET, specify the type of resource

More on that later…

WildFly: Java EE 7 IDEs







WildFly: New Web Server (Undertow)

WildFly: New Web Server (Undertow)

- Flexible and high-performance

WildFly: New Web Server (Undertow)

- Blocking / non-blocking based on NIO

WildFly: New Web Server (Undertow)

- Composition/handler based architecture

WildFly: New Web Server (Undertow)

- Lightweight & fully embeddable

WildFly: New Web Server (Undertow)

- Supports Servlet 3.1 & HTTP upgrade

WildFly: New Web Server (Undertow)

- `mod_cluster` supported

WildFly: New Web Server (Undertow)

- http://undertow.io

WildFly New Web Server: Undertow

NonBlockingHandler.java

```
Undertow.builder() (1)
  .addListener(8080, "localhost")
  .setHandler(new HttpHandler() { (2)
    @Override
    public void handleRequest(final HttpServerExchange exchange)
      throws Exception {
      exchange.getResponseHeaders()
              .put(Headers.CONTENT_TYPE, "text/plain");
      exchange.getResponseSender()
              .send("Hello World");
    }
}).build().start(); (3)
```

(1) Same API used for WildFly integration, fluent builder API

(2) Can create multiple handlers

(3) Build the server and start to listen for connections

Undertow benchmarks

```
techempower@lg01:~$ wrk -d 30 -c 256 -t 40 http://10.0.3.2:8080/byte
Running 30s test @ http://10.0.3.2:8080/byte
  40 threads and 256 connections
  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency   247.05us    3.52ms 624.37ms   99.90%
    Req/Sec    27.89k     6.24k   50.22k    71.15%
  31173283 requests in 29.99s, 3.83GB read
  Socket errors: connect 0, read 0, write 0, timeout 9
Requests/sec: 1039305.27
Transfer/sec:    130.83MB
```

This is output from Wrk testing a single server running Undertow using conditions similar to Google's test (1-byte response body, no HTTP pipelining, no special request headers) **1.039 million requests per second**.

http://www.techempower.com/blog/2014/03/04/one-million-http-rps-without-load-balancing-is-easy/

WildFly: Port Reduction

WildFly: Port Reduction

- HTTP Upgrade to reduce the number of ports in the default installation to just two
  - 8080 for applications with JNDI, EJB & JMS multiplexed
  - 9990 for management, for both HTTP/JSON & Native API

WildFly: Port Reduction

- Only overhead is the initial HTTP Upgrade request/response

WildFly: Role Based Access Control

WildFly: Role Based Access Control

- Pre-defined administrative and privileged Roles
  - Monitor, Operator, Maintainer, Deployer, Administrator, Auditor, Super User

WildFly: Role Based Access Control

- Roles is a set of Permissions

WildFly: Role Based Access Control

- Permissions specify which **Actions** (lookup, read, write) are allowed on resources

WildFly: Role Based Access Control

- Users or Groups are defined in Roles

WildFly: Administrative Audit Logging

WildFly: Administrative Audit Logging

- Logging of connection/authentication events

WildFly: Administrative Audit Logging

- Logging of management operations

WildFly: Administrative Audit Logging

- **Log message as JSON records**

WildFly: Administrative Audit Logging

- Audit logging handlers
  - Local file
  - Syslog (UDP / TCP / TLS)

WildFly: Automated Patching

WildFly: Automated Patching

- Allows libraries and configuration updates in an installation

WildFly: Automated Patching

- Patches are zip bundles with updates and metadata

WildFly: Automated Patching

- **Multiple one-off patches can be applied; invalidated by the next point/CP release**

WildFly: Automated Patching

- Rollbacks are possible

WildFly: Minimalistic "core" distribution

WildFly: Minimalistic "core" distribution

- 15 MB download

WildFly: Minimalistic "core" distribution

- **Rich management layer**

WildFly: Minimalistic "core" distribution

- **Fully concurrent service container with advanced capabilities**

WildFly: Minimalistic "core" distribution

- Modular class loading which enables multi-tenancy of applications

WildFly: Minimalistic "core" distribution

- Pluggable hot deployment layer

WildFly: Minimalistic "core" distribution

- Built-in lightweight web server

WildFly: Miscellaneous

WildFly: Miscellaneous

- Improved JDK8 compatibility

WildFly: Miscellaneous

- **RESTEasy 3**

WildFly: Miscellaneous

- **Hibernate search**

WildFly: Miscellaneous

- Per-deployment security permissions

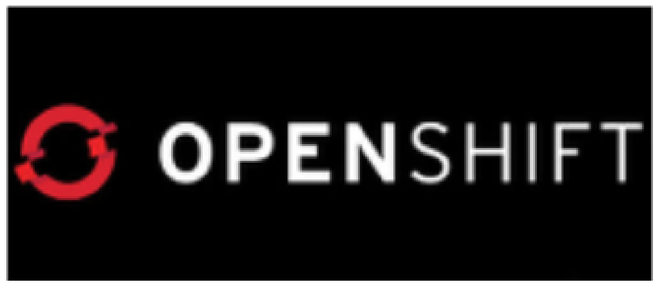
WildFly: Miscellaneous

- New public clustering API

WildFly: Miscellaneous

- Pruned: CMP, JAX-RPC, JSR 88

WildFly: In the cloud



Carry forward from AS 7.x

- Standalone and Managed Domain
- Centralized Administration
  - Command Line Interface (jboss-cli)
  - Admin Console
  - Configuration files

Standalone and Managed Domain

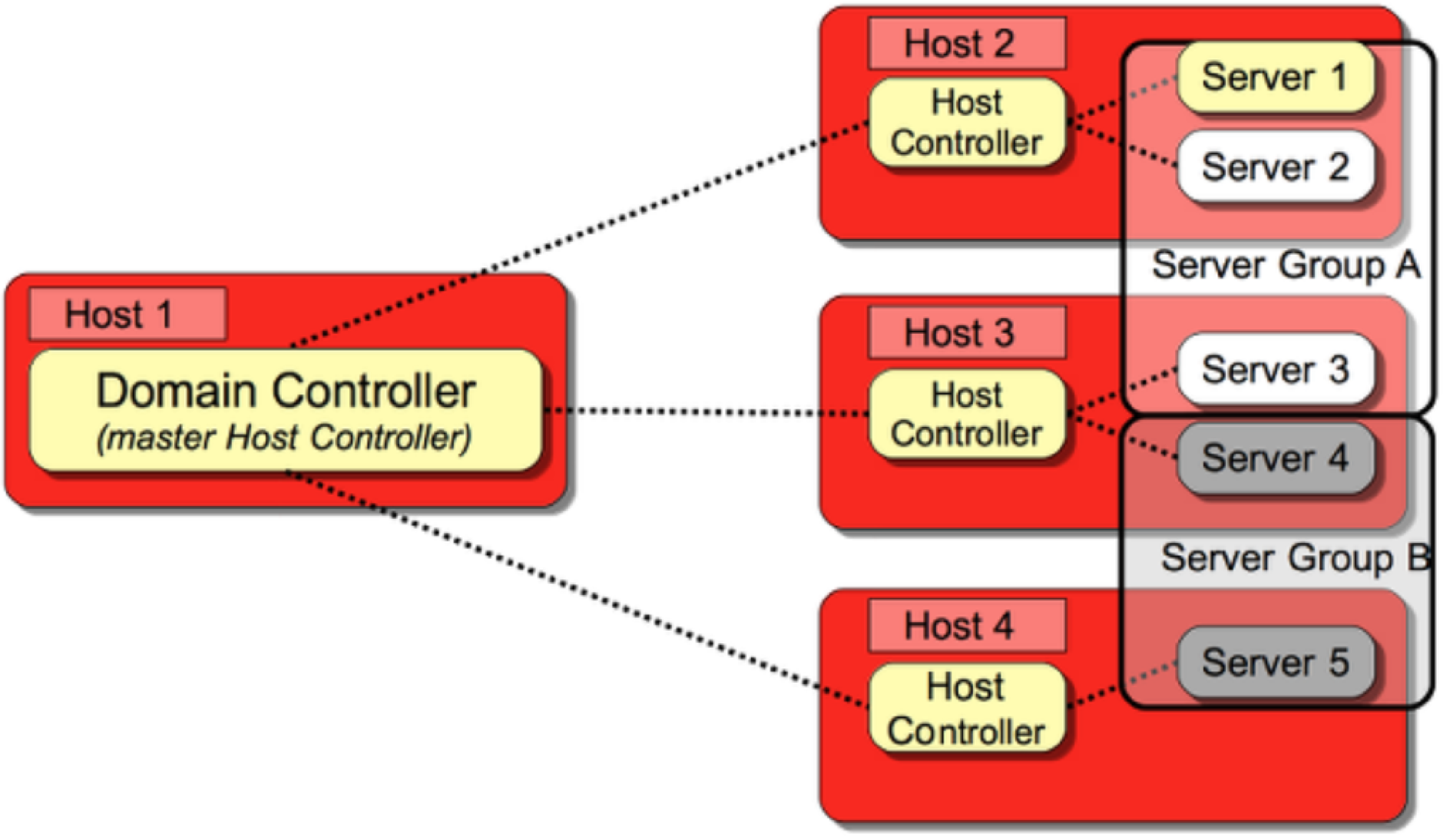
Standalone and Managed Domain

- **Standalone**: single independent instance

Standalone and Managed Domain

- **Managed domain**: manage multiple WildFly instances from a single control point
  - Host controller
  - Domain controller
  - Server group
  - Server

Managed Domain



Command Line Interface

Command Line Interface

- `jboss-cli.sh|bat`

Command Line Interface

- Connects to standalone instance or Domain controller

Command Line Interface

- Interactive mode: *nix-style shell
  - Contextual command and resource-tab completion

Command Line Interface

- Non-interactive mode: commands in files

Command Line Interface

- High-level compound operations

Command Line Interface

- **Persistent changes**

Admin Console

Admin Console

- **Simple**

Admin Console

- **Fast**

Admin Console

- **Lightweight**

Admin Console

- **Avoids XML configuration**

Admin Console

- **Single instance and domains**

Admin Console

- Mostly configuration, basic monitoring
  - Not a Red Hat JBoss Operations Network replacement

- Java EE 7 compliant
- lightweight
- managable
- highly scalable
- open source
- application server

Now available!

Dive into JMS 2.0

- Simplified API
- JMSContext Injection
- JMS Resource Definitions
- Shared Subscription
- Sending Messages Asynchronously
- MDB Configuration Properties

JMS 2.0

- JMS 1.1 - released in March 2002
- Java 7
- New Simplified API
- Compatible with JMS 1.1
- Fluent API
- Runtime exceptions

JMS 2.0 Example

SendMessage.java

```
@JMSDestinationDefinition(name="myQueue", interfaceName="javax.jms.Queue") (1)

@Resource(mappedName="myQueue")
Queue queue; (2)

@Inject
private JMSContext context; (3)
```

```
context.createProducer().send(queue, "Hello, JMS 2.0!"); (4)
```

1. Create destination resource during deployment
2. Fetch the queue resource
3. Main interface of the simplified API
4. Fluent builder API, runtime exceptions

JMS 2.0 Example

ReceiveMessage.java

```
@Resource(mappedName="myQueue")
Queue queue;

@Inject
private JMSContext context;

JMSConsumer consumer = context.createConsumer(queue); (1)
String text = consumer.receiveBody(String.class, 5000); (2)
// => "Hello, JMS 2.0!"
```

(1) Fluent builder API, runtime exceptions

(2) No cast required to receive a text message

JMS Destination Definitions

JMSDefinitions.java

```
@JMSDestinationDefinition(name="myQueue",                    (1)
        interfaceName="javax.jms.Queue",                     (2)
        properties = { "durable=false" }                     (3)
)

@JMSDestinationDefinition(name="myTopic",
        interfaceName="javax.jms.Topic"                      (4)
)
```

(1) Name of the destination

(2) JMS `Queue`

(3) Provider-specific properties

(4) JMS `Topic`

JMS ConnectionFactory Definitions

JMSDefinitions.java

```
@JMSConnectionFactoryDefinition(name="myFactory", (1)
        interfaceName = "javax.jms.QueueConnectionFactory", (2)
        minPoolSize = 5, (3)
        maxPoolSize = 20,
        clientId = "myclientID", (4)
        properties = { "initial-connect-attempts=5" } (5)
)
```

(1) Name of the JMS `ConnectionFactory`

(2) Type of the connection factory

(3) Min/Max size of the connection pool

(4) JMS properties

(5) Provider-specific properties

JMSContext

- Encapsulates both a JMS 1.1 `Connection` and `Session`
- `AutoCloseable`
- Auto-started
- Injectable in Java EE Web or EJB Containers

JMS Client with JMSContext

JMSClient.java

```
ConnectionFactory cf = (ConnectionFactory)namingContext.lookup("..."); (1)
Destination destination = (Destination)namingContext.lookup("...");

try (JMSContext context = cf.createContext(userName, password)) { (2)
  context.createProducer().send(destination, "hello");

  JMSConsumer consumer = context.createConsumer(destination); (3)
  String response = consumer.receiveBody(String.class, 5000);
}
```

1. Usual JNDI lookup to get the JMS Connection Factory & Destinations
2. `try-with-resources` statement to auto close the context
3. Context is automatically started when a consumer is created

JMS Context Injection

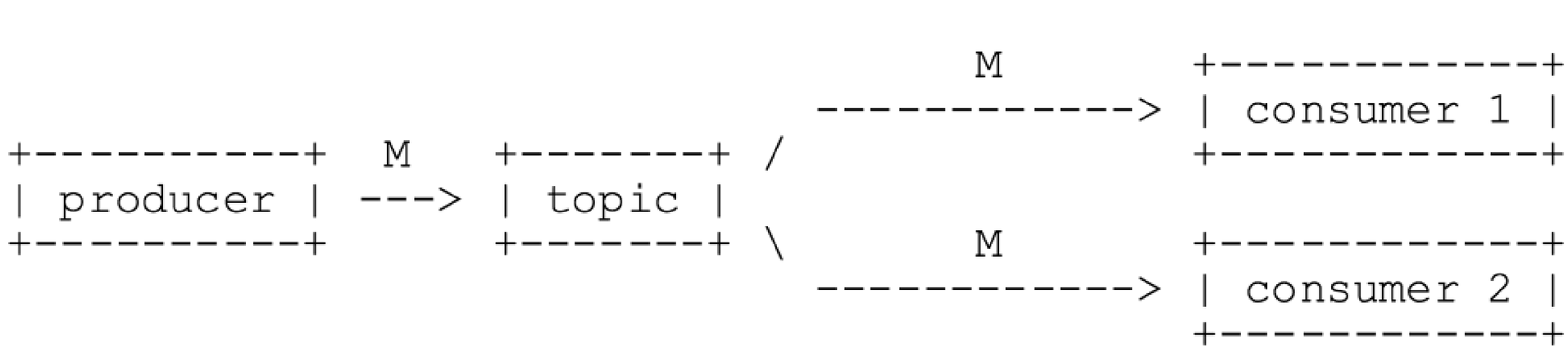
JMSClient.java

```
@Inject
// @JMSConnectionFactory("java:comp/DefaultJMSConnectionFactory") (1)
@JMSConnectionFactory("myFactory") (2)
@JMSPasswordCredential(userName="guest",password="password") (3)
@JMSSessionMode(JMSContext.AUTO_ACKNOWLEDGE) (4)
private JMSContext context;
```

1. Java EE 7 Default JMS Connection Factory…
2. … or you use your own
3. User credentials
4. Acknowledgement mode / Transactional behaviour

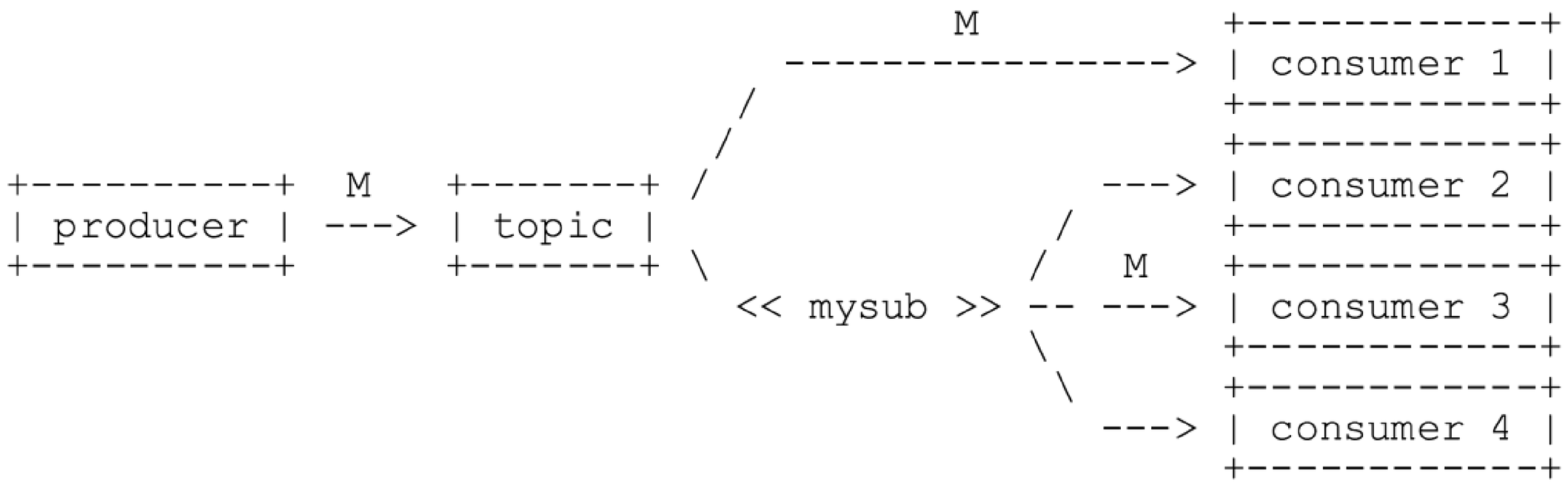
Shared Subscription

- **Restriction in JMS 1.1**
  - **No more than one consumer for a topic subscription**
    - **Impossible in Java SE application / multiple JVMs**
    - **Possible in a Java EE application using a pool of MDB**



Shared Subscription

- **Multiple consumers on the same topic subscription**



- **Durable / Non-durable shared consumers**

Shared Subscription

SharedSubscription.java

```
@Resource(mappedName="myTopic")
Topic topic;

@Inject
private JMSContext context;

String subscription = "...";

JMSConsumer consumer = context.createSharedConsumer(topic, subscription); (1)

context.setClientID("...");
JMSConsumer durableConsumer = context.createSharedDurableConsumer(topic,
subscription); (2)
```

(1) Non-durable shared consumer (topic + subscription)

(2) Durable shared consumer (topic + subscription + clientID)

Sending Messages Asynchronously

SendAsynchronously.java

```
JMSProducer producer = context.createProducer()
    .setAsync(new CompletionListener() {                    (1)
        @Override
        public void onCompletion(Message message) { }       (2)

        @Override
        public void onException(Message message, Exception exception) { }   (3)
    });
producer.send(destination, "Hello, Async!");                (4)
```

(1) Callback for completion

(2) Called when a message was sent successfully

(3) Called when a problem occurred and prevent the message to be sent

(4) Send the message asynchronously

MDB Configuration Properties

MDB Configuration Properties

- Java EE 6
  - `acknowledgeMode`
  - `messageSelector`
  - `destinationType`
  - `subscriptionDurability`

MDB Configuration Properties

- Java EE7
  - `destinationLookup`
  - `connectionFactoryLookup`
  - `clientId`
  - `subscriptionName`

MDB Configuration Properties

MDB.java

```
@MessageDriven(name = "MyMDB", activationConfig = {
    @ActivationConfigProperty(propertyName = "connectionFactoryLookup", (1)
                              propertyValue = "jms/MyConnectionFactory"),
    @ActivationConfigProperty(propertyName = "destinationType",
                              propertyValue = "javax.jms.Queue"),
    @ActivationConfigProperty(propertyName = "destinationLookup", (2)
                              propertyValue = "myQueue"),
    @ActivationConfigProperty(propertyName = "acknowledgeMode",
                              propertyValue = "Auto-acknowledge") })
public class MyMDB implements MessageListener {

    public void onMessage(Message message) { ... }
}
```

(1) standard portable property to lookup the connection factory

(2) standard portable property to lookup the destination

Q&A ?

Q&A ?

- **Thanks!**

References

- WildFly - http://wildfly.org, http://github.com/wildfly, @WildFlyAS
- JBoss EAP 6.2 - http://redhat.com/jboss
- Java EE 7 samples - https://github.com/javaee-samples/javaee7-samples
- Slides generated with Asciidoctor and DZSlides backend
- Original slide template - Dan Allen & Sarah White

Jeff Mesnil

http://jmesnil.net/