

# **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**

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

May 21, 2013

Simon Maple

27 comments



## 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!



**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



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 ?

- Previously called “JBoss Application Server”
- Upstream for Red Hat JBoss Enterprise Application Platform (JBoss EAP)

# What is WildFly 8 ?

- Previously called “JBoss Application Server”
- Upstream for Red Hat JBoss Enterprise Application Platform (JBoss EAP)
- Fast, lightweight, manageable

# What is WildFly 8 ?

- Previously called “JBoss Application Server”
- Upstream for Red Hat JBoss Enterprise Application Platform (JBoss EAP)
- Fast, lightweight, manageable
- Developer friendly

# What is WildFly 8 ?

- Previously called “JBoss Application Server”
- Upstream for Red Hat JBoss Enterprise Application Platform (JBoss EAP)
- Fast, lightweight, manageable
- Developer friendly
- Supports Java EE standards and beyond

# **What is WildFly 8 ?**

- Previously called “JBoss Application Server”
- Upstream for Red Hat JBoss Enterprise Application Platform (JBoss EAP)
- Fast, lightweight, manageable
- Developer friendly
- Supports Java EE standards and beyond
- Open source

# **WildFly 8 Main Features**

# WildFly 8 Main Features

- Java EE7 support

# WildFly 8 Main Features

- Java EE7 support
- High performance web server **Undertow**

# WildFly 8 Main Features

- Java EE7 support
- High performance web server **Undertow**
- Reduced port usage

# WildFly 8 Main Features

- Java EE7 support
- High performance web server **Undertow**
- Reduced port usage
- Role based access control

# WildFly 8 Main Features

- Java EE7 support
- High performance web server **Undertow**
- Reduced port usage
- Role based access control
- Auditing

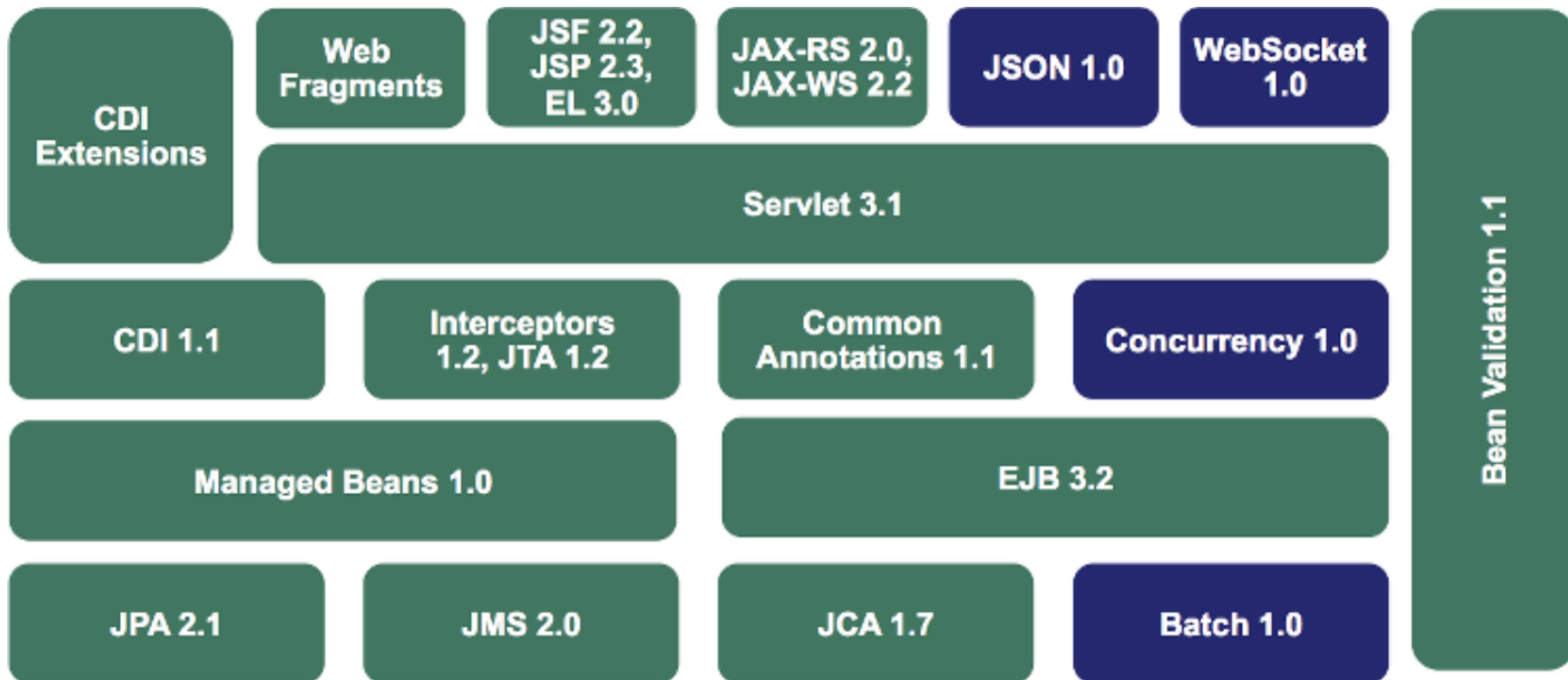
# WildFly 8 Main Features

- Java EE7 support
- High performance web server **Undertow**
- Reduced port usage
- Role based access control
- Auditing
- Automated patching

# WildFly 8 Main Features

- Java EE7 support
- High performance web server **Undertow**
- Reduced port usage
- Role based access control
- Auditing
- Automated patching
- Minimalistic "core" distribution

# WildFly: Java EE 7



# WildFly: Java EE 7 WebSocket

## 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

# WildFly: Java EE 7 Batch

## 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

# WildFly: Java EE 7 JSON

## 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

# WildFly: Java EE 7 Concurrency

## 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

# WildFly: Java EE 7 JAX-RS

## RunClient.java

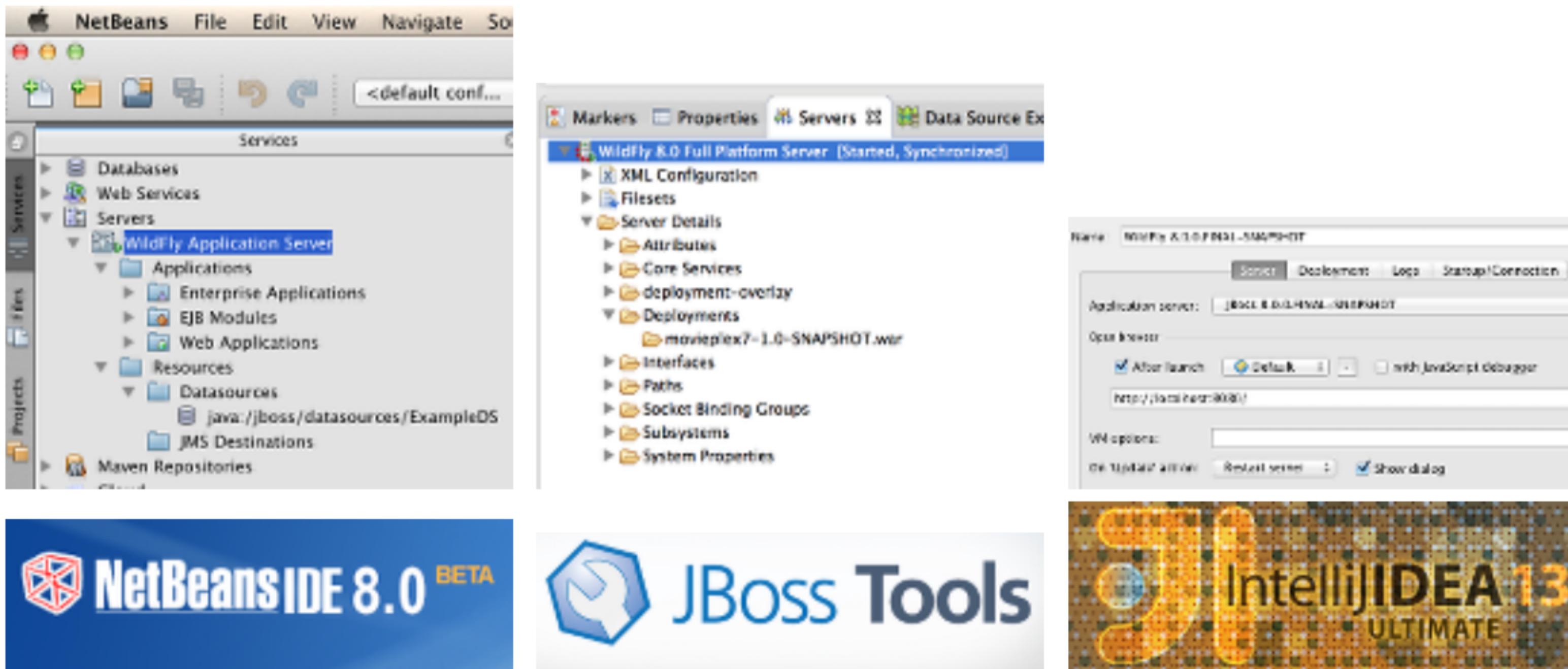
```
Client client = ClientBuilder.newClient(); ①  
WebTarget target = client.target("..."); ②  
target.register(Person.class);  
Person p = target  
    .path("{id}") ③  
    .resolveTemplate("id", "1")  
    .request(MediaType.APPLICATION_XML) ④  
    .get(Person.class); ⑤
```

- 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

**WildFly: Java EE 7 JMS**

**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)**

- Flexible and high-performance
- Blocking / non-blocking based on NIO

# **WildFly: New Web Server (Undertow)**

- **Flexible and high-performance**
- **Blocking / non-blocking based on NIO**
- **Composition/handler based architecture**

# WildFly: New Web Server **(Undertow)**

- Flexible and high-performance
- Blocking / non-blocking based on NIO
- Composition/handler based architecture
- Lightweight & fully embeddable

# WildFly: New Web Server **(Undertow)**

- **Flexible and high-performance**
- **Blocking / non-blocking based on NIO**
- **Composition/handler based architecture**
- **Lightweight & fully embeddable**
- **Supports Servlet 3.1 & HTTP upgrade**

# WildFly: New Web Server **(Undertow)**

- Flexible and high-performance
- Blocking / non-blocking based on NIO
- Composition/handler based architecture
- Lightweight & fully embeddable
- Supports Servlet 3.1 & HTTP upgrade
- **mod\_cluster supported**

# WildFly: New Web Server **(Undertow)**

- Flexible and high-performance
- Blocking / non-blocking based on NIO
- Composition/handler based architecture
- Lightweight & fully embeddable
- Supports Servlet 3.1 & HTTP upgrade
- `mod_cluster` supported
- <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

- 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
- 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

- Pre-defined administrative and privileged Roles
  - Monitor, Operator, Maintainer, Deployer, Administrator, Auditor, Super User
- Roles is a set of Permissions

# WildFly: Role Based Access Control

- Pre-defined administrative and privileged Roles
  - Monitor, Operator, Maintainer, Deployer, Administrator, Auditor, Super User
- Roles is a set of Permissions
- Permissions specify which Actions (lookup, read, write) are allowed on resources

# WildFly: Role Based Access Control

- Pre-defined administrative and privileged Roles
  - Monitor, Operator, Maintainer, Deployer, Administrator, Auditor, Super User
- Roles is a set of Permissions
- Permissions specify which Actions (lookup, read, write) are allowed on resources
- 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 connection/authentication events
- Logging of management operations

# WildFly: Administrative Audit Logging

- Logging of connection/authentication events
- Logging of management operations
- Log message as JSON records

# WildFly: Administrative Audit Logging

- Logging of connection/authentication events
- Logging of management operations
- Log message as JSON records
- 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

- Allows libraries and configuration updates in an installation
- Patches are zip bundles with updates and metadata

# WildFly: Automated Patching

- Allows libraries and configuration updates in an installation
- Patches are zip bundles with updates and metadata
- Multiple one-off patches can be applied; invalidated by the next point/CP release

# WildFly: Automated Patching

- Allows libraries and configuration updates in an installation
- Patches are zip bundles with updates and metadata
- Multiple one-off patches can be applied; invalidated by the next point/CP release
- Rollbacks are possible

# **WildFly: Minimalistic "core" distribution**

# **WildFly: Minimalistic "core" distribution**

- **15 MB download**

# **WildFly: Minimalistic "core" distribution**

- **15 MB download**
- **Rich management layer**

# **WildFly: Minimalistic "core" distribution**

- **15 MB download**
- **Rich management layer**
- **Fully concurrent service container with advanced capabilities**

# **WildFly: Minimalistic "core" distribution**

- **15 MB download**
- **Rich management layer**
- **Fully concurrent service container with advanced capabilities**
- **Modular class loading which enables multi-tenancy of applications**

# **WildFly: Minimalistic "core" distribution**

- **15 MB download**
- **Rich management layer**
- **Fully concurrent service container with advanced capabilities**
- **Modular class loading which enables multi-tenancy of applications**
- **Pluggable hot deployment layer**

# **WildFly: Minimalistic "core" distribution**

- **15 MB download**
- **Rich management layer**
- **Fully concurrent service container with advanced capabilities**
- **Modular class loading which enables multi-tenancy of applications**
- **Pluggable hot deployment layer**
- **Built-in lightweight web server**

# **WildFly: Miscellaneous**

# WildFly: Miscellaneous

- Improved JDK8 compatibility

# WildFly: Miscellaneous

- Improved JDK8 compatibility
- RESTEasy 3

# WildFly: Miscellaneous

- Improved JDK8 compatibility
- RESTEasy 3
- Hibernate search

# WildFly: Miscellaneous

- Improved JDK8 compatibility
- RESTEasy 3
- Hibernate search
- Per-deployment security permissions

# WildFly: Miscellaneous

- Improved JDK8 compatibility
- RESTEasy 3
- Hibernate search
- Per-deployment security permissions
- New public clustering API

# WildFly: Miscellaneous

- Improved JDK8 compatibility
- RESTEasy 3
- Hibernate search
- Per-deployment security permissions
- New public clustering API
- 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**

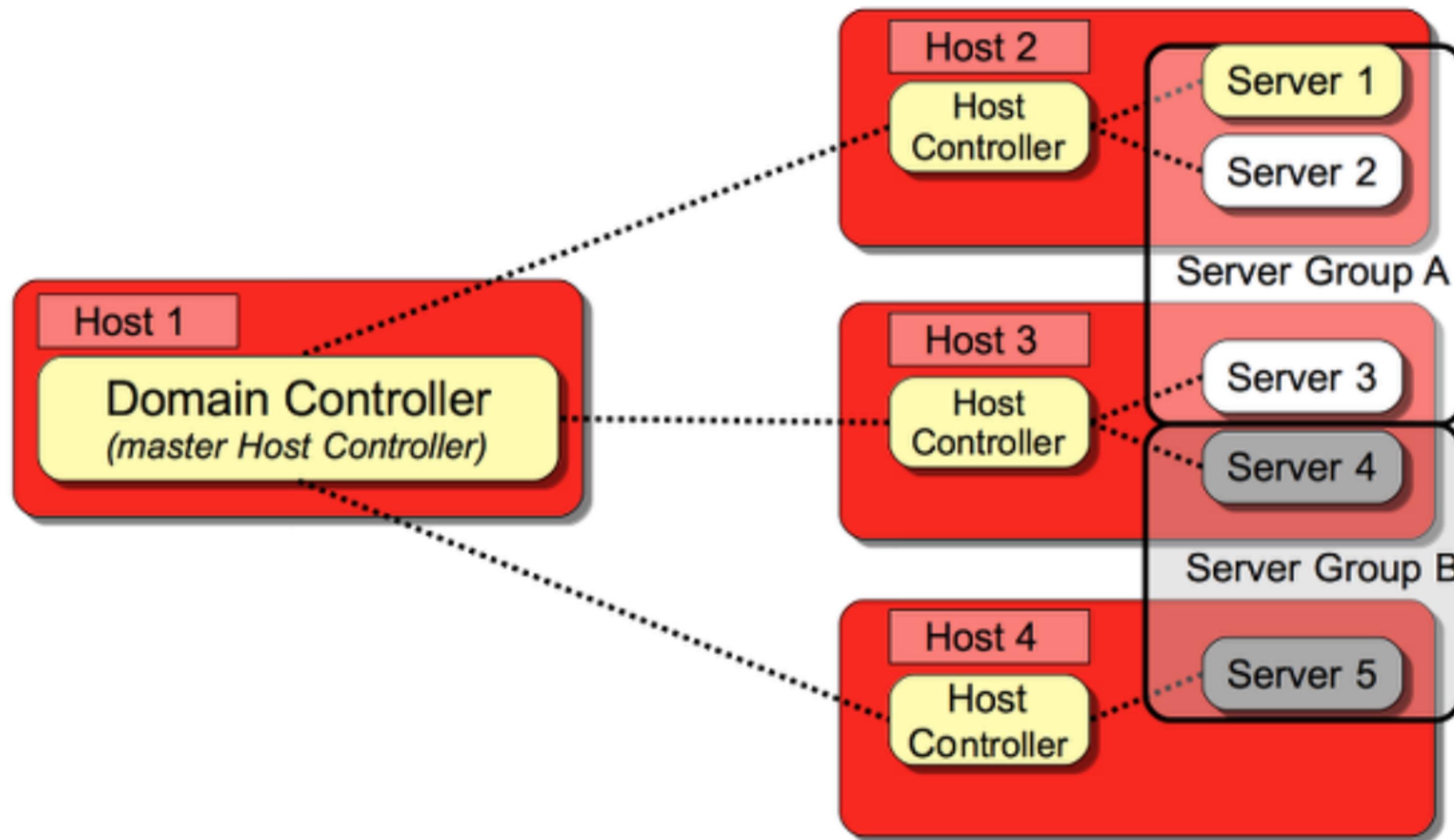
# Standalone and Managed Domain

- **Standalone: single independent instance**

# Standalone and Managed Domain

- **Standalone:** single independent instance
- **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

- `jboss-cli.sh|bat`
- **Connects to standalone instance or Domain controller**

# Command Line Interface

- `jboss-cli.sh|bat`
- Connects to standalone instance or Domain controller
- Interactive mode: \*nix-style shell
  - Contextual command and resource-tab completion

# Command Line Interface

- `jboss-cli.sh|bat`
- **Connects to standalone instance or Domain controller**
- **Interactive mode: \*nix-style shell**
  - Contextual command and resource-tab completion
- **Non-interactive mode: commands in files**

# Command Line Interface

- `jboss-cli.sh|bat`
- **Connects to standalone instance or Domain controller**
- **Interactive mode: \*nix-style shell**
  - Contextual command and resource-tab completion
- **Non-interactive mode: commands in files**
- **High-level compound operations**

# Command Line Interface

- `jboss-cli.sh|bat`
- **Connects to standalone instance or Domain controller**
- **Interactive mode: \*nix-style shell**
  - Contextual command and resource-tab completion
- **Non-interactive mode: commands in files**
- **High-level compound operations**
- **Persistent changes**

# **Admin Console**

# Admin Console

- Simple

# Admin Console

- Simple
- Fast

# Admin Console

- Simple
- Fast
- Lightweight

# Admin Console

- Simple
- Fast
- Lightweight
- **Avoids XML configuration**

# Admin Console

- Simple
- Fast
- Lightweight
- Avoids XML configuration
- Single instance and domains

# Admin Console

- Simple
- Fast
- Lightweight
- Avoids XML configuration
- Single instance and domains
- Mostly configuration, basic monitoring
  - Not a Red Hat JBoss Operations Network



- Java EE 7 compliant
- lightweight
- manageable
- 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",  
    interfaceName="javax.jms.Queue",  
    properties = { "durable=false" } 1  
)  
  
@JMSDestinationDefinition(name="myTopic",  
    interfaceName="javax.jms.Topic" 2  
)  
)
```

① Name of the destination

② JMS Queue

③ Provider-specific properties

④ JMS Topic

# JMS ConnectionFactory Definitions

## JMSDefinitions.java

```
@JMSSConnectionFactoryDefinition(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

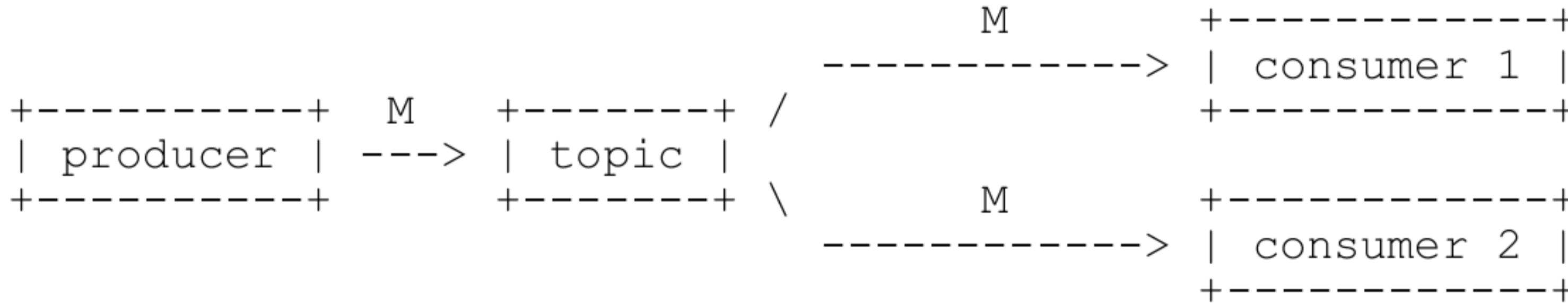
## JMSClient.java

```
@Inject  
// @JMSConnectionFactory("java:comp/DefaultJMSSConnectionFactory") 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

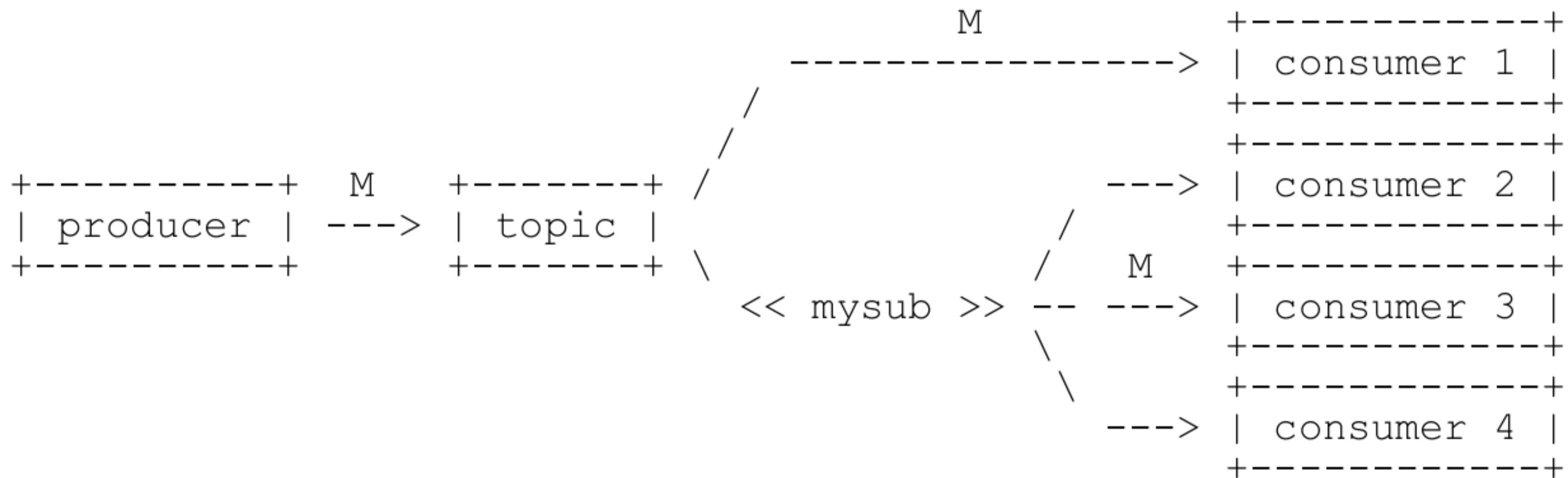
# Shared Subscription

- **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 EE 6**

- **acknowledgeMode**
- **messageSelector**
- **destinationType**
- **subscriptionDurability**

- **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/>