Why is WildFly 8 so ^$(@# good ?

Arun Gupta · Red Hat

Arun Gupta

— Director, Developer Advocacy, Red Hat Inc.

— O'Reilly and McGraw Hill author

— Fitness freak

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!

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


JBoss Application Server
has a new name...
and it's even
@#$%ing
faster !
WildFly
Learn more about WildFly >

`Objective:`

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

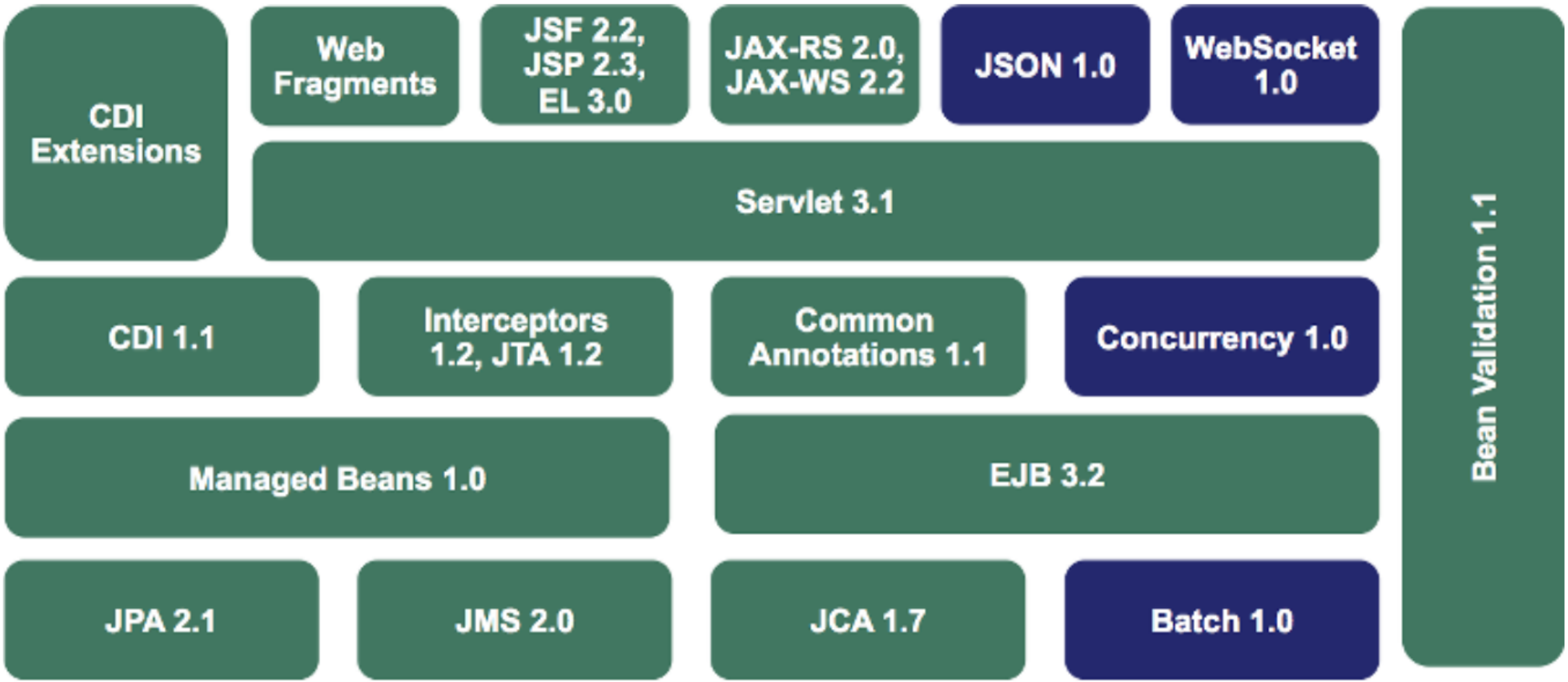
What is WildFly 8 ?

→ 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

→ Java EE7 support
→ High performance web server Undertow
→ Reduced port usage
→ Role based administration control & auditing
→ Automated patching

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

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

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

`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, specifies the type of resource

WildFly: Java EE 7 JMS

SendMessage.java

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

@Resource(mappedName="myQueue")
Queue syncQueue;

@Inject
// @JMSConnectionFactory("java:comp/DefaultJMSConnectionFactory") (2)
private JMSContext context; (3)
```

```
context.createProducer().send(syncQueue, "..."); (4)
```

1. Creates the destination resource during deployment
2. Default JMS connection factory
3. Main interface of the simplified API
4. Fluent builder API

WildFly: New web server (Undertow)

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

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) Start the handler in JVM

WildFly: Port reduction

→ HTTP Upgrade to reduce the number of ports in the default installation to just two
→ (indent) 8080 for applications with JNDI and EJB multiplexed
→ (indent) 9990 for management, for both HTTP/JON & Native API

→ Only overhead is the initial HTTP Upgrade request/response

WildFly: Role based access control

→ Management model consists of Users, Roles, Permissions
→ (indent) Pre-defined administrative and privileged roles
→ (indent) Users are defined in Roles
→ (indent) Roles is a set of Permissions
→ (indent) Permissions specify which Actions (lookup, read, write) are allowed on resources

WildFly: Administrative audit logging

→ Logging of connection/authentication events
→ Logging of management operations
→ Log message as JSON records
→ Audit logging handlers
→ (indent) Local file
→ (indent) Syslog (UDP / TCP / TLS)

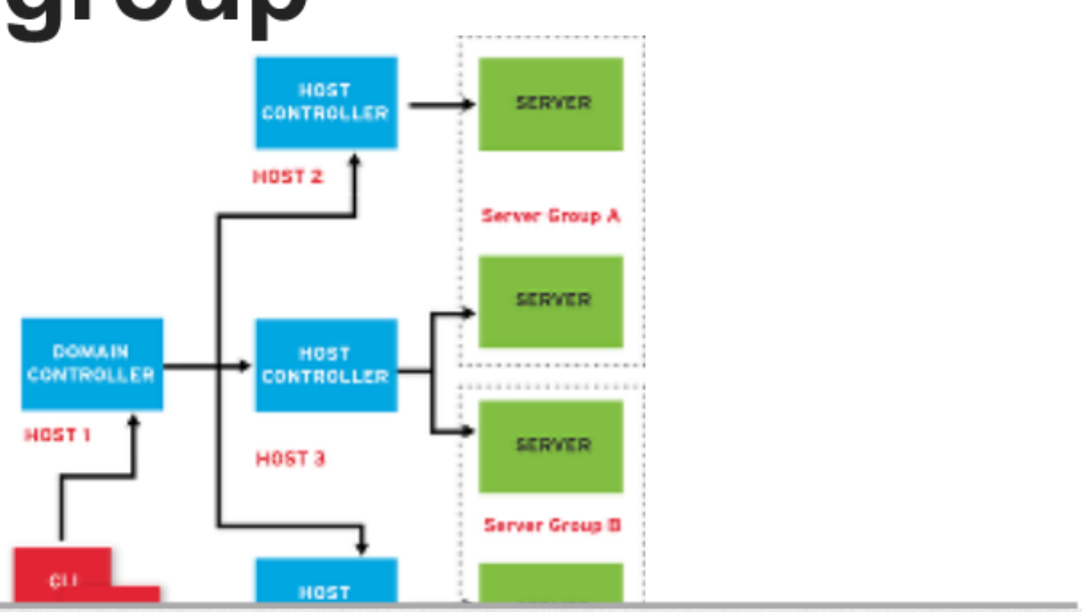
WildFly: Automated patching

→ 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

Carry forward from AS 7.x

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

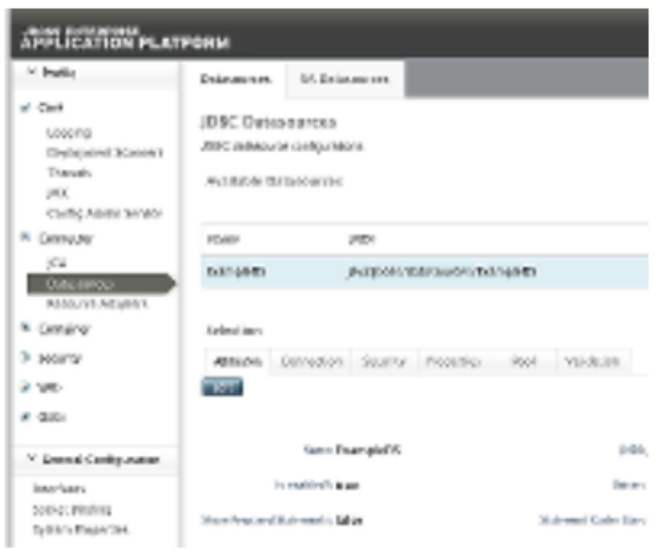
→ **Standalone**: single independent instance
→ **Managed domain**: manage multiple WildFly instances from a single control point
→ (indent) Host controller
→ (indent) Domain controller
→ (indent) Server group



Command Line Interface

→ `jboss-cli.sh|bat`
→ Connects to standalone instance or Domain controller
→ *nix-style shell syntax, resources as files
→ Contextual command and resource-tab completion
→ High-level compound operations
→ Persistent changes

Admin Console



→ Simple

→ Fast

→ Lightweight

→ Avoids XML configuration

→ Single instance and domains

→ Mostly configuration, basic monitoring

→ (indent) Not a Red Hat JBoss Operations

summary

WildFly is:
* Java EE 7 compliant
* lightweight
* manageable
* highly scalable
* open source
* application server

Now available!

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

Arun Gupta

@arungupta