Java EE 7 and HTML5: Developing for the Cloud

Arun Gupta, Java EE & GlassFish Guy
blogs.oracle.com/arungupta, @arungupta
The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle’s products remains at the sole discretion of Oracle.
Java EE 6 – Key Statistics

- 40+ Million Java EE 6 Component Downloads
- #1 Choice for Enterprise Developers
- #1 Application Development Platform
- Fastest implementation of a Java EE release
Top Ten Features in Java EE 6

1. EJB packaging in a WAR
2. Servlet and CDI extension points
3. Optional `web.xml`
4. Type-safe dependency injection
5. CDI Events
6. JSF standardizing on Facelets
7. EJBContainer API
8. @Schedule
9. EJB No Interface View
10. Web Profile
Today’s Cloud Offerings are all Proprietary

Infrastructure as a Service

Platform as a Service

Software as a Service
We’re moving Java EE into the Cloud!
Java EE 7 and 8 – Focus Areas

**Cloud**
- Provisioning
- Elastic & Autonomic Scalability
- Multi-Tenancy

**Modularity**
- Building on Jigsaw
- Focus on OSGi interop
- Supporting Profiles & Modular Applications

**HTML5**
- Emerging Web Standards require a programming model
- JSON, WebSockets, off-line, APIs & DOM
Java EE 7 Focus: Platform as a Service

• Next logical step for Java EE
  – J2EE  Java EE 6 : The Java EE Platform provides services
  – Java EE 7 : The Java EE Platform IS a service

• PaaS support entails evolutionary change

• Basic form of SaaS

• Provide way for customers and users to leverage public, private, and hybrid clouds
Roles

- Developer
- PaaS Customer/Tenant
- Deployer
- Application Submitter
- Application Administrator
- PaaS Provider
- PaaS Product Provider
- PaaS Account Manager
- PaaS Administrator

Diagram showing tenants and virtual machines.
Existing Java EE model

- Configure Java EE resources – JDBC, JMS etc
- Deploy Application EAR
Java EE 7 Model: Auto-Provision Services from Application Dependencies

• Provision and deploy application resources (e.g. LDAP stripe, data source instantiation and connection …)
• Extensible Deployment Models Supporting Multiple Frameworks
  • Spring, Seam, Play …

1. Provision and Initialize
2. Provision and Initialize
3. Provision and Initialize
4. Provision and Initialize
5. Deploy Application (EAR/GAR/SAR …)
Services

• Cloud apps consume services
  – Persistence, queueing, mail, caching, ...

• Service metadata facilitates ease of use when deploying into the cloud

```java
@DataSourceDefinition(
    name="java:app/jdbc/myDB",
    className="oracle.jdbc.pool.OracleDataSource",
    isolationLevel=TRANSACTION_REPEATABLE_READ,
    initialPoolSize=5
)
```
Services

- Cloud apps consume services
  - Persistence, queueing, mail, caching, ...
- Service metadata facilitates ease of use when deploying into the cloud

```java
@JMSConnectionFactoryDefinition(
    name="java:app/myJMSConnectionFactory",
    className="javax.jms.QueueConnectionFactory",
    resourceAdapterName="myJMSRA")

@JMSDestinationDefinition(
    name="java:app/myQueue",
    className="javax.jms.Queue",
    destinationName="myQueue1")
```
Services

• Cloud apps consume services
  – Persistence, queueing, mail, caching, …

• Service metadata facilitates ease of use when deploying into the cloud

```java
@MailSessionDefinition(
    name="java:app/mailSession",
    host="example.com",
    from="MyService@example.com"
)
```
Services

• Cloud apps consume services
  – Persistence, queueing, mail, caching, …

• Service metadata facilitates ease of use when deploying into the cloud

```java
@ConnectorResourceDefinition(
    name="java:app/myCustomConnector",
    className="com.extraServices.CustomButton.class",
    properties= {...}
)
```
Elasticity

- Service Levels
- Minimum and Maximum Instances
- Futures – Self Adjustment, Capacity On Demand

Elasticity Continuum

- Single node
  - Non-Elastic
- Java EE Multi-Node
  - Multi-Instance Clustering
- Dynamic Self Adjusting
  - SLA Driven Elasticity
  - Elastic Cluster
    - Capacity on Demand

Java EE 7 Focus: Move the Bar Right
Demo

PaaSing a Java EE Application in the Cloud

glassfish.org/javaone2011
Conference Planning in the Cloud

Java EE Application

JSF

JPA

EJB

Services

Metadata

Deploy

```xml
<glassfish-services>
  <service-description init-type="LB" name="ConferencePlanner-lb">
    <template id="LBNative"/>
    <configurations>
      <configuration name="https-port" value="50443"/>
      <configuration name="ssl-enabled" value="false"/>
      <configuration name="http-port" value="50080"/>
    </configurations>
  </service-description>
  <service-description init-type="JavaEE" name="ConferencePlanner">
    <characteristics>
      <characteristic name="service-type" value="JavaEE"/>
    </characteristics>
    <configurations>
      <configuration name="max.clustersize" value="4"/>
      <configuration name="min.clustersize" value="2"/>
    </configurations>
  </service-description>
...```

19 | Copyright © 2012, Oracle and/or its affiliates. All rights reserved.
Service Provisioning

Load Balancer

Java EE  Java EE  ...  Java EE

Database
Multi-tenancy in Java EE 7

• Support for separate isolated instances of the same app for different tenants
  – One application instance per tenant
  – Tenants correspond to units of isolation
  – Multitenant apps are declared as such
  – Each instance customized and deployed for a single tenant
  – Limited form of SaaS

• Mapping to tenant done by the container

• Tenant id available to application
  – E.g., under java:comp/tenantId
Java EE 7 – Candidate JSRs

- CDI Extensions
- Web Container Extensions
- JAX-RS 2.0
- JSF 2.2
- JSP 2.2
- EL 3.0
- Servlet 3.1
- CDI 1.1 / Interceptors 1.1 / JSR 250 1.1
- Managed Beans 1.0
- JPA 2.1
- JTA 1.1
- JMS 2.0
- EJB 3.2
- Bean Validation 1.1
- Jcache 1.0 (JSR 107)
- Concurrency Utilities 1.0 (JSR 236)
- State Management 1.0 (JSR 350)
- Batch Processing 1.0 (JSR 352)
- JSON 1.0 (JSR 353)
Java EE 7 Early Draft

• Requires Java SE 7
• Added new roles
• Resource definition metadata
  – DataSourceDefinition, JMSCredentialsProviderFactoryDefinition, JMSConnectionFactoryDefinition, JMSConnectionFactoryDefinition, JMSDestinationDefinition, MailSessionDefinition, ConnectorResourceDefinition
  – Pre-provisioned or shared resource
Java EE 7 Early Draft

- Default Data Source `java:comp/defaultDataSource`
- Default JMS Connection Factory `java:comp/defaultJMSConnectionFactory`
- Tenant Identifier: `java:comp/tenantId`
- Made optional
  - EJB Entity Beans, EJB QL, JAX-RPC, Deployment API, JAXR
Java Persistence API 2.1

- `@NamedStoredProcedureQuery`, `StoredProcedureQuery`
- Bulk update/delete using Criteria
- User-defined functions using `FUNCTION`
- Persistence Context Synchronization
JPA 2.1

Stored Procedures

```java
@Entity
@NamedStoredProcedureQuery(name="topGiftsStoredProcedure",
procedureName="Top10Gifts")
public class Product {

  // there are other setParameter methods for defining the temporal type
  // ... 

  query.execute();
  String response = query.getOutputParameterValue(1);
```
**JPA 2.1**

Criteria Update/Delete

CriteriaUpdate\<Customer\> q = cb.createCriteriaUpdate(Customer.class);
Root\<Customer\> c = q.from(Customer.class);
q.set(c.get(Customer_.status), "outstanding")
  .where(cb.lt(c.get(Customer_.balance), 10000));

@PersistenceContext EntityManager em;
Query query = em.createQuery(q);
query.executeUpdate();

### Criteria Delete

CriteriaDelete\<Customer\> q = cb.createCriteriaDelete(Customer.class);
Root\<Customer\> c = q.from(Customer.class);
q.where(cb.equal(c.get(Customer_.status), "inactive"),
  cb.isEmpty(c.get(Customer_.orders)));

@PersistenceContext EntityManager em;
Query query = em.createQuery(q);
query.executeUpdate();
JAX-RS 2.0
Client-side API

URL url = new URL("http://.../atm/balance");
HttpURLConnection conn = (HttpURLConnection) url.openConnection();
conn.setRequestMethod("GET");
conn.setDoInput(true);
conn.setDoOutput(false);
BufferedReader br = new BufferedReader(new InputStreamReader(conn.getInputStream()));
String line;
while ((line = br.readLine()) != null) {
    out.println(line);
}

Client client = ClientFactory.newClient();
String balance = client.target("http://.../atm/balance")
    .request("text/plain")
    .get(String.class);
@Provider
class LoggingFilter implements RequestFilter, ResponseFilter {
   @Override
   public FilterAction preFilter(FilterContext ctx) throws IOException {
      logRequest(ctx.getRequest());
      return FilterAction.NEXT;
   }

   @Override
   public FilterAction postFilter(FilterContext ctx) throws IOException {
      logResponse(ctx.getResponse());
      return FilterAction.NEXT;
   }
}
Client-side Async

Client client = ClientFactory.newClient();
Future<String> future = client.target("http://.../atm/{card}/balance")
    .pathParam("card", "1111222233334444")
    .queryParam("pin", "1234")
    .request("text/plain")
    .async()
    .get(
        new InvocationCallback<String>() {
            public void completed(String result) {
            }

            public void failed(InvocationException e) {
            }
        }
    );
JAX-RS 2.0
Bean Validation Integration

@Path("/")
class ProductResource {
    @POST
    @Consumes(MediaType.APPLICATION_FORM_URLENCODED)
    public void addProduct(@NotNull
                            @FormParam("productName") String name,
                            @NotNull
                            @Category
                            @FormParam("category") String category) {
        ...
    }
}
Server-side content negotiation

```java
@Path("/")
class ProductResource {
    @GET
    @Produces({ "text/xml;qs=0.75", "application/json"})
    public Product[] getProducts() {
        ...
    }
}
```
JavaServer Faces 2.2

- Integration with HTML5 forms
- Queue control for Ajax requests
- File Upload component (Non-Ajax & Ajax)
- Injection in all JSF artifacts – including converters & validators
  - @FaceletsResourceResolver
- Instantiating composite components in Java
- ...
Contexts & Dependency Injection 1.1

• Embedded mode to startup outside Java EE container
• Global ordering of interceptors and decorators
• API for managing built-in contexts
• Send Servlet events as CDI events
• . . .
@MethodValidated
public class OrderService {
    public OrderService(@NotNull CreditCardProcessor creditCardProcessor) { ... }

    public void placeOrder(
        @NotNull
        @Size(min=3, max=20) String customerCode,
        @NotNull @Valid Item item,
        @Min(1) int quantity) { ... }

    @NotNull @Size(min=1)
    public Set<CreditCardProcessor> getCreditCardProcessors() { ... }

    @NotNull @Future
    public Date getNextAvailableDeliveryDate() { ... }
}
Java Message Service 2.0

• Simplified API
  – Less verbose
  – Reduce the number of objects needed to send/receive message
  – Allow resource injection
  – Alternative, not replacement, for standard API
  – Remove JMSException, where possible

• Connection, Session and other objects are AutoCloseable

• New methods – no need for redundant arguments
JMS 1.1 – Sending a Message (Java EE)

Standard API

```java
@Resource(lookup = "jms/connectionFactory ")
ConnectionFactory connectionFactory;

@Resource(lookup="jms/inboundQueue")
Queue inboundQueue;

public void sendMessageOld (String payload) throws JMSException {
    try (Connection connection = connectionFactory.createConnection()) {
        Session session = connection.createSession();
        MessageProducer messageProducer = session.createProducer(inboundQueue);
        TextMessage textMessage = session.createTextMessage(payload);
        messageProducer.send(textMessage);
    }
}
```
JMS 2.0 – Sending a Message (Java EE)

New Simplified API

```java
@Resource(mappedName="jms/contextFactory")
ConnectionFactory connectionFactory;

@Resource(mappedName="jms/inboundQueue")
Queue inboundQueue;

public void sendMessage(String payload) {
    try (JMSContext context = connectionFactory.createContext()){
        context.send(inboundQueue,payload);
    }
}
```
@Inject
@JMSConnectionFactory("jms/contextFactory")
JMSContext context;

@Resource(mappedName="jms/inboundQueue")
Queue inboundQueue;

public void sendMessage(String payload) {
    context.send(inboundQueue, payload);
}
JSON 1.0 (JSR 353)

javax.json.*

• Object model API to represent JSON
  – Similar to DOM API in XML world

• Streaming API to produce/consume JSON
  – Similar to StAX API in XML world

JSR Supporters
  – fasterxml.com (Jackson)
  – Doug Crockford (json.org)

• Spec: json-processing-spec.java.net

• Reference Implementation: jsonp.java.net
JSON 1.0 (JSR 353)

- DOM-based APIs (javax.json.* package)
  - JSONBuilder – Builds a JSON object
  - JSONReader – Reads a JSON object or array from the stream
  - JSONWriter – Writes a JSON object or array to the stream

- Streaming APIs (javax.json.stream.* package)
  - JsonGenerator – Streaming JSON generator
  - JsonParser – Allows forward, read-only access to JSON
JsonObject value = new JsonBuilder()
    .beginObject()
    .add("firstName", "John")
    .add("lastName", "Smith")
    .add("age", 25)
    .beginObject("address")
        .add("streetAddress", "21 2nd Street")
        .add("city", "New York")
        .add("state", "NY")
        .add("postalCode", "10021")
    .endObject()
    .beginArray("phoneNumber")
        .beginObject()
            .add("type", "home")
            .add("number", "212 555-1234")
        .endObject()
        .beginObject()
            .add("type", "home")
            .add("number", "646 555-4567")
        .endObject()
    .endArray()
    .endObject()
    .build();
String json = "...";
JsonReader reader = new JsonReader(new StringReader(json));
JsonValue value = reader.readObject();
reader.close();

JsonWriter jsonWriter = new JsonWriter(new FileWriter(...));
JsonObject jsonObject = new JsonBuilder()
    .beginObject()
    . . .
    .endObject()
    .build();
jsonWriter.writeObject(jsonObject);
jsonWriter.close();
JSON 1.0 (JSR 353)

Code Sample – JsonGenerator, JsonParser

```java
JsonGenerator generator = new JsonGenerator(new FileWriter(...));
generator
  .beginObject()
  ...
  .beginArray()
  ...
  .endArray()
  .endObject()
.build();
generator.close();

String json = "...";
JsonParser parser = new JsonParser(new StringReader(json));
Iterator<Event> it = reader.iterator();
Event event = it.next();
```

START_OBJECT
END_OBJECT
START_ARRAY
END_ARRAY
KEY_NAME
VALUE_xxx
Transparency

• Oracle’s Java EE 7 JSRs are run in the open on java.net
  – http://javaee-spec.java.net
  – One project per spec – e.g., jpa-spec, jax-rs-spec, jms-spec…

• Publicly viewable Expert Group mail archive
  – Users observer list gets copies of all Expert Group emails

• Publicly viewable download area

• Publicly viewable issue tracker

• Commitment to update to JCP 2.8 Process
Status and Schedule

• All JSRs up and running
• Early Drafts
  – Java EE 7 Platform, JSF 2.2, JAX-RS 2.0, JPA 2.1, CDI 1.1, EJB 3.2, JMS 2.0, Bean Validation 1.1, Expression Language 3.0
• Final release target: Q2 2013
HTML 5 is Disruptive

• HTML 5 is the new UI across devices
  – Offline, Real-time Communication, File access, Semantic markup, Multimedia, CSS3

• HTML 5 Requires New Programming Model
  • Servers no longer generating markup language
    • Event Driven
    • JavaScript is part of the domain model
    • JSON is the payload, not just HTML
Java EE: Investments for HTML5

• WebSocket and Server Sent Events Infrastructure
• WebSocket SDK and JSR-356
  – Simplify the programming model for Java EE
WebSocket

- Part of HTML5 Specification
- Bi-directional
- Full-duplex
- Over a single TCP connection

Examples
- Real-time online games
- Collaborative platforms
- Social Networking
WebSocket Protocol – Handshake

tools.ietf.org/html/draft-ietf-hybi-thewebsocketprotocol-17

```
GET /mychat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: x3JJHMbDL1EzLkh9GBhXDw==
Sec-WebSocket-Protocol: chat, superchat
Sec-WebSocket-Version: 13
Origin: http://example.com
<EMPTY LINE>
```

HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: HSmrc0sMlYUkAGmm5OPpG2HaGWk=
Sec-WebSocket-Protocol: chat
<EMPTY LINE>
WebSocket Protocol – Framing

tools.ietf.org/html/draft-ietf-hybi-thewebsocketprotocol-17

- Wire format of messages
- Design Philosophy – Minimum framing required
- Data Frames
  - Text & Binary
    - Blob, ArrayBuffer
- Control Frames
  - Protocol-level signaling
WebSocket API

dev.w3.org/html5/websockets/

```
[Constructor(DOMString url, optional (DOMString or DOMString[]) protocols)]
interface WebSocket : EventTarget {
    readonly attribute DOMString url;

    // ready state
    const unsigned short CONNECTING = 0;
    const unsigned short OPEN = 1;
    const unsigned short CLOSING = 2;
    const unsigned short CLOSED = 3;
    readonly attribute unsigned short readyState;
    readonly attribute unsigned long bufferedAmount;

    // networking
    [TreatNonCallableAsNull] attribute Function? onopen;
    [TreatNonCallableAsNull] attribute Function? onerror;
    [TreatNonCallableAsNull] attribute Function? onclose;
    readonly attribute DOMString extensions;
    readonly attribute DOMString protocol;
    void close([Clamp] optional unsigned short code, optional DOMString reason);

    // messaging
    [TreatNonCallableAsNull] attribute Function? onmessage;
    attribute DOMString binaryType;
    void send(DOMString data);
    void send(ArrayBufferView data);
    void send(Blob data);
};
```
WebSocket SDK
websocket-sdk.java.net

• Annotation Driven Model
  – Allows you to turn POJOs into WebSocket Endpoints
  – Support for Server Sent Events
  – WebLogic 12.1.2 support
  – Runs on GlassFish 4.0 Early Builds Now!

• WebSocket SDK and JSR-356
  – Simplify the programming model for Java EE
  – Integrated with Java EE Platform e.g. CDI
WebSocket Example

import org.glassfish.websocket.api.annotations.WebSocketClose;
import org.glassfish.websocket.api.annotations.WebSocket;
import org.glassfish.websocket.api.annotations.WebSocketMessage;

@WebSocket(path="/HelloWorld")
public class HelloWorld{

    @WebSocketMessage
    public String sayHelloWorld(String caller) {
        return "hello " + caller + "!";
    }

    @WebSocketClose
    public void goodbye() {
        System.out.println("Adios");
    }
}

https://blogs.oracle.com/arungupta/entry/websockets_and_java_ee_7
Server-Sent Events

- Part of HTML5 Specification
- Server-push notifications
- Cross-browser JavaScript API: EventSource
- Message callbacks
- MIME type: text/eventstream
EventSource API

devel.w3.org/html5/eventsourcing/

```
[Constructor(DOMString url, optional EventSourceInit eventSourceInitDict)]
interface EventSource : EventTarget {
    readonly attribute DOMString url;
    readonly attribute boolean withCredentials;

    // ready state
    const unsigned short CONNECTING = 0;
    const unsigned short OPEN = 1;
    const unsigned short CLOSED = 2;
    readonly attribute unsigned short readyState;

    // networking
    [TreatNonCallableAsNull] attribute Function? onopen;
    [TreatNonCallableAsNull] attribute Function? onmessage;
    [TreatNonCallableAsNull] attribute Function? onerror;
    void close();
};

dictionary EventSourceInit {
    boolean withCredentials = false;
};
```
Server-Sent Events Example

Client-side

```javascript
var url = 'http://' + document.location.host + '/glassfish-sse/simple';
var eventSource = new EventSource(url);

eventSource.onmessage = function (event) {
    var theParagraph = document.createElement('p');
    theParagraph.innerHTML = event.data.toString();
    document.body.appendChild(theParagraph);
};
```

https://blogs.oracle.com/arungupta/entry/server_sent_events_using_glassfish
Server-Sent Events Example

Server-side Handler

```java
@ServerSentEvent("/simple")
public class MySimpleHandler extends ServerSentEventHandler {

    public void sendMessage(String data) {
        try {
            connection.sendMessage(data);
        } catch (IOException ex) {
            . . .
        }
    }
}
```

https://blogs.oracle.com/arungupta/entry/server_sent_events_using_glassfish
Server-Sent Events Example

Server-side Business Logic

```java
@Stateless
class SimpleEvent {

    @Inject @ServerSentEventContext("/simple")
    ServerSentEventHandlerContext<MySimpleHandler> simpleHandlers;

    @Schedule(hour="*", minute="*", second="*/10")
    public void sendDate() {
        for(MySimpleHandler handler : simpleHandlers.getHandlers()) {
            handler.sendMessage(new Date().toString());
        }
    }
}
```

https://blogs.oracle.com/arungupta/entry/server_sent_events_using_glassfish
SSE & WebSocket – competing technologies?

- WebSocket can send/receive, SSE is server-push
- SSE are over HTTP, WebSocket require servers to understand the protocol
- SSE provides simple and easy-to-use for server-push, WebSocket is slightly complex and has overhead
- WebSocket provide real-time updates, SSE can be configured
Project Avatar

Complete Solution for Dynamic Rich Clients

HTML 5 browser

HTML & Java hybrid application

Java application

JSON over WebSocket

Java EE Cloud
JPA-RS: Building Block of Project Avatar

• Exposes JPA mapped entities over REST via JAX-RS

• HTTP Methods
  – GET, PUT, POST, DELETE

• Content-Type and Accept-based content negotiation
  – XML or JSON

• Client
  – HTML5 with JavaScript (primary focus)
  – JavaFX
JPA-RS: Current Programming Model

GET http://…/order/4

Shop Persistence Unit

Customer | Product | Order

JAX-RS

JPA
JPA-RS: Thin Server Architecture

GET http://…/<pu-name>/<entity>/4
JPA-RS: Building Block of Project Avatar

Server-side Business Logic

• Persistence Unit Operations
  – /<root-uri>/<pu-name>/entity
  – /<root-uri>/<pu-name>/query
  – /<root-uri>/<pu-name>/metadata

• Supports invocation of @NamedQueries via HTTP

• Server-caching – EclipseLink clustered cache

• Dynamic Persistence also supported
  – Entities defined via metadata – no Java classes required
  – Enables persistence for HTML5/JavaScript apps
NoSQL Stores

- No standards
- Differing APIs and feature sets
- Some offer query language/API – some not
EclipseLink NoSQL

- Support JPA access to NoSQL databases
- Define annotations and XML to identify NoSQL stored entities (e.g., @NoSQL)
- Support JPQL subset for each
  - Key principal: leverage what’s available
- Initial support for MongoDB and Oracle NoSQL.
- Support mixing relational and non-relational data in single composite persistence unit
GlassFish Roadmap

GlassFish v3
- Java EE 6 support
- Single instance
- GlassFish Enterprise Mgr

GlassFish Server 3.1
- Centralized administration
- Clustering / HA
- GlassFish Server Control

GlassFish v3.0.1
- Oracle branding
- Oracle platform support
- Oracle interoperability

GlassFish Server 3.1.1
- Bug fixes
- Updated components
- Incremental features

GlassFish Server 3.1.2
- Bug Fixes
- Incremental features

GlassFish Server 4
- Java EE 7
- Multitenancy
- PaaS-enablement

2009 | 2010 | 2011 | 2012 | 2013

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.
Call to Action

• Java EE 7 Expert Group Project
  – [http://javaee-spec.java.net](http://javaee-spec.java.net)

• Java EE 7 Reference Implementation
  – [http://glassfish.org](http://glassfish.org)

• The Aquarium