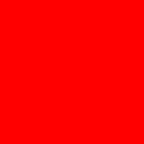


**ORACLE®**

## **Java EE 7: the New Cloud Platform**

Peter Doschkinow  
Senior Java Architect



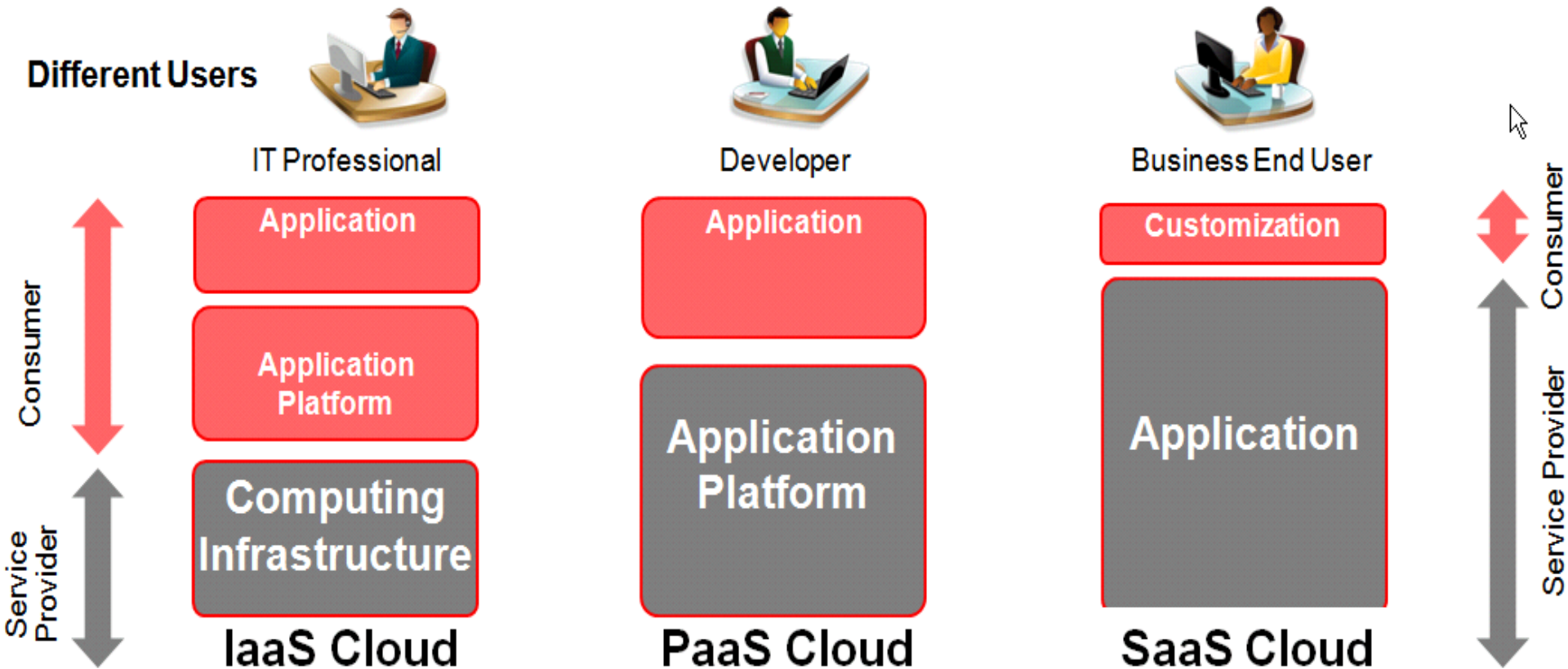
The following/preceding is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions.

The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

# Agenda

- Why Java EE as PaaS
- Java EE 7 as PaaS
  - Services
  - Deployment
  - Multi-tenancy
  - Roles
- Demo
- Java EE 7 API and Status

# Cloud Computing Service Models: IaaS, PaaS, SaaS



# The Java PaaS Marketplace

- Increasing number of vendors, but very different, vendor-specific offerings
  - Amazon Elastic Beanstalk
  - Google App Engine
  - Windows Azure
  - VmWare Cloud Foundry
  - CloudBees
  - Jelastic
  - Oracle Cloud Java service



# PaaS Features and Promises

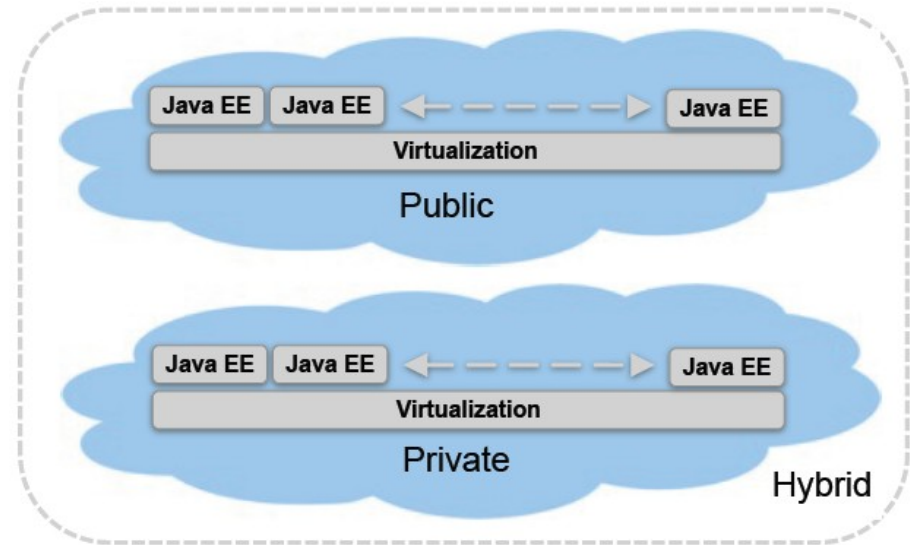
- Easier to use than IaaS: just deploy customer-created applications on provided middleware-stack/framework
- Using languages and tools supported by PaaS Provider
- No control of underlying cloud infrastructure
- Control over deployed applications and hosting environment configurations
- Automatic elasticity
- Better developer productivity, lower cost of management and faster time-to-market
- Cloud service model best suited for developers

# PaaS and Java EE

## Java EE design principles and capabilities

- Common programming model for enterprise developers
- Runtime handles application's infrastructure concerns
- Declarative resource references
- Scalable (scale-out) component models

# Java EE 7 Focus: PaaS



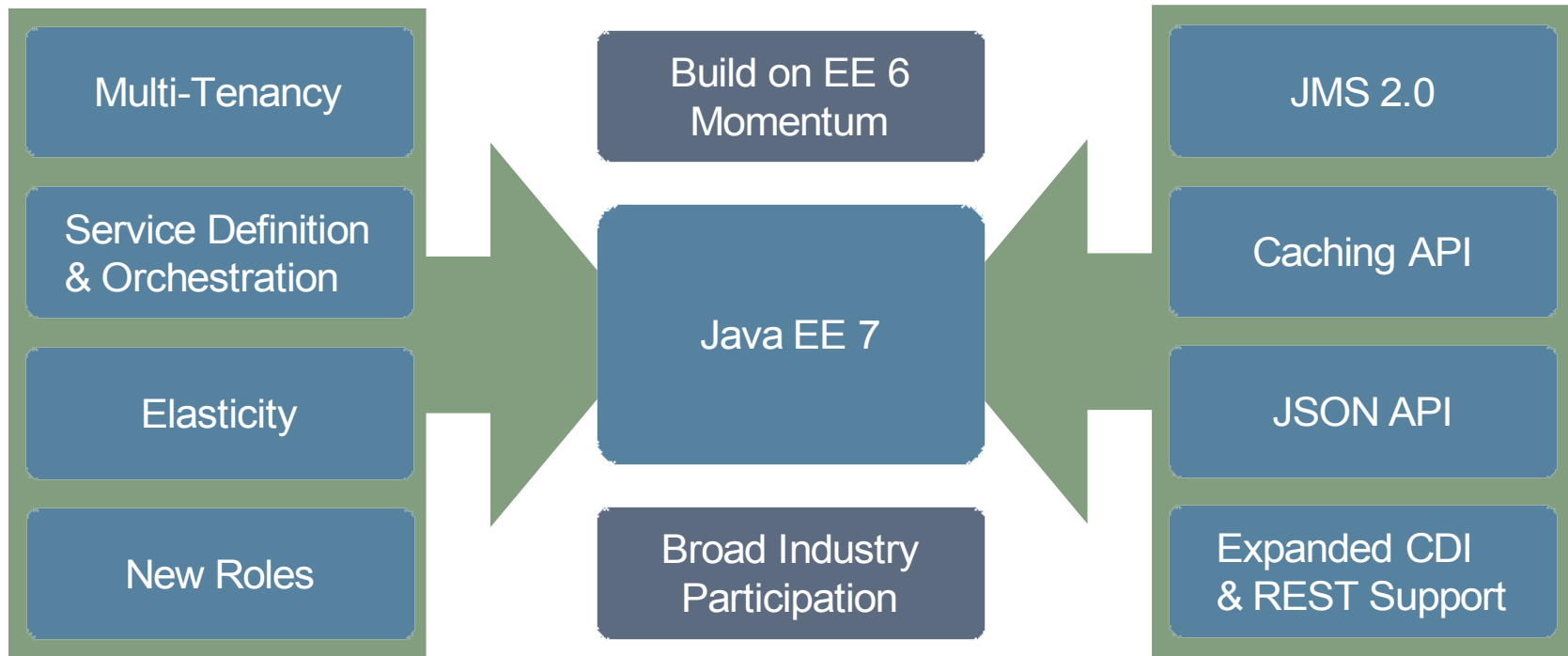
- Provide way for customers and users to leverage public, private, and hybrid clouds in a **standard** way
- PaaS support entails evolutionary change
- Next logical step for Java EE
  - J2EE → Java EE 6 : The Java EE Platform provides services
  - Java EE 7 : The Java EE Platform IS a service



# Java EE 7 PaaS Roadmap

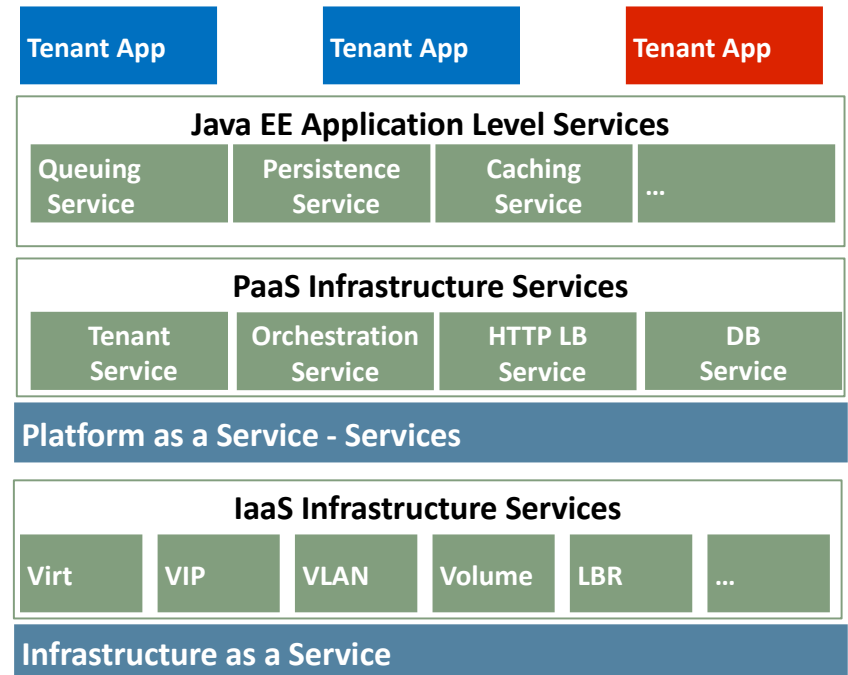
- Add metadata
  - For service provisioning and configuration
  - For QoS, elasticity
  - For sharing of applications and resources
  - For (re)configurability and customization
- Add useful APIs for cloud environment
  - JAX-RS client API, Caching API, State Management, JSON,...
- Extend existing APIs with support for multitenancy
- Define new platform roles to accommodate PaaS model

# Java EE 7 Design Goals



# Java EE 7 as PaaS

- Tenant applications consume services
- PaaS administrators host, configure, and manage application and infrastructure services
- Existing APIs in Java EE need to be updated to be service-enabled and tenant-aware
  - pluggable services are necessary



# Java EE Services

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

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

# Java EE Services

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

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

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

# Java EE Services

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

```
@MailSessionDefinition(  
name="java:app/mail/MySession",  
host="somewhere.myco.com",  
from="some.body@myco.com")
```

# Java EE Services

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

```
@ConnectorResourceDefinition(  
name="java:app/myCustomConnector",  
className="com.extraServices.CustomConnector")
```

# Java EE 7 Service

- An entity that is created, provisioned, managed, and monitored by or for the PaaS runtime/infrastructure
- A significant software function that is needed to execute an application
- Types
  - Provisioned - installed, configured, and managed by the platform; application scoped or shared
  - External - already exist in the enterprise, platform is configured to know about them
  - How it is shared - used by single or multiple environments, per-tenant or global
- Examples: LB, database, Java EE application service

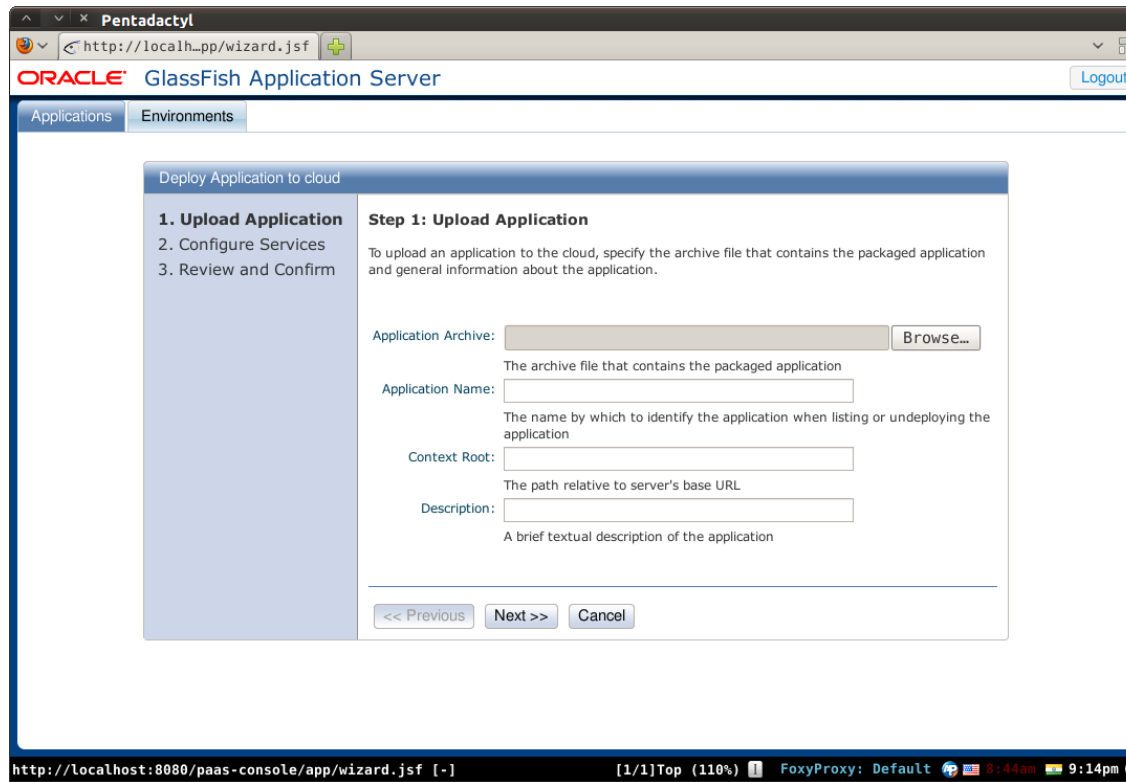


# Specification of Service Metadata

- Service Definition
  - Metadata used to provision and configure a Service
    - Service characteristics (functional and non-functional) specification → Template matching
    - Explicit Template specification
- Service(resource) Reference
  - Represents an application component's dependency on a Service
  - User specified through deployment descriptors or annotations
- Optional
  - When not specified(vanilla EE app), Orchestration Engine automatically handles service dependencies using default service definitions/templates

# PaaS Implications on Deployment

- Simplified PaaS application deployment
  - Single-click, self-service, “push to cloud”



The screenshot shows a web browser window titled "Pentadactyl" with the URL "http://localhost:pp/wizard.jsf". The page is for the "ORACLE GlassFish Application Server" and features a "Deploy Application to cloud" wizard. The wizard is currently on "Step 1: Upload Application".

**Deploy Application to cloud**

**1. Upload Application**  
2. Configure Services  
3. Review and Confirm

**Step 1: Upload Application**

To upload an application to the cloud, specify the archive file that contains the packaged application and general information about the application.

Application Archive:    
The archive file that contains the packaged application

Application Name:   
The name by which to identify the application when listing or undeploying the application

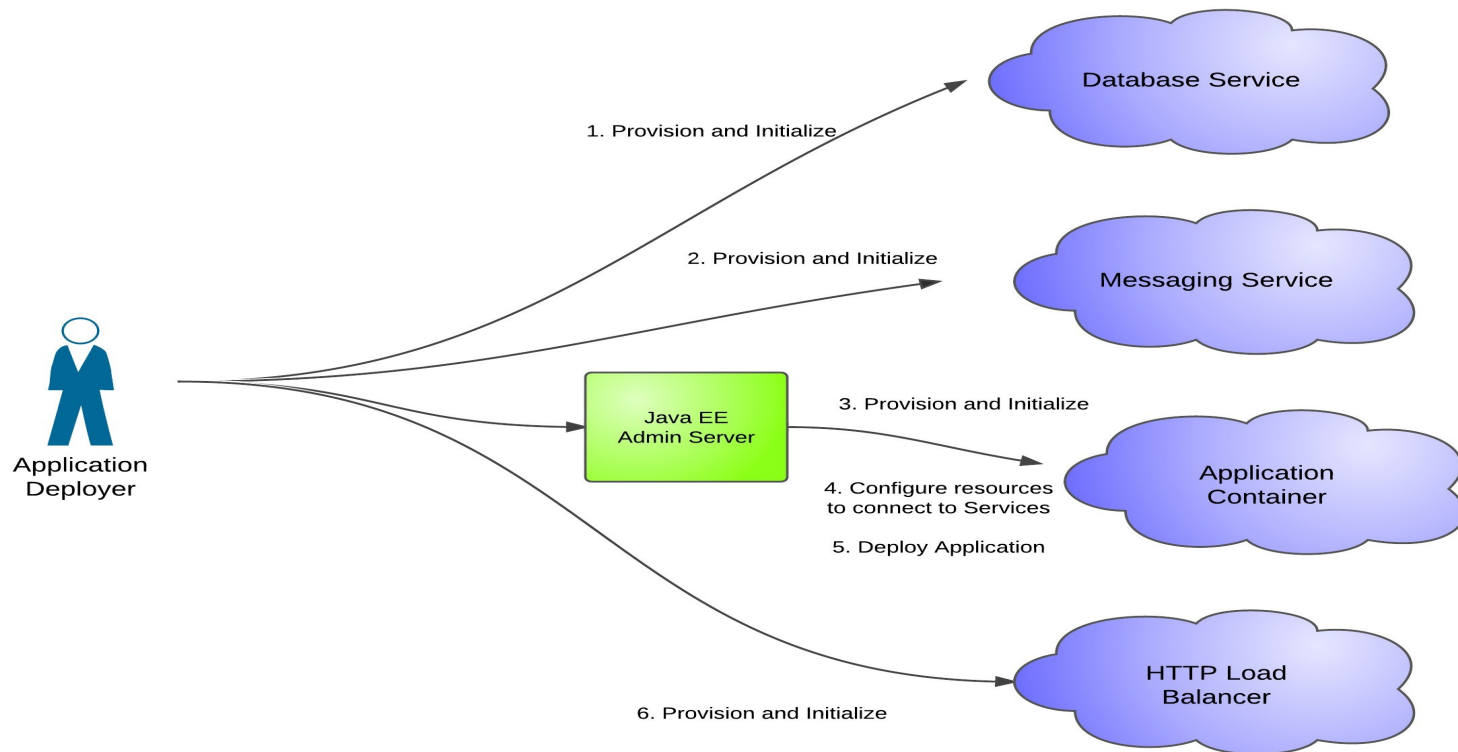
Context Root:   
The path relative to server's base URL

Description:   
A brief textual description of the application

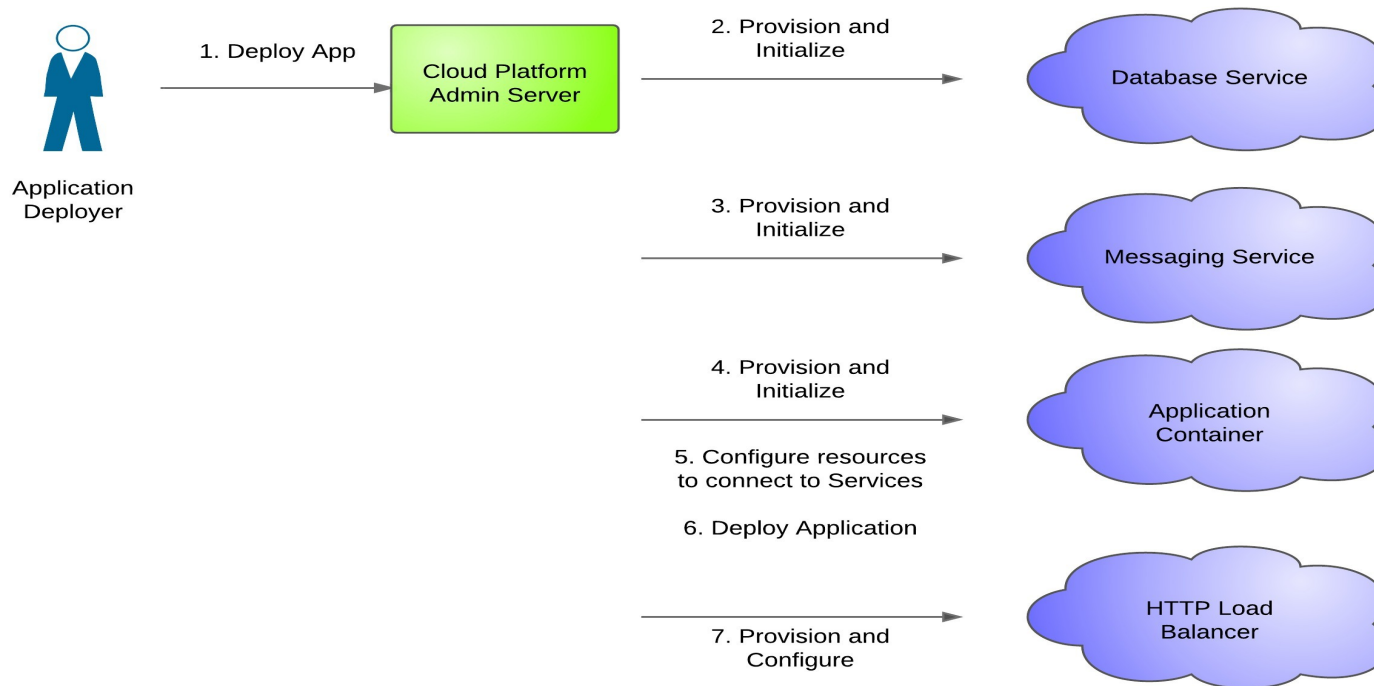
<< Previous   Next >>   Cancel

The browser's status bar at the bottom shows the URL "http://localhost:8080/paas-console/app/wizard.jsf [-]", zoom level "[1/1]Top (110%)", proxy "FoxyProxy: Default", and system time "8:44am 9:14pm".

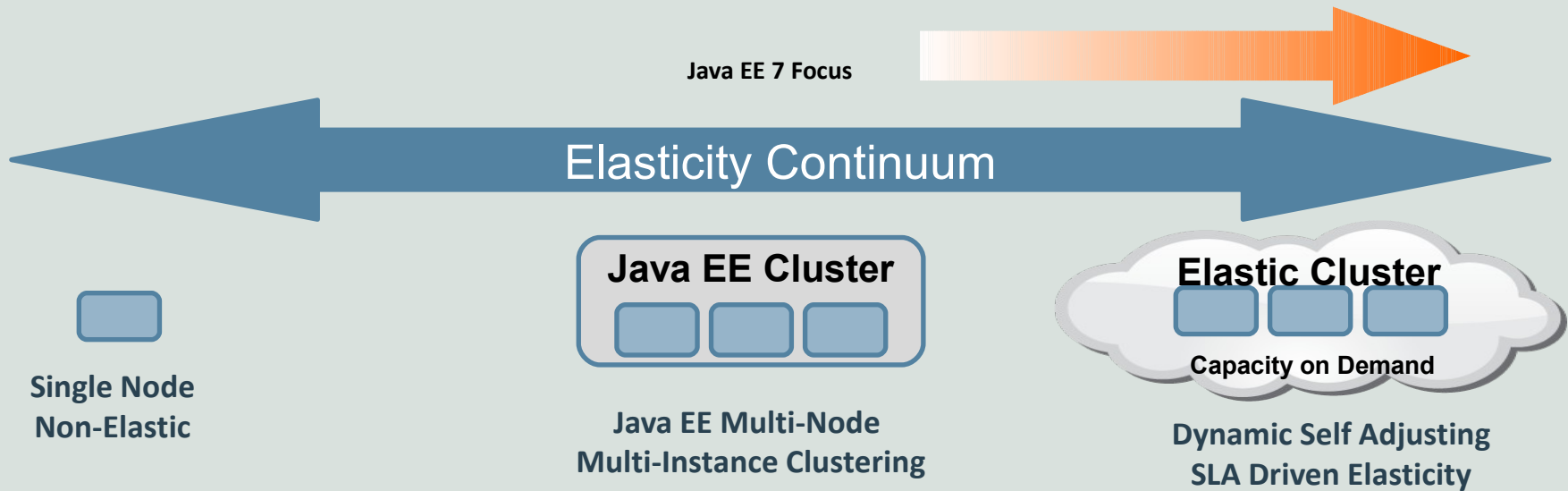
# Traditional Java EE App Deployment



# Java EE 7 PaaS App Deployment



# Java EE 7 Elasticity



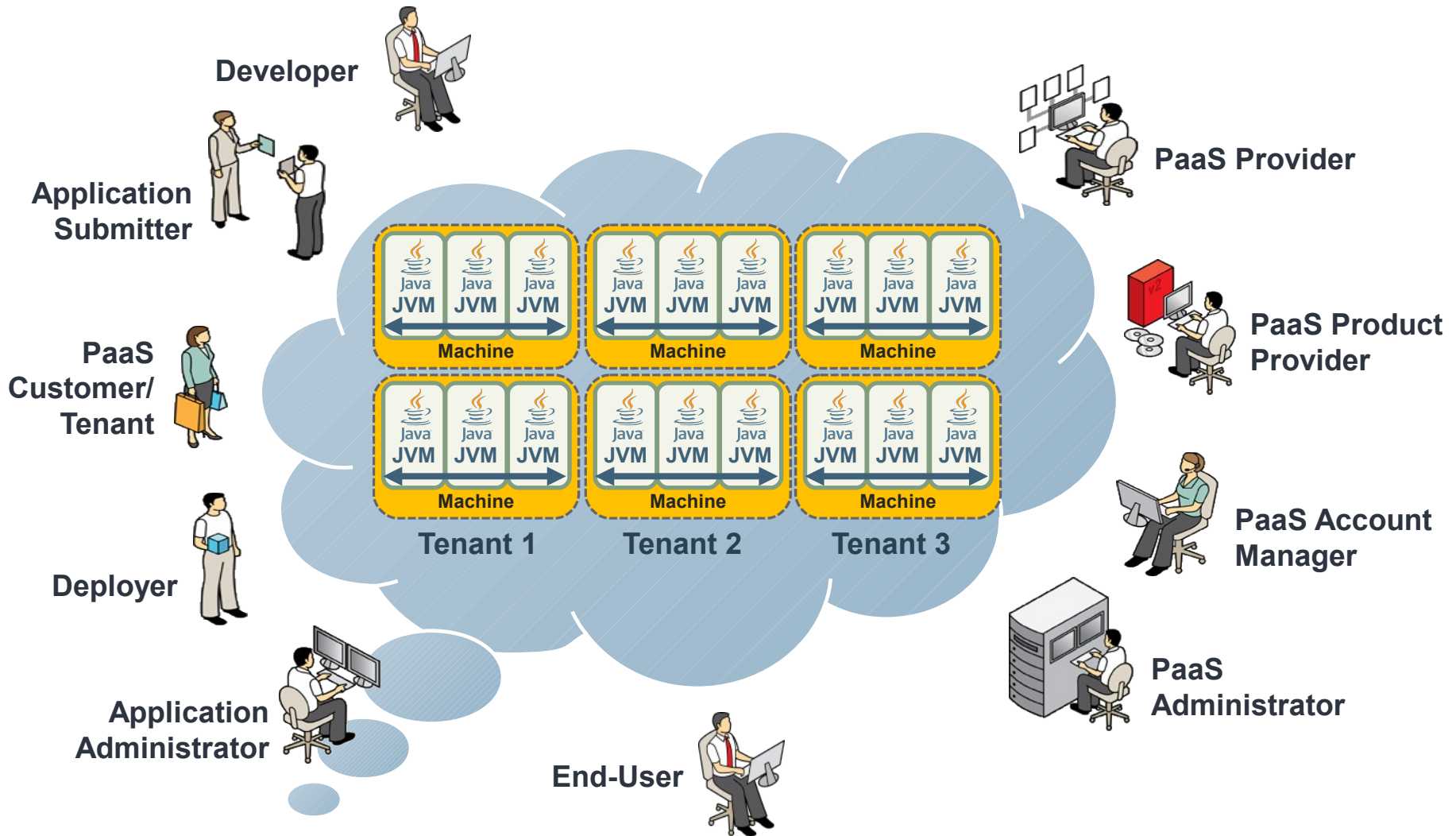
- Cluster elasticity :
  - Metrics provided by application
  - Application Server metrics (response time, etc..)
  - Virtual Machine information (CPU, Memory, Disk usages)
- Metrics sources
  - JMX Mbeans, JVM Monitoring tools, native tools

# Java EE 7 Multitenancy

Limited form of SaaS

- Support for separate isolated instances of the same app for different tenants
  - Multitenant apps are declared as such
  - Tenants correspond to units of isolation
  - One application instance per tenant
  - Each instance customized and deployed for a single tenant
- Mapping to tenant done by the container
- Tenant id available to application
  - E.g., under `java:comp/tenantId` or by injection

# Java EE 7 Roles



# GlassFish Server 4.0

GlassFish 4.0 

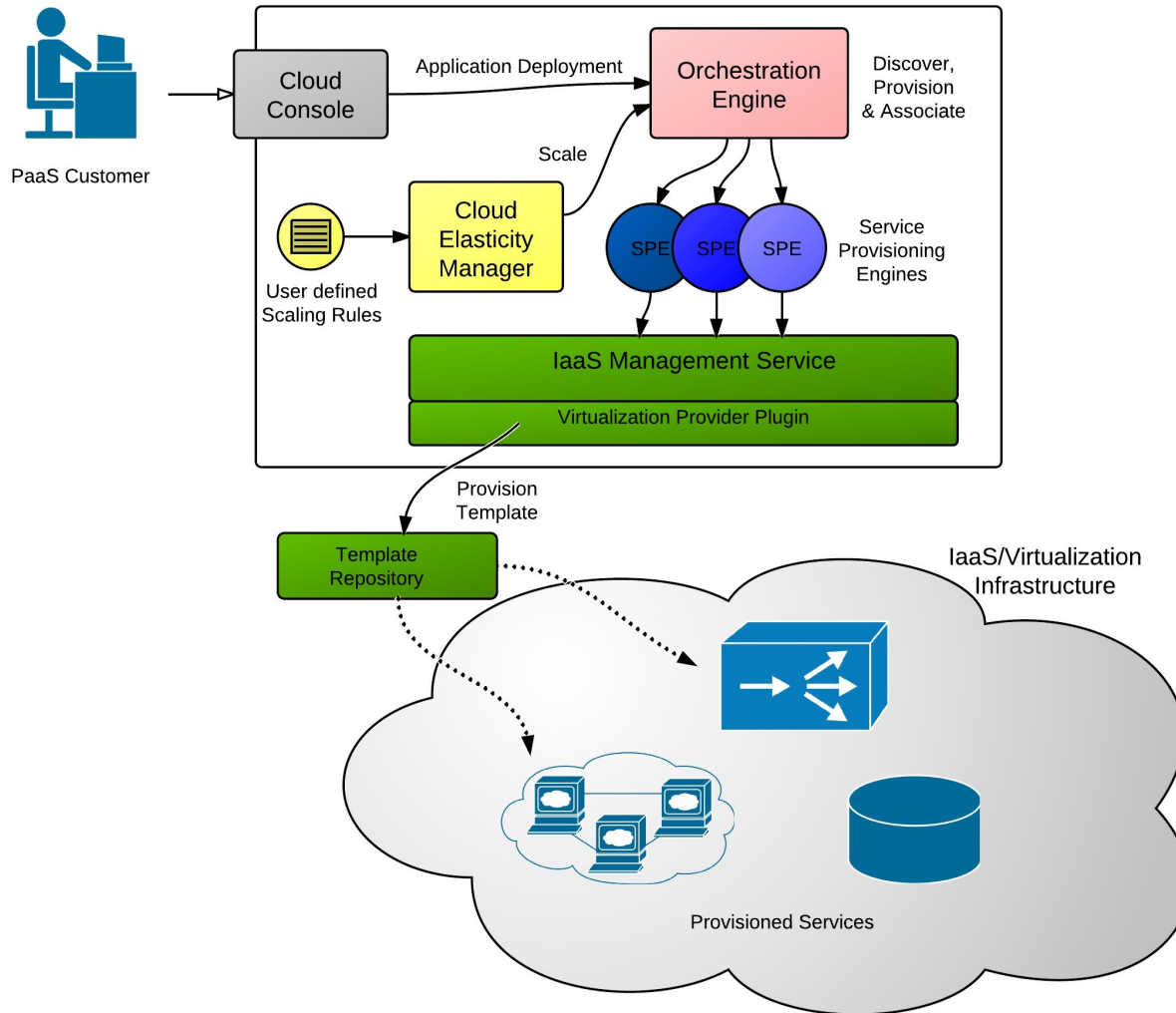


- Java EE 7 reference implementation
- Virtualization implementations
  - Laptop mode
    - Runs processes on the bare metal operating system.
  - Local mode
    - Locally installed hypervisor
    - Best fidelity to deployment scenario
  - Remote mode
    - Connects to remote hypervisors
- Transparent development

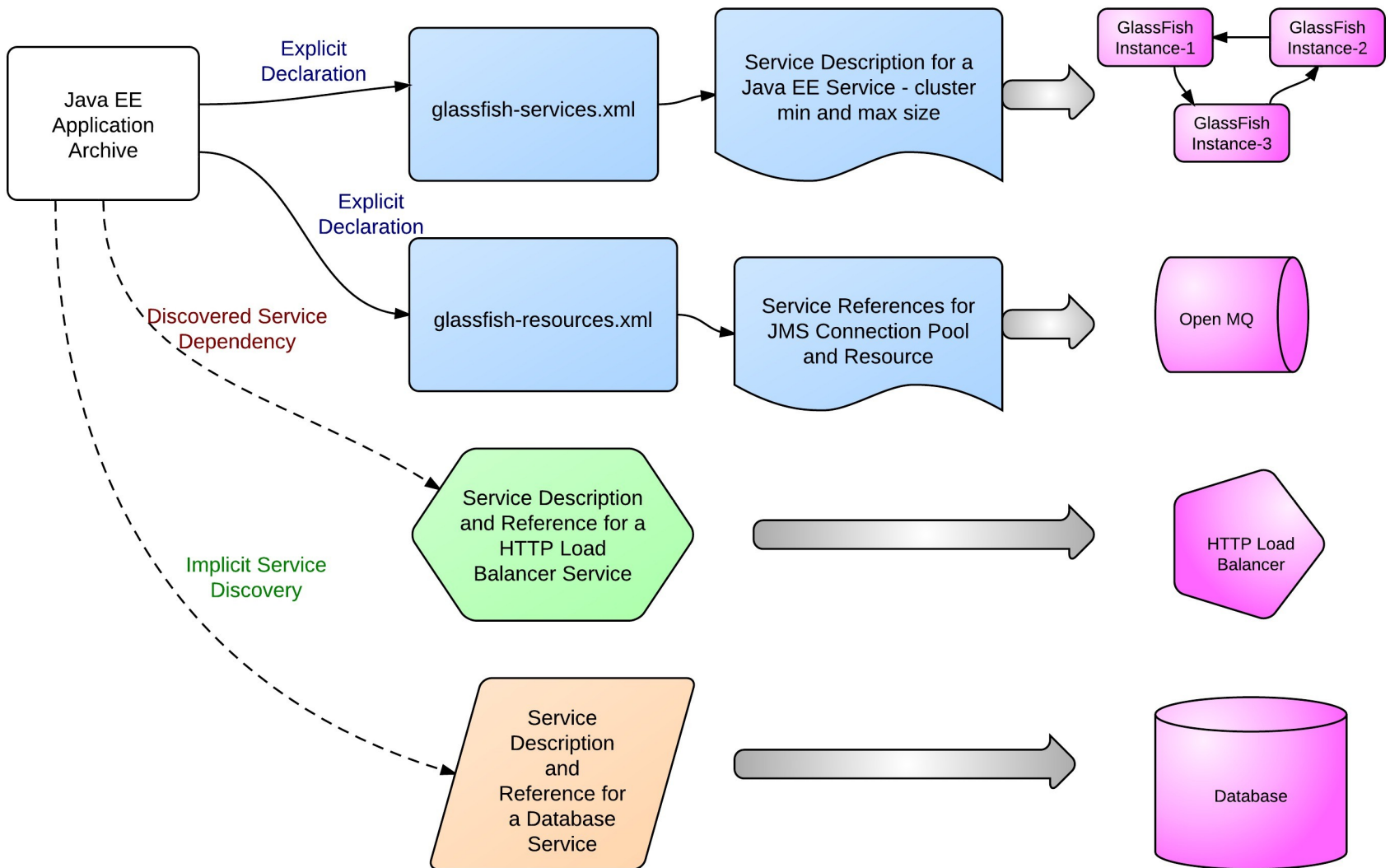


# GlassFish PaaS Architecture

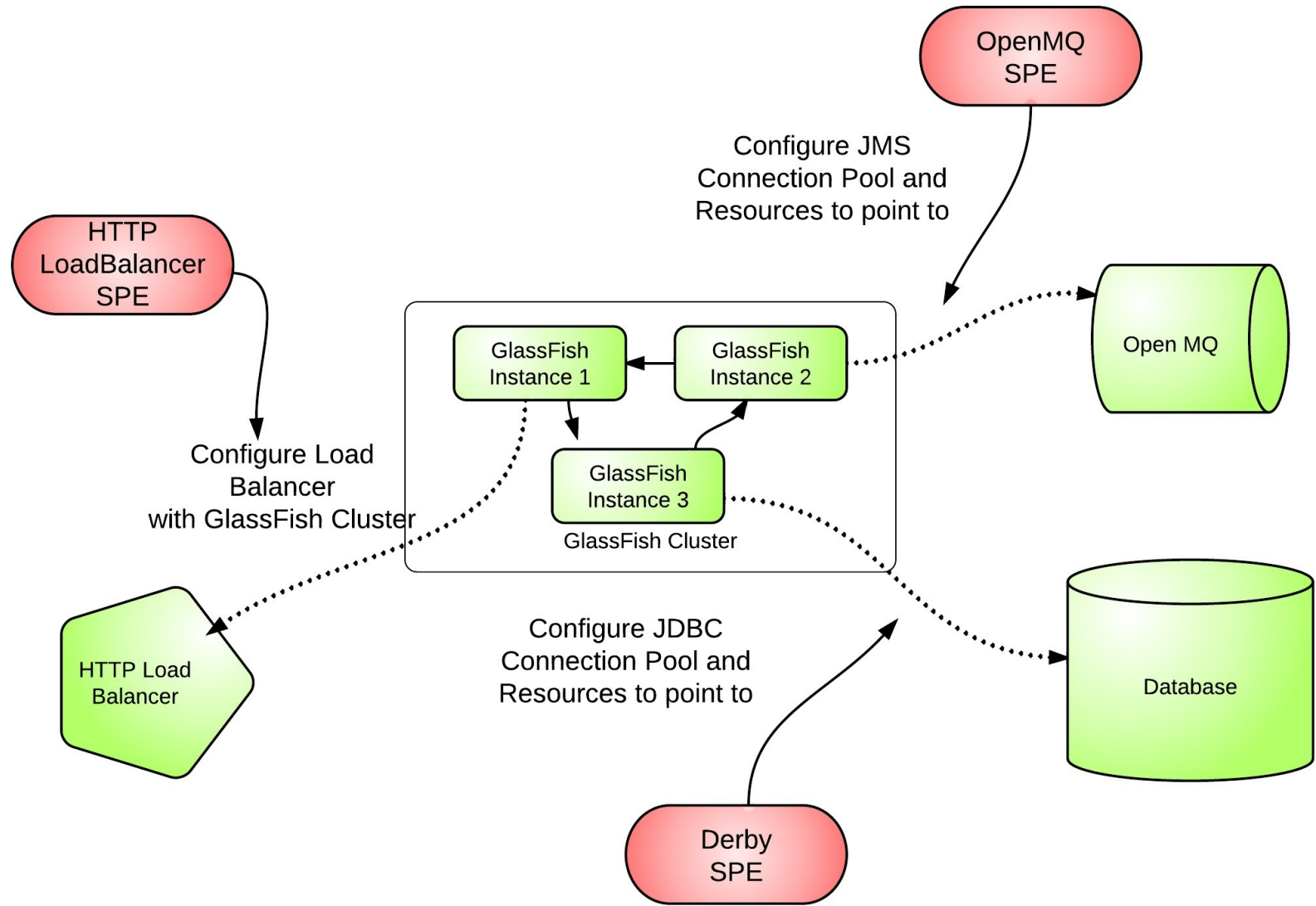
## Cloud Platform Admin Service (CPAS)



# Service Dependency Discovery and Provisioning

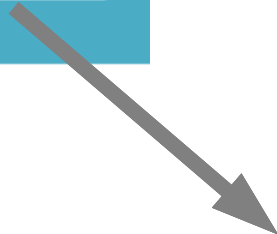
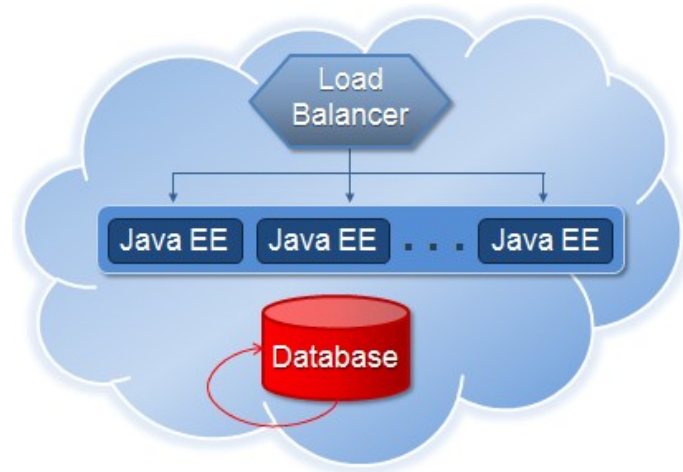
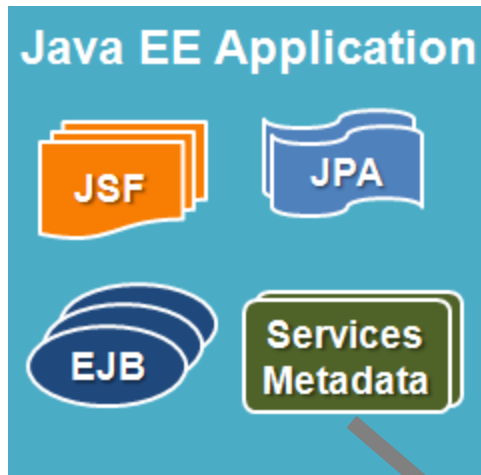


# Service Association



# PaaSing a Java EE Application

GlassFish 4.0 Demo at JavaOne: <http://glassfish.org/javaone2011>



```
<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>
...
</glassfish-services>
```

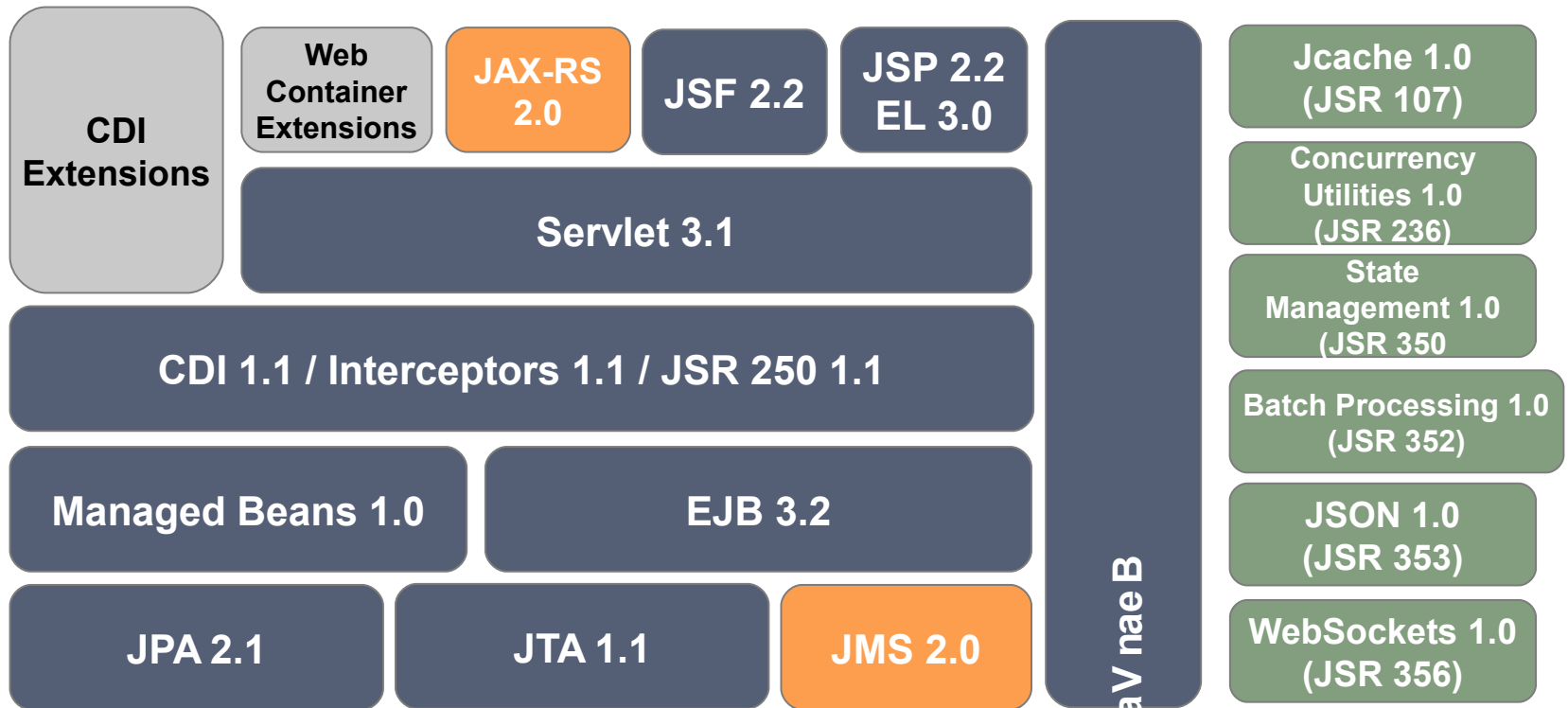
# Demo

- Dynamic service provisioning
  - Service dependencies are discovered from metadata and by application archive introspection
  - Discovered services (Java EE, database, load balancer) are provisioned
- Highly available cluster
  - With session failover
- Elasticity using auto-scaling
  - The Java EE cluster is automatically resized to meet growing demands

# Java EE 7 is not just Cloud-y

- Alignment of ManagedBeans across CDI, EJB, JSF, ...
  - POJO → ManagedBean → Enterprise JavaBean
  - Extension of container-managed transactions beyond EJB
- Further simplifications for ease-of-development
  - JMS 2.0 focus on ease-of-development
  - Expanded use of dependency injection
  - Expanded service metadata; improved configuration
- Pruning
  - EJB CMP and BMP, JAX-RPC, Deployment API
- Update to Web Profile

# Java EE 7 JSRs



# Transparency Checklist

<http://jcp.org/en/resources/transparency>

- Our 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 match JCP 2.8 Process

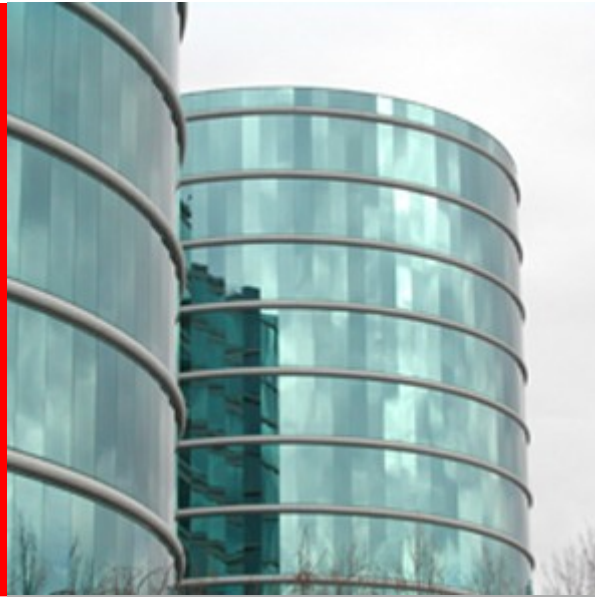


# Java EE 7 Status and Schedule

- All JSRs up and running
- Early drafts available for:
  - Java EE 7 (JSR 342)
  - Expression Language 3.0 (JSR 341)
  - Java Message Service 2.0 (JSR 343)
  - Enterprise JavaBeans 3.2 (JSR 345)
  - Contexts and Dependency Injection 1.1 (JSR 346)
  - Bean Validation 1.1 (JSR 349)
  - JavaServer Faces 2.2 (JSR 344)
  - Java Persistence API 2.1 (JSR 338)
  - Java API for RESTful Services 2.0 (JSR 339)
- Final release by Q2 2013
- Date-driven release: anything not ready will be deferred

# Links

- Java EE 7 java.net project
  - Archives, documents, mailing lists,...
  - <http://java.net/projects/javaee-spec>
- Component projects
  - <http://java.net/projects/XXX-spec>  
(where XXX = jpa, ejb, jms, servlet, jax-rs, jsf,...)
- GlassFish – Java EE 7 reference implementation
  - <http://glassfish.org>
- Feedback
  - [users@javaee-spec.java.net](mailto:users@javaee-spec.java.net)



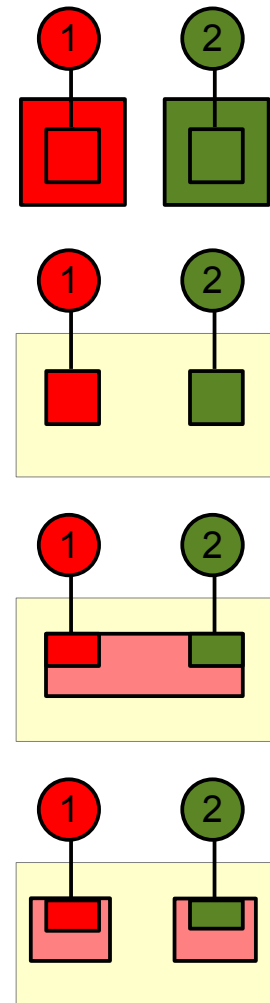
**ORACLE®**

## **Java EE 7: the new Cloud Platform**

Peter Doschkinow  
Senior Java Architect

# PaaS and Multi-tenancy: Some Models

- PaaS Platform on Demand
  - New runtime stack for each tenant
- PaaS Multitenant Containers
  - Isolated app partitions per tenant with shared runtime
- SaaS Multitenant Applications (SaaS-full)
  - Shared app instances, with tenant-specific customization
- SaaS-limited
  - Separate app instances, with tenant-specific customizations



# 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
- New mandatory API for integration of any JMS 2.0 provider with any Java EE server
- Connection, Session and other objects are AutoClosable

# Java Message Service 2.0

## sending a message the old way

```
@Resource(lookup = "jms/connectionFactory ")
ConnectionFactory connectionFactory;
@Resource(lookup="jms/inboundQueue")
Queue inboundQueue;
public void sendMessageOld (String payload) {
    Connection connection = null;
    try {
        connection = connectionFactory.createConnection();
        Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
        MessageProducer messageProducer = session.createProducer(inboundQueue);
        TextMessage textMessage = session.createTextMessage(payload);
        messageProducer.send(textMessage);
    }
    catch (JMSEException e) {
        // do something
    } finally {
        try {
            if (connection != null)
                connection.close();
        } catch (JMSEException e2) {
            // do something else    }}}}
```

# Java Message Service 2.0

sending a message the new way

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

public void sendMessageNew (String payload) {
    try (JMSContext context = connectionFactory.createContext();){
        context.send(inboundQueue,payload);
    }
}
```