2014 한국소프트웨어아키텍트대회

OSGi와 SOA 기반의 클라우드 플랫폼 아키텍처

OCE 의장 장진영
Rick.jang@opence.org
PaaS Mission

- Ready-made architecture
- All-in-One
- One-click integration
Innovation

Competitors

Provider who using PaaS

Design Install Integrate Develop Optimize for operation

plug Choose apps Modeling & Develop with Open APIs

Fast delivery & Low-cost by ready-made Architecture

DONE!
PaaS: has modularity issues

- Multiple Services
- Mature SaaS Level
Java for Enterprise PaaS

- Mission critical
- Transactional
- Legacy Integration

i.e. J2EE 8’s Multi-tenancy and PaaS Spec

- Optimized for SaaS app development (B2C)
- Cloud automation
Java Module System: ClassLoader

- Bootstrap Class Loader
- Extension Class Loader
- System Class Loader
- User Defined Class Loader
- User Defined Class Loader

i.e. Tomcat’s ClassLoader
OSGi: Standard module system for Java

• Standard
• Simplifies the complex classloader issues by encapsulating-‘Bundle’
• Spring DM and blueprint
• IBM J9 and Java jigsaw roadmap
Platforms using OSGi

- Eclipse plugin Marketplace
- Atlassian plugin Marketplace...
Platforms using OSGi

- Apache ServiceMix WAS
- Glassfish WAS
- JOnAS WAS
OSGi for PaaS

• More modular, so share more (at library level)
• More dynamic, so easily automated

• Provides standard J2EE framework including JAX-RS, JPA and Spring with whiteboard pattern.
Sharing Level

• Hypervisor-based PaaS vs. OSGi-based PaaS

Share less, More easy & Secure !

Share more, More cheap offering, More Competitive in the market !
OSGi based PaaS approaches

- Apache Ace and Amdatu
- uEclipse Gyrex
- Vaadin Arvue Master
- WSO2’s Stratos
- OCE Garuda

← Initiative from Korea!
An OSGi-based PaaS Architecture (OCE Garuda)

**Build Time**

- **Tools**
  - Visual Mashup
  - BPMN Process Modeler
  - Cloud IDE
  - Vaadin Designer

- **Artifacts**
  - Composite UI
  - BPMN (*.process)
  - Calls
  - JAVA
  - Vaadin UI

- **Package**
  - OSGi Bundle packager
  - Automated by CI tool (Jenkins)

- **Instance**
  - Master
  - HA Proxy
  - App1
  - App2
  - PaaS RT
  - OSGi Container
  - Node1
  - Node2

**Run-Time**

- **Instance**
  - Upload to App Store
  - Netra
  - App repo.
  - App Bundles (*.jar)

- **VM Template**
  - Image

**Stay In SCM (e.g. Git)**

Automated by CI tool (Jenkins)
Exposing Platform Services using OSGi and RESTful APIs

- Using Apache-CXF by white-board pattern
Whiteboard Pattern

Figure 2 Whiteboard actors in the OSGi framework
Extender Pattern

Looks for purple bundles

- Bundle A
- Bundle B
- Bundle C
- Bundle D

Matching bundles

Non-matching bundle
Services: RESTful Service

- Filter by annotation: @Path

```java
@Path("saveCustomer")
@Consumes(MediaType.APPLICATION_JSON)
@Produces(MediaType.TEXT_PLAIN)
public String saveCustomer(Customer customer) {
}
```
Services: Vaadin UI Service

- Using whiteboard pattern, platform pickups filtered by Class Type: com.vaadin.ui.UI

```java
@Title("CRM Web Application")
public class CustomerUI extends UI {
    ....
}
```
Services: Multi-tenancy

- Filtered by Bundle Header: X-Multi-tenant
- Inherent isolated class-loader
- Database isolation by JPA’s @Discriminator annotation
- Tenant information injection by ThreadLocal
- Also BPM engine provides tenant-specific Business process and logics(rule)
Cloud IDE: Auto OSGi Bundling by Maven

- Installation-free rapid development
- BPM platform connected to rapid ‘process app’ development
- Import from Cloud
  - Web Service Import
  - Database
- Developed results (apps) are packaged as OSGi bundles in a standard format
- J2EE Compliant

Cf. Google Apps Scripts
Cloud IDE: Vaadin UI Platform

UI Design in Vaadin wysiwyg designer

Source editing in Cloud IDE
Cloud IDE: BPMN Process Modeler

- BPMN Process Modeler
- Execution-capable process language
- GUI-based business rule definition
- Web service integration (*CSB)
- Business forms
- Data mapping
  - DB query automation
  - SQL tool
Reusing the mature SOA Architecture

Components

Services (Exposed by Apache-wink)

Enterprise Apps

Services

Process (on BPM)

CAS SSO / Apache DS EAM

App Registry (App Store)
Related works:

- Hypervisor-based Isolation
- Process-based Isolation (Linux Container)
- Gear
Conclusion

• OSGi makes implementing Enterprise PaaS easier by:

  – Modularity when to integrate existing J2EE technologies and legacy systems.
  – Dynamicity for handling components and services.
  – Service Oriented Architecture.
References

• Paul Bakkar, Bert Ertman, *Building Modular Cloud Apps with OSGi*, O’REILLEY


• Apache ACE Project, https://ace.apache.org/