Microservices mit Jersey und einem In-Memory Data Grid

JFS, Juli 2016

Peter Doschkinow
Michael Bräuer
Safe Harbor Statement

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.
Agenda

• Microservices
  – Java EE and Microservices
• Oracle lightweight technologies for microservices
• Demos
Characteristics of Existing Monolith Architecture

The status quo has served us well but there are new alternatives

- Three tiers
- Scale by cloning behind load balancer (X-axis scaling)
- One programming language
- Everything centralized – messaging, storage, database, etc

![Diagram showing monolith architecture]

- One large archive, including UI(s) and application code
- Feature-rich – support large, complicated applications, many use cases
- Provide 100% isolation between tenants
- Procured and manually set up
Existing Monolith Architecture Has its Limits

**Too Complex**
- Apps get too big and complicated for a developer to understand over time. Shared layers (ORM, messaging, etc) have to handle 100% of use cases – no point solutions

**Too Slow**
- Teams split up by function – UI, application, middleware, database, etc. Takes forever to get anything done due to cross-ticketing

**Too Fragile**
- A bug will quickly bring down an entire application. Little resiliency

**No Specialization**
- Different parts of applications have different needs – more CPU, more memory, faster network, etc.. Can not evolve at a different pace

**No Ownership**
- Code falls victim to “tragedy of the commons” – when there’s little ownership, you see neglect

**Inefficient Testing**
- Each time you touch the application, you have to re-test the whole thing. Hard to support continuous delivery
What Are Microservices?

Minimal function services that are deployed separately but can interact together to achieve a broader use-case

<table>
<thead>
<tr>
<th>Status Quo</th>
<th>Microservices</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single, Monolithic App</td>
<td>Many, Smaller Minimal Function Microservices</td>
</tr>
<tr>
<td>Must Test/Deploy/Scale Entire App</td>
<td>Can Test/Deploy/Scale Each Microservice Independently</td>
</tr>
<tr>
<td>One Database for Entire App</td>
<td>Each Microservice Has Its Own Datastore</td>
</tr>
<tr>
<td>In-process Calls Locally, SOAP Externally</td>
<td>REST Calls Over HTTP, Messaging, or Binary</td>
</tr>
<tr>
<td>Organized Around Technology Layers</td>
<td>Organized Around Business Capabilities (Bounded Ctx)</td>
</tr>
<tr>
<td>One Technology Stack for Entire App</td>
<td>Choice of Technology for Each Microservice</td>
</tr>
<tr>
<td>Developers Don’t Do Ops</td>
<td>Developers + Ops Support Production in Perpetuity</td>
</tr>
<tr>
<td></td>
<td>New infrastructure, design patterns and goals</td>
</tr>
</tbody>
</table>
Microservices vs. SOA

• Microservices
  – Smart endpoints, dumb pipes
  – Goals: Agility and Scalability
  – APIs: Private (Within the Application)
  – New Design Patterns and Infrastructure

• SOA
  – Dumb endpoints, smart pipes
  – Goals: Composition and Reuse
  – APIs: Published (Outside the Application)
  – Traditional Patterns and Infrastructure
Benefits of Microservices Come With Costs

Benefits

- Increase Agility
- Massive Scalability (development too!)
- Strong Module Boundaries
- Independent Deployment
  - Each team is free to deploy what/when they want
- Ability to Pick Different Technology
  - Each team can pick the best technologies for each microservice

Costs

- Distributed Computing
  - Microservice deployed separately, with latency separating each service
- Eventual Consistency
  - System as a whole is eventually consistent because data is fragmented
- Operational Complexity
  - Need mature DevOps team, with very high skills
Monolithic Deployment

- Infrequent releases (months)
- Limited scale
- High risk
- One technology

Gartner Application Architecture, Development & Integration Summit, May 2016
Microservices Deployment

- Release when ready
- Autonomous teams
- Scaling of development
- Polyglot

Gartner Application Architecture, Development & Integration Summit, May 2016
Microservices: The Bottom Line

– Majority of systems just fine as “monoliths”
– Majority of systems needing microservices could evolve into “hybrids”
– Few practical enterprise systems can or need to achieve microservices nirvana

... don’t even consider microservices unless you have a system that’s too complex to manage as a monolith.
The majority of software systems should be built as a single monolithic application. **Do pay attention to good modularity within that monolith**, but don’t try to separate it into separate services

http://martinfowler.com/bliki/MicroservicePremium.html
Microservices Related Technologies

- Frameworks: fat jars, “containerless”
  - Spring Boot, Dropwizard, Vert.x
  - WildFly Swarm, Payara Micro(GlassFish), KumuluzEE, TomEE Embedded
  - Grizzly(HTTP) + Jersey(JAX-RS) + Tyrus(WebSocket) + ...

- Virtualization
  - Docker Container, WebLogic Microcontainer

- Cloud
  - IaaS, PaaS

- Java libraries for reactive programming and microservice patterns
  - RxJava, Hystrix
Java EE and Microservices
Building blocks for pragmatic microservices

- JAX-RS
- JMS
- WebSocket
- Servlet
- JSON-P
- Bean Validation
- JAXB
- CDI
- EJB
- JPA
- JTA
- JCA

Administration  Monitoring  High Availability  Security  Resources
Java EE and Microservices
http://microprofile.io initiative

- JAX-RS
- JSON-P
- EJB
- Administration

- JMS
- Bean Validation
- JPA
- Monitoring

- WebSocket
- JAXB
- JTA
- High Availability

- Servlet
- CDI
- JCA
- Security

- Resources
WebLogic Multitenant Microcontainer for Microservices

Similar to Oracle Database pluggable/container databases

- Each microservice instance can have its own light-weight WebLogic container-like partition
- Partition isolation inside the JVM
- Easily move partitions between WebLogic hosts
- Each partition is exceptionally light
- Each WebLogic host can support hundreds of partitions
## Docker vs. WebLogic based Microservices

### Microservices on Docker Container
- Microservices either implement or rely on docker container frameworks for all non-functional requirements
- Stronger resource isolation
- Security issues
  - Container exploits propagate to all containers and the host
  - [http://www.boycottdocker.org/](http://www.boycottdocker.org/)

### Microservices on WLS Microcontainer
- Extreme lightweight microservices
- All non-functional requirements are inherited from WebLogic infrastructure
- Weaker resource isolation
Agenda

• Microservices
  – Java EE and Microservices

• Oracle lightweight technologies for microservices

• Demos
JAX-RS/Jersey

• JAX-RS 2.0
  – part of Java EE 7 (2013)
  – defines a standard API for
    • Implementing RESTful web services in Java
    • REST client API

• Jersey 2.0
  – provides production ready JAX-RS 2.0 reference implementation
  – brings several non-standard features
  – Current version is 2.22.1
Jersey for Microservices

- Integration with various HTTP containers and client transports
- Reactive/Async Client
- Test Framework, Monitoring and Tracing
- Support for SSE
- Dynamic reloading
- Various data bindings
- Security
- MVC view templates
- Weld (CDI) support
Oracle Coherence In-Memory Data Grid

• Distributed in-memory management of application objects
  – High performance, scalability and availability

• Typical use cases: caching, analytics, event processing

• Implements JCache JSR-107

• Support for Java SE 8 lambdas and Stream-like API

• Can be used as a service in a microservice architecture
  – For application state, configuration and service registration and lookup

• Lightweight
  – Start as a jar with a config file:
    • `java -Dcoherence.cacheconfig=<config-file> -jar <one-jar-containing-coherence-lib>`
Using Application Container Cloud Service For Microservices

A modern platform for lightweight application development

Developer Cloud Service

- Jersey + Grizzly
- Java SE Cloud Service
- Node Cloud Service
- Bring Your Own Container
- Other Polyglot Runtimes
- Configuration
- Service Discovery
- API Load Balancer
- Container Placement
- Docker Containers
- Application Container Cloud Service

Oracle Management Cloud - Management/Logging/Alerting

Oracle Cloud

Caching Cloud Service for State

Messaging Cloud Service

Database Cloud Service

NoSQL Cloud Service
Agenda

• Microservices
  – Java EE and Microservices
• Oracle lightweight technologies for microservices
• Demos
HTML5 App with Jersey+Tyrus+Grizzly: Drawing Board Demo

• Collaborative drawing
• Two-page application
  – List of drawings
  – Drawing
• Demonstrating
  – Server-side
    • Java EE 7: JAX-RS, JSON, WebSocket
    • Jersey specific: SSE, JSON-B
    • Lightweight integration Jersey+Tyrus+Grizzly – only ~10 MB footprint!
  – Client-side: AngularJS or JavaFX
Drawing Board Demo
light implementation – no app server

Clients
- HTML5 Browser
- WebView/WebKit
- JavaFX

Grizzly
- HTTP/S
- JSON
- SSE
- WS Endpoint (Tyrus)

DataProvider
- POJO (Drawings HashMap)

DrawingService
- query(...)
Splitting into microservices

**Clients**
- HTML5 Browser
- JavaFX
- HTTP/S
- JSON
- SSE

**Grizzly**
- JAX-RS, SSE (Jersey)
- DataProvider POJO (Drawings HashMap)

**Node.js**
- Socket.IO
- HTTP/S
- JSON
- Web Sockets
- DataProvider POJO (Drawings HashMap)

**JavaFX**
- WebView/WebKit
- send(...) onEvent(...) DrawingService.query(...) webSocketSend.send(...)
Drawing Board Demo

Splitting into microservices, deployed on the Application Container Cloud Service
https://dbmicrosse-gse00000361.apaas.em2.oraclecloud.com/
Drawing Board Demo

Splitting into microservices using a distributed cache service

Clients
- HTML5 Browser
  - send(...) onEvent(...) 
  - DrawingService.query(...) 
  - webSocketSend.send(...) 
- WebView/WebKit 
- JavaFX

Coherence Cluster
- JAX-RS/SSE Microservice
- Websocket Microservice

Grizzly
- HTTP/S
- JSON
- SSE

DataProvider uses a Coherence distributed Cache

JavaFX

Copyright © 2015, Oracle and/or its affiliates. All rights reserved. |
Drawing Board Demo
Scalability and HA for SSE and WS Microservices with OTD and WLS Multitenancy
Summary

• Microservices are a valuable architectural technique, but:
  – not necessarily for everyone
  – not necessary always
  – not necessarily all-at-once

• Building microservices with Jersey is easier
  – Many microservices-related features in Jersey are going to be standardized

• Oracle is working on providing a cloud platform for microservices
Integrated Cloud
Applications & Platform Services