Transactions in your microservices architecture

Dawn Parzych (CatchPoint)
What?

• Transaction and Micro-Services
• MicroProfile LRA (Long Running Actions)
• Demo
Rudy De Busscher

• Payara
  • Service team

• Involved in
  • Committer of MicroProfile
  • Committer in Eclipse EE4J groups
  • Java EE Security API Expert group member

@rdebusscher
https://blog.payara.fish/
https://www.atbash.be
Transaction

• Typical concept from the ‘Monolith’
• Start - End
• Typical short - to avoid locks
• Multiple datasources supported with XA transactions
Transaction

• Distributed - XA transactions
• 2 Phase Commit
• Not scalable
  • Locks
To be ACID or Not

- Atomicity: All or Nothing
- Consistency: All values are aligned
- Isolation: Different transactions can’t see ‘temporal’ values
- Durable: Reliably Stored

- CAP Theorem
  - Consistency - Availability - Partitioning
Long Running Transactions

• Multiple Datasource
• Several steps over long period
• Single Unit of Work?
Example

• Book Flight
• Book Hotel
• Payment
MicroService world

- No **Atomicity**
- Eventual **Consistency**
- No **Isolation**
- **Durable**
Saga Pattern

Ensures that each step of the business process has a compensating action to undo the work completed in the case of partial failures.
- Optimizing Enterprise Java for a Micro-services Architecture
- Based on some Java EE (Jakarta EE) specs

<table>
<thead>
<tr>
<th>Open Tracing 1.3</th>
<th>Open API 1.1</th>
<th>Rest Client 1.3</th>
<th>Config 1.3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fault Tolerance 2.0</td>
<td>Metrics 2.0</td>
<td>JWT Propagation 1.1</td>
<td>Health Check 2.0</td>
</tr>
<tr>
<td>CDI 2.0</td>
<td>JSON-P 1.1</td>
<td>JAX-RS 2.1</td>
<td>JSON-B 1.0</td>
</tr>
</tbody>
</table>

**MicroProfile 3.0**

- Blue = New
- Purple = Updated
- Orange = No change from last release (MicroProfile 2.2)
MP Long Running Action

- First release candidate available
- API more or less stable
MicroProfile LRA

• Long Running Actions

• Features
  • Loose coupling
  • Guaranteed a globally consistent outcome
  • Compensatable actions (SAGAs)
LRA Components

- Participant
  - JAX-RS resource
  - Enlist
- Participant
  - JAX-RS resource
- Participant
  - JAX-RS resource
- Coordinator
  - Callback
Demo
Some Key code concepts

- @LRA
  - LRA Level / defines transaction

- @Compensate
  - Participant level: Not present, not a participant.

- @Complete
  - Participant level

- LRA ID ( = URI)
  - Uniquely defines the Long Running Action (transaction)
  - Propagated through Header
LRA Types

- Required for each JAX-RS resource

<table>
<thead>
<tr>
<th>Participant</th>
<th>Active LRA</th>
<th>No LRA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required</td>
<td>Use Active</td>
<td>Start New</td>
</tr>
<tr>
<td>Requires New</td>
<td>Start New</td>
<td>Start New</td>
</tr>
<tr>
<td>Mandatory</td>
<td>Use Active</td>
<td>Error</td>
</tr>
<tr>
<td>Supports</td>
<td>Propagate</td>
<td>-</td>
</tr>
<tr>
<td>Never</td>
<td>Error</td>
<td>-</td>
</tr>
<tr>
<td>Not Supported</td>
<td>Not propagated</td>
<td>-</td>
</tr>
</tbody>
</table>

Not listed: type Nested
Distributed

Service A

Start

Service B

End

Service C
Returning Participant Status

- **Immediate**
  - Return type `ParticipantStatus, void`

- **Asynchronous return**
  - Return `CompletableFuture<ParticipantStatus>`

- **Idempotent @Complete/@Compensate**
  - Return type `ParticipantStatus`
    - `ParticipantStatus.Completing` / `ParticipantStatus.Compensating`

- **Through @Status method**
  - First call to `@Complete/@Compensate`
  - Following calls to `@Status`
By Default, LRA runs forever
But also by Default, immediately closed
Closed explicitly (end=true)
Define a timeout -> cancelled / all participants compensated.
Remove Participant

- @Leave
- Remove Participant from LRA
- Developer responsible for cleanup
  - No @Compensate/@Complete/@Status called
Feedback final status

• @AfterLRA
  • What was the final outcome

• For participants
• For ‘parties’
  • Can be used for any @LRA, not @Compensate annotated class.
Store LRA Info

- LRA only performs orchestration
- Participant needs to keep track of LRA
- Store LRA Id as part of your business data
- Extension
  - LRAData
  - ParticipantData
Takeaways

- Classic approach for Transactions and Long Running Action not applicable for micro-services
- MicroProfile LRA uses Compensatable actions
- Loosely coupled on top of JAX-RS resources
- Specification: In progress
Code

• Project https://github.com/eclipse/microprofile-lra

• Demo code
  • https://github.com/rdebuscher/mp-lra-demo
Thank You

Not using the Payara Platform yet? Download the open source software: Payara Server or Payara Micro

https://payara.fish/downloads

Need support for the Payara Platform?

https://payara.fish/support