

# Transactions for Microservices?

Tom Jenkinson

January 2018

RED HAT



Who has heard of:

Container

Docker

Kubernetes

Transactions

Two-phase commit (for example JTA)

Microservices



# System faults

- . Machines (and software!) fail...
  - Things get better with each generation
    - Individual machines have a lower probability of failure
    - Ran at scale the system as a whole sees an increase in the number of failures
- . Failures of centralized systems are difficult to handle
  - But failures of distributed systems are much more difficult!
    - Microservices are all about distributed systems

## Why we even need to consider this!

Whilst there are a number of architectural patterns still to be agreed upon by practitioners, some agreements appear to have solidified and one of these is that services should have control of their own data, possess their own data store instance and eschew distributed transactions.

# What is a transaction and how can it help me?

- Mechanistic aid to achieving “**correctness**”
  - Provides an “**all-or-nothing**” property to work that is conducted within its **scope**
  - **Guarantees** shared state is protected from conflicting updates from multiple users
- There are many models...

- “Correctness” in this case is moving a program from one consistent state to another
  - Between the consistent states is where you need the transaction as you are in inconsistent state at that time
- Provides an “all-or-nothing” property to work that is conducted within its scope
  - Even in the presence of failures
- Ensures that shared state is protected from conflicting updates

- . from multiple users
  - Resulting in the “Guaranteed” shared global consensus on the outcome
- . Facilitates the development of software that is more reliable and responsive to faults in other components
- . There are many models
  - Not just two-phase commit!



The graphic on the right tries to capture that these properties are graded from say 0 to 1

- All or nothing
- When it is achieved? Now-eventual?
- Can others see the updates during the process
- Will these changes be guaranteed to be seen in the future

# Transactions for Microservices?

- Microservice interactions are typically complex
  - Involving many parties
  - Possibly extending to different organisations
  - Potentially lasting for hours or days
- They must be coordinated somehow to deal with failures in these interactions
- So can full-ACID transactions help us to achieve “correctness”?

## Well, yes but..

- Full-ACID transactions implicitly assume
  - One service can **block** another service
  - You can cope with resources being locked for periods of **time**
- Therefore full-ACID does not work optimally in
  - Loosely coupled environments!
  - Long duration activities!

- Closely coupled environment
  - One service can block another service
    - Must have control over the availability of these services
- Short-duration activities
  - Where you can cope with resources being locked for periods of time



# If full-ACID isn't suitable, what can we do?

- Relax some of the properties!
  - Let's take a simple case of booking a seat on a plane and travel insurance....

# Relax atomicity

- Sometimes it may be desirable to cancel some work without affecting the remainder
- Similar to nested transactions
  - Work performed within scope of a nested transaction is provisional
  - Failure does not affect enclosing transaction
- In our example:
  - It may still be preferential to get the airline seat even without travel insurance
    - The travel insurance might have been a nested transaction

For example, WS-BA allows non atomic outcomes [1] but LRA does not:

our LRA solution is atomic in the sense that when the LRA is closed/cancelled **ALL** participants are told to complete/compensate and this satisfies the definition of atomic. That said we can still get the partial outcomes that you refer to later by starting nested LRAs and (if required) cancelling those half way through the enclosing LRA. Tangentially related is the fact that compensations can themselves start new LRAs (since the compensation may need to run in the context of another long running action).

[1] WS-BA

(<http://docs.oasis-open.org/ws-tx/wstx-wsba-1.1-spec-os/wstx-wsba-1.1-spec-os.html>) says:

"A coordinator for an AtomicOutcome coordination type **MUST** direct all participants either to close or to compensate. A coordinator for a MixedOutcome coordination type **MUST** direct all participants to an outcome but **MAY** direct each individual

participant to close or compensate. All Business Activity coordinators **MUST** implement the AtomicOutcome coordination type. A Business Activity coordinator **MAY** implement the MixedOutcome coordination type."

# Relax consistency

- ACID transactions are all about strong global consistency
  - All participants remain in lock-step
- But that does not scale well
  - Replication researchers have known about this for years
    - Weak consistency replication protocols developed for large scale (number of replicas and physical deployment)
  - Weak/eventual consistency is now a generally accepted approach
- In our example:
  - It might have very little consequence to other users of the system if the reservation of the airline seat is seen by other users of the system before the purchase of the insurance assuming eventually all updates are consistent


# Relax isolation

- . Internal isolation or resources should be a decision for the service provider
  - E.g., commit early and define compensation activities
  - However, it does impact applications
    - . Some users may want to know a priori what isolation policies are used
- . Undo can be whatever is required
- . Regarding the example:
  - It might have very little impact to other users of the system if the reservation of the airplane seat is seen by other users but subsequently released back to the pool of available seats if a transaction is undone

- . Undo can be whatever is required
  - Before and after image
  - Entirely new business processes

# Relax durability

- Although this is possible in some models its use is not appropriate in the use cases we try to address
- For example, if durability is relaxed:
  - We might be able to book the airline seat but subsequently a power loss of the transaction manager means we never complete the travel insurance booking



Because there is no isolation the system can become inconsistent.

But the compensatory actions will bring the system back into a consistent state (eventually).

For example if a business participant fails then the client may decide to cancel the action. That leaves a situation where earlier participants have done something that will never be consistent with the target outcome (eg a PO + payment + shipment). After the compensations have been run the system ought to be consistent again.

There is loose coupling because there is no locking. The benefit of no locking is that it is

safe to perform actions that have a long duration.

In XA there can be implicit data sharing between services. That should never happen in loosely coupled systems.

2 services connect to the same DB table

In a loosely coupled service based environment there is sometimes a need for different services to provide consistency guarantees. Typical examples include:

- order processing involving three services (take order, bill customer, ship product). If the shipping service finds that it is out of stock then the customer will have been billed with no prospect of receiving his item;
- an airline overbooks a flight which means that the booking count and the flight capacity are inconsistent.

There are various ways that systems overcome such inconsistency but it would be advantageous to provide a generic solution which handles failure conditions, maintains state for those flows that span long periods of time and ensures that correcting activities are called correctly.

Traditional techniques for guaranteeing consistency in distributed environments has



focused on XA transactions where locks may be held for long periods thereby introducing strong coupling between services and decreasing concurrency to unacceptable levels. Additionally, if such a transaction aborts then valuable work which may be valid will be rolled back. In view of these issues an alternative approach is desirable.

### Goals

- support long running actions
- no strong coupling between services
- allow actions to finish early
- allow compensating actions if a business activity is cancelled
- support short running actions that may need to be started to facilitate the execution of a step in a long running action

## The result: Long Running Action

- A draft specification contributed to the MicroProfile community
- Defines an API for to coordinate activities allowing both
  - Long running activity
  - Loosely coupled processes
- Provides:
  - CDI annotations which can be applied to JAX-RS resources
  - A pure java based coordination API
- Built on the premise that each action defines a corresponding “undo” action
  - Forward compensation during activity is allowed
    - Keep business process making forward progress

RED HAT



- Loosely coupled processes
  - The coordinator does not understand the business logic
- 
- 
- Although the coordinator implementation runs on WildFly Swarm and OpenShift it should run fine in other environments.
  - Running participants in environments such as Spring Boot should be feasible

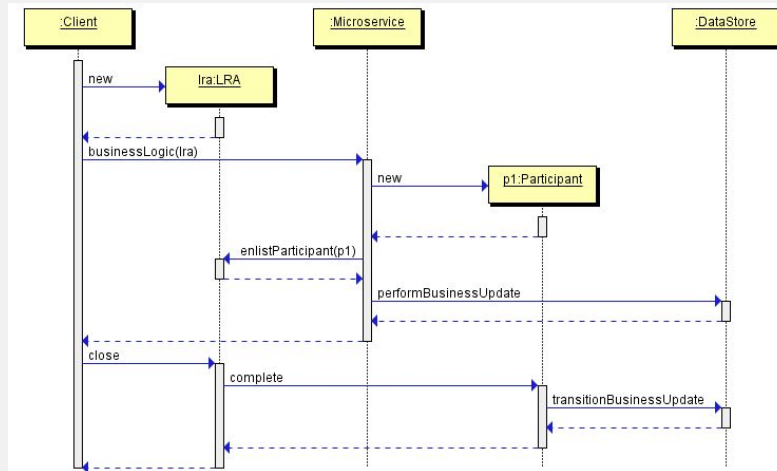
We propose a compensation based approach in which participants make changes visible but register a compensatory action which is performed if

something goes wrong.

Within the LRA model, an activity reflects business interactions: all work performed within the scope of an activity is required to be compensatable.

Therefore, an activity's work is either performed successfully or undone. How services perform their work and ensure it can be undone if compensation is required are implementation choices and is not exposed to the LRA model which simply defines the triggers for compensation actions and the conditions under which those triggers are executed. In other words, an LRA coordinator is concerned only with ensuring participants obey the protocol necessary to make an activity compensatable (and the semantics of the business interactions are not part of the model). Issues such as isolation of services between potentially conflicting activities and durability of service work are assumed to be implementation decisions. The coordination protocol used to ensure an activity is completed successfully or compensated is not two-phase and is intended to better model interactions between microservices. Although this may result in non-atomic behaviour for the overall business activity, other activities may be started by the service to attempt to compensate in some other manner.

# Long Running Actions: Actors and interactions



- In the model, an LRA is tied to the scope of an activity so that when the activity terminates the LRA coordination protocol will be automatically performed either to accept or to compensate the work. For example, when a user reserves a seat on a flight, the airline reservation centre may take an optimistic approach and actually book the seat and debit the user's account, relying on the fact that most of their customers who reserve seats later book them; the compensation action for this activity would be to un-book the seat and credit the user's account.
- As in any business interaction, service activities may or may not be compensatable. Even the ability to compensate may be a transient

- capability of a service. A Compensator (or simply LRA participant) is the LRA participant that operates on behalf of a service to undo the work it performs within the scope of an LRA or to compensate for the fact that the original work could not be completed.
- 
- 
- 
- a window exists (per LRA) where you can't process requests because of the spof where you need to detect the coordinator is dead
  - register compensations and the ability to close/cancel the LRA blocks progress with a single coordinator
  - May be reduced by adding Health check to your PaaS for the coordinator to restart the coordinator
  - We are currently working on further addressing the requirements of fault tolerance while avoiding the single point of failure that is oft-argued
    - Specialised REST load balancer for PaaS
      - We are looking at load balancing to a cluster of coordinators with affinity via the x-lra header

- This is a SPOF per TX but not per coordinator
- Taking advantage of presumed nothing to allow any coordinator to field any request
  - Use the persistent storage to “communicate”

# Live coding demo

RED HAT



# Notable tech used in the demo



RED HAT



## OpenShift

PaaS - Enterprise Kubernetes

used to provision the docker containers which host LRA and microservices

Allows stateful and stateless apps so I can have persistent logs for the LRA coordinator

## Minishift

neat tool to run openshift locally

will be using it to run all of the microservices

## WildFly swarm

"right size" -- i.e compose -- Microservices running on Java EE into a fat-jar

with only the parts of EE I require

it does this by asking the user to select fractions - think component for items such as

it's MicroProfile compatible

## Narayana

Project I lead

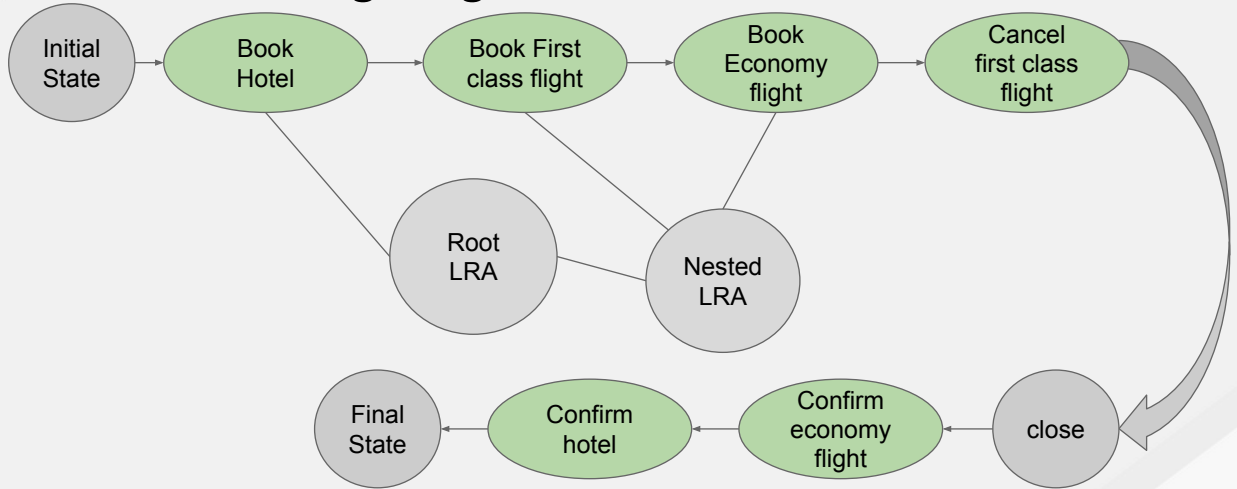
30 years of expertise in the area of transaction processing. JTS, WS, STM, C++ been there done it.

Provides the LRA coordinators

Using 5.7.0 can be downloaded from [narayana.io](http://narayana.io)



# What we are going to do?



RED HAT



This source code is available in our repository

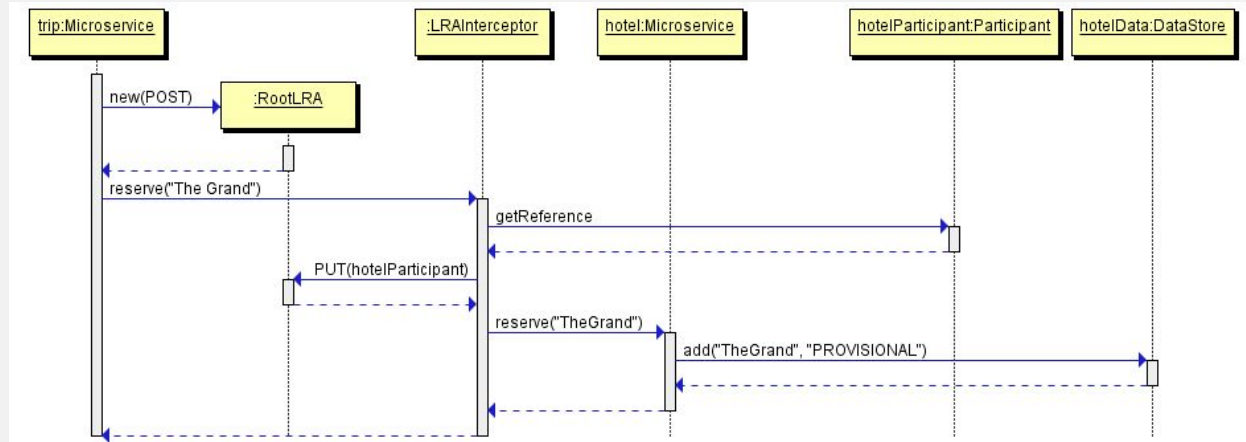
<https://github.com/jbosstm/conferences/>

Slide shows the overall structure of the application which is pretty much the "hello world" of Transaction applications

- \* a business process highlighted in green
  - \* a trip booking microservice
  - \* hotel and flight microservices
- \* root and nested coordinators created and used by the microservices
  - \* what's particularly special is we are using them in a Saga and where locking is entirely reasoned in the business logic in the application.
    - \* f.ex the "locking" of flights is the business level seat being confirmed
    - \* we find this is a natural way to view locks and one that resonates with developer and analyst alike
  - \* running in different processes to simulate how it might be desirable to have a coordinator under the control of different trust domains

I now have a few sequence diagrams to help ground us as we walkthrough the application:

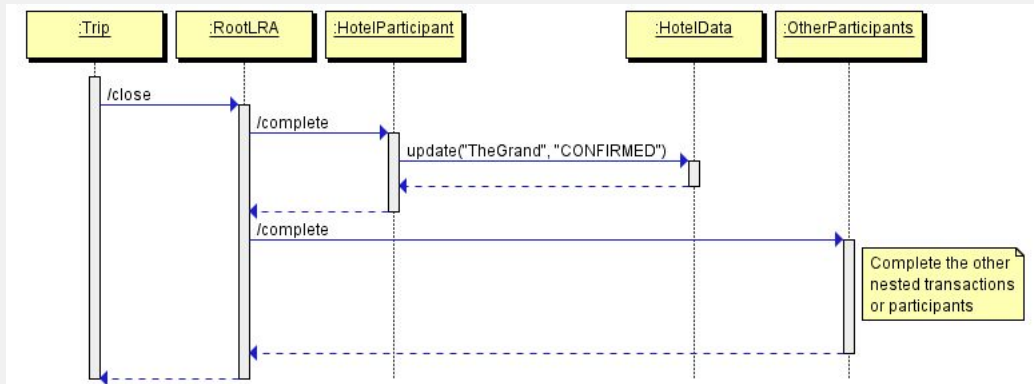
# Deep-dive: Enlisting the Hotel Microservice



Slide 1: shows the interactions between the trip microservice and the hotel microservice

- \* I have called out some of the work that the LRA implementation is doing for you as LRA Interceptor
- \* The business logic is going to PROVISIONAL

# Deep-dive: Completing the booking



Slide 4: shows what happens when a client will ask the trip microservice to complete

- \* The nested LRA is called again but as the nested coordinator knows the outcome it won't reissue command

# Running the demo!

Over to the IDE :)

RED HAT



Will be using a set of preconfigured folders

Not trying to hide transaction code - just a lot to cover

You might spot some dockerfiles in here

used later when I take the microservices and run them on the cloud - minishift

# Conclusions

- Transactions for Microservices?
  - Yes!
  - There are many transaction models that don't necessitate full-ACID
    - Long Running Action specification
- Future work in MicroProfile
  - If you don't agree with this model then get involved and help define more

# Where can I find out more?

Narayana.io/community

- Forums
- IRC
- Blog
- <https://github.com/jbosstm/conferences/201801-transactions-microservices/>

MicroProfile mailing list

- “The need for transactions in the MicroProfile”
  - <https://groups.google.com/forum/#!starred/microprofile/CJirjFkM9Do>

# THANK YOU



[plus.google.com/+RedHat](https://plus.google.com/+RedHat)



[facebook.com/redhatinc](https://facebook.com/redhatinc)



[linkedin.com/company/red-hat](https://linkedin.com/company/red-hat)



[twitter.com/RedHatNews](https://twitter.com/RedHatNews)



[youtube.com/user/RedHatVideos](https://youtube.com/user/RedHatVideos)

RED HAT

