Java. Cloud. Leadership.

WildFly-Swarm and transactions DRAFT

Mark Little, VP

Transactions and microservices?

- “Transactions should be contained within a single service”
  - “A microservice should be tied to a single database”
- “Atomicity is overrated”
- “Transactions limited scalability”

When to use transactions

- When you need ACID semantics!
- Or ...
  - When you have a need to guarantee consensus in the presence of failures
  - When you need isolation and consistency across failures
- Relaxing ACID semantics is possible
- Recoverable transactions may be sufficient

```java
@Path("/")
public class MyResource
{
    @GET
    @Produces("text/plain")
    public String init() throws Exception
    {
        return "Active";
    }

    @Path("begincommit")
    @GET
    @Produces("text/plain")
    public String beginCommit() throws Exception
    {
        UserTransaction txn = (UserTransaction) new InitialContext().lookup("java:comp/UserTransaction");
        String value = "Transaction ";

        try
        {
            txn.begin();

            value += "begun ok";

            try
            {
                txn.commit();

                value += " and committed ok";
            }
            catch (final Throwable ex)
            {
                value += " but failed to commit";
            }
```

```java
package org.wildfly.swarm.examples.transactions;

import org.wildfly.swarm.container.Container;
import org.wildfly.swarm.jaxrs.JAXRSDeployment;
import org.wildfly.swarm.transactions.TransactionsFraction;

/**
 * @author nmcl
 */

public class Main {
    public static void main(String[] args) throws Exception {
        Container container = new Container();

        /*
     * Use specific TransactionFraction even though it doesn't do
         * any more than the default one - for now.
         */

        container.subsystem(new TransactionsFraction(4712, 4713));

        // Start the container

        container.start();

        /*
         * Now register JAX-RS resource class.
         */

        JAXRSDeployment appDeployment = new JAXRSDeployment(container);
        appDeployment.addResource(MyResource.class);

        container.deploy(appDeployment);
    }
}
```

Software Transactional Memory

- ACI … no D
- Framework for building transactions
- Using JTA where wanted
- Volatile updates, even shared between multiple services, more appropriate
- Compensations

```
@Optimistic
public class SampleLockable implements Sample
{
   public SampleLockable (int init)
   {
      _isState = init;
   }

   @ReadLock
   public int value ()
   {
      return _isState;
   }

   @WriteLock
   public void increment ()
   {
      _isState++;
   }

   @WriteLock
   public void decrement ()
   {
      _isState--;
   }

   @State
   private int _isState;
}
```

```
MyExample ex = new MyExample(10);
Container<Sample> theContainer = new Container<Sample>();
AtomicAction act = new AtomicAction();

act.begin();

obj1.increment();

act.commit();
```