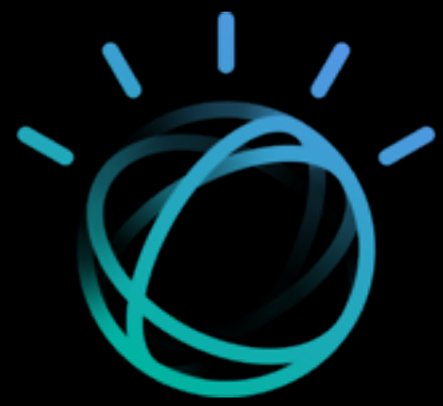
IBM Liberty Profile

Propagação de Transações em Ambiente Kubernetes



Introdução

Objetivo Geral

O objetivo deste documento e apresentar uma solução para a propagação de transações utilizando Liberty Profile em ambiente Kubernetes.

Delineamento do Problema

O Liberty Profile é uma runtime de execução java aderente aos profiles MicroProfile, WebProfile e JEE Profile desenvolvido e otimizado para execução em containers. O Liberty Profile é distribuído nas modalidades Open Source (openliberty.io) ou licenciado pela IBM, podendo executar em infraestruturas tradicionais ou em ambiente Kubernetes. O Liberty Profile também é distribuído como parte do licenciamento do IBM Cloud Private e, portanto, apresenta-se como runtime natural para execução de código java dentro deste ambiente kubernetes. Parte do desafio da modernização da infra-estrutura para adoção de um ambiente Kubernetes é garantir que determinados requisitos de arquitetura, muitas vezes incompatíveis ou inexistentes dentro de um modelo de microserviços, coexistam com o processo de modernização. Um exemplo disto é a propagação de transações entre servidores de aplicação. Embora esta funcionalidade esteja madura e seja natural para aplicações EJB desenvolvidas sobre a plataforma WebSphere Traditional Profile, este nível de acoplamento transacional pode impor desafios para execução em infraestrutura Kubernetes, tais como:

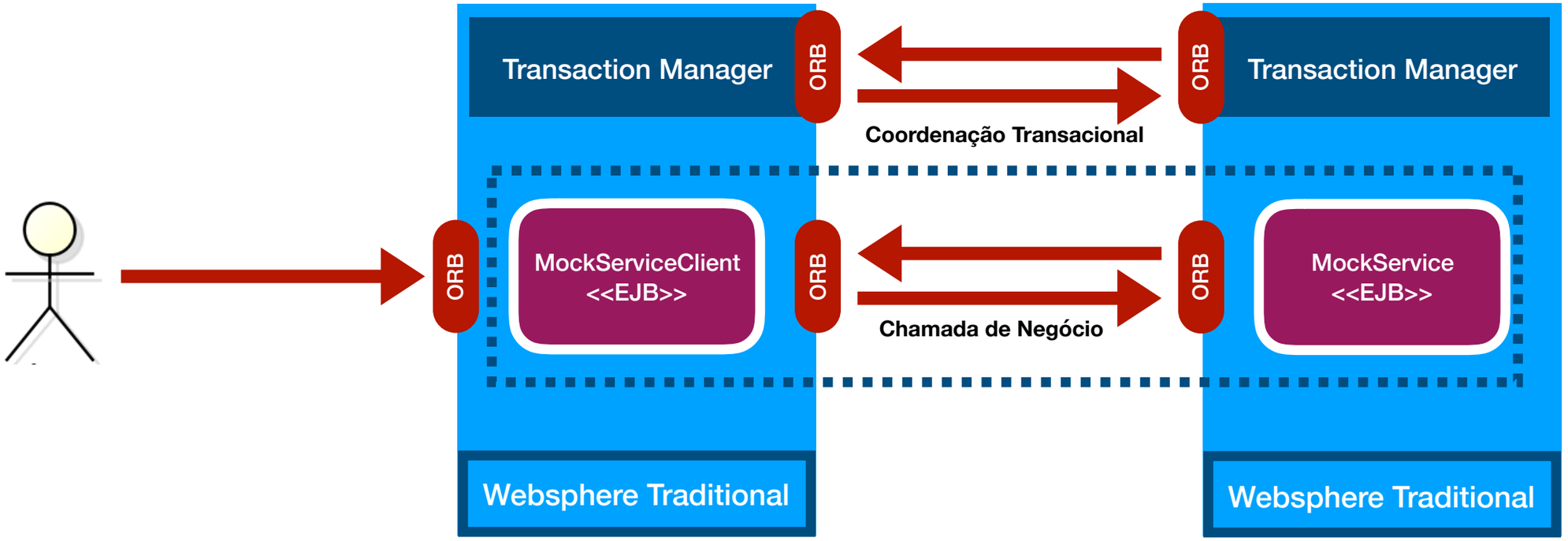
- Complexidade de comunicação imposta pelo protocolo IIOP e pelo ORB
- Limitações quanto ao balanceamento de carga
- Impossibilidade de uso do Ingress Controller do Kubernetes para balanceamento e virtualização de hosts.
- Comunicação Bidirecional pouco transparente

Este trabalho buscará apresentar uma alternativa ao protocolo RMI/IIOP para propagação de transações em ambiente Kubernetes permitindo que aplicações legadas ou dependentes do modelo de desenvolvimento orientado a componentes JEE tenham condições de executar em ambiente Kubernetes e participem do processo de adoção de uma cloud privada.

Descrição do Cenário

Neste cenário teremos duas aplicações, cada uma executando em um diferente servidor de aplicações. A aplicação cliente é chamada Enterprise-Client e a aplicação de backend é chamada Enterprise-Server. O cenário assume como premissa que o contexto de transação iniciado na aplicação Enterprise-Client precisa ser propagado para o Enterprise-Server. Portanto, o Enterprise-Client assume como coordenador da transação e o Enterprise-Server o participante

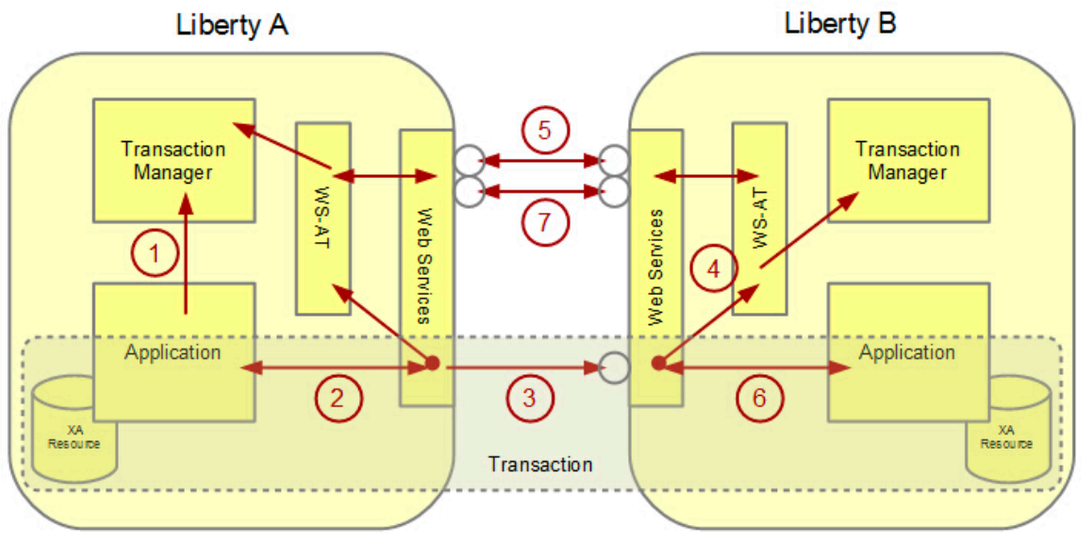
A ilustração abaixo representa o modelo tradicional implementado utilizando WAS BASE.



Descrição do Cenário

Na ilustração abaixo observamos o mesmo esquema de funcionamento, contudo, utilizando o padrão WS-AtomicTransaction 2.0 para implementação da propagação da transação.

O cenário foi implementado de forma que os componentes EJB sejam mantidos inalterados, incluíndo suas interfaces, adicionando um stub para comunicação via JAX-WS 2.2 e anotações pertinentes com esta especificação. A comunicação via JAX-WS permitirá a propagação da transação dentro do ambiente Kubernetes utilizando o protocolo HTTP, e portanto, elegível para balanceamento via Ingress Controller. A propagação poderá ocorrer tanto entre participantes dentro do ambiente Kubernetes quanto vindo de um Liberty Profile localizado fora deste ambiente.



The WS-AT support is an interoperability protocol that introduces no new programming interfaces for transactional support. Global transaction demarcation is provided by standard enterprise application use of the Java™ Transaction API (JTA) UserTransaction interface. If an application component that is running under a global transaction makes a web services request, a WS-AT CoordinationContext is implicitly propagated to the target web services, but only if the appropriate application deployment descriptors have been set, as described in the topic about configuring transactional deployment attributes.

If the application server is the system hosting the target endpoint for a web services request that contains a WS-AT Coordination Context, the application server automatically establishes a subordinate JTA transaction in the target run time environment that becomes the transactional context under which the target web services application runs.

Implementação

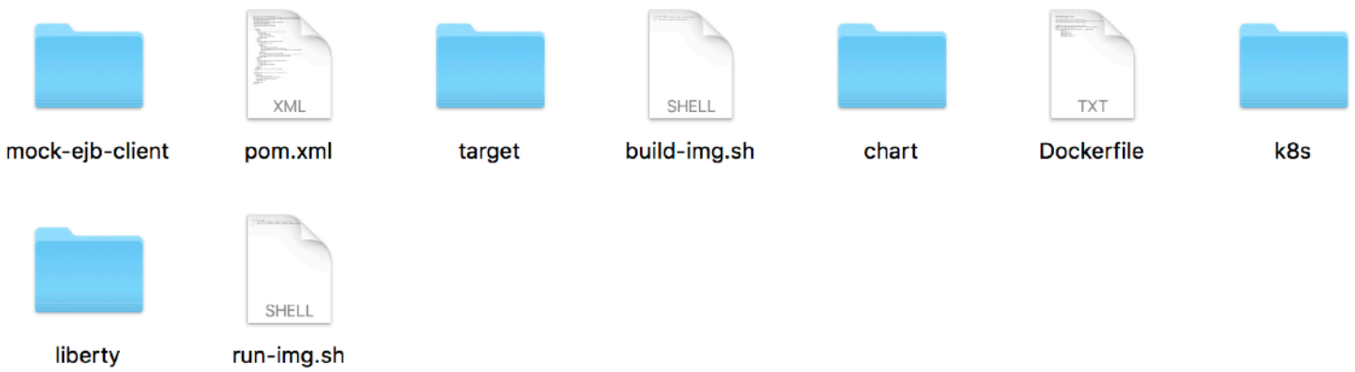
O Asset desenvolvido visa implementar o comportamento descrito na página anterior de uma maneira compatível com o ambiente Kubernetes. A execução em Kubernetes nos impõe algumas implicações:

- Em geral, a comunicação entre pods é estabelecida por intermedio de um Service.
- O Service apresenta uma abstração entre as várias possíveis replicas de um Pod Stateless, atuando como um balanceador.
- Por padrão o Kubernetes não cria registro do hostname do Pod em seu DNS.
- A comunicação em um cenário de propagação de transação não é unidirecional, portanto, o servidor Chamado necessitará acessar um callback no serviço de WS-AtomicTransaction do servidor Chamador a fim de se registrar como participante da transação. E da mesma forma o servidor Chamador precisará chamar o serviço de WS-AtomicTransaction no servidor Chamado a fim de mudar o estado da transação.
- As comunicações paralelas ao negócio destinadas a manutenção do estado transacional, devem manter a identidade dos servidores participantes, portanto, o balanceamento de carga via Service não poderá ocorrer.

A fim de atender os critérios descritos aqui descreveremos a seguir a configuração proposta.

Implementação

A implementação foi baseada em build utilizando MAVEN. O sistema de arquivos para ambos projetos (EJB Cliente e EJB BackEnd) apresentam a seguinte estrutura:



mock-ejb / mock-ejb-client -> Esta pasta contem o projeto do ejb-jar utilizado para implementar o componente EJB.

target -> Pasta com o resultado da compilação do MAVEN

k8s -> Pasta com os artefatos para deployment em Kubernetes

liberty -> Pasta com as configurações do Liberty que serão inseridas dentro da imagem do container

pom.xml -> Arquivo com as configurações de build do MAVEN

build-img.sh -> Script responsável pela criação da imagem, executar ./build-img.sh 0.0.1 sendo 0.0.1 o número da tag.

Dockerfile -> Arquivo responsável pela montagem da imagem

run-img.sh -> Script para executar o container localmente, executar ./run-img.sh 0.0.1 sendo 0.0.1 o número da tag.

Implementação

A configuração do EJB de Backend é baseada em EJB 3.1 com JAX-WS 2.2 e WS-AtomicTransaction 2.0. Fundamentalmente a configuração foi baseada em anotações e seguiram os seguintes passos:

(A) Preparação do EJB de BackEnd

```
package ibm.sample.mock.service;

import ibm.sample.mock.view.MockServiceLocal;

@WebService(targetNamespace="http://sample.com", name="MockServiceWS", serviceName="MockServiceWS")
@Stateless
@Local(MockServiceLocal.class)
@Remote(MockServiceRemote.class)
@TransactionManagement(value=TransactionManagementType.CONTAINER)
@TransactionAttribute(value=TransactionAttributeType.REQUIRED)
public class MockService implements MockServiceRemote {

    public MockService() {}

    @WebMethod
    @Override
    public double meanOperation( @WebParam(name = "vector") double[] vector) {
        double sum = 0;
        for (double number : vector) {
            sum = sum + number;
        }

        return sum / vector.length;
    }

    @WebMethod
    @Override
    public double sd(@WebParam(name = "vector") double[] vector) {
        double mean = this.meanOperation(vector);

        double sum = 0;
        for (double number : vector) {
            sum = sum + (Math.pow((mean - number), 2));
        }
        return Math.sqrt(sum / vector.length);

    }

}
```

Implementação

A configuração do EJB de Backend é baseada em EJB 3.1 com JAX-WS 2.2 e WS-AtomicTransaction 2.0. Fundamentalmente a configuração foi baseada em anotações e seguiram os seguintes passos:

(B) Geração dos Stubs Web Service JAX-WS 2.2 para incorporação no projeto do EJB Cliente. Este execução deste passo é necessário realizar o deployment do EJB de Backend para que sua interface seja automaticamente descrida como um WSDL pelo Liberty Profile. Para a geração dos stubs o Liberty oferece um utilitário chamado wsimport.sh localizado em <LIBERTY_HOME>/bin/jaxws/wsimport.sh. Para criação dos stubs executar:

```
./wsimport -clientjar minhaLibCliente.jar -target 2.2 http://enterprise-server.local:32420/mock-ejb-0.0.1-SNAPSHOT/MockServiceWS?wsdl
fazendo parse do WSDL...


Fazendo download do WSDL e dos metadados associados


Gerando o código...


Compilando o código...


Arquivando os artefatos gerados em ./minhaLibCliente.jar.
```

Implementação

A configuração do EJB de Backend é baseada em EJB 3.1 com JAX-WS 2.2 e WS-AtomicTransaction 2.0. Fundamentalmente a configuração foi baseada em anotações e seguiram os seguintes passos:

(C) Preparação do EJB Consumidor. Este passo é baseado em anotações JAX-WS realizadas no consumidor do Serviço. É necessário também que o WSDL do serviço a ser consumido seja depositado em META-INF/wsdl/ pois ele será referenciado desta maneira pelo stub.

Como podemos perceber ao lado, o próprio EJB consumidor foi marcado para expor uma interface Web Service. A demarcação transacional respeitará a anotação REQUIRES_NEW, iniciando uma nova transação a partir deste ponto. A anotação @WebServiceRef faz a injeção do Stub na Interface Remota MockServiceWS (derivada da interface remota do EJB de BackEnd). É possível utilizar outras formas de lookup do serviços, por exemplo, InitialContext programaticamente, ou ainda, injeção via @Resource, para isso o serviço deve estar configurado no ejb-jar.xml utilizando a notação <service-ref/>.

```
package ibm.sample.mock.service;

import ibm.sample.mock.service.model.InvalidOperationException;

@WebService(targetNamespace="http://sample.com", name="MockServiceClientWS", serviceName="MockServiceClientWS")
@Stateless
@Remote(MockServiceClientRemote.class)
@TransactionManagement(value=TransactionManagementType.CONTAINER)
@TransactionAttribute(value=TransactionAttributeType.REQUIRES_NEW)
public class MockServiceClient implements MockServiceClientRemote {

    @WebServiceRef(MockServiceWS_Service.class)
    MockServiceWS mockService;

    public MockServiceClient() {

    }

    @WebMethod
    @Override
    public OperationResult callOperation(@WebParam(name = "operation") Operation operation) throws InvalidOperationExceptic
        OperationResult result = new OperationResult();

        if (operation.getOperationType().equalsIgnoreCase("mean")) {

            double mean = mockService.meanOperation(operation.getVectorAsList());
            result.setOperationResult(mean);
            result.setOperationType("MEAN");
            result.setReturnCode("OK");
            return result;
        } else

        if (operation.getOperationType().equalsIgnoreCase("sd")) {

            double sd = mockService.sd(operation.getVectorAsList());
            result.setOperationResult(sd);
            result.setOperationType("SD");
            result.setReturnCode("OK");
            return result;

        } else {
            throw new InvalidOperationException("Operacao do Tipo: "+operation.getOperationType()+ " é invalido.");
        }
    }
}
```

Implementação

A configuração do EJB de Backend é baseada em EJB 3.1 com JAX-WS 2.2 e WS-AtomicTransaction 2.0. Fundamentalmente a configuração foi baseada em anotações e seguiram os seguintes passos:

(D) Configuração do Liberty Profile. A configuração do Liberty é injetada dentro do container durante sua montagem. As instruções para copia das configurações podem ser encontradas dentro do Dockerfile. A imagem ao lado demonstra a configuração utilizada neste cenário.

Para a configuração de WS-AtomicTransaction utilizamos a notação wsAtomicTransaction ativada pela feature wsAtomicTransaction-1.2. Esta notação, uma vez configurada, permitirá a propagação automática de transações via protocolo WS-AT dada as seguintes condições:

- Existe uma transação Ativa
- Tanto o Cliente quanto o BackEnd Suportam WS-AT 1.2

A notação wsAtomicTransaction foi configurada para receber o POD_IP como variável de ambiente. Esta configuração é necessária para informar ao servidor destino como realizar o callback para acesso ao serviço WS-AT deste servidor.

```
<server description="Enterprise-Server">

  <featureManager>
    <feature>javaee-7.0</feature>
    <feature>mpJwt-1.0</feature>
    <feature>mpRestClient-1.0</feature>
    <feature>apiDiscovery-1.0</feature>
    <feature>adminCenter-1.0</feature>
    <feature>mpConfig-1.2</feature>
    <feature>localConnector-1.0</feature>
    <feature>wsAtomicTransaction-1.2</feature>
  </featureManager>

  <variable name="defaultHostName" value="localhost" />

  <keyStore id="defaultKeyStore" password="Liberty" />

  <httpEndpoint host="${env.POD_IP}" httpPort="9080" httpsPort="9443"
      id="defaultHttpEndpoint"/>

  <orb id="defaultOrb">
    <serverPolicy.csiv2>
        <layers>
            <attributeLayer identityAssertionEnabled="true" />
            <transportLayer sslEnabled="true" />
        </layers>
    </serverPolicy.csiv2>
  </orb>

  <iiopEndpoint id="defaultIiopEndpoint" host="${env.POD_IP}" iiopPort="2809">
    <iiopsOptions iiopsPort="9402" sslRef="defaultSSLConfig" />
  </iiopEndpoint>

  <quickStartSecurity userName="admin" userPassword="admin" />

  <wsAtomicTransaction SSLEnabled="false" SSLRef="defaultSSLConfig"
           externalURLPrefix="http://${env.POD_IP}:9080" clientAuth="false"/>

  <application location="mock-enterprise.ear">
  </application>

    <logging  traceSpecification="Transaction=all:
              com.ibm.ws.transaction.services.WebAppTransactionCollaboratorImpl=all:
              org.apache.cxf.*=all:com.ibm.ws.jaxws.*=all"
        traceFileName="trace.log"
        maxFileSize="100"
        maxFiles="10"
        traceFormat="BASIC" />
</server>
```

Referências

Este asset esta disponível via git no endereço:

https://github.com/deivsonrayner/liberty-wsatomicTransaction

Outras referências:

OASIS WS-AtomicTransaction Spec.

http://docs.oasis-open.org/ws-tx/wstx-wsat-1.2-spec-os/wstx-wsat-1.2-spec-os.html

Visão Geral de Propagação de Transação via WS-AT usando Liberty Profile

https://www.ibm.com/support/knowledgecenter/en/SSEQTP_liberty/com.ibm.websphere.wlp.doc/ae/cwlp_dep_wsatoverview_lib.html

Configurações de Segurança para os Serviços WS-AT

https://www.ibm.com/support/knowledgecenter/en/SSEQTP_liberty/com.ibm.websphere.wlp.doc/ae/rwlp_dep_wsat_sec_lib.html

Obrigado!