adaptTo()

APACHE SLING & FRIENDS TECH MEETUP

BERLIN, 23-25 SEPTEMBER 2013

Inter-Sling communication with a message queue

Tomasz Rękawek

Agenda

- Code bookmarks
  - http://bit.ly/adaptto-jms
- Case 1: shared session
  - Running ActiveMQ in Sling
  - JMS connection provider
- Case 2: JMS-based discovery API implementation
  - Sling Message Listener
- Case 3: Reverse replication request
  - Targeting instance by its run mode

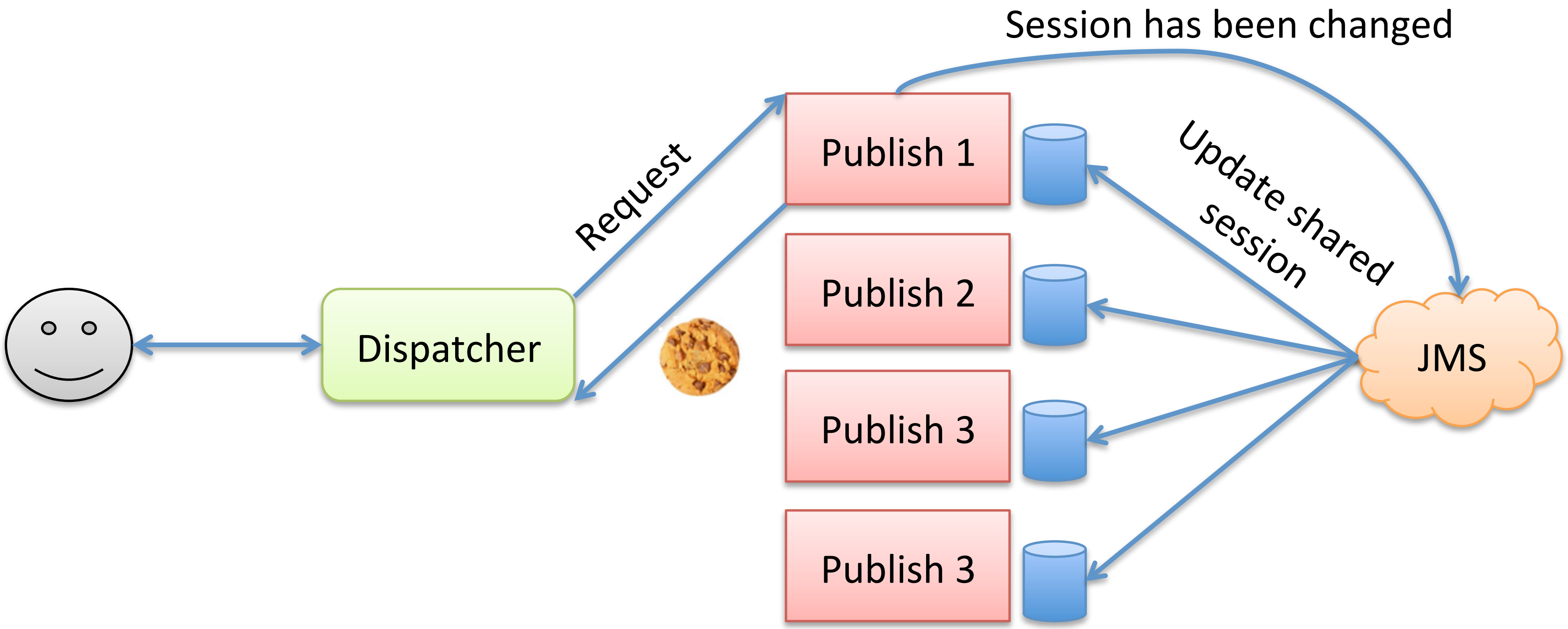
Solution overview

Shared session

Shared session

- We have *n* publishes and the dispatcher with `stickySession` enabled
- We use the servlet session
- In case one publish stops responding, the user will be redirected to the other publish
- We want the session to be transferred too

JMS-based shared session



Shared session storage – stores session from all publishes, with assigned unique shared session id

Cookie containing unique shared session id (different than JSESSION_ID) and unique instance id

Shared session

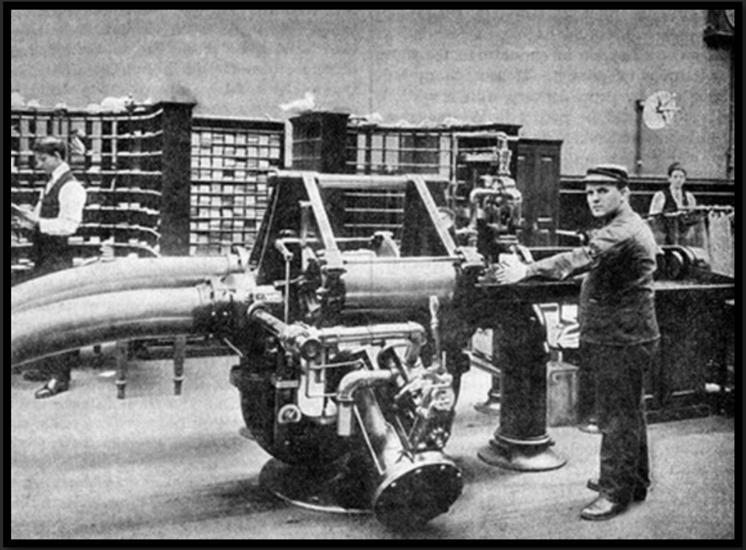
- If a publish notices that the instance id in the cookie is different from its own, it'll copy shared session values to the `HttpSession`
- The Administrator can configure names of the shared session properties (as a list of regular expressions)

Shared session – demo

- Local infrastructure description:
  - One author + two publishes
  - All available under `local.cq` domain
  - We can switch between publishes using `?publish2` query string
- http://local.cq/bin/cognifide/session.txt
- http://local.cq/bin/cognifide/session.txt/add
- http://local.cq/bin/cognifide/session.txt?publish2

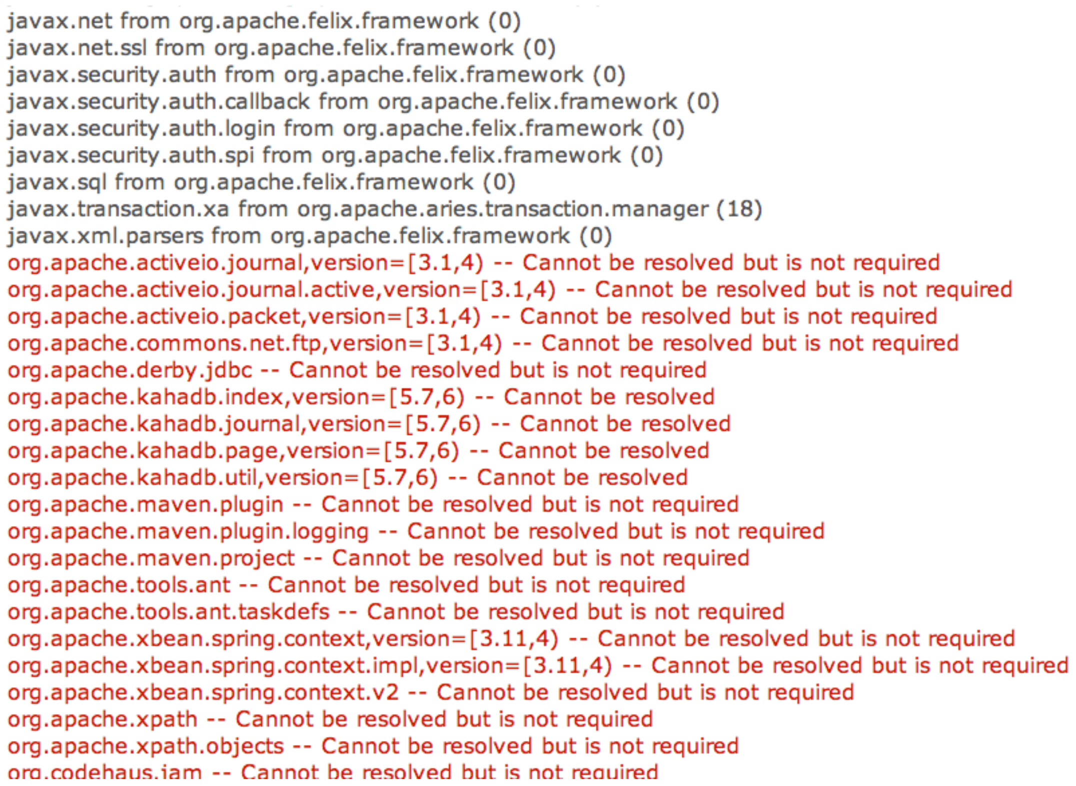
Shared session implementation details

Integrating ActiveMQ



Rough start: dependencies

- I tried to use the `activemq-core` package. This is what I saw in the /system/console:



```
javax.net from org.apache.felix.framework (0)
javax.net.ssl from org.apache.felix.framework (0)
javax.security.auth from org.apache.felix.framework (0)
javax.security.auth.callback from org.apache.felix.framework (0)
javax.security.auth.login from org.apache.felix.framework (0)
javax.security.auth.spi from org.apache.felix.framework (0)
javax.sql from org.apache.felix.framework (0)
javax.transaction.xa from org.apache.aries.transaction.manager (18)
javax.xml.parsers from org.apache.felix.framework (0)
org.apache.activeio.journal,version=[3.1,4) -- Cannot be resolved but is not required
org.apache.activeio.journal.active,version=[3.1,4) -- Cannot be resolved but is not required
org.apache.activeio.packet,version=[3.1,4) -- Cannot be resolved but is not required
org.apache.commons.net.ftp,version=[3.1,4) -- Cannot be resolved but is not required
org.apache.derby.jdbc -- Cannot be resolved but is not required
org.apache.kahadb.index,version=[5.7,6) -- Cannot be resolved
org.apache.kahadb.journal,version=[5.7,6) -- Cannot be resolved
org.apache.kahadb.page,version=[5.7,6) -- Cannot be resolved
org.apache.kahadb.util,version=[5.7,6) -- Cannot be resolved
org.apache.maven.plugin -- Cannot be resolved but is not required
org.apache.maven.plugin.logging -- Cannot be resolved but is not required
org.apache.maven.project -- Cannot be resolved but is not required
org.apache.tools.ant -- Cannot be resolved but is not required
org.apache.tools.ant.taskdefs -- Cannot be resolved but is not required
org.apache.xbean.spring.context,version=[3.11,4) -- Cannot be resolved but is not required
org.apache.xbean.spring.context.impl,version=[3.11,4) -- Cannot be resolved but is not required
org.apache.xbean.spring.context.v2 -- Cannot be resolved but is not required
org.apache.xpath -- Cannot be resolved but is not required
org.apache.xpath.objects -- Cannot be resolved but is not required
org.codehaus.jam -- Cannot be resolved but is not required
```

Providing necessary dependencies

- Some dependencies are taken from other OSGi bundles – we don't worry about them
- Some have to be embedded

```
<Embed-Dependency>activemq-core,geronimo-j2ee-
management_1.1_spec,ejb-api,jaxrpc-api</Embed-Dependency>
```

- Some can be ignored
  - Check the original `pom.xml` for optional dependencies

```
<Import-Package>
!javax.jmdns.*,!org.apache.activeio.*,!
org.apache.activemq.jaas.*,!org.apache.camel.*,!
org.apache.commons.net.*,…,*
</Import-Package>
```

Result: `sling-jms-activemq`

- We've created an OSGi bundle with all necessary dependencies embedded
- Optional dependencies are ignored and marked not-to-import
- Bundle provides ActiveMQ to any Sling instance

Shared session implementation details

JMS Connection Provider

Creating JMS connection with ActiveMQ

- In order to create JMS connection we need to import some ActiveMQ classes:

```
import javax.jms.Connection;
import org.apache.activemq.ActiveMQConnectionFactory;

ActiveMQConnectionFactory factory;
factory = new ActiveMQConnectionFactory();
Connection connection = factory.createConnection();
```

- So all JMS-related code will be also dependent on the ActiveMQ
- What if we want to change JMS implementation?

JMS connection provider

- We use OSGi to separate JMS implementation from it's interface
- `JmsConnectionProvider` – custom OSGi service providing `javax.jms.Connection`
- Bundle `sling-jms-api` contains service interface
- Implementation: `sling-jms-impl-activemq`
- Using `sling-jms-api` and the connection provider makes us independent from the JMS implementation

Example usage of JMS connection provider

```
@Component
public class MyComponent implements MessageListener {

   @Reference
   private JmsConnectionProvider connectionProvider;

   private javax.jms.Connection connection;

   @Activate
   protected void activate() throws JMSException {
      connection = connectionProvider.getConnection();
   }

   @Deactivate
   protected void deactivate() throws JMSException {
      connection.close();
   }
}
```

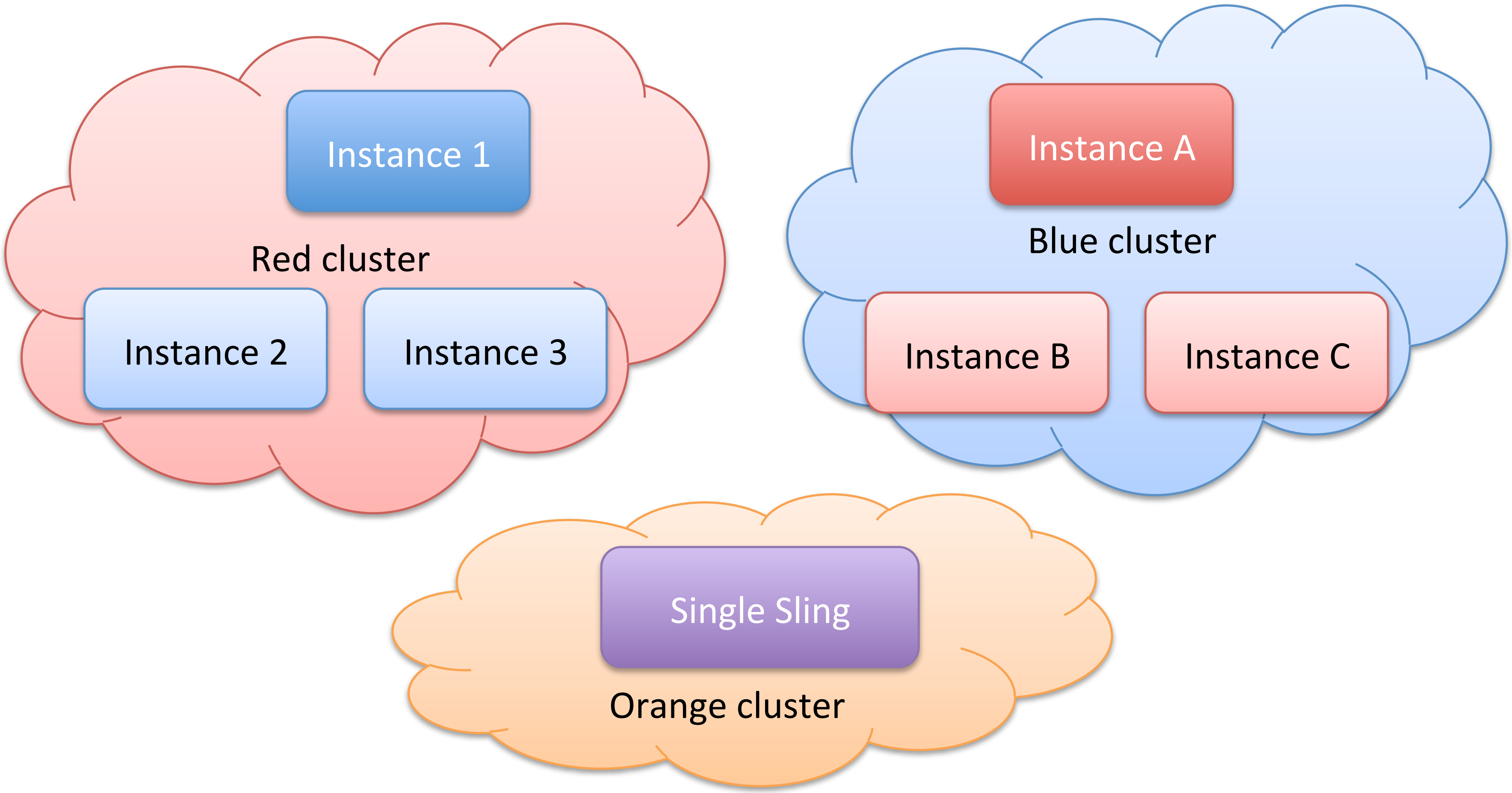
API overview & our JMS implementation

Discovery API

Sling Discovery API

- New Sling API
- Each instance can expose a list of key-value properties to other instances
- Example usage: metadata for workflow offloading in CQ
- Properties are not meant to be messaging tool, they shouldn’t change too often
- Instances are grouped into clusters, each cluster has elected leader

Topology


Instance 1
Red cluster
Instance 2
Instance 3
Instance A
Blue cluster
Instance B
Instance C
Single Sling
Orange cluster

Discovery OSGi services

- `DiscoveryService` provides `TopologyView`
- Custom instance properties can be added with `PropertyProvider` implementations
- Changes in topology can be observed with `TopologyEventListener`

JMS discovery implementation

- There is a default HTTP-based implementation
  - requires providing all instance URLs on each instance
- But JMS is a natural choice for the transport layer here
- `sling-jms-discovery` – a new, JMS-based discovery implementation that doesn’t require any configuration

Discovery election

- If some instance notices there is no leader for some cluster, it sends message: `WHO_IS_LEADER`
- If no one responds in 10 seconds, it sends the second message: `ELECTION`
- Every instance in a given cluster has to respond with their Sling instance id in the `VOTE` message
- After 5 seconds instance with the smallest id is chosen and it sends the `I_AM_LEADER` message

Discovery – demo

- http://local.cq:4503/bin/jms/discovery/info.txt

JMS discovery implementation details

Sling Message Consumer

adaptTo()

Sling Message Listener

Writing JMS consumers in OSGi is all about

```
@Component
public class MyComponent implements MessageListener {

   @Reference
   private JmsConnectionProvider connectionProvider;
   private Connection connection;
   private Session session;
   private MessageConsumer consumer;

   @Activate
   protected void activate() throws JMSException {
      connection = connectionProvider.getConnection();
      session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
      Destination dest = session.createTopic("my topic");
      consumer = session.createConsumer(dest);
      consumer.setMessageListener(this);
      connection.start();
   }

   @Deactivate
   protected void deactivate() throws JMSException {
      consumer.close();
      session.close();
      connection.close();
   }
}
```

Sling Message Listener

Why don't we

```
@SlingMessageConsumer(
    destinationType = DestinationType.TOPIC,
    subject = "my topic")
public class MyComponent implements MessageListener {

        public void onMessage(Message msg) {
// …
```

- `@SlingMessageConsumer` annotation is transformed into `@Service`, `@Component` and `@Properties` by our `sling-jms-scr` plugin
- `MessageListener` services are collected by our `MessageConsumerRegistry` OSGi component
- Registry component creates JMS consumers and forward messages to appropriate listeners

Sling Message Listener – filtering

In the message listener configuration you can add the `filter` property to filter out incoming messages. Use the LDAP format:

```
@SlingMessageConsumer(
   destinationType = DestinationType.TOPIC,
   subject = "my topic"
   filter = "(requestType=VOTE)")
```

Solution overview

Reverse Replication Request

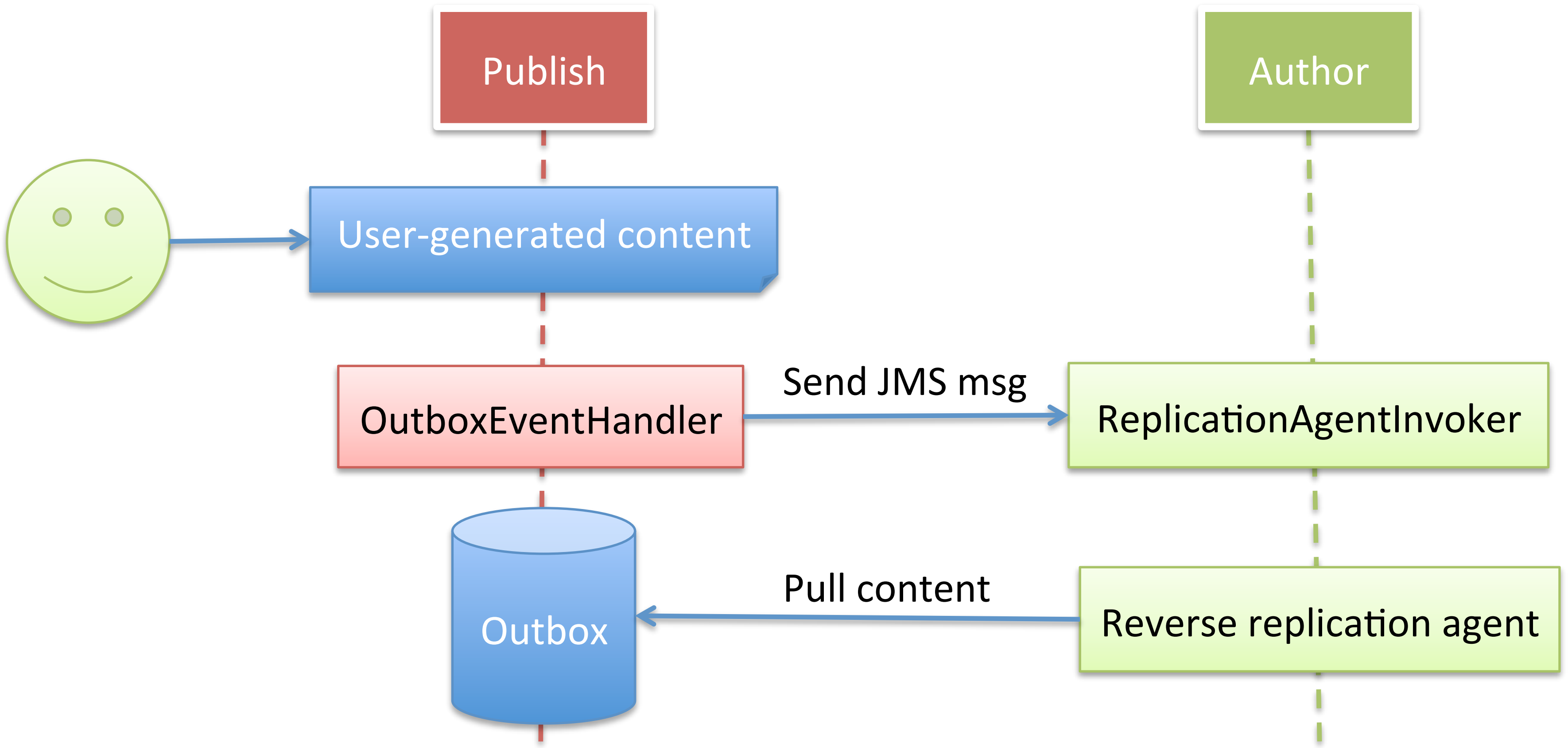
Reverse replication request

- In CQ, the Author gathers content from publish instances every 30 seconds (configurable in the `ReverseReplicator` service)
- Why can’t publish send user-generated content to the Author (push instead of pull)?
  - Security issues – publishes are often in DMZ
- But publish can *inform* the Author there is something new to pull

Reverse replication request

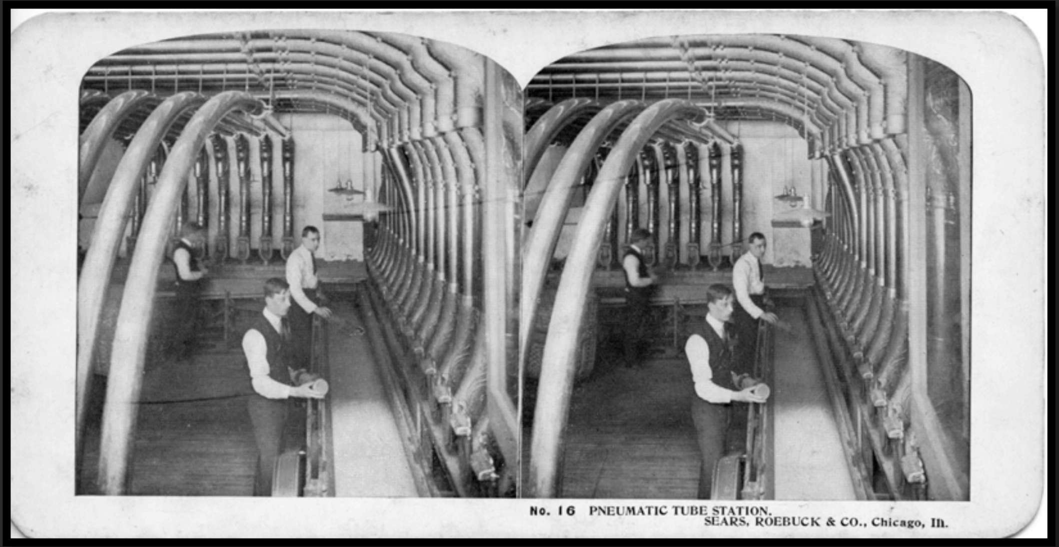
- `OutboxEventHandler` watches the outbox node and sends information every time there is some new content to pull
- `ReplicationAgentInvoker` receives this information and invokes the reverse replication process
- It’s a good idea to disable the out-of-the-box `ReverseReplicator` service, so we don’t have two reverse replications at the same moment

Reverse replication request flow



Reverse replication request implementation details

Targeting messages with the Sling run mode


No. 16 PNEUMATIC TUBE STATION.
SEARS, ROEBUCK & CO., Chicago, Ill.

Send messages only to the Author

- The outbox event handler should send messages only to the Author
- We can filter it manually…
- …or use `@SlingMessageConsumer` for the consumer and add `MessageConsumerProperties.DESTINATION_RUN_MODE` to the message properties
- Messages will be filtered automatically

Final remarks

Extra tools created during JMS research

- Embedded ActiveMQ broker configurable within the OSGi component
- Out-of-the-band JCR binaries transfer
  - You can send a message with metadata and some JCR path and the binary itself will be sent via HTTP
- Utilities to deal with serialization problems when sending `ObjectMessage`

Bundles overview

- sling-jms-activemq
  - ActiveMQ + dependencies
- sling-jms-api
  - JMS Connection Provider
  - Sling Message Consumer
  - Blob & Object message utils
- sling-jms-impl-activemq
  - Implementation of the API tools
- sling-jms-scr
  - Annotation processor for Sling Message Consumer
- sling-jms-sandbox
  - Example usage of the API tools
- Open-sourced use cases
  - sling-jms-discovery
  - cq-jms-replication
  - sling-jms-session

Try it yourself!

- https://github.com/Cognifide/PoC-Sling-JMS
  - All bundles
  - Some documentation
  - Temporary repo, will be transferred, so clone it while you can ;)

That’s all folks!