Deploying FuseMQ in enterprise using Fuse Fabric

Dejan Bosanac
FuseSource
Presenter: Dejan Bosanac

- Senior Software Engineer at FuseSource - [http://fusesource.com](http://fusesource.com)
- Apache ActiveMQ committer and PMC member
- Co-author of ActiveMQ in Action
- Blog:  
  - [http://www.nighttale.net/](http://www.nighttale.net/)
- Twitter:  
  - [http://twitter.com/dejanb](http://twitter.com/dejanb)
Agenda

- Problems of large enterprise deployments
- Fuse Fabric in nutshell
- FuseMQ and Fuse Fabric
  - Creating brokers
  - Connecting
  - Topologies
- Fuse Management Console
Problems of large deployments
Main problems

- Installing brokers on multiple hosts
  - ssh, untar, set directories and environment
- Setting configuration manually for every broker
  - copying xml config, tweaking, testing
- Updating configuration across cluster
- Upgrading brokers

It’s very tedious and error-prone process
Problems – Traditional best-practice tips

- Keep XML as a template and configure node-specific details through properties
- Keep configuration in SVC system (git, svn, ...)
- Keep configuration separate from installation for easier upgrades

Deployment with Fuse Fabric moves it to the next level
Problems - Clients

- Topology is very “static”
- Clients need to be aware of topology
- Clients need to know broker locations
- Changes are not easy as clients need to be updated
- Adding new resources (brokers) requires client updates
- Not suitable for “cloud” deployments

Fuse Fabric makes deployments more “elastic”
Fuse Fabric in a nutshell
Fuse Fabric in a nutshell

- How Fabric can help?
  - It provides centralized distributed broker configuration
  - It provides centralized distributed broker registry
  - Uses OSGi and Apache Karaf for easy spawning new broker instances
  - It provides additional tools for centralized configuration and monitoring (Fuse Management Console)
Fuse Fabric in a nutshell

- **Installation**
  - Features and bundle versions centrally stored and managed
  - Easy installation and upgrade

- **Configuration**
  - Stored in one place
  - Versioned

- **Discovery**
  - All brokers registered in central registry
  - Allows clients to connect without knowing broker locations
  - Allows easy creation of advanced topologies
Fuse Fabric Architecture

- Zookeeper

  - Replicated in-memory tree
  - Similar to file system
  - Highly-available
  - Distributed
  - Support network split
  - Proven track record

**Ideal for distributed configuration and locking**
Containers

- Apache Karaf instances provisioned through central registry (Zookeeper)
Fuse Fabric Architecture

Profiles:

- Zookeeper nodes with conventional names
- OSGi configuration for the node (so we know what features and bundles should be used)
- Other configuration (centralized broker configuration)
- Versioned
Fuse Fabric - Profile

FuseFabric:karaf@root> profile-display default
Profile id: default
Version : 1.0
Parents :
Associated Containers :

Container settings
--------------------
Repositories :
    mvn:org.fusesource.fabric/fuse-fabric/7.0-SNAPSHOT/xml/features

Features :
    fabric-agent
    karaf
    fabric-jaas
    fabric-core
Fuse Fabric - Profile

Agent Properties:

```
org.ops4j.pax.url.mvn.repositories =
http://repo1.maven.org/maven2,
http://repo.fusesource.com/nexus/content/repositories/releases,
http://repo.fusesource.com/nexus/content/groups/ea,
http://repository.springsource.com/maven/bundles/release,
http://repository.springsource.com/maven/bundles/external,
http://scala-tools.org/repo-releases

org.ops4j.pax.url.mvn.defaultRepositories =
file:${karaf.home}/${karaf.default.repository}@snapshots,
file:${karaf.home}/local-repo@snapshots
```

Configuration details
----------------------------

PID: org.fusesource.fabric.zookeeper
zookeeper.url ${zk:root/ip}:2181
FuseMQ and Fuse Fabric
FuseMQ features

- mq-base profile
  - Defines OSGi features and bundles to be installed
  - Defines basic broker settings

- mq-create command
  - Helper command for creating brokers
  - Creates a new profile based on mq-base
  - Optionally creates new containers
  - Assigns the profile to containers (essentially starts the broker)
MQ – Creating broker

FuseFabric:karaf@root> mq-create --create-container broker1 fusebroker
MQ profile fusebroker ready
Successfully created container broker1
MQ Profile

FuseFabric:karaf@root> profile-display fusebroker
Profile id: fusebroker
Version : 1.0
Parents  : mq-base
Associated Containers : broker1

Configuration details
-----------------------
PID: org.fusesource.mq.fabric.server-fusebroker
  standby.pool default
  connectors openwire
  broker-name fusebroker
  data data/fusebroker
  config zk:/fabric/configs/versions/1.0/profiles/mq-base/broker.xml
  group default
MQ – Assigning profile

FuseFabric:karaf@root> container-create-ssh --host 192.168.1.106
--user dejanb --password xxx broker1

FuseFabric:karaf@root> mq-create --assign-container broker1 fusebroker
MQ profile fusebroker ready
Profile successfully assigned to broker1
MQ - Benefits

What did we achieve with this?

- We can easily create new brokers with the same profiles
- We can create new profile version with updated broker version and/or changed configuration
- We can easily update all (or some) brokers by applying the new profile
MQ Profile - Management

- Create a new profile version
  - with upgraded bundles
  - and configuration changes
- Try it out on a non-production container
- Deploy to one or a few production containers
- Roll the full upgrade
- Easy rollback if anything goes wrong
Broker Registry
Broker Registry

- Brokers are organized in groups (clusters)
  - Cluster can have any number of brokers (with different names)
  - Put in “default” group if not specified
Connecting to the Broker

- Clients need to have ZooKeeper URL
- There is a new discovery protocol (called fabric)
- Connecting is as easy as defining the group
ActiveMQConnectionFactory factory =
new ActiveMQConnectionFactory("discovery:(fabric:default)");
Connecting - Reconnecting

- Clients don’t need to know brokers location
- Works like a failover transport
- Supports options for tuning reconnecting options

```
discovery:(fabric:default)?reconnectDelay=1000&useExponentialBackOff=false
```
<camelContext xmlns="http://camel.apache.org/schema/spring">
  <!-- Do your magic here -->
</camelContext>

<bean id="activemq" class="org.apache.activemq.camel.component.ActiveMQComponent">
  <property name="brokerURL" value="discovery:(fabric:discovery)"/>
</bean>
- Create master slave configuration by starting multiple brokers with the same name (in the same group)
  - First one stared becomes a master
  - Everyone else is a slave
  - Locked on Zookeeper node
  - When master dies, a first slave to get a lock becomes next master
FuseFabric:karaf@root> mq-create --create-container broker1 fusebroker

FuseFabric:karaf@root> mq-create --create-container broker2 fusebroker
Master/Slave

- No more relying on shared storage locking
- You’ll still need shared storage for preserving the state among brokers
- Easy creating non-persistent master slave configurations
- Clients again don’t need to know topology as fabric discovery will do that work
Master/Slave

- Multiple master slave over the same containers
  - Resource utilization

```
mq-create --create-container broker1,broker2,broker3 hq-broker
mq-create --assign-container broker1,broker2,broker3 web-broker
```
Networks

- Controlled through profile
- Uses fabric discovery, just as clients

mq-create --group us-east --networks us-west --create-container us-east1,us-east2 us-east

mq-create --group us-west --networks us-east --create-container us-west1,us-west2 us-west
Elastic clusters

- Request-reply pattern over JMS
- Load Balance Traffic
- Non-persistent, not-connected brokers
- Elastic cluster
  - Allow adding new brokers, without updating clients
  - Allow rebalancing of clients
Elastic clusters

```
mq-create --create-container broker1 broker1
mq-create --create-container broker2 broker2
mq-create --create-container broker3 broker3
```
Tooling
Fuse Management Console

- Centralized Unified Console
- Web UI for managing and monitoring infrastructure
- Uses Fabric to discover resources
- Features
  - Container Management
  - Profile Management
  - Centralized Security
  - Centralized Monitoring
# FMC – containers

## Containers

<table>
<thead>
<tr>
<th>Name</th>
<th>Active</th>
<th>Provisioned</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>broker1</td>
<td>✔️</td>
<td>✔️</td>
<td>1.0</td>
</tr>
<tr>
<td>root</td>
<td>✔️</td>
<td></td>
<td>1.0</td>
</tr>
</tbody>
</table>

### broker1

- **Type**: Managed Container
- **Profiles**: fusebroker
- **Location**
  - **Local IP**: 192.168.1.111
  - **Local Hostname**: debian-bosnacs-macbook-pro-2.local
  - **Public IP**: 
  - **Public Hostname**: 
  - **Manual IP**: 
  - **Resolver**: Local Hostname
  - **Provision Status**: Success
### Containers / broker1

<table>
<thead>
<tr>
<th>Profiles</th>
<th>Name:</th>
<th>broker1</th>
<th>Status:</th>
<th>online</th>
<th>Provision Status:</th>
<th>Success</th>
</tr>
</thead>
</table>

**Process ID:** 8939@deanj-bosanac-macbook-pro-2.local  
**JVM:** Java HotSpot(TM) 64-Bit Server VM (Apple Inc.)  
**CPU time:** 26 seconds  
**Up time:** 3 minutes  
**OS type:** Mac OS X 10.5.8  
**Architecture:** x86_64  
**CPU cores:** 2  
**load average:** 0.70

**CPU Usage:** 0.98%  
**Physical Memory:** 604.10 MB free  
**Heap Memory:** 119.76 MB used

**Threads:**  
45 running  
152 peak

**Swap:**  
2.00 GB free  
0 bytes total

**File Descriptors:**  
156 used  
10240 max

**Native Memory:**  
49.95 MB used  
50.19 MB alloc  
130.00 MB max
## FMC – broker view

### Containers / broker1 / Brokers / broker1 : Queues

<table>
<thead>
<tr>
<th>Queue Name</th>
<th>Memory Limit</th>
<th>Memory Usage</th>
<th>Producer Count</th>
<th>Consumer Count</th>
<th>Max Enqueue Time</th>
<th>Min Enqueue Time</th>
<th>Average Enqueue Time</th>
<th>Enqueue Count</th>
<th>Dequeue Count</th>
<th>Dispatch Count</th>
<th>Inflight Count</th>
<th>Max Page Size</th>
<th>Cursor Memory Usage</th>
<th>Cursor Percent Usage</th>
<th>Cursor Full</th>
<th>Does Cursor Have Space</th>
<th>Messages Buffered</th>
<th>Use Cache</th>
<th>Producer Flow Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>FABRIC DEMO</td>
<td>1.00 MB</td>
<td>0%</td>
<td>1</td>
<td>1</td>
<td>90 ms</td>
<td>1 ms</td>
<td>1 ms</td>
<td>281</td>
<td>282</td>
<td>281</td>
<td>0</td>
<td>200</td>
<td>0 bytes</td>
<td>0</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
</tr>
</tbody>
</table>
# FMC - Profiles

## Profiles

<table>
<thead>
<tr>
<th>Versions</th>
<th>Profiles</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Name</strong></td>
<td><strong>Containers</strong></td>
</tr>
<tr>
<td>1.0</td>
<td>2</td>
</tr>
</tbody>
</table>

## Profiles

<table>
<thead>
<tr>
<th>Name</th>
<th>Containers</th>
</tr>
</thead>
<tbody>
<tr>
<td>aws-eu2</td>
<td>0</td>
</tr>
<tr>
<td>camel</td>
<td>0</td>
</tr>
<tr>
<td>cloud</td>
<td>0</td>
</tr>
<tr>
<td>cloudservers-uk</td>
<td>0</td>
</tr>
<tr>
<td>cloudservers-us</td>
<td>0</td>
</tr>
<tr>
<td>cxf</td>
<td>0</td>
</tr>
<tr>
<td>default</td>
<td>0</td>
</tr>
<tr>
<td>dosgi</td>
<td>0</td>
</tr>
<tr>
<td>esb</td>
<td>0</td>
</tr>
</tbody>
</table>
Profiles / mq-base

Version: 1.0
Parent Profiles: karaf

Features | Fuse Application Bundles (0) | Bundles (0) | Repositories (0) | Config Properties (0) | System Properties (0) | Config Files (4) |
--- | --- | --- | --- | --- | --- | --- |
org.fusesource.insight.graph.json
org.fusesource.mq.fabric.template.properties
org.fusesource.fabric.agent.properties
broker.xml

Add new config file (example: com.foo.myservice.properties):
Future

- More things for developers
  - Make it even easier to write applications for Fuse Enterprise

- More things for operations
  - Visualization of clusters
  - Centralized logging (collect and search all logs centrally)
Conclusion

- Helps with complex and large deployments
- Use central registry for distributed configuration and locking
- Make clients location agnostic of brokers (needed for cloud deployments)
- Easy upgrades and updates
- Support for incremental patching
- Tools
Questions