Deploying Apache ActiveMQ for Reliability and Scalability

Rob Davies
CTO
FuseSource
Presenter: Rob Davies

- CTO at FuseSource - “the experts in open source integration and messaging”  
  [http://fusesource.com](http://fusesource.com)
- Apache ActiveMQ, ServiceMix and Camel PMC member
- Co-creator of ActiveMQ, ServiceMix and Camel
- Co-author of ActiveMQ in Action:
  - Chapter 5: ActiveMQ Message Store
  - Chapter 10: Deploying ActiveMQ in the Enterprise
  - Chapter 11: ActiveMQ Broker Features In Action
  - Chapter 12: Advanced Client Options
  - Chapter 13: Tuning ActiveMQ for Performance
- Twitter:  [http://twitter.com/rajdavies](http://twitter.com/rajdavies)
Apache ActiveMQ – Enterprise Features

- Failover
- High Availability
- Clustering
- Scaling
Failover

- Java and C++ clients support seamless failover
Apache ActiveMQ – Synchronous sends

ActiveMQ Producer

ActiveMQ Broker

ActiveMQ Consumer

ActiveMQ Consumer

ActiveMQ Consumer
Apache ActiveMQ – Synchronous sends

Synchronous send: JMS Client

Set `alwaysSyncSend` on the `ActiveMQConnectionFactory`

You can set:
`sendTimeout` on the `ActiveMQMessageProducer`

**BTW** -

In ActiveMQ 5.6 – you can get a callback – e.g.:

```java
producer.send(session.createTextMessage("Hello"), new AsyncCallback() {
    public void onSuccess() {}

    public void onException(JMSException exception) {
        exception.printStackTrace();
    }
});
```
Apache ActiveMQ – Enterprise Features

JMS async send can be made reliable

- failover:// transport (default) detects network outages
- failover:// can replay messages by enabling `trackMessages`
  
  ```
  failover:(tcp://hostA:61617,tcp://hostB:61617)?trackMessages=true
  ```
ActiveMQ High Availability
Three types of Master/Slave.

- Fully replicated Master/Slave
- JDBC Master/Slave
- Shared File System Master/Slave
Apache ActiveMQ – High Availability

**Fully Replicated Master/Slave**

- Shared nothing
- Fully replicated
  - All messages
  - All acknowledgements
  - All transactions
- Slave does not start any transports or network connections
Apache ActiveMQ – High Availability

Fully Replicated Master/Slave

<broker masterConnectorURI="tcp://masterhost:62001" shutdownOnMasterFailure="false">

failover://(tcp://masterhost:61616,tcp://slavehost:61616)?randomize=false
**Fully Replicated Master/Slave - Limitations**

- Only one slave can be connected to a Master
- No automatic failback
- No automatic synchronization after master re-start

**Fully Replicated Master/Slave - Recovery**

- *Shutdown the slave broker (clients don’t need to be)*
- *Copy the message database to the master*
- *Re-start the master and the slave*
JDBC Master/Slave

- Extreme reliability – but not as fast
- Recommended if already using an enterprise database
- No restriction on number of slaves
- Simple configuration
- Configurable lockKeepAlivePeriod
Apache ActiveMQ – High Availability

Shared File system
Master/Slave

ActiveMQ Slave

ActiveMQ Master

ActiveMQ Slave

Larry’s
Removals

ActiveMQ Client

File System
Shared File Master/Slave

- Recommended if you have a SAN, or DRDB or NFS
- No restriction on number of slaves
- Simple configuration
- N.B. – ensure file locking works – and times out – NFSv4 good!
- On KahaDB the lock is `databaseLockedWaitDelay`

`failover://(tcp://host1:61616,tcp://host2:61616)`
ActiveMQ Broker Topologies
Networks

- Link ActiveMQ Brokers together
- Use Store and Forward
- Are uni-directional by default
- All Destinations are global
Store and Forward
**Apache ActiveMQ – Broker Topologies**

*Bi-directional network*

[Diagram showing a bi-directional network with two ActiveMQ brokers: one local and one remote, connected by a firewall.]
Bi-directional network - Configuration

```xml
<networkConnectors>
  <networkConnector uri="static://(tcp://backoffice:61617)"
    name="bridge"
    duplex="true"
    conduitSubscriptions="true"
    decreaseNetworkConsumerPriority="false">
  </networkConnector>
</networkConnectors>
```
Combining HA and Networks
Apache ActiveMQ – Broker Topologies

Networks – Configuration for master/slave (from 5.6)

```
<networkConnectors>
  <networkConnector uri="masterslave:(tcp://master,tcp://slave)"/>
</networkConnectors>

Which is the same as:

```
<networkConnectors>
  <networkConnector uri="static:failover:(tcp://master,tcp://slave)?randomize=false&maxReconnectAttempts=0"/>
</networkConnectors>
```
Networks – Configuration – Filters: dynamicallyIncludedDestinations

```xml
<networkConnectors>
  <networkConnector uri="static:(tcp://remote:61617) ">
    <dynamicallyIncludedDestinations>
      <queue physicalName="free.food.">
      <queue physicalName="free.beer.">
      <topic physicalName="cricket.scores.">
    </excludedDestinations>
  </networkConnector>
</networkConnectors>
```
Networks – Configuration – Filters: \textit{staticallyIncludedDestinations}

\[
<\text{networkConnectors}>
\begin{align*}
<\text{networkConnector uri}="\text{static:(tcp://remote:61617)}?\text{useExponentialBackOff=false}"/>& \\
<\text{staticallyIncludedDestinations}> \\
<\text{queue physicalName}="\text{management.queue-1}"/>& \\
<\text{queue physicalName}="\text{management.queue-2}"/>& \\
<\text{queue physicalName}="\text{global.}"/>& \\
<\text{topic physicalName}="\text{global.}"/>& \\
</\text{staticallyIncludedDestinations}>& \\
</\text{networkConnectors}>
\]
Networks – Configuration – networkTTL=2
ActiveMQ – geographically dispersed data centers: redundant links – Topic support

Enable duplicate subscriptions over the network:

```xml
<networkConnectors>
  <networkConnector uri="static:(tcp://brokerB:61617)" name="A-B" networkTTL="2" suppressDuplicateTopicSubscriptions="false">
  </networkConnector>
  <networkConnector uri="static:(tcp://brokerC:61618)" name="A-C" networkTTL="2" suppressDuplicateTopicSubscriptions="false">
  </networkConnector>
</networkConnectors>

Ensure every Topic message is only sent through one network connection - the one with the highest priority:

```xml
<destinationPolicy>
  <policyMap>
    <policyEntries>
      <policyEntry topic=""></policyEntry>
  </policyEntries>
</policyMap>
</destinationPolicy>
```
ActiveMQ Scaling
ActiveMQ – Vertical Scaling
Apache ActiveMQ – Vertical Scaling

Reduce the number of Broker Threads

ACTIVEMQ_OPTS=

"-Xmx1024M -Dorg.apache.activemq.UseDedicatedTaskRunner=false"

Reduce thread usage by Destinations

<destinationPolicy>
  <policyMap>
    <policyEntries>
      <policyEntry queue=""> optimizeDispatch="true" />
    </policyEntries>
  </policyMap>
</destinationPolicy>
Vertical Scaling: Other things to improve Vertical Scaling

- Use the nio transport on the ActiveMQ broker
- Increase the amount of memory available to the broker
- Reduce the default JVM stack size of each thread by use of -Xss option
- Use call backs for synchronous sends
- Use LevelDB!
ActiveMQ – Horizontal Scaling
**Horizontal Scaling:** Increase load capacity using many brokers

- Use ActiveMQ Networks
- Messages are forwarded between brokers with interested consumers
- Networks lift the limits of using a single machine

**Problems:**

- Complex topologies can lead to non-optimal message routing
- Orphaned Messages on failure (use networks and clusters)
Horizontal Scaling – client-side partitioning

Hybrid of Vertical and Horizontal Scaling
• Multiple broker nodes are used by the clients
• Brokers are NOT networked
• The client application send message to different brokers, typically based on some defined partitioning of the data.

Pros
• You can use all the tuning techniques used in Vertical scaling
• Have better Horizontal scalability than using Network Of Brokers (Less broker cross talk)

Cons
• Added complexity required on the end user Application
FuseSource Knows How To Build Enterprise Apps

Help throughout the software development lifecycle...

- **Get started**
  - training videos
  - webinars
  - tutorials
  - documentation
  - white papers

- **Try it out**
  - tech overviews
  - training
  - project planning
  - pilot workshop
  - pilot subscription

- **Build right**
  - dev. subscription
  - arch. workshop
  - best practices
  - QoS development
  - training

- **Deploy safely**
  - prof. subscription
  - health check
  - perf. workshop
  - HA workshop
  - training

Available for Free

FuseSource Paid Engagement
Embedding Apache Camel
ActiveMQ with embedded Camel

- Co-location: fail-fast
- Advanced routing in the Broker – reduce latency and improve performance
ActiveMQ with embedded Camel: import camel into ActiveMQ broker config:

```xml
<beans>
  <broker brokerName="testBroker" xmlns="http://activemq.apache.org/schema/core">
    <transportConnectors>
      <transportConnector uri="tcp://localhost:61616"/>
    </transportConnectors>
  </broker>
  <import resource="camel.xml"/>
</beans>
```
ActiveMQ with embedded Camel: Setup Camel Context in usual way

```xml
<camelContext errorHandlerRef="errorHandler" xmlns="http://camel.apache.org/schema/spring">
  <route>
    <from uri="activemq:queue:test.queue"/>
    <choice>
      <when>
        <xpath>$foo = 'bar'</xpath>
        <to uri="activemq:topic:topic.bar"/>
      </when>
      <when>
        <xpath>$foo = 'cheese'</xpath>
        <to uri="activemq:topic:topic.cheese"/>
      </when>
      <otherwise>
        <to uri="activemq:topic:topic.all"/>
      </otherwise>
    </choice>
  </route>
</camelContext>
```
Introducing FuseMQ Enterprise
Why Fuse Fabric?

- Configuration of Apache ServiceMix and Apache ActiveMQ is complex
- Enterprise deployments of our software requires a lot of upfront knowledge, and it's easy to get wrong
- Enterprises need to deploy across different environments, on-premise, on a private cloud, on a public cloud and all of the above
- Enterprise deployments need location transparency, and support of failover of endpoints
Fuse Fabric – Key Features

- Support Hybrid deployments – on premise, on cloud, on both
  - Endpoints can be relocated
  - Endpoints can be load balanced
  - Endpoints can be elastic
  - Endpoints can be highly available

- Distributed Configuration
  - Configuration has to be accessed across multiple domains
  - Configuration has to be highly available

- Runtime Registry – allows discovery of:
  - services
  - Endpoints
  - FuseMQ message brokers

- Distributed Management
  - Easy elastic scaling of services
  - Monitoring and Control of resources
FuseMQ Enterprise Use cases
Ingestion for BigData Architecture:

Web Servers

Camel HTTP Log Reader

FuseMQ Cluster

Fuse ESB

Camel Hbase/HDFS

HDFS

Web Servers

Camel HTTP Log Reader

FuseMQ Cluster

Fuse ESB

Camel Hbase/HDFS

HDFS

Web Servers

Camel HTTP Log Reader

FuseMQ Cluster

Fuse ESB

Camel Hbase/HDFS

HDFS

Web Servers

Camel HTTP Log Reader

FuseMQ Cluster

Fuse ESB

Camel Hbase/HDFS

HDFS
Example of Distributed Application Integration

FuseMQ Clusters

Fuse ESB

Head Office

FuseSource
integration everywhere
**M2M Deployments**

**MQTT**

- MQTT support in Fuse Message Broker 5.6 and FuseMQ Enterprise 7.1
- MQTT protocol is extremely lightweight – with many M2M clients
- Using Fuse Messaging products – seamless integration between MQTT, Stomp, OpenWire (JMS, C++/C) and more (AMQP in the future).

Machine-to-Machine (M2M) solutions such as industrial control, smart buildings, asset tracking, traffic control and healthcare monitoring, are an essential and integral part of nearly all industry, enterprise and daily life. Inherent to M2M is the need to connect objects in the physical world, via sensors, actuators and other devices, into monitoring, control, business, and consumer software systems, often over constrained wireless networks.
Deploying FuseMQ in enterprise using Fuse Fabric

Dejan Bosanac
FuseSource
Presenter: Dejan Bosanac

- Senior Software Engineer at FuseSource - http://fusesource.com
- Apache ActiveMQ committer and PMC member
- Co-author of ActiveMQ in Action
- Blog:
  - http://www.nighttale.net/
- Twitter:
  - 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*
Fuse Fabric Architecture

- Containers
  - Apache Karaf instances provisioned through central registry (Zookeeper)
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:

```java
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
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
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>
Topologies
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
Master/Slave

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

```bash
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
### 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**: 192.168.1.111
- **Local Hostname**: debian-bosilac-macbook-pro-2.local
- **Public Hostname**: 
- **Manual IP**: 
- **Resolver**: Local Hostname
- **Provision Status**: Success
Containers / broker1

Profiles

Name: broker1  Status: online  Provision Status: Success

Process ID: 8939@dejan-bosanac-macbook-pro: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
**Containers / broker1 / Brokers / broker1 : Queues**

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

#### Profiles

<table>
<thead>
<tr>
<th>Name</th>
<th>Containers</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>2</td>
<td>✓</td>
</tr>
</tbody>
</table>

#### Profiles

<table>
<thead>
<tr>
<th>Name</th>
<th>Containers</th>
</tr>
</thead>
<tbody>
<tr>
<td>aws-ec2</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>cf</td>
<td>0</td>
</tr>
<tr>
<td>default</td>
<td>0</td>
</tr>
<tr>
<td>docsi</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 [1]  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):  Add
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