A4M36JEE

Clustering & Scalability

Václav Chalupa
Quality Engineer
Red Hat

Radoslav Husár
Software Engineer
Red Hat

November 21, 2014
Agenda

• Clusters
  • HA
  • Load-balancing
  • Scalability
• JGroups
• Infinispan
• Clustering in WildFly 8
• mod_cluster
Cluster in General

“A computer cluster consists of a set of loosely connected computers that work together so that in many respects they can be viewed as a single system.”

Wikipedia
Motivation

• Interconnected
• But independent
• Made possible with:
  • high-speed networking
  • cheap commodity hardware
• Improve performance and/or availability
• Scale to handle higher load
Lets Define “Our” Cluster for Today

A cluster is a collection of WildFly 8 servers that communicate with each other so as to improve the availability of services by providing the following capabilities:

- High Availability
- Scalability
- Fail-over
- Fault Tolerance
High Availability / HA

Capability to support server applications that can be reliably utilized with a minimum down-time.
Scalability

Capability to handle a large number of requests without service response time degradation.
Fail-over

Capability for a cluster node to take over the tasks or requests processed by a failing node.
Fault Tolerance

Guarantee of correct behavior in the event of a failure.
So, why do we need Cluster?

Potential problems with deploying critical applications on a single node:

• ?
• ?
What does Java EE say about clustering?

- Err, not much.
WildFly Clustering Areas

- Web session replication
- Stateful Session Bean (SFSB) replication
- Entity bean replication (2nd level caching)
- SingletonService
- mod_cluster auto-configuration
- HornetQ (JMS) clustering
  - not covered here today
Making Deployments Clustered

• Distributed web sessions
  • Add `<distributable/>` tag to `web.xml`
  • Uses ‘web’ cache container, by default

• Clustered Stateful Session Beans
  • Annotate `@Stateful` (WildFly 8.0.0.Final)
  • Automatically clustered unless: `passivationCapable=false`
  • Uses ‘ejb’ cache container, by default
Notice about EJBs clustering

@Clustered annotation needed previously

- No more needed for WildFly
- EJBs clustered automatically
- You can disable clustering of SFSB by:
  @Stateful(passivationCapable=false)
  - From EJB 3.2
Distributable Sessions

• All session attributes must be serializable:
  • Must implement `java.io.Serializable`
  • Most native Java objects implement this functionality

• After updating any immutable objects stored in session:
  • `HttpSession.setAttribute()` must be called to inform the session replication that the session has changed

Ideally, sessions should be kept relatively small

• Less network traffic between the each clustered VM
Distributable Sessions – Immutable objects

- Known immutable values:
  - Null, java.util.Collections.EMPTY_LIST/EMPTY_MAP/EMPTY_SET
- The value type is or implements immutable type:
  - Boolean, Byte, Character, Double, Float, Integer, Long, Short
  - java.lang.Enum, StackTraceElement, String
  - ...
- The value type is annotated with:
  - @org.wildfly.clustering.web.annotation.Immutable
Application Must be Cluster-Aware

- Don't spawn custom services that should be singleton in the cluster:
  - Timers, whatnot
  - Locking becomes complex

- Don't store data as flat files
  - Store over NAS/SAN (NFS)
  - Use DB
  - Use data grid
EE @Singleton

Not cluster-wide singleton!

- @Singleton per JVM as spec dictates
- @Clustered @Singleton could be cluster-wide singleton (not yet)
SingletonService (HA Singleton)

Create singleton service in ServiceActivator (MSC)

- SingletonService is started only on one node
  - `start(StartContext): org.jboss.msc.service.Service`

Example:

https://github.com/wildfly/quickstart/tree/master/cluster-ha-singleton
Clustered 2LC

- JPA/Hibernate 2nd level cache
  - Infinispan is default 2nd level cache provider
- `persistence.xml` no longer needs to define `hibernate.cache.region.factory_class`
  - Uses “hibernate” cache container, by default
  - Non-clustering profiles use local-cache
- Provides eviction & expiration support
  - “ha” profiles use clustered caches
- Invalidation-cache for entities/collections
Operational Modes

- Clustering is orthogonal to
  - Standalone mode or
  - Domain mode

- Clustering in domain “easier” to manage
- More on next lecture on management!
Changes from AS 4/5/~6

- All clustering services start on demand and stop when no longer needed
- Lifecycle example:
  - Deploy app1, starts channel and cache
  - Deploy app2
  - Undeploy app1
  - Undeploy app2, stops cache and channel

- Starting a server with no deployments will not start any channels/caches
Changes from AS 4/5/~6

- Infinispan replaced JBoss Cache as clustering toolkit and session cache
- Configuration is now centralized.
- No more farm deployment.
  - Domains and server groups provide this functionality.
- No out-of-box HA Singleton deployer.
  - Deploy application backend to only one node
- No HA JNDI (replaced with client JNDI).
Extensions for Clustering in WildFly 8

org.jboss.as.clustering.jgroups
• Provides the communication between cluster nodes

org.jboss.as.clustering.infinispan
• Provides the replicated caching functionality

org.jboss.as.mod_cluster
• Provides integration and configuration with mod_cluster software load balancer
Predefined Profiles

• Standalone mode
  • standalone-ha.xml
  • standalone-full-ha.xml

$ bin/standalone.sh -c standalone-ha.xml
Predefined Profiles

- Domain mode
  - ha profile
  - full-ha profile

Use “ha” profile from domain.xml:

```xml
<server-group name="clustered-group" profile="ha">
  <socket-binding-group ref="ha-sockets"/>
</server-group>
```

- $ bin/domain.sh
JGroups
What is not reliable?

Messages get:

- **Lost and dropped**
  - Too big (UDP has a size limit), no fragmentation
  - Buffer overflow at the receiver, switch (NIC, IP network buffer)
- **Delivered in different order**

- We don't know the members of the cluster (multicast)
  - No notification when new node joins, leaves, or crashes
- Faster sender might overload slower receiver
  - Flow control absence
So what Is JGroups ?

**Toolkit for reliable cluster communication**

- Provides:
  - Fragmentation
  - Message retransmission
  - Flow control, Ordering
  - Group membership, membership change notification

- **LAN or WAN based**
  - IP multicasting transport default for LAN
  - TCP transport default for WAN
Architecture of JGroups

send

state

compress

flow

NAKACK

FRAG

UDP

receive

Channel

Network

send

state

compress

flow

NAKACK

FRAG

UDP

receive

Channel
A Message

- **src, dest: Address**
  - Address: identity of a member (of the cluster)
  - src: filled in when sending (by JGroups)
  - dest: null == send to all members of the group

- **buffer: byte[]**

- **headers: hashmap of headers**
  - each protocol can add/remove its own headers
  - example: sequence number for reliable retransmission

- **Message travels across the network**
Address

• A cluster consists of members
• Each member has its own address
• The address uniquely identifies one member

• Address is an abstract class
  • Implemented as a UUID
  • UUID is mapped to a physical address
• An address can have a logical name
  • For instance 'a'
  • If not set, JGroups picks the name, e.g. „host-16524“
View

- List of members (Addresses)
- Is the same in all members:
  - A: \{A,B,C\}
  - B: \{A,B,C\}
  - C: \{A,B,C\}

- Updated when members join or leave
- All members receive all views in the same order
- `Channel.getView()` returns the current view
API

- **Channel**: similar to `java.net.MulticastSocket`
  - But with built-in group membership, reliability

- **Operations**:  
  - Create a channel with a configuration (program. or xml)  
  - Connect to a group named 'x'
    - Everyone that connects to "x" will see each other  
  - Send a message to all members of 'x'  
  - Send a message to a single member  
  - Receive a message  
  - Be notified when members join, leave (crashes included)  
  - Disconnect from the group  
  - Close the channel
API (Code)

JChannel ch = new JChannel("udp.xml");
ch.setReceiver(new ReceiverAdapter() {
    @Override public void receive(Message msg) {
        System.out.println("msg from " + msg.getSrc() + ": " + msg.getObject());
    }
    @Override public void viewAccepted(View new_view) {
        System.out.println("new view: " + new_view);
    }
});
ch.connect("demo-group");
System.out.println("members are: " + ch.getView().getMembers());
Message msg = new Message(null, null, "Hello world");
ch.send(msg);
ch.close();
State transfer

State is data shared by all nodes in a cluster

- Stock quotes
- HTTP web sessions
- Messages received in the same order will update the state consistently across a cluster
- To add state transfer to an application, it has to
  - Add STATE_TRANSFER to the config
  - Implement the state transfer callbacks
- A new joiner needs to acquire state
Group Topology
Protocols (1)

- Transport
  - UPD (IP Multicast), TCP, TCP_NIE, LOOPBACK
- Member discovery
  - PING, TCPPING, TCGOSSIP, MPING
- Failure detection (freeze up, crash)
  - FD, FD_SOCK, VERIFY_SUSPECT, MERGE
- Reliable transmission and Ordering
  - Sequence numbers, lost messages are retransmitted
- Distributed Garbage Collection
  - Agreement on all received messages
Protocols (2)

- Group Membership
  - GMS
  - New view on membership change
- Flow control
  - FC
  - Fast sender does not overwhelm slow ones
- Fragmentation
  - FRAG, FRAG2
  - Big messages are transmitted as smaller ones
Protocols (3)

- State Transfer
  - STATE_TRANSFER
  - New member receives the state of the group
- Security
  - ENCRYPT, AUTH
- Debugging
  - PERF, TRACE, STATS
- Simulation and testing
  - DELAY, SHUFFLE, LOSS, PARTITIONER
JGroups Ergonomics

- Idea: observe the environment and adjust stack configuration dynamically
  - One configuration doesn't rule them all
  - Scale from small to large clusters
  - Shift from private to public cloud providers
  - Account for traffic patterns
- WIP
- You can contribute if you like networks.
CACHE STORE / PERSISTENCE

- Store data from memory to other kind of storage
  - File System, Relational Database, Other NoSQL stores
- Passivation support (spillover to disk)
PASSIVATION IN WILDFLY

<max-active-sessions>
  1000
</max-active-sessions>

- Disabled by default
- Controls maximum number of sessions to keep in memory, rest will be passivated.
EVICION and PERSISTENCE in AS

- Handle too many active sessions
- Passivation - eviction from memory to disk
- A way to be nice to users (keep sessions for longer time) and not crash the AS (with OOMs)
- Possibly handle restarts/upgrades
Cache Modes
Local mode

- Single node
- Non-clustered environment
  - Unaware of other instances on network
- Why use LOCAL cache in AS?
Replication mode

- Each node contains all the entries

Advantages
- N node cluster tolerates N-1 failures
- Read friendly – we don't need to fetch data from owner node
- Instant scale-in, no state transfer on leave

Disadvantages
- Write unfriendly, put broadcast to every node
- Doesn't scale well
- When node joins all state has to be transferred to new node
- Heap size stays the same when we add nodes
Replication mode

```java
cache.put(K, V)
```

![Diagram of cache servers](image)
Distribution mode

- Advantages
  - Scales – number of replications is independent of cluster size, depends only on number of owners
  - Number of owners set to compromise between failure tolerance and performance
  - Virtual heap size = numNodes * heapSize / numOwners

- Disadvantages
  - Not every node is an owner of the key, GET may require network hops
  - Node join and leave requires state transfer (rehash)
DISTRIBUTION

cache.put(K, V)

Cache on Server1

Cache on Server2

Cache on Server3

Cache on Server4

cache.get(K)
Invalidation mode

- Suitable for RDBMS off-loading, used with shared cache store
- Entry exists in node's local cache => it's valid and can be returned to requestor
- Entry doesn't exist in node's local cache => it's retrieved from the persistent store
- If a node modifies/removes entry it's invalidated in other nodes
- Low internode message traffic, PUT sends only invalidation messages and they are small.
INVALIDATION

cache.put(K, V_new)

Cache on Server1

K, V_new

Cache on Server2

K, V_old

Cache on Server3

Cache on Server4
Sync vs Async mode

- Synchronous
  - All operations get confirmation that the other relevant cluster nodes reached the desired state

- Asynchronous
  - All operations block only until they perform local changes, we don't wait for JGroups responses.
  - Better throughput but no guarantees on data integrity in cluster.
Using Infinispan from AS

• Customizing Infinispan Caches

• Eager vs. lazy startup mode
  • `<replicated-cache ... start="LAZY|EAGER">`

• JNDI binding
  • `<cache-container ... jndi-name="...">`
  • Assumes java:global namespace if unqualified
Using Directly

- On-demand injection of cache container

```java
@ManagedBean
public class CustomBean<K, V> {
    @Resource(lookup = "java:jboss/infinispan/customcontainer")
    private org.infinispan.manager.CacheContainer container;
    private org.infinispan.Cache<K, V> cache;

    @PostConstruct
    public void start() {
        this.cache = this.container.getCache();
    }
}
```
Load-balancers & mod_cluster
What is mod_cluster?

- Set of modules for Apache HTTPd and Tomcat-based web servers
  - requires Apache HTTPd 2.2.8+
  - requires JBoss AS 5.0+ or Tomcat 6+

- Similar to mod_jk and mod_proxy enables HTTPd to be a load-balancer in front of Java web servers

- JBoss.org LGPL project
Architecture #1

- Client requests proxied to back-end server
  - AJP, HTTP, HTTPS protocols
  - transparent to request handling on Java side
- **Key difference:**
  back channel from back-end to the front-end
  - Life-cycle information
  - Load-balancing information
  - Uses HTTP/HTTPS
Architecture #2
Overview of Key Benefits

- Simplified configuration
  - Dynamic configuration instead of static
  - HTTPd need not be preconfigured with cluster topology
  - Little configuration on the HTTPd and web server side

- Improved load-balancing
  - Load calculation done on the server side where more information is available

- Fine grained life-cycle control
  - Undeploy a running web app without 404s
Dynamic Configuration

- Backend web servers register with HTTPd at startup
- Backend web server register applications' as they are available
- No more static topology configuration on the HTTPd
  - No workers.properties
  - No uriworkermap.properties
- Auto-discovery
  - HTTPd servers advertise themselves for web servers to register with them using UDP multicast
  - No topology information
No more `worker.properties` & `uriworkermap.properties`

```properties
worker.list=lb
worker.lb.type=lb
worker.lb.balance_worker=node1,node2

worker.node1.type=ajp13
worker.node1.host=192.168.2.1
worker.node1.port=8009
worker.node1.lbfactor=1

worker.node2.type=ajp13
worker.node2.host=192.168.2.2
worker.node2.port=8009
worker.node2.lbfactor=1

/webapp/*=loadbalancer
/newwebapp/*=loadbalancer
```
Better Load-balancing

- **Problem**: load-balancer lacks information needed to make optimal load-balancing decision
  - Knows of: number of requests, sessions, sent/received bytes, response times
  - Ignores: backend server metrics, i.e. CPU usage, available memory, DB connection pool
  - Ignores: activity of other load-balancers

- **Solution**: backend web servers inform balancer how much load they can handle
  - Factor is a number between 1 to 100
  - Relative factors are used to make decisions
  - Backend servers have configured set of metrics
Load Metrics

• Metric tracked by the backend server to help make decision
  • e.g. available memory, CPU usage
• Multiple readings are combined to overall load factor
  • Older readings decline in importance/weight
• Highly configurable
  • Weights can be assigned to metrics, e.g. 50% CPU usage and 50% connection pool usage
  • Pluggable custom classes for metrics
List of Load Metrics

- Web tier usage:
  - active sessions, busy connections, bytes send and received, request count
- System utilization
  - CPU utilization, system memory usage, JVM heap usage, number of threads
- JCA connection pool usage
- Custom – build your own
Rolling Upgrades

- **Problem:** How to roll an upgrade without downtime?
  - Most downtime caused by upgrades, not crashes.
  - New release might be binary incompatible and cannot re-join the cluster.
    - Application and session incompatibilities
    - Major JBoss AS version upgrades (6.0 to 7.1)
    - Component upgrades (Infinispan)
    - DB Schema upgrades
  - General problem with large flat clusters.
    - State transfers, merges, scalability
Rolling Upgrades

- Solution: mod_cluster load balancing groups (mod_jk's domains)
  - 20 node cluster == 2 load balancing groups of 10 nodes, each LB group is a cluster
  - Session is replicated to all nodes within the LB group
  - In case of crash, fail-over happens within the LB group only
  - If there are no alive servers in LB group the session is lost forever and ever
Rolling Upgrades

- Upgrade entire domain at once.
  - Disable all contexts in the domain (mod_cluster manager)
  - No new sessions are created on disabled nodes.
  - Existing sessions are still directed to its' nodes.
  - Drain all sessions – all sessions expired in the domain.
  - Shutdown and perform an upgrade.
  - Start the group (enabled).
Installation HTTPd

- HTTPd modules and Java side:
  
  http://www.jboss.org/mod_cluster/downloads/

- Supported platforms
  
  - Linux x86, x64, ia64
  - Solaris x86, SPARC
  - Windows x86, x64, ia64
  - HP-UX PA-RISC, ia64
  - build your own from sources

- Distributes will full distribution or just use the modules

- Straightforward migration
HTTPd Configuration

LoadModule proxy_module modules/mod_proxy.so
LoadModule proxy_ajp_module modules/mod_proxy_ajp.so
LoadModule cluster_slotmem_module modules/mod_cluster_slotmem.so
LoadModule manager_module modules/mod_manager.so
LoadModule proxy_cluster_module modules/mod_proxy_cluster.so
LoadModule advertise_module modules/mod_advertise.so

<IfModule manager_module>
  #Listen 127.0.1.1:6666
  Listen *:6666
  ManagerBalancerName mycluster

  <VirtualHost *:6666>
    KeepAliveTimeout 300
    MaxKeepAliveRequests 0
    AdvertiseFrequency 5
    ServerAdvertise On
    EnableMCPMReceive On
    AllowDisplay On
    <Location />
      Order deny,allow
      Allow from 127.0.1
    </Location>
    <Location /mod_cluster_manager>
      SetHandler mod_cluster-manager
      Order deny,allow
      #Deny from all
      #Allow from 127.0.1
      Allow from all
    </Location>
  </VirtualHost>
</IfModule>
WildFly Configuration

Comes out-of-box in standalone-ha.xml profile

- Or add to your existing profile:

```xml
<extensions>
  ...
  <extension module="org.jboss.as.mod_cluster"/>
  ...
</extensions>

  ...
  <subsystem xmlns="urn:jboss:domain:modcluster:1.0">
    <mod-cluster-config advertise-socket="modcluster"/>
  </subsystem>

  ...
  <socket-binding-group name="standard-sockets" ...>
    <socket-binding name="modcluster" port="0" multicast-address="224.0.1.105" multicast-port="23364"/>
  ...
```
Demo: Try This At Home (Demo in LAB)

- Deployment
  - One HTTPd with mod_cluster
  - Two WildFly instances
  - No static configuration – dynamic auto-discovery
- Scenario
  - WAR demo application
  - Client GUI to generate load and track load-balancing
- Part of distribution so you can try yourself!
Questions?
Thank you!
Community

- http://www.wildfly.org/
- http://www.jgroups.org/
- http://www.infinispan.org/
- https://www.jboss.org/mod_cluster
- http://www.jboss.org/