



PV243

Clustering & Scalability

Radoslav Husár

Software Engineer
Red Hat

April 13, 2015

Open Topics

- Clusters
- JGroups
- Infinispan
- Clustering in WildFly 8
- mod_cluster



Why cluster?

- Interconnected
- But independent
- Made possible with
 - high-speed networking
 - and cheap commodity hardware
- Improve performance and/or availability
- Scale to handle higher load



WildFly Cluster

- 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
 - Failover
 - Fault Tolerance



High Availability / HA

- Capability to support server applications that can be reliably utilized with a minimum down-time; a service has a very high probability of being available.



Scalability

- Capability to handle a large number of requests by without service response time degradation;
- a service can handle a large number of requests by spreading the workload across multiple servers.



Failover

- If a service fails, the client can continue processing its tasks as another cluster member takes over the client's requests.



Fault Tolerance

- Guarantee of correct behaviour in the event of a failure.



What does Java EE say about clustering?

- Not much.



WildFly 8 Clustering Areas

- Web session replication
- Stateful Session Bean replication
- SingletonService
- Entity bean replication (2nd level caching)
- 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 `@Clustered @Stateful`
 - automatically clustered unless *`passivationCapable=false`*
- – Uses “ejb” cache container, by default



Distributable Sessions

- All session attributes must be serializable
 - Must implement `java.io.Serializable`
 - Most native Java objects implement this functionality
- After updating objects which are stored in the session
 - `HttpSession.setAttribute()` 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



Application Must be Cluster-Aware

- Don't spawn custom services that should be singleton in the cluster.
 - Mailer threads, whatnot.
 - Locking becomes complex
- Don't store data as flat files
 - Store over NAS/SAN (NFS)
 - Use DB
 - Use data grid



SingletonService

```
MyService service = new MyService();
SingletonService<Environment> singleton =
    new SingletonService<Environment>(service, MyService.SERVICE_NAME);
singleton.setElectionPolicy(new PreferredSingletonElectionPolicy(
    new NamePreference(SingletonService.DEFAULT_CONTAINER),
    new SimpleSingletonElectionPolicy()));
ServiceController<Environment> controller =
    singleton.build(CurrentServiceContainer.getServiceContainer())
    .addDependency(
        ServerEnvironmentService.SERVICE_NAME, ServerEnvironment.class, service.getEnvInjector())
    .install();
controller.setMode(ServiceController.Mode.ACTIVE);
```



EE6 @Singleton

- Not cluster-wide singleton!
- @Singleton per JVM as spec dictates



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.
- No HA JNDI (replaced with client JNDI).



Extensions for Clustering in WildFly

- `org.jboss.as.clustering.jgroups`
the JGroups extension, which provides the communication between cluster nodes
- `org.jboss.as.clustering.infinispan`
the Infinispan extension, which provides the replicated caching functionality
- `org.jboss.as.mod_cluster`
extension to provide integration and configuration with `mod_cluster` software load balancer



Predefined Profiles

- Standalone mode
 - *standalone-ha.xml*
 - *standalone-full-ha.xml*
- `$./bin/standalone.sh -server-config standalone/configuration/standalone-ha.xml`



Predefined Profiles

- Domain mode

- *ha profile*
- *full-ha profile*

- Use “ha” profile from domain.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

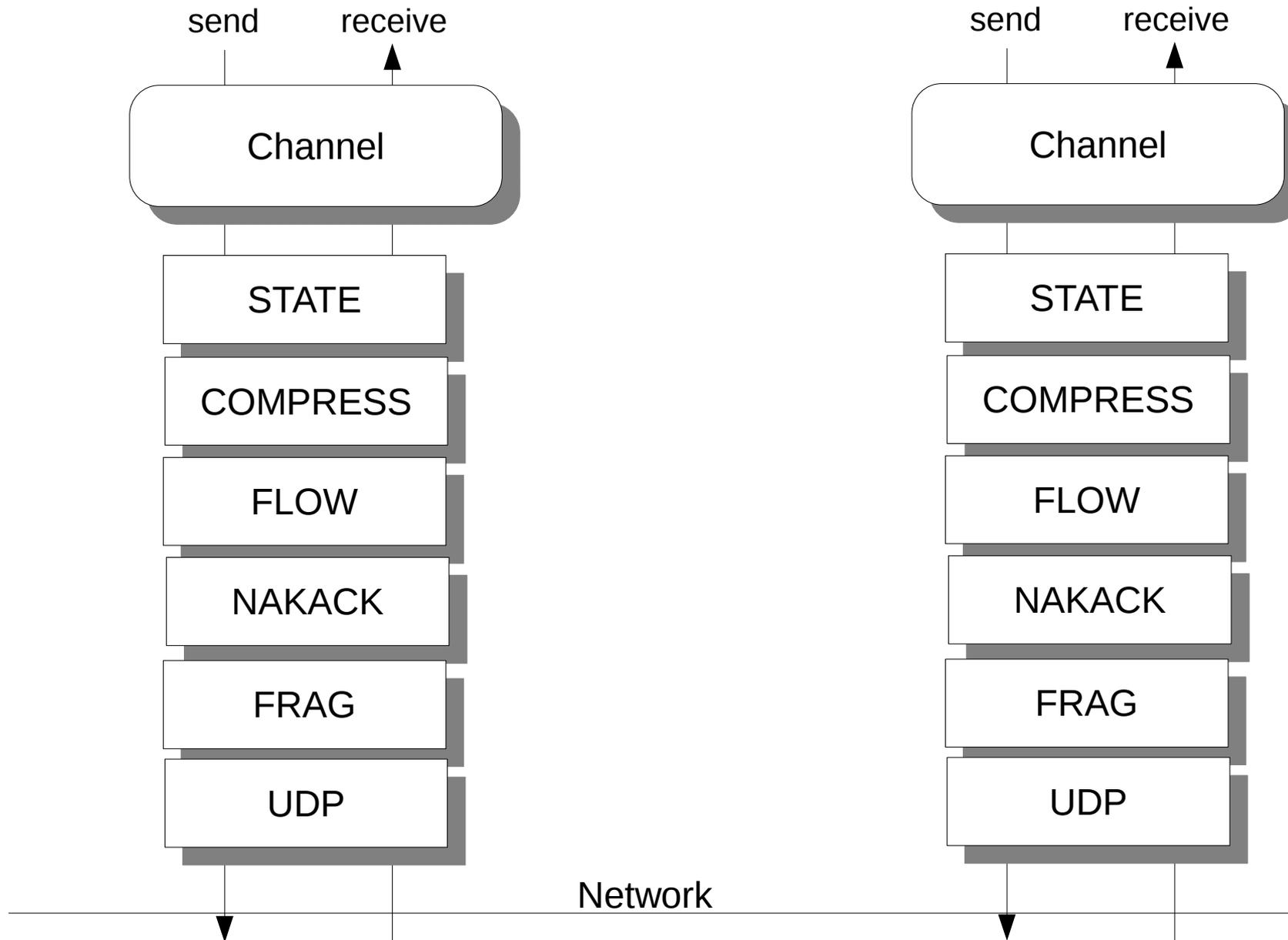


So what Is JGroups ?

- 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



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 (including crashes)
 - 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

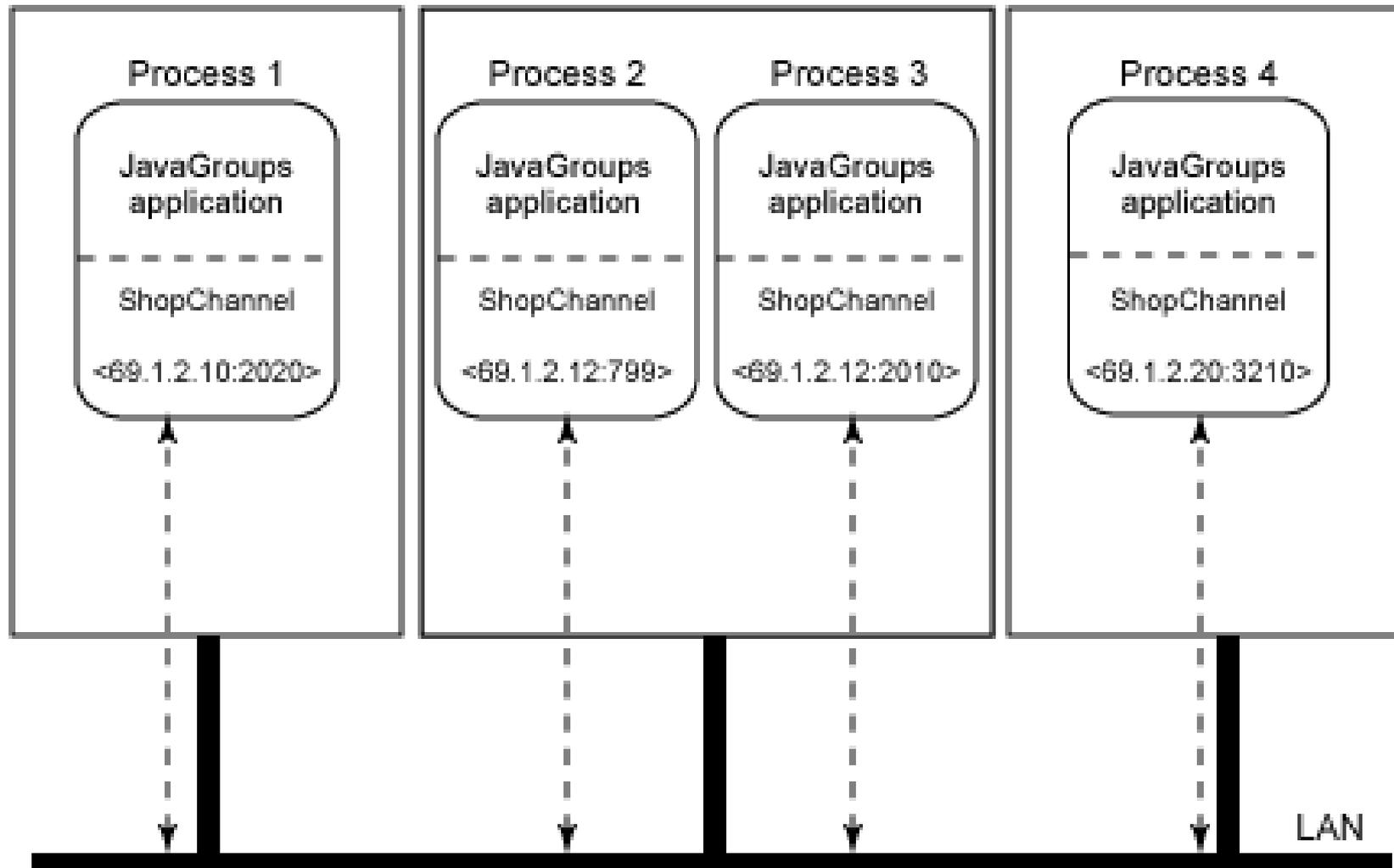


State transfer API

- `JChannel.getState()` called by state requester
- `ReceiverAdapter`:
 - `byte[] getState()`
 - Called on state provider
 - Needs to return serialized state
 - `void setState(byte[] state)`
 - Called on state requester
 - Needs to set state



Group Topology



Protocols (1)

- Transport
 - UDP (IP Multicast), TCP, TCP_NIE, LOOPBACK
- Member discovery
 - PING, TCPPING, TCPGOSSIP, MPING
- Failure detection (freeze up, crash)
 - FD, FD_SOCKET, 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





Infinispan

Infinispan

- Open source data grid platform
- Distributed key/value store
- Transactional (JTA & XA)
- Low-latency (in-memory)
- Java-based (with Scala sprinkles)
- Remote access not only from JVM
- Optionally persisted to disk
- Feature-rich
- Very actively developed



Let's look at API first though...

- Map-like key/value store
- JSR-107 Java Temporary Caching API
 - `javax.cache.Cache` interface
- Asynchronous API

- CDI API
- Upcoming JPA-like layer
- Hibernate OGM



TRANSACTIONS

- Transactions are optional, designed for from beginning
 - TRANSACTIONAL
 - NON_TRANSACTIONAL
- Transactional possible locking modes
 - OPTIMISTIC
 - PESSIMISTIC
- And 2 isolation modes available
 - REPEATABLE_READ
 - READ_COMMITTED



TRANSACTIONS

```
Cache cache = cacheManager.getCache();
```

```
TransactionManager tm =  
cache.getAdvancedCache().getTransactionManager();
```

```
tm.begin();  
cache.put(k1, v1);  
cache.remove(k2);  
tm.commit();
```



QUERYING

- Based on Hibernate Search

```
@Indexed
@ProvidedId
public class Event {
    @Field String title;
    @Field String annotation;
    @Field @DateBridge(resolution=Resolution.DAY) Date day;
```

....

```
org.apache.lucene.search.Query luceneQuery = queryBuilder.phrase()
    .onField( "title" )
    .andField( "annotation" )
    .sentence( "something" )
    .createQuery();
```

```
CacheQuery query = searchManager.getQuery( luceneQuery,
Event.class );
```

```
List<Event> objectList = query.list();
```



DISTRIBUTED EXECUTORS

- Leverage familiar ExecutorService, Callable abstractions
- Expand it to distributed, parallel computing paradigm
- Looks like a regular ExecutorService
- Feels like a regular ExecutorService
- The “magic“ that goes on Infinispan grid is completely transparent to users

MAP REDUCE...



EXPIRATION

- Specify maximum time entries
 - stay in cache (lifespan)
 - stay in cache untouched (maxIdle)
- Can set default expiration in cache config
- Can explicitly set lifespan or maxIdle with every PUT

```
cache.put("Bad smell", "I'll begone in 30 seconds", 30,  
TimeUnit.SECONDS);  
cache.put("Annoying Girlfriend", "If you don't tell me you  
love me every 5 minutes I 'll be gone!", -1,  
TimeUnit.SECONDS, 5, TimeUnit.MINUTES);
```



EXPIRATION in AS

- HTTP Sessions expire
 - Timeout in web.xml
- SFSB Sessions expire
 - @CacheConfig annotation
- Sessions expire so that
 - Don't consume resources
 - They don't get abused if they are not invalidated



EVICTIOIN

- Set maximum # of entries to keep in cache
- Multiple out-of-box eviction strategies
 - UNORDERED
 - FIFO
 - LRU – Least recently used
 - LIRS – Low Inter-Reference Recency Set



CACHE STORE / PERSISTENCE

- Store data from memory to other kind of storage
 - File System (FileCacheStore)
 - Relational Database (JdbcBinaryCacheStore, JdbcStringBasedCacheStore)
 - Other NoSQL stores (Cassandra, JClouds BlobStore, RemoteCacheStore)
- Not only in-memory
 - Write-through caching
 - Write-behind caching
- Passivation support (spillover to disk)
- Preloading & warm start support



PASSIVATION IN WILDFLY 8

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

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

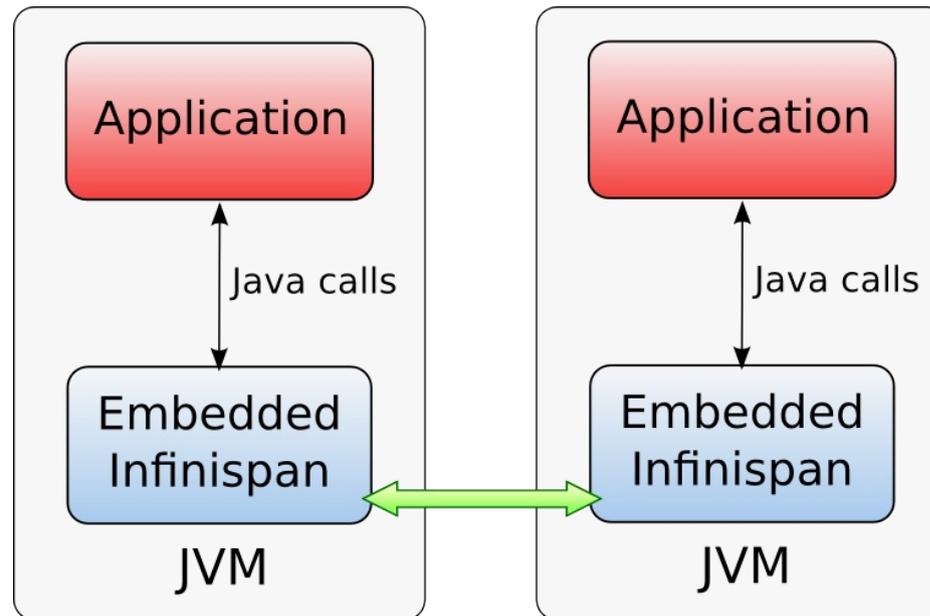


EVICTIOIN and PERSISTENCE in AS

- 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



Embedded Access Mode



Cache Modes



LOCAL

- Single node
- Non-clustered environment
 - Unaware of other instances on network
- Why use LOCAL cache

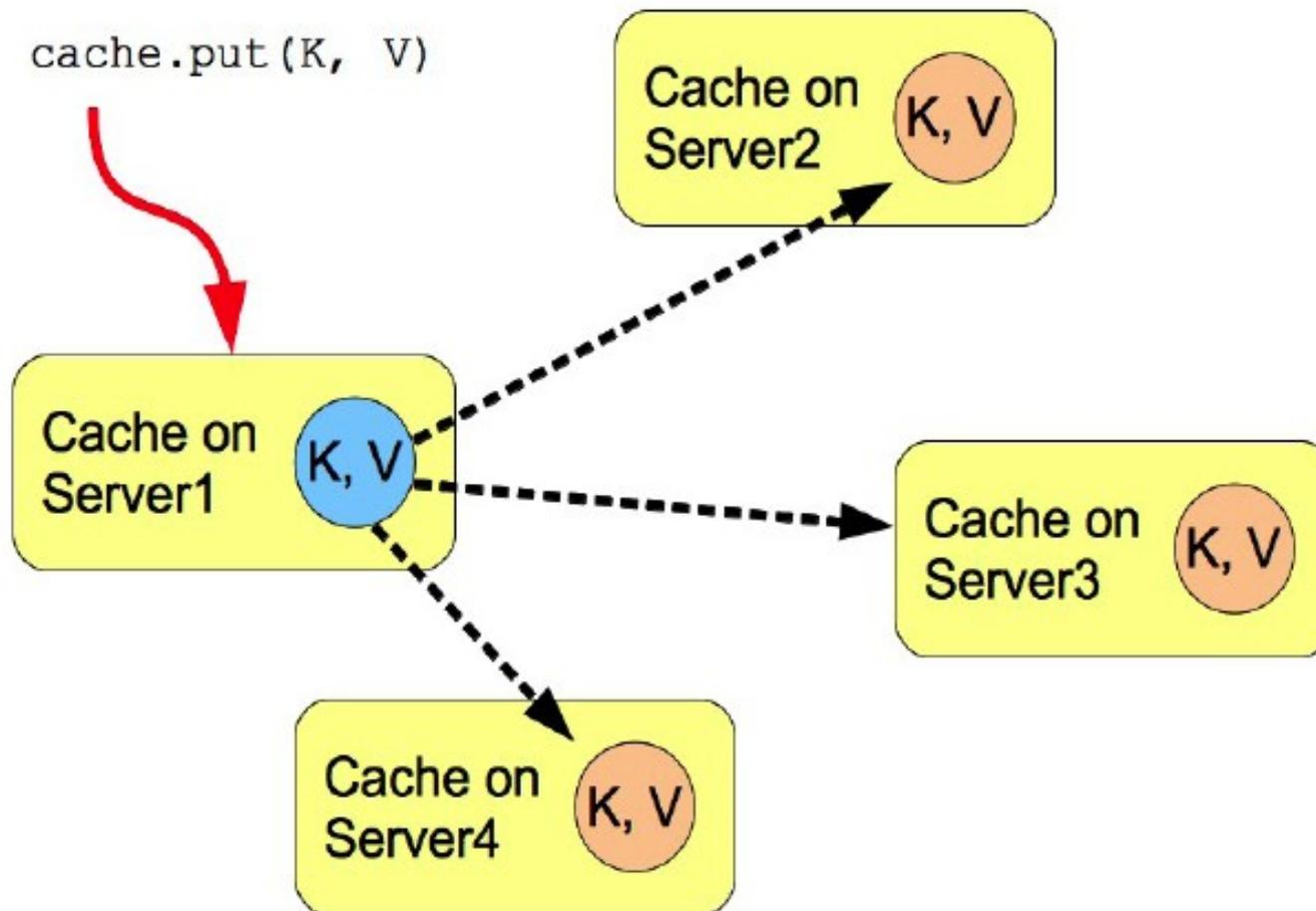


Replication mode

- 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
 - Do we need read-friendly in session clustering?
 - Instant scale-in, no state transfer on leave
- Disadvantages
 - Write unfriendly, put must be to every node
 - Doesn't scale
 - Upon join all state has to be transferred to new node
 - Heap size stays the same when we add nodes



REPLICATION



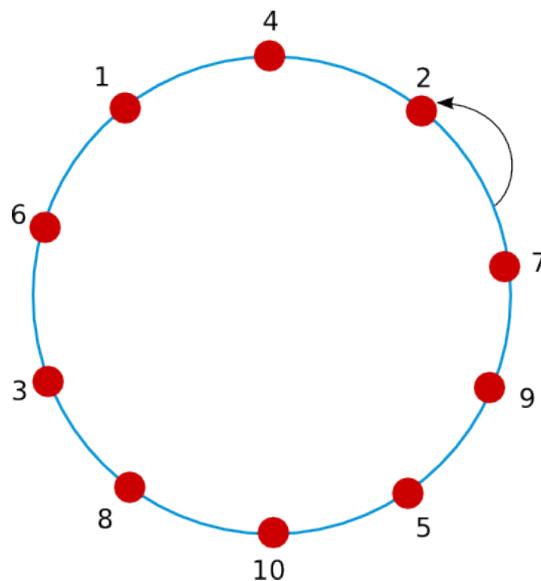
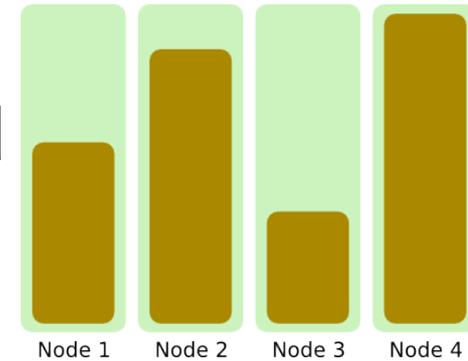
DISTRIBUTION

- 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)

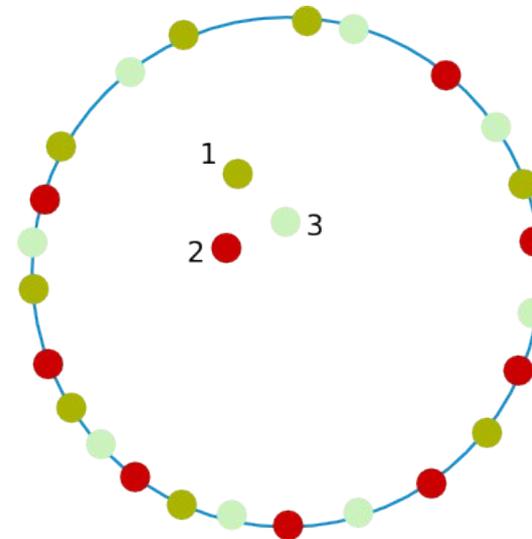


Consistent Hash function

- Even distribution of entries – balanced load
- Less expected rehash on node leave / join
- How usable in clustering?
- Who decides where the session will be stored?



Hash wheel

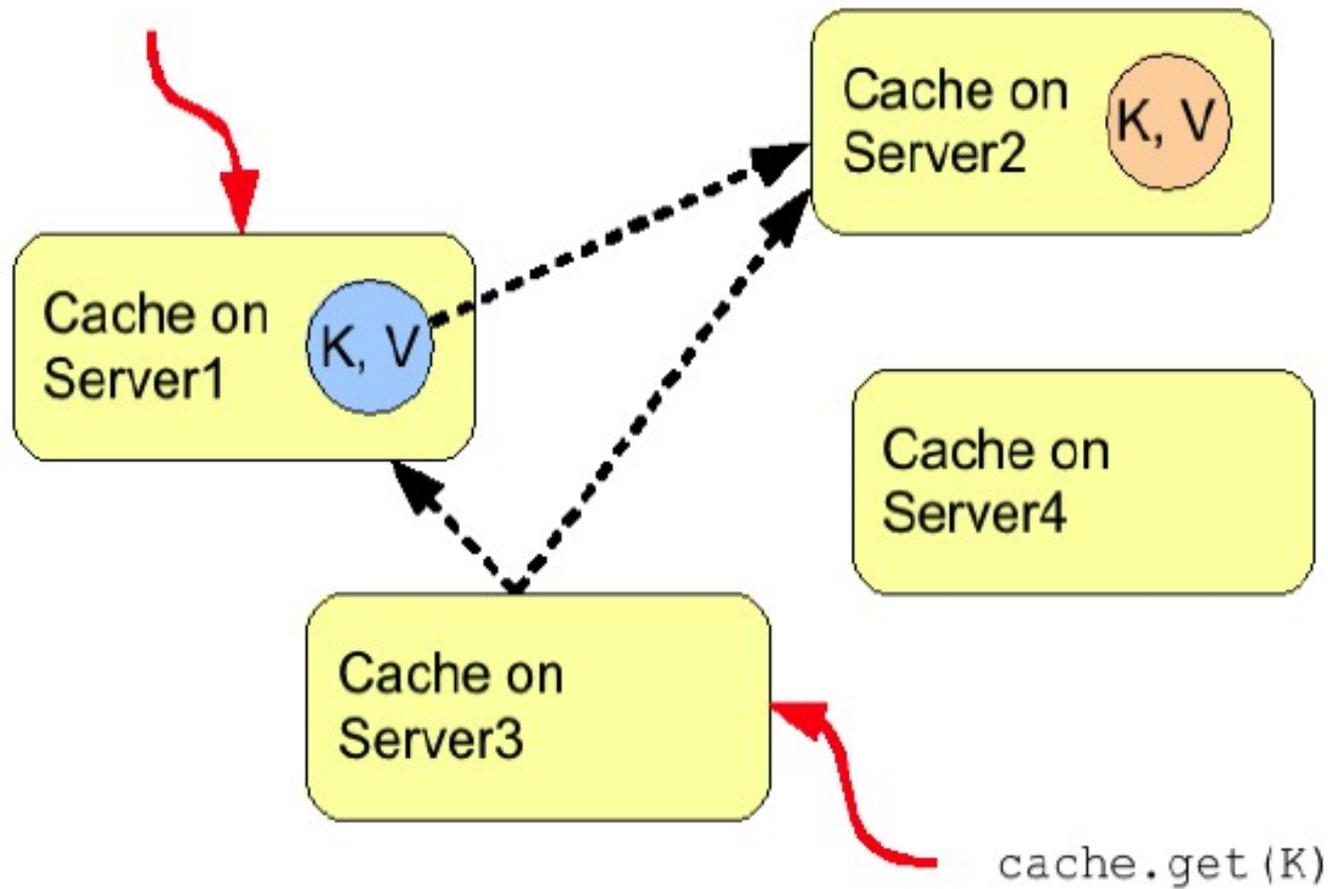


Virtual nodes



DISTRIBUTION

`cache.put (K, V)`

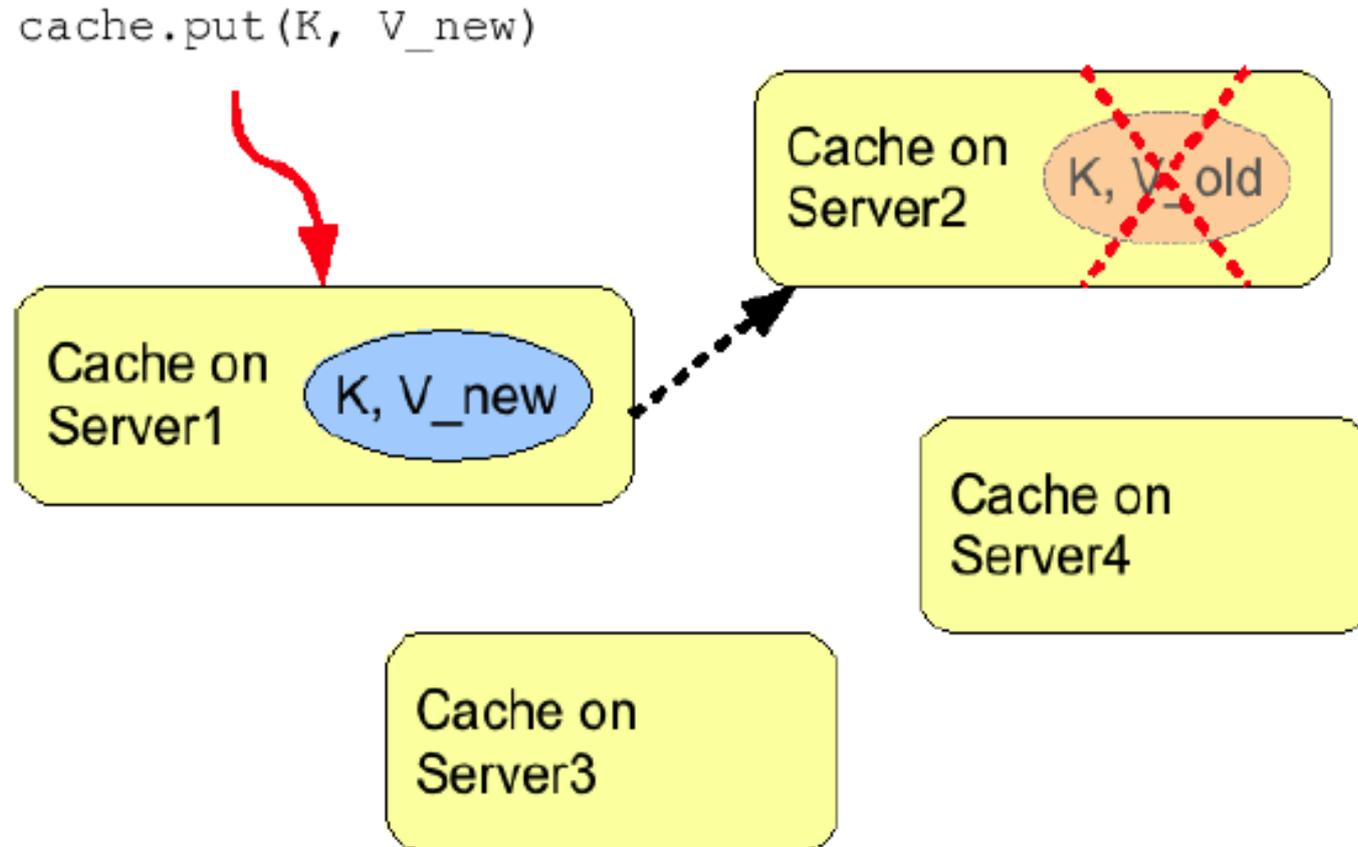


INVALIDATION

- Usable when often read, but rarely written (change entries)
- If entry exists in node's local cache
 - it's valid and can be returned to requestor
- If 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 cluster traffic, each PUT issues small invalidation message
- When use in clustering?
 - Suitable for RDBMS off-loading, used with shared cache store



INVALIDATION



SYNC and ASYNC

- Synchronous
 - All operations get confirmation that the other relevant cluster nodes reached the desired state
 - Implications to response times
 - 2PC
- 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.
- When use which?



Using Infinispan from AS

- Customizing Infinispan Caches
- JNDI binding
 - `<cache-container ... jndi-name="...">`
 - Assumes `java:global` namespace if unqualified



Using Directly

- On demand injection of cache container

@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

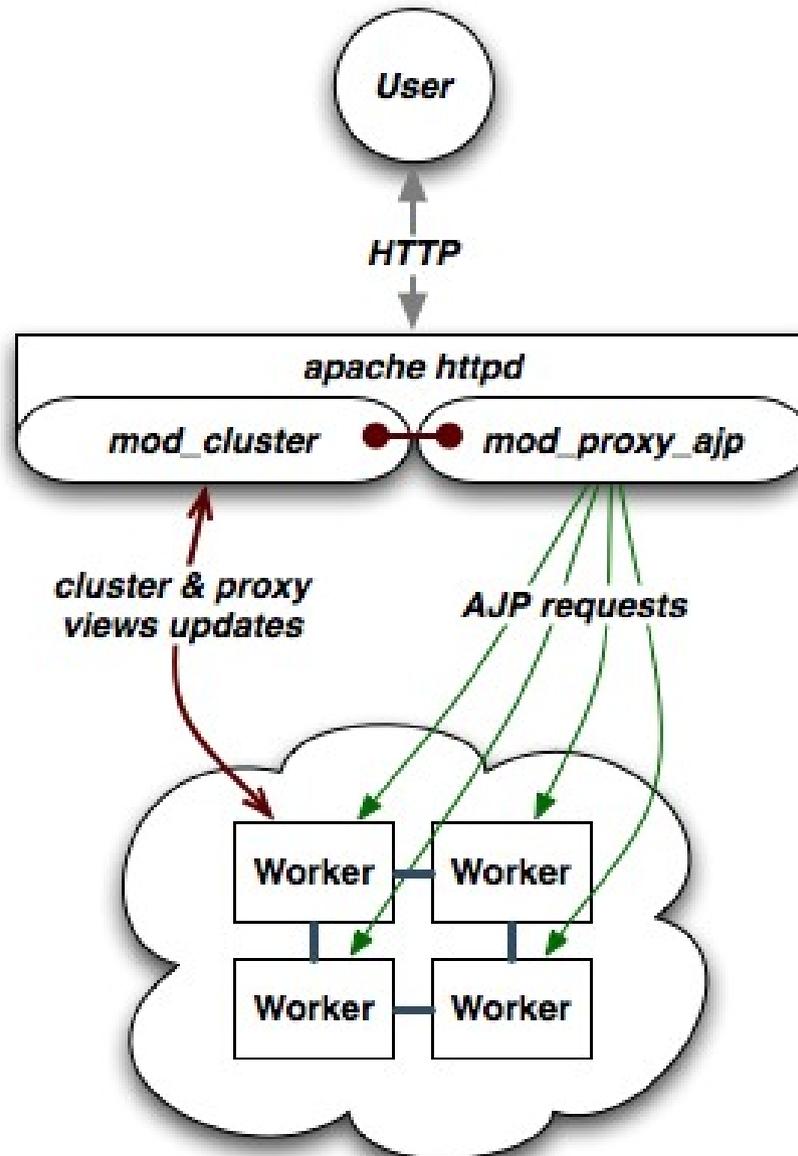


Architecture

- 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 advertize themselves for web servers to register with them using UDP multicast
 - No topology information



No more `worker.properties` & `uriworkermap.properties`

```
worker.list=lb  
worker.lb.type=lb  
worker.lb.balance_workers=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, failover 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 slotmem_module modules/mod_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

Listen 192.168.1.1:8000

<VirtualHost 192.168.1.1:8000>
  <Directory />
    Order deny,allow
    Deny from all
    Allow from 192.168.2.
  </Directory>

  KeepAliveTimeout 60
  MaxKeepAliveRequests 0
  AdvertiseGroup 224.0.1.105:23364
</VirtualHost>
```



Configuration in WildFly

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

```
./bin/standalone.sh -c standalone-  
ha.xml
```

- Or add to your existing profile:

```
<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"/>  
...
```



mod_cluster Subsystem Operations

- add
- add-custom-metric
- add-metric
- add-proxy
- disable
- disable-context
- enable
- enable-context
- list-proxies
- stop-context
- read-proxies-configuration
- read-proxies-info
- refresh
- remove
- remove-custom-metric
- remove-metric
- remove-proxy
- reset
- stop



Possible Demo

- 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





Thank you!

