



Management and Monitoring

Rostislav Svoboda

Senior Quality Assurance Engineer, JBoss by Red Hat

Advanced Java EE Lab @ CVUT

Dec 14th 2012

About me

- Senior Quality Assurance Engineer, JBoss by Red Hat
- ~3 years with company
- Experience with JBoss AS since 2005
- Known components: JBossWS Native stack, mod_jk, ISAPI, NSAPI, HornetQ, AS 5/6 testsuite
- Main focus: JBossWS CXF stack, performance testing, AS7 / EAP6 model, IPv6, some internal stuff

rsvoboda@redhat.com



Agenda

- Monitoring
 - JDK tools
 - System tools
 - AS7 specifics
- AS7 goals, architecture
- AS7 Domain Model
- AS7 Management
 - CLI / Scripting + Java API + HTTP API
 - WebUI + RHQ / JON



Monitoring – motivation

Everything is working as expected, right ?

No bugs, only features ...

We will learn how to do some basic investigation and JVM monitoring.



JDK tools - JAR level investigation

`jar tf $file` or `unzip -l $file`

```
jar tf jboss-modules.jar
```

`javap -classpath $file FQCN`

```
javap -classpath jboss-modules.jar org.jboss.modules.JarModuleLoader
```

```
javap -private -classpath jboss-modules.jar org.jboss.modules.JarModuleLoader
```

`javap -c -classpath $file FQCN`

```
javap -c -classpath jboss-modules.jar org.jboss.modules.JarModuleLoader
```

bytecode disassembled into the bytecode instructions defined by the JVM specification



JDK tools – info about process

`jps -l [-m -v]`

JDK specific

`jinfo $PID`

Java system properties + VM flags



JDK tools – info about memory

`jmap $PID`

`jmap -heap $PID`

java heap summary

`jmap -dump:file=heap-dump $PID`

dump java heap in hprof binary format

`jhat heap-dump`

Check <http://127.0.0.1:7000/>



JDK tools – stack trace and JVM stats

`jstack -l $PID`

stack traces of Java threads for a given Java process, thread locking issues

`kill -QUIT $PID` or `Ctrl + \`

detail in java process console - same as `jstack` + heap details

`jstat -gcutil -t $PID 1s 30`

summary of GC statistics, 30 snapshots, each second one generated

`jstat -class $PID`

Class loader statistics



JDK tools – GUI

jconsole \$PID

Heap and Non-Heap memory usage, CPU usage, VM summary
number of threads and classes, stack trace for each thread
MBeans details

jvisualvm

nicer look & feel, based on NetBeans platform
Heap and PermGen memory usage, CPU usage, VM summary
number of threads and classes, details for each thread, not stack trace
lightweight CPU and memory profiling + sampling



System tools – OS, CPU

OS

Linux uname -a, cat /etc/redhat-release

Solaris uname -a

Windows hostname, ver

CPU

Linux top, htop, cat /proc/cpuinfo,

Solaris top,

Windows Ctrl+Alt+Delete --- Task Manager



System tools – memory, disk

Memory

Linux free, vmstat -a

Solaris vmstat -a, vmstat 3

Windows Ctrl+Alt+Delete --- Task Manager

Disk

Linux df -h, du -h

Solaris df -h, du -h

Windows Ctrl+Alt+Delete --- Task Manager --- advanced monitoring



System tools – processes, ports

Processes

Linux ps aux, top, kill -9

Solaris ps -aef, pargs, pfiles, kill -9

Windows Ctrl+Alt+Delete --- Task Manager, taskkill /F /T /PID %PID%

WMIC PROCESS get Caption,Commandline,Processid

Ports

Linux netstat -natup, tcpdump

Solaris netstat -an

Windows netstat -a -n



AS7 specifics

JDR - JBoss Diagnostic Reporter

bin/jdr.sh [.bat]

JBoss specific tool for diagnostic

add at least one user into ManagementRealm using bin/add-user.sh

jconsole

bin/jconsole.sh [.bat]

Management of JBoss AS7 is exposed over a native interface build on top of JBoss Remoting, also JSR-160 connector is provided to make JMX remotely accessible.



Advanced tools

- your IDE debugger
- your IDE profiler
- JProfiler - <http://www.ej-technologies.com/products/jprofiler/overview.html>
- Java Decompiler - <http://java.decompiler.free.fr/>
- TDA - Thread Dump Analyzer - <http://java.net/projects/tda/>
- MAT - Memory Analyzer - <http://www.eclipse.org/mat/>

- Sysinternals tools - - <http://technet.microsoft.com/en-us/sysinternals/bb842062>
- Wireshark - <http://www.wireshark.org/>



AS7 motivation aka look at the mirror

- Legacy subsystems
- Boot time
- Memory footprint
- Testability
- Modularity
- Administration options
- Not “good enough”

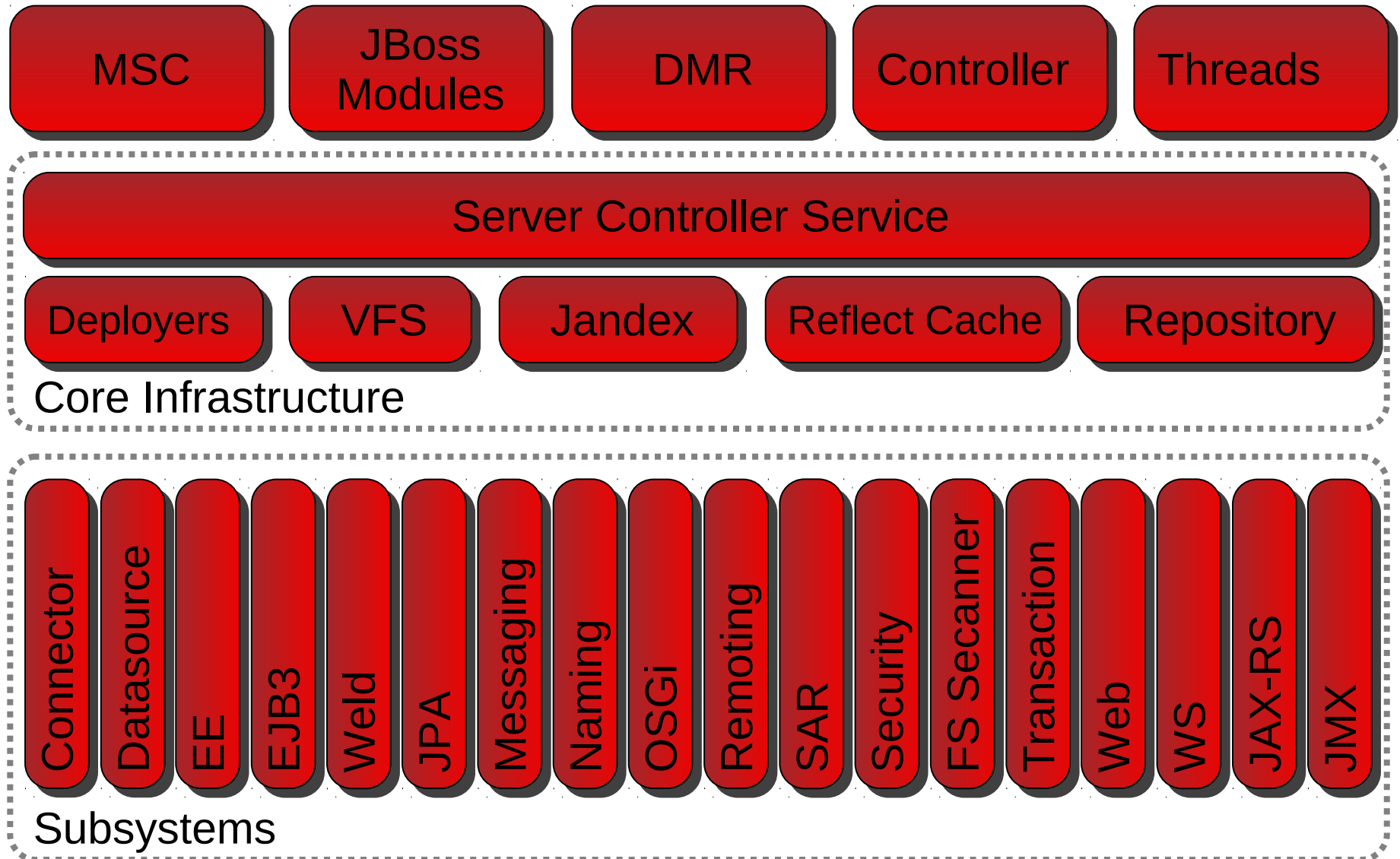


AS7 goals and features

- Make it smaller and faster
- Remove legacy stuff
- Remove unnecessary abstraction layers
- Improve manageability
- Multi-node management
- Simplify configuration
- Modularize



AS7 Architecture



MSC - Modular Service Container

- <https://github.com/jbossas/jboss-msc>
- Replacing the JBossAS 5 & 6 microcontainer
- Small and lightweight
- Everything is a service
- Services have only two states Up & Down
- Additional lifecycle states modeled using additional services
- New algorithms for checking module/service dependencies at deploy and run time
- Only the services you need are loaded and started
- Services are unloaded when not needed



MSC - Modular Service Container – verbose

JBoss AS 7 starts and deploys all services in parallel, this complex problem is solved by new service container.

MSC is essentially an advanced concurrent and scalable state machine. It analyzes the dependencies between all services on the fly and attempts to start as many as it can at the same time, while still adhering to the relationship requirements. This means fast startup and multiple deployments in parallel.

- MSC configured to use a threadpool size of double the number of CPUs
- Annotation indexing, caching of reflection metadata
- StAX based parsing
- Lazy loading functionality
- Massive threading



JBoss Modules - Modular Classloading System

- <https://github.com/jbossas/jboss-modules>
- <https://community.jboss.org/wiki/ModuleCompatibleClassloadingGuide>
- <https://docs.jboss.org/author/display/MODULES/Home>
- See module.xml files in AS7 modules subdirectory

- Standalone implementation of
 - modular (non-hierarchical) class loading and
 - execution environment for Java
- No more classloader hell, scoping of classloaders more narrowly defined
- You must explicitly allow deployed services and applications to see and use dependencies
- Transitive dependencies are hidden, by default



JBoss Modules - verbose

In addition to parallel services (MSC), JBoss AS 7 also has modularity and concurrent class loading. By segmenting classes into proper modules, the app server can naturally optimize access patterns, and only look in the spot that truly owns the classes.

Also, since the visibility between modules is intentionally limited, searching is cheap. In the case of JBoss Modules, module resolution and class lookup are constant. All of this has a very high degree of concurrency, even a significant portion of class definition.

In other words, rather than a single class loader which loads all JARs into a flat class path, each library becomes a module which only links against the exact modules it depends on, and nothing more.



DMR – Dynamic Model Representation

- <https://github.com/jbossas/jboss-dmr>
- <https://docs.jboss.org/author/display/AS71/Detyped+management+and+the+jboss-dmr+library>

- Central detyped management API
- All management operations operate with/on DMR
- Compatibility is stressed
- Self describing
- Convertible from/to JSON
- Represents simple and complex types
 - int, long, big int, double, big dec, boolean, string, bytes, list, object, property, expression



DMR – Dynamic Model Representation

- `org.jboss.dmr.ModelNode`
 - wrapper around some value, typically some basic JDK type
 - expressions are quite interesting
- `org.jboss.dmr.ModelType`
 - just enum for supported types
- `org.jboss.dmr.Property`
 - `String => ModelNode` tuple

```
bsh % ModelNode node = new ModelNode();
```

```
bsh % print(node.getType());
```

```
UNDEFINED
```

```
bsh % node.set(1);
```

```
bsh % print(node.getType());
```

```
INT
```

```
bsh % node.set(true);
```

```
bsh % print(node.getType());
```

```
BOOLEAN
```

```
bsh % node.set("Hello, world");
```

```
bsh % print(node.getType());
```

```
STRING
```



Domain model – key goals

- Self-contained, stable configuration and management API
 - End user configuration centralized in a few files
 - Configuration separated from service wiring
 - Everything in configuration schema exposed via management API
 - Schema files for all configurations
 - Stable == no incompatible changes



Domain model – key goals

- Manage multiple servers via a single control point
 - Configure a cluster
 - Start/stop nodes in a cluster
 - Rolling deployment to a cluster
 - Apply a patch to a set of servers (TBD)
- Management API exposed via:
 - Typed Java interface
 - REST
 - CLI
- Domain API is the only supported management API



Domain vs. standalone

Standalone

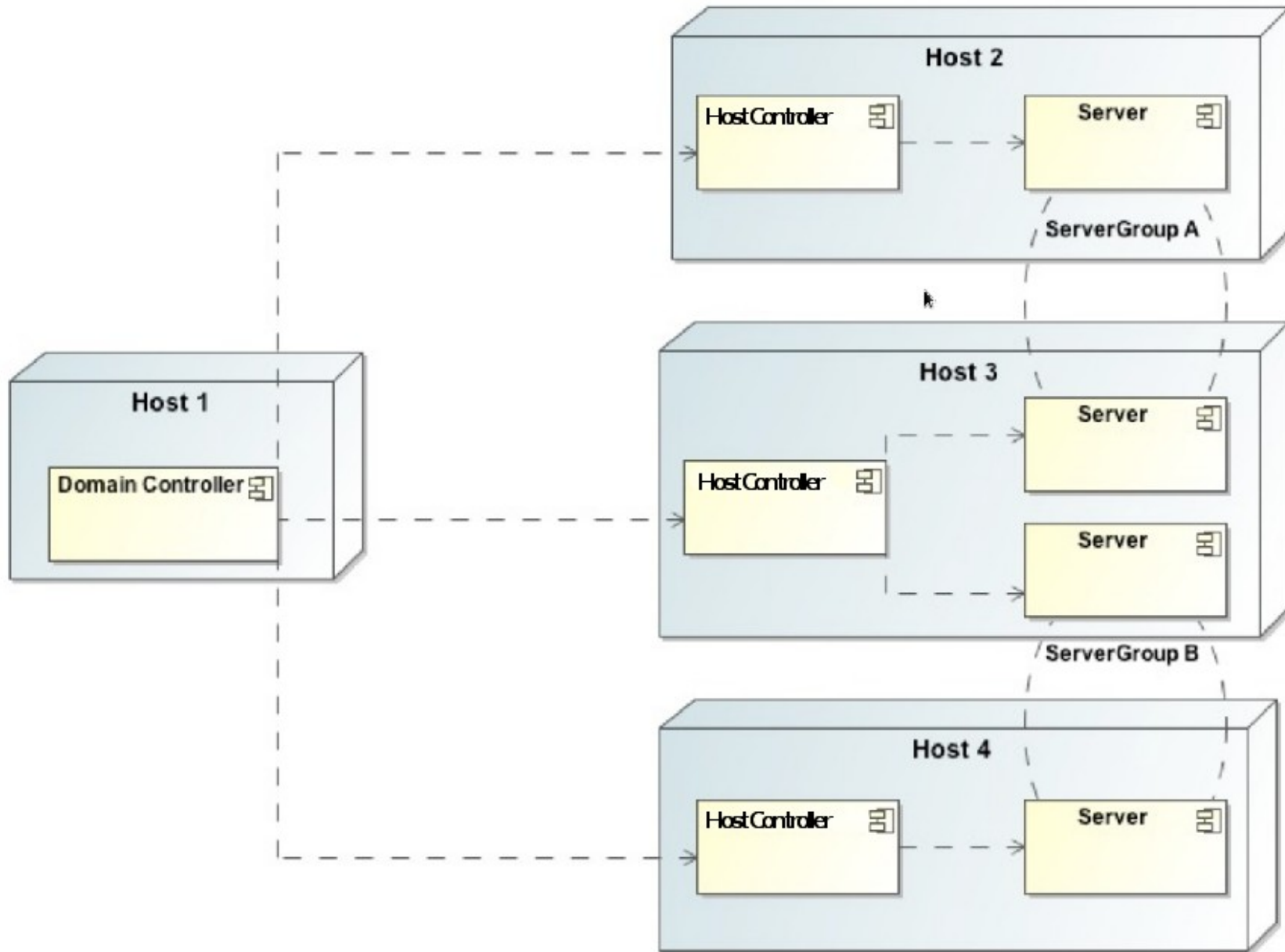
- Traditional JBoss single JVM server
- Management facilities IN-VM
- No lifecycle management, just shutdown
- Development and embedded solutions

Domain

- Multi-JVM, multi-server model
- Lifecycle managed by Process Controller (PC)
- Management coordinated by Domain Controller (DC)
- Multiple server instances per host managed by Host Controller (HC)
- HC on master node is DC



Domain model



Domain model - terms

- <https://community.jboss.org/wiki/DomainManagementModelDesign>
- Domain
- Cluster
- Server
- ServerGroup
- Profile
- Subsystem
- Module



Domain model - schema

- <https://community.jboss.org/wiki/JBossASDomainSchema>
- docs/schema/jboss-as-config_1_2.xsd in AS7 distribution
- docs/schema/*.xsd
- standalone.xml, domain.xml, host.xml

```
<server-groups>
  <server-group name="main-server-group" profile="default">
    <jvm name="default">
      <heap size="64m" max-size="512m"/>
    </jvm>
    <socket-binding-group ref="standard-sockets"/>
  </server-group>
  <server-group name="other-server-group" profile="default">
    <jvm name="default">
      <heap size="64m" max-size="512m"/>
    </jvm>
    <socket-binding-group ref="standard-sockets"/>
  </server-group>
</server-groups>

  <servers>
    <server name="server-one" group="main-server-group" auto-start="true">
      <jvm name="default"/>
    </server>
    <server name="server-two" group="main-server-group" auto-start="true">
      <jvm name="default">
        <heap size="64m" max-size="256m"/>
      </jvm>
      <socket-binding-group ref="standard-sockets" port-offset="150"/>
    </server>
    <server name="server-three" group="other-server-group" auto-start="false">
      <socket-binding-group ref="standard-sockets" port-offset="250"/>
    </server>
  </servers>
```



Domain management

- End user configuration centralized in a few files
- Config changes made via management tools persisted back to the config file
- Secure remote access via:
 - Native Java interface
 - HTTP/REST + JSON
 - CLI
- All about DMR and detyped management API
 - `<socket-binding name="management-native" interface="management" port="${jboss.management.native.port:9999}"/>`
 - `<socket-binding name="management-http" interface="management" port="${jboss.management.http.port:9990}"/>`



Detyped Management API

- <https://community.jboss.org/wiki/AS7DetypedManagementAPI>
 - <https://community.jboss.org/wiki/DetypedDescriptionOfTheAS7ManagementModel>
 - <https://community.jboss.org/wiki/FormatOfADetypedOperationRequest>
 - <https://community.jboss.org/wiki/FormatOfADetypedOperationResponse>
-
- The management client libraries needed to be forward compatible
 - It is highly unlikely that an API that consists of hundreds of Java types could be kept forward compatible
 - A detyped API works by making it possible to build up arbitrarily complex data structures using a small number of Java types



CLI

- Command line management tool for the AS 7 server
- Command `bin/jboss-cli.sh` or `bin/jboss-cli.bat`
- Interactive mode
- Non-interactive mode
- Batch mode
- GUI mode
- Operations based on model
- Generic CLI commands



CLI

```
$ ./bin/jboss-cli.sh --connect controller=IP_ADDRESS
[standalone@IP_ADDRESS:9999 /] /system-property=foo:add(value=bar)
[standalone@IP_ADDRESS:9999 /] /system-property=foo:read-resource
{
  "outcome" => "success",
  "result" => {"value" => "bar"}
}
[standalone@IP_ADDRESS:9999 /] /system-property=foo:remove
{"outcome" => "success"}
```

```
[domain@IP_ADDRESS:9999 /] /system-property=foo:add(value=bar)
[domain@IP_ADDRESS:9999 /] /system-property=foo:read-resource
[domain@IP_ADDRESS:9999 /] /system-property=foo:remove
```

```
[domain@IP_ADDRESS:9999 /] /host=master/system-property=foo:add(value=bar)
[domain@IP_ADDRESS:9999 /] /host=master/system-property=foo:read-resource
[domain@IP_ADDRESS:9999 /] /host=master/system-property=foo:remove
```

```
[domain@IP_ADDRESS:9999 /] /host=master/server-config=server-one/system-property=foo:add(value=bar)
[domain@IP_ADDRESS:9999 /] /host=master/server-config=server-one/system-property=foo:read-resource
[domain@IP_ADDRESS:9999 /] /host=master/server-config=server-one/system-property=foo:remove
```



CLI

- <https://community.jboss.org/wiki/CommandLineInterface>
- <https://community.jboss.org/wiki/GenericTypeCLICommands>
- <https://community.jboss.org/wiki/CLICompoundValueFormat>
- <https://community.jboss.org/wiki/CLINon-interactiveMode>
- <https://community.jboss.org/wiki/CLIBatchMode>
- <https://docs.jboss.org/author/display/AS71/CLI+Recipes>

- <https://community.jboss.org/wiki/JBossAS7Command-linePublicAPI>



Java API

- Native management interface uses an open protocol based on the JBoss Remoting library
- The management protocol is an open protocol, so a completely custom client could be developed without using prepared libraries (e.g. using Python or some other language)
- Maven artifact `org.jboss.as:jboss-as-controller-client`
- <https://docs.jboss.org/author/display/AS71/The+native+management+API>



Java API

```
ModelControllerClient client = ModelControllerClient.Factory.  
    create(InetAddress.getByName("localhost"), 9999);
```

```
ModelNode op = new ModelNode();  
op.get("operation").set("read-resource");  
op.get("recursive").set(true);  
op.get("include-runtime").set(true);  
op.get("recursive-depth").set(10);
```

```
ModelNode returnVal = client.execute(op);  
System.out.println(returnVal.get("result").toString());  
client.close();
```



HTTP API

- <http://localhost:9990/management>
- Sometimes called REST API
- HTTP request in JSON like format
- The default operation is read-resource
- add user into ManagementRealm using `bin/add-user.sh`

- <https://docs.jboss.org/author/display/AS71/The+HTTP+management+API>
- <https://community.jboss.org/wiki/HTTPJSON-likeAPI>



HTTP API

- GET

- <http://localhost:9990/management?recursive&include-runtime&json.pretty>
- [management?operation=resource-description&recursive&operations](http://localhost:9990/management?operation=resource-description&recursive&operations)
- [management/subsystem/web/connector/http?include-runtime&json.pretty](http://localhost:9990/management/subsystem/web/connector/http?include-runtime&json.pretty)
- [management/subsystem/web?operation=operation-names&json.pretty](http://localhost:9990/management/subsystem/web?operation=operation-names&json.pretty)

- POST

- `curl --digest -L -D - http://localhost:9990/management --header "Content-Type: application/json" -d '{"operation":"read-resorce","json.pretty":1}' -u ferda`
- `curl --digest -L -D - http://localhost:9990/management --header "Content-Type: application/json" -d '{"operation":"read-attribute","address":[{"host":"master"}, {"server":"server-one"}],"name":"server-state","json.pretty":1}' -u ferda`



HTTP API

```
curl --digest -L -D - http://localhost:9990/management --header "Content-Type: application/json" -d '{"address" : [{ "socket-binding-group" : "standard-sockets" }, { "socket-binding" : "test" }], "operation" : "add", "port" : 8181, "json.pretty":1}' -u ferda
```

```
curl --digest -L -D - http://localhost:9990/management --header "Content-Type: application/json" -d '{"address" : [{ "subsystem" : "web" }, { "connector" : "test-connector" }], "operation" : "add", "socket-binding" : "test", "scheme" : "http", "protocol" : "HTTP/1.1", "enabled" : true, "json.pretty":1}' -u ferda
```

```
curl --digest -L -D - http://localhost:9990/management --header "Content-Type: application/json" -d '{"address" : [{ "subsystem" : "web" }, { "connector" : "test-connector" }], "operation" : "remove", "json.pretty":1}' -u ferda
```

```
curl --digest -L -D - http://localhost:9990/management --header "Content-Type: application/json" -d '{"address" : [{ "socket-binding-group" : "standard-sockets" }, { "socket-binding" : "test" }], "operation" : "remove", "json.pretty":1}' -u ferda
```



Web console

The screenshot shows the JBoss Application Server 7.1 web console. The browser address bar indicates the URL is localhost:9990/console/App.html#server-instances. The page title is "JBoss Application Server 7.1". The navigation menu includes "Profiles", "Server", and "Runtime". The "Runtime" tab is active, showing "Server Instances".

Server: master: server-one

Server Instances

Server Status (Host: master)

Server instances represent the server runtime state. This includes the virtual machine status, as well as deployments and subsystem specific state (i.e. datasource pool sizes).

[Stop](#)

Server	Server Group	Status	Active
server-one	main-server-group		✓
server-three	other-server-group		⊘
server-two	main-server-group		✓

1-3 of 3

Status

[Availability](#) [Environment Properties](#)

Server Instance: server-one Server Configuration: server-one

Running?: true

1.1.0.FINAL [Settings](#) [Logout](#)



RHQ / JON

The screenshot displays the JBoss ON web interface for a resource named 'DomainController'. The browser address bar shows 'localhost:7080/coregui/#Resource/10011/Summary/Activity'. The navigation menu includes Dashboard, Inventory (selected), Reports, Bundles, Administration, and Help. The user is logged in as 'rhqadmin'.

The main content area is titled 'DomainController JBossAS7 Host CONTROLLER'. It features several tabs: Summary, Inventory, Alerts, Monitoring, Events, Operations, and Configuration. The 'Summary' tab is active, showing a list of resource metrics:

- Resource: Measurements
 - Maximum request time 118ms
 - Number of management requests per Minute 150.62
 - Time used for management requests per Minute 6.34s
 - see more...
- Resource: Event Counts
 - No event counts based off display criteria.
- Resource: OOB Metrics
 - No OOB conditions found

On the right side, there are three panels:

- Resource: Alerts**: A table with columns 'Creation Time', 'Name', 'Condition Text', 'Priority', and 'Status'. It displays 'No results found using specified criteria.' and has 'Delete' and 'Acknowledge' buttons.
- Resource: Operations**: A table with columns 'Date Submitted', 'Operation', 'Requestor', and 'Status'. It displays 'No items to show.' and has 'Delete' and 'Force Delete' buttons.
- Resource: Configuration Updates**: A table with columns 'Version', 'Date Submitted', 'Date Completed', 'Status', 'User', and 'Update Type'.

At the bottom left, there are 'Edit Mode' and 'Reset' buttons. The left sidebar shows a tree view of the system hierarchy, with 'DomainController' selected under 'JBossAS7 Host Controllers'.



That's all ...

