WildFly

Monitoring and Management

Ladislav Thon

Senior Software Engineer, Red Hat

Advanced Java EE Lab @ FEL ČVUT

Nov 2018

Agenda

- Monitoring
  - System tools, JDK tools, WildFly tools
- Management
  - WildFly history and overview
  - Architecture, management model, RBAC
  - CLI, Web console, Java API, HTTP API
- WildFly in the clouds
  - Thorntail
  - OpenShift

WildFly Monitoring

Monitoring – motivation

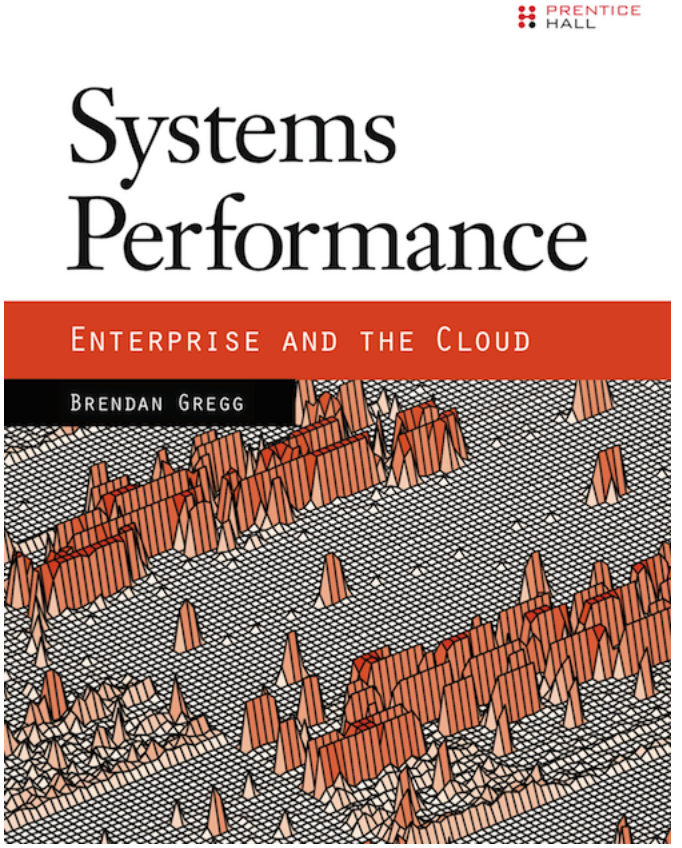
You are using WildFly, so bright future lies ahead…

Remember Murphy's law!

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

System tools

- OS info
- Processes
- Memory
- Disk
- Network



System tools – OS info, processes

- Gather basic info about OS and hardware
  - `cat /etc/os-release, uname -a, dmesg`
  - `cat /proc/cpuinfo, cat /proc/meminfo`
  - `lstopo, lshw, lscpu, lsmem, lspci, lsusb`

- Monitoring and managing running processes
  - `top, htop, ps aux, pstree -a -T`
  - `kill [-9] <PID>`
  - `pgrep, pkill`
  - `pidstat, mpstat`

System tools – memory, disk

- Monitoring memory usage
  - `free -m`
  - **`vmstat`** `-w 1`
- Monitoring disk usage
  - `df -h, du -h`
  - `mount, findmnt, lsblk [-f]`
  - `iostat, iotop`

System tools – network, others

- Monitoring network usage
  - `netstat, ss`
  - `ifstat, iftop`
  - Wireshark
- Some more advanced tools
  - `sar`
  - `strace, ltrace`
  - `perf`
  - `dtrace, systemtap, bpftrace`

JDK tools – inspect JARs

- List files in given JAR archive
  - `jar`
  - `unzip`
- Disassemble the `class` file
  - `javap`

JDK tools – JVM processes

- List running JVMs
  - `jps [-l] [-m] [-v]`
- Show system properties and JVM flags
  - `jinfo <PID>`
- Send diagnostic commands
  - `jcmd [<PID>] [<command>]`

JDK tools – memory

- Inspect heap, create heap dump
  - `jmap -heap <PID>`
  - `jmap -histo[:live] <PID>`
  - `jmap -dump[:live],file=<path> <PID>`
- Analyze heap dump
  - `jhat <path>`
  - VisualVM (`jvisualvm`)
- Native memory tracking (NMT)
  - `-XX:NativeMemoryTracking=[summary|detail]`
  - `jcmd <PID> VM.native_memory ...`

JDK tools – stack trace and JVM stats

- Show stack traces of all threads
  - `jstack [-l] <PID>`
- JVM statistics monitoring (GC, JIT compiler, …)
  - `jstat -gcutil <PID> 1s`
  - `jstat -compiler <PID> 1s`
  - `...`

JDK tools – GUI

- `jconsole`
  - Basic statistics (heap, threads, …)
  - JMX console
- VisualVM (`jvisualvm`)
  - Basic statistics (heap, threads, …)
  - CPU profiler, heap profiler, heap dump analyzer
- Java Mission Control (`jmc`)
  - Includes Java Flight Recorder
  - Profiler without safepoint bias

Other Java tools

- Your IDE can offer a debugger, profiler, or decompiler
- JProfiler: https://www.ej-technologies.com/products/jprofiler/overview.html
- YourKit: https://www.yourkit.com/
- Honest Profiler: https://github.com/jvm-profiling-tools/honest-profiler
- Java Decompiler: http://jd.benow.ca/
- Eclipse Memory Analyzer (MAT): https://www.eclipse.org/mat/
- JITWatch: https://github.com/AdoptOpenJDK/jitwatch
- GCViewer: https://github.com/chewiebug/GCViewer

WildFly tools

- `jconsole`
  - `$WF_HOME/bin/jconsole.sh`
  - Includes WildFly management plugin
- JBoss Diagnostic Reporter
  - `$WF_HOME/bin/jdr.sh`
  - Creates a .zip with useful diagnostic info (configuration, runtime state, log files)
- Management API

WildFly Management

WildFly history and overview

- Called JBoss AS before
- Why was AS7 rewritten from scratch?
  - Legacy subsystems
  - Boot time
  - Memory footprint
  - Bad modularity
  - Administration options
  - Not “good enough”

WildFly history and overview

- JBoss AS 7
  - Java EE 6
  - No legacy stuff
  - Better manageability
  - Multi-node management
  - Simplified configuration
  - Modular

WildFly history and overview

- WildFly 8
  - Successor of JBoss AS 7
  - “Even #@*%ing faster”
  - Java EE 7
  - New web server: Undertow
  - Port reduction
  - Management RBAC
  - Audit logging
  - Patching

WildFly history and overview

- WildFly 9
  - HTTP/2 support
  - Undertow as a load balancer
  - Graceful shutdown

- WildFly 10
  - Java 8+
  - ActiveMQ Artemis
  - JavaScript support with hot reloading
  - HA singleton

WildFly history and overview

- WildFly 11
  - New security infrastructure: Elytron
  - Remote EJB/JNDI over HTTP
  - WildFly OpenSSL
- WildFly 12
  - New quarterly release model
  - Partial Java EE 8
  - Improved Java 9 support

WildFly history and overview

- WildFly 13
  - Feature-complete Java EE 8
  - Improved Java 10 support
  - New provisioning infrastructure: Galleon
- WildFly 14
  - Java EE 8 certified
  - New connection pool: Agroal

WildFly architecture

- Core
  - Modularity, management
- Extensions
  - Each provides one or more subsystems
  - All the interesting features
- Clients for management interface
  - CLI
  - Web console
  - Custom – Java API, HTTP API

Core

- JBoss Modules
  - First thing started
  - Modular and concurrent classloading
  - O(1) dependency resolution
  - Module only sees what it imports (depends on)
- JBoss MSC (Modular Service Container)
  - Everything is a service (with lifecycle)
  - Services are deployed on demand and in parallel
- Extensible management layer
  - Mediate access to service container
  - Provides persistent configuration model for the AS

Management

- API and tools
- Requirements:
  - Simple, powerful, stable
  - As few compile-time and runtime dependencies as possible
  - Backwards compatibility
- Solution: detyped API
  - Only a few types: boolean, int, long, double, big int, big decimal, string, byte array, key-value pair, list, object, expression, type
    - Similar to JSON
  - Automatic type conversion
  - JBoss DMR (Dynamic Model Representation)

Management

- Everything operates on DMR
  - Management interface
  - Subsystems internals
- Structured as a tree
  - Attributes
  - Operations
  - Children
  - Metadata (type, constraints, documentation, …)
- Rendered documentation: http://wildscribe.github.io/

CLI

- `$WF_HOME/bin/standalone.sh -c standalone-full-ha.xml`
- Interactive CLI usage
  - `$WF_HOME/bin/jboss-cli.sh -c`
    - Low-level operations
      - `:whoami, :read-resource, :read-attribute, :read-resource-description, :read-operation-description, ...`
    - High-level commands
      - `cd, ls, pwd, read-attribute, echo-dmr, ...`
      - `deploy, undeploy, jms-queue, data-source, ...`
  - `$WF_HOME/bin/jboss-cli.sh -c --gui`

CLI

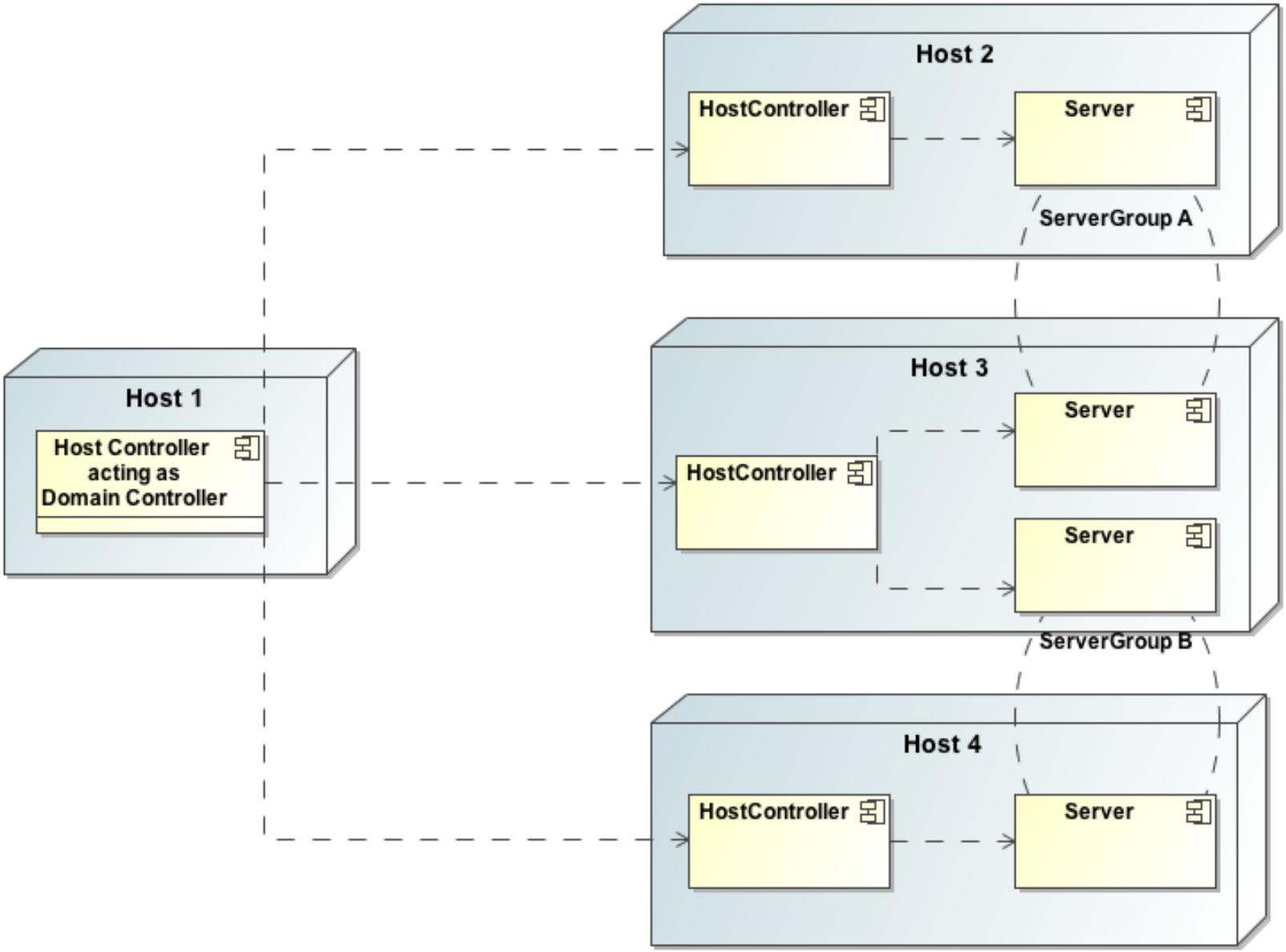
- Non-interactive CLI usage
  - `$WF_HOME/bin/jboss-cli.sh -c command=:whoami`
  - ```
    cat > test.cli <<EOF
    version
    :whoami
    EOF
    ```
  - `$WF_HOME/bin/jboss-cli.sh -c file=test.cli`


    ```

Operating modes

- Standalone server
  - Single JVM, single machine
  - Individual management
  - No lifecycle management
- Managed domain
  - Multiple JVMs, multiple machines
  - Lifecycle managed by process controller (PC)
  - Servers on single machine managed by host controller (HC)
  - Centralized management coordinated by domain controller (DC)
    - DC is actually a plain HC on a selected master node
  - **Difference only in management, not in features**

Managed domain



Managed domain

- Manage multiple servers via single control point
  - Configure a cluster
  - Start/stop nodes in a cluster
  - Deploy application to the entire cluster in one go
  - …
- All configuration centralized in a few files
- All configuration exposed via management API

Managed domain – terms

- **Domain** – set of host controllers and servers
- **Host controller** – miniature WildFly instance that manages servers
- **Server** – WildFly instance that runs applications
- **Server group** – set of servers that are managed as a unit
- **Subsystem** – block of configuration and group of services
- **Profile** – set of subsystems
- **Cluster** – server group with clustering services configured

Setting up a cluster in managed domain

- `$WF_HOME/bin/domain.sh`
- `$WF_HOME/bin/jboss-cli.sh -c`

```
# remove demo servers and server groups
/server-group=main-server-group:stop-servers(blocking=true)
/server-group=other-server-group:stop-servers(blocking=true)

/host=master/server-config=server-one:remove
/host=master/server-config=server-two:remove
/host=master/server-config=server-three:remove

/server-group=main-server-group:remove
/server-group=other-server-group:remove
```


Setting up a cluster in managed domain

```
# create server groups for application servers and load balancers
/server-group=application-servers:add(profile=full-ha,
    socket-binding-group=full-ha-sockets)
/server-group=load-balancers:add(profile=load-balancer,
    socket-binding-group=load-balancer-sockets)

# create 2 application servers and 1 load balancer
/host=master/server-config=app-server-1:add(group=application-servers,
    socket-binding-port-offset=100,auto-start=false)
/host=master/server-config=app-server-2:add(group=application-servers,
    socket-binding-port-offset=200,auto-start=false)
/host=master/server-config=loadbal-1:add(group=load-balancers,auto-start=false)

# start all servers
/server-group=application-servers:start-servers(blocking=true)
/server-group=load-balancers:start-servers(blocking=true)

# deploy a clustered app to all application servers
deploy my-app.war --server-groups=application-servers
```


Role Based Access Control (RBAC)

- Different users have different sets of permissions to read and update parts of the management tree
- Replaces the simple permission scheme used in JBoss AS 7, where authenticated users have all permissions

- Core concepts:
  - **Role** – named set of permissions
  - **Mapping** users and groups to roles

RBAC roles

- Not given permissions for "security sensitive" items:
  - **Monitor** – read only
  - **Operator** – Monitor + modify runtime state
  - **Maintainer** – Operator + modify persistent config
  - **Deployer** – Operator + modify "application resources"
- Given permissions for "security sensitive" items:
  - **Administrator** – all permissions, but cannot read or write resources related to the administrative audit logging system
  - **Auditor** – can read anything, can modify the resources related to the administrative audit logging system
  - **SuperUser** – all permissions

Java API

Maven artifact `org.wildfly.core:wildfly-controller-client`

```
try (ModelControllerClient client =
        ModelControllerClient.Factory.create("localhost", 9990)) {

    ModelNode op = new ModelNode();
    op.get("address").setEmptyList().add("subsystem", "undertow");
    op.get("operation").set("read-resource");
    op.get("recursive").set(true);
    op.get("include-runtime").set(true);

    ModelNode result = client.execute(op);
    System.out.println(result.get("result", "instance-id"));
}
```


Lab – overview

- *management-00*: Initial commit, start here
- *management-01*: Read product version
- *management-02*: Read resources recursively, including runtime-only info
- *management-03*: Read resource description of `http-remoting-connector` in the Remoting subsystem
- *management-04*: connect remotely to running WildFly

Lab 01 – Product version

- `ProductVersionApp.java`
- Get product version

- Hint:
  - `:read-attribute(name=product-version)`

Lab 02 – Read resources recursively

- `ResourcesRecursivelyApp.java`
- Get all resources recursively
  - Include runtime-only information
  - Limit recursion

- Hint:
  - `:read-resource(include-runtime=true, recursive-depth=5)`

Lab 03 – Read resource description in Remoting

- `ResourcesDescriptionRemotingSubsystemApp.java`
- Read resource description for `http-remoting-connector` in the Remoting subsystem
  - Read recursively

- Hint:
  - `/subsystem=remoting/http-connector=http-remoting-connector:read-resource-description(recursive=true)`

Lab 04 – Connect remotely

- `RemoteConnectionApp.java`
- Preparation:
  - Add user with username and password
    - `$WF_HOME/bin/add-user.sh`
  - Disable local auth
    - `/core-service=management /security-realm=ManagementRealm /authentication=local:remove()`
- Use the login/password you created
- Verify by running `:whoami`

HTTP API

- http://localhost:9990/management
- HTTP + JSON (not really REST)
- Requires management user (`add-user.sh`) and DIGEST auth
- Use GET for simple reading operations (resources, attributes)
  - `/management?recursive&include-runtime`
  - `/management?operation=attribute&name=product-version`
  - `/management/subsystem/infinispan/cache-container/ejb`
- Use POST, with DMR serialized to JSON as request body, for complex operations

Web Console

http://localhost:9990/console/

- Check environment properties
- Reload server

- Check `ExampleDS` datasource configuration, create `TestDS`
- Create JMS queue in the Messaging subsystem

WildFly In The Clouds

Thorntail

- “Just enough app server”
  - Select what you need (e.g. JAX-RS + CDI)
  - WildFly, MicroProfile, etc.
- Fraction
  - Well-defined collection of capabilities
  - Often maps directly to a WildFly subsystem
- Uberjar
  - Self-contained, executable JAR
- Requires JDK 8+ and Maven
- https://thorntail.io/

Thorntail

```
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>io.thorntail</groupId>
            <artifactId>bom</artifactId>
            <version>${version.io.thorntail}</version>
            <scope>import</scope>
            <type>pom</type>
        </dependency>
    </dependencies>
</dependencyManagement>

<dependencies>
    <dependency>
        <groupId>io.thorntail</groupId>
        <artifactId>jaxrs</artifactId>
    </dependency>
</dependencies>
```


Thorntail

```
<build>
    <plugins>
        <plugin>
            <groupId>io.thorntail</groupId>
            <artifactId>thorntail-maven-plugin</artifactId>
            <version>${version.io.thorntail}</version>
            <executions>
                <execution>
                    <goals>
                        <goal>package</goal>
                    </goals>
                </execution>
            </executions>
        </plugin>
    </plugins>
</build>
```


OpenShift

- Cloud
  - IaaS, PaaS, SaaS etc.
  - “Pet vs. cattle”
- Containers
  - Docker
- Kubernetes
  - https://kubernetes.io/
- OpenShift
  - https://www.okd.io/
  - https://www.openshift.com/

Monitoring in this brave new world

- Health checks
- Application metrics
  - Prometheus (OpenMetrics), Grafana
- Distributed tracing
  - OpenTracing, Jaeger
- Aggregated logging
  - Elastic Search, Logstash / Fluentd, Kibana

Thank you for your attention.

Questions?

WildFly

Monitoring and Management

Ladislav Thon

Senior Software Engineer, Red Hat

Advanced Java EE Lab @ FEL ČVUT

Nov 2018

Agenda

- Monitoring
  - System tools, JDK tools, WildFly tools
- Management
  - WildFly history and overview
  - Architecture, management model, RBAC
  - CLI, Web console, Java API, HTTP API
- WildFly in the clouds
  - Thorntail
  - OpenShift


redhat.
WildFly Monitoring

Monitoring – motivation

You are using WildFly, so bright future lies ahead…

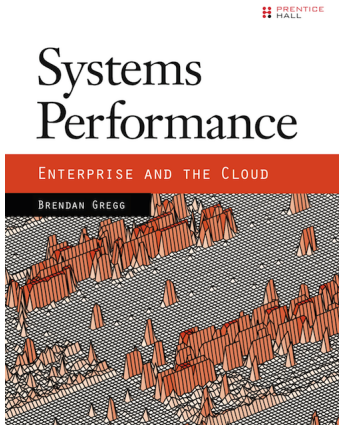
Remember Murphy's law!

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

System-level monitoring (memory, disk, network, …)
Application-level monitoring
- JVM
- WildFly
- deployed app

System tools

- OS info
- Processes
- Memory
- Disk
- Network



Everything we'll show is for Linux.

On Windows, Sysinternals are good:
https://docs.microsoft.com/en-us/sysinternals/

System tools – OS info, processes

- Gather basic info about OS and hardware
  - `cat /etc/os-release, uname -a, dmesg`
  - `cat /proc/cpuinfo, cat /proc/meminfo`
  - `lstopo, lshw, lscpu, lsmem, lspci, lsusb`
- Monitoring and managing running processes
  - `top, htop, ps aux, pstree -a -T`
  - `kill [-9] <PID>`
  - `pgrep, pkill`
  - `pidstat, mpstat`

Start a WildFly standalone server first.

System tools – memory, disk

- Monitoring memory usage
  - `free -m`
  - **`vmstat`** `-w 1`
- Monitoring disk usage
  - `df -h, du -h`
  - `mount, findmnt, lsblk [-f]`
  - `iostat, iotop`

System tools – network, others

- Monitoring network usage
  - `netstat, ss`
  - `ifstat, iftop`
  - Wireshark
- Some more advanced tools
  - `sar`
  - `strace, ltrace`
  - `perf`
  - `dtrace, systemtap, bpftrace`

JDK tools – inspect JARs

- List files in given JAR archive
  - `jar`
  - `unzip`
- Disassemble the `class` file
  - `javap`

jar tf jboss-modules.jar
jar tfv …
unzip -l jboss-modules.jar

javap -cp jboss-modules.jar org.jboss.modules.Module
javap -private -c …

JDK tools – JVM processes

- List running JVMs
  - `jps [-l] [-m] [-v]`
- Show system properties and JVM flags
  - `jinfo <PID>`
- Send diagnostic commands
  - `jcmd [<PID>] [<command>]`

```
PID=$(jps -l | grep jboss-modules.jar | cut -d ' ' -f 1)

jinfo $PID

jcmd
jcmd $PID help
jcmd $PID VM.version
```

JDK tools – memory

- Inspect heap, create heap dump
  - `jmap -heap <PID>`
  - `jmap -histo[:live] <PID>`
  - `jmap -dump[:live],file=<path> <PID>`
- Analyze heap dump
  - `jhat <path>`
  - VisualVM (`jvisualvm`)
- Native memory tracking (NMT)
  - `-XX:NativeMemoryTracking=[summary|detail]`
  - `jcmd <PID> VM.native_memory ...`

jmap -heap $PID
jmap -histo:live $PID
jmap -clstats $PID
jmap -dump:live,file=mydump.hprof $PID

edit $WF_HOME/bin/standalone.conf to include NMT
restart WildFly (and set $PID again!)
jcmd $PID VM.native_memory

Don’t use jhat, it’s a waste of time.
Eclipse MAT is best.

JDK tools – stack trace and JVM stats

- Show stack traces of all threads
  - `jstack [-l] <PID>`
- JVM statistics monitoring (GC, JIT compiler, …)
  - `jstat -gcutil <PID> 1s`
  - `jstat -compiler <PID> 1s`
  - `...`

jstack -l $PID
kill -3 $PID

jstat -gcutil -t $PID 1s
jstat -gccause -t $PID 1s

jstat -compiler -t $PID 1s
jstat -printcompilation -t $PID 1s

JDK tools – GUI

- `jconsole`
  - Basic statistics (heap, threads, …)
  - JMX console
- VisualVM (`jvisualvm`)
  - Basic statistics (heap, threads, …)
  - CPU profiler, heap profiler, heap dump analyzer
- Java Mission Control (`jmc`)
  - Includes Java Flight Recorder
  - Profiler without safepoint bias

Other Java tools

- Your IDE can offer a debugger, profiler, or decompiler
- JProfiler: https://www.ej-technologies.com/products/jprofiler/overview.html
- YourKit: https://www.yourkit.com/
- Honest Profiler: https://github.com/jvm-profiling-tools/honest-profiler
- Java Decompiler: http://jd.benow.ca/
- Eclipse Memory Analyzer (MAT): https://www.eclipse.org/mat/
- JITWatch: https://github.com/AdoptOpenJDK/jitwatch
- GCViewer: https://github.com/chewiebug/GCViewer

WildFly tools

- `jconsole`
  - `$WF_HOME/bin/jconsole.sh`
  - Includes WildFly management plugin
- JBoss Diagnostic Reporter
  - `$WF_HOME/bin/jdr.sh`
  - Creates a .zip with useful diagnostic info (configuration, runtime state, log files)
- Management API


redhat.
WildFly Management

WildFly history and overview

- Called JBoss AS before
- Why was AS7 rewritten from scratch?
  - Legacy subsystems
  - Boot time
  - Memory footprint
  - Bad modularity
  - Administration options
  - Not “good enough”

WildFly history and overview

- JBoss AS 7
  - Java EE 6
  - No legacy stuff
  - Better manageability
  - Multi-node management
  - Simplified configuration
  - Modular

WildFly history and overview

- WildFly 8
  - Successor of JBoss AS 7
  - “Even #@*%ing faster”
  - Java EE 7
  - New web server: Undertow
  - Port reduction
  - Management RBAC
  - Audit logging
  - Patching

WildFly history and overview

- WildFly 9
  - HTTP/2 support
  - Undertow as a load balancer
  - Graceful shutdown
- WildFly 10
  - Java 8+
  - ActiveMQ Artemis
  - JavaScript support with hot reloading
  - HA singleton

WildFly history and overview

- WildFly 11
  - New security infrastructure: Elytron
  - Remote EJB/JNDI over HTTP
  - WildFly OpenSSL
- WildFly 12
  - New quarterly release model
  - Partial Java EE 8
  - Improved Java 9 support

WildFly history and overview

- WildFly 13
  - Feature-complete Java EE 8
  - Improved Java 10 support
  - New provisioning infrastructure: Galleon
- WildFly 14
  - Java EE 8 certified
  - New connection pool: Agroal

WildFly architecture

- Core
  - Modularity, management
- Extensions
  - Each provides one or more subsystems
  - All the interesting features
- Clients for management interface
  - CLI
  - Web console
  - Custom – Java API, HTTP API

Core

- JBoss Modules
  - First thing started
  - Modular and concurrent classloading
  - O(1) dependency resolution
  - Module only sees what it imports (depends on)
- JBoss MSC (Modular Service Container)
  - Everything is a service (with lifecycle)
  - Services are deployed on demand and in parallel
- Extensible management layer
  - Mediate access to service container
  - Provides persistent configuration model for the AS

Management

- API and tools
- Requirements:
  - Simple, powerful, stable
  - As few compile-time and runtime dependencies as possible
  - Backwards compatibility
- Solution: detyped API
  - Only a few types: boolean, int, long, double, big int, big decimal, string, byte array, key-value pair, list, object, expression, type
    - Similar to JSON
  - Automatic type conversion
  - JBoss DMR (Dynamic Model Representation)

Management

- Everything operates on DMR
  - Management interface
  - Subsystems internals
- Structured as a tree
  - Attributes
  - Operations
  - Children
  - Metadata (type, constraints, documentation, …)
- Rendered documentation: http://wildscribe.github.io/

3 kinds of attributes/operations:
- read-only (can even be runtime-only, such as metrics)
- read-write runtime state
- read-write persistent configuration

CLI

- `$WF_HOME/bin/standalone.sh -c standalone-full-ha.xml`
- Interactive CLI usage
  - `$WF_HOME/bin/jboss-cli.sh -c`
    - Low-level operations
      - `:whoami, :read-resource, :read-attribute, :read-resource-description, :read-operation-description, ...`
    - High-level commands
      - `cd, ls, pwd, read-attribute, echo-dmr, ...`
      - `deploy, undeploy, jms-queue, data-source, ...`
  - `$WF_HOME/bin/jboss-cli.sh -c --gui`

```
:whoami
:read-resource(recursive=true)
:read-resource(include-runtime=true)

:read-attribute(name=product-version)
read-attribute product-version
read-attribute product-version --verbose

:read-operation-description(name=whoami)
read-operation whoami

batch
discard-batch / run-batch
```

CLI

- Non-interactive CLI usage
  - `$WF_HOME/bin/jboss-cli.sh -c command=:whoami`
  - ```
    cat > test.cli <<EOF
    version
    :whoami
    EOF
    ```
  - `$WF_HOME/bin/jboss-cli.sh -c file=test.cli`


    ```

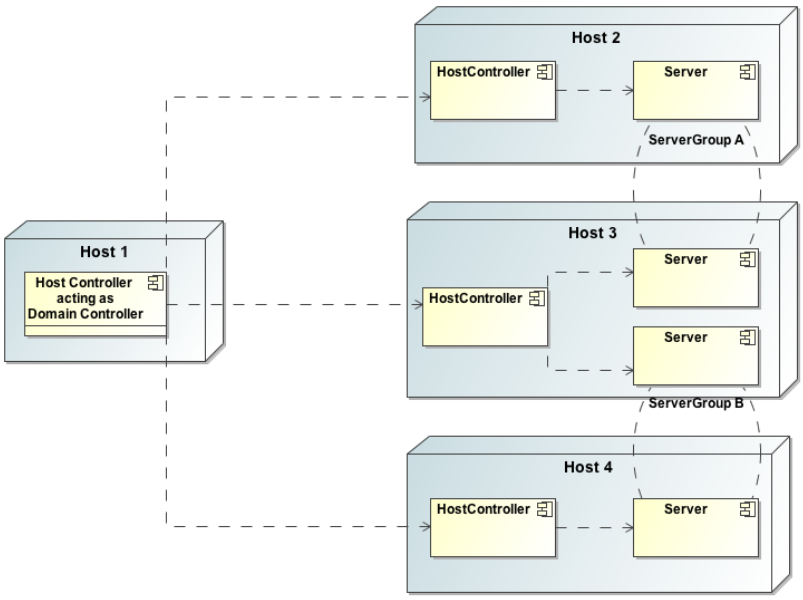
Operating modes

- Standalone server
  - Single JVM, single machine
  - Individual management
  - No lifecycle management
- Managed domain
  - Multiple JVMs, multiple machines
  - Lifecycle managed by process controller (PC)
  - Servers on single machine managed by host controller (HC)
  - Centralized management coordinated by domain controller (DC)
    - DC is actually a plain HC on a selected master node
  - **Difference only in management, not in features**

```
unzip wildfly-14.0.1.Final.zip
cp tinyClusteredWebapp.war
   wildfly-14.0.1.Final/standalone/deployments/
mv wildfly-14.0.1.Final wildfly-14.0.1.Final-node1
cp -a wildfly-14.0.1.Final-node1
   wildfly-14.0.1.Final-node2
./wildfly-14.0.1.Final-node1/bin/standalone.sh
   -c standalone-ha.xml -Djboss.node.name=node1
./wildfly-14.0.1.Final-node2/bin/standalone.sh
   -c standalone-ha.xml -Djboss.node.name=node2
   -Djboss.socket.binding.port-offset=100
http localhost:8[01]80 [Cookie:JSESSIONID=...]

unzip wildfly-14.0.1.Final.zip
./wildfly-14.0.1.Final/bin/domain.sh
pstree -a -T
```

Managed domain



Blue boxes correspond to machines.

Yellow boxes correspond to JVM processes.

The picture doesn’t show process controllers, but they are not crucial for understanding the concept.

Managed domain

- Manage multiple servers via single control point
  - Configure a cluster
  - Start/stop nodes in a cluster
  - Deploy application to the entire cluster in one go
  - …
- All configuration centralized in a few files
- All configuration exposed via management API

Managed domain – terms

- **Domain** – set of host controllers and servers
- **Host controller** – miniature WildFly instance that manages servers
- **Server** – WildFly instance that runs applications
- **Server group** – set of servers that are managed as a unit
- **Subsystem** – block of configuration and group of services
- **Profile** – set of subsystems
- **Cluster** – server group with clustering services configured

Setting up a cluster in managed domain

- `$WF_HOME/bin/domain.sh`
- `$WF_HOME/bin/jboss-cli.sh -c`

```
# remove demo servers and server groups
/server-group=main-server-group:stop-servers(blocking=true)
/server-group=other-server-group:stop-servers(blocking=true)

/host=master/server-config=server-one:remove
/host=master/server-config=server-two:remove
/host=master/server-config=server-three:remove

/server-group=main-server-group:remove
/server-group=other-server-group:remove
```


Setting up a cluster in managed domain

```
# create server groups for application servers and load balancers
/server-group=application-servers:add(profile=full-ha,
    socket-binding-group=full-ha-sockets)
/server-group=load-balancers:add(profile=load-balancer,
    socket-binding-group=load-balancer-sockets)

# create 2 application servers and 1 load balancer
/host=master/server-config=app-server-1:add(group=application-servers,
    socket-binding-port-offset=100,auto-start=false)
/host=master/server-config=app-server-2:add(group=application-servers,
    socket-binding-port-offset=200,auto-start=false)
/host=master/server-config=loadbal-1:add(group=load-balancers,auto-start=false)

# start all servers
/server-group=application-servers:start-servers(blocking=true)
/server-group=load-balancers:start-servers(blocking=true)

# deploy a clustered app to all application servers
deploy my-app.war --server-groups=application-servers
```


Role Based Access Control (RBAC)

- Different users have different sets of permissions to read and update parts of the management tree
- Replaces the simple permission scheme used in JBoss AS 7, where authenticated users have all permissions

- Core concepts:
  - **Role** – named set of permissions
  - **Mapping** users and groups to roles

RBAC roles

- Not given permissions for "security sensitive" items:
  - **Monitor** – read only
  - **Operator** – Monitor + modify runtime state
  - **Maintainer** – Operator + modify persistent config
  - **Deployer** – Operator + modify "application resources"
- Given permissions for "security sensitive" items:
  - **Administrator** – all permissions, but cannot read or write resources related to the administrative audit logging system
  - **Auditor** – can read anything, can modify the resources related to the administrative audit logging system
  - **SuperUser** – all permissions

Java API

Maven artifact `org.wildfly.core:wildfly-controller-client`

```
try (ModelControllerClient client =
        ModelControllerClient.Factory.create("localhost", 9990)) {

    ModelNode op = new ModelNode();
    op.get("address").setEmptyList().add("subsystem", "undertow");
    op.get("operation").set("read-resource");
    op.get("recursive").set(true);
    op.get("include-runtime").set(true);

    ModelNode result = client.execute(op);
    System.out.println(result.get("result", "instance-id"));
}
```


Lab – overview

- *management-00*: Initial commit, start here
- *management-01*: Read product version
- *management-02*: Read resources recursively, including runtime-only info
- *management-03*: Read resource description of `http-remoting-connector` in the Remoting subsystem
- *management-04*: connect remotely to running WildFly

Lab 01 – Product version

- `ProductVersionApp.java`
- Get product version

- Hint:
  - `:read-attribute(name=product-version)`

Lab 02 – Read resources recursively

- `ResourcesRecursivelyApp.java`
- Get all resources recursively
  - Include runtime-only information
  - Limit recursion

- Hint:
  - `:read-resource(include-runtime=true, recursive-depth=5)`

Lab 03 – Read resource description in Remoting

- `ResourcesDescriptionRemotingSubsystemApp.java`
- Read resource description for `http-remoting-connector` in the Remoting subsystem
  - Read recursively
- Hint:
  - `/subsystem=remoting /http-connector=http-remoting-connector :read-resource-description (recursive=true)`

Lab 04 – Connect remotely

- `RemoteConnectionApp.java`
- Preparation:
  - Add user with username and password
    - `$WF_HOME/bin/add-user.sh`
  - Disable local auth
    - `/core-service=management /security-realm=ManagementRealm /authentication=local:remove()`
- Use the login/password you created
- Verify by running `:whoami`

$WF_HOME/standalone/configuration/mgmt-users.properties

Local auth needs to be disabled for this excersise, because when playing locally, it will always succeed.

HTTP API

- http://localhost:9990/management
- HTTP + JSON (not really REST)
- Requires management user (`add-user.sh`) and DIGEST auth
- Use GET for simple reading operations (resources, attributes)
  - `/management?recursive&include-runtime`
  - `/management?operation=attribute&name=product-version`
  - `/management/subsystem/infinispan/cache-container/ejb`
- Use POST, with DMR serialized to JSON as request body, for complex operations

Web Console

http://localhost:9990/console/

- Check environment properties
- Reload server

- Check `ExampleDS` datasource configuration, create `TestDS`
- Create JMS queue in the Messaging subsystem

JMS queue: Messaging -> Destinations
JNDI name: java:jboss/exported/jms/queue/test


redhat.
WildFly In The Clouds

Thorntail

- “Just enough app server”
  - Select what you need (e.g. JAX-RS + CDI)
  - WildFly, MicroProfile, etc.
- Fraction
  - Well-defined collection of capabilities
  - Often maps directly to a WildFly subsystem
- Uberjar
  - Self-contained, executable JAR
- Requires JDK 8+ and Maven
- https://thorntail.io/

Thorntail

```
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>io.thorntail</groupId>
            <artifactId>bom</artifactId>
            <version>${version.io.thorntail}</version>
            <scope>import</scope>
            <type>pom</type>
        </dependency>
    </dependencies>
</dependencyManagement>

<dependencies>
    <dependency>
        <groupId>io.thorntail</groupId>
        <artifactId>jaxrs</artifactId>
    </dependency>
</dependencies>
```


Thorntail

```
<build>
    <plugins>
        <plugin>
            <groupId>io.thorntail</groupId>
            <artifactId>thorntail-maven-plugin</artifactId>
            <version>${version.io.thorntail}</version>
            <executions>
                <execution>
                    <goals>
                        <goal>package</goal>
                    </goals>
                </execution>
            </executions>
        </plugin>
    </plugins>
</build>
```


OpenShift

- Cloud
  - IaaS, PaaS, SaaS etc.
  - “Pet vs. cattle”
- Containers
  - Docker
- Kubernetes
  - https://kubernetes.io/
- OpenShift
  - https://www.okd.io/
  - https://www.openshift.com/

Monitoring in this brave new world

- Health checks
- Application metrics
  - Prometheus (OpenMetrics), Grafana
- Distributed tracing
  - OpenTracing, Jaeger
- Aggregated logging
  - Elastic Search, Logstash / Fluentd, Kibana

Thank you for your attention.

Questions?