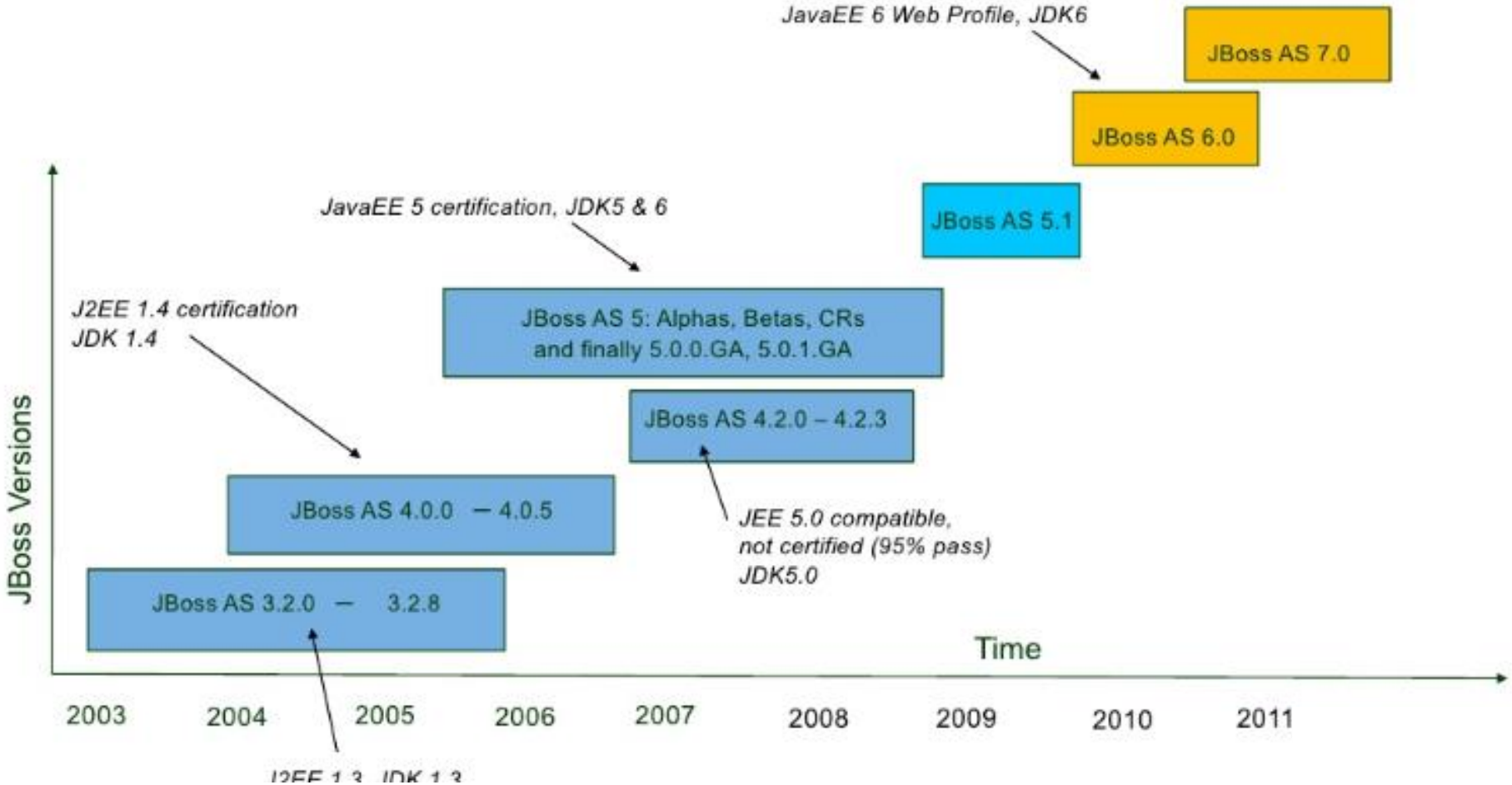
JBoss Application Server 7

JBoss AS Timeline



Motivations For AS7

- Improve Usability
- Increase Manageability
- Simplify Configurations
- Highly Performant - GO FAST!

Key Features of AS7

Key Features of AS

- **Fast** and **Lightweight**
- Supports **domain** (multi-node) management
- Multiple consistent **management interfaces**
  - CLI, Java API, HTTP API, Console
- Unified, **user-focused configuration**
- **Modular**
  - Only APIs, no AS implementation exposure
  - **True isolation**

Two Operational Modes

- **Standalone**
  - Traditional JBoss single JVM server
  - Management facilities IN-VM
- **Domain**
  - Multi-JVM, multi-server model
  - Management coordinated by *Domain Controller Process*
  - Multiple server instances (JVMs) per Host
  - Full lifecycle managed by *Process Controller*

Standalone Mode

- Standalone is a ***single AS process*** for use in development, where the additional management functionality is not required
- Provides a ***similar development experience to previous versions*** of the AS, allowing for a deployment to be dropped in the deployments folder and automatically deployed
- Can still be managed by the **same tools and API's** as domain mode

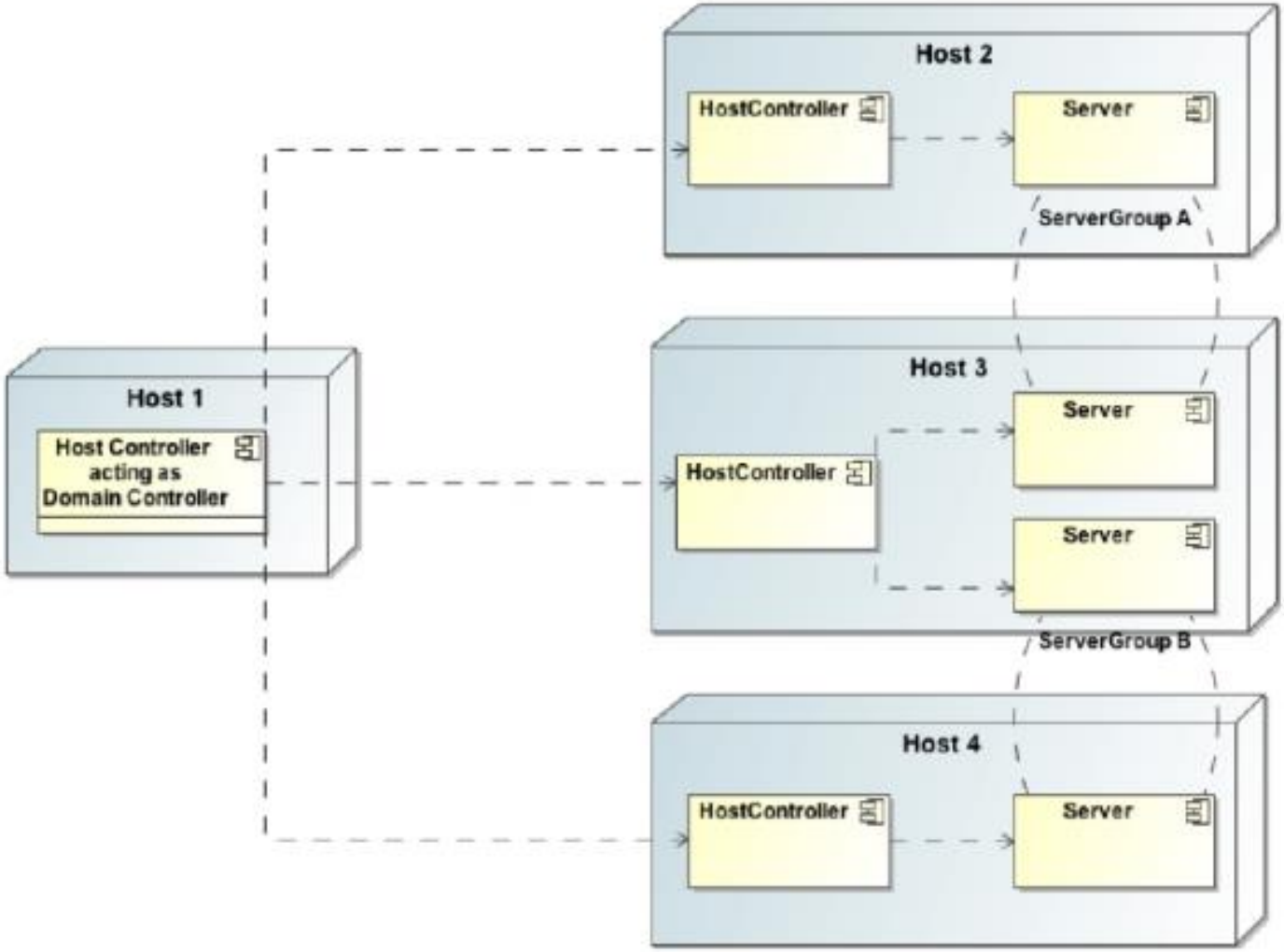
Domain Mode

- Easy management of **multiple AS instances**
- Managed from a **single point**, all have access to the same domain configuration
- Allows for management and configuration updates to be pushed out to all servers
- Domain Mode has **three separate processes**:
  - Process Controller
  - Host Controller
  - Server Instance

Domain Mode Processes

- **Process Controller**
  - Responsible for managing and starting / restarting processes
  - Extremely simple, not much that can go wrong
- **Host Controller**
  - One host controller is the domain controller, the rest are slaves
  - Domain controller is responsible for pushing out configuration changes over the domain

Domain Mode


Host 2
HostController
Server
ServerGroup A
Host 1
Host Controller acting as Domain Controller
Host 3
HostController
Server
Server
ServerGroup B
Host 4
HostController
Server

My File's Have Changed?

- Where are they?

File Layout

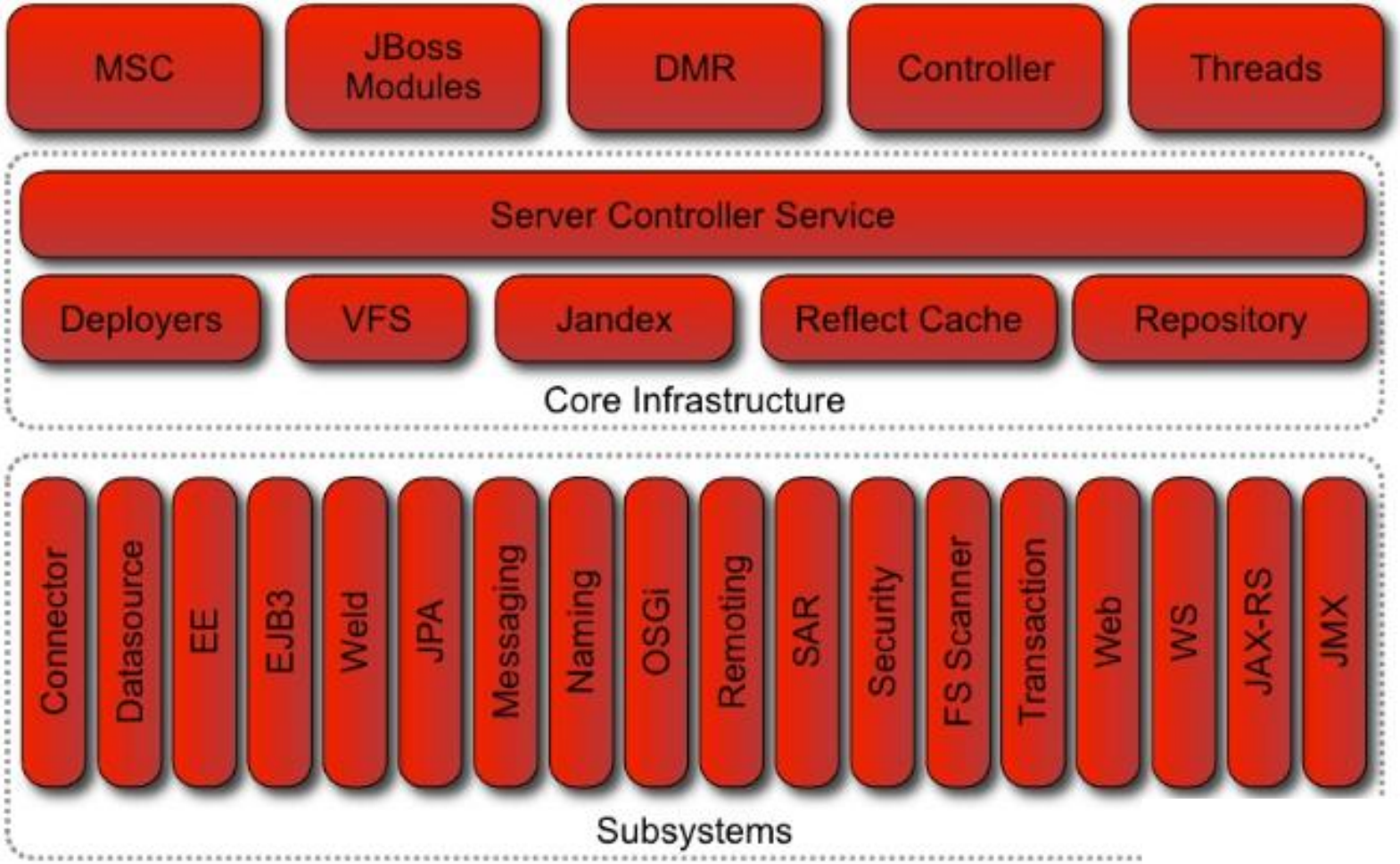
- jboss-7.0.0.Beta3
  - bin
    - standalone.conf — Standalone Mode JVM Parameters
    - standalone.sh — Standalone Mode
    - domain.sh — Domain Mode
    - jboss-admin.sh — Command Line Interface
  - modules — Static JBoss Module Definitions
  - standalone
    - configuration
      - standalone.xml — Standalone Unified Configuration
    - deployments — File System Deployment
    - logs
    - data — Internal Data (includes repository)

File Layout - Domain

- jboss-7.0.0.Beta3
  - domain
    - configuration
      - domain.xml — Domain Wide Unified Configuration
      - host.xml — Host Controller Configuration
    - servers
      - server-one — Server "One" JVM instance data
        - logs
        - data
      - server-two — Server "Two" JVM instance data
        - logs
        - data

Architecture

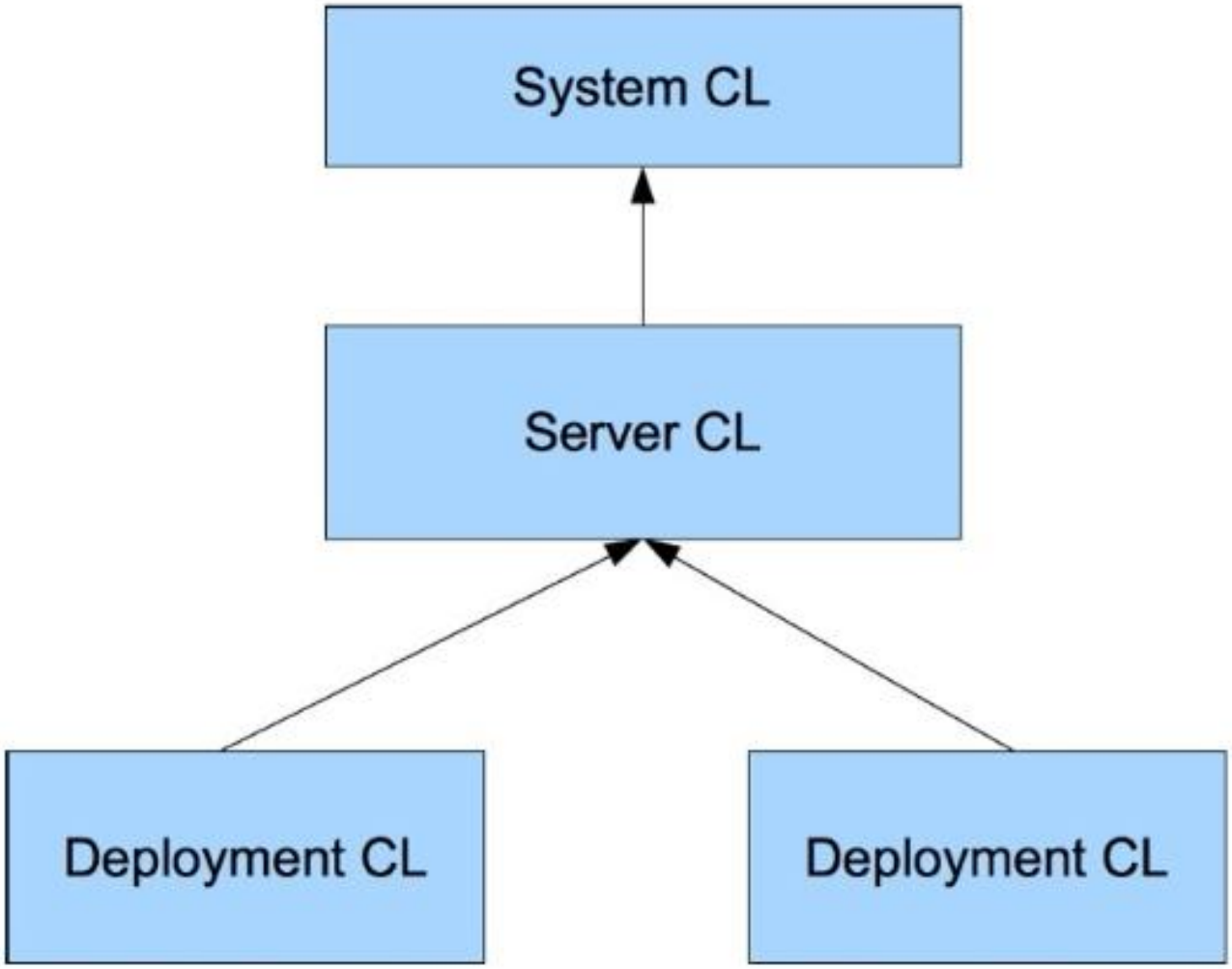
JBoss AS 7 Architecture



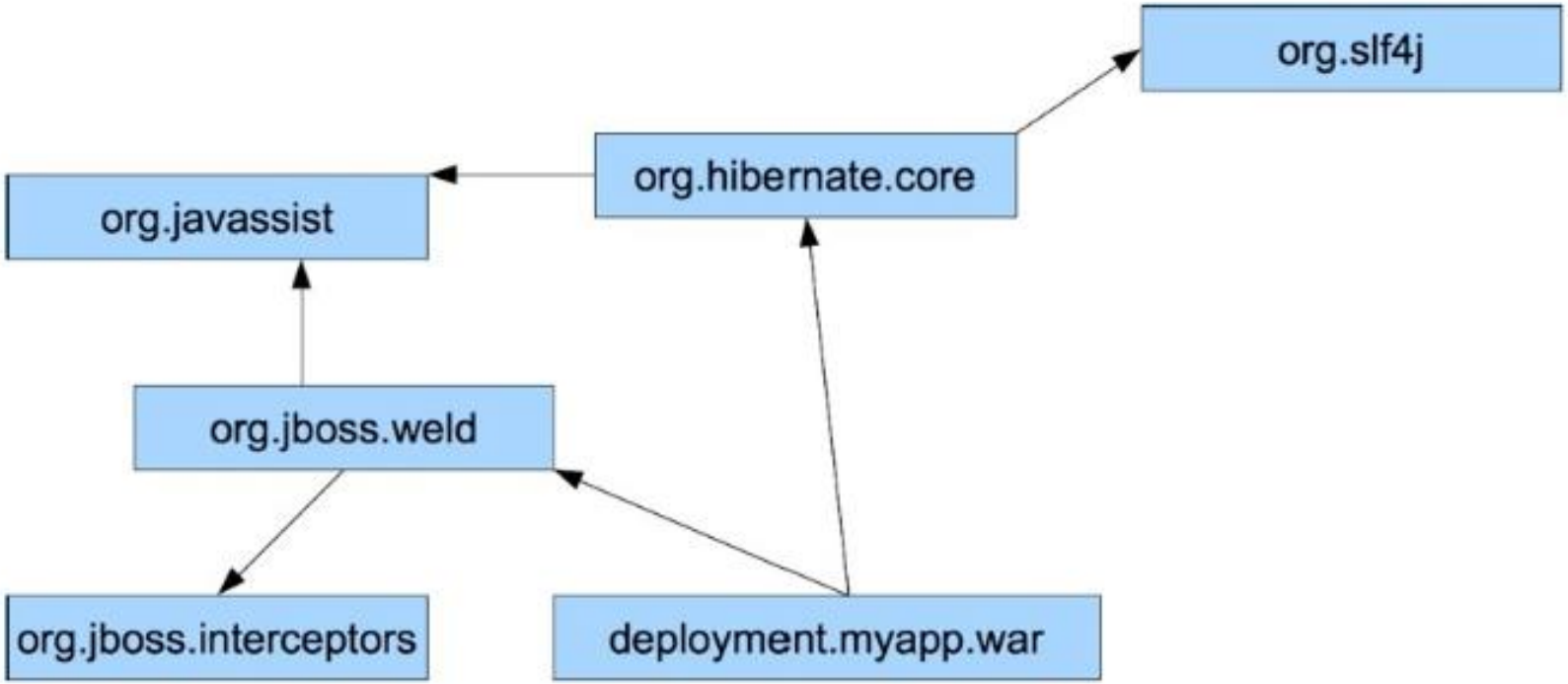
Classloading

JBoss Modules

Hierarchical CL



Modular CL


org.slf4j
org.hibernate.core
org.javassist
org.jboss.weld
org.jboss.interceptors
deployment.myapp.war

JBoss Modules

JBoss Modules

- Provides extremely fast modular class loading
  - O(1) Dependency resolution
  - Concurrent CL (lockless in most VMs)
- "Pure" modular class loading
  - Modules only see what they import (includes JDK classes!)
- External module definitions
  - Don't have to break open the JAR
- Dynamic
  - Modules can be redefined

JBoss Modules

- Provides extremely fast modular class loading
  - O(1) Dependency resolution
  - Concurrent CL (lockless in most VMs)
- "Pure" modular class loading
  - Modules only see what they import (includes JDK classes!)
- External module definitions
  - Don't have to break open the JAR
- Dynamic
  - Modules can be redefined
- Extensible
  - JBoss OSGi implemented on modules

modules.xml

```
<module xmlns="urn:jboss:module:1.0" name="org.jboss.weld.api">

   <resources>
      <resource-root path="weld-api-1.1.Final.jar"/>
   </resources>

   <dependencies>
      <module name="javax.api"/>
      <module name="javax.enterprise.api"/>
      <module name="javax.inject.api"/>
      <module name="javax.persistence.api"/>
      <module name="javax.interceptor.api"/>
      <module name="javax.servlet.api"/>
   </dependencies>
</module>
```

Jar that provides the module classes

Other modules that this module can see

User Deployments

- User deployments are modules too
- Sets up dependencies on some modules automatically (e.g. JPA, Hibernate, WebServices)
- The user can also set up their own dependencies on app server modules

User Deployment Details

- Each sub-deployment in an ear is *it's own module*
- Sub-deployments in an EAR *do not have access* to other sub-deployments by defaUlt
- Allows for individual ejb-jar's to have dependencies on different versions of classes
- Also provide an *relaxed* isolation mode, which automatically set up dependencies between all the sub deployments in the ear
- Dependencies can be set up using the manifest, a custom deployment descriptor, or on a global level

JBoss Modular Service Controller (MSC)

Modular Service Container

- Small, lightweight & efficient
- Highly concurrent & scalable state machine
- Only two non-error, non-transition states - stop & start

Services

- In AS7 almost everything is a service
- Services are objects that can be **started** and **stopped**
- Services can have **dependencies** on other services
- When all a services dependencies are satisfied it will attempt to start
- If a dependency going to be stopped, then MSC will stop all dependent services first
- Services can ***inject*** dependent services

Everything is a Service!

- As mentioned previously almost everything in AS7 is a service, including:
  - EJB's (actually 2+ services)
  - JNDI Bindings
  - Servlets
  - The deployment itself
- Individually shut down and restart, with all dependencies being maintained

JEE

- Still using the same underlying projects, but with completely new integration code.
- Boot process has been *highly optimized*
- Annotation scanning is done by scanning the deployments bytecode, preventing expensive class loading unless it is absolutely necessary
- Services start asynchronously where possible (e.g. Weld and Hibernate can both be starting at the same time)

Dynamic Model Representation

(DMR)

Management - DMR

- **Central De-typed Management API**
  - All management operations operate with/on DMR
  - Backwards compatible!
- Can be used to control a single standalone server or an entire domain
- De-types (i.e. string based) API uses a small set of Java classes
- Various transports (Java Remoting, JSON over HTTP)
- All management interfaces are based on this API

Dynamic Model Representation (DMR)

- **Central De-typed Management API**
  - All management operations operate with/on DMR
  - Backwards compatible!
- **Represents simple and complex types**
  - `int, long, big int, double, big dec, boolean, string, bytes, list, object, property, expression`
- **Auto-converts like dynamic languages**
- **Self describing**
- **Convertible to/from JSON**
- **Also has a defined binary protocol (optionally b64)**

Management

- **ONE configuration file**
  - `standalone.xml / domain.xml`
- Management API that allows for **persistent changes** to the configuration
- Management API **can manage all servers** in the domain
- Management **console** to provide user friendly management in a web browser
- **Command line** tool for use in scripts

Management via Configuration

```
<bean name="TransactionManager"
class="com.arjuna.ats.jbossatx.jta.TransactionManagerService">
        <annotation>@org.jboss.aop.microcontainer.aspects.jmx.JMX
(name="jboss:service=TransactionManager",
exposedInterface=com.arjuna.ats.jbossatx.jta.TransactionManagerServiceMBean.class,
registerDirectly=true)</annotation>
        <annotation>@org.jboss.managed.api.annotation.ManagementObject
(name="TransactionManager",componentType=@org.jboss.managed.api.annotation.ManagementCompone
nt(type = "MCBean", subtype =
"JTA"),targetInterface=com.arjuna.ats.jbossatx.jta.TransactionManagerServiceMBean.class)
        </annotation>

        <property name="transactionTimeout">300</property>
        <property name="objectStoreDir">${jboss.server.data.dir}/tx-object-store</property>
```

```
<subsystem xmlns="urn:jboss:domain:transactions:1.0">
            <recovery-environment socket-binding="txn-recovery-environment"
                                  status-socket-binding="txn-status-manager"/>
            <core-environment socket-binding="txn-socket-process-id"/>
</subsystem>
```

Management via API

- Add a datasource via management API

```
request = new ModelNode();
request.get("address").add("subsystem", "datasources");
request.get("address").add("data-source", “java:/DefaultDS”);
request.get("operation").set("add");
request.get("jndi-name").set(“java:/DefaultDS”);
request.get("enabled").set("true");
request.get("connection-url").set("jdbc:h2:mem:test;DB_CLOSE_DELAY=-1");
request.get("driver-class").set("org.h2.Driver");
request.get("driver-name").set("h2");
request.get("security").get("user-name").set("sa");
request.get("security").get("password").set("sa");
request.get("pool-name").set("DefaultDS");
ModelNode result =
client.execute(OperationBuilder.Factory.create(request).build());
```

Management via Command Line

- Scriptable command line management tool
- Uses the management API internally
- Allows access to high level user friendly commands:

```
create-jms-queue --name testQueue
```

- Also allows direct access to the domain model, giving access to the full functionality of the management API

JBoss AS 7.1.1

Configuring Application Server

Follow the Lab Document