Operations and Monitoring with Spring

Eberhard Wolff

Regional Director and Principal Consultant

SpringSource

A Short History of Spring

- Spring is a platform independent framework for the development of (Enterprise) Java Applications

- Originally started as an easier to use alternative for Java EE development
- In particular compared to EJB
- See "J2EE Development Without EJB"

- But actually there is more to it

Programming Model vs. Infrastructure

- Programming Model:
  A set of APIs to use as a developer
- Infrastructure:
  Something to run your software on

- Spring defines a Programming Model but no Infrastructure
- …you can use Java EE, a simple Servlet container…

- Java EE defines a Programming Model *and* an Infrastructure

Bundling Programming Model and Infrastructure

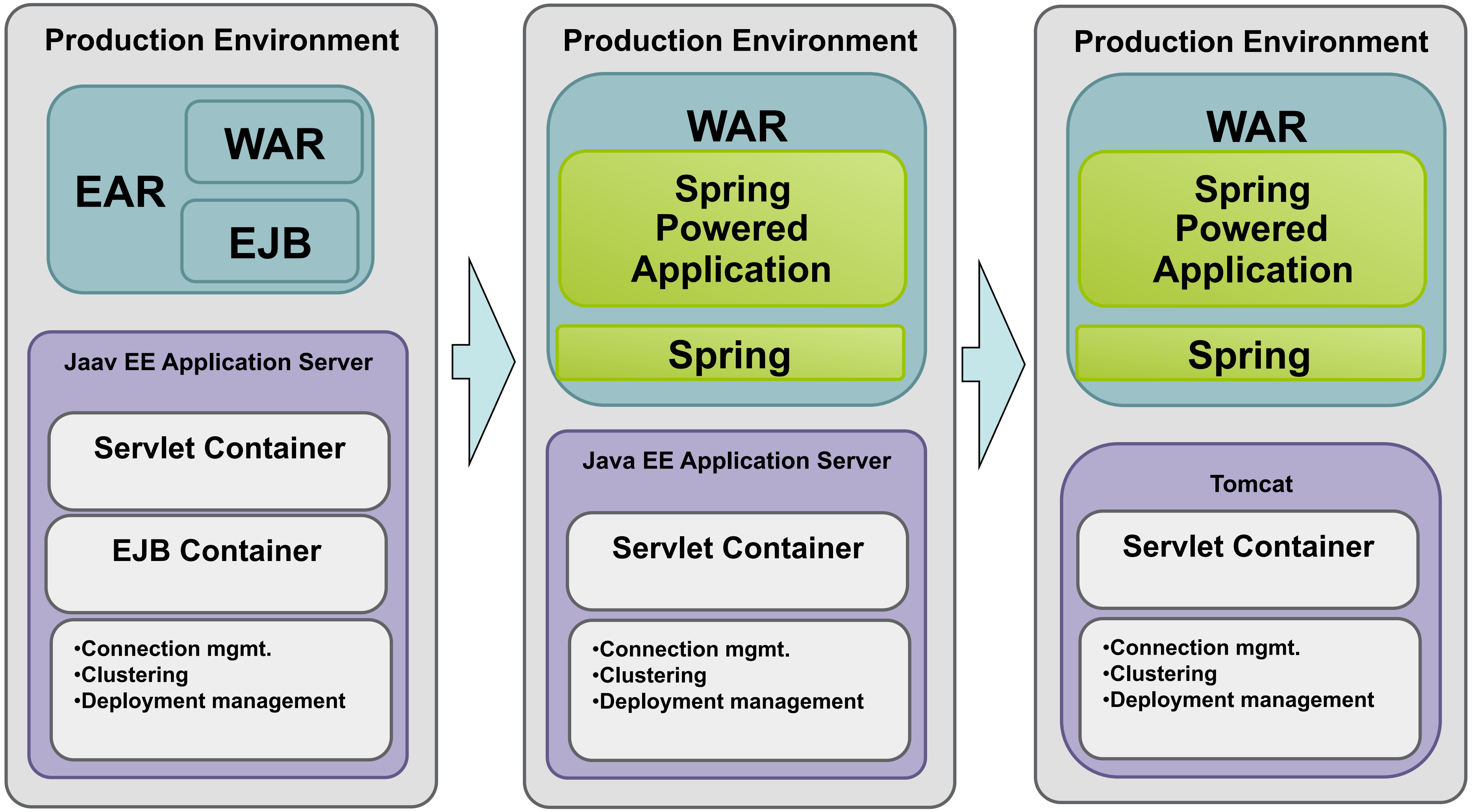
- Bundling leads to some problems

- To upgrade the programming model you need to upgrade the infrastructure
- Operations will not like that idea

- You are limited to a certain infrastructure – what do you do concerning OSGi?
- You only get the benefit from OSGi if you use its non OSGi deployment model

- Users realize how flexible they are using Spring

Spring's Impact on Java EE

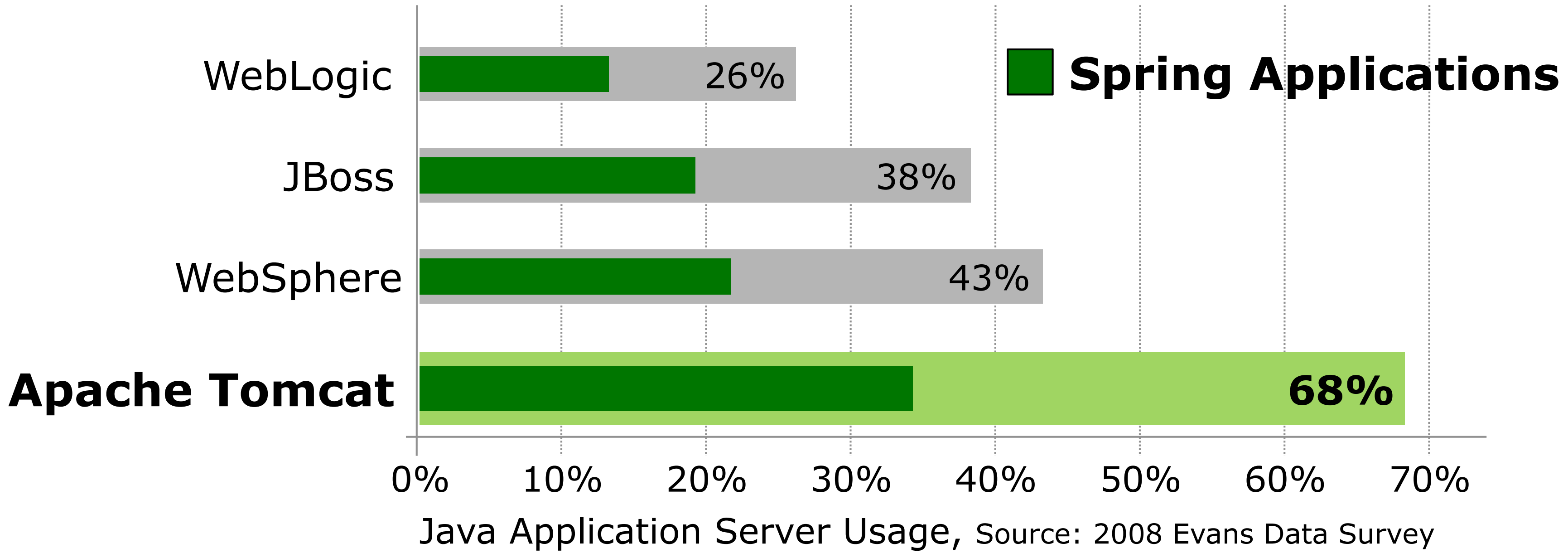


So: What is the advantage of Spring over Java EE 6?

- Type error: Cannot compare infrastructure + programming model to programming model

- But seriously…
- Spring actually exists – Java EE 6 is still being standardized
- You don't need to upgrade your server (many are still on Java EE 5)
- Actually this question is not important to many: They don't run on Java EE anyway

Today’s De Facto Standards
Spring and Tomcat



- Obviously the Java EE platform is often not needed
- A Servlet container is enough
- …and too complex

Is Spring + Servlet container as powerful as Java EE?

- "Classic" features of a Java EE server

Feature	**Spring Solution**
Transactions	No real 2PC but smart solutions for many scenarios
Security	Spring Security is much better
Distribution	Not too important any more Spring Remoting offers even more features
Persistence	Java EE's persistence (JPA) can be used – and several other
Naming	Supported by Dependency Injection
Connection / Resource Pooling	Supported by Servlet Containers and DataSource implementations

So...

- Let's add transaction support (JTA) to Tomcat
- Then we have a full blown Application Server

- But Tomcat is very successful already
- …and Spring offers solutions
  - for O/R mappers
  - for JMS + a DataSource
- JTA seems to be a not too important

- Is there anything else we need to think about?
- Infrastructure is not just for developers!

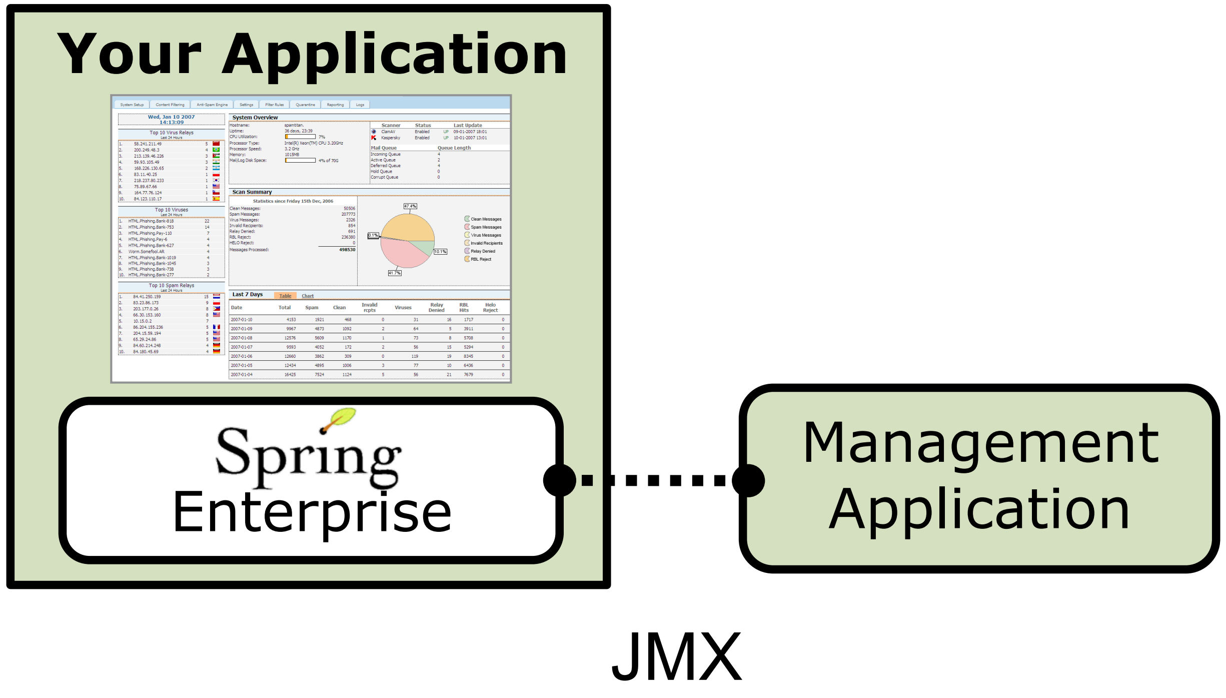
Operations

- Operations cares about
  - monitoring
  - administration
- They should be able to look into application
- Usually there is more than one Application Server – how can you handle larger installations?
- Individual updates of parts of an application are important
- Virtualization / Cloud

Spring Enterprise - Looking into Applications

Spring Enterprise - Looking into Applications

- Certified Spring
- Instrumented Spring
  - Monitor Spring apps across infrastructure
    - JMX flows into Management Application
  - Track app performance
- Just different JARs
- No code changes



Demo: Spring Enterprise

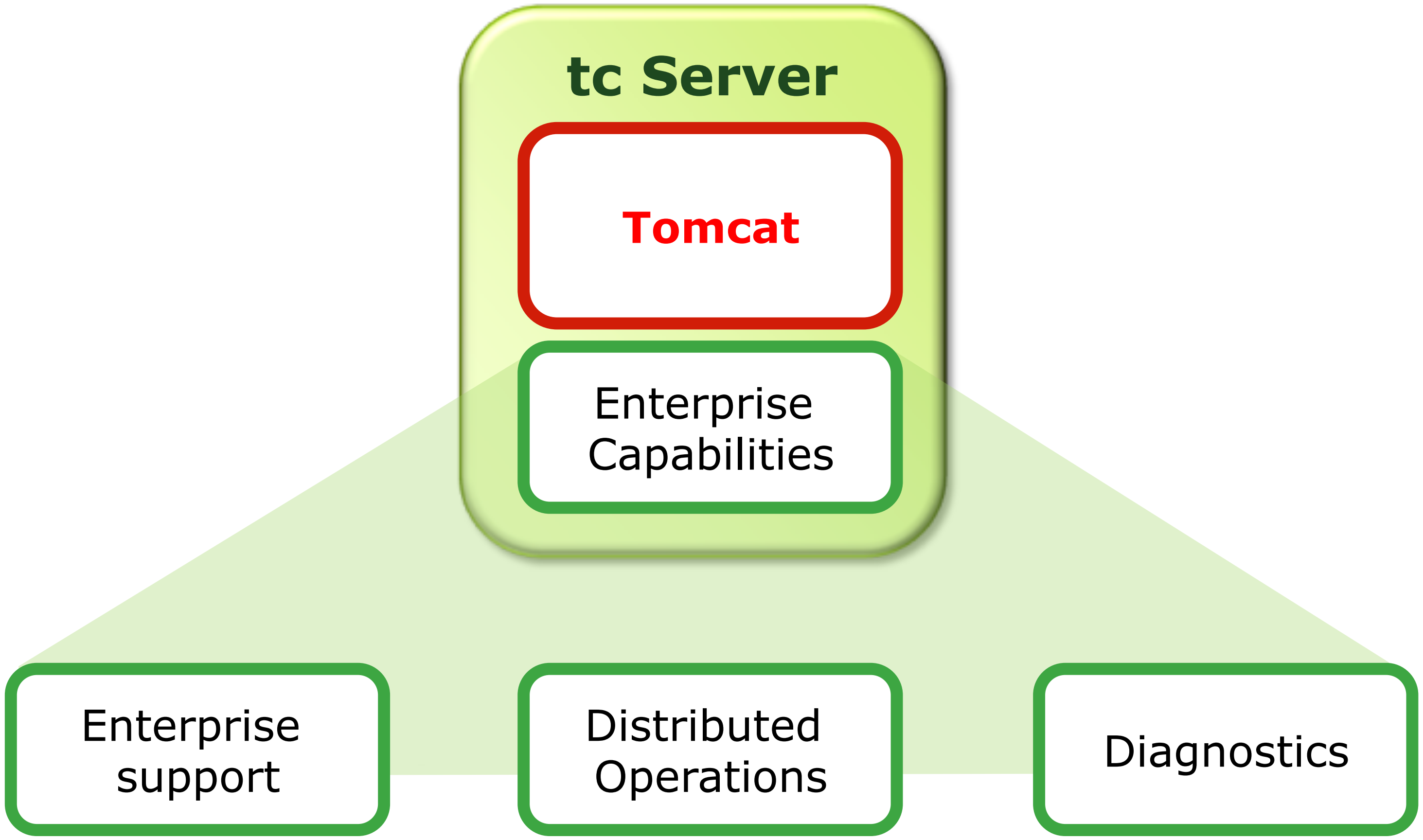
Handling large clusters

Cluster

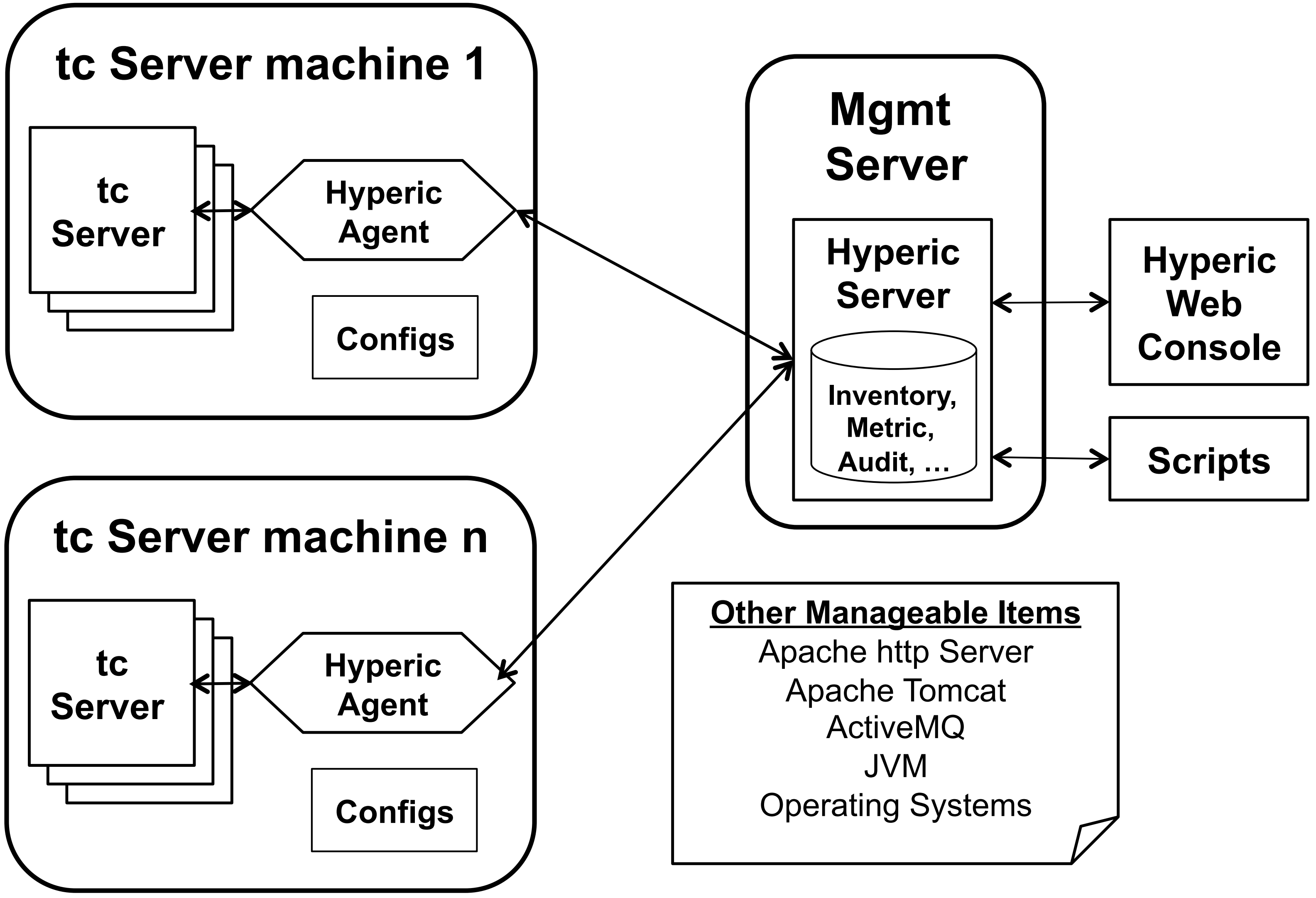
- Usually there is more than one Tomcat server in an installation
- How can you manage them?
- Ideally centralized

SpringSource tc Server

Tomcat you know. Enterprise capabilities you need.



Product Architecture



Other Manageable Items

Apache http Server

Apache Tomcat

ActiveMQ

JVM

Operating Systems

tc Server

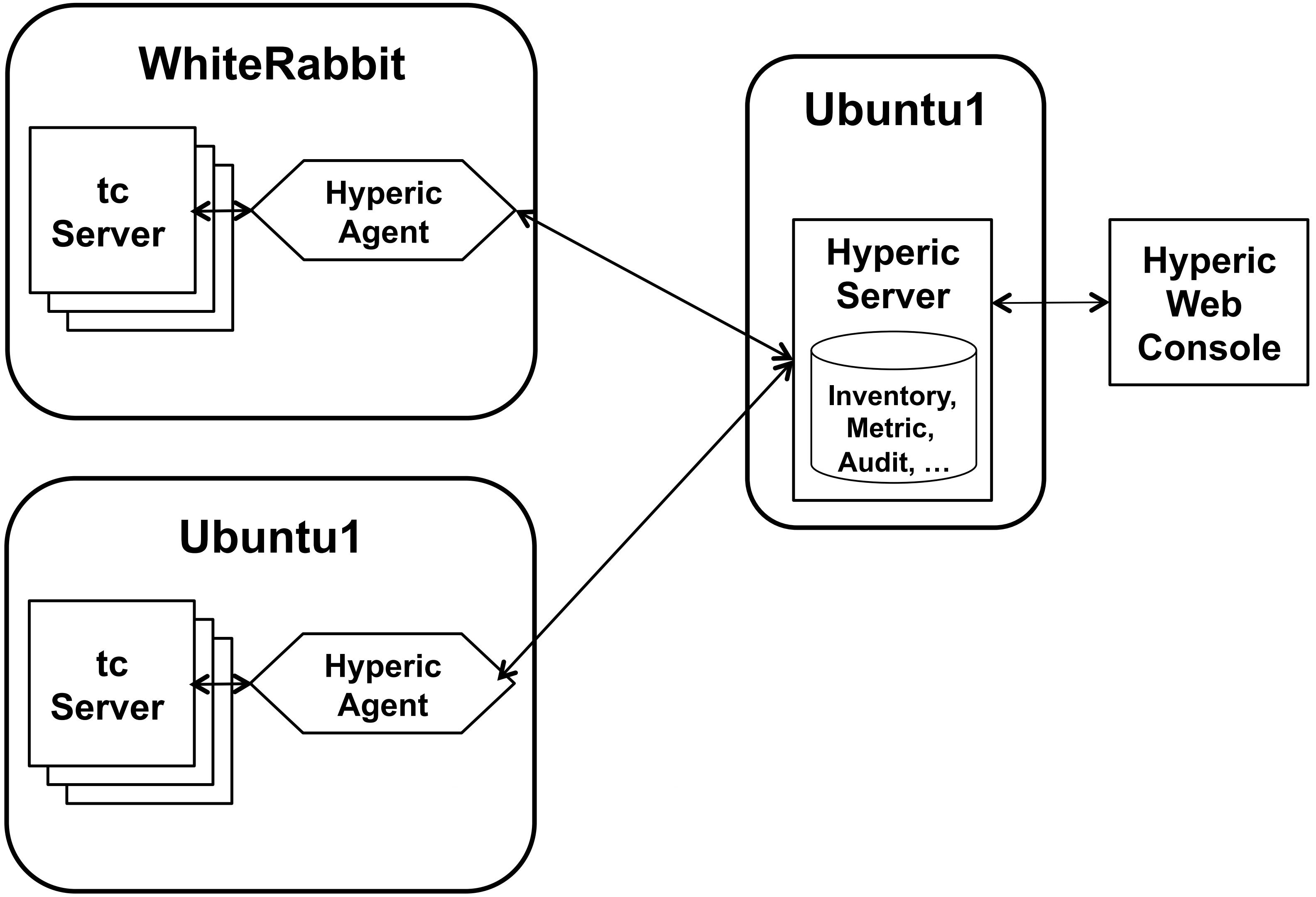
- Binaries almost unchanged – no lock in
- Centralized monitoring and administration

- Groups of server allow to deal with a group of servers – not just one
  - Start / stop / restart
  - Deployment
  - Configuration including JVM options

- Easy operations of large Tomcat installations

Demo: tc Server

Demo


WhiteRabbit
tc Server
Hyperic Agent
Ubuntu1
tc Server
Hyperic Agent
Ubuntu1
Hyperic Server
Inventory, Metric, Audit, …
Hyperic Web Console

Updating parts of an application

dm Server

- Modularization is key to maintainable software
- Modularization at runtime gives more power to Operations
- Updates of parts of the application
- Determining the source of an error
- etc

- On the client and in the embedded world OSGi has succeeded as a standard for modularization
- OSGi enters the server market…

OSGi

It's a module system

- Partition a system into a number of modules – "bundles"

- Dynamic: Bundles can be installed, started, stopped, uninstalled and updated
- ...at runtime
- better operations

- Strict visibility rules
- Resolution process satisfies dependencies of a module
- Understands versioning

It's even service-oriented

- Bundles can **publish** services… *dynamically*!
- **Service Registry** allows other bundles to **consume** services
- Services come and go at runtime
  - … *transparently* when using Spring-DM

OSGi Bundle

- The fundamental unit of deployment and modularity in OSGi
- Just a JAR file
  - with additional entries in **`META-INF/MANIFEST.MF`**
- Common manifest headers:
  - **`Bundle-SymbolicName`**
  - **`Bundle-Version`**
  - **`Bundle-Name`**
  - **`Bundle-ManifestVersion`**
  - **`Bundle-Vendor`**

Import / Export -Package

Declares package-level dependencies of your bundle.

```
Import-Package: com.xyz.foo;
Import-Package:
  com.xyz.foo;version="1.0.3"
Import-Package:
  com.xyz.foo;version="[1.0.3,1.0.3]"
Import-Package:
  com.xyz.foo;version="[1.0.3,1.1.0)",
  com.xyz.bar;version="[1.0.3,2.0.0)"
Export-Package: com.xyz.foo
Export-Package: com.xyz.foo;version="1.0.5"
```

>= 1.0.3; e.g., 1.0.3.GA, 1.0.4, etc.

Spring Dynamic Modules & SpringSource dm Server

Spring-DM: ApplicationContext

- Configuration files in `/META-INF/spring`
- Automatically merged
- ..and `ApplicationContext` is created

Service export and import

```
<beans ...>
  <osgi:service ref="customerDAO"
  interface="dao.ICustomerDAO" />
  <osgi:reference id="dataSource"
  interface="javax.sql.DataSource" />
</beans>
```

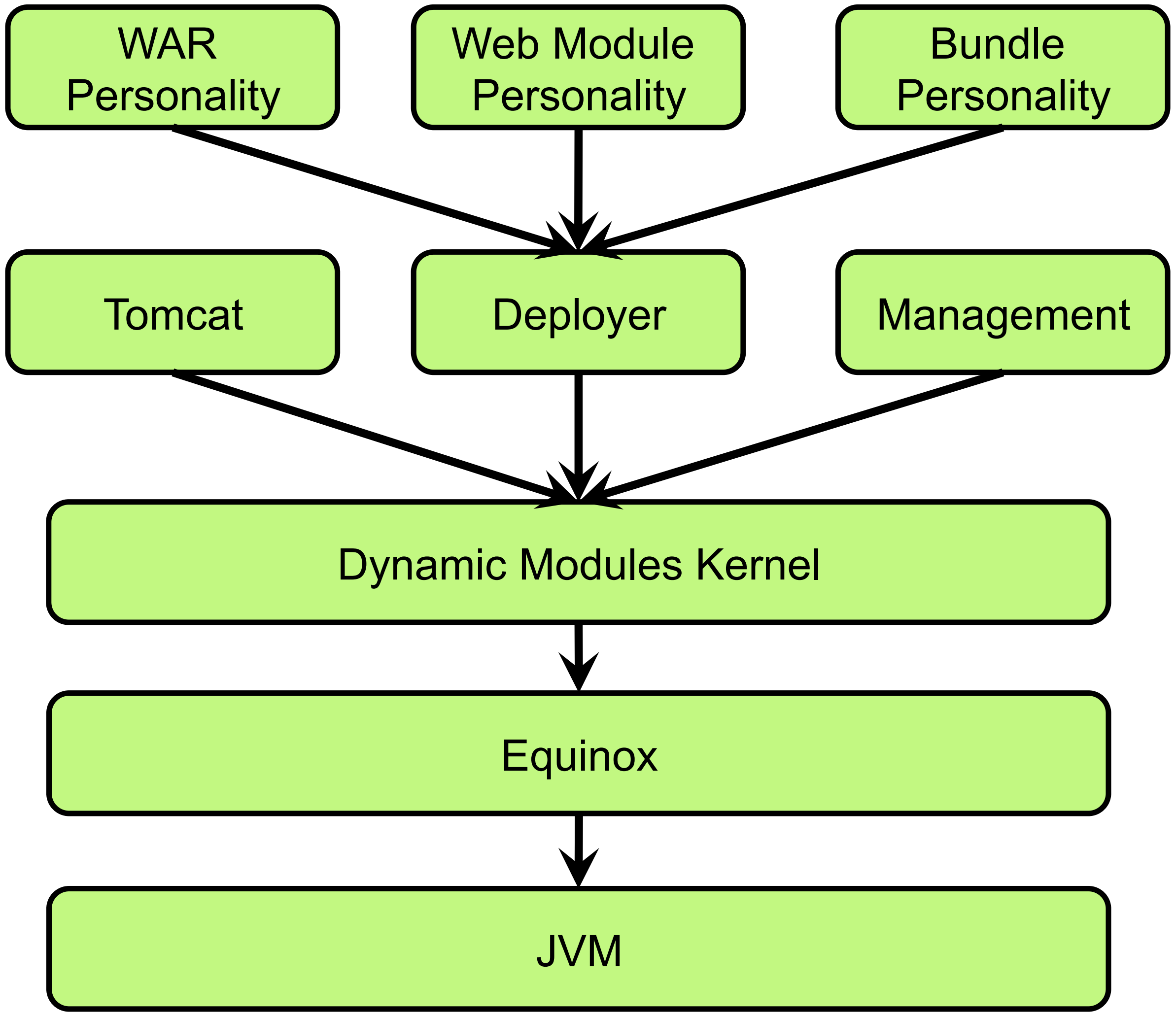
Service export and import

- Dynamic services automatically dealt with
- i.e. method calls are buffered
- Purely declarative
- No dependencies on OSGi in the code
- No resource leaks

- Not solved in Spring Dynamic Modules:
  - Easy import of libraries
  - Using JPA or Hibernate in OSGi
  - Seamless Web Support
  - Notion of an application
- Enter dm Server

dm Server Platform

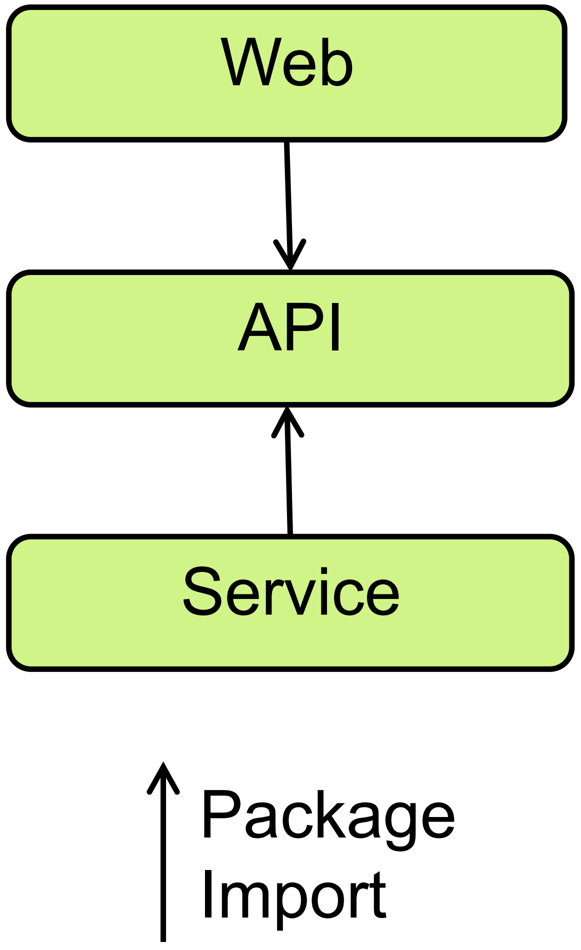
- Modular profiles
- Bundle repository
- Library provisioning
- Serviceability
  - FFDC
  - Logging / Tracing
- Built on Equinox
- Modular architecture
  - Subsystems
  - Bundles
- Small footprint



Bundles for the example

- Web
- Service
- API: only interfaces and domain classes
  - Implementation can be exchanged

- Could add infrastructure: `DataSource` / `PlatformTransactionManager`

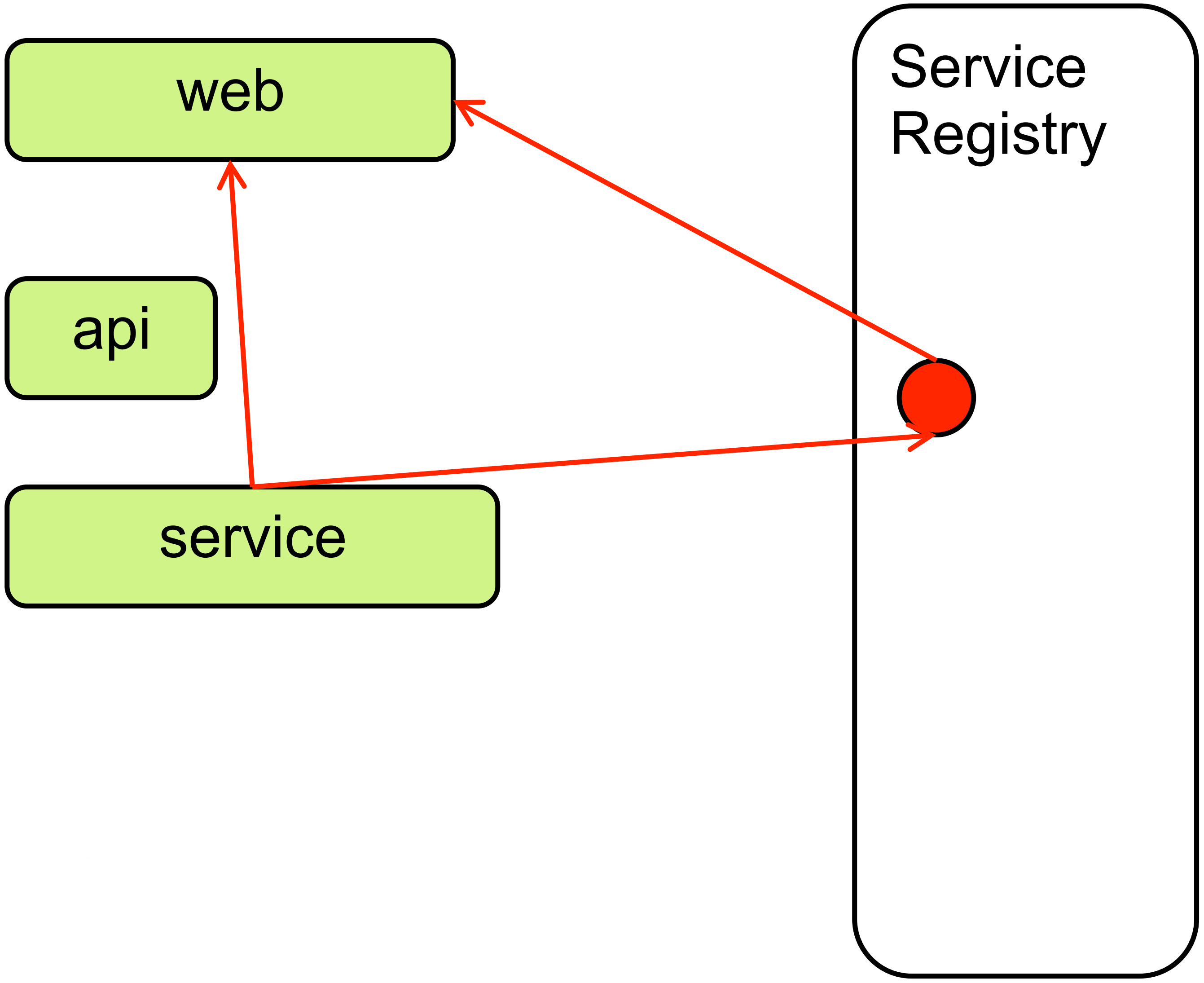


Bundles & Types

- Only dependencies to the API
- Therefore: implementation can be *exchanged even at* runtime
- No direct dependencies to any implementation
- Not shown: dependencies to external bundles
- … can be installed in dm Server
- … modular middleware!

Bundles & Services

Services: Publish / Consume



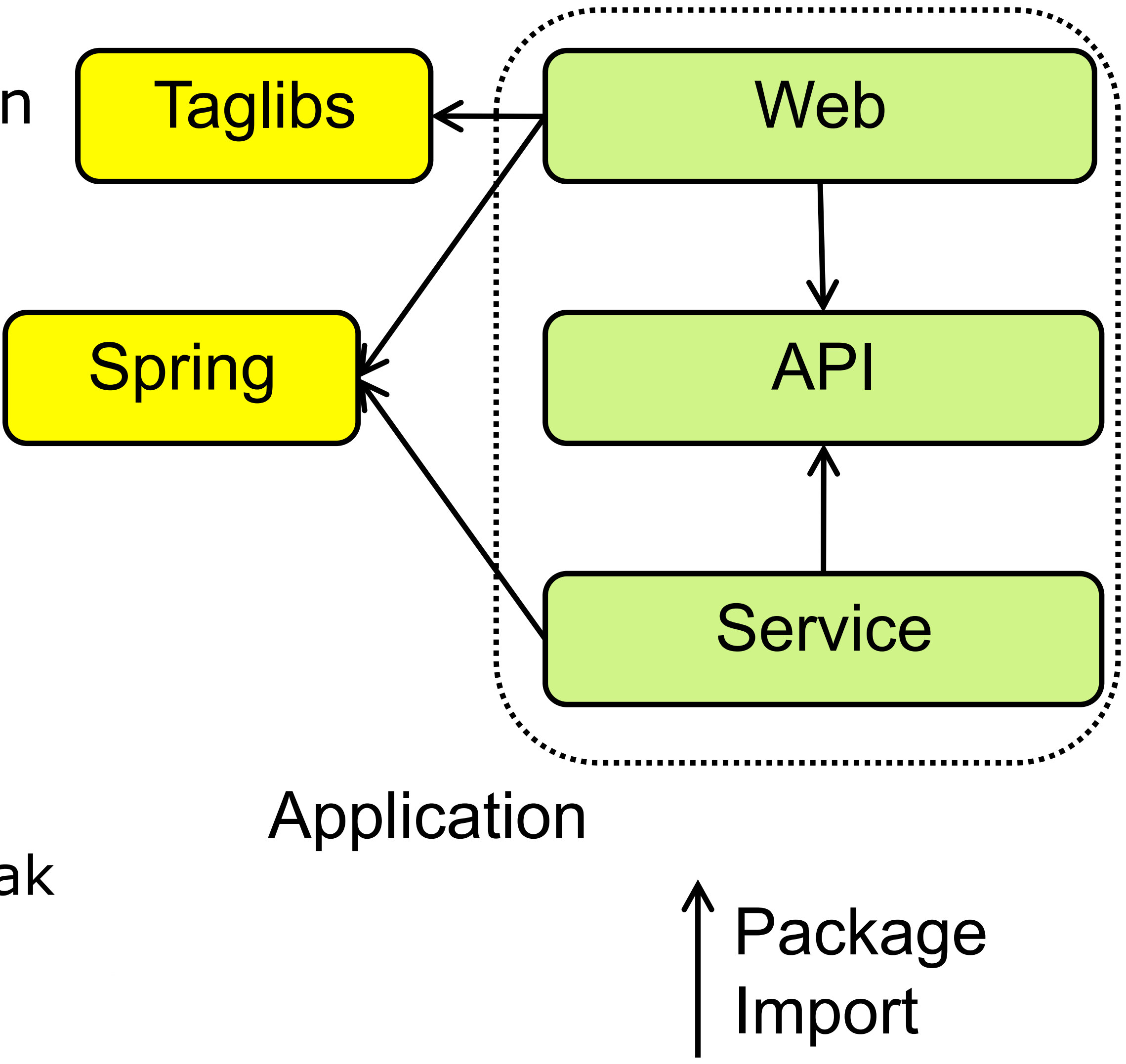
Bundles & Services

- Infrastructure can use the same principle as application services
- i.e. `DataSource` and `PlatformTransactionManager` are just another service

- Can I still run on plain Java EE?
- Yes: instead of OSGi Service directly inject Spring Beans
- no more more dynamic services / modularization
- No code change needed
- Application can run on Java EE or OSGi

PAR

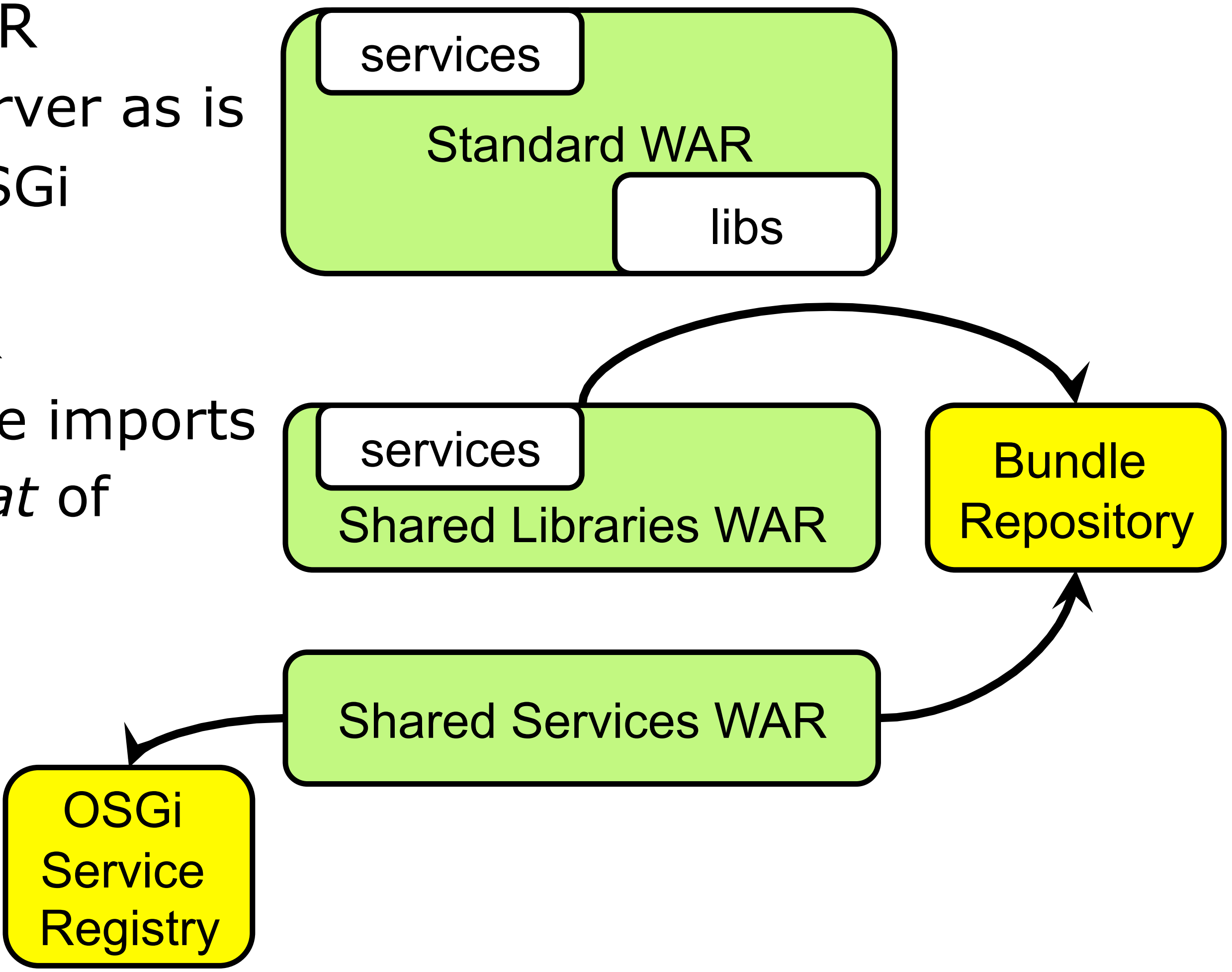
- Packaging format for all modules in an application
- JAR with Application-* manifest headers
- Single unit: deploy, refresh, undeploy
- Application boundaries
  - Scoping of types and services
  - DataSource does not leak out of the application
  - Hibernate can change domain objects



Web Migration: From WAR to PAR

Web Application Deployment Options

- Standard Java EE WAR
  - supported on dm Server as is
  - converted into an OSGi bundle
- Shared Libraries WAR
  - WAR + OSGi package imports
  - Eradicate *library bloat* of monolithic Java EE
  - WARs
- Shared Services WAR
  - Uses OSGi services with Spring's <osgi:reference>
- Web Module



Roadmap

dm Server 2.0 Roadmap

- SpringSource dm Server 2.0: 2009
- Cloning bundles
  - solves problems around static variables and more
- Shared Repository
  - make a repository available to other servers
- Plan Files
  - Define an application as a collection of bundles
  - Does not contain the bundles, more flexible
- Distributed and improved Management
  - operation on a group of servers
  - like tc Server for Tomcat
- Modular Web Applications

Support for Enterprise OSGi Standards

- **RFC 66**: Web Container for OSGi (RI based on dm Server)
- **RFC 119**: Distributed OSGi
- **RFC 124**: Blueprint Service (RI based on Spring-DM)
- **RFC 139**: JMX interface for OSGi
- **RFC 142**: JNDI and OSGi integration

Note: OSGi has an impact on operations

- New deployment model
- Updates of bundles possible

- How much less regression testing do you actually do?
- Is redeploying just a part of an application OK for operations?

Virtualization / Cloud

- Deployment blueprints define machines and their characteristics
- vApp define a deployment format based on such blueprints

- dm Server, tc Server etc will be configurable using vApp properties (e.g. ports)
- vApps can be pre defined – just add your application
- vApps can optimize for network traffic or availability

- Deployment and set up of the VMs is trivial
- …in your private or a public cloud

Summary

Summary

- Spring is a portable programming model
- This help the move from full blown Java EE to Servlet containers
- An important part of infrastructures is support for operations

- Issues:
  - Monitoring e.g. Spring Enterprise
  - Cluster e.g. tc Server
  - Modularization at runtime e.g. OSGi / dm Server
  - Virtualization / Cloud

Questions?

Eberhard Wolff
eberhard.wolff@springsource.com

http://SpringSource.com