TERRACOTTA

Transparently Clustered Spring

Jonas Bonér - Eugene Kuleshov

Terracotta, Inc.

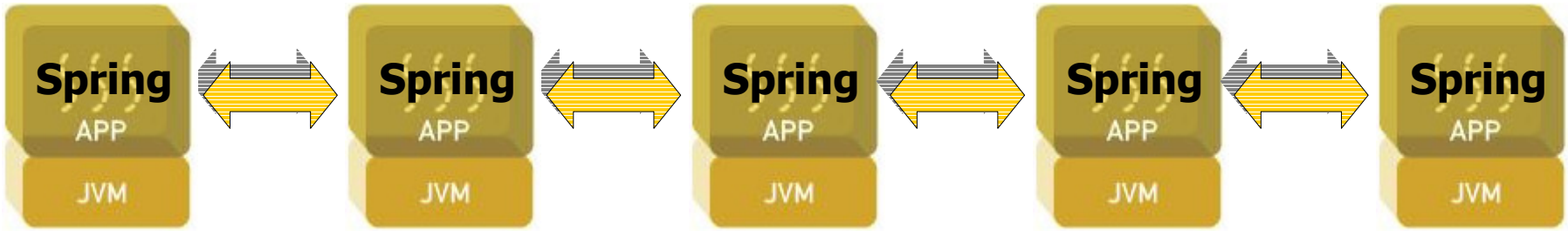
Agenda

- Overview of clustering strategies today
  - Benefits and drawbacks
- The ideal solution
  - The need for JVM-level clustering
- Introduction to Terracotta for Spring
  - Overview and demos
- Summary and QA

TERRACOTTA

Problem overview

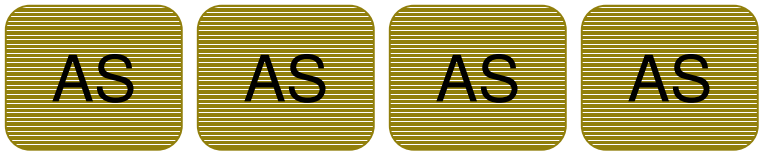
- One application per JVM is simple – but not enough
- Planned/Unplanned downtime and/or capacity stimuli necessitate a scaled-out app-tier response



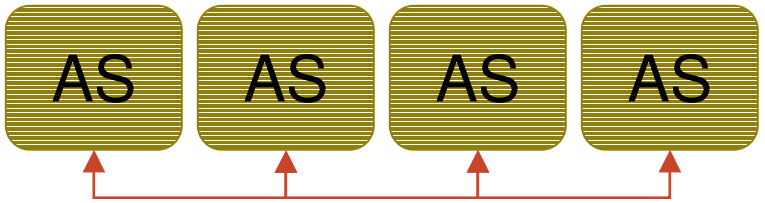
- Clustering state across JVM's is a non-trivial problem - **Lets look at the approaches.**

TERRACOTTA

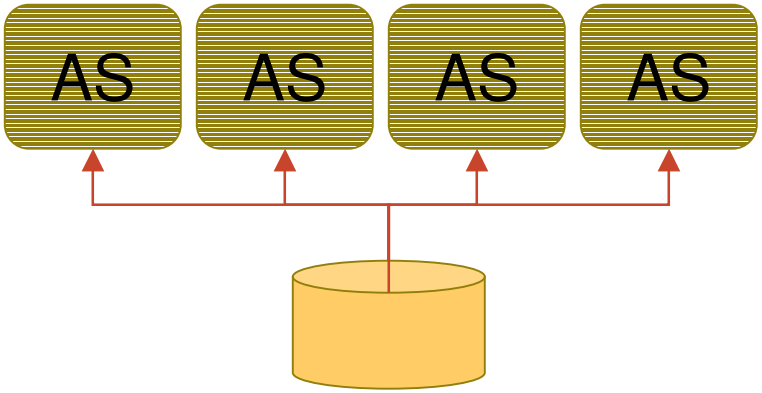
Typical clustering strategies



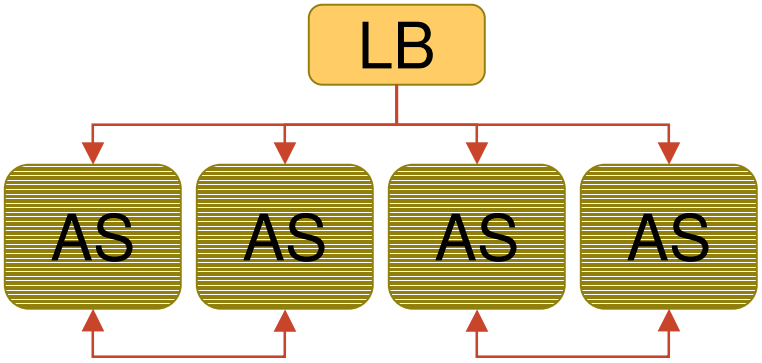
none



Broadcast P2P: push



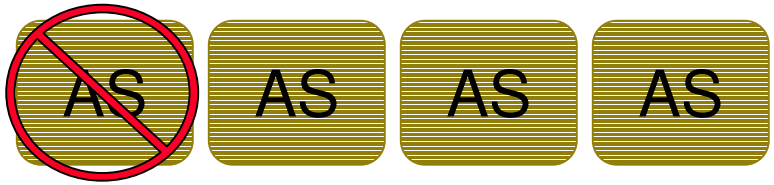
SoR | DB: pull



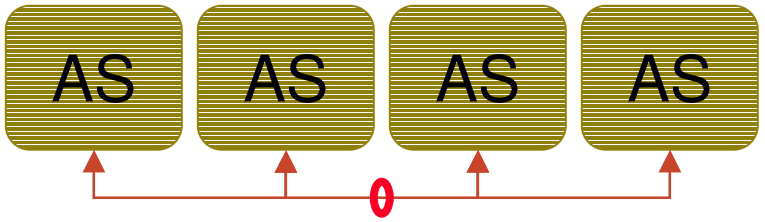
Buddy system

TERRACOTTA

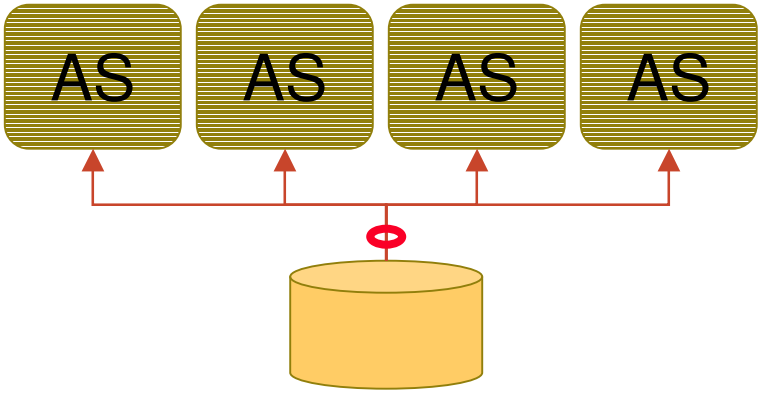
Typical clustering strategies: No silver bullet



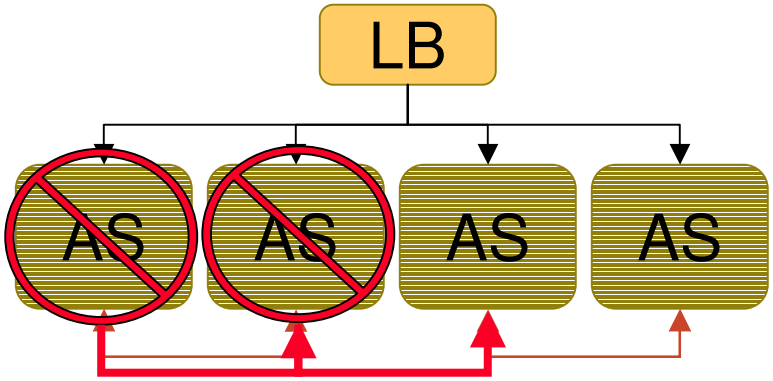
Single point of failure



Broadcast bottleneck



SoB DB bottleneck



Cache sync storms

TERRACOTTA

All strategies suffers from use of Serialization

Why is Serialization a problem?

1. Breaks Java's "pass-by-reference" semantics
   - Domain model is perturbed
   - Developers need to maintain references manually
2. Can impact scalability
   - Can not keep track of actual changes
   - Flattens and sends whole object graphs over the wire
3. Forces use of unnatural, verbose and error-prone coding rules

   For example, we need to:
   - `get()` an instance, even if we already have a reference to it
   - `put()` changes back
   - implement some event callback mechanism (`onMessage(msg)`)
   - invoke `publish(msg)` after write

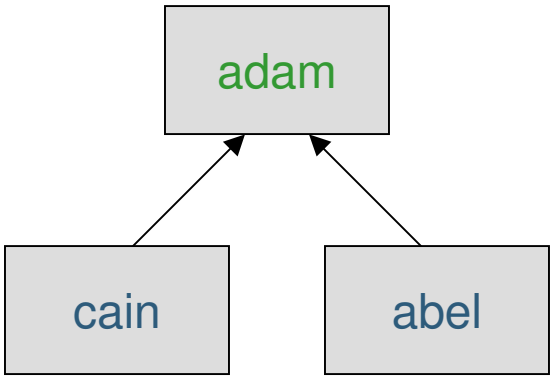
   These things are easy to forget

TERRACOTTA

How does Serialization perturb your domain model?

Java has “pass-by-reference” semantics

```
// let's create one father and two sons
Person adam = new Person(“Adam”, null);
Person cain = new Person(“Cain”, adam);
Person abel = new Person(“Abel”, adam);
```



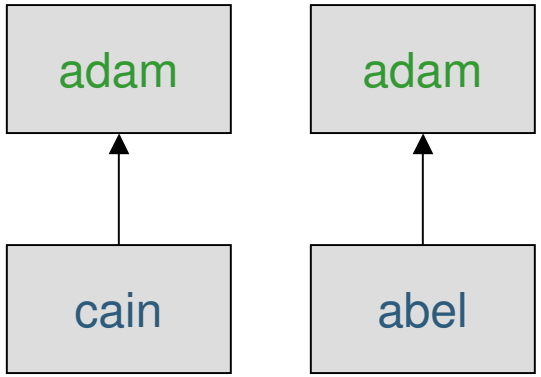
Object Identity **Is** preserved

TERRACOTTA

How does Serialization perturb your domain model?

Serialization breaks regular object references

```
// but... if we serialize Cain and Abel
Person _cain = (Person)Serializer.clone(cain);
Person _abel = (Person)Serializer.clone(abel);
```



Object Identity **Is NOT** preserved

TERRACOTTA

Why is Object Identity important?

- If broken, then you have to:
  - Maintain the relational maps between objects yourself
  - Layer some kind of primary-key mechanism onto your domain objects
- This forces you to:
  - Think like a relational database designer
  - Rip the domain model apart and then manually stitch it back together with keys

Why is Serialization-based clustering harder to scale?

- Field updates
  - ⇒ push whole object graph
  - ⇒ too much data is sent over wire
- Coarse-grained locks
  - ⇒ locking top-level object, regardless of scope of change
  - ⇒ premature lock contention

There has to be a better way!

We need Simplicity AND Scale-out

- Simplicity at runtime requires …
  - Preservation of Object Identity
  - Preservation of the semantics of the Java Memory Model (JMM)
- Scale-out requires…
  - Fine-grained replication
  - Runtime lock optimization for clustering
  - Runtime caching for data access

TERRACOTTA

Ideally, Clustered Java would…

- Use natural Java semantics
- Turn a single-JVM application into a clustered one, **without**:
  1. Code changes
  2. Semantic changes

- What we ultimately need is:

  Clustering at the JVM level

TERRACOTTA

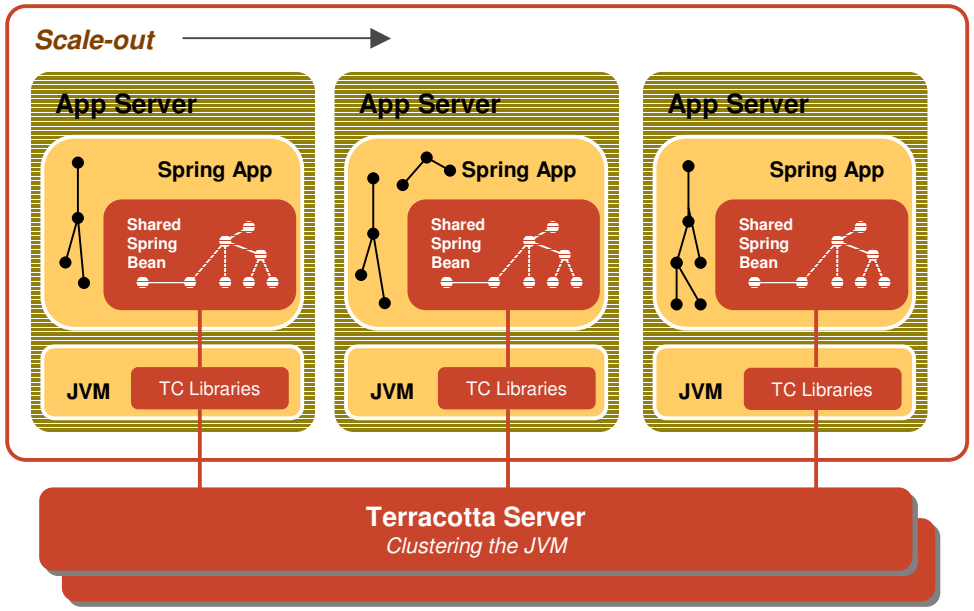
The Ideal Solution

1. Preserves your domain model
2. Does not require usage of Java Serialization
3. Requires no application code changes
4. Reduces amount of replication overhead

- Terracotta for Spring:
  - Recognizes that clustering is a deployment/operational artifact and delivers it as an infrastructure service that:
  - Clusters the JVM and shares any arbitrary Spring bean and its references by:
    - Plugging into the Java Memory Model and automatically detects what changed in the "clustered" domain model
    - Only replicating what changed to where needed

TERRACOTTA

Terracotta for Spring: Core Clustering Services

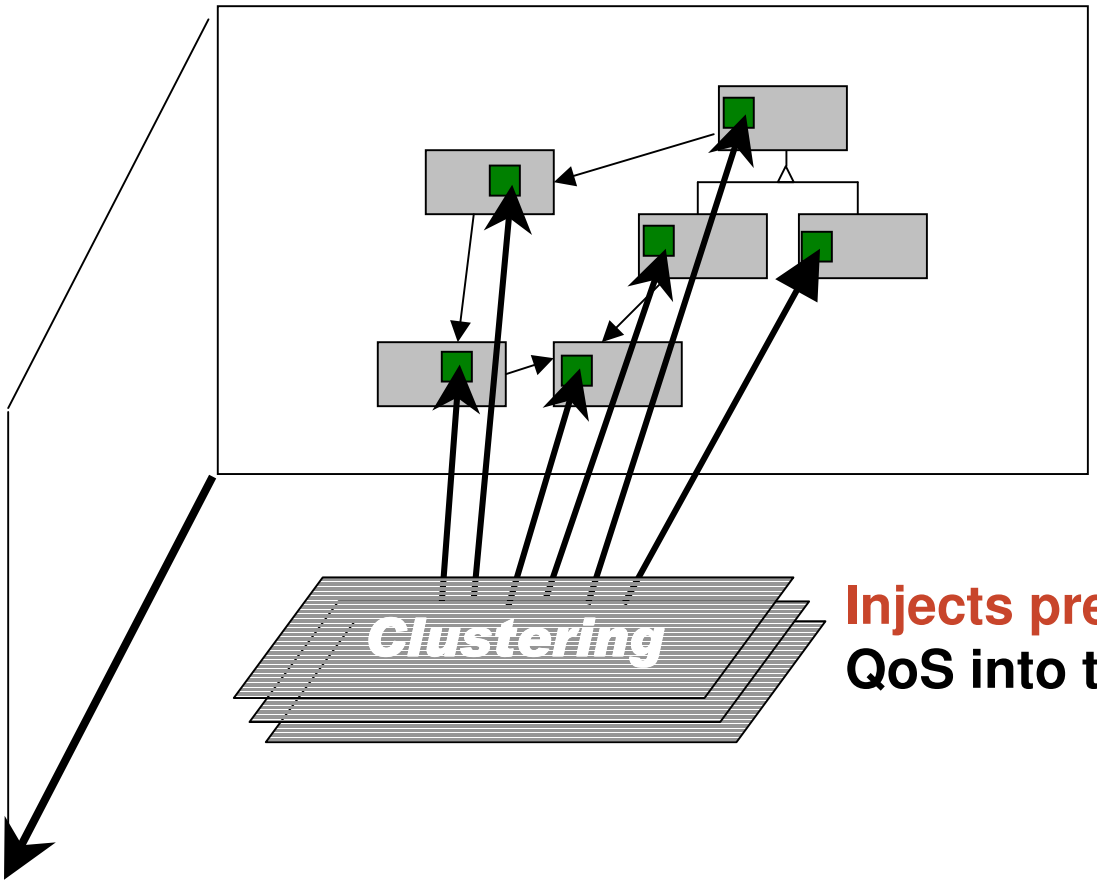


- **Transparency**
  - Runtime clustering for Spring
  - No API
  - Natural Spring Semantics
- **Sharing**
  - Fine Grained / Field Level
  - Only Where Resident
- **Coordination**
  - Distributed Events
  - Distributed Wait Notify
  - Distributed Method Call
  - Fine Grained Locking
- **Memory Management**
  - Dynamic Faulting and Flushing
  - Large Virtual Heaps

TERRACOTTA

How it works

Terracotta injects Quality of Services at runtime



Developer focuses solely on the business logic, using POJOs, Spring Beans, EJBs etc.

Injects pre packaged QoS into the application

Terracotta for Spring

TERRACOTTA

The Spring Framework

- Life-cycle
  - Defines and drives object life cycle (creates and destroys beans)
- Scope
  - *Singleton* – scoped by application context
  - *Prototype* – scoped by user (factory returns a new one every time)
  - *Session (or custom) scoped beans* – scoped by session or custom code
- Assembly
  - Well defined components with declarative dependencies
- Allows us to naturally layer clustering services on top

TERRACOTTA

DEMO

Clustering JMX state in a Web application

TERRACOTTA

JMX Demo: The Problem

- *Spring*'s support for *AOP* and *JMX* allows to capture runtime information from the application and make it available to the management tools, **but…**

- In a clustered application you **do not** get an aggregated view of the application state

- You have to manage or monitor **each node individually or code for a cluster-based view.**

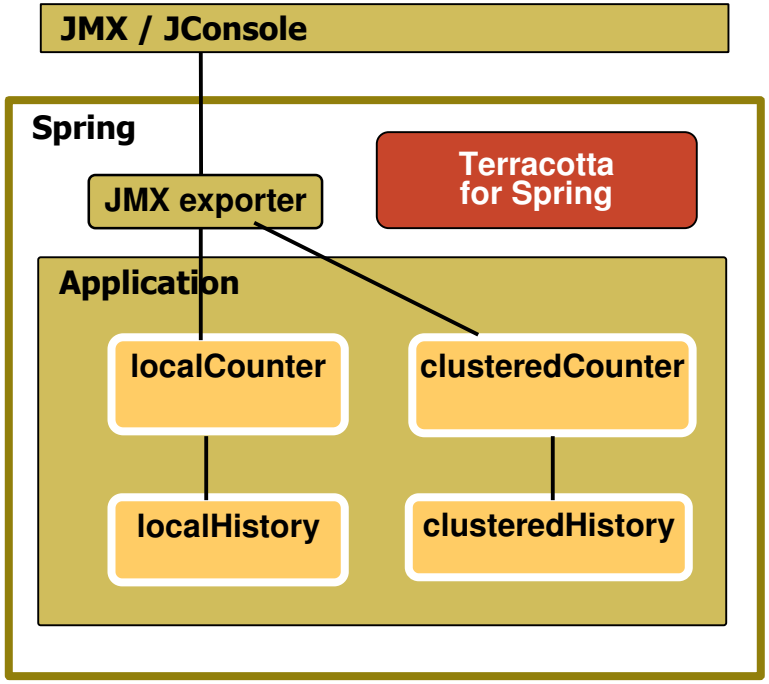
- This demo shows how *Terracotta for Spring*
  - **Shares data throughout the cluster**
  - Clustered state is **made available through JMX with no code-changes.**
  - Creates a **single access point** for monitoring and management

TERRACOTTA

JMX Demo: Spring Configuration

- Two “counter” service beans (local and clustered)
- Two “history” interceptors (local and clustered)

```
<bean id="localCounter
      class="demo.jmx.Counter"/>

<bean id="clusteredCounter"
      class="demo.jmx.Counter"/>

<bean id="localHistory"
      class="demo.jmx.HistoryQueue"/>

<bean id="clusteredHistory"
      class="demo.jmx.HistoryQueue"/>
```



TERRACOTTA

JMX Demo: Terracotta Configuration

- Terracotta for Spring can declaratively cluster Spring beans with **zero code changes**
- Using a simple XML configuration we declare which beans should be clustered
- Clustered state is made available through JMX

Terracotta config

```
<spring>
  <application name="tc-jmx">
    <application-contexts>
      <application-context>
        <paths>
          <path>*/applicationContext.xml</path>
        </paths>
        <beans>
          <bean name="clusteredCounter"/>
          <bean name="clusteredHistory"/>
        </beans>
      </application-context>
    </application-contexts>
  </application>
</spring>
```

Spring config

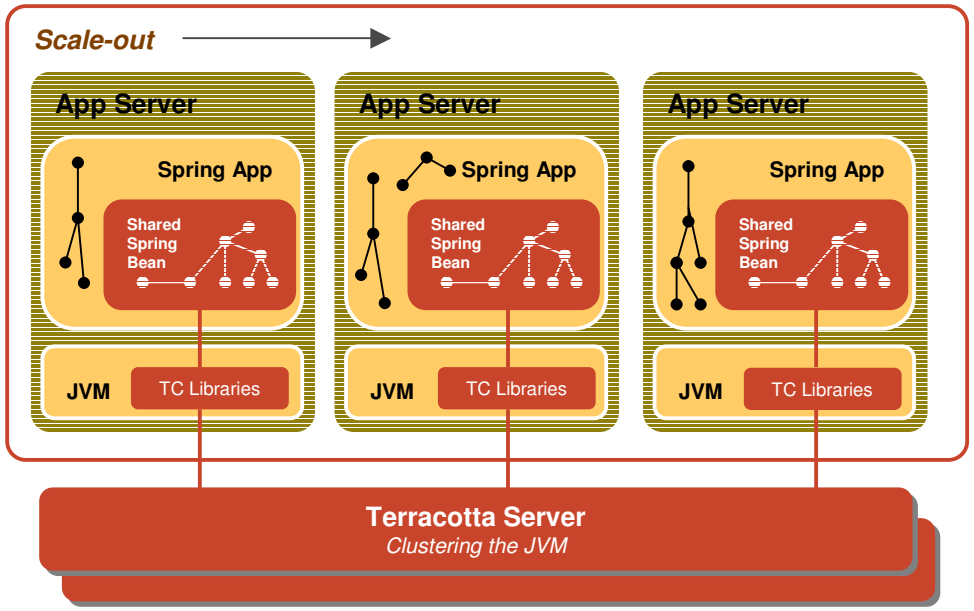
```
<bean id="localCounter"
      class="demo.jmx.Counter"/>

<bean id="clusteredCounter"
      class="demo.jmx.Counter"/>

<bean id="localHistory"
      class="demo.jmx.HistoryQueue"/>

<bean id="clusteredHistory"
      class="demo.jmx.HistoryQueue"/>
```

JMX Demo: Conclusion

- Drops In and Out
  - Zero code changes needed
  - Declarative configuration
- Natural Clustering of Spring Beans
  - Plain POJOs – **no Java Serialization**
- Clustered state is made available through JMX
  - Single point of monitoring and management

TERRACOTTA

Terracotta for Spring Features



- Runtime Clustering Service
  - Drops In and Out
  - Runtime Visibility of your Spring application
- Spring Framework Support:
  - Clustered Singleton Beans
  - Clustered Session Scoped Beans (and custom scoped beans)
  - WebFlow (including continuations)
  - Availability of Clustered State via JMX
  - Distributed Asynchronous Application Context Events (Cluster wide thread coordination)

TERRACOTTA

Drops In and Out

- No changes to existing code necessary
- Declarative configuration in Terracotta XML file
- Spring style configuration

```
<spring>
  <jee-application name="MyWebApp">
    <application-contexts>
      <application-context>
        <paths>
          <path>*/applicationContext.xml</path>
        </paths>
        <beans>
          <bean name="clusteredBean"/>
        </beans>
      </application-context>
    </application-contexts>
  </jee-application>
</spring>
```

TERRACOTTA

Natural Clustering of Spring Beans

- Supported types are:
  - Singleton beans (including interceptors)
  - Session scoped beans
  - Custom scoped beans
- Life-cycle semantics preserved
- Scope semantics preserved - within the same "logical" `ApplicationContext`

TERRACOTTA

Sharing JMX state

- Shared beans can be exposed through Spring JMX
- Coherent view of the aggregate state throughout the cluster
- One single point of management and monitoring

TERRACOTTA

Spring WebFlow

- Spring WebFlow stores conversational state in session
- Terracotta for Spring clusters Http Session transparently
- Just need a single line of config to cluster WebFlow state (default or continuation-based)

TERRACOTTA

DEMO

Clustering Spring WebFlow's continuations (conversational state)

TERRACOTTA

Distributed Reliable Events

- Spring has a simple event/messaging facility in the `ApplicationContext`

- Similar to the Observer pattern
  1. Publish event to the context using `publishEvent(event)`
  2. All beans that implements the `ApplicationListener` interface will receive the event

- Turn Spring `ApplicationContext` events into Distributed Reliable Events

TERRACOTTA

Summary

- Spring has increased momentum in the enterprise as an application framework of choice
- Scaling-Out Spring applications is more important than ever
- Simplified, Efficient Clustering at runtime:
  - While preserving the natural semantics for Spring
- Terracotta for Spring can address these issues today by Clustering at the JVM level

TERRACOTTA

Availability

- Terracotta for Spring
  - Available for download: TODAY
  - Free production license for up to 2 nodes
- Download Terracotta for Spring today at
  - http://www.terracottatech.com/downloads.jsp
  - Additional inquiries: Contact sales@terracottatech.com
- Download Spring 1.x and 2.0 today at
  - http://www.springframework.org
- Spring Training and Education Services are available today at
  - http://www.Interface21.com
  - Additional inquiries: Contact info@interface21.com

TERRACOTTA

For More Information

- http://www.terracottatech.com/
- http://blog.terracottatech.com/

- http://springframework.org/

- http://jonasboner.com/
- http://jroller.com/page/eu

TERRACOTTA

Questions?

Thank You

www.terracottatech.com

TERRACOTTA