Web Sessions User Guide

Innovation Release

Version 4.3.5

April 2018

Document ID: WS-UG-435-20180417

Table of Contents

1 About Web Sessions

What is Web Sessions?

Terracotta Web Sessions is the fast, reliable way to get the scalability benefits of a stateless web architecture without overloading your database or rewriting your application. Supporting the Java Servlet session standard, Terracotta Web Sessions works with your favorite Web framework, your favorite application container, and your custom session objects.

Easy Setup

Terracotta Web Sessions is simple to use and a good fit for a wide range of use cases, from departmental applications to large-scale web and e-commerce sites.

Sessions Visibility

See the performance of your entire application cluster at a glance. Also browse the live contents of user sessions for easy debugging and problem resolution. Terracotta Web Sessions makes it easy to configure and optimize your sessions and watch the performance boost you're getting in real time.

In-Memory Speed

Terracotta dynamically partitions and optimizes the locality of user session data to provide coherent and reliable access to hot sessions at in-memory speed. This architecture yields very high application throughput and very low session access latency, no matter how many application servers and concurrent users you have.

Scale

Terracotta Web Sessions makes it easy to add additional servers to your application tier. High availability for sessions at scale is critical to increasing application throughput without downtime or disrupting the customer experience.

High Performance and High Availability

Enjoy high availability (99.999% uptime) for your session data without sacrificing application performance or punishing your database. Terracotta Web Sessions can take full advantage of BigMemory's fault tolerance and Fast Restartable Store to ensure session data is always available, even in the event of server failure or restart.

Session Consistency

Terracotta session data is always guaranteed to be coherent, consistent, and up-to-date. This ensures that your customers' experience is unaffected when they are routed to different application servers in the event of server failure or restarts.

Efficiency

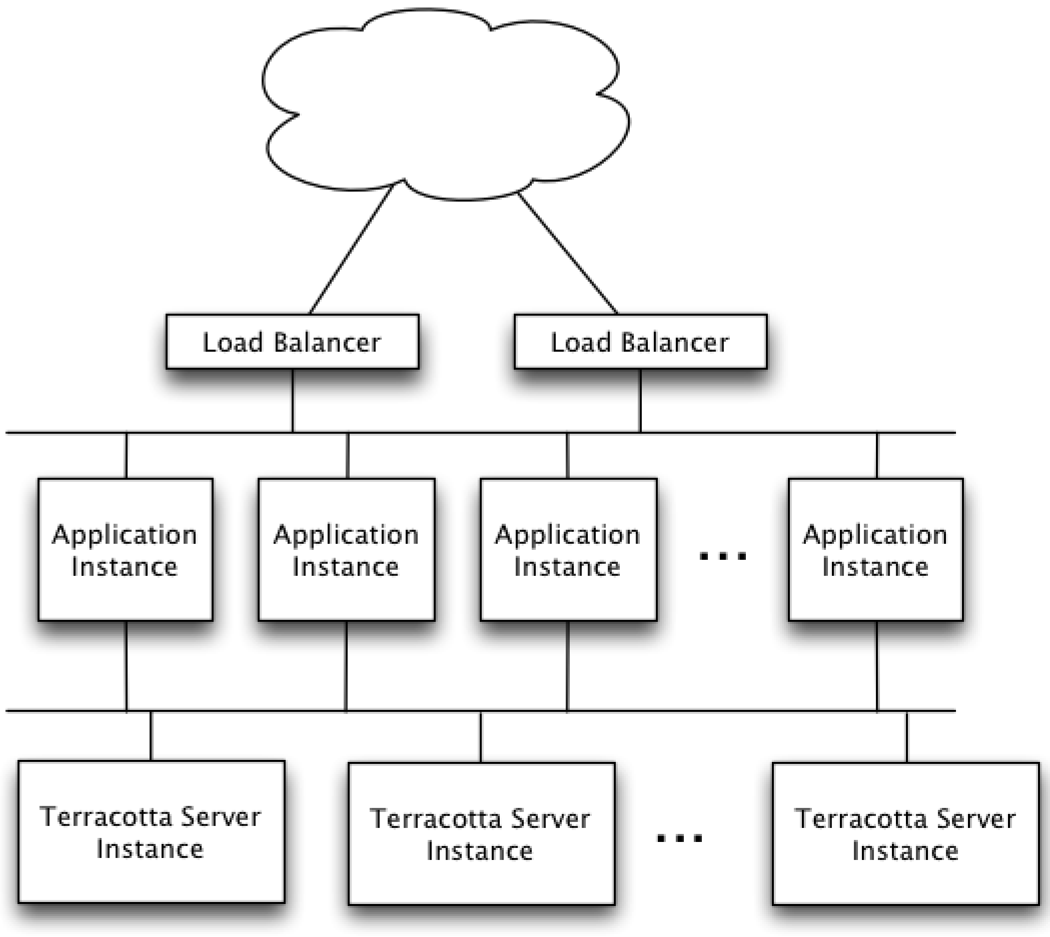
Terracotta keeps only the locally active session data in memory and sends only the changed data over the network when sessions are updated. This efficient use of memory and network resources enables Terracotta Web Sessions to outperform competitive solutions.

Broad Container Support

Terracotta Web Sessions works with most major application servers and a wide range of popular web frameworks.

Architecture

The following diagram shows the architecture of a typical Terracotta-enabled web application.



The load balancer parcels out HTTP requests from the Internet to each application server. To maximize the locality of reference of the clustered HTTP session data, the load balancer uses HTTP session affinity so all requests corresponding to the same HTTP session are routed to the same application server. However, with a Terracotta-enabled web application, any application server can process any request. Terracotta Web Sessions clusters the sessions, allowing sessions to survive node hops and failures.

The application servers run both your web application *and* the Terracotta client software, and are called "clients" in a Terracotta cluster. As many application servers may be deployed as needed to handle your site load.

For more information about the Terracotta clusters, refer to the *BigMemory Max Administrator Guide*.

2 Installing and Configuring Web Sessions

Requirements

- JDK 1.6 or higher.
- An application server.
- All clustered objects must be serializable.

Installing the Session JARS

Download Terracotta Web Sessions

Download the BigMemory Max kit, which includes Terracotta Web Sessions.

Unpack the Kit

Unpack the distribution into a directory on your system. In the following instructions, we will refer to the directory as <terracotta>/. Where forward slashes ("/") are given in directory paths, substitute back slashes ("\") for Microsoft Windows installations.

Copy the License Key

Copy the license key to the terracotta distribution directory. This file, called terracotta-license.key, was attached to an email you received after registering for the download.

```
%> cp terracotta-license.key <terracotta>/
```

Copy the JAR files

To cluster your application's web sessions, add the following two JAR files to your application's classpath, and place these files in the WEB-INF/lib folder of the Web Application Archive (WAR) on all of your application servers. These files are under ${TERRACOTTA_HOME}/.

```
sessions/lib/web-sessions-<version>.jar
```

<version> is the current version of the Terracotta Web Sessions JAR.

```
apis/toolkit/lib/terracotta-toolkit-runtime-ee-<version>.jar
```

This particular JAR contains the Terracotta client libraries.

Configuring Your Web Sessions Cluster

Terracotta servers, and Terracotta clients running on the application servers in the cluster, are configured with a Terracotta configuration file, tc-config.xml by default. Servers that are not started with a specified configuration will use a default configuration.

To add Terracotta clustering to your application, you must specify how Terracotta clients get their configuration by including the source in your web.xml file. Add the following configuration snippet to web.xml:

```
<filter>
 <filter-name>terracotta</filter-name>
 <!-- The filter class is specific to the application server. -->
 <filter-class>org.terracotta.session.{container-specific-class}</filter-class>
 <init-param>
   <param-name>tcConfigUrl</param-name>
   <!-- <init-param> of type tcConfigUrl has a <param-value> element containing
        the URL or filepath (for example, /lib/tc-config.xml) to tc-config.xml.
    If the Terracotta configuration source changes at a later time,
    it must be updated in configuration. -->
   <param-value>localhost:9510</param-value>
 </init-param>
<!-- The following init-params are optional.
 <init-param>
   <param-name>synchronousWrite</param-name>
   <param-value>false</param-value>
 </init-param>
 <init-param>
   <param-name>sessionLocking</param-name>
   <param-value>false</param-value>
 </init-param>
 <init-param>
   <param-name>maxBytesOnHeap</param-name>
   <param-value>128M</param-value>
 </init-param>
 <init-param>
   <param-name>maxBytesOffHeap</param-name>
   <param-value>4G</param-value>
 </init-param>
 <init-param>
   <param-name>rejoin</param-name>
   <param-value>false</param-value>
 </init-param>
 <init-param>
   <param-name>nonStopTimeout</param-name>
   <param-value>-1</param-value>
 </init-param>
 <init-param>
   <param-name>concurrency</param-name>
   <param-value>0</param-value>
 </init-param>
<!-- End of optional init-params. -->
</filter>
<filter-mapping>
 <!-- Must match filter name from above. -->
 <filter-name>terracotta</filter-name>
 <url-pattern>/*</url-pattern>
 <!-- Enable all available dispatchers. -->
 <dispatcher>ERROR</dispatcher>
 <dispatcher>INCLUDE</dispatcher>
 <dispatcher>FORWARD</dispatcher>
 <dispatcher>REQUEST</dispatcher>
</filter-mapping>
```

`<filter-name>` can contain a string of your choice. However, the value of `<filter>/<filter-name>` must match `<filter-mapping>/<filter-name>`.

Choose the appropriate value for <filter-class> from the following table.

Container	Value of <filter-class>
GlassFish	org.terracotta.session.TerracottaGlassfish31xSessionFilter
JBoss 6.1.x	org.terracotta.session.TerracottaJboss61xSessionFilter
JBoss 7.1.x	org.terracotta.session.TerracottaJboss71xSessionFilter
JBoss 7.2.x	org.terracotta.session.TerracottaJboss72xSessionFilter
Jetty 8.1.x	org.terracotta.session.TerracottaJetty81xSessionFilter
Jetty 9.0.x	org.terracotta.session.TerracottaJetty90xSessionFilter
Resin	org.terracotta.session.TerracottaResin40xSessionFilter
Tomcat 6.0.x	org.terracotta.session.TerracottaTomcat60xSessionFilter
Tomcat 7.0.x	org.terracotta.session.TerracottaTomcat70xSessionFilter
WebLogic 10.3.x	org.terracotta.session.TerracottaWeblogic103xSessionFilter
WebLogic 12.1.x	org.terracotta.session.TerracottaWeblogic121xSessionFilter
WebSphere 7.0.x	org.terracotta.session.TerracottaWebsphere70xSessionFilter
WebSphere 8.0.x	org.terracotta.session.TerracottaWebsphere80xSessionFilter
WebSphere 8.5.x	org.terracotta.session.TerracottaWebsphere85xSessionFilter

Ensure that the Terracotta filter is the first `<filter>` element listed in web.xml. Filters processed ahead of the Terracotta filter may disrupt its processing.

web.xml should be in /WEB-INF if you are using a WAR file.

Starting the Cluster

For Terracotta to function properly, make sure that your `JAVA_HOME` setting is set.

To start the cluster

1. Start the Terracotta server.

   UNIX/Linux:

   ```
   [PROMPT] ${TERRACOTTA_HOME}/server/bin/start-tc-server.sh
   ```

   Microsoft Windows:

   ```
   [PROMPT] ${TERRACOTTA_HOME}\server\bin\start-tc-server.bat
   ```

2. Start your application servers.
3. (Optional) Start your management console (any JMX monitoring tool is fine) and connect it to your application server. For more information, refer to "Managing Web Sessions Using JMX" on page 24.

Configuring a Terracotta Cluster

This step shows you how to run clients and servers on separate machines and add failover (High Availability). You will expand the Terracotta cluster and add High Availability by doing the following:

- Moving the Terracotta server to its own machine
- Creating a cluster with multiple Terracotta servers
- Creating multiple application nodes

These tasks bring your cluster closer to a production architecture.

Note: For additional details about configuring a Terracotta cluster, see the "Configuring a Terracotta Server Array" and "Terracotta Configuration Parameters" chapters in the *BigMemory Max Administrator Guide*. For additional details about configuring a cluster for high availability, see the *BigMemory Max High-Availability Guide*.

To configure a Terracotta cluster

1. Shut down the Terracotta cluster.
2. Create a Terracotta configuration file called tc-config.xml with contents similar to the following:

   ```
   <?xml version="1.0" encoding="UTF-8"?>
   <!-- All content copyright Terracotta, Inc., unless otherwise indicated.
    All rights reserved. -->
   <tc:tc-config xsi:schemaLocation="http://www.terracotta.org/schema/terracotta-9.xsd"
   xmlns:tc="http://www.terracotta.org/config"
   ```

```
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <servers>
   <!-- Sets where the Terracotta server can be found.
    Replace the value of host with the server's IP address. -->
   <server host="server.1.ip.address" name="Server1">
     <data>%(user.home)/terracotta/server-data</data>
     <logs>%(user.home)/terracotta/server-logs</logs>
   </server>
   <!-- If using a standby Terracotta server, also referred to as
    an ACTIVE-PASSIVE configuration, add the second server here. -->
   <server host="server.2.ip.address" name="Server2">
     <data>%(user.home)/terracotta/server-data</data>
     <logs>%(user.home)/terracotta/server-logs</logs>
   </server>
 </servers>
 <!-- Sets where the generated client logs are saved on clients. -->
 <clients>
   <logs>%(user.home)/terracotta/client-logs</logs>
 </clients>
</tc:tc-config>
```

3. Install Terracotta on a separate machine for each server you configure in tc-config.xml.
4. Copy the tc-config.xml to a location accessible to the Terracotta servers.
5. Install the Sessions JAR files on each application node you want to run in the cluster. Be sure to install your application and any application servers on each node. For information about installing the Sessions JAR files, see "Installing the Session JARS" on page 10.
6. Edit web.xml on each application server to list both Terracotta servers:

   ```
   <param-value>server.1.ip.address:9510,server.2.ip.address:9510</param-value>
   ```

7. Start the Terracotta server in the following way, replacing "Server1" with the name you gave your server in tc-config.xml:

   UNIX/Linux

   ```
   [PROMPT] ${TERRACOTTA_HOME}/server/bin/start-tc-server.sh -f <path/to/tc-config.xml> \
      -n Server1
   ```

   Microsoft Windows

   ```
   [PROMPT] ${TERRACOTTA_HOME}\server\bin\start-tc-server.bat -f <path\to\tc-config.xml> ^
      -n Server1
   ```

   If you configured a second server, start that server in the same way on its machine, entering its name after the `-n` flag. The second server to start up becomes the hot standby, or PASSIVE. Any other servers you configured will also start up as standby servers.

8. Start all application servers.
9. Start your JMX console tool and view the cluster. For information about using JMX with Web Sessions, see "Managing Web Sessions Using JMX" on page 24.

3 Running the Shopping Cart Example

Starting the Example

We'll use the "Cart" sample in the sessions samples directory of the Terracotta distribution to demonstrate clustered Web Sessions.

Before starting the sample application, make sure you have performed the following steps as described in "Installing the Session JARS" on page 10.

- Download Terracotta Web Sessions.
- Unpack the kit.
- Copy your license key to the distribution directory.

To start the example

1. Start the Terracotta Server

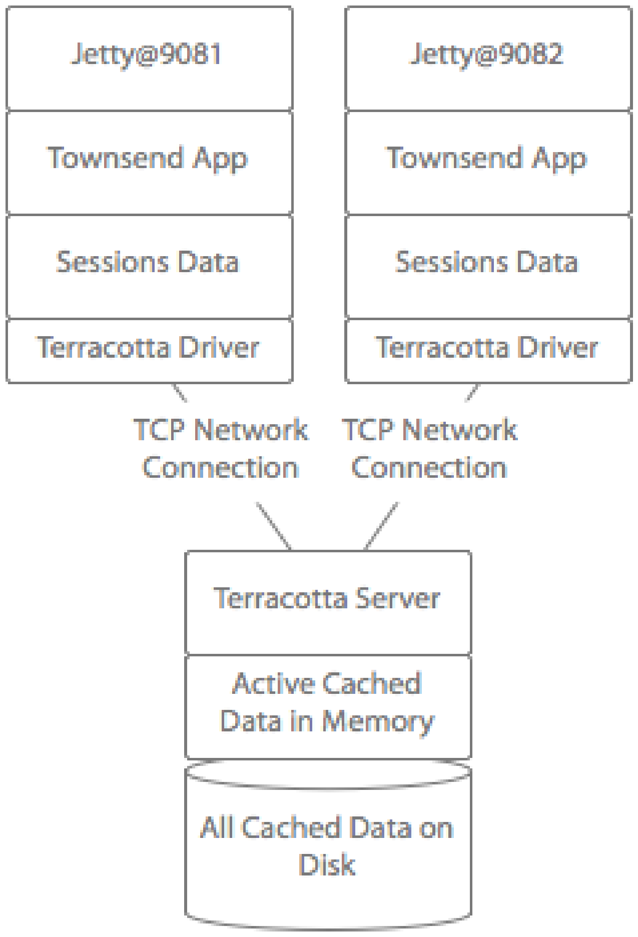
```
%> cd <terracotta>/sessions/code-samples/cart
%> bin/start-sample-server.sh
```

2. Start the Sample

```
%> bin/start-sample.sh
```

Viewing the Example

The Shopping Cart sample starts up two instances of Jetty, each connected to the Terracotta server for access to shared session data. The sample application uses Terracotta to store session data for scalable high availability. Any session can be read from any application server.



View the Cart

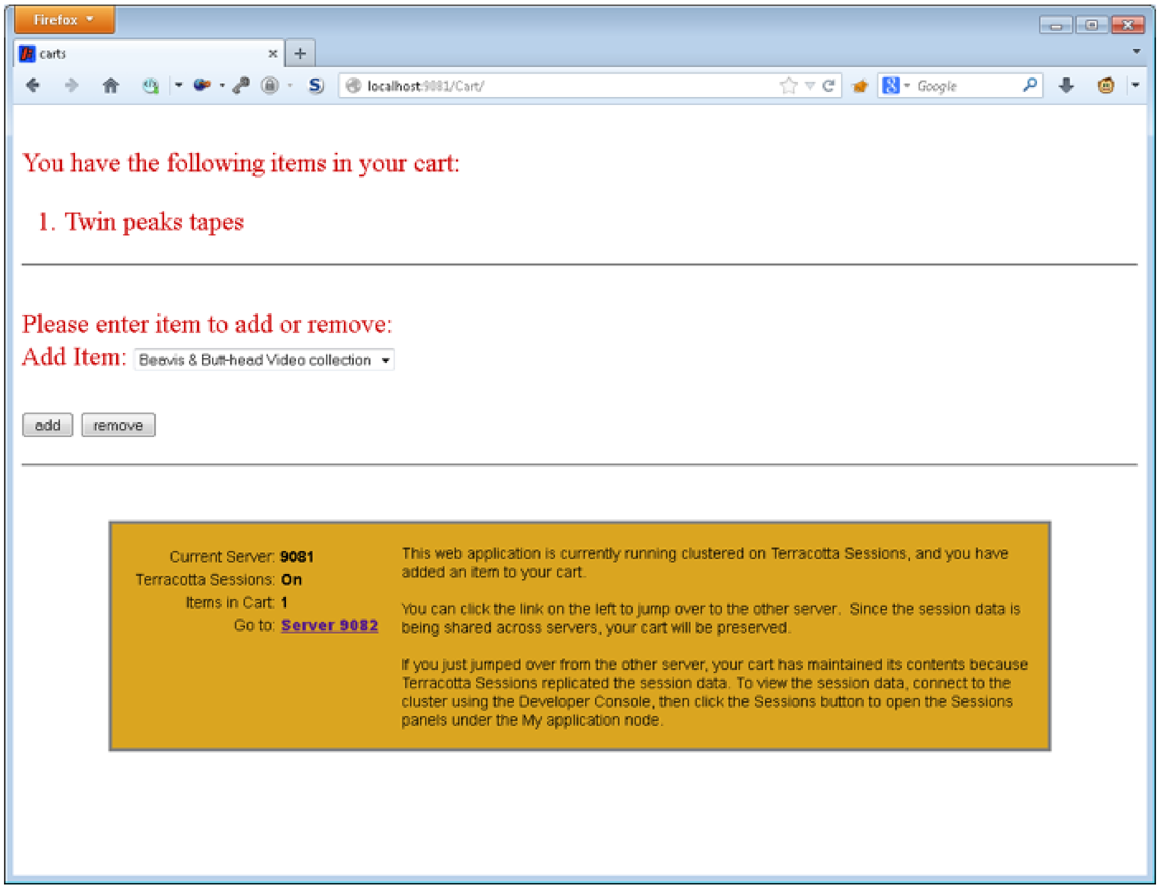
Once the Jetty instances have started, you can view the sample application by visiting the following links:

- http://localhost:9081/Cart
- http://localhost:9082/Cart

Note: These links *will not work* unless you have the sample running.

Create Session Data

After Jetty loads the sample application, select an item to place in the cart. It should look something like this in your browser:



You have the following items in your cart:

1. Twin peaks tapes

Please enter item to add or remove:
Add Item: Beavis & Butt-head Video collection

Current Server: **9081**
Terracotta Sessions: **On**
Items in Cart: **1**
Go to: **Server 9082**

This web application is currently running clustered on Terracotta Sessions, and you have added an item to your cart.

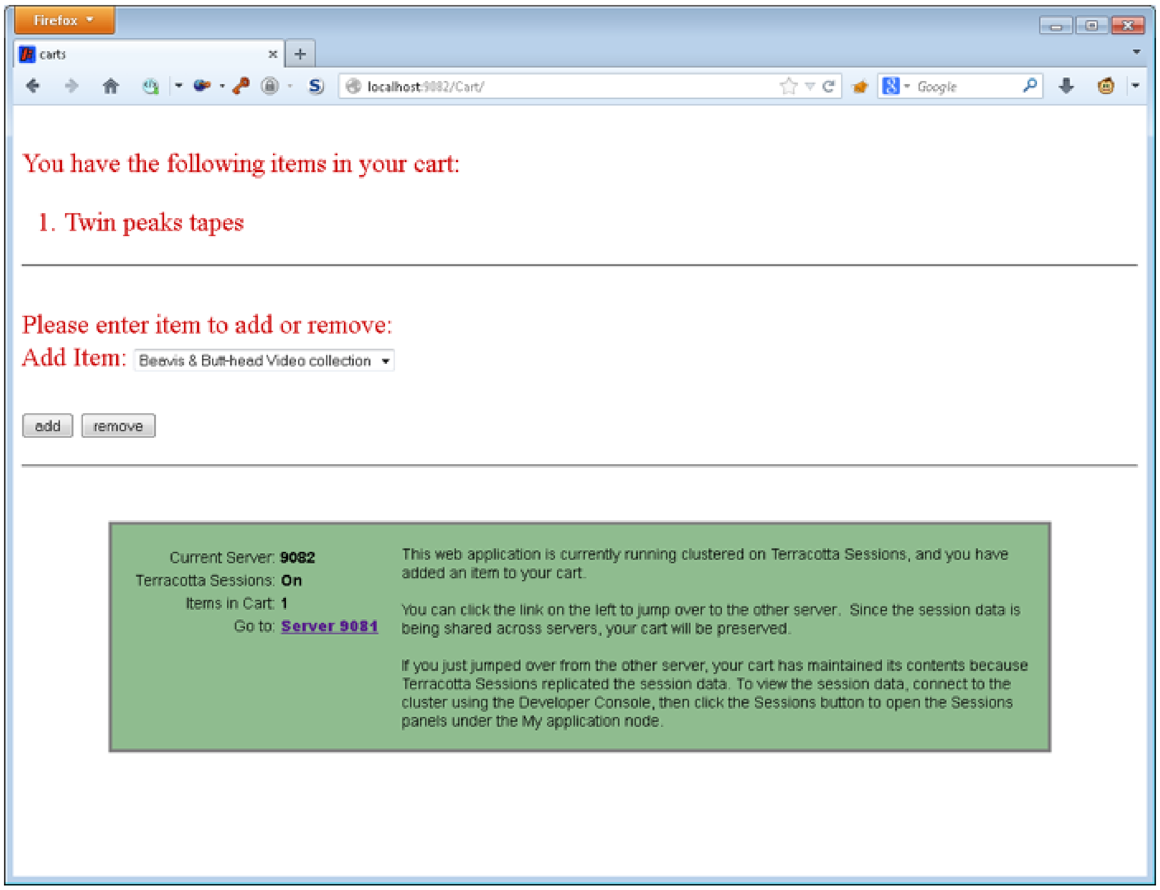
You can click the link on the left to jump over to the other server. Since the session data is being shared across servers, your cart will be preserved.

If you just jumped over from the other server, your cart has maintained its contents because Terracotta Sessions replicated the session data. To view the session data, connect to the cluster using the Developer Console, then click the Sessions button to open the Sessions panels under the My application node.

Notice that your item is stored in the session and displayed on the page.

Viewing the Session in Other VMs

To see the session data automatically made consistent and available in the other application server, click the Server link in the colored box.



You have the following items in your cart:

1. Twin peaks tapes

Please enter item to add or remove:
Add Item: Beavis & Butt-head Video collection

Current Server: **9082**
Terracotta Sessions: **On**
Items in Cart: **1**
Go to: **Server 9081**

This web application is currently running clustered on Terracotta Sessions, and you have added an item to your cart.

You can click the link on the left to jump over to the other server. Since the session data is being shared across servers, your cart will be preserved.

If you just jumped over from the other server, your cart has maintained its contents because Terracotta Sessions replicated the session data. To view the session data, connect to the cluster using the Developer Console, then click the Sessions button to open the Sessions panels under the My application node.

Notice that your product browsing history is intact, even on the other application server. Web Sessions is automatically and transparently clustering your session data and sharing it between the two server instances on demand. Make another change to your cart and then click the other server link. You will observe that the session data is shared across the two server instances at a fine-grained, field-change level, independent of the application code.

4 Using Web Sessions

Optional Configuration Attributes

While Terracotta Web Sessions is designed for optimum performance with the configuration you set at installation, in some cases it may be necessary to use the configuration attributes described in the following sections.

Session Locking

By default, session locking is off in Terracotta Web Sessions. If your application requires disabling concurrent requests in sessions, you can enable session locking.

To enable session locking, add an `<init-param>` block as follows:

```
<filter>
<filter-name>terracotta-filter</filter-name>
<filter-class>org.terracotta.session.TerracottaContainerSpecificSessionFilter</filter-class>
<init-param> <param-name>tcConfigUrl</param-name>
<param-value>localhost:9510</param-value>
</init-param> <init-param>
<param-name>sessionLocking</param-name>
<param-value>true</param-value> </init-param>
</filter>
```

If you enable session locking, see *Deadlocks When Session Locking Is Enabled* in "Common Issues" on page 28.

Synchronous Writes

Synchronous write locks provide an extra layer of data protection by having a client node wait until it receives acknowledgement from the Terracotta Server Array that the changes have been committed. The client releases the write lock after receiving the acknowledgement. Note that enabling synchronous write locks *can substantially raise latency rates, thus degrading cluster performance*.

To enable synchronous writes, add an `<init-param>` block as follows:

```
<filter>
<filter-name>terracotta-filter</filter-name>
<filter-class>org.terracotta.session.TerracottaContainerSpecificSessionFilter</filter-class>
<init-param> <param-name>tcConfigUrl</param-name>
<param-value>localhost:9510</param-value>
</init-param> <init-param>
<param-name>synchronousWrite</param-name>
<param-value>true</param-value> </init-param>
</filter>
```

Sizing Options

Web Sessions gives you the option to configure both heap and off-heap memory tiers.

- Memory store - Heap memory that holds a copy of the hottest subset of data from the off-heap store. Subject to Java garbage collection (GC).
- Off-heap store - Limited in size only by available RAM. Not subject to Java GC. Can store serialized data only. Provides overflow capacity to the memory store.

Note: If using off-heap, see information about allocating direct memory in "Configuring Storage Tiers" in the *BigMemory Max Configuration Guide*.

To set the sizing attributes, add one or both `<init-param>` blocks to your web.xml as follows:

```
<filter>
<filter-name>terracotta-filter</filter-name>
<filter-class>org.terracotta.session.TerracottaContainerSpecificSessionFilter</filter-class>
<init-param> <param-name>tcConfigUrl</param-name>
<param-value>localhost:9510</param-value>
</init-param> <init-param>
<param-name>maxBytesOnHeap</param-name>
<param-value>128M</param-value> </init-param>
<init-param>
<param-name>maxBytesOffHeap</param-name>
<param-value>2G</param-value> </init-param>
</filter>
```

Nonstop and Rejoin Options

The nonstop timeout is the number of milliseconds an application waits for any cache operation to return before timing out. Nonstop allows certain operations to proceed on clients that have become disconnected from the cluster. One way clients go into nonstop mode is when they receive a "cluster offline" event. Note that a nonstop cache can go into nonstop mode even if the node is not disconnected, such as when a cache operation is unable to complete within the timeout allotted by the nonstop configuration.

To set the nonstop timeout, add an `<init-param>` block to your `web.xml` as follows:

```
<filter>
<filter-name>terracotta-filter</filter-name>
<filter-class>org.terracotta.session.TerracottaContainerSpecificSessionFilter</filter-class>
<init-param> <param-name>tcConfigUrl</param-name>
<param-value>localhost:9510</param-value>
</init-param> <init-param>
<param-name>nonStopTimeout</param-name>
<param-value>30000</param-value> </init-param>
</filter>
```

Tuning Nonstop Timeout

You can tune the timeout value to fit your environment. The following information provides additional guidance for choosing a `nonStopTimeout` value:

- In an environment with regular network interruptions, consider increasing the timeout value to prevent timeouts for most of the interruptions.
- In an environment where cache operations can be slow to return and data is required to always be in sync, increase timeout value to prevent frequent timeouts. For example, a locking operation may exceed the nonstop timeout while waiting for a lock. This would trigger nonstop mode only because the lock couldn't be acquired in time. Setting the method's timeout to less than the nonstop timeout avoids this problem.
- If a nonstop cache employs bulk loading, be aware that a timeout multiplier may be applied by the bulk-loading method.

Concurrency

The concurrency attribute allows you to set the number of segments for the map backing the underlying server store managed by the Terracotta Server Array. If concurrency is not explicitly set (or set to "0"), the system selects an optimized value.

To configure or tune concurrency, add an `<init-param>` block to your web.xml as follows:

```
<filter>
<filter-name>terracotta-filter</filter-name>
<filter-class>org.terracotta.session.TerracottaContainerSpecificSessionFilter</filter-class>
<init-param> <param-name>tcConfigUrl</param-name>
<param-value>localhost:9510</param-value>
</init-param> <init-param>
<param-name>concurrency</param-name>
<param-value>256</param-value> </init-param>
</filter>
```

Tuning Concurrency

The server map underlying the Terracotta Server Array contains the data used by clients in the cluster and is segmented to improve performance through added concurrency. Under most circumstances, the concurrency value is optimized by the Terracotta Server Array and does not require tuning.

If an explicit and fixed segmentation value must be set, use the concurrency attribute, making sure to set an appropriate concurrency value. A too-low concurrency value could cause unexpected eviction of elements. A too-high concurrency value may create many empty segments on the Terracotta Server Array (or many segments holding a few or just one element).

The following information provides additional guidance for choosing a concurrency value:

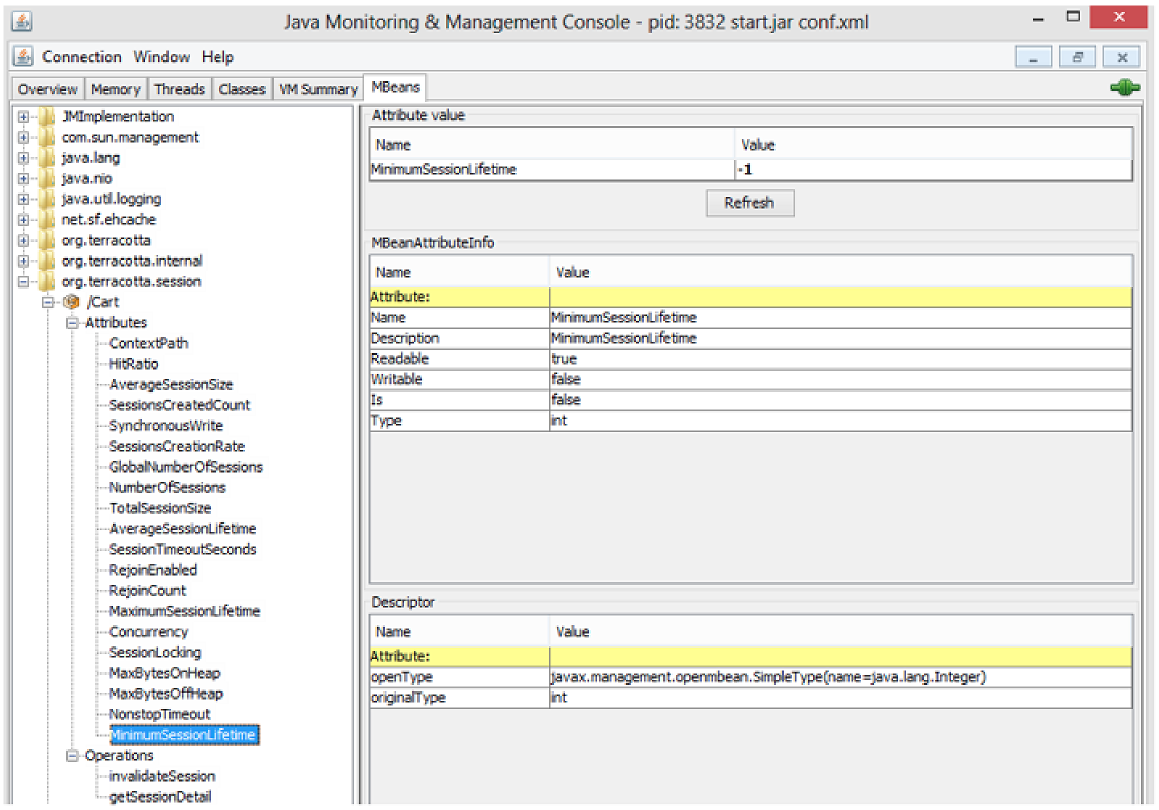
- In general, the concurrency value should be no less than the number of active servers in the Terracotta Server Array, and optimally at least twice the number of active Terracotta servers.
- With extremely large data sets, a high concurrency value can improve performance by hashing the data into more segments, which reduces lock contention.
- In environments with very few cache elements, set concurrency to a value close to the number of expected elements.

Managing Web Sessions Using JMX

Terracotta exposes JMX MBeans to gather statistics. You can view the statistics using any JMX monitoring application such as JConsole or VisualVM.

To use JMX with Web Sessions, start your management console and connect it to your application server.

The console below displays the Web Sessions MBeans.


Java Monitoring & Management Console - pid: 3832 start.jar conf.xml
Connection Window Help
Overview Memory Threads Classes VM Summary MBeans
Refresh

- JMImplementation
- com.sun.management
- java.lang
- java.nio
- java.util.logging
- net.sf.ehcache
- org.terracotta
- org.terracotta.internal
- org.terracotta.session
  - /Cart
    - Attributes
      - ContextPath
      - HitRatio
      - AverageSessionSize
      - SessionsCreatedCount
      - SynchronousWrite
      - SessionsCreationRate
      - GlobalNumberOfSessions
      - NumberOfSessions
      - TotalSessionSize
      - AverageSessionLifetime
      - SessionTimeoutSeconds
      - RejoinEnabled
      - RejoinCount
      - MaximumSessionLifetime
      - Concurrency
      - SessionLocking
      - MaxBytesOnHeap
      - MaxBytesOffHeap
      - NonstopTimeout
      - MinimumSessionLifetime
    - Operations
      - invalidateSession
      - getSessionDetail

Attribute value

Name	Value
MinimumSessionLifetime	-1

MBeanAttributeInfo

Name	Value
Attribute:	
Name	MinimumSessionLifetime
Description	MinimumSessionLifetime
Readable	true
Writable	false
Is	false
Type	int

Descriptor

Name	Value
Attribute:	
openType	javax.management.openmbean.SimpleType(name=java.lang.Integer)
originalType	int

The following table describes the Web Sessions MBeans.

MBean	Data Type	Description
Average Session Lifetime	Integer	Average local session lifetime (in seconds) @return average session lifetime, or -1 if no sessions have expired.
Average Session Size	Long	Average local session size (in bytes).
Concurrency	(Integer)	Number of segments for the map backing the underlying server store managed by the Terracotta Server Array. If concurrency is not explicitly set (or set to "0"), the system selects an optimized value.
Context Path	String	Context path for this servlet context.

MBean	Data Type	Description
Global Number Of Sessions	Long	Total number of live sessions that exist in the cluster.
Hit Ratio	Double	Local hit ratio of the sessions cache @return hit ratio n, or NaN if unavailable.
Max Bytes Off Heap	Long	Maximum number of bytes allowed in the off-heap tier.
Max Bytes On Heap	Long	Maximum number of bytes allowed in the heap tier.
Maximum Session Lifetime	Integer	Maximum local session lifetime (in seconds) @return maximum session lifetime, or -1 if no sessions have expired.
Minimum Session Lifetime	Integer	Minimum local session lifetime (in seconds) @return minimum session lifetime, or -1 if no sessions have expired.
Nonstop Timeout	Long	Number of milliseconds an application waits for any cache operation to return before timing out, or -1 if no timeout is configured.
Number Of Sessions	Long	Number of session objects in local memory.
Rejoin Enabled	Boolean	Whether client is configured to automatically rejoin its cluster if ejected.
Session Locking	Boolean	Whether session locking is enabled.
Session Timeout Seconds	Integer	Number of seconds a session waits before timing out, or -1 if no timeout is configured.
Sessions Created Count	Long	Number of sessions created locally since initial startup.
Sessions Creation Rate	Long	Rate of local session creation since initial startup. This number represents the number sessions created in the previous minute.

MBean	Data Type	Description
Synchronous Write	Boolean	Whether configured to synchronously flush enclosed changes to the Terracotta Server Array, blocking the unlocking thread until changes have been acknowledged as committed.
Total Session Size	Long	Total local session size (in bytes).

Managing Web Sessions Using the REST API

The REST API is enabled automatically and no setup is required. For general information about the REST API, refer to the *Terracotta REST Developer Guide*. For Web Sessions, requests can be made using `/agents/sessions` with `/contexts` and `/details` options.

Send requests to the management port of any server in the Terracotta Server Array, for example:

```
http://<server-id>:9540/tc-management-api/v2/agents/sessions/contexts
```

The above request will return the available contexts on all nodes, including the agentIds and the context paths.

To request details about a particular context on all nodes, add the context path, for example:

```
/agents/sessions/contexts/Cart
```

The above request will return details about the context "/Cart" on all nodes, including agentIds, number of sessions, hit ratios, and sessions counts.

Note that context paths can include more than one directory, for example "/aaa/bbb."

When working with `/details` you will need the agentIds. AgentIds are returned with `/context` requests. Note that the agentId is not unique to a node. Each context generates its own agentId.

Once you have the agentIds, you can view session details or invalidate a session.

To view session details, such as creation time, last access time, and max inactive interval:

```
GET: /agents;ids=xxx/sessions/details/Cart?sessionId=1234
```

To invalidate a session:

```
DELETE: /agents;ids=xxx/sessions/details/Cart?sessionId=1234
```

Common Issues

The following sections summarize common issues than can be encountered when clustering Web Sessions.

Sessions Time Out Unexpectedly

Sessions that are set to expire after a certain time instead seem to expire at unexpected times, and sooner than expected. This problem can occur when sessions hop between nodes that do not have the same system time. A node that receives a request for a session that originated on a different node still checks local time to validate the session, not the time on the original node. Adding the Network Time Protocol (NTP) to all nodes can help avoid system-time drift. However, note that having nodes set to different time zones can cause this problem, even with NTP.

This problem can also cause sessions to time out later than expected, although this variation can have many other causes.

Changes Not Replicated

Terracotta Web Sessions must run in serialization mode. In serialization mode, sessions are clustered, and your application must follow the standard servlet convention on using setAttribute() for mutable objects in replicated sessions.

Deadlocks When Session Locking Is Enabled

In some containers or frameworks, it is possible to see deadlocks when session locking is in effect. This happens when an external request is made from inside the locked session to access that same session. This type of request fails because the session is locked.

Events Not Received on Node

Most Servlet spec-defined events will work with Terracotta clustering, but the events are generated on the node where they occur. For example, if a session is created on one node and destroyed on a second node, the event is received on the second node, not on the first node.