Clustering Configuration for Jetty

{scrollbar}

Standard and Geronimo Specific Deployment Descriptor Updates

In order to cluster a Web application, standard and Geronimo specific deployment descriptors must be updated as follows:

- standard deployment descriptor, i.e. web.xml: the Web application must be marked as distributable. In other words, the optional "distributable" element must be defined.
- Geronimo specific deployment descriptor, i.e. geronimo-web.xml: the "clustering-wadi" substitution group must be defined. This element provides a Web application specific clustering configuration, which overrides the default clustering configuration. The default clusering configuration is defined by the GBean WADIJettyClusteringBuilder declared by the org.apache.geronimo.configs/jetty6-clustering-builder-wadi//car module.

Clustering Configuration Overview

Four configurations are of interest to us:

- *org.apache.geronimo.configs/clustering//car*: it defines the runtime dependencies that a clustered Web application must have. For instance, it imports the geronimo-clustering JAR into the classpath of such Web applications. Also, it defines a BasicNode GBean, which defines an unique name for this Geronimo instance in a cluster.
  In order to give a specific name to a Geronimo instance, this module can be reconfigured by updating the clusterNodeName property defined within var/config/config-substitutions.properties: clusterNodeName=MyOwnNodeName

- *org.apache.geronimo.configs/jetty6-clustering-wadi//car*: it defines the runtime dependencies of WADI, which is the actual clustering provider. Also, it defines a WADI implementation of the Jetty6 clustering contracts.
- *org.apache.geronimo.configs/jetty6-clustering-builder-wadi//car*: it defines the build time dependencies along with a WADIJettyClusteringBuilder GBean, which declares the default clustering configuration and knows how to process the "clustering-wadi" element.

- *org.apache.geronimo.configs/wadi-clustering//car*: it defines WADI specific GBeans used to further configure the behavior of a clustered application. This is in this module, by overriding the DefaultBackingStrategyFactory GBean, that the default number of replicas (set to 2 out-of-the-box) to be maintained across the cluster for a given session is configured.

Creating a new Geronimo Instance

The simplest way to create a new Geronimo instance is to:

- create a new directory, say node2, under the Geronimo installation dir;
- copy the var directory into it; and
- change the PortOffset property within node2/var/config/config-substitutions.properties. For instance, the creation of a new instance named node2 could be achieved like this: cd $GERONIMO_HOME mkdir node2 cp -r var node2 perl -pi -e 's/PortOffset=0/PortOffset=1/' node2/var/config/config-substitutions.properties

- give to this instance a unique name by updating the clusterNodeName property within node2/var/config/config-substitutions.properties: cl usterNodeName=NODE2 In order to start this new instance, you need to set the org.apache.geronimo.server.name system property to the name of the new instance. In the case of the above example, node2 can be started like this from the gshell prompt: geronimo/start-server -G server.name=node2 -b

- ensure that the system property java.net.preferIPv4Stack is set to true when a Geronimo instance is started. Uncomment the following line within etc/rc.d/start-server,default.groovy: // Uncomment this property so that Tribes work on Mac OS X command.properties['java. net.preferIPv4Stack'] = true System Property java.net.preferIPv4Stack
  The system property java.net.preferIPv4Stack must be set to true otherwise Tribes, the group communication used by WADI, fails (at least on MAC OS X).

Deploying a Demonstration Web Application and Testing Clustering

Deploying a Demonstration Web application

- Start an instance, e.g. the out-of-the-box one: geronimo/start-server -D node.name=red -b System Property node.name

The system property node.name is only required by a WADI Web application, which will be deployed later to verify the installation.

- Start the org.apache.geronimo.configs/jetty6-clustering-builder-wadi//car module if need be: deploy/start org.apache.geronimo.configs /jetty6-clustering-builder-wadi/2.0-SNAPSHOT/car Why should this module be started
  If this module is not started, then the "clustering-wadi" element is not recognized and the deployment fails.

- Deploy the Demonstration Web application and start it.
  WADI provides a demonstration Web application, which can be downloaded from this link: http://snapshots.repository.codehaus.org/org /codehaus/wadi/wadi-webapp/2.0-SNAPSHOT/wadi-webapp-2.0-20080318.132141-11.war curl -o wadi-webapp-2.0-SNAPSHOT.war http://snapshots.repository.codehaus.org/org/codehaus/wadi/wadi-webapp/2.0-SNAPSHOT/wadi-webapp-2.0-20080318.132141-11.war // and from gshell deploy/distribute /<full path>/wadi-webapp-2.0-SNAPSHOT.war deploy/start org.codehaus.wadi/wadi-webapp/2.0-SNAPSHOT/war

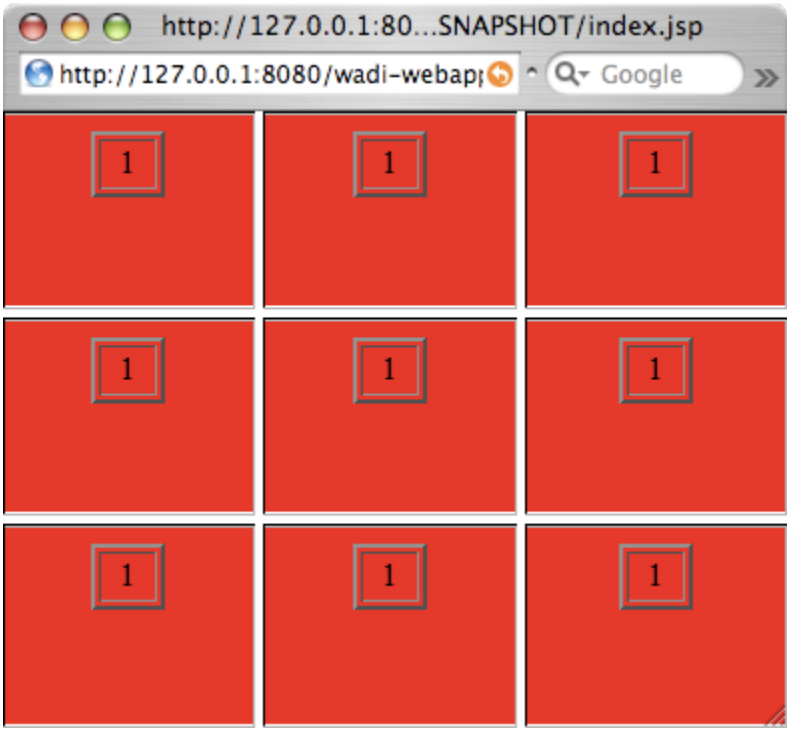
- Start a second instance, e.g. the node2 instance previously created: geronimo/start-server -D node.name=yellow -G server.name=node2 -b

- Start the Web application deployed at step 2. deploy/start --port 1100 org.codehaus.wadi/wadi-webapp/2.0-SNAPSHOT/war

Testing Clustering

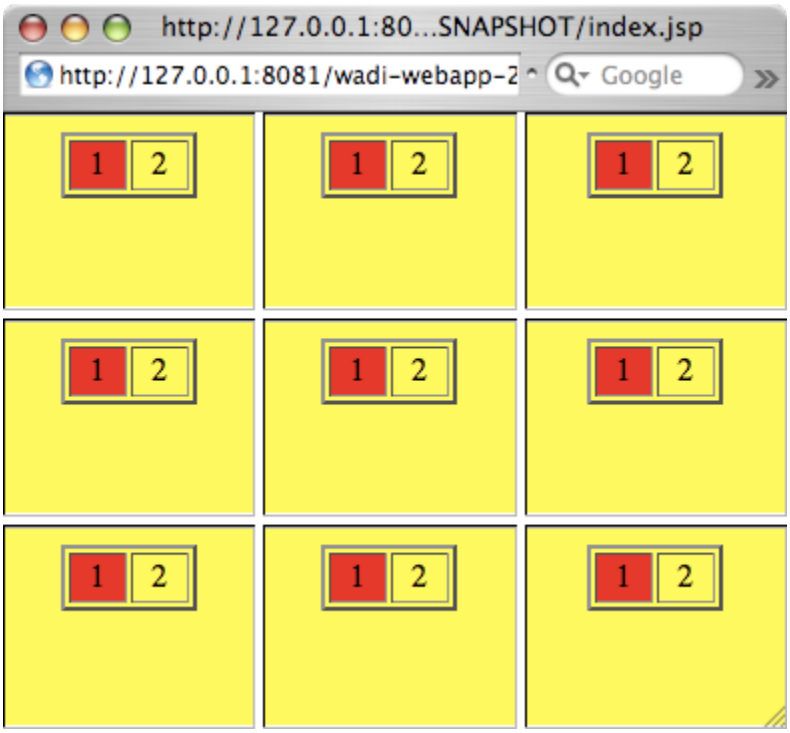
These two above instances are now running a clustered version of the demonstration Web application. This application implements a visual way to observe where a session is updated and where it is accessed.

- Let's create a session on the out-of-the-box instance, the red node, by accessing this URL: http://127.0.0.1:8080/wadi-webapp/index.jsp
  You should see:



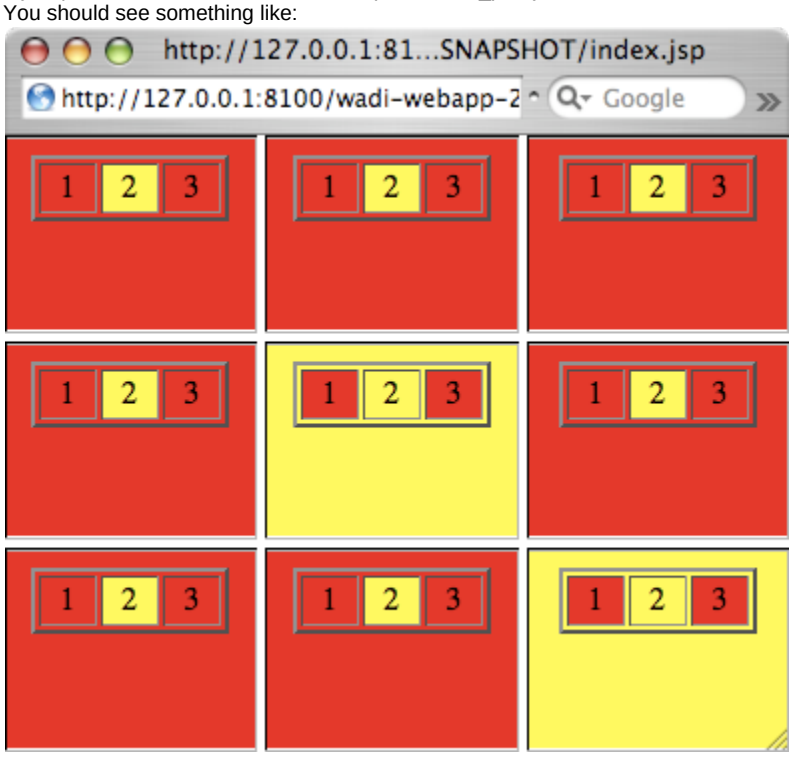
  - The 1 in red means that the session has been updated on the red node. The red background of each frame indicates that the session has been accessed by the red node.
  - In your browser, you can verify that the JSESSIONID cookie ends with NODE. This information can be leveraged by load-balancers to provide sticky load-balancing.

- Let's now request the same session from node2, the yellow node: http://127.0.0.1:8081/wadi-webapp/index.jsp
  You should see:



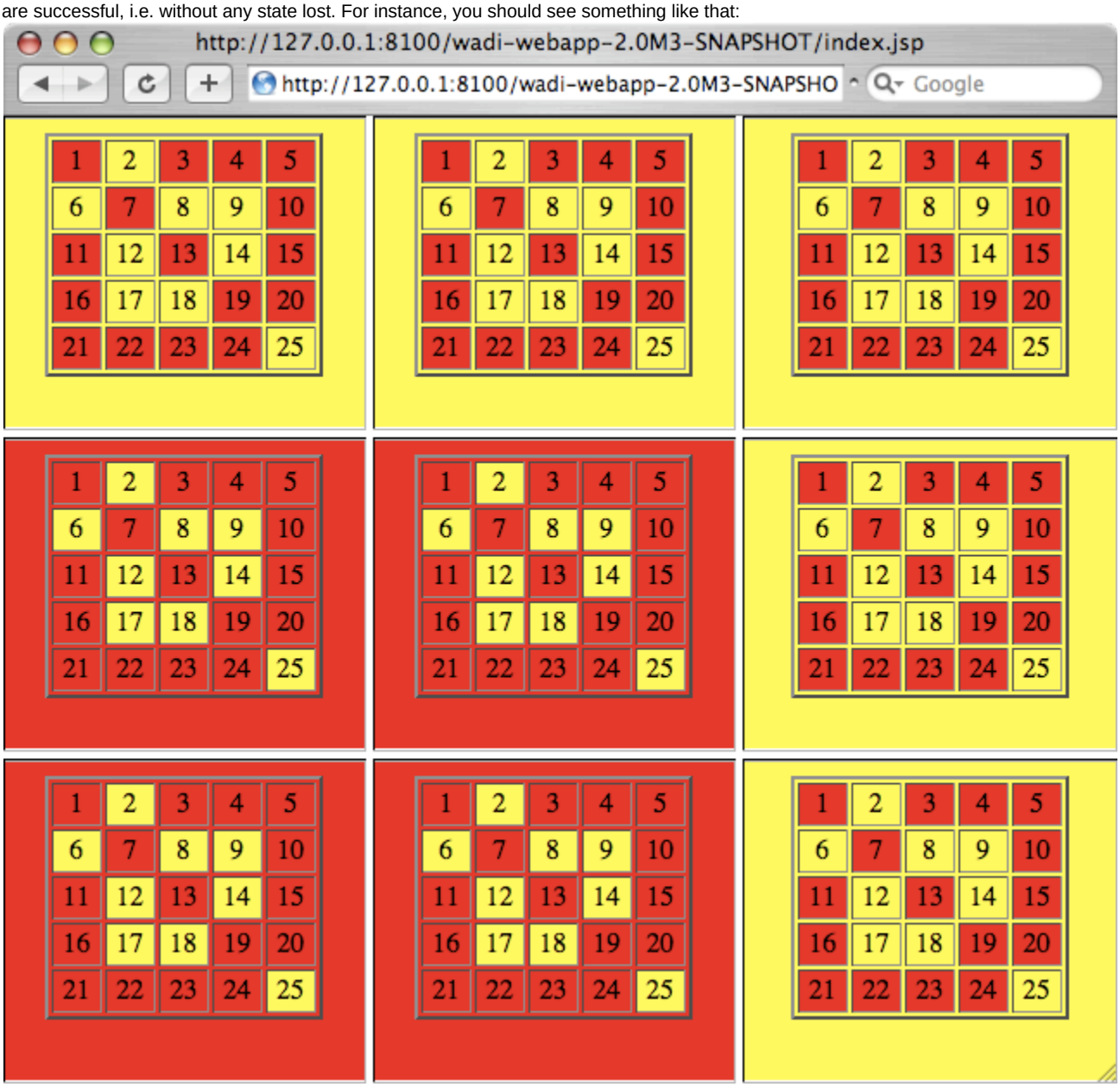
  - The 2 in yellow means that the session has been updated on the yellow node. The session state, i.e. the 1 in red, has been migrated from the red to the yellow node. The yellow background of each frame indicates that the session has been accessed by the yellow node.
  - In your browser, you can verify that the JSESSIONID cookie for the /wadi path ends with NODE2. As expected, the session cookie has been updated to reflect the new location of the session. This is not working as expected
    If you see a 1 instead of a 2, then the nodes do not see each other. The first thing to verify is that the system property java.net.preferIPv4Stack is indeed set to true. If it is set to true, then it is likely that multicasting is not properly working on the boxes running the nodes. Look at your firewall and network configurations to ensure that multicasting is allowed.

- If you put these two instances behind Apache mod_proxy and access the same URL: http://127.0.0.1:8100/wadi-webapp/index.jsp
  You should see something like:



- The 3 may be in yellow or red and the background color of each frame is a mix of yellow and red. This mix of colors indicates that the session has been migrated between the red and yellow instances depending on the target instance selected by mod_proxy for each frame.

- You can access repetitively the reverse proxy URL to confirm the robustness of the session migration implementation. All the requests are successful, i.e. without any state lost. For instance, you should see something like that:



- To confirm that replication is working fine, you can kill -9 a node and access the URL of the remaining node and observe that the state is properly restored.