Yellowfin Clustering Guide

Release 7.0

February 2014

Yellowfin ®
Release 7
Clustering Guide

Version: 1.3
Published: February 2014

Table of Contents

Yellowfin Clustering Overview 4

Application Level Messaging 4

Container Level Session Replication 5

Yellowfin Database Clustering 6

Licensing 6

Installing a Yellowfin Cluster 7

File System Replication 7

Reinstallation 7

Yellowfin for Application Messaging 8

Modifications to web.xml 10

web.xml file parameters 11

Background Tasks 13

System Tasks 13

Task Scheduler 13

Container Level Session Replication 14

Caveats 14

Modifications to web.xml 15

Modifications to server.xml 15

Modifications to ROOT.xml 17

•

Yellowfin Clustering Overview

Yellowfin can be clustered on multiple servers to allow for high-availability and load-balancing. The actual load-balancing (multiplexing requests from external requests) can be achieved with a hardware load-balancer, or load-balancing software. This guide will outline the modifications required to Yellowfin's configuration to enable clustering, but not the external environment that multiplexes the incoming requests.

It is required that the load-balancing infrastructure delivers network traffic to the Yellowfin application server transparently, as if the packets were connecting directly to the application server directly. If Session Replication is not enabled, the load-balancing infrastructure will need to provide “sticky-session” functionality, where traffic for a user’s session should be sent to the same node.

Yellowfin supports clustering at 2 levels:

1. Application
2. Container

Application Level Messaging

Yellowfin will communicate with other nodes when application wide messages need to be sent. This communicates messages like refreshing caches, and logging out concurrent users. This is controlled by a background servlet in Yellowfin that will communicate with other nodes.

Yellowfin also has cluster safe webservice calls, which allows a webservice Single-Sign-On request to come from an application to any cluster node, and return a token id. The token id will then be valid on any node that the token id is sent to.

Container Level Session Replication

The Yellowfin application has also been written to allow user session information to be replicated onto multiple cluster nodes concurrently. User session information is the memory footprint that is stored on the server for each user session. Enabling Session Replication allows for the users session to continue, even if the cluster node they were connected to experiences a failure.

Without Session Replication, failure of a cluster node will destroy a users session, and they will need to login again to another node.

Session Replication is a function of the Java Container. An example of how this can be achieved with Tomcat will be discussed in this document.

Yellowfin Database Clustering

A Yellowfin Cluster must share a single Yellowfin Database instance. The database could be a single database instance, that is shared across all Yellowfin nodes, or a database that is clustered itself. It is assumed that database clustering / replication is done by the client, as every database implements this in a different way. Database Clustering / Replication should be transparent to Yellowfin, as Yellowfin will have a single database URL for the entire cluster.

Licensing

The Yellowfin Licence file is stored in the Yellowfin database, and because of this, the Licence must contain a reference to all hostnames in the cluster. A clustered licence can be requested from Yellowfin.

Installing a Yellowfin Cluster

As a Yellowfin Cluster requires only one database instance, it means that the Yellowfin Installation process will differ slightly when installing a cluster.

The first Yellowfin Cluster node can be installed the same as standalone instance of Yellowfin. The command-line or GUI installer can be used.

Additional Yellowfin Cluster nodes do not require the Yellowfin database. There are two options to options for installing additional nodes:

File System Replication

Assuming that the file system structure on each node is the same, a new node can be created by duplicating the Yellowfin installation directory onto another computer. The only change required to each node, are changes to the web.xml file which is discussed later in this document.

Reinstallation

After the initial node is installed, and the Yellowfin database is created, the Yellowfin installer can be used for the installation of subsequent nodes, however, this process will create additional Yellowfin databases. The installer can be allowed to create these databases, but these can be removed once the installation process is finished. Each node's web.xml can then be modified to point to the database created with the first nodes installation.

Yellowfin for Application Messaging

Each Yellowfin node must have the ClusterManagement servlet enabled for the node to become "Cluster Aware". The ClusterManagement servlet is enabled by adding an additional configuration block to the web.xml file on each node.

The Application Messaging is performed using webservices, allowing different nodes to initiate cache clearing or changes to licensing details.

Multicast Cluster Messaging (DYNAMIC mode)

Yellowfin inter-node messaging is now handled by a highly configurable multicast library called JGroups. Using this method will automatically find other nodes in the cluster sharing the same Yellowfin database.
Dynamic mode also allows for multiple cluster instances to run on the same server.

The default configuration of JGroups uses UDP multicast messages to determine group membership and find new nodes. There may be environments where these types of messages can not be sent. For example, Amazon does not allow multicast packets on its internal network between nodes. The Multicast Cluster Messaging adapter allows you to pass a XML configuration file to configure JGroups to use other methods for node discovery. This file can be referenced by passing the path to the BroadcastConfiguration servlet parameter within the ClusterManagement servlet.

Webservice Cluster Messaging (LEGACY mode)

Yellowfin's legacy cluster messaging is handled by AXIS webservices. This requires that all nodes be defined at startup, and that the service end-point, port, user and password be defined in each node's web.xml file. Legacy mode does not allow cluster instances to reside on the same host.

Modifications to web.xml for DYNAMIC mode.

The following servlet definition needs to be added to the web.xml on each node:

```
<!-- Cluster Management -->
<servlet>
	<servlet-name>ClusterManagement</servlet-name>
	<servlet-class>com.hof.mi.servlet.ClusterManagement</servlet-class>
	<init-param>
		<param-name>ClusterType</param-name>
		<param-value>DYNAMIC</param-value>
	</init-param>
	<init-param>
		<param-name>SerialiseWebserviceSessions</param-name>
		<param-value>true</param-value>
	</init-param>
	<init-param>
		<param-name>CheckSumRows</param-name>
		<param-value>true</param-value>
	</init-param>
	<init-param>
		<param-name>EncryptSessionId</param-name>
		<param-value>true</param-value>
	</init-param>
	<init-param>
		<param-name>EncryptSessionData</param-name>
		<param-value>true</param-value>
	</init-param>
	<load-on-startup>11</load-on-startup>
</servlet>
```

Modifications to web.xml for LEGACY mode.

The following servlet definition needs to be added to the web.xml on each node:

```
<!-- Cluster Management -->
<servlet>
	<servlet-name>ClusterManagement</servlet-name>
	<servlet-class>com.hof.mi.servlet.ClusterManagement</servlet-class>
	<init-param>
		<param-name>ServiceUser</param-name>
		<param-value>admin@yellowfin.com.au</param-value>
	</init-param>
	<init-param>
		<param-name>ServicePassword</param-name>
		<param-value>test</param-value>
	</init-param>
	<init-param>
		<param-name>ServiceAddress</param-name>
		<param-value>/services/AdministrationService</param-value>
	</init-param>
	<init-param>
		<param-name>ServicePort</param-name>
		<param-value>80</param-value>
	</init-param>
	<init-param>
		<param-name>ClusterHosts</param-name>
		<param-value>
			192.168.4.184
		</param-value>
	</init-param>
	<init-param>
		<param-name>SerialiseWebserviceSessions</param-name>
		<param-value>true</param-value>
	</init-param>
	<init-param>
		<param-name>CheckSumRows</param-name>
		<param-value>true</param-value>
	</init-param>
	<init-param>
		<param-name>EncryptSessionId</param-name>
		<param-value>true</param-value>
	</init-param>
	<init-param>
		<param-name>EncryptSessionData</param-name>
		<param-value>true</param-value>
	</init-param>
	<load-on-startup>11</load-on-startup>
</servlet>
```

web.xml file parameters

The following properties outline the options in the ClusterManagement servlet definition.

Property	Value
`ClusterType`	Either DYNAMIC or LEGACY. DYNAMIC will us multicast messaging and automatically find other nodes in the cluster. LEGACY is the default, and will use webservices to communicate with a defined list of cluster nodes.
`BroadcastConfiguration`	A JGroups configuration file. This allows you to configure cluster messaging to work in environments where multicast networking is not available. This is for DYNAMIC mode only and is optional. By default, JGroups will use the configuration defined in udp.xml.
`ServiceUser`	User that will validate the webservice connection to other nodes. For LEGACY mode only.
`ServicePassword`	Password for the ServiceUser. For LEGACY mode only.
`ServicePasswordEncrypted`	True/False. Is the contents of the ServicePassword encrypted.
`ServiceAddress`	Address of the Yellowfin webservice. For LEGACY mode only.
`ServicePort`	Port on which Yellowfin is running. For LEGACY mode only.
`ClusterHosts`	This is a comma-seperated list of all nodes in the Cluster. These can include IP addresses or hostnames. For LEGACY mode only.
`SerialiseWebserviceSessions`	True/False. This is required if using Single Sign-On on the Cluster. It can serialise tokens to the database so that the token can be accessed from any node.
`CheckSumRows`	True/False. Security option to check sum the serialised webservice session records in the database. This helps prevent modification to the

	table which could lead to the creation of unauthorised sessions in Yellowfin.
`EncryptSessionId`	True/False. Security option to encrypt the serialised webservice session id in the database. This helps prevent modification to the table which could lead to the creation of unauthorised sessions in Yellowfin.
`EncryptSessionData`	True/False. Security option to encrypt the serialised webservice session records in the database. This helps prevent modification to the table which could lead to the creation of unauthorised sessions in Yellowfin.

Background Tasks

Each Yellowfin node will be configured by default to run background tasks, which includes the broadcasting of reports. This potentially could send reports multiple times to end users. Because of this it is recommended that background tasks be enabled on one node only.

System Tasks

Disable the background system tasks by removing (or commenting-out) the following XML block from the web.xml. This will disable system tasks, like Group Flattening, LDAP synchronization, Event Archiving, Document Cleanup and Average Run time calculations.

```
<servlet>
	<servlet-name>SystemTaskManager</servlet-name>
	<servlet-class>com.hof.servlet.SystemTaskManager</servlet-class>
	<load-on-startup>8</load-on-startup>
</servlet>
```

Task Scheduler

Disable the task scheduler by adding the following to the web.xml file, inside the MIStartup Servlet block.
This will disable all tasks visible in the Task Schedule menu within the Yellowfin interface. This includes broadcasting, populating cached filters, scheduled reports and populating cached views.

```
<init-param>
        <param-name>DisableTaskScheduler</param-name>
        <param-value>TRUE</param-value>
</init-param>
```

Container Level Session Replication

Container Level Session Replication allows Yellowfin to be deployed in a fail-safe environment where in the event of a node in the cluster failing, a currently connected Yellowfin user can continue their session on another node. This is achieved with Session Replication at the Application Server level, and is not a function of Yellowfin itself, and hence is not a supported component of Yellowfin. This configuration also requires a load-balancer to evenly distribute incoming requests to different cluster nodes, which will not be addressed in this guide.

Different Application Servers may have different ways of implementing this functionality. This example is shown using Tomcat, which is shipped with Yellowfin.

Setting up session replication consists of the following steps:

1. Modifications to web.xml
2. Modifications to server.xml
3. Modifications to ROOT.xml

More information regarding Tomcat Session Replication can be found at http://tomcat.apache.org

Caveats

There are several caveats to using Session Replication with Yellowfin (or any Java web application). The two main downsides of this are Session Replication overhead, and memory usage.

There is additional overhead in replicating each session to all nodes in the cluster. This happens after each user's request, and by default, only the changed session elements are copied to other nodes. This means all nodes will be doing additional work each time a user does something on any connect node.

The function of Session Replication is to replicate a user's memory footprint on the server, to all nodes in the cluster. This means the sessions for all users in the cluster will be stored on each node, taking up additional memory. Expanding the cluster with additional nodes is then restricted by the amount of memory that each node has.

It should be considered if this overhead is worth the added fail-safe functionality that it brings. If Session Replication is not used, and a node fails, a user's session will be destroyed, but they'll be able to login to another node to continue their work.

Modifications to web.xml

The text “<distributable/>” needs to be added to the web.xml file on each node underneath the “<webapp>” line.

```
<web-app>
	<distributable/>
	<!-- System Event and Debug classes -->
	<listener>
		<listener-class>com.hof.servlet.SysSessionListener</listener-class>
	</listener>
```

Modifications to server.xml

Significant changes need to be made to the server.xml file in the Yellowfin/appserver/conf directory.
Firstly, an addition must be made to the “<engine>” node in the xml.

Change from:

```
<Engine name="Catalina" defaultHost="localhost" >
```

To:

```
<Engine name="Catalina" defaultHost="localhost" jvmRoute="node01">
```

The jvmRoute value must be different for each node. In this example the number behind the word "node" is incremented for each additional node.

Secondly, an additional XML entry must be made within the "<host>" XML block.

Add:

```
<Cluster className="org.apache.catalina.cluster.tcp.SimpleTcpCluster"
managerClassName="org.apache.catalina.cluster.session.DeltaManager"
expireSessionsOnShutdown="false" useDirtyFlag="true"
notifyListenersOnReplication="true">
       <Membership className="org.apache.catalina.cluster.mcast.McastService"
       mcastAddr="228.0.0.4"
       mcastPort="45564"
       mcastFrequency="500"
       mcastDropTime="3000"/>
       <Receiver className="org.apache.catalina.cluster.tcp.ReplicationListener"
       tcpListenAddress="192.168.4.199"
        tcpListenPort="4001"
       tcpSelectorTimeout="100"
       tcpThreadCount="6"/>
       <Sender className="org.apache.catalina.cluster.tcp.ReplicationTransmitter"
       replicationMode="pooled"
        ackTimeout="15000"
       waitForAck="true"/>


             <Valve className="org.apache.catalina.cluster.tcp.ReplicationValve"
       filter=".*\.gif;.*\.js;.*\.jpg;.*\.png;.*\.htm;.* \.html;.*\.css;.*\.txt;"/>


             <Deployer
className="org.apache.catalina.cluster.deploy.FarmWarDeployer"
       tempDir="/tmp/war-temp/"
       deployDir="/tmp/war-deploy/"
       watchDir="/tmp/war-listen/"
       watchEnabled="false"/>


             <ClusterListener
       className="org.apache.catalina.cluster.session.ClusterSessionListener" />

</Cluster>
```

The IP address “192.168.4.199” should be replaced with the address of the network interface that is to receive session replication messages.

The IP address “228.0.0.4” is a multicast address that session replication messages will be received on. This does not need to be modified, but a linux/unix server may need to register a valid multicast route to receive broadcasts on this address. This can be done by running the following command on the console:

```
sudo route add -net 224.0.0.0 netmask 224.0.0.0 dev eth0
```

This syntax may be different for different flavours of linux / unix. This example uses “eth0” as the network device that will be used for this route.

Modifications to ROOT.xml

Remove the lines from the ROOT.xml in the Yellowfin/appserver/conf/Catalina/localhost directory:

```
<Manager className="org.apache.catalina.session.PersistentManager" debug="0"
distributable="false" saveOnRestart="false">
      <Store className="org.apache.catalina.session.FileStore" />
</Manager>
```

These lines were previously added to suppress session serialisation to disk in Yellowfin.