WSO2 Carbon Clustering

LabKit:

Prerequisites and

Hands-on Exercises

**Goal**

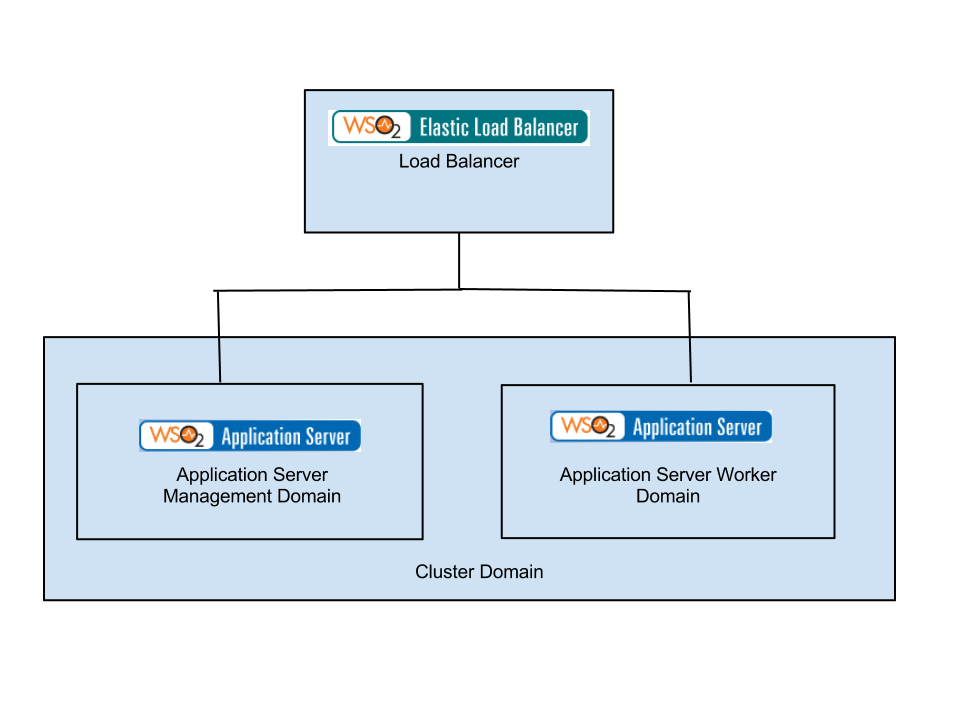
We are going to setup a cluster in our local machines.

The cluster will be composed of these product instances:

* Load Balancer – used to load balance the requests between Application Server cluster nodes.
* WSO2 Application Server management node – all deployments and configurations are performed through this node.
* WSO2 Application Server worker node – Application Server requests are served by this node.

Keep in mind that although those steps are for clustering WSO2 AS, they apply to all WSO2 products.

In the end of this labkit, the worker/manager cluster set up will be as follows:

****

**Pre-requisites:**

1. Download WSO2 App Server 5.2.1
   * <http://docs.wso2.com/display/AS521/Getting+Started>
2. Extract downloaded WSO2 App Server archive to a local directory (2 times).
   * The extracted directories will be AS\_WRKR\_HOME, AS\_MGR\_HOME
3. Download WSO2 Elastic Load Balancer 2.1.1
   * <https://docs.wso2.com/display/ELB211/Getting+Started>
4. Extract downloaded WSO2 Elastic Load Balancer archive to a local directory
   * The extracted directory will be ELB\_HOME

**Setting up WSO2 Elastic Load Balancer**

The WSO2 Elastic Load Balancer automatically distributes incoming traffic across multiple WSO2 product instances. It enables you to achieve greater levels of fault tolerance in your cluster, and provides the required balancing of load needed to distribute traffic.

You configure the ELB with the overall definition of the cluster in mind and define how it should distribute the load.

Open the <ELB\_HOME>/repository/conf/loadbalancer.conf file, locate the “appserver” part and edit it as follows:

**appserver {**

**domains{**

# We specify below the domain name which is used to identify

# the cluster. On startup, nodes that are configured with

# this domain name can join the cluster.

**wso2.as.domain {**

# The tenant\_range attribute is used to handle

# tenant-aware load-balancing. This attribute allows us

# to partition tenants into several clusters, so that

# when there is a large number of tenants to work with,

# we can instruct each cluster to work only with a

# particular tenant or a few selected tenants. Here,

# we are not enabling tenant partitioning, so we have

# used an asterisk (\*) in front of tenant\_range to

# represent all possible tenants.

**tenant\_range \*;**

# The ELB creates one group management agent per

# cluster to manage the service groups.

# We use the group\_mgt\_port attribute to specify

# the port for this cluster's group management agent.

**group\_mgt\_port 4500;**

# All the service requests must be routed to the worker

# nodes through the ELB, which is the front end to the

# entire cluster. We specify the worker sub-domain and

# use the hosts attribute to configure the publicly

# accessible host name (as.wso2.com) that clients

# use to send their requests to the cluster.

# We will map the host to the IP address of the ELB.

**worker {**

**hosts as.wso2.com;**

**}**

# Below we provide access to the management node from

# outside the private network so external clients can

# upload applications and perform other management tasks.

# We configure the mgt sub-domain and we will map the

# host to the IP address of the ELB.

**mgt {**

**hosts mgt.as.wso2.com;**

**}**

**}**

**}**

**}**

In summary, we have configured the load balancer to handle requests sent to as.wso2.com and to distribute the load among the worker nodes in the worker sub-domain of the wso2.as.domain cluster.

**Setting up cluster configuration on the ELB**

Previously, we configured several properties of the cluster such as domain name and sub-domain, but we didn’t define them there. We now define these properties as we build the cluster.

1. Open the <ELB\_HOME>/repository/conf/axis2/axis2.xml file.
2. Locate the clustering section and verify or configure the properties as follows (some of these properties are already set correctly by default):  
   1. Enable clustering for this node:   
      <clustering class="org.wso2.carbon.core.clustering.hazelcast.HazelcastClusteringAgent" enable="true">
   2. Set the membership scheme to wka to enable the Well Known Address registration method (this node will send cluster initiation messages to WKA members that we will define later):  
      <parameter name="membershipScheme">wka</parameter>
   3. Specify a domain name for the ELB node (note that this domain is for potentially creating a cluster of ELB nodes and is not the cluster of AS nodes that the ELB will load balance):  
      <parameter name="domain">wso2.carbon.lb.domain</parameter>
   4. Specify the port used to communicate with this ELB node. If this port number is already assigned to another server, the clustering framework will automatically increment this port number. However, if two servers are running on the same machine, you must ensure that a unique port is set for each server.  
      <parameter name="localMemberPort">4000</parameter>  
        
      Important : The localMemberPort should NOT be the same value as the group\_mgt\_port value specified in the loadbalancer.conf file.
   5. Specify the name of the host or IP address used to communicate with the ELB node. This is the host name or IP address that ELB uses to communicate with the members of cluster. In other words, ELB advertises itself to the outside world using the value given in localMemberHost. When you specify a host name as the value of this parameter, you must follow a name resolution method such as /etc/hosts mapping or DNS. If you use an IP, you must specify that as the WKA member to declare in other products.  
      <parameter name="localMemberHost">elb.as.wso2.com</parameter>

**Mapping the host name to the IP**

In the ELB, we configured 2 host names in loadbalancer.conf to front service requests. We must now map these host names to the actual IP addresses.

Open the server's /etc/hosts file and add the following line:

MGT\_ mgt.as.wso2.com

127.0.0.1 as.wso2.com

**Starting the ELB server**

Start the ELB server by typing the following command in the terminal:

$ sh <ELB\_HOME>/bin/wso2server.sh

The ELB should print logs to the server console indicating that the cluster initialization is complete.

**Configuring the Manager Node**

Configuring clustering for the manager node is similar to the way you configured it for the load balancer node, but the localMemberPort is 4100 instead of 4000, and you define the load balancer node as the well-known member.

1. Open the <AS\_MGR\_HOME>/repository/conf/axis2/axis2.xml file.
2. Locate the clustering section and verify or configure the properties as follows (some of these properties are already set correctly by default):
   1. Enable clustering for this node:  
      <clustering class="org.wso2.carbon.core.clustering.hazelcast.HazelcastClusteringAgent" enable="true">
   2. Set the membership scheme to wka to enable the well-known address registration method (this node will send cluster initiation messages to WKA members that we will define later):  
      <parameter name="membershipScheme">wka</parameter>
   3. Specify the name of the cluster this node will join:  
      <parameter name="domain">wso2.as.domain</parameter>
   4. Specify the host used to communicate cluster messages:  
      <parameter name="localMemberHost">127.0.0.1</parameter>
   5. Specify the port used to communicate cluster messages:  
      <parameter name="localMemberPort">4100</parameter>
   6. Add the following lines to the <members> group:  
      <members>  
       <member>  
       <hostName>127.0.0.1</hostName>  
       <port>4500</port>  
       </member>  
      </members>
   7. Change the following clustering properties. Ensure that you set the value of the subDomain as mgt to specify that this is the manager node, which will ensure that traffic for the manager node is routed to this member.  
      <parameter name="properties">  
       <property name="backendServerURL" value="https://${hostName}:${httpsPort}/services/"/>  
       <property name="mgtConsoleURL" value="https://${hostName}:${httpsPort}/"/>  
       <property name="subDomain" value="mgt"/>  
      </parameter>

Since we run multiple WSO2 Carbon based products in same host, we must avoid the possible port conflicts. Thus we must change the port offset to avoid port conflicts.

1. Open <AS\_MGR\_HOME>/repository/conf/carbon.xml.
2. Locate the <Ports> tag and appropriately change the value of its sub-tag titled Offset. This value increments based on the number of products you are running. Since the ELB is running on the same server, the value should be 1.  
   <Offset>1</Offset>
3. Locate the <HostName> tag and add the cluster host name:  
   <HostName>as.wso2.com</HostName>
4. Locate the <MgtHostName> tag and uncomment it. Make sure that the management host name is defined as follows:  
   <MgtHostName>mgt.as.wso2.org</MgtHostName>

Configuring the catalina-server.xml file

1. Specify the following configurations in the catalina-server.xml file located in the <AS\_MGR\_HOME>/repository/conf/tomcat/ folder.

<Connector protocol="org.apache.coyote.http11.Http11NioProtocol"

port="9763"

proxyPort="8280"

--------

/>

<Connector protocol="org.apache.coyote.http11.Http11NioProtocol"

port="9443"

proxyPort="8243"

--------

/>

**Starting the AS Manager server**

Start the AS manager server by typing the following command in the terminal:

$ sh <AS\_MGR\_HOME>/bin/wso2server.sh

The AS should print logs to the server console indicating that its initialization is complete.

INFO - HazelcastGroupManagementAgent Member joined [fd94ad87-6ab1-4829-b343-b84172af89a7]: /127.0.0.1:4100

INFO - MemberUtils Added member: Host:127.0.0.1, Remote Host:null, Port: 4100, HTTP:9764, HTTPS:9444, Domain: wso2.as.domain, Sub-domain:mgt, Active:true

INFO - HazelcastGroupManagementAgent Application member Host:127.0.0.1, Remote Host:null, Port: 4100, HTTP:9764, HTTPS:9444, Domain: wso2.as.domain, Sub-domain:mgt, Active:true joined application cluster

**Configuring the Worker Node #1**

Configuring clustering for the worker nodes implies that the localMemberPort will vary for each worker node. Also we will add the subDomain property, and we will add the ELB and the Manager instance to the well-known members.

1. Open the < AS\_WRKR\_HOME >/repository/conf/axis2/axis2.xml file.
2. Locate the Clustering section and verify or configure the properties as follows (some of these properties are already set correctly by default):
   1. Enable clustering for this node:  
      <clustering class="org.wso2.carbon.core.clustering.hazelcast.HazelcastClusteringAgent" enable="true">
   2. Set the membership scheme to wka to enable the Well Known Address registration method (this node will send cluster initiation messages to WKA members that we will define later):  
      <parameter name="membershipScheme">wka</parameter>
   3. Specify the name of the cluster this node will join:  
      <parameter name="domain">wso2.as.domain</parameter>
   4. Specify the host used to communicate cluster messages.  
      <parameter name="localMemberHost">127.0.0.1</parameter>
   5. Specify the port used to communicate cluster messages (if this node is on the same server as the ELB, manager node, or another worker node, be sure to set this to a unique value, such as 4001 and 4002 for worker nodes 1 and 2). Since it is the first worker node we are configuring, we choose the port 4001  
      <parameter name="localMemberPort">4001</parameter>
   6. Define the sub-domain as worker by adding the following property under the <parameter name="properties"> element:  
      <property name="subDomain" value="worker"/>
   7. Define the ELB and manager nodes as well-known members of the cluster by providing their host name and localMemberPort values. The manager node is defined here because it is required for the Deployment Synchronizer (seen in the following optional lab) to function in an efficient manner. The deployment synchronizer uses this configuration to identify the manager and synchronize deployment artifacts across the nodes of a cluster.  
      <members>  
      <member><hostName>127.0.0.1</hostName><port>4100</port></member>  
      <member><hostName>127.0.0.1</hostName><port>4500</port></member>  
      </members>

**Adjusting the port offset and host name**

1. Open <AS\_WRKR\_HOME>/repository/conf/carbon.xml.
2. Locate the <Ports> configuration and change the value of its sub-tag as follows on each worker node:  
   <Offset>2</Offset>   
     
   We set the offset to 2, because there are already two Carbon products (ELB and AS manager) running on this server.
3. While making changes to the carbon.xml file, specify the host name as follows:  
   <HostName>as.wso2.com</HostName>

**Starting the AS server**

Start the AS server by typing the following command in the terminal:

$ sh <AS\_WRKR\_HOME>/bin/wso2server.sh -DworkerNode=true

The additional -DworkerNode=true argument indicates that this is a worker node.   
  
This parameter basically makes a server read-only. A node with this parameter will not be able to do any changes such as writing or making modifications to the deployment repository etc.   
  
This parameter also enables the worker profile, where the UI bundles will not be activated and only the back end bundles will be activated once the server starts up.

When you configure the axis2.xml (under the clustering section), the cluster sub domain must indicate that this node belongs to the "worker" sub domain in the cluster.

When starting this worker node, it should display logs in the console indicating that the cluster initialization is complete.

Notice also that the ELB console should have these new messages:

INFO - HazelcastGroupManagementAgent Member joined [64e04d13-a41f-4139-81c1-546c6863f231]: /127.0.0.1:4001

INFO - MemberUtils Added member: Host:127.0.0.1, Remote Host:null, Port: 4001, HTTP:9765, HTTPS:9445, Domain: wso2.as.domain, Sub-domain:worker, Active:true

INFO - HazelcastGroupManagementAgent Application member Host:127.0.0.1, Remote Host:null, Port: 4001, HTTP:9765, HTTPS:9445, Domain: wso2.as.domain, Sub-domain:worker, Active:true joined application cluster

When you terminate one node, all nodes identify that the node has left the cluster. The same applies when a new node joins the cluster.

**Configuring the Worker Node #2**

Now we are adding a new worker node to the worker cluster.

1. Copy the AS\_WRKR\_HOME to another directory. This copy will be AS\_WRKR2\_HOME.
2. Open <AS\_WRKR2\_HOME>/repository/conf/carbon.xml
3. Locate the <Ports> configuration and change the value of its sub-tag as follows on each worker node:  
   <Offset>3</Offset>
4. Open the <AS\_WRKR2\_HOME>/repository/conf/axis2/axis2.xml file.
5. In this file, redefine the port used to communicate cluster messages (if this node is on the same server as the ELB, manager node, or another worker node, be sure to set this to a unique value, such as 4001 and 4002 for worker nodes 1 and 2).  
   <parameter name="localMemberPort">4002</parameter>

**Starting the AS server**

Start the AS server by typing the following command in the terminal:

$ sh <AS\_WRKR2\_HOME>/bin/wso2server.sh -DworkerNode=true

When starting the second worker, it should display logs in the console indicating that the cluster initialization is complete.

Notice also that the ELB console should have these new messages:

INFO - HazelcastGroupManagementAgent Member joined [fe60210f-58a9-4357-9ae0-cd7f7a9a25e5]: /127.0.0.1:4002

INFO - MemberUtils Added member: Host:127.0.0.1, Remote Host:null, Port: 4002, HTTP:9766, HTTPS:9446, Domain: wso2.as.domain, Sub-domain:worker, Active:true

INFO - HazelcastGroupManagementAgent Application member Host:127.0.0.1, Remote Host:null, Port: 4002, HTTP:9766, HTTPS:9446, Domain: wso2.as.domain, Sub-domain:worker, Active:true joined application cluster

**Testing the cluster**

To test the cluster in action:

* Log in to the management console of Application Server management node (use the management console URL displayed in the terminal when you started the node)
* Deploy the 2 applications found in the same folder that this document and named:
  + HiRollerBank.war
  + PizzaShop.war
* Refresh the applications list to confirm these war files are correctly deployed.

We must now deploy the same WAR files on the 2 worker nodes. Since we do not have activated Deployment Synchronizer yet (see next lab), we must do this manually:

* Shut down the worker node #1.
* Paste the 2 war files into <AS\_WRKR\_HOME>/repository/deployment/server/webapps
* Paste the 2 war files into <AS\_WRKR2\_HOME>/repository/deployment/server/webapps
* Restart both worker nodes

We can now test the deployed applications:

* Log in to the management console of Application Server management node:  
  https://mgt.as.wso2.com:9643/
* Go to Applications list and click on “Try it”

You will notice that the requests are served by the worker nodes of the cluster.

Shut down the worker node #1. The deployed applications are still available since the worker node #2 can respond to requests.

Optional LabKit

Deployment Synchronizer

**SVN-based Deployment Synchronizer**

Deployment Synchronizer provides capability to synchronize deployment artifacts across the nodes of a product cluster.

All Carbon-based products can use Deployment Synchronizer (DepSync) to ensure the same status is maintained across all nodes in the cluster. It maintains a central repository of the <PRODUCT\_HOME>/repository/deployment/server folder, which is where deployment configurations are stored for all Carbon-based products, and uses that repository to synchronize the nodes.

**Pre-requisites**

To execute on the server that is identified to host the SVN repository:

1. Download and install VisualSVNServer:
   1. win32 executables:   
      <http://www.visualsvn.com/files/VisualSVN-Server-3.0.0-win32.msi>
   2. x64 executables:   
      <http://www.visualsvn.com/files/VisualSVN-Server-3.0.0-x64.msi>
2. After the VisualSVNServer installation, create an empty SVN repository and (at least) one SVN user that will have the rights to write into this repo.

**Giving SVN capabilities to cluster nodes**

1. Download and install SVNKit (svnClientBundle-1.0.0.jar) from http://dist.wso2.org/tools/svnClientBundle-1.0.0.jar to the <PRODUCT\_HOME>/repository/components/dropins folder. <PRODUCT\_HOME> corresponds to the various server products installed in the cluster. Following the previous lab, you will have to drop the downloaded jar into:
   1. <AS\_MGR\_HOME>/repository/components/dropins
   2. <AS\_WRKR\_HOME>/repository/components/dropins
   3. <AS\_WRKR2\_HOME>/repository/components/dropins
2. Download http://maven.wso2.org/nexus/content/groups/wso2-public/com/trilead/trilead-ssh2/1.0.0-build215/trilead-ssh2-1.0.0-build215.jar and copy it to the <PRODUCT\_HOME>/repository/components/lib folder. <PRODUCT\_HOME> corresponds to the various server products installed in the cluster. Following the previous lab, you will have to drop the downloaded jar into:
   1. <AS\_MGR\_HOME>/repository/components/lib
   2. <AS\_WRKR\_HOME>/repository/components/lib
   3. <AS\_WRKR2\_HOME>/repository/components/lib

**Enabling DepSync on the manager node**

You configure DepSync in the <AS\_MGR\_HOME>/repository/conf/carbon.xml file on the manager node by making the following changes in the <DeploymentSynchronizer> tag:

1. Enable the DepSync feature: <Enabled>true</Enabled>
2. Automatically commit local repository changes to the central repository (only enable this on the manager node, which is the node that receives server admin requests as per our deployment pattern):  
   <AutoCommit>true</AutoCommit>
3. Automatically update the local repository with changes from the central repository:  
   <AutoCheckout>true</AutoCheckout>
4. Specify the repository type (in this case, Subversion): <RepositoryType>svn</RepositoryType>
5. Specify the location of the repository with the access protocol:  
   <SvnUrl>http://<PathToRepository>/<RepoName>/</SvnUrl>
6. Specify the user name you defined in the last section: <SvnUser>repouser</SvnUser>
7. Specify the password you defined in the last section: <SvnPassword>repopassword</SvnPassword>
8. Enable tenant-specific configurations if needed:  
   <SvnUrlAppendTenantId>true</SvnUrlAppendTenantId>

The final configuration for the management node should be similar to the following:

<DeploymentSynchronizer>

<Enabled>true</Enabled>

<AutoCommit>true</AutoCommit>

<AutoCheckout>true</AutoCheckout>

<RepositoryType>svn</RepositoryType>

<SvnUrl>http://private.svn.server/depsync.repo/</SvnUrl>

<SvnUser>repouser</SvnUser>

<SvnPassword>repopassword</SvnPassword>

<SvnUrlAppendTenantId>false</SvnUrlAppendTenantId>

</DeploymentSynchronizer>

Now that we have completed configuring DepSync on the manager node, we will configure it on the worker nodes.

**Enabling DepSync on the worker nodes**

You enable DepSync on the worker nodes the same way as on the manager node, with one change: set <AutoCommit>false</AutoCommit>, since worker nodes do not handle server admin requests.

Therefore, the final configuration for worker nodes will be as follows:

<DeploymentSynchronizer>

<Enabled>true</Enabled>

<AutoCommit>false</AutoCommit>

<AutoCheckout>true</AutoCheckout>

<RepositoryType>svn</RepositoryType>

<SvnUrl>http://private.svn.server/depsync.repo/</SvnUrl>

<SvnUser>repouser</SvnUser>

<SvnPassword>repopassword</SvnPassword>

<SvnUrlAppendTenantId>false</SvnUrlAppendTenantId>

</DeploymentSynchronizer>

**Testing the DepSync**

To test the DepSync in action:

* Log in to the management console of Application Server management node (use the management console URL displayed in the terminal when you started the node)
* Deploy the 2 applications found in the same folder that this document and named:
  + HiRollerBank.war
  + PizzaShop.war
* Refresh the applications list to confirm these war files are correctly deployed.
* Changes will get reflected into other nodes after 60 seconds (default synchronization period)