

WSO2 ESB / Apache Synapse Clustering Guide

Copyright

© 2008 WSO2, Inc.

WSO2 Licenses this document to you under the terms of the Apache License, version 2.0. You may not use this document except under the terms of the License. You may obtain a copy of the License at <http://www.apache.org/licenses/LICENSE-2.0>

Authors

Eric Hubert, Jamba! GmbH
Ruwan Linton, WSO2
Asanka Abeysinghe, WSO2
Afkham Azeez, WSO2

Table of Contents

1 Introduction.....	3
1.1 About this Guide.....	3
1.2 Motivation for Clustering.....	3
1.3 Clustering Basics.....	3
2 Clustering in Apache Synapse.....	4
2.1 Configuration.....	4
2.2 Clustering Implementation Details.....	6
3 Clustering Scenarios.....	7
3.1 High Availability.....	7
3.2 Stateful Mediators and Endpoints.....	8
3.2.1 Caching Mediator.....	8
3.2.2 Throttling Mediator.....	9
3.2.3 Statefull Load balancing Endpoint.....	10
4 References.....	10

1 Introduction

1.1 *About this Guide*

This guide briefly outlines the need for clustering in an enterprise application deployment, summarizes some clustering basics, and describes how to configure clustering support available in the Apache Synapse light-weight ESB, as well as the commercially supported WSO2 ESB based on Apache Synapse. It also gives some insights in implementing clustering, targeted at users wishing to deepen their knowledge and help them get started on implementing stateful custom mediators.

1.2 *Motivation for Clustering*

Achieving high availability and scalability are important requirements for many enterprise application deployments. Enterprise Service Buses (ESB) have become the de-facto standard for integrating autonomous entities. Therefore these requirements have also become essential components of an ESB functionality. A common approach to high availability and high scalability is through the formation of computer clusters.

1.3 *Clustering Basics*

A computer cluster is a group of coupled computers that works closely together. In many respects they can be viewed as if they were a single computer. The components, often also referred to as members or nodes of a cluster, are commonly connected to each other through fast local area networks. Software running on each node of a cluster has to facilitate the advantage of having redundant hardware and/or software resources, to increase the availability and/or performance of the whole system. It therefore, has to implement the concept of fault-tolerant components, and share any existing state information among members, so that any redundant component is able to take over the work of another in case it is required. These concepts will be described in detail, in the following few paragraphs.

In order to form a cluster to increase performance of a software system, you typically need to install the software on different computers in your network, and spread work amongst those. This can be achieved using software or hardware load balancers that are able to distribute incoming requests to different nodes of a cluster. This distribution can be based on a very simple strategy/algorithm or on sophisticated ones. The most common and the very basic strategy is round-robin, where, each request is assigned to the next node in a list, until the end of the list is reached and request processing starts from the beginning of this list. Through increasing the numbers of nodes in a cluster, you decrease the average load on each node and are able to handle more work in a given time period. In essence, you are better able to scale your performance requirements.

Using a cluster to increase availability in a software system, adds concepts of fault-tolerance and failover. If and when a component fails, it will be detected by the cluster, which then repeats the failed operation on an identical component residing in the cluster. Only if the nature of the request being processed is completely stateless, that the same operation can be safely repeated on an identical but independent component. For an example, a load balancer can detect the unavailability of a component, which may mean a component that has stopped running or in a faulty state, and delegate requests coming to that component to another. The same does not work if an operation is stateful, and means a component is required to maintain state information between subsequent invocations. To make a failover process work in such a scenario, any stateful component needs to communicate its state information across all identical components within the cluster. This process is also referred to as state replication and needs to be explicitly supported by the underlying cluster framework.

2 Clustering in Apache Synapse

The Apache Synapse ESB is build on top of the powerful Apache Axis2 Web service stack. Most of Synapse components are stateless and do not need any specific clustering support. Therefore, by default, clustering support is not activated on the Synapse ESB.

However, there are a few mediators for e.g. the cache mediator, the throttling mediator as well as the stateful load balancing mediator, which need to maintain state information. If these specific mediators are used in a clustered deployment, Apache Synapse then needs to be configured in order to make use of underlying clustering support available in the Apache Axis2 framework. This support enables Apache Synapse to replicate any state information of mediators, between different cluster nodes.

2.1 Configuration

In order to enable clustering support in Apache Synapse ESB, you need to ensure that multi casting is enabled on your network and the following section in the axis2.xml is uncommented:

```
<cluster class="org.apache.axis2.clustering.tribes.TribesClusterManager">
  <parameter name="AvoidInitiation">>false</parameter>
  <parameter name="domain">wso2esb.domain</parameter>

  <!-- cluster membership management configuration -->
  <parameter name="multicastAddress">228.0.0.4</parameter>
  <parameter name="multicastPort">45564</parameter>
  <parameter name="multicastFrequency">500</parameter>
  <parameter name="multicastMemberDropTime">3000</parameter>
  <parameter name="tcpListenHost">127.0.0.1</parameter>
  <parameter name="tcpListenPort">4000</parameter>

  <contextManager class="org.apache.axis2.clustering.context.DefaultContextManager">
    <listener class="org.apache.axis2.clustering.context.DefaultContextManagerListener"/>
    <replication>
      <defaults>
        <exclude name="local_*/>
        <exclude name="LOCAL_*/>
      </defaults>
      <context class="org.apache.axis2.context.ConfigurationContext">
        <exclude name="UseAsyncOperations"/>
        <exclude name="SequencePropertyBeanMap"/>
        <exclude name="WORK_DIR"/>
        <exclude name="NextMsgBeanMap"/>
        <exclude name="RetransmitterBeanMap"/>
        <exclude name="StorageMapBeanMap"/>
        <exclude name="CreateSequenceBeanMap"/>
        <exclude name="ConfigContextTimeoutInterval"/>
        <exclude name="ContainerManaged"/>
        <exclude name="wso2esb.generated.pages"/>
        <exclude name="my.mercury.*/>
      </context>
      <context class="org.apache.axis2.context.ServiceGroupContext">
        <exclude name="*/>
      </context>
      <context class="org.apache.axis2.context.ServiceContext">
        <exclude name="*/>
      </context>
    </replication>
  </contextManager>
</cluster>
```

This is a default configuration, which needs to be changed only if you need to make adjustments to reflect requirements specific to your network environment, or would like to fine tune the state replication process. Be careful with any change you do though!

The class attribute of the cluster element specifies the main class of the clustering implementation. This class should implement the `org.apache.axis2.clustering.ClusterManager` interface. The Axis2 built-in clustering implementation is based on Tribes. Therefore, Tribes based ClusterManager implementation is specified there by default.

The `AvoidInitiation` parameter specifies whether clustering should be initialized automatically on start up. By default, this is set to false in the Synapse ESB, which will initialize clustering on start up. The Apache Synapse ESB does not handle programmatical initialization and hence users are not supposed to change the value of this parameter.

The domain parameter defines the name of the domain of a cluster. All nodes that use the same domain name belongs to the same cluster, as long as network settings are also identical. This allows us to create multiple clusters in the same network by specifying different domain names.

The parameters `multicastAddress` and `multicastPort` are used for cluster membership management through multicasting. The parameter `multicastFrequency` defines how often multicasts are sent to detect changes in the cluster membership (added or removed members). The parameter `multicastMemberDropTime` defines the time frame after which a member will be dropped from the cluster (loses its membership). The `multicastFrequency` as well as the `multicastMemberDropTime` are specified in milliseconds.

Actual state replication in Synapse is done using TCP. Therefore, parameters `tcpListenHost` and `tcpListenPort`, are used to specify the IP-Address and port used for inter-member-communication.

Apart from these global parameters the configuration contains a `contextManager` section used to configure session data replication. The class attribute of the `ContextManager` element specifies the implementing class of the

`org.apache.axis2.clustering.context.ContextManager` interface. There is an associated listener class implementing the `org.apache.axis2.clustering.context.ContextManagerListener` interface. This class is specified in the class attribute of the listener element.

Data to be exclude in the replication process can be specified in the replication element. Session data is replicated for three context types (namely the `ConfigurationContext`, the `ServiceGroupContext` and the `ServiceContext`). In Apache Synapse, only the `ConfigurationContext` will be used. You can specify which data to be excluded in each of these context types by listing them under the appropriate context element. Each context element can have one or more exclude elements. The name attribute of the exclude element specifies the name of the property to exclude. The default element of the replications section contains the data to be exclude from all context types. It is possible to specify complete property names or the prefix or suffix of property names. Prefixes and suffixes are specified using the asterisk (*) character. For example, according to the above configuration, all session data beginning with the name `my.mercury` will not be replicated for the `ConfigurationContext`. All session data beginning with names `local_` and `LOCAL_` will not be replicated in all contexts. As `ServiceGroupContext` and `ServiceContext` are not used, everything will be excluded from replication. Please change these settings with care and only if you really require to!

Additional Configuration

Interfaces

To set a member's IP (used by other members to identify the member), this can be done by editing the `<parameter name="tcpListenHost">127.0.0.1</parameter>` with the actual IP of the node replacing the value 127.0.0.1. This is an optional setting and the system will pick up the IP address of the first interface listed, and use it as the default.

Database

The WSO2 ESB comes with an embedded Derby database. In order to share a common database among the WSO2 ESB cluster nodes, instead of the embedded Derby databases, modify the `sever.xml` file (`$ESB_HOME/webapp/WEB-INF/classes/conf`) by changing the following entry:

```
<StartEmbeddedDerby>>false</StartEmbeddedDerby>
```

New database settings can be applied by modifying the `wso2esb.hibernate.cfg.xml` and the `registry.xml` (`$ESB_HOME/webapp/WEB-INF/classes/conf`).

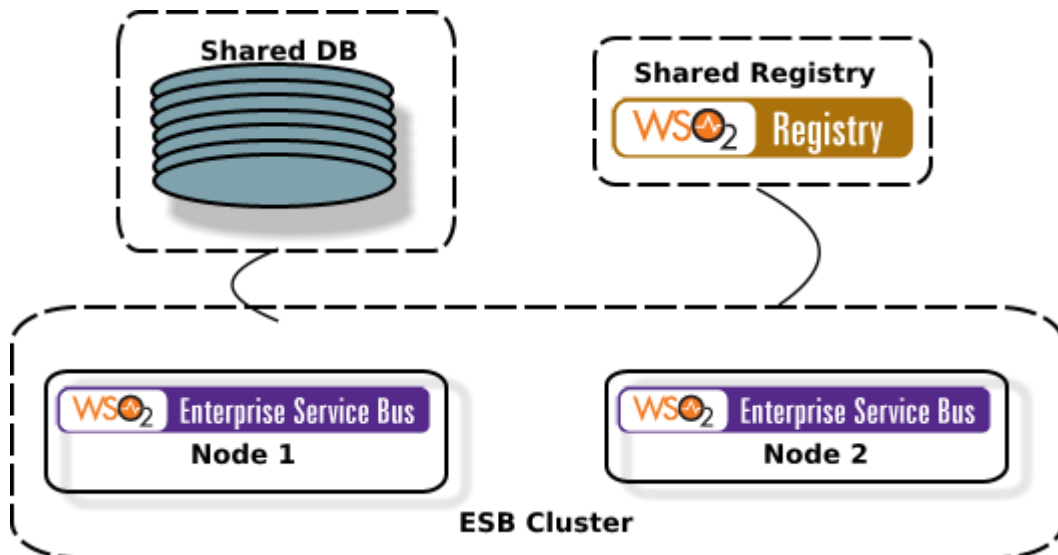
Registry

The WSO2 ESB also comes with an embedded Registry. In order to share a common registry among the cluster nodes modify the `sever.xml` file (`$ESB_HOME/webapp/WEB-INF/classes/conf`) by changing the following block:

```
<Registry>
  <importRoot>file:registry</importRoot>
  <type>remote</type>
  <configuration>
    <url>http://localhost:8081/wso2registry</url>
    <username>admin</username>
    <password>admin</password>
  </configuration>
</Registry>
```

Also change the `synapse.xml` by adding the following entry:

```
<registry provider="org.wso2.esb.registry.WSO2Registry">
</registry>
```



2.2 Clustering Implementation Details

The default clustering implementation is Apache Synapse is based on the Apache Tribes implementation and is a robust Group Communication Framework (GCF). Tribes clustering implementation discovers group members based on membership multi cast. Axis2 also supports plugging in different clustering implementations based on different Group Communication Frameworks (GCFs).

Axis2 clustering stores session data in three levels. Data that should only be available to a service is stored in the ServiceContext. Common data for all services in a service group is stored in the ServiceGroupContext. Common data for all the service groups is stored in the ConfigurationContext. In Apache Synapse, the first two types are not used as all proxy services are stateless in nature. This is the reason why, the default configuration listed in the previous abstract excludes anything of these two context types from the replication.

In a clustered environment, the Apache Synapse ESB keeps all state information in the ConfigurationContext. Apache Synapse does not replicate session data by default because most mediators are stateless. Therefore, in order to ensure that all nodes share the same information, stateful mediators or endpoints need to take care of session replication.

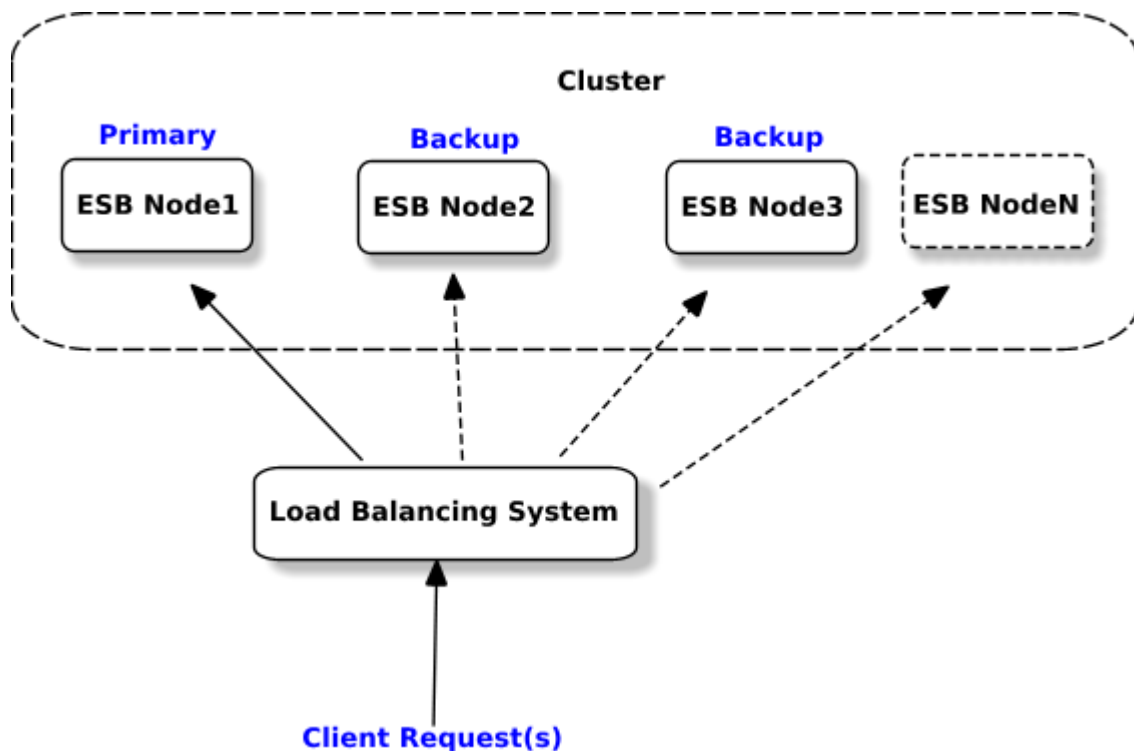
If you are going to write your own mediator or endpoint implementation which needs to maintain state information, you can have a look at one of the built-in stateful mediators, e.g. the Cache Mediator defined in `org.apache.synapse.mediators.builtin.CacheMediator`. All these implementations use static `replicate-Method` of the Apache Axis2 Clustering helper class `org.apache.axis2.clustering.context.Replicator`.

3 Clustering Scenarios

3.1 High Availability

As illustrated in figure 1, session replication in Apache Synapse and in WSO2 ESB, can be used to achieve high availability for stateful entities within a clustered ESB. All client requests are always directed to Node 1, which is elected as the primary server by the load balancing system. Node 2 and Node 3 are kept as back up servers and no requests will be directed under normal operations. With all three nodes configured as a cluster, session state changes in Node 1 are replicated among nodes 2 and 3.

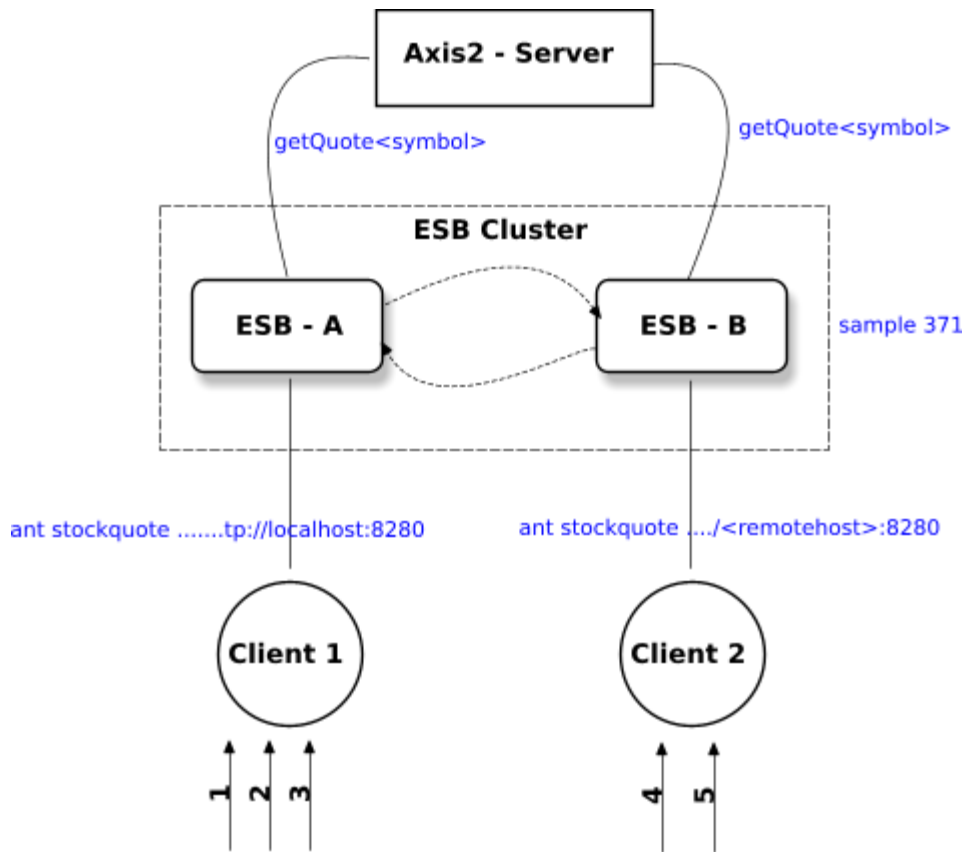
Now if Node 1 fails, the load balancing system elects Node 2 as the primary server with the other two nodes considered as backups. From that point onwards, client requests will be directed to Node 2. The client may not notice any change or data loss, as session data is also made available in Node 2.



Dynamic Routing

Use Linux-HA (Linux High Availability) as the Load Balancing/Fail over system or a DNS based fail-over system and configure nodes with the fail-over system. Fail-over system will route requests to the available node based on availability and priority.

3.2.2 Throttling Mediator



As illustrated above, configure two Synapse instances in a cluster with Axis2 server running 'StockQuoteSampleService'. Refer sample 371 of the Synapse sample guide for detailed instructions.

Start the two Synapse instances with sample 371.

Send three requests from Client 1 to the first server (ESB-A) and client will receive a stock quote for the given symbol. Send two more requests from client 1 (with a total of 3 requests so far) and client will experience the same result.

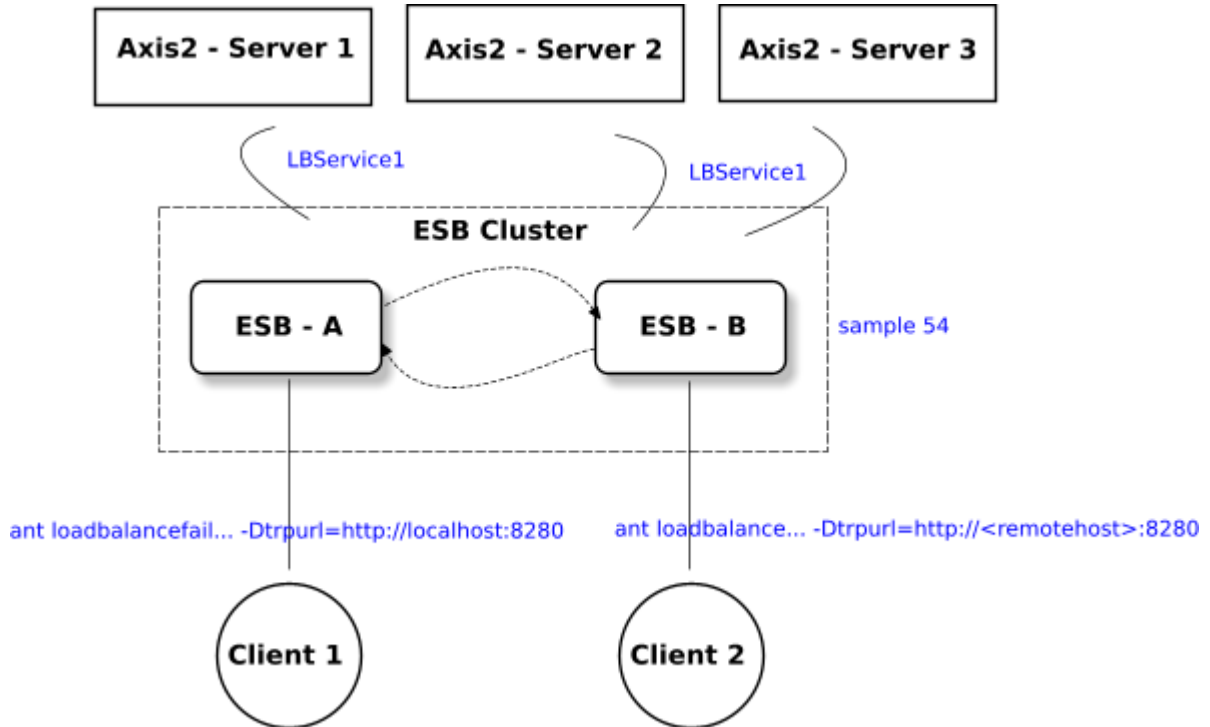
Start client 2 and send a request to server 2 (ESB-B), client will receive a quote for the given symbol. Send a request again from client 2 and client will receive a message with ****Access Denied****.

Send a request from client 1 to server 1 and the client will get the 'Access Denied' message.

Sample has configure to throttle the incoming requests and the maximum count has set to 4, requests coming after that will be rejected with an appropriate message.

In the clustered configuration the application context will be shared so the number of requests coming to the two server instances will be treated in one context and the request count will be the same. Therefore after sending the 4th request from client 2, system will continue reject messages from either client.

3.2.3 Stateful Load Balancing Endpoint



As described in sample 54, run 3 instances of Axis2 servers with the 'LoadbalanceFailoverService' sample service.

Modify the Synapse configuration file for sample 54, synapse_sample_54.xml by giving each endpoint a unique name e.g <endpoint name="EPR1">

Start the two server nodes with sample 54. Send requests to node 1 using client 1, as explained in the sample 54. Identify the session and the server relationship. Send requests from client 2 to node 2 identify the session and the server relationship, compare with the results got with node 1.

The results will be same for node 1 and node 2.

4 References

1. Download WSO2 ESB [<http://wso2.org/downloads/esb/>]
2. ESB Configuration Language [http://wso2.org/project/esb/java/1.7.1/docs/ESB_Configuration_Language.html]
3. ESB Samples Guide [http://wso2.org/project/esb/java/1.7.1/docs/ESB_Samples.html]
4. Documentation Index [http://wso2.org/project/esb/java/1.7.1/docs/docs_index.html]