MuleSoft Blueprint: Load Balancing Mule for Scalability and Availability

Introduction

Integration applications almost always have requirements dictating high availability and scalability. In this Blueprint we'll learn how Mule applications can be implemented to meet such requirements. We'll start off by looking at trivial load balancing approaches, using TCP load balancers and JMS. We'll then dig into Mule's HA Clustering, which facilitates high availability and load sharing for Mule applications. In the course of this discussion we'll also see how Mule's support for VM queueing leads to additional opportunities to introduce reliability into your integration applications.

Developing Applications for Scalability and High Availability

Certain considerations must be taken when developing an application that needs to be deployed for high availability. The primary concern when developing an application is determining what state, if any, must be shared so that it is available for all nodes.

The classic example of shared state in a web application is an HTTP session. Typically a session cookie is used on the client side to associate a user with particular data stored on the server. For a single node this session data can be stored in memory, but in a load balanced environment the data must be stored in a manner that enables each node to access it.

Similar examples exist in integration applications. For example, consider a Mule application that must poll a shared NFS directory using the File transport. This application will function properly as long as its only running a single Mule server. Adding more Mule servers, however, produces a race condition as each server attempts to process the same file simultaneously.

A similar condition may exist with the idempotent-message-filter. The idempotent-message-filter keeps a record of messages accepted by the filter and rejects those that have already been passed. By default Mule stores the state of the idempotent filter on the filesystem local to the Mule server. This state, the list of messages processed by the filter, must be shared with other Mule nodes as they are added to the environment.

BEST PRACTICE: Strive to make your flows stateless whenever possible.

The most effective way to avoid problems of shared state is to write your Mule flows so that they are stateless. Flows that do require shared state should be kept minimal. In addition to avoiding the concurrency issues described in the previous section, this approach will also optimize the performance of your applications since stateless flows tend to perform better than stateful flows.

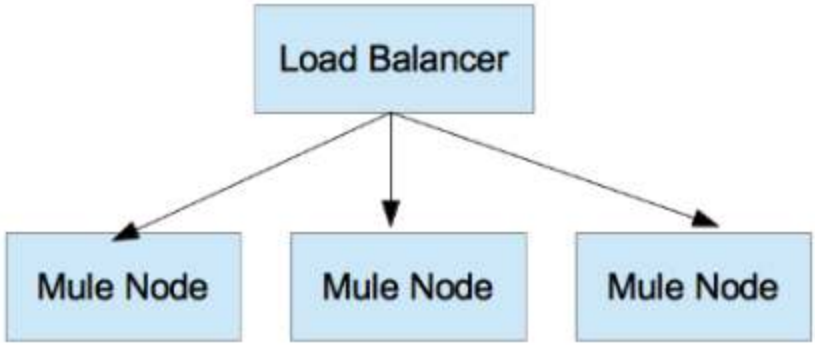
Horizontally Scaling Stateless Mule Applications with Load Balancing

Load balancing provides the easiest mechanism to achieve high availability and scaling with Mule applications. It distributes traffic across multiple Mule nodes running the same Mule application.

Most load balancing mechanisms have a variety of algorithms to choose from. Stateless algorithms, like round-robin, simply alternate traffic between nodes. More complex session-based algorithms use a token, like an HTTP session cookie, to “pin” a client to a particular node or group of nodes.

Load Balancing TCP Based Transports

TCP based transports, such as HTTP, are usually load balanced with either a hardware or software load balancer. Hardware load balancers, such F5’s BIG-IP, or software load balancers, such as HAProxy, are typically used to load balance traffic across a farm of Mule nodes.



Using TCP based load balancing is usually trivial with Mule. The load balancer is typically based "in front" of the Mule nodes.

BEST PRACTICE: Use load balancing without HA clustering when you know your Mule flows are stateless

Load Balancing with JMS

Many JMS brokers, such as ActiveMQ and HornetQ, support competing consumption of JMS queues. This allows multiple clients to consume off the same JMS queue while the JMS broker distributes messages equally to each.

This behavior enables the JMS broker to have a load balancing approach that is similar to the TCP-based one described in the previous section. Additional flexibility can be achieved by using JMS reliability headers in conjunction with competitive consumption. This facilitates the weighting of messages to control which are given priority for processing, a feature not typically possible with TCP based protocols.

NOTE: In order to take advantage of QoS headers ensure that the "honorQosHeaders" attribute is set to "true" in your Mule configuration.

BEST PRACTICE: Check your JMS brokers documentation to confirm the behavior of multiple consumers on a single queue.

BEST PRACTICE: Use JMS reliability headers in conjunction with competing consumption to achieve QoS in a load balanced environment

Using Mule HA Clusters to Distribute State

Horizontal load balancing is a powerful technique for making your Mule applications “web scale”, at the cost of not maintaining state between the nodes. There are, however, times when sharing states is unavoidable. Mule HA Clustering can be used when these situations arise.

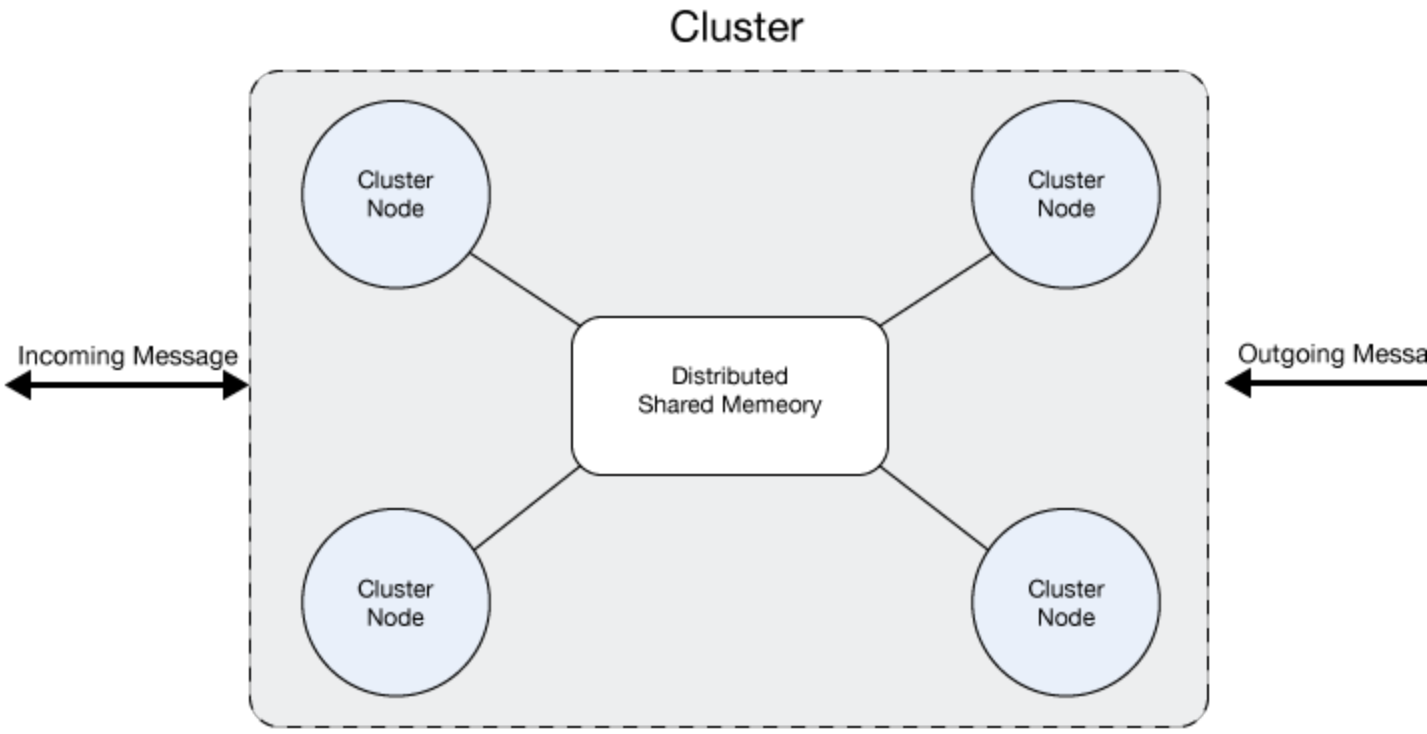
The use of any of the following features in your Mule application likely indicates you have a requirement for shared state:

- The file transport
- The sftp/ftp transport
- JMS topics
- Quartz scheduling
- Message sequencing
- The use of the request-reply exchange pattern on asynchronous transports (ie, JMS or VM.)
- The idempotent-message-filter
- The caching scope
- Any sort of polling of a shared resource
- The use of redelivery policies
-

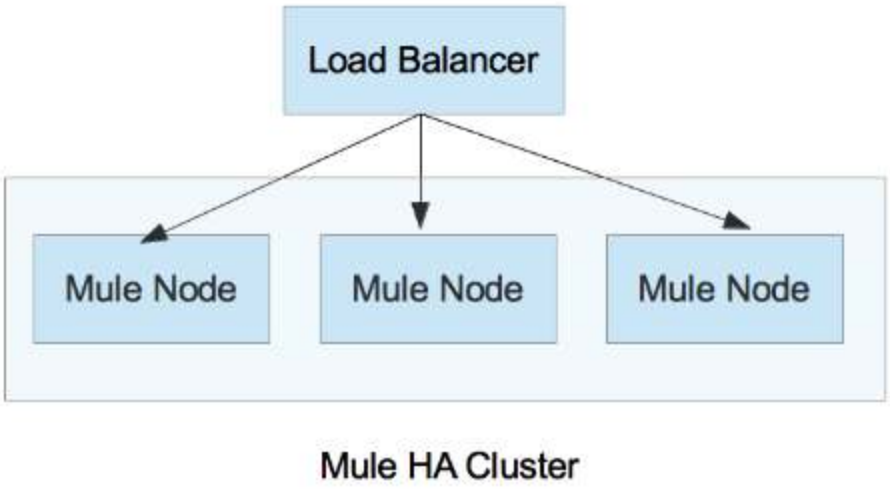
All Mule features, including message processors and aggregation, become “cluster aware” when deployed in a cluster. This is automatic and doesn’t require intervention from the developer or changes during application development. Mule Clusters function just like a single server.

When two or more Mule servers are joined in a cluster, they create a common memory space that serves as the foundation for distributed primitives like queues, collections and semaphores. Through the use of these primitives, the nodes in a cluster can create shared message stores to synchronize state, shared queues to balance load, and semaphores to synchronize access to common resources.

All communication and synchronization happens automatically and in the background, so any Mule application can become a clustered application just by being deployed in a cluster of Mule servers.



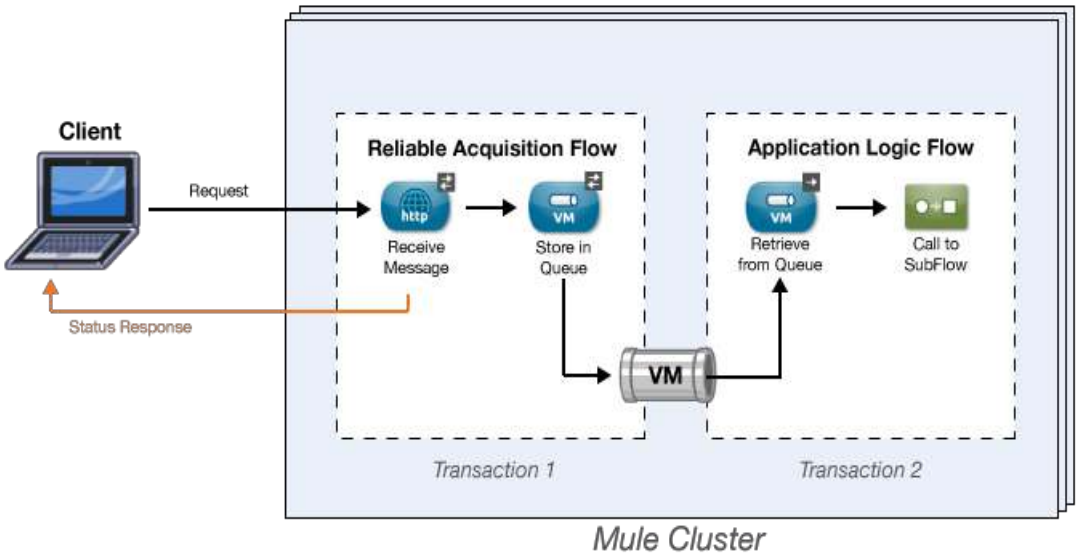
NOTE: Using Mule Cluster in conjunction with TCP based transports like HTTP still requires the use of a front-end load balancer to redirect traffic across the cluster.



Mule HA Cluster

Decomposing Flows for Reliability with the VM Transport

A reliability pattern is a design that results in reliable messaging for an application even if the application receives messages from a non-transactional transport. A reliability pattern couples a reliable acquisition flow with an application logic flow, as shown in the following diagram.



The reliable acquisition flow (that is, the left-hand part of the diagram) delivers a message reliably from an inbound endpoint to an outbound endpoint, even though the inbound endpoint is for a non-transactional transport. The outbound endpoint can be any type of transactional endpoint such as VM or JMS. If the reliable acquisition flow cannot deliver the message, it ensures that the message isn't lost:

- For socket-based transports like HTTP, this means returning an "unsuccessful request" response to the client so that the client can retry the request.
- For resource-based transports like File or FTP, it means not deleting the file, so that it can be reprocessed.

The application logic flow (that is, the right-hand side of the diagram) delivers the message from the inbound endpoint (which uses a transactional transport) to the business logic for the application.

The following transports can be used to decouple flows for reliability in a Mule HA Cluster:

- VM
- JDBC
- Persistent JMS

BEST PRACTICE: Using the VM transport in an HA cluster is usually the easiest way to implement reliable acquisition in a flow.

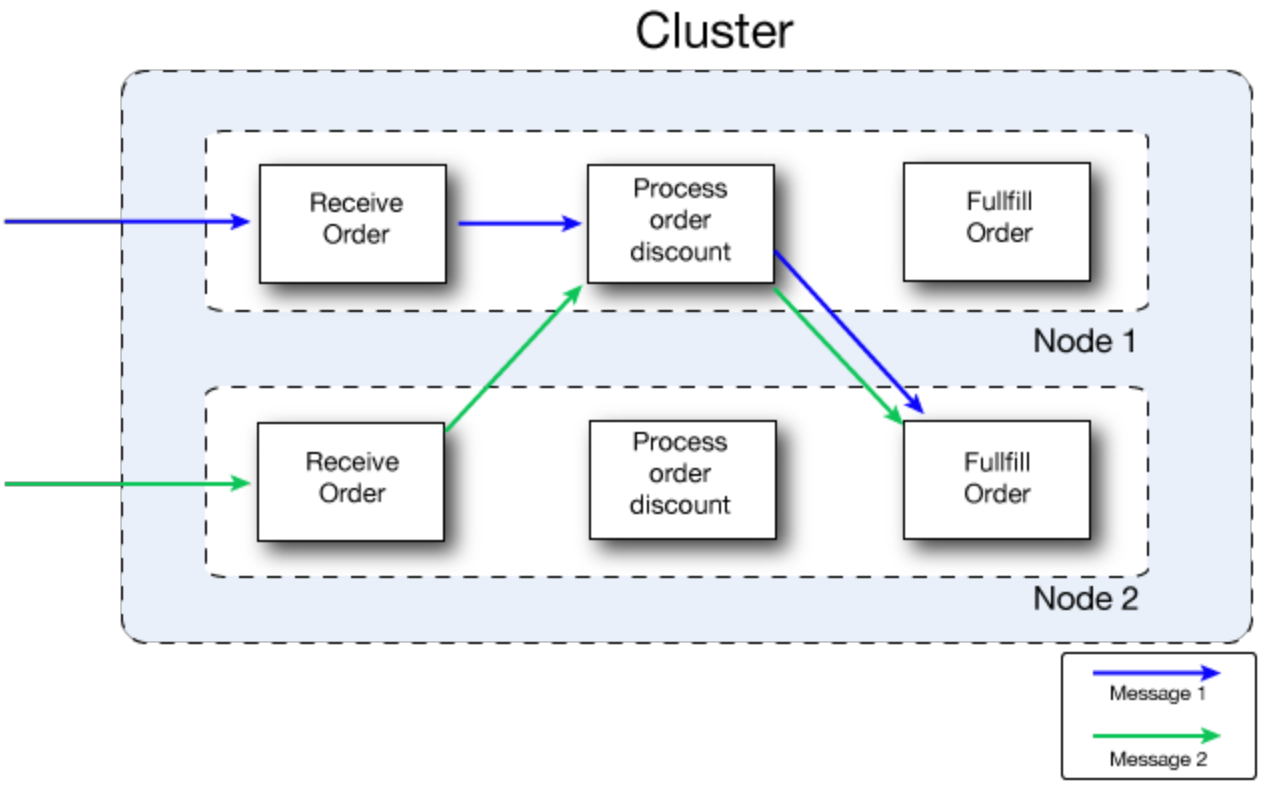
NOTE: Complete information about the implementation of Reliability Patterns can be found here: http://www.mulesoft.org/documentation/display/mmc/Reliability+Patterns

Load Sharing with the VM Transport in a Mule Cluster

Messages passed on a VM queue in a Mule cluster will be automatically load balanced to receiving flows. This makes the VM transport an attractive alternative to JMS queuing when a Mule cluster in available. The following benefits can be realized:

- No additional operational infrastructure is required.
- VM queues are typically order of magnitudes faster than JMS queues
- Clustering of a JMS infrastructure for high availability isn't necessary

The following figure illustrates workload sharing in more detail. Both nodes process messages related to order fulfillment. However, when one node is heavily loaded, it can move the processing for one or more steps in the process to another node. Here, processing of the Process order discount step is moved to Node 1, and processing of the Fulfill order step is moved to Node 2.



Conclusion

In this Blueprint we learned how to develop Mule applications for scalability and availability. First we looked at load balancing, which provides an easy way to achieve horizontal scalability when your flows don't require shared state. When shared state is a requirement, which is frequently the case in many integration scenarios, then horizontal load balancing becomes a challenge because the applications state must be synchronized on an external resource, typically a database or messaging system. Mule HA clustering greatly simplifies this requirement by transparently providing a distributed memory grid to automatically share state between Mule instances. We saw how this feature can be further leveraged to implement reliability patterns and dynamic load sharing using VM queues.

Further information about load balancing Mule can be found on our online documentation or by contacting MuleSoft Services.