Fault Tolerant Clustering - A Concrete Example

Table of Contents

In order to highlight the behavior of the Restcomm JAIN SLEE server when run in fault tolerant clustering mode, a concrete situation is described below.

Example Overview

The example below outlines a situation where two Restcomm JAIN SLEE nodes (`Node-1` and `Node-2`) are configured to run in fault tolerant mode. Both nodes have the same deployable unit containing a custom . A null activity is used by the s to communicate.

The service is a simple application that relays information received in INFO messages to another incoming dialog. This might for example be a rudimentary chat application. The flow outlines the clustered execution of the base case of this application, e.g. receive one message on one dialog and send it out on another dialog.

The example starts with the creation of two entities. One of the entities is the receiver, and the other one is the sender. The receiver will create an and register this with the naming facility. The sender retrieves the and sends a message to the receiver using an event. The example ends when the receiver sends the acquired message on its dialog as an INFO message.

Creating Sbb entities

The base case starts with a INVITE being received in `Node-1`. The event will be routed inside `Node-1` and trigger the creation of a new entity (`Sbb-1`). The entity in turn create and attach to a new null activity (`ACI-1`). Concurrently another INVITE is received in `Node-2`, which causes the JAIN SLEE container to create another entity (`Sbb-2`). Both s and s are fully clustered, so even though they are created in different physical nodes, they are logically inside the same container. In the image below, the cloud represents this logical relationship.

Note that balancer is not a requirement. It is present in this example to show basic message flow with balancer in front of deployed containers.

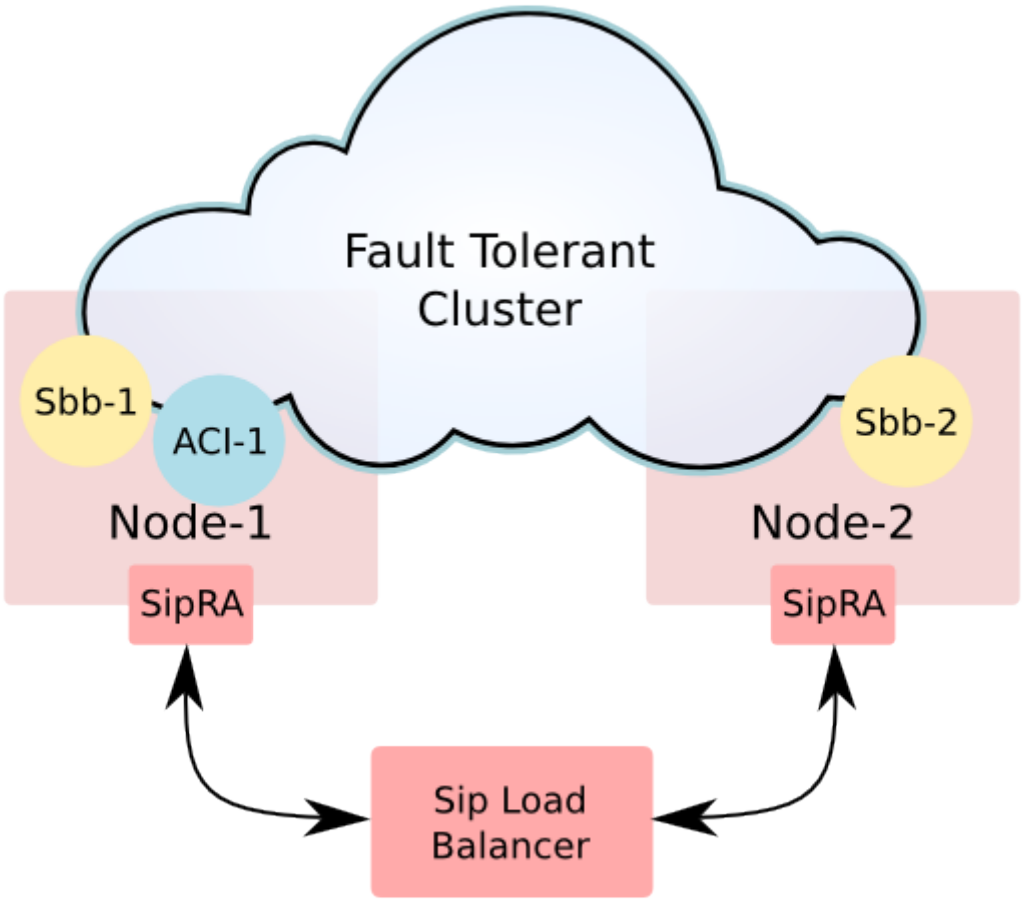

Figure 1. Fault tolerant cluster consisting of two nodes that both have one created on SBB entity for chat application. The SIP load balancer is multiplexing the SIP traffic.

Relaying the Message

After the s have been set up, a INFO is received by `Node-2` and relayed to `Sbb-2`. `Sbb-2` then looks up `ACI-1` from the ACI naming facility. `Node-2` retrieves the clustered state of `ACI-1` and de-serializes it for `Sbb-2`. Note that this means that `ACI-1` is currently being handled in `Node-2`.

When `ACI-1` has been retrieved from the cluster, `Sbb-2` fires a MessageEvent on `ACI-1`. Since events are not clustered, the event will be routed only on `Node-2`. The event will, however, be delivered to all attached s. This is achieved by `Node-2` retrieving the `Sbb-1` entity from the cluster and delivering the event to it.

The MessageEvent is parsed by `Sbb-1` and an outgoing INFO message is constructed with the appropriate payload. `Sbb-1` then forwards the INFO message to the SipRA. The incoming dialog that spawned `Sbb-1` is in `Node-1`, hence the SipRA will retrieve the activity object from the cluster and send the INFO message. The load balancer will then handle de-multiplexing. Note that retrieving the activity object from the clustered state only works because the SipRA is explicitly handling the replication of the SIP activity objects. Had another than the SipRA been used, a similar kind of clustering would have been needed to be implemented using the `FaultTolerantResourceAdaptor` interfaces.

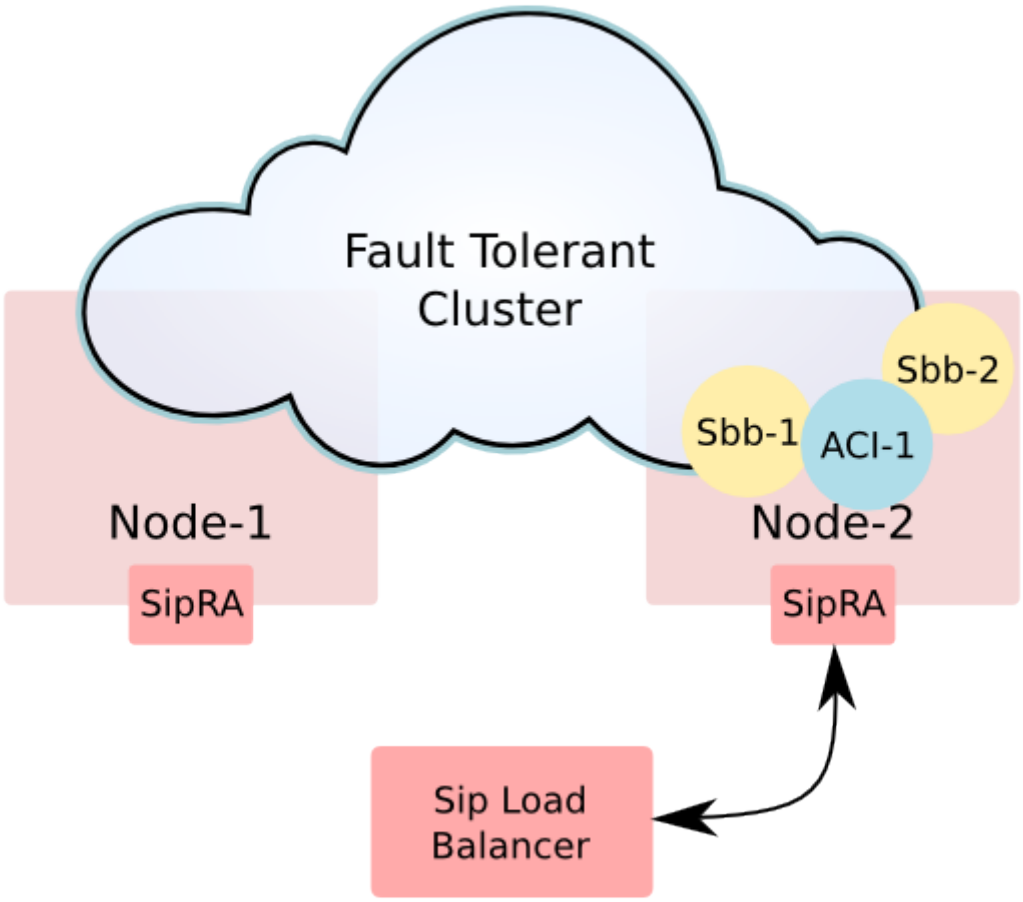

Figure 2. The situation when relaying the INFO message. Both SBB entities are running in the same node.