GlassFish ESB Resilience Options

Discussion Paper
February 2010

Michael.Czapski@sun.com

Abstract

This document discusses considerations and options applicable to arcitecting resilient, highly available solutions using the GlassFish ESB suite and its underlying infrastructure.

Table of Contents

Introduction

It is expected that business solutions, whether designed in accordance with the Service Oriented Architecture principles, or designed following any of the other accepted architectural principles, are robust, reliable and available. Robustness, reliability and availability, in this context, means not just that solutions are free of design and implementation defects but are also architected and deployed in such a way that business users can access them when needed, in spite of any failures that may occur.

In an ideal world all applications will always be available for use. In the real world this may not be possible, or may not be possible at a reasonable cost.

This document discusses resilience options, available to the designers of GlassFish ESB solutions and considerations that need to be entered into when designing GlassFish ESB solutions for resilience and high availability.

High Availability versus Disaster Recovery

Discussion in this document is intended to allow a designer to make better choices when designing solutions with the goals of:

- Increasing Operational Availability
- Reducing Downtime
- Minimising or Eliminating Data Loss

To facilitate discussion and provide the necessary context it is important to understand the kinds of events that might impact availability of GlassFish ESB solutions, the expected severity of these events, the expected duration of service disruption and the kids of measures that can be taken to minimise adverse consequences. It is also important to understand which of these events can be accommodated using standard disaster recovery and high availability measures and which are GlassFish ESB-specific or can be assisted by GlassFish ESB facilities and technology.

So, what do we protect against?

- Natural or Man-made Disasters?
  - Loss of Data Centre
  - Loss of Key Personnel
  - Loss of Power
- Failures of Servers, Networks, Storage?
- Failures of Application Servers / Containers?
- Failures of Applications?

- Failures of Application Components?

The set of questions to which the answer is “Yes” will largely dictate the most appropriate approach to achieving infrastructure resilience.

For example achieving resilience in face of disasters will most likely lead to a disaster recovery approach, whereas achieving resilience in face of application server failures will most likely lead to application server clustering approach or application clustering approach.

It must be borne in mind that the more severe the issue (flood, earthquake, terrorist act, civil disorder) the less sensible it is to depend on a single-site high availability solution as the sole means of achieving resilience. It is likely that not only will the infrastructure in the site become unusable but the personnel needed to operate this infrastructure may be unable to get to the site, and that the customers will be unable to use the service anyway. In short, high availability/disaster recovery decisions must be looked at in context.

Disaster Recovery, the architectural design which addresses catastrophic events, implies the following expectations:

- Longer downtime: Hours, Days
- Semi-manual or manual recovery
- Multiple data centres and networks are affected

High Availability is the architectural design which addresses System and Application failures. It implies the following expectations:

- Shortest downtime: Seconds, Minutes
- Automatic system recovery
- Single data centre and/or network are affected

The expected outcomes, in terms of acceptable duration of downtime, acceptable effort in recovery and acceptable cost, will largely dictate the approach to resilience

The disaster recovery approach leads to longer downtimes and greater recovery effort and implies lesser cost then the high availability approach

Disaster Recovery is invoked when data, equipment or site loss occurs. These events and measures implemented to combat them, are standard measures, applicable to any IT installation or solution implementation. Data should be routinely backed up and media should be kept off site. Arrangements should be in place to repair or replace faulty equipment. If warranted, arrangements should be in place to resume operations at an alternative site. Because these are standard resilience measures this document will not discuss them further. It will be assumed that routine backups are available for data

recovery, if required. It will be assumed that alternate site is available as a passive site or as an active site, if required.

Similarly, because GlassFish ESB solutions can be deployed to a variety of hardware and operating system platforms, operating system clustering will not be discussed in details. Reader is referred to their operating system documentation for information on whether and how operating system clustering can be accomplished.

This document will deal with the kinds of measure an architect can put in place to improve or guarantee availability of GlassFish ESB-based solutions. High Availability is the focus.

Component Dependencies and Failure Points

Components of a computing solution can be logically depicted as nodes in a dependency hierarchy. Failure points are component-related and also can be depicted as existing at different level of the dependency hierarchy. Illustration 1, is a good vehicle to present the dependency hierarchy in the context of a single computing and storage platform.

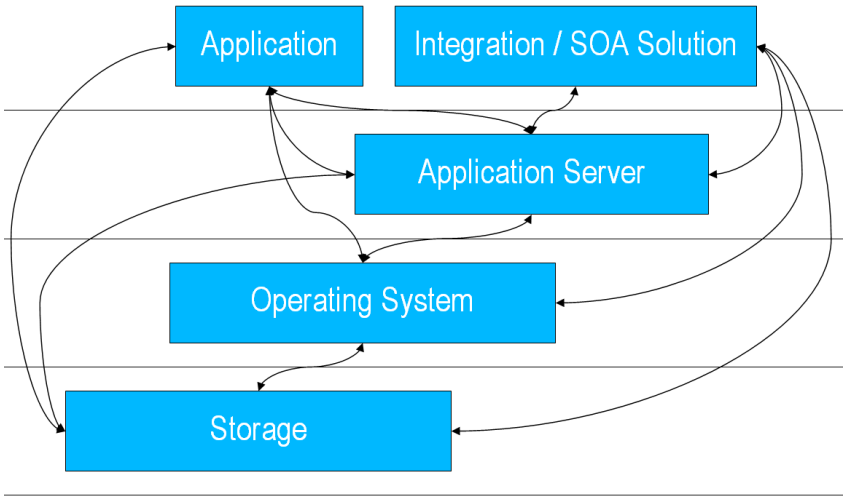

Illustration 1: Dependency Hierarchy

A typical application is dependent on a number of underlying infrastructure components, potentially with their own hierarchical dependencies. Each of these components is subject to failure.

When viewed as a hierarchy of dependent components the solution presents failure points at different levels in the hierarchy.

In some cases addressing a failure point in a component higher in the hierarchy may eliminate the need to address a failure point in a component, upon which it depends, lower in the hierarchy. An example of this is application clustering, which typically eliminates the need to address application server and operating system platform failures but not storage failures.

It is frequently not necessary to configure resilience at each level of the hierarchy. It is essential, however, to analyse the solution, identify failure points and address them at the appropriate level.

The notion of failure points at different levels of the hierarchy is best illustrated with specific examples.

The BPEL Service Engine can be clustered (BPEL SE-specific application cluster) in such a way that multiple, independently deployed BPEL service engines can execute concurrently in multiple, unrelated application servers, on multiple, unrelated physical machines, by sharing a highly available persistence store, as shown in Illustration 2.

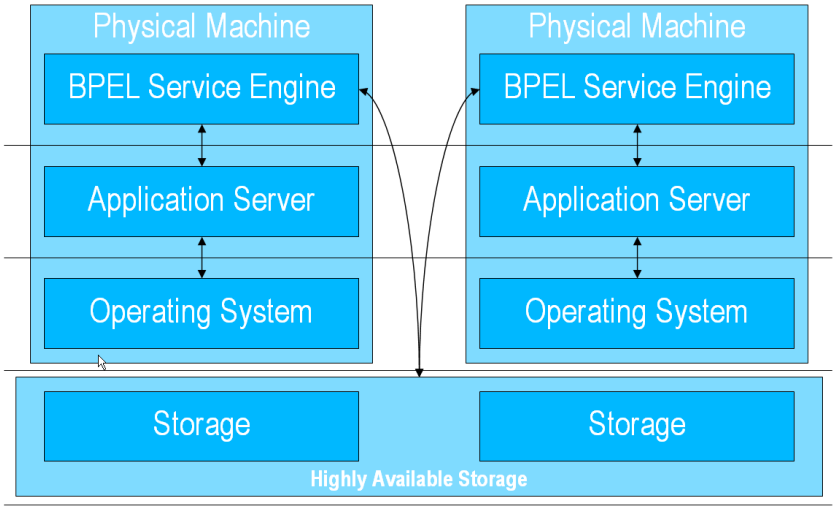

Illustration 2: BPEL SE Application Cluster

The properties of the BPEL SEs, when configured for application clustering, are such that a failure of one (or the underlying application server, or the underlying physical machine), will cause in-flight business process instances to be automatically failed over to the surviving BPEL SEs and continued from the point of interruption.

In this specific case there is no requirement for application server clustering or for operating system clustering. It is necessary only to ensure high availability of share storage.

Multiple Application Servers can be configured as application server clusters to provide high availability for Web Applications which maintain session state between requests (pages). Both load balancing and session failover can be addressed this way. HTTP Load Balancer is used to direct requests to available application servers. In-memory replication capability in the GlassFish Application Server addresses situations where a host fails and the session is failed over to another host. The use of in-memory replication does not require persistent storage. Highly available database (HADB) is required to address session recovery when the entire application server cluster fails. The use of HADB increases availability to 99.999%. Illustration 3 depicts this kind of configuration.

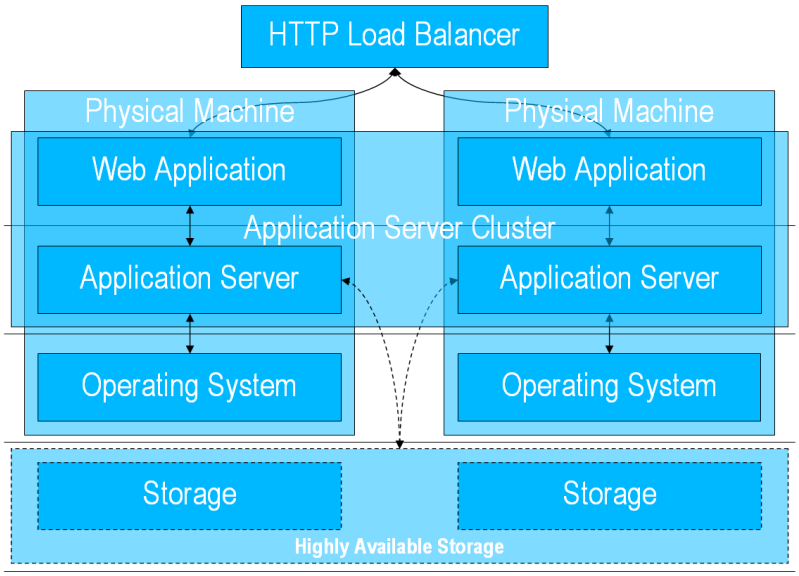

Illustration 3: Stateful Web Application

In this case operating system clustering (lower in the hierarchy) is not required.

Well designed Web Services should be stateless – no “session” should be maintained between service invocations. It is only necessary to ensure that at least one application server, to which the service is deployed, is available, and that the HTTP load balancer is aware of which provider is available. There is no requirement for either application server clustering or operating system clustering. This configuration is shown in Illustration 4.

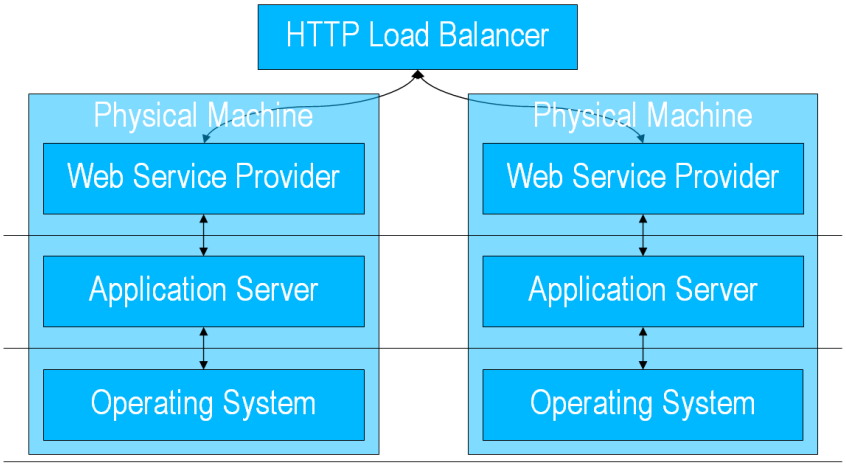

Illustration 4: Stateless Web Service

Staged Event Driven Architecture (SEDA) and Message-oriented Middleware (MOM)

deployments rely on a messaging infrastructure, typically JMS-compliant. Availability of this infrastructure is critical to the availability of the solutions that use it. The Sun Java System Message Queue (SJSMQ) JMS implementation can be configured to provide service high availability, where there is always a JMQ broker with which to communicate, and service and data high availability, where not only there always is a broker with which to communicate, but also the messages queued at the failed broker are available to the surviving brokers.

JMQ does not use an Application Server so Application Server, and discussions relating to it, are irrelevant to JMQ resilience discussions. JMQ Cluster (which is an application cluster) does not require operating system clustering either. It requires highly available shared storage for service and data availability (the highest level of availability for JMQ). This configuration is depicted in Illustration 5.

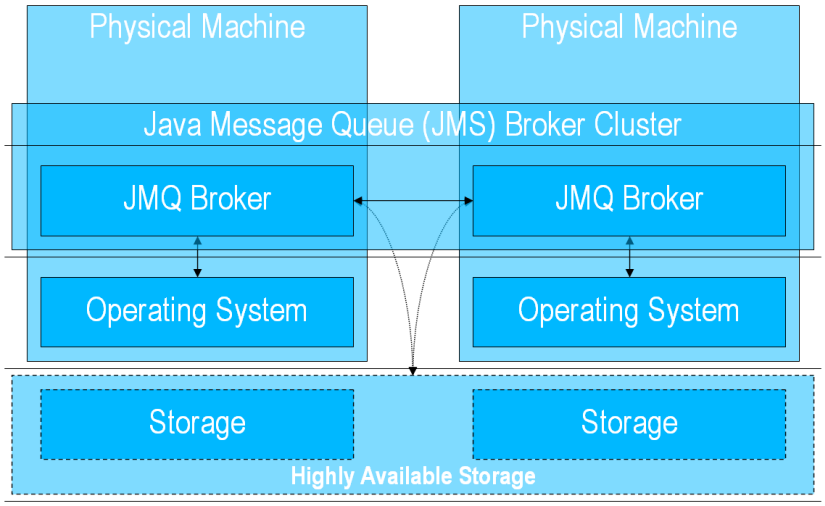

Illustration 5: Java MQ Broker Cluster-based HA

To satisfy the requirement for JMQ service availability without data availability, no highly available shared storage is required.

Some solutions rely on session-oriented TCP/IP connectivity where an external application establishes a connection to an inbound adapter and uses this connection to send messages until it is stopped or until the receiver disappears. If the sending application can re-try connection when the receiver to which it sends messages fails, an operating system cluster using a cluster-wide shared IP address, or a Load Balancer-based solution can be used to provide a failover receiver, which will resume receipt of messages from the sender, or which will carry its share of message traffic. Illustration 6 depicts this kind of solution.

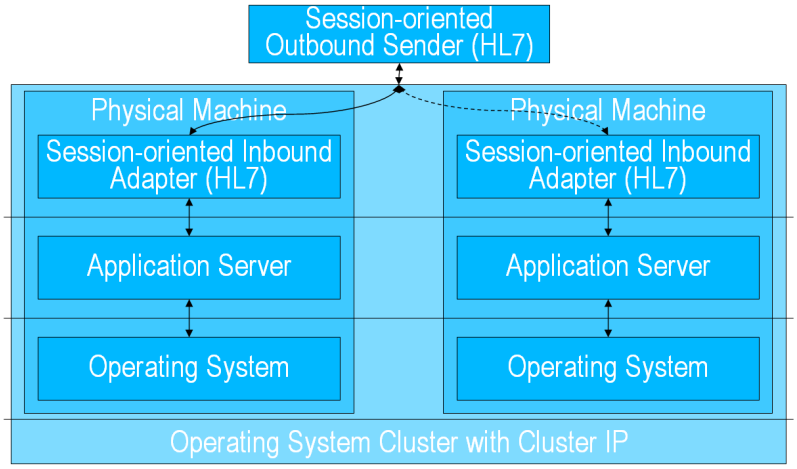

Illustration 6: Cluster-IP or Load Balancer-based HA for TCP-based protocols

More often then not, solutions manipulate application data in persistent storage, whether file system-based or RDBMS-based. To be highly available, these solutions must be provided with continuous access to the application data stores. This is over and above highly available storage required for business process instance persistence, JMQ message persistence, web application session persistence and all other forms of persistence that support high availability of the application infrastructure (BPEL SE, IEP SE), application server or the operating system. Illustration 7 depicts an example of this kind of configuration.

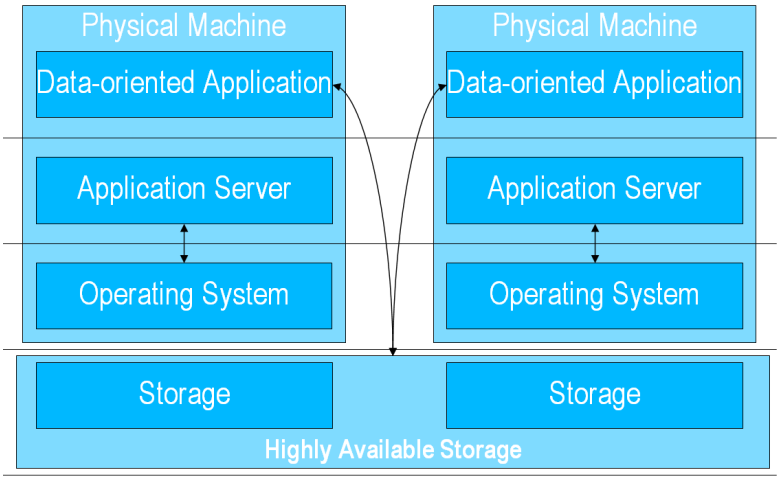

Illustration 7: Application using shared, highly available data storage

All these are examples of solution components which have different characteristics from the high availability stand point. Careful consideration is required to arrive at a resilience solution that addresses all the different requirements.

High Availability

Having looked at examples of different GlassFish ESB solution configurations we identified a number of failure points that may require different solutions. These are:

- Storage failure
- Machine failure
- Application Server failure
- Application failure

We can now look in somewhat greater detail at the properties of each and discuss briefly what measures might be available to mitigate failures.

Storage

Looking at the dependency hierarchy from the bottom up one considers technology options available to address high availability requirements at each level.

A number of options exist to address availability of storage. The costs, complexity and transparency vary. While highly available storage is frequently essential for highly available integration and SOA solutions, specifics of these are well beyond the scope of this document. GlassFish ESB high availability solution architecture would typically assume that the storage it uses is highly available.

Some of the solutions might involve:

- Shared, Network-attached, Highly-available Storage
- Shared Distributed, Replicated RDBMS
- Shared, Distributed Backing/Persistence Stores

It is critical that the shared storage is hosted externally to any GlassFish ESB infrastructure component which may fail, or that shared storage, when co-hosted with GlassFish ESB components like application servers, is configured in such a way that a failure of a host (which would bring both the application server node and the storage node down) does not make shared storage unavailable to the surviving appreciation server nodes.

Machine / Operating System

Operating System-specific Clustering is designed to recover from system failures – failures of hardware, network and storage. OS clustering requires multiple physical machines – at least two. Higher-level components must be Cluster-aware to be effectively managed under operating system clustering. Resource-specific Agents monitor resource availability and notify cluster infrastructure of resource failures. Failures trigger failover to alternate hosts.

Operating system clusters deal with resilience of components which are a) visible at the

operating system level and b) can be manipulated (started/stopped/pinged) from the level of the operating system.

GlassFish Application Server includes Data Services / Cluster Agents that can be used with an OS Cluster.

GlassFish ESB solutions exist inside the application server. They are not visible at the operating system level as distinct components and can not be manipulated in any way unless the application server is operational. Consequently, operating system clustering can generally be used to start/stop/ping application servers executing on cluster members, but not applications or application components hosted by the application server. It is, in a sense, an all-or-nothing proposition for the solutions hosted inside application servers.

GlassFish Application Server

Application server clustering technology was invented to address the requirements of session-oriented web applications, session-oriented enterprise Java beans (SFSB) and entity beans. In effect, only session-oriented web applications, stateful session beans and entity beans can benefit from application server clustering. Other forms of applications deployable to an application server derive no specific benefit from being run in an application server cluster as distinct form a collection of independent application server instances.

GlassFish ESB Binding Components, while not generally benefiting from being deployed to an Application Server Clutters, are cluster-aware and are engineered to ensure “cluster safe” operation. An example of this is the File BC, which uses distributed locking mechanism to ensure that multiple instances of the BC, deployed to different applicatirn server cluster members, polling the same shared directory for the same file, do not both acquire and try to process the same file at the same time. Other binding components, for example the HTTP BC, are exposed independently on different cluster members, or invoke external services, without concern for cluster safety since they are inherently operating on distinct messages/message exchanges.

GlassFish ESB documentation discusses specific cluster configuration of different binding components [1].

Applications

The GlassFish ESB / OpenESB infrastructure uses the GalssFish Application Server as the runtime container for its solutions.

The purpose of GalssFish ESB-based solutions is to provide application integration and to support SOA-based applications. By their very nature, integration solutions are intermediaries between cooperating external applications / message sources and destinations. Different kinds of external systems use different kinds of communication protocols with properties that may assist in development of resilient solutions or may hinder it.

Different means of achieving high availability exist for different kinds of applications.

Sessionless and stateless applications, for example web services or SEDA messaging, can

achieve resilience and high availability by horizontal scaling (multiple instances concurrently executing on multiple, possibly heterogeneous, platforms). If appropriate, and supported by the communication protocol, a load balancer can be used to distribute workload between platforms which are a part of the horizontally scaled solution.

Session-oriented applications, for example web applications, can be deployed to application server clusters with session fail over and load balancer-based distribution with session stickiness

Achieving high availability for applications, which are of a different kind form the above, will vary. One of the most critical aspects of high availability for applications which “service” external systems in the sense of receiving messages from external systems, is the ability of the external system in question to recognize that a specific instance of the service provider/message consumer is not available, and retry. Retry may involve “reconnection” or redirection of message flow to an alternate receiver. For session-oriented TCP-based external senders able to reconnect and resend on failure, a cluster-wide IP address, always pointing to an available instance of a receiver, or a solution involving a TCP load balancer, may be appropriate. For other kinds of consumers/senders this may not be appropriate.

Inbound messaging from external systems requires external systems to be able to recognise receiver failures and redirect messaging to alternate receivers, including resending the failed message, if any. If external systems are unable to do this in an automated manner then failure recovery and service resumption may require human involvement and manual procedures at the external system end.

Ensuring messages are processed in the order in which they were received from the sending system is critical in certain industries and solutions. Designing high availability solutions for these kinds of environments poses special challenges.

In contrast, resilience of the outbound message processing requires that on the one hand the solutions themselves are able to recover from external system failure and on the other hand that in-flight messages are able to be processed to their exit points on any of the surviving members of the high availability solution.

Messages my be received from external senders using the session-oriented TCP/IP protocol, the sessionles HTTP protocol, polling a local or a remote file systems, polling a database, etc.. Each of these means of message delivery requires a different method to address high availability. OS Cluster Shared IP address or a TCP load balancer for one, a HTTP load balancer and multiple independent receivers for the other, highly available local file system or database for the other, etc.. Properties of each method have to be considered before a combined solution is arrived at.

Business logic can be implemented using a variety of execution languages. Some languages, such as Java, are suitable for rapidly processing non-transactional data or for processing short-lived transactions. Other languages, such as BPEL, implement long lived logic where a message may live for seconds, minutes, days, months, or longer, before its processing is completed. The likelihood of a failure affecting any single message in the former case is much smaller then in the later case. The potential number of affected messages in the former case is likely to be much less then in the later case. The ability to simply re-try message processing in the former case is likely to be greater then in the later,

where multiple processing steps with side effects could have been executed by the time the failure occurred.

SEDA solutions, using JMS as a messaging infrastructure for connecting parts of the solution together, pose both different challenges and provide opportunities not available with other methods. For example JMS Queues with concurrent consumers distributed over multiple independent platforms could provide both load balancing and failover.

GlassFish ESB HA versus GlassFish Application Server Cluster HA

To get a better understanding of why GlassFish Application Server Cluster will not provide all the answers, one must look at characteristics of GlassFish ESB components, therefore solutions typically deployed to the GalssFish ESB.

In general, web applications are not constructed using the GlassFish ESB tooling and technologies so GlassFish Application Server Cluster functionality, aimed at providing high availability for web applications, is not applicable to GlassFish ESB-based solutions.

Stateful Session Beans and remote EJBs are typically not part of GlassFish ESB solutions so GlassFish Application Server Cluster functionality aimed at these is not applicable either.

The ability to deploy a load balancer plugin in front of the GlassFish Application Server Cluster may be of benefit to these solutions which use HTTP-based Adapters, including web services and Application Adapters such as PeopleSoft Adapter. Since HTTP protocol-based solutions in GlassFish ESB are stateless, only the load balancer is really required. If it were a stand-alone HTTP load balancer it would not required an application server cluster.

GlassFish ESB-based solutions can not take advantage of the major benefits of application server clustering – session persistence and failover. The ability to use a stand-alone TCP or HTTP load balancers with GlassFish ESB solutions that use TCP-based and HTTP-based protocols minimizes the benefit of using a GlassFish cluster Load Balancer plugin as well.

The GlassFish ESB itself provides application cluster functionality for specific components, namely the BPEL Service Engine and the IEP Service Engine. Both support long running processes and shared persistence store-based failover between instances hosted on different platforms.

The BPEL Service Engine, as has been mentioned before, is clusterable in the sense of being able to be configured as an application cluster. Multiple instances of BPEL SE can concurrently execute on multiple independent platforms, whether themselves clustered or not. All BPEL SE instances would share a highly available persistence store. Specific BPEL processes, which are intended to be long running and which are intended to be resumable, would be configured to persist state at different points during execution to this shared store. On failure of one instance surviving instances would detect the failure and would elect an instance to pick up and resume the interrupted processes. Process state of interrupted processes is available in the persistence store so the process instances can be resumed from the last persistence point. The BPEL SE clustering is configurable on per-BPEL Engine basis, supports correlation failover and does not require Application Server- or Operating System-based clustering. It merely requires two or more independent instances of GlassFish ESB infrastructure and a highly available shared persistence store.

The IEP Service Engine is also clusterable in the sense of being able to form an application cluster with other instances of the IEP SE executing on different platforms. Similar to BPEL SE clustering, the IEP SE persists events in the shared store. Events sent to different instances of the IEP process are processed by a single instance. A failure of a single instance results in the event stream and the persistence store being taken over and resumed by a surviving instance. IEP clustering does not require Application Server- or Operating System-based clustering.

Most Service Engines and Binding Components, which when running inside an application server cluster could attempt concurrent access to the same physical resources, for example files in a file system, are cluster-ware so that potential concurrent access is coordinated cluster-wide and potential conflicts are avoided. For example the File BC ensures that only one of the readers/writes in a cluster reads/writes a file at a time.

Java Message Queue Cluster

While not strictly speaking a part of the GlassFish ESB, the Java MQ JMS infrastructure is used in many GlassFish ESB solutions. When used in such solutions it will enhance architect's ability to design highly available solutions because it both provides distributed message persistence and itself can be configured to be highly available. For this reason it is worthwhile to be aware of different kinds of JMQ clusters and consider their properties when designing highly available GlassFish ESB solutions.

Conventional Broker Cluster provides service availability. Such cluster is intended to ensure that a JMS client can connect to a JMS Broker. When a broker fails clients re-connect to another broker in the cluster. While this ensures service availability messages stored by the failed broker locally, not available. Recovery of messages is delayed until the failed broker resumes operation. This has potential SLA implications, and message sequence preservation implications for delayed messages. This kind of JMS Cluster does not offer data availability.

Highly Available Broker Cluster offers both service and data availability. When a broker fails another takes over the clients and the shared persistent storage used to store messages in flight. This kind of cluster requires HA JDBC Persistence Store.

Concluding Remarks

Ultimately, the solution architecture, the components it uses, and platform configuration, will dictate whether a single high availability solution will be appropriate or multiple solutions will have to be developed to take advantage of these characteristics which assist in achieving the goal of resilience and to work around these which do not.

By its nature, SOA lends itself naturally to HTTP load balancer-based high availability through horizontal scaling. Whether this is achievable depends on the nature of the services being implemented and the resources the services use.

By its nature, SEDA solutions lend themselves to horizontal scaling, load balancing and single point of failure avoidance using the JMS competing consumers pattern. Solutions can be broken up into smaller solutions with JMS persistence points in-between. Each smaller solution may be able to be horizontally scaled over multiple platforms. Using the

JMS Competing Consumer pattern multiple solutions can process messages from the single JMS Queue concurrently. Whether this is appropriate will depend on other factors, most notably whether message sequence preservation is important.

Session-oriented, TCP/IP-based protocols lend themselves to “Cluster IP” and active/passive cluster-based resilience solutions.

Requirement for in-flight process instance persistence or event stream persistence will dictate the need for a a HA database and HA shared storage.

All these factors are additive in the sense that a SOA-based solution may use TCP/IP-based session-oriented adapters, may involve long running processes and may need to access file systems for reading or writing. Some of these matters will not become apparent until a significant part of the solution architecture has been developed, and may require redesign for resilience and high availability.

Summary

This document discussed resilience options, available to designers of GlassFish ESB solutions, and considerations that need to be entered into when designing GlassFish ESB solutions for resilience and high availability.

Disaster recovery and high availability were discussed and contrasted to set the context for subsequent discussion of high availability options.

A component dependency hierarchy was presented and examples of solutions, in which high availability is addressed at the different level in the hierarchy, were presented and discussed.

High availability of storage, operating system, application servers and applications was discussed in greater detail, with characteristics, resilience options and applicability of resilience options to GlassFish ESB solutions presented and discussed.

GlassFish Application Server cluster characteristics and GlassFish ESB components and application clustering characteristics were compared and contrasted to address circumstances in which one is more appropriate then the other.

JMS high availability was discussed since JMS is used in many GlassFish ESB solutions.

It should be obvious that careful consideration of the numerous options is required to determine the best means of architecting highly available GlassFish ESB solutions.

Combination of application, application server and operating system clustering may be required to achieve appropriate degree of resilience.

There is no single prescription for HA. Relevant aspects of each solution have to be analysed and decisions have to be made at multiple points to arrive at a strategy that will be most appropriate for the circumstances.

Appendix

BPEL SE Clustering Example

The material presented in “Clustering GlassFish ESB: BPEL Failover and Load Balancing (V1.3)” [2] illustrates both GlassFish Application Server clustering, the use of the GlassFish Load Balancer plugin, and BPEL SE clustering and failover.

The single, end-to-end tutorial leads the reader through the steps of establishing a GlassFish Application Server Cluster, setting up BPEL Service Engines to use shared persistence store, and proving that BPEL SE failover works in a BPEL SE cluster.

A simple solution, designed to experience BPEL failover in a GlassFish Application Server cluster, is developed and exercised.

A cluster of two application server instances, each hosting a BPEL service engine (SE) and two binding components (BC), HTTP and FILE, is created. BPEL processes running in each BPEL SE are fed test messages.

In the initial scenario messages are sent directly to the HTTP BC on each instance. Load balancer is not used.

Once the messages have arrived and business process instances have been created, one of the application server instances is killed. The BPEL SE on the surviving instance picks up the in-flight work and completes all business process instances.

Subsequently, load balancing is added to the scenario and load balancer-mediated instance creation on and failover is demonstrated.

Topics include:

- Building and configuring a GlassFish cluster of two application server instances
- Creating and deploying a simple BPEL process that we can use for BPEL failover testing
- Establishing the soapUI test cases we need to demonstrate BPEL failover
- Seeing BPEL failover in action
- Installing, configuring and testing the Sun Web Server load balancing plug-in to front the GlassFish Application Server cluster

HL7 Messaging High Availability Examples

The material presented in “GlassFish ESB v2.2 Field Notes - Exercising Load Balanced, Highly Available, Horizontally Scalable HL7 v2 Processing Solutions” [3] leads the reader through the process of constructing a heterogeneous, non-clustered collection of hosts, and using it to implement and exercise three load balanced, highly available GlassFish ESB-based solutions. The environment consists of a number of independent “machines”, which are not a part of an Operating System Cluster. Each “machine” hosts a GlassFish Application Server. Application Servers are independent of one another and are not clustered. The intent of this material is to demonstrate that load balanced, highly available, horizontally scalable solutions, based on the GlassFish ESB software alone, can be

designed and implemented.

The specific class of solutions to which this discussion applies is the class of solutions which are exposed as request/reply services, for example HL7 messaging with explicit Application Acknowledgment, or Request/Reply Web Services or JMS in Request/Reply mode, which implement business logic as short lived processes, and which are atomic, or idempotent, or tolerant of duplicate messages.

In the document only high availability and scalability of receiver solutions is addressed. This aspect is the focus because a failure to process a message by a receiver may result in message loss.

The document discusses an exercise involving an example load balanced, highly available, horizontally scalable healthcare environment, processing HL7 v2 messages. Discussion includes customization of generic GlassFish ESB v2.2 VMware Virtual Appliances for a specific Load Balancing and High Availability exercise and deploying ready-made GlassFish ESB solutions. The exercise for HL7 BC-based, Web Service-based and JMS-based highly available, load balanced, and horizontally scalable receivers, processing HL7 v2.3.1 messages, is conducted and discussed.

The objective is to convince the reader that for the applicable class of GlassFish ESB-based solutions, load balancing and dynamic failover without message loss works. For that class of solutions this provides for high availability and horizontal scalability without resorting to Application Server or Operating System clustering.

References

1. “Configuring GlassFish ESB for Clustering”, Available: http://wiki.open-esb.java.net/Wiki.jsp?page=Jbi_clusteringAbout_c, Accessed: January 23rd, 2010
2. “Clustering GlassFish ESB: BPEL Failover and Load Balancing (V1.3)”, Available: http://wikis.sun.com/download/attachments/47881337/GFESBBPELHALBTutorialV1_3.pdf, Accessed: January 23rd, 2010
3. “GlassFish ESB v2.2 Field Notes - Exercising Load Balanced, Highly Available, Horizontally Scalable HL7 v2 Processing Solutions”, Available: http://blogs.sun.com/javacapsfieldtech/entry/glassfish_esb_v2_2_field1, Accessed: January 23rd, 2010

Acknowledgements

Contributors: Michael Czapski (michael.czapski@sun.com)