One Pager: SAF/AMF support for SailFin release.

Table of Contents

1. Introduction

1.1. Project/Component Working Name

SAF/AMF support in SailFin

1.2. Name(s) and e-mail address of Document Author(s)/Supplier

Nandini Ektare : nandini.ektare@sun.com

Hong Zhang : hong.zhang@sun.com

1.3. Date of This Document

09/17/2007

2. Project Summary

2.1. Project Description

Enable support for AMF (Availability Management Framework) in an environment where availability of SailFin instances is managed by SAF (Service Availability Framework) based middleware.

2.2. Risks and Assumptions

2.2. Risks and Assumptions

Definition and Configuration of SailFin cluster topology / application (un)deployments is still done by the SailFin Domain Application Server (aka DAS). SailFin cluster has to reach a "steady-state" before the AMF within SAF takes over the control. This one-pager addresses management of SailFin cluster runtime only through AMF.

The translation of the domain configuration information (presently available in DAS) to AMF understandable form is done using separate SailFin tools. Availability and stability of these translation tools, though orthogonal, is core to proper functioning of the presented solution.

3. Problem Summary

3.1. Problem Area

Production deployments of the SailFin need the ability to leverage the SAF implementations which are gaining popularity in the telecommunications industry as they allow system designers to build highly available applications that are interoperable and portable across a variety of compliant middleware and platforms.

This one-pager proposes a mechanism to plug SailFin into a SAF compliant implementation.

More information on SA Forum efforts, their two interface specifications: AIS (Application Interface Specification) and HPI (Hardware Platform Interface) and role of AMF within the AIS is available here

Within SAF, AMF specifies a software entity that provides service availability by coordinating redundant resources within a cluster to deliver a system with no single point of failure. This framework provides a set of APIs to enable highly available applications. It drives the high availability state of various system components, and monitors their health by invoking callback functions of these components. It also manages the readiness state without exposing it to components and further allows a component to query the framework for information about a given component's high availability state, using functions defined in the set AMF APIs.

3.2. Justification

4. Technical Description

4.1. Details

4.1.1. Start/Stop of Instances

As part of providing service availability through AMF, AMF controls start/stop of the instances which is traditionally done by the Node Agent.

For this to work seemlessly,

A. The assumption that SailFin cluster reaches a "steady state" is vital. Steady state implies that the cluster has been defined and configured through SailFin DAS

> Every member instance of the cluster is aware of this final cluster topology. Traditionally, every member instance gets this awareness through the Instance Synchronization process which Node Agent performs on behalf of it's managed instances. So, in other words, clustered instances need to be synced-up before AMF starts controlling their lifecycle.
>
> Note: E*very time* a cluster undergoes configuration changes including changes post-creation, steady state has to be achieved before AMF takes over the cluster runtime management.

B. AMF starts a SailFin clustered instance using the startserv (startserv.bat) script for that instance[1]. AMF stops a SailFin clustered instance using the stopserv (stopserv.bat) script for that instance. These scripts are present in the bin directory of the SailFin clustered instance. For example <nodeagent-install-dir>/<instance-name>/bin/startserv.

Note: AMF has to ensure that the clustered instance (and it's associated IMQ broker process) has stopped completely before invoking startserv script.

[1] Every Instance obtains an environment variable "SA_AMF_COMPONENT_NAME" during startup. This variable has to be passed while invoking the startserv script.

C. Following are recommended steps in case node agent managing a clustered instance has to be started/stopped.

Start the node agent with –startinstances=false and –syncinstances=false

Use OS tools to kill node agent explicitly. Otherwise stop-node-agent will try to stop existing running instances (Alternative if an RFE is implemented: stop-node-agent –stopinstances=false. However it's not planned for implementation)

Note: By design, if a node-agent has not started an instance, it will not watchdog it. This takes care of the issue that if AMF were to control instance lifecycle, Node agent's watch dogging capability can interfere.

D. SailFin clustered instance needs to load an AMF-enabling (native) code during startup. This code is referred to as JAA-AMF (Java Availability API - Application Management Framework) code. JAA-AMF Provider code has to be loaded during Instance startup

A system-all LifeCycle Module within instances will load the JAA-AMF provider code. This can be handled by creating the domain with a profile like "amf". *(see GlassFish Profiles Specification)*. This profile will create a DAS with a pre-configured system-all LifeCycle Module and include the provider code binaries. These two artefacts will become available on clustered instances as part of reaching the "steady state" for AMF management.

The Provider code will be in the form of a native library. The provider code requires the environment variable "SA_AMF_COMPONENT_NAME" which is passed during startup.

4.1.2. Start/Stop of Application

In addition to managing the starting and stopping of the instances, the AMF also manages the starting and stopping of the applications deployed in the instances.

AMF will manage the starting and stopping of the applications through JSR77 MBean start/stop interface.

1. Deployment: The (root) JSR77 MBean should be available after the deployment, even if the application was deployed

   in a disabled state.

2. Server (instance) start up: The (root) JSR77 MBeans should be available for all deployed applications after the server is started, even if the applications were previously deployed in disabled state.

3. The implementation of the JSR77 MBean start/stop should enable an authenticated client to start/stop (load/unload) an application:

   For standalone modules, load/unload the modules. Additionally:

   for standalone resource adapter, it will also load all the associated connector configurations and resources.

   for standalone SIP module, equivalent MBean will be created and used to load/unload sip module.

   For application, the root JSR77 MBean will be invoked to load/unload all the modules of the application including any SIP modules if they exist.

4. Recovery of xa resources will also be handled by SAF. SAF will use an application server api to disable xa resource recovery during application startup by application server. Later SAF will initiate the xa resource recovery.

Note: Definition/configuration of instances forming the cluster and deployment/undeployment of the applications to this cluster are still the responsibility of DAS and Node Agent.

4.2. Bug/RFE Number(s)

None

4.3. In Scope

Start/Stop of SailFin Instances by AMF

Loading AMF provider plug-in during SailFin instance startup

Start/Stop of Applications deployed on SailFin Instances

4.4. Out of Scope

Definition/configuration of clusters and instances in the SailFin and their translation to AMF.

Synchronization of instances.`

Deployment/Undeployment of the applications to this cluster.

4.5. Interfaces

4.5.1 Exported Interfaces

4.5.2 Imported interfaces

com.sun.appserv.connectors.spi.ResourceRecoveryManager
[available in appserv-ext.jar]

```
    /**
     * to enable lazy recovery, setting lazy to "true" will disable recovery during
     * application startup by application server
     * @param lazy boolean
     */
    public static void setLazyRecovery(boolean lazy)

    /**
     * to recover xa resources
     * @param force boolean to override "lazyRecovery"
     */
    public static void recoverXAResources(boolean force)
```

Usage :

```
// [application startup's recoverXAResources will be a no-op since lazy-recovery is set to 'true' ]
ResourceRecoveryManager.setLazyRecovery(true);
...
...
ResourceRecoveryManager.recoverXAResources(force=true);  // will do actual recovery
```

4.5.3 Other interfaces (Optional)

Interface	Stability	Exporting Project: Name, Specification or other Link.	Comments
			Location of the scripts:

startserv (startserv.bat) stopserv (stopserv.bat)			<nodeagent-install-dir>/<instance-name>/bin/startserv <nodeagent-install-dir>/<instance-name>/bin/stopserv For example: <installdir>/publish/glassfish/nodeagents/i1/bin/startserv

4.6. Doc Impact

None

4.7. Admin/Config Impact

None

4.8. HA Impact

The proposal provides a way to replace existing HA capabilities within SailFin with the industry standards proposed by SAF for achieving high-availability.

4.9. I18N/L10N Impact

None.

4.10. Packaging & Delivery

None

//Installation scripts would provide the JAA-AMF Provider native library. Details in the SailFin FS.

4.11. Security Impact

None.

4.12. Compatibility Impact

None.

4.13. Dependencies

http://wiki.glassfish.java.net/Wiki.jsp?page=FunctionalSpecsOnePagers

5. Reference Documents

SAF and GlassFish : An Architectural document

SA Forum Home

6. Schedule

6.1. Projected Availability

November 2007