*White Paper*

# High Availability Services with SAS® Grid Manager

August 11, 2014

## Revision History

| Version | Date | Author | Description of Changes |
|---|---|---|---|
| 1.0 | Aug. 29, 2011 | Robert Jackson | Initial revision. |
| 1.1 | Aug. 11, 2014 | Scott Parrish | Remove references to ESD_CORP_DOMAIN from EGO Service Director example plugin configuration file under “Without Corporate DNS Updates” section. |

Table of Contents

# Introduction

In a SAS environment, there are certain services that need to always be available and accessible. These services are vital to the running applications and their ability to process SAS jobs. Example services include:

- SAS Metadata Server
- SAS Object Spawner
- Platform Process Manager
- Platform Grid Management Service
- Web Application Tier Components

High Availability for these services can be achieved using the EGO capabilities provided by the Platform Suite for SAS which is a third party component included with the SAS Grid Manager product. Although Platform Suite for SAS can be licensed without SAS Grid Manager the EGO High Availability capabilities require a SAS Grid Manager license.

In order for SAS Grid Manager to provide High Availability for SAS services, the following aspects must be addressed:

- **Service Resource Redundancy** – Provide alternate resources for execution of essential services. Allowing critical functions to execute on multiple physical or virtual nodes eliminates a single point of failure for the grid.
- **Service Resiliency** – Provide a mechanism to monitor the availability of services on the grid and to automatically restart a failed service on the same resource or on an alternate when necessary.
- **Service Relocation** – Provide a method for allowing access to services running on the grid without requiring clients to have any knowledge of the physical location of the service.

## Scope

This document addresses the requirements for implementing High Availability (HA) services running in a SAS grid environment using the EGO capabilities of Platform Suite for SAS which is included with SAS Grid Manager. Configuration examples are included that provide details for configuring essential SAS services to be Highly Available in the grid.

## Terms

- **High Availability (HA)** – A system design protocol and associated implementation that ensures a certain absolute degree of operational continuity during a given measurement period. Availability refers to the ability of the user community to access the system, whether to submit new work, update or alter existing work, or collect the results of previous work. If a user cannot access the system, it is said to be unavailable.
- **Failover** – The capability to move over automatically to a redundant or standby resource upon the failure or abnormal termination of a previously active application or service.
- **Switchover/Migrate** – The capability to move over to a redundant or standby resource upon manual user request.
- **Fault Tolerance** – The capability to continue proper operation after a failure has occurred.
- **Platform Enterprise Grid Orchestrator (EGO)** – Component of Platform Suite for SAS that enables High Availability. It is automatically enabled when Platform Suite for SAS is installed and configured.
- **IP Load Balancer** – A hardware device that is used to distribute IP connections across multiple hosts.
- **Virtual Hostname** – A name or alias that is substituted for the actual hostname to be accessed.
- **Domain Name System (DNS)** – System for translating human readable hostnames into numerical identifiers associated with networking equipment for the purpose of locating and addressing these devices.
- **Fully Qualified Domain Name (FQDN)** - The unambiguous name for a host which specifies the system hostname along with all domain levels up to and including the top level domain.

# Service Resource Redundancy

The first consideration for making any service Highly Available within the grid is to ensure that redundant resources are available for execution of the service. Resources within the grid might become unavailable in the event of a fault. Some failures that can impact grid resources include:

- Power outages
- Failed CPU, RAM, Disks, Network Interface Cards, etc…
- Over-temperature related shutdown
- Security breaches
- Application, middleware, and operating system failures

Grid resources could also possibly be taken offline manually by an administrator for maintenance, but no matter the reason for a resource being offline, Service Resource Redundancy allows alternate resources within the grid to be used avoiding service outages which would disrupt the grid's ability to execute jobs. For the grid to be truly HA, any critical services must be allowed to execute on two or more nodes within the grid, and note that only the LSF server nodes are eligible to be execution resources for HA services. For SAS services there are two possible redundancy schemes to consider:

**Active-Passive Redundancy –** The service will execute on only one node at any given time but alternate (standby) resources are available for executing the service.
**Active-Active Redundancy –** The service will execute on multiple nodes concurrently with each service instance providing full capabilities to consumers.

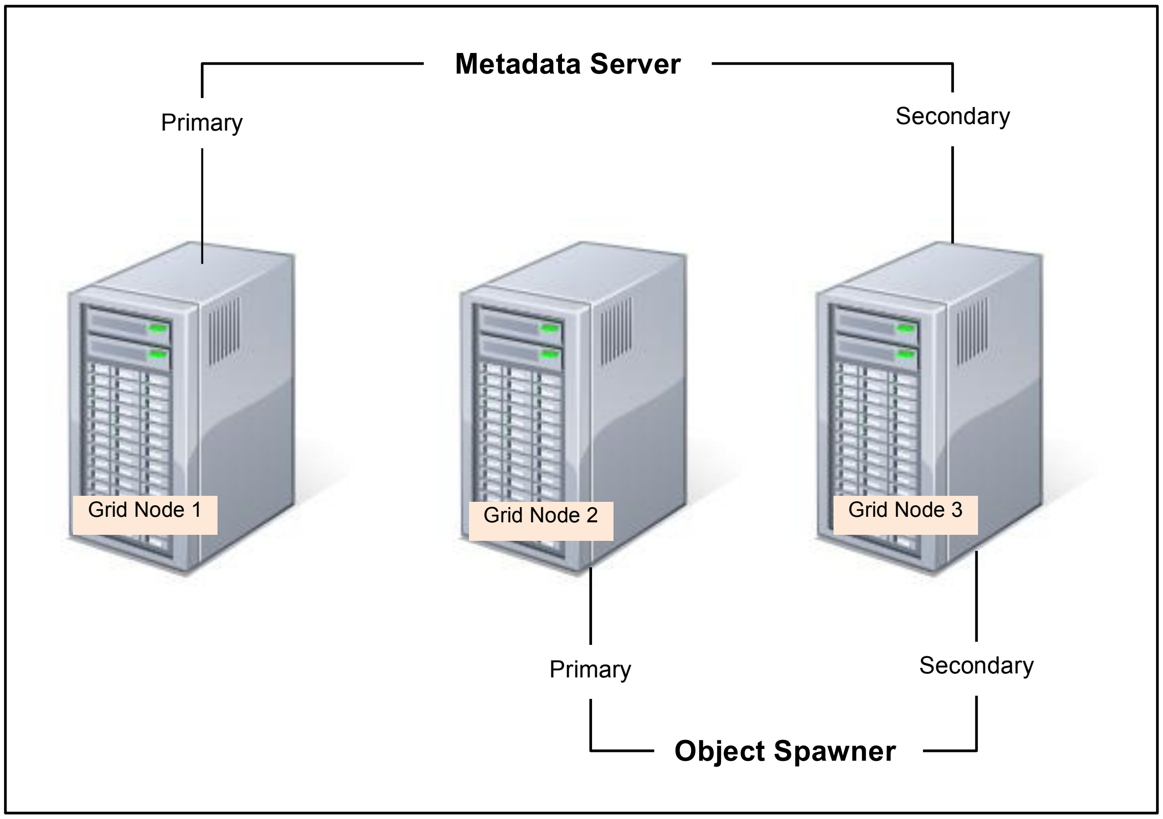
## Active-Passive Redundancy

When using an active-passive (cold standby) redundancy scheme multiple nodes are capable of executing the service but only one instance of the service is executing at any given time. The normal execution node, referred to as the primary; the first alternate node, is the secondary; optional second alternate, would be the tertiary. By default the service will be started on the primary resource when it is available and the alternates remain in cold standby; meaning the service is not concurrently executing on multiple resources and it will only be started on an alternate when the primary becomes unavailable. If a service is started on an alternate resource and the primary is recovered and becomes available, the service will not be automatically moved back to the primary (non-revertive failover); the recovered primary becomes the first alternate (secondary) and remains in cold standby.

NOTE: The example configurations presented in this document will only include primary and secondary resources, but if tertiary resources are required refer to Appendix A for configuration details.

A active-passive redundancy scheme is pictured below in Figure 1 which includes two critical services configured across three grid nodes.

***Figure 1: Example Active-Passive Resource Redundancy Scheme***



The Metadata Server is configured to primarily run on Grid Node 1, but can failover to Grid Node 3. The Object Spawner has Grid Node 2 as its primary resource, but can also failover to Grid Node 3. This is a simple example but illustrates how under normal conditions the grid resources would be fully used, but providing Resource Redundancy can increase the availability of vital services within the grid if resources become unavailable allowing the grid to continue executing jobs without disruption.

When deciding which grid resources will be used as alternates for critical services it is important to consider the potential utilization of resources within the grid when a service is started on an alternate node. For example, utilizing the scheme shown above, if Grid Node 2 becomes unavailable Grid Node 3 will be executing the Object Spawner. If instead Grid Node 1 were chosen as standby for the Object Spawner, Grid Node 1 would then be providing both services. Executing both services might increase the load on Grid Node 1 to at or above 100%, which quite possibly could degrade the grid's ability to execute jobs. Again, a simple example, but it is important to develop an optimal redundancy scheme for the critical grid services such that the grid can continue to function after a fault has occurred.

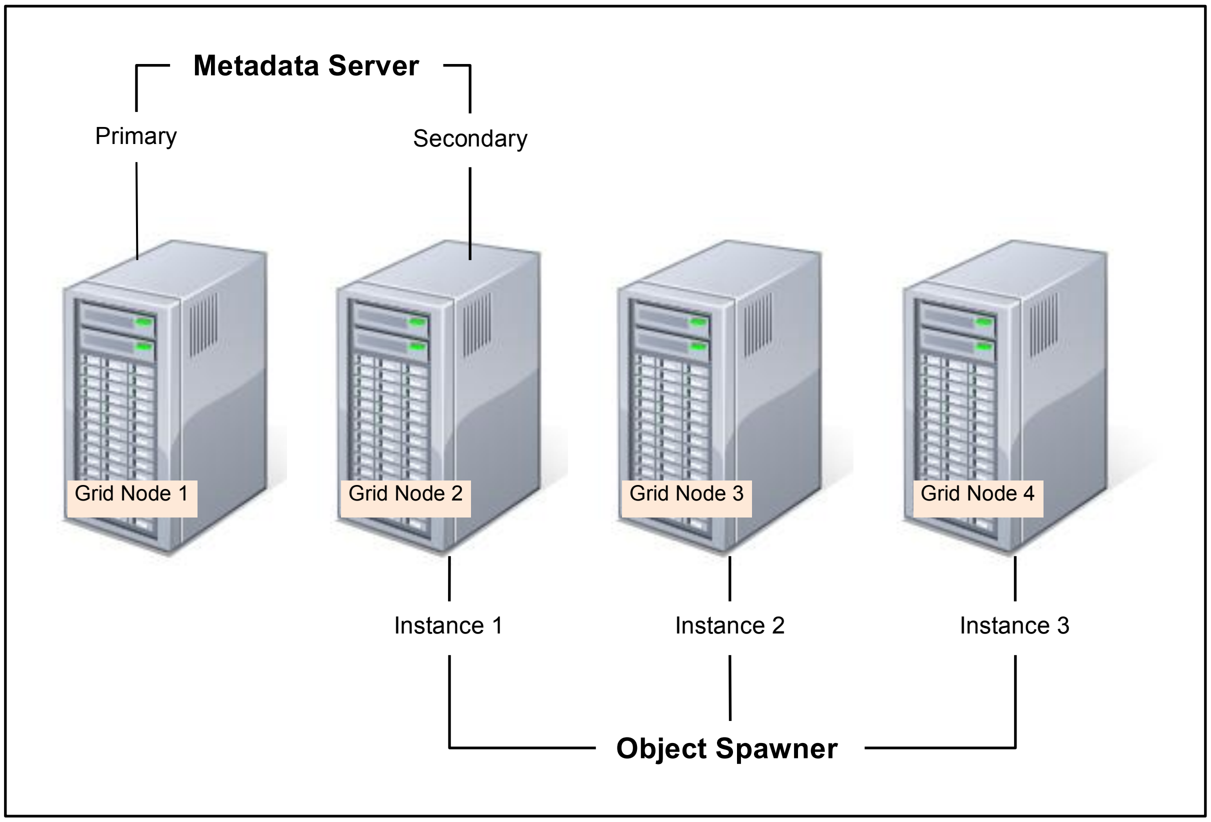
For the grid to provide redundancy for any given service, the application providing the service must be able to execute on each node configured to provide the service. This requires that all necessary software be installed or somehow accessible and appropriately licensed, plus all required application configuration and data must be available and properly synchronized. Using the grid's shared storage for the SAS configuration simplifies much of the redundancy setup, and the examples presented in this document assume this approach is used.

## Active-Active Redundancy

In some deployments for certain SAS services an active-passive redundancy scheme isn't the preferred configuration. For example, a SAS Object Spawner which implements load balancing provides Resource Redundancy for the service in an active-active configuration; meaning the service executes on multiple resources concurrently. This type of redundancy scheme can still use some aspects of the HA functionality provided by the SAS Grid Manager to improve service availability. It is possible that some services still use active-passive redundancy while others are configured for active-active redundancy. An example of such a mixed redundancy scheme is shown below in Figure 2.

NOTE: Although most of the discussion and examples in this document will be based on a active-passive redundancy scheme, any known differences in configuration for an active-active scheme will be noted where applicable.

***Figure 2: Example Mixed Resource Redundancy Scheme***



As with the previous example, the Metadata Server is configured to primarily run on Grid Node 1, but in this layout it would failover to Grid Node 2. The Object Spawner will have three concurrent instances running on Grid Nodes 2, 3, and 4. Again this is a simple example but it illustrates how active-passive and active-active redundancy can be combined to increase service availability while also achieving the desired resource utilization.

# Service Resiliency

To increase service availability there needs to be some facility to monitor execution of the services within the grid and to automatically restart failed services or move them to an alternate resource when necessary. The resiliency, or fault tolerance, of services within the grid is handled by the EGO Service Controller. Each service that will be monitored within the grid must be defined in the EGO Service Controller configuration.

NOTE: It is not required for a service to have Resource Redundancy to take advantage of the resiliency provided by the EGO Service Controller. Even when a service will only execute on a single node within the grid the EGO Service Controller can be used to provide execution resiliency for the service. To add such a service to the EGO Service Controller configuration set both the **`Primary Host`** and **`Failover Host`** to be the same node.

NOTE: All machines used to run services only, those that do not contribute computing power to the grid, for which SAS Grid Manager will provide High Availability are counted in the total number of cores for the SAS Grid Manager license. A machine dedicated to running the SAS Metadata Server for which SAS Grid Manager will provide High Availability is a good example.

## EGO Service Controller

The EGO Service Controller is a Platform EGO component that is always executed on the EGO master host (Grid Control Server) and is responsible for starting and monitoring configured services. It ensures that all the defined services are running by detecting failures and restarting service instances. The EGO Service Controller reads configuration files for the services that must be instantiated. These configuration files are XML documents which contain the service definition that provides details such as how to start and stop the service, parameters describing types of resources the service may require, and so on. There is one service definition file dedicated to each service to be managed by the EGO Service Controller. These files can be created and edited using Platform RTM for SAS, but in deployment scenarios where this utility cannot be used the service definitions can also be created manually. The examples in this section were generated using Platform RTM for SAS version 2.06; for instructions on manually creating EGO Service Controller definitions refer to Appendix A.

## Managing HA Services with Platform RTM for SAS

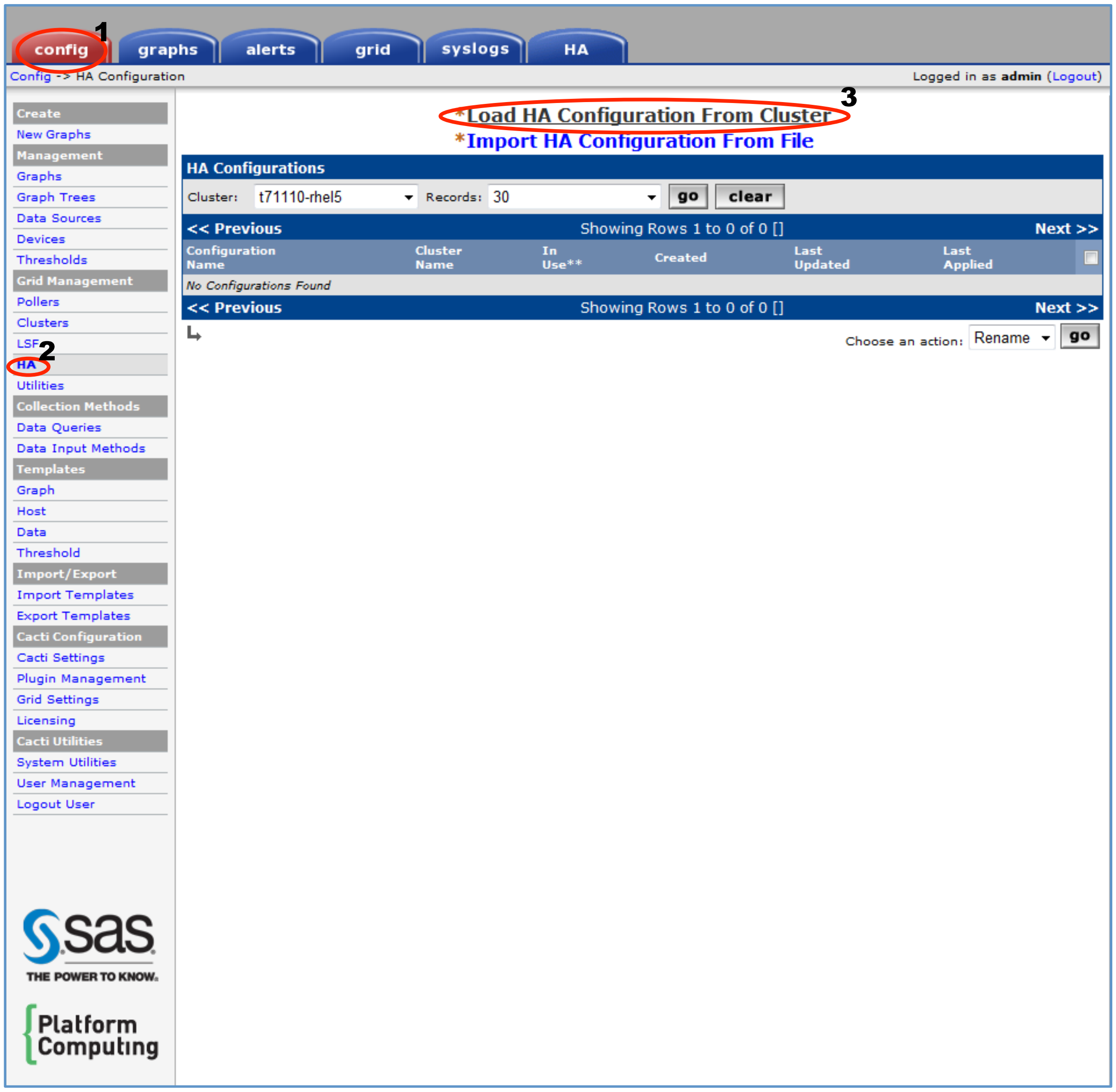
Platform RTM for SAS is a web-based operational dashboard for the SAS grid environment. It provides capabilities for configuring, monitoring, and controlling many aspects of the grid environment including HA services. Defining an HA Application inside RTM will create the necessary definition for the service so that the EGO Service Controller will begin monitoring and managing the execution of the service within the grid. The exact steps for creating HA Applications within RTM are detailed in the following sections.

## Loading HA Configuration

Before defining HA Applications inside RTM, an HA configuration must be created for the cluster that is to be managed. The action within RTM to be performed is ***Load HA Configuration From Cluster** which will read in any existing EGO Service Controller definitions and create the initial configuration inside RTM. Even if no EGO Service Controller definitions exist this operation is still required, but the new configuration will be empty with *No Configured Applications*.

Referring to the figure below, perform the following steps to load the HA configuration:

1) Select **config** from the tab list at the top of the screen.
2) On the left-hand menu, under the **Grid Management** section select **HA**.
3) In the operation pane, select ***Load HA Configuration From Cluster** and then answer **yes** when prompted for confirmation.



HA Configurations

Cluster: t71110-rhel5 Records: 30

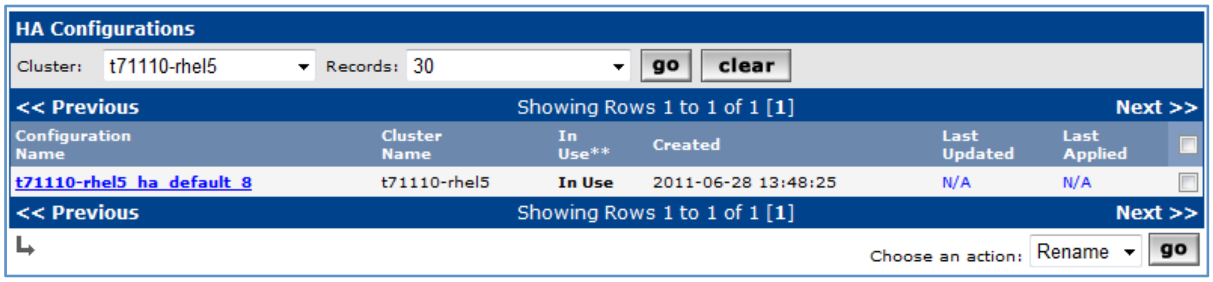
Showing Rows 1 to 0 of 0 []

| Configuration Name | Cluster Name | In Use** | Created | Last Updated | Last Applied |
|---|---|---|---|---|---|
| *No Configurations Found* | | | | | |

Showing Rows 1 to 0 of 0 []

Choose an action: Rename

Once the configuration has been loaded, the operation pane will refresh and a table showing the newly created configuration will appear.


HA Configurations

Cluster: t71110-rhel5 Records: 30 go clear

<< Previous Showing Rows 1 to 1 of 1 [1] Next >>

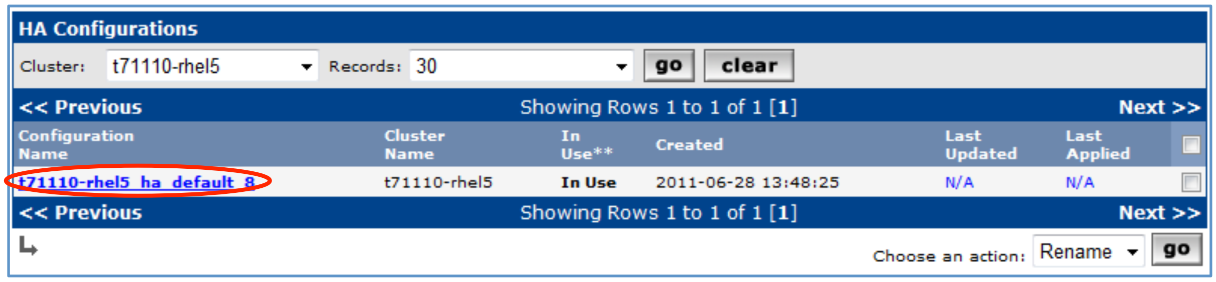
| Configuration Name | Cluster Name | In Use** | Created | Last Updated | Last Applied | ☐ |
|---|---|---|---|---|---|---|
| t71110-rhel5_ha_default_8 | t71110-rhel5 | In Use | 2011-06-28 13:48:25 | N/A | N/A | ☐ |

<< Previous Showing Rows 1 to 1 of 1 [1] Next >>

Choose an action: Rename go

## Defining HA Applications

Once an HA configuration has been created, to edit the configuration and define HA Applications click on the **Configuration Name** link.


HA Configurations

Cluster: t71110-rhel5 Records: 30 go clear

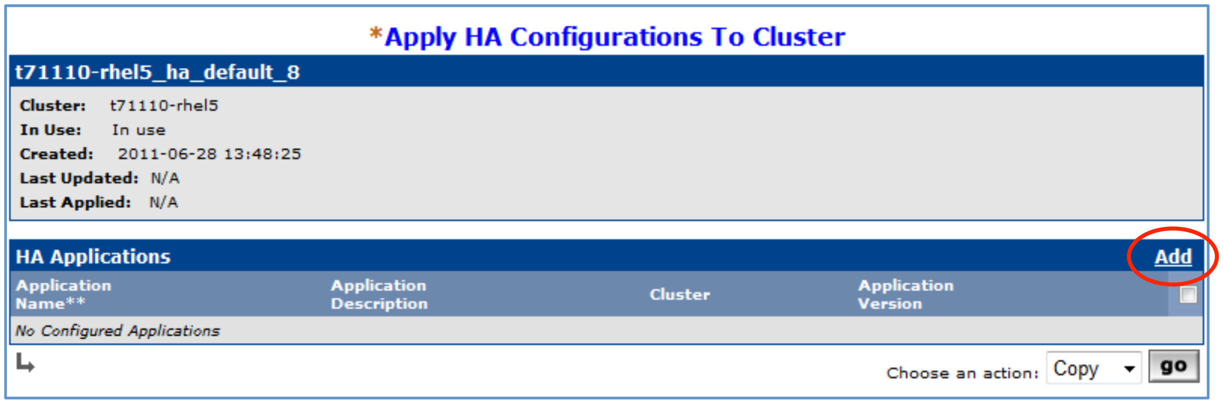
<< Previous Showing Rows 1 to 1 of 1 [1] Next >>

| Configuration Name | Cluster Name | In Use** | Created | Last Updated | Last Applied | ☐ |
|---|---|---|---|---|---|---|
| t71110-rhel5_ha_default_8 | t71110-rhel5 | In Use | 2011-06-28 13:48:25 | N/A | N/A | ☐ |

<< Previous Showing Rows 1 to 1 of 1 [1] Next >>

Choose an action: Rename go

The RTM HA configuration is initially created with *No Configured Applications* as shown in the HA Applications table below; to create a service definition required for the EGO Service Controller, select the **Add** link on the far right of the header row in the HA Applications table.


*Apply HA Configurations To Cluster

t71110-rhel5_ha_default_8

**Cluster:** t71110-rhel5
**In Use:** In use
**Created:** 2011-06-28 13:48:25
**Last Updated:** N/A
**Last Applied:** N/A

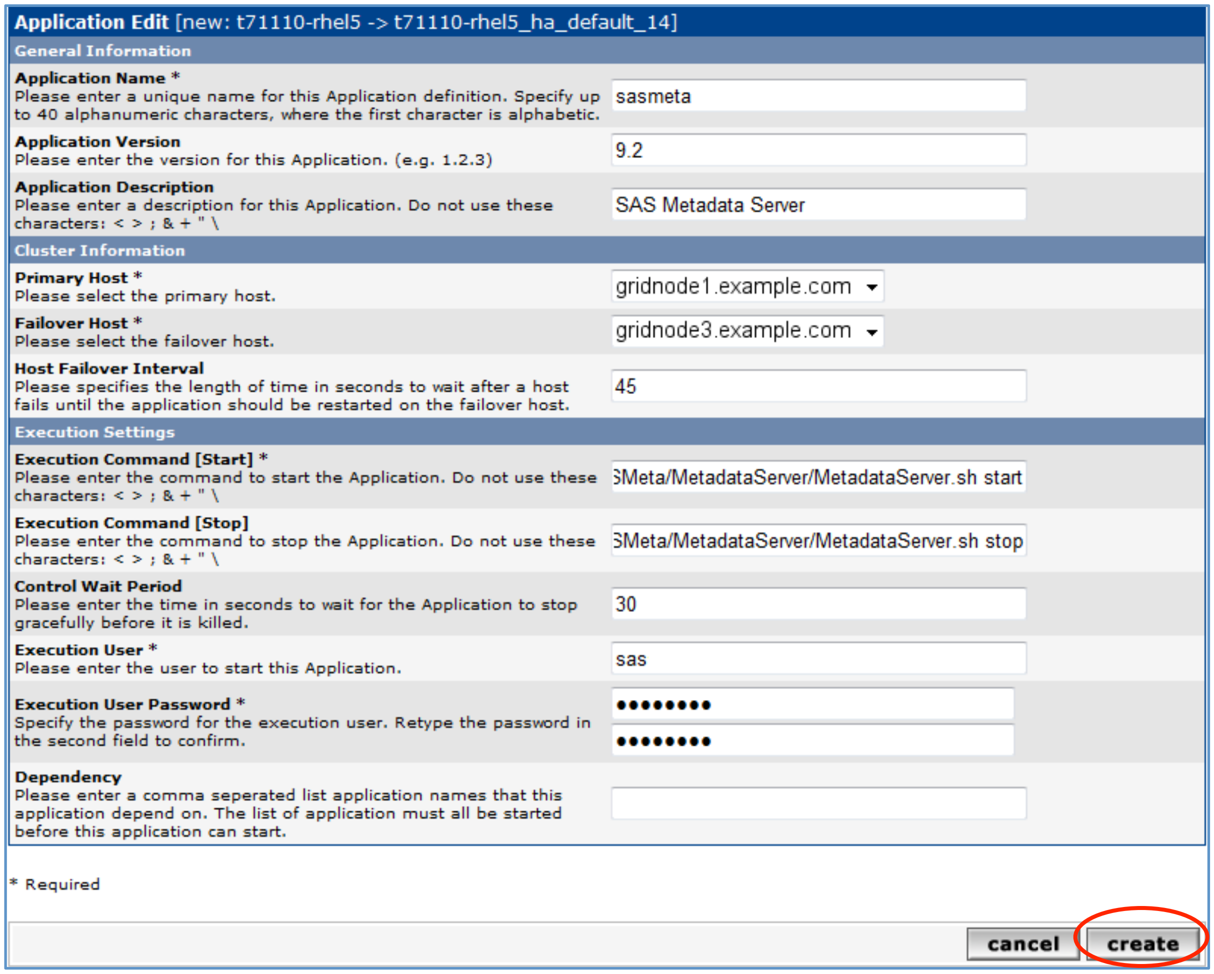
HA Applications Add

| Application Name** | Application Description | Cluster | Application Version | ☐ |
|---|---|---|---|---|
| *No Configured Applications* | | | | |

Choose an action: Copy go

The Application Edit form will be displayed, this form is used to create the EGO Service Controller definition for the service. The important settings to be entered in this form are described below:

- **Application Name** – Short name, or alias, for this service.
- **Primary Host / Failover Host** – Select the primary and secondary grid nodes used for execution of this service.
- **Execution Command [Start] / [Stop]** – Commands to start and stop the service; these commands must be executable on both the primary and failover host using the exact same command line.
- **Execution User and Password** – User and password for execution of the service; the specified user must have the necessary permissions to start and stop this service.
- **Dependency** – Optional; specifies the Application Name of any other EGO services that must be running before this service can be started.



**Application Edit** [new: t71110-rhel5 -> t71110-rhel5_ha_default_14]

**General Information**

| | |
|---|---|
| **Application Name** * Please enter a unique name for this Application definition. Specify up to 40 alphanumeric characters, where the first character is alphabetic. | sasmeta |
| **Application Version** Please enter the version for this Application. (e.g. 1.2.3) | 9.2 |
| **Application Description** Please enter a description for this Application. Do not use these characters: < > ; & + " \ | SAS Metadata Server |

**Cluster Information**

| | |
|---|---|
| **Primary Host** * Please select the primary host. | gridnode1.example.com |
| **Failover Host** * Please select the failover host. | gridnode3.example.com |
| **Host Failover Interval** Please specifies the length of time in seconds to wait after a host fails until the application should be restarted on the failover host. | 45 |

**Execution Settings**

| | |
|---|---|
| **Execution Command [Start]** * Please enter the command to start the Application. Do not use these characters: < > ; & + " \ | SMeta/MetadataServer/MetadataServer.sh start |
| **Execution Command [Stop]** Please enter the command to stop the Application. Do not use these characters: < > ; & + " \ | SMeta/MetadataServer/MetadataServer.sh stop |
| **Control Wait Period** Please enter the time in seconds to wait for the Application to stop gracefully before it is killed. | 30 |
| **Execution User** * Please enter the user to start this Application. | sas |
| **Execution User Password** * Specify the password for the execution user. Retype the password in the second field to confirm. | •••••••• / •••••••• |
| **Dependency** Please enter a comma seperated list application names that this application depend on. The list of application must all be started before this application can start. | |

* Required

Once all the desired settings have been entered into this form, click on the **create** button to submit the information and then answer **yes** when prompted for confirmation to complete adding the service definition.

The important service definition settings for some critical SAS services are discussed below. Refer to Defining SAS Services in EGO Service Controller Configuration for details on how to better determine the exact settings to be used.

### *SAS Metadata Server*

For the SAS Metadata Server, the key settings for the Application Edit form are as follows:

- **Application Name** – Any service alias; e.g. `sasmeta`
- **Primary Host / Failover Host** – Select the primary and secondary grid nodes configured to execute the Metadata Server.
- **Execution Command [Start] / [Stop]**
  - Start (UNIX)

```
@full path to level config@/SASMeta/MetadataServer/MetadataServer.sh start
```

  - Stop (UNIX)

```
@full path to level config@/SASMeta/MetadataServer/MetadataServer.sh stop
```

Where **`@full path to level config@`** is replaced with the full directory path on the node where the SAS configuration level is located.

  - Start (Windows)

```
To Be Added
```

  - Stop (Windows)

```
To Be Added
```

- **Execution User and Password** – e.g. `sas`
- **Dependency** – None

### *SAS Object Spawner*

For the SAS Object Spawner, the key settings for the Application Edit form are as follows:

- **Application Name** – Any service alias; e.g. `sasobjspawner`
- **Primary Host / Failover Host** – Select the primary and secondary grid nodes configured to execute the Object Spawner.

NOTE: When configuring Object Spawner for active-active redundancy using load balancing, create multiple HA Applications, one for each node that will execute the Object Spawner, and set both the Primary Host and Failover Host to be the same node.

- **Execution Command [Start] / [Stop]**
  - Start (UNIX)

```
@full path to level config@/ObjectSpawner/ObjectSpawner.sh start
```

  - Stop (UNIX)

```
@full path to level config@/ObjectSpawner/ObjectSpawner.sh stop
```

Where **`@full path to level config@`** is replaced with the full directory path on the node where the SAS configuration level is located.

  - Start (Windows)

```
To Be Added
```

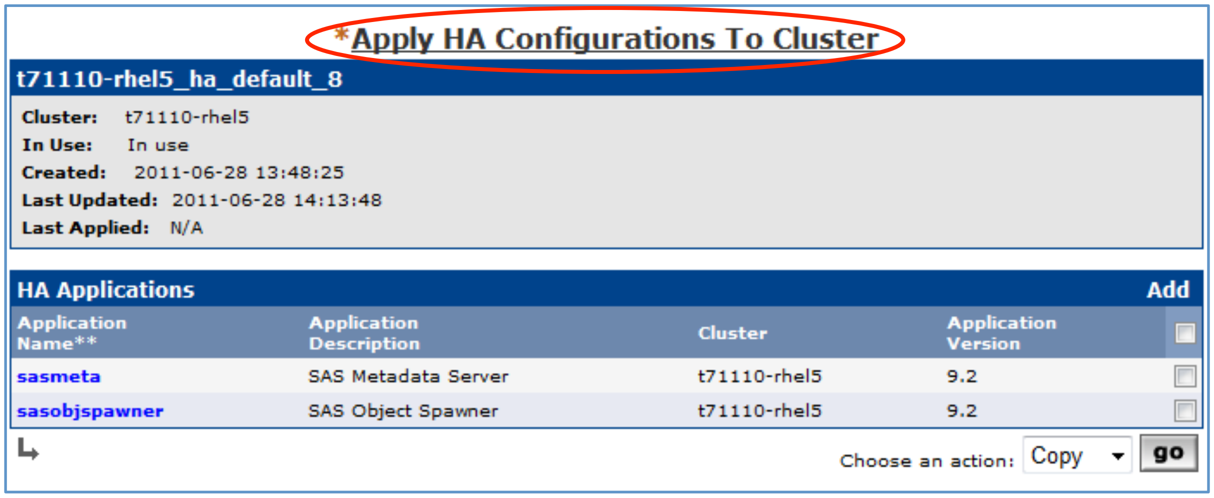
  - Stop (Windows)

```
To Be Added
```

- **Execution User and Password** – e.g. `sas`
- **Dependency** – Application Name for SAS Metadata service; e.g. `sasmeta`

## Applying HA Configuration

Once all the desired applications have been defined inside RTM, the configuration must be applied to the cluster for the settings to take effect. In the top portion of the operation pane, select the link ***Apply HA Configurations To Cluster**.



*Apply HA Configurations To Cluster

t71110-rhel5_ha_default_8

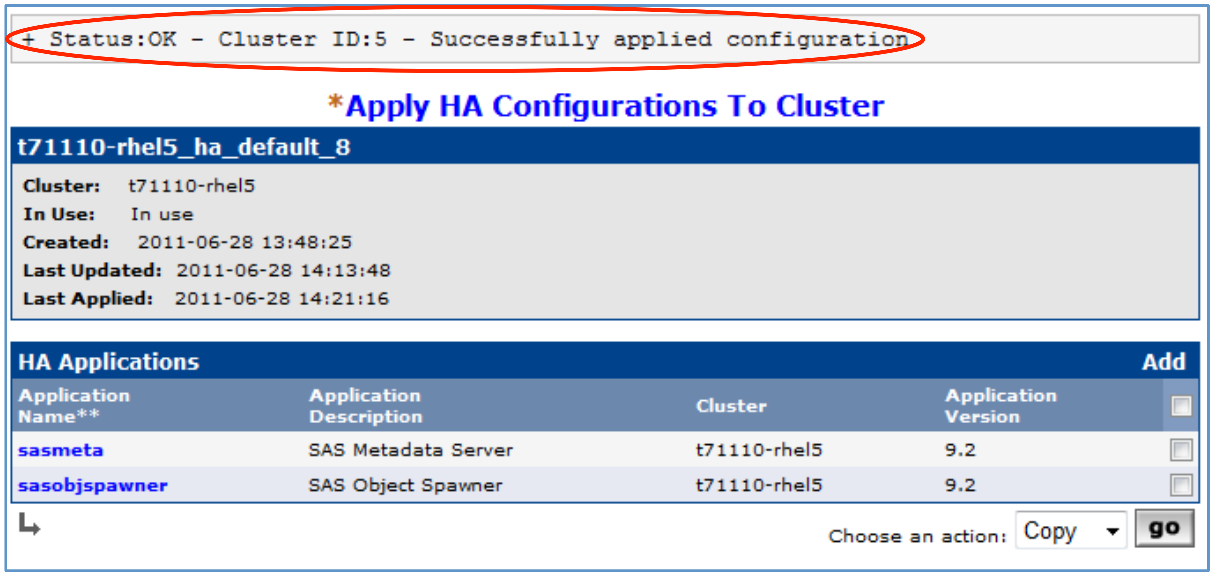
**Cluster:** t71110-rhel5
**In Use:** In use
**Created:** 2011-06-28 13:48:25
**Last Updated:** 2011-06-28 14:13:48
**Last Applied:** N/A

HA Applications — Add

| Application Name** | Application Description | Cluster | Application Version | |
|---|---|---|---|---|
| sasmeta | SAS Metadata Server | t71110-rhel5 | 9.2 | ☐ |
| sasobjspawner | SAS Object Spawner | t71110-rhel5 | 9.2 | ☐ |

Choose an action: Copy go

Once the HA configuration has been applied, the operation pane will be redisplayed with a Status message. If the configuration was correctly applied to the cluster, a message similar to the one shown below should be displayed.



```
+ Status:OK - Cluster ID:5 - Successfully applied configuration
```

*Apply HA Configurations To Cluster

t71110-rhel5_ha_default_8

**Cluster:** t71110-rhel5
**In Use:** In use
**Created:** 2011-06-28 13:48:25
**Last Updated:** 2011-06-28 14:13:48
**Last Applied:** 2011-06-28 14:21:16

HA Applications — Add

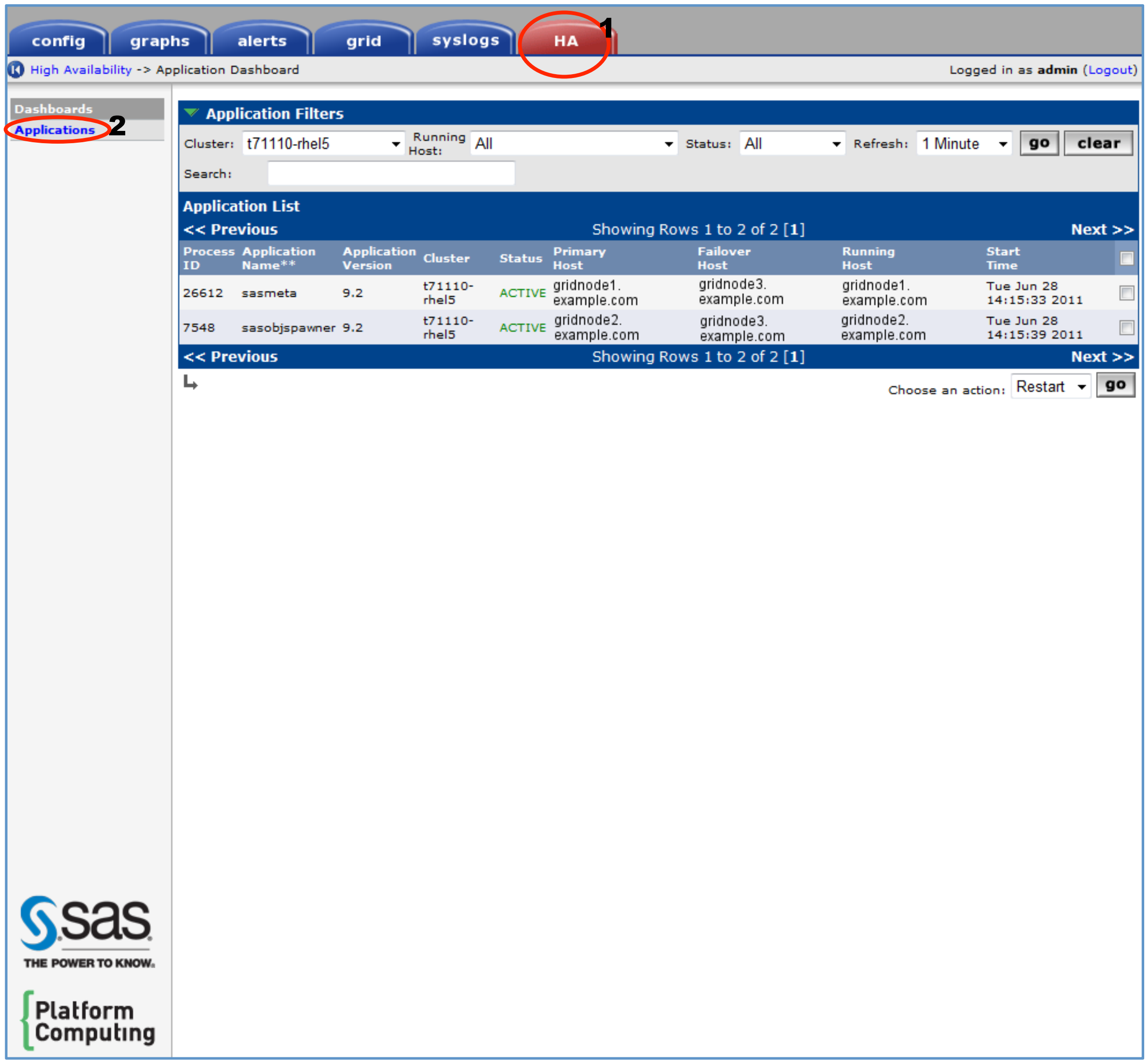
| Application Name** | Application Description | Cluster | Application Version | |
|---|---|---|---|---|
| sasmeta | SAS Metadata Server | t71110-rhel5 | 9.2 | ☐ |
| sasobjspawner | SAS Object Spawner | t71110-rhel5 | 9.2 | ☐ |

Choose an action: Copy go

## Monitoring and Controlling HA Applications

RTM also includes monitoring and control capabilities for HA services. To see the current status or to change the state of any defined service, perform the following:

1) Select **HA** from the tab list at the top of the screen.
2) On the left-hand menu, under the **Dashboards** section select **Applications**.



Cluster: t71110-rhel5 Running Host: All Status: All Refresh: 1 Minute

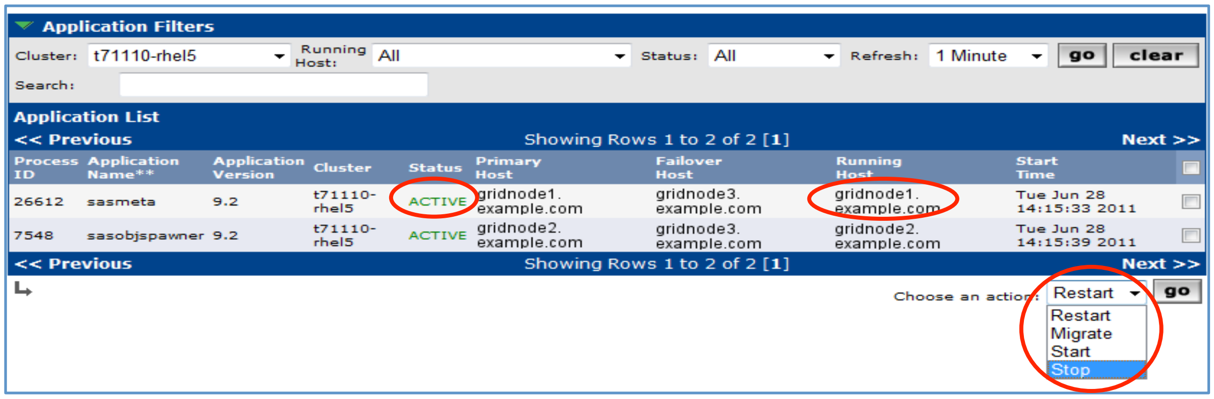
Search:

Showing Rows 1 to 2 of 2 [1]

| Process ID | Application Name** | Application Version | Cluster | Status | Primary Host | Failover Host | Running Host | Start Time | |
|---|---|---|---|---|---|---|---|---|---|
| 26612 | sasmeta | 9.2 | t71110-rhel5 | ACTIVE | gridnode1.example.com | gridnode3.example.com | gridnode1.example.com | Tue Jun 28 14:15:33 2011 | ☐ |
| 7548 | sasobjspawner | 9.2 | t71110-rhel5 | ACTIVE | gridnode2.example.com | gridnode3.example.com | gridnode2.example.com | Tue Jun 28 14:15:39 2011 | ☐ |

Choose an action: Restart

The Application List table shows the current **Status** of each defined service and other various bits of important information including the **Running Host**, or the node which is currently executing the service within the grid. From this screen a service can be stopped, started, restarted, or migrated. Migrating a service will manually switchover the service to the alternate redundant node.



Application Filters

Cluster: t71110-rhel5 Running Host: All Status: All Refresh: 1 Minute

Search:

Application List

<< Previous Showing Rows 1 to 2 of 2 [1] Next >>

| Process ID | Application Name** | Application Version | Cluster | Status | Primary Host | Failover Host | Running Host | Start Time | |
|---|---|---|---|---|---|---|---|---|---|
| 26612 | sasmeta | 9.2 | t71110-rhel5 | ACTIVE | gridnode1.example.com | gridnode3.example.com | gridnode1.example.com | Tue Jun 28 14:15:33 2011 | ☐ |
| 7548 | sasobjspawner | 9.2 | t71110-rhel5 | ACTIVE | gridnode2.example.com | gridnode3.example.com | gridnode2.example.com | Tue Jun 28 14:15:39 2011 | ☐ |

<< Previous Showing Rows 1 to 2 of 2 [1] Next >>

# Service Relocation

Services that are both redundant and resilient are technically Highly Available as they can recover from faults and transition to alternate execution resources when necessary. The service will be available a high percentage of the time, but exactly how would a client determine where the service is currently located if it can be moved around between different grid nodes? It makes little difference if the service is HA if it can't be accessed by clients after it is restarted on a secondary resource. In order to locate an HA service within the grid there must be some mechanism to allow access to a running service without requiring clients to have knowledge of which grid node is currently executing it. This aspect of High Availability is quite possibly the most important and the most complex. For Service Relocation within the grid there are two different approaches to consider:

- **IP Load Balancer (Hardware)** – hardware based solution which requires the addition of an IP Load Balancing switch.
- **EGO Service Director (Software)** – software based solution that utilizes the EGO Service Director included with Platform Suite for SAS.

With either solution the fundamental concepts are the same:

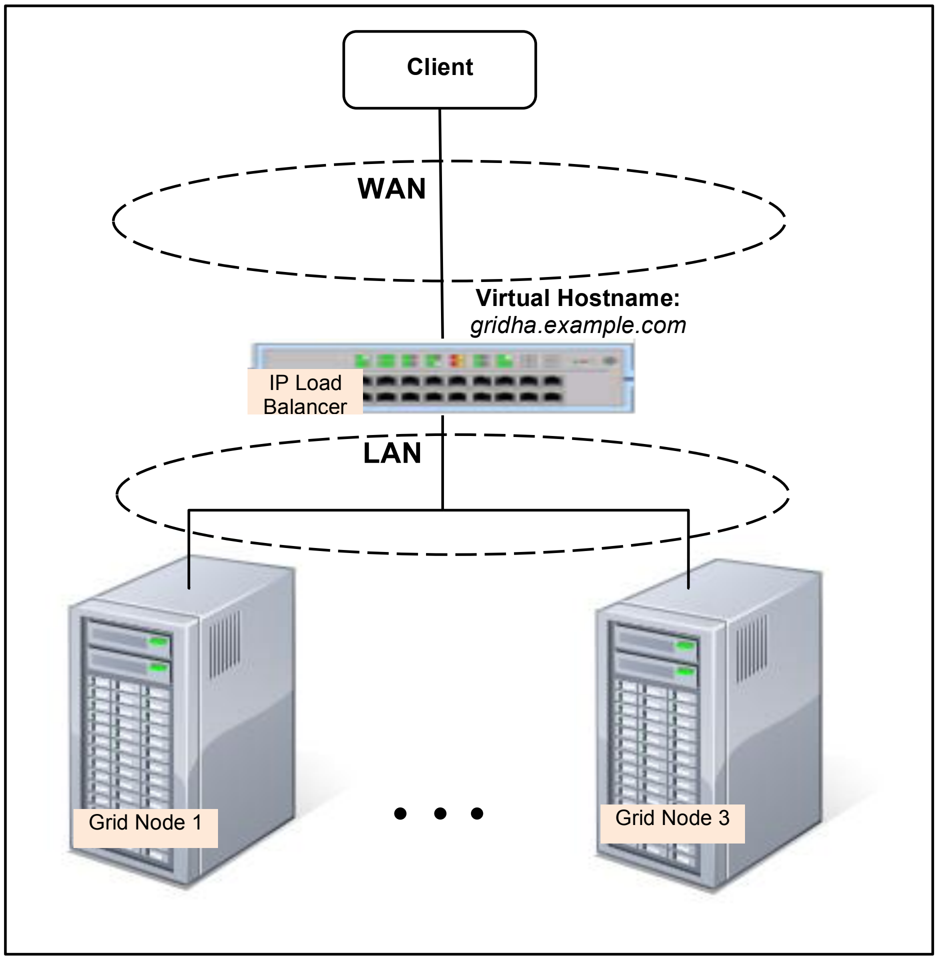
- Define a virtual hostname for the services that are to be HA within the grid.
- Any client wishing to access an HA grid service must use the virtual hostname.
- The virtual hostname is resolved to the current physical location of the service within the grid.

The method used to resolve the virtual hostname to the service's current physical location is completely hidden from the client; it is exactly the same as any other host to which a connection is to be made. However, the detail of how virtual hostnames are resolved is where the two solutions differ.

## IP Load Balancer (Hardware)

The IP Load Balancer approach requires additional hardware to be installed and configured for the grid, thus this is frequently referred to as the Hardware approach. In some deployments the required hardware may already be present in the network allowing the grid to leverage existing infrastructure. Figure 3 below shows a logical network diagram with a load balancer configured with a single virtual hostname (e.g. *gridha.example.com)*. In this configuration the clients would connect using the virtual hostname *gridha.example.com*, this hostname resolves to the IP address of the load balancer switch. Based on the load balancer configuration, the switch would then decide which physical host is available to respond to client requests. When services are failed over or migrated to redundant resources by the EGO Service Controller the load balancer is responsible for detecting the change and routing new connections to the correct destination.

***Figure 3: IP Load Balancer Logical Network***



The following example scenario describes how a service would be relocated during failover.

- A client attempts a connection to *gridha.example.com* port number 8561.
- The IP Load Balancer has been configured to monitor port 8561 on Grid Node 1 and Grid Node 3 and redirect connections to either port; i.e. common Round Robin scheduling.
- The SAS Metadata server is currently executing on Grid Node 1 listening on port 8561, while Grid Node 3 is the secondary resource and currently is not executing the service.
- The IP Load Balancer will redirect all incoming connections for port 8561 to Grid Node 1.
- Grid Node 1 is taken offline by a fault or manually by an administrator.
- The IP Load Balancer detects the disruption and traffic destined for port 8561 is momentarily disrupted as there is no destination available.
- EGO Service Controller starts up the SAS Metadata Server on Grid Node 3.
- IP Load Balancer detects that port 8561 is now in-service on Grid Node 3.
- The IP Load Balancer will redirect all incoming connections for port 8561 to Grid Node 3.
- The client establishes a new connection to the SAS Metadata Server now executing on Grid Node 3.

## EGO Service Director (Software)

The EGO Service Director is a Platform EGO service that functions as a locating mechanism for other Platform EGO services. The EGO Service Director contains a stand-alone DNS server, which can serve as the authoritative name server for the EGO DNS subdomain and respond to DNS queries for EGO HA services. The EGO Service Director relies on the EGO Service Controller to provide location information and state change notifications of service instances. Each service which is defined in the EGO Service Controller configuration will be associated with a DNS record which includes a virtual hostname, or alias, for the service. When a service instance begins execution, enters the RUN state, the EGO Service Director will add a corresponding DNS record with the service's current location information. When a service instance transitions from RUN to any other state, the EGO Service Director will delete the associated DNS record. When the location of the EGO Service Director DNS server or any other EGO HA service is changed, the corresponding DNS record in the DNS database is updated with the IP address for the new location. In order for clients to recognize this address change in a timely manner it is crucial that every attempt to resolve an EGO service's virtual hostname receive the most up to date information. For this reason, the Time-To-Live (TTL) setting for EGO virtual hostnames and operating system DNS caching mechanisms can negatively impact the overall time required to realize EGO service relocation. The TTL setting for EGO virtual hostnames should be tuned to a marginally low duration as the total failover time is directly proportional, (e.g. total failover time = time to detect service failure + time to initialize redundant service + *time for client to realize service relocation*).

The virtual hostnames for the EGO HA services that are dynamically updated by the EGO Service Director will always be in a subdomain of the corporate DNS domain. For instance, if the corporate domain were `example.com` all virtual hostnames for EGO HA services by default would be in the subdomain `ego.example.com`. The name of the EGO subdomain can be configured as required, but the dynamic virtual hostnames for the EGO services must always be inside the EGO subdomain. The host portion of the virtual hostname for an EGO service will be the Application Name assigned to this service in the EGO Service Controller definition. So for example, if the EGO Service Controller were to define an application named `sasmeta`, the EGO Service Director will track the location of this service and dynamically update the virtual hostname `sasmeta.ego.example.com`.

The following example scenario describes how a service would be relocated during failover.

- The SAS Metadata server is currently executing on Grid Node 1, while Grid Node 3 is the secondary resource and currently is not executing the service.
- A client attempts a connection to *sasmeta.ego.example.com*.
- The client's DNS query for the virtual hostname *sasmeta.ego.example.com* will return the IP address of Grid Node 1.
- Grid Node 1 is taken offline by a fault or manually by an administrator.
- EGO Service Controller informs EGO Service Director of the service state change and the EGO Service Director dynamically deletes the DNS record for *sasmeta.ego.example.com*.
- Any DNS queries for the virtual hostname *sasmeta.ego.example.com* will return 'Host Unknown' as there is currently no destination available.
- EGO Service Controller starts up the SAS Metadata Server on Grid Node 3.
- EGO Service Controller informs EGO Service Director of the service state change and EGO Service Director dynamically adds a DNS record for *sasmeta.ego.example.com* with the IP of Grid Node 3.
- DNS queries for the virtual hostname *sasmeta.ego.example.com* will now return the IP address of Grid Node 3.
- Clients can reconnect to the SAS Metadata Server now executing on Grid Node 3.

When utilizing the EGO Service Director for Service Relocation within the grid there is one key decision that will determine which of the possible setups can be used:

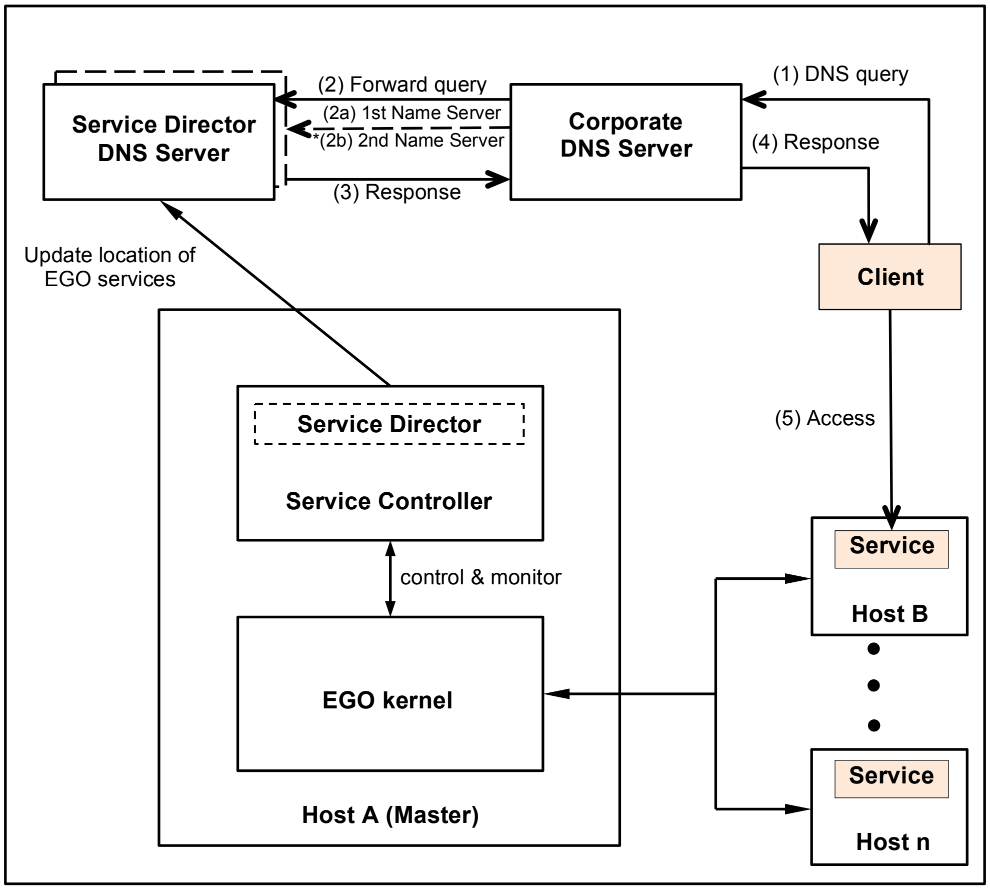
- Will the EGO Service Director be allowed to send dynamic updates to the corporate DNS server?
    - **No** – The only possible configuration is described below in EGO Service Director without Dynamic Updates of Corporate DNS Server.
    - **Yes** – There are two possibilities; refer to EGO Service Director with Dynamic Updates of Corporate DNS Server Allowed below for more details.

The EGO Service Director cannot be configured with Platform RTM for SAS, please refer to Appendix B for detailed configuration instructions and examples for each of the setups described below.

## EGO Service Director without Dynamic Updates of Corporate DNS Server

In this configuration the EGO Service Director DNS is setup to master a subdomain, or subzone, which will be used to resolve the location of virtual hostnames for the HA services within the grid. The corporate DNS server will forward all queries for any of the virtual hostnames within that subdomain to the EGO Service Director DNS server. It is important that the corporate DNS server be configured with multiple Name Server records for the EGO subdomain, one for each of the redundant nodes that can possibly execute the EGO Service Director DNS server. The detailed flow of a DNS query for an EGO service virtual hostname is shown below in Figure 4.

***Figure 4: EGO Service Director – No Corporate DNS Updates***



In the example above, when the query is forwarded to the EGO Service Director DNS, step 2, the flow includes sub-steps 2a and 2b. This is shown to highlight the fact that the EGO Service Director itself is configured to be HA so there will be redundant resources configured to execute the service and this means the corporate DNS server must have multiple Name Server records configured for the EGO subdomain. On initial startup and after an EGO Service Director failover or switchover, the corporate DNS server can't possibly know which Name Server is active and so it may be required to forward the query to multiple servers before receiving a response. The figure only depicts two Name Servers for the EGO subdomain, but if there are more redundant nodes configured for the EGO Service Director, the query could quite possibly need to be forwarded to each of these nodes (i.e. max forwards = number of EGO Service Director redundant nodes).

## EGO Service Director with Dynamic Updates of Corporate DNS Server Allowed

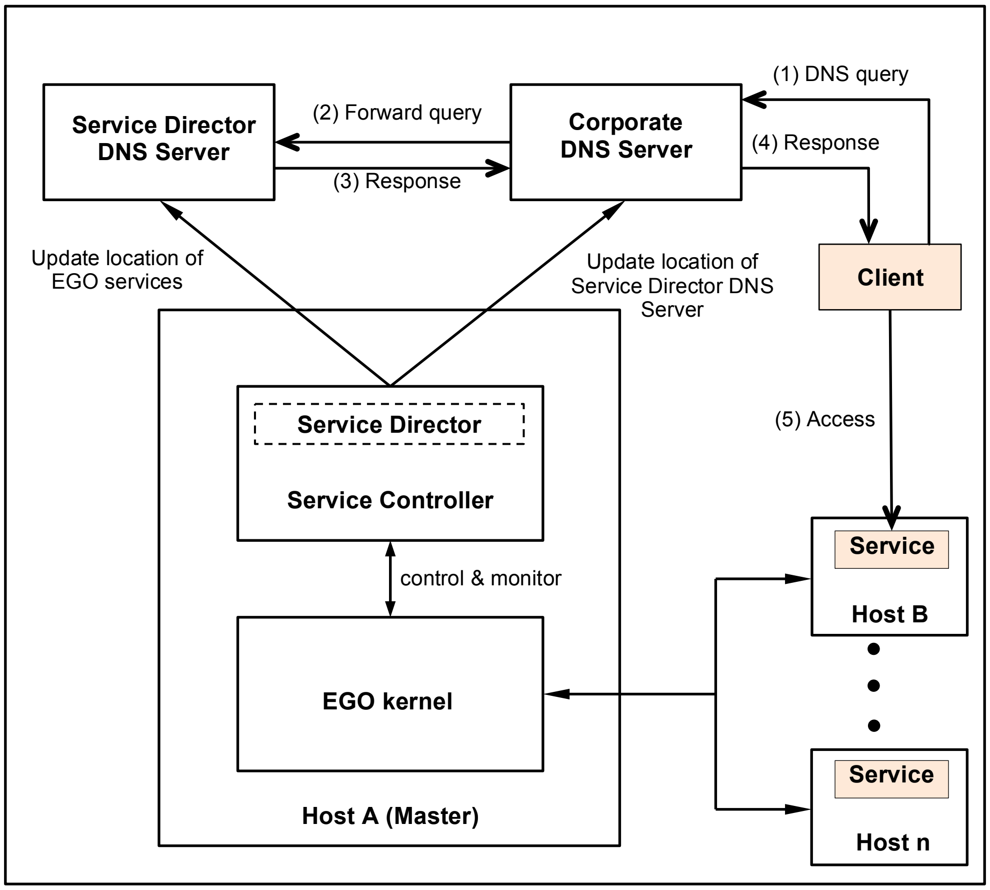
The EGO Service Director's job is to dynamically update the location of all the services under the control of the EGO Service Controller, and allowing these updates to be sent directly to the corporate DNS server will produce more efficient DNS queries. The two distinct possibilities for these updates are detailed in the following sections.

### *Corporate DNS Updates: EGO Service Director DNS Server Location only*

Since the EGO Service Director itself is configured to be a resilient service managed by the EGO Service Controller, its location would be dynamically updated if the service were to failover or be manually switched to a different host. Allowing the location of the EGO Service Director DNS server to be dynamically updated in the corporate DNS server will eliminate the potential inefficiencies of having multiple Name Server records for the EGO subdomain. For this configuration to be possible, the virtual hostname of the EGO Service Director DNS server must reside inside the corporate domain, e.g. if the corporate domain were *example.com* and the EGO subdomain *ego.example.com*, the EGO Service Director virtual hostname would be something like *egonameserver.example.com*.

Just as with the previous configuration, the EGO Service Director DNS is setup to master a subdomain which is used to resolve the location of virtual hostnames for all EGO services within the grid, the only difference in this setup is that the corporate DNS server is configured with only a single Name Server record for the EGO subdomain. This Name Server record is configured using the virtual hostname of the EGO Service Director DNS server which will be dynamically updated in the corporate DNS with the current location of this service. Details for a DNS query are shown below in Figure 5.

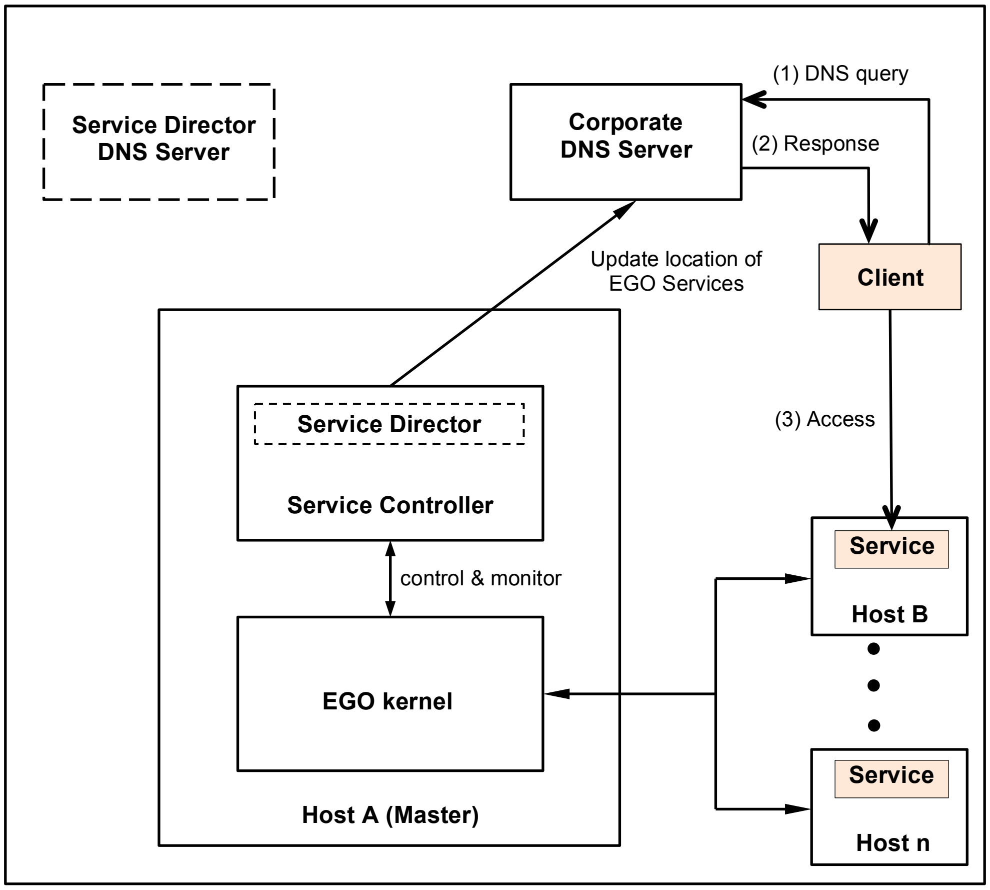
***Figure 5: EGO Service Director – Corporate DNS Updates for EGO Service Director DNS Server***



### *Corporate DNS Updates: Location of All EGO HA Services*

In both of the previous setups, the corporate DNS server must delegate resolution for the EGO virtual hostnames to the EGO Service Director DNS server, but in most scenarios the EGO subdomain will only contain a small handful of hostnames. The efficiency of the DNS queries could be substantially improved if the corporate DNS server knew the latest location information for the EGO services and could respond with authority to queries for the EGO virtual hostnames. Details for a DNS query with this setup are shown below in Figure 6.

***Figure 6: EGO Service Director – Corporate DNS Updates for All HA Services***



The EGO Service Director DNS server is shown in the figure as it is still required, but it would not receive any dynamic updates and would not be involved with any DNS queries. In this setup the EGO subdomain is mastered by the corporate DNS server so it isn't necessary to involve the EGO Service Director DNS when querying an EGO virtual hostname.

This would be the most efficient setup as far as DNS queries are concerned, but there is one potential pitfall with this configuration. As previously noted, the DNS Time-To-Live for the EGO virtual hostnames must be set low such that clients will recognize location changes for services and not be delayed by DNS name caching, but in this configuration the EGO Service Director DNS server is not in control of resolving these hostnames. This means that the TTL for the EGO virtual hostnames is entirely controlled by the corporate DNS server and must be tuned correctly or failover times for EGO HA services could be unacceptably high. The details for tuning TTL in the corporate DNS servers are beyond the scope of this document as various DNS server software will have differing methods, but if using the above setup make certain to consider this issue in the configuration.

## Comparison Hardware –VS– Software

Both of the approaches for Service Relocation have their pluses and minuses, and exactly which of these solutions might be best for any given grid deployment really must be decided on a case by case basis. Table 1 below contains a point by point comparison of the two solutions:

***Table 1: Pros and Cons of Service Relocation Alternatives***

| IP Load Balancer (Hardware) | EGO Service Director (Software) |
|---|---|
| **Advantages** | **Disadvantages** |
| Single IP address; no IP address change during failover, clients and dependent services can maintain or reestablish connections after failover. | Multiple IP addresses; IP address change during failover requires some clients and dependent services to be restarted. |
| Quick failover; switching incoming traffic to the failover host is nearly instantaneous. | Potentially slower failover; DNS Time-To-Live, propagation of DNS updates, and corporate DNS setups can cause unexpected failover delay. |
| Minimal risk for enterprise network disruption as the configuration changes are localized to the LAN segment containing the IP Load Balancer. | Attempts to minimize failover time with improper DNS configuration tuning could cause network delays for the entire enterprise. |
| Less software complexity; fewer third party configuration steps required for SAS deployment. | Increased software complexity; additional third party configuration steps required for SAS deployment. |
| No conflicts with corporate IT policies. | Possible conflicts with corporate IT policies; many enterprises have strict policies disallowing usage of secondary DNS servers. |
| **Disadvantages** | **Advantages** |
| More expensive; additional hardware to purchase and maintain. | Less expensive; no additional hardware to purchase and maintain. EGO Service Director is included with SAS Grid Manager license costs. |
| Full fault tolerance requires an HA solution for the IP Load Balancer; could increase total cost and add complexity to the overall solution. | EGO Service Director leverages the EGO functionality to provide full fault tolerance with zero added expense. |

Both solutions are totally viable and the right option must be determined based on the specific customer’s IT requirements.

# SAS Configuration for HA Services

In order to take full advantage of the High Availability of services within the grid all clients and dependent services must be correctly configured. The necessary configuration changes are to make certain that all references to EGO HA services use the virtual hostnames created for the services.

## Configuring Virtual Hostnames

As discussed in the previous section, depending on the solution used for Service Relocation the virtual hostname for an HA service can either be the hostname assigned to the IP Load Balancer or the alias for the service within the EGO subdomain. No matter which solution is used to relocate a service all SAS configuration must use the appropriate virtual hostname, never the IP address or the real hostname of a single grid node.

### Initial SAS Deployment

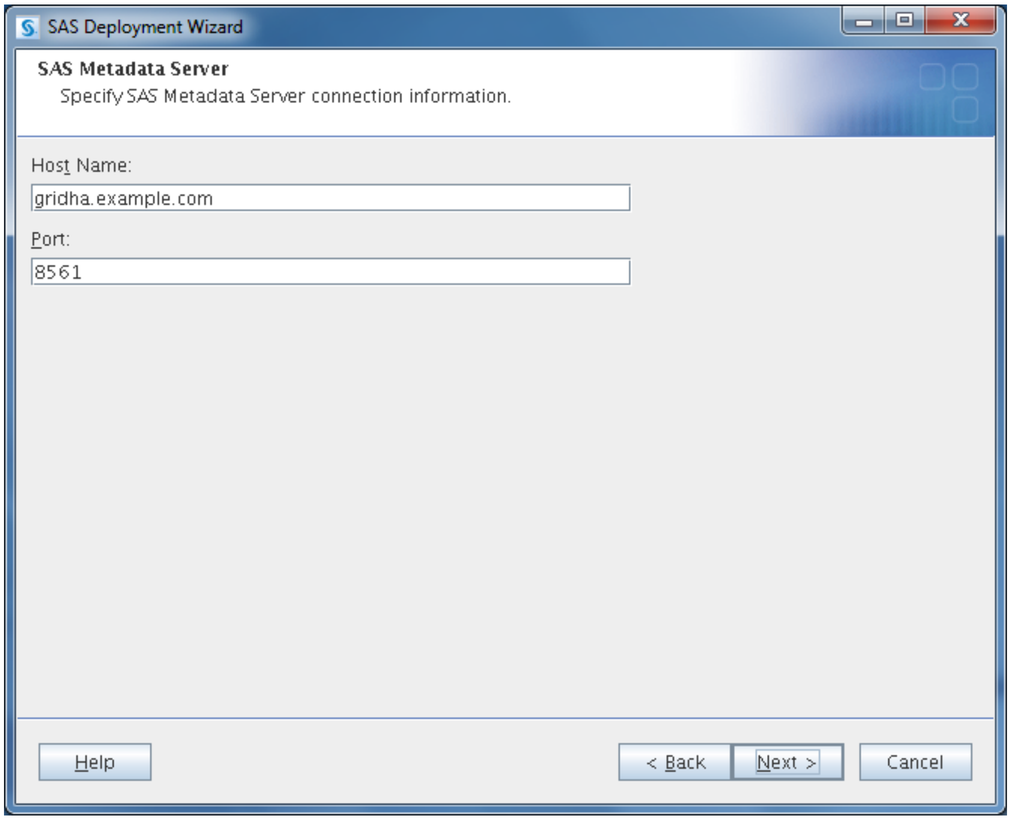
When configuring HA during the SAS deployment the virtual hostnames for the HA services can be entered directly into the Deployment Wizard. The sections below provide some helpful tips when initially deploying important SAS HA services. Examples are provided for both Service Relocation solutions.

#### *IP Load Balancer (Hardware)*

When using an IP Load Balancer for HA Service Relocation use the hostname assigned to the IP Load Balancer in the SAS configuration when referencing any of the HA services.

- **SAS Metadata Server**

When entering the SAS Metadata Server connection information use the hostname of the IP Load Balancer for **`Host Name`** as shown below. This is the only difference from a normal SAS deployment for configuring SAS Metadata Server as HA.



**SAS Metadata Server**
Specify SAS Metadata Server connection information.

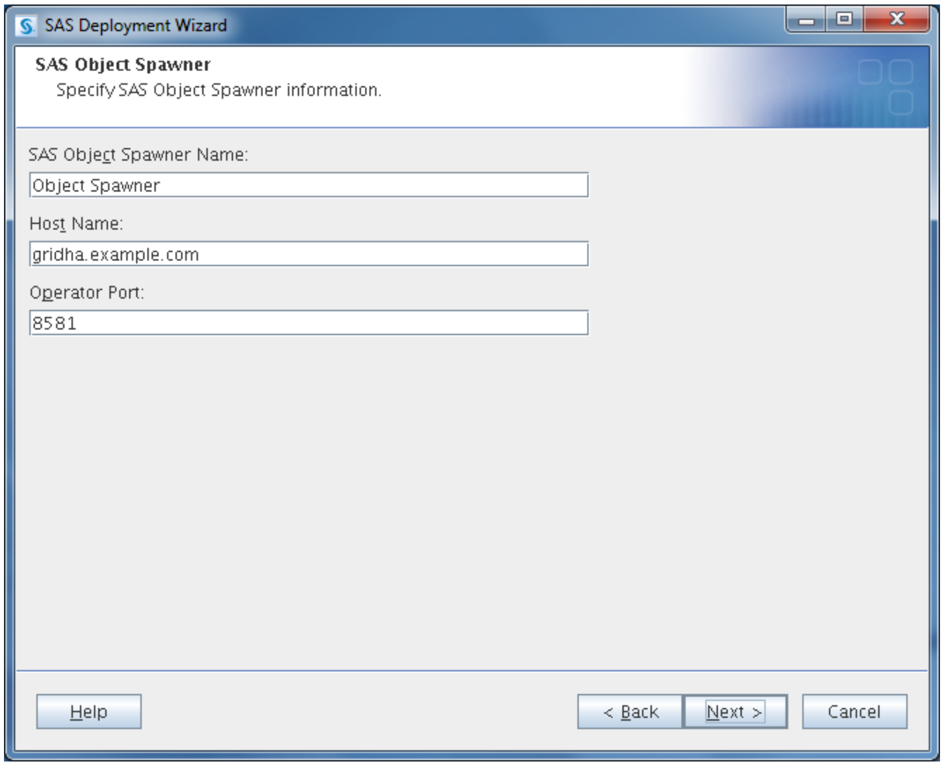
Host Name: gridha.example.com

Port: 8561

- **SAS Object Spawner**

NOTE: To configure the SAS Object Spawner as HA, the **`Prompting Level`** must be set to **`Custom`**, else the dialog shown below will not be presented.

Enter the Object Spawner information using the hostname of the IP Load Balancer for **`Host Name`**.



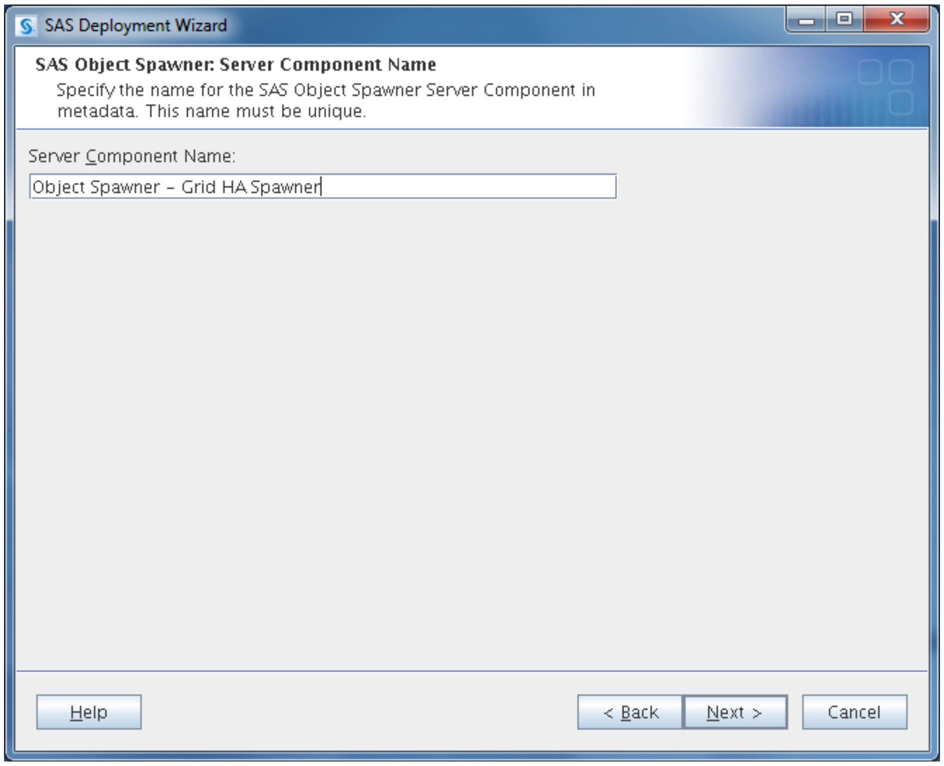
SAS Object Spawner Name: Object Spawner

Host Name: gridha.example.com

Operator Port: 8581

Proceed to verify that all SAS logical servers being deployed that utilize this Object Spawner are configured with **`Host Name`** being correctly set the same as the Object Spawner, (using the hostname of the IP Load Balancer). A few examples are included below; not all the example components, or possibly additional components, may be included with any specific deployment.

Modify the **`Server Component Name`** to highlight the fact that this Spawner is not associated with a single server. This is not required, but recommended; the name entered can be anything desired.

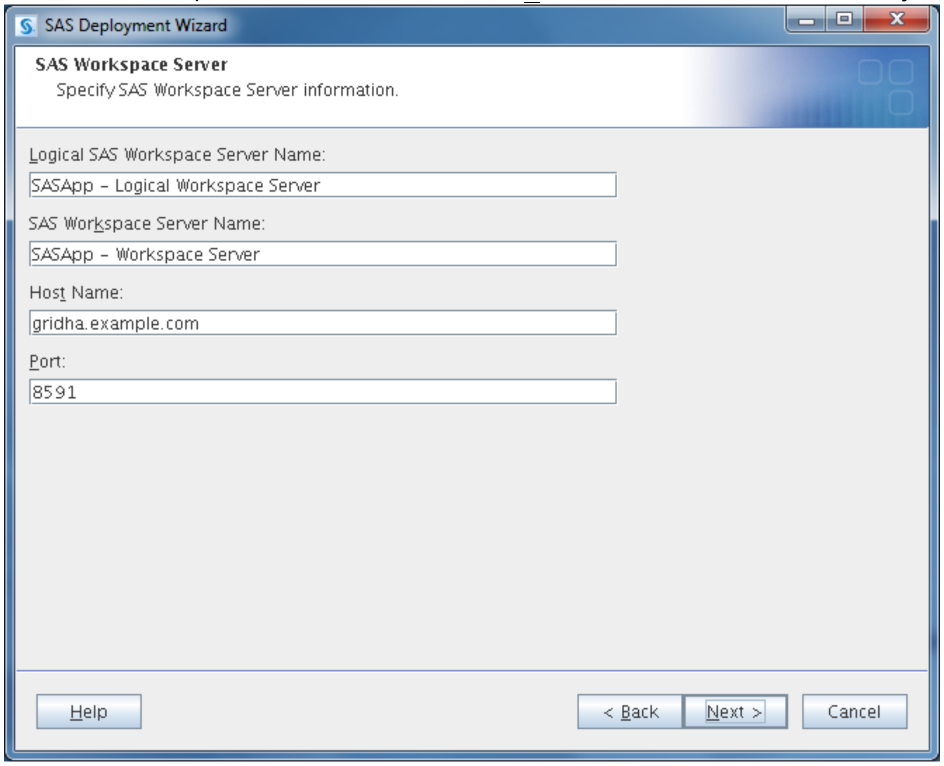


**SAS Object Spawner: Server Component Name**

Specify the name for the SAS Object Spawner Server Component in metadata. This name must be unique.

Server Component Name:
Object Spawner – Grid HA Spawner

The SAS Workspace Server must have **`Host Name`** set the same as the Object Spawner.



**SAS Workspace Server**

Specify SAS Workspace Server information.

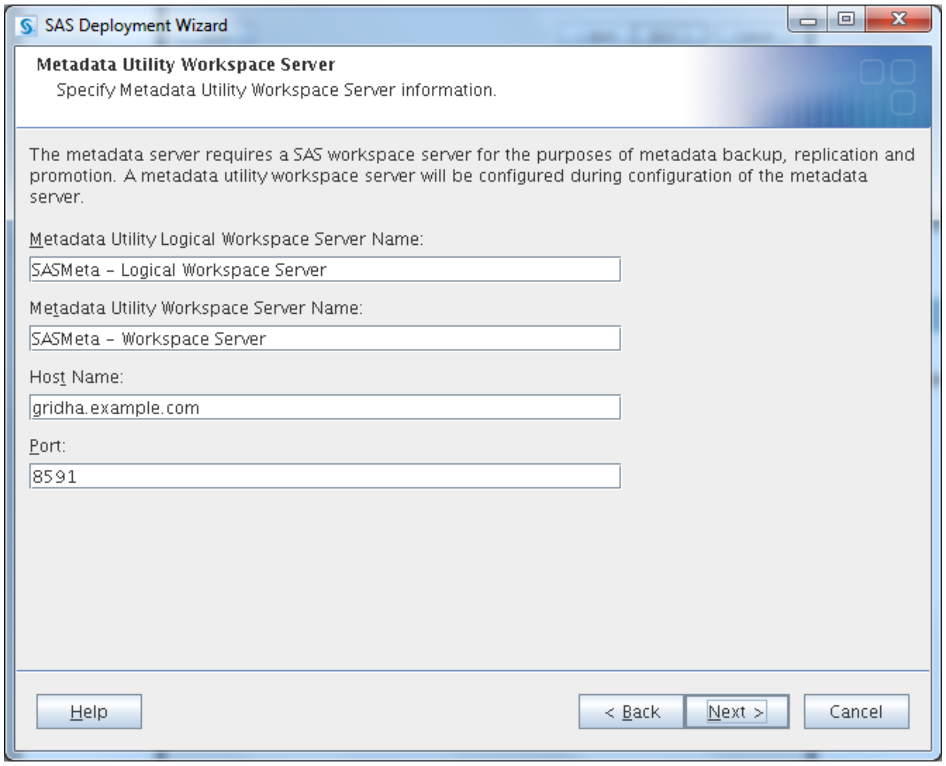
Logical SAS Workspace Server Name:
SASApp – Logical Workspace Server

SAS Workspace Server Name:
SASApp – Workspace Server

Host Name:
gridha.example.com

Port:
8591

The Metadata Utility Workspace Server must have **`Host Name`** set the same as the Object Spawner.



**Metadata Utility Workspace Server**

Specify Metadata Utility Workspace Server information.

The metadata server requires a SAS workspace server for the purposes of metadata backup, replication and promotion. A metadata utility workspace server will be configured during configuration of the metadata server.

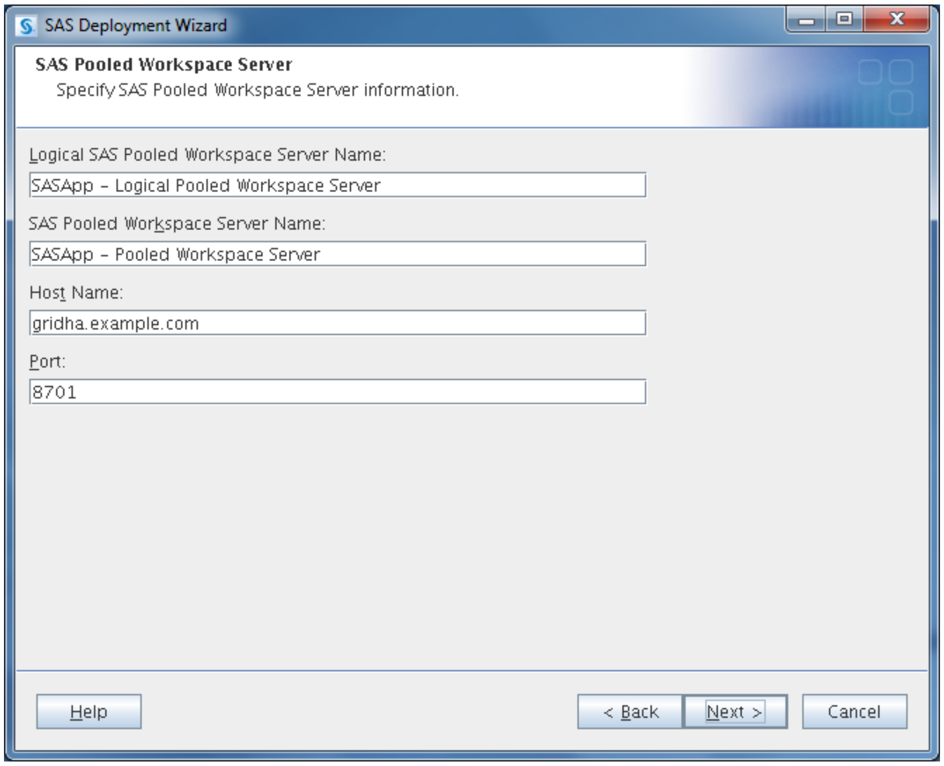
Metadata Utility Logical Workspace Server Name: SASMeta - Logical Workspace Server

Metadata Utility Workspace Server Name: SASMeta - Workspace Server

Host Name: gridha.example.com

Port: 8591

The SAS Pooled Workspace Server must have **`Host Name`** set the same as the Object Spawner.



**SAS Pooled Workspace Server**

Specify SAS Pooled Workspace Server information.

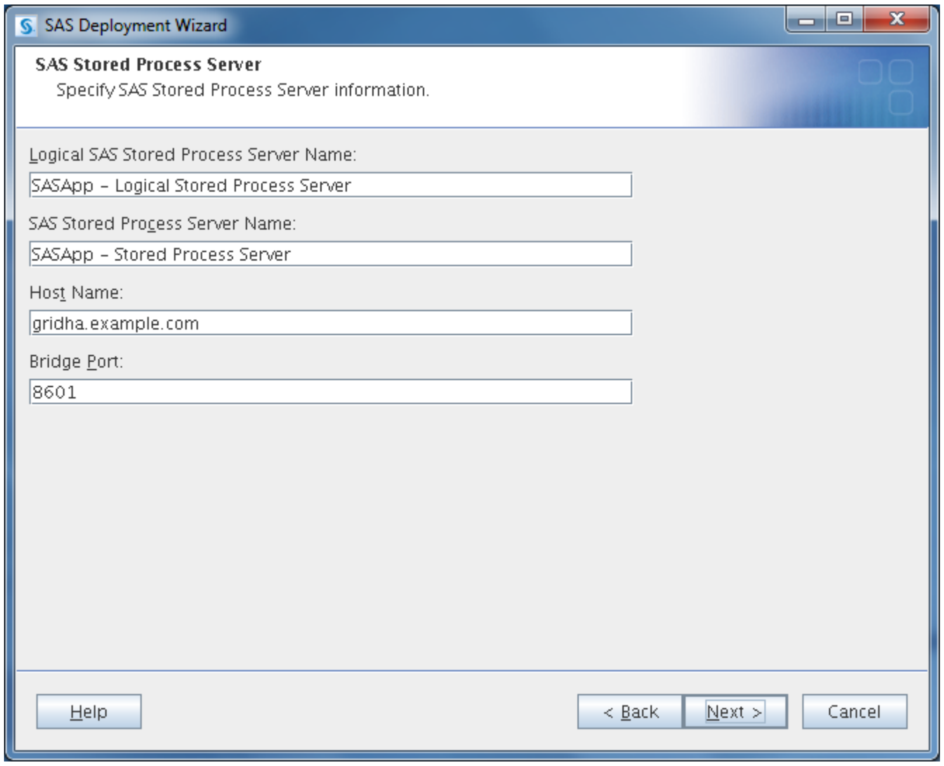
Logical SAS Pooled Workspace Server Name: SASApp - Logical Pooled Workspace Server

SAS Pooled Workspace Server Name: SASApp - Pooled Workspace Server

Host Name: gridha.example.com

Port: 8701

The SAS Stored Process Server must have **`Host Name`** set the same as the Object Spawner.



SAS Stored Process Server

Specify SAS Stored Process Server information.

Logical SAS Stored Process Server Name: SASApp – Logical Stored Process Server

SAS Stored Process Server Name: SASApp – Stored Process Server

Host Name: gridha.example.com

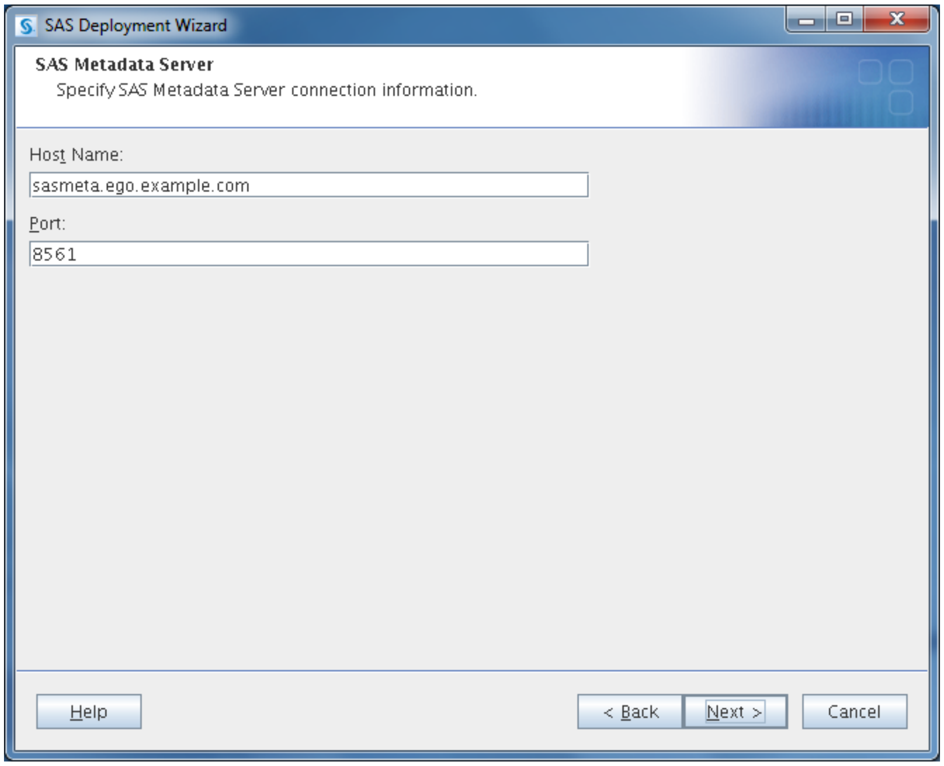
Bridge Port: 8601

### *EGO Service Director (Software)*

When using the EGO Service Director for HA Service Relocation a different virtual hostname is created for every service which is defined in the EGO Service Controller configuration. This allows the services to be independently switched to alternate resources, but does require the specific virtual hostnames to be correctly referenced in the SAS configuration. Some example SAS configurations are included below.

- **SAS Metadata Server**

When entering the connection information use the EGO subdomain virtual hostname for **`Host Name`** of the SAS Metadata Server as shown in the example below. This is the only difference from a normal SAS deployment when configuring SAS Metadata Server as HA.


SAS Deployment Wizard

SAS Metadata Server

Specify SAS Metadata Server connection information.

Host Name: sasmeta.ego.example.com

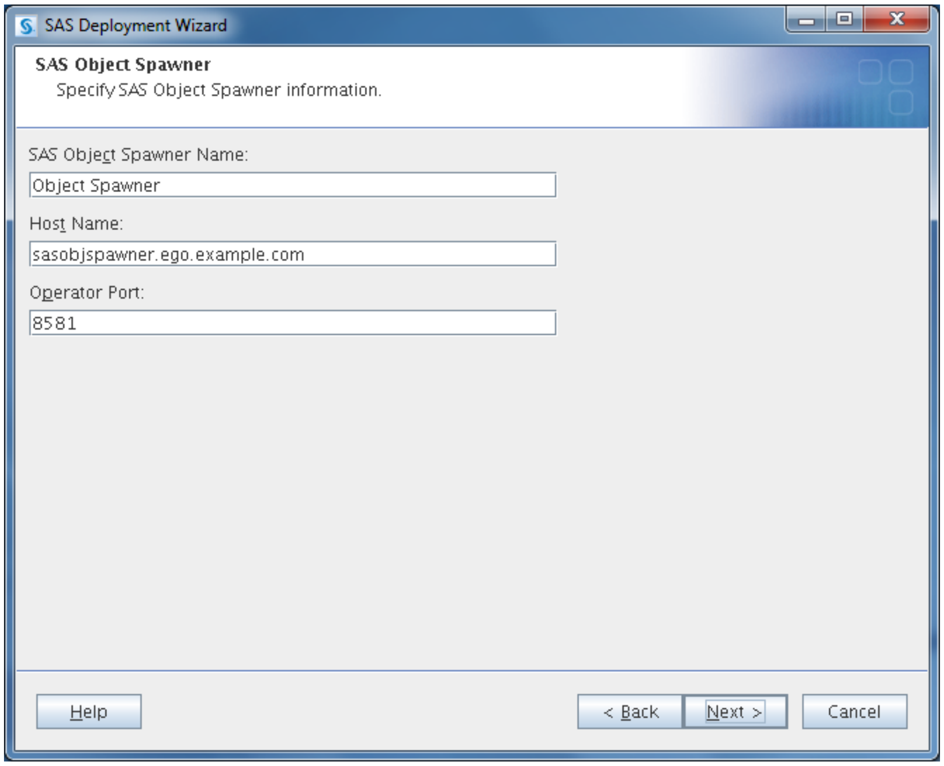
Port: 8561

NOTE: In order for the deployment wizard to complete successfully the **`Host Name`** entered here must correctly resolve, but since the EGO Service Controller cannot start the service until after the deployment has been completed, this presents a bit of a dilemma. One simple solution is to temporarily add an entry into the local server's network `hosts` file for the virtual hostname of the SAS Metadata Server making certain to remove these entries once the deployment is completed. A preferable solution would be to add a DNS `A` record for the virtual hostname of the SAS Metadata Server inside the EGO subdomain assigning the IP address of the deployment host; refer to Appendix B for more details on this possibility.

- **SAS Object Spawner**

NOTE: To configure the SAS Object Spawner as HA, the **`Prompting Level`** must be set to **`Custom`**, else the dialog shown below will not be presented.

Enter the Object Spawner information using the EGO subdomain virtual hostname for **`Host Name`**.



SAS Object Spawner Name:
Object Spawner

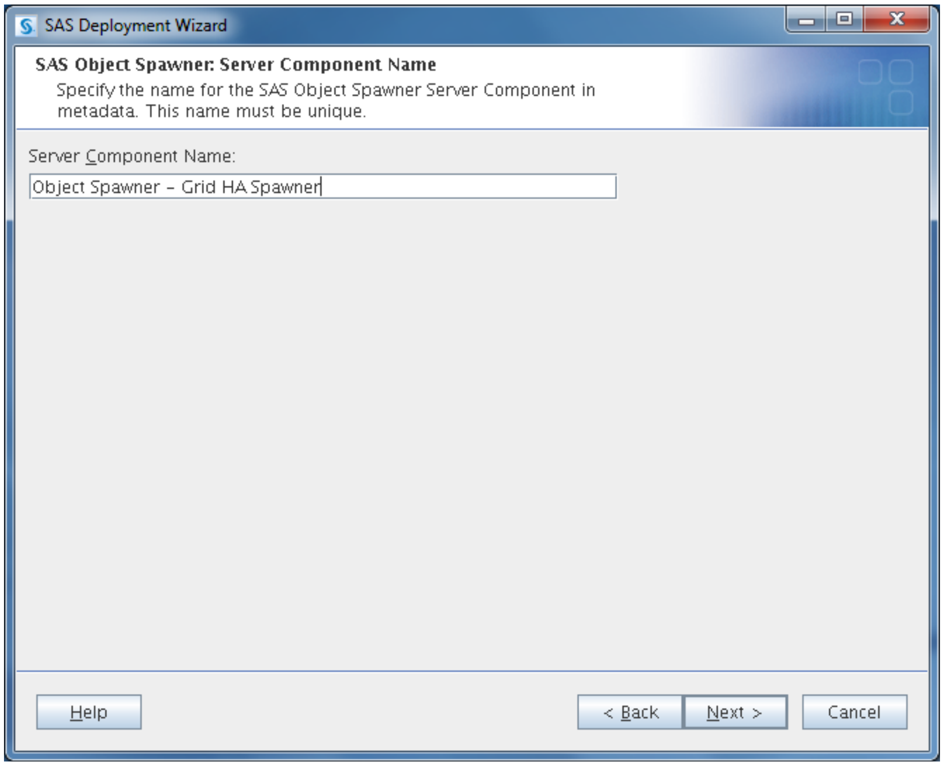
Host Name:
sasobjspawner.ego.example.com

Operator Port:
8581

NOTE: In order for the deployment wizard to complete successfully the **`Host Name`** entered here must correctly resolve, but since the EGO Service Controller cannot start the service until after the deployment has been completed, this presents a bit of a dilemma. One simple solution is to temporarily add an entry into the local server's network `hosts` file for the virtual hostname of the SAS Object Spawner making certain to remove these entries once the deployment is completed. A preferable solution would be to add a DNS `A` record for the virtual hostname of the SAS Object Spawner inside the EGO subdomain assigning the IP address of the deployment host; refer to Appendix B for more details on this possibility.

Proceed to verify that all SAS logical servers being deployed that utilize this Object Spawner are configured with **`Host Name`** being correctly set the same as the Object Spawner, using the EGO subdomain virtual hostname. A few examples are included below; not all the example components, or possibly additional components, may be included with any specific deployment.

Modify the **Server Component Name** to highlight the fact that this Spawner is not associated with a single server. This is not required, but recommended; the name entered can be anything desired.



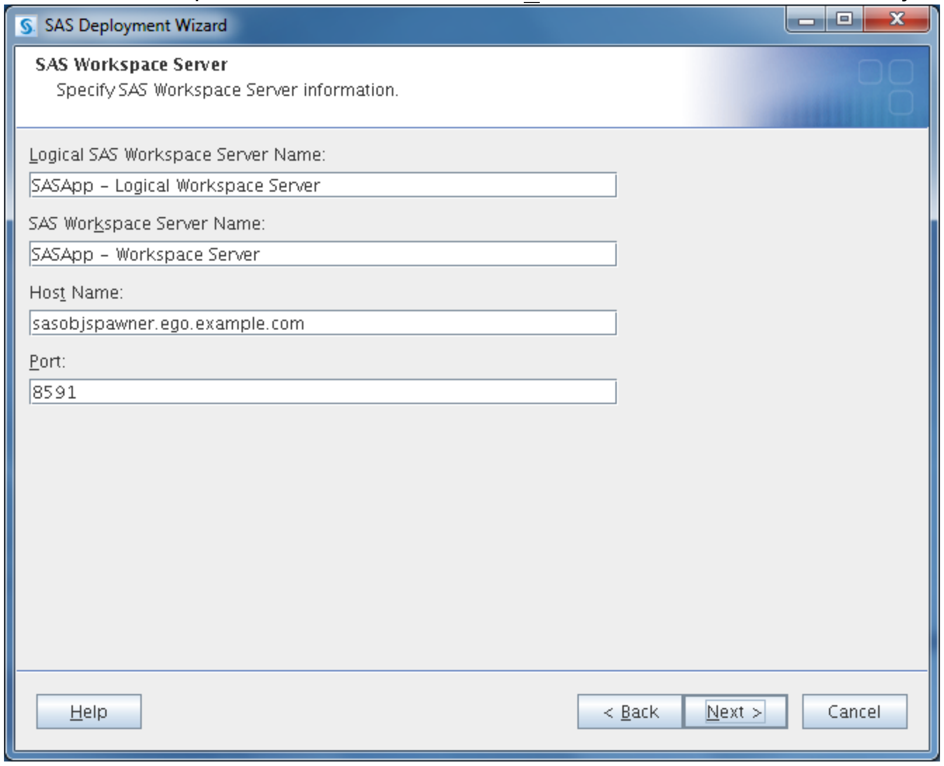
SAS Deployment Wizard

**SAS Object Spawner: Server Component Name**

Specify the name for the SAS Object Spawner Server Component in metadata. This name must be unique.

Server Component Name: Object Spawner – Grid HA Spawner

The SAS Workspace Server must have **Host Name** set the same as the Object Spawner.



SAS Deployment Wizard

**SAS Workspace Server**

Specify SAS Workspace Server information.

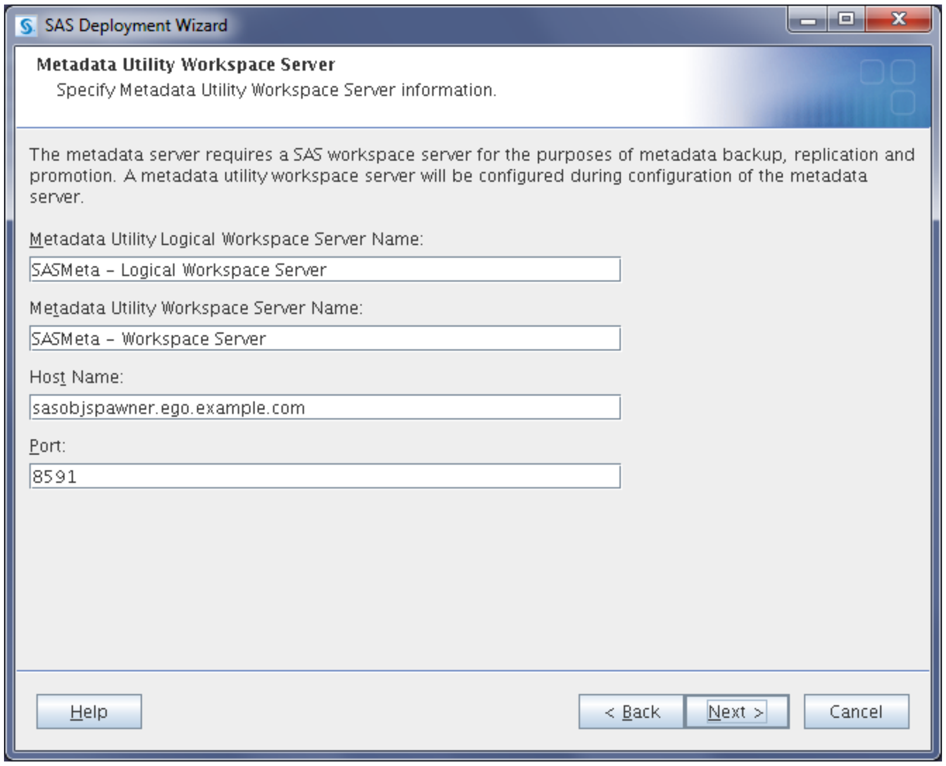
Logical SAS Workspace Server Name: SASApp – Logical Workspace Server

SAS Workspace Server Name: SASApp – Workspace Server

Host Name: sasobjspawner.ego.example.com

Port: 8591

The Metadata Utility Workspace Server must have **`Host Name`** set the same as the Object Spawner.


SAS Deployment Wizard

**Metadata Utility Workspace Server**

Specify Metadata Utility Workspace Server information.

The metadata server requires a SAS workspace server for the purposes of metadata backup, replication and promotion. A metadata utility workspace server will be configured during configuration of the metadata server.

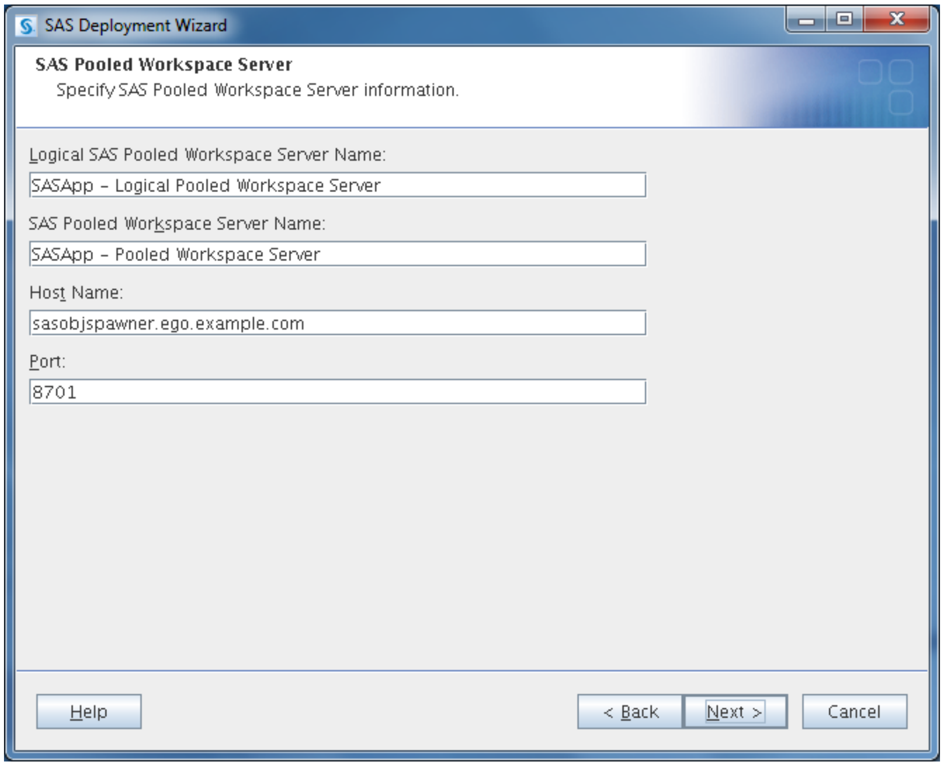
Metadata Utility Logical Workspace Server Name: SASMeta - Logical Workspace Server

Metadata Utility Workspace Server Name: SASMeta - Workspace Server

Host Name: sasobjspawner.ego.example.com

Port: 8591

The SAS Pooled Workspace Server must have **`Host Name`** set the same as the Object Spawner.


SAS Deployment Wizard

**SAS Pooled Workspace Server**

Specify SAS Pooled Workspace Server information.

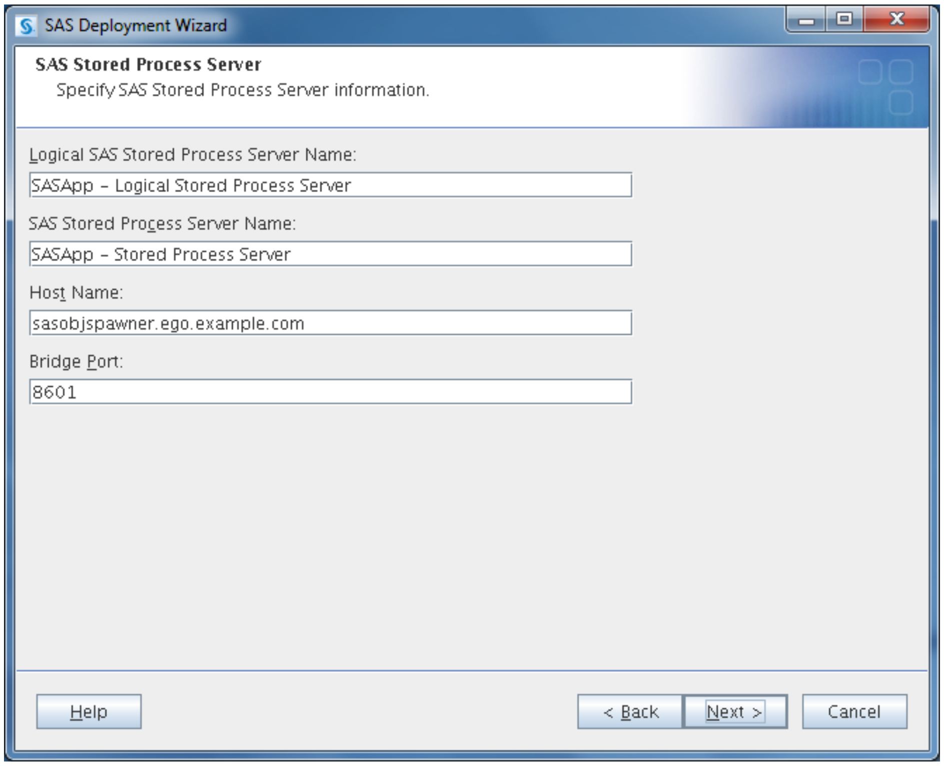
Logical SAS Pooled Workspace Server Name: SASApp - Logical Pooled Workspace Server

SAS Pooled Workspace Server Name: SASApp - Pooled Workspace Server

Host Name: sasobjspawner.ego.example.com

Port: 8701

The SAS Stored Process Server must have **Host Name** set the same as the Object Spawner.


SAS Deployment Wizard

SAS Stored Process Server
Specify SAS Stored Process Server information.

Logical SAS Stored Process Server Name: SASApp – Logical Stored Process Server

SAS Stored Process Server Name: SASApp – Stored Process Server

Host Name: sasobjspawner.ego.example.com

Bridge Port: 8601

## Modifying an Existing SAS Deployment

To Be Added

## SAS Service Startup Scripts

The scripts used to startup the SAS server processes will need to be adjusted to accommodate the HA services. There are several considerations to take into account when making adjustments to these scripts, the following sections discuss the necessary modifications and give details for making these changes.

To simplify management of the service dependencies it is highly recommended that all services running within the grid be managed by the EGO Service Controller. For services that will not be HA, meaning they can only execute on a single resource, allowing the EGO Service Controller to manage these services can still provide service resiliency even without resource redundancy. When defining non-HA services within the EGO Service Controller configuration select the single execution node for both the **Primary Host** and the **Failover Host**. The discussion that follows, along with the provided examples, assumes all SAS services are being managed by the EGO Service Controller.

### Helper Scripts

The examples included in this section make references to utility scripts which have been published to assist with integrating the suggested HA configuration changes; these scripts are available for download from:

**http://support.sas.com/rnd/scalability/grid/HA/gridha.html**

A general description and an example usage for the helper scripts are shown below.

**UNIX Scripts**

- **`ego_server.sh`** – This script will be used when interacting with the EGO Service Controller to start, stop, and status managed services.

```
/shared/config/ego_server.sh start sasobjspawner
```

- **`server_wrap.sh`** – This script could be used when creating the EGO Service Controller definition; wrapping the **`Execution Command [Start]`** to keep the service's start function from exiting. This may be necessary if the normal server control script start function (typically will fork daemon processes and exit) results in the EGO Service Controller declaring the service as failed.

```
/shared/config/server_wrap.sh /shared/config/Lev1/SASMeta/MetadataServer/MetadataServer.sh
```

**Windows Scripts**

To Be Added

### Defining SAS Services in EGO Service Controller Configuration

All SAS services will need to be defined in the EGO Service Controller configuration. To easily determine the full list of services to be added including the execution user that should be used to control the services and the execution commands that should be used to start and stop each service, refer to the `sas.servers` script in the SAS configuration level directory.

**UNIX Example**

The relevant excerpts from an example `sas.servers` script are shown below.

```
      …
SERVERUSER=sas

#*****
# Commands passed to the server control scripts
#*****
STARTCMD=start
STOPCMD=stop
```

```
      …
start_metadata_server()
{
      …
    $METADATA_SERVER_DIR/MetadataServer.sh $STARTCMD >/dev/null 2>&1
      …
}
      …
start_object_spawner()
{
      …
    $OBJECT_SPAWNER_DIR/ObjectSpawner.sh $STARTCMD >/dev/null 2>&1
      …
}
      …
stop_metadata_server()
{
      …
    $METADATA_SERVER_DIR/MetadataServer.sh $STOPCMD >/dev/null 2>&1
      …
}
      …
stop_object_spawner()
{
      …
    $OBJECT_SPAWNER_DIR/ObjectSpawner.sh $STOPCMD >/dev/null 2>&1
      …
}
      …
start_servers ()
{
      …
METADATA_SERVER_DIR=/shared/config/Lev1/SASMeta/MetadataServer
METADATA_SERVER_LOGS=/shared/config/Lev1/SASMeta/MetadataServer/Logs
METADATA_SERVER_CONTEXT=SASMeta

  if [ -x "$METADATA_SERVER_DIR/MetadataServer.sh" ];
  then
  {
    start_metadata_server;
#echo back from start meta 1
    if [ "$?" -eq 0 ];
      then
      {
        $Logmsg "SAS Metadata Server 1 is UP";
      }
      else
      {
        $Logmsg "SAS Metadata Server 1 is NOT up"
        $Logmsg "The remaining SAS servers will NOT be started as a result."
        return 1;
      }
    fi
  }
  else
  {
    $Logmsg "Could not access SAS Metadata Server 1 script"
    $Logmsg "SAS Metadata Server 1 is NOT up"
    $Logmsg "The remaining SAS servers will NOT be started as a result."
    return 1;
  }
  fi
      …
OBJECT_SPAWNER_DIR=/shared/config/Lev1/ObjectSpawner
OBJECT_SPAWNER_LOGS=/shared/config/Lev1/ObjectSpawner/Logs
OBJECT_SPAWNER_CONTEXT=NULL

  if [ -x "$OBJECT_SPAWNER_DIR/ObjectSpawner.sh" ]
  then
  {
    start_object_spawner;
    if [ "$?" -eq 0 ];
    then $Logmsg "SAS Object Spawner 1 is UP";
    else $Logmsg "SAS Object Spawner 1 is NOT up";
    fi;
  }
  else
  {
    $Logmsg "Could not access SAS Object Spawner 1 script"
    $Logmsg "SAS Object Spawner 1 is NOT up"
  }
  fi
      …
}

stop_servers ()
{
      …
OBJECT_SPAWNER_DIR=/shared/config/Lev1/ObjectSpawner
OBJECT_SPAWNER_LOGS=/shared/config/Lev1/ObjectSpawner/Logs

  if [ -x "$OBJECT_SPAWNER_DIR/ObjectSpawner.sh" ];
  then stop_object_spawner; fi;


METADATA_SERVER_DIR=/shared/config/Lev1/SASMeta/MetadataServer
METADATA_SERVER_LOGS=/shared/config/Lev1/SASMeta/MetadataServer/Logs

  if [ -x "$METADATA_SERVER_DIR/MetadataServer.sh" ];
  then stop_metadata_server; fi;


}
      …
```

### Windows Example

To Be Added

The function `start_servers` will contain a block of code for starting each service and there will be a separate `start_<server_name>` function that actually initializes the service. Likewise, the function `stop_servers` will contain a block of code for stopping each service and there will be a separate `stop_<server_name>` function to actually shutdown the service. Each service listed in the `sas.servers` script will need to be somehow added to the EGO Service Controller configuration. There are three possibilities for this integration:

- **Independent Service** – All service transitions (start, stop, and restart) can be performed without consideration for the current state of any other service.
- **Dependent Service** – One or more service transitions (start, stop, and restart) must consider the current state of another service.
- **Service Group** – The service is bundled with one or more other services that will be transitioned together as a group.

#### *Independent Service*

When a service is completely independent all that is required is to create an EGO Service Controller definition (refer to Defining HA Applications for instructions) using the values from the `sas.servers` script. Use the value of `SERVERUSER` for the **`Execution User`**, the command line that would be executed by the `start_<server_name>` function as the **`Execution Command [Start]`**, and the command line from the `stop_<server_name>` function as the **`Execution Command [Stop]`**.

### *Dependent Service*

The scripts for starting up the SAS servers must account for the possible dependencies between the various services; these can be generally classified in two categories:

**Startup Dependency** – The service cannot initialize until the dependency service is available.
**Restart Dependency** – The service not only requires the dependency service to be available at initialization but also requires restarting if the dependency service is restarted.

- **Startup Dependency**

When defining services within the EGO Service Controller configuration dependencies can be specified and these will be enforced when the services are started; meaning the services will always be initialized in dependency order. What the EGO Service Controller doesn't account for however, is the time required to fully initialize the required services. For example, when the SAS Metadata Server is configured as a dependency for Object Spawner, the EGO Service Controller will always start the Metadata Server before the Object Spawner, but if the Metadata Server takes time to initialize before it begins listening for connections, when the Object Spawner is started by the EGO Service Controller it might not be able to connect to the Metadata Server as expected. For this type of startup sequencing the EGO Service Controller dependencies do not provide an adequate solution and a different approach must be considered. NOTE: When timing constraints are not a consideration the startup dependencies can be handled by specifying all dependencies when creating the EGO Service Controller definition for the service.

To handle the sequencing of startup dependencies the recommended solution is to create a startup script for the top-level service which provides the dependency and have that script also start all the dependent services. In order for this to function correctly the EGO Service Controller definitions for each of the dependent services must be manually edited to change `<sc:StartType>` from `AUTOMATIC` to `MANUAL` as this is not a supported operation in Platform RTM for SAS (refer to Appendix A: EGO Service Controller Manual Config for more details). Changing each of the dependent services so that they are not automatically started by the EGO Service Controller will allow this top-level script to start these services only after their dependency is fully initialized. Once the services are started the EGO Service Controller will monitor their execution and restart them if necessary, providing the expected Service Resiliency. The EGO Service Controller definition for the top-level service will continue to have `<sc:StartType>` of `AUTOMATIC`, but it must be modified to change the **`Execution Command [Start]`** and **`Execution Command [Stop]`** to utilize this newly created script that will sequence the dependencies.

To create the top-level script to sequence the startup dependencies follow the steps below:

1. Start with a copy of the `sas.servers` script from the SAS configuration level directory.
2. Decide which services will need to be grouped; it's possible that not all services listed in the `sas.servers` script have the same dependency. For any service that isn't dependent, remove all references to that service from the new script. If multiple dependency groups are necessary create multiple copies of the `sas.servers` script and modify each copy as needed.
3. For the top-level service, leave the existing `start_<server_name>` and `stop_<server_name>` functions untouched along with the corresponding blocks within the `start_servers`, `stop_servers`, and `server_status` functions.
4. For each dependent service, make the following modifications:
   - o Locate the related block for the service within the `start_servers` function. Replace the entire block with something similar to the following:

```
"${@CMD_EGOSERVER@}" start @SC_APPLICATION_NAME@
```

Where `@CMD_EGOSERVER@` is the full path to the `ego_server.sh` helper script described above in Helper Scripts and **`@SC_APPLICATION_NAME@`** is the EGO Service Controller **Application Name** for the service to be started.

- Locate the related block for the service within the `stop_servers` function. Replace the entire block with something similar to the following:

```
"${@CMD_EGOSERVER@}" stop @SC_APPLICATION_NAME@
```

Where **`@CMD_EGOSERVER@`** is the full path to the `ego_server.sh` helper script described above in Helper Scripts and **`@SC_APPLICATION_NAME@`** is the EGO Service Controller **Application Name** for the service to be stopped.

- Locate the related block for the service within the `server_status` function. Replace the entire block with something similar to the following:

```
"${@CMD_EGOSERVER@}" status @SC_APPLICATION_NAME@
```

Where **`@CMD_EGOSERVER@`** is the full path to the `ego_server.sh` helper script described above in Helper Scripts and **`@SC_APPLICATION_NAME@`** is the EGO Service Controller **Application Name** for the service to be queried.

### UNIX Example

The relevant excerpts from an example modified `sas.servers` script are shown below which demonstrates how to sequence the startup of the SAS Metadata Server and the Object Spawner.

```
      …
start_servers ()
{
      …
METADATA_SERVER_DIR=/shared/config/Lev1/SASMeta/MetadataServer
METADATA_SERVER_LOGS=/shared/config/Lev1/SASMeta/MetadataServer/Logs
METADATA_SERVER_CONTEXT=SASMeta

  if [ -x "$METADATA_SERVER_DIR/MetadataServer.sh" ];
  then
  {
    start_metadata_server;
#echo back from start meta 1
    if [ "$?" -eq 0 ];
      then
      {
        $Logmsg "SAS Metadata Server 1 is UP";
      }
      else
      {
        $Logmsg "SAS Metadata Server 1 is NOT up"
        $Logmsg "The remaining SAS servers will NOT be started as a result."
        return 1;
      }
    fi
  }
  else
  {
    $Logmsg "Could not access SAS Metadata Server 1 script"
    $Logmsg "SAS Metadata Server 1 is NOT up"
    $Logmsg "The remaining SAS servers will NOT be started as a result."
    return 1;
  }
  fi
      …
# Start Object Spawner
/shared/config/ego_server.sh start sasobjspawner
# Start any additional dependent EGO services
      …
}

stop_servers ()
{
      …
# Stop Object Spawner
/shared/config/ego_server.sh stop sasobjspawner

# Stop any additional dependent EGO services

METADATA_SERVER_DIR=/shared/config/Lev1/SASMeta/MetadataServer
METADATA_SERVER_LOGS=/shared/config/Lev1/SASMeta/MetadataServer/Logs

  if [ -x "$METADATA_SERVER_DIR/MetadataServer.sh" ];
  then stop_metadata_server; fi;


}

server_status()
{

  $Logmsg "SAS servers status:"

METADATA_SERVER_DIR=/shared/config/Lev1/SASMeta/MetadataServer

    if [ -f $METADATA_SERVER_DIR/server.pid ];
    then
    {
      pid=`cat $METADATA_SERVER_DIR/server.pid`
      kill -0 $pid >/dev/null 2>&1
      if [ $? -eq 0 ]; then
         $Logmsg "SAS Metadata Server 1 (pid $pid) is running..."
      else
         $Logmsg "SAS Metadata Server 1 is stopped"
      fi
    }
    else
      $Logmsg "SAS Metadata Server 1 is stopped"
    fi

# Status Object Spawner
/shared/config/ego_server.sh status sasobjspawner

# Status any additional dependent EGO services
      …
}
      …
```

### Windows Example

To Be Added

- **Restart Dependency**

To handle restart dependencies a service would either need to detect when its required services are restarted and restart itself or the initialization of the top-level service must account for restarting each of its dependents. Since it can't be assumed a service would be able to detect the restart of a dependency, the recommended approach is the latter. To manage the restart dependencies use the same approach as for sequencing startup dependencies described in the previous section. It is important to note when and how the top-level service will be "restarted", as this can be easily confused with the "restart" functionality provided by the `sas.servers` script. The EGO Service Controller will use the service's **`Execution Command [Start]`** to initialize the service, including when "restarting" the service after it has failed on the current host and after a failover to an alternate host. Since the EGO Service Controller definition configures the **`Execution Command [Start]`** to execute the modified `sas.servers` script with "start" as the argument, the service is not being "restarted", from the script's perspective it will always execute "start" functionality. So, if any service has a restart dependency on another service the sequencing scripts must make certain the dependent service is stopped before the start of the top-level service can begin.

Using the previously modified copy of `sas.servers` as a starting point, make the following additional modifications:

1. Decide which services have the restart dependency; it's possible that some services have only a startup dependency while others actually have the restart requirement. It is important not to just assume that all services need to be restarted when any dependency is restarted; if the restart isn't required it introduces unnecessary additional service downtime, so don't do it.
2. Locate the block in the `start_servers` function that starts the top-level service; immediately before this block, for each service that must be restarted add a command to stop the dependent service; something similar to the following:

```
"${@CMD_EGOSERVER@}" stop @SC_APPLICATION_NAME@
```

Where **`@CMD_EGOSERVER@`** is the full path to the `ego_server.sh` helper script described above in Helper Scripts and **`@SC_APPLICATION_NAME@`** is the EGO Service Controller **Application Name** for the service to be stopped.

## UNIX Example

Excerpts from an example modified `sas.servers` script are shown below.

```
      …
start_servers ()
{
      …
# If Object Spawner is running, it must be stopped
/shared/config/ego_server.sh stop sasobjspawner

# Stop any additional dependent EGO services
      …
METADATA_SERVER_DIR=/shared/config/Lev1/SASMeta/MetadataServer
METADATA_SERVER_LOGS=/shared/config/Lev1/SASMeta/MetadataServer/Logs
METADATA_SERVER_CONTEXT=SASMeta

  if [ -x "$METADATA_SERVER_DIR/MetadataServer.sh" ];
  then
  {
    start_metadata_server;
#echo back from start meta 1
    if [ "$?" -eq 0 ];
      then
      {
        $Logmsg "SAS Metadata Server 1 is UP";
      }
      else
      {
        $Logmsg "SAS Metadata Server 1 is NOT up"
        $Logmsg "The remaining SAS servers will NOT be started as a result."
        return 1;
      }
    fi
  }
  else
  {
    $Logmsg "Could not access SAS Metadata Server 1 script"
    $Logmsg "SAS Metadata Server 1 is NOT up"
    $Logmsg "The remaining SAS servers will NOT be started as a result."
    return 1;
  }
  fi
      …
# Start Object Spawner
/shared/config/ego_server.sh start sasobjspawner

# Start any additional dependent EGO services
      …
}
```

#### Windows Example

To Be Added

### *Service Groups*

In certain deployment scenarios it may be required or desirable to keep a group or cluster of services always executing together on the same node, while High Availability is still needed for these services. This will be an exception to the previous recommendation of defining all SAS services individually within the EGO Service Controller configuration as the grouped services are not desired to be controlled independently. To support this scenario, the service grouping will be defined within the EGO Service Controller configuration as a single service and then would be controlled by a single group control script. This new script is similar to the sequencing script from the previous section, the difference is that none of the services being controlled are individually defined in the EGO Service Controller configuration; meaning all the services referenced within the new script will be started exactly as they would be in a normal SAS deployment rather than using EGO specific commands.

To create a service group, follow the steps below:

1. Start with a copy of the `sas.servers` script from the SAS configuration level directory.
2. For any service that will not be part of the service group remove all references to that service from the new script. If multiple service groups are necessary create multiple copies of the `sas.servers` script and modify each copy as needed. It is not necessary to modify any of the other logic within the script, simply minimize the list of services being controlled by this script to those services being grouped.
3. Create a new EGO Service Controller definition for the service group with the **`Execution Command [Start]`** and **`Execution Command [Stop]`** both configured to execute this new script.

## Verifying HA Service Configuration

### Testing Service Resource Redundancy

The following test scenarios can be used to verify that services are able to be executed on each of the redundant nodes. The Service Startup and Shutdown tests should be completed successfully before beginning the EGO HA configuration.

#### Service Startup and Shutdown

1. On one of the redundant nodes configured to execute the service use the command line specified in the EGO Service Controller definition as **`Execution Command [Start]`** to startup the service. If this is active-active redundancy, repeat this step for each of the redundant nodes.

2. Once service startup has been verified, use the command line specified in the EGO Service Controller definition as **`Execution Command [Stop]`** to shut down the service. If this is active-active redundancy, repeat this step for each of the redundant nodes.
3. Verify access to the service using a SAS client. For example, to verify Metadata Server the SAS Management Console could be used to connect.
4. If this is active-passive redundancy, repeat steps 1 and 2 for each of the redundant nodes.

## Testing Service Resiliency and Relocation

The following test scenarios can be used to verify that the EGO Service Controller will correctly detect service failure and restart services as required, and that when service execution moves to an alternate resource the service will be accessible by clients and dependent services as expected.

### *Service Restart*

- **Restart on Current Node**
  - Verify access to the service using a SAS client. For example, to verify Metadata Server the SAS Management Console could be used to connect.
  - Kill the service processes or use the command line specified in the EGO Service Controller definition as **`Execution Command [Stop]`** to shut down the service.
  - Verify the service is correctly restarted on the same node. If the service has any restart dependent services, verify that all the dependents are correctly restarted. If the service is part of a service group, verify that all the group services are available.
  - Re-verify access to the service using a SAS client.
- **Failover to Alternate Node**

NOTE: These scenarios do not apply to services using active-active redundancy.

  - Shutdown the primary node while it is currently executing the service.
  - Verify the service is correctly started on the standby node. If the service has any restart dependent services, verify that all the dependents are correctly restarted. If the service is part of a service group, verify that all the group services are available.
  - Verify the Service Relocation to the secondary node by accessing the service using a SAS client. For example, to verify the relocation of Metadata Server the SAS Management Console could be used to connect.
  - If there is a tertiary resource, shutdown the standby while it is currently executing the service (while leaving the primary node offline).
  - Verify the service is correctly started on the tertiary node. If the service has any restart dependent services, verify that all the dependents are correctly restarted. If the service is part of a service group, verify that all the group services are available.
  - Verify the Service Relocation to the tertiary node by accessing the service using a SAS client.
  - Bring the primary node back online and shutdown the service on the secondary or tertiary by killing the service processes or using the command line specified in the EGO Service Controller definition as **`Execution Command [Stop]`** to shut down the service.
  - Verify the service is correctly restarted on the primary node. If the service has any restart dependent services, verify that all the dependents are correctly restarted. If the service is part of a service group, verify that all the group services are available.
  - Verify the Service Relocation to the primary node by accessing the service using a SAS client.

### *Service Switchover/Migration*

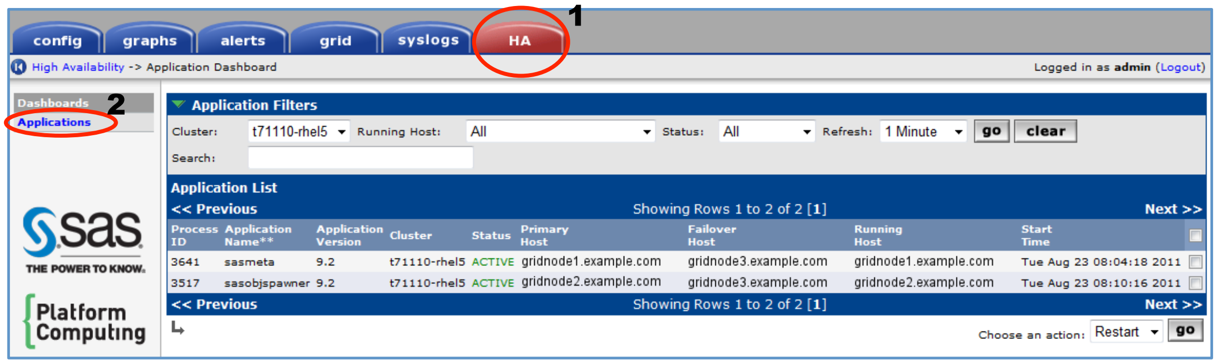
- **Using Platform RTM for SAS**

NOTE: These scenarios do not apply to services using active-active redundancy.

RTM includes capabilities to manually switchover, or migrate, HA services to a redundant node.

- To access the HA Application migration functionality, perform the following:

1) Select **HA** from the tab list at the top of the screen.
2) On the left-hand menu, under the **Dashboards** section select **Applications**.



Cluster: t71110-rhel5 Running Host: All Status: All Refresh: 1 Minute

Search:

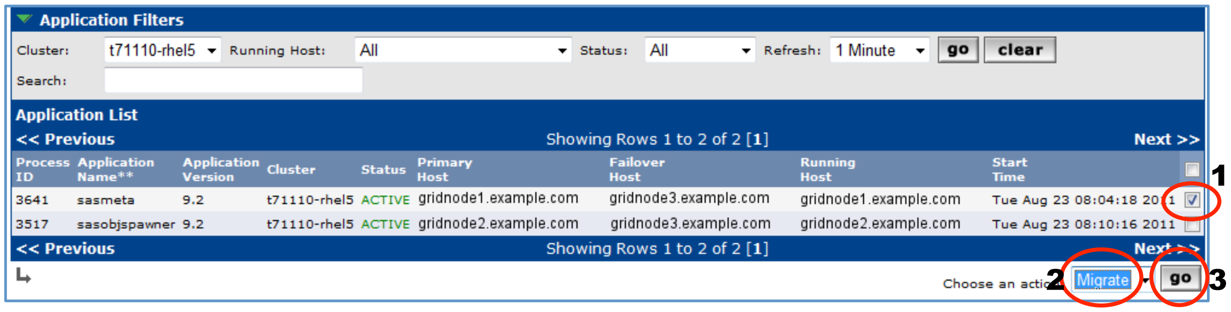
<< Previous Showing Rows 1 to 2 of 2 [1] Next >>

| Process ID | Application Name** | Application Version | Cluster | Status | Primary Host | Failover Host | Running Host | Start Time | ☐ |
|---|---|---|---|---|---|---|---|---|---|
| 3641 | sasmeta | 9.2 | t71110-rhel5 | ACTIVE | gridnode1.example.com | gridnode3.example.com | gridnode1.example.com | Tue Aug 23 08:04:18 2011 | ☐ |
| 3517 | sasobjspawner | 9.2 | t71110-rhel5 | ACTIVE | gridnode2.example.com | gridnode3.example.com | gridnode2.example.com | Tue Aug 23 08:10:16 2011 | ☐ |

<< Previous Showing Rows 1 to 2 of 2 [1] Next >>

- Verify the service **Status** is `ACTIVE` and that the **Running Host** is the primary node.
- To migrate the service to the secondary node, perform the following:

1) Check the action box in the far right column next to the service to be migrated.
2) From the **Choose an action** pull-down menu, select **Migrate**.
3) Click on the **go** button.



Cluster: t71110-rhel5 Running Host: All Status: All Refresh: 1 Minute

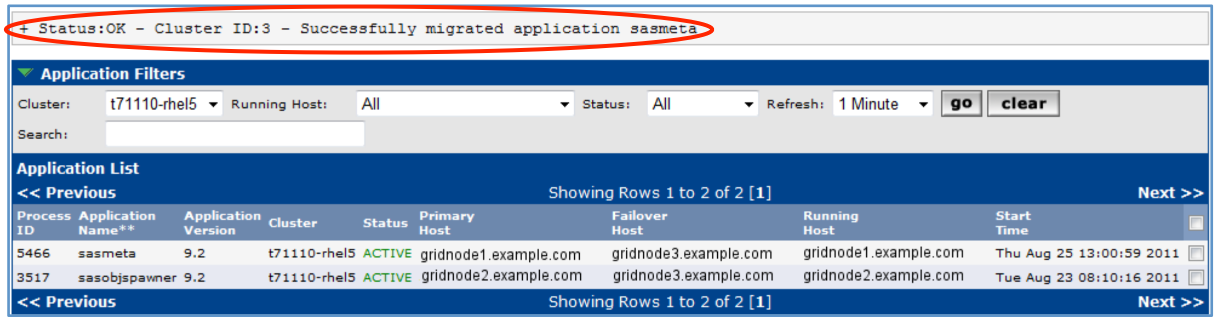
Search:

<< Previous Showing Rows 1 to 2 of 2 [1] Next >>

| Process ID | Application Name** | Application Version | Cluster | Status | Primary Host | Failover Host | Running Host | Start Time | ☐ |
|---|---|---|---|---|---|---|---|---|---|
| 3641 | sasmeta | 9.2 | t71110-rhel5 | ACTIVE | gridnode1.example.com | gridnode3.example.com | gridnode1.example.com | Tue Aug 23 08:04:18 2011 | ✓ |
| 3517 | sasobjspawner | 9.2 | t71110-rhel5 | ACTIVE | gridnode2.example.com | gridnode3.example.com | gridnode2.example.com | Tue Aug 23 08:10:16 2011 | ☐ |

<< Previous Showing Rows 1 to 2 of 2 [1] Next >>

- Enter the LSF Administrator password if necessary and answer **yes** when prompted for confirmation to begin the service migration.
- Once the migration has been submitted, the operation pane will be redisplayed with a Status message. If the migration initiated successfully, a message similar to the one shown below should be displayed.



```
+ Status:OK - Cluster ID:3 - Successfully migrated application sasmeta
```

Cluster: t71110-rhel5 Running Host: All Status: All Refresh: 1 Minute

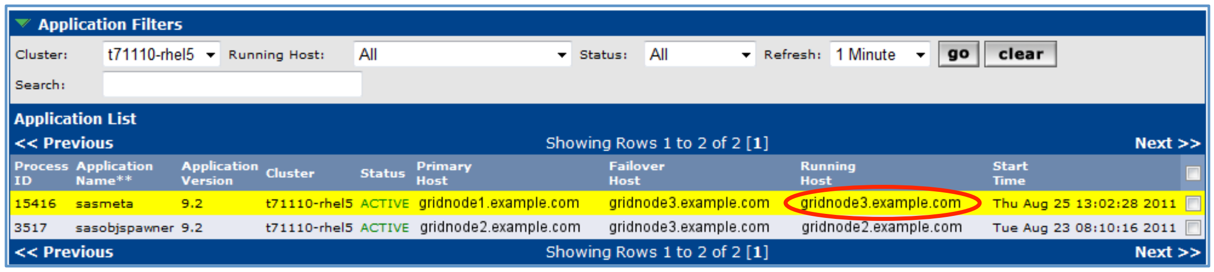
Search:

<< Previous Showing Rows 1 to 2 of 2 [1] Next >>

| Process ID | Application Name** | Application Version | Cluster | Status | Primary Host | Failover Host | Running Host | Start Time | ☐ |
|---|---|---|---|---|---|---|---|---|---|
| 5466 | sasmeta | 9.2 | t71110-rhel5 | ACTIVE | gridnode1.example.com | gridnode3.example.com | gridnode1.example.com | Thu Aug 25 13:00:59 2011 | ☐ |
| 3517 | sasobjspawner | 9.2 | t71110-rhel5 | ACTIVE | gridnode2.example.com | gridnode3.example.com | gridnode2.example.com | Tue Aug 23 08:10:16 2011 | ☐ |

<< Previous Showing Rows 1 to 2 of 2 [1] Next >>

- Once the service begins executing on the secondary node, the next operation pane refresh will update with the row for the particular service highlighted in yellow as shown below; this is normal and emphasizes the fact that this service is now executing on a resource other than its configured primary.



Application Filters

Cluster: t71110-rhel5 Running Host: All Status: All Refresh: 1 Minute go clear

Search:

Application List

<< Previous Showing Rows 1 to 2 of 2 [1] Next >>

| Process ID | Application Name** | Application Version | Cluster | Status | Primary Host | Failover Host | Running Host | Start Time |
|---|---|---|---|---|---|---|---|---|
| 15416 | sasmeta | 9.2 | t71110-rhel5 | ACTIVE | gridnode1.example.com | gridnode3.example.com | gridnode3.example.com | Thu Aug 25 13:02:28 2011 |
| 3517 | sasobjspawner | 9.2 | t71110-rhel5 | ACTIVE | gridnode2.example.com | gridnode3.example.com | gridnode2.example.com | Tue Aug 23 08:10:16 2011 |

<< Previous Showing Rows 1 to 2 of 2 [1] Next >>

- Verify the Service Relocation to the secondary node by accessing the service using a SAS client. For example, to verify the relocation of Metadata Server the SAS Management Console could be used to connect.
- NOTE: To return service execution to the primary node, repeat the service migration steps.

- **Using egosh command line**

NOTE: These scenarios do not apply to services using active-active redundancy.

To execute the commands shown below the user must have LSF administrator privilege.

- The `egosh` utility requires a separate login and to change service states or make any changes the login must be an administrative user. To logon using the default administrator account execute the following:

```
egosh user logon -u Admin -x Admin
```

- Verify that the service is currently executing on the primary node; use the `service list` command to verify the service is in the `STARTED` state and the `RESOURCE` is correctly listed as the primary node.

```
egosh service list –l
```

- Manually close the primary resource.

```
egosh resource close @PRIMARY_NODE@
```

Where `@PRIMARY_NODE@` is the hostname of the primary node.

- Stop and then restart the service. Make certain to pause after stopping the service to give sufficient time for service shutdown. Use the `service list` command to verify the service is in the `DEFINED` state before attempting to restart.

```
egosh service stop @SERVICE_NAME@
...PAUSE...
egosh service start @SERVICE_NAME@
```

Where `@SERVICE_NAME@` is the **`Application Name`** assigned to this service in the EGO Service Controller definition.

- The service should be restarted on the secondary node; use the `service list` command to verify the service is in the `STARTED` state and the `RESOURCE` is correctly listed as the secondary node.
- Verify the service is correctly started on the standby node. If the service has any restart dependent services, verify that all the dependents are correctly restarted. If the service is part of a service group, verify that all the group services are available.

- Verify the Service Relocation to the secondary node by accessing the service using a SAS client. For example, to verify the relocation of Metadata Server the SAS Management Console could be used to connect.
- If there is a tertiary node configured for the service, manually close the secondary resource, leaving the primary also closed.

```
egosh resource close @SECONDARY_NODE@
```

Where `@SECONDARY_NODE@` is the hostname of the secondary node.

- Stop and then restart the service again. Make certain to pause after stopping the service to give sufficient time for service shutdown. Use the `service list` command to verify the service is in the `DEFINED` state before attempting to restart.
- The service should be restarted on the tertiary node; use the `service list` command to verify the service is in the `STARTED` state and the `RESOURCE` is correctly listed as the tertiary node.
- Verify the service is correctly started on the tertiary node. If the service has any restart dependent services, verify that all the dependents are correctly restarted. If the service is part of a service group, verify that all the group services are available.
- Verify the Service Relocation to the tertiary node by accessing the service using a SAS client.
- Once the service startup has been verified, prepare to restart the service back on the primary node by first opening both the primary and secondary resources.

```
egosh resource open @PRIMARY_NODE@
egosh resource open @SECONDARY_NODE@
```

Where `@PRIMARY_NODE@` and `@SECONDARY_NODE@` are the hostnames of the primary and secondary nodes respectively.

- Stop and then restart the service again. Make certain to pause after stopping the service to give sufficient time for service shutdown. Use the `service list` command to verify the service is in the `DEFINED` state before attempting to restart.
- The service should be restarted on the primary node; use the `service list` command to verify the service is in the `STARTED` state and the `RESOURCE` is correctly listed as the primary node.
- Verify the service is correctly started on the primary node. If the service has any restart dependent services, verify that all the dependents are correctly restarted. If the service is part of a service group, verify that all the group services are available.
- Verify the Service Relocation to the primary node by accessing the service using a SAS client.

# Appendix A: EGO Service Controller Manual Config

NOTE: The examples below make reference to LSF and EGO specific environment variables; it is assumed the LSF environment setup has been sourced in the configuration environment.

## EGO Service Controller Definitions

To utilize the EGO Service Controller for service resiliency a definition must be created for each service to be managed. As previously discussed, Platform RTM for SAS is the easiest method of creating these definitions, but in some deployment scenarios RTM might not be available. RTM also doesn't support some advanced EGO Service Controller definition options that might be needed to meet specific deployment requirements; tertiary redundant server or additional environment variables are two examples. In these situations the service definitions can be manually edited and added to the EGO Service Controller configuration. The EGO Service Controller definitions are located in the following configuration directory:

```
${EGO_ESRVDIR}/esc/conf/services
```

There is a template EGO Service Controller definition file, `service.xml.TMPL`, located in the above directory, the contents are shown below:

```
<?xml version="1.0" encoding="UTF-8"?>
<sc:ServiceDefinition xmlns:sc="http://www.platform.com/ego/2005/05/schema/sc"
xmlns:ego="http://www.platform.com/ego/2005/05/schema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xsi:schemaLocation="http://www.platform.com/ego/2005/05/schema/sc ../sc.xsd
http://www.platform.com/ego/2005/05/schema ../ego.xsd" ServiceName="Dummy">
  <sc:Version>1.2</sc:Version>
  <sc:Description></sc:Description>
  <sc:MinInstances>1</sc:MinInstances>
  <sc:MaxInstances>5</sc:MaxInstances>
  <sc:Priority>10</sc:Priority>
  <sc:MaxInstancesPerSlot>1</sc:MaxInstancesPerSlot>
  <sc:MaxInstancesPerHost>1</sc:MaxInstancesPerHost>
  <sc:NeedCredential>false</sc:NeedCredential>
  <sc:ControlPolicy>
    <sc:StartType>MANUAL</sc:StartType>
    <sc:MaxRestarts>10</sc:MaxRestarts>
    <sc:HostFailoverInterval>PT60S</sc:HostFailoverInterval>
    <sc:Dependency type="OnStart">DB</sc:Dependency>
  </sc:ControlPolicy>
  <sc:AllocationSpecification>
    <ego:ConsumerID>/SampleApplications/EclipseSamples</ego:ConsumerID>
    <!-- The ResourceType specifies a "compute element" identified by the URI used below -
->
    <sc:ResourceSpecification
ResourceType="http://www.platform.com/ego/2005/05/schema/ce">
      <ego:ResourceGroupName>ComputeHosts</ego:ResourceGroupName>
      <ego:ResourceRequirement>LINUX86</ego:ResourceRequirement>
    </sc:ResourceSpecification>
  </sc:AllocationSpecification>
  <sc:ActivityDescription>
     <ego:Attribute name="hostType"
type="xsd:string">LINUX86</ego:Attribute>
     <ego:ActivitySpecification>
       <ego:Command>sleep 60</ego:Command>
       <ego:ExecutionUser>sampleOSuser</ego:ExecutionUser>
       <ego:WorkingDirectory>/tmp</ego:WorkingDirectory>
       <ego:Umask>0777</ego:Umask>
     </ego:ActivitySpecification>
  </sc:ActivityDescription>
  <sc:ActivityDescription>
     <ego:Attribute name="hostType"
type="xsd:string">NTX86</ego:Attribute>
     <ego:ActivitySpecification>
       <ego:Command>sleep 60</ego:Command>
       <ego:ExecutionUser>sampleOSuser</ego:ExecutionUser>
       <ego:WorkingDirectory>C:\</ego:WorkingDirectory>
       <ego:Umask>0777</ego:Umask>
     </ego:ActivitySpecification>
  </sc:ActivityDescription>
  <sc:ActivityDescription>
     <ego:Attribute name="hostType" type="xsd:string">all</ego:Attribute>
     <ego:ActivitySpecification>
       <ego:Command>sleep 60</ego:Command>
       <ego:ExecutionUser>sampleOSuser</ego:ExecutionUser>
       <!-- executable to control/shutdown service instance -->
       <ego:JobController>/path/cmd</ego:JobController>
       <!-- time in seconds to wait for job control to take effect,
service
           instance will be killed if it is already terminated after
wait period
       -->
       <ego:ControlWaitPeriod>PT10S</ego:ControlWaitPeriod>
       <ego:WorkingDirectory>/tmp</ego:WorkingDirectory>
       <ego:Umask>0777</ego:Umask>
     </ego:ActivitySpecification>
  </sc:ActivityDescription>
</sc:ServiceDefinition>
```

The template itself shows some of the advanced configuration possibilities that can't be accomplished through the RTM user interface. The important sections of the file are highlighted in bold, and these settings are further discussed below.

- **`ServiceName`** – Set this to a short alias for the service. If the EGO Service Director is used for Service Relocation this will be the host component of the virtual hostname for the service.
- **`Description`** – Set this to a short description for the service.
- **`MaxInstances`** – Set this 1 when using active-passive redundancy scheme or to the desired number of concurrently executing services when using active-active scheme.
- **`StartType`** – Change this to **`AUTOMATIC`** if this service is to be automatically started by the EGO Service Controller.
- **`Dependency`** – If there are other services defined to be managed by the EGO Service Controller that must be started before this service, include one **`Dependency`** clause for each of these services with the value equal to the **`ServiceName`** from the other service definition. If no dependencies are needed, delete or comment out the example line.
- **`ConsumerID`** – Change this value to **`/ManagementServices/LSFManagementServices`** which is the value required for RTM to recognize this definition.
- **`ResourceGroupName`** – Update this to **`ManagementHosts`** if the service is to be executed only on LSF master candidate nodes, or leave the setting of **`ComputeHosts`** if the service can execute on any grid compute node.
- **`ResourceRequirement`** – This setting can be used as written to allow the service to be executed on any grid node meeting specific **`hostType`**, but most often should be updated to list

the specific redundant nodes which are configured to execute the service. An example is shown below:

```
<ego:ResourceRequirement>
    select('@hostA@' || '@hostB@') order('@hostB@')
</ego:ResourceRequirement>
```

Where `@hostA@` is the FQDN of the primary execution host and `@hostB@` is the FQDN of the secondary. If a tertiary host is required, add `@hostC@` as follows:

```
select('@hostA@' || '@hostB@' || '@hostC@') order(1*'@hostA@' + 2*'@hostB@' + 3*'@hostC@')
```

If using an active-active redundancy scheme list all the nodes that will concurrently execute the service. For example, if the SAS Object Spawner is configured for load balancing across four nodes the select statement would look like the following:

```
select('@hostA@' || '@hostB@' || '@hostC@' || '@hostD@')
```

The order clause can be omitted in this case as all nodes will execute the service concurrently.

- **`ActivityDescription`** – The template includes a fairly complex example of how the service can be configured with different parameters for starting and stopping which depend on the **`hostType`** of the execution node. This could be required in heterogeneous grid environments, but in general the **`ActivityDescription`** from the template can often be simplified to something similar to the following:

```
  <sc:ActivityDescription>
    <ego:Attribute name="hostType" type="xsd:string">all</ego:Attribute>
    <ego:ActivitySpecification>
      <ego:Command>@START_COMMAND@</ego:Command>
      <ego:JobController>@STOP_COMMAND@</ego:JobController>
      <ego:ExecutionUser>@USER_ACCOUNT@</ego:ExecutionUser>
      <ego:EnvironmentVariable name="EGO_APP_TYPE">RTM_HA</ego:EnvironmentVariable>
      <ego:EnvironmentVariable name="EGO_SERVICE_VER">@APP_VER@</ego:EnvironmentVariable>
      <ego:EnvironmentVariable name="@VAR_NAME@">@VAR_VALUE@</ego:EnvironmentVariable>
      <ego:ControlWaitPeriod>PT30S</ego:ControlWaitPeriod>
      <ego:Umask>0022</ego:Umask>
    </ego:ActivitySpecification>
  </sc:ActivityDescription>
```

  - Replace `@START_COMMAND@` with the command line required to start the service.
  - Replace `@STOP_COMMAND@` with the command line required to stop the service.
  - Replace `@USER_ACCOUNT@` with the user name on the server that should be used for execution.
  - Add the RTM required environment variables **`EGO_APP_TYPE`** and **`EGO_SERVICE_VER`** replacing `@APP_VER@` with the software version for the service being defined.
  - If any specific environment variables are needed to execute **`Command`** or **`JobController`**, these can be defined as shown, one **`EnvironmentVariable`** clause per required setting. Replace `@VAR_NAME@` with the name of variable and `@VAR_VALUE@` with the associated value.

## Monitoring and Controlling HA Services

Once the services have been defined in the EGO Service Controller configuration, the command line utility `egosh` can be used to monitor and control the services. The `egosh` utility requires a separate login and to change service states or make any changes the login must be an administrative user. To login using the default administrator account execute the following:

```
# egosh user logon -u Admin -x Admin
```

Anytime a new service definition is created or modifications are made to an existing definition the EGO Service Controller must be reconfigured to pick up the changes. This is accomplished by restarting the LIM on the EGO master host using the command:

```
# egosh ego restart -f
```

NOTE: The restart command will start up any services that have **`StartType`** set to **`AUTOMATIC`** regardless of the state the service was in when the restart was executed.

The following session capture shows some useful commands for working with services.

```
# egosh service list
SERVICE  STATE    ALLOC CONSUMER RGROUP RESOURCE SLOTS SEQ_NO INST_STATE ACTI
sbatchd  DEFINED        /Cluster Intern
ServiceD DEFINED        /Managem Manage
sasobjsp DEFINED        /Managem Manage
res      DEFINED        /Cluster Intern
WebServi DEFINED        /Managem Manage
sasmeta  DEFINED        /Managem Manage

# egosh service start sasmeta
Starting service <sasmeta>. Run <service view> to ensure startup

# egosh service list
SERVICE  STATE    ALLOC CONSUMER RGROUP RESOURCE SLOTS SEQ_NO INST_STATE ACTI
sbatchd  DEFINED        /Cluster Intern
ServiceD DEFINED        /Managem Manage
sasobjsp DEFINED        /Managem Manage
res      DEFINED        /Cluster Intern
WebServi DEFINED        /Managem Manage
sasmeta  STARTED  230   /Managem Manage t71110-r 1     1      RUN        1189

# egosh service view sasmeta
-----------------------------------------------------------------
Basic Service INFO
Service Name       : sasmeta
Service Description: "SAS Metadata Server"

state       pri   maxInstance minInstance maxPerSlot  maxPerHost  credFlag
STARTED     10    1           1           1           1           0

The Allocation Requirement INFO
Name           : sasmeta
Consumer Name  : /ManagementServices/EGOManagementServices
Resource Group: ManagementHosts
Requirement    : select('gridnode1.example.com' || 'gridnode3.example.com')
order('gridnode3.example.com')

minSlots       maxSlots       EXCLUSIVE      TILE
0              1              No             1

The Associated Activity Specifications
all:
Command            : /shared/config/Lev1/SASMeta/MetadataServer/MetadataServer.sh start
Execution User     : sas
Umask              : 022
Working Dir        :
InstanceControl Cmd: /shared/config/Lev1/SASMeta/MetadataServer/MetadataServer.sh stop
ControlWaitPeriod  : 30

ENV<0>             : EGO_APP_TYPE=RTM_HA
ENV<1>             : EGO_SERVICE_VER=9.2
RLIMIT-CPU         : -
RLIMIT-FSIZE       : -
RLIMIT-DATA        : -
RLIMIT-STACK       : -
RLIMIT-CORE        : -
RLIMIT-RSS         : -
RLIMIT-NOFILE      : -
```

```
RLIMIT-AS           : -

The Control Policy INFO
StartType: AUTOMATIC
Service Dependency: NO DEPENDENCY
MaxRestarts     MaxHeartBeatInterval MaxMissedHeartBeats  HostFailoverInterval
10              -                    -                    45

The Allocation Situation
Allocation ID:232
RESOURCE                                         TYPE                SLOTS
gridnode3.example.com                        X86_64              1

Service Instances INFO
STATE      ACTIVITY_ID      RESOURCE                                 SEQ_NO
RUN        1191             gridnode3.example.com                1

# egosh service stop sasmeta
Service <sasmeta> has been stopped successfully

# egosh service list
SERVICE  STATE    ALLOC CONSUMER RGROUP RESOURCE SLOTS SEQ_NO INST_STATE ACTI
sbatchd  DEFINED        /Cluster Intern
ServiceD DEFINED        /Managem Manage
sasobjsp DEFINED        /Managem Manage
res      DEFINED        /Cluster Intern
WebServi DEFINED        /Managem Manage
sasmeta  DEFINED        /Managem Manage

# egosh service view sasmeta
-----------------------------------------------------------------
Basic Service INFO
Service Name       : sasmeta
Service Description: "SAS Metadata Server"

state       pri   maxInstance minInstance maxPerSlot  maxPerHost  credFlag
DEFINED     10    1           1           1           1           0

The Allocation Requirement INFO
Name           : sasmeta
Consumer Name : /ManagementServices/EGOManagementServices
Resource Group: ManagementHosts
Requirement    : select('gridnode1.example.com' || 'gridnode3.example.com')
order('gridnode3.example.com')

minSlots       maxSlots       EXCLUSIVE      TILE
0              1              No             1

The Associated Activity Specifications
all:
Command             : /shared/config/Lev1/SASMeta/MetadataServer/MetadataServer.sh start
Execution User      : sas
Umask               : 022
Working Dir         :
InstanceControl Cmd: /shared/config/Lev1/SASMeta/MetadataServer/MetadataServer.sh stop
ControlWaitPeriod   : 30

ENV<0>              : EGO_APP_TYPE=RTM_HA
ENV<1>              : EGO_SERVICE_VER=9.2
RLIMIT-CPU          : -
RLIMIT-FSIZE        : -
RLIMIT-DATA         : -
RLIMIT-STACK        : -
RLIMIT-CORE         : -
RLIMIT-RSS          : -
RLIMIT-NOFILE       : -
RLIMIT-AS           : -

The Control Policy INFO
StartType: AUTOMATIC
```

```
Service Dependency: NO DEPENDENCY
MaxRestarts     MaxHeartBeatInterval MaxMissedHeartBeats  HostFailoverInterval
10              -                    -                    45

The Allocation Situation
No Allocation

Service Instances INFO
 Number of service instances:<0>
```

# Appendix B: EGO Service Director Configuration

NOTE: The examples below make reference to LSF and EGO specific environment variables; it is assumed the LSF environment setup has been sourced in the configuration environment.

## EGO Service Controller Definition

Utilizing the EGO Service Director to provide HA Service Relocation requires that the EGO Service Director itself be HA. The required service definition is already present for the EGO Service Controller in the file:

```
${EGO_ESRVDIR}/esc/conf/services/named.xml
```

The following modifications are required to properly enable the EGO Service Director:

- Set the service to be automatically started: Change the following line:

```
<sc:StartType>MANUAL</sc:StartType>
```

To:

```
<sc:StartType>AUTOMATIC</sc:StartType>
```

- Update the `ResourceRequirement` definition to specify the grid nodes that will be used to execute the EGO Service Director. Change the following:

```
<ego:ResourceRequirement>select('LINUX86' ||'X86_64' ||
SOLX8664')</ego:ResourceRequirement>
```

To:

```
<ego:ResourceRequirement>
    select('@hostA@' || '@hostB@') order('@hostB@')
</ego:ResourceRequirement>
```

Where `@hostA@` is the FQDN of the primary execution host and `@hostB@` is the FQDN of the secondary. If a tertiary host is required, add `@hostC@` as follows:

```
select('@hostA@' || '@hostB@' || '@hostC@') order(1*'@hostA@' + 2*'@hostB@' + 3*'@hostC@')
```

- Double check that the value of ESD_CONF environment variable is set to the correct location. This might be correct as is, but a safe change is to replace the current setting with the following:

```
<ego:EnvironmentVariable name="ESD_CONF">${EGO_ESRVDIR}/esd/conf</ego:EnvironmentVariable>
```

The completed xml file should look similar to the following example:

```
<?xml version="1.0" encoding="UTF-8"?>
<sc:ServiceDefinition xmlns:sc="http://www.platform.com/ego/2005/05/schema/sc"
xmlns:ego="http://www.platform.com/ego/2005/05/schema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xsi:schemaLocation="http://www.platform.com/ego/2005/05/schema/sc ../sc.xsd
http://www.platform.com/ego/2005/05/schema ../ego.xsd" ServiceName="ServiceDirector">
  <sc:Version>1.2</sc:Version>
  <sc:Description>EGO: EGO Service Director</sc:Description>
  <sc:MinInstances>1</sc:MinInstances>
  <sc:MaxInstances>1</sc:MaxInstances>
  <sc:Priority>10</sc:Priority>
  <sc:MaxInstancesPerSlot>1</sc:MaxInstancesPerSlot>
  <sc:MaxInstancesPerHost>1</sc:MaxInstancesPerHost>
  <sc:NeedCredential>FALSE</sc:NeedCredential>
  <sc:ControlPolicy>
    <sc:StartType>AUTOMATIC</sc:StartType>
    <sc:MaxRestarts>1</sc:MaxRestarts>
    <sc:HostFailoverInterval>PT1M0S</sc:HostFailoverInterval>
  </sc:ControlPolicy>
  <sc:AllocationSpecification>
    <ego:ConsumerID>/ManagementServices/EGOManagementServices</ego:ConsumerID>
<!--The ResourceType specifies a "compute element" identified by the URI used below-->
    <sc:ResourceSpecification
ResourceType="http://www.platform.com/ego/2005/05/schema/ce">
      <ego:ResourceGroupName>ManagementHosts</ego:ResourceGroupName>
      <ego:ResourceRequirement>
         select('@hostA@' || '@hostB@') order('@hostB@')
       </ego:ResourceRequirement>
    </sc:ResourceSpecification>
  </sc:AllocationSpecification>
  <sc:ActivityDescription>
    <ego:Attribute name="hostType" type="xsd:string">all</ego:Attribute>
    <ego:ActivitySpecification>
      <ego:Command>${EGO_TOP}/7.0/scripts/egosrvloader.sh named -u lsfadmin -
f</ego:Command>
      <ego:ExecutionUser>root</ego:ExecutionUser>
       <ego:EnvironmentVariable name="ESD_CONF">
         ${EGO_ESRVDIR}/esd/conf
       </ego:EnvironmentVariable>
      <ego:Umask>0022</ego:Umask>
    </ego:ActivitySpecification>
  </sc:ActivityDescription>
</sc:ServiceDefinition>
```

## EGO Service Director DNS Server Configuration

To control the behavior of the EGO Service Director DNS server there are 3 important configuration files:

- **EGO Subzone Definition File**

```
${EGO_ESRVDIR}/esd/conf/named/namedb/db.@EGO_SUBDOMAIN@
```

- **DNS Server Configuration**

```
${EGO_ESRVDIR}/esd/conf/named/conf/named.conf
```

- **EGO Service Director Plugin Configuration**

```
${EGO_ESRVDIR}/esd/conf/esddefault.xml
```

For each of the possible EGO Service Director DNS setups, these three files must be properly configured to correctly match the corporate DNS server configuration. The following sections give example configurations which can be used as a starting point for configuring the EGO Service Director DNS server. All the examples below are based on the following environment assumptions:

- The corporate domain name is `example.com.`
- The ego subdomain is `ego.example.com.`
- The virtual hostname for the EGO Service Director DNS Server is `egonameserver.`
- The Time-To-Live for EGO virtual hostnames is set to 0, when possible.

## Without Corporate DNS Updates

- **EGO Subzone Definition File**

```
$ORIGIN .
$TTL 0  ; 0 seconds
ego.example.com   IN SOA  egonameserver. root. (
                                1715       ; serial
                                10800      ; refresh (3 hours)
                                900        ; retry (15 minutes)
                                604800     ; expire (1 week)
                                0          ; minimum (0 seconds)
                                )
                     NS      egonameserver.ego.example.com.
;
; Add A records for the sub zone as required
$ORIGIN ego.example.com.
sasmeta              A       192.168.1.100
sasobjspawner        A       192.168.1.101
```

TTL for the subdomain is set to 0 seconds in this example; this setting should be tuned as required.

The `A` records for the `sasmeta` and `sasobjspawner` are examples only and not required. Setting these aliases to the IP of the service's primary execution host will allow the virtual hostname for the service to resolve prior to the definition being added to the EGO Service Controller configuration; which may be required to complete the SAS deployment wizard.

- **DNS Server Configuration**

Remove or comment the lines in the global options section which enable query forwarding:

```
//        forwarders { @IPaddr@; @IPaddr@; };
//        forward first;
```

Update the name of the key that will be used to permit dynamic updates, change the default lines:

```
key ego. {
        algorithm HMAC-MD5.SIG-ALG.REG.INT;
        secret "@EGO_SECRET_VALUE@";
};
```

To:

```
key key.ego.example.com {
    algorithm HMAC-MD5.SIG-ALG.REG.INT;
    secret "@EGO_SECRET_VALUE@";
};
```

Update the subzone definition to be used for the EGO virtual hostnames, change the following:

```
zone "ego." IN {
        type master;
        file "db.ego";
        allow-update { key ego.; };
};
```

To:

```
zone "ego.example.com" IN {
        type master;
        file "db.ego.example.com";
        allow-update { key key.ego.example.com; };
        forwarders { };
};
```

- **EGO Service Director Plugin Configuration**

```
<?xml version="1.0" encoding="UTF-8"?>
<ESDDefaultPluginConfiguration>
  <ESD_EGO_NAMESERVER>egonameserver</ESD_EGO_NAMESERVER>
  <ESD_EGO_DOMAIN>ego.example.com</ESD_EGO_DOMAIN>
  <ESD_EGO_KEY name="key.ego.example.com">@EGO_SECRET_VALUE@</ESD_EGO_KEY>
</ESDDefaultPluginConfiguration>
```

The `ESD_EGO_KEY` will be used to sign dynamic update requests sent to the EGO Service Director DNS server. The name and value must correspond to the key previously defined in the DNS server configuration above.

- **Corporate DNS Configuration**

The corporate DNS server must be changed to delegate a subzone for the EGO virtual hostnames. Multiple Name Server entries should be added for this subzone, one for each redundant host that is configured to possibly execute the EGO Service Director based on the EGO Service Controller definition. The Name Server records will utilize the actual corporate hostnames for the EGO Service Director hosts.

## Corporate DNS Dynamic Updates for Location of EGO Service Director DNS Only

- **EGO Subzone Definition File**

```
$ORIGIN .
$TTL 0   ; 0 seconds
ego.example.com    IN SOA  egonameserver.example.com. root. (
                                1715       ; serial
                                10800      ; refresh (3 hours)
                                900        ; retry (15 minutes)
                                604800     ; expire (1 week)
                                0          ; minimum (0 seconds)
                                )
                        NS      egonameserver.example.com.
;
; Add A records for the sub zone as required
$ORIGIN ego.example.com.
sasmeta                A       192.168.1.100
sasobjspawner          A       192.168.1.101
```

TTL for the subdomain is set to 0 seconds in this example; this setting should be tuned as required.

The `A` records for the `sasmeta` and `sasobjspawner` are examples only and not required. Setting these aliases to the IP of the service's primary execution host will allow the virtual hostname for the service to resolve prior to the definition being added to the EGO Service Controller configuration; which may be required to complete the SAS deployment wizard.

- **DNS Server Configuration**

Remove or comment the lines in the global options section which enable query forwarding:

```
//        forwarders { @IPaddr@; @IPaddr@; };
//        forward first;
```

Update the name of the key that will be used to permit dynamic updates, change the default lines:

```
key ego. {
        algorithm HMAC-MD5.SIG-ALG.REG.INT;
        secret "@EGO_SECRET_VALUE@";
};
```

To:

```
key key.ego.example.com {
    algorithm HMAC-MD5.SIG-ALG.REG.INT;
    secret "@EGO_SECRET_VALUE@";
};
```

Update the subzone definition to be used for the EGO virtual hostnames, change the following:

```
zone "ego." IN {
        type master;
        file "db.ego";
        allow-update { key ego.; };
};
```

To:

```
zone "ego.example.com" IN {
        type master;
        file "db.ego.example.com";
        allow-update { key key.ego.example.com; };
        forwarders { };
};
```

- **EGO Service Director Plugin Configuration**

```
<?xml version="1.0" encoding="UTF-8"?>
<ESDDefaultPluginConfiguration>
  <ESD_EGO_NAMESERVER>egonameserver</ESD_EGO_NAMESERVER>
  <ESD_EGO_DOMAIN>ego</ESD_EGO_DOMAIN>
  <ESD_CORP_DOMAIN>example.com</ESD_CORP_DOMAIN>
  <ESD_EGO_KEY name="key.ego.example.com">@EGO_SECRET_VALUE@</ESD_EGO_KEY>
  <ESD_CORP_KEY name="key.example.com">@CORP_SECRET_VALUE@</ESD_CORP_KEY>
</ESDDefaultPluginConfiguration>
```

The `ESD_EGO_KEY` name and value must correspond to the key previously defined in the DNS server configuration above. The `ESD_CORP_KEY` would be used when updating the virtual hostname location for the EGO Service Director DNS Server, the name and value must be set to correspond with the corporate DNS configuration. Since the virtual hostname is within the corporate domain, the `ESD_CORP_KEY` must be permitted for updating the corporate DNS domain. This could possibly not be required, depending on the policy for dynamically updating the corporate DNS. When either key is not defined in this file the dynamic update requests will not be signed.

- **Corporate DNS Configuration**

The corporate DNS server must be changed to delegate a subzone for the EGO virtual hostnames. The Name Server for this subzone should be set to the virtual hostname of the EGO Service Director DNS Server, `egonameserver.example.com` in this example. In order for this configuration to work successfully this virtual hostname must statically resolve within the corporate DNS. This means that an initial `A` record needs to be present in the corporate DNS for this virtual hostname, often referred to as a 'glue' record. The IP for this record should be set to the address of the primary execution host for the EGO Service Director, but once the EGO Service Director DNS server is initialized a dynamic update with the correct location information will be sent to the corporate DNS server.

## Corporate DNS Dynamic Updates for Location of all EGO HA Services

- **EGO Subzone Definition File**

  Not used in this setup.

- **DNS Server Configuration**

No subzone will be defined for the EGO Service Director DNS server, so comment out or remove the default definitions:

```
//key ego. {
//        algorithm HMAC-MD5.SIG-ALG.REG.INT;
//        secret "@EGO_SECRET_VALUE@";
//};
```

And:

```
//zone "ego." IN {
//        type master;
//        file "db.ego";
//        allow-update { key ego.; };
//};
```

- **EGO Service Director Plugin Configuration**

```
<?xml version="1.0" encoding="UTF-8"?>
<ESDDefaultPluginConfiguration>
  <ESD_EGO_NAMESERVER>egonameserver</ESD_EGO_NAMESERVER>
  <ESD_EGO_DOMAIN>ego</ESD_EGO_DOMAIN>
  <ESD_CORP_DOMAIN>example.com</ESD_CORP_DOMAIN>
  <ESD_EGO_KEY name="key.example.com">@CORP_SECRET_VALUE@</ESD_EGO_KEY>
  <ESD_CORP_KEY name="key.example.com">@CORP_SECRET_VALUE@</ESD_CORP_KEY>
</ESDDefaultPluginConfiguration>
```

It is not a typographic error above; the value for both the `ESD_EGO_KEY` and `ESD_CORP_KEY` must be set to the secret keys corresponding to the corporate DNS configuration. The `ESD_CORP_KEY` will be used when updating the corporate domain, (virtual hostname of the EGO Service Director DNS server), and the `ESD_EGO_KEY` is used when updating the EGO subdomain, (EGO service virtual hostnames). Since defining the EGO subdomain in the corporate DNS configuration is optional in this setup, further discussed below, the `ESD_EGO_KEY` would have the same value as `ESD_CORP_KEY` if no EGO subdomain exists, else it must correspond to the secret for the EGO subdomain in the corporate DNS configuration. In this setup, both of these keys are optional depending on the corporate DNS dynamic update policies, but the settings must correctly align with the corporate DNS configuration.

- **Corporate DNS Configuration**

The corporate DNS server should be changed to add a subzone for the EGO virtual hostnames. The Name Server entries for this subzone would be the same as the corporate zone since the corporate DNS server would be serving as master for this new zone. The TTL for hosts in the new zone should be set to 0 or some marginally low value. In this setup, it isn't absolutely required to add the new EGO subzone to the corporate DNS server, provided that the TTL for EGO virtual hostnames can be set as required. Just as with the previous configuration, the virtual hostname for the EGO Service Director DNS server, `egonameserver.example.com` in this example, must statically resolve within the corporate DNS when the EGO Service Director initializes.

SAS INSTITUTE INC. WORLD HEADQUARTERS SAS CAMPUS DRIVE CARY, NC 27513

TEL: 919 677 8000 FAX: 919 677 4444 U.S. SALES: 800 727 0025 WWW.SAS.COM SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration. Other brand and product names are trademarks of their respective companies. Copyright © 2011, SAS Institute Inc. All rights reserved