http://www.sun.com/bigadmin/features/articles/smf_example.jsp
May 09, 2008

BigAdmin System Administration Portal

Feature Article

Using Service Management Facility (SMF) in the Solaris 10 OS: A Quick Example

Don Turnbull, April 2007

Introduction

The Service Management Facility is a new, unified model for services and service management that is included in the Solaris Operating System. SMF provides a deeper, more functional view into the processes managed during startup and shutdown of a Solaris system. In addition, processes managed through SMF can have dependencies and they are monitored to allow for restarts if a process fails or is improperly stopped.

SMF is a core part of the predictive self-healing technology available in the Solaris 10 OS, and it provides automatic recovery from software and hardware failures as well as administrative errors. In addition, SMF-managed services can be delegated to non-root users. Finally, SMF is a follow-on to the legacy method of starting and stopping services, though `/etc/rc` scripts will continue to run when present for backward compatibility.

Deployment of services through SMF provides a much more consistent and robust environment. First, users can query the Solaris OS with a simple command (`svcs -a`) to determine if a service is running, instead of attempting a connection and wondering if the connection will succeed. Additionally, critical services can be restarted automatically in the event of a problem, such as someone inadvertently killing a service, a bug causing a core dump, or other process failures occurring. Further, SMF provides detailed and common logging as well as robust error handling to prevent services from hanging after a system state change. Please see the man page for `smf`(5) for more information.

After a typical software installation, there can be a half dozen or more processes that need to be started and stopped during system startup and shutdown. In addition, these processes may depend on each other and may need to be monitored and restarted if they fail. For each process, these are the logical steps that need to be done to incorporate these as services in SMF:

- Create a service manifest file.
- Create a methods script file to define the start, stop, and restart methods for the service.
- Validate and import the service manifest using `svccfg`(1M).
- Enable or start the service using `svcadm`(1M).
- Verify the service is running using `svcs`(1).

Using SAS processes as an example, we will create two services, one for the SAS Metadata Server (OMR) and one for the SAS Object Spawner. In this example, the Object Spawner cannot attempt to start before the OMR is started and should be stopped before the OMR is stopped.

Configuring the OMR Service

Step 1

Create the manifest file according to the description in the `smf_method`(5) man page. For clarity, this file should be placed in a directory dedicated to files related to the application. In fact, the service will be organized into a logical folder inside SMF, so having a dedicated folder for the files related to the application makes sense. However, there is no specific directory name or location requirement enforced inside SMF.

In the example, the OMR service will be organized in SMF as part of the SAS application folder. This is a logical grouping; there is no physical folder named `sas` associated with SMF. However, when managing the service, the service will be referred to by `application/sas/metadata`. Other SAS-related processes can later be added and identified under `application/sas` as well. For the example, the file `/var/svc/manifest/application/sas/metadata.xml` should be created containing the following:

```
<?xml version="1.0"?>
<!DOCTYPE service_bundle
   SYSTEM "/usr/share/lib/xml/dtd/service_bundle.dtd.1">

<service_bundle type='manifest' name='SAS:Metadata'>
  <service
     name='application/sas/metadata'
     type='service'
     version='1'>
     <create_default_instance enabled='false' />
     <single_instance />

     <dependency
       name='multi-user-server'
       grouping='optional_all'
       type='service'
       restart_on='none'>
          <service_fmri value='svc:/milestone/multi-user-server' />
     </dependency>
     <exec_method
       type='method'
       name='start'
       exec='/lib/svc/method/sas/metadata %m'
       timeout_seconds='60'>
       <method_context>
         <method_credential user='sas' />
       </method_context>
     </exec_method>

     <exec_method
       type='method'
       name='restart'
       exec='/lib/svc/method/sas/metadata %m'
       timeout_seconds='60'>
       <method_context>
         <method_credential user='sas' />
       </method_context>
     </exec_method>

     <exec_method
       type='method'
       name='stop'
       exec='/lib/svc/method/sas/metadata %m'
```

```
      timeout_seconds='60' >
      <method_context>
        <method_credential user='sas' />
      </method_context>
    </exec_method>

    <property_group name='startd' type='framework'>
      <propval name='duration' type='astring' value='contract' />
    </property_group>

    <template>
      <common_name>
        <loctext xml:lang='C'>
          SAS Metadata Service
        </loctext>
      </common_name>
      <documentation>
        <doc_link name='sas_metadata_overview' iri=
'http://www.sas.com/technologies/bi/appdev/base/metadatasrv.html'
        />
        <doc_link name='sas_metadata_install' uri=
            'http://support.sas.com/rnd/eai/openmeta/v9/setup' />
      </documentation>
    </template>
  </service>
</service_bundle>
```

The manifest file basically consists of two tagged stanzas that have properties that define how the process should be started, stopped, and restarted and also define any dependencies. The first tag, *<service_bundle>* defines the name of the service bundle that will be used to group services and as part of the parameters in `svcs` commands (`svcs`, `svcmgr`, and so on). The interior tag, *<service>*, defines a specific process, its dependencies, and how to manipulate the process. Please see the man page for `service_bundle`(4) for more information on the format of manifest files.

Step 2

Create the methods scripts. This file is analogous to the traditional `rc` scripts used in previous versions of the Solaris OS. This file should be a script that successfully starts, stops, and restarts the process. This script must be executable for all the users who might manage the service, and it must be placed in the directory and file name referenced in the `exec` properties of the manifest file. For the example in this procedure, the correct file is `/lib/svc/method/sas/metadata`, based on the manifest file built in Step 1. See the man page for `smf_method`(5) for more information on method scripts.

```
#!/sbin/sh
# Start/stop client SAS MetaData service
#
.. /lib/svc/share/smf_include.sh
SASDIR=/d0/sas9-1205
SRVR=MSrvr
CFG=$SASDIR/SASMain/"$SRVR".sh

case "$1" in
'start')
        $CFG start
        sleep 2
        ;;
'restart')
        $CFG restart
```

```
        sleep 2
        ;;
'stop')
        $CFG stop
        ;;
*)
        echo "Usage: $0 { start | stop }"
        exit 1
        ;;
esac
exit $SMF_EXIT_OK
```

Step 3

Validate and import the manifest file into the Solaris service repository to create the service in SMF and make the service available for manipulation. The following commands shows the correct file name to use for the manifest in this example.

```
# svccfg
svc:> validate /var/svc/manifest/application/sas/metadata.xml
svc:> import /var/svc/manifest/application/sas/metadata.xml
svc:> quit
```

Step 4

Enable the service using the `svcadm` command. The `-t` switch allows you to test the service definition without making the definition persistent. You would exclude the `-t` switch if you wanted the definition to be a permanent change that persists between reboots.

```
# svcadm enable -t svc:/application/sas/metadata
```

Step 5

Verify that the service is online and verify that the processes really are running by using the `svcs` command.

```
# svcs -a | grep sas
online   8:44:37 svc:/application/sas/metadata:default

# ps -ef | grep sas
.....
sas 26791  1  0  08:44:36 ?  0:00 /bin/sh /d0/SASMain/MSrvr.sh
.....
```

Configuring the Object Spawner Service

Now, in the example, both the OMR process (above) and the Object Spawner process were to be configured. The Object Spawner is dependent on the OMR. The remainder of this document describes configuring the dependent Object Spawner process.

Step 1

The manifest file for the Object Spawner service is similar to the manifest file used for the OMR service. There are a few small changes and a different dependency. The differences are highlighted in bold in the following:

```
<?xml version="1.0">
```

```
<!DOCTYPE service_bundle
   SYSTEM "/usr/share/lib/xml/dtd/service_bundle.dtd.1">

<service_bundle type='manifest' name='SAS:ObjectSpawner'>
  <service
    name='application/sas/objectspawner'
    type='service'
    version='1'>
    <create_default_instance enabled='false' />
    <single_instance />
    <dependency
       name='sas-metadata-server'
       grouping='optional_all'
       type='service'
       restart_on='none'>
          <service_fmri value='svc:/application/sas/metadata' />
    </dependency>
    <exec_method
      type='method'
      name='start'
      exec='/lib/svc/method/sas/objectspawner %m'
      timeout_seconds='60'>
      <method_context>
        <method_credential user='sas' />
      </method_context>
    </exec_method>

    <exec_method
      type='method'
      name='restart'
      exec='/lib/svc/method/sas/objectspawner %m'
      timeout_seconds='60'>
      <method_context>
        <method_credential user='sas' />
      </method_context>
    </exec_method>

    <exec_method
      type='method'
      name='stop'
      exec='/lib/svc/method/sas/ objectspawner %m'
      timeout_seconds='60' >
      <method_context>
        <method_credential user='sas' />
      <method_context>
    <exec_method>

    <property_group name='startd' type='framework'>
      <propval name='duration' type='astring' value='contract' />
    </property_group>

    <template>
      <common_name>
        <loctext xml:lang='C'>
          SAS Object Spawner Service
        </loctext>
      </common_name>
      <documentation>
        <doc_link name='sas_metadata_overview' iri=
'http://www.sas.com/technologies/bi/appdev/base/metadatasrv.html'
        />
        <doc_link name='sas_metadata_install' uri=
            'http://support.sas.com/rnd/eai/openmeta/v9/setup' />
      </documentation>
```

```
    </template>
  </service>
</service_bundle>
```

Step 2

After creating the manifest file, create the script `/lib/svc/method/sas/objectspawner`:

```
#!/sbin/sh
# Start/stop client SAS Object Spawner service
#
.. /lib/svc/share/smf_include.sh
SASDIR=/d0/sas9-1205
SRVR=ObjSpa
CFG=$SASDIR/SASMain/"$SRVR".sh

case "$1" in
'start')
        $CFG start
        sleep 2
        ;;
'restart')
        $CFG restart
        sleep 2
        ;;
'stop')
        $CFG stop
        ;;
*)
        echo "Usage: $0 { start | stop }"
        exit 1
        ;;
esac
exit $SMF_EXIT_OK
```

Step 3

Validate and import the manifest file in the same manner as was used for the OMR service:

```
# svccfg
svc:> validate /var/svc/manifest/application/sas/objectspawner.xml
svc:> import /var/svc/manifest/application/sas/objectspawner.xml
svc:> quit
```

Step 4

Enable the new service in the same manner as was used for the OMR service:

```
# svcadm enable -t svc:/application/sas/objectspawner
```

Step 5

Finally, verify that the service is up and running in the same manner as was used for the OMR service:

```
# svcs -a | grep sas
online   10:28:39 svc:/application/sas/metadata:default
online   10:38:20 svc:/application/sas/objectspawner:default
```

```
# ps -ef | grep sas
.....
sas 26791  1  0  18:44:36 ?  0:00 /bin/sh /d0/SASMain/MSrvr.sh
sas 26914  1  0  18:18:49 ?  0:00 /bin/sh /d0/SASMain/ObjSpa.sh
.....
```

In a real situation, there will be other processes owned by SAS and that have the service process as parent process. However, the fact that the script defined as the method script to the process is running is proof that the output from `svcs -a` is correct; the service is running.

Demonstrating SMF Functionality

To show that the services will automatically restart themselves as configured, kill off the current Metadata server (PID 26791).

```
# kill 26791
```

The `ps`(1) command shows that the previous Metadata PID of 26791 no longer exists and a new PID, 27035, is now running. All restarted without user intervention.

```
# ps -ef | grep sas
.....
sas 27035  1  0  18:44:36 ?  0:00 /bin/sh /d0/SASMain/MSrvr.sh
sas 26914  1  0  18:18:49 ?  0:00 /bin/sh /d0/SASMain/ObjSpa.sh
.....
```

Given that the Metadata Server is a critical component of the SAS 9 deployment, SMF functionality greatly adds to the availability of the SAS application environment. Most applications have similar situations. Through SMF, an application's processes can be automated, monitored, and managed with less overhead and less administrator customization of the Solaris OS.