SANE 2006 – Solaris SMF

15/05/06 11:03

Liane = tech-lead voor SMF, werkte vroeger in het kernel team. Her presentation can be downloaded from her blog as soon as possible.

Agenda

- SMF basics
- SMF components and implementation
- Solaris Integration Points
- SMF service development
- Troubleshooting and recovery
- SMF future

Motivation behind SMF

- There's a big difference between a service and a normal program.
- OSes provide little support for service-based management.
  - What services are available on the system?
  - Are any broken? How are they broken?
  - What does this service depend on?
- Service availability is paramount. The OS doesn't matter that much when it comes to availability calculations.
- Lack of knowledge of service boundary and inter-service relationships -> limits error handling ability of system
- Ticking time-bombs inherent in file-based configuration. No easy roll-back, no change management, no immediate checking of changes.

Service management is divided into three parts:

- Service delivery -> description of what you deliver
- Administrative interaction -> how to stop/start, etc
- Management of service by system -> what will the system do in various situations?

Solaris 10 SMF solution

- All services are elevated to first-class system objects
- Consistent configuration handling
  - generic meta-config common (enabled, dependencies, methods, etc)
  - configuration explicitly commited -> no time bombs

    - administrative intent preserved -> config changes made to the system are preserved and protected from being overwritten. Think: modified init scripts that are killed by a patch install.
    - restore to known-good snapshots -> roll-back / change management.
- Securely delegate administration to non-root users through RBAC.
- Services automatically restarted in dependency order:
    - administrative error, hardware error, software fault. So if service A core dumps and B depends on A, they're both restarted.
    - parallel startup comes along for the ride.
- Easy access to information about faulted services.

Predictive self healing

- Solaris Fault Manager provides detection and diagnosis of hardware errors, leading to isolation and deactivation of faulty components and precies, accurate messaging.
- SMF makes Solaris services self-healing. Hardware faults which previously cause system restarts are now isolated to the effected services. Services are also automatically restarted in the face of hardware and software faults.
- Bottom line: you will be told –what- is failing and –what- is affected, instead of having to figure it out by going over multiple log messages.
- Software fault handling is much more complex than hardware fault handling. This makes SMF a tricky piece of software to build.

A new system daemon, svc.startd, has taken over most of init's responsibilities.

- init still uses inittab and /etc/rcX.d
- svc.startd can automatically restart services
    - If daemon X is enabled it is started at boot time and restarted if it dies.
    - If daemon X is disabled it is never started. Not at boot time and not after installing a patch or upgrade or whatever.

Each service defined for SMF has a defined state. For a list of known service states, refer to page 1 of the handout.

Service dependencies

- svc.startd starts service in dependency order and also allows for parallel startup of non-related services. -> faster boot.
- If a service encounters a hardware error, only the service and services which depend on it are restarted (not the whole system).

Service method

- svc.startd uses a service's methods to manipulate it.
  - Start & stop methods
  - Refresh method
- Methods can be scripts, binaries or keywords.
- Methods can specify context (user, group, privileges (think user-init) and resource management settings).
- Nice :) You can tell SMF to run daemon X as non-privileged user Y, but to connect to privileged port Z. Nifty.

SMF configuration

- Service meta-config is kept in the repository (enabled/disabled, state, dependencies, methods, etc).
- The repository is controlled by svc.configd.
- The repository is currently stored in /etc/svc/repository.db (a database (currently SQLLight), not a flat file).

Service instances

- services are presented as instance nodes which are children of service nodes.
- Both service nodes and instance nodes can have properties
- If an instance doesn't have property X, the service's X is used.
- allows similar services to share properties (e.g. same webserver, different ports).

Service names

- Fault Management Resource Identifier (FMRI)

- URI syntax: svc:/system/cron:default
  - system/cron = service name
  - default = instance name
- The namespace is flat. The slashes do not refer to directories or whatever.
- Commands accept abbreviations (system/cron, cron)

Property structure

- A property has a type and zero or more values.
- Properties are grouped in named groups which have a tag (framework, dependency, etc)
- Example: “general/enabled” = group/property

Property snapshots

- Snapshot of instance = copy of properties + properties of service node
- Snapshots are automatically taken (successful start, before upgrade)
- Snapshots can be used to roll-back to previous situations
- SMF components usually use the “running” snapshot so changes can be commited by updating that snapshot.

Repository use

- Services are allowed to store their config in the repository
  - avoids writing a new parser
  - benefits from snapshots
  - sue libscf and svcprop
  - Example: “rpc/bin” has “config/verbose_logging” and “config/local_only” (this is a cool feature by the way!)
- All users may read properties.
- Write privileges may be delegated via RBAC (see S5:smf_security)

Service manifests

- XML file which describes dependencies, methods, service-specific properties.
- delivered into /var/svc/manifest

- manifests are loaded into the repository at boot-time when there are new/changed manifests.
- manifests in /var/svc/manifest should NOT be customized. These changes should be made in the repository using svccfg.

Commands

- svcs -> lists state, state-time, FMRI, dependencies of (enabled) services.
- svcadm -> manipulates services. Runs asynchronous so it returns immediately, even if its tasks are still working in the background (can be circumvented with "-s"). Changes can also be made to persist until the next reboot (temporarily).
- svccfg -> import/export manifests, apply/extract profiles. Also has an interactive mode. Also able to dump (archive) a readable XML file with the complete config.
- svcprop -> properties of services and instances for listing, fetching (for scripts) and stuff like that.

Delegated restarters

- Not all services fit svc.startd's service model.
- SMF allows a service to be a delegated restarter for other services like stop/start/restart and setting states.
- svc.startd still handles enable/disable.
- Currently inetd is the only delegated restarter. Configuration is still stored in the repository and not in inetd.conf.
- More delegated restarters will come soon.

Inetd is not used anymore. Everything that was in there has been converted to SMF. When unconverted services appear in inetd.conf, inetd gives off an alarm. The inetconv command can be used to convert services to SMF.

Currently there is no way of quickly showing all temporary changes affecting the system right now. They're working on implementing this :)

<pauze> Vanaf nu type ik minder. Ik lees de sheets nog eens door

Milestone = SMF model voor runlevels = a collection of dependencies which declare a specific state of system-readiness. None = kale kernel + root prompt. All = meekijken hoe alle services worden opgestart, vanaf "none".

Legacy services are not monitored by SMF, while they are started by it at boot time. At a later point in time someone may stop a legacy service, but the state will still be "legacy_run".

At boot time all messages from starting services are logged into files. They are –not- displayed at the console screen, thus cleaning up your boot messages.

Contract = mechanism to express relationships between a process and the kernel-managed resources it depends upon. They are visible via the /system/contract filesystem. Contracts can actually take various actions based upon events happening within the group of processes tied to it.

Any long-running process can be fitted to SMF. First, create a manifest and then modify your rc scripts.

Benefits:

- visible and manageable using standard solaris tools.
- new tools will automatically see your service
- built-in restart due to administrative error, software or hardware fault.
- dependencies reduce complex install-time logic and eliminate start-time checks
- using SMF simplifies participation in Solaris innovations: FMA, resource management delegated administration.

Common integration

- create basic manifest
- convert init script to methods
- minimal testing
- split monolithic services into separate restartable components. (advanced)
- customized error handling. (advanced)

Manifest creation:

- name your service
- identify whether your service may have multiple instances
- identify your service model (contract, transient, wait)
- identify how your service is stopped and started
- determine faults to be ignored
- identify dependencies
- identify dependants
- create (if appropriate) a default instance
- create template info to describe your service
- /usr/bin/xmllint can be used to debug your XML code...

Model contract = expects at least one process in the service when the start method exits

Model transient = services usually don't have processes in the servcie when the start method exits

When reverting to an older snapshot SMF cannot show you the differences easily. You can write a script to do so and Liane actually noted this feature request down :)

One of the future features in SMF is "monitors" which can actually tell you if your applications/service is functioning properly. Not by seeing which processes are running, but by checking its real functionality.