Solaris 11 Customer Maintenance Lifecycle

Gerry Haskins
Director, Software Lifecycle Engineering
Solaris Systems
Oracle
Dublin, Ireland

Keywords: Solaris 11 IPS maintenance strategy proactive reactive policy best practice

Disclaimer

Introduction

With the release of Solaris 11, now is a good time to discuss how Solaris 11 will be maintained by customers in their production environment.

Solaris 11 is the first mainstream product release designed for use in enterprise production environments to utilize the Image Packaging System (IPS) package architecture.

IPS has previously been showcased in Solaris 11 Express and OpenSolaris.

The requirements of enterprise production customers are significantly more rigorous than those for a proof of concept or early access release.

This presentation deals with how IPS, and the maintenance policies around Solaris 11, are designed to meet the requirement of enterprise production customers.

In particular, this presentation:

- Provide some background information on IPS features and Solaris maintenance best practices
- Highlights some of the key advantages of IPS over the old SVR4-based package and patch architecture
- Compares and contrasts the customer maintenance lifecycle experience between Solaris 11 and Solaris 10
- Highlights the associated maintenance policy changes for Solaris 11 designed to provide a better customer experience

Image Packaging System (IPS)

Image Packaging System (IPS) is a single tier packaging architecture designed to replace Sun's old System V Release 4 (SVR4) based two tier packaging and patch architecture.

With IPS there are no more patches. All deliverables, including bug fixes, are made at the package level.

IPS packages are smart and fat.

Fat packages contain everything - SPARC and x86 deliverables, man pages, etc.

IPS is smart because only that portion of the package which is relevant to the target system is downloaded. For example, if the target system is SPARC 4v, then only the necessary objects for SPARC 4v are downloaded. And only objects which deliver a change delta compared to the installed version are downloaded.

IPS advantages over the SVR4-based architecture

Two of the key IPS and Solaris 11 architects, Bart Smaalders and David Comay, spent a lot of time working with me around the Solaris 10 11/06 (Update 3) time frame to understand the deficiencies and limitations of the SVR4 based two tier packaging and patch architecture.

While the SVR4 based architecture had served Sun well, it struggled to cope with patching Solaris 10 due to the introduction of large features in Solaris 10 Update releases such as ZFS, Trusted Extensions, Secure By Default, OPL platform support, etc.

By the Solaris 10 11/06 (Update 3) time frame, it was proving extremely difficult to create Solaris 10 Kernel patches of release quality. Many patch revisions failed pre-release testing, primarily because the patching architecture couldn't cope with applying arbitrary change to a live boot environment on systems running Zones.

I wrote a detailed analysis of the root causes and Bart and David spent time to really understand the issues and help me to come up with solutions.

The short term solution was the introduction of Deferred Activation Patching, which uses Loopback Filesystem (lofs) mounts to keep the live boot environment in a consistent state during patching. This is achieved by mounting the old objects over the newly patched objects, so that if the object is invoked, it's guaranteed to be consistent with what's running in memory. When the system is rebooted, the lofs mounts are torn down, exposing the patched functionality. See http://blogs.oracle.com/patch/entry/solaris_10_kernel_patchid_progression.

The long term solution is Image Packaging System (IPS). Bart and David architected some of the deficiencies of the SVR4-based process out of IPS.

For example, SVR4-based patches can contain free format pre- and post-patch scripts. Being free format, they can potentially contain anything - e.g. testing whether an install precondition is met, stopping and restarting a daemon, etc. While the engineers writing them usually do a good job of ensuring they work when installed to a live boot environment, they may not neglect to do all the due diligence required to ensure the scripts work in more complex scenarios, such as installation to an alternate boot environment, the miniroot, a diskless client, a non-global zone, etc. And being free format also makes them hard to test, since each is unique and hence it's hard to automate testing as it's hard to automatically know what the nominal result should be.

IPS replaces such free format scripts with a predefined set of permitted actions. These actions are coded to work in all supported install scenarios. This eliminates a whole class of error opportunity.

Even more importantly, Solaris developers on Solaris 10 and below tend to use a quick and easy method called "BFU" (Blindingly Fast Updates) to stream their changes onto a system for unit and link testing. This completely bypasses the SVR4-based packaging and patch architecture. Hence developers are not necessarily aware of the limitations imposed by the SVR4-based architecture.

For example, moving an object from one package to another sounds like it should be a trivial exercise. But it's extremely difficult to do in the SVR4-based patch architecture. It's possible to do it if the object is moving from a package whose name is alphabetically before the package name it's moving too, e.g. from pkg A to pkg B. But it's extremely difficult indeed to move an object in the other direction, e.g. from a pkg B to pkg A.

Because developers aren't exposed to these limitations when they use BFUs, they sometimes code changes in a way which is very difficult for the SVR4-based patch architecture to deliver. The resulting complexity can sometimes cause problems.

In IPS, all Solaris developers use exactly the same IPS packaging architecture when unit and link testing code changes, as customers will do when installing them once they are released. This ensures that any IPS related limitations can be understood and dealt with prior to release.

This may sound like a minor change, but it's not. One of the process improvements implemented during Solaris 10 was to ensure non-bug-fix code putbacks to the Solaris 10 Sustaining Gate, such as critical new hardware support, didn't destabilize the Sustaining Gate, thereby compromising our ability to deliver bug fixes for customer escalated issues. Such putbacks had been a recurring source of error. We implemented this by insisting the developers create an IDR (Interim Diagnostic or Relief, which utilize the SVR4-based patch architecture) for independent testing prior to putting back the code to the Sustaining Gate. Surprisingly, we found very few issues indeed during independent testing. The reason being that the very act of the developers creating deliverables using the SVR4-based patch architecture meant that they saw and fixed architecture related deficiencies with their deliverables prior to putting their changes back to the code base. Hence, we're using the same concept for all IPS deliverables from all Solaris developers.

There are many other improvements in IPS and Solaris 11.

For example, Solaris 11 utilizes a ZFS Root filesystem in order to leverage the compelling advantages of ZFS snapshots. The 'beadm' utility is used to manage these boot environment snapshots. Together, they behave like an enhanced and well integrated Live Upgrade with much reduced disk and processing overhead.

Solaris 11 Deliverables: Updates, SRUs, CPUs, and IDRs

Solaris 11 Updates may be released periodically to provide new software features, enhancements to existing features, platform support, etc.

Support Repository Updates (SRUs) delivering bug fixes will be released on a monthly basis.

Every third SRU will be renamed a Critical Patch Update (CPU) in line with standard Oracle policy. The National Vulnerability Database is updated about a week before CPUs are released. Oracle releases CPUs for many products in the Oracle stack, currently on the Tuesday closest to the 15th of January, April, July, and October. This predictable cadence enables customers to pre-plan maintenance windows to apply the latest security fixes.

The SRU/CPU process can be seen already in Solaris 11 Express.

IDRs may be created for specific customers to diagnose or provide relief for specific issues.

The IPS IDR mechanism leverages IPS functionality so that an IDR can be auto-superseded (auto-obsoleted) when a final fix for all the issues it addresses is released. This simplifies System Administration on Solaris 11.

IPS Delivery Options

By default, IPS downloads are from an internet based repository.

A Release Repository enables test and evaluation of the initial Solaris 11 release and Solaris 11 Updates without a support contract.

A separate Support Repository, accessible by customers with an appropriate support contract, is a superset of the contents of the Release Repository and contains the original Solaris 11 release, Updates, SRUs, and CPUs.

Customers can set up a local repository to mirror the official Oracle repository for systems which cannot be connected directly to the internet.

ISO images will also be available for download from My Oracle Support.

Customer Maintenance Lifecycle

Many enterprise customers follow a similar maintenance lifecycle.

The two key types of maintenance are proactive maintenance (prevention) and reactive maintenance (break/fix).

Proactive maintenance can be further broken down into:

- Major proactive maintenance windows where major changes to the hardware, OS, and software stacks are deployed. This is often related to the deployment of new hardware, which may require a later version of the Operating System, or it may be related to the upgrade of other key components of the software stack such as the Oracle database and/or ISV applications.

- Minor proactive maintenance windows which are typically used to deploy bug fixes in a preventative manner - i.e. the prevention is better than cure philosophy. This is because planned maintenance downtime is usually far cheaper and less disruptive to a business than unplanned downtime. In Solaris 10, this might take the form of applying the Solaris OS Recommended Patchset or CPU. In Solaris 11, the equivalent would be applying an SRU or CPU. Firmware should also be updated as needed in both cases.

The key characteristic of proactive maintenance is that there is sufficient time to allow for pre-deployment testing prior to roll-out across the enterprise.

Reactive maintenance typically consists of either:

- Application of fixes to security vulnerabilities. In some industries, industry regulation or internal business risk practices require that fixes to security vulnerabilities be deployed across the enterprise within a short, sometimes very short, time period.

- Application of fixes to address specific issues encountered in the production environment - i.e. break/fix scenarios.

The key characteristic of reactive maintenance is that there is little time for comprehensive pre-deployment testing prior to roll-out. This implies that most customers want to minimize the amount of change they need to deploy in reactive maintenance situations on the premise that minimizing change in theory should help minimize the risk of introducing new regressions.

Major proactive maintenance windows where new hardware platforms and/or major OS or Database upgrades are deployed are typically scheduled for every 18, 2 4, or 30 months, depending on what's driving the change.

Many customers schedule minor proactive maintenance windows for every 3, 6, or 9 months.

Reactive maintenance occurs on an "as needed" basis.

Solaris 10 "dim sum" Patching

Oracle Sun releases over 2,300 patch revisions each year. This is not just for Solaris, but for all products using SVR4-based patches to deliver bug fixes and other enhancements.

For Solaris 10 alone, around 500 Solaris OS patch revisions have been released in the last 12 months for SPARC and another 480 for x86.

A total of over 5,000 SPARC and 4,600 x86 Solaris 10 OS patch revisions have been released to date.

This can result in a patch combinatorics issue which IPS architect Bart Smaalders refers to as "dim sum" patching, named after Chinese "dim sum" lunch buffets where customer pick and choose whatever they want from the buffet.

Customers can and frequently do end up with unique patch combinations on their systems which have never been tested as a unit by Oracle Sun.

For example, a customer could have Solaris 10 6/06 (Update 2) installed, patched with the Solaris 10 10/08 (Update 6) Kernel, the ZFS bug fix wad from Solaris 10 8/07 (Update 4), and a 'zoneadmd' sustaining patch from September 2011.

Issues with patch combinatorics are very rare thanks to rigorous Solaris development processes to ensure patch dependencies are complete and correct. The Oracle Sun Patch System Test also test the boundary conditions of patch combinatorics - namely patches on their own and all patches together.

But from a risk perspective, it's still less than ideal to be running a unique software combination. For example, unique software combinations can expose latent race conditions. These can be difficult to diagnose, because they are only exposed on such unique systems.

Oracle Sun recommends customers stay as up to date as possible with the Solaris OS Recommended Patchset.

But even this can be updated several times a month as patches are released which meet its inclusion criteria. Clearly it's impractical for customers to install every update to the Solaris OS Recommended Patchset.

Therefore, even if customers apply the Solaris OS Recommended Patchset every 3 or 6 months, there are many discreet Recommended Patchset "baselines" released within even this time frame, so different customers may be on different "baselines" because there are so many of them.

Solaris 11 Maintenance

We want to avoid replacing "dim sum" patching with "dim sum" packaging in Solaris 11.

We want to collate customers onto a relatively small number of discreet Baselines.

We believe this will provide a better customer experience by providing higher quality and a "safety in numbers" effect.

Oracle Sun QA teams will intensely test each Baseline as a unit, resulting in higher quality.

Having many customers install the Baseline means that pervasive issues should be identified quickly and hence fixed quickly.

Bug fixes will typically be released in monthly Support Repository Updates (SRUs) with every 3rd SRU being renamed as a CPU (see above). These can be viewed as discreet maintenance Baselines. So can Update releases.

There will be a relatively small number of such discreet Baselines.

We expect customers to deploy one of these Baselines of their choosing in proactive maintenance windows.

We recognize the customer requirement to perform reactive maintenance as outlined above. This may cause customers to move temporarily away from a Baseline.

Customers may also need to deviate from a Baseline due to constraints imposed by ISVs - for example incompatibilities with specific package updates in Solaris due to their use of private Solaris interfaces (which is naughty) as they are subject to change.

We simply want to keep such deviations to a minimum.

We want to encourage customers to stay as close to a Baseline as possible and to revert to a Baseline (or as close as possible to it) in their next proactive maintenance window.

System Minimiziation, a.ka. Security Hardening

Customers will still be able to minimize systems.

Indeed, minimization is actively encouraged. Reducing the install footprint speeds up updates and reduces the opportunity for vulnerabilities to be exploited.

Thanks to IPS being a single tier packaging architecture, adding packages to a system at any stage if needs change is easy.

Adding packages to a system was a lot more complex under the SVR4-based model, as it was critical that any added packages be patched up to the same level as the rest of the installed system, and any patches pre-applied to the added package be applied to the rest of the installed system. This is called "incremental patching". Otherwise, the system would be in a compromised state, where patch dependency checking could fail, potentially resulting in system corruption.

Solaris 11 Granularity for Reactive Maintenance

Solaris 11 consists of an indivisible core of Solaris as well as non-core packages which can be updated independently in reactive maintenance scenarios.

The indivisible core of Solaris consists of a number of interdependent packages and is analogous to the Kernel patch in Solaris 10 and below. It consists of the Kernel, libc, libxml, ssh, core drivers needed to access the package repository, but not ssl, etc. It includes those core parts of the Solaris Operating System and Network stack which have private interfaces between them. Even in reactive maintenance situations, if any part of this indivisible core needs to be updated, the whole core needs to be updated as a unit, similar to a Kernel patch in previous Solaris releases. It's simply too risky to allow this core of Solaris to be sub-divided.

Non-core packages use the IPS "facet" feature, "version-lock", to enable them to be updated independently of a Baseline in reactive maintenance situations.

For example, it's possible to update Thunderbird, Mozilla, *Office, etc. independently of core Solaris and independently of each other too.

By default, non-core packages will be updated along with the baseline - e.g. to the Baseline provided by an SRU.

But they can be uncoupled from such updates by setting the IPS "version-lock" facet to false in such packages. This enables them to be moved forward independently.

Such packages can also be frozen to prevent them from being updated to a Baseline along with the rest of Solaris.

These features are key to addressing break/fix reactive maintenance issues.

Furthermore, we plan to maintain a Security package, which will be an empty package specifying optional dependencies on all Solaris packages which deliver security fixes. Installing the empty Security package will pull in updates to all Solaris packages on the system for which security fixes are available.

This security package will be updated whenever new security fixes are released.

Where a system has been minimized (i.e. unnecessary packages have been removed), IPS will respect that minimization when updating the system, unless an update to a pre-existing package includes a new mandatory dependency on an additional package.

FOSS, Userland, and Desktop deliverables in Solaris 11

The intention is to keep FOSS and Userland components delivered in Solaris 11 as up to date as possible with the community based versions.

For Desktop components, the intention is to keep in sync with Oracle Enterprise Linux (OEL) as far as possible.

The goal is to provide customers with a familiar and up-to-date user environment.

In previous Solaris releases, we tended to apply our strict rules to ensure compatibility for core Solaris onto FOSS, Userland, and Desktop components too, resulting in sometimes shipping outdated versions. In Solaris 11, we intend to try to keep a more sensible balance between compatibility and up-to-dateness, to avoid FOSS, Userland, and Desktop appearing outdated.

Core Solaris will continue to enforce strict compatibility to enable us to continue to provide our long standing binary compatibility guarantee which protects customer investments in Solaris.

Delivering Change in Solaris 11

As mentioned above, Solaris 11 Updates will be released periodically as needed.

Solaris 11 Updates may introduce new platform support, new software features, enhancements to existing features, bug fixes for non-customer-escalated issues, bug fixes for customer escalated issues, etc. This is similar to Solaris 10 Updates.

Solaris Updates provide intensely tested, high quality, stepping-stones to new and improved functionality, including a wealth of bug fixes.

In between Solaris 11 Updates, monthly SRUs/CPUs will be released as discussed above. This are primarily to deliver bug fixes for customer escalated issues, but low risk features, particularly platform support enhancements, and bug fixes for non-customer-escalated issues may also be included from time to time.

All code changes for Solaris are integrated to the tip of the source code tree. This enables a single sustaining code branch to provide bug fixes for all customers on a Solaris release.

For example, for Solaris 10, there's a single set of cumulative patches which patch all Solaris 10 versions. Similarly, in Solaris 11, there will be a single set of cumulative package updates which will apply to all Solaris 11 versions.

In Solaris 10, if a bug fix went into a patch which forms part of a Solaris 10 Update or the bug fix was integrated after an Update was released, then the cumulative nature of the resultant patch means that it will also contain any feature code changes and other bug fixes to the objects in question.

Similarly, in Solaris 11, SRUs/CPUs for one Update cease once the next Update is released. So if a bug fix is released in an SRU published after Solaris 11 Update 1 releases, then that SRU will also contain any feature code changes and other bug fixes to the objects in question, exactly as is the case in Solaris 10.

Feature code is typically switched off by default in Solaris Updates so that customers installing a patch or SRU/CPU to get a bug fix don't get surprised by new feature functionality.

Irrespective of the whether the packaging architecture is SVR4-based or IPS, the more a customer keeps up to date in proactive maintenance windows, the smaller the code delta will be in reactive maintenance situations.

That is, change, and hence risk, can be minimized in reactive maintenance scenarios by keeping the system as up to date as possible in scheduled proactive maintenance windows.

IDRs provide Interim Diagnostics and Relief. That is, diagnostic code to root cause customer issues or initial relief for the issue provided by the sustaining engineer working to fix the issue. Once the fix has been identified, it will go through the normal test and release process before being released for general consumption in a Solaris 11 SRU/CPU or Update.

Solaris 11 Maintenance Best Practices and Policies

We recommend customers schedule proactive maintenance windows to update to a Solaris 11 Baseline (i.e. an SRU, CPU, or Solaris 11 Update) at least once every 12 months.

Most Solaris 10 customers already schedule proactive maintenance windows for as least this frequency. So just keep the same proactive maintenance practice for Solaris 11.

While limited "dim sum" packaging is allowed to facilitate customer requirements for reactive maintenance, we want customer to revert to a well tested Baseline (or to as near to a Baseline as is possible) in proactive maintenance windows.

To prevent rampant "dim sum" packaging without reversion to a Baseline, all Solaris 11 deliverables will specify a requirement on the Solaris Baseline released 24 months previously.

This means that applying any change to a Solaris 11 system will update packages on a system up to the package version contained in the 24 month old Baseline if they not already at that package version or a later package version.

Obviously, this won't take effect until 24 months after Solaris 11 releases.

The 24 month limit recognizes that some customers legitimately have proactive maintenance cycles lasting more than 12 months.

It's still perfectly possible to leave a system untouched for years. Such systems are still supported and issues on such systems will still be investigated, including issuing diagnostic IDRs where needed.

It's just that if any change is made to such a system, it'll be brought up to at least the 24 month old Baseline level.

The reason for this policy is to provide customers with a better Solaris experience so that they don't encounter issues which have already been fixed a long time ago.

It is intended that IDRs will be provided on any Baseline less than 12 months old. For example, if in May 2012 a customer reports an issue on Solaris 11 2011.11, an IDR may be provided for Solaris 11 2011.11.

If the customer subsequently updates to a Solaris 11 2012.05 SRU Baseline, and the issue isn't fixed in that release, another IDR may be provided for that Baseline.

If the customer is running a Baseline between 12 and 24 months old, an IDR providing relief based on the 12 month old individual package or indivisible core Solaris may be provided. That is, the customer doesn't need to update their entire Baseline to install the IDR, just the individual package or indivisible core Solaris plus the IDR.

If the customer is running a Baseline older than 24 months, an IDR providing diagnostics may still be provided on their Baseline. But an IDR providing relief to the issue will update the system to the 24 month old Baseline.

This policy facilitates a level of "dim sum" packaging necessary to satisfy legitimate reactive maintenance situations, while ensuring that customers return to a Solaris Baseline in due course.

This is not as radical a change as it may at first seem.

In Solaris 10, 86% of IDRs produced are based on patch levels less than 12 months old. This shows that most customers are good at keeping systems reasonably up to date in proactive maintenance windows.

As with Solaris 10 and earlier releases, a key point to remember is that the more a customer keeps up to date in proactive maintenance windows, the more discreet the change they can get in reactive maintenance scenarios.

Solaris 11 maintenance policies and deliverables will be regularly reviewed and feedback is welcome. Our desire is to provide the best possible customer experience on Solaris 11.

Everything contained in this document is for guidance only and is subject to change. You are reminded about the disclaimer on forward looking statements at the beginning of this document.

Contact address:

Gerry Haskins
Oracle
East Point Business Park
Dublin 3
Ireland

Phone:	+353-1-8199254
Email	Gerry.Haskins@oracle.com
Internet:	http://blogs.oracle.com/patch