Patching at Large with SUSE Manager

Marc Laguilharre, Premium Support Engineer, Marc.Laguilharre@suse.com
Silvio Moioli, Developer, moio@suse.com
Good morning, my name is Marc Laguilharre, I’m working in the Premium Support team for 20 years. That means that I have a limited number of major customers, mostly based in France. Currently I have four, and three of them use SUSE Manager.

I am co-presenting this session with Silvio Moioli, who has been working at SUSE Manager as a developer for the last six years.

In the last two years Silvio focused on performance and scalability improvements of the product, and is currently coordinating a group of five developers in that area.

I will of course give customer-focused view to this case study, while Silvio can tell you more about SUSE Manager inner workings and mechanisms.
This is the question we would like to tell you about today, sharing how one of our customers implements patching of a relatively large server landscape with SUSE Manager.
We are going to cover:
• an introduction to the customer and their specific environment
• an overview on the architecture of the solution
• some of the best practices in deploying and running large SUSE Manager solutions
• some customer-specific adaptations
• some troubleshooting steps
• in the end, time is reserved for your questions and answers
First things first, let’s give some context about this customer. As far as the name goes…
…we are unfortunately not authorized to communicate the customer's name directly in this presentation.

What we can say in general is:

• it's one of the biggest financial institutions in France
• it has a very important worldwide placing as well, according to Forbes

Customer's management did not approve the SUSE Manager Team to be present today because they are external employees. We might try to fix that for the next SUSECON!
We can also tell you this is a retail bank, and a pretty large one.

On the technical context side, this customer has a very big Red Hat installed base. In fact, virtually all of the 10,000 systems we are talking about are based on Red Hat OSs. This gives me the occasion to also give a bit of historical context – in fact, before migrating to SUSE Manager, this customer used to use a competing product we will not name here...
Well, OK, I guess we can name it!

As you might have guessed the product was Red Hat’s Satellite 6. For various reasons, Satellite was not really able to satisfy our customer’s requirements, so later it was decided to migrate to SUSE Manager.

Pain points included:
- Subscription model was not flexible enough, customer had to develop their own subscription tool to prevent incorrect assignment of physical subscriptions to ESX servers
- Red Hat’s answer was to have a dedicated VMWare cluster Linux, but this was not possible because of customer’s ESX’s team constraints
- Pulp (the repository management component in Satellite 6 had severe reliability issues, ultimately the customer had to develop a custom script to patch from a custom repo
- Despite the architecture originally presented, the solution could not go beyond 8,000 client systems because of bugs, customer gave up after a two year project
Other management products

- HP Server Automation – monitoring (legacy)
- BMC Client Management – configuration (legacy)
- Several VMware products – virtualization
- CNTLM – Active Directory integration

Some more technical context.

SUSE Manager is not the only management solution in place at the customer, other components are present from several vendors. Some of them have an overlap in features with Manager, especially in the Linux space, and some are in the process of being replaced.

BMC Client Management was formerly known as Marimba.
Organizational context

- SUSE Manager team: ownership of update infrastructure
- Security team: channel management, application of updates
- Provisioning team: ownership of virtual infrastructure
- Need for automation from all interested parties

More context – now on the organizational part.

The SUSE Manager team at our customer is very knowledgeable and young. They do administration and also have good development capabilities. They are reactive and accurate and creative... Perhaps sometimes even a little bit too creative! We like it that way - it is always possible to drive a fast car slow, while the opposite is much harder to accomplish.

I am very lucky to work with them and we have very good communication with SUSE Support, SUSE Consulting and SUSE R&D. To give you an idea, more than 300 emails were exchanged last year alone, and about 100 Service Requests were opened (some about SUSE Manager, others about SUSE Linux Enterprise Expanded Support). They see Premium Support (an assigned support engineer) as added value for their company.

That’s all about the customer context.
Solution architecture and best practices
Overall architecture has 3 layers: the main SUSE Manager Server, several Proxies and then clients. Network topology in this customer’s case is not particularly complicated.
This already allows me to start talking about best practices:
• few or no minions should be directly connected to the Server
• the number of Proxies must be adequate to the expected network traffic, which in turn depends on products and update cadences
• Clients should be distributed “evenly” to Proxies
Audience question: is there someone not familiar with the traditional/minion distinction?
For more best practices, let’s focus on individual nodes.
SUSE Manager workloads tend to exhibit CPU peaks during registration, patching. Not so much CPU activity is otherwise expected in a healthy system.

Plenty of RAM is advisable, as the SUSE Manager Server among other things hosts a Postgres database server that greatly benefits from any available memory.

Similarly, storage is pretty important and at this size we recommend local SSDs, in RAID if possible. In this case our customer used VMware datastore, which is a Fiber Channel SAN over Ethernet. This is not optimal from a performance point of view, but so far deemed acceptable.
• 1 Proxy per ~2,000 minions
• 16 GB RAM
• 1 CPU, 4 cores

• Hardware requirements on Proxies are significantly lower
• Bandwidth plays a much more important role
• It is advisable to add more (even smaller) Proxies in case of performance problems - 2,000 clients is already above the normal recommendation
Managed Systems

- Salt recommended!

- Salt minions are in general strongly recommended
- Traditional systems are still supported for the foreseeable future, in this case they are used to cover RHEL 5
- Coming next: recommended SUSE Manager features
Content Staging
(package prefetching)
Typical timeline for a round of updates: a maintenance window must be wide enough to accommodate time for downloading and applying the downloaded packages.

In some circumstances such as in the example here, downloading might even be the dominating factor - depending on available bandwidth.
What this feature allows you to do is to anticipate the downloading of packages, so that the critical maintenance window is shortened. In most cases, downloading can happen in the background without side effects well outside the maintenance window.

This allows to shorten the maintenance window time significantly. Equivalently, one defines a “download window” which is separated from the critical maintenance window.

This functionality is optionally enabled and needs two parameters to function.
The two parameters are in green at the bottom.

staging_window defines the length of the download window. Individual downloads will be spread randomly to minimize peak load on the HTTP server.

staging_advance defines “how long earlier” the staging window will open wrt the scheduled time of the patch action.

If staging_window equals staging_advance, the download window will close immediately before patching starts.
These are the parameters to be set in order to configure the feature. They get activated on next Tomcat and Taskomatic restart.

Once the feature is configured, it must be activated on an Org-by-Org basis.
This is the place in the UI where this feature is globally enabled.

Notes:
there is an equivalent functionality for traditional clients
the feature also works for new package installations
One of the core parameters to keep in mind when sizing and operating a SUSE Manager patching infrastructure at scale is the number of minions that are patched in parallel.

With thousands of systems, we naturally want to avoid a serial approach – unless we have months at disposal – but there are also limits to the capacity of SUSE Manager and, possibly, other third-party services involved to process massive input resulting from the patch application on (possibly) very many servers all at once.

Ideally, we should determine the maximum safe number of minions that can be patched in parallel without unexpected side effects and always use it, in order to minimize patch time. This feature limits SUSE Manager (in particular, Salt) to a given number of minions concurrently.
I can explain how this works with diagrams and Salt command line examples. In this case, all minions registered to the SUSE Manager Server above are updated simultaneously – likely the SUSE Manager Server (and any other infrastructure needed to carry out the update, for example repo HTTP servers) will receive traffic from all of them at the same time.
The --batch command line option allows one to limit the amount of simultaneously patched minions. In this case the first two are processed, and as soon as one is finished...

...another one starts processing.

We added support in Salt to expose this mechanism via the API (initially this was only available through the command line interface) and adapted SUSE Manager to make use of it.

This feature is enabled by default but the number of concurrent minions still can be changed manually (the default is sufficient for a small installation).
Please note this feature is expected for SUSE Manager 3.2.8.

A similar feature also exists for traditional clients, please refer to the manuals for configuration details.

This last feature allows me to introduce the next section, which is about tuning.
Is anyone from Gibson in the audience? Please accept my apologies for having used a Japanese guitar photo!

When the desired number of minions to be processed in parallel changes, and in general for all big installations, a number of other parameters might need adjustment to increase the SUSE Manager Server capacity - allocating more memory to specific components, allow them to use more worker threads/processes, etc.

A full tuning guide covering all SUSE Manager components (Apache httpd, Tomcat, Salt, PostgreSQL…) is well beyond the scope of this presentation, but we will go through some of the most important ones now.
/etc/rhn/rhn.conf

- Taskomatic threads: org.quartz.threadPool.threadCount = 100
- Taskomatic cycle time: org.quartz.scheduler.idleWaitTime = 1000
- Internal thread pool: java.message_queue_thread_pool_size = 100
- Java-reserved database connections: hibernate.c3p0.max_size = 150
- Presence ping timeout: java.salt_presence_ping_timeout = 20,
  java.salt_presence_ping_gather_job_timeout = 20
- Search server maximum memory: rhn-search.java.maxmemory=4096
Other configuration options

- Apache httpd: number of connections
- PostgreSQL: smdba autotune
- Tomcat: maximum heap memory
Please note many other parameters and explanations are available in the product’s official guides.

One further important, although somewhat implicit, configuration parameter is the SUSE Manager version. It would be easy to just reduce that to a recommendation to always stick to the latest and greatest, but we want to give some more context here.
We have established a continuous development cycle improving SUSE Manager – addressing bugs, improving performance and adding features.

Different SUSE Manager versions get a different portion of those changes, and deciding which version to use is your choice.
At any point in time we support two versions of SUSE Manager in parallel — right it's versions 3.1 and 3.2. Each release is supported for a total of about two years.

Every year, we publish a new version. When that will happen (it is expected this summer)...
…what will happen is that the oldest version, 3.1, goes out of support.

As you can see we have two pictures here - one representing still and one representing sparkling water. We use this analogy to explain what kind of changes we do to each version.

The “still” version only gets bug fixes. The “sparkling” or “fresh” version gets bugfixes, performance improvements and some new features as well.

It is up to you to choose the version that suits better. In general, the “still” version is more stable but receives less improvements — both in terms of new features and performance-wise. “Fresh” is what we typically recommend for large scale scenarios that benefit from all latest additions.

Regardless of the choice, we continuously improve the product and produce maintenance updates every few weeks. Keeping Servers, Proxies, clients and bootstrap repos up to date is strongly recommended.
Another couple of high-level suggestions from our part.

First suggestion: take the turtle approach when scaling up. Scale up slowly, and stop if any problem is found before new confounding elements enter the picture, making it more and more difficult to understand what is happening.
Second suggestion: we have been talking about many best practices so far including architecture, hardware sizing, product features, tuning, and versions. You might feel a bit lost (either at this point, or at any later point during your SUSE Manager experience). If you do, definitely get some help from experts!

Especially the involvement of consulting in the initial phases of a large SUSE Manager project can be vital. Our experience in this case underlines this particularly well.

I’m now handing it over to Marc who will talk to you about features and measures that are specific to this customer.
Especially the involvement of consulting in the initial phases of a large SUSE Manager project can be vital. Our experience in this case underlines this particularly well.

I’m now handing it over to Marc who will talk to you about features and measures that are specific to this customer.
Key auto-acceptance
(automated registration)

This is a Salt feature disabled by default but actively used by many of our customers, including the one subject to this case study.

The feature essentially allows one to bypass the standard check for new minions the very first time a Salt master is contacted by them.
The standard mechanism employs a so-called “fingerprint”, a long hexadecimal string that represents the new system. Ideally, the security-conscious sysadmin will check the fingerprint as displayed on the Salt Master and the Salt minion match before accepting the minion’s key, thus making it part of the managed infrastructure.

On one hand, this ensures no rogue minions get managed by the Salt master - on the other hand this might become difficult, if not impossible or simply not needed, if the minions are deployed automatically (depending on how exactly the customer is doing the deployment).

When the key acceptance mechanism is not needed, it can be disabled as in this case.
After applying this configuration change and restarting the Salt master, every minion will be automatically onboarded into SUSE Manager as soon as the minion starts.
SUSE Manager ships by default with an internal AAA (authentication, authorization and access control) mechanism. Authentication can optionally be delegated to the Linux PAM modules.

In our customer’s case, PAM was used to delegate authentication to Active Directory through the SSSD daemon. In general, the easiest setup employs `winbind`, but this was not compatible with the specifics of this customer’s Active Directory implementation.
Excerpt from the SSSD configuration file (/etc/sssd/sssd.conf):

```
[sssd]
config_file_version = 2
services = pam,nss
domains = FOREST1.ZCORP
domain/FOREST1.ZCORP debug_level = 6
id_provider = ad
auth_provider = ad
enumerate = true
case_sensitive = false
ldap_sasl_authid = "SUMA$"
krb5_realm = FOREST2.ZCORP
krb5_renewable_lifetime = 1d
krb5_renew_interval = 1h
ldap_id_mapping = false
ldap_user_name = sAMAccountName
ldap_user_gecos = displayName
```
SUSE Manager offers an XMLRPC API for scripting and integration with third party products, in addition to Salt’s own API. The vast majority of functionality available via the UI is also exposed via API, to the extent some of our customers basically never look at the Web console!

In this customer’s case, the availability of the API is essential to allow multiple teams working with SUSE Manager, exposing the bits that are relevant for each one.
Script examples

- `add_systems_in_cisco.py`: adds systems from a text file to a custom group (in batches)
- `cancel_all_actions.py`: cancels any Actions in progress
- `cancel_action.py`: makes sure no Action is accidentally still running in work hours (triggered via cron)
- `clean_denied_keys.sh`: clears any denied Salt keys (triggered via cron)
- `failed_systems.py`: creates a CSV report of systems that failed an Action

A programming course over the use of the API is unfortunately a topic on its own deserving more than one presentation, here we at least want to describe some use cases that are important for this customer to give an idea on what can be accomplished.
We have two feature requests from this customer currently being evaluated to basically have functionality from segregate.py into the main product. That script works well as a temporary measure.

Also note that a new feature to define environments integrated into SUSE Manager is in development as of today.
One of SUSE Manager’s main features is the delivery of content to systems, in particular software packages. SUSE Manager Proxies, notably, fulfill the role of distribution caching nodes to adapt to different network topologies and conserve bandwidth.

In some cases, though, it has been noted that customers already have internal content distribution mechanisms, sometimes tailored to the specific needs or network topologies, and prefer to use those instead of SUSE Manager’s integrated facilities. A feature was added in version 3.2.7 to address that.
This is the standard behavior: packages and Salt’s ZeroMQ control messages follow the same path. What if we wanted to use a different path?
In this case, SUSE Manager is configured not to serve packages directly through the Proxy, but using a third-party CDN, which could be as simple as a different http proxy or as complex as a global content distribution network hosted in a cloud.
Configuration of this feature, which is a Salt exclusive, happens via Pillars. The first step is to create a pillar top file.

This example states that the pkg_download_points.sls file (which we will present in a moment) applies to all minions.
In this example, we conditionally apply alternate download endpoints based on the minion’s FQDN. Any other grain (or other pillar, like system groups) could have been used.
Alternative solution: yum priorities

- Provided via `yum-plugin-priorities` package
- Allows to define “backup repos” to be used when the main one is unavailable
- `yum-config-manager --setopt='suse*.skip_if_unavailable=1' --save`
- `timeout`, `retries` and `enabled` parameters need to be set on each repo line

This feature is relatively recent in SUSE Manager, our customer needed some solution sooner than that. In this case, a solution based on Yum’s repo priorities was devised.

This is a mechanism native to yum that will result in using alternative repos in case the main one is unavailable after a certain number of retries.

Customer put all together in a cron job running every 5 minutes. Result is a one-liner, albeit not very elegant.

```
echo "*/5 * * * * root yum-config-manager --setopt='suse*.skip_if_unavailable=1' --save 1>/dev/null ; grep -q 'timeout' /etc/yum.repos.d/susemanager:channels.repo || sed -i '/name=/a timeout=15' /etc/yum.repos.d/susemanager:channels.repo; grep -q 'retries' /etc/yum.repos.d/susemanager:channels.repo || sed -i '/timeout=/a retries=2' /etc/yum.repos.d/susemanager:channels.repo ; sed -i s'/enabled=0/enabled=1/' /etc/yum.repos.d/my-http.repo" > /etc/cron.d/repos.cron
```
The salt-minion daemon normally operates on read-write filesystems (for obvious reasons).

It is still anyways possible to run salt-minion (with minimal configuration change) even in case the filesystem is remounted read-only in case of catastrophic failures. This at least allows an operator to reboot the systems from the Salt master (using magic sysrq keys from the /proc filesystem, for example).
Configuration of minions on read-only filesystems

```
service salt-minion stop

    echo 'tmpfs /var/cache/salt/minion/proc/ tmpfs
defaults,size=5M 0 0' >> /etc/fstab

    mount /var/cache/salt/minion/proc/

    service salt-minion start
```

Caveat: this solution was devised by our enterprising customer and it works for their use cases, but this is not officially endorsed yet! Use with caution.
Here are some general guidelines and tips that helped us when dealing with SUSE Manager troubleshooting, especially in big scale scenarios.
Follow Action processing

- You can find information about triggered Actions in `rhn_web_ui.log` and `rhn_web_api.log`.
- Use `taskotop` for Taskomatic jobs.
- A Web UI with the same information displayed in `taskotop` is available.
- In general, use a top-down approach.

Typically the first thing you want to do is to make sure your Actions are being executed correctly. In order to do that, you can use the UI or command line tools.

Problem resolution can often be tackled best with a top-down approach, trying to isolate components that do not behave as expected.

For example, when seeing an issue, one should wonder: is it due to Taskomatic, or Salt, or the UI?

SUSE Manager – Under the Hood [TUT1039] is a good session to learn more about SUSE Manager internals.
Important support input

- Output files from `supportconfig, sosreport`
- Timestamp when the issue occurred
- Salt event bus dump
  
  `salt-run state.event pretty=True | tee /tmp/error-salt.lst`

If it really looks like an issue, your support engineer will probably need this data to tell more.
Advanced cases

• Database dumps
• SUSE can create test environments quickly via Terraform to try to replicate and isolate issues
• Give feedback!

If it really looks like an issue, your support engineer will probably need this data to tell more.

Feedback in general is always useful and some of the features presented today were actually developed with this customer, for this customer - and made general and available to all SUSE Manager (and Uyuni) users!
Q&A