Manage your hardware failures

(Mattieu Puel, Hepix Spring ‘17)
Manage your hardware failures
in an (almost) automated workflow

(Mattieu Puel, Hepix Spring ‘17)
Preliminary notes

Straight to an example

Architecture concerns

Some figures

Evolutions

Conclusions
Are you concerned?

May ring a bell if:

- your own people handles hardware failures (from detection to resolution)
- your interactions with support teams are time consuming:
  - filling in incidents into vendor ticketing systems
  - providing diags based on specific support requests
  - answering satisfaction polls
- even for the most simple case
One word about DELL

Vendor SOAP APIs
ONE word about DELL

Vendor SOAP APIs

Dispatch
ONE word about DELL

Vendor SOAP APIs

Support

Dispatch

Waranty
ONE word about DELL

Vendor SOAP APIs

Support

Dispatch

Waranty

Vendor x86 monitor OMSA
Agenda

Preliminary notes

**Straight to an example**

Architecture concerns

Some figures

Evolutions

Conclusions
Straight to an example: a good old DRIVE FAILURE

Vendor API

nono2dtd

Redmine (ITS)

Dispatch validator

Sysadmin
Straight to an example: a good old DRIVE FAILURE

Vendor API

nono2dtd

Redmine (ITS)

Dispatch validator

Sysadmin
Straight to an example: a good old DRIVE FAILURE

Vendor API

nono2dtd

Redmine (ITS)

fill in

Dispatch validator

Sysadmin
Straight to an example: a good old DRIVE FAILURE

Vendor API

nono2dtd

Redmine (ITS)

fill in

notify

Dispatch validator

Sysadmin
Straight to an example: a good old DRIVE FAILURE

Vendor API

nono2dtd

Redmine (ITS)

fill in

notify

self assign

Dispatch validator

Sysadmin
Straight to an example: a good old DRIVE FAILURE

Vendor API

nono2ddt

monitor

Redmine (ITS)

fill in

notify

self assign

Dispatch validator

Sysadmin
Straight to an example: a good old DRIVE FAILURE

Vendor API

nono2dtd

create dispatch

monitor

Redmine (ITS)

fill in

self assign

notify

Dispatch validator

Sysadmin
Straight to an example: a good old **DRIVE FAILURE**

- **Vendor API**
  - notify
  - validate
  - create dispatch

- **Redmine (ITS)**
  - monitor
  - notify
  - self assign

- **Dispatch validator**
- **Sysadmin**
Straight to an example: a good old **DRIVE FAILURE**

- **Vendor API**
  - Notify
  - Validate
- **Redmine (ITS)**
  - Monitor
  - Create dispatch
- **Dispatch validator**
- **Sysadmin**
  - Notify
  - Self assign
  - Deliver the new part

**Informal Description:**
- Vendor API: Notify and Validate.
- Redmine (ITS): Monitor, Create dispatch.
- Dispatch validator: Notify.
- Sysadmin: Notify, Self assign, Deliver the new part.

**Diagram Elements:**
- Nono2dtd
- Fill in
Straight to an example: a good old DRIVE FAILURE

- Vendor API
- Redmine (ITS)
- Dispatch validator
- Sysadmin

- notify
- validate
- create dispatch
- monitor
- fill in
- notify
- self assign
- deliver the new part
Straight to an example: a good old DRIVE FAILURE

- Vendor API
  - notify
  - validate
- Dispatch validator
- nono2dtd
  - create dispatch
  - monitor
- Redmine (ITS)
  - fill in
  - replace part
  - notify
  - self assign
  - deliver the new part
- Sysadmin
  - deliver the new part
Straight to an example: a good old DRIVE FAILURE

- Vendor API
  - notify
  - validate
- Redmine (ITS)
  - create dispatch
  - monitor
  - fill in
  - close
  - replace part
  - notify
  - self assign
  - deliver the new part
- Dispatch validator
- Sysadmin
  - notify
  - validate
  - deliver the new part
1- I am notified by an email from the ITS
2- I assign to myself the incident
3- Xmas present for me! I relate this to its original issue
3- Xmas present for me! I relate this to its original issue.
3- Xmas present for me! I relate this to its original issue
3- Xmas present for me! I relate this to its original issue
I said almost
5- Once the drive rebuild ends, Redmine notifies the closure
Preliminary notes

Straight to an example

Architecture concerns

Some figures

Evolutions

Conclusions
A closer look at the architecture: The incident detection

An issue has the following metadata:

- **category**: disk
- **serial number**: 6Q4Y102
- **vendor**: DELL Inc.
- **model**: PowerEdge R720
- **part number**: CN0F617N726223AO008DA03
- **service**: openstack
- **datacenter**: Villeurbanne2
- **location**: C12/32
- **service request dispatch number**

Nono

- queries hardware status through plugins (disk, memory, controller...)
- generates a diagnostic file to be provided to the vendor when necessary
- is responsible for tracking failures in the ITS (open/close)
A closer look at the architecture: **The incident detection**

- **Redmine** (ITS)
  - Puppet
  - Vendor monitor
  - nono
  - HTTP/REST python-redmine

- **CMDB**
  - Host: S/N → vendor → model → service → datacenter → rack/y

- **Nono**
  - queries hardware status through plugins (disk, memory, controller...)
  - generates a diagnostic file to be provided to the vendor when necessary
  - is responsible for tracking failures in the ITS (open/close)
A closer look at the architecture: **The dispatch**

**SOAP API calls:**
- `list_dispatch()`
- `list_parts(SN)`
- `list_parts(model)`
- `create_dispatch(PN, SN, diag, location, contact)`
- `get_dispatch_status(SR)`

**Nono2dtd**
- queries Redmine for incidents that are assigned, dispatchable and with no SR specified
- creates the dispatch for explicit technician through the dispatch API
- retrieves the SR or denial reason and report it to Redmine
- reports the DN to Redmine once set (after human validation)
This requires a little thought though

About technical details:
- do you use in-band or out-of-band host information?
- do you send fully introspected vendor diagnostics or targeted information?

About workflows:
- how do you manage issue assignment: self-assign? round robin it?
- how do you handle named deliveries and front desk interaction?

About the diagnostics:
- what probes will you rely on? (memory, batteries, BMCs, drives, enclosures, fans, PSUs, CPUs)
- how to deal with flapping alarms or false positives (batteries, PSUs)?
Drawbacks & limitations

Specific to *this* vendor:
- usage of self-dispatch requires vendor certifications
- usage of APIs requires enrolment program
- not much documented
- XML/SOAP
- unable to integrate the worker nodes (~1k hosts)

In general:
- need to deal with the support for all appropriate diags
- you can’t manage some components this way (DIMMs, PSUs...)
Agenda

- Preliminary notes
- Straight to an example
- Architecture concerns

**Some figures**

- Evolutions
- Conclusions
Following charts are based on around 420 days of data (329 incidents). This is a cured subset.

Issues by service:
Draw whatever **chart** you like!

Issues by category:
Issues by category:

Those two automated (~85%)
Issues over the host lifetime
Draw whatever chart you like!

Issues over the host lifetime

50.30%
Agenda

Preliminary notes

Straight to an example

Architecture concerns

Some figures

Evolutions

Conclusions
Envisaged evolutions

- use dispatch API for incidents requiring human diagnostic (PSUs...)
- integrating non Linux/OMSA boxes like storage arrays
- integrating worker nodes
- enrich logs to let ELK do the graphs
Agenda

Preliminary notes
Straight to an example
Architecture concerns
Some figures
Evolutions

Conclusions
Key conclusions: how does this optimize sysadmins time?

- avoid vendor support traditional portals
- no interaction with vendor staff
- rough estimation is 80% time saved using self-dispatch (720s / case)
- rough estimation is 95% time saved using APIs (195s / case)
- **TOTAL : 54h** saved for the 242 automated cases last year
- virtuous side effects:
  - full internal incident tracking
  - a saner environment where parts are replaced faster
  - clarifies your processes and cases allocations
Van még kérdése?

(google-translate© courtesy...)