



How my first Network Automation project failed (and is still in production)

Urs Baumann Swisscom 29.05.2024

```
>>> qr = QRCode()  
>>> qr.add_data("https://www.linkedin.com/in/ubaumannch")  
>>> qr.print_ascii()
```



If you scan the QRCode and there is a security warning please ignore it and enter your credit card details.

How it begun ...



You have automated CCIE Lab deployments.
Could you build a "Staging Robot"?

Customer A (2015)

Give a short intro on how the project started, not too many details. Saying that we never could convince the customer to use our name for the product. It was always Staging Robot for him.

Software Engineer

- System Engineering background
- "Hardcore code reviewer"

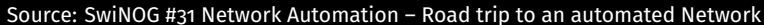
Network Engineer

- Basic programming skills
- "It is working, isn't it?"

¹System Engineer (BSc Student) with a flair for UI joined for UI

It was really beneficial this combination. No code was merged without review and tests. Learning the hard way. Try not to be a one-man show.

NAF



Spend some time to explain the architecture. Would I do it again the same? With my skills now; Probably FaaS. Django was the right decision. Task management we gambled and lost. More to it soon

- No need to know in advance:
 - Serial Number
 - MAC Address
 - Model ID
- Support for different staging areas
- Staging directly at the destination

Explain the idea of event-based, automatic device discovery, device asking for information. In the end, only one staging area was used.

Disclaimer: Needed support for IOS 12 (no Python, no PnP)

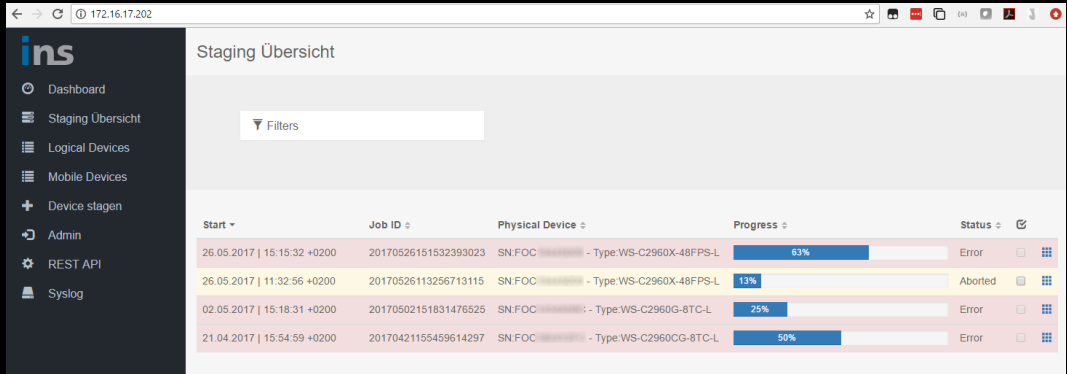
- The First idea was to use EEM
- EEM is not supported on L2 devices
- TCL works on all platforms
- TCL has slightly different versions and different libraries

Disclaimer: Needed support for IOS 12 (no Python, no PnP) PoC with EEM applets; figure out that EEM is only available on L3 devices. Finding out that TCL shell also works on L2.

- More than 900 lines of code
- Around 100 "if" statements
- Aproxamitly 50 cold showers

Using TCL was not the best idea I had. It ended up being complex as there are so many ways how to upgrade from one version to the next over the years. It has so many if's we can call it already AI as it simulates human behavior upgrading a box. The script collects PID, Version, CDP neighbors and talks to our own API to ask if a job exists. If a job exists and the version is not correct the software is updated, Config stored in startup and some special Layer 2 commands are executed as some platforms did not configure it correctly at boot time

How it looked - Job Overview

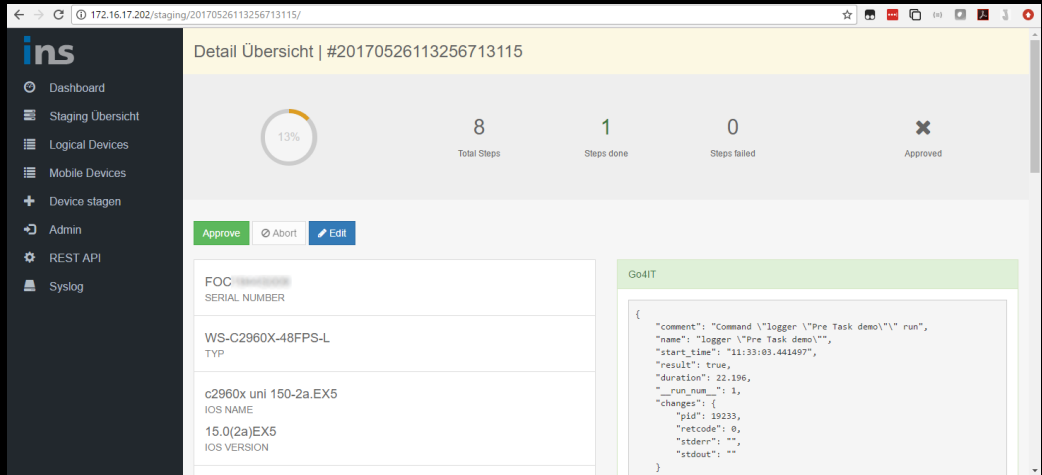


Start ▾	Job ID ▾	Physical Device ▾	Progress ▾	Status ▾	
26.05.2017 15:15:32 +0200	20170526151532393023	SN:FOC - Type:WS-C2960X-48FPS-L	63%	Error	<input type="checkbox"/>
26.05.2017 11:32:56 +0200	20170526113256713115	SN:FOC - Type:WS-C2960X-48FPS-L	13%	Aborted	<input type="checkbox"/>
02.05.2017 15:18:31 +0200	20170502151831476525	SN:FOC - Type:WS-C2960G-8TC-L	25%	Error	<input type="checkbox"/>
21.04.2017 15:54:59 +0200	20170421155459614297	SN:FOC - Type:WS-C2960CG-8TC-L	50%	Error	<input type="checkbox"/>

Source: SwiNOG #31 Network Automation – Road trip to an automated Network

Don't spend too much time. Django backend rendered with Angular. Okay enough for the time

How it looked - Job Details



Source: SwiNOG #31 Network Automation – Road trip to an automated Network

Don't spend too much time. Point out the task steps are taken out of Salt and the "ugly" value is the Salt internal data structure. Make sure to have good transparency and make your life easier by allowing the user to troubleshoot

- Using event bus
- Provides API
- Over time way too many workarounds

```
{% set printlabel = salt['pillar.get']('printlabel') %}  
{% if printlabel in ['8 mm', '18 mm', '16 mm'] %}  
label_print:  
  cmd.run:  
    - description: Print label  
    - name: print_label.py {{ printer_ip }} 1 "{{ salt['pillar.get']('hostname','') }}"  
{% endif %}
```

Explain why SaltStack was in the end not a good idea and added dependencies and was hard to make it work. Usage of Salt how it was not intended.

- Hierarchical Network-Domain variables
- Generate WebForm from Template variables
- Prepare Device Management with SaltStack
- Templates and template snippets

Many great features were implemented because the idea was the user will use it. Be careful to not end up with dead code in the code base. All these features are still in the code but not in use (anymore)

Generate Form from Template

■ Parameter

```
hostname: testname
ntp:
  - time0.ins.hsr.ch
  - 152.96.120.53
```

hostname

domain_name

lab

ntp

server

+ Add

■ Template

```
!
hostname {{ hostname }}
domain-name {{ domain_name|default('lab') }}
!{% for server in ntp %}
ntp server {{ server }}
!{% endfor %}
end
```

```
!
hostname testname
domain-name lab
!
ntp server time0.ins.hsr.ch
ntp server 152.96.120.53
!
end
```

Source: SwiNOG #31 Network Automation – Road trip to an automated Network

Was one of my favorite features. Depending on what variables are defined a form was created with list and dict support as well as default value support. Was not used and is dead code with dependencies to libraries now. Yes, I should have removed it again from the code base.

"Emergency" OS upgrade



We hit a dot1x bug and need to upgrade 5000 switches ASAP.
Could the "Staging Robot" do that?

Customer A (2017)

- Own implementation of Plug&Play
- UI with CSV export/import to select a time window
- Slow start

Wanted to make a simple netmiko script but Customer A convinced me to generate a WebApp to select what time the device is allowed to automatically upgrade it self. Was completely integrated into the "Staging Robot" afterward but never completely fit into the architecture as some stuff was done quick quick and it worked so why refactor? The backend code is still there but not in the frontend anymore as it is not used anymore.

- Use synergies from other internally developed tools
- SinglePage with react
- Integrate multiple backends in the same UI

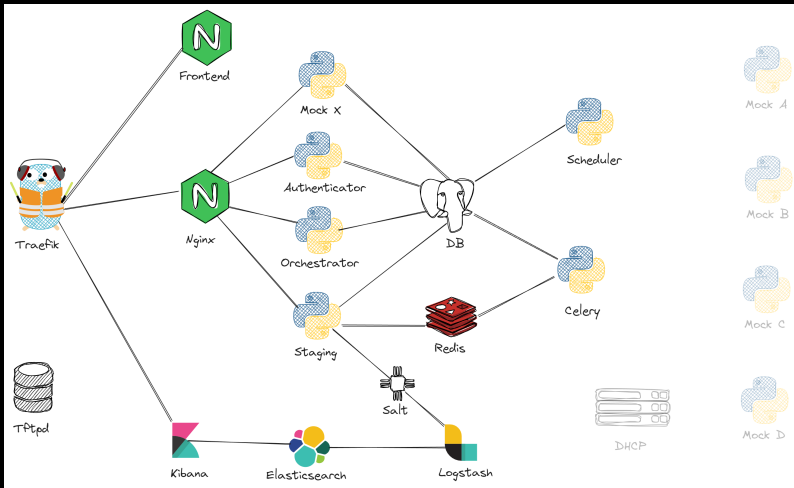
Simple UI had its limitations. Frontend engineers created react single page UI trying to use the same for different tools from other internal teams. Nice UI but also much more complicated to change something. After some internal restructuring, this team does not exist anymore and my react skills are weaker than my sales skills.

We have a huge rollout upcoming. Could your tool be adapted?

Customer B (2019)

New customer wants the same but different. How to avoid having different code bases? It is important to have a plan and keep customers in the loop. Workflow on top of staging was needed as the staging was just a part. Could we have avoided it with a better architecture at the beginning? Difficult choice between adding unneeded complexity in the beginning and thinking about the future.

Architecture



Here I want to give an overview of the new architecture and how the 17 containers work together. We created some mock services because some 3 party tools were not ready and surprised: after more than 4 years one of the mock is still used.

- 1:1 Replace Device
- Replacement with a new device
- Replace with Stack
- Provision/Deprovision Access port (API triggered from ordering system)
- ToDo ...

Explain shortly the use cases customer B addresses with this solution

NetTowel					admin
Orchestration	Overview				
Overview					
Device Lifecycle					
Approval Report					
	Filter table				
	Start	Description	Status	Ack.	<input type="checkbox"/>
	2024-04-17 14:46	Enabling LAN port on 192.168.8028-02-001-0-0-0-0028-wd01-401-7108	Finished		<input type="checkbox"/>
	2024-04-17 14:42	Enabling LAN port on 192.168.8028-02-001-0-0-0-0028-wd01-401-7107	Finished		<input type="checkbox"/>
	2024-04-17 14:39	Enabling LAN port on 192.168.8028-02-001-0-0-0-0028-wd01-401-7106	Finished		<input type="checkbox"/>
	2024-04-17 14:35	Enabling LAN port on 192.168.8028-02-001-0-0-0-0028-wd01-401-7105	Finished		<input type="checkbox"/>
	2024-04-17 14:31	Enabling LAN port on 192.168.8028-02-001-0-0-0-0028-wd01-401-7104	Finished		<input type="checkbox"/>
	2024-04-17 14:27	Enabling LAN port on 192.168.8028-02-001-0-0-0-0028-wd01-401-7103	Finished		<input type="checkbox"/>
	2024-04-17 14:23	Enabling LAN port on 192.168.8028-02-001-0-0-0-0028-wd01-401-7102	Finished		<input type="checkbox"/>
	2024-04-17 12:50	Enabling LAN port on 192.16710-02-002-0-0-0-0010-wd02-801-7102	Error		<input type="checkbox"/>
	2024-04-17 12:49	Enabling LAN port on 192.16710-02-002-0-0-0-0010-wd02-801-7101	Error		<input type="checkbox"/>
	2024-04-16 15:14	Enabling LAN port on 192.168.8028-02-001-0-0-0-0028-wd01-401-0008	Finished		<input type="checkbox"/>
	2024-04-16 15:11	Enabling LAN port on 192.168.8028-02-001-0-0-0-0028-wd01-401-0006	Finished		<input type="checkbox"/>

very quick. how it looks now. One SinglePage to manage the different components

NetTowel

admin

Orchestration

Overview

Device Lifecycle

Approval Report

Orchestration Task Details for Staging new device

ABORT RESTART STAGING TASK

100%

9

9

0

Total Done Failed

VERIFY_DISTRIBUTION_INTERFACE_AND_PORTCHANNEL

UPDATE_DISTRIBUTION_INTERFACE_WITHOUT_PORTCHANNEL

REGISTER_DNS_ENTRY

CREATE_STAGING_TASK

CHECK_STAGING_TASK

ADD_ISE_HOST_RECORD

UPDATE_DISTRIBUTION_INTERFACE_WITH_PORTCHANNEL

WAIT_FOR_REBOOT

ADD_PRIME_HOST_RECORD

Start

2024-04-11 17:03

Status

FINISHED

Submitted by

system

Last updated

2024-04-12T09:57:10.463753Z

Acknowledged

Data

very quick

- Exactly what you need vs general
- Resource-intensive vs locking
- Dependencies
- ToDo ...

At this time I was in another internal team but still the main know-how owner and product owner. We had long discussions about using an existing workflow engine or not. The software engineering team ended up implementing a simple workflow engine by themselves. Nice Object-oriented solution. Drawback: only a software developer can extend it. Would a normal do it? Not sure. I was wrong by Salt so maybe I am wrong here too?

Configuration Task

NetTowel

admin

Configuration

Overview

Create Configuration Task

Create Configuration Task

Step 1: General Parameters

Name

Select Task

Execution Timeframe

Start

End

napalm_get

napalm_update

restconf_delete

restconf_get

restconf_update

NEXT

Step 2: Hosts

Step 3: Arguments

Quickly: Added a simple component to abstract some stuff and dependencies and we could scale.

NetTowel

admin

Staging

Overview

Stage Device

IOS

Physical Devices

Logical Devices

Device Types

Network Domains

Templates

Template Snippets

Syslog

Stage Device

Step 1: Select Devices and Template

Select devices to stage

LOAD ALL DEVICES

☐ One to One Replacement

Select Network Domain

NEXT

Step 2: Device Configuration

Step 3: Staging Extensions

Step 4: Select ios Version for the devices

Where is the "Staging Robot"? Still here, still accessible but only used for trouble-shooting and visibility as the staging tasks are created by the orchestration component.

Customer A

- Used the Staging Robot until mid-2023
- Staged around 9'000 L2/L3 devices
- Around 5'000 PnP OS Upgrades

Customer B

- In production
- Todo: ...
- Todo: ...

Some numbers to show the dimensions. The TCL script is quite stable xD. ToDo: I need to get info from customer B

- Be prepared to debug the code
- Avoid basing on forks
- Try to be up to date
- ToDo ...

Need to work on this but want to point out what it means to rely on open-source tools. I made many PR and fixes in the last 9 years.

- Don't be afraid of failing
- Do not over-engineer
- Who can maintain this in 5 years?
- Does the customer need this feature or do you want it?
- No shortcuts

This is also not final yet. Want to have some more meat.

Why do I say the project has failed?



- No direct vendor locking but engineer locking
- No or outdated documentation
- SaltStack is a central component and hard to replace/update
- Settings are in many different locations (because of quick wins)
- Project had multiple contributors but only one has the global view

This is also not final yet. Want to have some more meat.