The Single Source of Truth for Network Automation

Andy Davidson <andy@asteroidhq.com>
March 2018
CEE Peering Days 2018, DKNOG 8, UKNOF 40

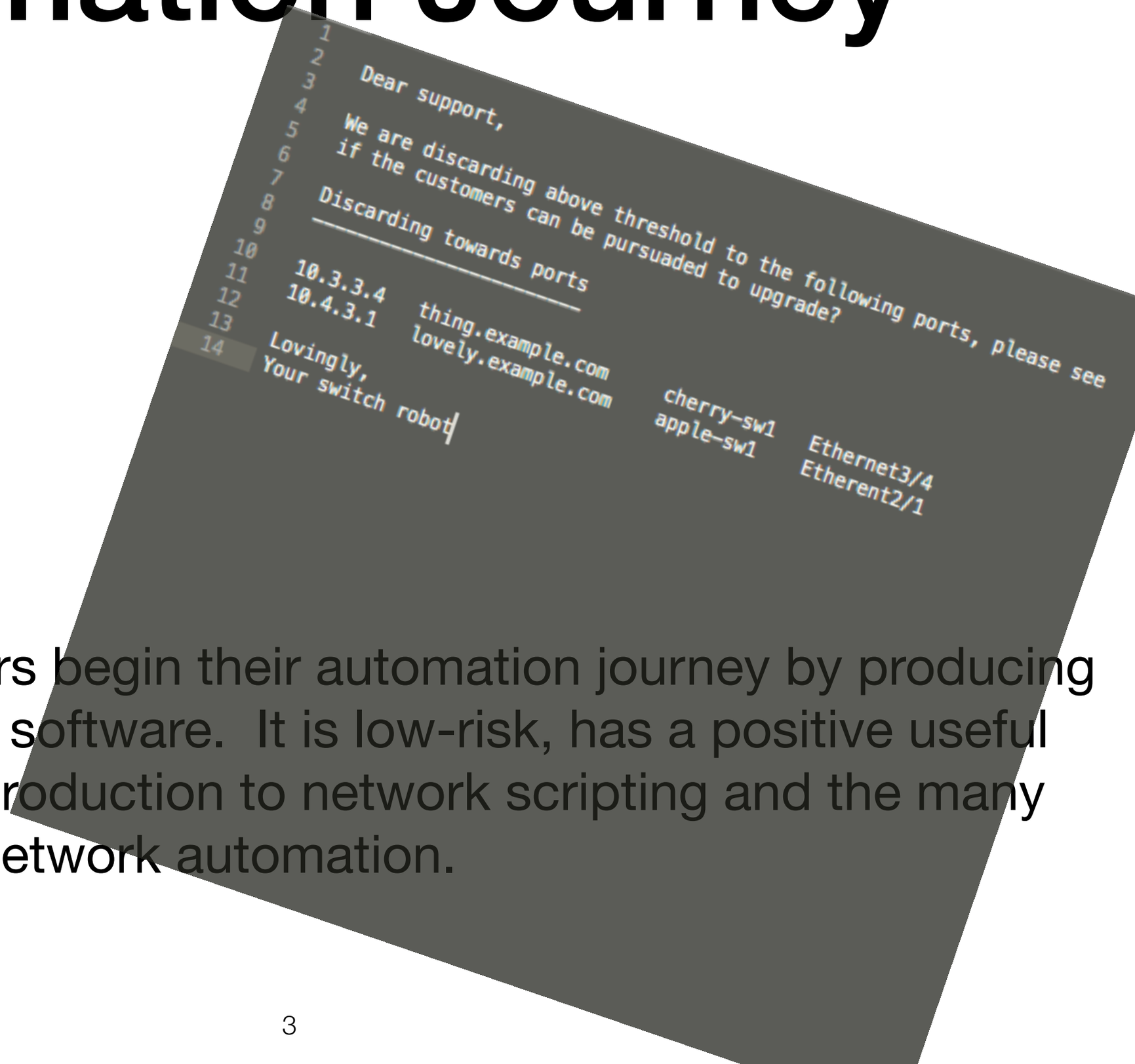
Automation Journey

Reporting

Most network engineers begin their automation journey by producing some simple reporting software. It is low-risk, has a positive useful impact, and a good introduction to network scripting and the many libraries that support network automation.

Automation Journey

Reporting



Most network engineers begin their automation journey by producing some simple reporting software. It is low-risk, has a positive useful impact, and a good introduction to network scripting and the many libraries that support network automation.

Automation Journey

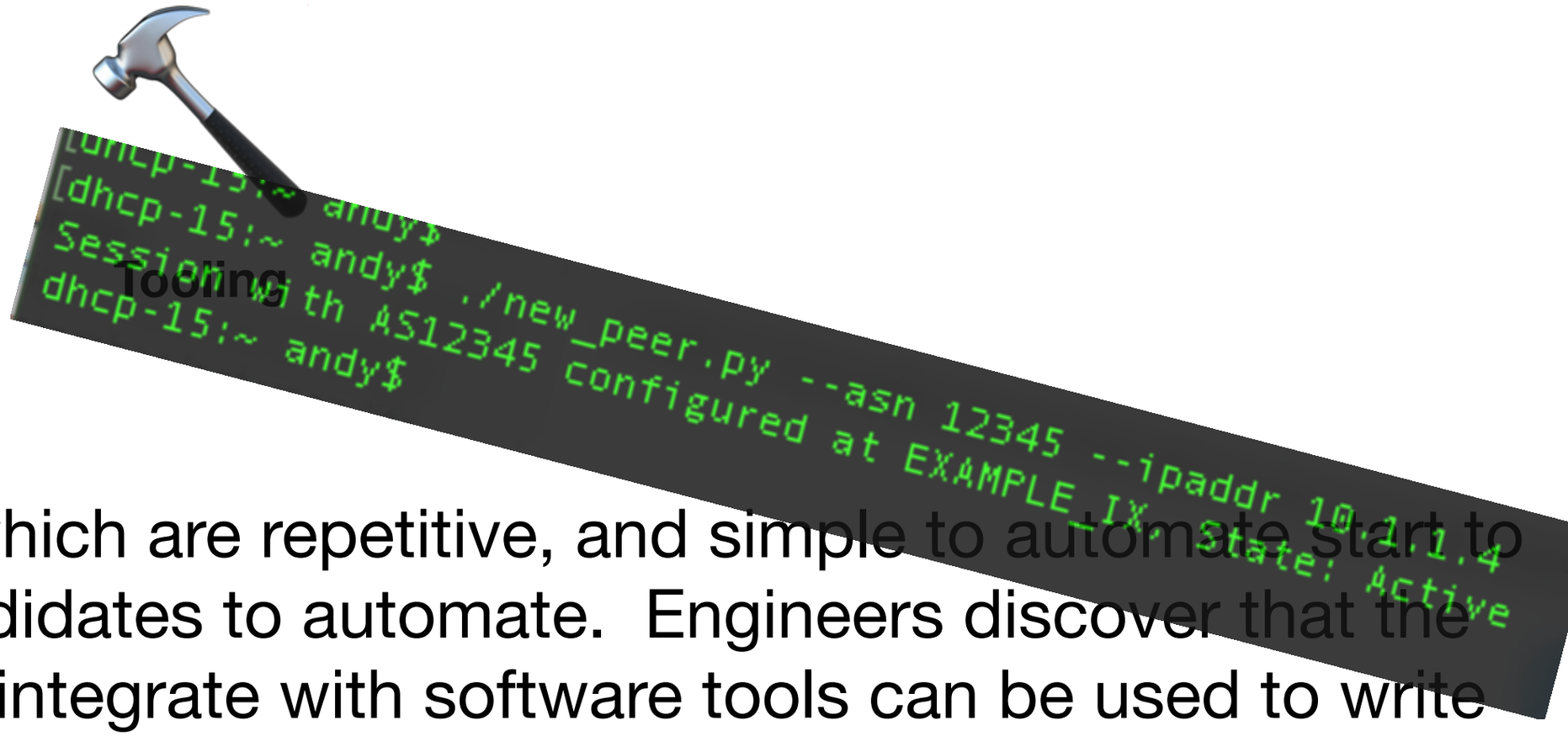
Reporting

Tooling

Eventually, tasks which are repetitive, and simple to automate start to look like great candidates to automate. Engineers discover that the great libraries that integrate with software tools can be used to write as well as read configuration, and simple standalone tools are created.

Automation Journey

Reporting



Tooling

Eventually, tasks which are repetitive, and simple to automate start to look like great candidates to automate. Engineers discover that the great libraries that integrate with software tools can be used to write as well as read configuration, and simple standalone tools are created.

Automation Journey

Reporting

Tooling

Application

More complex tools are eventually produced. Engineers begin to “configure the network and not the device”, so state becomes a problem (I mean state becomes properly managed). This takes the look and feel of a proper application.

Automation Journey

Reporting

Tooling

Application

More complex tools are eventually produced. Engineers begin to “configure the network and not the devices” state becomes a problem (I mean state becomes properly managed). This takes the look and feel of a proper application.

Automation Journey

Reporting

Tooling

Application

Business

The ultimate place to reach is a fully automated and integrated business with a set of processes enforced and delivered by software. “Configure the product, not the network”.

Generally solved by businesses with scale challenges (mass access, hosting) but now a commonplace medium sized ISP/IXP requirement.

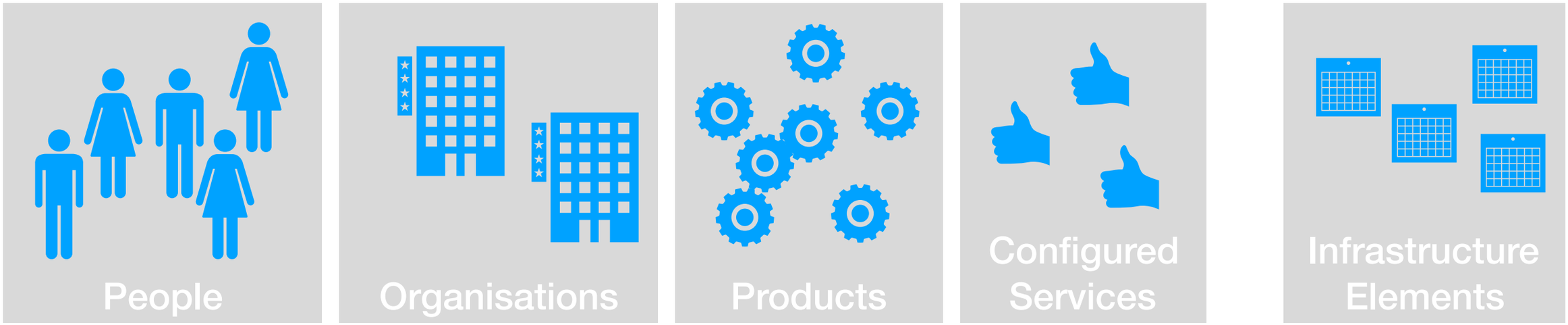
This presentation..

- Offer a technical perspective/thoughts on architecture on Greenfield deployment at the ‘automated business’ end of the spectrum
- What motivated this decision?
    - Replication - “as a service” product
    - Efficiency, leanness
    - Service assurance (rapid provisioning, ongoing high availability)
    - Integration with third party peering networks, Euro-IX, PeeringDB
    - Experience in this field, and frustration with traditional model
    - Chance to align business and technical process from the start - in our “DNA”

This presentation .. (2)

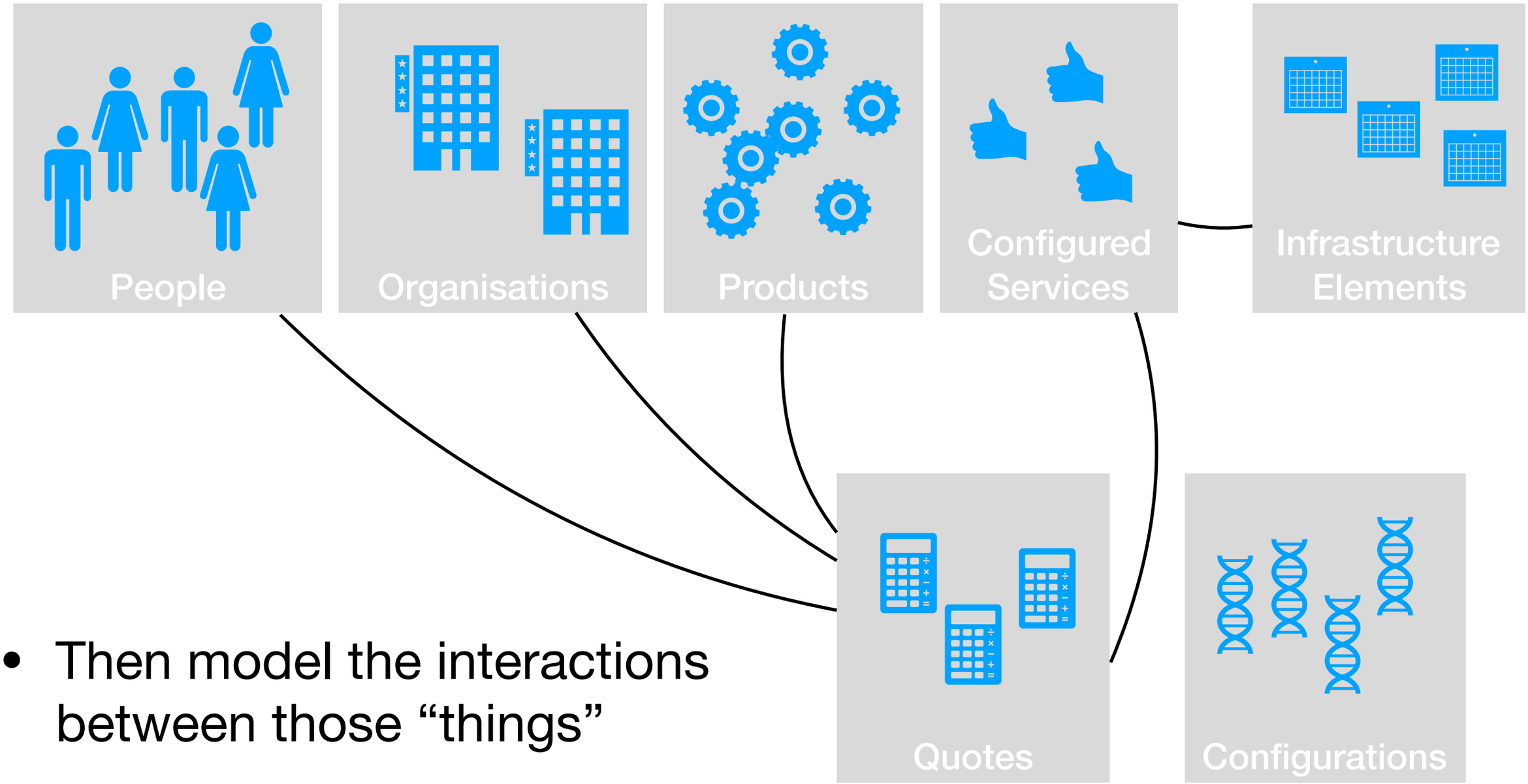
- Data model
    - Why and how to build a data model to support integrated automated business
- Software architecture for network centric businesses
    - Abstraction
    - APIs & API integration with customers
- Software testing
- Useful third party tools

What I mean, “data model”?



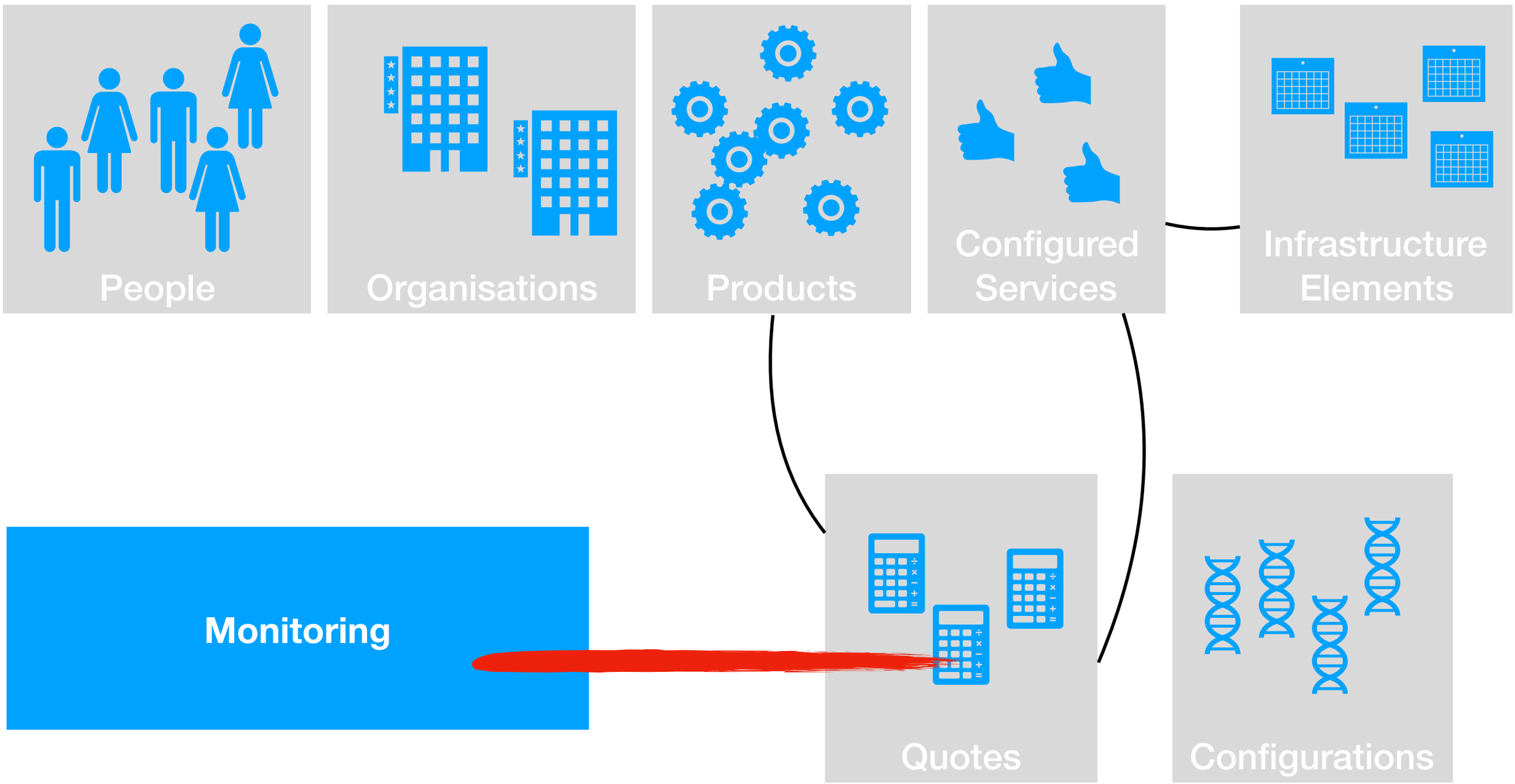
- A description of the things your business needs to ‘know’ in order to operate
- Start with the steady state of the business

What I mean, “data model”?

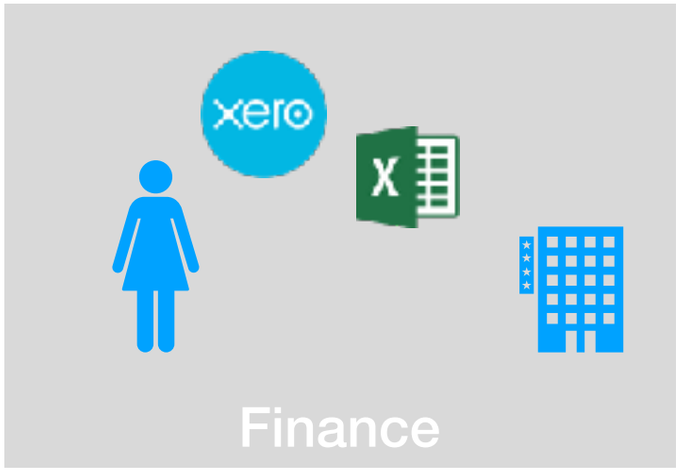


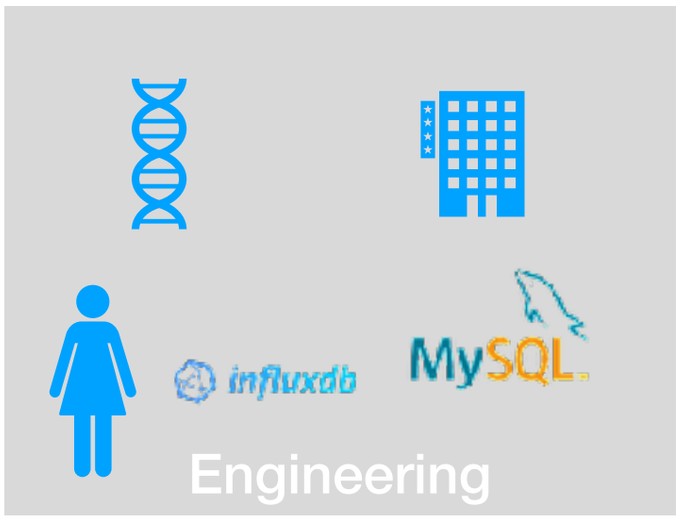
- Then model the interactions between those “things”

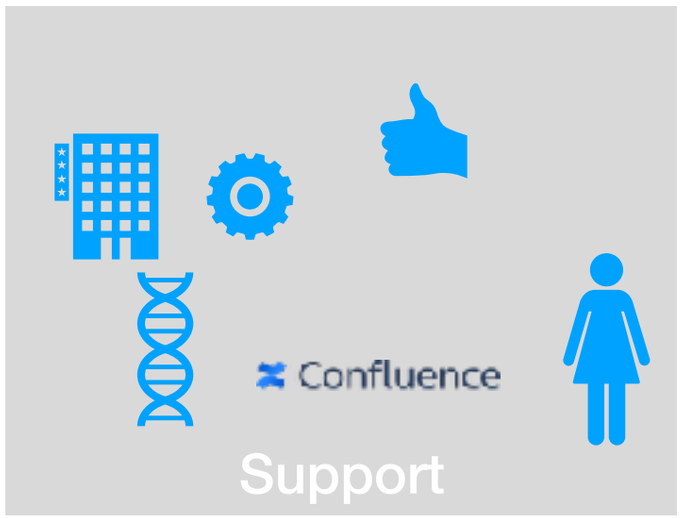
Why to care from an engineering point of view?



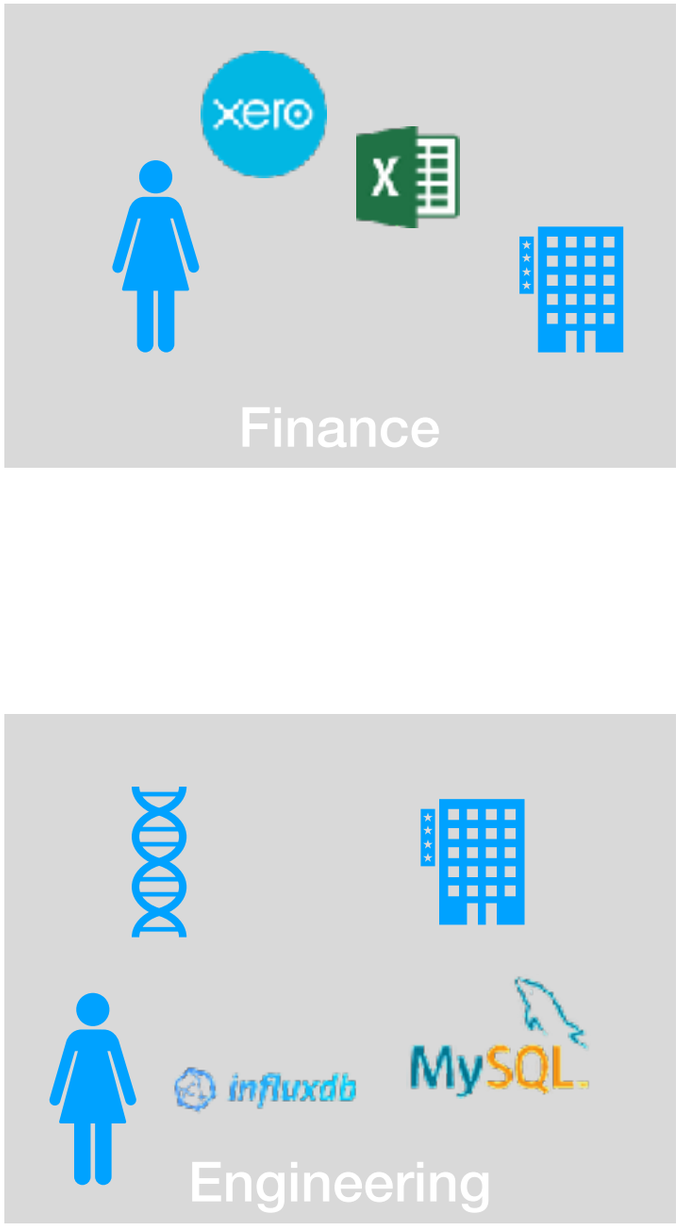
Where does/shall data live?



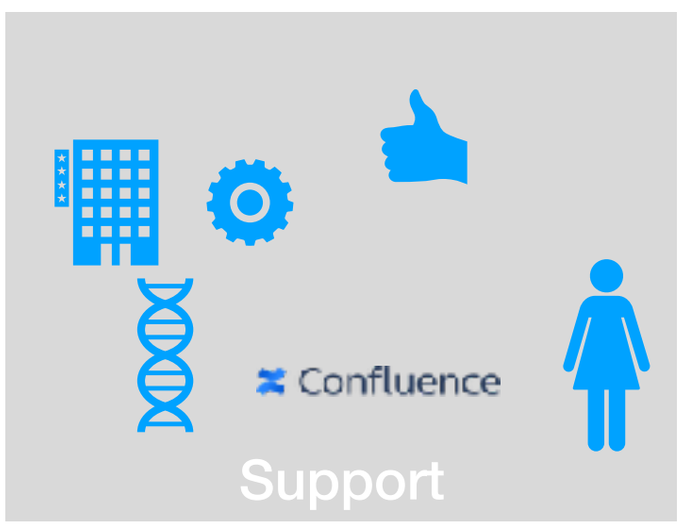




Where does/shall data live?

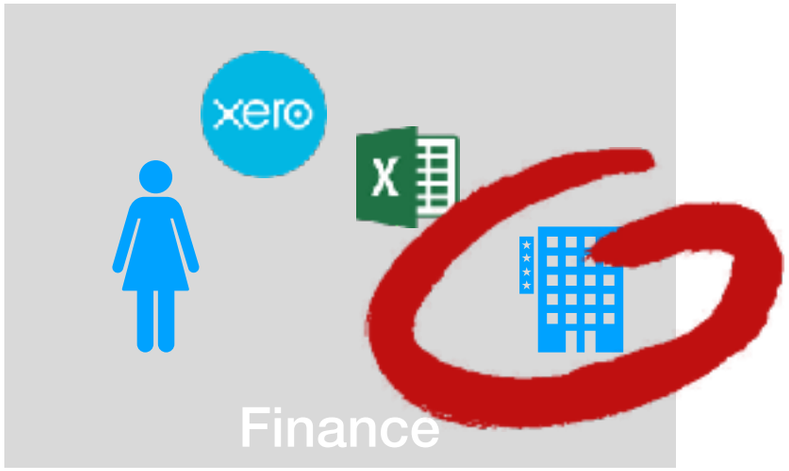


Fundamentally it is *fine* for data to “live” in different tools and databases

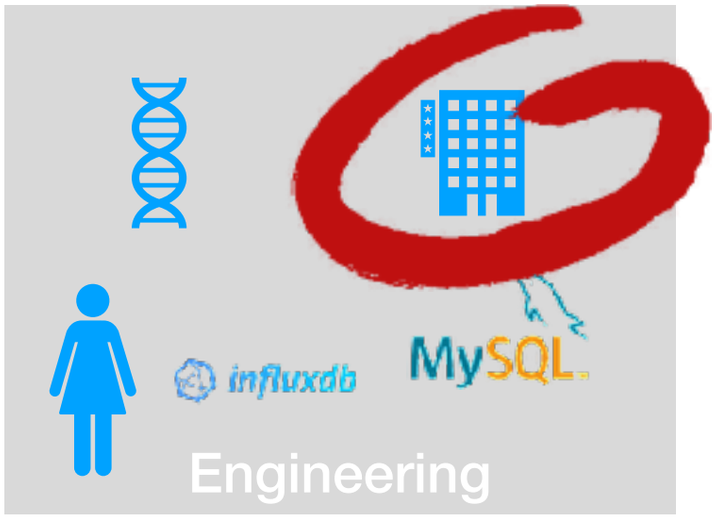


Where does/shall data live?

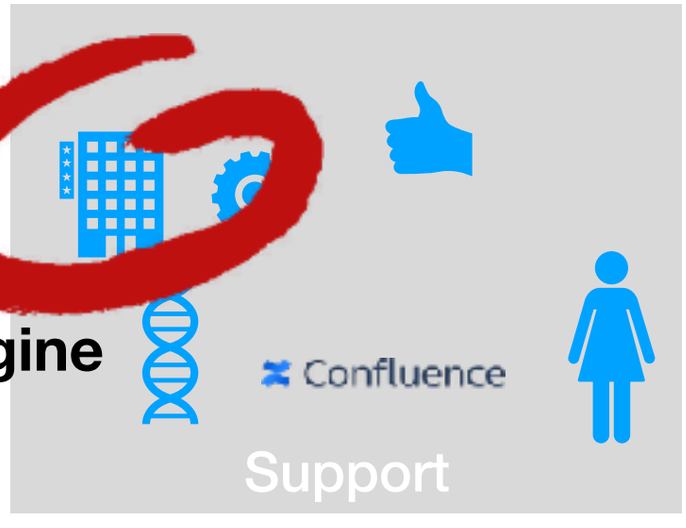
Search Engine, Inc.



Search Engine (Netherlands) B.V.

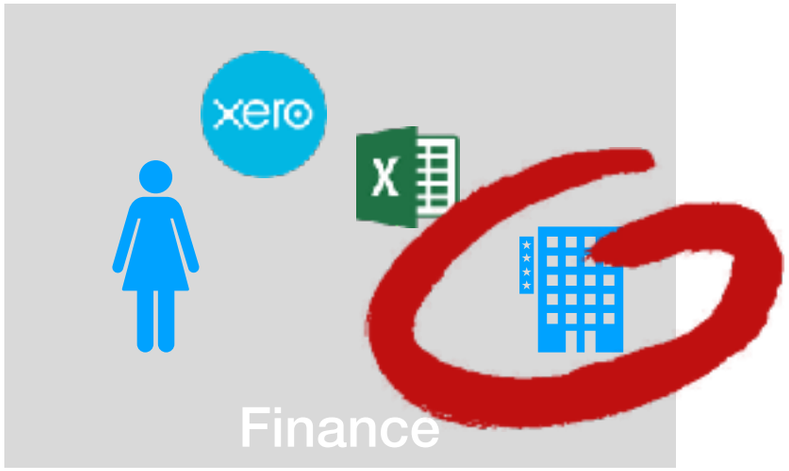


Search Engine

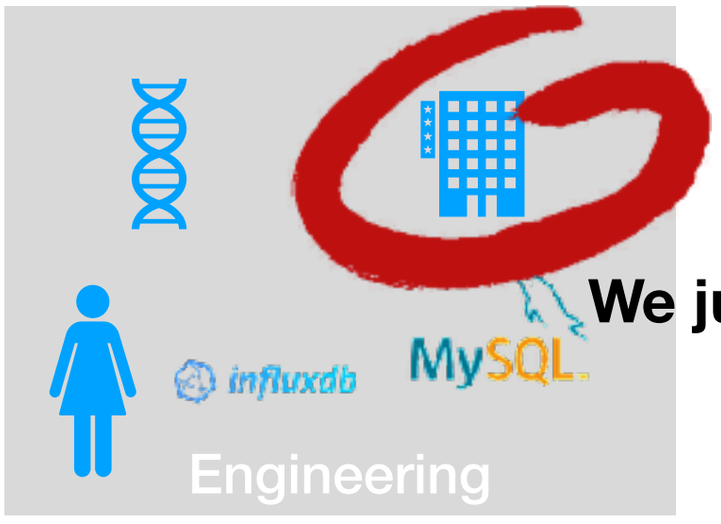


Where does/shall data live?

Search Engine, Inc.

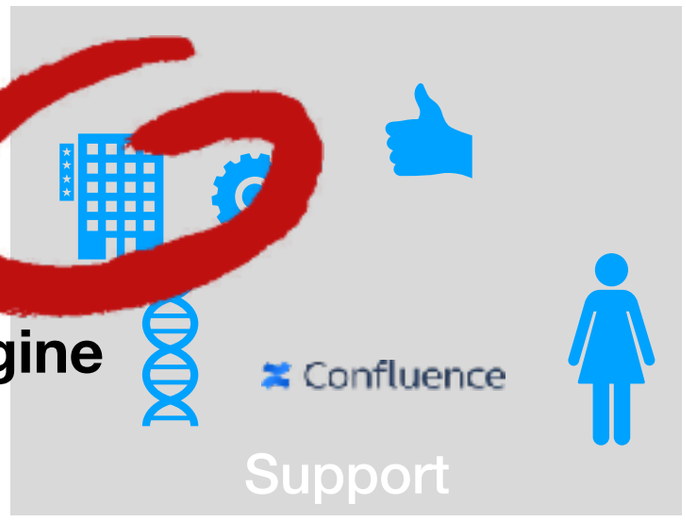


Search Engine (Netherlands) B.V.



We just deal with Fred

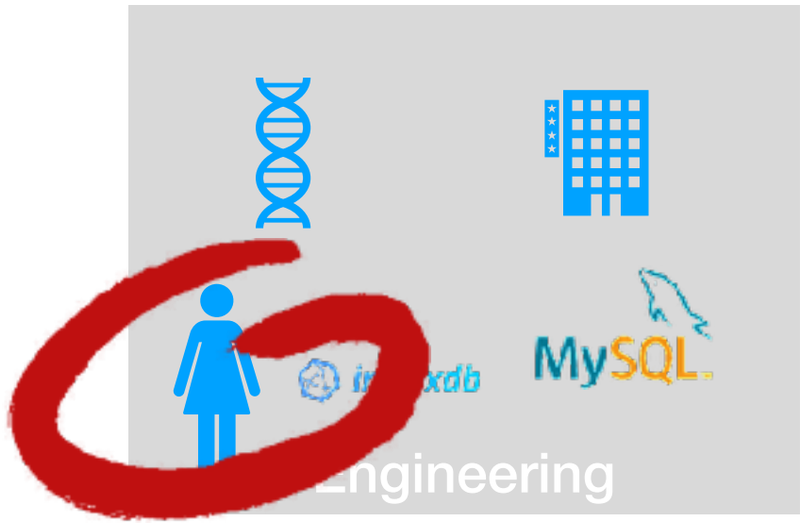
Search Engine

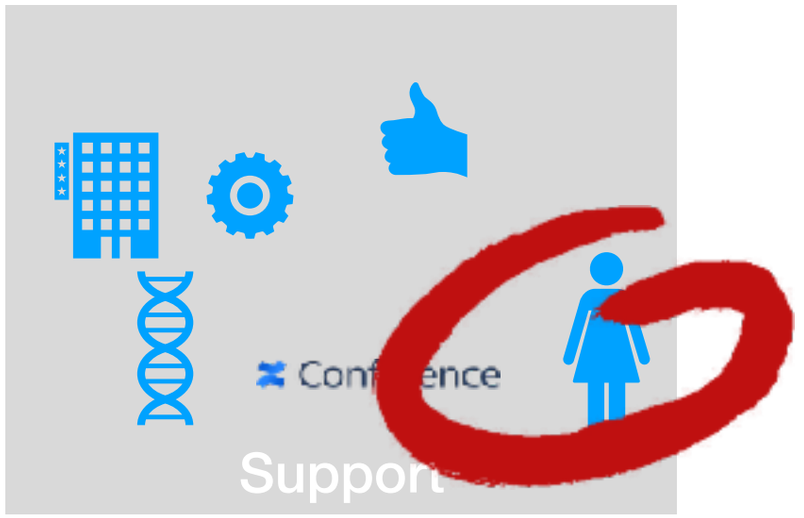


Where does/shall data live?

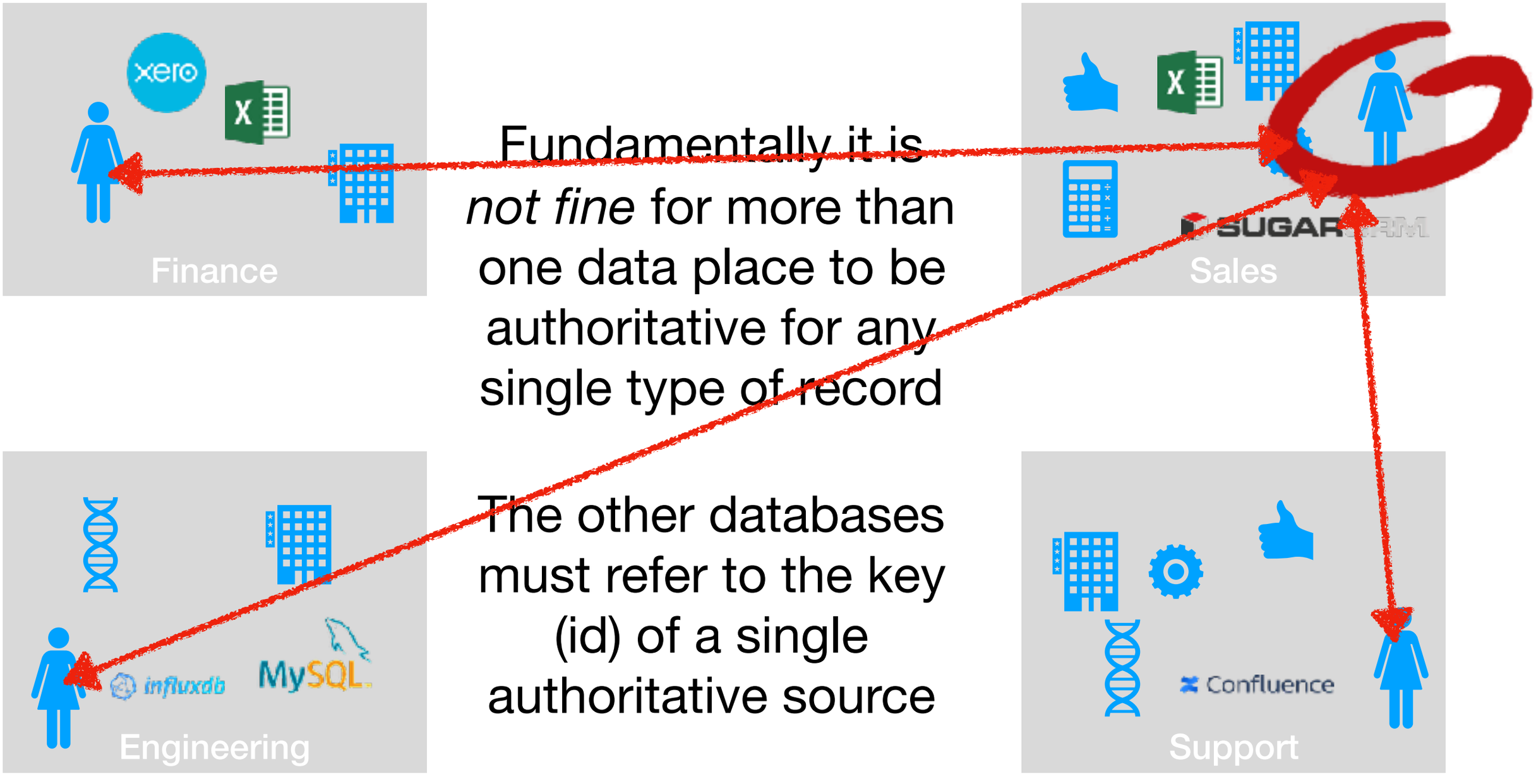
Fundamentally it is *not fine* for more than one data place to be authoritative for any single type of record

The other databases must refer to the key (id) of a single authoritative source





Where does/shall data live?



We will talk about how to configure and enforce that shortly.

Rules of Engagement

- Store any item of data **ONCE**
  - Easy to ensure that it is correct
  - “Third normal form”
- Give every record a **unique** ID which has nothing to do with the record
  - (ASN is not to be used as ID!)
- Decide **where** it will be authoritative
- Requires buy in and planning from across the business.

Separate your customer/ infrastructure data

Port
port_id
customer_id
bridge_id
port_name

Service
service_id
port_id
service_item1
product_id

Ensure infrastructure centric and customer-centric data is not in the same table
This will make your data substantially easier to maintain in terms of portability

Database Fashions

- Document store -vs- RDBMS
- Developers like document stores because they are very extensible and less strict
    - “Storage” cost reduced, so now we can be lazy
- Strict is a benefit / feature

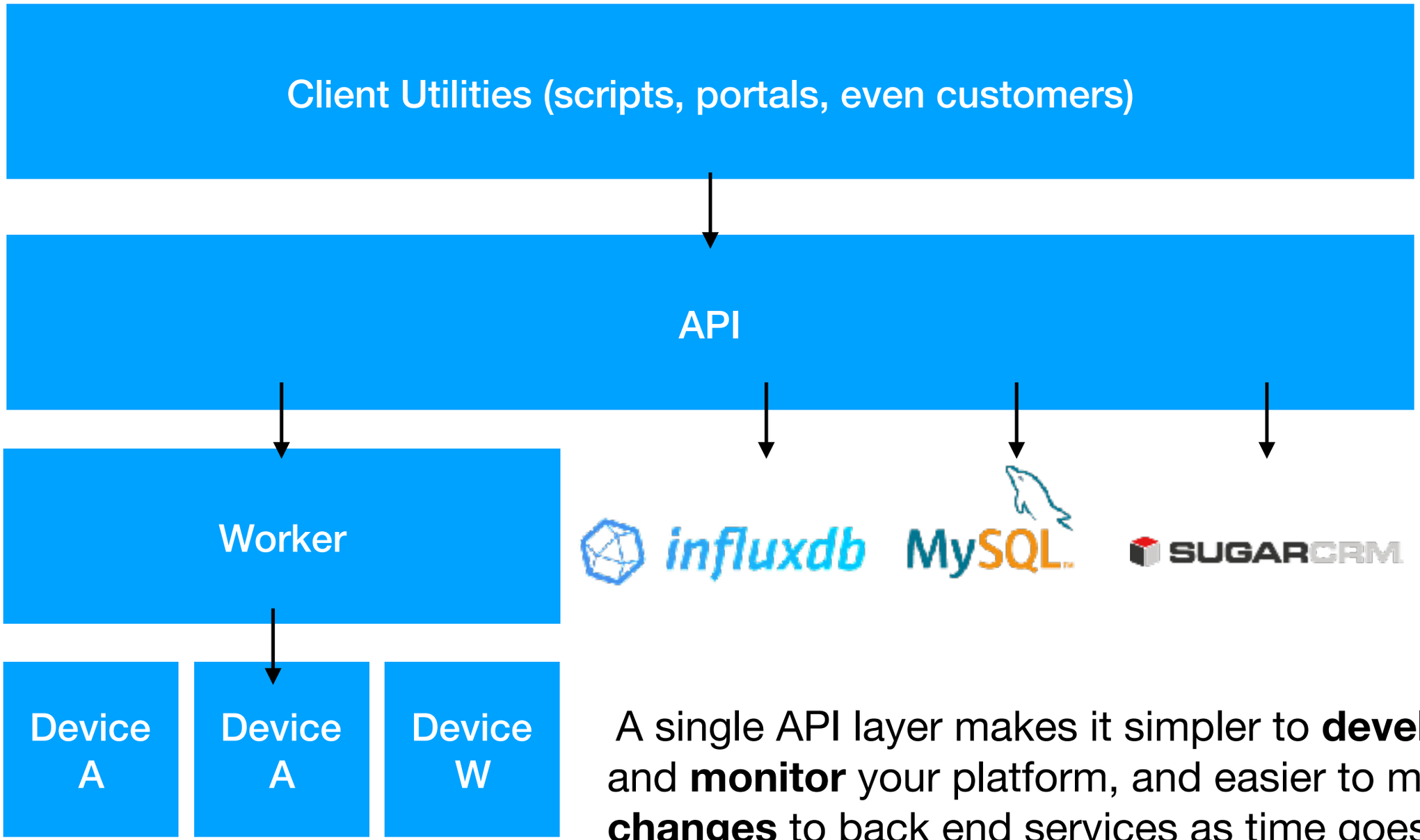
Common Data Stores in Engineering

- SQL - Truths about users, ports, services, ‘state’, e.g. MySQL

- Time Series - e.g. Port utilisation, light level, error count, e.g. InfluxDB

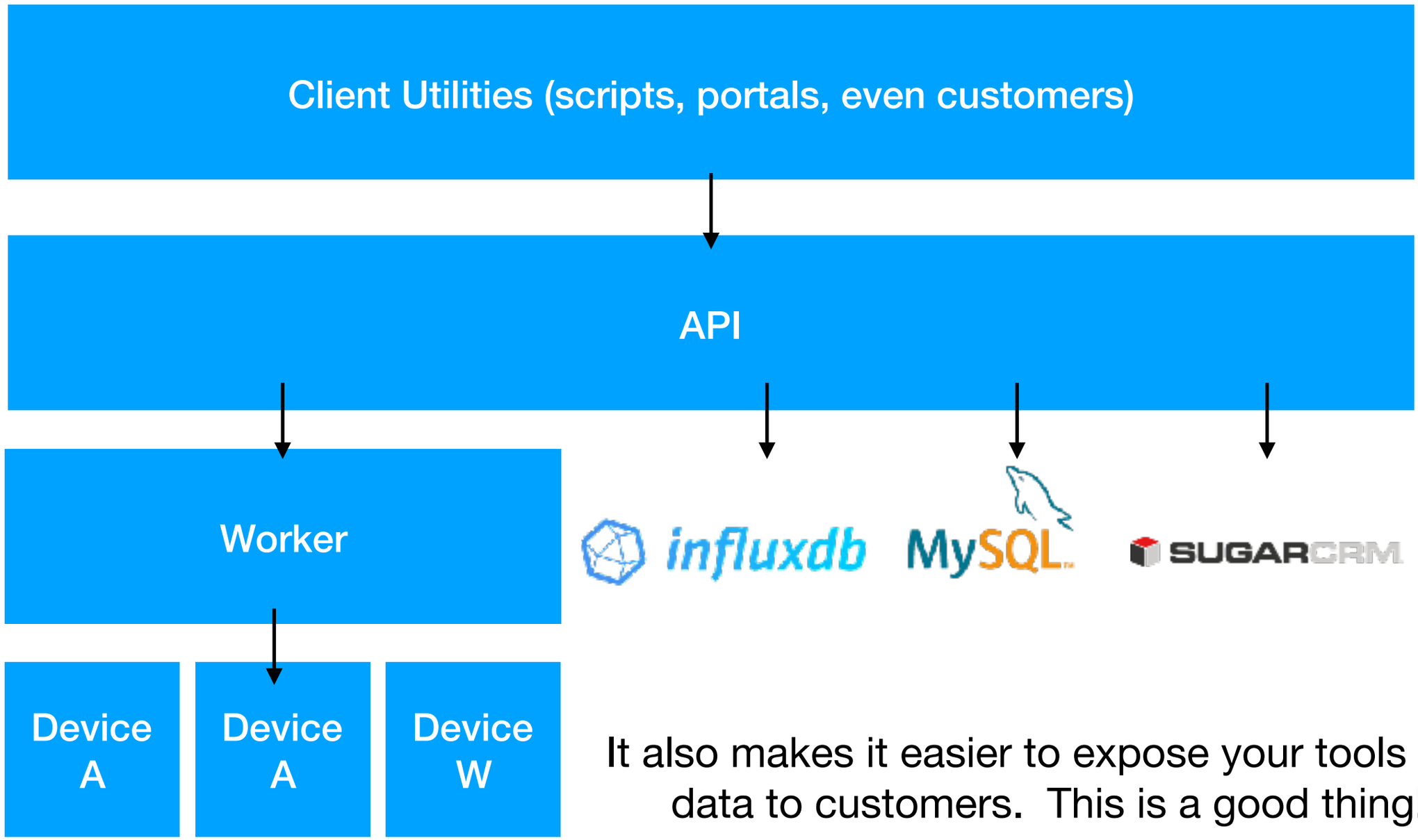
- Third Party - Someone else’s sorted data, e.g. CRM, e.g. EuroIX/PeeringDB

General Architecture



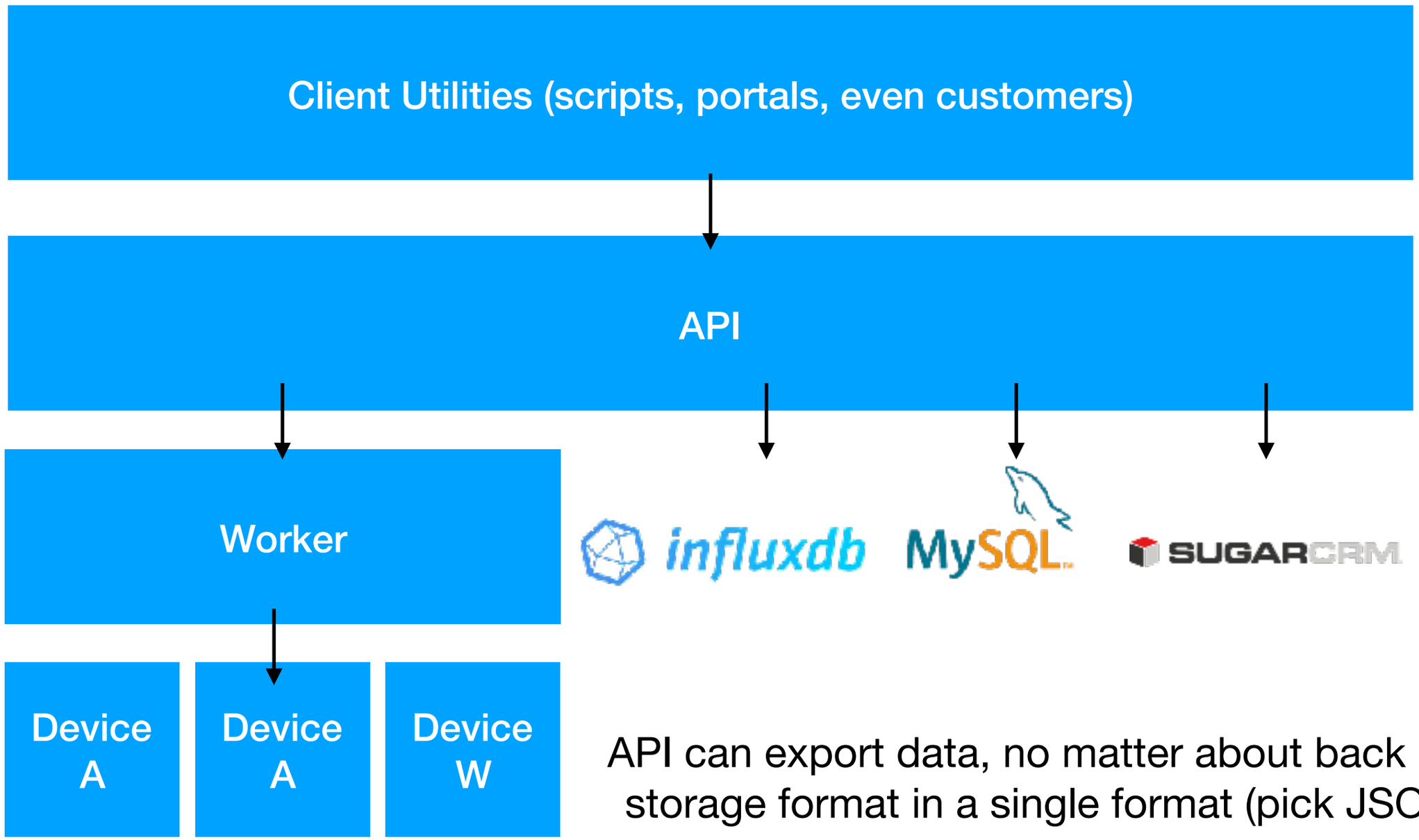
A single API layer makes it simpler to **develop** and **monitor** your platform, and easier to make **changes** to back end services as time goes by

General Architecture



It also makes it easier to expose your tools and data to customers. This is a good thing!

General Architecture



API can export data, no matter about back end storage format in a single format (pick JSON)

Worker, BIRD

Internal SQL


```
{
    "asn":      ,
    "collector_bgp4_last_changed": "2018-01-07 12:39:33",
    "collector_bgp4_observed_time": "2018-01-08 21:10:27",
    "collector_bgp4_routes": 11,
    "collector_bgp4_state": "Established",
    "collector_bgp6_last_changed": "2018-01-07 12:39:40",
    "collector_bgp6_observed_time": "2018-01-08 21:10:27",
    "collector_bgp6_routes": 5,
    "collector_bgp6_state": "Established",
    "exchange_id': 1,
    "exchange_name": "Asteroid Amsterdam Internet Exchange",
    "exchange_shortname": "ams",
    "mac_address':
    "network_id': 7,
    "organisation_id":
    "peering_dnsname":      ,
    "peering_ip6addr": "2001:7f8:b6::      ",
    "peering_ipaddr": "185.1.      ",
    "port_id": 7,
    "route_server_config": 1,
    "routeserver_bgp4_last_changed": "2018-01-08 11:02:08",
    "routeserver_bgp4_observed_time": "2018-01-08 20:45:58",
    "routeserver_bgp4_routes_filtered": 3,
    "routeserver_bgp4_routes_imported": 7,
    "routeserver_bgp4_state": "Established",
    "routeserver_bgp6_last_changed": "2018-01-08 20:34:05",
    "routeserver_bgp6_observed_time": "2018-01-08 20:45:58",
    "routeserver_bgp6_routes_filtered": 0,
    "routeserver_bgp6_routes_imported": 5,
    "routeserver_bgp6_state": "Established",
    "service_id":  ,
    "vlan_8021q_tag": "101",
    "vlan_id": 1,
    "vlan_name': "AMS_PEERING"
}
```


GET https://sputnik.asteroidhq.com/port/2

Pretty Raw Preview JSON


```
    "port_enabled": 1,
    "port_first_link_enable_time": "6th-February-2018 17:29:53",
    "port_id": 2,
    "port_identifier": "Ethernet1/2",
    "port_original_lineitem_id": 1,
    "port_original_quote_id": 1,
    "port_repeatbill_id": 1,
    "port_repeatbill_mrc": 0,
    "port_speed_name": "10000",
    "port_supervisor_lock": 0,
    "port_switch_admin_enable": 1,
    "port_switch_last_flapped_time": "13th-January-2018 00:01:49",
    "port_switch_link_enable": 1,
    "port_switch_observed_time": "2nd-March-2018 14:51:06",
    "port_type": "physical",
    "stats": {
        'backtime': '1w',
        'backtime_days': '7',
        'bps_in': [
            [
                "2018-02-23T14:30:00Z",
                1198.0533333333333,
                1519396200
            ],
            [
                "2018-02-23T15:00:00Z",
                1336.2933333333333,
                1519398000
            ],
            [
                "2018-02-23T15:30:00Z",
                1231.8666666666666,
                1519399800
            ],
            [
                "2018-02-23T16:00:00Z",
```


Worker, Arista

InfluxDB

Models <> Templates

- Once you have confidence in your data model you can harness the power of templated configuration

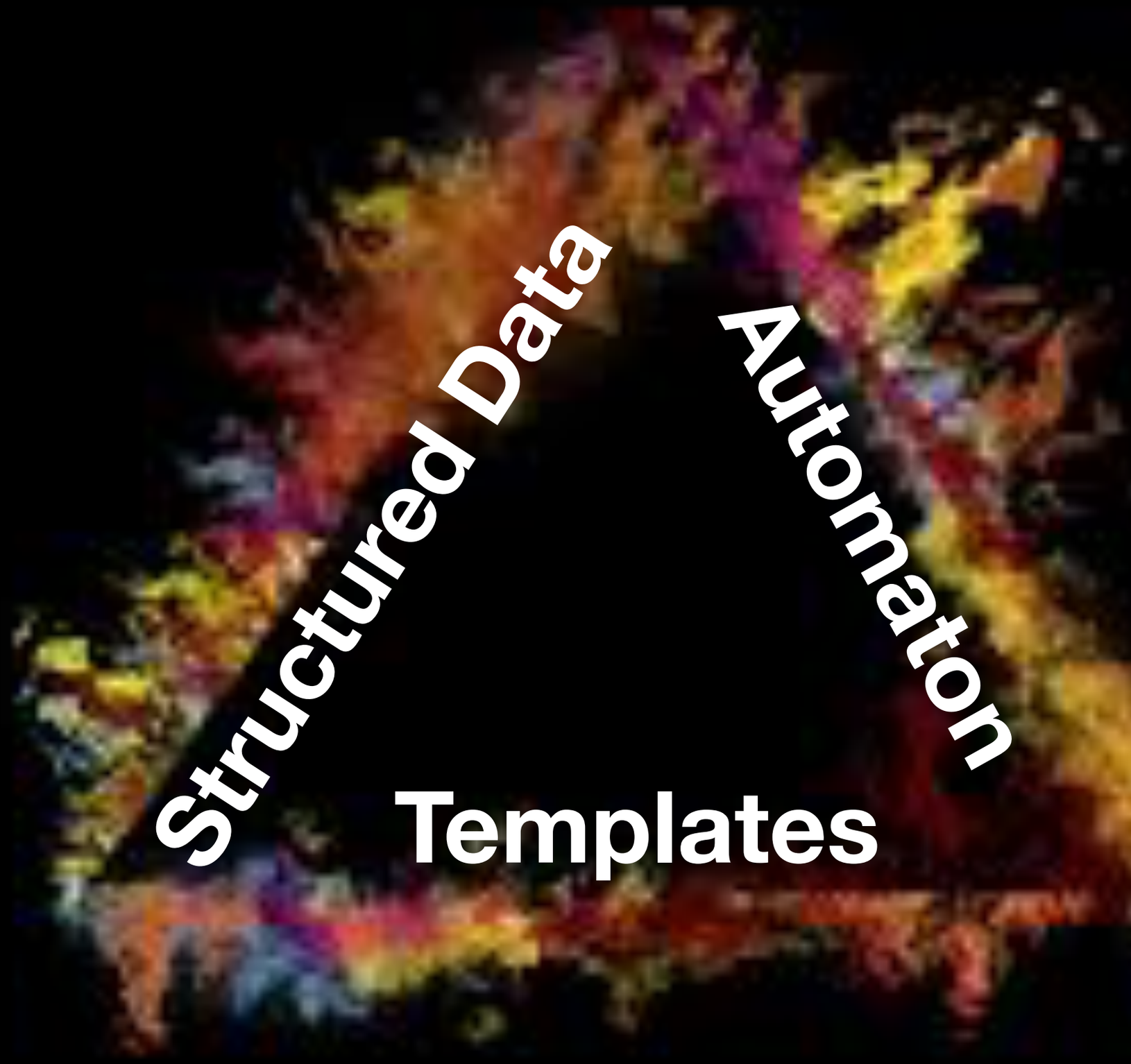
- Once your data model extends across the business you can do that with greater accuracy and devolved control

- e.g. at Asteroid, our sales people can deliver exchange ports directly from the **quotation**

  - So can customers

  - Simultaneous delivery of monitoring from the quotation

Automation fire triangle



Templates - Jinja

```
{% for network in exchange_data.json.member_list %}
### AS{{ network.asnum }}  -  {{ network.name }}

protocol bgp pb_as{{ network.asnum }} {
    description "Collector for AS{{ network.asnum }} - {{ network.name }} ";
    local as collectorasn;
    source address collectoraddress;
    neighbor {{ network["connection_list"][0]["vlan_list"][0]["ipv4"]["address"] }}  as {{ network.asnum }};
    import all;
    export none;
}

{% endfor %}
```

- Generate any kind of configuration
- Takes variables from your JSON API
- Facilitates programatic methods in configuration strophe
  - Loops
  - Conditionals

Automation - Ansible

```
2
3   - name: get route-server settings for this ixp
4     uri: url="https://sputnik.asteroidhq.com/export/euroix/{{ exchange_id }}/participants.json" body=yes
5     register: exchange_data
6     tags:
7       - softconfig
8
9   - name: install bird IPv4 config
10    template: >
11      src=templates/collector4.conf.j2
12      dest=/etc/bird/bird.conf
13      owner=root
14      group=root
15      mode=0640
16      vars:
17        exchange_data: {{ exchange_data }},
18    register: bird4_changed
19    tags:
20      - softconfig
21
22  - name: restart bird (IPv4)
```

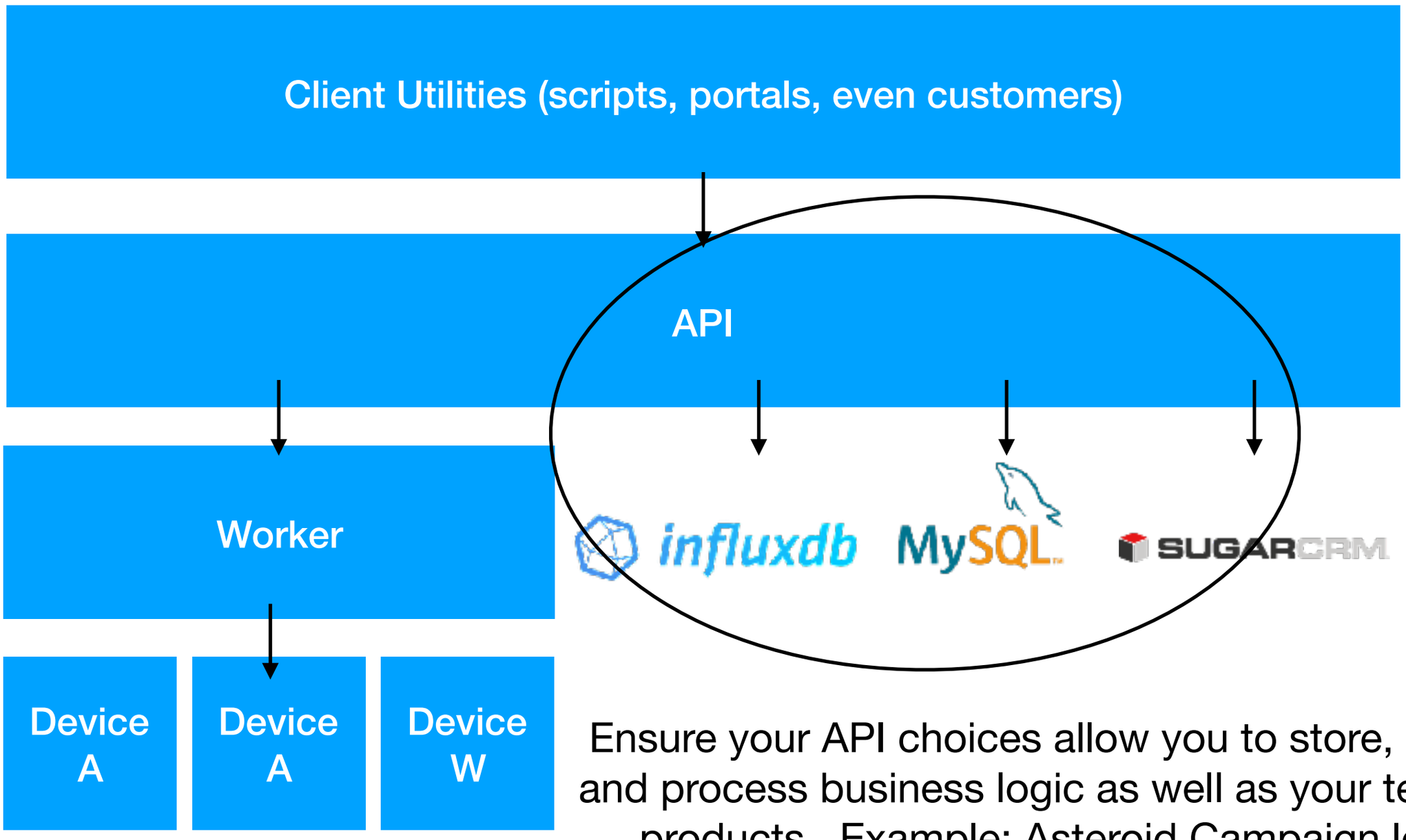
Conditional Logic without script

```
- name: Collect Prefix information
  uri: url='http://46.51.199.18/prefix/{{ item["connection_list"][0]["vlan_list"][0]["ipv4"]["as_macro"] }}' body=yes
  register: prefix
  with_items: "{{ exchange_data['json']['member_list'] }}"
  when: item['connection_list'][0]['vlan_list'][0]['ipv4']['routeserver'] == true
  tags:
    - softconfig
```

Advantages

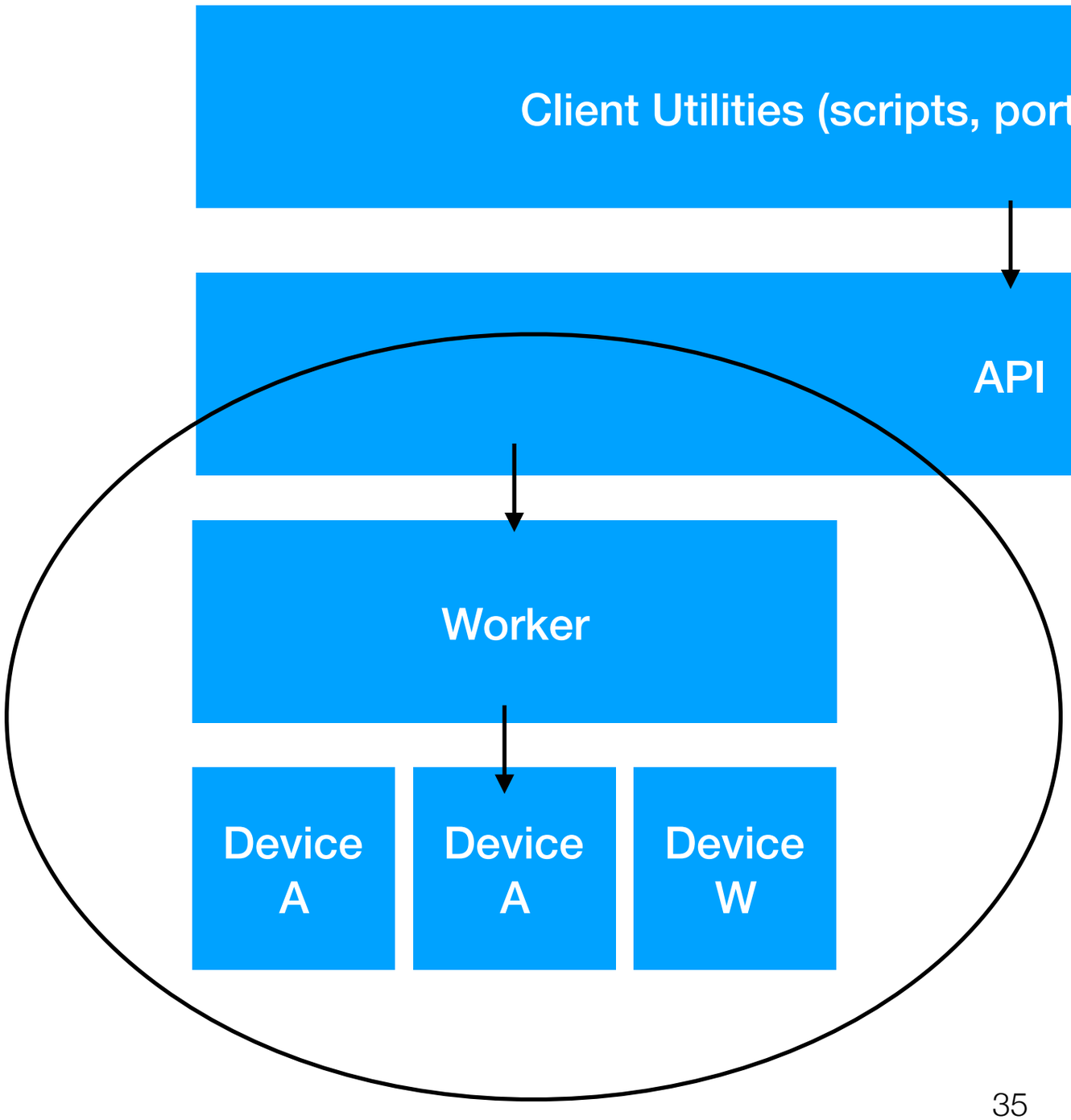
- API layer lightweight
  - Retrieve and update database records
  - Write in a familiar type-safe language (I chose Python)
- Automation layer lightweight
  - Essentially Ansible configuration files
  - Configuration “easier” than coding?

Business Logic



Ensure your API choices allow you to store, retrieve and process business logic as well as your technical products. Example: Asteroid Campaign logic.

Worker Architecture



Must consider:

- Inter Process Communication
- Job and network state
- Device independent
- Vendor failure behaviour
- Device swap-outs

Inter-Process communication

- Message Queue based?
  - e.g. RabbitMQ
  - 👍 Quite good support in major scripting languages
  - 👍 Fault tolerant, order matters, guaranteed delivery, HA
  - 👎 Extra software to support & Centralised
- Web Services
  - 👍 Same technology stack as central API
  - 👍 Inherently extensible
  - 👍 Decentralised
  - 👎 Extra software to write and more state to manage

Device Independence

- I chose to write a different worker per back end technology
- 👎 A bit of copy/paste code, which is an anti-pattern
- 👍 No stress trying to treat different vendors generically
- 👍 NAPALM allows me to continue with Ansible
- 👍 Can swap out a switch/server architecture for sure

Device Swap-outs

- Using Ansible/NAPALM for switch configuration allows a process for rolling full configuration in event of device failure

- No need for specific software feature, an operational process is ok

Software Testing

```
[tatou:test_example andy$ ls
__pycache__     test_example.py
[tatou:test_example andy$ cat test_example.py
#!/usr/bin/env python

import unittest

class UnitTests(unittest.TestCase):
    def test_can_add_oneone(self):
        sum = 1 + 1
        self.assertTrue(sum == 2)
[tatou:test_example andy$ pytest
============================= test session starts ==============================
platform darwin -- Python 2.7.10, pytest-3.0.5, py-1.4.32, pluggy-0.4.0
rootdir: /Users/andy/test_example, inifile:
plugins: cov-2.4.0
collected 1 items

test_example.py .

=========================== 1 passed in 0.01 seconds ===========================
tatou:test_example andy$
```

- Write the test first
- Red, Green, Refactor mantra

Integration vs Unit Testing

```python
class UnitTests(LoginTests):
    def test_regular_user_can_view_own_contact(self):
        result_json = sputnik_get(self, "https://127.0.0.1:5001/contact/6")
        self.assertTrue(result_json["contact_firstname"] == 'Regular')

    def test_regular_user_can_not_view_others_contact(self):
        with self.assertRaises(AuthException):
            result_json = sputnik_get(self, "https://127.0.0.1:5001/contact/3")

    def test_can_not_load_administrator_systems_list_if_not_auth(self):
        with self.assertRaises(AuthException):
            result_json = sputnik_get(self, "https://127.0.0.1:5001/exchange/2/systems")
```

- If you are like me, you will prefer Integration tests
- Write lots, and remember to cover desired exceptions
- Run on your development instance after every change
- “Back to Zero” testing catches unexpected failures

The Joy of Errors

```
tatou:sputnik andy$ pytest
======================================== test session starts ========================================
platform darwin -- Python 2.7.10, pytest-3.0.5, py-1.4.32, pluggy-0.4.0
rootdir: /Users/andy/src/sputnik, inifile:
plugins: cov-2.4.0
collected 188 items

tests/test_api_asteroid.py ..........................................
tests/test_api_whitelabel.py ........................................
tests/test_regular.py ...............................................
tests/test_unit.py ..............................................F..............................

============================================= FAILURES ==============================================
_____________________________ UnitTests.test_can_load_organisations_quotes ______________________________

self = <test_unit.UnitTests testMethod=test_can_load_organisations_quotes>

    def test_can_load_organisations_quotes(self):
        ql = QuoteList()
        testquote = ql.list_quotes_organisation(2)[0]
        self.assertTrue(testquote["quote_currency"] == "EUR")
>       self.assertTrue(testquote["quote_lineitems"][0]["quote_lineitem_desc"] == "10Gbit Peering Port, Stellar Internet Exchange")
E       IndexError: tuple index out of range

tests/test_unit.py:391: IndexError
================================== 1 failed, 187 passed in 5.82 seconds ==================================
```

Ubiquity of JSON for an ISP

- Especially in Peering!
- PeeringDB
- Euro-IX IXF-DB
- Asteroid JSON


```
{
    "bridge_hostname": "ams-sw1",
    "demarcation": {
        "crossconnect_paying_organisation_id": null,
        "crossconnect_ref": null,
        "demarcation_id": 2,
        "demarcation_label": "1/1/3 and 4",
        "port_id": 2,
        "rack_location": "Nikhef rack 108"
    },
    "exchange_id": 1,
    "exchange_main_peering_vlan_id": 1,
    "exchange_name": "Asteroid Amsterdam Internet Exchange",
    "exchange_product_id": 2,
    "exchange_product_name": "10Gbit Peering Port",
    "exchange_tag": "ams",
    "light_level": {
        "latest_power_db": -2.8215313251307437
    },
    "organisation_id": 3,
    "organisation_name": "Slashme BV",
    "outstanding_work": [],
    "peering_service": {
        "mac_address": null,
        "peering_dnsname": "asteroid-amsterdam.slashme.org",
        "peering_ip6addr": "2001:7f8:b6::63fb:1",
        "peering_ipaddr": "185.1.34.10",
        "route_server_config": 1,
        "service_id": 4,
        "vlan_id": 1,
        "vlan_number": "101"
    },
    "permit_config_change": true,
    "port_bundle_group": null,
    "port_bundletype": "none",
    "port_customer_status": "Live",
    "port_description": "CUST::Slashme BV",
    "port_enabled": 1,
    "port_first_link_enable_time": "6th-February-2018 17:29:53",
    "port_id": 2,
    "port_identifier": "Ethernet1/2",
    "port_original_lineitem_id": 1,
    "port_original_quote_id": 1,
    "port_repeatbill_id": 1,
    "port_repeatbill_mrc": 0,
    "port_speed_name": "10000",
    "port_supervisor_lock": 0,
    "port_switch_admin_enable": 1,
    "port_switch_last_flapped_time": "13th-January-2018 00:01:49",
    "port_switch_link_enable": 1,
    "port_switch_observed_time": "2nd-March-2018 14:51:05",
    "port_type": "physical",
    "stats": {
        "backtime": "1w",
        "backtime_days": "7",
        "bps_in": [
            [
                "2018-02-23T14.30.00Z",
                198.0533333333333,
                1519396200
            ],
            [
                "2018-02-23T15:00:00Z"
```

Summary

- Single source of truth under the control of all departments
- Which is used to configure services and network
- Accessible to all departments
- Customer self service
- Provision from quote
- “Information in one place and tool”
- Account Managers can do troubleshooting

Any Questions?

Andy Davidson <andy@asteroidhq.com>
www.asteroidhq.com