Introduction

Ravindra Kudache

Who ?

G-Research

"G-Research is a leading quantitative research and technology company. By using the latest scientific techniques, we produce world-beating predictive research and build advanced technology to analyse the world's data".

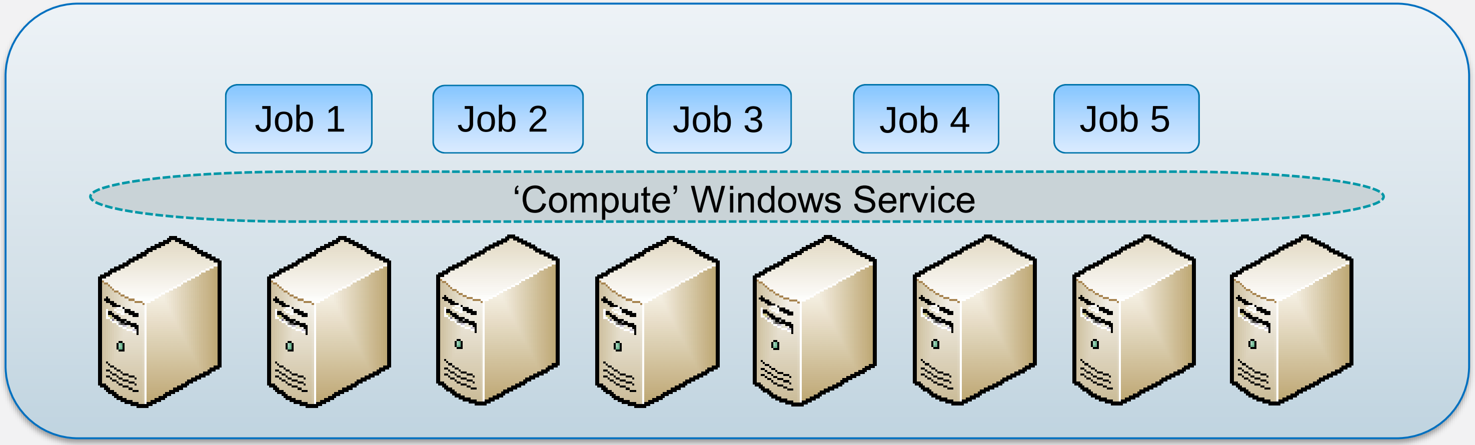
Windows Engineering

Windows Engineering are responsible for automated Windows provisioning, software delivery and configuration management across the estate. We strongly adhere to the principles of IaC (Infrastructure-as-Code).

What ?

These 1000 physical windows servers makeup a High-Throughput Computing System, which we refer to as the ‘Farm’. We have multiple farms hosting tens of thousands of servers and this is just one small subset.

The Farm provides compute resource that allows for many computational tasks to be completed over a long period of time. Mechanisms in the farm include; job queueing, scheduling, prioritizing and resource management. Nodes are predominantly **‘stateless’**

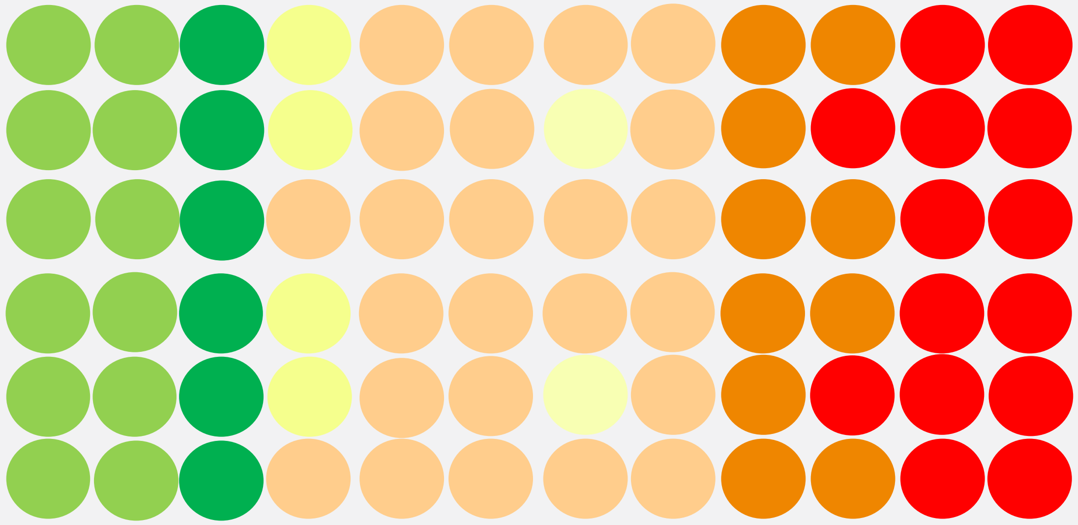


Why ?

< 30 Days

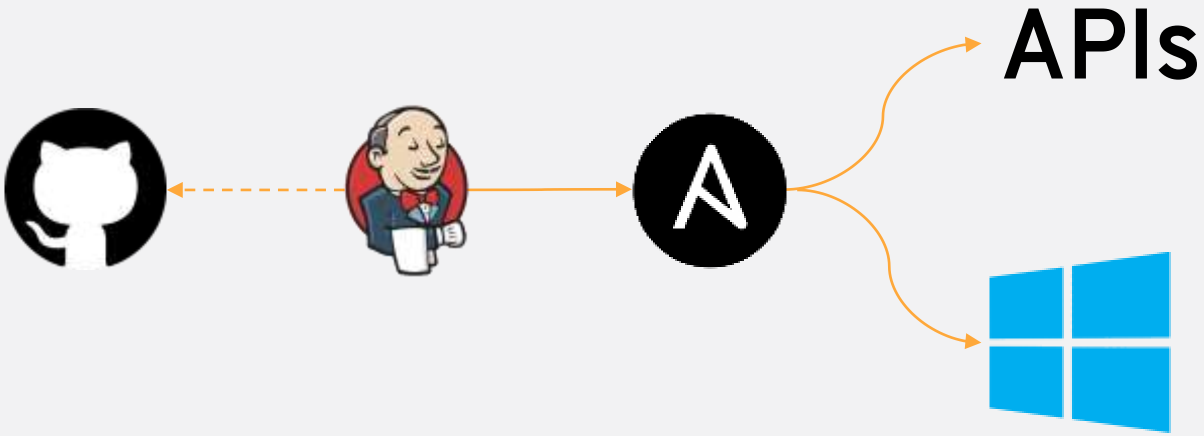
- Our objective is to have no nodes in the farm older than **30 days**
- This prevents the build up of unintentional artefacts on older nodes
- This allows frequent updating of software
- Servers do not require patching
- The automated mechanism allows for more consistent node builds
- Reduce manual intervention and time spent on farm builds (5 teams)
- Reduce troubleshooting time.

The Problem

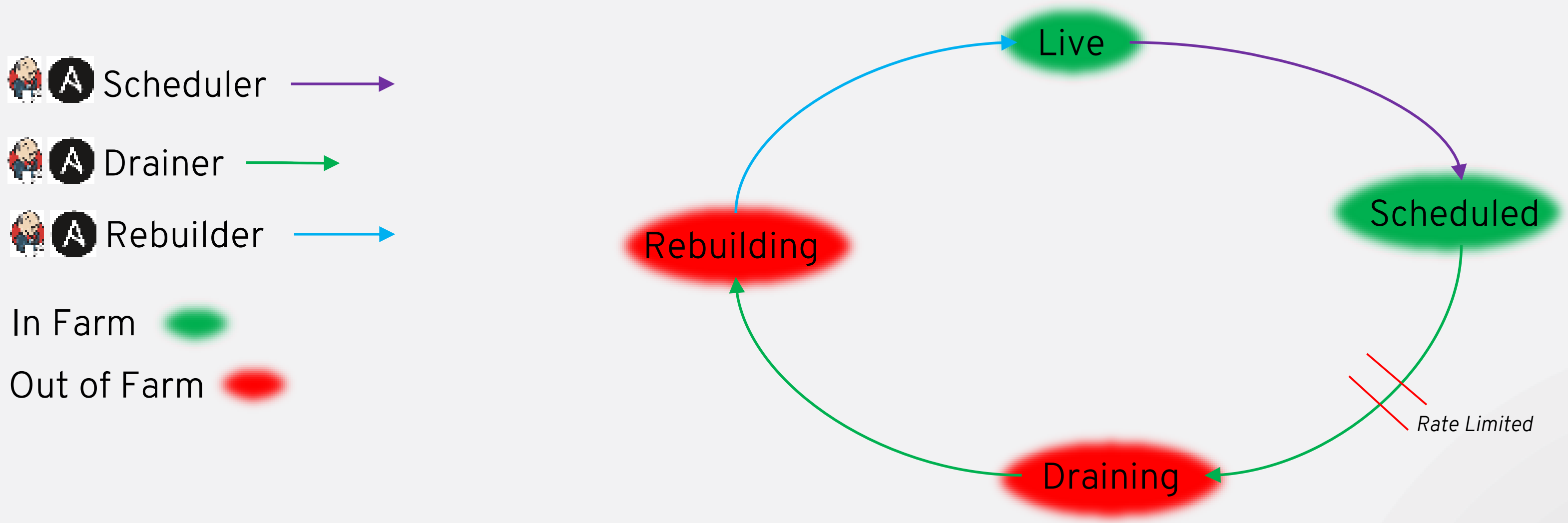


The Rebuild Process

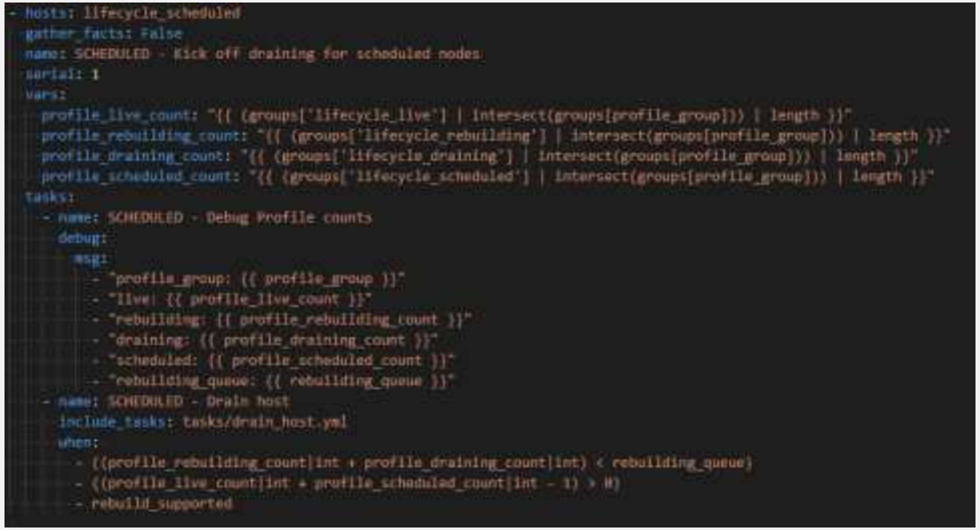
Process Overview



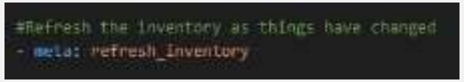
Process Overview



Process Overview - Drainer



```yaml
- hosts: lifecycle_scheduled
  gather_facts: False
  name: SCHEDULED - Kick off draining for scheduled nodes
  serial: 1
  vars:
    profile_live_count: "{{ (groups['lifecycle_live'] | intersect(groups[profile_group])) | length }}"
    profile_rebuilding_count: "{{ (groups['lifecycle_rebuilding'] | intersect(groups[profile_group])) | length }}"
    profile_draining_count: "{{ (groups['lifecycle_draining'] | intersect(groups[profile_group])) | length }}"
    profile_scheduled_count: "{{ (groups['lifecycle_scheduled'] | intersect(groups[profile_group])) | length }}"
  tasks:
    - name: SCHEDULED - Debug Profile counts
      debug:
        msg:
          - "profile_group: {{ profile_group }}"
          - "live: {{ profile_live_count }}"
          - "rebuilding: {{ profile_rebuilding_count }}"
          - "draining: {{ profile_draining_count }}"
          - "scheduled: {{ profile_scheduled_count }}"
          - "rebuilding_queue: {{ rebuilding_queue }}"
    - name: SCHEDULED - Drain host
      include_tasks: tasks/drain_host.yml
      when:
        - ((profile_rebuilding_count|int + profile_draining_count|int) < rebuilding_queue)
        - ((profile_live_count|int + profile_scheduled_count|int - 1) > 0)
        - rebuild_supported
```



```yaml
#Refresh the inventory as things have changed
- meta: refresh_inventory
```

Process Overview

Live -> Scheduled
- Used to see the overall health of the process
- All nodes ending up in scheduled indicates rate issues

Scheduled -> Draining
- Rate limited by the number of nodes in Draining and Rebuilding
- Uses ansible inventory to calculate if transition is possible

Draining -> Rebuilding
- Checks whether jobs are still running on the node
- Triggers external “Rebuilder” Jenkins job/playbook

Rebuilding -> Live
- Processed by generic rebuild process on a fixed version
- Time taken only needs to be of the order of draining time

Inventory Overview

Overall Inventory

CMDB

	Live	Scheduled	Draining	Rebuilding
Profile A	1	1	1	1
Profile B	1	1	1	1
Profile C	1	1	1	1

Compute Farm

Profile A	Profile B	Profile C	Profile D
4	4	4	4

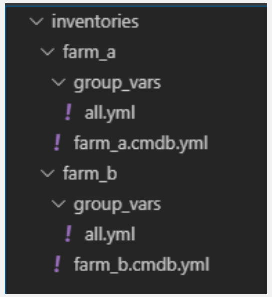
Ansible Inventory V1

CMDB

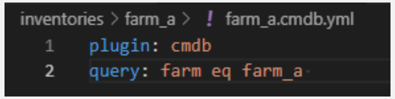
Live	Scheduled	Draining	Rebuilding
4	4	4	4

Ansible Inventory V1

Inventory Files



```
inventories
  farm_a
    group_vars
      all.yml
    farm_a.cmdb.yml
  farm_b
    group_vars
      all.yml
    farm_b.cmdb.yml
```



inventories > farm_a > farm_a.cmdb.yml

```yaml
plugin: cmdb
query: farm eq farm_a
```

Issues:

- All Compute-Farm facts had to be calculated per node post inventory
- Each nodes awareness of other nodes had to be calculated separately
- Calculation of compute farm groups was done in YAML 'code'
- Setting variables per 'profile' had to be done post inventory
- Playbooks were hard to follow

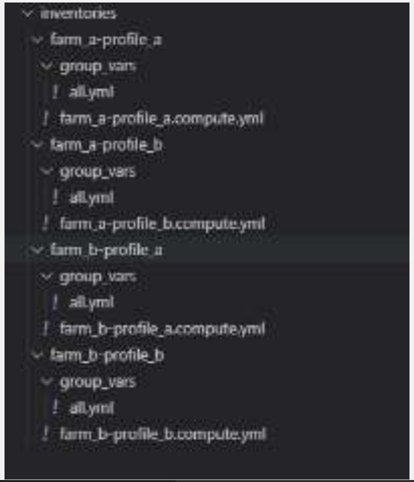
Ansible Inventory V2

CMDB + Compute Farm

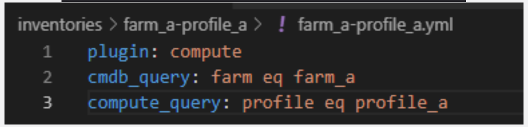
	Live	Scheduled	Draining	Rebuilding
Profile A	1	1	1	1

Ansible Inventory V2

Inventory Files



```
v inventories
  v farm_a-profile_a
    v group_vars
      ! all.yml
    ! farm_a-profile_a.compute.yml
  v farm_a-profile_b
    v group_vars
      ! all.yml
    ! farm_a-profile_b.compute.yml
  v farm_b-profile_a
    v group_vars
      ! all.yml
    ! farm_b-profile_a.compute.yml
  v farm_b-profile_b
    v group_vars
      ! all.yml
    ! farm_b-profile_b.compute.yml
```



inventories > farm_a-profile_a > ! farm_a-profile_a.yml

```yaml
plugin: compute
cmdb_query: farm eq farm_a
compute_query: profile eq profile_a
```

Advantages

- Simple to get number of nodes out of farm
- Possible to set per profile variables
- Ansible playbooks are much simpler
- Easier for external teams to contribute
- Able to use “refresh_inventory” to update node states

Issues:

- Each farm requires multiple playbooks runs via Jenkins to cover the whole farm
- Setting per general profile variables has to be done in every inventory
- Setting up support for new profiles is still complex

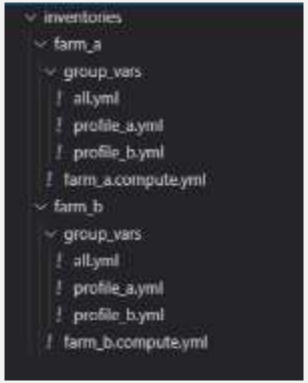
Ansible Inventory V3

CMDB + Compute Farm

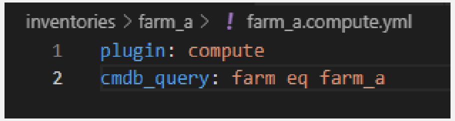
	Live	Scheduled	Draining	Rebuilding
Profile A	1	1	1	1
Profile B	1	1	1	1
Profile C	1	1	1	1
Profile D	1	1	1	1

Ansible Inventory V3

Inventory Files

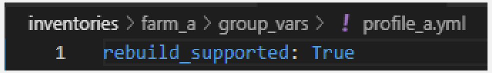


```
inventories
  farm_a
    group_vars
      all.yml
      profile_a.yml
      profile_b.yml
    farm_a.compute.yml
  farm_b
    group_vars
      all.yml
      profile_a.yml
      profile_b.yml
    farm_b.compute.yml
```



inventories > farm_a > ! farm_a.compute.yml

```yaml
plugin: compute
cmdb_query: farm eq farm_a
```



inventories > farm_a > group_vars > ! profile_a.yml

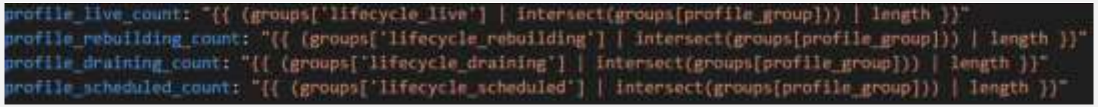
```yaml
rebuild_supported: True
```

Advantages

- Simple to add profiles
- Single playbook run per farm
- All nodes included even when not rebuild supported
- Even easier for external teams to contribute

Issues:

- Getting the profile + lifecycle cross section is more complex:



```yaml
profile_live_count: "{{ (groups['lifecycle_live'] | intersect(groups[profile_group])) | length }}"
profile_rebuilding_count: "{{ (groups['lifecycle_rebuilding'] | intersect(groups[profile_group])) | length }}"
profile_draining_count: "{{ (groups['lifecycle_draining'] | intersect(groups[profile_group])) | length }}"
profile_scheduled_count: "{{ (groups['lifecycle_scheduled'] | intersect(groups[profile_group])) | length }}"
```

Logging

Ansible:
- Logs playbook, play, and task results to splunk via a callback plugin
- Includes timing information
- Sends chat messages on error

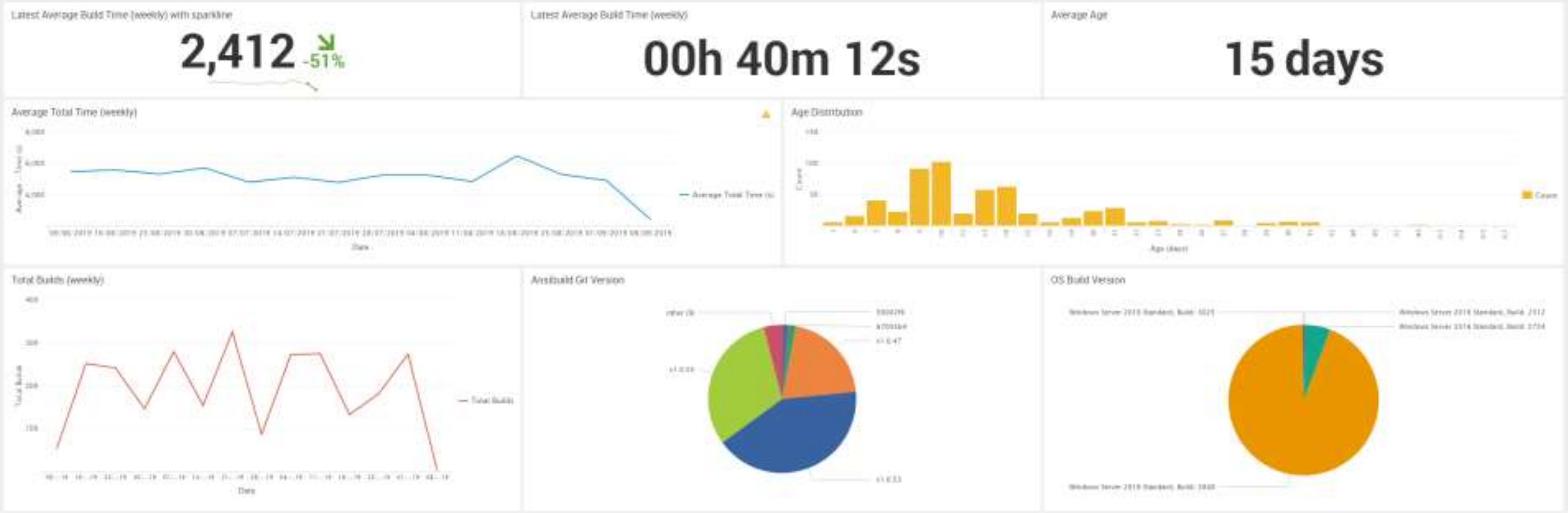
Jenkins:
- Full playbook output retained for 30 days
- Job meta information logged to splunk

Compute Farm:
- Snapshot of node states are logged to splunk

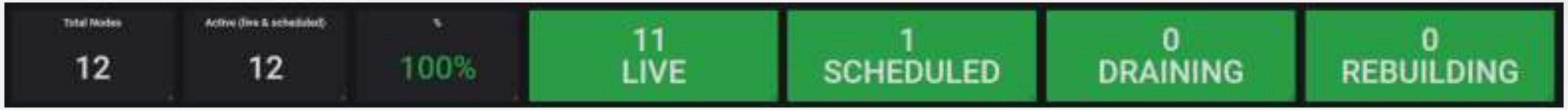
Compute Nodes:
- Splunk forwarder logs inventory information, including image version

CMDB:
- Snapshot of node states are logged to splunk

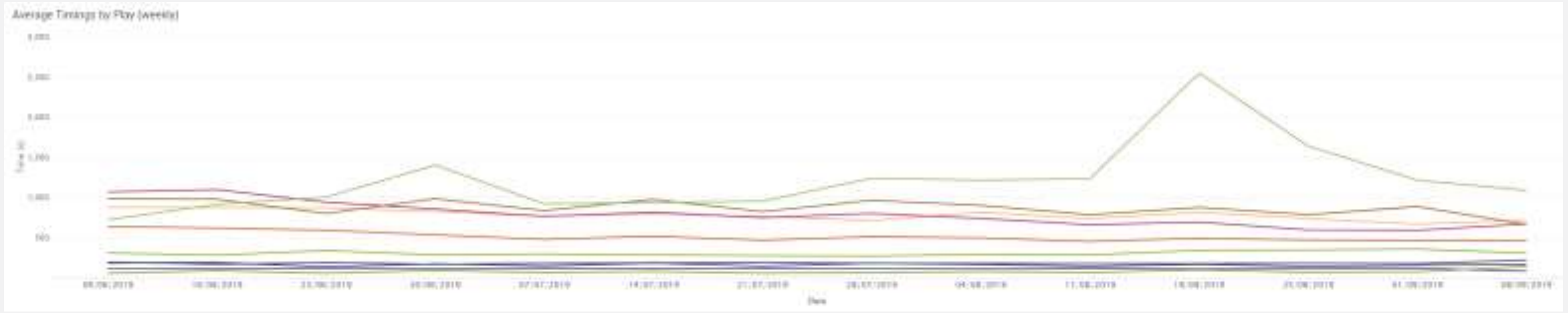
Logging - Views



Logging - Views

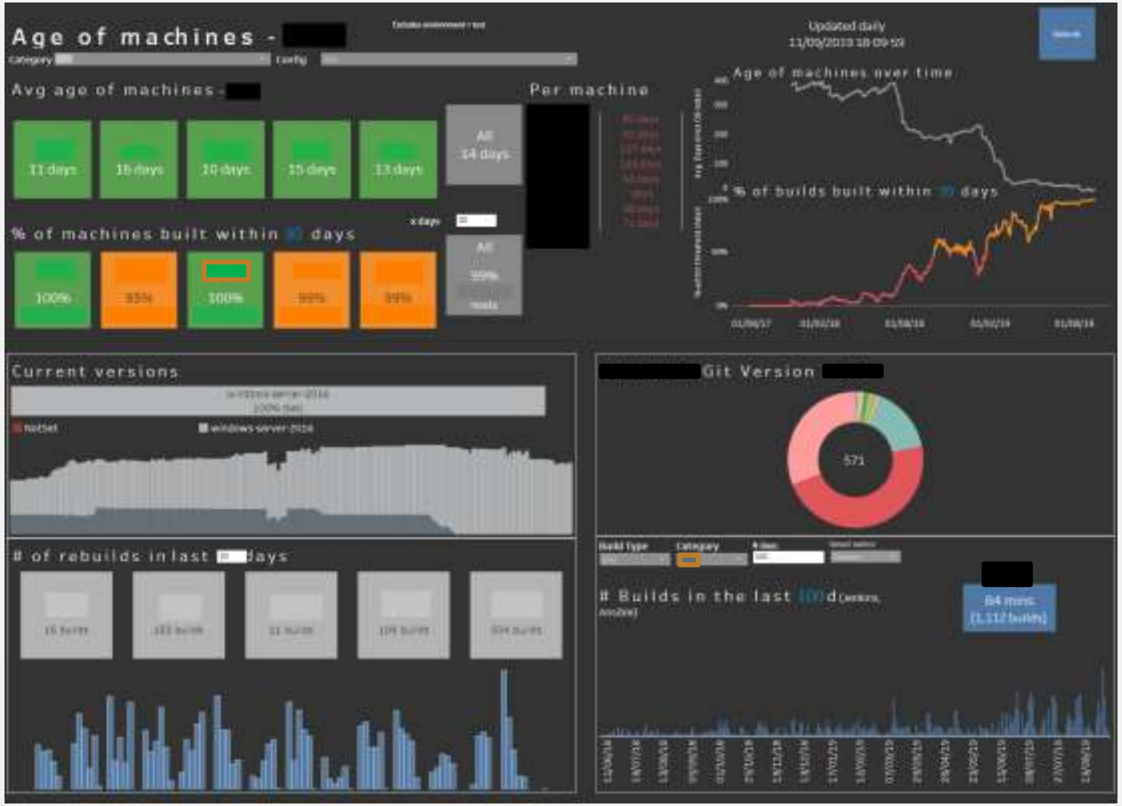


Total Nodes	Active (live & scheduled)	%				
12	12	100%	11 LIVE	1 SCHEDULED	0 DRAINING	0 REBUILDING



Results

Results - Deployment

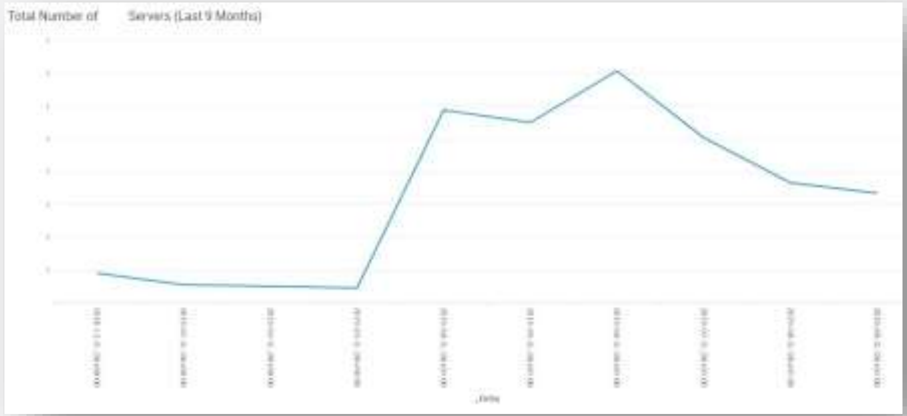


Result - Builds

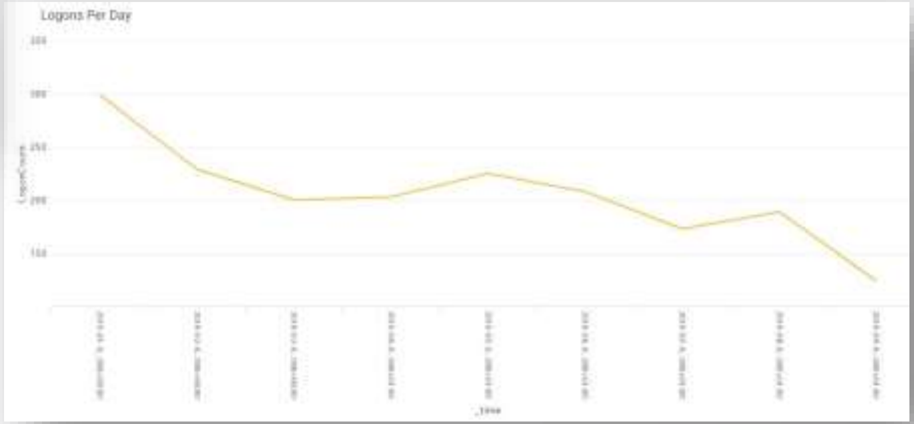
	Pre	Post
Teams Involved	5	1
Manual steps	12	2
Build Time (includes test and into 'Farm')	1-2 days	90mins

Results - Logons

Server count



Logons



Result – Operational

- Servers do not require patching
- Software deployment within 30 days (.net)
- Nodes rebuilt instead of troubleshooting
- Teams can focus on other priorities now build, troubleshooting and management have been reduced.
- Fewer snowflakes !

Lessons Learnt

Project Design

- Consider support and external contributors
- Use modular tasks and roles
- Good inventory is critical for simple playbooks