Using Django for (stop gap) network automation management

Ed Crewe

What is ideal network automation?

It is full software defined networking...

Central master control plane - device agents

Central master config & state database -
device discovery and / or subscription

Auto conformance of live state to config

Management plane - open config markup

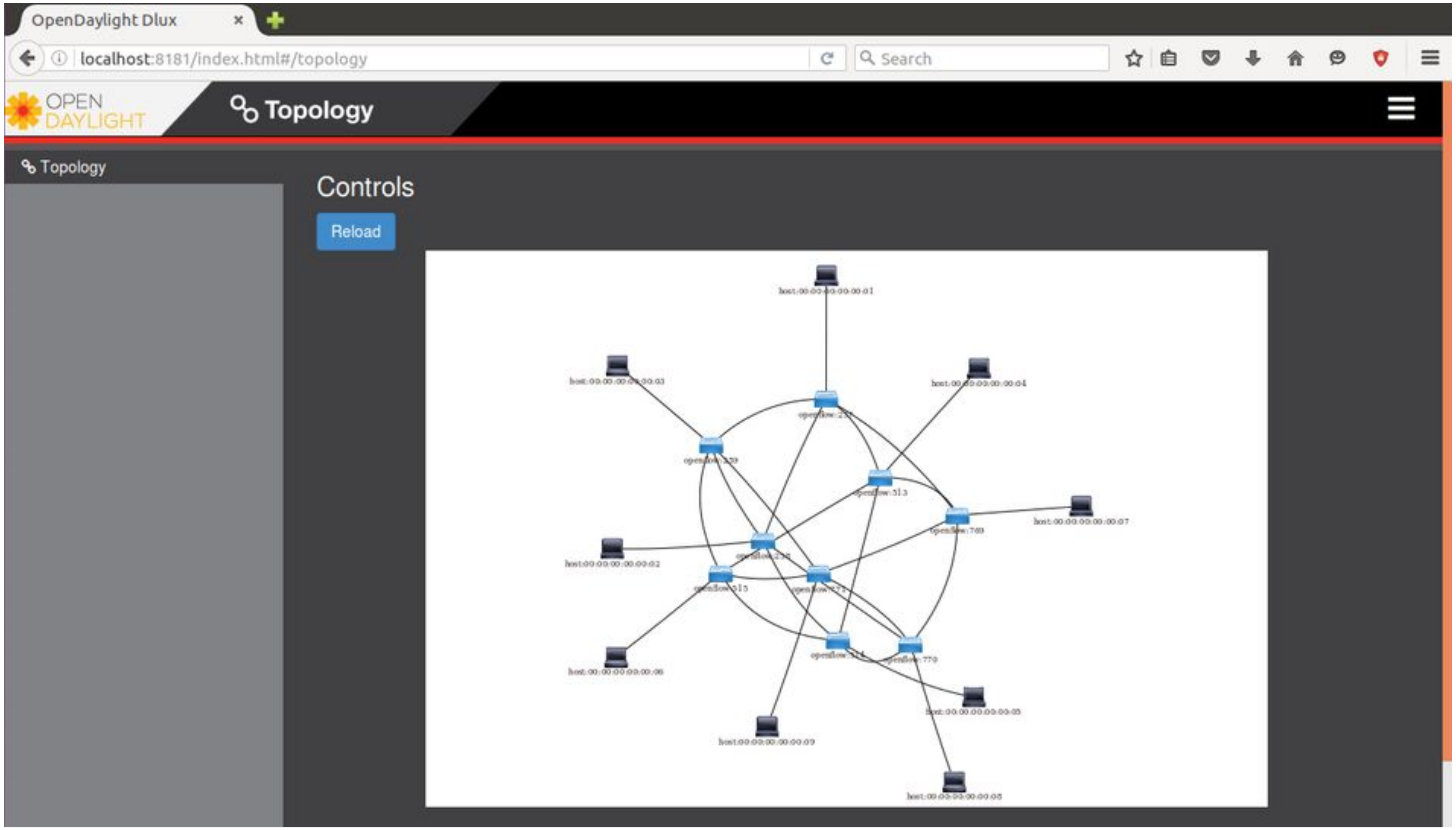
Zero touch provisioning of newly added devices

Dynamic topology generation and reconfiguration

GUI modelling of network topology to live state

Telemetry / status reporting - triggered reconfiguration

Open Daylight - leaf and spine network



The stop gap solution

Fairly static topology generation based on supplied seed hardware metadata

Dedicated **network deployment server (nds)** per leaf and spine unit - receives new config and push code as rpms

Ansible playbooks from rpms pushes vendor specific config to devices

Cronjob for regular reconformance of network

Logging of all push / commit errors from devices

Web GUI to reconcile config artifact (rpm) versions to device push errors

REST API to allow move to more centralised management via nds

Network / NDS status query, reporting and monitoring

Pros and Cons

Few direct dependencies

Simplicity

Isolation / security

Performance - low latency of device control plane

Scalability - controller load

Interoperability - vendor config for older devices

Robust - if nds dissapears devices still run

Backwards compatibility

Poor central control

No central telemetry of network state

No dynamic configuration

No self healing

Vendor specific code

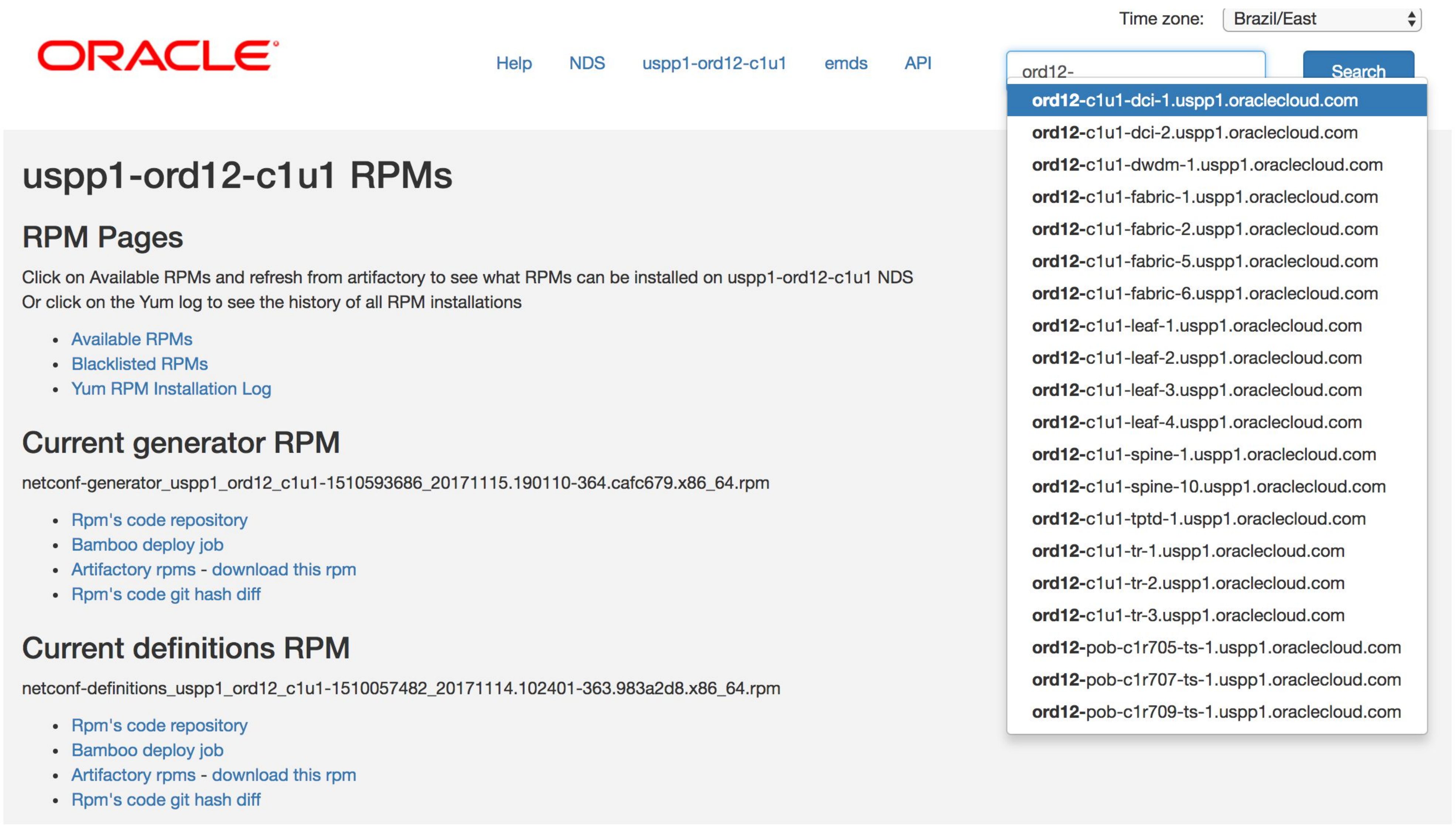
Redhat family specific

Reliant on build & artifact services

No central data, audit or control

Poor data persistence (reliance on nds logs)

Django NDS web



- ord12-c1u1-dci-1.uspp1.oraclecloud.com
- ord12-c1u1-dci-2.uspp1.oraclecloud.com
- ord12-c1u1-dwdm-1.uspp1.oraclecloud.com
- ord12-c1u1-fabric-1.uspp1.oraclecloud.com
- ord12-c1u1-fabric-2.uspp1.oraclecloud.com
- ord12-c1u1-fabric-5.uspp1.oraclecloud.com
- ord12-c1u1-fabric-6.uspp1.oraclecloud.com
- ord12-c1u1-leaf-1.uspp1.oraclecloud.com
- ord12-c1u1-leaf-2.uspp1.oraclecloud.com
- ord12-c1u1-leaf-3.uspp1.oraclecloud.com
- ord12-c1u1-leaf-4.uspp1.oraclecloud.com
- ord12-c1u1-spine-1.uspp1.oraclecloud.com
- ord12-c1u1-spine-10.uspp1.oraclecloud.com
- ord12-c1u1-tptd-1.uspp1.oraclecloud.com
- ord12-c1u1-tr-1.uspp1.oraclecloud.com
- ord12-c1u1-tr-2.uspp1.oraclecloud.com
- ord12-c1u1-tr-3.uspp1.oraclecloud.com
- ord12-pob-c1r705-ts-1.uspp1.oraclecloud.com
- ord12-pob-c1r707-ts-1.uspp1.oraclecloud.com
- ord12-pob-c1r709-ts-1.uspp1.oraclecloud.com

uspp1-ord12-c1u1 RPMs

RPM Pages

Click on Available RPMs and refresh from artifactory to see what RPMs can be installed on uspp1-ord12-c1u1 NDS
Or click on the Yum log to see the history of all RPM installations

- Available RPMs
- Blacklisted RPMs
- Yum RPM Installation Log

Current generator RPM

netconf-generator_uspp1_ord12_c1u1-1510593686_20171115.190110-364.cafc679.x86_64.rpm

- Rpm's code repository
- Bamboo deploy job
- Artifactory rpms - download this rpm
- Rpm's code git hash diff

Current definitions RPM

netconf-definitions_uspp1_ord12_c1u1-1510057482_20171114.102401-363.983a2d8.x86_64.rpm

- Rpm's code repository
- Bamboo deploy job
- Artifactory rpms - download this rpm
- Rpm's code git hash diff

Component applications

Network App

1. netconf-generator - creates configs and pushes via ansible
2. netconf-definitions - provides the seed hardware metadata

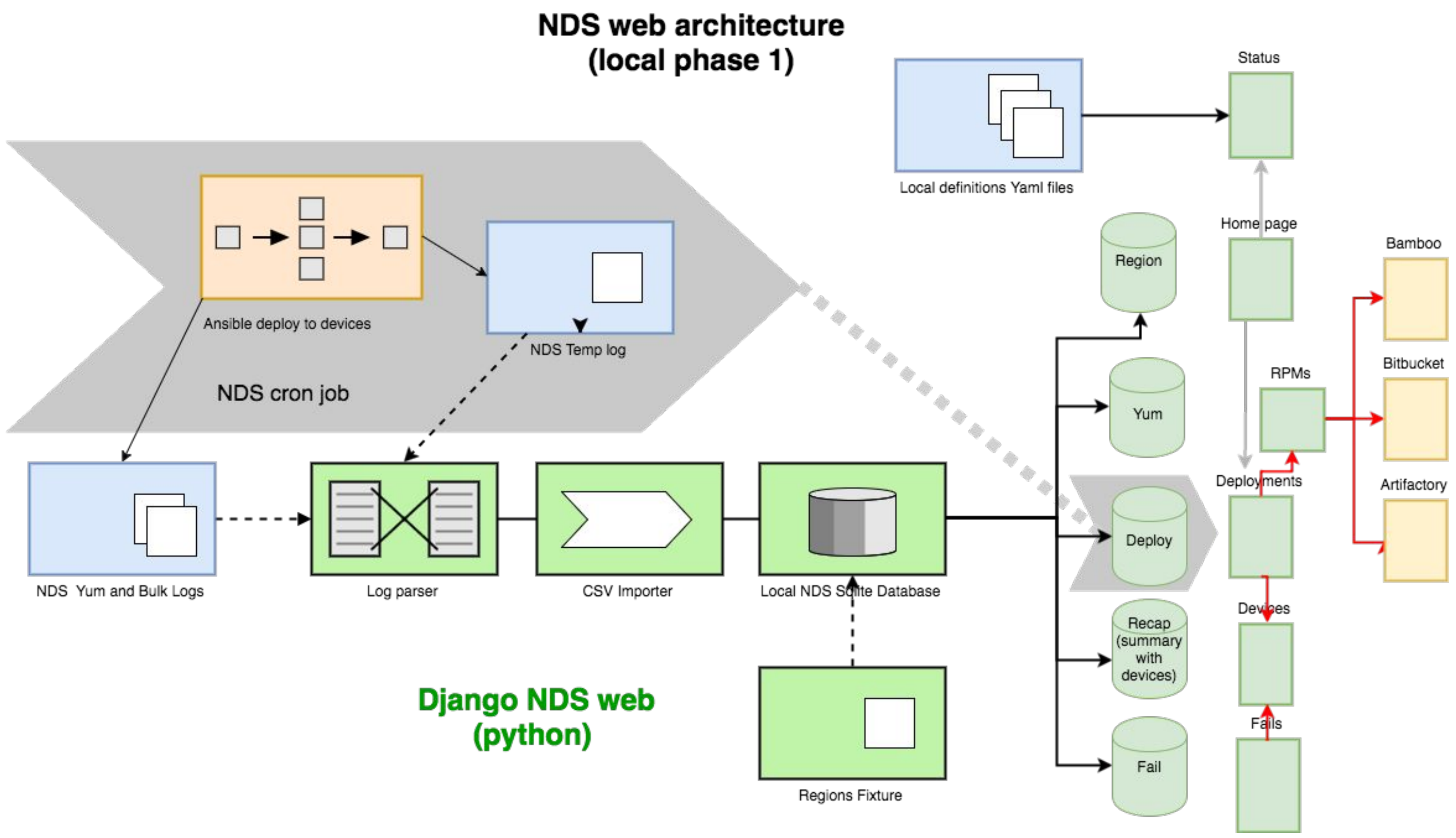
Web Core

1. ndsweb - core app
2. ndsrest - rest api
3. ndsregion - NDS / leaf and spine unit data

Web Apps

1. ndslog - Ansible push, device error & yum rpm logs
2. emds - job scheduler for device tasks eg. reporting
3. rpmblacklist - software / configuration artifact rollback
4. ndscurrentstate - config status
5. ndsmonitor - server status

Architecture diagram



Netconf config generation

Regular push conformance of network topology

Build step (on build servers)

- Code takes seed hardware data in yaml validates it into a temporary database and generates the output topology as yaml
- Input topology yaml for Ansible tasks with templates for generating vendor specific leaf & spine device configs, along with dhcp etc.
- Create push code and config rpm artifacts for deploy to NDS
- Artifact deployment is logged (NDS yum rpm log)

Push step (on NDS)

- Run through the config pushes for all the devices in the unit
- (Juniper / Cisco) Devices have their own internal config database with commit checking of config changes.

Ansible push

- Ansible is agent less (SSH based) generic config management
- Uses playbooks of tasks (manifests of modules)
- Uses Python vendor specific modules for optimal config features / compatibility - eg. Juniper’s junos_eznc
- Pushes out to each device in parallel to check config, update, commit check and commit
- Device commit failures or other errors reported by devices are logged by Ansible
- Final summary status of full topology conformance is logged

Log parsing

Purpose is to link together topology generation code, config and deployment repositories with end device errors to allow easy debugging by network engineers
(Error causes = network changes outside of the unit, load issues and other infrastructure issues as well as hardware failure)

- Parses standard Ansible CLI style output … so non-log format, turning it into a series of logs
- Uses django-csvimport in bulk load mode to split data into various log related models / tables
- Also loads rpm artifact log - artifacts use a naming convention to allow easy matching to Bamboo deploys, Artifactory object versions and original Git source code changes of generator code and configs

Build artifacts / deploy server integration

If errors occur due to code or configuration changes can be reverted via an artifact blacklisting app.

Integrates with the Bamboo build server which provides a list of available generator artifacts from artifactory.

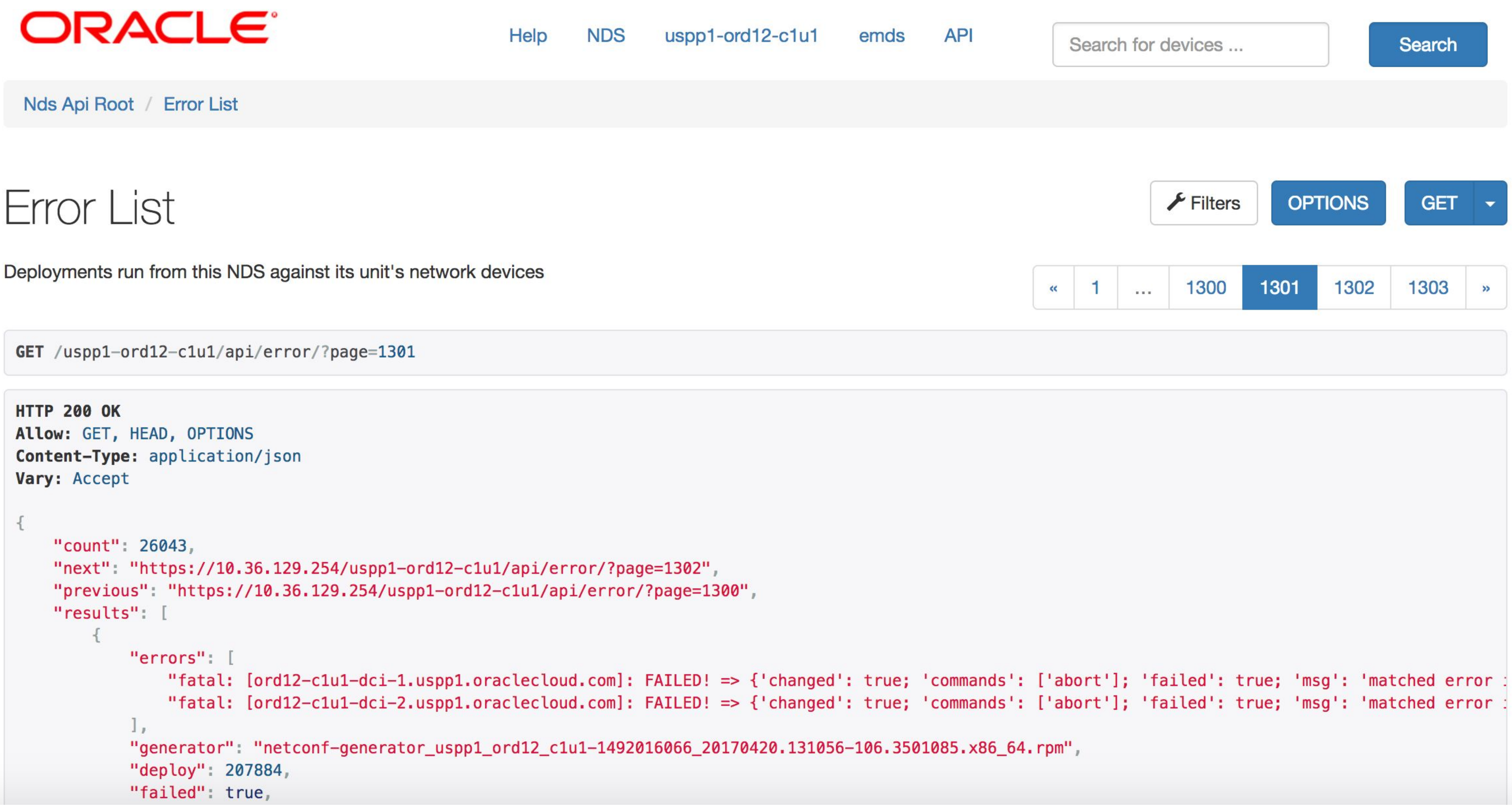
Device task scheduler

Status and diagnostic tasks can take time to aggregate from all devices (a large unit may have hundreds of devices)

So a job scheduler to gather this data asynchronously and then expose via a REST service is available.

REST API (for central management app)

Nds-rest-framework wraps up the standard log model -> resorce and custom summary resources for use by central management application (Java based app by another team)
This is all provided by django-rest-framework with django-filters - with the API using its automatic admin UI, but with a slightly more JSON content exposing stylesheet that integrates it into the web UI look and feel



Error List

Deployments run from this NDS against its unit's network devices

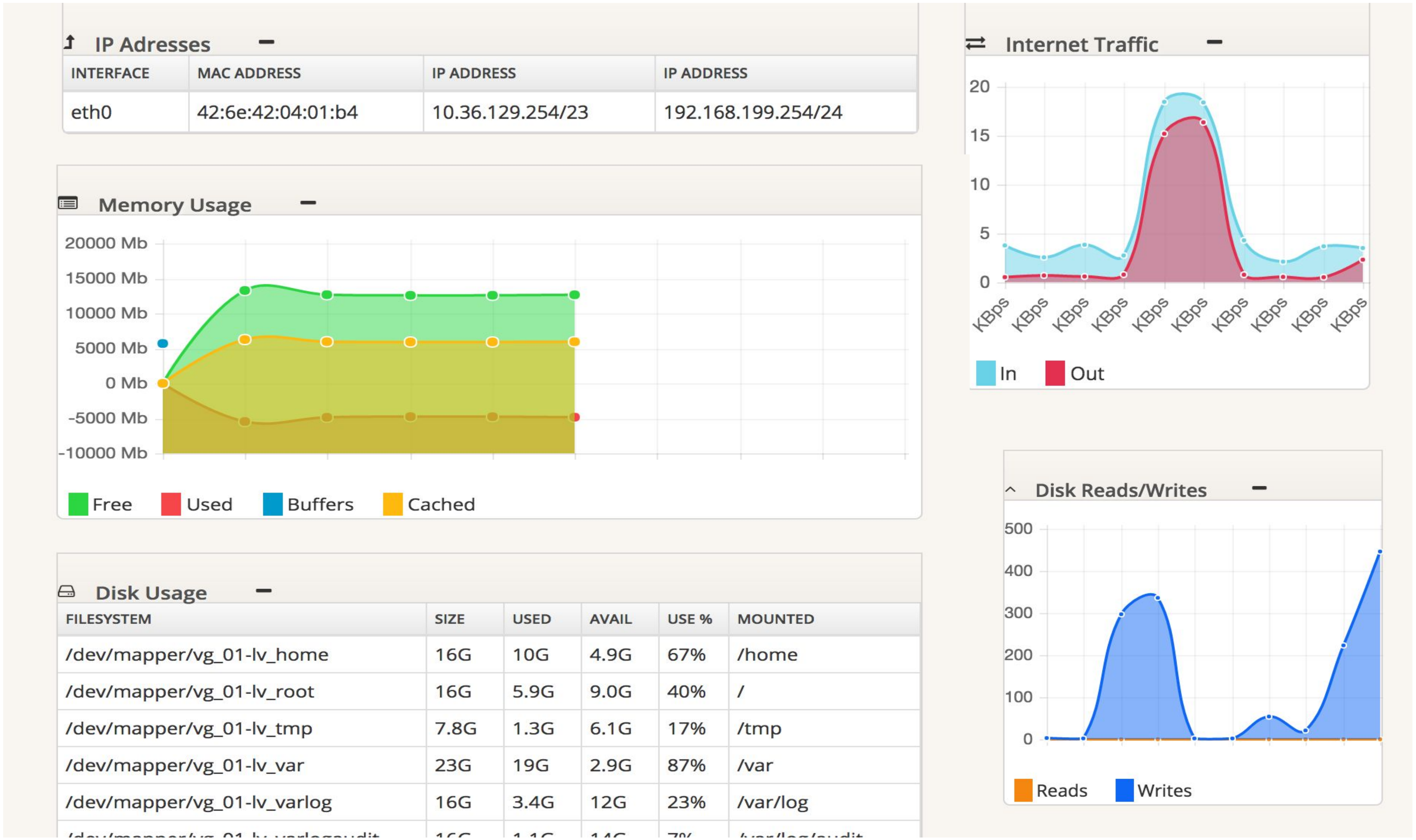
```
GET /uspp1-ord12-c1u1/api/error/?page=1301
```

```
HTTP 200 OK
Allow: GET, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

{
    "count": 26043,
    "next": "https://10.36.129.254/uspp1-ord12-c1u1/api/error/?page=1302",
    "previous": "https://10.36.129.254/uspp1-ord12-c1u1/api/error/?page=1300",
    "results": [
        {
            "errors": [
                "fatal: [ord12-c1u1-dci-1.uspp1.oraclecloud.com]: FAILED! => {'changed': true; 'commands': ['abort']; 'failed': true; 'msg': 'matched error
                "fatal: [ord12-c1u1-dci-2.uspp1.oraclecloud.com]: FAILED! => {'changed': true; 'commands': ['abort']; 'failed': true; 'msg': 'matched error
            ],
            "generator": "netconf-generator_uspp1_ord12_c1u1-1492016066_20170420.131056-106.3501085.x86_64.rpm",
            "deploy": 207884,
            "failed": true,
```

Server status monitor

https://github.com/k3oni/pydash
(For trend data use sysstats or a newer server telemetry app, eg. Oracle cloud EM)



IP Adresses

INTERFACE	MAC ADDRESS	IP ADDRESS	IP ADDRESS
eth0	42:6e:42:04:01:b4	10.36.129.254/23	192.168.199.254/24

Disk Usage

FILESYSTEM	SIZE	USED	AVAIL	USE %	MOUNTED
/dev/mapper/vg_01-lv_home	16G	10G	4.9G	67%	/home
/dev/mapper/vg_01-lv_root	16G	5.9G	9.0G	40%	/
/dev/mapper/vg_01-lv_tmp	7.8G	1.3G	6.1G	17%	/tmp
/dev/mapper/vg_01-lv_var	23G	19G	2.9G	87%	/var
/dev/mapper/vg_01-lv_varlog	16G	3.4G	12G	23%	/var/log
/dev/mapper/vg_01-lv_varlogaudit	16G	1.1G	14G	7%	/var/log/audit

End to end testing with Docker compose

ORACLE

Triggers the build of all the dependent services for NDS web and mocks the devices to allow for testing of device pushes, device fails etc. This is hooked up to Jenkins CI jobs triggered to validate merge requests (with notifications on slack) …

```
          Name                           Command               State                  Ports
---------------------------------------------------------------------------------------------------------------
artifactory-e2e               /bin/sh -c /entrypoint-art ...   Up      127.0.0.1:8081->8081/tcp
bamboo-e2e                    /bin/sh -c source /root/.b ...   Up      127.0.0.1:8085->8085/tcp
bitbucket-e2e                 /sbin/tini -- /entrypoint. ...   Up      127.0.0.1:7990->7990/tcp,
127.0.0.1:7999->7999/tcp
e2e_ldap                      /ldap/slapd.sh                   Up      127.0.0.1:389->389/tcp, 636/tcp
e2e_mysql                     /entrypoint.sh mysqld            Up      127.0.0.1:3306->3306/tcp, 33060/tcp
ndstest_e2e_uspp1_ord12_c1u1   /nds/entrypoint.sh              Up      127.0.0.1:44301->443/tcp,
127.0.0.1:55001->80/tcp
ndstest_e2e_uspp1_ord12_tp1    /nds/entrypoint.sh              Up      127.0.0.1:44302->443/tcp,
127.0.0.1:55002->80/tcp
netdef-e2e                    java -jar net-def.jar serv ...   Up      127.0.0.1:4010->4010/tcp,
127.0.0.1:4012->4012/tcp
netui-e2e                     /usr/share/nginx/html/setu ...   Up      127.0.0.1:8082->80/tcp
yumrepo-e2e                   /data/entrypoint.sh              Up      127.0.0.1:8880->80/tcp
> Waiting for services to start!
..................
```

Questions

Thanks,
Ed Crewe

http://edcrewe.com/

Talk is linked from the meetup site

https://www.meetup.com/python-dbbug/events/244781627/