Continuous Integration for your database migrations

Created by: Michael Still <mikal@stillhq.com>,
Joshua Hesketh <josh@nitrotech.org>
Modified: 2014-05-13

rackspace®
the open cloud company

About Me

Some quick things about me:

- I work for Rackspace
- Primarily on OpenStack
- I am currently President for Linux Australia
- I am passionate about open source
- I helped organise linux.conf.au 2009, PyCon Australia 2012+2013

Overview

- Motivation for testing migrations (and why you should do it too)
- Test infrastructure and how it ties together
- How we test database migrations
- Some interesting bugs uncovered by our work

Terminology

- **Schema version**: a single database schema, represented by a number
- **Database migration**: the process of moving between schema versions
- **Dataset:** a copy of a real deployment used to test against
- **Sqlalchemy**: the database ORM that OpenStack nova uses
- **Zuul**: A gating system developed by openstack-infra to launch jobs
- **Idempotent:** Idempotence is the property of certain operations in mathematics and computer science, that can be applied multiple times without changing the result beyond the initial application.

Migration 138 (Drop server name column)

```
def upgrade(migrate_engine):
    meta = MetaData()
    meta.bind = migrate_engine

    instances = Table('instances', meta, autoload=True)
    server_name = instances.columns.server_name
    server_name.drop()

def downgrade(migrate_engine):
    meta = MetaData()
    meta.bind = migrate_engine
    instances = Table('instances', meta, autoload=True)
    server_name = Column('server_name',
                         String(length=255))
    instances.create_column(server_name)
```

What is continuous integration?

"Continuous integration is the practice, in software engineering, of merging all developer working copies with a shared mainline several times a day. It was first named and proposed as part of extreme programming."

http://en.wikipedia.org/wiki/Continuous_Integration

- **The OpenStack project defines CI as:**
  - Running unit and integration tests on all patches as part of the review process ("check"). These checks vote on reviews much like human reviewers.
  - Once a patch is approved by human reviewers, a subset of these tests are re-run before merge ("gate").

Motivation

- While our tests covered lots of unit tests, and even upgrades of trivial systems, we weren't testing upgrades on real production data.
- We found the following things while doing this work:
  - Schema drift – some deployments had schemas that weren't possible with our current models and migrations. Upgrades didn't work for those people.
  - Performance issues – migrations which took way longer than reasonable on non-trivial data sets.
  - Broken downgrades – while many migrations implemented downgrades, several of them didn't work with non-trivial data.
- Aside: are downgrades important? How do you handle a failed upgrade in a production deployment? Are backups from before the upgrade sufficient?

Goals

- Catch slow migrations.
- Catch migrations that may not work against real datasets.
- Test new migrations but also existing migrations.
  - Changes to core components may affect the overall performance.
- Ease pains for operators.
- Catch problems early.
  - Migrations can not be changed once landed (as a rule).

Managing your nova database

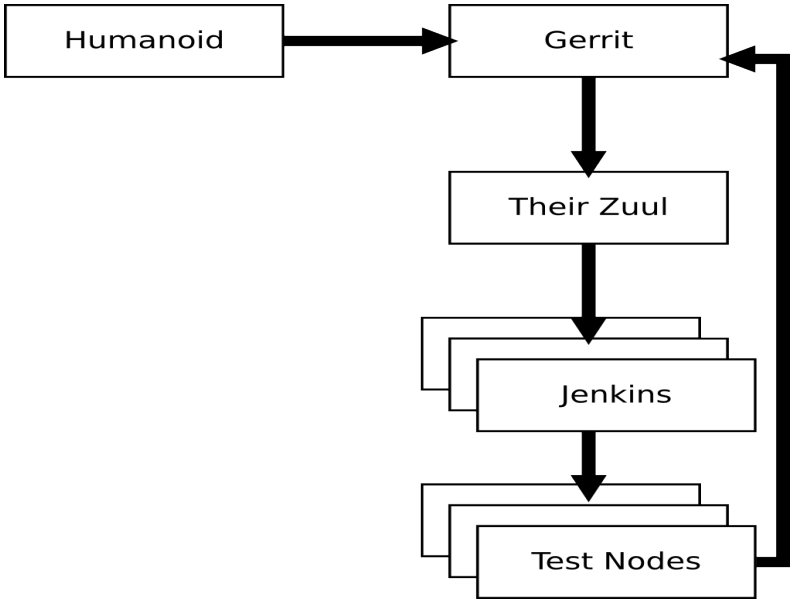
```
nova-manage db sync --version 230
```

Testing database migrations

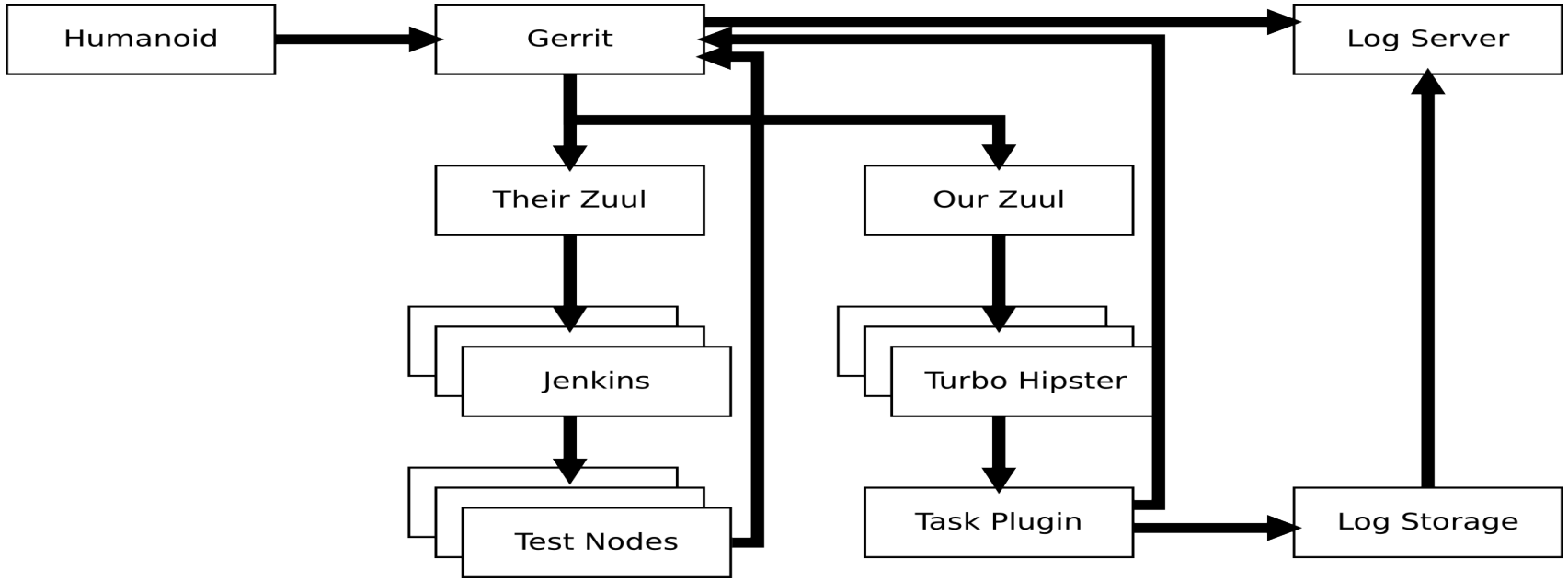
```
nova-manage db sync --version 230
```

- Timing
  - How long does moving from one version to another take?
- innodb stats
  - `show status like 'innodb%';`
  - How many reads, writes, deletes, updates etc. for a given dataset?

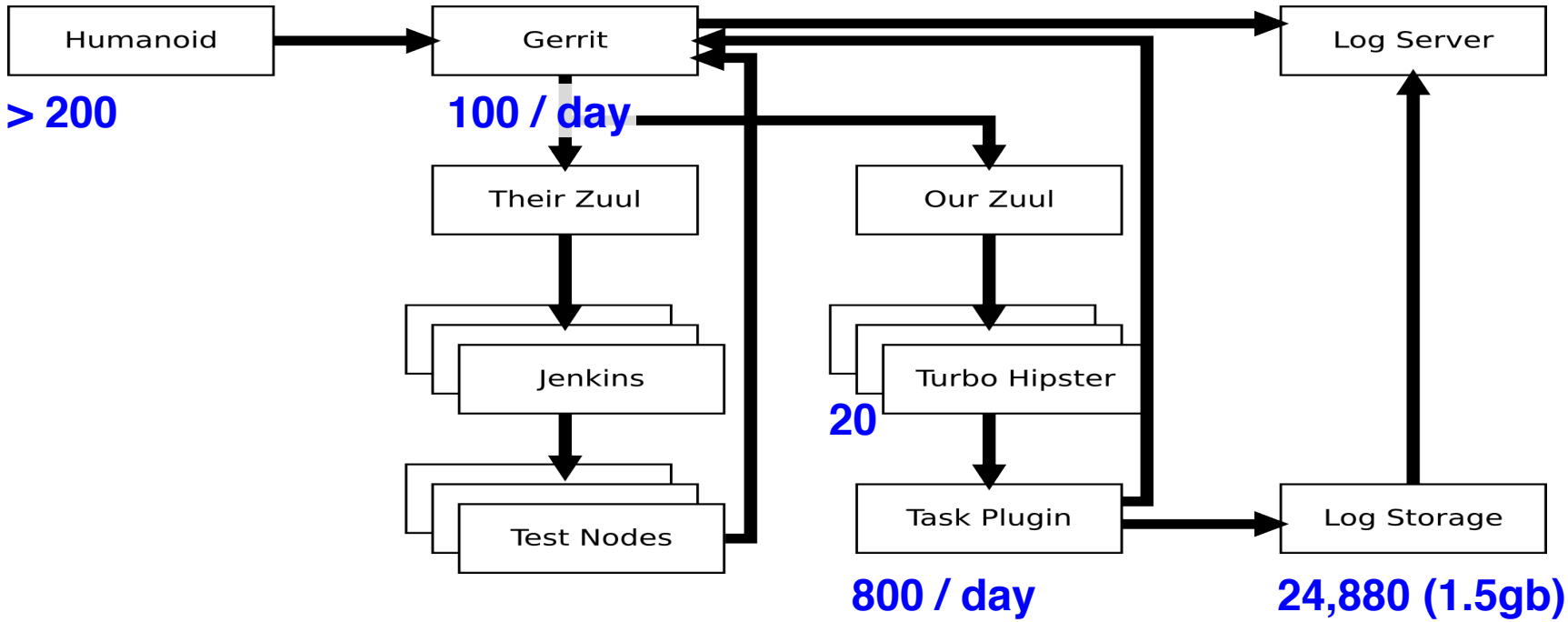
Implementation



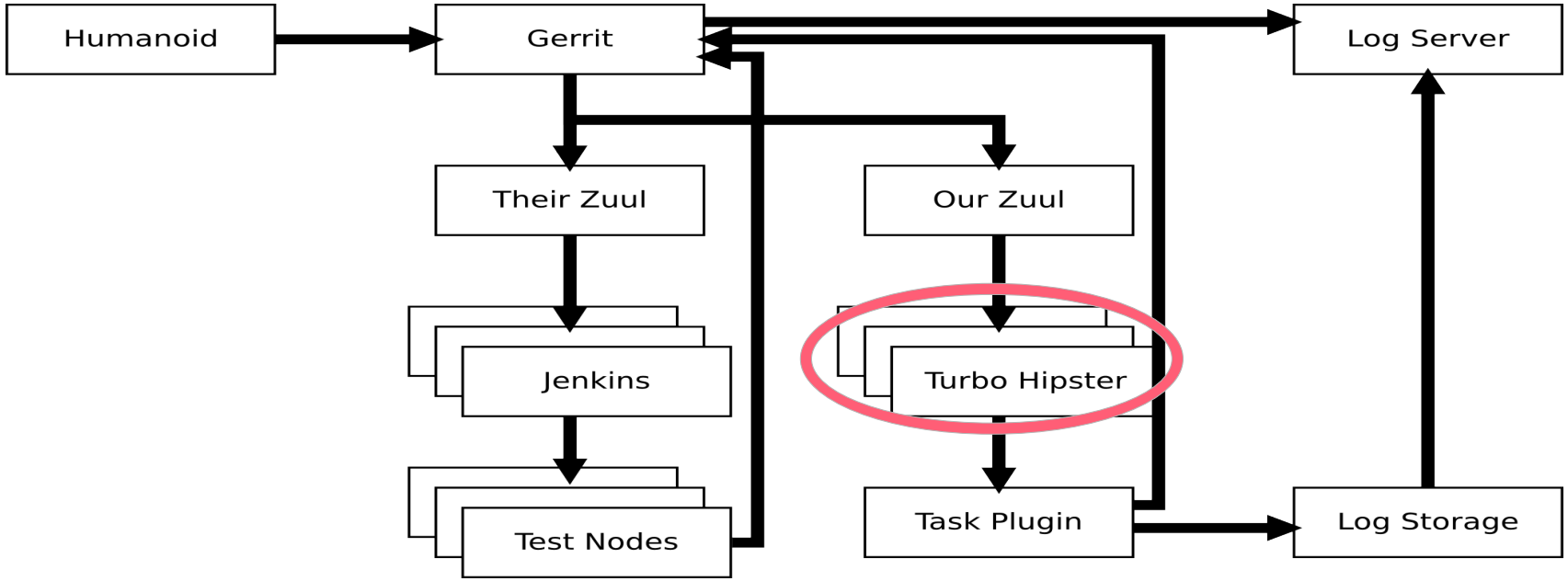
Implementation



Implementation



Implementation



Implementation

Turbo Hipster

- Turbo Hipster is a test runner.
  - A series of task plugins
  - Registration with Zuul
  - Running task plugins when requested
  - Log storage
  - Results

- The actual voting in Gerrit is handled by Zuul on the basis of the results returned by Turbo Hipster

/etc/turbo-hipster/config.yaml

```
zuul_server:
  gerrit_site: http://review.openstack.org
  git_origin: git://git.openstack.org
  gearman_host: ourzuul.rcbops.com

debug_log: /var/log/turbo-hipster/debug.log
jobs_working_dir: /var/lib/turbo-hipster/jobs
git_working_dir: /var/lib/turbo-hipster/git
pip_download_cache: /var/cache/pip

publish_logs:
  type: swift
  authurl: …

conf_d: /etc/turbo-hipster/conf.d/
```

/etc/turbo-hipster/conf.d/db_migration.yaml

```
plugins:
    - name: real_db_upgrade
      function: build:real-db-upgrade_nova_mysql_devstack_131007
      datasets_dir: /var/lib/turbo-hipster/datasets_devstack_131007

    - name: real_db_upgrade
      function: build:real-db-upgrade_nova_percona_devstack_150
      datasets_dir: /var/lib/turbo-hipster/datasets_devstack_150
…
```

/var/lib/turbo-hipster/dataset/config.json

```
{
    "Innodb_rows_read": {
        …
    },
    "XInnodb_rows_changed": {
        …
     },
    "database": "nova_datasets_user_001",
    "db_pass": ”pass",
    "db_user": ”user",
    "logging_conf": "logging.conf",
    "maximum_migration_times": {
        …
     },
    "project": "openstack/nova",
    "seed_data": "nova_user_001.sql",
    "type": "mysql"
}
```

/var/lib/turbo-hipster/dataset/config.json

```
{
    "Innodb_rows_read": {
        …
    },
    "XInnodb_rows_changed": {
        …
     },
    "database": "nova_datasets_user_001",
    "db_pass": ”pass",
    "db_user": ”user",
    "logging_conf": "logging.conf",
    "maximum_migration_times": {
        …
     },
    "project": "openstack/nova",
    "seed_data": "nova_user_001.sql",
    "type": "mysql"
}
```

The real_db_upgrade plugin

- A bit of an anti-climax, its really not that complicated
- For each patchset
  - Checkout the code
  - Bootstrap a new test database
  - Upgrade to the current state of trunk (ie, run migrations)
  - Upgrade to the state of the patch
  - Downgrade to the first migration in this release
  - Upgrade again
- Pass / fail is based on analysis of the logs from the shell script run.

The real_db_upgrade plugin

- A bit of an anti-climax, its really not that complicated
- For each patchset
  - Checkout the code
  - Bootstrap a new test database
  - Upgrade to the current state of trunk (ie, run migrations)
  - Upgrade to the state of the patch
  - Downgrade to the first migration in this release
  - Upgrade again
- Pass / fail is based on analysis of the logs from the shell script run.

The real_db_upgrade plugin

- A bit of an anti-climax, its really not that complicated
- For each patchset
  - Checkout the code
  - Bootstrap a new test da
  - Upgrade to the current s
  - Upgrade to the state of the patch
  - Downgrade to the first migration in this release
  - Upgrade again

```
./gerrit-git-prep.sh
# Added for turbo-hipster
git branch -D working || true
git checkout -b working
```

- Pass / fail is based on analysis of the logs from the shell script run.

The real_db_upgrade plugin

- A bit of an anti-climax, its really not that complicated
- For each patchset
  - Checkout the code
  - Bootstrap a new test database
  - Upgrade to the current state of trunk (ie, run migrations)
  - Upgrade to the state of the patch
  - Downgrade to the first migration in this release
  - Upgrade again
- Pass / fail is based on analysis of the logs from the shell script run.

The real_db_upgrade plugin

- A bit of an anti-climax, its really not that complicated
- For each patchset
  - Checkout the code
  - Bootstrap a new test database
  - Upgrad
  - Upgrad
  - Downg
  - Upgrad

```
echo "Restoring test database $6"
mysql -u $4 --password=$5 -e "drop database $6"
mysql -u $4 --password=$5 -e "create database $6"
mysql -u $4 --password=$5 $6 < $7
```

- Pass / fail is based on analysis of the logs from the shell script run.

The real_db_upgrade plugin

- A bit of an anti-climax, its really not that complicated
- For each patchset
  - Checkout the code
  - Bootstrap a new test database
  - Upgrade to the current state of trunk (ie, run migrations)
  - Upgrade to the state of the patch
  - Downgrade to the first migration in this release
  - Upgrade again
- Pass / fail is based on analysis of the logs from the shell script run.

The real_db_upgrade plugin

- A bit of an anti-climax, its really not that complicated
- For each
  - Checkou
  - Bootstra
  - Upgrade
  - Upgrade
  - Downgra
  - Upgrade

```
echo "Database is from Folsom! Upgrade via
Grizzly”
git branch -D stable/grizzly || true
git remote update
git checkout -b stable/grizzly
git reset --hard remotes/origin/stable/grizzly
pip_requires
db_sync "grizzly"
```

- Pass / fail is based on analysis of the logs from the shell script run.

The real_db_upgrade plugin

- A bit of an anti-climax, its really not that complicated
- For each
  - Checkou
  - Bootstra
  - Upgrade
  - Upgrade
  - Downgra
  - Upgrade
- Pass / fail is based on analysis of the logs from the shell script run.

```
echo "Database is from Folsom! Upgrade via
Grizzly”
git branch -D stable/grizzly || true
git remote update
git checkout -b stable/grizzly
git reset --hard remotes/origin/stable/grizzly
pip_requires
db_sync "grizzly"
```

The real_db_upgrade plugin

- A bit of an anti-climax, its really not that complicated
- For each
  - Checkou
  - Bootstra

```
echo "Database is from Folsom! Upgrade via
Grizzly"
git branch -D stable/grizzly || true
git remote update
```

```
echo "MySQL counters before upgrade:"
mysql -u $4 --password=$5 $6 -e "show status like 'innodb%';"

for i in `seq $start_version $increment $end_version` do
  nova-manage --verbose db sync --version $i
  echo "MySQL counters after upgrade:"
  mysql -u $4 --password=$5 $6 -e "show status like 'innodb%';"
done
```

The real_db_upgrade plugin

- A bit of an anti-climax, its really not that complicated
- For each
  - Checkou
  - Bootstra

```
echo "Database is from Folsom! Upgrade via
Grizzly"
git branch -D stable/grizzly || true
git remote update
```

```
echo "MySQL counters before upgrade:"
mysql -u $4 --password=$5 $6 -e "show status like 'innodb%';"

for i in `seq $start_version $increment $end_version` do
  nova-manage --verbose db sync --version $i
  echo "MySQL counters after upgrade:"
  mysql -u $4 --password=$5 $6 -e "show status like 'innodb%';"
done
```

The real_db_upgrade plugin

- A bit of an anti-climax, its really not that complicated
- For each patchset
  - Checkout the code
  - Bootstrap a new test database
  - Upgrade to the current state of trunk (ie, run migrations)
  - Upgrade to the state of the patch
  - Downgrade to the first migration in this release
  - Upgrade again
- Pass / fail is based on analysis of the logs from the shell script run.

The real_db_upgrade plugin

- A bit of an anti-climax, its really not that complicated
- For each patchset
  - Checkout the code
  - Bootstrap a new test database
  - Upgrade to the current state of trunk (ie, run migrations)
  - Upgrade to the state of the patch
  - Downgrade to the first migra
  - Upgrade again
- Pass / fail is based on an

```
git checkout working
echo "Now test the patchset"
pip_requires
db_sync "patchset"
```

The real_db_upgrade plugin

- A bit of an anti-climax, its really not that complicated
- For each patchset
  - Checkout the code
  - Bootstrap a new test database
  - Upgrade to the current state of trunk (ie, run migrations)
  - Upgrade to the state of the patch
  - Downgrade to the first migration in this release
  - Upgrade again
- Pass / fail is based on analysis of the logs from the shell script run.

The real_db_upgrade plugin

- A bit of an anti-climax, its really not that complicated
- For each patchset
  - Checkout the code
  - Bootstrap a new test database
  - Upgrade to the current state of trunk (ie, run migrations)
  - Upgrade to the state of the patch
  - Downgrade to the first migration in this release
  - Upgrade aga

```
db_sync "patchset" $last_stable_version
```

- Pass / fail is based on analysis of the logs from the shell script run.

The real_db_upgrade plugin

- A bit of an anti-climax, its really not that complicated
- For each patchset
  - Checkout the code
  - Bootstrap a new test database
  - Upgrade to the current state of trunk (ie, run migrations)
  - Upgrade to the state of the patch
  - Downgrade to the first migration in this release
  - Upgrade again
- Pass / fail is based on analysis of the logs from the shell script run.

The real_db_upgrade plugin

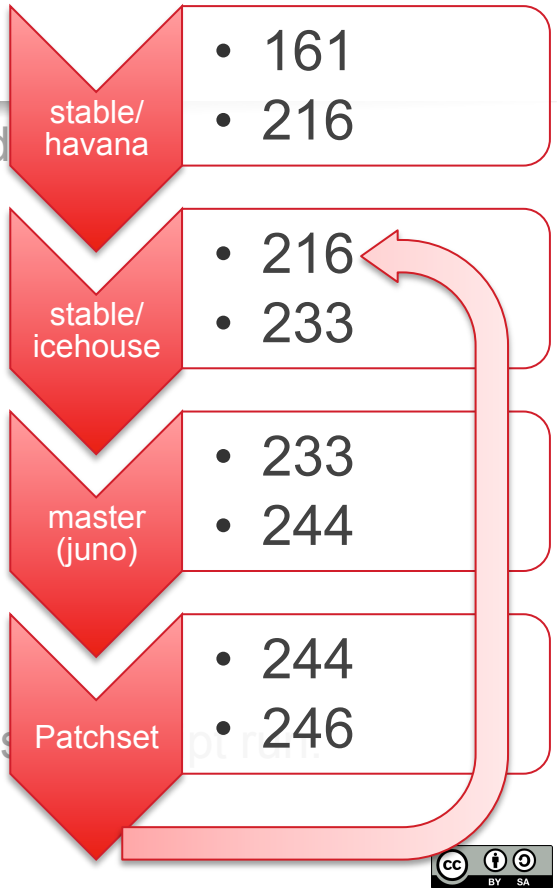
- A bit of an anti-climax, its really not that complicated
- For each patchset
  - Checkout the code
  - Bootstrap a new test database
  - Upgrade to the current state of trunk (ie, run migrations)
  - Upgrade to the state of the patch
  - Downgrade to the first migration in this release
  - Upgrade again

```
db_sync "patchset"
```

- Pass / fail is based on analysis of the logs from the shell script run.

The real_db_upgrade plugin

- A bit of an anti-climax, its really not that complicated
- For each patchset
  - Checkout the code
  - Bootstrap a new test database
  - Upgrade to the current state of trunk (ie, run migrations)
  - Upgrade to the state of the patch
  - Downgrade to the first migration in this release
  - Upgrade again
- Pass / fail is based on analysis of the logs from the s pt run



The real_db_upgrade plugin

- A bit of an anti-climax, its really not that complicated
- For each patchset
  - Checkout the code
  - Bootstrap a new test database
  - Upgrade to the current state of trunk (ie, run migrations)
  - Upgrade to the state of the patch
  - Downgrade to the first migration in this release
  - Upgrade again
- Pass / fail is based on analysis of the logs from the shell script run.

Resulting log files

```
2013-12-24 05:38:08,338 [output] 216 -> 217...
2013-12-24 05:38:08,451 [output] done
2013-12-24 05:38:08,451 [output] 217 -> 218...
2013-12-24 05:38:08,455 [output] done
2013-12-24 05:38:08,455 [output] 218 -> 219...
2013-12-24 05:38:08,460 [output] done
2013-12-24 05:38:08,460 [output] 219 -> 220...
2013-12-24 05:38:08,464 [output] done
2013-12-24 05:38:08,464 [output] 220 -> 221...
2013-12-24 05:38:08,469 [output] done
2013-12-24 05:38:08,469 [output] 221 -> 222...
2013-12-24 05:38:08,473 [output] done
```

Resulting log files

```
2014-01-16 07:03:51,248 [output] Innodb_row_lock_waits  0
2014-01-16 07:03:51,248 [output] Innodb_rows_deleted    0
2014-01-16 07:03:51,248 [output] Innodb_rows_inserted   0
2014-01-16 07:03:51,248 [output] Innodb_rows_read   5
2014-01-16 07:03:51,248 [output] Innodb_rows_updated    1
2014-01-16 07:03:51,248 [output] Innodb_read_views_memory   88
2014-01-16 07:03:51,248 [output] Innodb_descriptors_memory  8000
2014-01-16 07:03:51,249 [output] Innodb_s_lock_os_waits 2
2014-01-16 07:03:51,249 [output] Innodb_s_lock_spin_rounds  60
2014-01-16 07:03:51,249 [output] Innodb_s_lock_spin_waits   2
```

Resulting log files

```
2014-01-16 07:03:51,248 [output] Innodb_row_lock_waits  0
2014-01-16 07:03:51,248 [output] Innodb_rows_deleted    0
2014-01-16 07:03:51,248 [output] Innodb_rows_inserted   0
2014-01-16 07:03:51,248 [output] Innodb_rows_read   5
2014-01-16 07:03:51,248 [output] Innodb_rows_updated    1
2014-01-16 07:03:51,248 [output] Innodb_read_views_memory   88
2014-01-16 07:03:51,248 [output] Innodb_descriptors_memory  8000
2014-01-16 07:03:51,249 [output] Innodb_s_lock_os_waits 2
2014-01-16 07:03:51,249 [output] Innodb_s_lock_spin_rounds  60
2014-01-16 07:03:51,249 [output] Innodb_s_lock_spin_waits   2
```

/var/lib/turbo-hipster/dataset/config.json

```
{
    "Innodb_rows_read": {
        …
    },
    "XInnodb_rows_changed": {
        …
     },
    "database": "nova_datasets_user_001",
    "db_pass": ”pass",
    "db_user": ”user",
    "logging_conf": "logging.conf",
    "maximum_migration_times": {
        …
     },
    "project": "openstack/nova",
    "seed_data": "nova_user_001.sql",
    "type": "mysql"
}
```

/var/lib/turbo-hipster/dataset/config.json

```
{
    "Innodb_rows_read": {
        "148->149": 110000,
        "151->152": 3470000,
        "159->160": 200000,
        "160->161": 390000,
        "202->203": 260000,
        "205->206": 140000,
        "215->216": 930000,
        "default": 100000
    },
    "XInnodb_rows_changed": {
          …
     },
    "database": "nova_datasets_user_001",
    "db_pass": ”pass",
…
```

/var/lib/turbo-hipster/dataset/config.json

```
{
    "Innodb_rows_read": {
        …
    },
    "XInnodb_rows_changed": {
        "148->149": 110000,
        "151->152": 3200000,
        "184->185": 140000,
        "193->194": 150000,
        "202->203": 520000,
        "203->204": 260000,
        "205->206": 190000,
        "215->216": 280000,
        "229->230": 140000,
        "default": 100000
    },
…
```

/var/lib/turbo-hipster/dataset/config.json

```
…
    "maximum_migration_times": {
        "134->135": 200.0,
        "135->134": 97.0,
        … …
        "206->205": 106.0,
        "215->216": 137.0,
        "229->230": 122.0,
        "230->229": 84.0,
        "230->231": 120,
        "231->230": 120,
        "default": 60
     },
    "project": "openstack/nova",
    "seed_data": "nova_user_001.sql",
    "type": "mysql"
}
```

Return results

Database migration testing successful..

gate-real-db-upgrade_nova_mysql_devstack_131007 SUCCESS in 4m 12s
gate-real-db-upgrade_nova_mysql_devstack_150 SUCCESS in 4m 17s
gate-real-db-upgrade_nova_mysql_user_001 SUCCESS in 14m 52s
gate-real-db-upgrade_nova_percona_devstack_131007 SUCCESS in 4m 20s
gate-real-db-upgrade_nova_percona_devstack_150 SUCCESS in 4m 11s
gate-real-db-upgrade_nova_percona_user_001 SUCCESS in 14m 22s
gate-real-db-upgrade_nova_mysql_user_002 FAILURE - Did not find the end of a migration after a start in 4m 16s (non-voting)
gate-real-db-upgrade_nova_percona_user_002 FAILURE - Did not find the end of a migration after a start in 4m 47s (non-voting)

Security

Security

- This is all a little bit scary
- We're running code on our workers provided by third parties
- Those people have registered on Gerrit, but could be pretty much anyone

- Mitigation:
  - Only a limited set of people have access to the worker nodes, so placing data on the file system inside your patch isn't very interesting.
  - The untrusted code from third parties is tested with networking turned off.
  - We check logs for suspicious data before serving them to the real world.
  - We're also working on dataset anonymization.

Security

- Running a process with networking turned off…
  - We initially explored LXC as an option

Security

- Running a process with networking turned off…
  - We initially explored LXC as an option
  - Network namespaces (netns) is much simpler

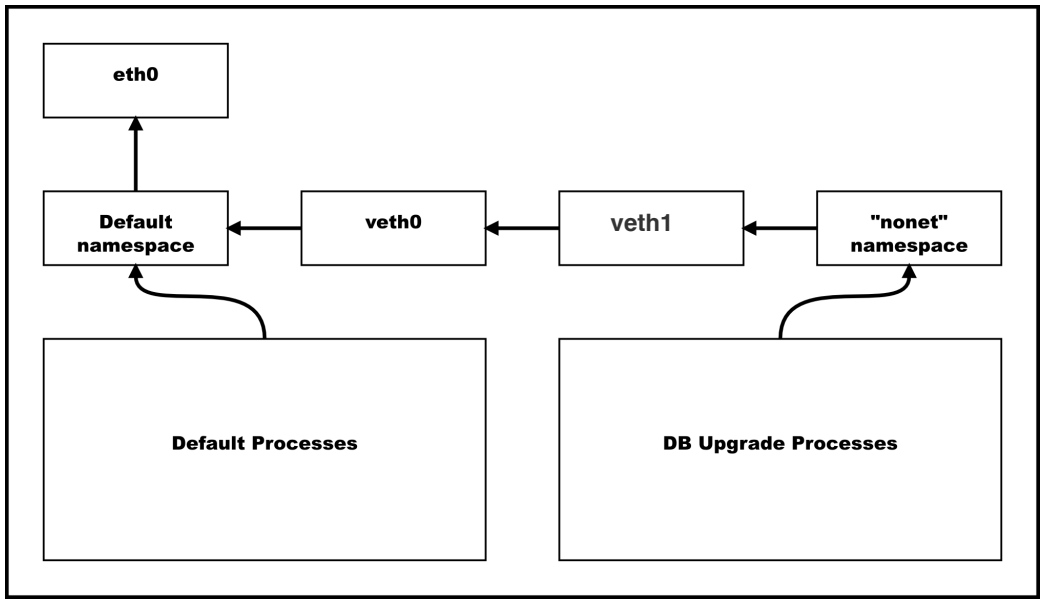
Security

- Running a process with networking turned off…
  - We initially explored LXC as an option
  - Network namespaces (netns) is much simpler

```
ip netns add nonet
ip link add veth0 type veth peer name veth1
ifconfig veth0 172.16.0.1/24 up
ip link set veth1 netns nonet
ip netns exec nonet ifconfig veth1 172.16.0.2/24 up
/sbin/iptables -A INPUT -p tcp --dport 3306 -i eth0 -j DROP
/sbin/iptables -A INPUT -p tcp --dport 3306 -i eth1 -j DROP
```

Security

- Running a process with networking turned off…
  - We initially explored LXC as an option
  - Network namespaces (netns) is much simpler



Security

- Running a process with networking turned off…
  - We initially explored LXC as an option
  - Network namespaces (netns) is much simpler

```
ip netns exec nonet …command to upgrade db…
```

DIY

- Set up zuul
  - Read zuul's documentation
  - Configure it to trigger on patchsets you're interested in
  - Launch a job per a dataset you wish to test
- Set up turbo-hipster
  - Configure the datasets and register them as jobs against zuul
  - Tweak or write your own migration shell script

Interesting bugs found

Date: Tue, 11 Jun 2013 20:29:05 +1000
Subject: Slow db migration in havana
From: Michael Still <mikal@stillhq.com>
To: openstack-operators@lists.openstack.org

Hi.

I just wanted to make sure people are aware of a slow db migration which landed in havana today. I discovered this while testing a CI script I am working on which tests DB migrations.

The migration is number 186, which landed with this review https://review.openstack.org/#/c/29251/11 -- it rewrites the block device mapping tables, and will therefore affect people making heavy use of volumes.

This migration takes just under 20 minutes with my test database of 45k block device mappings and 35k shadow block device mappings, on an otherwise unloaded cloud instance.

Cheers,
Michael

Interesting bugs found

Date: Tue, 11 Jun 2013 20:29:05 +1000
Subject: Slow db migration in havana
From: Michael Still <mikal@stillhq.com>
To: openstack-operators@lists.openstack.org

Hi.

I just wanted to make sure people are aware of a [illegible] which landed in havana today. I discovered this while testing a CI script I am [illegible] which tests DB migrations.

The migration is number 186, [illegible] this review https://review.openstack.org/#/c/29251/11 -- it rewrites the block devi[illegible]es, and will therefore affect people making heavy use of volumes.

This migration [illegible] just under 20 minutes with my test database of 45k block device mappings and 35k shadow block device mappings, on an otherwise unloaded cloud instance.

Cheers,
Michael

https://review.openstack.org/#/c/32744/1/nova/db/sqlalchemy/migrate_repo/versions/186_new_bdm_format.py

Interesting bugs found

Change instances config_drive column from string to Boolean

config drive feature was added early in commit d963e25906b75a48c75b6e589deb2a53f75d6ee3, config_drive column is varchar type but only store Boolean value, so when using nova boot --config-drive true to build instances, a string 'True' is stored in DB,

The function still works because a string is regard as True in statement, this patch changes column type from varchar to Boolean, and changes the string 'True' to a Boolean type.

Closes-Bug: 1193438

Change-Id: I6c2951a890dc213a053e0ccbc0685ecf0afbdf1a

Interesting bugs found

Patch Set 2: **Do not merge**

Is there any reason to make this change apart from db cleanliness?

The original config drive API allowed you to specify the UUID for a disk image which was used to build the config drive, which is why this column started out as a string. When config drive was re-written that functionality was dropped as being insecure, but we didn't want to change the API without bumping the API version number.

So... We haven't deprecated that API version, and we might still have clients expecting to send a UUID for this call.

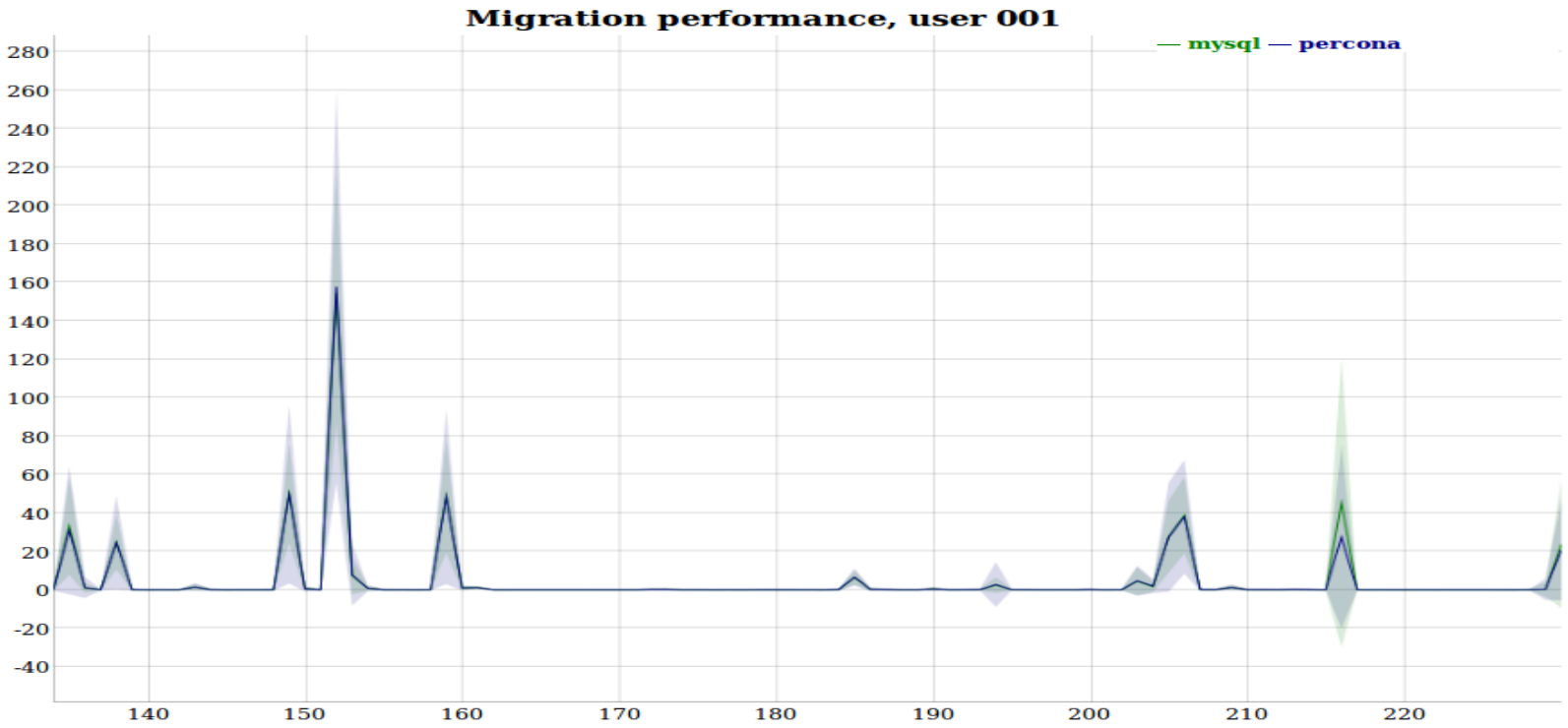
Also, rewriting this column is a very expensive operation for large deployments, so unless we have a really solid reason to do it, I'd prefer we just leave this as is.

Secondary effects

- We now have a really good database of how long various migrations take, and the relative performance of various SQL engines.
  - In fact, we set the time limits for each migration based on a historical analysis of these logs, so the system learns over time.
  - There are some flaws in this though.

- http://www.rcbops.com/graphs/results.html

Secondary effects



Where to from here?

- zuul triggers (eg github).
- Dataset anonymization.
- More scalable workers with nodepool.
- Testing other engines (eg postgres, mariadb etc.).
- Reduce false positives.

- zuul and turbo-hipster are both open source and can be found at
  - http://git.openstack.org

THANK YOU

Questions?
Joshua.hesketh@rackspace.com

the **open cloud** company

RACKSPACE® HOSTING | 5000 WALZEM ROAD | SAN ANTONIO, TX 78218
US SALES: 1-800-961-2888 | **US SUPPORT:** 1-800-961-4454 | WWW.**RACKSPACE**.COM