Mutually Assured Quality

Jason Etheridge
Hat Model @Equinox

Denish Patel
Database Architect @OmniTI

Quality Assurance

What is quality? assurance?

- Quality is fitness for purpose.
- Quality Assurance (QA) is a *process*, a set of methods, that aims to produce fitness for purpose with the least amount of effort (or, tries to ensure that it is right the first time)
- Software developers like to automate this stuff as much as they can.

So we tend to think “tests”,

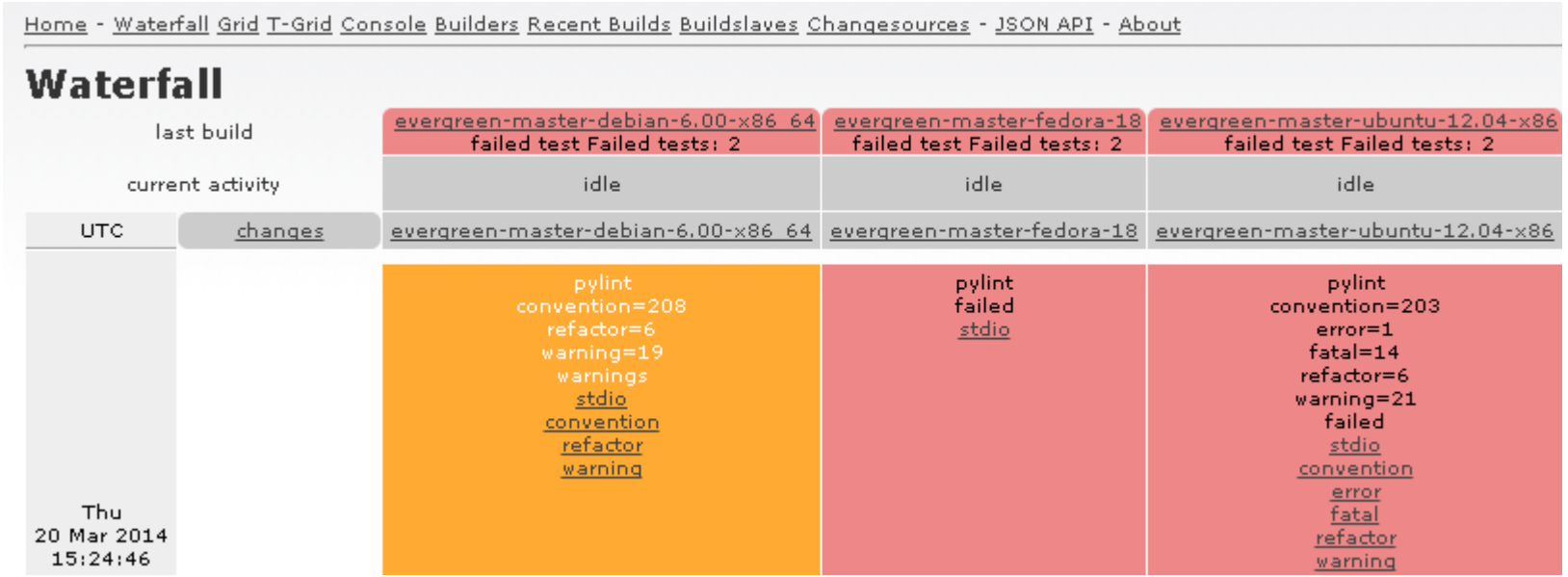
but before we talk about that, what else is there besides software testing? Idea testing. We need to define “purpose” in Fit For Purpose.

Development in the open helps:

- bug reports and ideas / conversation
- design / consensus / craft / version control
- code review / pull requests / sign-offs

Breaking the Build

http://testing.evergreen-ils.org/buildbot/



Home - Waterfall Grid T-Grid Console Builders Recent Builds Buildslaves Changesources - JSON API - About

Waterfall

		evergreen-master-debian-6.00-x86_64	evergreen-master-fedora-18	evergreen-master-ubuntu-12.04-x86
last build		failed test Failed tests: 2	failed test Failed tests: 2	failed test Failed tests: 2
current activity		idle	idle	idle
UTC	changes	evergreen-master-debian-6.00-x86_64	evergreen-master-fedora-18	evergreen-master-ubuntu-12.04-x86
Thu 20 Mar 2014 15:24:46		pylint convention=208 refactor=6 warning=19 warnings stdio convention refactor warning	pylint failed stdio	pylint convention=203 error=1 fatal=14 refactor=6 warning=21 failed stdio convention error fatal refactor warning

The more the merrier

http://testing.evergreen-ils.org/~live/

[test.sh output] [installer_installer.sh output] [Previous Runs] [Git]

Test Output Summary

HTML generated on Fri Mar 21 04:37:09 EDT 2014

- initializing installer - **Passed**
- configure timezone - **Passed**
- configure CPAN - **Passed**
- Installing some build essentials - **Passed**
- creating opensrf user and environment - **Passed**
- cloning git repositories - **Passed**
- Installing OpenSRF pre-requisites - **Passed**
- Installing Evergreen pre-requisites - **Passed**
- Installing Evergreen database pre-requisites - **Passed**
- setting ld.so.conf and rsyslog and /etc/hosts and ejabberd - **Passed**

one last slide

```
BEGIN;

SELECT plan(2);

UPDATE config.internal_flag SET enabled = FALSE WHERE name = 'cat.bib.use_id_for_tcn';
INSERT INTO biblio.record_entry (marc, last_xact_id)
VALUES ('<record xmlns="http://www.loc.gov/MARC21/slim"/>', 'test');

SELECT matches((SELECT tcn_value FROM biblio.record_entry
                WHERE id = CURRVAL('biblio.record_entry_id_seq')),
                '^AUTOGENERATED-',
                'TCN is autogenerated when cat.bib.use_id_for_tcn is disabled');

UPDATE config.internal_flag SET enabled = TRUE WHERE name = 'cat.bib.use_id_for_tcn';
INSERT INTO biblio.record_entry (marc, last_xact_id)
VALUES ('<record xmlns="http://www.loc.gov/MARC21/slim"/>', 'test');
SELECT is((SELECT tcn_value FROM biblio.record_entry
           WHERE id = CURRVAL('biblio.record_entry_id_seq')),
           (SELECT CURRVAL('biblio.record_entry_id_seq')::TEXT),
           'TCN equals BRE ID when cat.bib.use_id_for_tcn is enabled');

ROLLBACK;
```

Postgres DB Performance

Review Hardware

- CPU - 8+ cores
- RAM - 64GB+
- Disks - SSDs , RAID 10
- Network - Min Gigbit, 10Gigbit

Review OS

- Operating System
- Max. Shared Memory
- Min. Shared Memory
- Kernel Swappiness
- Cache
- Swap
- Scheduler

Review OS

Linux Kernel 2.6.32 has performance issues with keeping the THP enabled.

Disabled these settings:

- Transparent Huge Pages
- Transparent Huge Pages Defrag

Postgres Config/Memory parameters

- shared_buffers (25% of RAM or 8GB)
- effective_cache_size (70-75% of RAM)
- checkpoint_completion_target=0.9
- checkpoint_segments=100
- max_connections
- work_mem=25MB
- maintenance_work_mem=10MB

Postgres Config/ Log parameters

- logging_collector => 'on'
- log_destination => 'stderr'
- log_filename => 'postgresql-%Y-%m-%d_%H%M%S.log'
- log_line_prefix => '%m [%r] [%p]: [%l-1] user=%u,db=%d,e=%e '
- log_min_duration_statement => 1000ms

Postgres Config/Log parameters

- log_autovacuum_min_duration => '0'
- log_lock_waits => 'on'
- log_temp_files => '0'
- log_checkpoints => 'on'
- log_statement => 'ddl'

Slow Query Log Analysis/pgbadger

Most frequent queries (N) ^

Rank	Times executed	Total duration	Min/Max/Avg duration (s)	Query
1	**29,373**	18h36m0.229s	1.791s/1m12.368s/2.280s	`SELECT "atev".id FROM action_trigger.event AS "atev" INNER JOIN action_trigger.event_definition AS "atevdef" ON ( "atevdef".id = "atev".event_def ) WHERE ( "atevdef".granularity IS NULL ) AND "atev".run_time < '' AND "atev".state = '';` Show examples
2	**14,146**	8h19m56.070s	1.737s/4m32.526s/2.120s	`SELECT count ( * ) FROM action_trigger.event WHERE state = '';` Show examples
3	**1,600**	16h2m53.812s	20.940s/7m13.839s/36.109s	`WITH system_count AS (SELECT bre.id AS bib, count (ac.id) AS count FROM asset.COPY ac JOIN asset.call_number acn ON ac.call_number = acn.id JOIN biblio.record_entry bre ON bre.id = acn.record WHERE bre.id IN (SELECT target FROM ACTION.hold_request ahr WHERE ahr.hold_type = '' ::text) AND ac.deleted = '' GROUP BY bre.id ORDER BY bre.id) , lib_count AS (SELECT bre.id AS bib, count (ac.id) AS count FROM asset.COPY ac JOIN asset.call_number acn ON ac.call_number = acn.id JOIN biblio.record_entry bre ON bre.id = acn.record WHERE bre.id IN (SELECT target FROM ACTION.hold_request ahr WHERE ahr.hold_type = '' ::text) AND ac.deleted = '' AND ac.circ_lib IN (...) GROUP BY bre.id ORDER BY bre.id) , bib_format AS (SELECT mba.id AS bib, cvm.VALUE AS format FROM metabib.record_attr mba, config.coded_value_map cvm WHERE mba.attrs ::hstore - > '' = cvm.code AND cvm.ctype = '' AND mba.id IN (SELECT target FROM ACTION.hold_request ahr WHERE ahr.hold_type = '' ::text AND ahr.pickup_lib IN (...)) GROUP BY mba.id, cvm.VALUE ORDER BY mba.id) SELECT ahr.target, rmsr.title, rmsr.author, f.format, count (ahr.target) , COALESCE (lc.count, '') , sc.count FROM ACTION.hold_request ahr JOIN reporter.materialized_simple_record rmsr ON rmsr.id = ahr.target LEFT OUTER JOIN lib_count lc ON lc.bib = ahr.target JOIN system_count sc ON sc.bib = ahr.target JOIN bib_format f ON f.bib = ahr.target WHERE ahr.hold_type = '' ::text AND ahr.fulfillment_time IS NULL AND ahr.cancel_time IS NULL AND ahr.pickup_lib IN (...) GROUP BY ahr.target, rmsr.title, rmsr.author, f.format, lc.count, sc.count --we want the ratio OF holds TO copies but the titles WITH 0 OWNED will NOT --display because that result IS NULL, so test FOR total holds TO catch those HAVING (count(ahr.target) / lc.count) > 0 OR count(ahr.target) > 0 ORDER BY f.format, count(ahr.target) DESC;` Show examples
4	**1,585**	11h34m7.880s	1.000s/7m0.984s/26.276s	`SELECT * FROM unapi.bre ( '', '', '', '', '', '', '', NULL, NULL, '' ) AS "unapi.bre";` Show examples
5	**1,376**	58m28.853s	1.887s/31.933s/2.550s	`SELECT cn.record FROM asset.call_number cn JOIN asset.COPY cp ON ( cp.call_number = cn.id ) WHERE cp.barcode = '';` Show examples

Slow Query Log Analysis/ pgbadger

Slowest queries ^

Rank	Duration (s)	Query
1	7h30m59.554s	SELECT * -- bib SEARCH: #CD_documentLength #CD_meanHarmonic #CD_uniqueWords core_limit(10000) LIMIT(1000) estimation_strategy(inclusion) keyword: "crusade" AND "art" item_type(a) depth(0) FROM SEARCH.query_parser_fts (1::INT, 0::INT, $ core_query_16668 $ SELECT m.source AS id, ARRAY_ACCUM (DISTINCT m.source) AS records, 1.0 / ((AVG ((COALESCE (ts_rank_cd (x86a60f8_keyword.index_vector, x86a60f8_keyword.tsq, 14) * x86a60f8_keyword.weight, 0.0)) + (COALESCE (ts_rank_cd (x8715028_keyword.index_vector, x8715028_keyword.tsq, 0) * x8715028_keyword.weight, 0.0)) + (COALESCE (ts_rank_cd (x86807b0_keyword.index_vector, x86807b0_keyword.tsq, 0) * x86807b0_keyword.weight, 0.0))) + 1 * COALESCE (NULLIF (FIRST (mrd.attrs @ > hstore ('item_lang', $ _16668 $ eng $ _16668 $)) , FALSE) ::INT * 5, 1))) ::NUMERIC AS rank, FIRST (mrd.attrs - > 'date1') AS tie_break FROM metabib.metarecord_source_map m JOIN metabib.record_attr mrd ON (m.source = mrd.id) LEFT JOIN (SELECT fe.*, fe_weight.weight, x.tsq /* SEARCH */ FROM metabib.keyword_field_entry AS fe JOIN config.metabib_field AS fe_weight ON (fe_weight.id = fe.FIELD) JOIN (SELECT to_tsquery ('keyword', COALESCE (NULLIF ('(' || *btrim* (regexp_replace (search_normalize (split_date_range ($ _16668 $ AND $ _16668 $)) , E '(?:\\s+|:)', '&', 'g') , '&|') || ')', '()') , '')) AS tsq) AS x ON (fe.index_vector @@ x.tsq)) AS x86a60f8_keyword ON (m.source = x86a60f8_keyword.source) LEFT JOIN (SELECT fe.*, fe_weight.weight, x.tsq /* SEARCH */ FROM metabib.keyword_field_entry AS fe JOIN config.metabib_field AS fe_weight ON (fe_weight.id = fe.FIELD) JOIN (SELECT to_tsquery ('keyword', COALESCE (NULLIF ('(' || *btrim* (regexp_replace (search_normalize (split_date_range ($ _16668 $ crusade $ _16668 $)) , E '(?:\\s+|:)', '&', 'g') , '&|') || ')', '()') , '')) AS tsq) AS x ON (fe.index_vector @@ x.tsq)) AS x8715028_keyword ON (m.source = x8715028_keyword.source) LEFT JOIN (SELECT fe.*, fe_weight.weight, x.tsq /* SEARCH */ FROM metabib.keyword_field_entry AS fe JOIN config.metabib_field AS fe_weight ON (fe_weight.id = fe.FIELD) JOIN (SELECT to_tsquery ('keyword', COALESCE (NULLIF ('(' || *btrim* (regexp_replace (search_normalize (split_date_range ($ _16668 $ art $ _16668 $)) , E '(?:\\s+|:)', '&', 'g') , '&|') || ')', '()') , '')) AS tsq) AS x ON (fe.index_vector @@ x.tsq)) AS x86807b0_keyword ON (m.source = x86807b0_keyword.source) WHERE 1 = 1 AND mrd.attrs @ > (hstore ('item_type', $ _16668 $ a $ _16668 $)) AND ((x86a60f8_keyword.id IS NOT NULL) AND (((x8715028_keyword.id IS NOT NULL AND x8715028_keyword.VALUE ~ * $ _16668 $ crusade $ _16668 $)) AND (((x86807b0_keyword.id IS NOT NULL AND x86807b0_keyword.VALUE ~ * $ _16668 $ art $ _16668 $)))) GROUP BY 1 ORDER BY 4 ASC NULLS LAST, 5 DESC NULLS LAST, 3 DESC LIMIT 10000 $ core_query_16668 $ ::TEXT, $$ $$ ::INT [ ], $$ $$ ::INT [ ], NULL::INT, 1000::INT, 10000::INT, 'f' ::BOOL, 'f' ::BOOL, NULL::INT);
2	6h16m25.163s	SELECT * -- bib SEARCH: #CD_documentLength #CD_meanHarmonic #CD_uniqueWords core_limit(10000) LIMIT(1000) estimation_strategy(inclusion) keyword: "hemingway" AND "critical" depth(0) FROM SEARCH.query_parser_fts (1::INT, 0::INT, $ core_query_31570 $ SELECT m.source AS id, ARRAY_ACCUM (DISTINCT m.source) AS records, 1.0 / ((AVG ((COALESCE (ts_rank_cd (x85fb9b8_keyword.index_vector, x85fb9b8_keyword.tsq, 14) * x85fb9b8_keyword.weight, 0.0)) + (COALESCE (ts_rank_cd (x86b9b18_keyword.index_vector, x86b9b18_keyword.tsq, 0) * x86b9b18_keyword.weight, 0.0)) + (COALESCE (ts_rank_cd (x87686e0_keyword.index_vector, x87686e0_keyword.tsq, 0) * x87686e0_keyword.weight, 0.0))) + 1 * COALESCE (NULLIF (FIRST (mrd.attrs @ > hstore ('item_lang', $ _31570 $ eng $ _31570 $)) , FALSE) ::INT * 5, 1))) ::NUMERIC AS rel, 1.0 / ((AVG ((COALESCE (ts_rank_cd (x85fb9b8_keyword.index_vector, x85fb9b8_keyword.tsq, 14) * x85fb9b8_keyword.weight, 0.0)) + (COALESCE (ts_rank_cd (x86b9b18_keyword.index_vector, x86b9b18_keyword.tsq, 0) * x86b9b18_keyword.weight, 0.0)) + (COALESCE (ts_rank_cd (x87686e0_keyword.index_vector, x87686e0_keyword.tsq, 0) * x87686e0_keyword.weight, 0.0))) + 1 * COALESCE (NULLIF (FIRST (mrd.attrs @ > hstore ('item_lang', $ _31570 $ eng $ _31570 $)) , FALSE)

2013-05-06 19:13:17

Query Tuning/Example

- PostgreSQL logged 672,530 queries for a day
  - The most time consuming query was logged 229,246 times, runtime 2.061897s
  - SELECT * FROM unapi.bre ('166563', 'marcxml', 'record', '{holdings_xml,mra, acp,acnp,acns,bmp}', 'WORCESTER_MA', '2', 'acn=>5,acp=>5', NULL, NULL, '142') AS "unapi.bre";
  - Total runtime : 2.04s was spent calling unapi.holdings_xml function
  - EXPLAIN ANALYZE - http://explain.depesz.com/s/P1l , 146 lines long!
- unapi.bre => unapi.holdings_xml calls => evergreen.ranked_volumes()
  - select * FROM evergreen.ranked_volumes(2992059, 1, 0, '"acn"=>"5", "acp" =>"5"', NULL, 328, '{mra,acp,acnp,acns,bmp}'::text[]);
  - Explain Analyze: http://explain.depesz.com/s/DYd
- evergreen.ranked_volumes() => execution time was **400ms**
  - After rewrite , it runs in **16ms!**

Monitoring

- Proactive monitoring is recommended
  - # of connections
  - # of locks
  - etc.
- Tools
  - Circonus (www.circonus.com)
  - Nagios
  - Zabbix
  - Ganglia

Specific Recommendations

- Postgresql.conf tuning
- Upgrade Postgres to 9.2.X
  - 20-25% performance increase
- Postgres full text search
  - Much Much faster than %ILIKE% queries
- Use Built-in Replication
  - SLONY replication is being used
  - Streaming replication is built-in
    - read-only secondary db

Questions?

- Jason Etheridge
  - Email: jason@esilibrary.com However, the public mailing list is better: OPEN-ILS-GENERAL
  - QA whitepaper: http://nox.esilibrary.com/~jason/qareport/qa.html
- Denish Patel
  - Email : denish@omniti.com
  - Twitter: @DenishPatel
  - Blog: www.pateldenish.com