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/

<table>
<thead>
<tr>
<th>UTC</th>
<th>changes</th>
<th>evergreen-master-debian-6.00-x86_64</th>
<th>evergreen-master-fedora-18</th>
<th>evergreen-master-ubuntu-12.04-x86</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thu 20 Mar 2014 15:24:46</td>
<td></td>
<td>pylint convention=208, refactor=6, warning=19</td>
<td>pylint failed stdio</td>
<td>pylint convention=203, error=1, fatal=14, refactor=6, warning=21</td>
</tr>
</tbody>
</table>

- pylint
- convention=208
- refactor=6
- warning=19
- stdio
- convention
- refactor
- warning
The more the merrier

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

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
- Building OpenSRF - Passed
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

- `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'
<table>
<thead>
<tr>
<th>Rank</th>
<th>Times executed</th>
<th>Total duration</th>
<th>Min/Max/Avg duration (s)</th>
<th>Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>29,373</td>
<td>18h36m0.229s</td>
<td>1.791s/1m12.368s/2.280s</td>
<td><code>SELECT &quot;stev&quot;.id FROM action_trigger.event AS &quot;stev&quot; INNER JOIN action_trigger.event_definition AS &quot;stevdef&quot; ON (&quot;stevdef&quot;.id = &quot;stev&quot;.event_def) WHERE (&quot;stevdef&quot;.granularity IS NULL) AND &quot;stev&quot;.run_time &lt; '2023-01-01 00:00:00' AND &quot;stev&quot;.state = '');</code></td>
</tr>
<tr>
<td>2</td>
<td>14,146</td>
<td>8h19m56.070s</td>
<td>1.737s/4m32.526s/2.120s</td>
<td><code>SELECT COUNT ( * ) FROM action_trigger.event WHERE state = '');</code></td>
</tr>
<tr>
<td>3</td>
<td>1,600</td>
<td>16h2m53.812s</td>
<td>20.940s/7m13.839s/36.109s</td>
<td><code>WITH system_count AS (SELECT bre.id AS bib, COUNT ( ac.id ) AS COUNT FROM asset_COPY ac JOIN asset.call_number scn ON ac.call_number = scn.call_number WHERE biblio.record_entry bre ON bre.id = ac.record WHERE bre.id IN ( SELECT target FROM ACTION_hold_request shr WHERE shr.hold_type = 'text' AND ac.deleted = 'false' 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 scn ON ac.call_number = scn.call_number WHERE biblio.record_entry bre ON bre.id = ac.record WHERE bre.id IN ( SELECT target FROM ACTION_hold_request shr WHERE shr.hold_type = 'text' AND ac.deleted = 'false' GROUP BY bre.id ORDER BY bre.id ) ), bib_format AS (SELECT mba.id AS bib, mba.VALUE AS format FROM metabib.record_attr mba, config.code_value_map cvm WHERE mba.attr_value = 'text' AND cvm.CODE AND cvm.type = 'system' AND mba.id IN ( SELECT target FROM ACTION_hold_request shr WHERE shr.hold_type = 'text' AND mba.id ORDER BY mba.id ) ) SELECT shr.target, mba.title, mba.author, f.format, COUNT ( shr.target ) FROM ACTION_hold_request shr WHERE shr.hold_type = 'text' AND shr.pickup_lib IN ( ... ) GROUP BY mba.id, mba.VALUE ORDER BY mba.id ) SELECT shr.target, mba.title, mba.author, f.format, COUNT ( shr.target ) COALESCE ( lc.count, 0 ), sc.count FROM ACTION.hold_request shr JOIN reporter.materialized_simple_record mrmr ON mrmr.id = shr.target LEFT OUTER JOIN lib_count lc ON lc.call_number = mrmr.call_number WHERE biblio.record_entry bre ON bre.id = mrmr.magazine WHERE biblio.record_entry bre ON bre.id = mrmr.magazine APPLY COUNT ( COUNT ( shr.target ) ) AS 'unapi.bib' WHERE COUNT ( COUNT ( shr.target ) ) &gt; 0 OR COUNT ( COUNT ( shr.target ) ) &gt; 0 ORDER BY f.format, COUNT ( shr.target ) DESC;</code></td>
</tr>
<tr>
<td>4</td>
<td>1,585</td>
<td>11h34m7.880s</td>
<td>1.000s/7m0.984s/26.276s</td>
<td><code>SELECT * FROM unapi.bre (',', ',', ',', ', ', ', ', ', ', NULL, NULL, ' ') AS 'unapi.bre';</code></td>
</tr>
<tr>
<td>5</td>
<td>1,376</td>
<td>58m28.853s</td>
<td>1.887s/31.933s/2.550s</td>
<td><code>SELECT rc.record FROM asset.call_number cn JOIN asset.COPY cp ON ( cp.call_number = cn.id ) WHERE cp.barcode = '');</code></td>
</tr>
</tbody>
</table>
### Slow Query Log Analysis (pgbadger)

<table>
<thead>
<tr>
<th>Rank</th>
<th>Duration (s)</th>
<th>Query</th>
</tr>
</thead>
</table>
| 1    | 7h30m59.554s | ```
```
| 2    | 6h16m25.163s | ```
```
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 - [Link](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: [Link](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 %LIKE% 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

● Denish Patel
  ○ Email: denish@omniti.com
  ○ Twitter: @DenishPatel
  ○ Blog: www.pateldenish.com