TEXAS ADVANCED COMPUTING CENTER

WWW.TACC.UTEXAS.EDU

Json ingestion improvements

Robert McLay

May 19, 2022

XALT: Outline

- Ingestion Definition
- Questions about how site install XALT and Ingestion
- Execution Filtering
- New Package Filtering
- Debugging *.json file ingestion
- Tracking Down when Json Ingestion is slow
- Debugging *.json syslog ingestion

Ingestion Definition

- XALT generates Json records (either file or syslog or ...)
- Ingestion is the phase where Json records are added to the MySQL DB.

How sites install XALT for users?

- XALT is installed locally on each node? (TACC does this)
- XALT is in a shared global location?
- Other ways?

How sites install XALT for Ingestion?

- TACC uses a VM to store the syslog transmission style
- We have also used a shared global file location to ingest json records
- Other ways?

Dealing with different types of XALT installation

- This talk assumes that installing XALT users is more difficult
- At TACC, XALT can only be updated on Maintenance Days
- This may get easier in the future at TACC with rolling instations
- Whereas changes to ingestion are easy.
- We only change the XALT installation on the VM.
- Is this different for your site?

Executable Pre-ingestion filtering

- Because some sites have trouble updating XALT for users
- There was a request to allow filtering of *.json records
- These are transmitted records but not yet ingested

A new python array to filter executables

```
pre_ingest_patterns = [
#    percent    path pattern
     [0.0,      r'.*foobar'],
     [0.01,     r'.*BAZ'],
]
```

- This python array is converted to a flex routine
- This flex code is converted to C code and compiled into a shared library (libpreIngest.so)
- Python allows for standard libraries be connected to python programs

How filtering is shown in xalt_configuration_report

```
% xalt_configuration_report
...
*----------------------*
 Array: ingestPatternA
*----------------------*
================== /path/to/site/config.py ==================
   0: 0.0000 => .*foobar
   1: 0.0100 => .*BAZ
================== src/tmpl/xalt_config.py ==================
   2: 1.0000 => .*
```

- Note that src/tmpl/xalt_config.py provides the default patterns.
- So you don't have to.

Connecting C shared libraries w/ Python

```
from   ctypes import *    # used to interact with C shared libraries

libpath      = os.path.join(dirNm, "../lib64/libpreIngest.so")
libpreIngest = CDLL(libpath)
pre_ingest_filter = libpreIngest.pre_ingest_filter
pre_ingest_filter.argtypes = [c_char_p]
pre_ingest_filter.restype  = c_double
...
exec_prob = pre_ingest_filter(exec_path.encode())
prob      = random.random() # 0 < prob <= 1.0
if (prob <= exec_prob):
   # ingest
```

- The ctypes python package provide the magic.
- Once the boilerplate code is provided, it is just one line to find the probability to keep or not.
- This is very fast filtering

Pkg Filtering added to Ingestion

- XALT 2.10.37 added Pkg filtering to xalt_record_pkg prgm
- It also added the same filtering to Ingestion
- It uses the same ctypes package to integrate it with python.
- This means that you can filter package w/o re-installing XALT everywhere.

Protecting XALT against endless loops from site Config.py files

```
# A site had in their site Config.py:
path_patterns = [
  ['KEEP',  r'optenvHPC.*'],
  [ 'SKIP',  r'.*'],
  ]
```

- where XALT was stored in /opt/envHPC/xalt/...
- XALT must protect sites from tracking all xalt programs
- Otherwise you can get an endless loop.

Preventing Endless XALT loops

- XALT has a system config file: src/tmpl/xalt_config.py
- Site has a Config.py file
- This files control how XALT filters
- The system file has head_path_patterns
- This forces XALT to skip all XALT executabls
- Also skip the unix system logger command which write to syslog.
- XALT uses logger in testing.

Configuration Report

```
*----------------------*
 Array: pathPatternA
*----------------------*
====== src/tmpl/xalt_config.py =====
   1: SKIP => .*logger
   6: SKIP => .*xalt_syshost
   7: SKIP => .*xalt_record_pkg
======= Config.py ============
...
====== src/tmpl/xalt_config.py =====
  21: KEEP => .*
```

- Abbreviated patterns from xalt_configuration_report

Debugging Json ingestion

- Issue #46 shows a detailed discussion where a site had trouble ingesting.
- It wasn't clear where the slowdown was happening
- xalt_file_to_db.py and xalt_syslog_to_db.py got the -D option
- This adds debug printing for the internal steps

XALT searches first for link.*.json files

```
link: /home/user/.xalt.d/link.rios.*.json
  --> Trying to open file
  --> Trying to load json
  --> Sending record to xalt.link_to_db()
  --> Trying to connect to database
  --> Starting TRANSACTION
  --> Searching for build_uuid in db
  --> Trying to insert link record into db
  --> Success: link recorded
  --> Trying to insert objects into db
  --> Trying to insert functions into db
  --> Done
```

XALT searches for run.*.json files next

```
run: /home/user/.xalt.d/run.rios.*.json
  --> Trying to open file
  --> Trying to load json
  --> Sending record to xalt.run_to_db()
  --> Trying to connect to database
  --> Starting TRANSACTION
  --> Searching for run_uuid in db
  --> Trying to insert run record into db
  --> Success: stored full xalt_run record
  --> Trying to insert objects into db
  --> Trying to insert env vars into db
  --> Done
```

Finally XALT searches for pkg.*.json

```
--> Found 10 pkg.*.json files

--> Success: pkg entry "R:bar" stored
--> Success: pkg entry "R:foo" stored
--> Success: pkg entry "R:acme" stored
--> Failed to record: pkgFilter blocks "R:base"
--> Success: pkg entry "python:json" stored
--> Success: pkg entry "python:linecache" stored
--> Success: pkg entry "python:struct" stored
--> Success: pkg entry "python:base64" stored
--> Success: pkg entry "python:codecs" stored
```

Debugging Results

- Site used ∼user/.xalt.d
- BTW: This has a race condition
- The ∼/.xalt.d directory has to exist first
- Walking a parallel file system is slow when looking ∼/.xalt.d directories.
- I encourage the site to switch to a global shared location

Similarly for xalt_syslog_to_db.py

```
--> Trying to connect to database
--> Starting TRANSACTION
--> Searching for build_uuid in db
--> Trying to insert link record into db
--> Success: link recorded
--> Trying to insert objects into db
--> Trying to insert functions into db
--> Done

--> Trying to connect to database
--> Starting TRANSACTION
--> Searching for run_uuid in db
--> Trying to insert run record into db
--> Success: stored full xalt_run record
--> Trying to insert objects into db
--> Trying to insert env vars into db
--> Done

--> Success: pkg entry "python:token" stored
--> Success: pkg entry "python:tokenize" stored
--> Success: pkg entry "python:linecache" stored
```

- But there there are no file names given and the link, run and pkg records are mixed together

Conclusions

- Debugging ingestion is not practical for thousands for records.
- But it is useful.
- Next Meeting June 16 10:00am U.S. Central (15:00 UTC)

Future Topics?

- ???
- Others?