RYOT engadget YAHOO! SPORTS MAKERS Aol. TechCrunch HUFFPOST YAHOO! MAIL ① by Aol. YAHOO! tumblr. verizon digital media services YAHOO! FINANCE BrightRoll YAHOO! GEMINI

AMC Tech Corner

Introduction

Alan M. Carroll, Code Slinger Extraordinare

This is a whirl wind tour of my current development projects, both current and future.

- Transaction Box
- Control of DSO reload
- YAML configuration for cache, “records.config”, and IPAllow / SNI Configuration
- Change to _internal_msg_buffer
- Down server changes
- libSWOC
- Managed Traffic Server

Transaction Box

State of Play

Introduction

Transaction Box (“TxnBox”) is a Traffic Server plugin. It is intended to

- Substitute for a variety of other plugins, including many Lua based ones, to simplify plugin configuration.
- Provide a YAML based configuration as agreed by the community. In many cases this obviates the need to update other plugins to YAML, they can be replaced by TxnBox.
- Preview a possible replacement for “remap.config”. This was decided on a year ago at the Beijing Summit, where in conjunction with Miles I started this work.
- Designed to be easily extensible so that new minor functionality can be added, rather than having to write a new plugin.
- Good documentation.

TxnBox Basics

- YAML configuration.
- Global and remap configuration.
- Support for most hooks (ever expanding) – everything is tied to a specific hook.

Repository: https://github.com/SolidWallOfCode/txn_box

Documentation: http://docs.solidwallofcode.com/txn_box

Slack Channel: traffic-server-txnbox

Configuration Elements

Directives make things happen

Comparisons compare data for conditional action

Extractors get data for directives and comparisons.

Modifiers adjust data from extractors.

TxnBox 0.2.11 deployed

- TxnBox 0.2.11 is currently running in YCPI production.
- Used primarily for ramping – controlling the distribution of requests across multiple upstreams.
- Static file capability tested in production.

Enhancements

Based on experience with production and feedback on the TxnBox channel, I made a number of enhancements.

- AuTest support for QA (thanks Brian!). The basic support is operational, adding additional tests.
- Static file support.
- Response replacement (serve configuration specified content instead of the upstream response).
  - Use: replace an upstream 403 with a 204 in specific circumstances.
- Configuration – file globbing, post-load hook, and global reload.
- The big rename, and removing the remap specific elements.

Zeerust – past futures

Many of the capabilities described at the last summit as futures are now implemented.

- IPSpace is available as an experimental feature.
- Dynamic regular expressions.
- Tuple support (creation, use, comparison).
- Documentation support and content.
- Transaction variables.
- Comparisons: combination, IP address, numeric, boolean
- Duration.
- Some TLS data (SNI name)
- Random numbers.

https://tvtropes.org/pmwiki/pmwiki.php/Main/Zeerust

Internals

Several major restructuring of internals to make extending TxnBox easier.

- Feature expression support – parsing, type determination.
- “Type” initialization for directives at config load time.
- Directive arguments.
- Per config and per context storage for Directives, Extractors.

Static Files

TxnBox can load and store a file from the file system during global configuration loading and then extracted where useful.

- File can be periodically checked for modification and automatically reloaded.
- Content is extracted and so can be used anywhere, such as an HTTP field value or as the body of a response.

post-load

Based on feedback I added a new pseudo-hook, “post-load”.

- Directives on this hook are called once per configuration load, immediately after that configuration loading has finished.
- Directives can return error conditions which invalidate the configuration.
  - Therefore a directive can cross check in an order independent way other directives.
  - On initial load, this fails out the TS process. On reload it prevents loading the new configuration.
- Global data is loaded here, such as static files and IPSpaces.

Config file glob support

- TxnBox 0.2.11 supports the standard “file glob” support such as is available in a Bash shell.
- The original request was to be able to load a directory.
- Current pattern support allows loading a directory, or only specific files in a directory (such as only “.yaml” files).
- Multiple patterns can be used to load multiple sets of configuration files.
- Pattern matching tracks loaded configuration files so that if a file matches multiple patterns, it is only loaded once.
  - If one file *must* be loaded first, it can be explicitly called out even if it matches a later pattern.
  - Order of loading is based on sorted order, again the same as shell file globbing.
- Usable in global and remap configurations.

IPSpace

- Load IPSpace during post-load hook
- Columns can be integer, string, flag set, or enumeration.
- Modifier to convert IP address to IPSpace row, then extractors to pull out column data.
- Supports the same auto-reload capability of static files.

Example: Ramp

This is used on a remap rule from “ex.one” to “ex.one”, which is the default destination. This configs shifts 20% of the traffic to “staging.ex.one”.

“remap.config”

```
map http://ex.one http://ex.one @plugin=txn_box.so @pparam=txn_box/ramp.yaml
```

“ramp.yaml”

```
with: random<0,999>
select:
- lt: 200 # 20% ramp
  do:
  - ua-req-host: staging.ex.one
```

Example: Static File

This sets the “Token” field in the proxy request to the contents of the file “token.txt” if it is not already set.

```
when: post-load # called immediately after configuration is loaded
do:
- text-block-define:
    path: “token.txt” # look in TS config directory
    name: “token”
    duration: hours<8> # check for updates every 8 hours

when: proxy-req # proxy request - SEND_REQUEST_HDR_HOOK
do:
- proxy-req-field<Token>: # set the value for the “Token” field.s
  - proxy-req-field<Token> # fetch current value
  - else: text-block<token> # use text-block if no current value
```

Still to do

- String matching acceleration (thanks Damian!).
- Argument indirection for dynamic arguments.
- Comparison local modifiers.
- More tuple operators (for-all, for-every).
- Dynamic DSO loading / invocation.

Minor Updates

Plugin Control of DSO Reload

This work has been done by Damian, with design input from Gancho and myself, and community feedback.

- Add a new plugin API call to control whether a plugin can have its DSO reloaded at run time.
  - Must be called from TSPluginInit
- Default value is configuration value.
- The plugin implementor knows best if the plugin DSO can be reloaded successfully.
  - Should not bother the administrator, who will generally have no idea. Don’t let him do stupid things.
- Global switch too heavy handed in some situations.
- Enables a plugin to restore pre-ATS 9 behavior for loading.
- Does not change current ATS 9 default behavior.

internal_msg_buffer

This is used to store content for responses that are generated internally. It is currently an allocated string and has very poor performance when doing many updates – the buffer is repeatedly reallocated and copied.

A change has been discussed for a long time, which is to convert this to an MIOBuffer. I've been working on that but instead of an MIOBuffer an IOBufferChain seems a better idea. This will, in addition to far better performance on repeated writes, also enable removing various limitations and inefficiencies in the template system.

Down server recheck

Current no protection is made for a down server. ATS will continue to pound on it as requests come in.

This work will move more of the DNS controlling information in to the HostDB record, where it can be stored independent of the HttpSM. This is required to coordinate upstream connections among different instances of HttpSM to the same upstream.

The logic will have a block period during which no connection attempt will be made. After that, the next request will be allowed upstream while other requests are blocked. If the “leaker” succeeds, the upstream is marked as available. If the leaker fails, the block period becomes active, postdated to the start time of the upstream connection attempt.

Yamlization

Cache Configuration

There are currently 4 distinct configuration files for configuring the ATS cache.

- “storage.config” - defines the raw storage.
- “volume.config” - creates the cache volumes and stripes.
- “hosting.config” – assignment of specific domains / hosts to specific cache volumes
- “cache.config” – cache knob tweaking for requests

Rather than converting these individually to YAML, I propose having a single cache configuration file. I would also like to simplify the configuration and core implementation by removing the “cache.config” and “hosting.config” features from the core and making them available via the plugin API instead.

https://tvtropes.org/pmwiki/pmwiki.php/Main/Zeerust

Design

```
cache: # configuration root key
  storage: # list of storage blocks
  - path: “/dev/nme1-0” # path to storage.
    size: “auto” # size of storage [optional for disks]
    tag: “ssd” # reference tag, a name.
    seed: “ssd-seed” # seed string for cache consistent hash
  volumes: # list of volume definitions
  - tag: “linear” # name of volume
    ramcache: enable # default is enable
    store: # list of storage elements to use.
    - use: “ssd” # tag of the storage unit, can be “*” to mean “all”
      size: “100%” # amount of storage to use.
```

Notes

- Extend plugin API to specify the cache volume. Use TxnBox or custom plugin to control.
  - Enables much finer grained or different criteria to control volume assignment.
- Extend plugin API for cache options (e.g., store despite cookies, etc.).

records.config

This was discussed on the mailing list.

- Use a tree form.
- Drop the “proxy.config” prefix.
- Otherwise looking at a direct translation from flat names to a tree.

Issues:

- Entry points for traffic_ctl configuration settings.
  - Unlikely to be able to support updating arbitrary subtrees.
    Most like need explicit “configuration points” that can be updated.

IPAllow / SNI Configuration

IPAllow is used to control access by IP address. This overlaps with SNI configuration which also has IP address based access control. Since one of the larger goals of going to YAML is to have a more unified, consistent configuration, these should be more unified and not independent yet interfering mechanisms.

inbound.yaml

The proposal is to remove “ipallow.config” and “sni.config” and replace them with “inbound.yaml” which configures the initial handling of inbound connections.

- TLS based – non-TLS is handled as a special exception to specifying an SNI name.
- Must handle “no SNI name” as an explicit case.
- Each SNI name has an IP map of ranges -> properties / actions
- Properties would be the current SNI options plus the method list from IP Allow.
- Support a “global” record which is applied before the SNI specific one.

libSWOC

Improved core utilities

libSWOC

- libSWOC is a fork of ATS core utilities, with a few added classes.
- Forked to enable rapid development.
- Approaching reasonable maturity, could be pulled back into TS core.
- Being used by other projects, may not be able to shut down.
- May be needed for some other work (e.g. IPAllow/SNI work).
- Could import as done with libYAML-CPP.

Updates

- TextView much improved.
- Added Lexicon which will be handy for handling enumerations.
- Added IPSpace to upgrade from IPMap.
- Added MemArena.
- More extensive documentation.

Managed ATS

Reasons

I’ve been kicking this around for a couple years, now getting pressure (validation?) from others. As elastic services become more common in corporate infrastructure and therefore move toward a more dynamicenvironment, ATS must become easier / more convenient to manage in an automated fashion. Operating in such environment is becoming a critically required feature.

The goal can be summed up in the phrase “deployment convenience”. In a dynamic environment, it must be easy and fast to deploy new instances.

Note: Kit’s Ingress Controller is a project on the same theme but focusing on a different aspect of this same issue.

Enhancing ATS for managed environments

What is needed to help ATS migrate to managed environments? Susan Hinrichs, Bryan Call, and I are involved in an effort inside the VZ Edge group.

- Standard based administrative control (JSON RPC project). (Damian, Alan)
- Standard based configuration (YAML/JASON configuration files). (Alan, Will)
  - Conversion projects underway or getting started for critical configuration files.
- Minimal install footprint.
  - Defaults for missing files. (Damian)
  - Fewer configuration files. (Alan)
- Stability in limited resource environments. (Fei)
- Best practices for common scenarios. (Fei, Susan, Bryan)

It’s JSON, all the way down

We’ve already committed to YAML as the base configuration syntax. Using JSON/YAML for other management interfaces provides a much more consistent style for automated management and simplifies internal development.

- JSON handling logic internally works the same for all of the external interfaces – parsing RPC becomes literally identical to parsing configuration files. This helps enforce consistency in code and use.
- Developers do not need to learn yet another tool chain, the build system is not made more complex.
- If the administrator can handle JSON configuration, he has the tools to handle JSON interfaces. “If you do JSON, you’re all set for automated management of ATS”.