YAHOO!

TSQA v4

Jason Kenny

Goals

- Easy to run the tests
- Run all the test, or a subset
- Easy to run in automated or manual setup
- Easy to write tests
- Easy to run tests against existing build or different builds
- Easy to add tests
- Precise diagnostic.
- Anything that goes wrong should be reported, such as a process did not start
- Should be extensible

What is TSQA v4

- It is a mix of:
  - Reusable Gold Testing System (AuTest for short)
    - https://bitbucket.org/dragon512/reusable-gold-testing-system (MIT license)
    - Documentation: https://bitbucket.org/dragon512/reusable-gold-testing-system/wiki/Home
  - A set of extension for AuTest to customize the ease to create tests for ATS to be checked into TrafficServer under the tests/ directory
    - Until pull request is done:
      - https://bitbucket.org/dragon512/ats_tests

So what is Reusable Gold Testing System

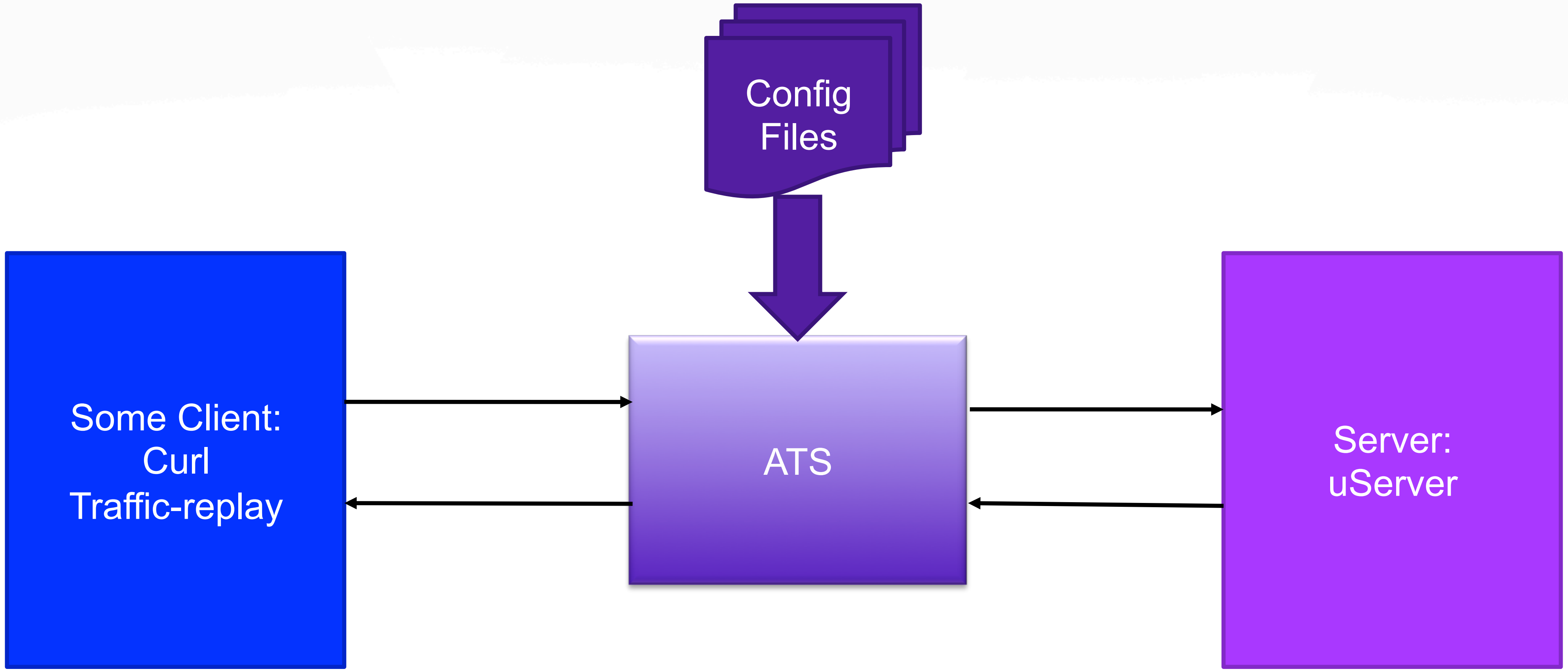
- It is a testing system for doing gold based testing
- Started as a testing system for Parts (an extension for SCons) when I was at Intel.
  - The testing system was useful in other projects at Intel, so I got permission to make it a separate project.
  - Then it was called gtest, not to be confused with Google's version, because of this I had to via Intel's lawyers call it Reusable Gold Testing System.
  - Since then various fixes have been made to better address needs to adapt testing of other types of systems

What has been Improved

- Better support for working with many different Processes
  - Ability to delay start of a process chain based on some condition, such as a port is open.
- Improvements to the gold tester:
  - Better support to wild card section with the {} characters
- Ability to define custom file type with special APIs
- Ability to write data to files on a certain event
- Updates to allow more extensibility of the system
- Standardized all events across the system.
- First stab at script generation to help replay any step for debugging purposes
- Other internal tweaks…

So what about the Tests for ATS?

Tests look like this...



Difference in writing a test –TSQA (1 of 2)

```
import os
import subprocess
import logging
import requests
import random
import uuid
import time

import helpers
import tsqa.test_cases
import tsqa.utils

log = logging.getLogger(__name__)


class TestCacheGeneration(helpers.EnvironmentCase):
    '''
    Test the cache object generation ID.
    '''

    def _fetch(self, path):
        url = 'http://127.0.0.1:{}/{}'.format(
            self.configs['records.config']['CONFIG']['proxy.config.http.server_ports'],
            path
        )
        log.debug('get {}'.format(url))
        return requests.get(url, headers={'x-debug': 'x-cache,x-cache-key,via,x-cache-generation'})

    def _dump(self, response):
        log.info('HTTP response {}'.format(response.status_code))
        for k, v in response.headers.items():
            log.info('    {}: {}'.format(k, v))

    def _ctl(self, *args):
        cmd = [os.path.join(self.environment.layout.bindir, 'traffic_ctl')] + list(args)
        out, _ = tsqa.utils.run_sync_command(
            cmd,
            env=self.environment.shell_env, stdout=subprocess.PIPE, stderr=subprocess.STDOUT
        )
        return out

    @classmethod
    def setUpEnv(cls, env):

        cls.configs['plugin.config'].add_line('xdebug.so')

        cls.configs['remap.config'].add_line(
            'map /default/ http://127.0.0.1/ @plugin=generator.so'
        )
…
```

Difference in writing a test – TSQA (2 of 2)

```
cls.configs['remap.config'].add_line(
        'map /generation1/ http://127.0.0.1/' +
        ' @plugin=conf_remap.so @pparam=proxy.config.http.cache.generation=1' +
        ' @plugin=generator.so'
    )
    cls.configs['remap.config'].add_line(
        'map /generation2/ http://127.0.0.1/' +
        ' @plugin=conf_remap.so @pparam=proxy.config.http.cache.generation=2' +
        ' @plugin=generator.so'
    )

    # Start with cache generation turned off
    cls.configs['records.config']['CONFIG']['proxy.config.http.cache.generation'] = -1
    # Wait for the cache so we don't race client requests against it.
    cls.configs['records.config']['CONFIG']['proxy.config.http.wait_for_cache'] = 1
    cls.configs['records.config']['CONFIG']['proxy.config.config_update_interval_ms'] = 1

  def test_generations_are_disjoint(self):
    """Test that the same URL path in different cache generations creates disjoint objects"""
    objectid = uuid.uuid4()

    # First touch is a MISS.
    ret = self._fetch('default/cache/10/{}'.format(objectid))
    self.assertEqual(ret.status_code, 200)
    self.assertEqual(ret.headers['x-cache'], 'miss', msg=ret)
    self.assertEqual(ret.headers['x-cache-generation'], '-1')

    # Same URL in generation 1 is a MISS.
    ret = self._fetch('generation1/cache/10/{}'.format(objectid))
    self.assertEqual(ret.status_code, 200)
    self.assertEqual(ret.headers['x-cache'], 'miss')
    self.assertEqual(ret.headers['x-cache-generation'], '1')

    # Same URL in generation 2 is still a MISS.
    ret = self._fetch('generation2/cache/10/{}'.format(objectid))
    self.assertEqual(ret.status_code, 200)
    self.assertEqual(ret.headers['x-cache'], 'miss')
    self.assertEqual(ret.headers['x-cache-generation'], '2')

    # Second touch is a HIT.
    ret = self._fetch('default/cache/10/{}'.format(objectid))
    self.assertEqual(ret.status_code, 200)
    self.assertEqual(ret.headers['x-cache'], 'hit-fresh', msg=ret.headers['x-cache'])
    self.assertEqual(ret.headers['x-cache-generation'], '-1')
```

Difference in writing a test - AuTest

```
import uuid
Test.Summary = '''
Test that the same URL path in different cache generations creates disjoint objects
'''
# need Curl
Test.SkipUnless(Condition.HasProgram("curl","Curl need to be installed on sytem for this test to
work"))
Test.ContinueOnFail=True
# Define default ATS
ts=Test.MakeATSProcess("ts")
# setup some config file for this server
ts.Disk.records_config.update({
            'proxy.config.body_factory.enable_customizations': 3,  # enable domain specific body
factory
            'proxy.config.http.cache.generation':-1, # Start with cache turned off
            'proxy.config.http.wait_for_cache': 1,
            'proxy.config.config_update_interval_ms':1,

        })
ts.Disk.plugin_config.AddLine('xdebug.so')
ts.Disk.remap_config.AddLines([
            'map /default/ http://127.0.0.1/ @plugin=generator.so',
            'map /generation1/ http://127.0.0.1/' +
            ' @plugin=conf_remap.so @pparam=proxy.config.http.cache.generation=1' +
            ' @plugin=generator.so',
            'map /generation2/ http://127.0.0.1/' +
            ' @plugin=conf_remap.so @pparam=proxy.config.http.cache.generation=2' +
            ' @plugin=generator.so'
        ])

objectid = uuid.uuid4()
```

```
#first test is a miss for default
tr=Test.AddTestRun()
tr.Processes.Default.Command='curl "http://127.0.0.1:{0}/default/cache/10/{1}" -H "x-debug:\
 x-cache,x-cache-key,via,x-cache-generation" --verbose'.format(ts.Variables.port,objectid)
tr.Processes.Default.ReturnCode=0
# time delay as proxy.config.http.wait_for_cache could be broken
tr.Processes.Default.StartBefore(Test.Processes.ts,ready=2)
tr.Processes.Default.Streams.All="gold/miss_default-1.gold"
# Same URL in generation 1 is a MISS.
tr=Test.AddTestRun()
tr.Processes.Default.Command='curl "http://127.0.0.1:{0}/generation1/cache/10/{1}" -H "x-debug:\
 x-cache,x-cache-key,via,x-cache-generation" --verbose'.format(ts.Variables.port,objectid)
tr.Processes.Default.ReturnCode=0
tr.Processes.Default.Streams.All="gold/miss_gen1.gold"
# Same URL in generation 2 is still a MISS.
tr=Test.AddTestRun()
tr.Processes.Default.Command='curl "http://127.0.0.1:{0}/generation2/cache/10/{1}" -H "x-debug\
: x-cache,x-cache-key,via,x-cache-generation" --verbose'.format(ts.Variables.port,objectid)
tr.Processes.Default.ReturnCode=0
tr.Processes.Default.Streams.All="gold/miss_gen2.gold"
# Second touch is a HIT for default.
tr=Test.AddTestRun()
tr.Processes.Default.Command='curl "http://127.0.0.1:{0}/default/cache/10/{1}" -H "x-debug:\
 x-cache,x-cache-key,via,x-cache-generation" --verbose'.format(ts.Variables.port,objectid)
tr.Processes.Default.ReturnCode=0
tr.Processes.Default.Streams.All="gold/hit_default-1.gold"
```

Difference in what is tested – What is the Same

Both tests same logic of

- Cache test of:
  - Default url - cache miss and cache generation code of -1
  - Gen1 url - cache miss and cache generation code of 1
  - Gen2 url - cache miss and cache generation code of 2
  - Default url – cache hit and cache generation code of -1

Difference in what is tested – What is added

Autests also tests:

- Traffic_server started
- Traffic_server ran with no errors in diags.log
  - It did not exit early with some return code error
- Running on default values (TSQA can get corrupted with modified config files in the build)
- Gen1 and 2 second pass have hits
- Request ran with out issue
  - Return codes checked
- Header is checked for a number of other differences based on the gold file
- Tests all cases as sometimes cache might miss and hit other case (not seen in TSQA case)

Basic test – test that Traffic server runs

```
Test.Summary = '''Test that Trafficserver starts with default configurations.'''
# Do we have the requirements we need to run the test
Test.SkipUnless(Condition.HasProgram("curl","Curl need to be installed on sytem for this test to work"))
# Define a trafficserver process
p=Test.MakeATSProcess("ts")
# Create a test run so we can test something
t = Test.AddTestRun("Test traffic server started properly")
# add check that is still running after this step is finished
t.StillRunningAfter = Test.Processes.ts
# define the default process to run curl based on the dynamic port for this test
t.Processes.Default.Command = "curl http://127.0.0.1:{0}“.format(p.port)
# test that curl did not fail
t.Processes.Default.ReturnCode = 0
# make sure the traffic server process start before the curl command,
#and that the curl command run once the correct port is ready
t.Processes.Default.StartBefore(Test.Processes.ts, ready = When.PortOpen(Test.Processes.ts.port))
```

Extension added

- Commandline for point to your ats install i.e. --ats-bin <pathto/bin>
  - Has error check to make sure path is good
  - Has check to make sure traffic_layout is not broken
- Variables are added to store values from traffic_layout
- API function to allow creating an environment for ATS to run in based on default values compiled in the system. Not one defined in the builds config files.
- Functions to deal with config files
- Functions to create and deal with origin server

Extensions : ATS server

- Define a traffic server process
  - MakeATSProcess(name,command=[traffic_server],select_ports)
  - Process that is returns has file objects for:
    - Log files
      - Squid.log
      - Error.log
      - Diag.log
    - Config files
      - Records.config
      - Cache.config
      - Congestion.config
      - … Should have all of them.
  - Defines attributes with ports values:
    - port – value of ipv4 port
    - Portv6 – value of ipv6 port
    - manager_port – manager port for communication
    - admin_port – admin port
  - Process Environment has all the variable set to run ATS in the sandbox correctly

Extensions : ATS server (cont.)

- Special files types for records.config.

  Records.config example1:

```
ts=Test.MakeATSProcess("ts",command="traffic_manager")

# setup some config file for this server
ts.Disk.records_config.update({
        'proxy.config.body_factory.enable_customizations': 3,
        'proxy.config.http.cache.generation':-1,
        'proxy.config.config_update_interval_ms':1,
    })
```

  Records.config example2:

```
ts.Disk.records_config['proxy.config.config_update_interval_ms']=3
```

Extensions : ATS server (cont.)

- General config files can add lines.

Example 1:

```
ts.Disk.remap_config.AddLines([
        'map /default/ http://127.0.0.1/ @plugin=generator.so',
        #line 2
        'map /generation1/ http://127.0.0.1/' +
        ' @plugin=conf_remap.so @pparam=proxy.config.http.cache.generation=1' +
        ' @plugin=generator.so',
])
```

Example 2:

```
ts.Disk.remap_config.AddLine('map /default/ http://127.0.0.1/ @plugin=generator.so‘)
```

Copy config files

- Copy a config file:

  CopyConfig(from, asfile)

  Example:

```
ts1 = Test.MakeATSProcess("ts1",select_ports=False)
ts1.Setup.ts.CopyConfig('config/records_8090.config','records.config')

ts2 = Test.MakeATSProcess("ts2",select_ports=False)
ts2.Setup.ts.CopyConfig('config/records_8091.config','records.config')
```

Create an origin server

- Define a origin server
  - MakeOriginServer(name,public_ip)
  - Server has extra API (to grow as we need) such as *addResponse(…)*

    Example:

```
server=Test.MakeOriginServer("server")

request_header={"headers": "GET / HTTP/1.1\r\nHost: www.example.com\r\n\r\n", "timestamp": "1469733493.993", "body":
""}

response_header={"headers": "HTTP/1.1 200 OK\r\nConnection: close\r\n\r\n", "timestamp": "1469733493.993", "body": ""}

#add response to the server dictionary
server.addResponse("sessionfile.log", request_header, response_header)
```

Still plan to add to the extension

- Items based on feedback
- A curl function to make it easier to define the curl command line
- Server response API to take wild cards
- TrafficReplay clients functions

What is next?

- Need to fix –J option
- Add various APIs for controlling streams output better
- Fix some bugs
- Add some reporting tweaks and features
- Add some common report format generators to allow data to be used by existing dashboard systems
- Address user feedback

- Give feedback! Write Tests! System will only get better if we all help.

YAHOO!

Demo

- Show details of what happens

- QUESTIONS??

Thank you!!

- Feedback welcome!
- Links:
  - https://bitbucket.org/dragon512/reusable-gold-testing-system (MIT license)
  - Documentation: https://bitbucket.org/dragon512/reusable-gold-testing-system/wiki/Home
  - Tests until pull request is done:
    - https://bitbucket.org/dragon512/ats_tests

YAHOO!