**During Upgrade Tests Suite**

The during-upgrade tests suite was created to measure the uptime of critical services while a project is upgrading from branch to branch. The test has been constructed using python curl calls (urllib2, request) to minimize extra layers.

**Pipeline Flow:**

1. Start During Tests
2. Upgrade Begins
3. Upgrade Ends
4. Stop During Tests
5. Aggregate results

**Project’s Tested:**

* Swift
* Nova
* Keystone

**Test Scenario Used:**

* Swift

1. Create container
2. Create object
3. Delete object
4. Delete container

Goals:

Verify core services s-proxy, s-acct, s-object, s-container are functioning properly.

* Nova

1. Create VM
2. Delete VM

Goals:

Verify n-cpu, n-con, n-api functioning properly

* Keystone

1. Generate token
2. Validate token

Goals:

Verify Keystone is up and able to generate and validate tokens.

**To do:**

* Swift

1. The Swift team would like to identify a way to validate the background processes are running
2. The Swift team would like the api uptime test (another suite) to use a health check as opposed to doing a get\_account call.

* Nova

1. We still need to speak to them to discuss testing intervals and accurately measuring down time.

* Keystone

1. Verified that this is the right testing made a few comments about using version 3 instead of version 2
2. Would like to run the test.

**How the tests are run (test run simultaneously):**

* Clone the repo
* Set the parameters, see here **https://github.com/osic/rolling-upgrades-during-test/blob/master/README.md**
* python call\_test.py –d
* Create /usr/during.uptime.stop to stop
* Swift

1. Get a token
2. Get the swift url

* If the Swift url is not available alert that Swift may not be installed. End Test

1. Create container – if returns (201, 202) success
2. Create Object – if returns (201,202) success
3. Delete Object – if returns (204) success
4. Delete Container – if returns (204) success
5. Send status
6. Check for STOP file (if there stop)
7. Repeat

* Nova

1. Get a token
2. Get the Nova url

* If the Nova url is not available alert that Nova may not be installed. End Test

1. Create a VM

* Check VM status for ‘ACTIVE’ or ‘ERROR’ (if ‘ERROR’ fail of course)
* If status is not set to ‘ACTIVE’ in (x) amount of time send fail
* Once fail is sent check if VM has been created if no then 5, else 4

1. Delete VM – if gets status code (204) success

* If it doesn’t get status code 204 it will send fail and keep attempting
* It will not create an extra VM

1. Send status
2. Check for STOP file (if there stop)
3. Repeat

* Keystone

1. Get a token – if no (503, 404) success
2. Validate token – if no (503, 404) success
3. Send status
4. Check for STOP file (if there stop)
5. Repeat

**How are Results Stored:**

Aggregated Results – /output/during.uptime.out

* After the stop file has been created the test will stop and all results will be aggregated
* With each iteration of a scenario a True or False (1 or 0) was appended to the pipeline
* At the end of a run the data is aggregated combined together and reported back in json

Status files – /output/<project>\_status.json

* A 1 or 0 and the time are sent for each run of a scenario to a file so that we can identify downtime to the minute (second in some cases). These are json files as well.
* Different from the test above because they are written to a file during the test.

Ex. Aggregated:

{"keystone": {"successful\_requests": 1, "down\_time": 0, "start\_time": "2016-11-29T14:36:14", "project": "keystone", "success\_pct": 100.0, "end\_time": "2016-11-29T14:36:15", "uptime\_pct": 100.0, "total\_requests": 1, "failed\_requests": 0}, "swift": {"successful\_requests": 1, "down\_time": 0, "start\_time": "2016-11-29T14:36:14", "project": "swift", "success\_pct": 100.0, "end\_time": "2016-11-29T14:36:16", "uptime\_pct": 100.0, "total\_requests": 1, "failed\_requests": 0}, "nova": {"successful\_requests": 1, "down\_time": 0, "start\_time": "2016-11-29T14:36:14", "project": "nova", "success\_pct": 100.0, "end\_time": "2016-11-29T14:36:33", "uptime\_pct": 100.0, "total\_requests": 1, "failed\_requests": 0}}

Ex. Status:

{"status": 1, "timestamp": "2016-11-29T14:36:15", "service": "swift"}

\*\*ElasticSearch parsing script parses the json results and pushes them to ElasticSearch.

**How is the uptime percentage calculated:**

1. Before a test runs a timer starts
2. After the test is finished this timer stops

The time gets appended to a variable aggregating the total time of the test

* duration

1. If the test fails, another variable is also populated which aggregates all of the down time

* total\_down\_time

1. The uptime percentage is calculated as

* ((total\_down\_time/duration) \* 100)

\*\*\*Still working to find best solution all suggestions are acceptable.

**Tips for Debugging:**

call\_test.py

* If it fails in call test debug it sequentially

test\_<project>.py

* If it fails in test, start debugging by going to the actual test in the test file and following it
* test\_<scenario>