Integration Testing Pipeline

Using SWIM to test SWIM
By Shovik Guha

What I set out to do initially

- Build a general pipeline for testing SWIM applications over time
- What is a test? (short term vs. long term)
- Goals: Autonomous, Configurable, Transportable
- Wanted an objective process for testing

Tools I used

- Swimshd, made modifications to fit my use case, exit after didSync()
- JMX buoy, made modification to generalize for Canary buoy
- Python, parse data and generate graph
- Shell script, used as wrapper

Test Structure

- Repo contains general outlines for ways to write the tests
- Main and Simulator Separation, internal vs. external
- Functional, unit like tests, vs long running metric collection
- Shell / Python scripts also are expandable

Running the Example Tests

- Install jmx via swimshd
- Confirm with “ss -l”
- Include jmx args in VM arguments of program you want to inspect
- Specify duration of the test as a program argument
- Run the getData scripts, use nohup and &
- Parse data with python after you are done collecting

Running an Actual Test on the Dev Site

- Separate Main class from Simulator
- Follow exact same process as in the previous slide

Analyzing the Results

- Run the Python parser to produce the graphs
- Graphs and runtime context stored in directory, can scp directory to view
- This makes is to tests can be run on remote servers, docker containers, raspberry pis, etc.
- Test Server produces an example of a “good” app
- Running the test on the dev site shows that there is a memory leak present

Canary (Work in Progress)

- Realtime component of the testing pipeline
- Does basic forecasting with EWMA on CPU usage, alerts when CPU is tending high
- Generalized the Canary by modifying cast / mold patterns
- Added Heap Usage observer, alerts when heap usage is high relative to the amount committed
- Can expand to any of the fields viewable through JMX

Next Steps

- Running multiple tests in parallel (RPI’s / docker)
- Jenkins integration
- UI

Thanks!