Fault Injection Presentation

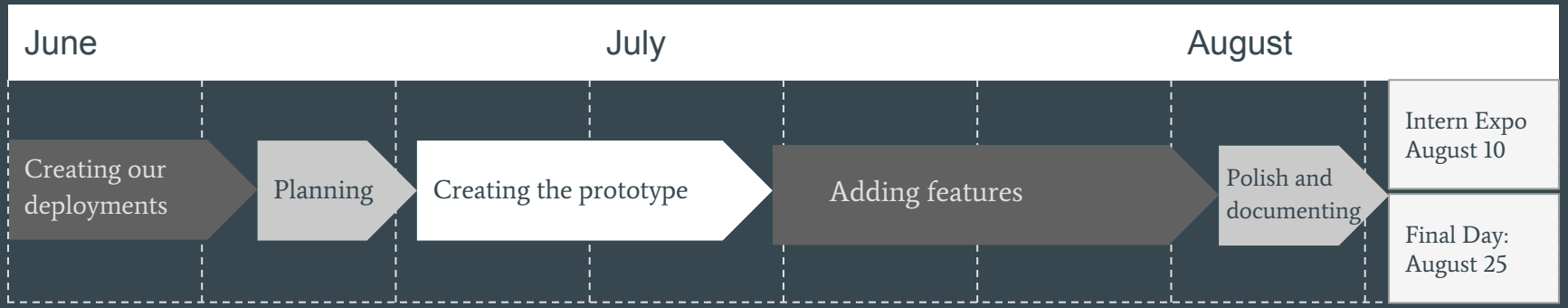
Daniel Pivonka & Steven Brzozowski
July 21, 2017

Rev 01

Objective

Create a fault insertion mechanism that can run concurrently with any deployed cloud environment that will help test and validate the fault resilience built into the deployments

Timeline



- Deployed with HCI and learned how to diagnose failed deployments

- Created documents / flowcharts fleshing out what we want to do

- Followed the planning documents to make a functioning fault injector which can cause node and service faults while monitoring the state of the deployment

Plan to add:

- Extended setup scripts
- Fully fleshed out stateful/stateless modes
- Refine structure to make it easier to develop functionality in the future

- Develop guides for future developers to adopt within their development efforts

Goals

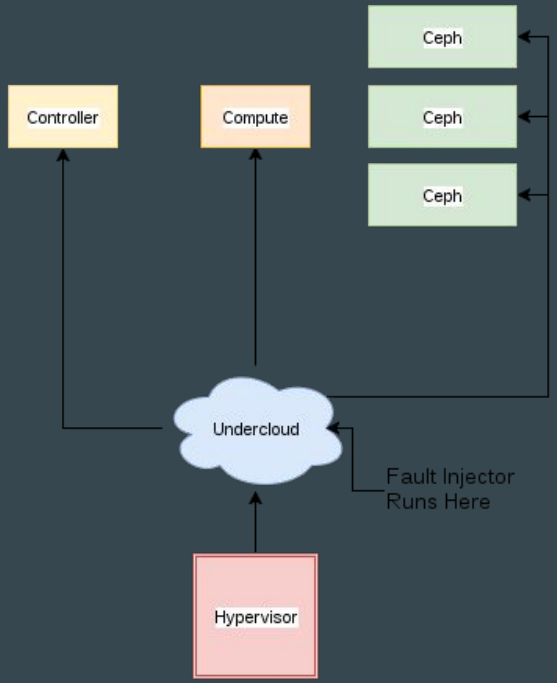
1. Build a functioning prototype and gather initial feedback
2. Demonstrate how this tool can and should be used
3. Design it to be easily extended
4. Develop our tool to work with the various deployments (i.e. HCI, containerized, etc.)
5. Run load generation and look for anomalies in the expected application resiliency
6. Create documentation

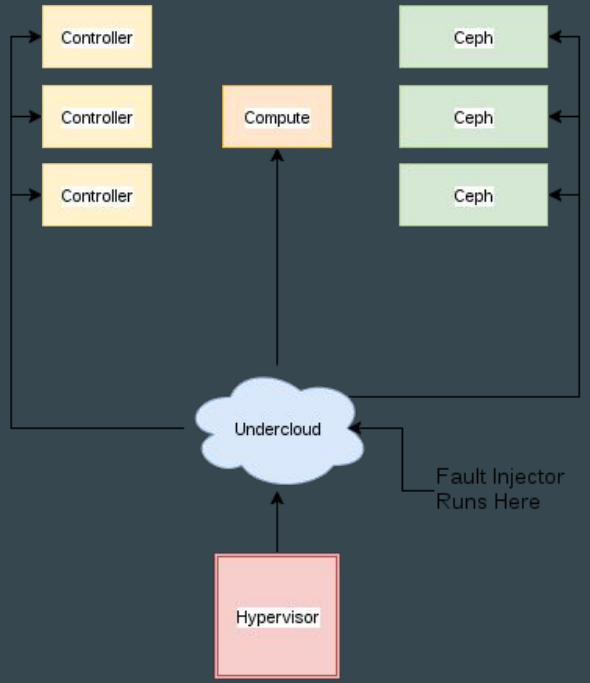
Non-Goals

- Build our own monitoring/load generation tools
  - There are already suitable tools which can run alongside our solution
    - For our deployment, we use CBT to generate load and CollectD to collect statistics for monitoring
- Run node and service fault domains simultaneously
  - Raises the overall complexity greatly for a relatively small benefit
- Create several plugins
  - Focus on Ceph, creating a base on which to build

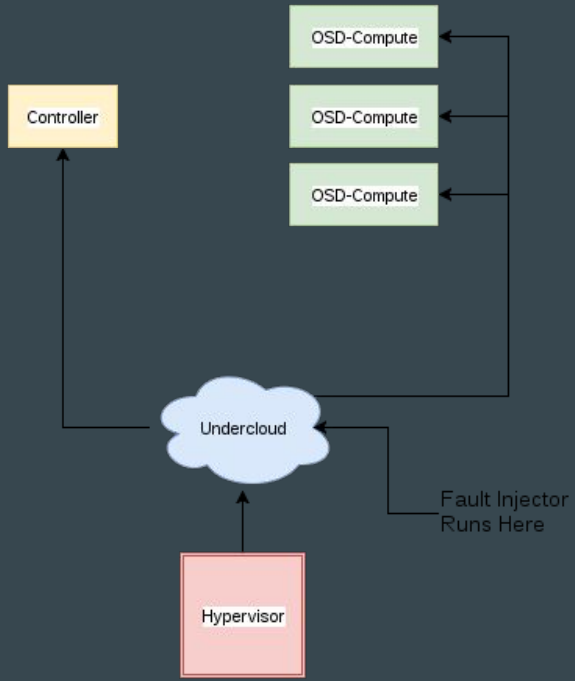
How

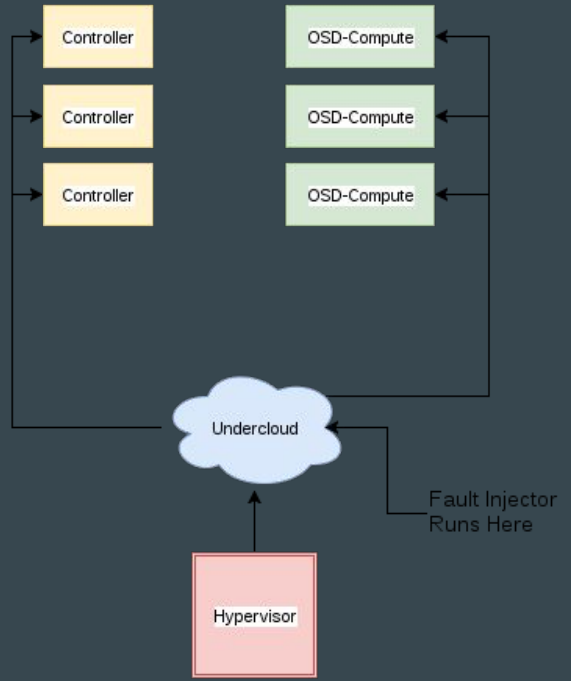
- Create an extendable framework using Python and Ansible Playbooks which can be extended through the use of plugins
- This framework can be run standalone, but will also allow performance testing and monitoring tools to run alongside
- The program has the ability to record and repeat specific runs to rule out the possibility of random chance causing a failure.





Deployed Environment - Standard





Deployed Environment - Hyper-Converged

Fault Injection Modes

Stateless

- Entire pieces of hardware selected for fault
- Unaware of the services running on the hardware
- Up to *n* faults executed at a time

Stateful

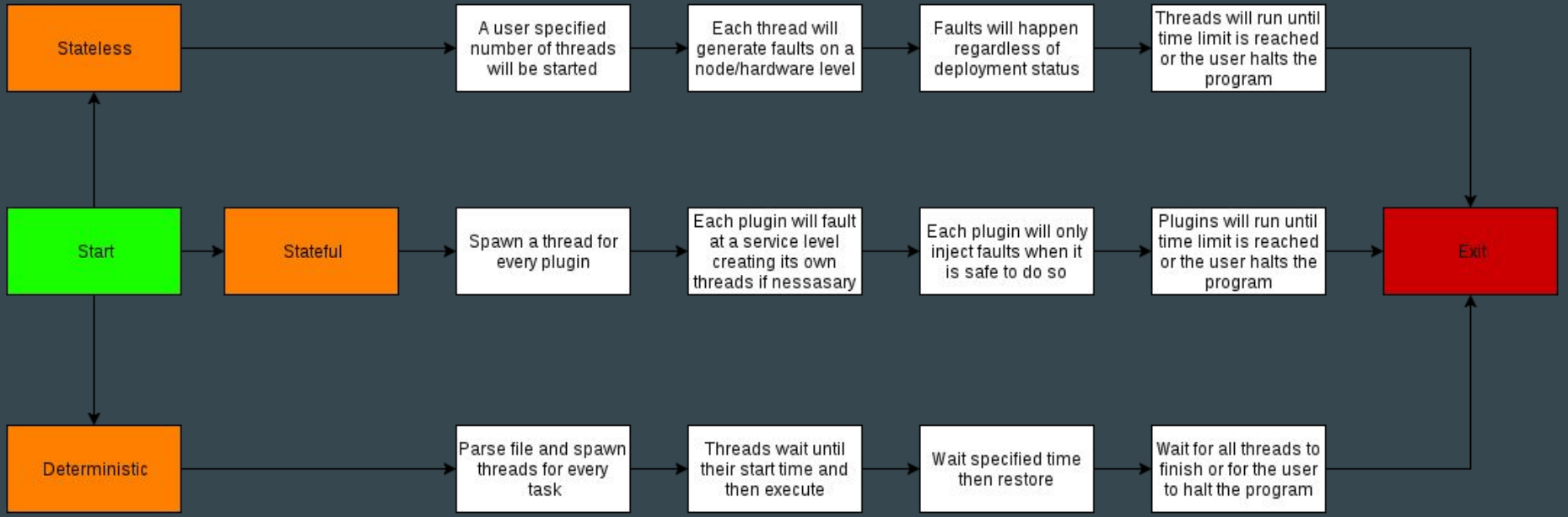
- Aware of the environment's current state
- Faults services and is aware of the service resiliency. Multiple services can be faulted concurrently

Deterministic

- All stateful and stateless runs are logged to their own individual files
- Example of file on next slide

Domain	Type	Target IP	Start Time	End Time	Wait Time (s)	Additional Data
Ceph	ceph-osd-fault	192.168.24.16	0:01:04.3222	0:06:20.5626	415	0
Ceph	ceph-mon-fault	192.168.24.19	0:03:04.2623	0:07:13.3986	470	-
Ceph	ceph-osd-fault	192.168.24.16	0:02:27.1999	0:07:41.3772	520	3
Ceph	ceph-mon-fault	192.168.24.19	0:07:12.6922	0:09:20.6666	260	-
Ceph	ceph-mon-fault	192.168.24.11	0:07:45.7628	0:09:55.3162	23	-
Ceph	ceph-osd-fault	192.168.24.6	0:07:41.4763	0:10:09.5025	160	2
Ceph	ceph-mon-fault	192.168.24.18	0:10:12.7892	0:11:23.7090	100	-
Ceph	ceph-osd-fault	192.168.24.6	0:06:28.6844	0:11:34.5337	450	5

Deterministic Mode File



Fault Injector Diagram

Generating Load

- Leave load generation up to the fault plugin creators
- As the creators of the Ceph fault plugin, we will use the Ceph Benchmarking Tool to generate realistic load

Monitoring Example

We use Grafana and a CollectD plugin for Openstack

Extendability

Currently our framework contains a template fault class in addition to single Ceph-based fault class. However, The system is designed to support multiple plugin types running simultaneously.

Framework

- Deployment class for storing attributes of deployment
  - The deployment class contains a list of Node objects
  - Each node object stores attributes of the node

- Fault class is a base class for creating faults
  - The fault class contains the deployment object
  - The fault class contains functions for stateless, stateful, and deterministic modes that can be implemented when applicable
  - Each custom fault class would define functions to execute the fault

- The setup.py tool provided gathers data on Openstack deployments
  - The tool will create the config.yaml needed for our tool
  - The tool has the -c flag that adds ceph information to the config.yaml

Within The Next 6 Weeks...

Planning to:

- Add support for containerized deployments
- Add additional features
- Polish our tool
- Create documentation
- Run our tool alongside QE tests over a long period of time on large scale deployment

Not Planning to:

- Support versions of Openstack prior to Ocata although our tool may work with older versions