Fault Injection

Architecture

blocked URL

Trogdor

The Fault Injector is named Trogdor.

Daemons

Trogdor has two Java daemons: an agent and a coordinator.

The agent process is responsible for actually implementing the faults. For example, it might run iptables, send signals to processes, generate a lot of load, or do something else to disrupt the computer it is running on. We run an agent process on each node where we would like to potentially inject faults. So it will run alongside the brokers, zookeeper nodes, etc.

The coordinator process is responsible for communicating with the agent processes and for scheduling faults. For example, the coordinator can be instructed to create a fault immediately on several nodes. Or it can be instructed to create faults over time, based on a pseudorandom seed.

Both the coordinator and the agent expose a REST interface that accepts objects serialized via JSON. There is also a command-line program which makes it easy to send messages to either one without manually crafting the JSON message body.

FaultSpecs

A FaultSpec is the specification for a fault. For example, this fault spec describes a network partition:

```
{
    "class": "org.apache.kafka.trogdor.fault.NetworkPartitionFaultSpec",
    "startMs": 1000,
    "durationMs": 30000,
    "partitions": [["node1", "node2"], ["node3"]]
}
```

FaultSpecs should always be serializable to and from JSON. Fault specs should be platform independent: notice that there was nothing about iptables, or Windows, or anything platform specific in the NetworkPartitionFaultSpec.

All fault specifications have:

- A "class" field describing their type. This contains a full Java class name.
- A "startMs" field describing when the fault should start. This is given in terms of milliseconds since the UNIX epoch.
- A "durationMs" field describing how long the fault should last. This is given in terms of milliseconds.

In general, we plan on customizing FaultSpecs to Apache Kafka. So there should be a fault spec for cutting off communication with ZooKeeper, a fault spec for suspending the active controller, and so forth.

Platform

Trogdor's Platform class encapsulates supports for a particular computing environment.

There are at least two important computing environments for us:

- BasicPlatform (the standard Linux platform)
- Windows (eventually)

Node

A node is an individual element of a cluster. Each node has a unique name. Currently, we assume that all the nodes in the cluster are known ahead of time, and that a trogdor agent process is running on each node.

Fault

Fault objects implement particular induced problems. Each fault object has:

- An ID string which uniquely identifies it
- A FaultSpec describing what it should do
- A set of target node names describing the nodes which are affected by it

Unlike the FaultSpec class, the Fault class is platform-specific. For example, a network partition fault may be implemented differently on the cloud versus on a local Linux environment, even though the specification is the same.

It makes sense to use composition to create more specific faults out of more general faults. For example, a fault for suspending the active Kafka controller could be built off of a fault for suspending a generic process. A fault for partitioning the ZooKeeper nodes from the broker nodes could be built on top of the generic network partition fault.

Code

The code for Trogdor lives under `./tools/src/main/java/org/apache/kafka/trogdor.`

There are several directories:

- common: common utilities and functions.
  - SignalLogger: logs a helpful message when we shut down because of a signal such as SIGTERM, SIGSEGV, etc.
- coordinator: the code for the Trogdor coordinator.
  - NodeManager: a thread which manages the coordinator's communication with a particular agent
  - CoordinatorClient: a command-line program for sending messages to the coordinator
- agent: the code for the Trogdor agent.
  - AgentClient: a command-line program for sending messages to the agent
- fault: code for describing and implementing faults
  - FaultSet: a convenient way of storing and iterating over faults sorted by start and end time
- rest: code for implementing REST communication, including message types, etc.
  - RestExceptionMapper: maps Java exceptions to HTTP status codes, in an attempt to make error reporting clearer
- basic: an implementation of Platform for a basic Linux environment

Ducktape Integration

What if you want to perform fault injection as part of a system test? For this case, we need ducktape integration.

TrogdorService

There is a TrogdorService which will handle starting the Trogdor coordinator and Trogdor agents for you. This is similar to the existing KafkaService, ZooKeeperService, etc. classes which handle starting other daemons.

However, unlike other service daemons, TrogdorService operates on nodes which already have a service running. That is why the constructor of TrogdorService takes an agent_nodes parameter. TrogdorService will start Trogdor agents on the agent_nodes that are passed in, but it will not take "ownership" of these nodes. The services which originally allocated those nodes (KafkaService, ZookeeperService, etc.) are responsible for de-allocating them. For this reason, TrogdorService should be shut down before other services.

TrogdorService allocates a single node of its own for the Trogdor Coordinator. As mentioned earlier, it is better to run the coordinator on its own node so that it is not affected by the faults that it is creating elsewhere in the cluster.

Changes to AK Ducktape Services

Some of the ducktape service implementations assume that they are the only thing running on their node. They make assumptions which generally fall into two categories:

- Assuming that their service is the only thing using `/mnt`.
  - For example, KafkaService runs "`rm -rf /mnt/*`" when shutting down.
  - Another example is that JMXService is hard-coded to write to `/mnt/jmx_tool.log`.
- Using `killall java` or similar shell commands which may affect processes not managed by the service
  - There was an example of this in KafkaService (a little more subtle than killall, but the same idea)

We need to fix these services so that TrogdorAgent can run on the same node. The fixes are mostly easy:

- Namespacing temporary files
  - For example, creating Kafka files in /mnt/kafka rather than /mnt, and cleaning up by running rm -rf /mnt/kafka rather than rm -rf /mnt/*

- Introducing more precise ways of managing processes
    - For example, adding a command to use `jcmd` to check for java processes, rather than `ps aux`

Upgrading to the Latest Ducktape

There has not been a new ducktape release in a while. Apache Kafka is still using the 0.6.0 release of ducktape, which was made in mid-2016.

To upgrade AK to the latest ducktape, we need several things

- A way of quickly running the latest ducktape and latest AK together in a docker container. This was added in a few pull requests to ducker-ak.
- The cluster node allocation code in Ducktape has been modified to take operating system into account.
    - There are some incompatibilities between this and the AK code.
    - Also, the ducktape code should be revised a bit to be more generic. It should allow for the possibility to allocate nodes based on tags in the future, rather than hard-coding allocation based on operating system.
- There are some minor miscellaneous version incompatibilities that need to be fixed.
- A new ducktape release will need to be made

Kibosh

Overview

Kibosh is a fault-injecting filesystem for Linux. It is written in C using FUSE.

Kibosh acts as a pass-through layer on top of existing filesystems. Usually, it simply forwards each request on to the underlying filesystems, but it can be configured to inject arbitrary faults.

Injecting Faults

Faults are injected by writing JSON to the control file. The control file is a virtual file which does not appear in directory listings, but which is used to control the filesystem behavior.

Example

```
# Mount the filesystem.
$ ./kibosh /kibosh_mnt /mnt

# Verify that there are no faults set
$ cat /kibosh_mnt/kibosh_control
{"faults":[]}

# Configure a new fault.
$ echo '{"faults":[{"type":"unreadable", "code":5}]}' > /kibosh_mnt/kibosh_control
```

Code

Kibosh is written using the Linux kernel C coding style. The only exception is that tabs are 4 spaces and do not use a hard tab character.

We try to minimize the native dependencies that are used. Kibosh has three dependencies:

- The CMake build system
- libfuse
- json-parser

For convenience, the source code for json-parser is included in the main repo as `json.c` and `json.h`. It is BSD licensed.

Code here: https://github.com/confluentinc/kibosh

- fault.c: stuff for parsing fault JSON and implementing faults
- fault_unit.c: unit tests for the fault stuff
- file.c: implements the FUSE operations directly dealing with file writing, reading, etc.
- fs.c: deals with allocating `struct kibosh_fs` objects

Pull Requests

- Ducktape PRs
    - https://github.com/confluentinc/ducktape/pull/168
    - Run AK system test builder against our new version of ducktape (in progress)
- Trogdor PRs
    - https://github.com/apache/kafka/pull/3699
- Kibosh PRs
    - https://github.com/confluentinc/kibosh/pull/1