OpenWhisk on Mesos

Tyson Norris/Dragos Dascalita Haut, Adobe Systems, Inc.
OPENWHISK ON MESOS
OPERATIONAL EVOLUTION
CUSTOMER FOCUSED DEVELOPMENT
CUSTOMER FOCUSED EXTENSIONS
ALEXA SKILLS - A SERVERLESS USE CASE

- Demo
- Associate Alexa skills with serverless functions
I feel like we should speak maybe for a 3-4 minutes why do we care so much about serverless and end that with this demo, and then get into more details of how. WDTY? I could cover the motivation part too.

Tyson: not just why serverless is important for us, but also "why running your own platform" (Serverless infrastructure vs app specific serverless)
OPENWHISK

“Apache OpenWhisk is a serverless, open source cloud platform that executes functions in response to events at any scale.”

Or: functions in docker containers

And: CLI + API = function management
OPENWHISK CONCEPTS

- Controller + Invoker
- Execution flow
OPENWHISK SCALING

- Add an Invoker
- Invoker will advertise itself via Kafka
- Controller will register it and monitor its health

But...

- We’re a Mesos shop.
  - We don’t want competing cluster (or container) managers
  - Can I use mesos to manage the cluster?
  - Can I use mesos to control my containers that invoke actions?
YES!

WITH SOME MINOR CHANGES
HOW TO MESOS IN AKKA

- OpenWhisk is an Akka application (Scala)
- OpenWhisk uses docker to launch containers
- Put a Mesos Actor in there!

But...

- What Mesos Actor?
MESOS FRAMEWORKS

MESOSIFICATION IN 3 STEPS

▸ Build a Mesos Actor
▸ Make OpenWhisk extensible
▸ Launch Mesos Tasks
A MESOS ACTOR IN AKKA

- Existing libraries didn’t fit (Scala, HTTP API)

Let’s build a new one!

- Akka HTTP + Akka Streams
- https://github.com/adobe-apiplatform/mesos-actor
A MESOS ACTOR IN AKKA

- Mesos HTTP + Protobuf messages → Scala classes + Akka Messages
A SHORT DEMO

- Mesos cluster using docker-compose
- SampleFramework
  - TaskMatcher – matching pending tasks to offers – default is “first match”
  - TaskBuilder – building TaskInfo protobuf from requirements – default is “verbatim”
- K.I.S.S. – A simplified interface to Mesos task launching
- DEMO
PENDING WORK

**MESOS-ACTOR TODO**

- Framework ID persistence (zookeeper, etc)
- Reconcile on startup (next update received will reset tasks)
- HA (multiple framework instances, leader election, shared task state - based on Akka Clustering)
- Multi-master redirect (redirect to current leader)
- Re-subscribe after disconnect (in master/framework failover)
- Mesos roles
A SERVERLESS PLATFORM ON MESOS

- YOUR servers
- YOUR Mesos cluster
- Operators expand/contract Mesos cluster
- OpenWhisk Alterations
  - Controller – none
  - Invoker – ContainerFactory
Original OpenWhisk behavior

- Client communicates with the controller via HTTP.
- The controller interacts with Kafka for activation.
- Invoker receives invoker health updates from the invoker host.
- Docker manages containers on the invoker host.
- Containers communicate with the invoker via HTTP.
OPENWHISK CHANGES

OPENWHISK INVOKER

- Invoker changes
  - Deploy: 1 per cluster (instead of 1 per host!)*
  - Configure: MesosContainerFactory SPI (instead of DockerContainerFactory)
  - Configure: LogDriverLogStore SPI (instead of DockerLogStore)

* HA? Details later
INVOKER - AFTER

Use mesos for container scheduling

- controller host
- invoker host
- mesos cluster

- client
- controller
- mesos
- invoker
- docker
- container

http

activation
HA IN THE AKKA CLUSTER
1. AKKA CLUSTER

- Shared container state
  - All cluster nodes have visibility to same container pool
- 1 leader
  - Single leader is responsible for streaming Mesos messages from master
2. FRAMEWORK FAILOVER

- Shared container state
  - Containers remain when scheduler stream is interrupted
  - New Containers cannot be created until new scheduler instance resumes
  - *Mitigate downtime with “prewarm containers”*
- 1 leader
  - New leader is elected by Akka
  - New leader will create new subscription to Mesos master with the same FrameworkID
  - Reconcile existing tasks
  - Resume managing containers
OPENWHISK CHANGES

OPENWHISK DETAILS
SPI - service provider interface: “Service Provider Interface (SPI) is an API intended to be implemented or extended by a third party. It can be used to enable framework extension and replaceable components.”

```scala
trait ContainerFactory {
}

trait LogStore {
  def containerParameters: Map[String, String]
  def logs(activation: WhiskActivation): Future[ActivationLogs]
  def collectLogs(container: Container, action: ExecutableWhiskAction)
    (implicit transid: TransactionId): Future[Vector[String]]
}
```
OPENWHISK CHANGES

OPENWHISK ALTERATIONS

- ContainerFactory SPI (PR#2659)
  - MesosContainerFactory (coming soon)
- LogStore SPI (PR#2695)
  - SplunkLogStore