Fault tolerant frameworks: Making use of CNI without Docker

Aaron Wood
Principal Software Engineer
Verizon Labs

Tim Hansen
Senior Software Engineer
Verizon Labs

September 15, 2017
Making the framework: V0 or V1?

V0

Requires C++ or bindings
- Java/Scala and Python bindings provided by Mesos
- Not the recommended path for development

V1

Streaming HTTP
- Allows for compression

No bindings required
- Use any language you want

JSON or Protobuf payloads between scheduler and Mesos

Obvious choice was V1 with compressed Protobuf payloads
Making the framework: Language

Go as scheduler, executor and accompanying SDK

Why Go?

- Increased developer speed
- Fast and light on memory
- Excellent concurrency primitives
- Single statically-linked binary for the scheduler and executor

Language of choice for Verizon Labs greenfield projects

Note: Gopher image created by Renee French taken from https://golang.org/doc/gopher/
Making the framework: SDK

All common functionality moved to a separate SDK

Reduced boilerplate

Required code that any scheduler can handle

Task lifecycle management

Resource (Offer) lifecycle management

RecordIO event decoder

Streaming HTTP
  • Scheduler and executor calls
  • Client with leader detection

Persistence (storage backend)

Scheduler, executor and common protobufs
Containers are containers.

There’s no real need for an extra daemon and client when Mesos can containerize tasks.
Isolation: Containers without Docker

**Pros**

Docker + dependencies don’t need to be installed, managed and patched/upgraded across the cluster

No more blocking if the Docker daemon gets stuck

Reduced attack surface

No need to manage “secrets” (aka encoded JSON) or setup an external credentials store

Broader support for container image specifications
Isolation: Containers without Docker

Cons

User namespaces currently not supported

Seccomp currently not supported

Showstopper bugs if using Mesos < 1.2.x

Image backends
  • MESOS-6875
  • MESOS-5028
  • MESOS-6327
  • MESOS-7280

Whiteout files
  • MESOS-6002
  • MESOS-6360
DEMO

g: build scheduler/main/main.go
  g: sched 6
  [1] 53635
[1] [INFO] [53635] unknown sched[0]:main.go:71][2017/08/28 17:47:03.04236942][Starting executor file server
[1] [INFO] [53635] unknown sched[0]:main.go:72][2017/08/28 17:47:03.04237470][Starting API server
[1] [INFO] [53635] unknown sched[0]:main.go:74][2017/08/28 17:47:03.04257466][Starting leader election socket server
[1] [INFO] [53635] unknown sched[0]:main.go:120][2017/08/28 17:47:03.04647738][we're leading.
[1] [INFO] [53635] unknown sched[0]:controller.go:106][2017/08/28 17:47:03.046023777][Restoring any persisted state from data store
[1] [INFO] [53635] unknown sched[0]:controller.go:114][2017/08/28 17:47:03.05012177][Starting periodic reconcile thread with a 15 minute interval.
[1] [INFO] [53635] unknown sched[0]:controller.go:282][2017/08/28 17:47:03.05378938][Subscribed with an ID of 00ad548b-7f65-4370-b2ef-af9e635e4a5-0002
[1] [INFO] [53635] unknown sched[0]:controller.go:42][2017/08/28 17:47:03.05932394][Not reconciling; Task manager is empty
[1] [INFO] [53635] unknown sched[0]:controller.go:42][2017/08/28 17:47:03.05932177][No tasks to launch.
[1] [INFO] [53635] unknown sched[0]:controller.go:104][2017/08/28 17:47:03.06108196][Suppressing offers
[1] [INFO] [53635] unknown sched[0]:controller.go:104][2017/08/28 17:47:03.06266645][Declining 1 offers

$ cat task.json

```json
{
  "name": "Mesos demo",
  "resources": { 'cores': 0.1, 'mem': 32.0, 'disk': { "size": 50.0 } },
  "command": "true; do /usr/local/bin/ncat -l -p 2000 -c /usr/bin/printf \"HTTP/1.1 200 \n\n\nIn framework task running!\"; done",
  "labels": { "type": "server" }
}
```

Verizon Proprietary and Confidential Information. Any third party use or disclosure is prohibited without Verizon’s express approval. All rights reserved.
## Container network interface: Overview

Standardized way of creating networks for containers.

CNI allows us to create a network at runtime for a container on any host.

<table>
<thead>
<tr>
<th>Supported Types of Plugins</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network</td>
</tr>
<tr>
<td>bridge</td>
</tr>
<tr>
<td>ipvlan</td>
</tr>
<tr>
<td>loopback</td>
</tr>
<tr>
<td>macvlan</td>
</tr>
<tr>
<td>ptp</td>
</tr>
<tr>
<td>vlan</td>
</tr>
</tbody>
</table>
Container network interface: Mesos

How does it interface with Mesos?

CNI configurations are placed on each node; the default is /etc/cni/net.d/<config>.conf

Configuration describes the CNI version, name, network type and other options according to the network type

The NetworkInfo protobuf attached to the task has the name set to CNI network to which it wishes to attach

Once the task is launched onto a node in the cluster, Mesos sends the information to CNI

CNI looks at its configuration and sees if network has a configuration

Network interface(s) are then created in the namespace for the container
Container network interface: Benefits

Automates network configuration and management for containers

Standardized with no vendor lock-in

Can support multiple networks and plug-ins per container

Isolation from other services for multi-tenant environments

End user visibility; looks like a private L2/L3 network

Implementation is decoupled from the interface to allow flexibility, i.e., change to an overlay mechanism like VXLAN or NVGRE without the end user noticing

IPv4 IPAM address management
Container network interface: Example
Container network interface: Example

```
core@node-50-11 ~ $ export PID=9783
core@node-50-11 ~ $ sudo nsenter --target $PID --mount --uts --ipc --net --pid
Update Strategy: No Reboots
Failed Units: 1
  update-engine-stub.service
f6cc7a81-4ba0-41ca-8fa8-ca908ae3d4e8 / # ip addr sho
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
   link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
   inet 127.0.0.1/8 scope host lo
     valid_lft forever preferred_lft forever
   inet6 ::1/128 scope host
     valid_lft forever preferred_lft forever
3: eth1@if22: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
   link/ether 0a:58:0b:32:0b:02 brd ff:ff:ff:ff:ff:ff
   inet 11.50.11.2/24 scope global eth1
     valid_lft forever preferred_lft forever
     inet6 fe80::3c4b:9aff:fe3c:2be/64 scope link
     valid_lft forever preferred_lft forever
5: eth0@if23: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
   link/ether 0a:58:0a:32:0b:00 brd ff:ff:ff:ff:ff:ff
   inet 10.50.11.12/24 scope global eth0
     valid_lft forever preferred_lft forever
     inet6 fe80::fc67:7ff:feee:ada9/64 scope link
     valid_lft forever preferred_lft forever
f6cc7a81-4ba0-41ca-8fa8-ca908ae3d4e8 / # ip routes sho
default via 11.50.11.1 dev eth1
10.50.11.0/24 dev eth0 proto kernel scope link src 10.50.11.12
11.50.11.0/24 dev eth1 proto kernel scope link src 11.50.11.2
f6cc7a81-4ba0-41ca-8fa8-ca908ae3d4e8 / # logout
core@node-50-11 ~ $
```
Container network interface: Improvements

What’s lacking?
IPv6 support
Support for dynamic traffic policy filtering
Support for dynamic updates to existing network configurations
Questions?
Thank you.