LINKERD

Scala the Cloud Native Way: Lessons Learned from Two Years of Linkerd in Production

Dennis Adjei-Baah

A little about me

🇬🇭 🇧🇼

Software Engineer @ **Buoyant**

Linkerd contributor

Twitter: **@dadjeib**

Github: **@dadjeibaah**

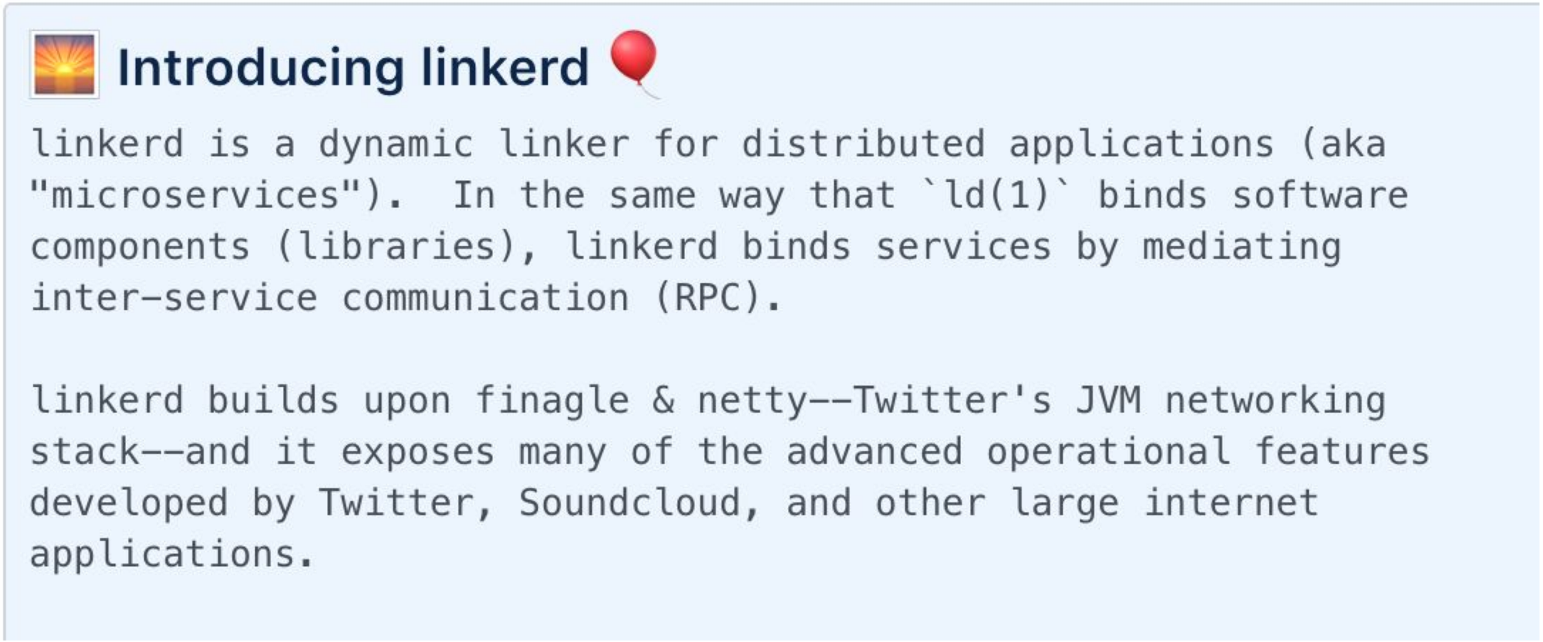
Linkerd

- Service mesh
  - **HTTP**, **HTTP/2**, gRPC request routing
  - reliable, secure, visible
- Built on Finagle/Netty
- Runs as an app with your microservices.

Why you would use it?

- Service discovery integration
- Load balancing
  - Request level load balancing
- Retries
- Advanced routing
  - Canary deployments
  - Blue/Green deployments
- TLS

Where it all started: 2015



🌅 **Introducing linkerd** 🎈

```
linkerd is a dynamic linker for distributed applications (aka
"microservices").  In the same way that `ld(1)` binds software
components (libraries), linkerd binds services by mediating
inter-service communication (RPC).

linkerd builds upon finagle & netty--Twitter's JVM networking
stack--and it exposes many of the advanced operational features
developed by Twitter, Soundcloud, and other large internet
applications.
```

LINKERD

CLOUD NATIVE COMPUTING FOUNDATION

24+ months in production

2k+ Slack channel members

7,000+ GitHub stars

20m+ DockerHub pulls

80+ contributors

400b+ production requests/mo

salesforce PayPal OfferUp

COMCAST monzo Expedia

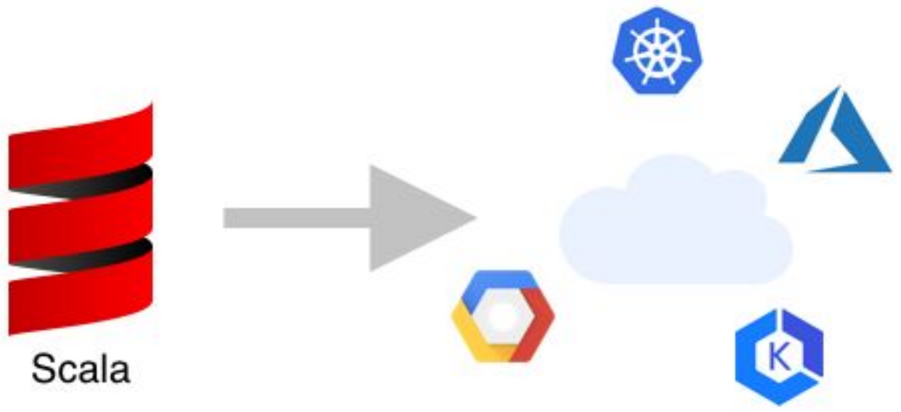
hush. Studyo credit karma

planet. CHASE

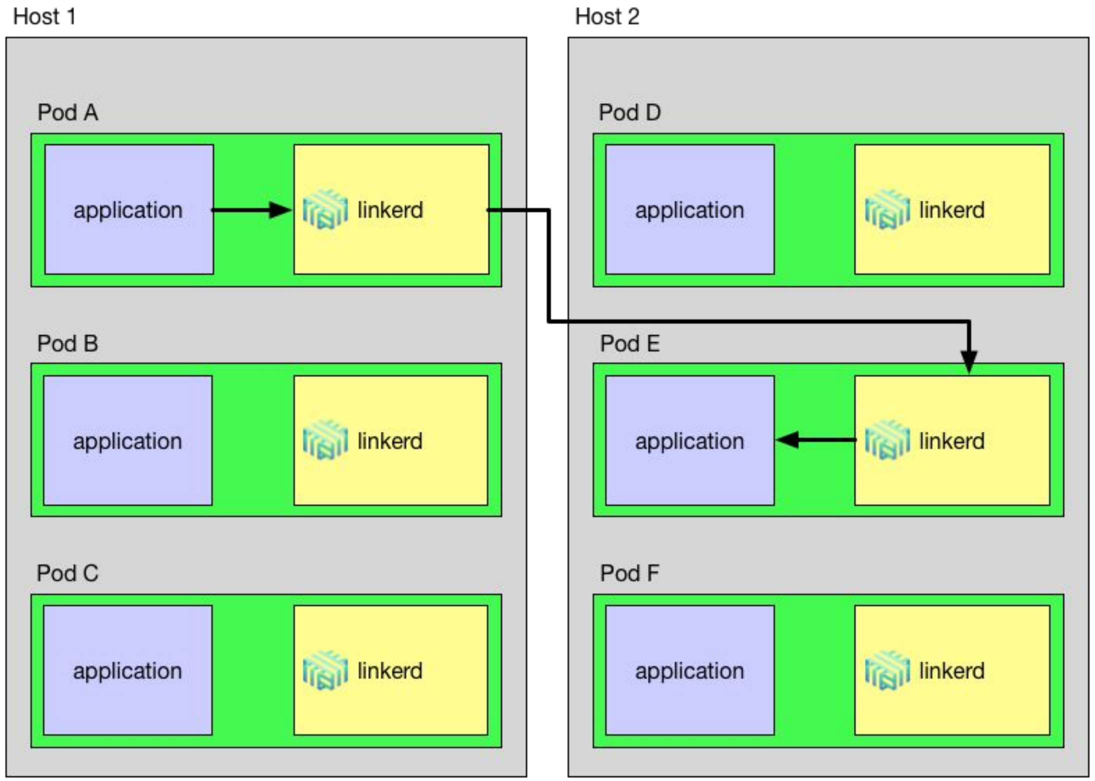
Challenges over those three years

1. Memory usage
2. Memory usage
3. Memory usage

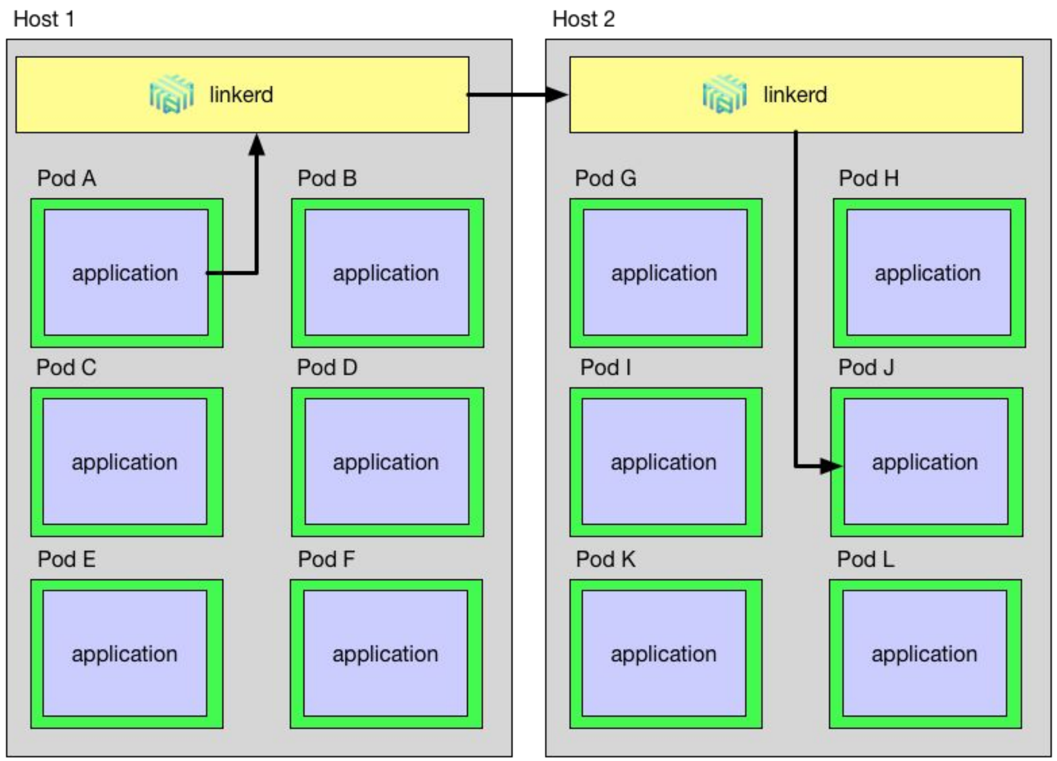
The world is going cloud native



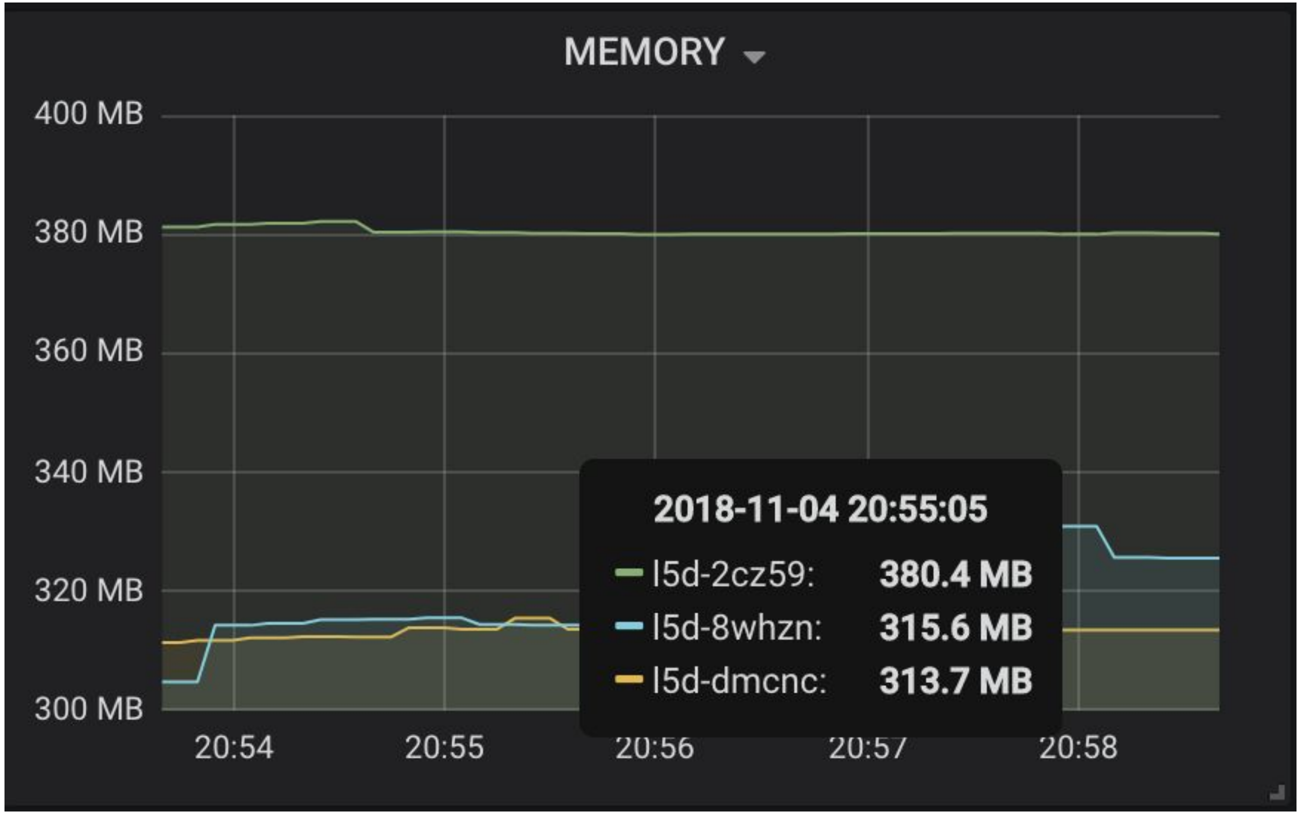
Linkerd runs as a distributed proxy



Alternative deployment: Per Host



Linkerd in production today



Optimizing Linkerd in production

Ways to reduce JVM memory footprint

- JVM tuning
- Use OpenJ9
- Use GraalVM
- Don’t use the JVM

Tuning JVM, Finagle, Netty

- JVM flags
  - Concurrent mark sweep collector, tiered compilation many more
- Finagle
  - Number of threads used to handle concurrency
  - Amount of memory used per thread to tame memory footprint
- Netty
  - Has similar settings help control memory used for

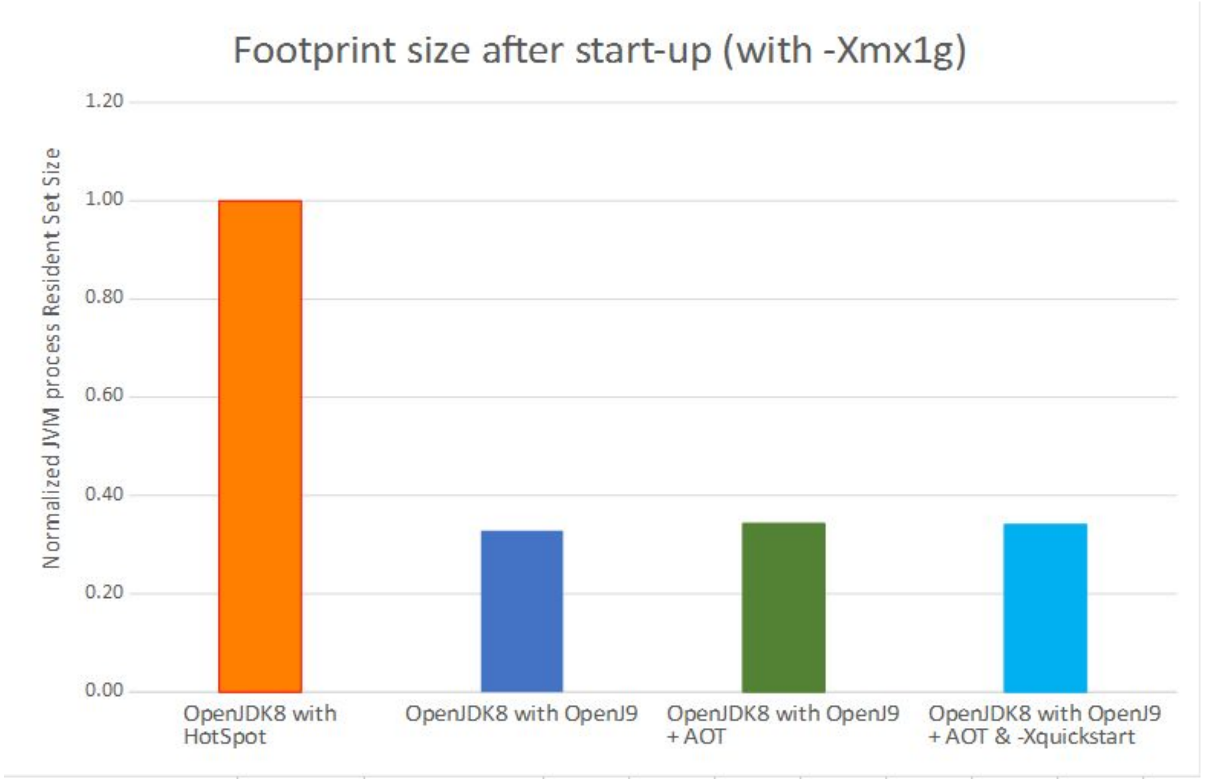
Finding the right tuning parameters for all environments is hard!

OpenJ9

- Built by IBM Java development team
- Tested for decades
- Designed for the cloud

https://pbs.twimg.com/profile_images/910853900181364736/bmWkKoLn_400x400.jpg

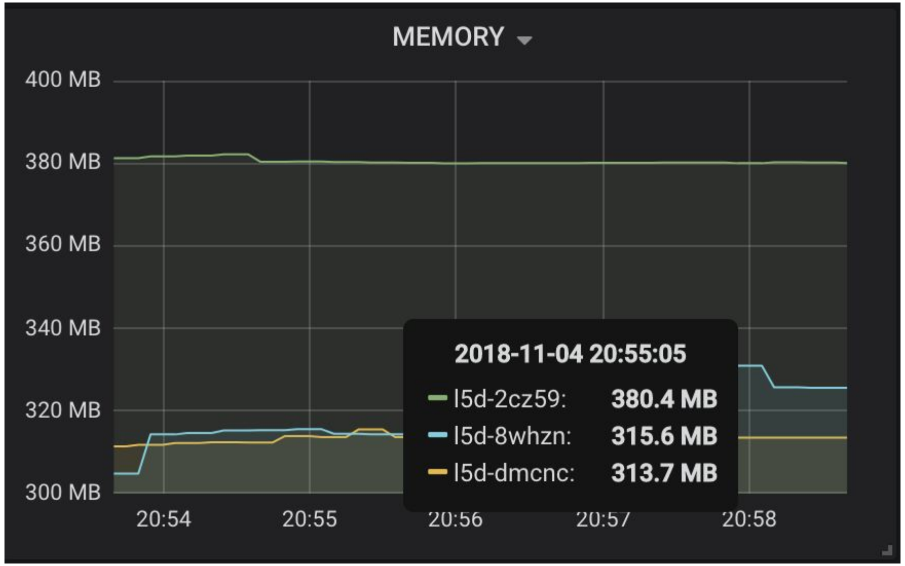
OpenJ9 benchmarks



https://www.eclipse.org/community/eclipse_newsletter/2018/april/images/perf_v8_startup_footprint.png

Linkerd + OpenJ9

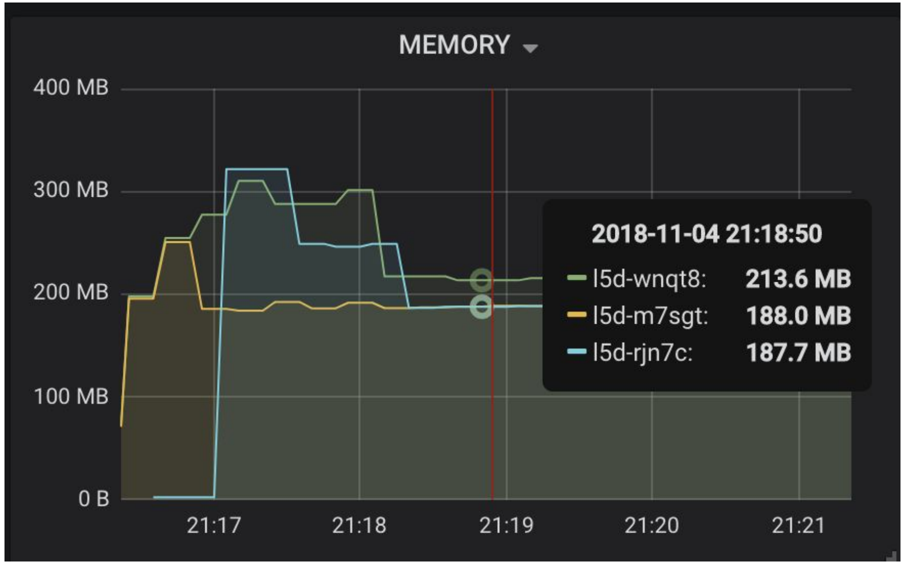
Before



2018-11-04 20:55:05

l5d-2cz59:	380.4 MB
l5d-8whzn:	315.6 MB
l5d-dmcnc:	313.7 MB

After



2018-11-04 21:18:50

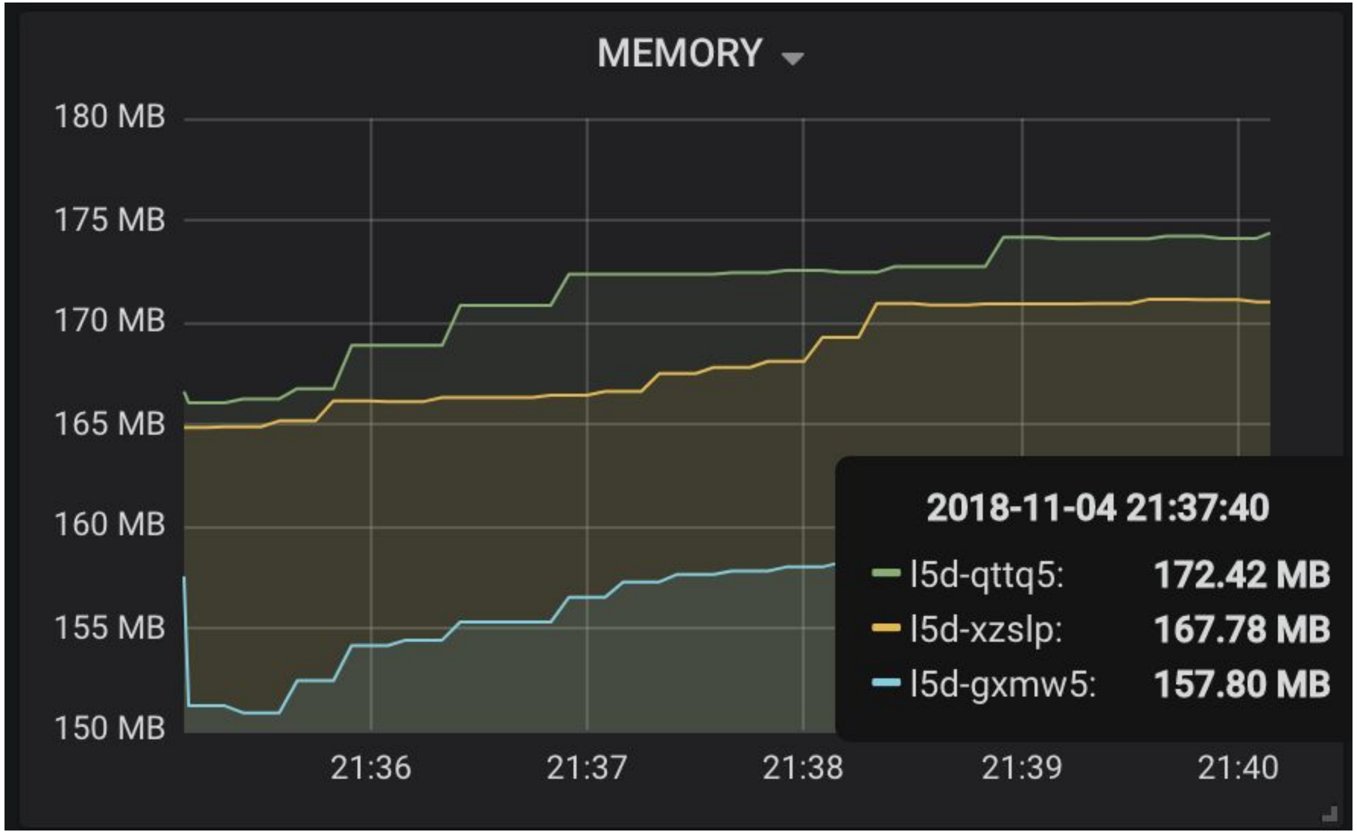
l5d-wnqt8:	213.6 MB
l5d-m7sgt:	188.0 MB
l5d-rjn7c:	187.7 MB

Other OpenJ9 optimizations

- -Xshareclasses
  - Store class data to disk to increase subsequent start up time
- -XX:+IdleTuningGcOnIdle
  - Trigger GC when VM perceives application as idle
- -Xtune:virtualized
  - Signals the VM that it running in a constrained cloud environment which makes it use threads more efficiently
- -Xgcpolicy
  - Configure GC behavior that optimizes garbage collection for a specific type of application.

Learn more here

Linkerd + OpenJ9 tuned



2018-11-04 21:37:40

l5d-qttq5:	172.42 MB
l5d-xzslp:	167.78 MB
l5d-gxmw5:	157.80 MB

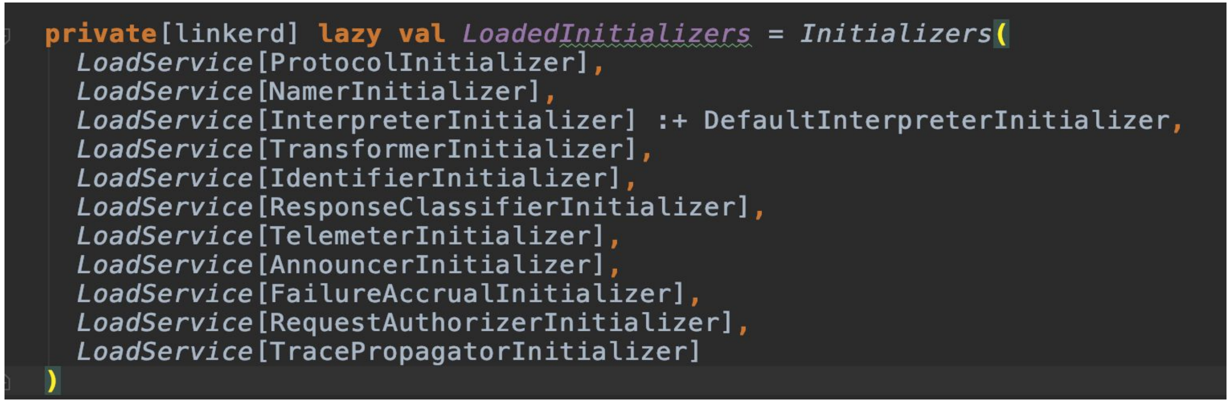
GraalVM

- One VM for multiple programming languages (R, Python, Java and more)
- Ahead of time compilation
- Build native images from Scala

https://www.graalvm.org/resources/img/graalvm.png

Work ongoing at Oracle

- Finagle and Netty on GraalVM
  - With some code modifications
- Making progress on getting Linkerd to work.
- Linkerd’s plugin system



```
private[linkerd] lazy val LoadedInitializers = Initializers(
  LoadService[ProtocolInitializer],
  LoadService[NamerInitializer],
  LoadService[InterpreterInitializer] :+ DefaultInterpreterInitializer,
  LoadService[TransformerInitializer],
  LoadService[IdentifierInitializer],
  LoadService[ResponseClassifierInitializer],
  LoadService[TelemeterInitializer],
  LoadService[AnnouncerInitializer],
  LoadService[FailureAccrualInitializer],
  LoadService[RequestAuthorizerInitializer],
  LoadService[TracePropagatorInitializer]
)
```

Contribute to Linkerd+GraalVM working group

- Working group mailing list
    - https://lists.cncf.io/g/cncf-linkerd-graal-wg
- Linkerd GraalVM slack channel
    - https://linkerd.slack.com/messages/CB3S9LMV5

Don’t use the JVM

- Only if you have really strict memory and throughput goals!
- JVM is more than capable in supporting high throughput
- Linkerd must be transparent
  - Require low system requests
  - Low latency added to proxied request

LINKERD 2.0

Current state of Linkerd

- Officially supported: OpenJDK with tuning
- Experimental: OpenJ9
- Future: GraalVM
- Minimalist rewrite: Linkerd 2.x
  - Off the JVM
  - Very few features compared to Linkerd 1.x (For now)

Recap

- Cloud native poses some challenges when working with the JVM
- JVM has a rich ecosystem e.g. we can build on top of Finagle and Netty
- Moving off the JVM is expensive
- End of the day all implementation details
- What problems we solve with Linkerd

Stay connected

Upcoming events:

- December 10: Linkerd 101 hands on class at KubeCon + CNCFCon Seattle
- December 11-13: lots of Linkerd talks at KubeCon!

Community:

- Join the linkerd-users mailing list
- Join the Linkerd Slack (2300+ users)
- **Twitter: @dadjeb**
- **Github: @dadjeibaah (Slides will be available there)**