BUOYANT

Creators of LINKERD

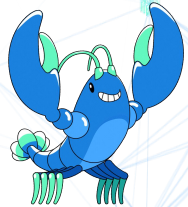


Top Five Things: Linkerd 2.16

Metrics, Gateway API retries & timeouts, IPv6, policy, and more!

Flynn, Technical Evangelist for Linkerd

flynn@buoyant.io | @flynn on slack.linkerd.io



@BuoyantIO

buoyant.io

O'REILLY®

BUOYANT
Creators of LINKERD

Linkerd: Up & Running

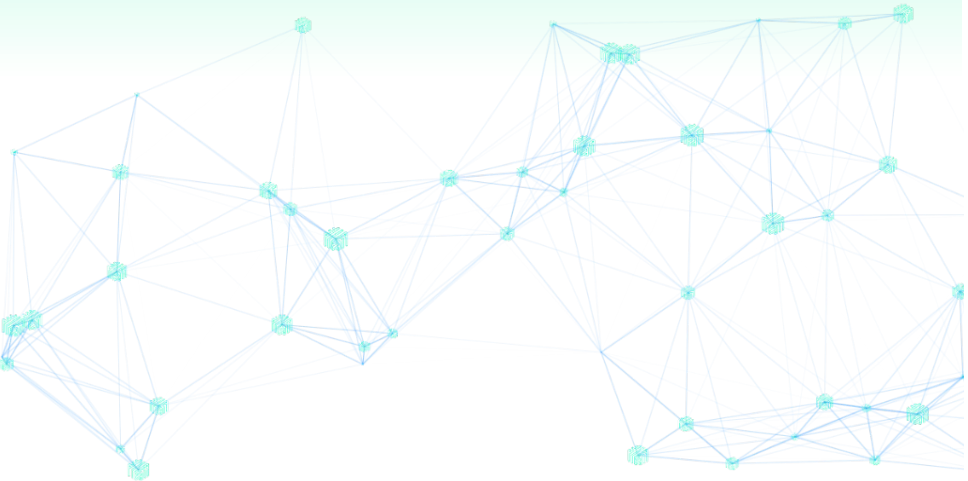
A Guide to Operationalizing a Kubernetes-Native Service Mesh



Download for free!

buoyant.io/**download/linkerd-up-and-running**

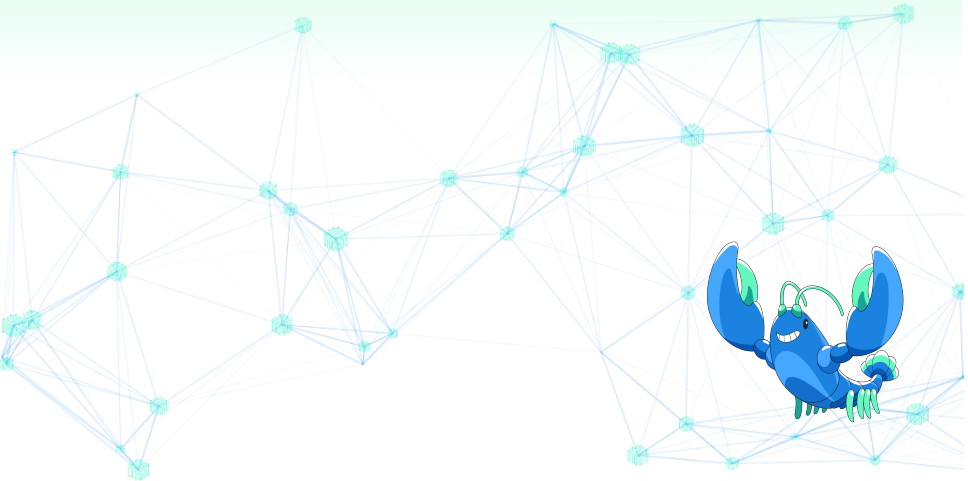
What's on the agenda?



- **Quick** discussion of the features, and then the
- **Demo!**

BUOYANT

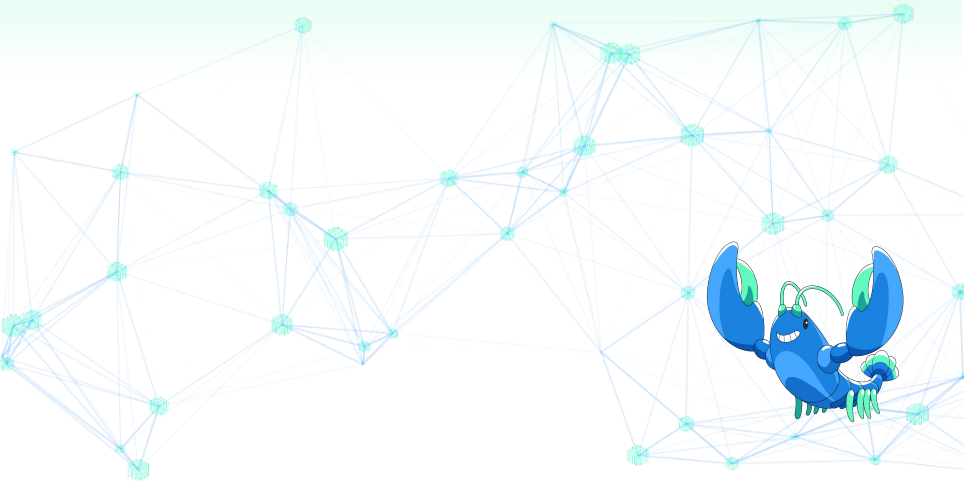
How do you follow along?



- https://github.com/BuoyantIO/service-mesh-academy/tree/main/2-16-features
- **For this demo, I'll be using `enterprise-2.16.0`** so I can show external workload automation
- You'll need a free Buoyant ID from https://enterprise.buoyant.io/
- Most things will also work with `edge-24.8.2` or later

BUOYANT

How do you follow along?



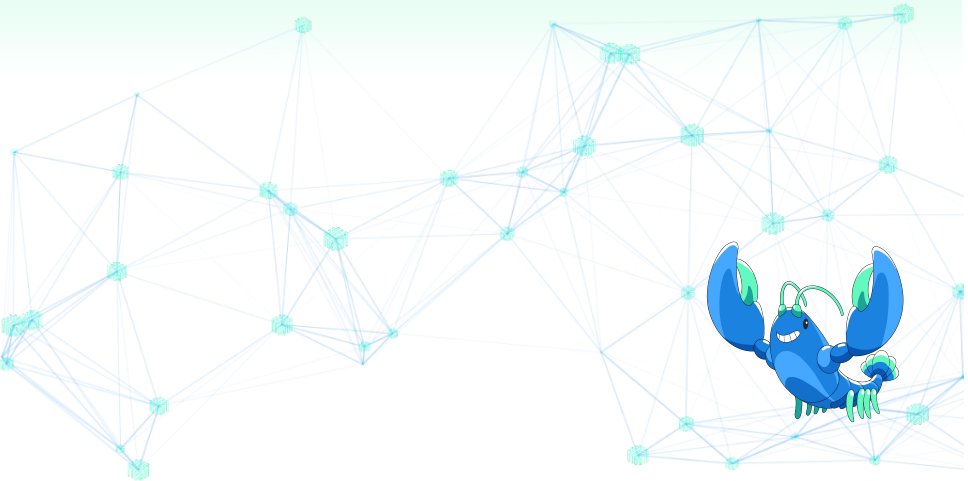
- This demo requires both kind and k3d:

  `https://kind.sigs.k8s.io/`
  `https://k3d.io/`

- Specifically, we'll be using kind with IPv6 and k3d with IPv4.

BUOYANT

How do you follow along?



- **kubectl**
  `https://kubernetes.io/docs/tasks/tools/`
- **kind**
  `https://kind.sigs.k8s.io/`
- **k3d**
  `https://k3d.io/`
- **step**
  `https://smallstep.com/docs/step-cli/`
- **bat**
  `https://github.com/sharkdp/bat`
- **helm**
  `https://helm.sh/docs/intro/quickstart`
- and Python.

BUOYANT

BUOYANT
SERVICEMESH
ACADEMY

Top Five Things You Need to Know about Linkerd 2.16

Top Five Things To Know About Linkerd 2.16!

1. GRPCRoute support!
2. Metrics, timeouts, and retries for HTTPRoute and GRPCRoute!
3. IPv6!
4. Policy audit mode!
5. External workload automation!
6. and yeah some bugfixes and such…

BUOYANT
Creators of LINKERD

BUOYANT
SERVICE MESH
ACADEMY

1. GRPCRoute and 2. Gateway API metrics, retries, and timeouts

BUOYANT
SERVICE MESH
ACADEMY

1 & 2. Gateway API



A Touch of History

- Gateway API (https://gateway-api.sigs.k8s.io/) is a project within Kubernetes' SIG-Networking
  - Started in 2020 primarily focused on north-south traffic and the Wild West of annotations on the Ingress resource
  - Extended in 2022 to sort out how to use Gateway API for meshes (the GAMMA initiative)
- Linkerd started to use the Gateway API in 2.12
  - Goal is Gateway API for talking about "classes of HTTP traffic" (including gRPC)
  - traffic shaping, retries, timeouts, auth policy, dynamic request routing, etc.

A Touch of History

- Linkerd first started using Gateway API's HTTPRoute resource
  - supported HTTP routing, traffic splitting, and authorization
  - did **not** support metrics, timeouts, retries, etc.
  - unpleasant (though possible) to use HTTPRoute for gRPC
- In general, from 2.12 to 2.15, HTTPRoute and ServiceProfile had non-overlapping functionality
  - you couldn't compose them
  - there were several things you could do with ServiceProfile that you couldn't do with HTTPRoute
- 2.16 starts fixing that!

Gateway API in 2.16

- **GRPCRoute**
  - 2.16 adds support for Gateway API GRPCRoute
  - Functionally like HTTPRoute, but better for gRPC

```
---
apiVersion: gateway.networking.k8s.io/v1beta1
kind: HTTPRoute
metadata:
  name: foo-listall-route
  namespace: foo
spec:
  parentRefs:
  - group: core
    kind: Service
    name: foo-svc
    port: 80
  rules:
  - matches:
      - path:
          type: PathPrefix
          value: "/listall"
```

BUOYANT
Creators of LINKERD

Gateway API in 2.16

- **GRPCRoute**
  - 2.16 adds support for Gateway API GRPCRoute
  - Functionally like HTTPRoute, but better for gRPC

```
---
apiVersion: gateway.networking.k8s.io/v1beta1
kind: HTTPRoute
metadata:
  name: foo-listall-route
  namespace: foo
spec:
  parentRefs:
  - group: core
    kind: Service
    name: foo-svc
    port: 80
  rules:
  - matches:
      - method: POST
        path:
          type: Exact
          value: /foo.v1.FooService/ListAll
```

BUOYANT
Creators of LINKERD

Gateway API in 2.16

- **GRPCRoute**
  - 2.16 adds support for Gateway API GRPCRoute
  - Functionally like HTTPRoute, but better for gRPC

```yaml
---
apiVersion: gateway.networking.k8s.io/v1beta1
kind: GRPCRoute
metadata:
  name: foo-listall-route
  namespace: foo
spec:
  parentRefs:
  - group: core
    kind: Service
    name: foo-svc
    port: 80
  rules:
  - matches:
      - method:
          type: Exact
          service: foo.v1.FooService
          method: ListAll
```

Gateway API in 2.16

- **Metrics**
  - 2.16 adds per-route metrics to both HTTPRoute and GRPCRoute!
- **Timeouts and retries**
  - 2.16 adds timeouts and retries to both HTTPRoute and GRPCRoute!
  - We do this with annotations since this isn't yet standardized in Gateway API; when it is, we can revisit things.
  - Fun fact: Linkerd is now the only major mesh to support both counted retries and budgeted retries!

Gateway API in 2.16

- Timeouts and retries can be applied at multiple levels, too
  - HTTPRoute
  - GRPCRoute
  - Service
- Wait, Service?
  - Yes: applying them directly to the Service means that they apply to *all* traffic through that Service, no matter what route got the traffic there.

BUOYANT
SERVICE MESH
ACADEMY

3. IPv6

https://buoyant.io/service-mesh-academy/ipv6-with-linkerd-future-proofing-your-network

A Touch of History

- "The Internet" is, ultimately, rooted in the Internet Protocol (*IP*)
- IP version 4 (IPv4) was standardized by RFC791 (**September 1981**)
  - 32-bit addresses, dotted-decimal notation, this is all familiar stuff
    - 172.31.4.64
    - 172.31.0.0/16
- IPv4 has the *minor problem* that, uh, we're out of addresses.
  - APAC ran out in 2011; by 2019 *everyone* was out
- We mostly dealt with this with RFC1918, which carves out some addresses to be not globally routable and assumes that NAT will make everything work.
  - Fun fact: Kubernetes pod & service CIDRs tend to be RFC1918 addresses.

BUOYANT
Creators of LINKERD

IPv6

- IPv6 (RFC8200, **July 2017**) tackles this address-space problem.
  - 128-bit addresses; CIDR is required when talking about ranges
  - Groups of four hex digits separated by colons
    - `fc00:f853:0ccd:e793:0000:0000:0000:0004`
    - `2607:f8b0:4006:0821:0000:0000:0000:200e`
  - You get to remove leading zeros in groups
    - `fc00:f853:ccd:e793:0:0:0:4`
    - `2607:f8b0:4006:821:0:0:0:200e`
  - You get to replace a single run of zeroes with "::"
    - `fc00:f853:ccd:e793::4`
    - `2607:f8b0:4006:821::200e`
- There's also a new `AAAA` DNS resource for IPv6 addresses.

BUOYANT
Creators of LINKERD

IPv6

- URLs change too: you enclose the address in square brackets for IPv6
  - So `http://172.31.4.64/`
  - But `http://[fc00:f853:ccd:e793::4]`
  - And `http://[fc00:f853:ccd:e793::4]:8000/`

BUOYANT
Creators of LINKERD

IPv6

- Three kinds of hosts in the world now:
  - IPv4-only
  - IPv6-only
  - Dualstack: both IPv4 and IPv6 supported
    - In the dualstack world, **IPv6 is preferred**
- Modern application languages and OSes Just Work™ in this world, as does Kubernetes as of 1.23, as does Linkerd as of 2.16!
  - It's honestly hard to demo in Linkerd because it's almost completely invisible.
  - Just remember to include both IPv4 and IPv6 CIDRs if you're doing NetworkAuthentication on dualstack nodes!

BUOYANT
SERVICE MESH
ACADEMY

4. Policy Audit Mode

Policy Audit Mode

- Prior to Linkerd 2.16, the way you tested authorization policy was by applying it.
  - guaranteed to accurately show you what your policy does!
  - also guaranteed to infuriate everyone when you break your application while trying to figure out how policy works!
- This is especially rough because policy is *the* most complex part of Linkerd from the user's point of view
  - *Everyone* gets policy wrong when they start changing things
  - …usually multiple times.

Policy Audit Mode

- In Linkerd 2.16, everywhere you can use a policy like `all-authenticated` or `deny`, you can now use `audit`
  - e.g. `config.linkerd.io/default-inbound-policy: audit`
  - This works cluster-wide or per-namespace, and we've added the new `accessPolicy` field to support it in Servers.
  - A Server with no `accessPolicy` defaults to `accessPolicy: deny`, for backward compatibility

```
---
apiVersion: policy.linkerd.io/v1beta3
kind: Server
metadata:
  name: foo-server
  namespace: foo
spec:
  accessPolicy: audit
  podSelector:
    matchLabels:
      app: web-svc
  port: http
```

Policy Audit Mode

- `audit` is like `deny`, except that instead of denying the connection, it will log it instead
  - In the `linkerd-proxy` logs, you'll see INFO level logs that include `authz.name=audit`
  - You'll also see `inbound_http_authz_allow_total` or `inbound_grpc_authz_allow_total` metrics that include the label `authz_name="audit"`
- As of 2.16, the `linkerd policy generate` command now defaults to `accessPolicy: audit` in its generated Servers
  - You can use `--disable-audit` if you really want to generate Servers with `deny` rather than `audit`

BUOYANT
Creators of LINKERD

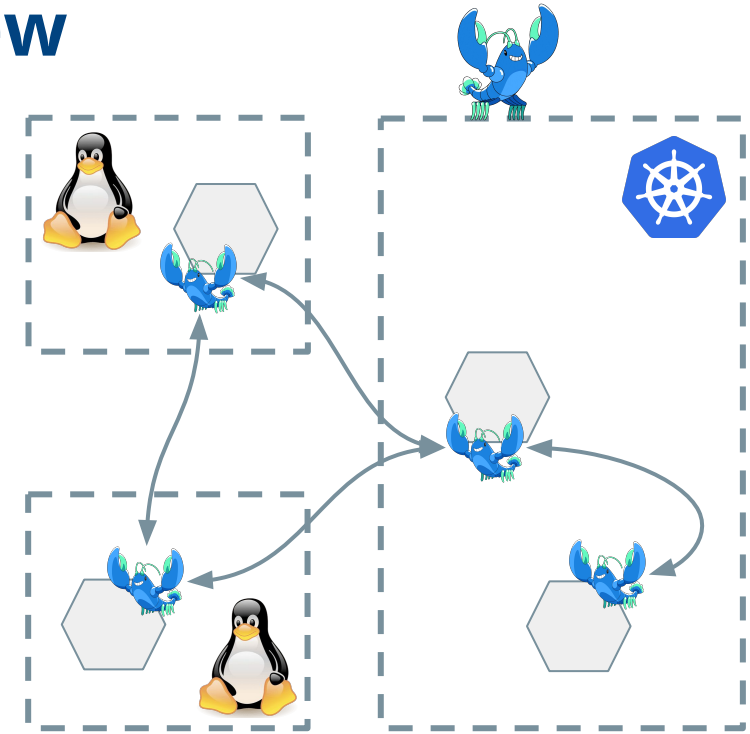
BUOYANT
SERVICE MESH
ACADEMY

5. External workload automation

https://buoyant.io/service-mesh-academy/linkerd-outside-of-kubernetes-the-easy-way

External Workload Overview

- Bring the mesh outside the cluster, all the way to VMs or edge devices
- Benefits
  - Access to legacy workloads
  - Access to smaller devices (the Linkerd proxy is very lightweight)
- Implementation
  - Run Linkerd proxy with custom iptables rules
- Challenges
  - Root of trust in shared system
  - Network / DNS
  - Currently assumes Linux



BUOYANT
Creators of LINKERD

External Workloads in Linkerd

- “Run the Linkerd proxy” is doing an awful lot of work in that sentence above; under the hood, this is a very complex process
  - Use SPIFFE for identity management
  - Set up the IP routing table
  - Connect back to the control plane to get policy
  - Set up an ExternalWorkload resource for each external workload instance
  - Set up Services linked to ExternalWorkload resources

External Workloads in Linkerd

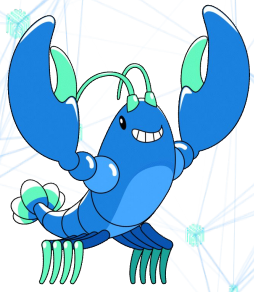
- BEL makes this easier with the *autoregistration controller* for managing external workloads
  - Instead of managing individual ExternalWorkload resources, you manually create a ExternalGroup resource instead
  - Install a Linux package on the edge device that knows how to handle the routing table and talk to the control plane
  - The autoregistration controller fields connections from external workloads, figures out their ExternalGroup, and manages the ExternalWorkloads for you

External Workloads in Linkerd

- The end result is that ExternalGroup and ExternalWorkload are parallel to Deployment and Pod
  - Create the ExternalGroup, let the controller manage the lower-level stuff for you
  - You can still manage ExternalWorkloads directly if you want to, of course, but it's really much easier not to

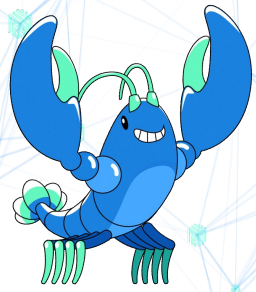
BUOYANT
SERVICE MESH
ACADEMY

DEMO 🤞



BUOYANT
SERVICE MESH
ACADEMY

Gotchas



Gotchas

- The big one: Linkerd Viz doesn't support Gateway API metrics
  - There's some really cool stuff coming here. Promise.
  - For now... it's pretty manual.
  - The "Metrics and Dashboards and Charts, Oh My!" SMA gets much further down the rabbit hole of how to work with metrics
- Related but smaller: `linkerd diagnostics proxy-metrics` requires a *Pod*; an ExternalWorkload won't cut it.
  - This is one of the few areas where you have to think about where your workload lives.
- Running IPv6 Kubernetes is still a lot more annoying than I'd like it to be.

Buoyant Enterprise for LINKERD

Rust-based network security and reliability for modern applications. Built on open source and designed for the enterprise.

- ➔ Zero-trust security and compliance across your entire network
- ➔ Global traffic management and control
- ➔ Full L7 application observability
- ➔ Built for the enterprise

Learn more & get a free trial at buoyant.io/**enterprise-linkerd**

BUOYANT
Creators of LINKERD



Updated Course!

BUOYANT
Creators of LINKERD

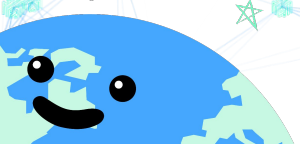


Get Certified!
Service Mesh 101, Online Course

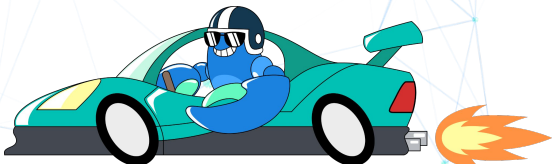
With hands-on LINKERD labs at learn.buoyant.io









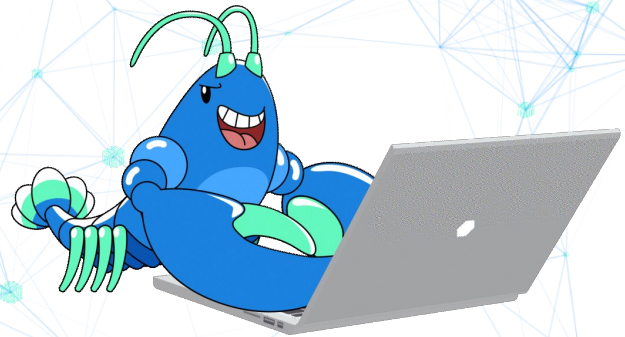


SERVICE MESH ACADEMY

Up Next on October 17

Linkerd and Backstage

SIGN UP TODAY!
buoyant.io/sma



BUOYANT
SERVICEMESH
ACADEMY

Q&A



BUOYANT
SERVICEMESH
ACADEMY

Thanks much!



flynn@buoyant.io
@flynn on slack.linkerd.io