What Happens when Service Mesh Maintainers Get a Taste of Their Own Mesh?

Risha Mars

Linkerd Maintainer, Software Engineer @ Buoyant

@marzipan

@rmars

Carol Scott

Linkerd Maintainer, Software Engineer @ Buoyant

@CarolScottSF

@scottcarol

What you'll hear today

- Working on Linkerd
- Becoming a Linkerd prod user while building our SaaS product (Dive)
- How we went from 😅 to 😍
- The ~~mistakes we made~~ adventures we had along the way
- What you can apply from our experience

Working on an awesome OSS project

- For most of its history, Buoyant has focused purely on building Linkerd.
- We spot potential issues through integration tests and edge releases
- Bug reports come in from Linkerd users
- But we've never been on-call for a Linkerd installation ourselves

Working on an awesome SaaS product

- Everything changed when we started building a SaaS product, Dive!
- We got to experience the joys of Linkerd… and also the bugs
- We got to practice what we preach
- User empathy levels… ENGAGE!

LINKERD

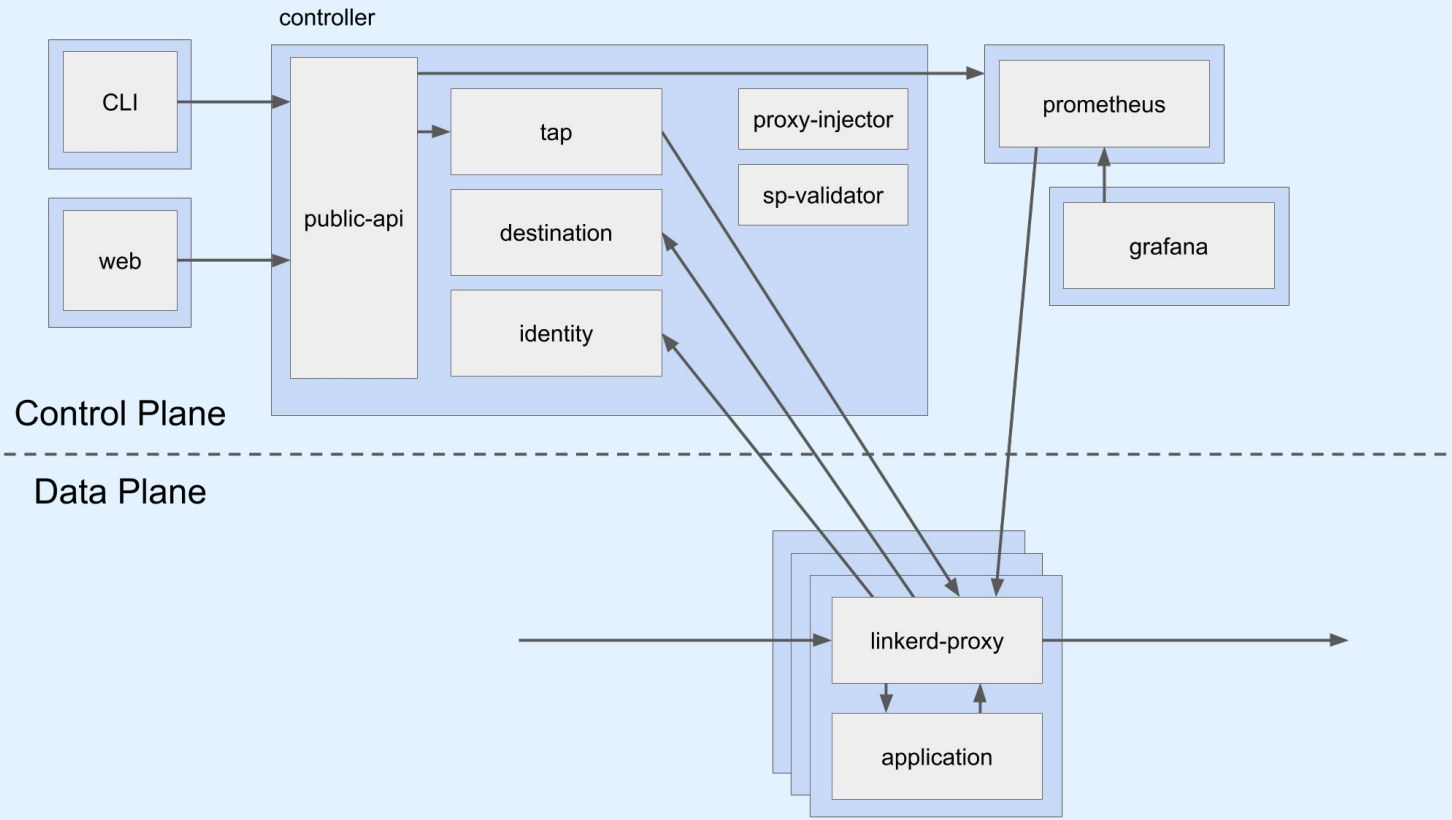
An open source *service mesh* and CNCF member project.

- 🔥 36+ months in production
- 🔥 3,500+ Slack channel members
- 🔥 10,000+ GitHub stars
- 🔥 100+ contributors

CLOUD NATIVE COMPUTING FOUNDATION

Ask
CHASE
GoDaddy
KAIROS
STRAVA
BIGCOMMERCE
Expedia
OfferUp
ebay
Cisco webex
COMCAST
planet.
Disney
Walmart

Linkerd


controller
CLI
web
public-api
tap
destination
identity
proxy-injector
sp-validator
prometheus
grafana
Control Plane
Data Plane
linkerd-proxy
application

Dive

DIVE

Buoyant

All workloads

- Dashboard
- Events
- Workloads
- Services
- Applications
- Clusters
- Add Event
- Add Cluster
- Give Feedback
- Docs

Connect Dive to Slack to get links, notifications, and commands!

Add to Slack

SERVICE

payments

A microservice that provides payment services.

Overview | Events | Service map | Preflight check | Promotions | Docs

Overview

The payments service runs on port 8082. Payments serves all web backend endpoints and all web frontend assets. We run 3 replicas in production.

It exposes two POST and two GET routes:

`/purchase` - (POST) Request to process a purchase.
`/reset` - (POST) Request to update resources.
`/test-mail` - (GET) Request to send test email to errors address.
`/ping` - (GET) Notice to check response and to wake up from idle.

Service level objectives

env : production

99.99% of all requests over a rolling 30-day window are served successfully.

Error budget 55%
Compliance 99.99%
SLI (Success Rate) 99.8%

1 of every 1000 requests can be served unsuccessfully

Workloads

Location	Type	Reqs/24h	Version	Last rollout	Tags
prod / payments / prod-payments	deploy	512.4k	2.7.6	2 days ago	3
staging / payments / staging-payments	deploy	12.2k	2.8	1 day ago	3
qa / payments / qa-payments	deploy	8.6k	2.7.6	2 days ago	3

Ongoing and scheduled events

Add event

payments — Promotion event — Deploy Payments service from Staging to QA — Kevin Ingelman • Ongoing • Started a few seconds ago — 3

qa / payments / qa-payments — Code rollout event — Rolled out v2.8 — Kevin Ingelman • Ongoing • Started a few seconds ago — 3

All events for this service

Added
Mon, 02 Jun 2020 15:04:05 MST

Application
None

Clusters
prod
staging

Owners
Carol Scott
Dennis Adjei-Baah
Kevin Ingelman

Tags
env : production
env : staging
env : qa
lang : python
squad : dora

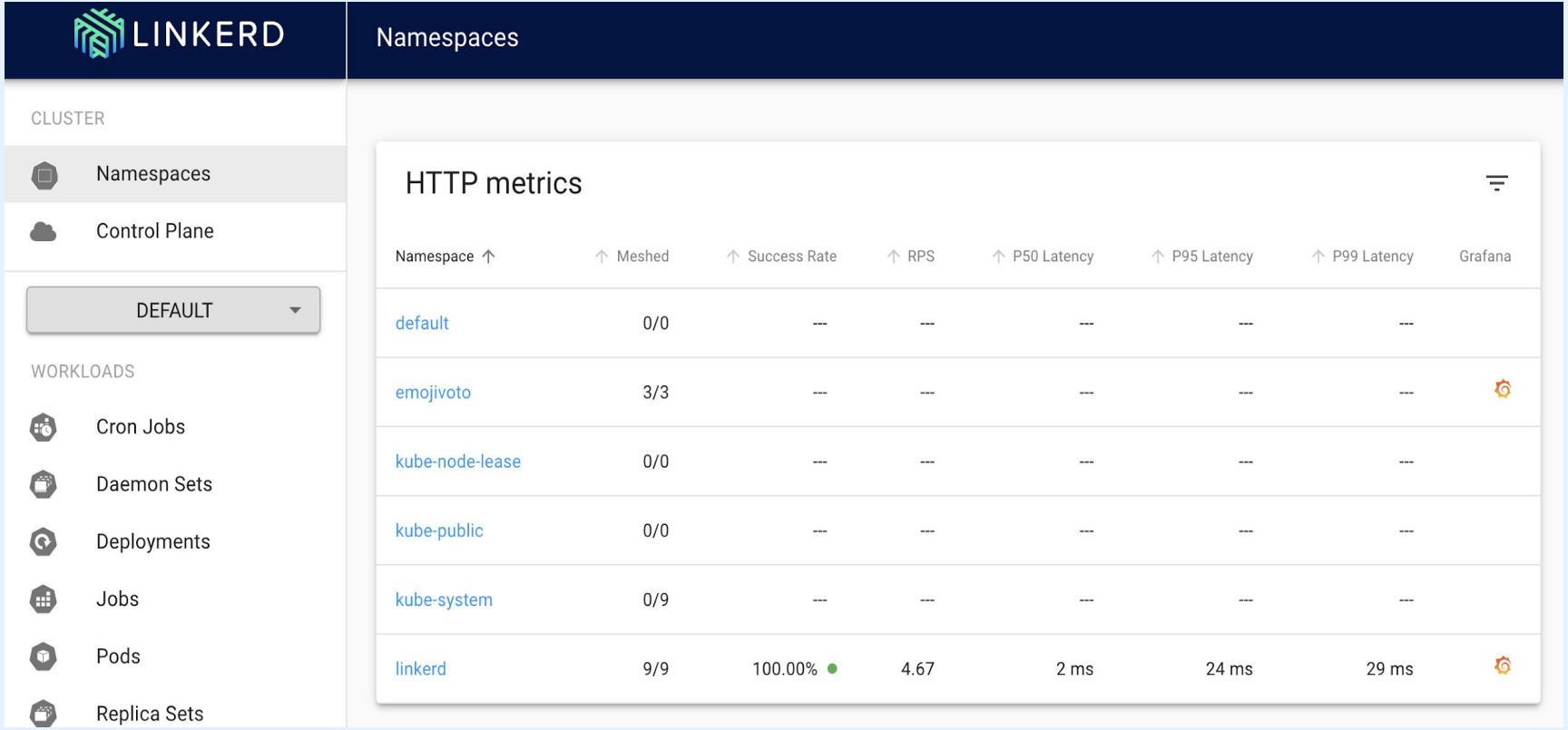
Notifications
Subscribe

Dive + Linkerd = Amazingness

- Linkerd Helps Dive
- Dive Helps Linkerd
- Linkerd and Dive Help Each Other

Linkerd Helps Dive

Why a Service Mesh?



Why a Service Mesh?

```
18:05:21 mars@dev ~/w/bcloud (main) $ curl https://run.linkerd.io/install-edge | sh
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  2971  100  2971    0     0   6573      0 --:--:-- --:--:-- --:--:--  6573

Validating checksum...
Checksum valid.

Linkerd edge-20.7.4 was already downloaded; making it the default 🎉

To force re-downloading, delete '/home/mars/.linkerd2/bin/linkerd-edge-20.7.4' then run me again.

Add the linkerd CLI to your path with:

  export PATH=$PATH:/home/mars/.linkerd2/bin

Now run:

  linkerd check --pre                     # validate that Linkerd can be installed
  linkerd install | kubectl apply -f -    # install the control plane into the 'linkerd' namespace
  linkerd check                           # validate everything worked!
  linkerd dashboard                       # launch the dashboard

Looking for more? Visit https://linkerd.io/2/next-steps
```

Investigate high latency

Our goal: Use linkerd metrics to investigate high latencies in our app

klingerf commented on Jan 28

We've been seeing high latency alerts for the bcloud-api deployment in our prod cluster since late last week. For example:

> Alert: P95 latency above 50ms for 10 minutes: bcloud-api (current value: 58.94999999999995ms)

We should try to figure out what changed, and bump the alert threshold if this is the new normal.

Investigate high latency

The investigation: Linkerd wasn’t giving us the metrics we expected it to, and we worried it wasn’t playing well with NGINX.

dadjeibaah commented on Jan 31

We tried to get more in-depth metrics on the route `/AddMetric` from the outbound nginx proxy. Unfortunately, we don't get route metrics from that proxy. Further investigation into why that is the case points to nginx overriding `l5d-dst-override` headers ONLY on requests to `bcloud-api`. This header is what is used to find service profiles to attach route metrics to. In this case, we can't attach route metrics for the ingress-nginx.

We think that behind that NGINX's GRPC configuration might be the interfering with linkerd-proxies `l5d-dst-override` header. Further investigation needs to be done to determine if that is 100% certain.

👍 1

Investigate high latency

The solution: Configure NGINX with the correct override header!

dadjeibaah commented on Feb 18

Worked with **@grampelberg** to figure out the header issue and it turns out we needed to set the nginx configuration annotation `grpc_set_header` with the `l5d-dst-override` header in bcloud-api's ingress config.

I've made this change in staging and we now get per route metrics for bcloud-api.

```
→  bcloud git:(dad/card-component) ✗ linkerd routes -n bcloud-staging  deploy/bcloud-api
ROUTE                    SERVICE   SUCCESS      RPS   LATENCY_P50   LATENCY_P95   LATENCY_P99
AddEvent             bcloud-api         -        -             -             -             -
AddMetric            bcloud-api   100.00%   0.3rps           5ms          19ms          20ms
DeploymentStream     bcloud-api         -        -             -             -             -
ServiceStream        bcloud-api         -        -             -             -             -
[DEFAULT]            bcloud-api         -        -             -             -             -
[DEFAULT]          bcloud-proxy         -        -             -             -             -
```

Viewing latency per route graphs in grafana's staging instance shows that The `AddMetrics` endpoint is contributing to high latency in bcloud-api.

Investigate high latency

The aftermath: Correct the docs -- oops, nevermind!

wmorgan commented on Feb 18

Very cool. A couple questions:

1. Do the docs in https://linkerd.io/2/tasks/using-ingress/#nginx need to be updated? Or did we just not follow them in our initial nginx setup?

Debugging slow requests

The goal: figure out why some of our graphql requests were slow

```
$ linkerd tap deploy/linkerd-prometheus -nlinkerd
req id=0:0 proxy=out src=10.1.0.83:48314 dst=10.1.0.82:4191 tls=true :method=GET :authority=10.1.0.82:4191 :path=/metrics
rsp id=0:0 proxy=out src=10.1.0.83:48314 dst=10.1.0.82:4191 tls=true :status=200 latency=1684µs
end id=0:0 proxy=out src=10.1.0.83:48314 dst=10.1.0.82:4191 tls=true duration=58µs response-length=3329B
req id=0:1 proxy=out src=10.1.0.83:47774 dst=10.1.0.84:9998 tls=true :method=GET :authority=10.1.0.84:9998 :path=/metrics
rsp id=0:1 proxy=out src=10.1.0.83:47774 dst=10.1.0.84:9998 tls=true :status=200 latency=4963µs
end id=0:1 proxy=out src=10.1.0.83:47774 dst=10.1.0.84:9998 tls=true duration=394µs response-length=2111B
req id=0:2 proxy=out src=10.1.0.83:55548 dst=10.1.0.89:4191 tls=true :method=GET :authority=10.1.0.89:4191 :path=/metrics
rsp id=0:2 proxy=out src=10.1.0.83:55548 dst=10.1.0.89:4191 tls=true :status=200 latency=1910µs
```

Debugging slow requests

The goal: figure out why some of our graphql requests were slow

Andrew Seigner <siggy@buoyant.io> Wed, Apr 8, 3:40 PM

to Alex, dive-dev

Oh! (As Adam just reminded me)... being able to see GraphQL POST request payloads would be _very_ useful in determining which queries are slow.

Debugging slow requests

The solution: use distributed tracing to find slow requests

Alex Leong <alex@buoyant.io>

to dive-dev, William

Tue, Apr 14, 11:50 AM

Hey Divers,

After discussing with the Linkerd team, I just wanted to give you an update on our thinking about each of these items:

* GraphQL Post Payloads

Tap bodies is on the Linkerd roadmap eventually but I highly highly recommend that you just add distributed tracing to Dive instead. Not only will this let you find slow requests and inspect their graphQL queries, it could potentially also allow you to follow the processing of that request and see specifically how long is taken by each query in a multi-query request. Adding OpenCensus to a go project is easy and I'd be happy to help out. Tarun, our resident OpenCensus expert, could probably help out too.

Distributed tracing helps! Sweet!

Jaeger UI | Lookup by Trace ID... | Search | Compare | System Architecture | About Jaeger

bcloud: GraphQL http request 7102b86

Find... | Alternate Views

Trace Start **July 30 2020, 11:31:47.606** | Duration **157.72ms** | Services **1** | Depth **2** | Total Spans **37**

0ms | 39.43ms | 78.86ms | 118.29ms | 157.72ms

Service & Operation | 0ms | 39.43ms | 78.86ms | 118.29ms | 157.72ms

bcloud GraphQL http request

GraphQL http request

Service: **bcloud** | Duration: **157.72ms** | Start Time: **0ms**

Tags: internal.span.format = proto | sampler.param = true | sampler.type = const

Process: client-uuid = 5deb8514a7ed102f | hostname = d0cf1fa92f58 | ip = 172.19.0.17 | jaeger.version = Go-2.23.0

SpanID: 7102b86111c58ed7

Thanks, Linkerd!

- The Linkerd team is super helpful and it’s a pleasure to interact with them (but we knew that already 😻)
- Maintainers know a lot about what their project does, and what it DOESN’T - which are both helpful
- P.S. When you’re installing a mesh, RTFM (Read The Full* Manual)... even if your team wrote the docs 😬

Dive Helps Linkerd

We live life on the edge

- We upgrade our staging cluster weekly to the latest edge release
- As a result, we sometimes discover bugs first!
- Some bugs we found were specific to upgrading from one Linkerd version to another!

`linkerd check` is our friend

```
07:44:22 mars@dev ~/w/bcloud (main) $ linkerd check
kubernetes-api
--------------
√ can initialize the client
√ can query the Kubernetes API

kubernetes-version
------------------
√ is running the minimum Kubernetes API version
√ is running the minimum kubectl version

linkerd-existence
-----------------
√ 'linkerd-config' config map exists
√ heartbeat ServiceAccount exist
√ control plane replica sets are ready
√ no unschedulable pods
√ controller pod is running
√ can initialize the client
√ can query the control plane API

linkerd-config
--------------
√ control plane Namespace exists
√ control plane ClusterRoles exist
√ control plane ClusterRoleBindings exist
√ control plane ServiceAccounts exist
√ control plane CustomResourceDefinitions exist
√ control plane MutatingWebhookConfigurations exist
√ control plane ValidatingWebhookConfigurations exist
√ control plane PodSecurityPolicies exist

linkerd-identity
```

Keeping our linkerd proxies up to date

The goal: Upgrade linkerd as part of our weekly routine

The process:

- Run `linkerd upgrade` to upgrade the control plane
- Upgrade the data plane by restarting each pod to pick up the new proxy
- Use `linkerd check --proxy` to see which proxies were out of date

Upgrading the proxies is tedious

The problem: linkerd check --proxy would only return the first out of date proxy.

```
linkerd-data-plane
------------------
√ data plane namespace exists
√ data plane proxies are ready
√ data plane proxy metrics are present in Prometheus
√ data plane is up-to-date
‼ data plane and cli versions match
    dive-viz/mysqld-exporter-staging-5f8d865b7f-czf4d running stable-2.8.1 but cli running edge-20.4.1
    see https://linkerd.io/checks/#l5d-data-plane-cli-version for hints
```

So the process would be:

- Restart a deploy to pick up the new proxy
- Run linkerd check --proxy to find the next pod to be updated
- Upgrade that proxy
- Repeat

Upgrading the proxies is tedious

The fix:

Improve proxy version diagnostics #4244

Merged **adleong** merged 3 commits into `master` from `alex/check-proxy-list` on Apr 16

Conversation 9 | Commits 3 | Checks 15 | Files changed 3

adleong commented on Apr 10 — Member

It can be difficult to know which versions of the proxy are running in your cluster, especially when you have pods running at multiple different proxy versions.

We add two pieces of CLI functionality to assist with this:

The `linkerd check --proxy` command will now list all data plane pods which are not up-to-date rather than just printing the first one it encounters:

```
!! data plane is up-to-date
    Some data plane pods are not running the current version:
        * default/books-84958fff5-95j75 (git-ca760bdd)
        * default/authors-57c6dc9b47-djldq (git-ca760bdd)
        * default/traffic-85f58ccb66-vxr49 (git-ca760bdd)
        * default/release-name-smi-metrics-899c68958-5ctpz (git-ca760bdd)
        * default/webapp-6975dc796f-2ngh4 (git-ca760bdd)
        * default/webapp-6975dc796f-z4bc4 (git-ca760bdd)
        * emojivoto/voting-54ffc5787d-wj6cp (git-ca760bdd)
        * emojivoto/vote-bot-7b54d6999b-57srw (git-ca760bdd)
        * emojivoto/emoji-5cb99f85d8-5bhvm (git-ca760bdd)
        * emojivoto/web-7988674b8b-zfvvm (git-ca760bdd)
        * default/webapp-6975dc796f-d2fbc (git-ca760bdd)
        * default/curl (git-7f6bbc73)
    see https://linkerd.io/checks/#l5d-data-plane-version for hints
```

The `linkerd version` command now supports a `--proxy` flag which will list all proxy versions running in the cluster and the number of pods running each version:

```
linkerd version --proxy
Client version: dev-7b9d475f-alex
Server version: edge-20.4.1
Proxy versions:
        edge-20.4.1 (10 pods)
        git-ca760bdd (11 pods)
        git-7f6bbc73 (1 pods)
```

New `linkerd check` output

```
linkerd-data-plane
------------------
√ data plane namespace exists
√ data plane proxies are ready
√ data plane proxy metrics are present in Prometheus
!! data plane is up-to-date
    Some data plane pods are not running the current version:
        * linkerd-dive/dive-agent-64bdf79994-cn2xv (edge-20.7.3)
        * emojivoto/web-9475ff858-7hcxj (edge-20.7.3)
        * linkerd/linkerd-prometheus-684b785b74-dchdq (edge-20.7.3)
        * linkerd/linkerd-tap-7c76bdf6c9-ztk72 (edge-20.7.3)
        * emojivoto/vote-bot-76cdf94c9b-brlk7 (edge-20.7.3)
        * linkerd/linkerd-sp-validator-f5b77fdbb-mxmxl (edge-20.7.3)
        * linkerd/linkerd-grafana-84d6d4b9c4-qzq25 (edge-20.7.3)
        * linkerd/linkerd-identity-57f5cb6f5d-z6fgf (edge-20.7.3)
        * linkerd/linkerd-web-d9544f75c-j9h2x (edge-20.7.3)
        * linkerd/linkerd-proxy-injector-85ccdf566f-5zsgl (edge-20.7.3)
        * linkerd/linkerd-destination-fb4654479-htlpn (edge-20.7.3)
        * linkerd/linkerd-controller-7d45bbd564-c6fkw (edge-20.7.3)
        * emojivoto/voting-8c7445b4-tpfvp (edge-20.7.3)
    see https://linkerd.io/checks/#l5d-data-plane-version for hints
√ data plane and cli versions match
```

New `linkerd version` output

```
07:44:21 mars@dev ~/w/bcloud (main) $ linkerd version --proxy
Client version: edge-20.7.4
Server version: edge-20.7.3
Proxy versions:
        edge-20.7.3 (14 pods)
        edge-20.7.4 (9 pods)
```

Can’t upgrade linkerd due to a render error

The goal: Upgrade linkerd from edge 20.5.3 to 20.5.4 as part of our weekly routine

Can't upgrade Linkerd from edge 20.5.3 to 20.5.4 due to render error #4471

Closed scottcarol opened this issue on May 22 · 2 comments

scottcarol commented on May 22 Member

Bug Report

What is the issue?

When upgrading from 20.5.3 to 20.5.4, I am getting the following error after running `linkerd upgrade`:

```
× Could not render upgrade configuration: render error in "grafana/templates/grafana.yaml": template: grafana/te
For troubleshooting help, visit: https://linkerd.io/upgrade/#troubleshooting
```

Assignees

kleimkuhler

Labels

area/cli

Projects

2.9 - backlog

Done

Can’t upgrade linkerd due to a render error

The workaround:

alpeb commented on May 22 Member

Thanks for the report **@scottcarol** 👍
The section for Grafana in `values.yaml` changed and we failed to document that.
To avoid the issue you can run `linkerd upgrade --addon-overwrite` which will reset the addons to their default values (assuming you haven't customized anything related to Grafana and tracing).

Currently we have an integration test for upgrades from the previous stable; I'll open an issue to make a new one to test upgrades from the previous edge so we can catch this sort of problems.

👍 1

Can’t upgrade linkerd due to a render error

The fix:

Add integration test for upgrading from edge #4557

Merged olix0r merged 12 commits into `master` from `kleimkuhler/add-upgrade-edge` on Jun 16

Conversation 14 | Commits 12 | Checks 17 | Files changed 5

kleimkuhler commented on Jun 4 • edited

Problem

#4557 changed the name of the function that `helm_upgrade_integration_tests` uses.

`install_stable()` was renamed to `latest_release_channel()` and now takes an argument for specifying either `edge` or `stable`.

`run_helm_upgrade_test` is a function used by the helm upgrade integration test and was not properly updated to use `latest_release_channel()`.

This silently passed integration tests because `run_helm_upgrade_test` started passing an empty string for the version to upgrade from, which results in the default behavior of `install_test.go` --and therefore still passes.

The case of the hanging linkerd check

The problem:

- As part of our normal workflow, we run linkerd check after upgrading
- After one upgrade, linkerd check --proxy wasn’t completing
- We noted the large number of evicted pods on our cluster
- Filed: `linkerd check --proxy` stalling on evicted pods #4690

The case of the hanging linkerd check

The fix:

Do not treat evicted pods as failed in healthchecks #4732

Merged zaharidichev merged 2 commits into `main` from `zd/do-not-treat-evicted-pods-as-failed` 20 days ago

We ❤️ our edge users (and their bug reports!)

BadCertificate in case of \n at the end of root cert #3880

Closed StupidScience opened this issue on Jan 2 · 25 comments

StupidScience comm

Disabling the Grafana add-on does not have any effect #4515

Open uhthomas opened this issue on May 29 · 1 comment

uhthomas commented on May 29 • edited

Bug Report

Traffic split shows no stats when filtered by namespace #3562

Closed stefanprodan opened this issue on Oct 11, 2019 · 15 comments

stefanprodan commented on Oct 11, 2019 • edited

Bug Report

Linkerd and Dive Help One Another

Dogfooding Dive

The problem: One of our Dive components was always out of SLO

Service level objectives

99% of all requests over a rolling 7-day window are served successfully.

Error budget
-289.7%

Compliance
96.1%

SLI (Success Rate)
100%

553 requests must be served successfully.

klingerf commented on May 5

It seems like 95% would probably work, but if we're going to drop it, I'd prefer to drop it to something that we know will be safe, since, as WM said in the description, seeing an immediately failing SLO as part of onboarding is really bad UX.

Dogfooding Dive

The investigation: one prometheus scrape fails every 2.8 hours

Prometheus scrapes to Go Processes around 99.9% SR #3836

Closed **siggy** opened this issue on Dec 16, 2019 · 11 comments

siggy commented on Dec 16, 2019 • edited Member

Bug Report

What is the issue?

We're observing 99% ~ 99.9% SR with a small Go service. The only traffic it receives are Prometheus scrapes (from our own Prometheus, not `linkerd-prometheus`). Both the Go service and our Prometheus are meshed.

Assuming prometheus scrapes every 10s, and goal of 99.9% (1 failure out of every 1000 requests), this is one failure every ~2.8 hrs == 10s * 1000 requests / 3600.

How can it be reproduced?

- Deploy a Go process in AKS
- Configure Prometheus to scrape the Go process every 10 seconds
- Mesh both sides of the connection

Dogfooding Dive

Helpful issue: improving Linkerd’s debug tooling

Include io error message on http metrics #4364

Closed **adleong** opened this issue on May 8 · 2 comments

adleong commented on May 8 Member

When an HTTP request fails due to an io error, the "error" label on the http response metric is "unclassified". This can make it very difficult to determine the cause of requests which fail due to io errors such as the remote closing the connection.

We should propagate the io error message onto the HTTP metrics to make these errors easier to identify and debug.

adleong mentioned this issue on May 8

Prometheus scrapes to Go Processes around 99.9% SR #3836 Closed

Dogfooding Dive

The fix: updating a timeout in Linkerd’s admin server

remove admin server timeouts #4350

Merged **adleong** merged 2 commits into `master` from `alex/an-idle-connection-is-the-devils-playground` on May 8

Conversation 7 | Commits 2 | Checks 15 | Files changed 1

adleong commented on May 7 • edited

Member

The Linkerd control plane components' admin servers have an idle connection timeout of 10 seconds. This means that they will close connections which have been idle for 10 seconds. These components are also configured with a 10 second period for liveness checks. This introduces a race condition where connections will be idle for approximately 10 seconds between liveness checks and can idle out, potentially causing the next liveness check to fail.

We remove the idle timeout so that the connection stays alive.

Dogfooding Dive

Look at those beautiful green 100% charts! 😻

Service level objectives

99% of all requests over a rolling 7-day window are served successfully.

Error budget
100%

Compliance
100%

SLI (Success Rate)
100%

551,570 requests can be served unsuccessfully.

Sharing (clusters) is caring

- Linkerd team doesn’t frequently get to do live debugging with a user
- Sharing our cluster helped the Linkerd team pinpoint an issue they’d been hearing about but couldn’t replicate (https://github.com/linkerd/linkerd2-proxy/pull/397)
- Spinning up a staging cluster for the Linkerd team permanently helped Dive’s cluster setup

Sharing (clusters) is caring

The investigation: a Dive engineer running Wireshark while pairing with a proxy maintainer

siggy commented on Nov 20, 2018

staggered timeout values, points to `LINKERD2_PROXY_BIND_TIMEOUT`. all bidirectional gRPC streams get cancelled after 30 seconds:

```yaml
- name: LINKERD2_PROXY_INBOUND_CONNECT_TIMEOUT
  value: 20s
- name: LINKERD2_PROXY_OUTBOUND_CONNECT_TIMEOUT
  value: 25s
- name: LINKERD2_PROXY_BIND_TIMEOUT
  value: 30s
```

http2.streamid == 17

No.	Time	Source	Destination	Protocol	Length	tcp.srcport	tcp.port	http2.streamid	http2.flags.end_stream	http2.type	Info	http2.rst_stream.error
227	20.970336	192.168.1.203	23.96.100.53	HTTP2	100	59426	59426,8086	17	False	HEADERS	HEADERS[17]	
230	20.971997	192.168.1.203	23.96.100.53	HTTP2	1414	59426	59426,8086	17	False	DATA	DATA[17]	
253	22.903860	192.168.1.203	23.96.100.53	HTTP2	1414	59426	59426,8086	17	False	DATA	DATA[17]	
275	24.870462	192.168.1.203	23.96.100.53	HTTP2	1414	59426	59426,8086	17	False	DATA	DATA[17]	
297	26.781764	192.168.1.203	23.96.100.53	HTTP2	1415	59426	59426,8086	17	False	DATA	DATA[17]	
319	28.842297	192.168.1.203	23.96.100.53	HTTP2	1415	59426	59426,8086	17	False	DATA	DATA[17]	
341	30.802133	192.168.1.203	23.96.100.53	HTTP2	1415	59426	59426,8086	17	False	DATA	DATA[17]	
348	30.866688	23.96.100.53	192.168.1.203	HTTP2	79	8086	8086,59426	17		WINDOW_UPDA…	WINDOW_UPDATE[17]	
365	32.795438	192.168.1.203	23.96.100.53	HTTP2	1415	59426	59426,8086	17	False	DATA	DATA[17]	
387	34.803211	192.168.1.203	23.96.100.53	HTTP2	1415	59426	59426,8086	17	False	DATA	DATA[17]	
408	36.793189	192.168.1.203	23.96.100.53	HTTP2	1414	59426	59426,8086	17	False	DATA	DATA[17]	
431	38.848856	192.168.1.203	23.96.100.53	HTTP2	1414	59426	59426,8086	17	False	DATA	DATA[17]	
453	40.928564	192.168.1.203	23.96.100.53	HTTP2	1414	59426	59426,8086	17	False	DATA	DATA[17]	
475	42.811100	192.168.1.203	23.96.100.53	HTTP2	1414	59426	59426,8086	17	False	DATA	DATA[17]	
479	42.875711	23.96.100.53	192.168.1.203	HTTP2	109	8086	8086,59426	0,0,17		WINDOW_UPDA…	WINDOW_UPDATE[0], PING[0], WINDOW_UPDATE[17]	
497	44.830284	192.168.1.203	23.96.100.53	HTTP2	1414	59426	59426,8086	17	False	DATA	DATA[17]	
519	46.927839	192.168.1.203	23.96.100.53	HTTP2	1414	59426	59426,8086	17	False	DATA	DATA[17]	
544	48.736976	192.168.1.203	23.96.100.53	HTTP2	1414	59426	59426,8086	17	False	DATA	DATA[17]	
568	50.719680	192.168.1.203	23.96.100.53	HTTP2	1414	59426	59426,8086	17	False	DATA	DATA[17]	
576	50.938284	192.168.1.203	23.96.100.53	HTTP2	79	59426	59426,8086	17		RST_STREAM	RST_STREAM[17]	CANCEL

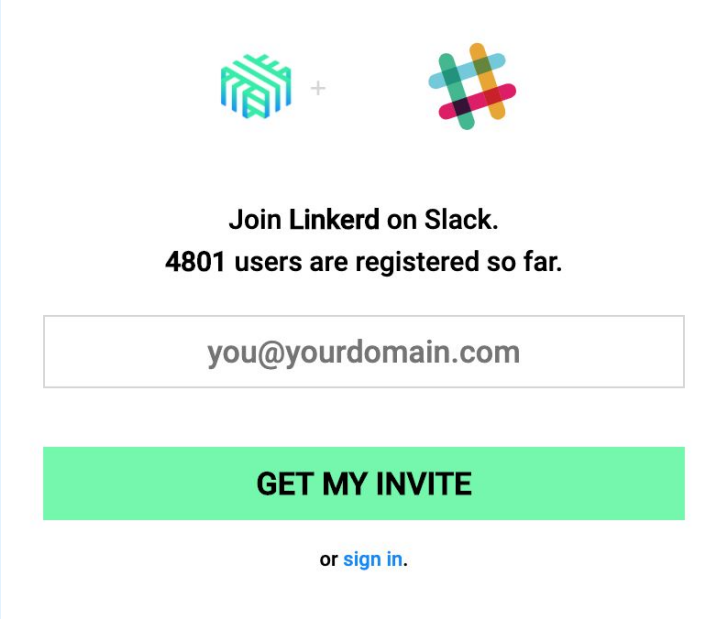
bcloud-agent.port.8086.with.proxy.timeouts.pcapng.zip

Conclusion

- Running Dive on Linkerd helped development in both
- “Tasting the mesh” led to a whole new class of bug discoveries
- Communication is key
- All Linkerd users benefit!

Join us on Slack!

https://slack.linkerd.io



DIVE

dive.co

@divedotco

divecommunity.slack.com

LINKERD

linkerd.io

@linkerd

slack.linkerd.io

github.com/linkerd

bit.ly/taste-the-mesh

References

https://linkerd.io/2/tasks/using-ingress/#nginx

https://github.com/linkerd/linkerd2/issues/1883

https://github.com/linkerd/linkerd2/issues/3836

https://github.com/linkerd/linkerd2/issues/4471

https://github.com/linkerd/linkerd2/pull/4557

https://github.com/linkerd/linkerd2/pull/4244

https://github.com/linkerd/linkerd2/issues/4690

https://github.com/linkerd/linkerd2/pull/4732