BUOYANT

Creators of LINKERD

Circuit Breaking and Dynamic Routing Deep Dive

Flynn, Technical Evangelist for Linkerd

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

What do you need for the workshop?

- A Kubernetes cluster. I'll be using `civo`, but the type of cluster doesn't much matter.
- To follow along, you'll also need `kubectl`, `bat`, and the `linkerd CLI`.
- Check out the workshop source at

  https://github.com/BuoyantIO/service-mesh-academy/tree/main/dynamic-routing-and-circuit-breaking

BUOYANT
Creators of LINKERD

What do you need for the workshop?

- Clone the workshop repo and **MAKE SURE YOU'RE USING AN EMPTY CLUSTER.**

  cd dynamic-routing-and-circuit-breaking
  `bash setup-demo.sh`

- Using k3d or the like?

  ```
  OVERRIDE_EMISSARY_IP=127.0.0.1 \
  bash setup-demo.sh
  ```

BUOYANT
Creators of LINKERD

What's on the agenda?

- Brief description of circuit breaking and dynamic request routing!
- **Workshop: dynamic request routing!**
- **Workshop: circuit breaking!**
- Gotchas for both!
- **Workshop: debugging!**

BUOYANT
Creators of LINKERD

Dynamic Request Routing

Dynamic request routing

In 2.12 and earlier...

- TrafficSplits (via SMI extension) are the primary way of dynamically controlling routing
- These allow coarse-grained routing behavior, based on the service name and percentage of traffic to split

Examples:

- Route 1% of traffic to `Foo` to `Foo-new`, and 99% to `Foo-orig` (progressive delivery)
- Route 100% of traffic to `Foo` to `Foo-west` (multi-cluster / failover)

Dynamic request routing

- In 2.13, route traffic based on (almost) any attribute of the request:
  - Headers!
  - Verbs!
  - (But not body)
- Examples:
  - Progressive delivery anywhere in the call graph
  - A/B testing anywhere in the call graph
  - Per-user canaries
  - etc.

Dynamic request routing

- Configured with the Gateway API HTTPRoute resource
  - Same resource introduced in Linkerd 2.12 for route-based auth
  - New capabilities for dynamic request routing:
    - use `parentRef` to associate the HTTPRoute with the Service you want to affect
    - use `backendRefs` to describe where you want the traffic to go

Gateway API and GAMMA

- The Gateway API (https://gateway-api.sigs.k8s.io/) is a project within Kubernetes' SIG-Networking
  - Started in 2020 primarily focused on tackling the "annotations wild West" of the Ingress resource
  - Now at version 0.7.0 and looking to reach 1.0.0 this year
- The GAMMA initiative is part of the Gateway API
  - Started in 2022 to sort out how to use Gateway API for meshes
  - Linkerd is quite active in GAMMA

Gateway API and GAMMA

- 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.
- Things we like about Gateway API:
  - powerful
  - flexible
  - good path to have it included in Kubernetes by default
  - someone else maintains the CRDs 😇

Gateway API and GAMMA

- Things we're actively working on within Gateway API/GAMMA:
  - Many things you can't currently do!
    - e.g. you can't yet configure retries; this is kind of important!
  - We can't yet pass Gateway API conformance tests
    - tests originally required being an ingress controller to pass, and we're not
    - this is why we use the `policy.linkerd.io` APIGroup at the moment
    - this is changing very quickly, so it'll likely be a nonissue soon

Circuit Breaking

Circuit Breaking

- Even newer than dynamic routing, but oft-requested!
- Don't hammer a failing workload endpoint with yet more traffic
  - When failure is detected, stop delivering requests to the failing endpoint (open the circuit breaker)
  - After a little while, try another request
  - If that succeeds, start delivering requests again (close the breaker)

Circuit Breaking in 2.13

- **Limited implementation**:
  - 2.13 can only open breakers when too many consecutive failures happen
  - “Failure” means HTTP 5yz responses
  - 2.13 configures circuit breakers with annotations on a Service
  - All the annotations have “failure-accrual” in their name
  - Docs: https://linkerd.io/2.13/tasks/circuit-breakers/
- This will be changing in future releases.

Circuit breaking in 2.13:

- Break the circuit after four consecutive request failures:

```
balancer.linkerd.io/failure-accrual: consecutive
balancer.linkerd.io/failure-accrual-consecutive-max-failures: 4
```

- Try reenabling traffic after 30 seconds:

```
balancer.linkerd.io/failure-accrual-consecutive-min-penalty: 30s
```

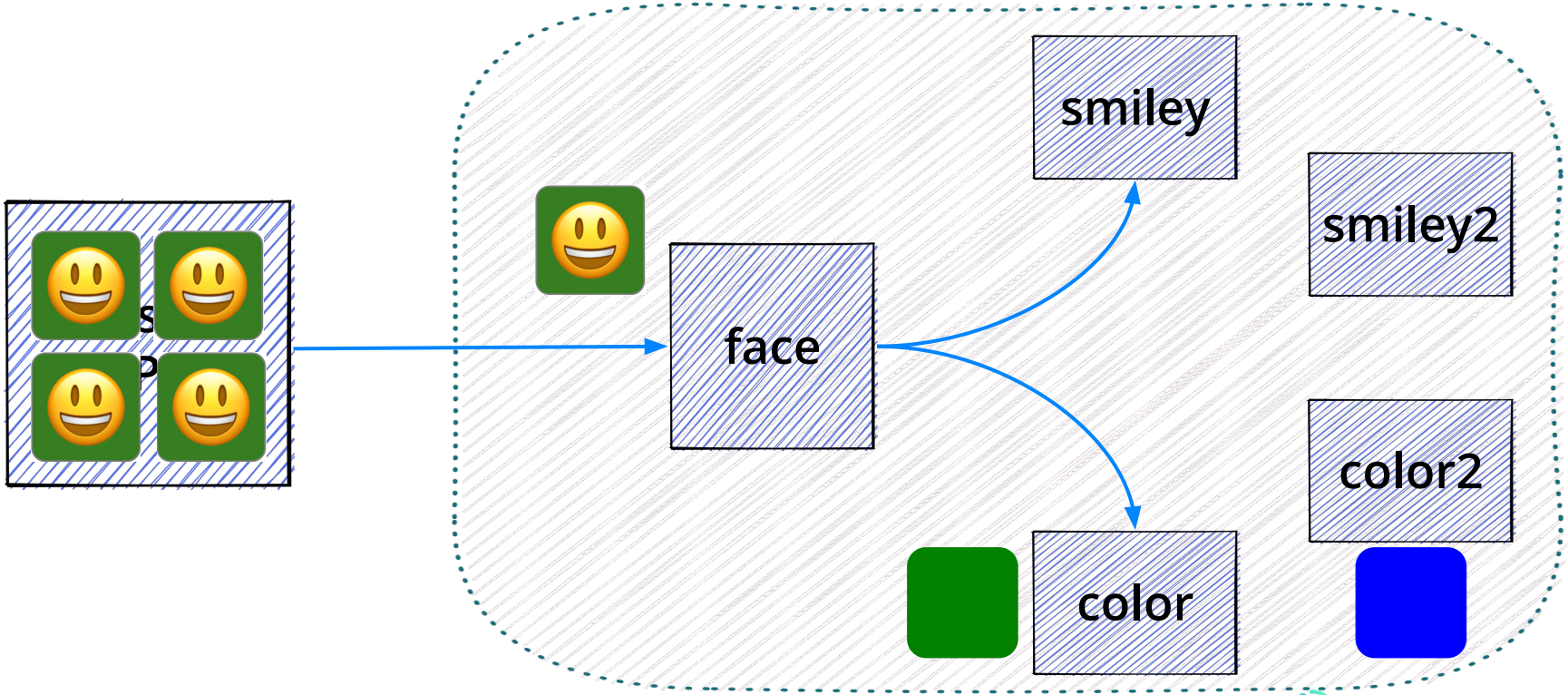
(This is for the first attempt. After that, the delay grows exponentially.)

- Don't ever wait more than 120 seconds between retries:

```
balancer.linkerd.io/failure-accrual-consecutive-max-penalty: 120s
```

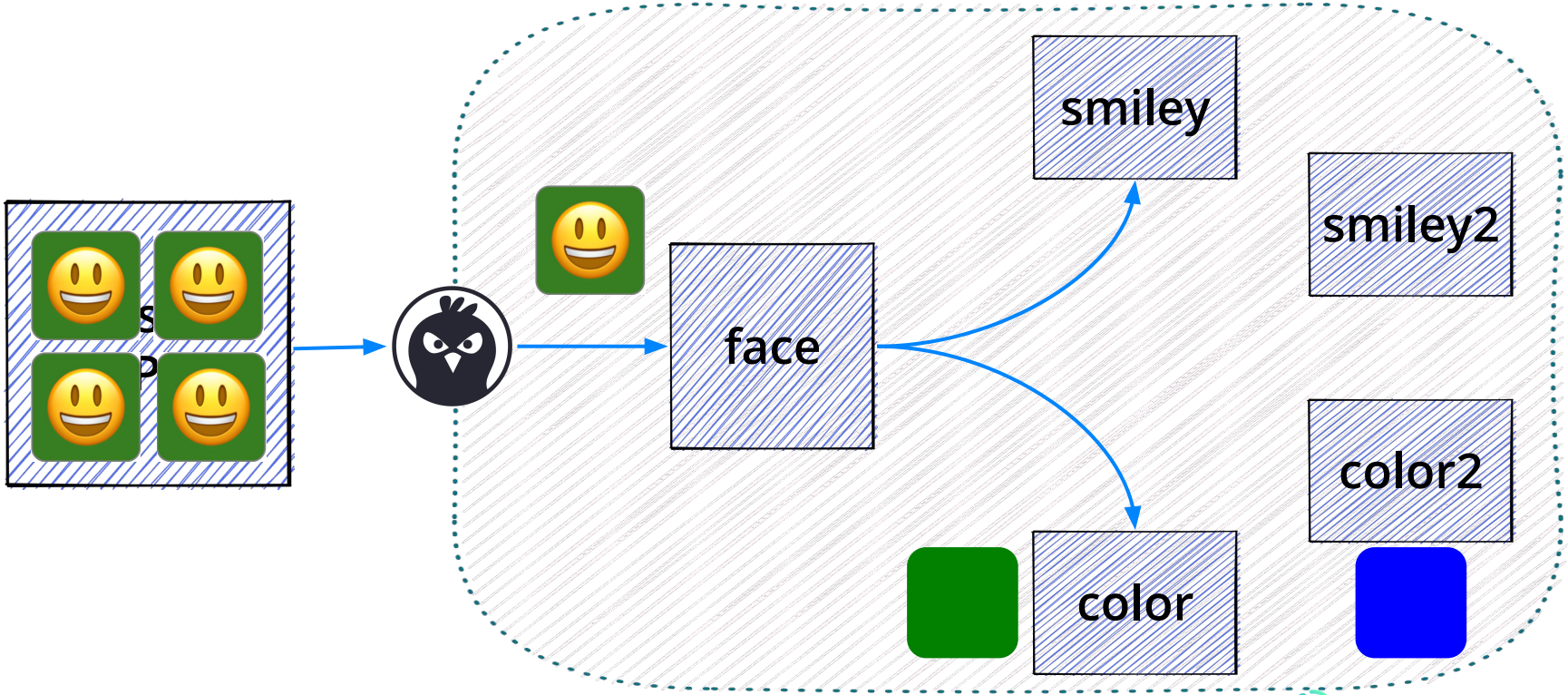
Demo Architecture

Demo Architecture



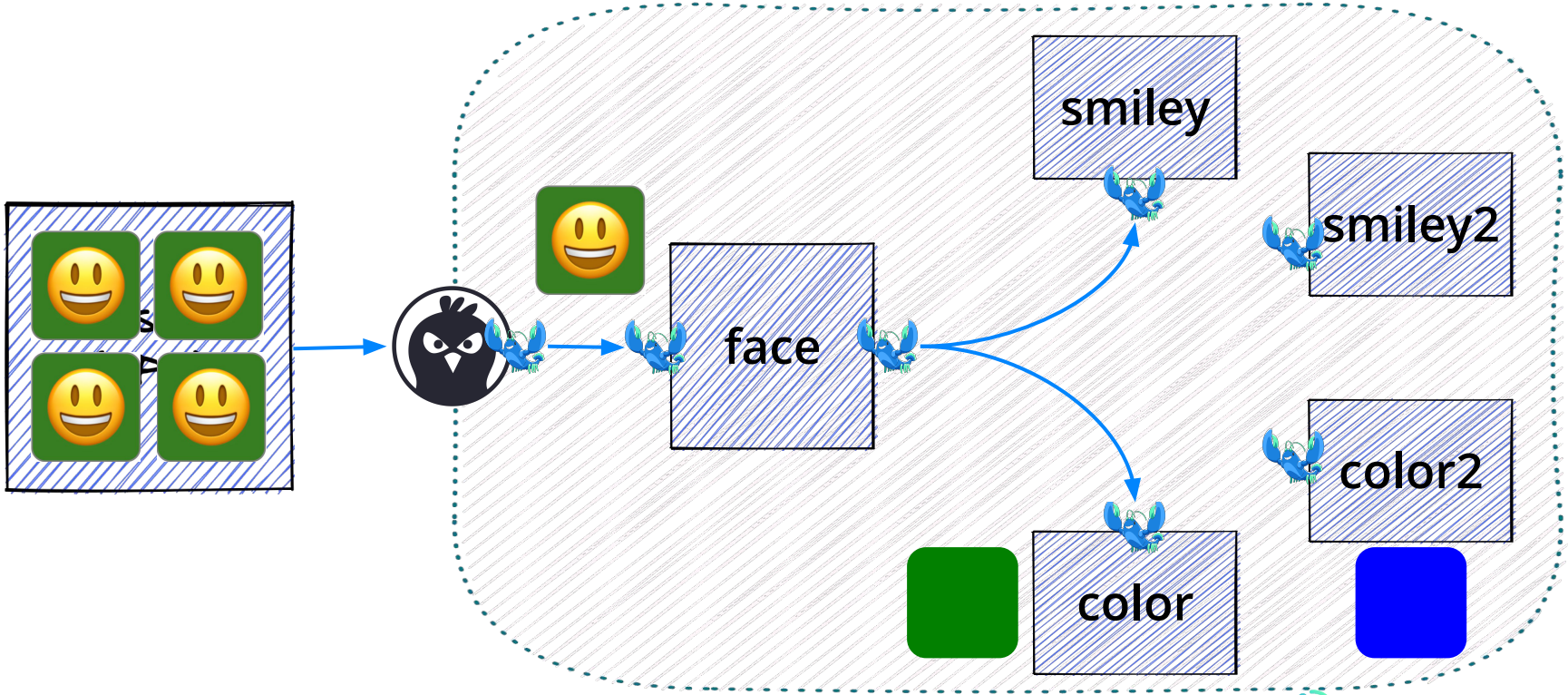
BUOYANT
Creators of LINKERD

Demo Architecture



BUOYANT
Creators of LINKERD

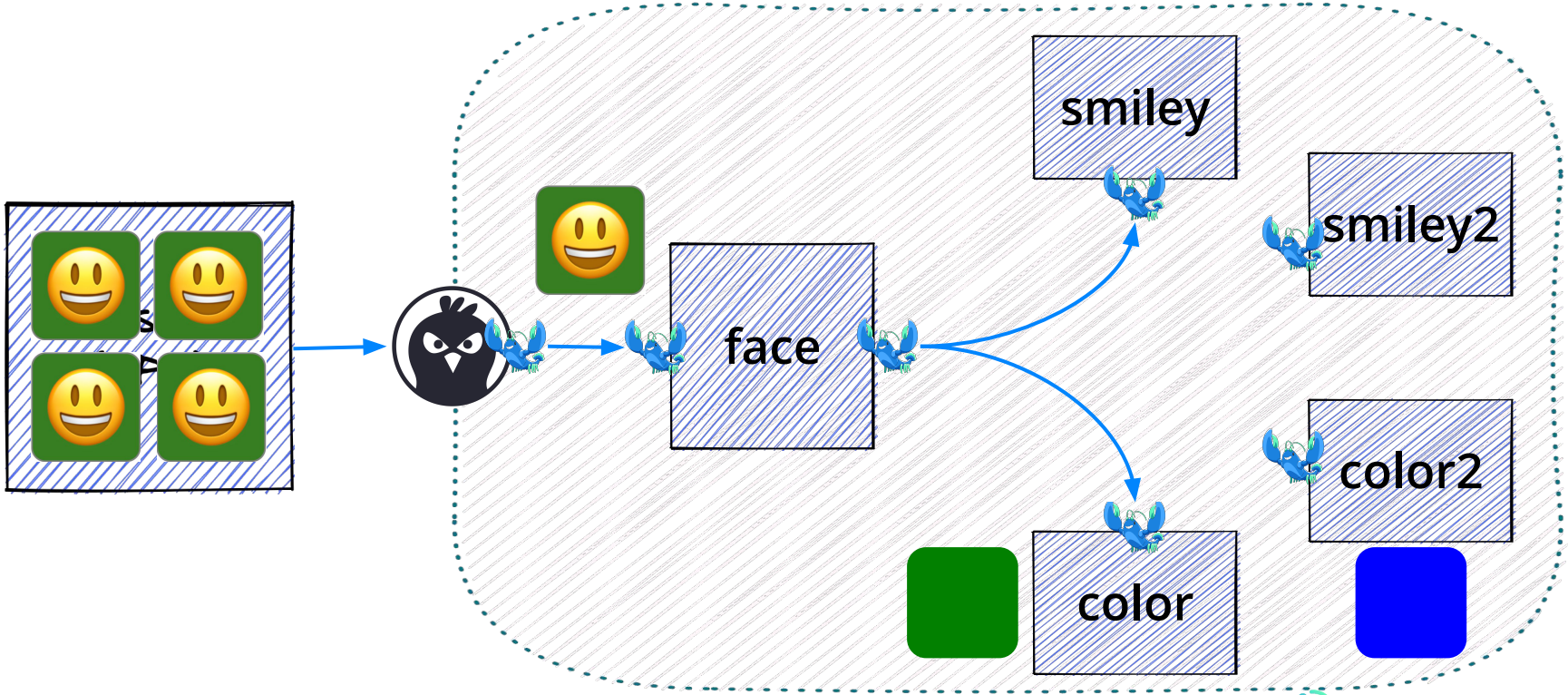
Demo Architecture



BUOYANT
Creators of LINKERD

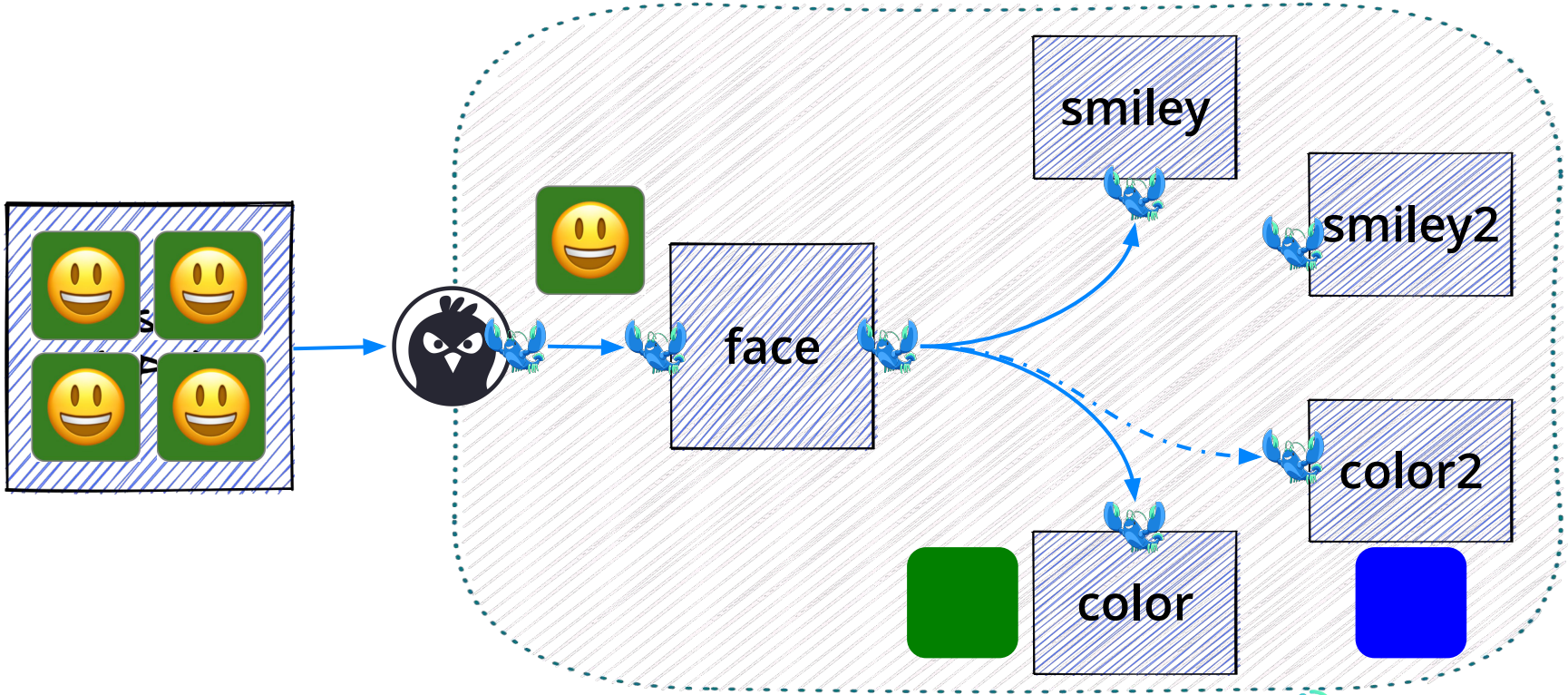
Demo Time

Demo Architecture



BUOYANT
Creators of LINKERD

Demo Architecture



BUOYANT
Creators of LINKERD

Gotchas

The Biggest Gotcha Of Them All

ServiceProfiles don’t compose with the shiny new features.

ServiceProfiles Don’t Compose With New Stuff

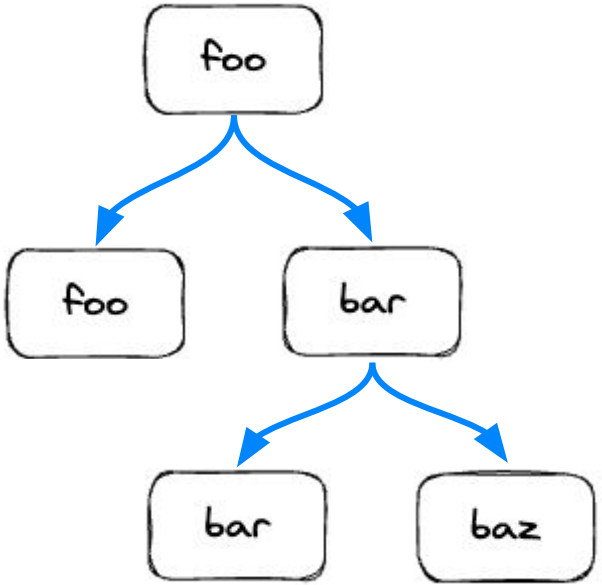
- If you have a ServiceProfile that defines routes, it will **take precedence** over
  - HTTPRoutes with conflicting routes
  - circuit breakers for workloads the ServiceProfile uses
- **This will be the case for the foreseeable future**
  - Doing it the other way around could yield too many surprising things when you upgrade
- There are still several things in 2.13 that you must do with ServiceProfile
  - Retries, timeouts, etc.
  - We are **actively working** on making all of this better. Quickly.

ServiceProfiles Don’t Compose With New Stuff

- Rules of thumb for debugging:
  - If your routes or breakers aren’t working, make sure you have no ServiceProfiles
  - If you have ServiceProfiles and remove them, you might need to restart Pods
    - Basically, the Linkerd proxy needs to decide if it’s running in 2.12 mode or 2.13 mode, and it might not switch cleanly.
  - The new `linkerd diagnostics policy` command can help when debugging routing issues.
- Back to the demo for a moment…

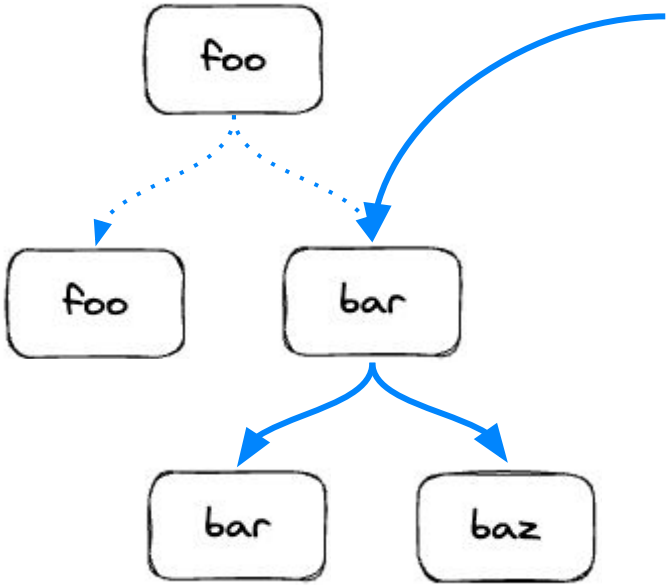
Dynamic Request Routing Gotchas

- HTTPRoutes don't stack
  - One HTTPRoute splits `foo` traffic 50/50 between `foo` and `bar`
  - Another HTTPRoute splits `bar` traffic 50/50 between `bar` and `baz`



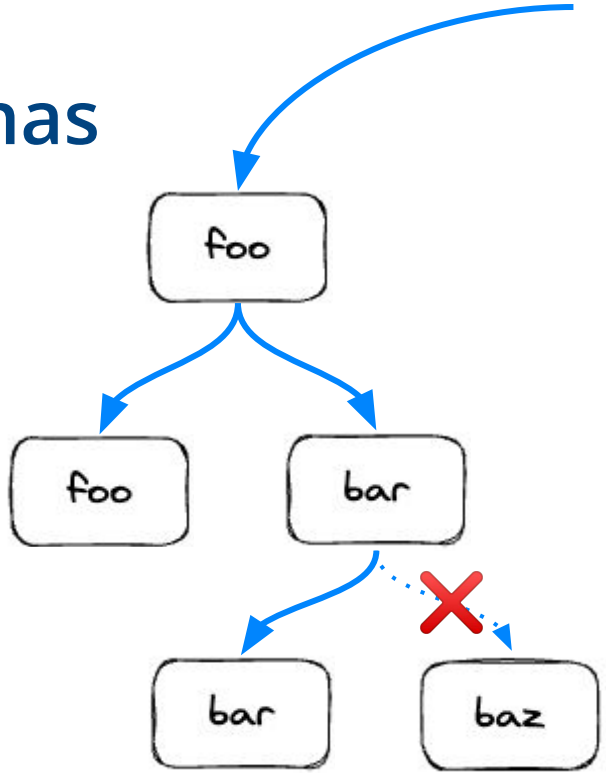
Dynamic Request Routing Gotchas

- HTTPRoutes don't stack
  - One HTTPRoute splits `foo` traffic 50/50 between `foo` and `bar`
  - Another HTTPRoute splits `bar` traffic 50/50 between `bar` and `baz`
  - Traffic sent directly to `bar` will get split between `bar` and `baz`



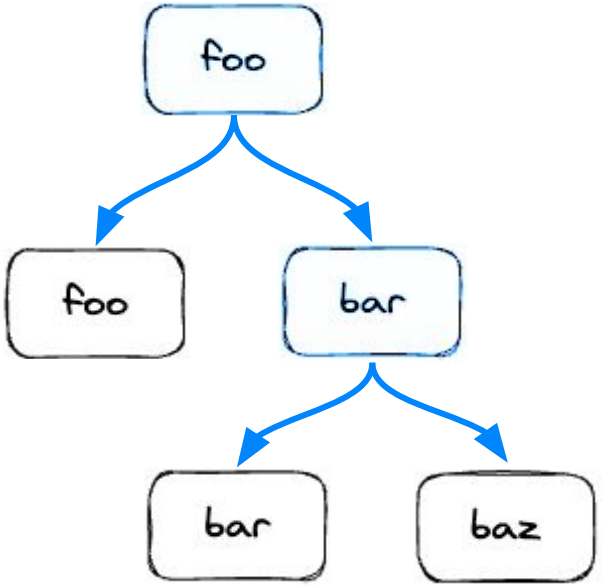
Dynamic Request Routing Gotchas

- HTTPRoutes don't stack
  - One HTTPRoute splits `foo` traffic 50/50 between `foo` and `bar`
  - Another HTTPRoute splits `bar` traffic 50/50 between `bar` and `baz`
  - Traffic sent directly to `bar` will get split between `bar` and `baz`
  - Traffic sent to `foo` will never get sent to `baz`



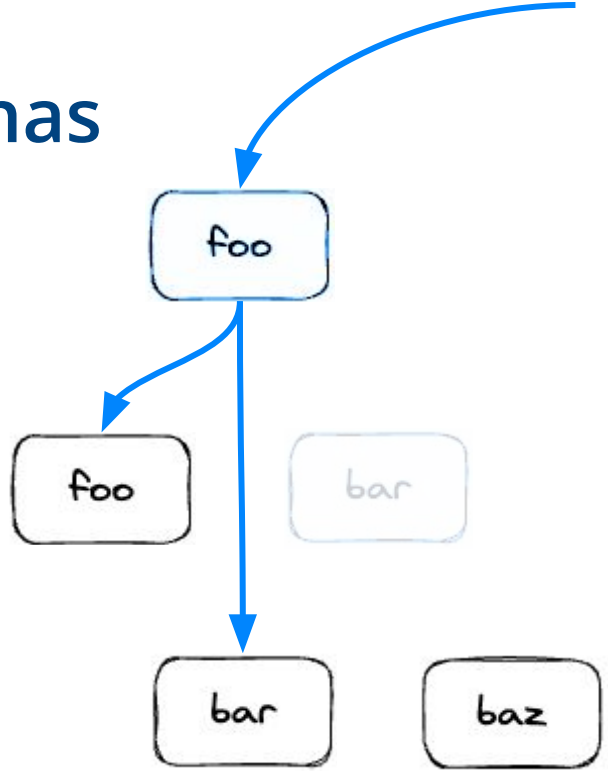
Dynamic Request Routing Gotchas

- HTTPRoutes distinguish between the “front end” of a Service and the “back ends”
- Routing only happens at the Service frontend (a ClusterIP)



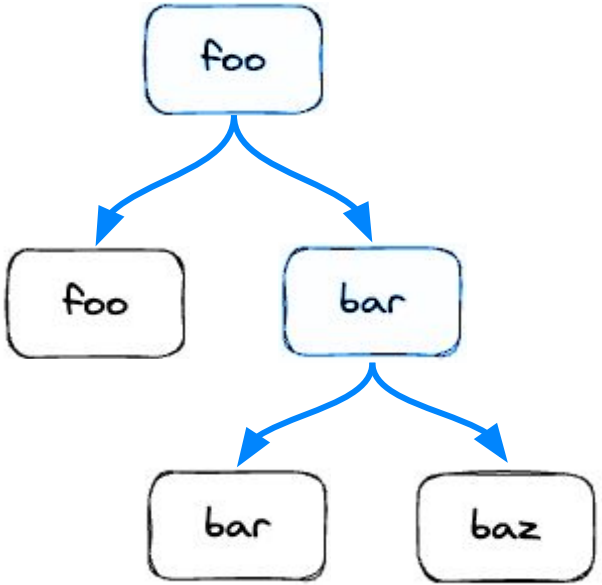
Dynamic Request Routing Gotchas

- HTTPRoutes distinguish between the “front end” of a Service and the “back ends”
- Routing only happens at the Service frontend (a ClusterIP)…
- …but once a routing decision is made, we go direct to a backend (an endpoint IP)



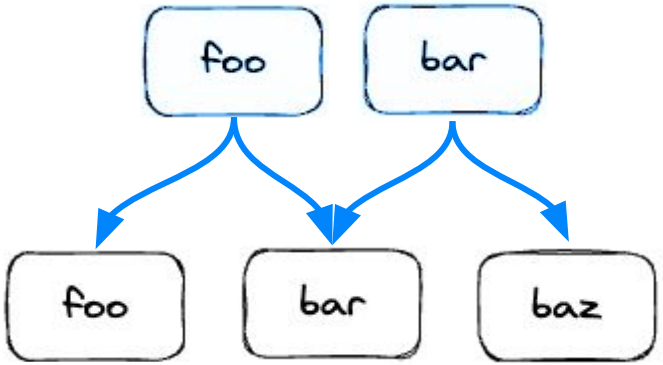
Dynamic Request Routing Gotchas

- So rather than setting up this…



Dynamic Request Routing Gotchas

- So rather than setting up this…
- …you were actually setting up this.



Q&A

BUOYANT'S SERVICE MESH ACADEMY

The creators of LINKERD

Monthly hands-on, engineer-focused training from the creators of the service mesh

June 15: Linkerd in Production 101: updated for 2.13!

SIGN UP TODAY!

buoyant.io/sma

NEW

Join Buoyant's LINKERD Forum

Unlike Slack, messages will stick around and be searchable, increasing usefulness for everyone.

Give it a try!

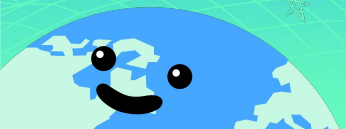
BUOYANT'S SERVICE MESH ACADEMY

The creators of LINKERD

NEW!

Fundamentals of the Service Mesh Online Course + Certification

With hands-on LINKERD labs



Fully automated LINKERD on any Kubernetes cluster

Buoyant Cloud automates upgrades, data plane version tracking, mesh health alerts, and much, much more.

BOOK A DEMO
buoyant.io/demo

BUOYANT

Creators of LINKERD

Thanks much!

flynn@buoyant.io

@flynn on slack.linkerd.io