Federating Clusters for Zero-Downtime Kubernetes

Dominik Táskai · Linkerd Ambassador

github.com/dtaskai/linkerd-multicluster-demo

Dominik Táskai

Senior Software Engineer @ GENESYS

Platform Engineering and Developer Experience

@dtaskai

@dtaskai

@dtaskai



LINKERD

A whole cluster disappears. Now what?

Failover as a runbook

Restore › **Repoint DNS** › **Wait**

An outage you already paid to survive, on paper.

The copy that never gets used

Your service already runs in another region, healthy and mostly idle.

When the main cluster fails, nothing sends it traffic.

Today: why the second copy can’t help on its own, what any fix has to solve, and one way to fix it.

Why whole clusters go away

The cluster is the blast radius. Replicas inside it don't help.

Regional or network outage

A zone, region or inter-region link fails. The obvious one, and rarely the most common.

Bad rollout or upgrade

A broken CNI, an admission webhook rejecting everything, a control-plane problem: one change, every workload.

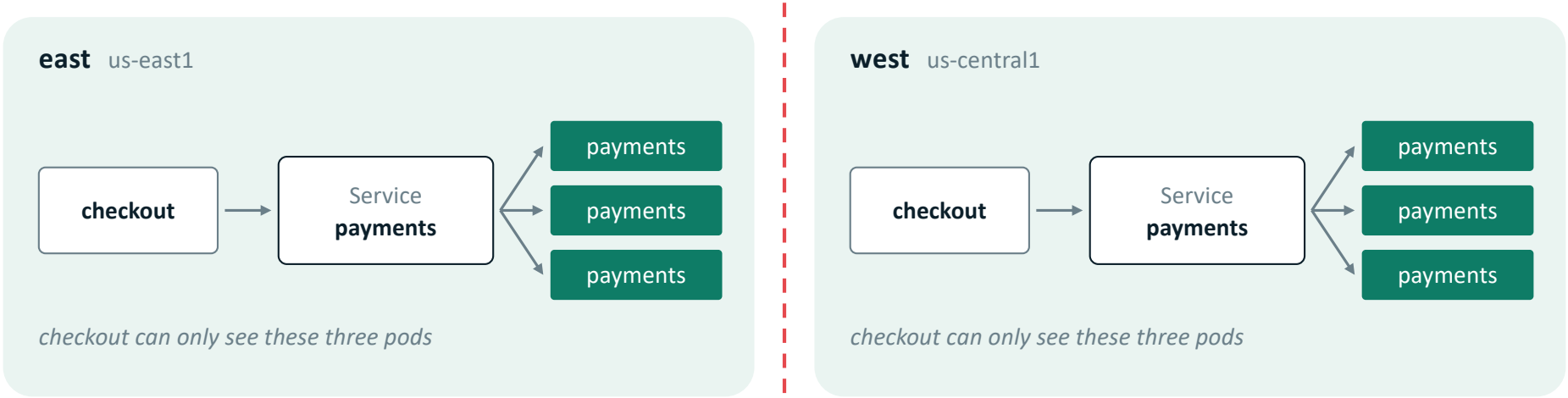
Capacity and quota exhaustion

Pod IP exhaustion, quota limits, node pools that can't scale up when you need them.

Expired cert or bad config

One cluster-wide dependency fails and everything that shares it fails together.

A Kubernetes Service stops at the cluster boundary



DNS

A cluster.local name only resolves inside the cluster that owns it.

Endpoints

The EndpointSlice lists local pods. Remote pods never show up.

Routing

kube-proxy only programs rules for local endpoints.

Identity

Each cluster issues its own service account identities.

Payments fails in east and checkout is stuck

1 Resolve

CoreDNS in east returns the payments ClusterIP.

```
10.112.4.20
```

›

2 Look up endpoints

The EndpointSlice is empty. All three pods are gone.

```
0 ready
```

›

3 Fail

Connection refused, or a 503 from the proxy in front.

```
503
```

›

4 Retry

Every retry lands on the same empty Service.

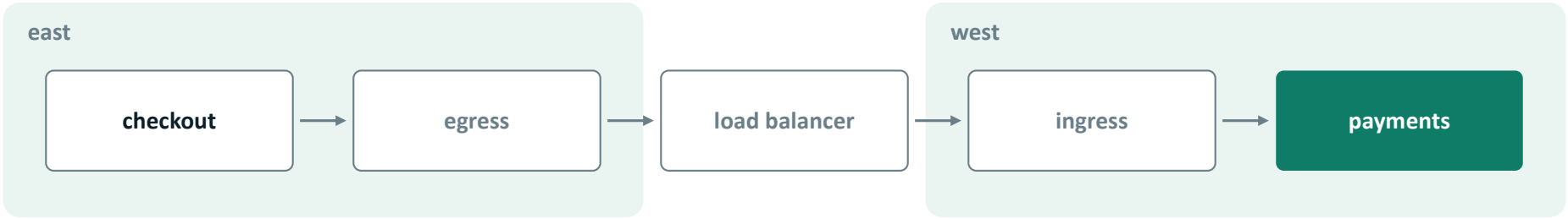
```
503, 503, 503
```

east payments 0 of 3 pods ready

west payments 3 of 3 pods ready, serving nothing

Nothing is wrong in west. Nothing in east knows west exists.

Crossing clusters today means the front door



`payments.west.example.com`

A second address

checkout needs west's external URL, plus code that decides when to use it.

Extra hops

A call between two services now goes out to a load balancer and back in.

Identity ends at the edge

mTLS stops at the ingress. West sees the load balancer, not checkout.

The wrong health signal

The load balancer checks the ingress, not whether payments pods are ready.

It works, and plenty of teams run this way. But every service pays for it, and none of it is automatic.

Where the usual workarounds break

DNS-based failover

TTLs and resolver caches keep clients on the dead side long after you flipped the record.

The untested standby

The passive copy drifted. Its first real traffic is the outage.

Slow detection

Health checks plus a human deciding to pull the trigger: minutes.

No capacity headroom

Survivors can't absorb the load. Failing over just moves the outage.

Partial failure

A half-dead cluster passes health checks and fails requests.

Identity and trust

Separate CAs mean no mutual auth. Cross-cluster mTLS becomes its own project.

What we want from the platform

	Runbook / DNS	Platform-level failover
Who decides	A human, off a dashboard	The endpoint list itself
Signal	DNS or external health check	Pods leave the load balancer
Recovery time	Minutes, if all goes well	Seconds
Client change	Per-app configuration	None
Identity	Bolted on later	Shared trust anchor, mTLS by default

Any fix has to solve four problems

Problem	The question it answers	If it's missing
1 **Discovery**	How does east learn that payments pods exist in west?	Remote pods never reach the endpoint list.
2 **Reachability**	Can a packet from east get to a pod in west, directly or through a gateway?	The name resolves, then the connection hangs.
3 **Identity**	Can west verify that the caller really is checkout?	No mTLS across the boundary, or it ends at the edge.
4 **Failover**	Who decides to send traffic to west, and how fast?	A person, a runbook, several minutes.

There’s more than one way to close the gap

Approach	Discovery	How traffic crosses	Worth knowing
Global DNS / GSLB	A DNS record per region, health-checked from outside	Through public or internal load balancers	TTLs and client caches set your failover time
Kubernetes MCS API (GKE MCS, Submariner)	ServiceExport and ServiceImport, one clusterset.local name	Straight to pods or ClusterIPs on a connected network	A standard for discovery. Workload mTLS is up to you
Cilium ClusterMesh	Global services, marked with an annotation	eBPF, pod to pod	Needs Cilium as the CNI in every cluster
Istio multicluster	Control planes read endpoints from every remote API server	Direct, or through an east-west gateway	Each control plane needs API access to the other clusters
Linkerd multicluster	A service-mirror controller per link copies remote Services	Through a gateway, or pod to pod	Mirror or federate, chosen per service with a label

Pick by what you already run and what your network allows. The rest of this talk uses Linkerd.

How Linkerd answers each of the four

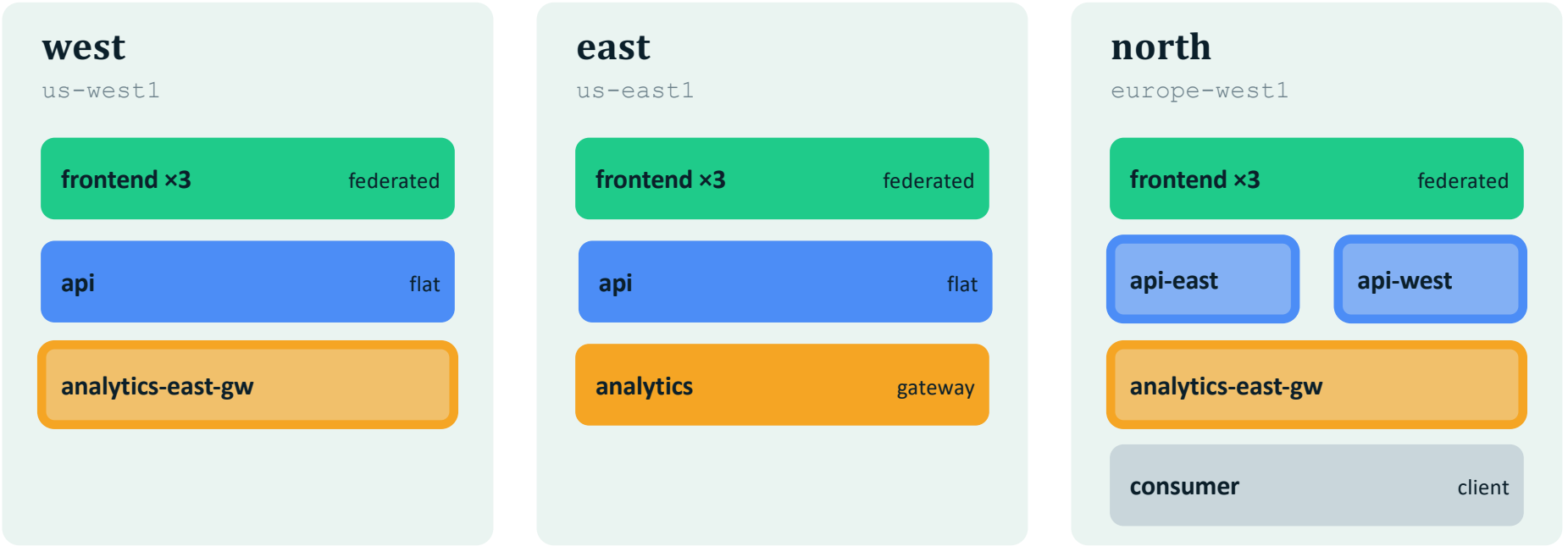
Problem	What Linkerd does	What you set up
1 Discovery	A service-mirror controller per Link copies exported Services from the remote cluster into the local one.	`Link + controller`
2 Reachability	Gateway mode needs only the gateway IP. Flat and federated modes go straight to remote pod IPs.	`Gateway or flat`
3 Identity	Every cluster chains to one trust anchor, so proxies use mTLS across the boundary.	`root.crt + issuers`
4 Failover	A federated Service holds endpoints from every member cluster. Dead pods drop out of the load balancer.	`federated=member`

Three modes, chosen per service by a label

Mode	Label	What happens	Network
Federated	`mirror.linkerd.io/federated=member`	Same-name services unioned into <svc>-federated, load balanced across all clusters	Flat (pod IPs routable)
Flat (pod-to-pod)	`mirror.linkerd.io/exported=remote-discovery`	Mirrored as <svc>-<cluster>, traffic goes straight to remote pods	Flat (pod IPs routable)
Hierarchical (gateway)	`mirror.linkerd.io/exported=true`	Mirrored as <svc>-<cluster>, traffic routed through a gateway	Gateway IP reachable

Hierarchical works on any network. Flat and federated need real pod-to-pod connectivity. Mix them freely on the same links.

The demo: 3 GKE clusters, 3 regions, 3 modes



Full-mesh links: 6 directional flat links + 1 gateway link from east (east-gw)

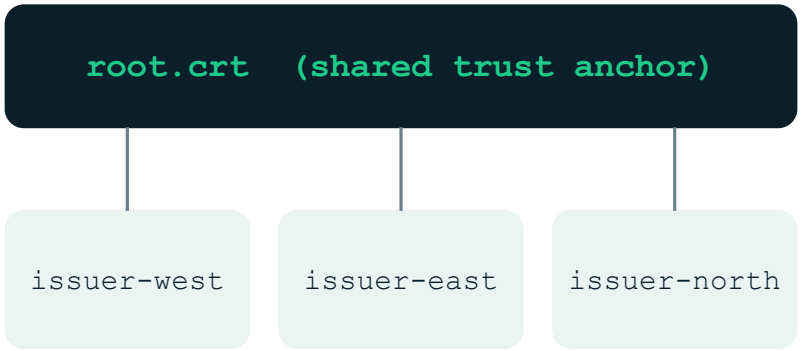
Foundations: flat network and shared trust

Address plan (must not overlap)

Cluster	Pod CIDR	Service CIDR
west	`10.100.0.0/14`	`10.104.0.0/20`
east	`10.108.0.0/14`	`10.112.0.0/20`
north	`10.116.0.0/14`	`10.120.0.0/20`

- Full-mesh VPC peering with import and export custom routes
- One zone per cluster, so num-nodes is the real node count

Trust hierarchy



Per-cluster issuers: rotate one compromised cluster without touching the others.

Links and controllers: one per link you consume

```
# west: gatewayless, one controller per link it consumes
linkerd --context west multicluster install --gateway=false \
  --set controllers[0].link.ref.name=east \
  --set controllers[1].link.ref.name=north \
  --set controllers[2].link.ref.name=east-gw \
  | kubectl --context west apply -f -

# flat link (no gateway)
linkerd --context east multicluster link-gen \
  --cluster-name=east --gateway=false \
  | kubectl --context west apply -f -
```

6 directional flat links

2 links from east: east and east-gw

3 / 2 controllers: west and north / east

Demo 1: from empty project to linked mesh

RUN

```
# step 0: config
cp env.example .env && source .env

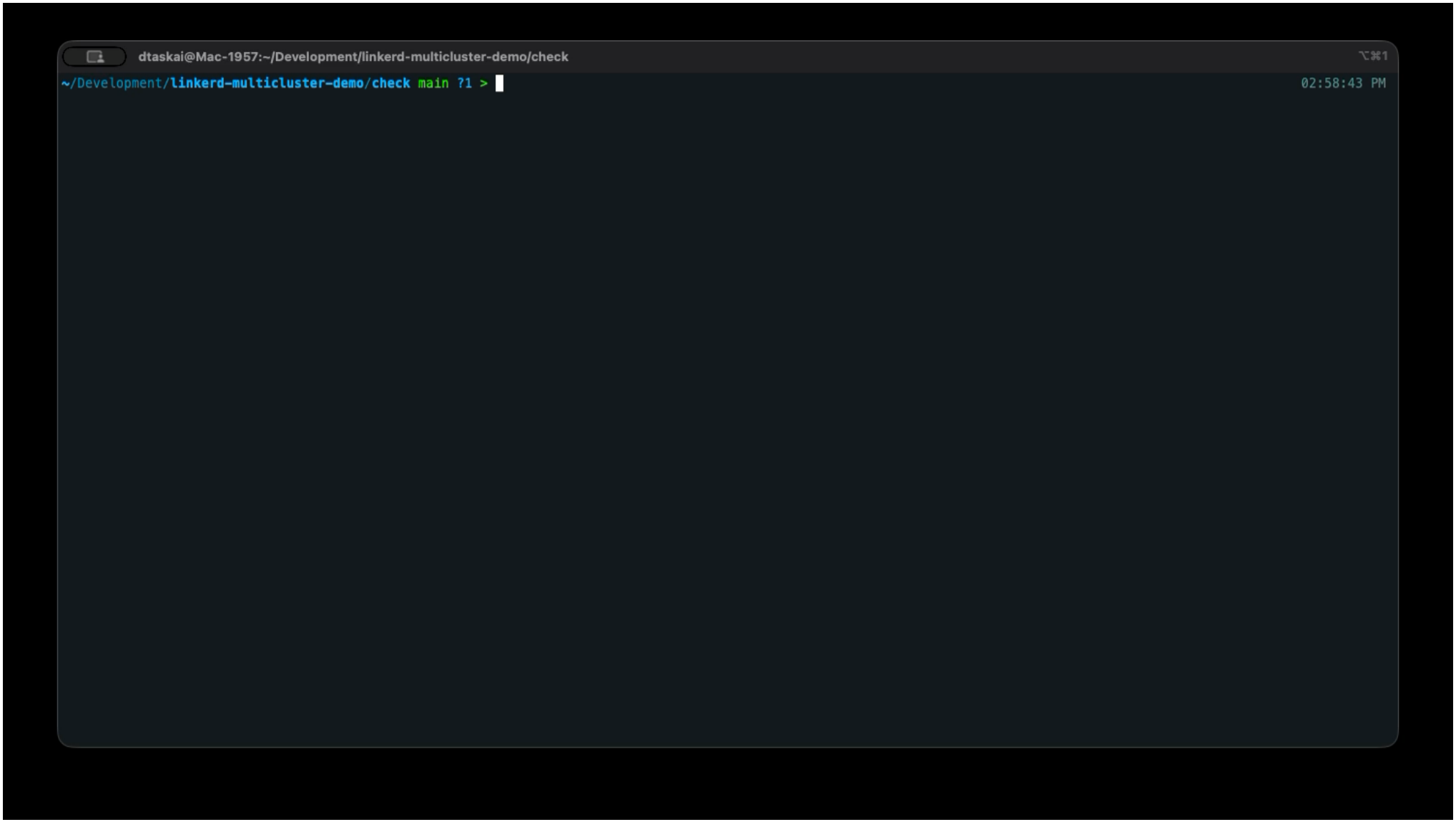
# pre-provisioned, walk through the script
./scripts/01-infra.sh

# shared trust anchor + control planes
./scripts/02-linkerd-install.sh

# multicluster + full-mesh links
./scripts/03-multicluster-setup.sh
linkerd --context <west east north> multicluster check
```

POINT OUT

- Only GCP_PROJECT needs changing in .env
- 01-infra.sh: non-overlapping CIDRs, peering routes, one zone per cluster
- 02: root.crt plus three issuer certs
- 03: six links plus east-gw, controllers per consumer
- multicluster check should report every linked cluster healthy


dtaskai@Mac-1957:~/Development/linkerd-multicluster-demo/check
⌥⌘1
~/Development/linkerd-multicluster-demo/check main ?1 >
02:58:43 PM

One label changes how a service crosses clusters

frontend
federated

```
kubectl --context $ctx -n mc-demo label \
  svc/frontend mirror.linkerd.io/federated=member
```

› `frontend-federated`

api
flat

```
kubectl --context <west east> -n mc-demo label \
  svc/api mirror.linkerd.io/exported=remote-discovery
```

› `api-west, api-east`

analytics
gateway

```
kubectl --context east -n mc-demo label \
  svc/analytics mirror.linkerd.io/exported=true
```

› `analytics-east-gw`

Demo 2: deploy and verify all three modes

RUN

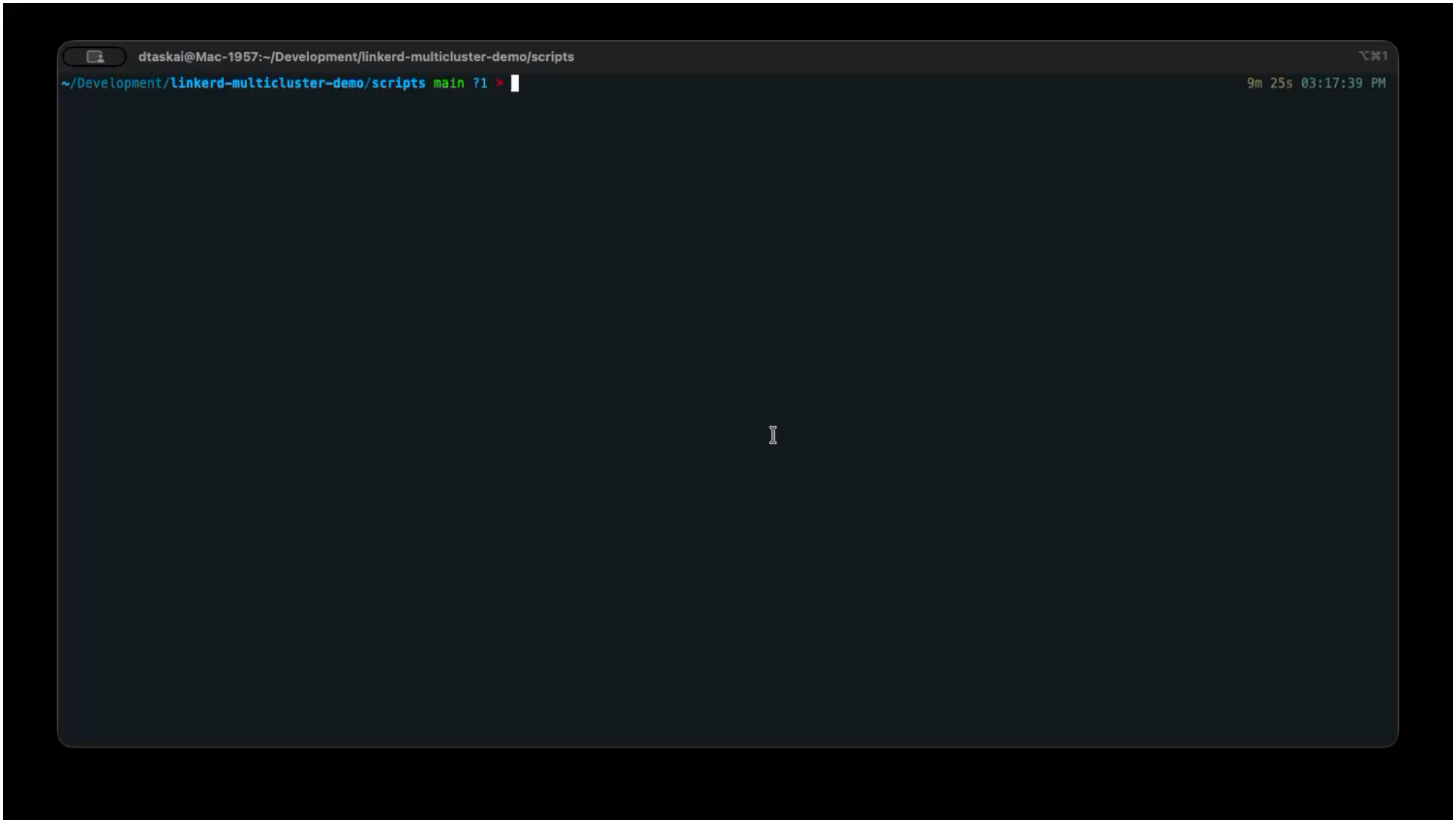
```
./scripts/04-deploy-app.sh

kubectl --context north -n mc-demo get svc

./scripts/05-verify.sh
# [federated]  frontend from east
# [flat-west]  api from west
# [gateway]    analytics from east
```

POINT OUT

- frontend-federated appears in every cluster within seconds
- Response strings reveal which cluster served each request
- Federated endpoints span all three clusters
- Flat mirror: endpoints are remote pod IPs
- Gateway mirror: one endpoint, east's gateway IP on :4143


dtaskai@Mac-1957:~/Development/linkerd-multicluster-demo/scripts
~/Development/linkerd-multicluster-demo/scripts main ?1 >
9m 25s 03:17:39 PM

Demo 3: chaos test, kill a whole cluster

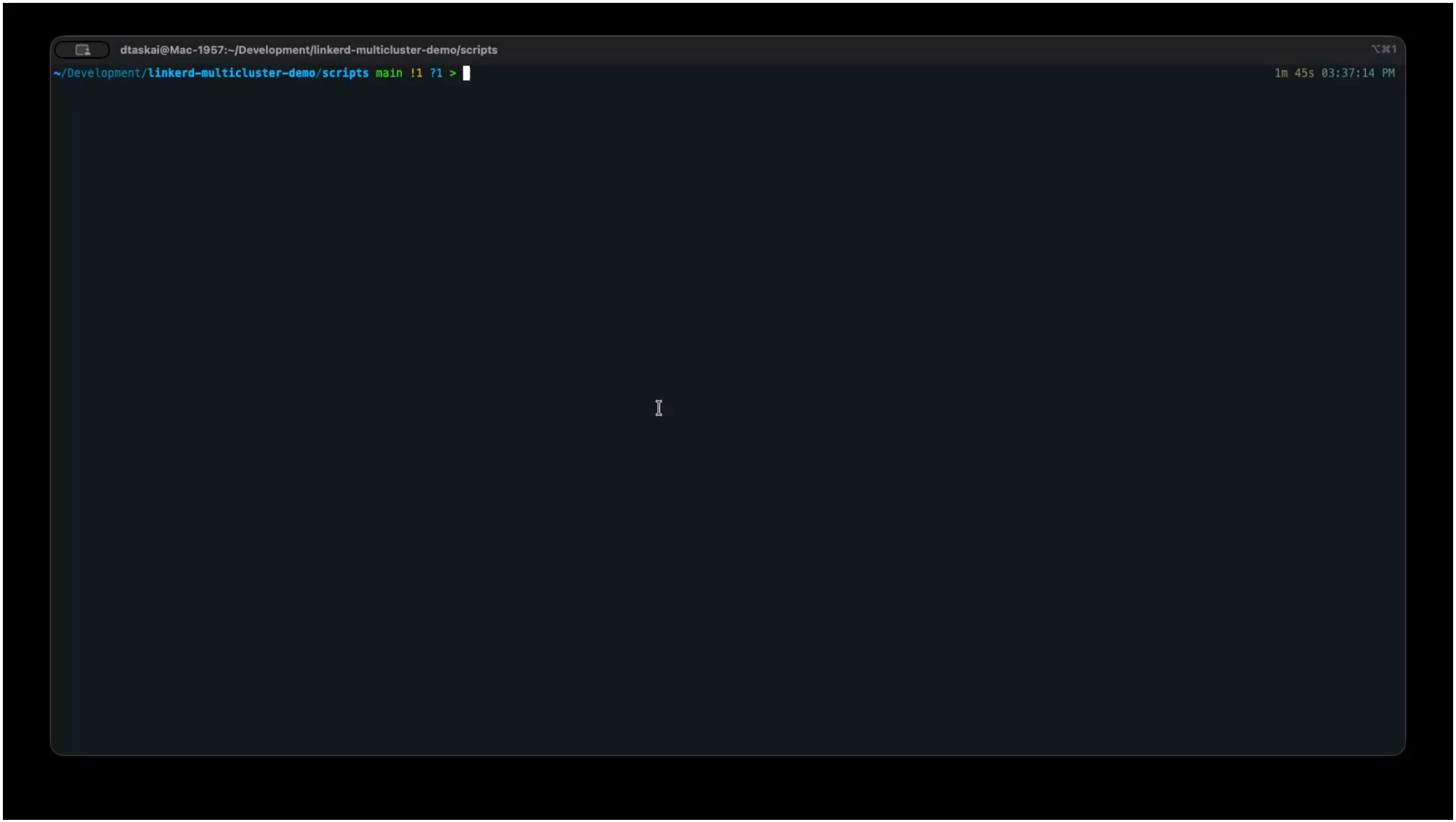
RUN

```
# scale every deployment in east to zero
./scripts/06-chaos-test.sh

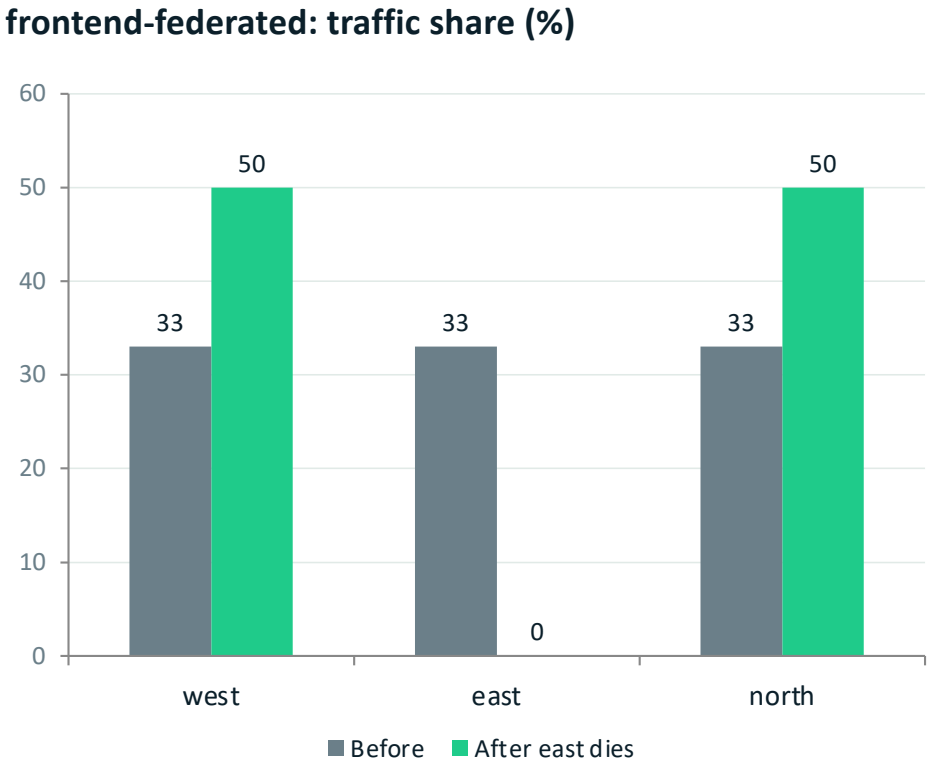
# bring east back
kubectl --context east -n mc-demo \
  scale deploy --all --replicas=1
kubectl --context east -n mc-demo \
  scale deploy/frontend --replicas=3
```

POINT OUT

- Federated: rebalances, zero errors, zero config changes
- api-east: 503s, the client asked for east explicitly
- analytics-east-gw: 502s, the gateway is down too
- Recovery takes 15-30 seconds
- Restore frontend to full replicas or east rejoins under-weighted (~14%)


dtaskai@Mac-1957:~/Development/linkerd-multicluster-demo/scripts
~/Development/linkerd-multicluster-demo/scripts main !1 ?1 >
1m 45s 03:37:14 PM

What the chaos test showed



Federated
Rebalances, zero errors

Flat: api-east
503s: remote pods are gone

Gateway: analytics-east-gw
502s: the gateway is down too

Federation fails over for you. Mirroring leaves the decision to the client.

Which mode should you pick?

Should the client be cluster-agnostic?	›	**Federated**
Need automatic failover with no app changes?	›	**Federated**
Client must pick a specific cluster?	›	**Flat mirror**
Traffic splitting with explicit weights?	›	**Flat mirror + TrafficSplit**
Singleton service in one cluster?	›	**Mirror (flat or gateway)**
No flat network between clusters?	›	**Gateway mirror**

Before you take this to production

- ✓ Bidirectional full-mesh links
- ✓ cert-manager with a shared CA
- ✓ Separate issuer cert per cluster
- ✓ NetworkPolicies for cross-cluster traffic
- ✓ Linkerd authorization policies
- ✓ Prometheus/Grafana: alert on federated endpoint count
- ✓ GitOps for Link CRs and multicluster config
- ✓ Test failover regularly in staging

Key takeaways

Federation

Automatic failover for services that can run anywhere. Not for ones pinned to a region.

Flat mirrors

Explicit, cluster-aware routing when data locality matters.

Gateway mirrors

Cross-cluster reach when a flat network isn't an option.

Start from the four problems, then pick the tool. With Linkerd, the per-service part is one label.

Q&A