
KubeCon
CloudNativeCon
North America 2024

KubeCon | CloudNativeCon

North America 2024

Topology Aware Routing

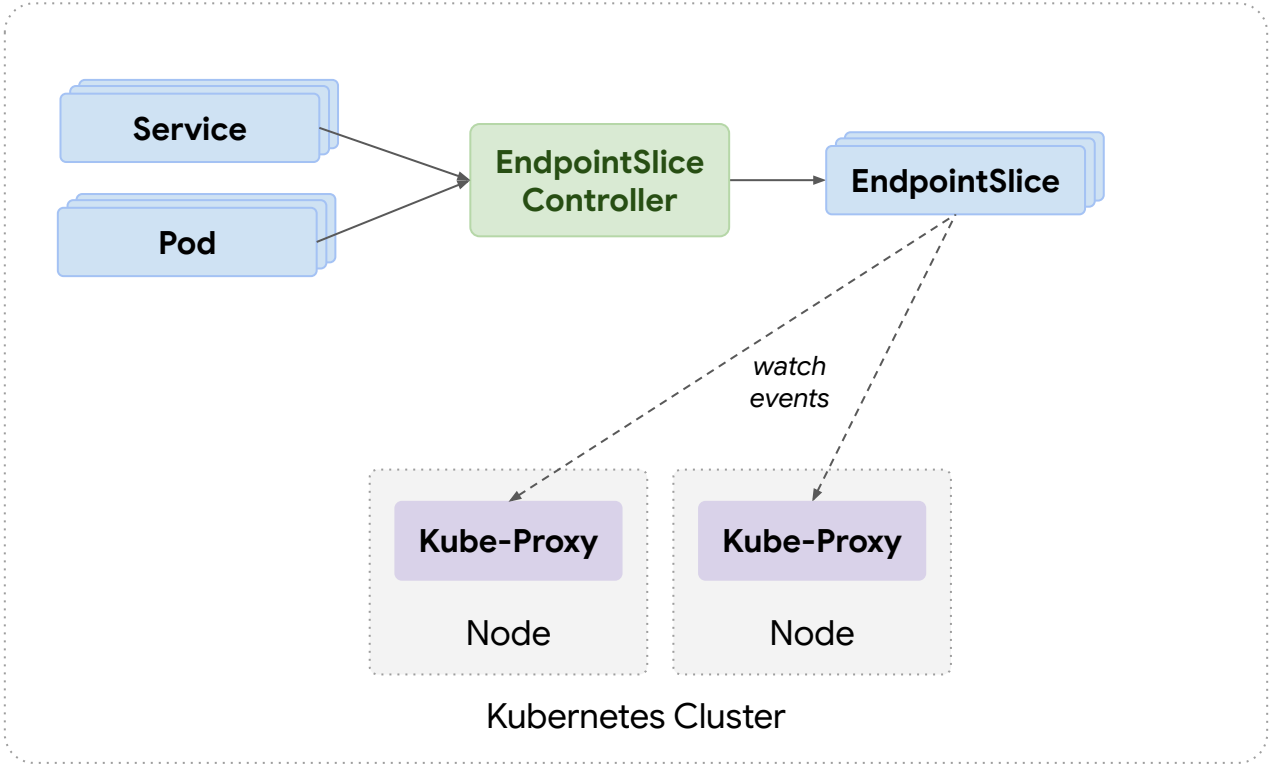
Understanding the Tradeoffs

Rob Scott, Google

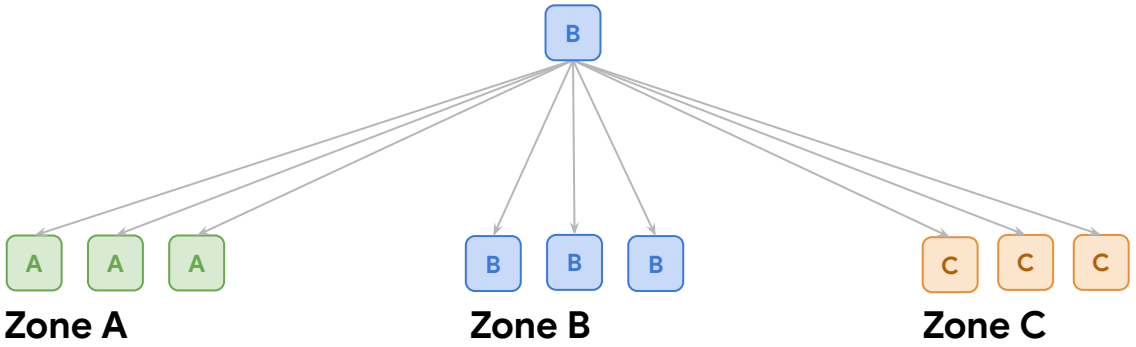
@robertjscott.ca

Context

Components


Service
Pod
EndpointSlice Controller
EndpointSlice
watch events
Kube-Proxy
Node
Kube-Proxy
Node
Kubernetes Cluster

Default: Spray Everywhere



Pros

- Least likely to overload traffic on a single endpoint
- Autoscaling is straightforward

Cons

- Can be expensive
- Does not provide zonal isolation


Wouldn’t it be nice if we could keep traffic closer to where it originated from?

The Ideal

- Keep traffic closer to where it originated from:
  - Minimize costs
  - Improve zonal isolation
- Fill closest endpoints and then spillover to next closest endpoints

What does “full” mean?

- It’s actually very difficult to define and may be different for everyone
  - Some combination of CPU, Memory, or GPU utilization
  - Number of active connections per endpoint
- Kubernetes API Server is not really a great solution for propagating this data
- Should we add some kind of new way to propagate per-endpoint metrics to dataplanes?

Something is better than nothing

- Don’t let the perfect be the enemy of the good
- Let’s start with something that’s achievable without boiling the ocean
- Will revisit a proper feedback loop once basic functionality in place


Attempt 1:
Topology Keys

Infinite Flexibility

REQUIRE Same zone OR region

```
topologyKeys:
- "topology.kubernetes.io/zone"
- "topology.kubernetes.io/region"
```

PREFER Same zone OR region

```
topologyKeys:
- "topology.kubernetes.io/zone"
- "topology.kubernetes.io/region"
- "*"
```

The Problems

- The spec was remarkably complicated to implement, and was never fully implemented
- There were no e2e tests for this
  - Most test suites run on a single cluster
  - e2e tests would need multiple variations of clusters/nodes with many different labels
- Although this approach was incredibly flexible, even the simple cases were relatively complex to configure
- This got stuck in alpha with no one able to drive it forward


Attempt 2:
Topology Hints

Working Theory

- Most people want to keep traffic close to where it originated from, as long as there's sufficient local capacity
- We should optimize for the common case and make it as simple as possible
- The traffic to each zone will be roughly proportional to the Nodes within that zone
- The vast majority of clusters exist within a single region
- The long tail of more complex approaches are better served with a highly customizable approach

Standard Topology Labels

- Standardize on the following labels:
  - `topology.kubernetes.io/region`
  - `topology.kubernetes.io/zone`
- Region and Zone are hierarchical
- Zones can not spread across regions
- These labels should be considered immutable
- A third key may be introduced in the future

How it Worked

Pods

Zone A

A A A A

Zone B

B B B

Zone C

C C

EndpointSlices

Original

For: All Zones

A A A
A B B
B C C

Topology Hints

For: Zone A

A A A

For: Zone B

B B B

For: Zone C

C C A

Key Terms

- **Desired Endpoints:** The number of endpoints a zone should be allocated
- **Overload:** The % difference between desired and allocated endpoints
- **Overload Threshold:** When the overload exceeds this, fall back to cluster-wide routing

Proportional Allocation

- Desired Endpoints for each zone are based on the **Allocatable Node CPU** in each zone
- A zone with 40 CPU cores should be allocated 2x more endpoints than a zone with 20 CPU cores
- Rationale: CPU Cores are roughly proportional to
  - The quantity of workloads scheduled in a zone
  - The expected number of requests we'd receive from the zone

Example

	Zone A	Zone B
CPU Cores	40	60
Endpoints by Location	5	5
Endpoints by Allocation	4	6

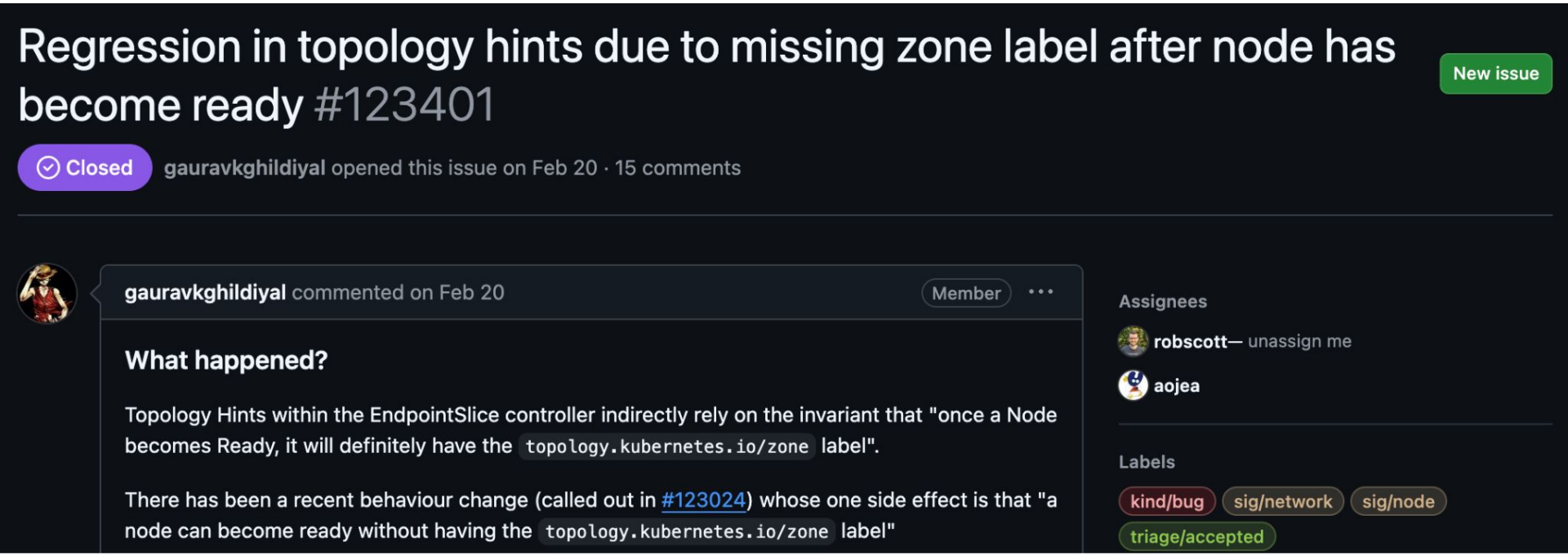
The guardrails we added were both confusing and fragile

Max Overload

# Endpoints	Per Zone	Max Overload	Hints Set?
3	1	0%	Yes
4	1.33	33%	No
5	1.67	67%	No
6	2	0%	Yes

When Overload > 20%, hints aren't set

Nodes Without Zone Labels



github.com/kubernetes/kubernetes/issues/**123401**


Attempt 3:
Traffic Distribution

Topology Aware Hints

Pods

Zone A

A A A A

Zone B

B B B

Zone C

C C

EndpointSlices

Original

For: All Zones

A A A
A B B
B C C

Topology Hints

For: Zone A

A A A

For: Zone B

B B B

For: Zone C

C C A

Traffic Distribution

Pods

Zone A

A A A A

Zone B

B B B

Zone C

C C

EndpointSlices

Original

For: All Zones

A A A
A B B
B C C

Topology Hints

For: Zone A

A A A A

For: Zone B

B B B

For: Zone C

C C

Definition

- `trafficDistribution: “PreferClose”`
- Indicates a preference for routing traffic to endpoints that are topologically proximate to the client. The interpretation of "topologically proximate" may vary across implementations and could encompass endpoints within the same node, rack, zone, or even region. **Setting this value gives implementations permission to make different tradeoffs, e.g. optimizing for proximity rather than equal distribution of load. Users should not set this value if such tradeoffs are not acceptable.**

Why Flexibility?

- Kubernetes ecosystem has grown to support many different dataplanes and ecosystems
- Some environments may have different topology constraints than zone and region
- Some dataplanes may be able to support more advanced heuristics, such as waterfall (“fill then spill”)
- Leaves room for additional terms and specificity as necessary

A Failure Story

The trouble with Topology Aware Routing: Sacrificing reliability in the name of cost savings

William Morgan

Jun 5, 2024 Blog › Linkerd

https://buoyant.io/blog/the-trouble-with-topology-aware-routing-sacrificing-reliability-to-avoid-cross-zone-traffic

What happens when things go wrong?

TAR is great when everything is healthy. But if one zone experiences problems—pods fail, or become slow, or traffic becomes unevenly distributed—the zone is left to its own devices.TAR prevents pods in the failing zone from *ever* being able to reach pods in the other zones to compensate.

https://buoyant.io/blog/the-trouble-with-topology-aware-routing-sacrificing-reliability-to-avoid-cross-zone-traffic

The Best Kind of Correct

- This was based on Pod health checks not catching the Pod being unavailable
- If no Pods in the local zone are healthy, traffic will spill over to other zones
- Example:
    - A zonal replica of a DB is unavailable and app health checks don’t cover that failure mode

1. **Without Topology Aware Routing**: you get a highly reliable multi-AZ cluster, but you have to pay for cross-zone traffic.
2. **With Topology Aware Routing**: you don't have to pay for cross-zone traffic, but you cannot recover from all in-zone failures by relying on other pods (even though they're in the same cluster.)

https://buoyant.io/blog/the-trouble-with-topology-aware-routing-sacrificing-reliability-to-avoid-cross-zone-traffic

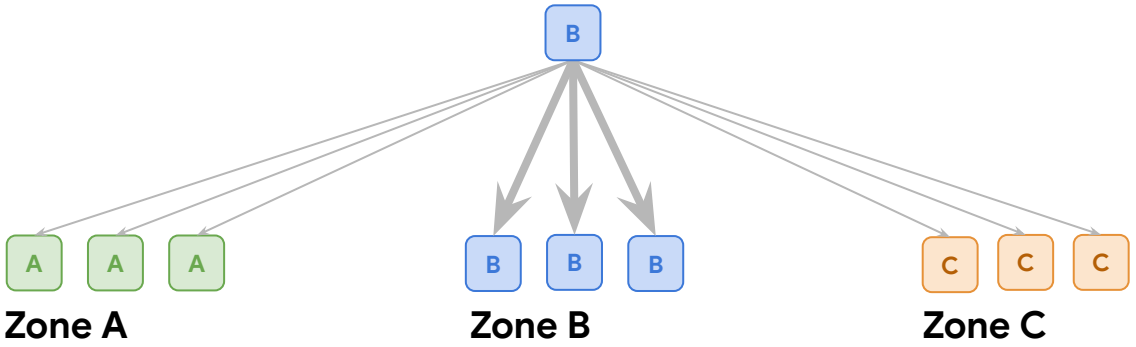
Recommendations

Topology Spread Constraints


```
topologySpreadConstraints:
- maxSkew: 1
  topologyKey: topology.kubernetes.io/zone
  whenUnsatisfiable: ScheduleAnyway
    labelSelector:
      matchLabels:
        app: example
```

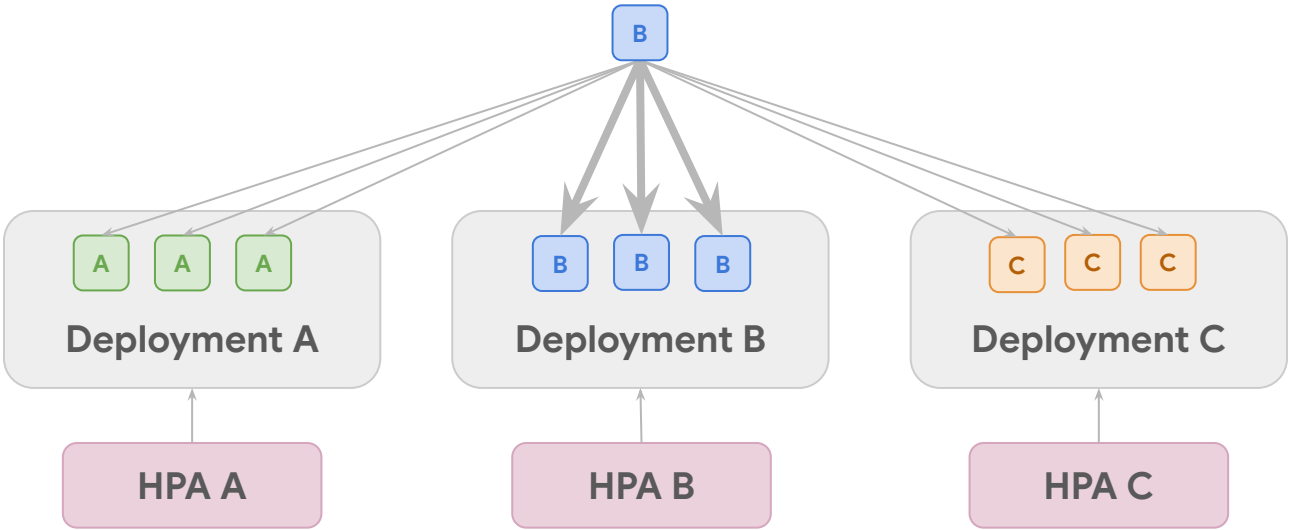
- For best results, use topology spread constraints when scheduling your workloads
- This will help ensure that your workloads are evenly distributed across zones

Autoscaling Interactions



- Routing traffic to the same zone increases the chance that endpoints in that zone will become overloaded
- Horizontal Pod Autoscaling may not scale up in this scenario (3 Pods in zone B at 100% util, 6 pods in other zones at 10% util == 40% avg util)

Autoscaling Recommendation



- Create separate deployment + HPA per zone

Recap

Recap

- Attempts:
  - 1: Topology Keys - infinite flexibility
  - 2: Topology Aware Hints - confusing attempt at guardrails
  - 3: Traffic Distribution - simple with some flexibility
- Tips:
  - Tradeoff: Preferring closer endpoints will negatively affect availability
  - Autoscaling: Pair with separate deployments and HPAs per zone
  - Otherwise, topology spread constraints will help keep Pods balanced

What’s Next

Further Integration

- All of the following will be implementation-specific. If you want your provider to support this feature, recommend sending a feature request:
  - Service type=LoadBalancer
  - Gateway API
  - Multi-Cluster Services (likely also needs to consider region)

Cilium + xDS

- Endpoint fullness - should we add some kind of new way to propagate per-endpoint metrics to dataplanes?
- xDS architecture is well equipped to solve this
  - Battle-tested load reporting service (LRS)
  - Centralized control plane can understand when spillover is necessary based on load reports
  - Scalability already proven with Envoy, gRPC, etc
  - Delta xDS - can propagate only what has changed

https://github.com/cilium/design-cfps/pull/14

Get Involved

- We could really use help to continue to push this forward
- Maybe your organization would benefit from the cost savings this would provide?
- What's needed:
  - More feedback (positive or negative)
  - Improved test coverage
  - Help pushing this to other integrations (Gateway, Multi-Cluster, LB, etc)

We’re Hiring

Google Cloud

Questions