KubeCon | CloudNativeCon

North America 2020

Virtual

Improving Network Efficiency with **Topology Aware Routing**

Rob Scott, Google

@robertjscott

Outline

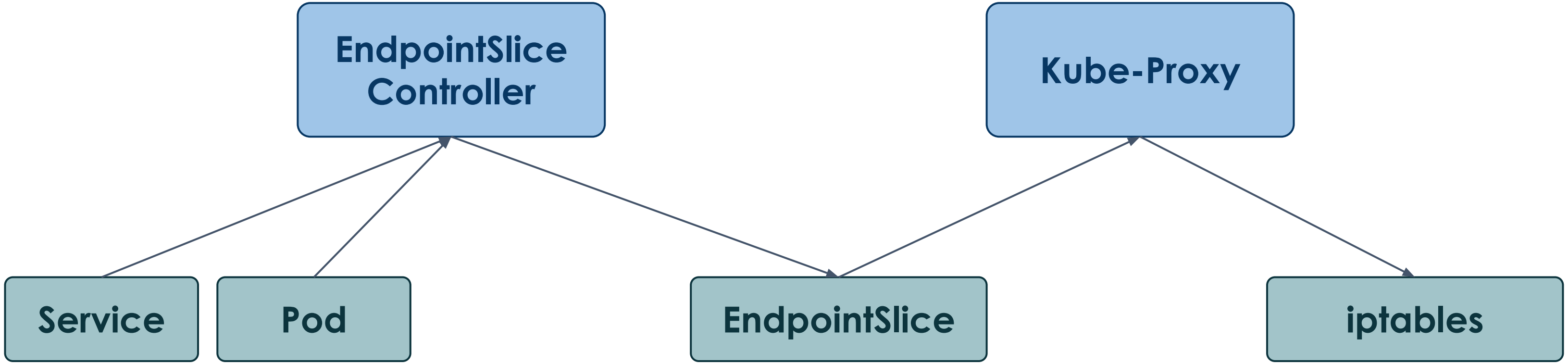
- Background
- Our first attempt
- Limitations
- Trying again
- Simulation results
- Long term vision

Disclaimers

- Topology aware routing is hard to get right
- This talk attempts to show the thought process behind our approach here
- Things could still change

Background

How it all works



iptables

```
-A KUBE-SVC-7PKYINUY4TAF2ZR4 -m statistic --mode random
  --probability 0.50000000000 -j KUBE-SEP-5FWDMO5BGH5HG6NF
-A KUBE-SVC-7PKYINUY4TAF2ZR4 -j KUBE-SEP-ZDPVNMFECRSHSEKA
```

- With iptables we rely on probabilities for load balancing.
- For a Service with 2 endpoints, the first endpoint will have a 50% chance of being chosen.
- If it is not chosen, the last endpoint has a 100% chance of being chosen.

Key Issues

- Traffic is distributed randomly across all endpoints, regardless of where it originates from
- In a 3 zone cluster, traffic is more likely to go to another zone than to stay in the current zone
- Every instance of kube-proxy needs to keep track of every endpoint in the cluster and manage iptables rules for them
- The larger a cluster gets, the more is required of kube-proxy and iptables - slower updates + more latency

Constraints

- Kube-Proxy doesn't handle requests directly, just programs iptables or ipvs.
  - Don't have visibility into request errors or timeouts.
  - Difficult to detect when an endpoint is overloaded.
- Kube-Proxy is deployed on each node, any significant changes could be expensive.
  - Endpoint updates need to be sent to each node.
  - More advanced logic increases CPU util on each node.

Our First Attempt

TopologyKeys

- We added a new alpha field to Services - `topologyKeys`
- Allowed endless flexibility
- Specify arbitrary topology keys in any order
- Kube-Proxy only routes to endpoints with matching labels
- A `*` could be used to indicate that traffic should be routed elsewhere if no labels matched

Examples

Require same zone or region

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

Examples

Prefer same zone or region

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

Limitations

Complexity

- Most users wanted the same thing - traffic should stay as close to where it originated from as possible.
- This approach required relatively complex configuration on each Service to achieve that.
- All the logic lived in kube-proxy:
    - Extra processing on each node.
    - All endpoints still needed to be delivered to each node.

Difficult to Implement

- Ideally topology keys would be given more weight if they appeared first
- This would be quite difficult to achieve without potentially overloading endpoints
- At first we just filtered endpoints matching any labels in topology keys
- If * was included in topology keys, all endpoints were passed through

The Ideal

- Ideally we would:
  - Prioritize endpoints matching earlier labels in the list.
  - Avoid overloading endpoints.
  - Avoid sending traffic nowhere.
  - Make * behave more like a failover configuration.
- This was quite difficult to achieve with such a flexible API.

Trying Again

Goals

- Build consensus around a small set of topology labels that will be clearly defined and officially supported.
- Develop a simple approach that covers most common use cases as automatically as possible.
- Only deliver the endpoints closest to each instance of kube-proxy to improve performance and scalability.

Goals

- Build consensus around a small set of topology labels that will be clearly defined and officially supported.
  - KEP 1659: Standard Topology Labels
- Develop a simple approach that covers most common use cases as automatically as possible.
  - KEP 2004: Topology Aware Routing
- Only deliver the endpoints closest to each instance of kube-proxy to improve performance and scalability.
  - KEP 2030: EndpointSlice Subsetting

Goals

- **Build consensus around a small set of topology labels that will be clearly defined and officially supported.**
  - KEP 1659: Standard Topology Labels
- Develop a simple approach that covers most common use cases as automatically as possible.
  - KEP 2004: Topology Aware Routing
- Only deliver the endpoints closest to each instance of kube-proxy to improve performance and scalability.
  - KEP 2030: EndpointSlice Subsetting

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

Goals

- Build consensus around a small set of topology labels that will be clearly defined and officially supported.
    - KEP 1659: Standard Topology Labels
- **Develop a simple approach that covers most common use cases as automatically as possible.**
    - KEP 2004: Topology Aware Routing
- Only deliver the endpoints closest to each instance of kube-proxy to improve performance and scalability.
    - KEP 2030: EndpointSlice Subsetting

Simulating Algorithms

- An automated approach meant we needed a good algorithm.
- We created a project to simulate the performance of different algorithms when run with millions of different inputs.
    - googleinterns/k8s-topology-simulator
- Evaluated 6 different algorithms, and found one that had the right combination of simplicity and performance

The Algorithm

Once a Service has enough endpoints, subset the EndpointSlices by zone. If a zone doesn't have enough endpoints, contribute some from a zone that does.

The Algorithm

Once a Service has enough endpoints, subset the EndpointSlices by zone. If a zone doesn't have enough endpoints, contribute some from a zone that does.

- Below a certain threshold, this approach results in a lot of churn and potential for overloaded endpoints
- Our testing showed that 3x the number of zones was a reasonable starting point
- We add padding on either side to prevent flapping between approaches

The Algorithm

Once a Service has enough endpoints, **subset the EndpointSlices by zone.** If a zone doesn't have enough endpoints, contribute some from a zone that does.

- We're introducing a new `endpointslice.kubernetes.io/for-zone` label that can be set on EndpointSlices.
- Kube-Proxy will be updated to only watch EndpointSlices where that label is not set or matches their current zone.

The Algorithm

Once a Service has enough endpoints, subset the EndpointSlices by zone. **If a zone doesn't have enough endpoints,** contribute some from a zone that does.

- Number of expected endpoints is based on the proportion of CPU cores in a zone.

Total Number of Endpoints:
12

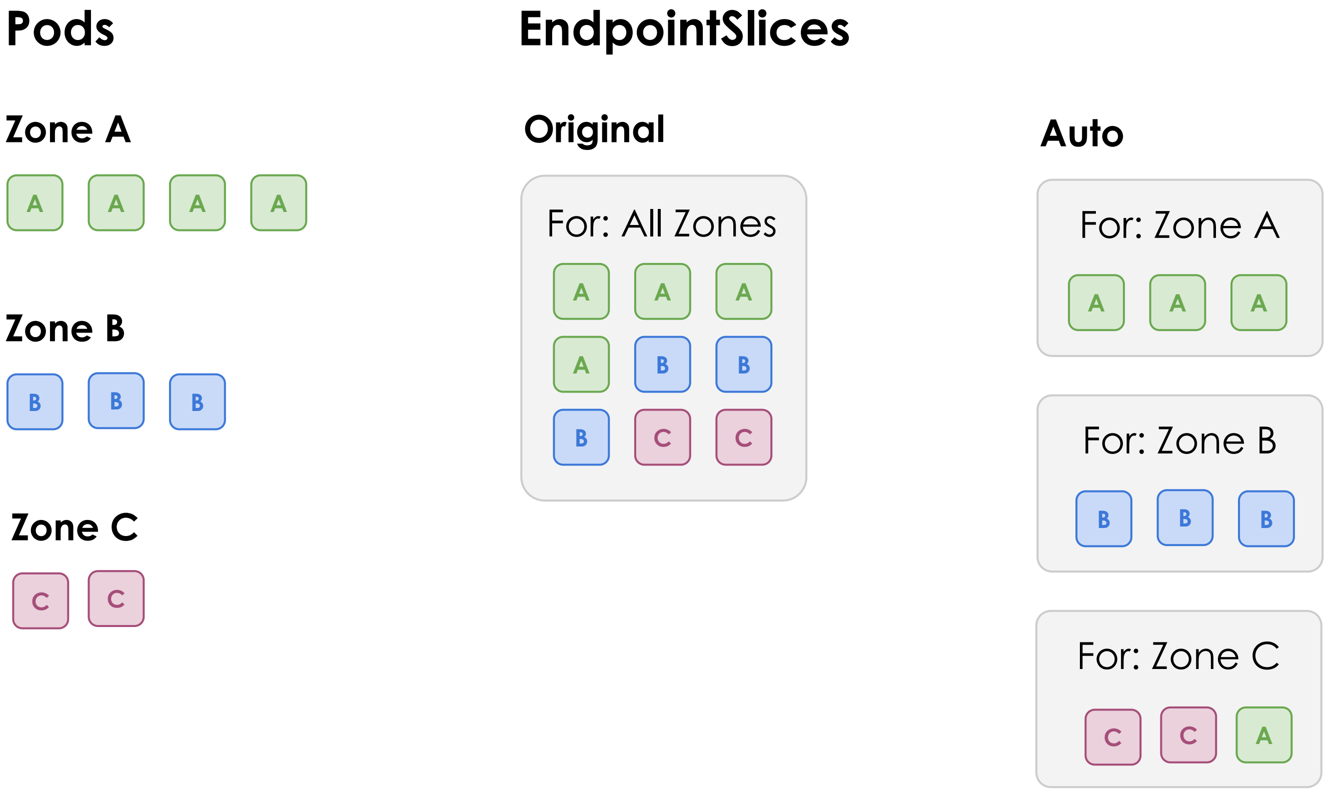
	CPU Cores	Expected Endpoints
zone-a	3	6
zone-b	2	4
zone-c	1	2

The Algorithm

Once a Service has enough endpoints, subset the EndpointSlices by zone. If a zone doesn't have enough endpoints, **contribute some from a zone that does.**

- To minimize churn, we only redistribute endpoints after a threshold has been passed
- We define an acceptable overload threshold, maybe 25%
- If we expected 10 endpoints in a zone:
  - 8 endpoints would be acceptable (10/8 => 25% overloaded)
  - 7 endpoints would not be (10/7 => 43% overloaded)

Example



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

Auto

For: Zone A

A A A

For: Zone B

B B B

For: Zone C

C C A

Goals

- Build consensus around a small set of topology labels that will be clearly defined and officially supported.
  - KEP 1659: Standard Topology Labels
- Develop a simple approach that covers most common use cases as automatically as possible.
  - KEP 2004: Topology Aware Routing
- **Only deliver the endpoints closest to each instance of kube-proxy to improve performance and scalability.**
  - KEP 2030: EndpointSlice Subsetting

Delivering EndpointSlices

- EndpointSlices can be labeled with:
  - `endpointslice.kubernetes.io/for-zone`
  - `endpointslice.kubernetes.io/for-region`
- Kube-Proxy will be updated to watch EndpointSlices with a matching zone or region
- For backwards compatibility, Kube-Proxy will continue to watch EndpointSlices without any zone or region specified

Summary

- 2 official topology labels: zone and region
- EndpointSlices can be delivered to zones or regions
- Users can opt-in to automatic topology aware routing on each Service
    - This will likely start as an annotation, may be expanded in the future

Simulation Results

Evaluation Criteria

- Percent of traffic that stayed In-Zone (45%)
- Overload - the proportion of extra traffic that any single endpoint might receive in a simulation
    - Max overload (20%)
    - Mean overload (20%)
- Proportion of new EndpointSlices required (15%)

Simulation Results

	Auto	Original
In-Zone Traffic	84.3%	38.8%
Overload	1.7%	0.0%
Extra Slices	36.9%	0.0%
Overall	**86.7%**	**72.5%**

Results from simulation of 39 million inputs for a 3 zone cluster.

Long Term Vision

In the next few months

- We need to test this in alpha and get feedback
  - Hopefully ready in Kubernetes 1.21
- Open questions:
  - How can we improve this approach?
  - Can we use a similar pattern for DNS?
  - What additional configuration will we need?
  - Can we eventually default to using this approach?

Longer term

- How can we implement topology aware routing with real time feedback?
- Ideally we could detect overloaded endpoints and route traffic elsewhere
- Can we do any of this redistribution of traffic without updating EndpointSlices on each change?

KubeCon | CloudNativeCon

North America 2020

Virtual

Thanks!

Rob Scott, Google

@robertjscott

KEEP CLOUD NATIVE EVERYWHERE

KubeCon | CloudNativeCon

North America 2020

Virtual