The untapped powers of services:

L7 Load Balancing without a Service Mesh

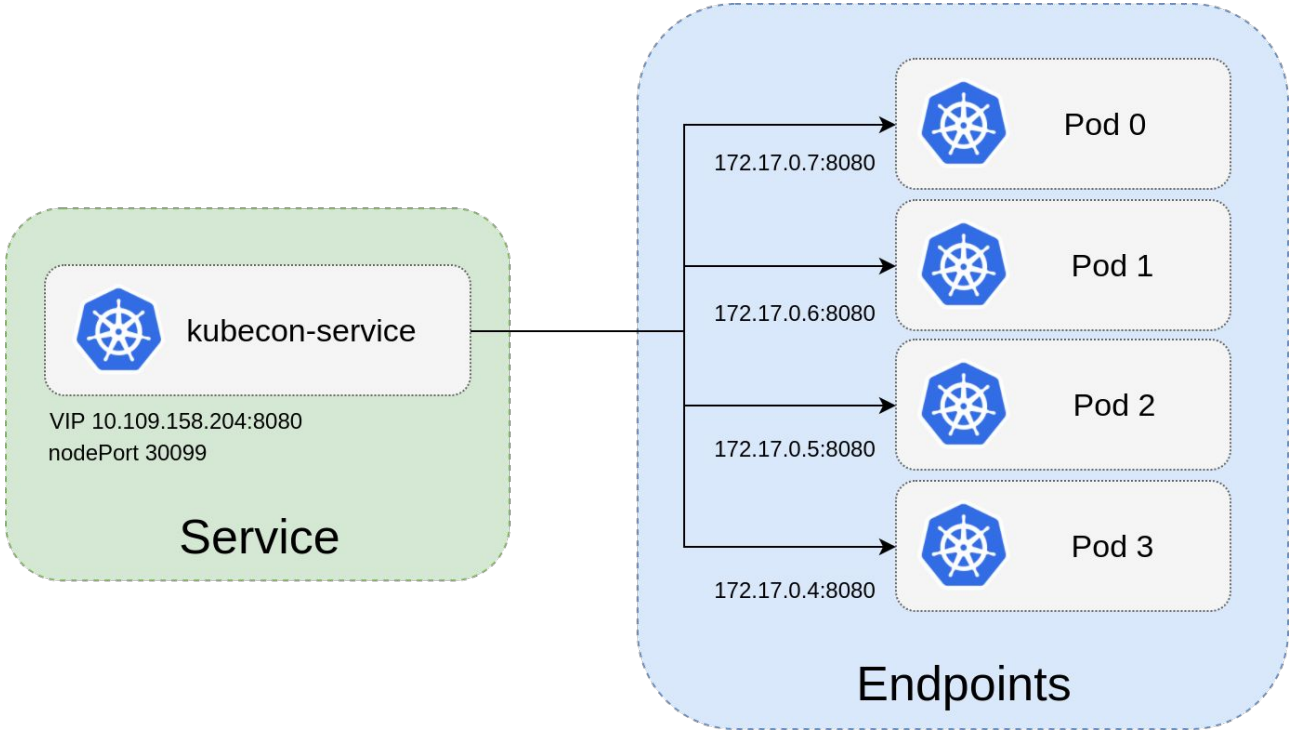
Damien Lespiau <damien@weave.works>

May 2nd 2018

What is this talk about?

- Why do we need L7 load balancing?
- Breakdown the different parts
- Various load balancing architectures
- Consistent hashing with bounded loads
- How hard can it be?

Kubernetes Service & Endpoints



Demo!

?!

```
{pod: "pod-0", "request_count": 100}
{pod: "pod-1", "request_count": 0}
{pod: "pod-2", "request_count": 0}
{pod: "pod-3", "request_count": 0}
```

What happened?

```
hey -n 100 -c 1 -q 10 $URL
```

- 1 connection to the VIP
- Connection pooling and keep-alive.
- Destination endpoint is chosen at the start of the connection
- Gets worse with HTTP/2 request multiplexing

Load Balancing

Load Balancing?

- Distribute the load (fairly)
- Affinity
- Locality
- Circuit Breaking

L4 vs L7

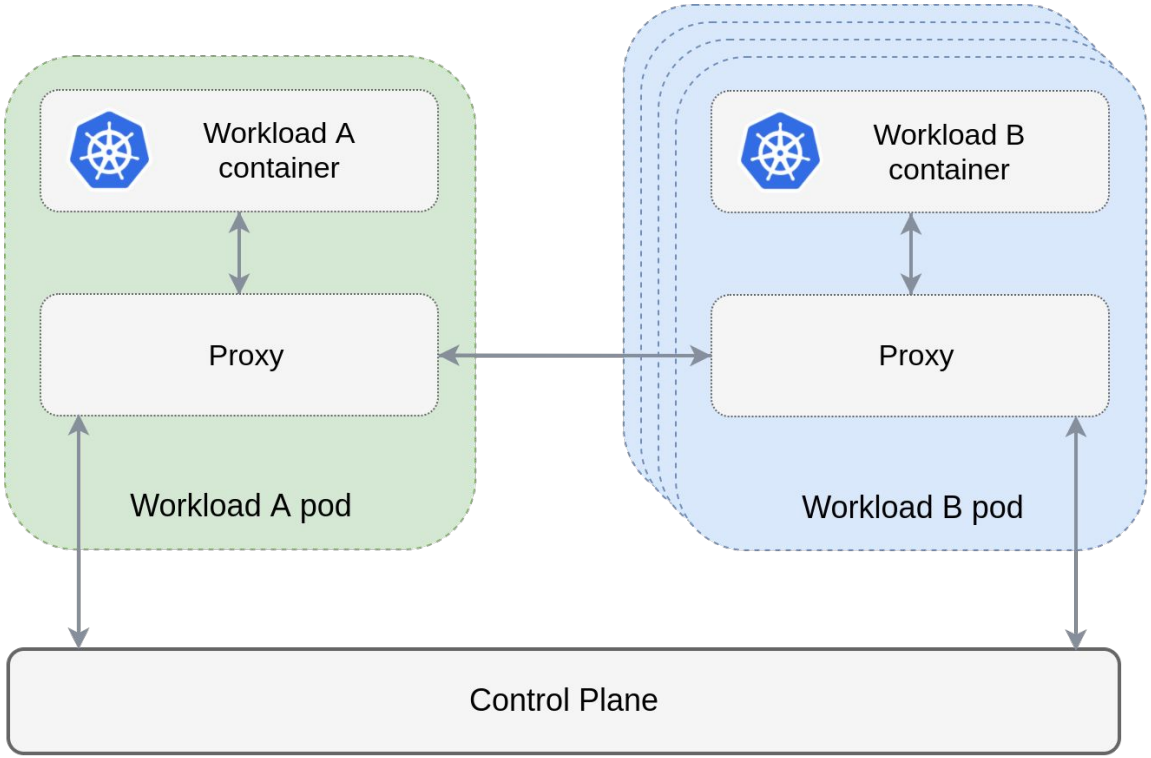
- L4 is about connections
- L4 affinity isn’t useful
- L7 is about requests
  - Better load distribution
  - Affinity
  - Passive circuit breaking

An incomplete landscape of L7 algorithms

	Random	Round Robin	Full scan, least loaded	Power of two, least loaded	Consistent hashing	Consistent hashing, bounded loads
HAProxy		✓	✓	✓	✓	✓
Linkerd			✓	✓		
Envoy	✓	✓		✓	✓	
Istio	✓	✓		✓	✓	
go-kit	✓	✓				

Architecture Boxes

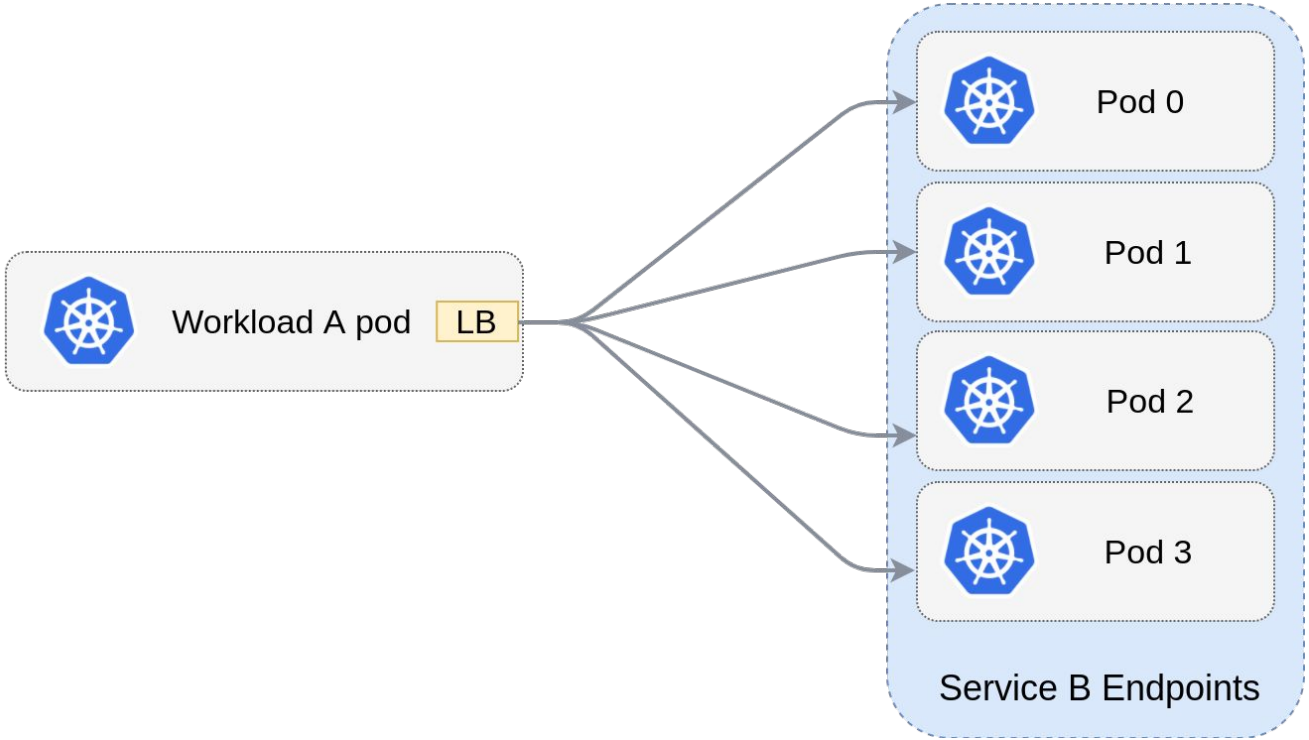
Sidecar proxy



Sidecar proxy

- Language agnostic
- Simple clients
- Proxy in the data path

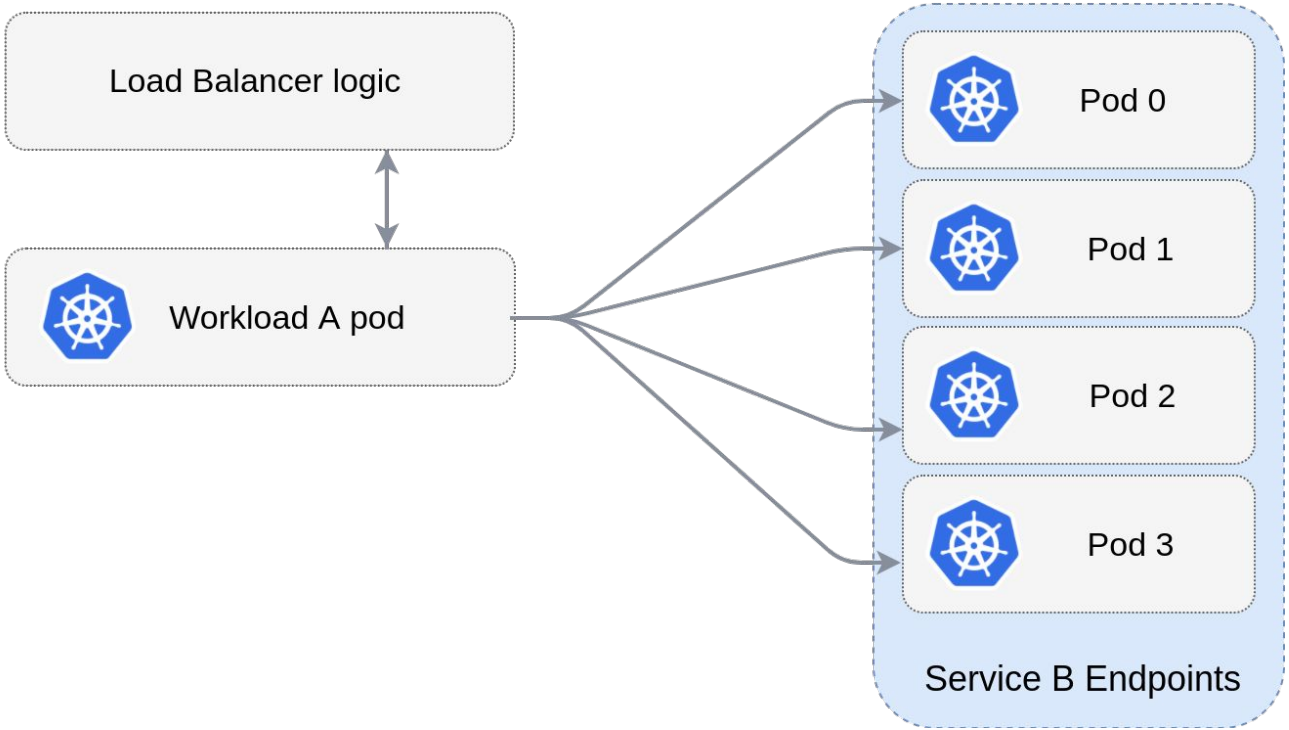
Client-side load-balancing



Client-side load-balancing

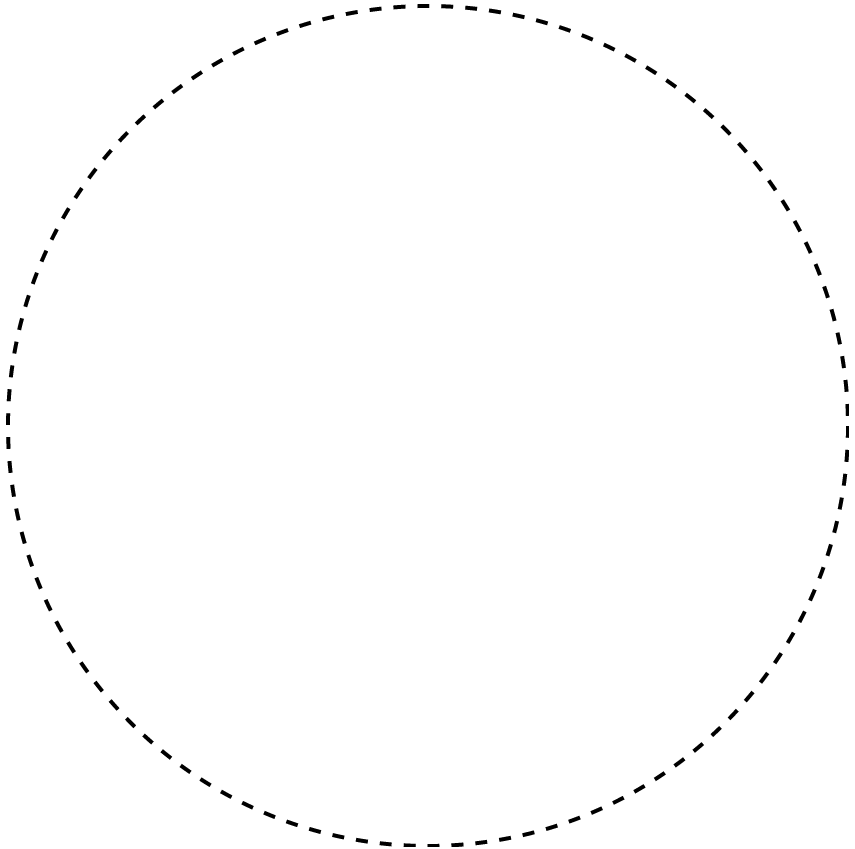
- Need library for each language
- No extra hop in the data path
- Full control over the desired behaviour

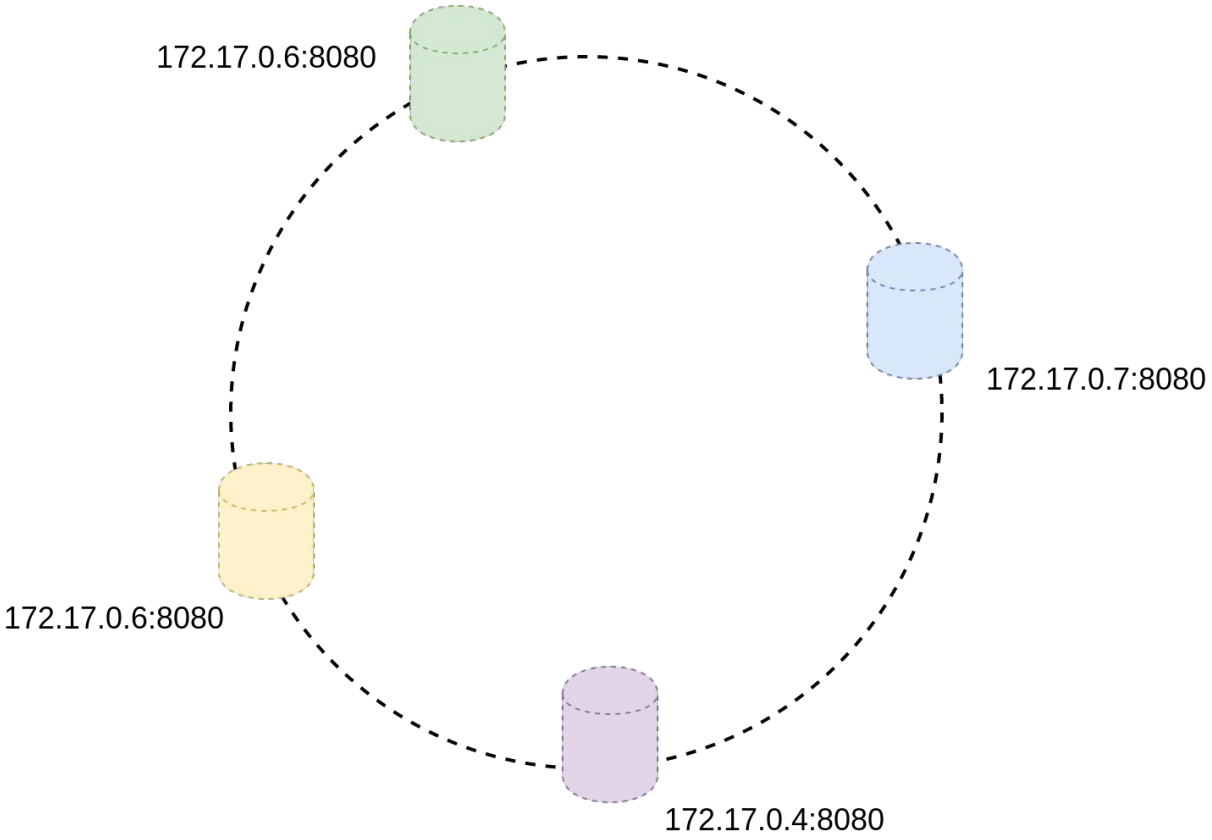
Look-aside load-balancing

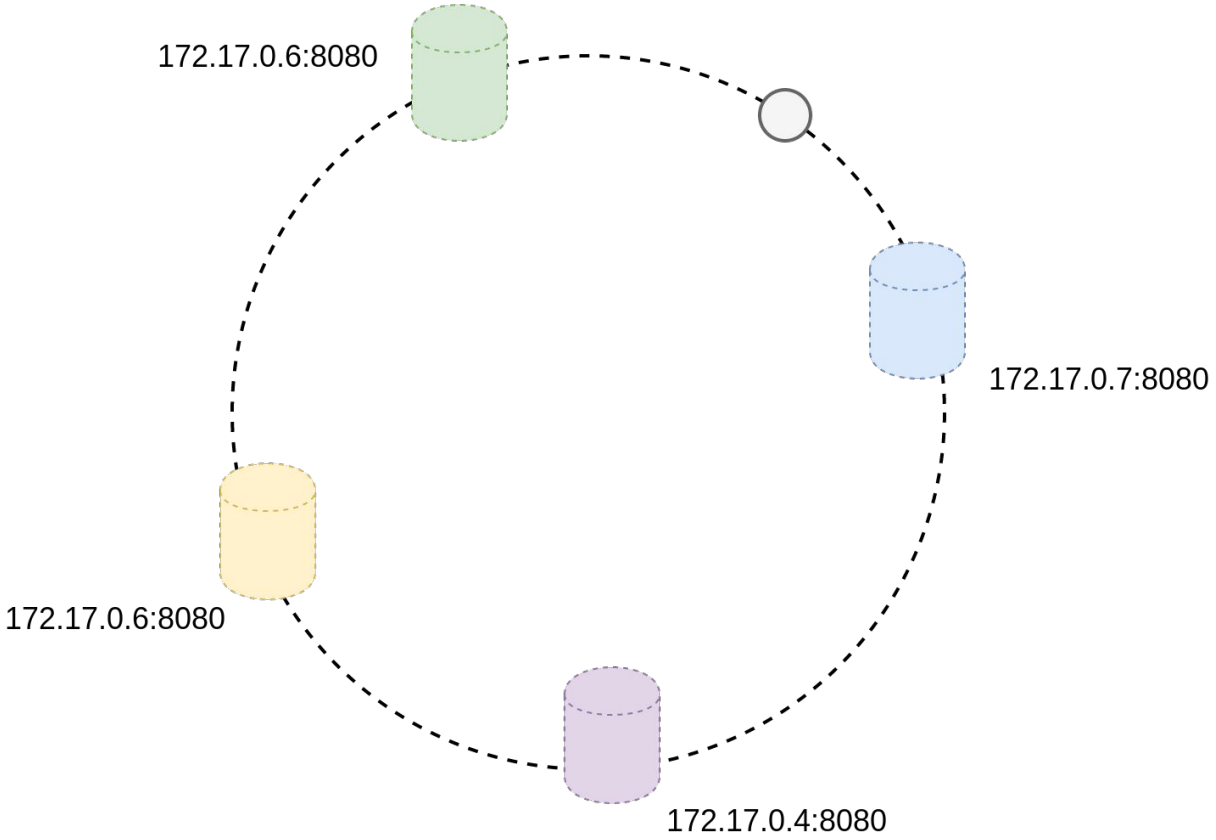


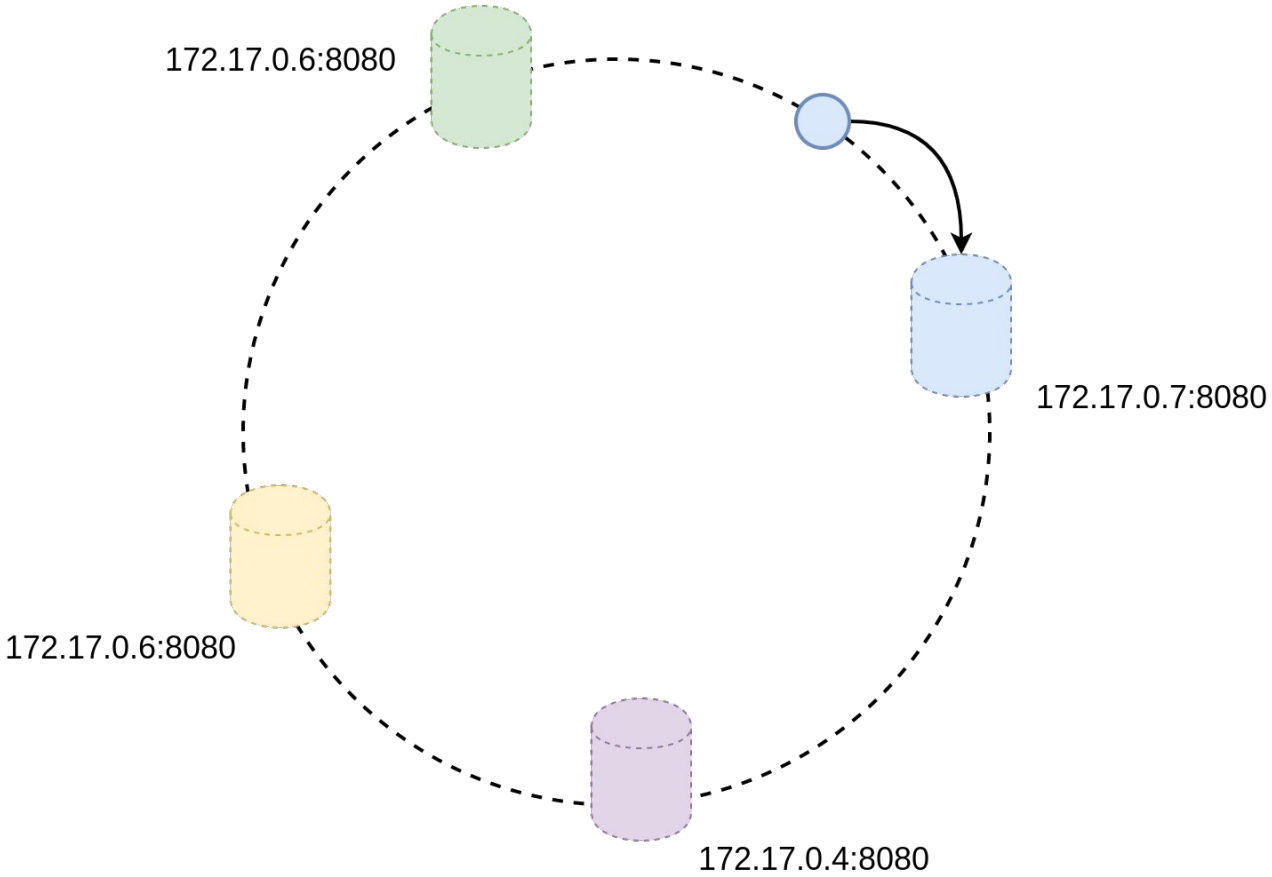
Affinity

Consistent hashing with bounded loads




172.17.0.6:8080
172.17.0.7:8080
172.17.0.6:8080
172.17.0.4:8080


172.17.0.6:8080
172.17.0.7:8080
172.17.0.6:8080
172.17.0.4:8080

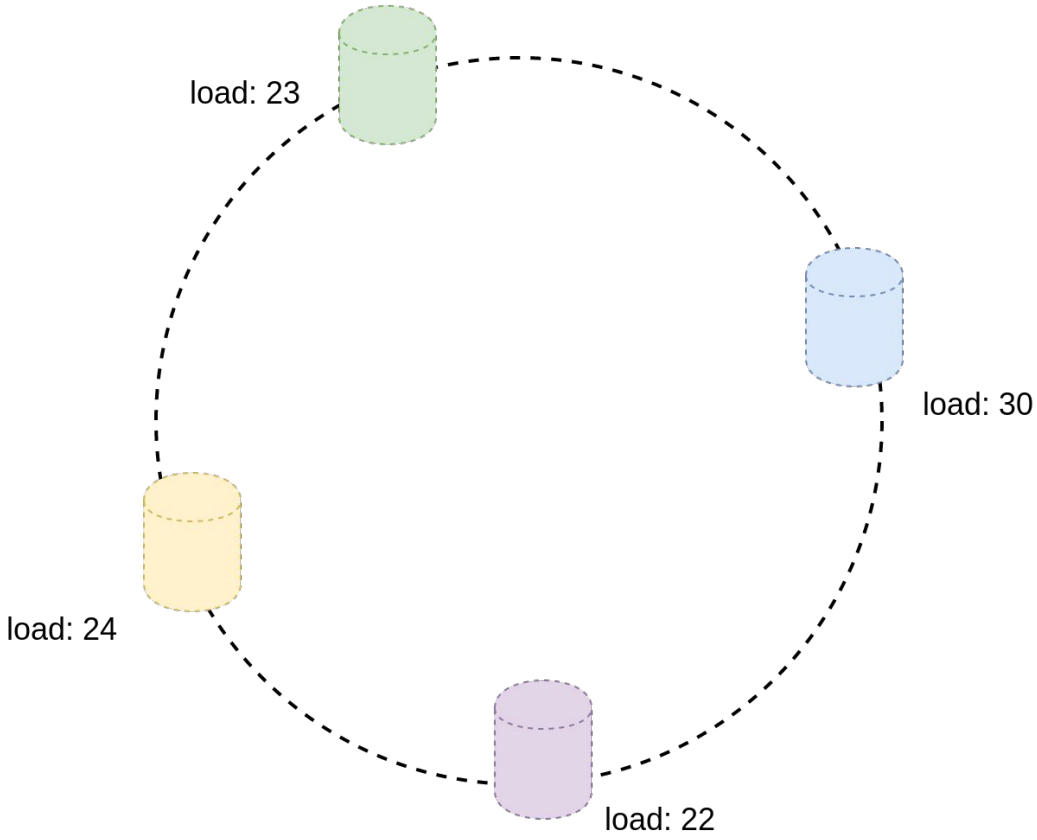

172.17.0.6:8080
172.17.0.7:8080
172.17.0.6:8080
172.17.0.4:8080

Bounded loads

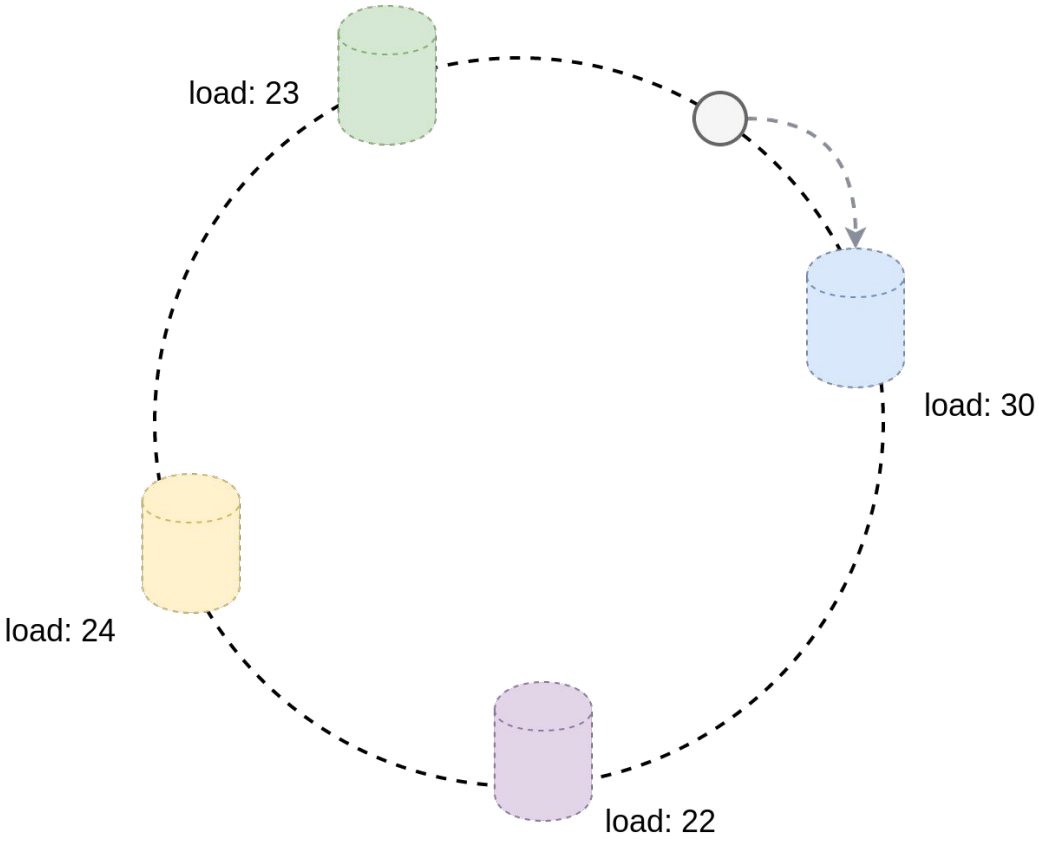
- Bound how much load an endpoint can serve compared to the average load:

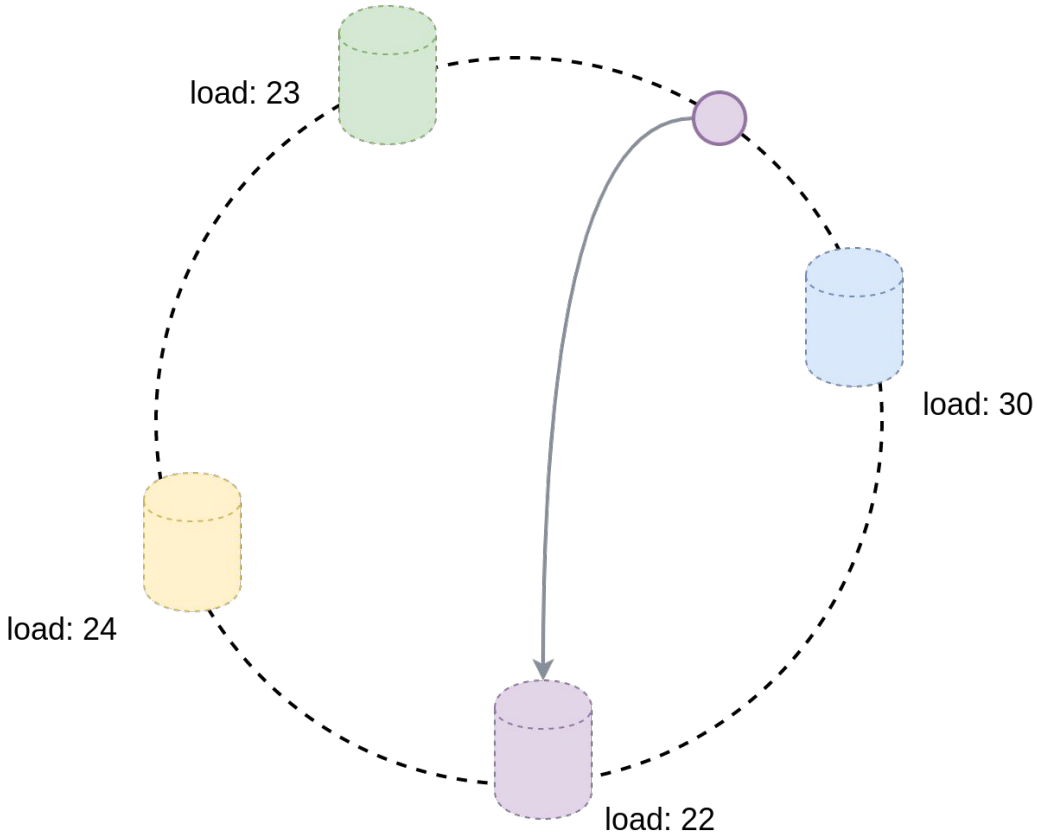
$$\mathbf{upperBound = c * averageLoad,\ c > 1}$$

$$\texttt{averageLoad = nRequests / nEndpoints}$$

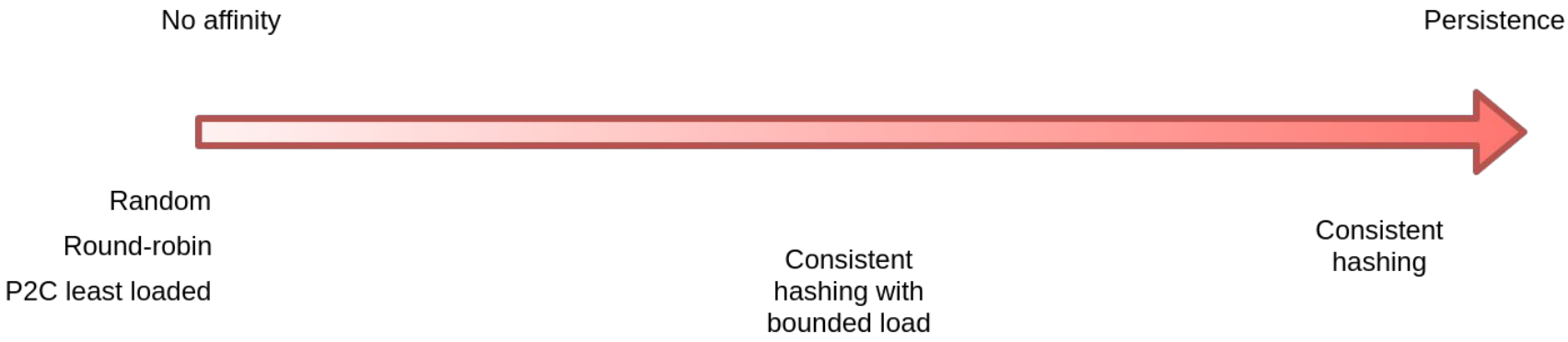

load: 23
load: 30
load: 24
load: 22

100 requests
Average load: 25
C = 1.20




load: 23
load: 30
load: 24
load: 22

The affinity scale

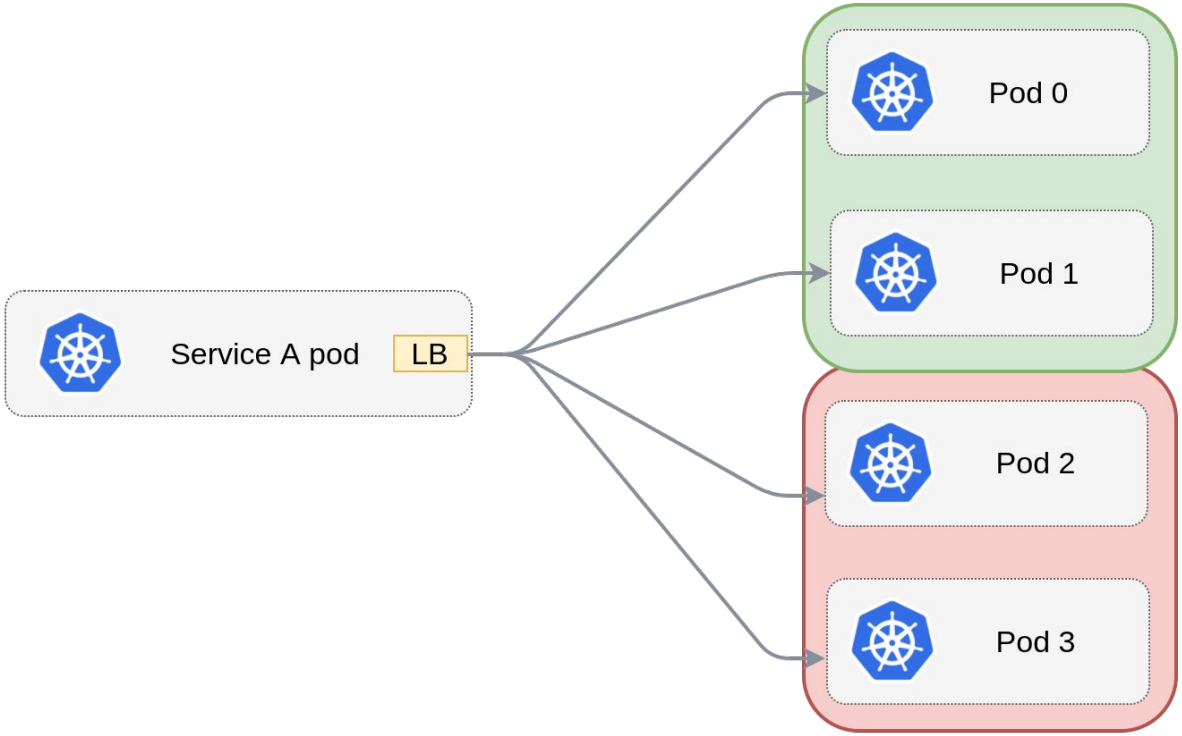


Locality

Endpoint Subsets

Endpoint Subsets

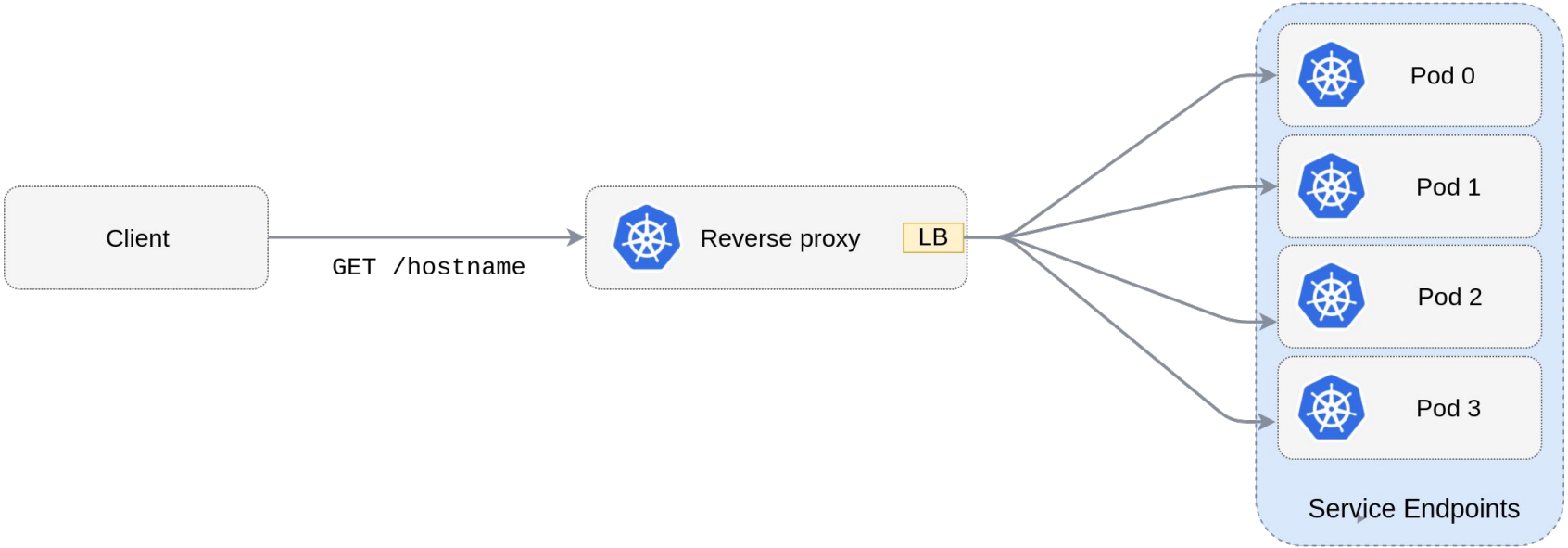
- Node-local
- AZ
- Service versions
- Prod/Dev
- ...



How hard can this be?

A toy reverse proxy

kubecon-proxy



Abstract load balancing algorithms

```
type LoadBalancer interface {
    SetEndpoints([]Endpoint)

    BeginRequest(key string) *Endpoint
    EndRequest(endpoint *Endpoint)
}
```

Watch an `Endpoints` object for changes

```
events, err := w.Client.CoreV1().Endpoints(service.Namespace).Watch(metav1.ListOptions{
    FieldSelector: fmt.Sprintf("metadata.name=%s", service.Name),
})

for {
    case event := <-events.ResultChan():
        endpoints := event.Object.(*corev1.Endpoints)
        switch event.Type {
        case watch.Added:
            fallthrough
        case watch.Modified:
                lb.SetEndpoints(members)
        case watch.Deleted:
                lb.SetEndpoints(nil)
        }
}
```

Reverse proxy with consistent hashing

```
func (p *proxy) ServeHTTP(w http.ResponseWriter, r *http.Request) {
    key := r.Header.Get("X-Affinity")

    endpoint := p.lb.BeginRequest(key)

    r.Host = endpoint.Address
    r.URL.Host = endpoint.Address
    r.URL.Scheme = "http"
    p.reverse.ServeHTTP(w, r)

    p.lb.EndRequest(endpoint)
}
```

Circuit breaking

- Relies on Kubernetes readinessProbe & livenessProbe
- Could be enhanced with passive circuit breaking

Demo!

Takeaways

Takeaways

- L7 load balancing is a necessity going forward
- Making a LB client library is relatively straightforward
- Use the Kubernetes API as a library
- Enhance Kubernetes Objects to use pod placement primitives
- Service VIP is bypassed by L7 load balancers
- What should be folded back into Kubernetes?

Demo code!

https://github.com/dlespiau/kubecon-l7lb

References

References

- Introduction to modern network load balancing and proxying
- The Power of Two Random Choices: A Survey of Techniques and Results
- SRE book - Load Balancing in the data center
- gRPC load balancing
- Consistent Hashing: Algorithmic Tradeoffs
- Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web
- Consistent Hashing with Bounded Loads
- Load Balancing is Impossible
- Predictive Load-Balancing: Unfair but Faster & more Robust

THANK YOU!

Questions, comments?

@__damien__

dlespiau

Kubernetes Services - iptables

```
-A PREROUTING -j KUBE-SERVICES
-A OUTPUT -j KUBE-SERVICES

-A KUBE-SERVICES -d 10.110.50.143/32 -p tcp -m tcp --dport 8080 -j KUBE-SVC-6NGWRYZJS3FYB3BU
-A KUBE-NODEPORTS -p tcp -m tcp --dport 30099 -j KUBE-SVC-6NGWRYZJS3FYB3BU

-A KUBE-SVC-6NGWRYZJS3FYB3BU -m statistic --mode random --probability 0.25000000000 -j KUBE-SEP-YNYGLM66N3WPJPGI
-A KUBE-SVC-6NGWRYZJS3FYB3BU -m statistic --mode random --probability 0.33332999982 -j KUBE-SEP-IVZ6K2KLB4O5IKJG
-A KUBE-SVC-6NGWRYZJS3FYB3BU -m statistic --mode random --probability 0.50000000000 -j KUBE-SEP-WWB334OGCNSQ7C2W
-A KUBE-SVC-6NGWRYZJS3FYB3BU -j KUBE-SEP-HSFXOVDVWGKWZJWU

-A KUBE-SEP-HSFXOVDVWGKWZJWU -p tcp -m tcp -j DNAT --to-destination 172.17.0.8:8080
-A KUBE-SEP-IVZ6K2KLB4O5IKJG -p tcp -m tcp -j DNAT --to-destination 172.17.0.6:8080
-A KUBE-SEP-WWB334OGCNSQ7C2W -p tcp -m tcp -j DNAT --to-destination 172.17.0.7:8080
-A KUBE-SEP-YNYGLM66N3WPJPGI -p tcp -m tcp -j DNAT --to-destination 172.17.0.3:8080
```

Moar connections

- Allowing 3 concurrent connections:

```
$ hey -n 100 -c 3 -q 10 $(minikube service --url kubecon-service)/hostname

                {"request_count":33}                    {"request_count":0}
                {"request_count":33}                    {"request_count":0}
                {"request_count":33}                    {"request_count":33}
                {"request_count":0}                     {"request_count":66}
```

- Disabling keep-alive:

```
$ hey -n 100 -c 1 -q 10 -disable-keepalive $(minikube service --url kubecon-service)/hostname

                                        {"request_count":26}
                                        {"request_count":25}
                                        {"request_count":25}
                                        {"request_count":24}
```

Consistent hashing - why?

- Want to route requests acting on the same object to the same endpoint among `n`

```
server = endpoints[hash(key) % n]
```

- Works fine when the list of endpoints is stable
- Changes the key -> endpoint mapping for most keys when endpoints are added removed
- When adding/removing an endpoint we'd like:
  - 1 / n keys to be remapped
  - Keys that don't move to still map to the same endpoint

Consistent hashing and load balancing

- Consistent hashing or rendez-vous hashing is no better than random for load balancing. Gets a lot worse if request keys aren't uniformly distributed (hot spots)
- Enters Consistent hashing with bounded loads (2016)
- Idea: ensure no host can serve more than a factor of the average load. If the hash key lands on an already too loaded endpoint, probe the next hosts in the ring until an endpoint with room is found

```
upperBound = c * averageLoad, c > 1
averageLoad = nRequests / nEndpoints
```