Acronis

#CyberFit

Demystifying Kubernetes Load Balancing

Cloud-Native & Platform Engineering Meetup

Anton Suslov, Lead DevOps Engineer

October 24th 2024

Acronis

- Team Lead for DevOps Cloud Infra team
  - Deployment framework
  - Kubernetes infrastructure
  - Maintaining internal cloud
  - Deployment of test environments
- Generally cause fewer production incidents than help solve

Anton Suslov

Lead DevOps Engineer

#CyberFit

Why should you listen to me?

- You have a self-managed Kubernetes cluster in production
- You enjoy long tirades about networking
- You want to have a deeper understanding of how load balancing works

DevOps Schools of thought

“It Just Works!”

“Why Doesn’t It Work?”

Why doesn’t it work

- You get woken up at night
- Packets are being dropped
- Nodes can’t connect to the api-server
- Something called “softirq” uses all your CPU

The journey to proper load balancing

- You are a simple man
- You have a bare-metal Kubernetes cluster
- You want to route HTTP requests, you install Ingress Controller
- You want load balancing, you install MetalLB
- Requests arrive, responses go out
- Life is good

The journey to proper load balancing

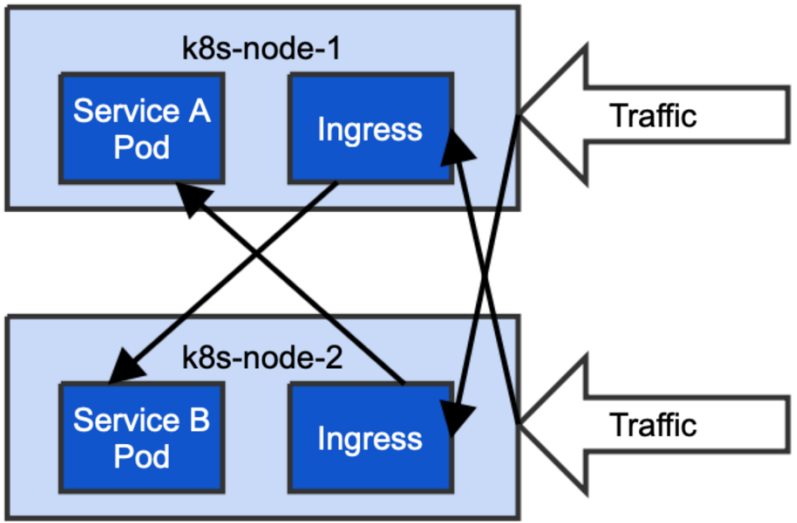
- Request volume grows
- Some of the nodes lose connectivity to k8s apiserver
- Extra traffic across the nodes due to load balancing
- Packet processing overhead (softirq) is high
- Some mitigations are in order

Understanding the costs of packet processing

- Packets are processed in Kernel interrupt context (irq/softirq)
- By default, all processing is done on CPU0
  - A huge bottleneck!
- You can deploy irqbalance to spread interrupt processing across cores
  - Each network interface is still pinned to a single core
- Use Receive Packet Steering to spread packet processing across cores
  - Single core that receives packets puts them into the processing queue
  - The queue is handled by multiple CPUs
    - Configurable via /sys/net/<ifname>/queues/rx-*/rps_cpus
  - Bottleneck is reduced

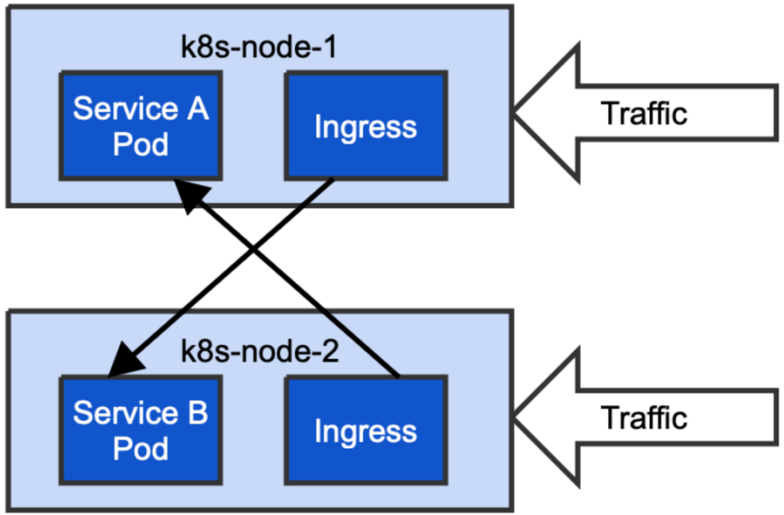
Reducing node cross-talk

- Service objects provide no locality guarantees by default
- Traffic coming to node 1 can be forwarded to a pod on node 2
- That's 2x overhead to receive and send traffic
- Kubernetes solution: externalTrafficPolicy field in Service objects
- Default behaviour is "Cluster"
- Set to "Local" to route traffic only on local Pods



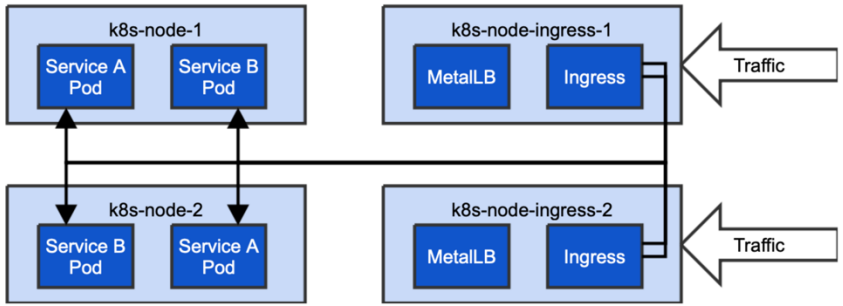
Reducing node cross-talk

- Service objects provide no locality guarantees by default
- Traffic coming to node 1 can be forwarded to a pod on node 2
- That's 2x overhead to receive and send traffic
- Kubernetes solution: externalTrafficPolicy field in Service objects
- Default behaviour is "Cluster"
- Set to "Local" to route traffic only on local Pods



Moving the Ingress load to dedicated nodes

- For balancing the traffic you are usually limited by the number of public IP addresses
- More worker nodes than public IPs
- Nodes running Ingress incur heavy softirq load
- Intermixing nodes leads to harder-to-predict load patterns and noisy neighboors
- Better to move LB and Ingress Controllers to dedicated machines



Acronis

Looks like we are done here?

#CyberFit

Oopsie-woopsie, your traffic is still unbalanced

- IP addresses are balanced equally, yet some dedicated nodes still have issues
- On no! Looks like I fell victim to DNS-based load balancing!
- Really wish I was just using the cloud…
  - How do they do it?
  - Must be some magic

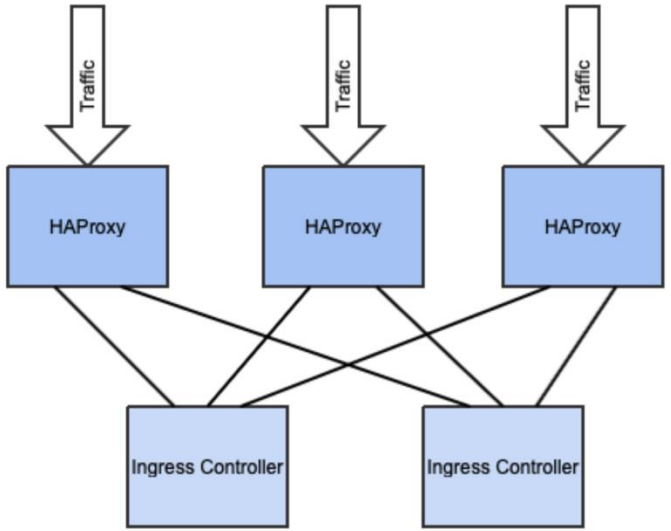
Load balancing in the clouds

- Cloud load balancers do not actually assign public IPs to your nodes
- Load balancer on the cloud side is configured via cloud-controller-manager
- The traffic is routed via nodePorts of your LoadBalancers
- Health-checks are done via healthCheckNodePort
- How do we replicate this?
  - Let's implement our own Load Balancer with HAProxy!

```
apiVersion: v1
kind: Service
metadata:
  name: load-balancer-example
spec:
  ports:
  - nodePort: 31752
    port: 80
    protocol: TCP
    targetPort: 80
  externalTrafficPolicy: Local
  healthCheckNodePort: 30779
  internalTrafficPolicy: Cluster
  selector:
    app: load-balancer-example
  type: LoadBalancer
```

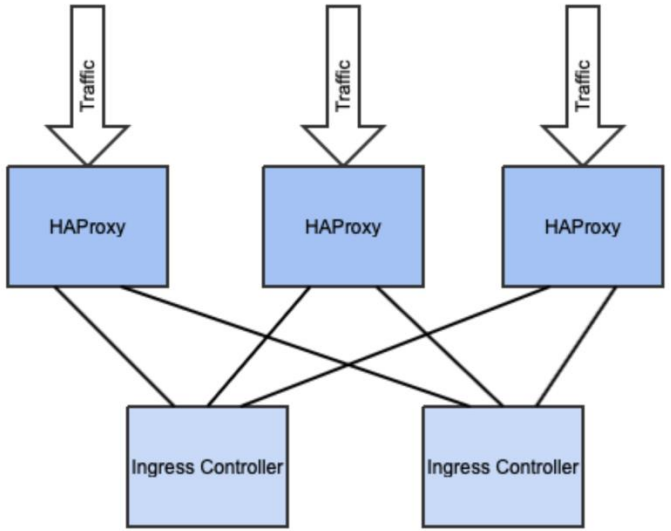
Fun with HAProxy

- Dynamic HAProxy configuration
  - A small config generator is needed to mimic cloud-controller-manager
- Implement health-checks based on healthCheckNodePort
- Public IPs are assigned by keepalived for failover purposes
- Use PROXY protocol to keep client IP address



Benefits

- Move HAProxy to dedicated nodes and mitigate the packet processing bottleneck
  - Most of the CPU will be available to just do packet processing
  - Proxy CPU use is negligible
- Ingresses get equal request load no matter the DNS strategy
- This load balancer scales from a tiny cluster to large production datacenters



What we have learned

- Load balancing is not that hard
- Processing network packets is not free
- Off-the-shelf tools help massively
- Maybe the real load balancing is the friends we found along the way

Acronis

Q&A session

#CyberFit

Acronis
Education

Grow your business with MSP Academy

Short modules.
Big impact.
Enroll today!