KubeCon | CloudNativeCon

Europe 2023

KubeCon | CloudNativeCon

Europe 2023

Autoscaling Elastic Kubernetes Infrastructure for Stateful Applications using Proxyless gRPC and Istio

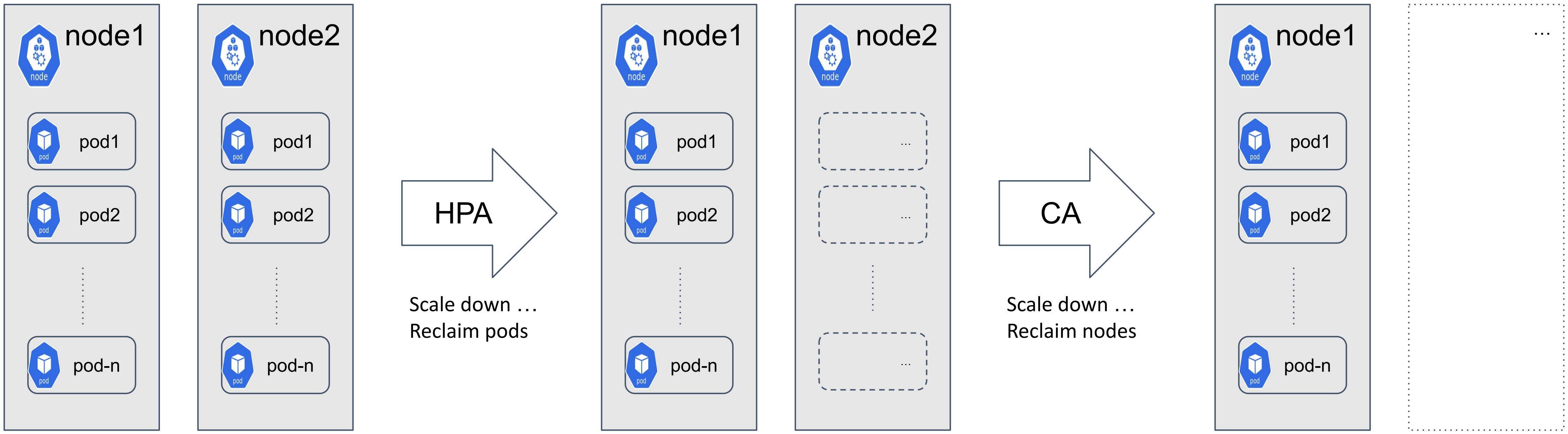
Sanjay Pujare - Google
Costin Manolache - Google

Agenda

- Kubernetes as Elastic and Autoscalable Infrastructure
- Stateful Applications and Impact of Autoscaling
- Cookie Based Stateful Session Affinity for Stateful Applications
- Why We Need Session Draining Support
- Configuring Using the New Gateway API
- Canary Deployment and Stateful Applications
- Using gRPC Observability to Verify Session Affinity (check o11y)
- Real Life Use-case (from Broadcom)
- Questions?

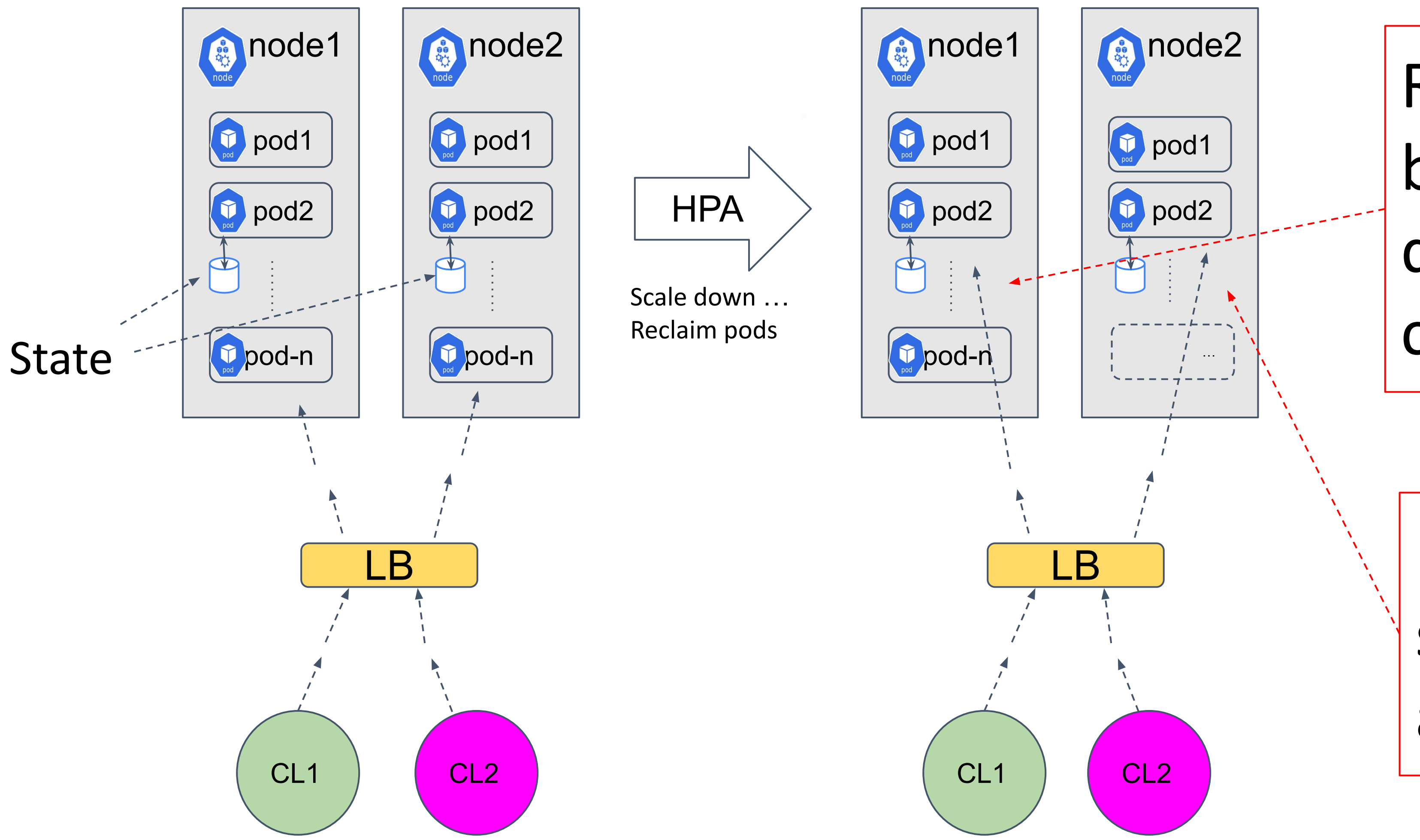
K8s as Autoscalable Elastic Infra

- K8s Popularity due to Autoscaling Features
- Capacity Management & Resource Optimization
- eg. Horizontal Pod Autoscaler (HPA) + Cluster Autoscaler (CA)



- Example of Scaling Down … to Illustrate Optimization

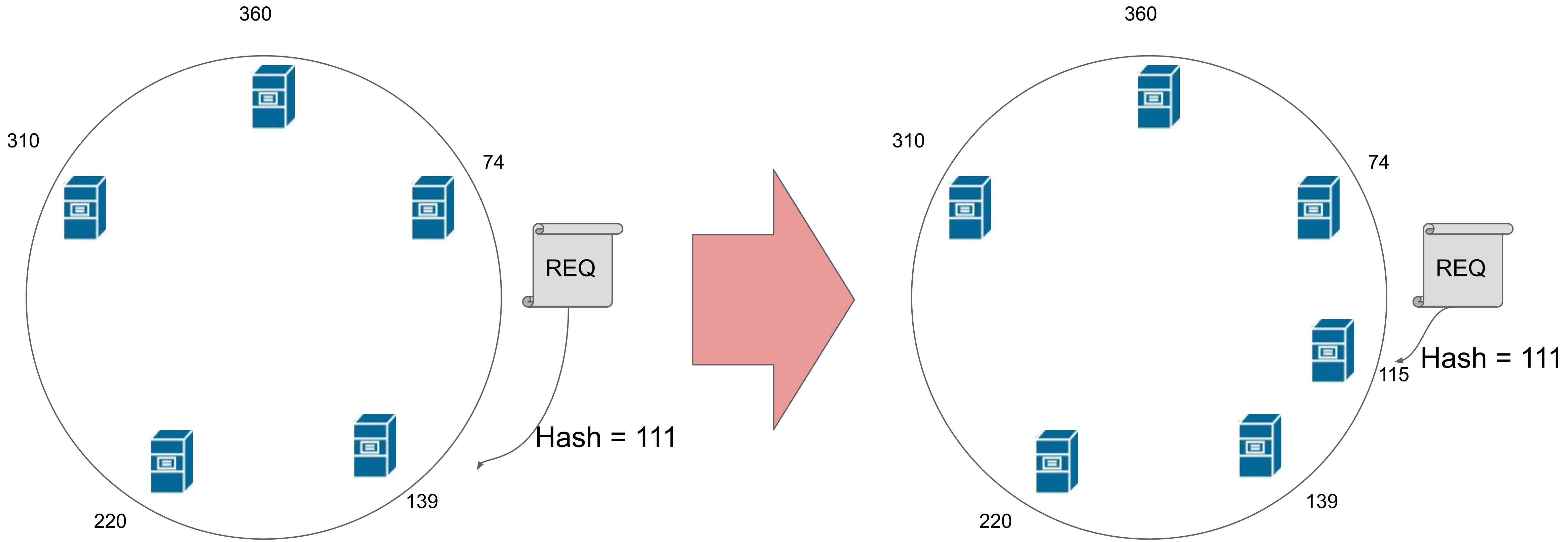
Stateful Applications & Autoscaling



RPCs to existing backends are “shifted” due to “ring-hash” ring changes

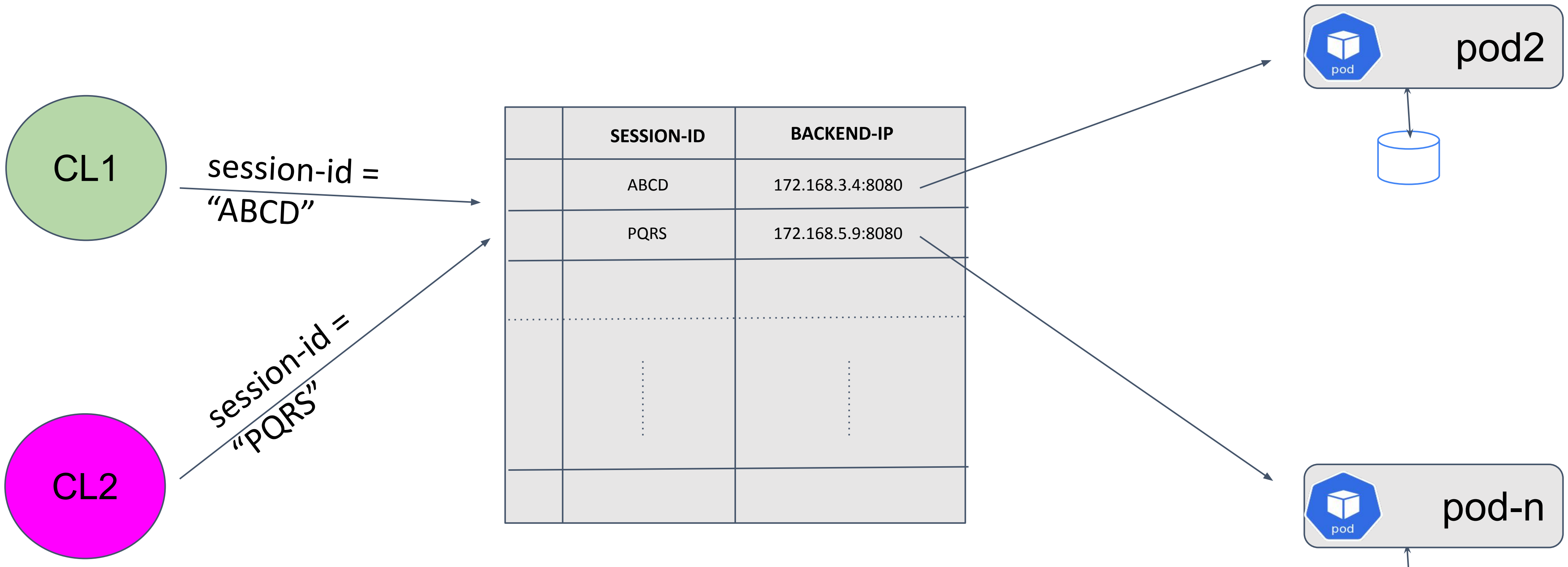
Backends with assigned sessions are removed as part of downscaling

Ring Hash Balancer Limitations



New host added: new hash mapping causes 1/Nth RPCs to be assigned to a different host where N = no. of backends

Stateful Needs Stateful Load Balancing!



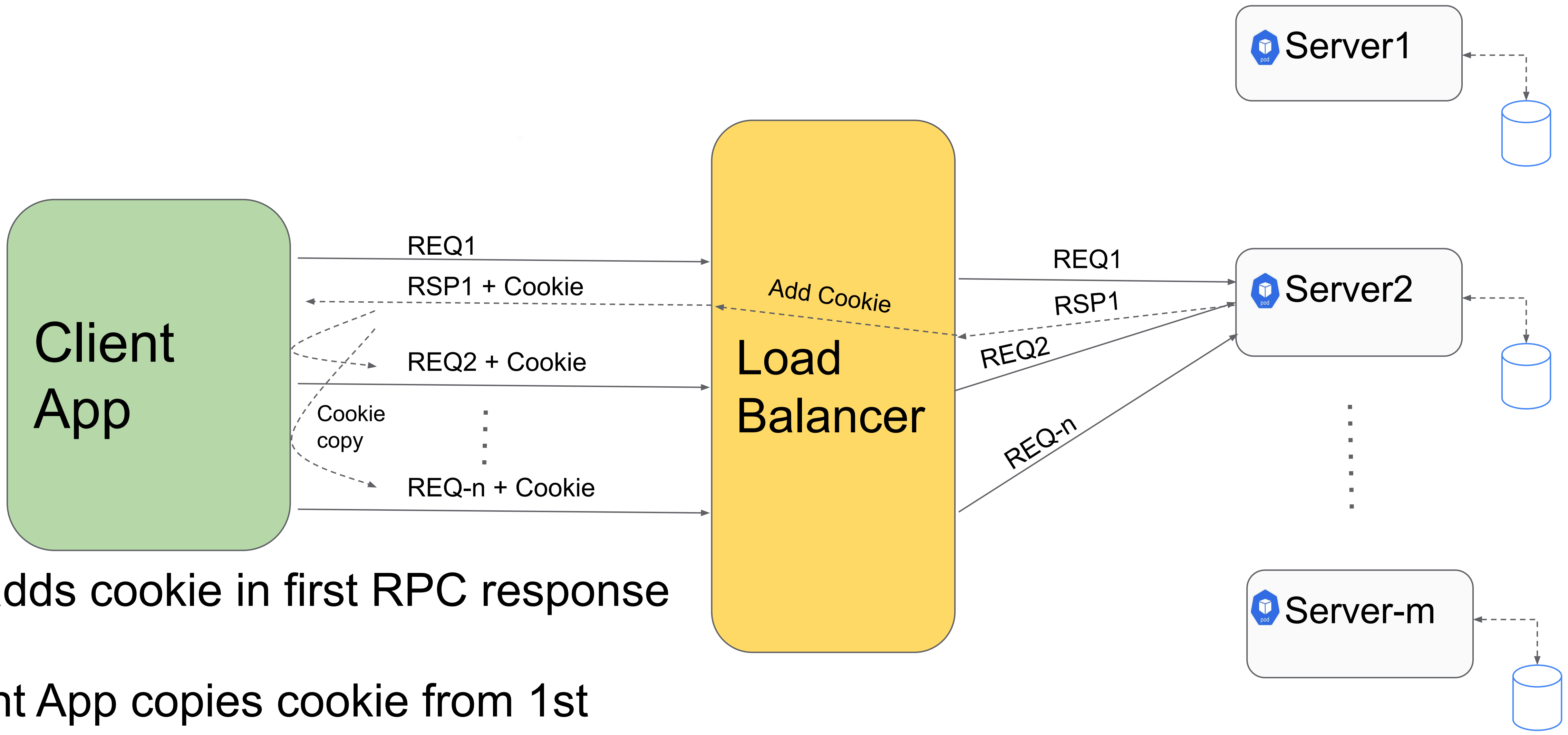
A “Hashtable” inside the Load Balancer

Session -id from the client is mapped to a backend-id or the address of the backend

Use a Cookie to Maintain State!

- Load Balancer routes first RPC in a session using some balancing algorithm
- RPC response contains “cookie” which encodes the backend that processed RPC
- Client includes “cookie” in all subsequent RPCs in the session
- Load Balancer decodes “cookie” value to get backend id and just sends RPC to that backend
- State maintained inside “cookie” which is held by client!

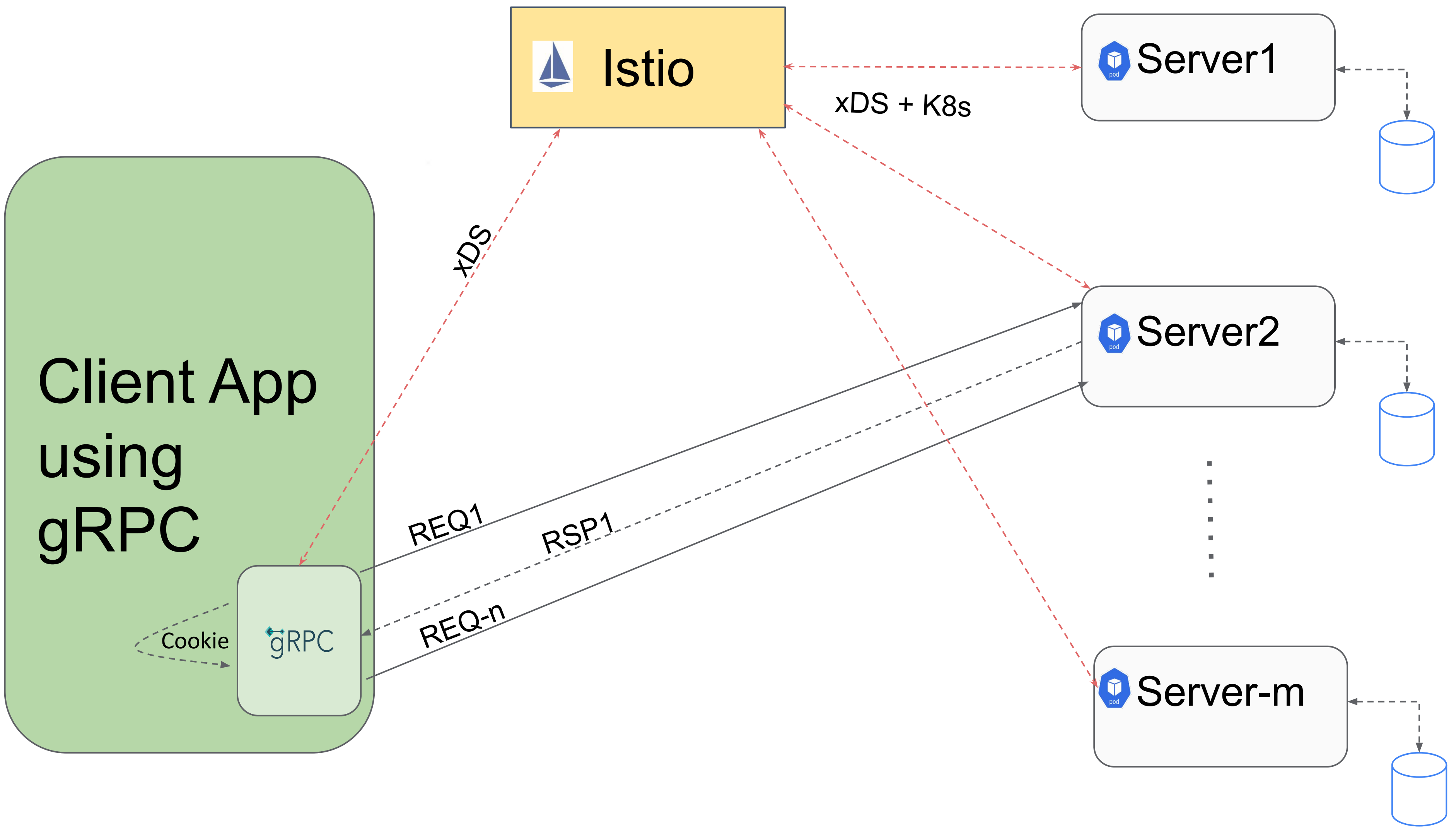
Stateful Session Affinity using Cookies



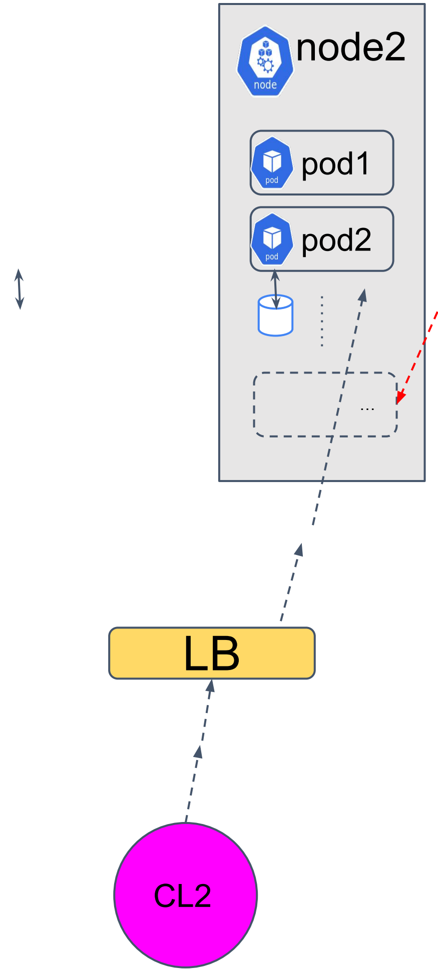
LB adds cookie in first RPC response

Client App copies cookie from 1st RPC response to all subsequent RPCs in the session

Stateful Session Affinity in gRPC



Need For Session DRAINING

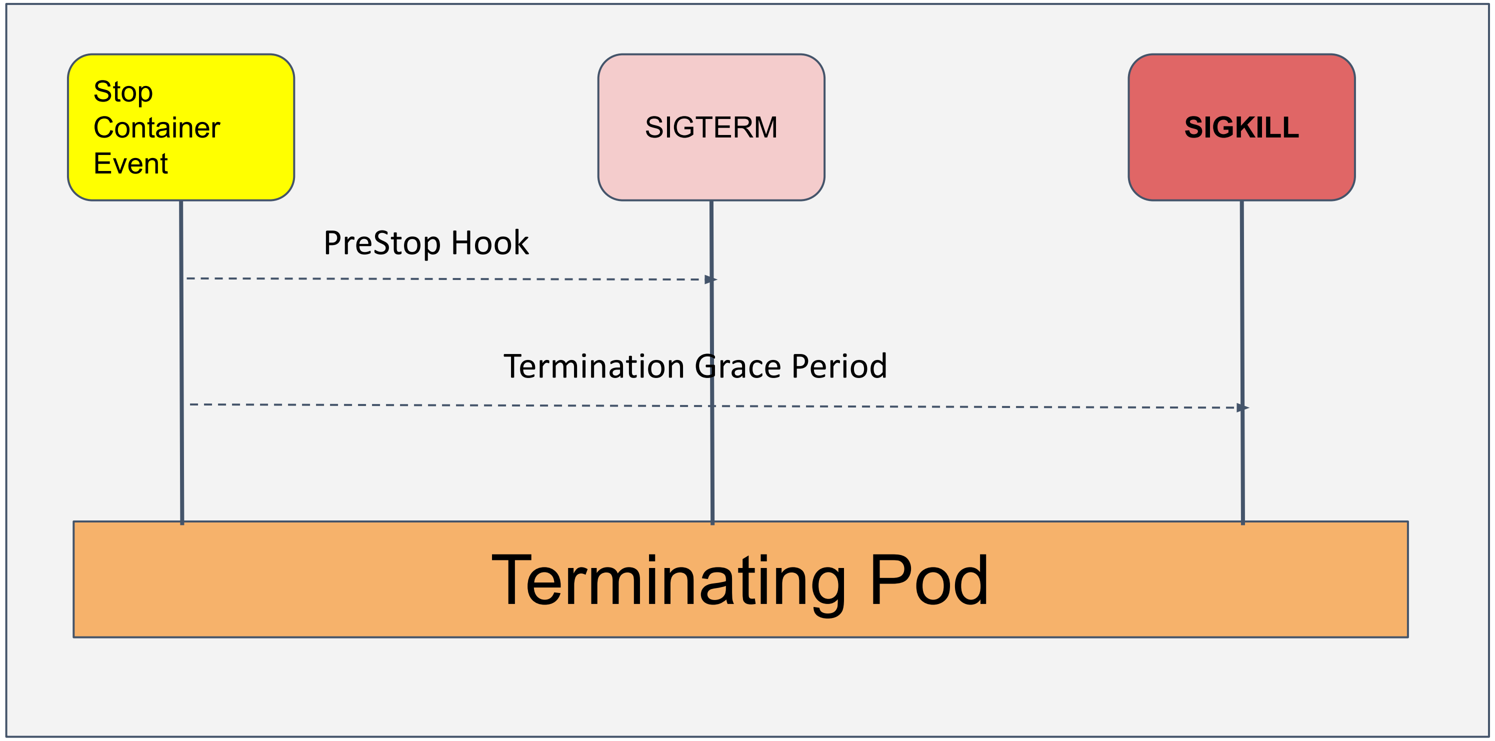


Backends with assigned sessions are removed as part of downscaling

- The other problem with downscaling!
- K8s will remove a pod even if it has assigned sessions!

- We need a special state for a pod - let’s say DRAINING - which has some special semantics
- When a pod is in DRAINING state, LB won’t assign new sessions to the pod
- But the pod will continue to receive RPCs of its assigned sessions
- But K8s should keep the pod around until all its sessions are complete!

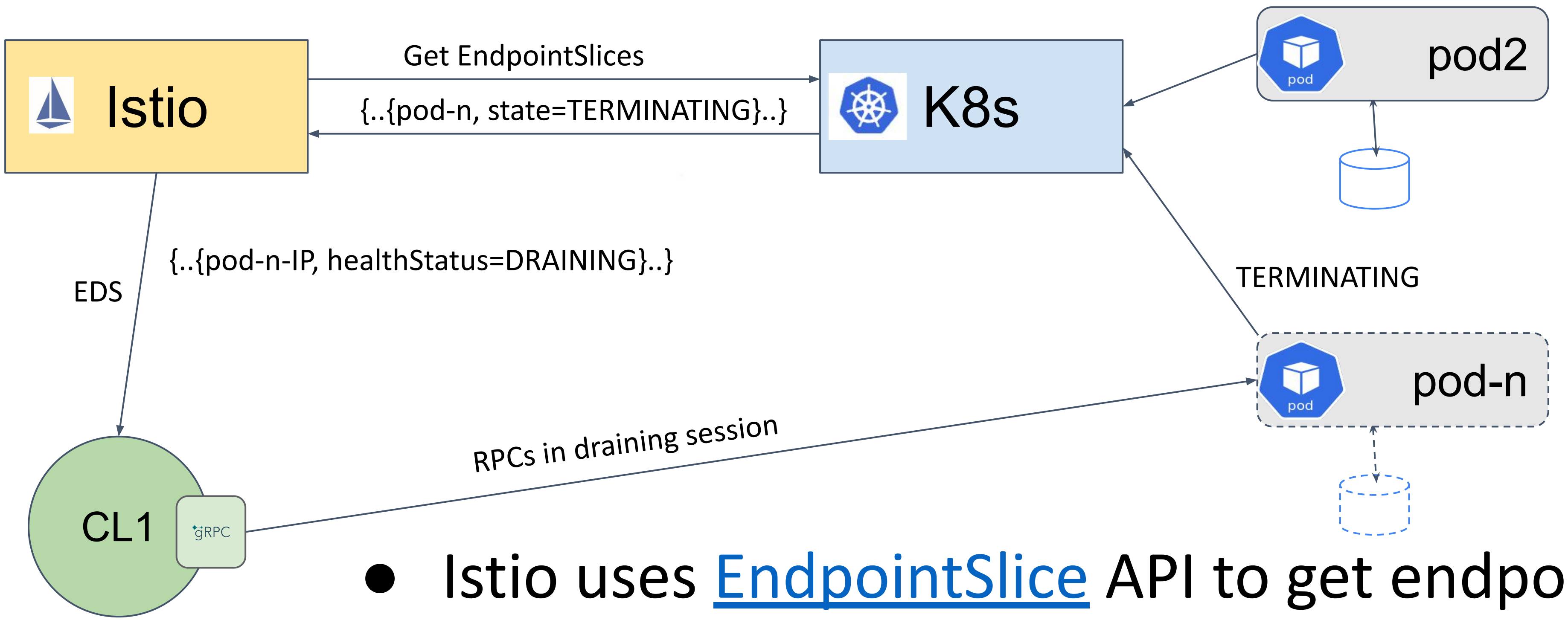
Kubernetes to the Rescue!



- Kubelet executes pod deletion driven by HPA scaledown event
- Kubelet marks the pod as TERMINATING and it calls the preStop hook
- preStop hook blocks as long as sessions are active

- Once the preStop hook returns, Kubelet sends SIGTERM
- Application sets the ***terminatingGracePeriod*** high enough for the sessions to drain

Istio Implements DRAINING State



- Istio uses [EndpointSlice]() API to get endpoints of a service
- K8s now includes endpoints corresponding to terminating pods
- Istio marks these endpoints with healthStatus=DRAINING when service has stateful session affinity enabled
- Proxyless gRPC client will send RPCs for existing sessions but will not send RPCs for new sessions i.e. RPCs without cookie to these endpoints

Canary Deployment Wrinkle!

KubeCon CloudNativeCon Europe 2023

```yaml
apiVersion: v1
kind: Service
metadata:
  name: helloworld
spec:
  ports:
    - port: 8080
      name: grpc-hw
  # No selector
---
apiVersion: v1
kind: Service
metadata:
  name: v1--helloworld
spec:
  ports:
    - port: 8080
      name: grpc-hw
  selector:
    app: helloworld
    version: v1
---
apiVersion: v1
kind: Service
metadata:
  name: v2--helloworld
spec:
  ports:
    - port: 8080
      name: grpc-hw
  selector:
    app: helloworld
    version: v2
```

```yaml
apiVersion:
gateway.networking.k8s.io/v1beta1
kind: HTTPRoute
metadata:
  name: http
spec:
  parentRefs:
    # The route applies to gRPC clients
    - kind: Service
      name: helloworld
  rules:
    # traffic splitting rule
    - backendRefs:
        - name: v1--helloworld
          port: 8080
          weight: 90
        - name: v2--helloworld
          port: 8080
          weight: 10
```

Traffic split bet v1 and v2:

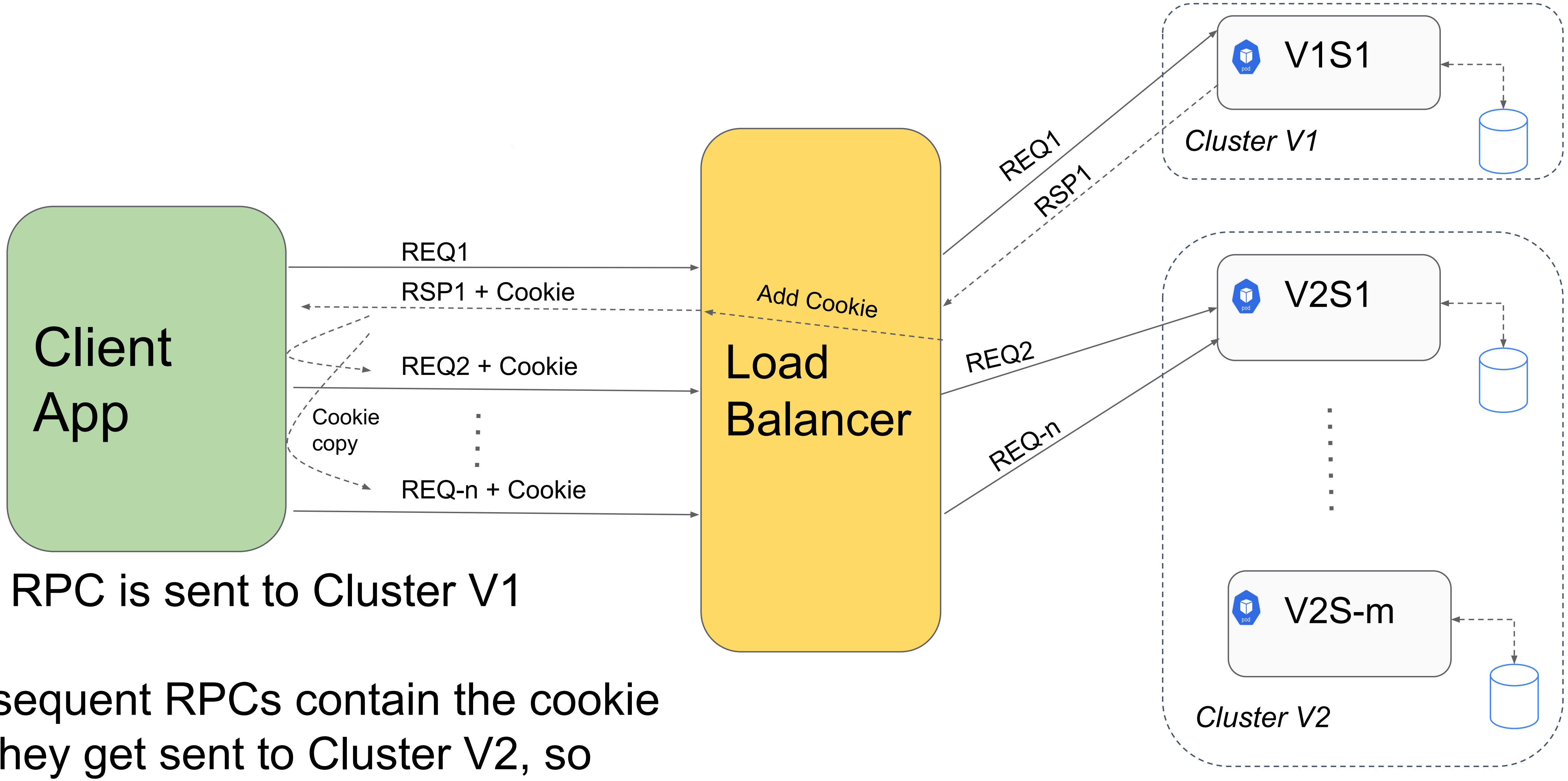
90% to v1
10% to v2

Cluster V1

Cluster V2

Istio Configuration fragment showing a canary deployment where 90% of the service traffic is sent to version1 and 10% is sent to version2

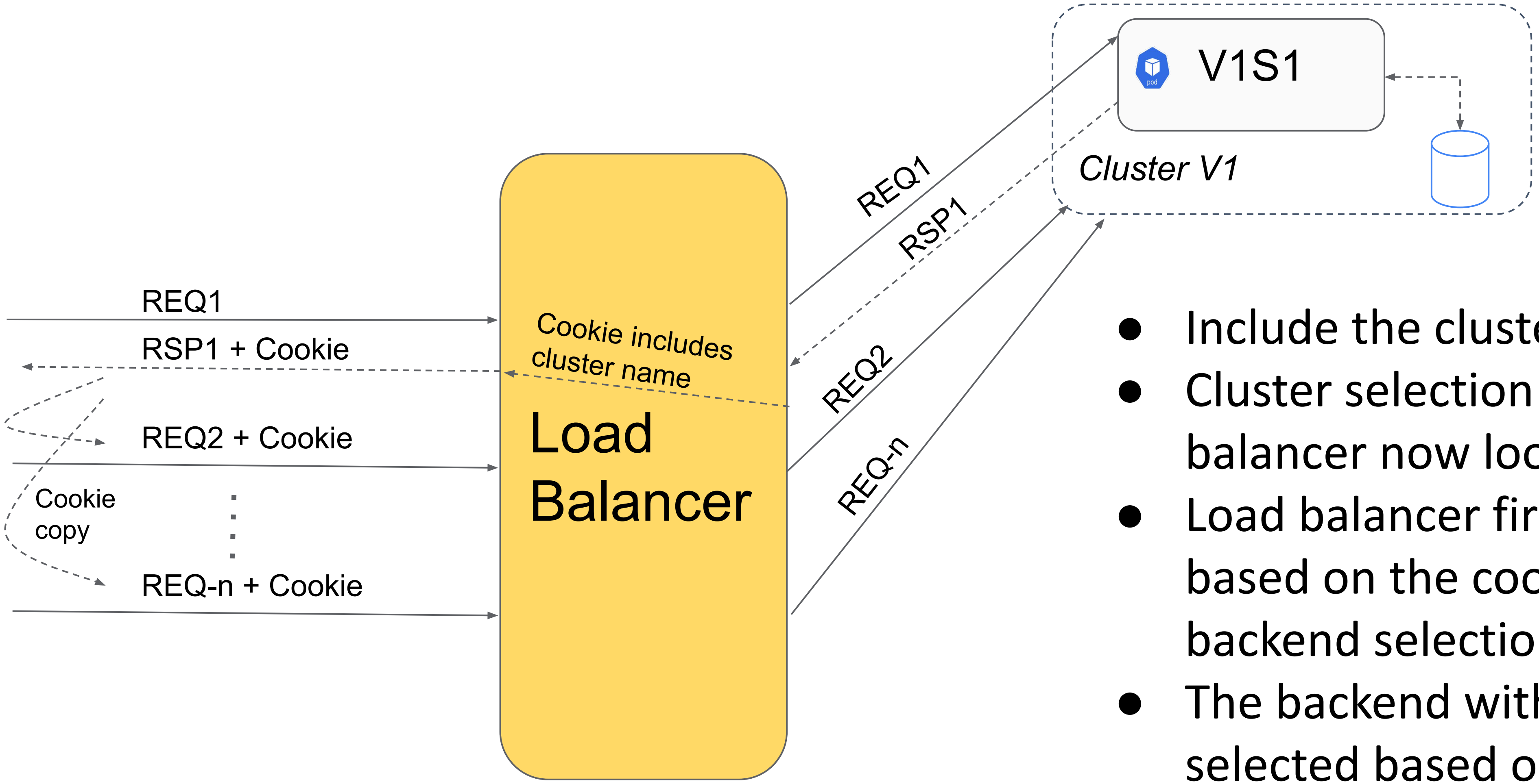
Canary Deployment Wrinkle!



First RPC is sent to Cluster V1

Subsequent RPCs contain the cookie but they get sent to Cluster V2, so session affinity is broken!

Canary Deployment Solution



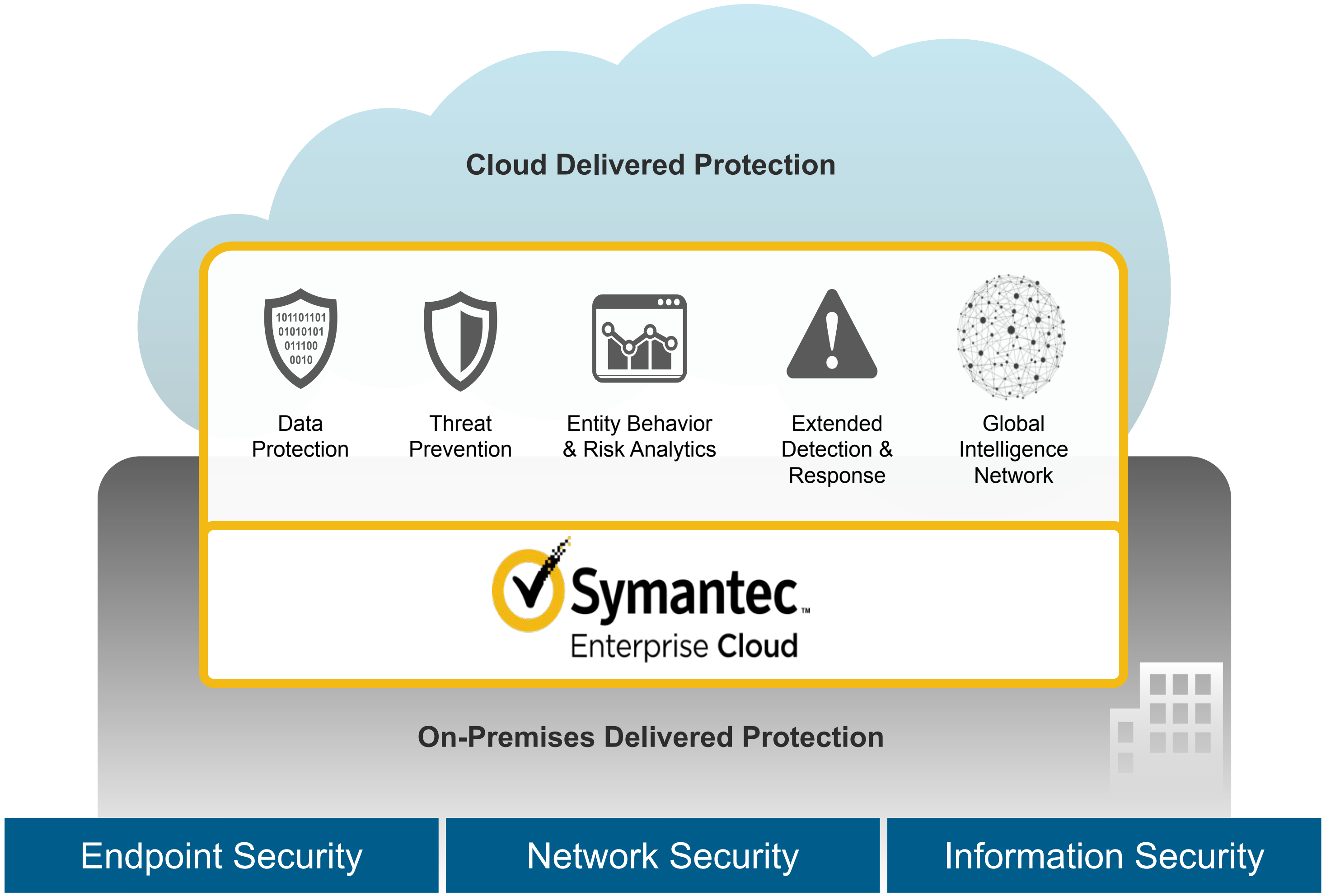
- Include the cluster name in the cookie
- Cluster selection part of the load balancer now looks at the cookie
- Load balancer first picks the cluster based on the cookie then delegates backend selection to that cluster
- The backend within the cluster is also selected based on the cookie

Cookie contents (base64 encoded but shown in plain-text here):

```
Set-Cookie: 192.168.20.6:8080;cluster:cluster-V1
```

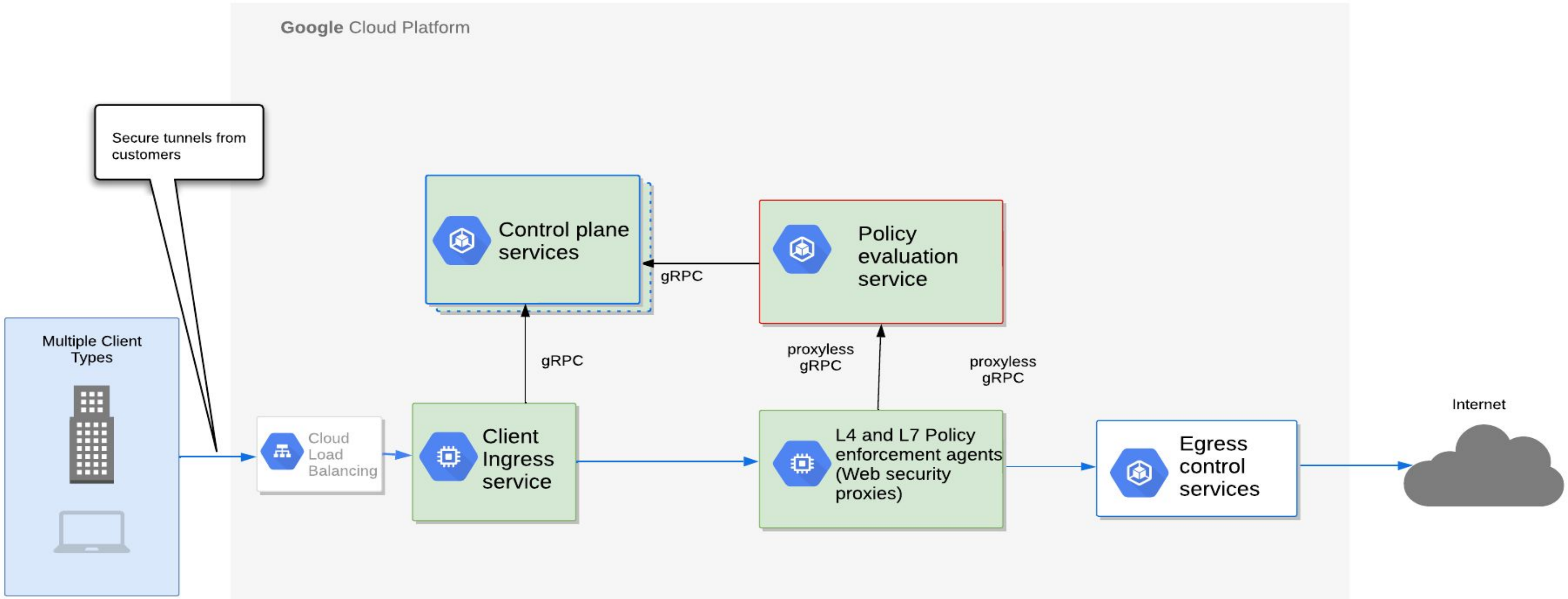
Use case:
Secure Users & Data No Matter Where They Reside

- SSE vendor - Secure Service Edge
- Complete cloud security stack
  - Proxy
  - Firewall
  - Browser isolation
  - CASB
  - ZTNA
  - Data Leakage Protection
  - Etc.



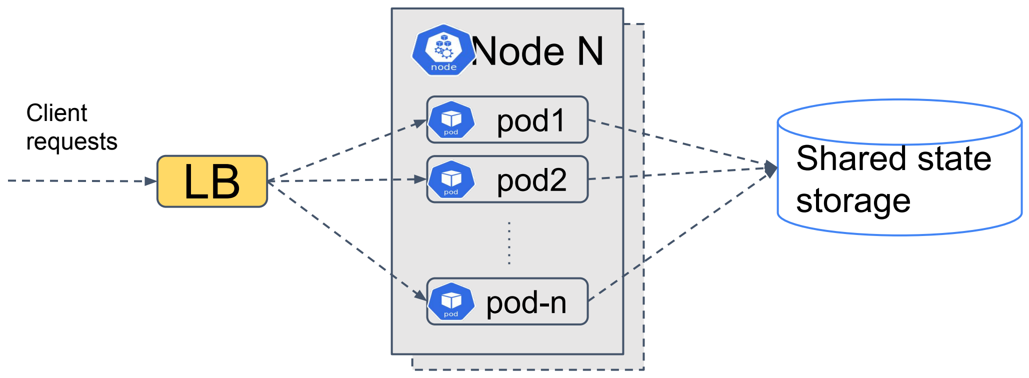
The need for Stateful Session Affinity

- Broadcom Micro-service performs WebSecurity policy evaluation on customer traffic
  - gRPC based, using Unary RPCs
  - RPC calls are issued at high frequency and are stateful
  - state spans across related RPCs that constitute a “session”



Challenges with conventional state sharing model

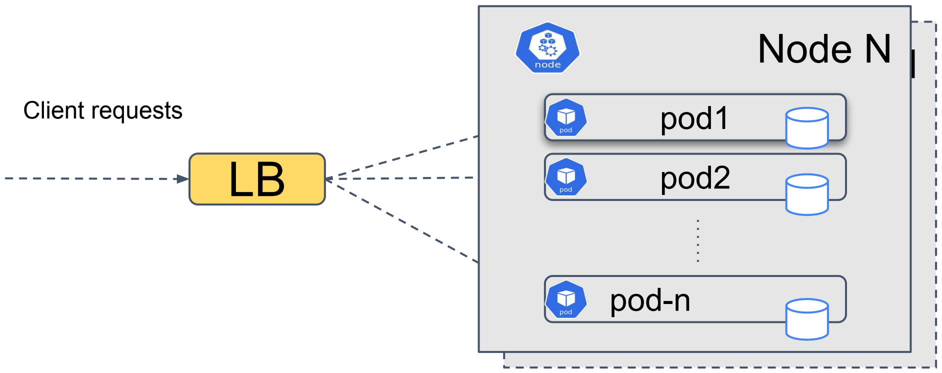
- Policy evaluation state mutates at high frequency
  - Performance-prohibitive to store off the service instance (e.g. external DB, etc)
  - Concurrency constraints on state mutations
  - Overhead of live state migration across instances is considerably high
  - State cache thrashing if requests are randomly distributed



Conventional state storage/sharing

Stateful Session Affinity enables local state-storage model

- State updates become pod-local
  - eliminate cross-pod update contentions
  - Highest efficiency possible (in-memory)
- Performance gains with better tenant cache utilization
  - No frequent flip flopping across pods



Pod-local state storage

Additional considerations for stateful services on K8s

- Load balancing requirements
  - Properly distribute requests according to backend load.
  - Coherently route sessions in a non-disruptive manner to avoid state trashing. Stateless session affinity would break sessions (Ring hash remapping problem).
  - Session-aware Canary routing. Don’t disrupt existing sessions during Canary upgrade progression stages.
- Service pods retain policy evaluation state
  - Scaling down or upgrade events are disruptive
  - Graceful draining of existing sessions. Pods should not terminate until active sessions are drained.
  - Load balancer must retain routing state for existing sessions while endpoints drain. New sessions should not be routed to draining endpoints.

Service Mesh to the rescue

- The advanced traffic routing requirements fit well within the scope of service mesh functionality
- Envoy (used in Istio sidecar deployments) added support for StatefulSessionFilter in v1.21
  - However no first-class configuration support in Istio CRDs at that time.
  - The sidecar model (Envoy based) adds significant latency and performance overheads
- Significant performance improvements when using proxyless gRPC deployments

Proxy based, 1C->3S setup

Concurrency	RPS	RPS actual	50 percentile RTT	90 percentile RTT
100	10k	8.2k	**10.66ms**	**15.72ms**
1000	3k	3k	**3.31ms**	**6.38ms**
1000	4k	4k	**6.19ms**	**13.25ms**
1000	10k	8.7k	**78.21ms**	**188.62ms**

Proxy-less, 1C->3S setup

Concurrency	RPS	RPS actual	50 percentile RTT	90 percentile RTT
1000	20k	20k	0.31ms	0.52ms
5000	20k	**20k**	**0.33ms**	**0.69ms**
1000	30k	**30k**	**0.35ms**	**0.90ms**

Current Status of the Feature

- Feature designed and implemented in gRPC: See gRFC A55
- Envoy already had the feature Stateful Session implemented
- Istio recently added support via labels on virtual service

```
labels:
    istio.io/persistent-session: my-cookie-name
```

- Google Traffic Director to add support using the new Gateway + GAMMA API (see next slide)

Why Gateway API?

- Gateway API models service networking in K8s
- Gateway API being projected as "Istio API v2"
  - Plans to make it the default API for traffic management in Istio
- GAMMA initiative within the Gateway API
  - Focus on service mesh technology and use-cases
  - New resources: MeshClass, Mesh, ServiceMeshBinding …
- Gateway API also part of managed K8s i.e. GKE through the GKE Gateway controller and for Google Traffic Director

Gateway API - Vendor Extension

```yaml
apiVersion: networking.gke.io/v1
kind: LBPolicy
metadata:
  name: payment-routing-policy
spec:
  default:
    sessionAffinity:
      type: GENERATED_STATEFUL_COOKIE
      name: "global-session-cookie"
      cookieTtlSec: 600
    connectionDraining:
      drainingTimeoutSec: 3600
  targetRef:
    group: ""
    kind: HTTPRoute
    name: payment-service-route
```

Specifies Cookie based Stateful Session Affinity policy

Specifies cookie name

Specifies cookie Time-to-live (in seconds)

Enables session draining with a timeout to specify max for draining state

Specifies that the policy is applied to a HTTPRoute (see next slide)

Gateway API & HTTPRoute/GRPCRoute

```yaml
apiVersion:
gateway.networking.k8s.io/v1beta1
kind: HTTPRoute
metadata:
  name: payment-service-route
  labels:
    gateway: payment-gw
spec:
  hostnames:
  - payment.service
  rules:
  - backendRefs:
    - name: payment-v1
      port: 50051
      weight: 90
    - name: payment-v2
      port: 50051
      weight: 10
```

Specifies target host name

Specifies weighted “clusters” where 90% traffic is sent to payment-v1 and 10% traffic is sent to payment-v2

Questions?

Questions?

Please scan the QR Code above
to leave feedback on this session