Cruise's Self-Driving Networking Journey

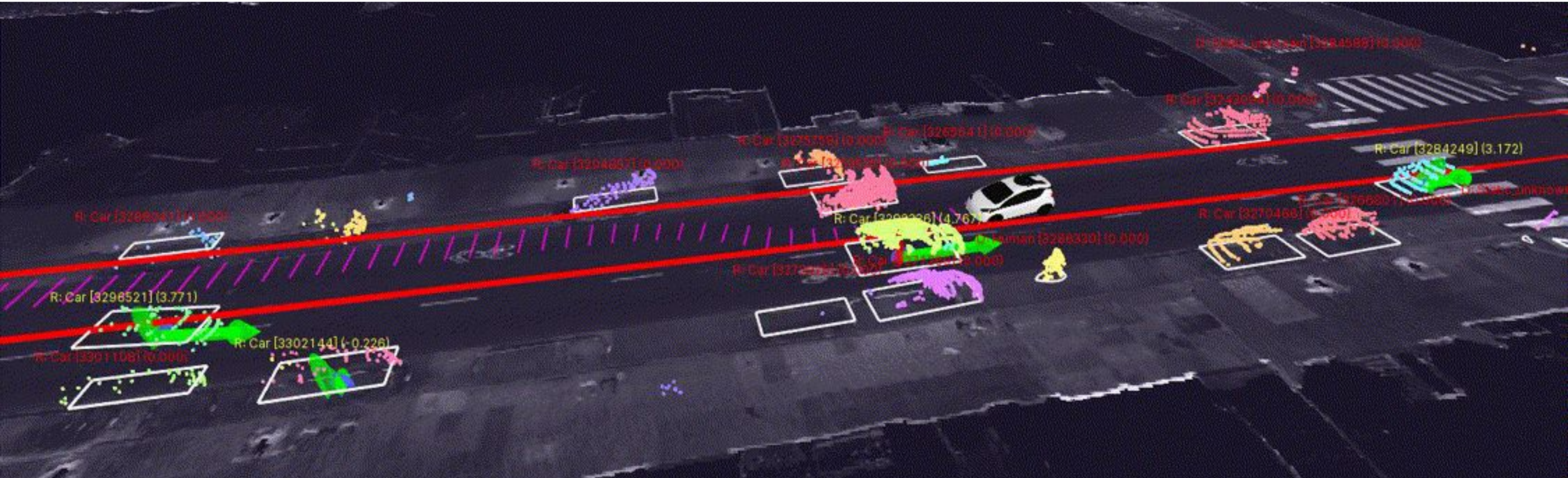
Bernard Van De Walle & Can "Jon" Yucel

@bvandewa

@canthefason

cruise

We're building the world's most advanced **self-driving vehicles** to safely connect people with the places, things, and experiences they care about.



https://getcruise.com/

https://cruise-automation.github.io/webviz/worldview/

Our Journey

Spanning multiple networks

GCP, AWS, On-prem

Multi-tenant clusters

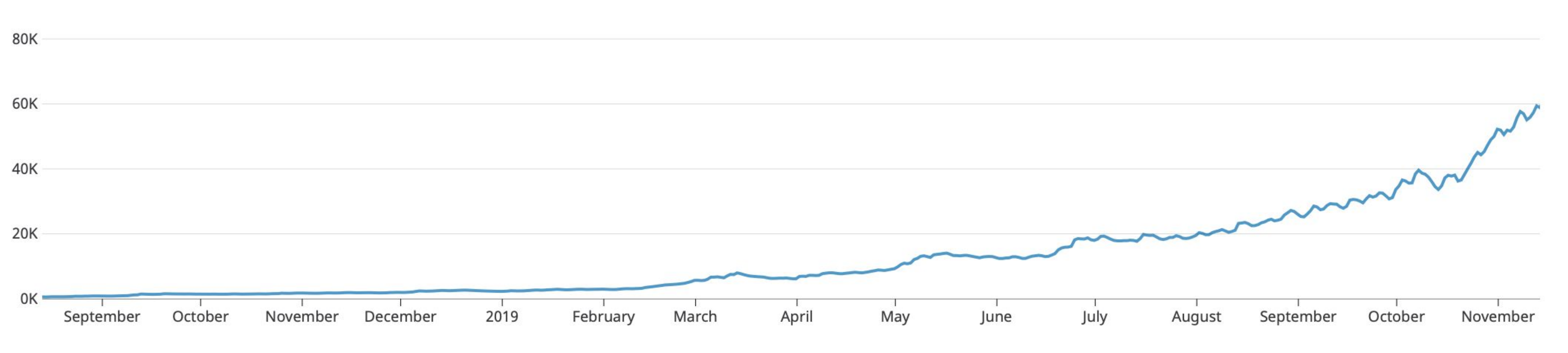
Multiple environments, multiple regions

Hybrid Cloud DNS

Route53, CoreDNS, Cloud DNS

Ingress Traffic

L7/L4 Load Balancers



Agenda

Network Connectivity

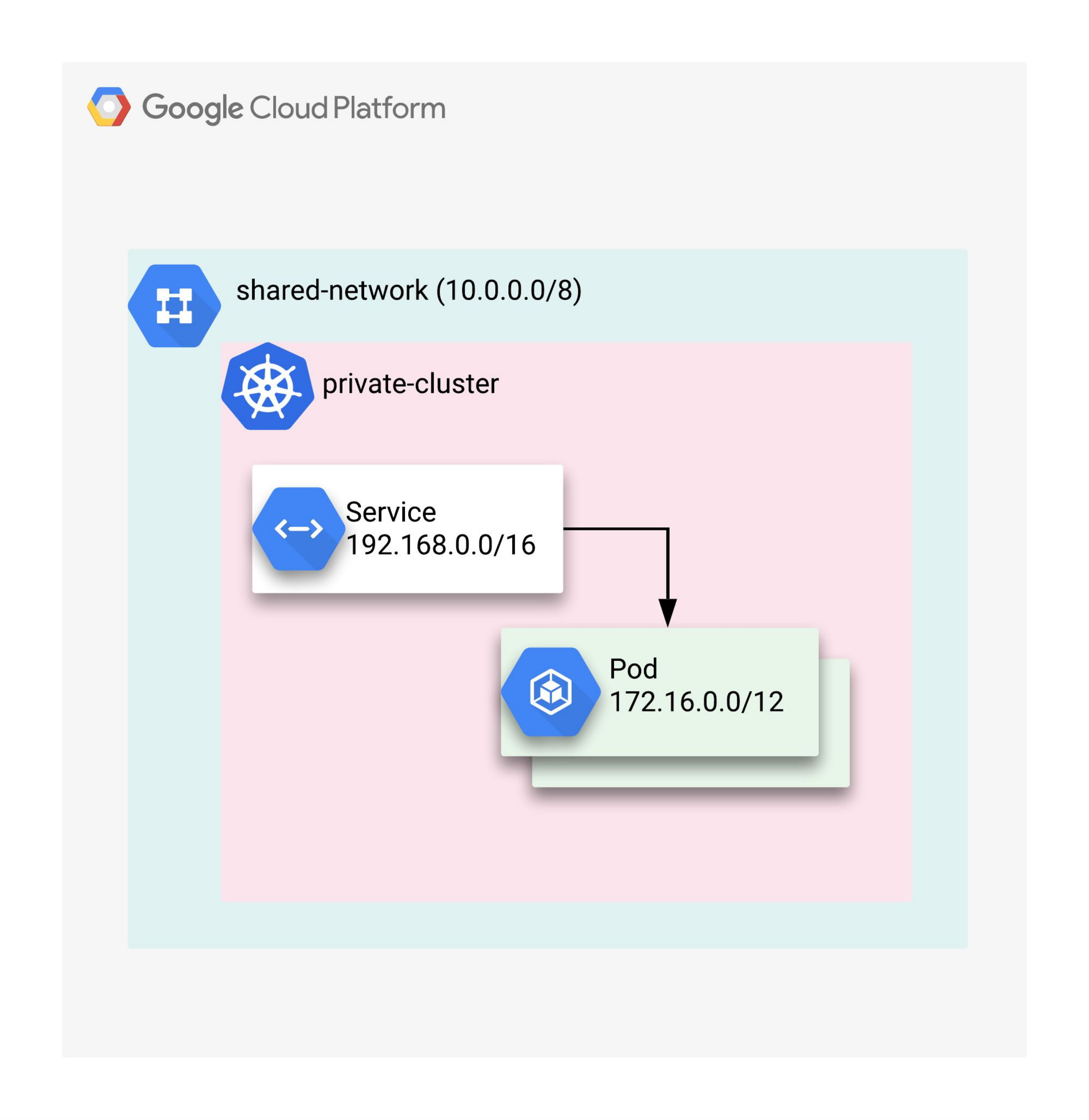
Monitoring/Logging

Security

Ingress Traffic

Hybrid DNS

cruise



Network Connectivity

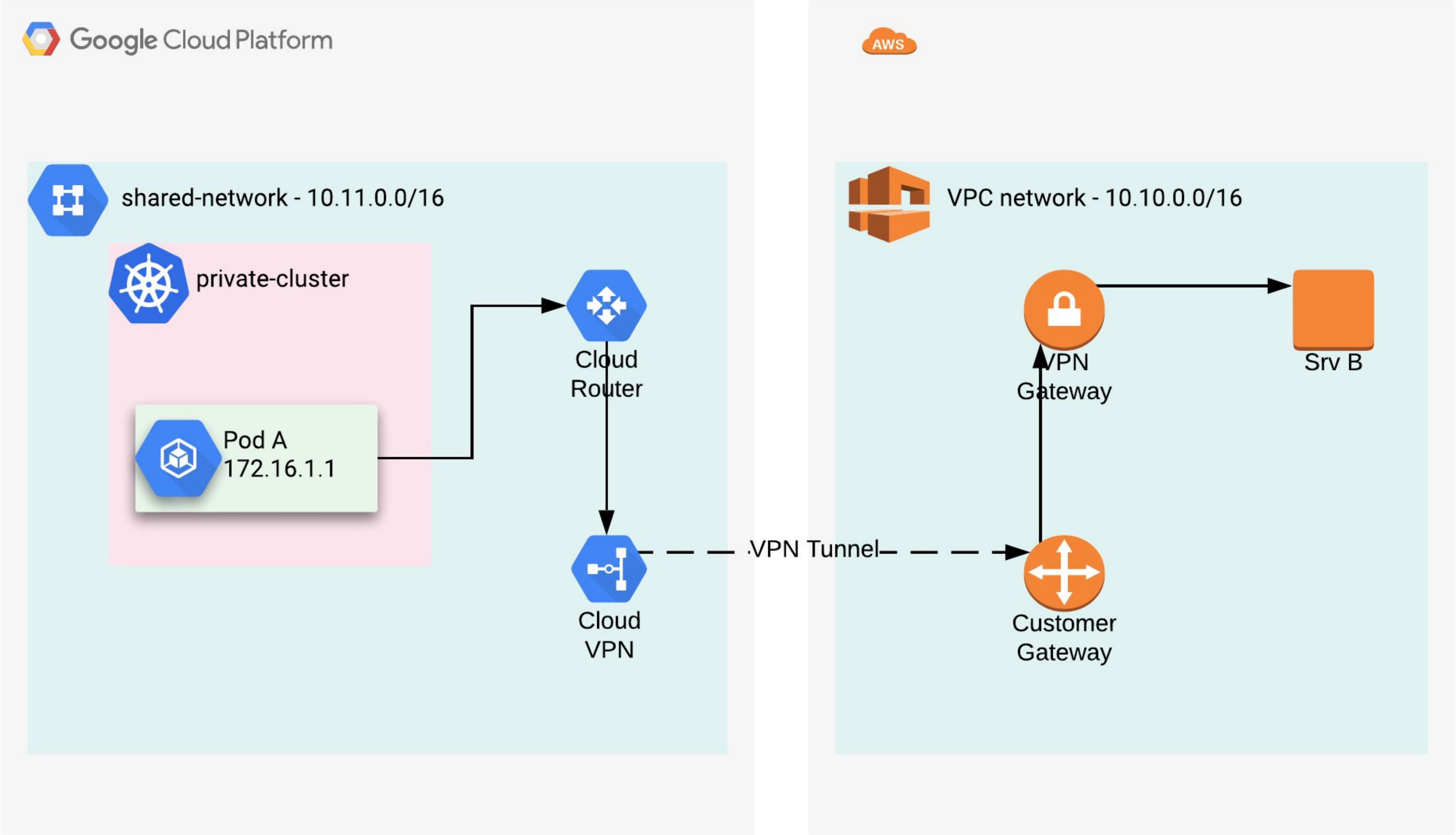
- **Goal:**
  - Isolate cluster from the public Internet
  - A cluster connected to our internal network
  - Repeatable configuration in multiple environments (dev, staging, prod)

- **IP Range Design Decisions:**
  - Class A for the nodes (10.0.0.0/8)
  - Class B for the pods (172.16.0.0/12)
  - Class C for the services (192.168.0.0/16)

Network Connectivity

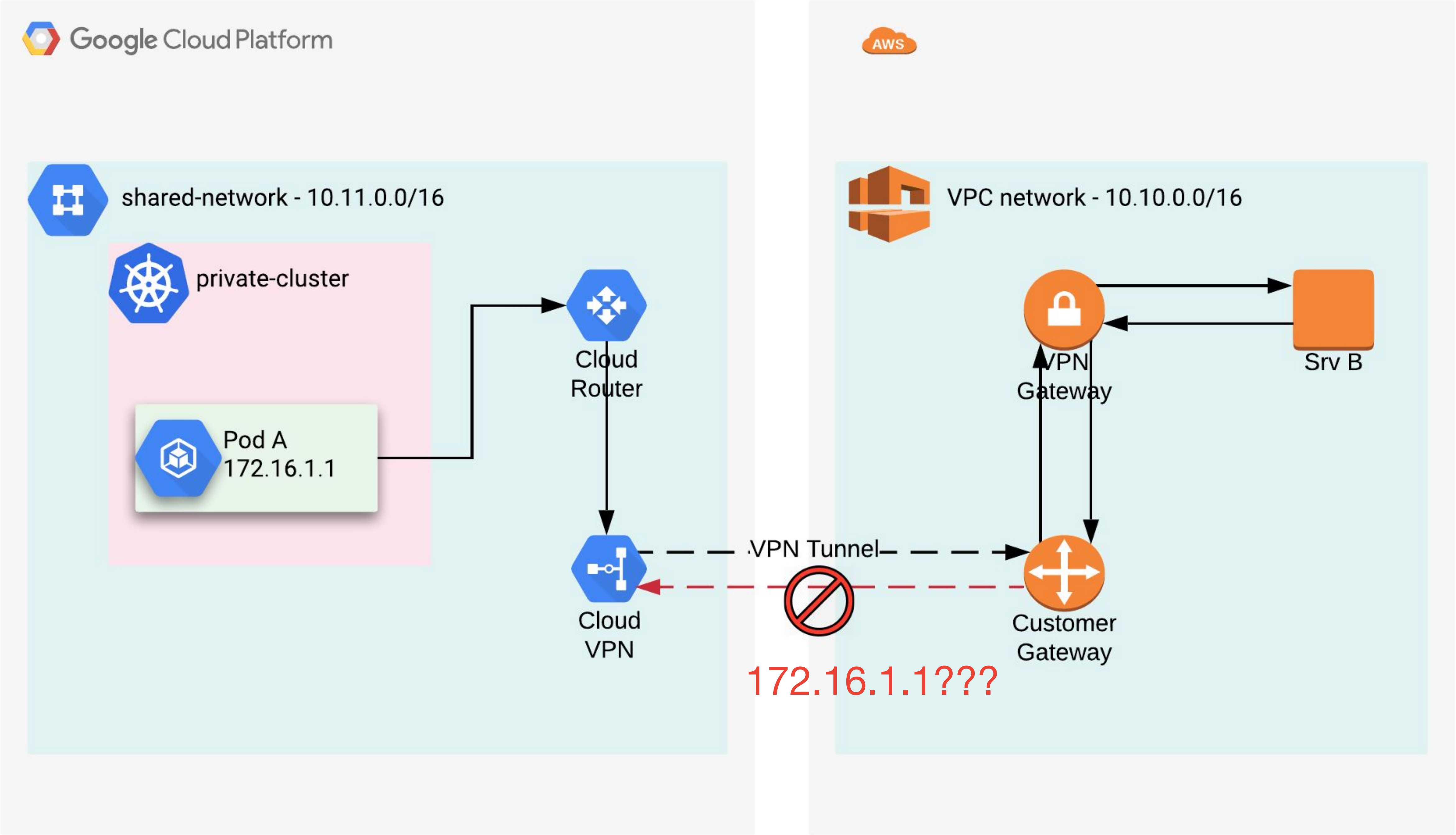
- **Goal:** Pod A can connect to Service B

- How to provide network connectivity?
  - Naive approach: VPN tunnels!



The Very First Problem!

- **Challenge:** Requests from Pod A to Srv B are all timing out!
- **Root cause**:
  - No return route for Pod IP range
  - Private GKE clusters come with IP MASQ agents
  - IP MASQ agent doesn't source NAT a request
- **Solution**: Tweak IP MASQ configuration!



cruise

Lesson learned:

Masquerade all requests outside of the network

Default Config

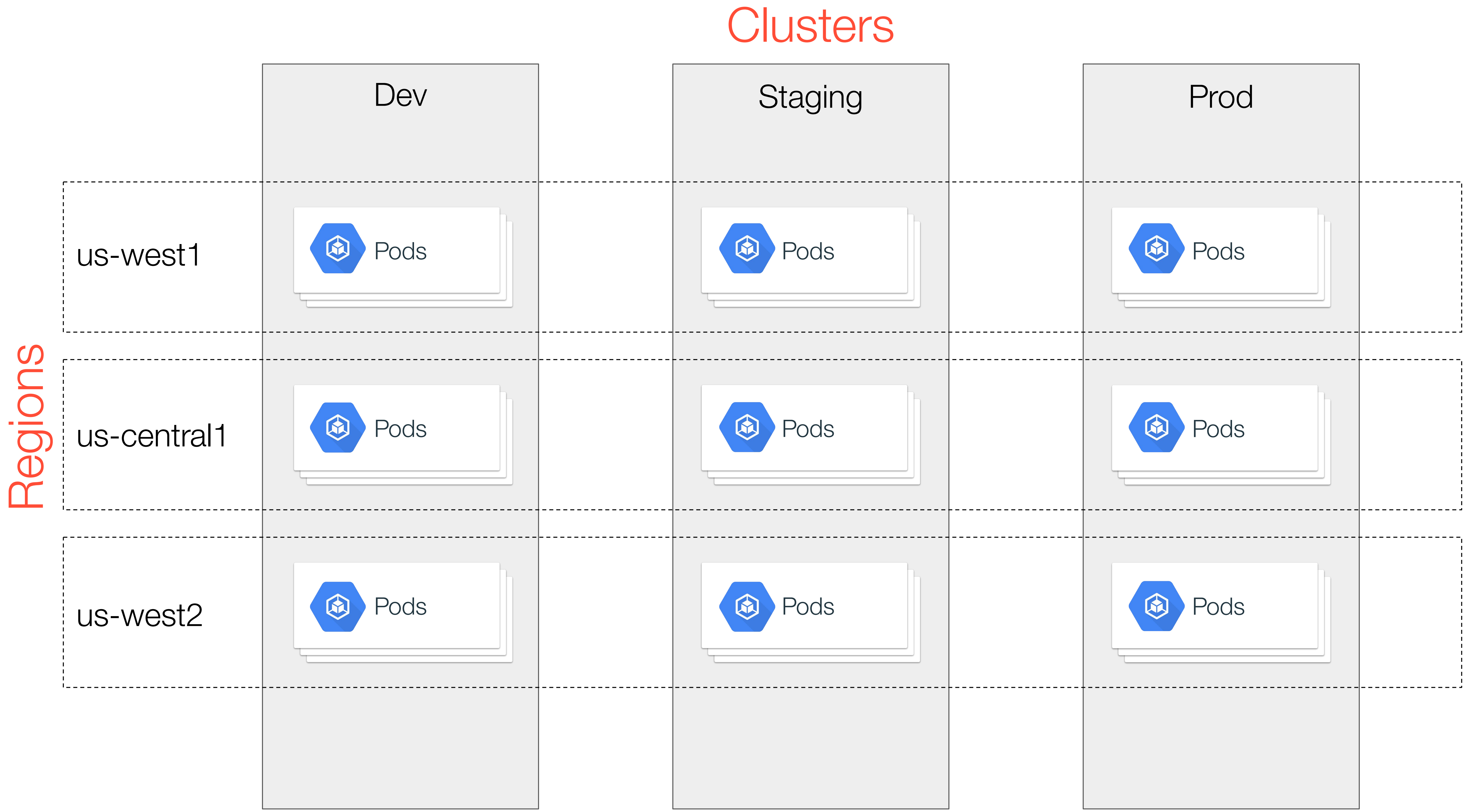
```
{
   nonMasqueradeCIDRs: [
      ~~10.0.0.0/8,~~
      172.16.0.0/12,
      192.168.0.0/16
   ]
}
```

vs.

Modified Config

```
{
   nonMasqueradeCIDRs: [
      10.11.0.0/16,
      172.16.0.0/12,
      192.168.0.0/16
   ]
}
```

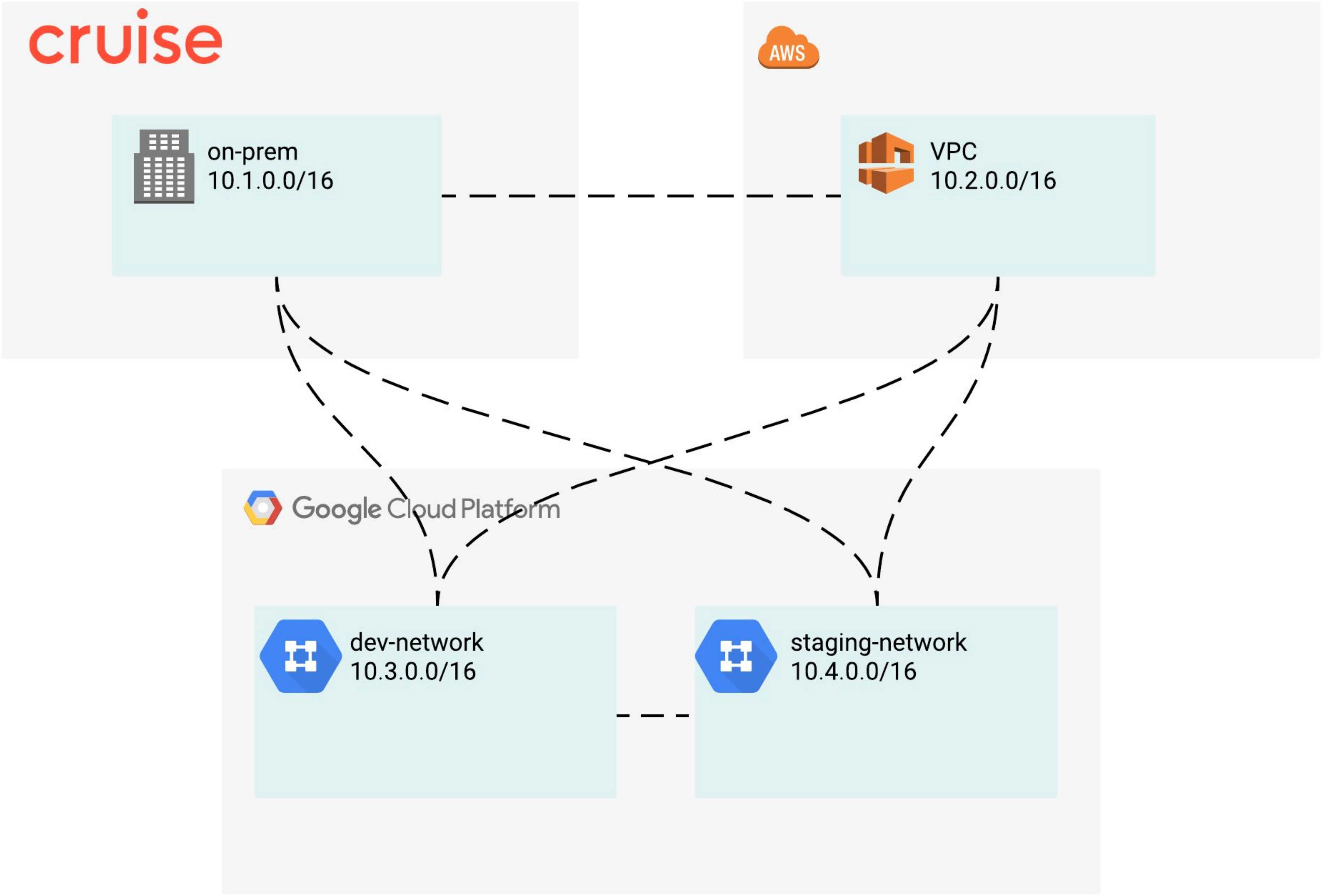
Environmental and Regional Clusters



cruise

Hybrid Networks

- **Gotchas of VPN tunnels**
  - n^2 tunnels
  - Static routes and route table management
  - Reduced performance

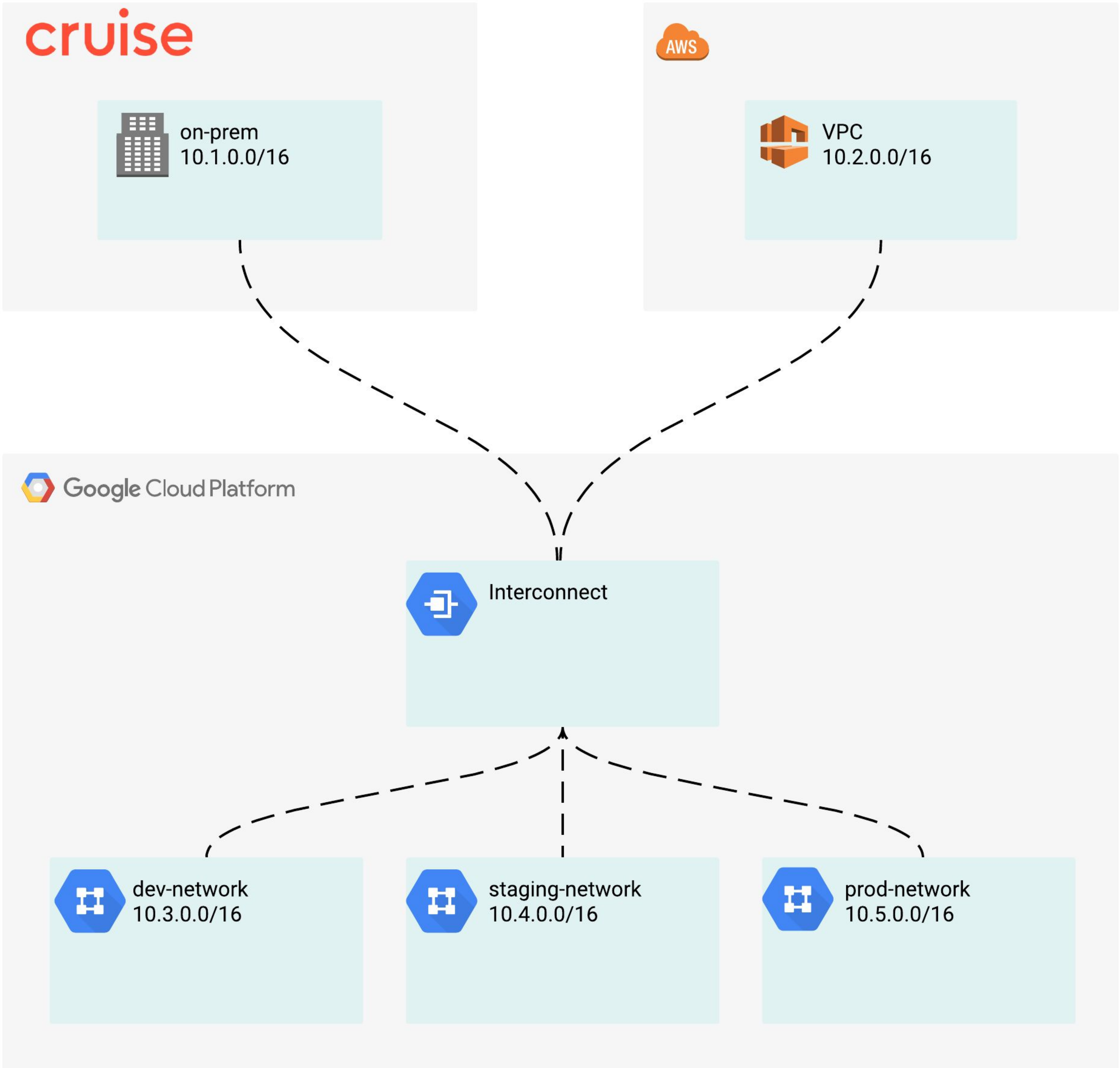


Hybrid Networks

- Interconnects solve n^2 tunnels problem.
  - Each network is connected through interconnects
  - No IPSEC tunnels

- BGP routers dynamically advertises all the routes

- Physical dedicated interconnects between networks provide high bandwidth

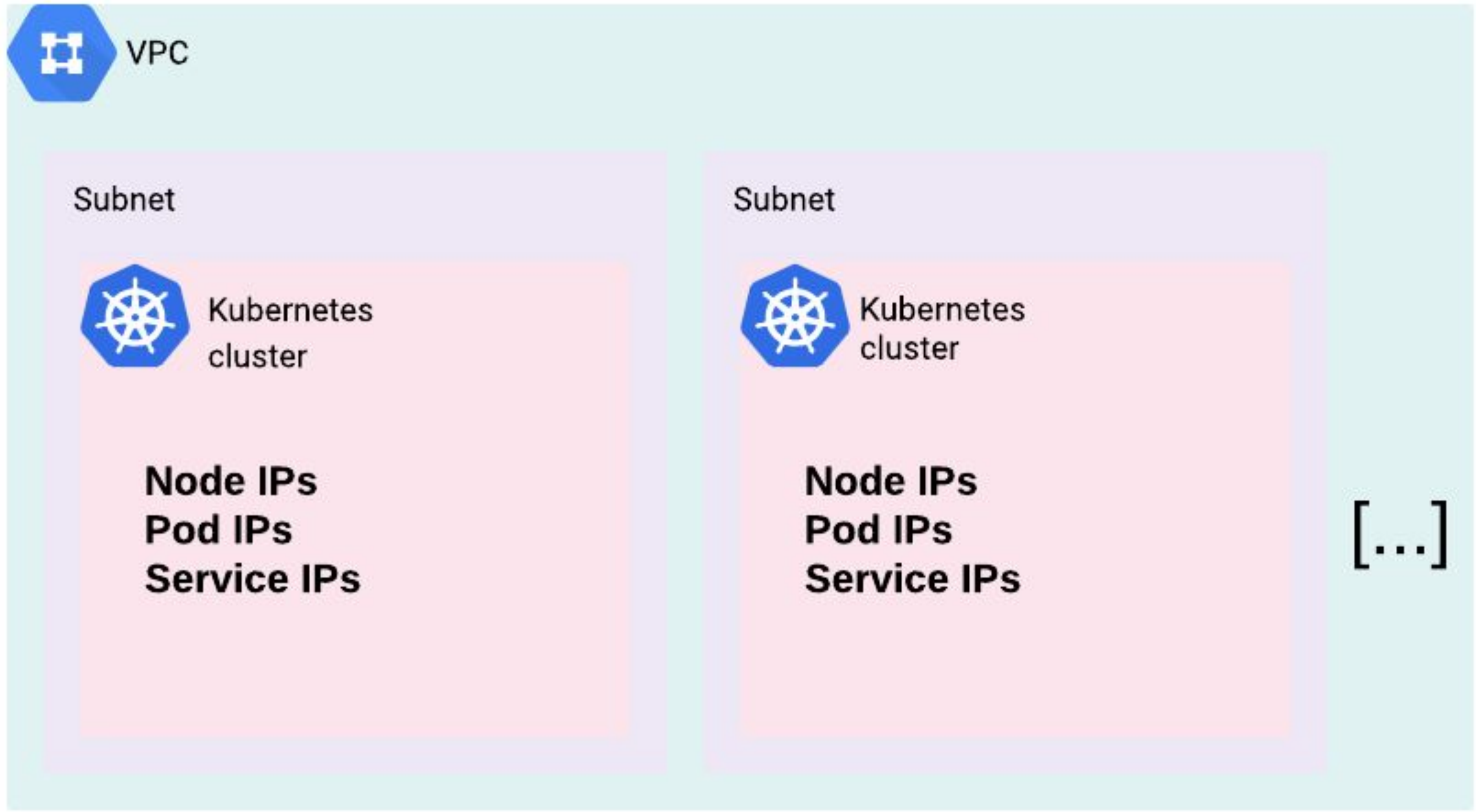


Container Platform Networking - http://bit.ly/cruise-networking

Multiple clusters

- One subnet per cluster.

- Those IP ranges depend on:
    - Environment (dev, staging, prod)
    - Region
    - Cluster sizing

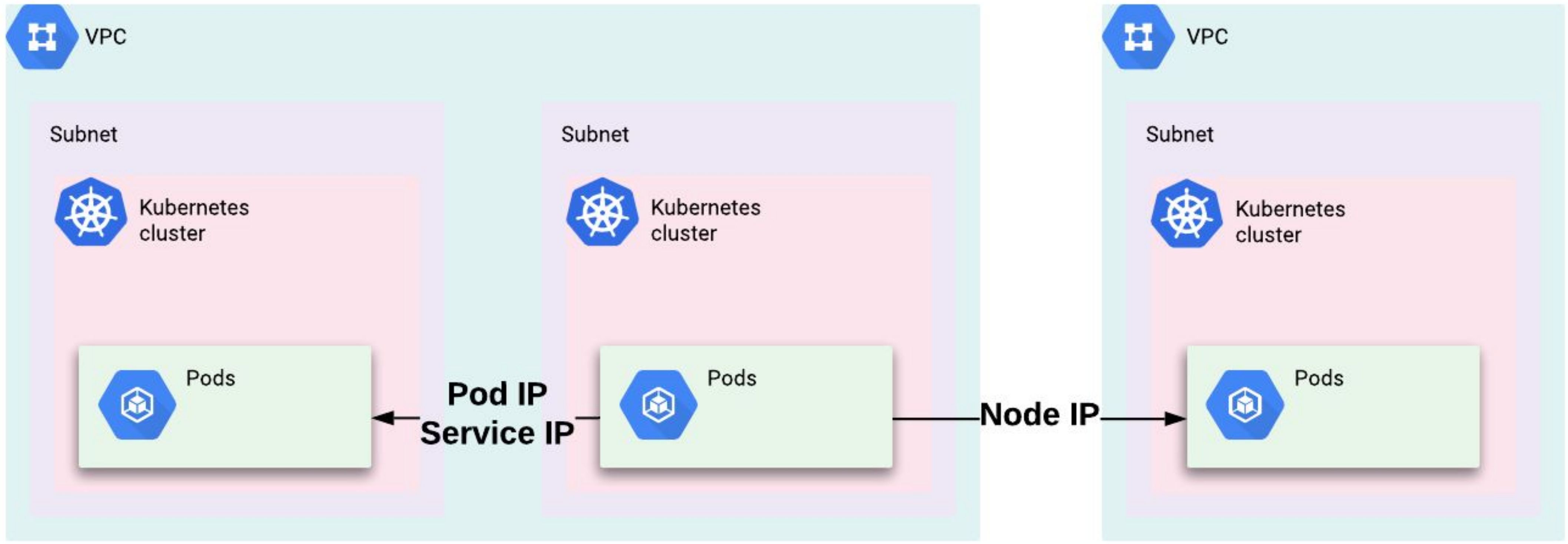


cruise

Constraints

- Node IPs globally unique
- Pod IPs locally unique per VPC (environment)
- Service IPs unique per VPC (environment)

Challenge: manual IP assignment process



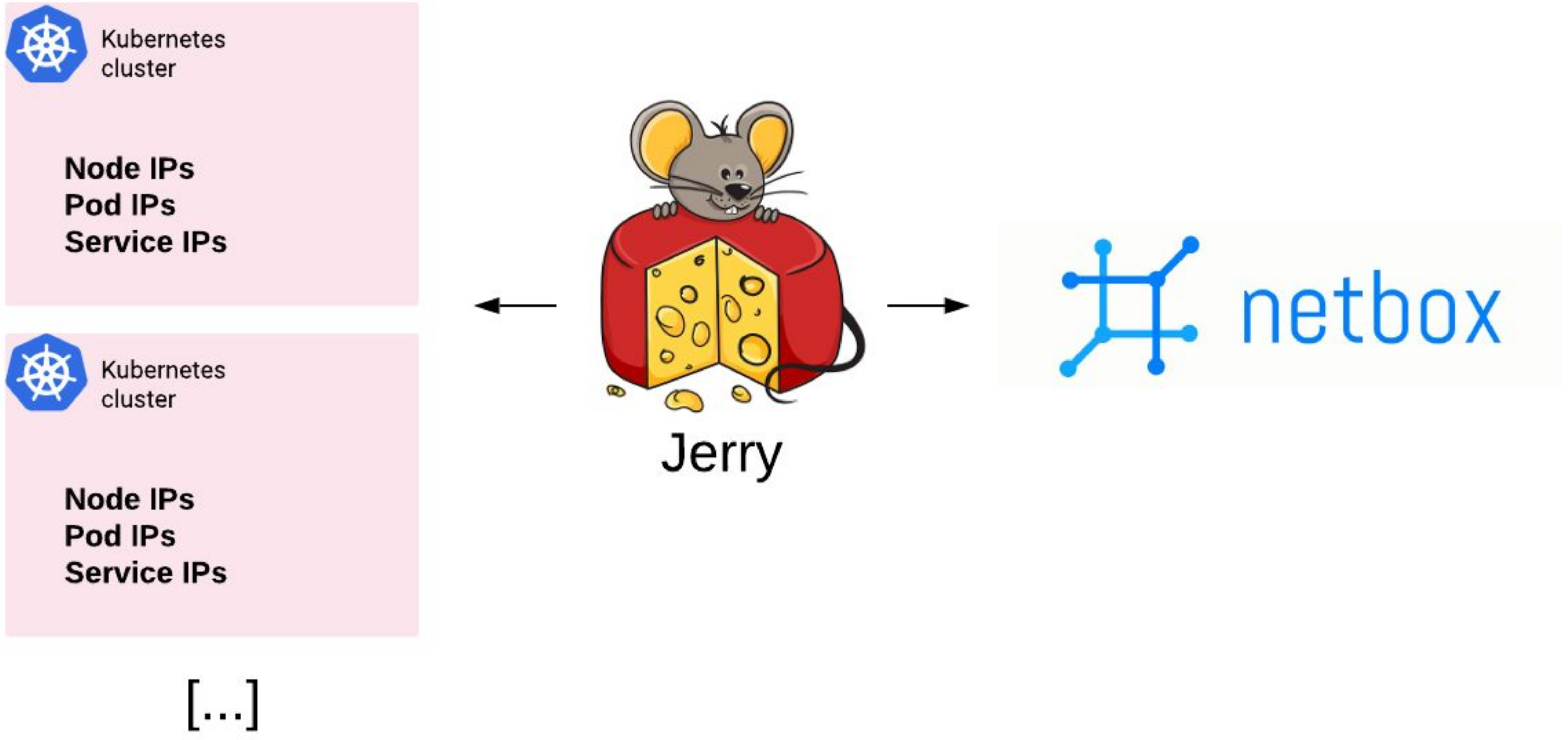
cruise

Jerry: Constraint based IP range assignments

4 parameters:

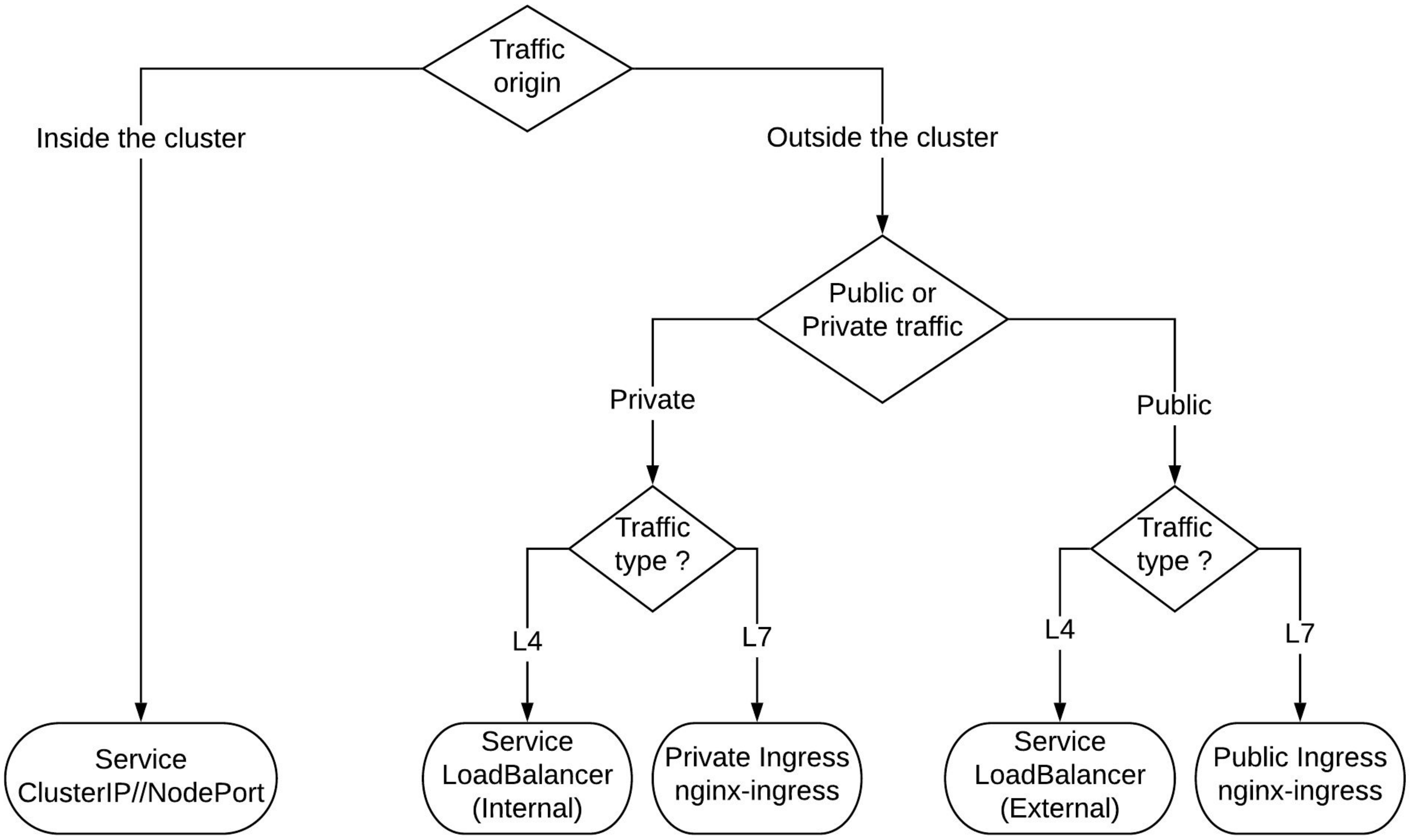
- Environment
- Region
- Max pods per node
- Max nodes

Syncs and validates IP ranges with Netbox

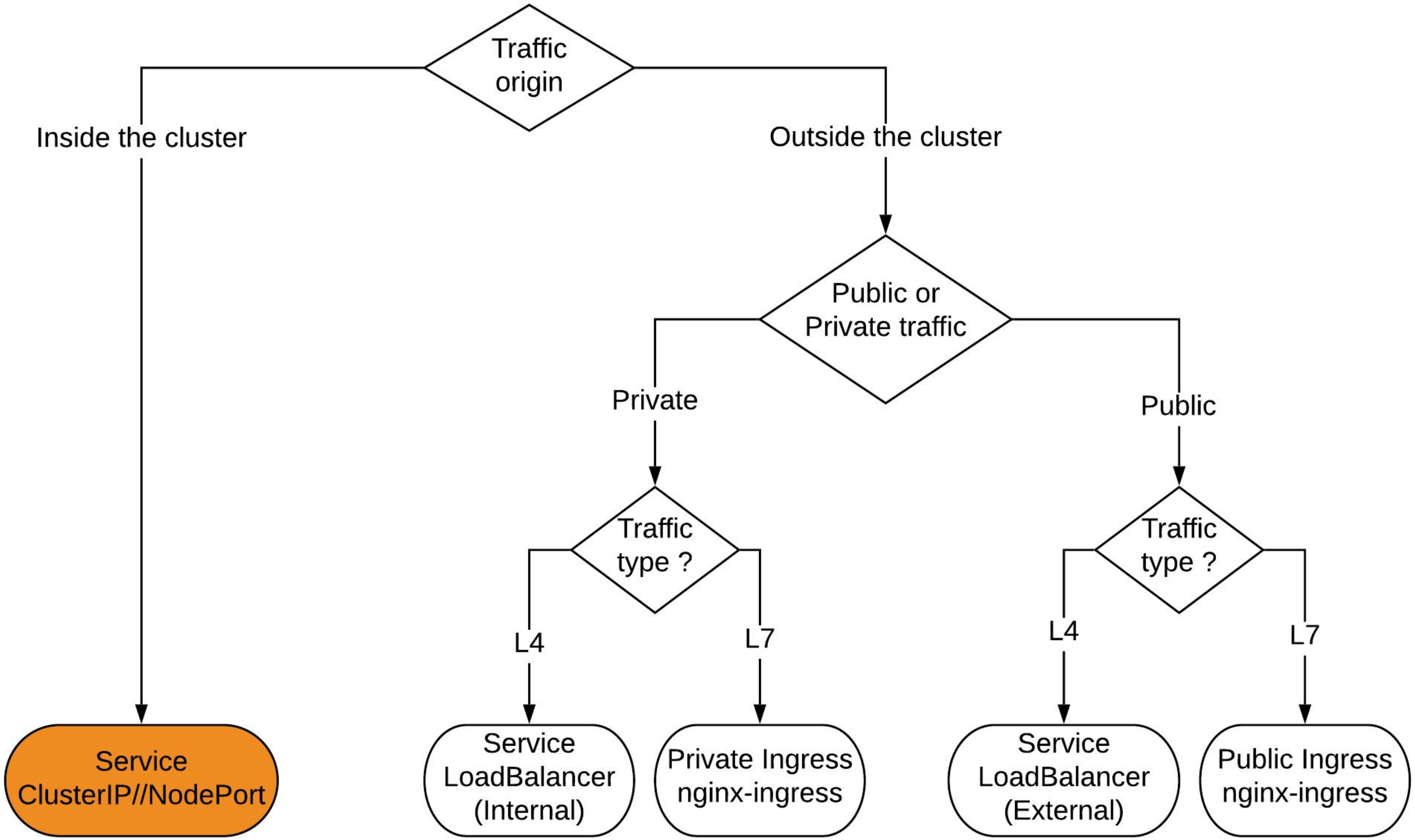


Lesson learned:
Meta Cluster visibility

Exposing services



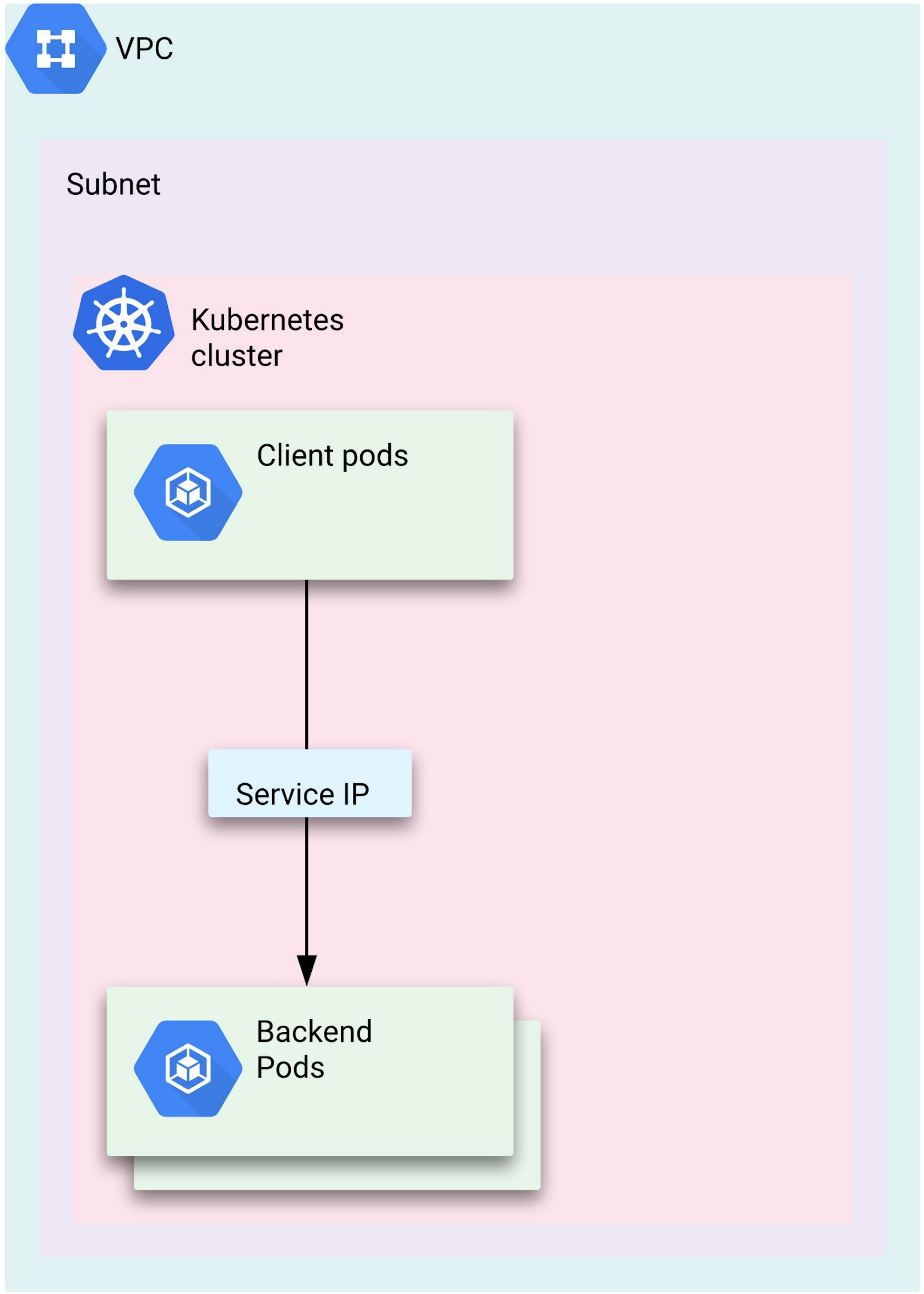
Exposing services



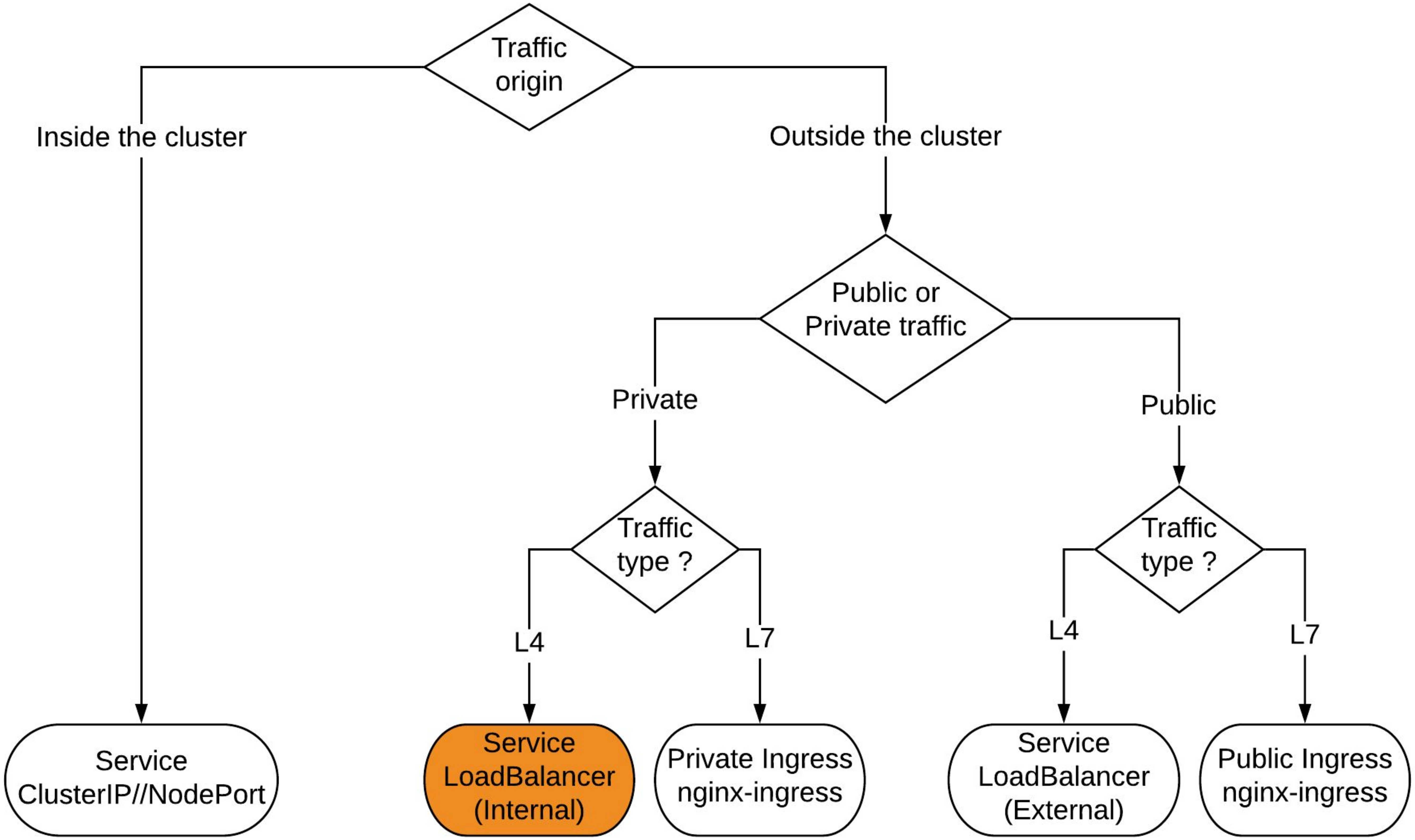
cruise

Intra-cluster: Kubernetes services

- Cluster IP
- Node ports



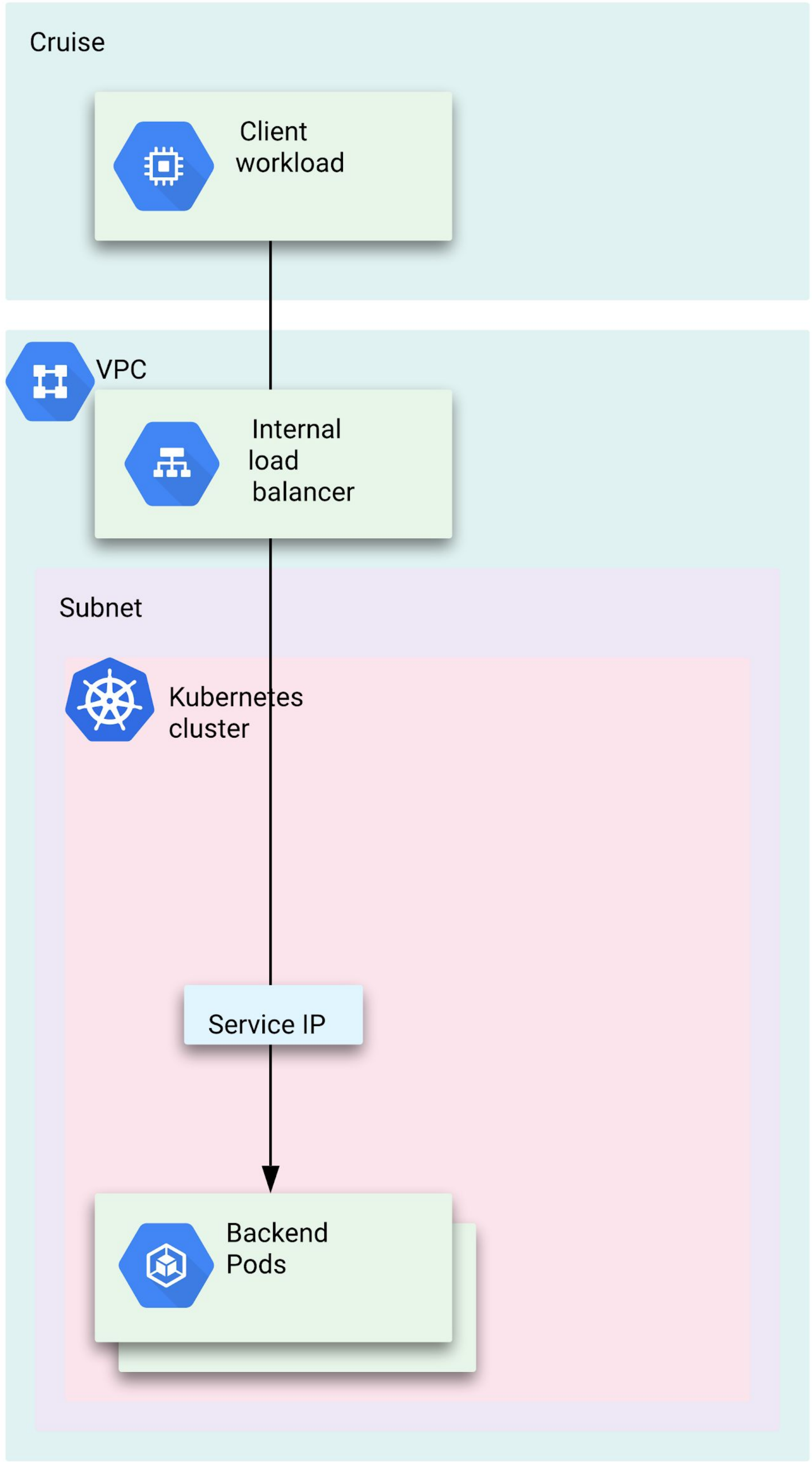
Exposing services



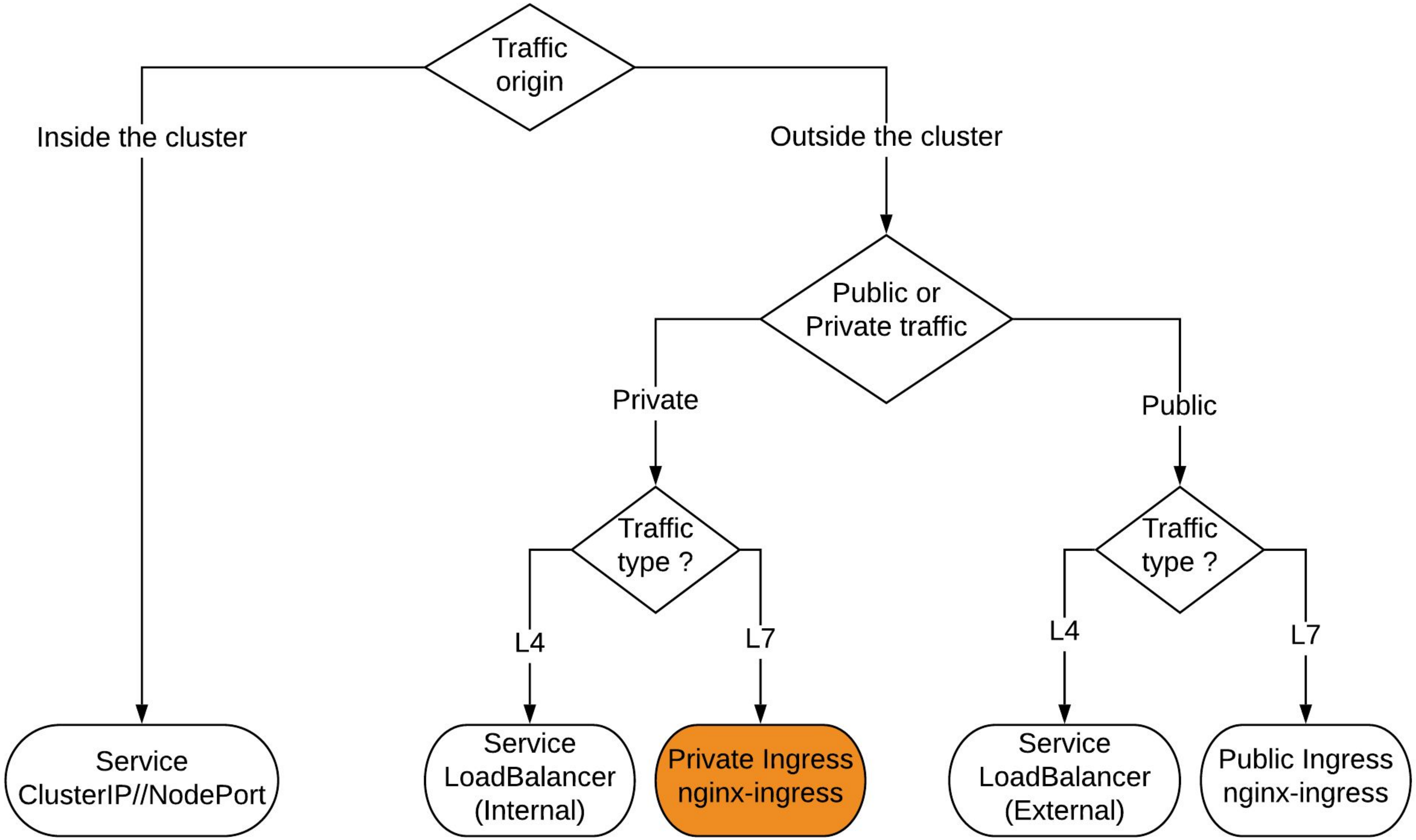
cruise

Private L4 traffic

- Internal load balancer on VPC
- Self-service with annotations

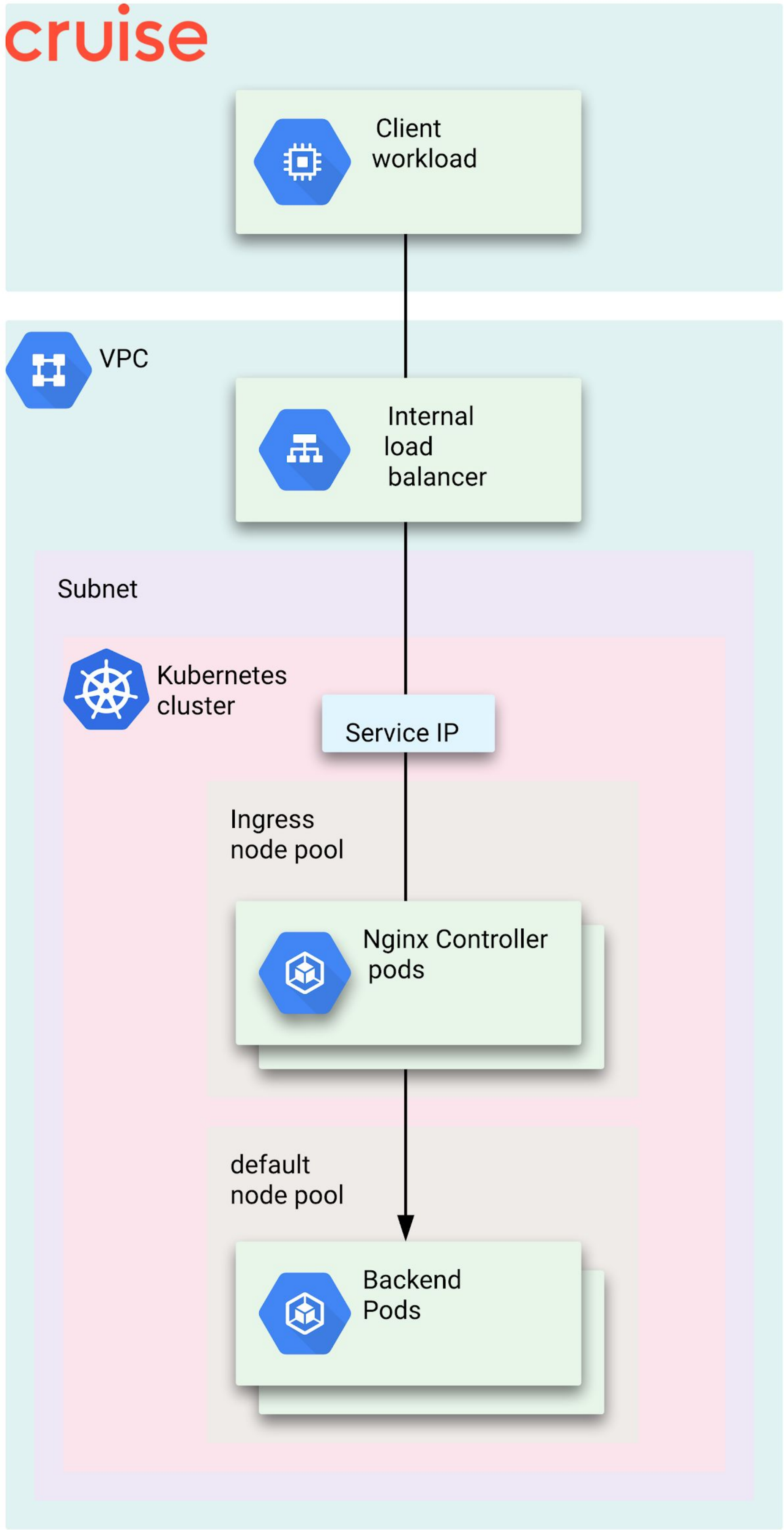


Exposing services



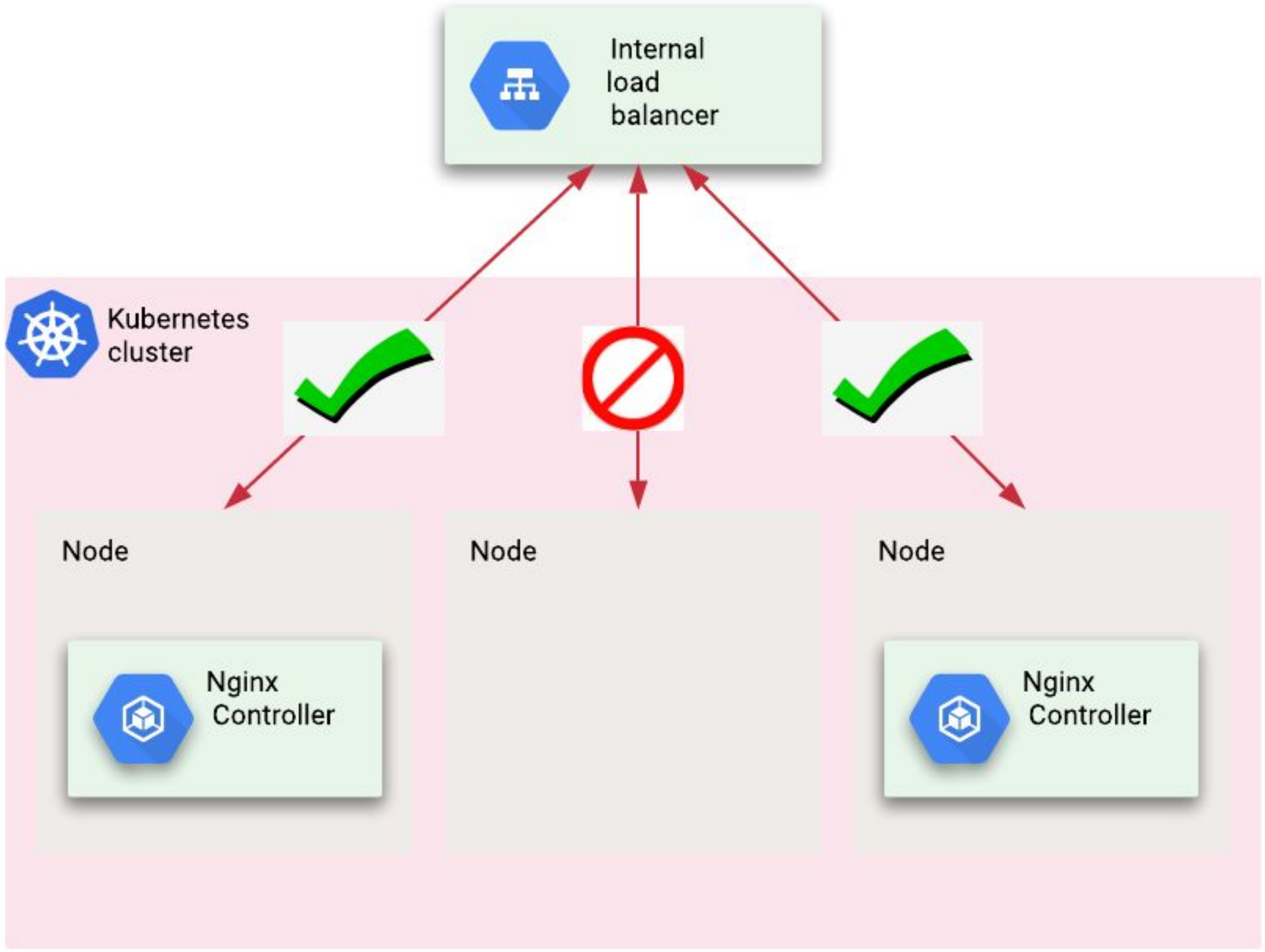
Private L7 traffic

- 95% of traffic
- Use standard **Ingress** resource
- Nginx-ingress
- Started with standard **in-cluster** controllers
- Dedicated node pool to avoid noisy neighbour // ressource



Private L7 traffic

- **externalTrafficPolicy** to avoid extra hop
- Load balancer Healthchecks the nodes. Only those with a controller available will succeed.

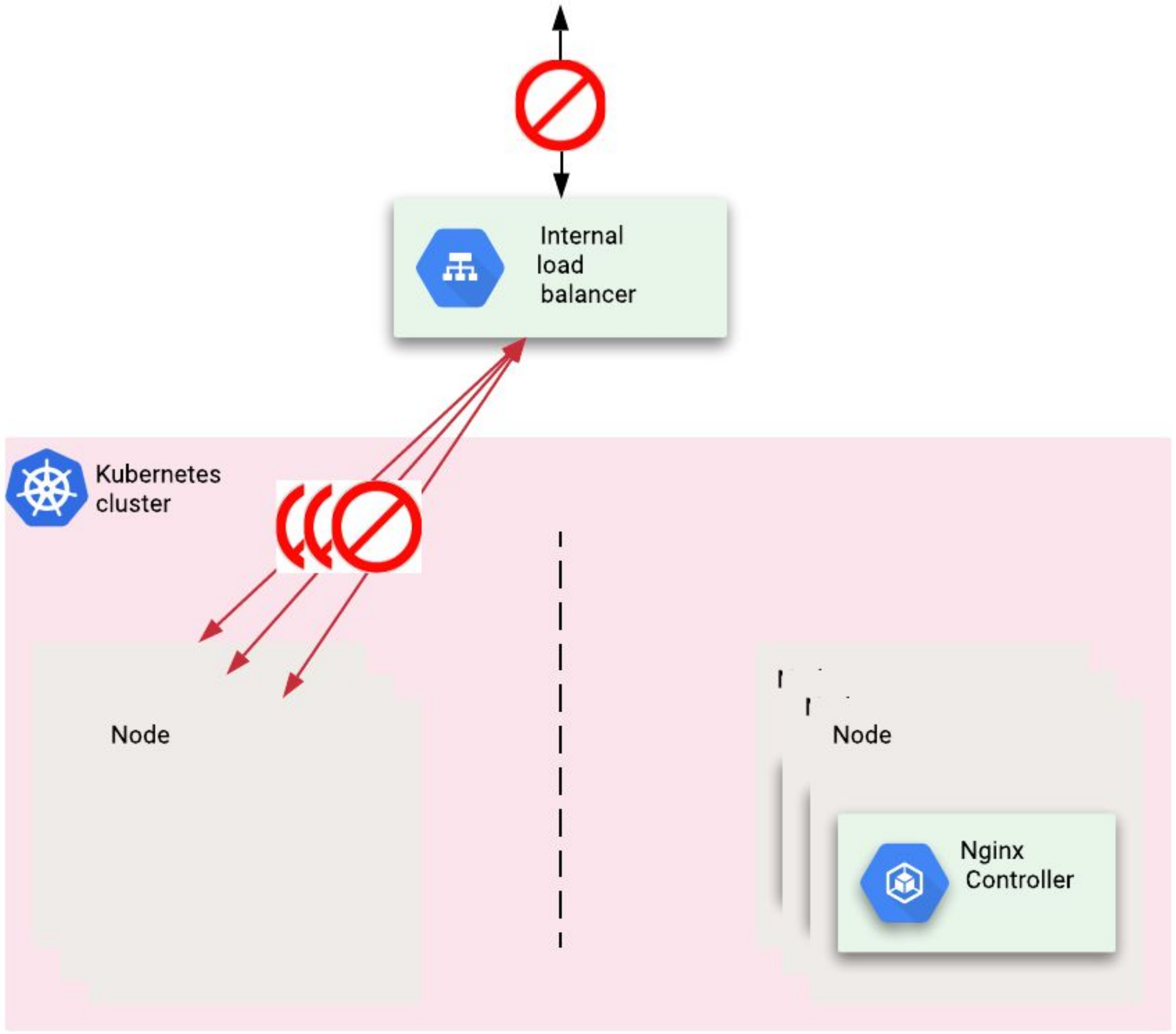


cruise

Private L7 traffic

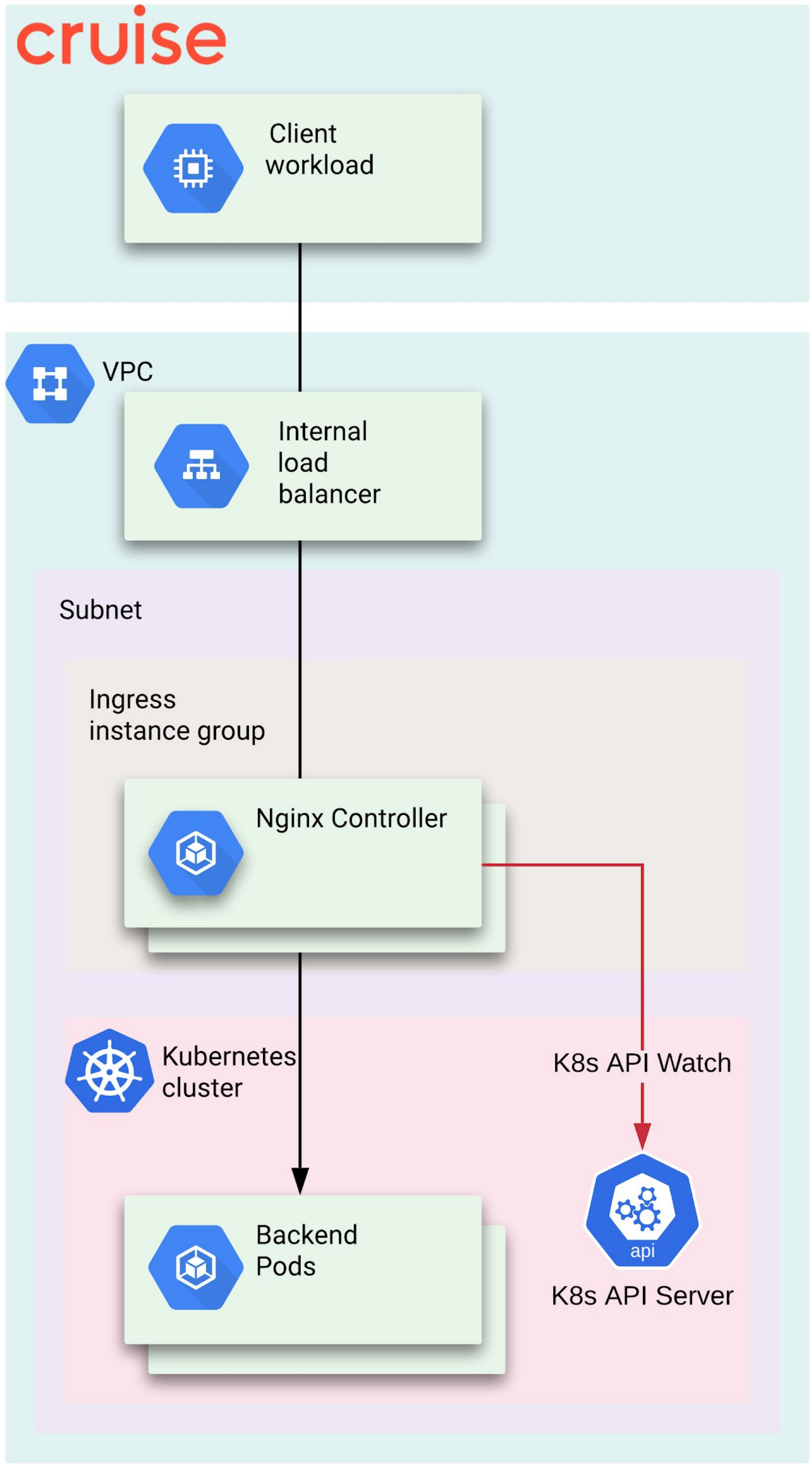
In-cluster **limitations**:

- Load Balancer healthcheck max. 250 nodes chosen randomly
- Decouple from Kubernetes management

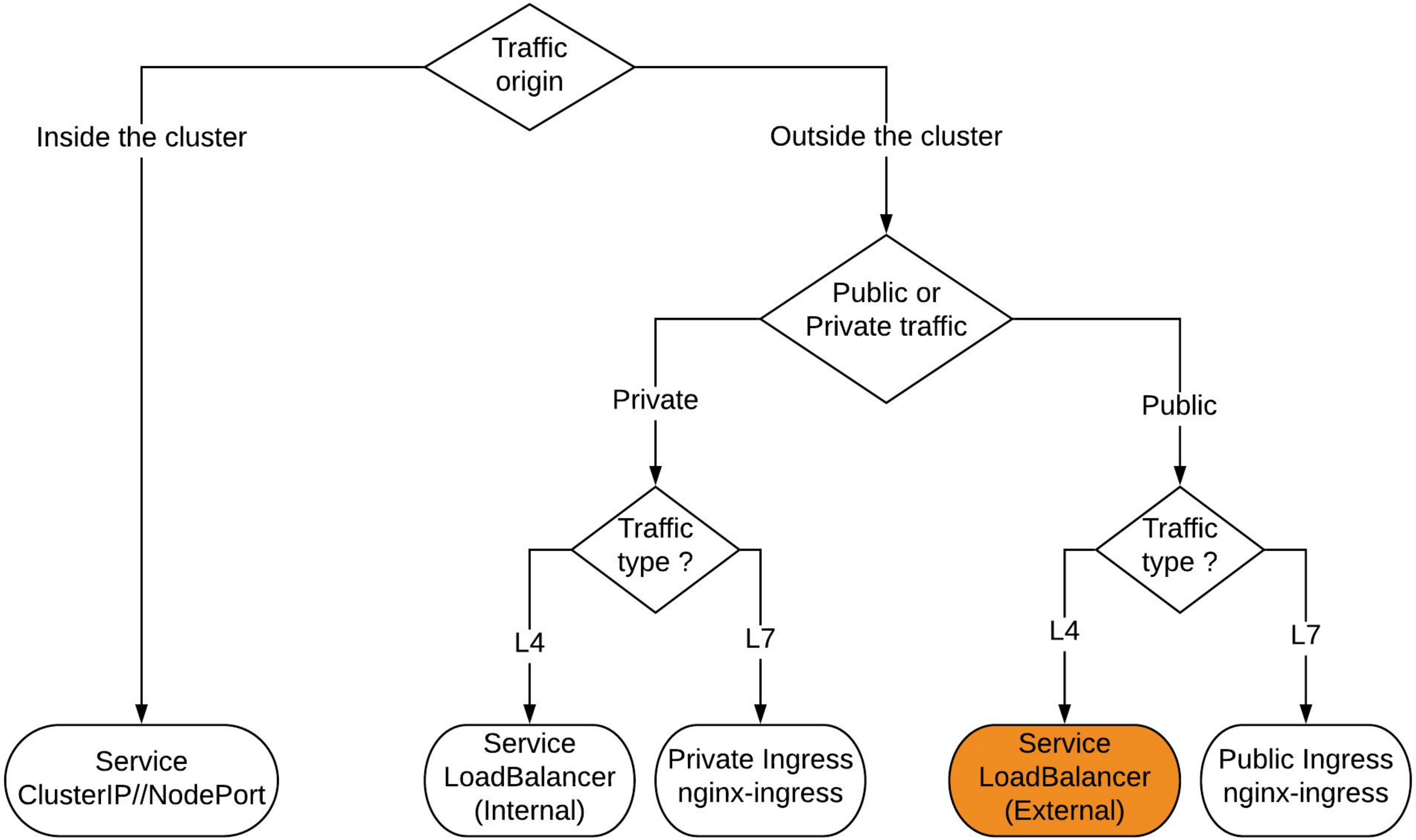


Private L7 traffic

- nginx-ingress **out-of-cluster**
- Managed in a separate instance group
- Watch Kube API outside the cluster
- Sends traffic directly to pods



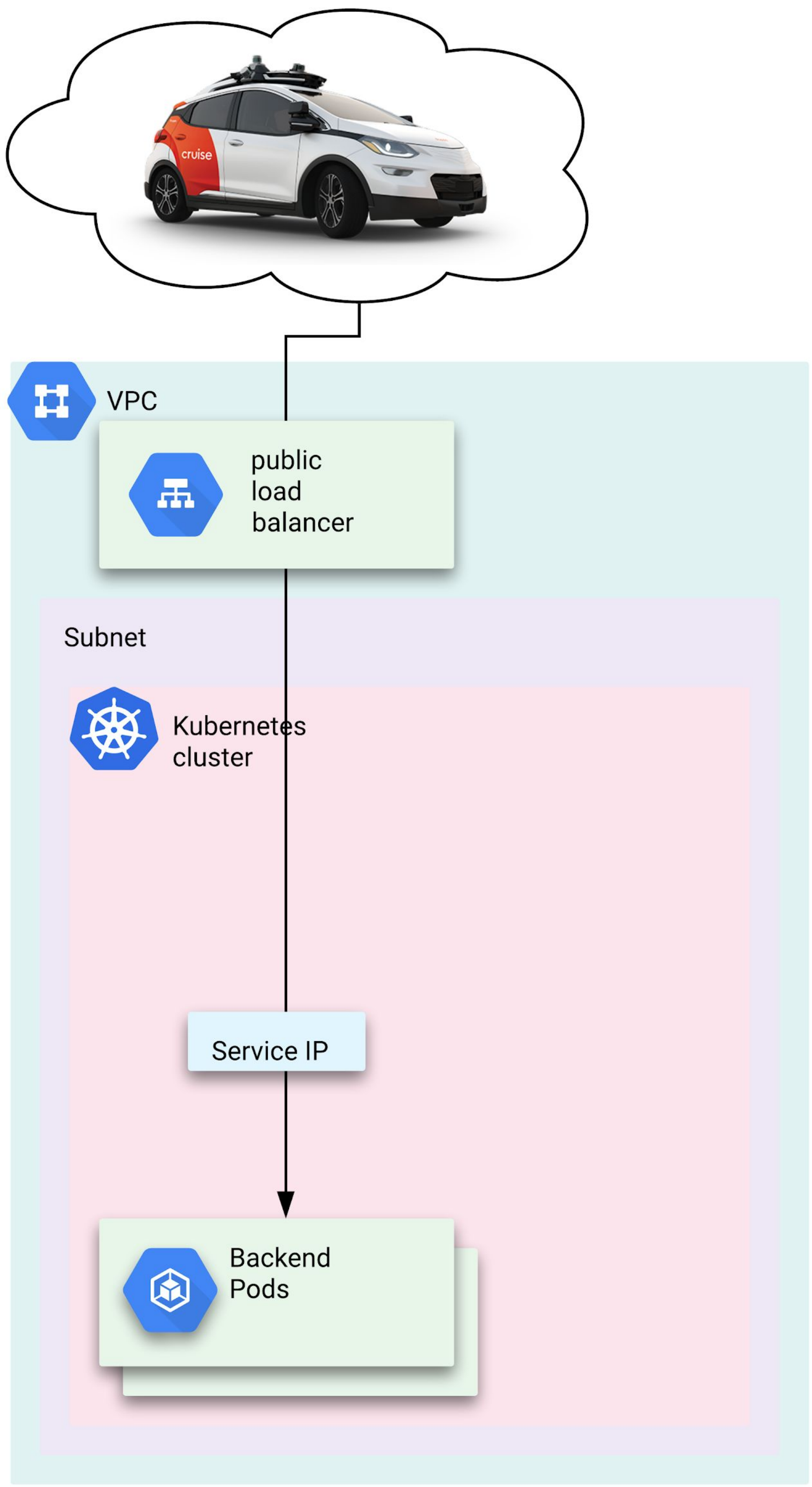
Exposing services



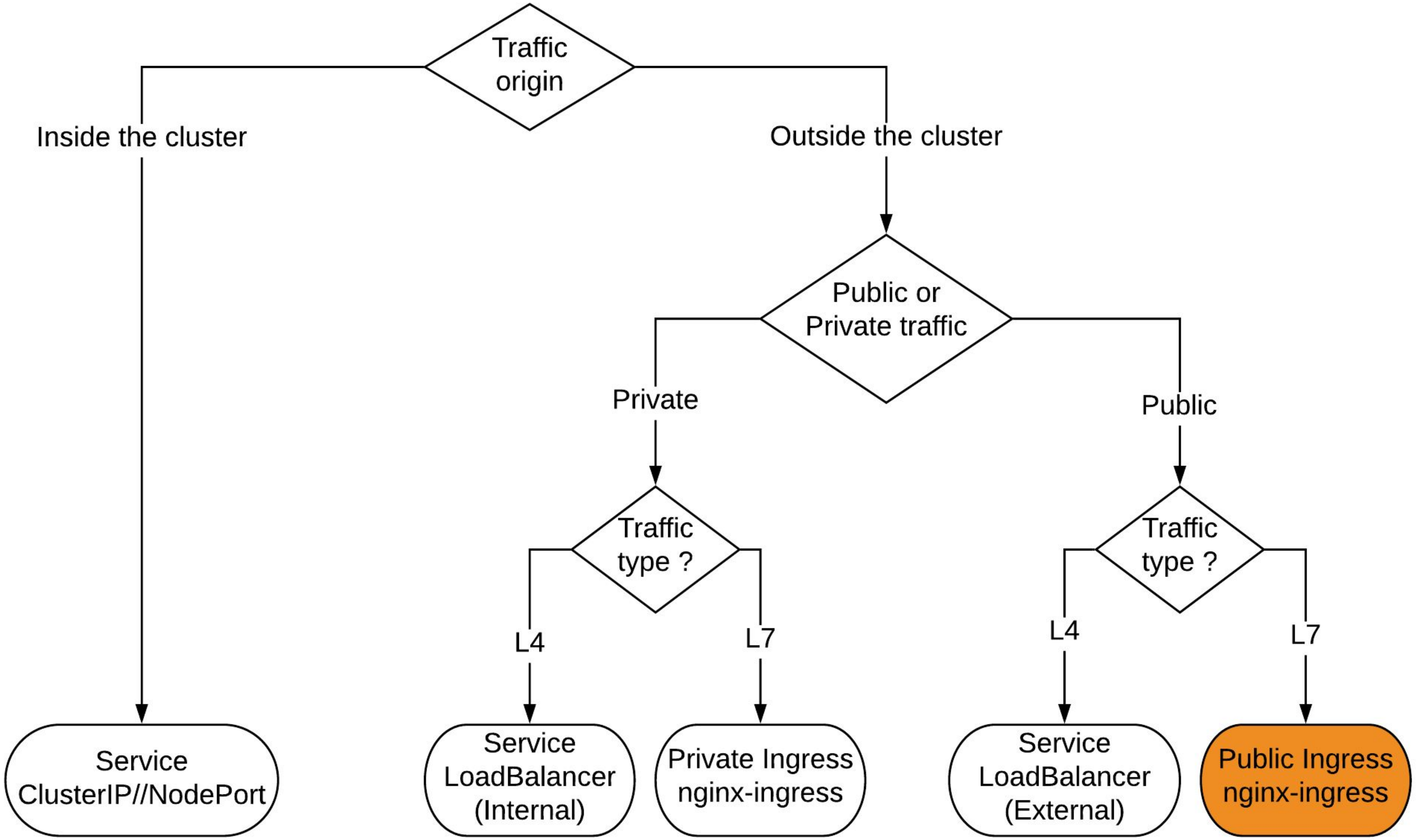
cruise

Public L4 traffic

- Public Load balancer on environment VPC
- Exceptional cases only (requires review)
- Firewalled and mTLS (unmanaged by Kubernetes)

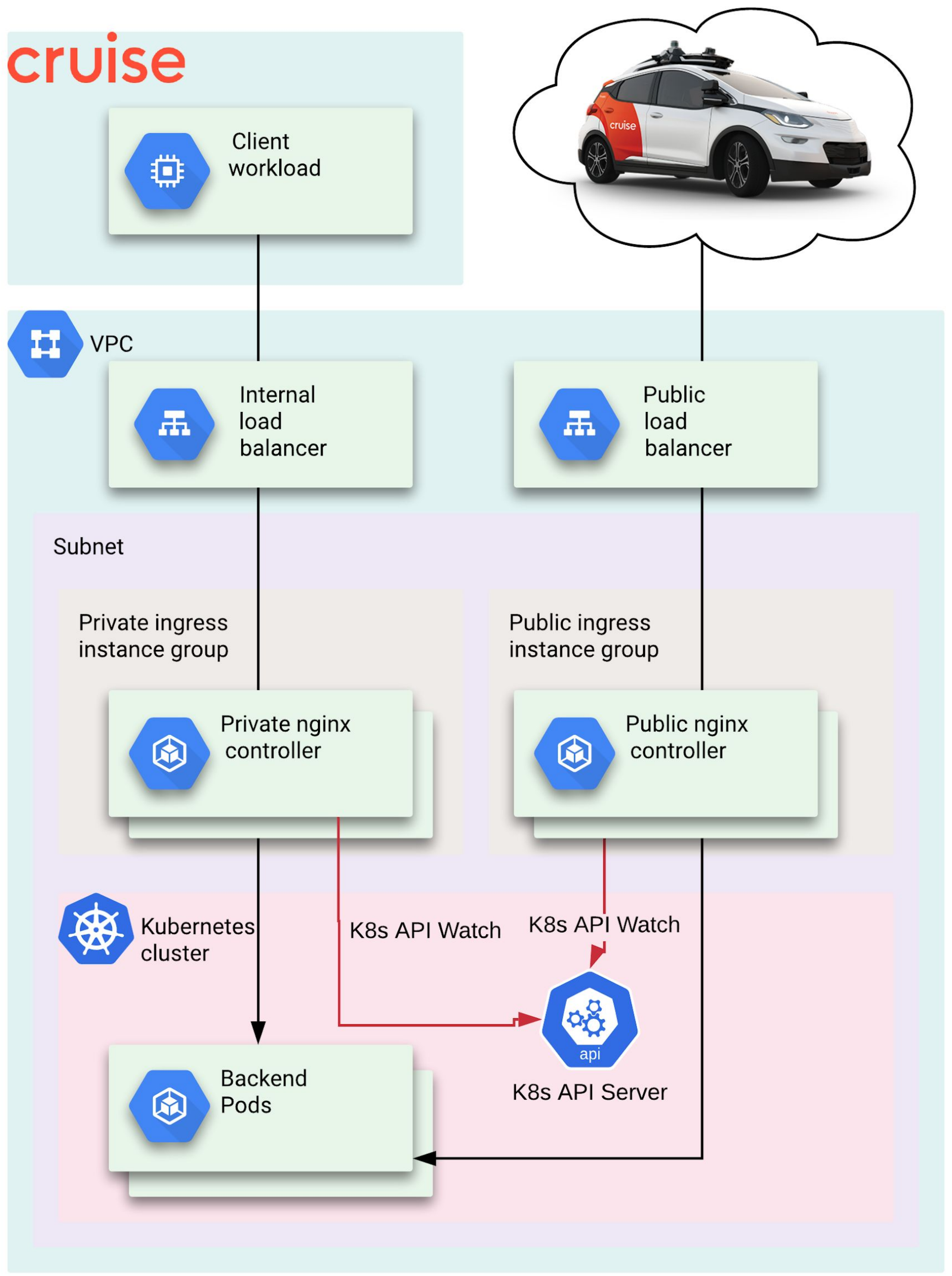


Exposing services



Public L7 traffic

- Started with L7 GLBC
- Evolved to **nginx-ingress**
- Same setup as private ingress
- Different nginx-ingress **annotation**



cruise

Lesson learned:

Support a small amount of options but support them **well**

Private ingress annotation:

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  annotations:
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/affinity: cookie
```

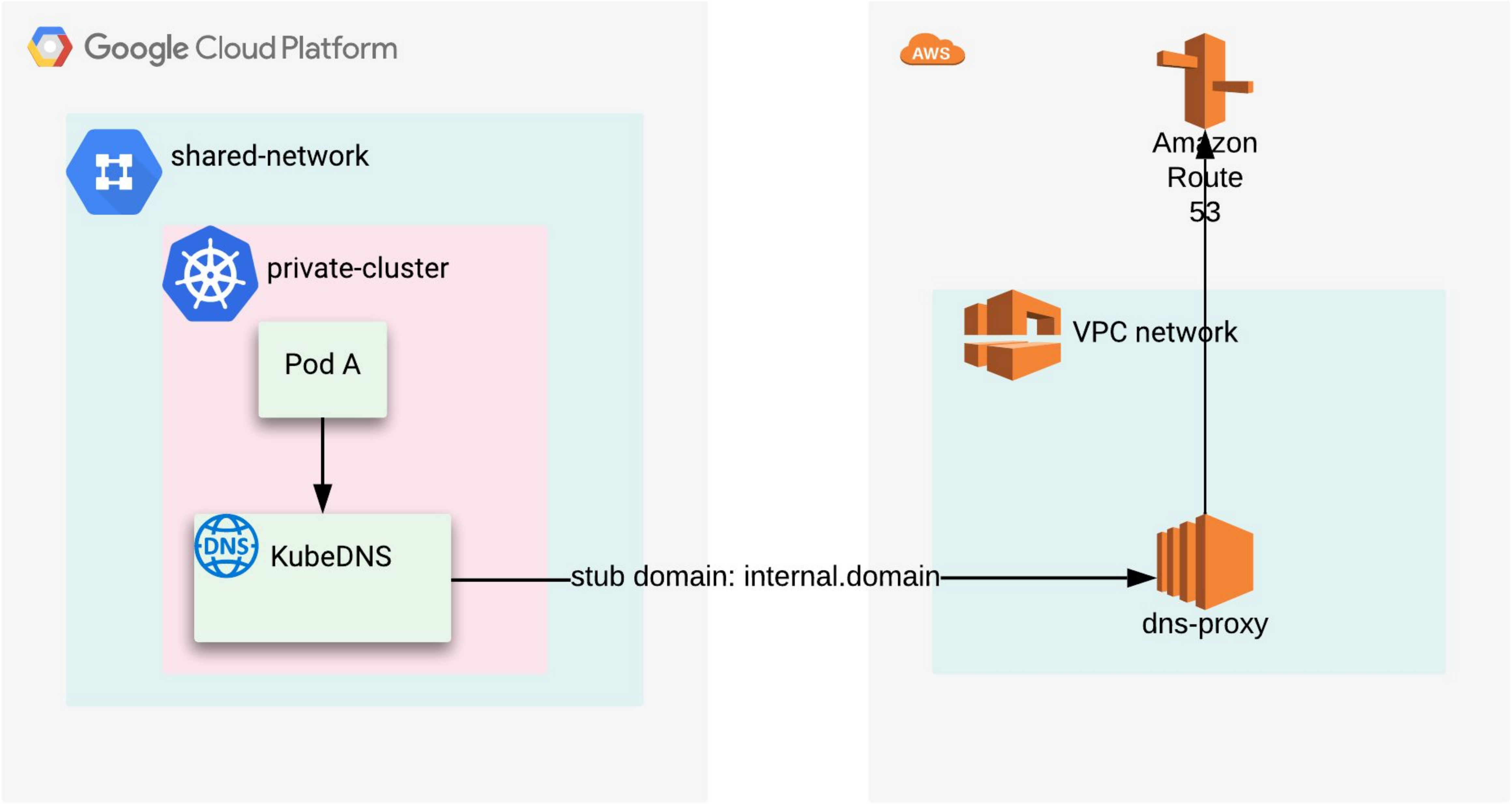
Public ingress annotation:

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  annotations:
    kubernetes.io/ingress.class: public
    nginx.ingress.kubernetes.io/affinity: cookie
    nginx.ingress.kubernetes.io/proxy-body-size: 20m
```

cruise

Hybrid DNS

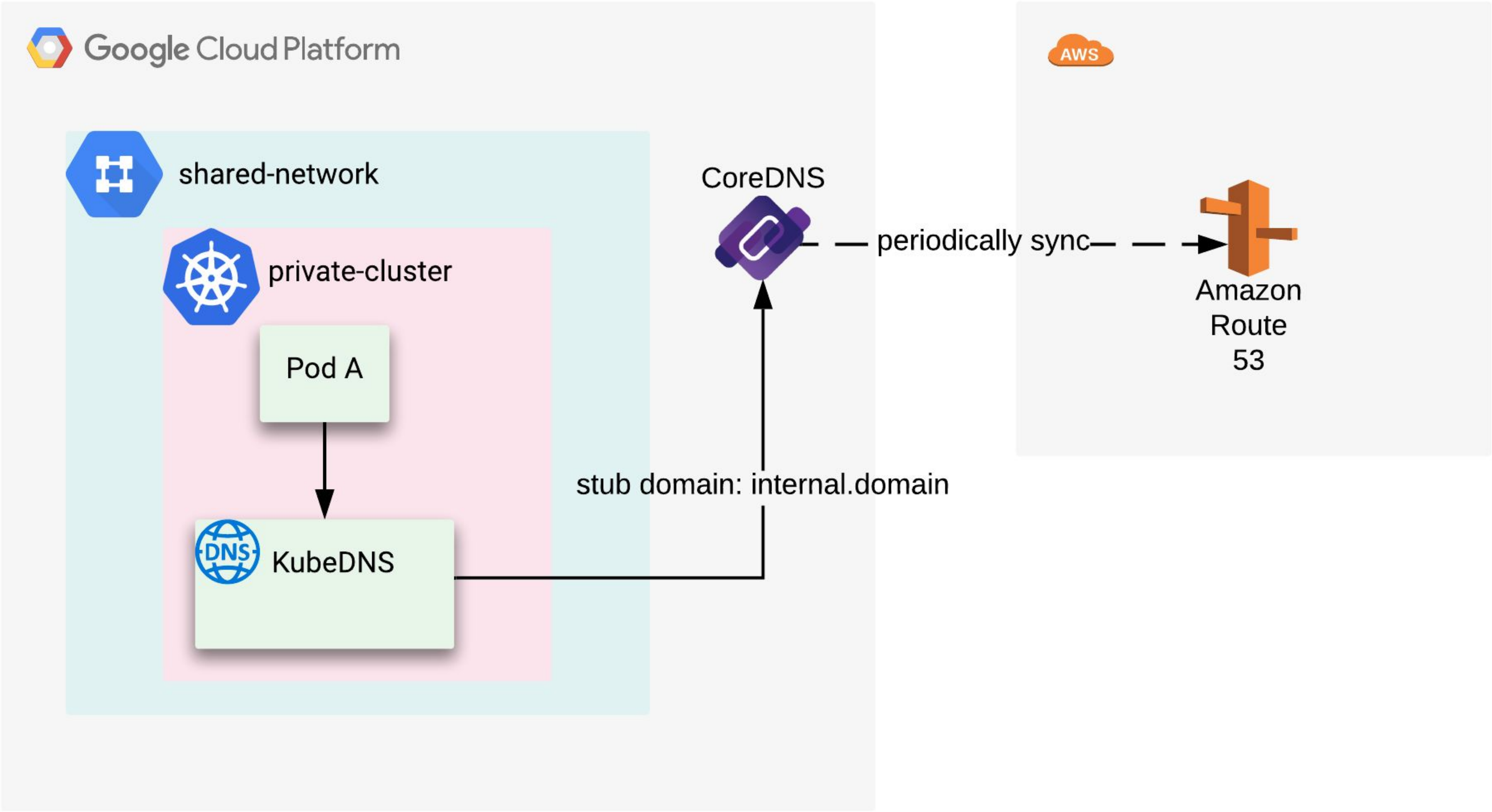
- **Goal:** DNS that works in hybrid environment
- **State:**
  - Domain records are stored in Route53
  - DNS proxies used for forwarding internal queries to Route 53
- **First attempt:** Configure KubeDNS to forward cruise domains to DNS proxies via stub domains
- **End result:** High latency!



cruise

Hybrid DNS

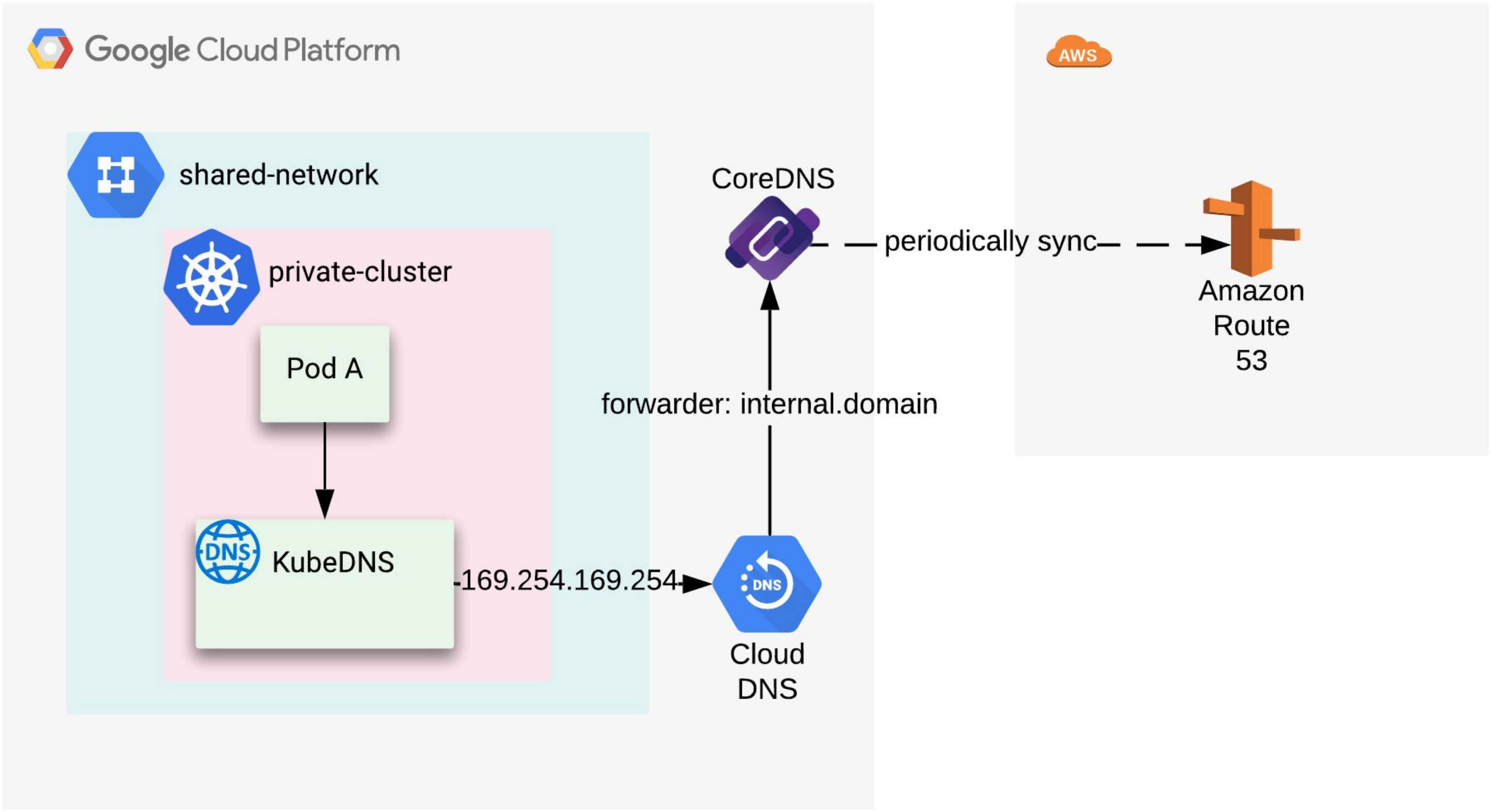
- **Solution**: Use CoreDNS and enable Route53 backend
- **Challenge:** Route53 plugin only supports A records and fetches all records one-by-one.
  - Added support for batch requests and allowed fetching all record types.
  - Configured CoreDNS to periodically pull from Route53.



cruise

Hybrid DNS

- **Challenge:** Connecting to CoreDNS via stub domains only solve the problem for the Kubernetes clusters.
- **Solution:** Incorporate Cloud DNS and forwarding zones.



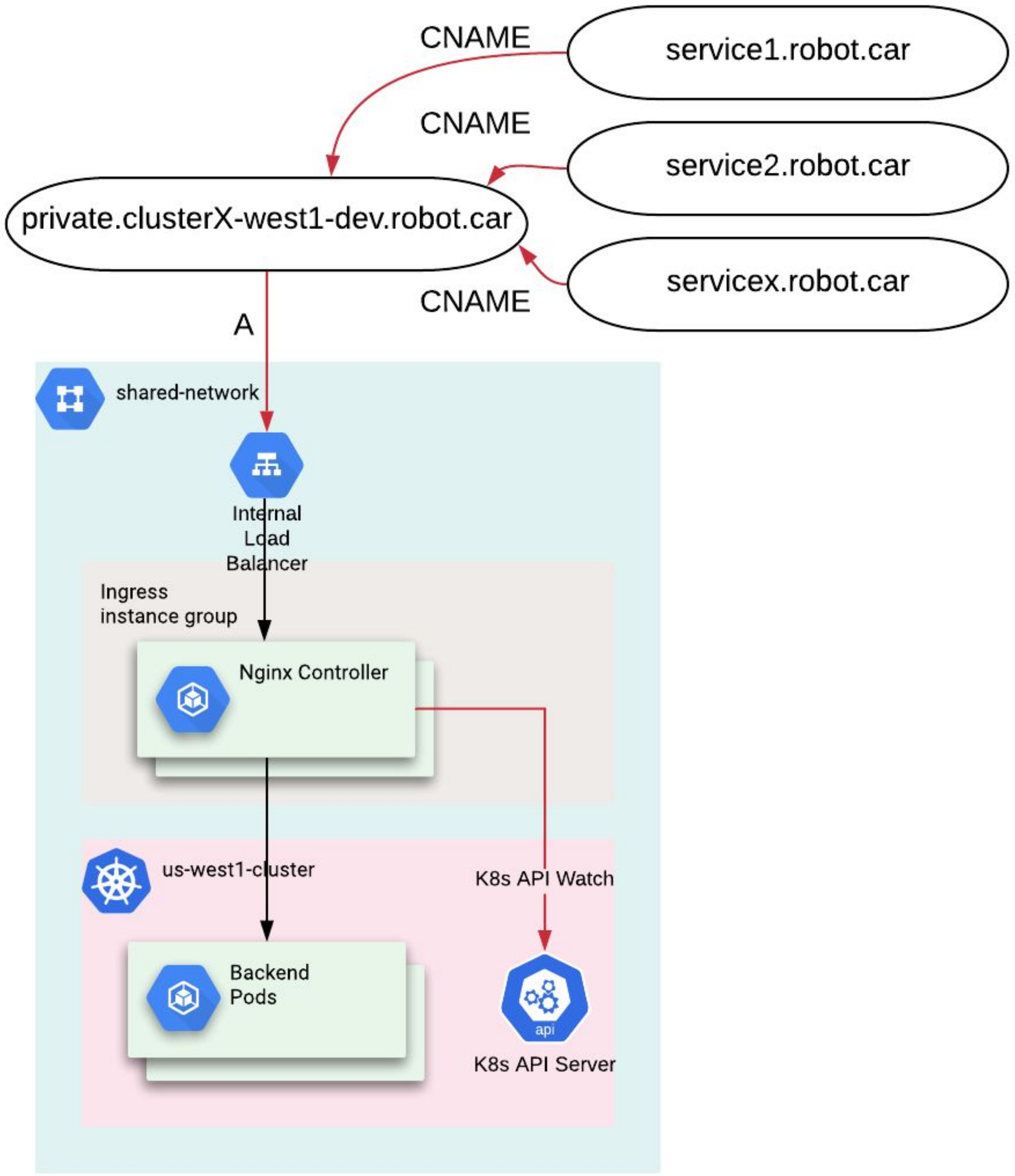
cruise

DNS configuration:

Each cluster publishes a DNS record for:

- Private Ingress endpoint
- Public Ingress endpoint

Tenants CNAME to the ingress endpoints



Ingress logging

- Structured logs with Fluentd
- Easy to search and filter
- Example filter:

```
jsonPayload.ingress_name="ingress-example"
jsonPayload.ingress_namespace="paas-tools"
```

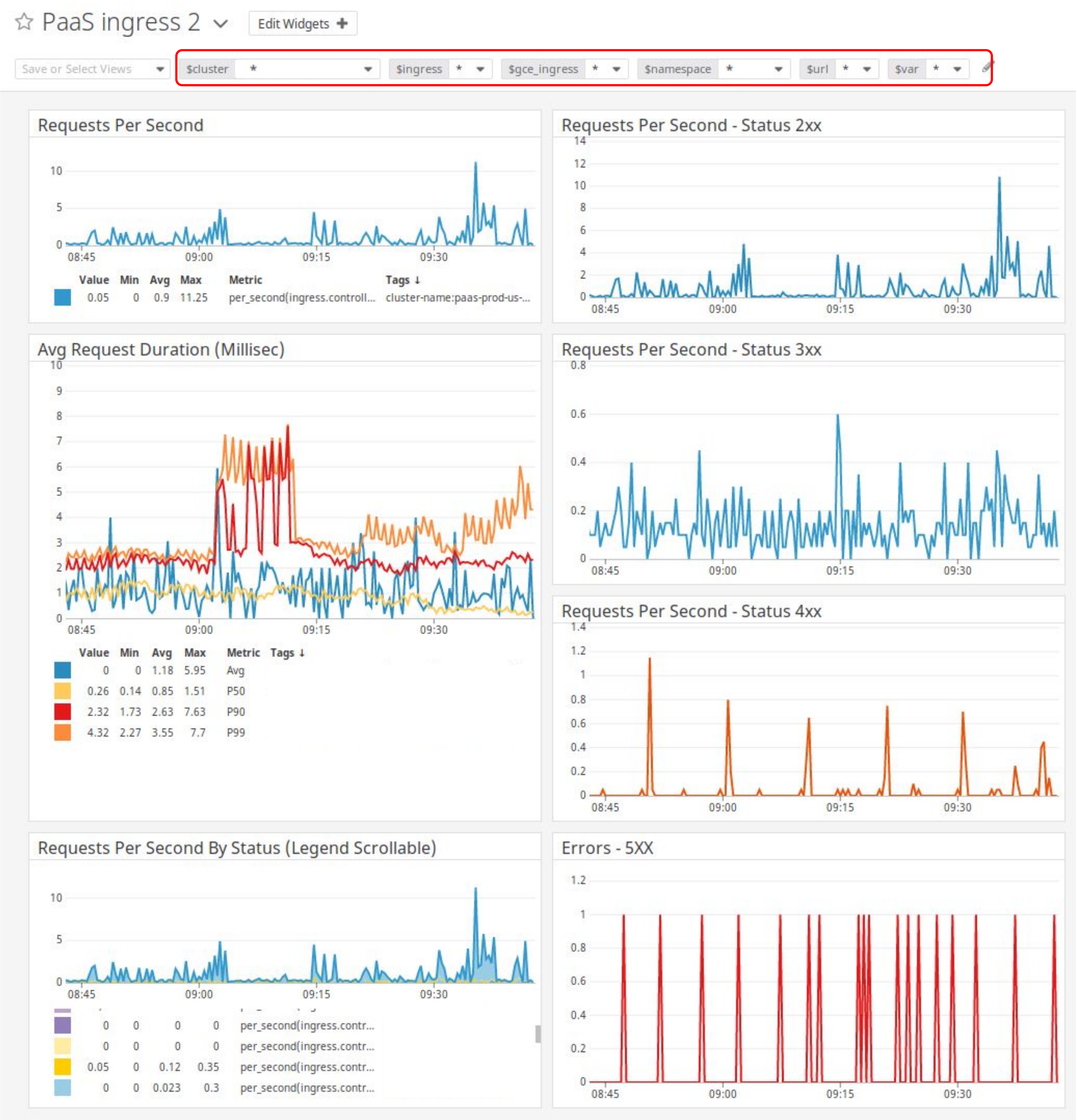

```
2019-11-19 09:37:08.000 PST GET 200 819 B 4 ms curl/7.58… /
10.252.10.2 - "GET /" 200 819 "-" "curl/7.58.0"
{                                                        Expand all | Collapse all
  httpRequest: {…}
  insertId: "98nchae9hx58v5cki"
  jsonPayload: {
    cluster: "paas-dev-us-west1"
    container_id: "89031a874870f3b64263bf452e48c46f8925f6b40583489019d46ce4b0c07ccb"
    container_name: "/nginx_nginx_1"
    host: "ingress-example.robot.car"
    ingress_name: "ingress-example"
    ingress_namespace: "paas-tools"
    log: "{"host": "ingress-example.robot.car", "httpRequest": {"latency": "0.004s", "referer": "-", "remoteIp": "10.252.10.2", "requestMethod": "GET", "requestSize": "38", "requestUrl": "/", "responseSize": "819", "status": "200", "userAgent": "curl/7.58.0"}, "ingress_name": "ingress-example", "ingress_namespace": "paas-tools", "referer": "-", "time": "19/Nov/2019:17:37:08 +0000"}"
    referer: "-"
    source: "stdout"
  }
  labels: {…}
  logName:
  receiveTimestamp: "2019-11-19T17:37:13.120505549Z"
  resource: {…}
  timestamp: "2019-11-19T17:37:08Z"
}
```

Monitoring

Ingress metrics

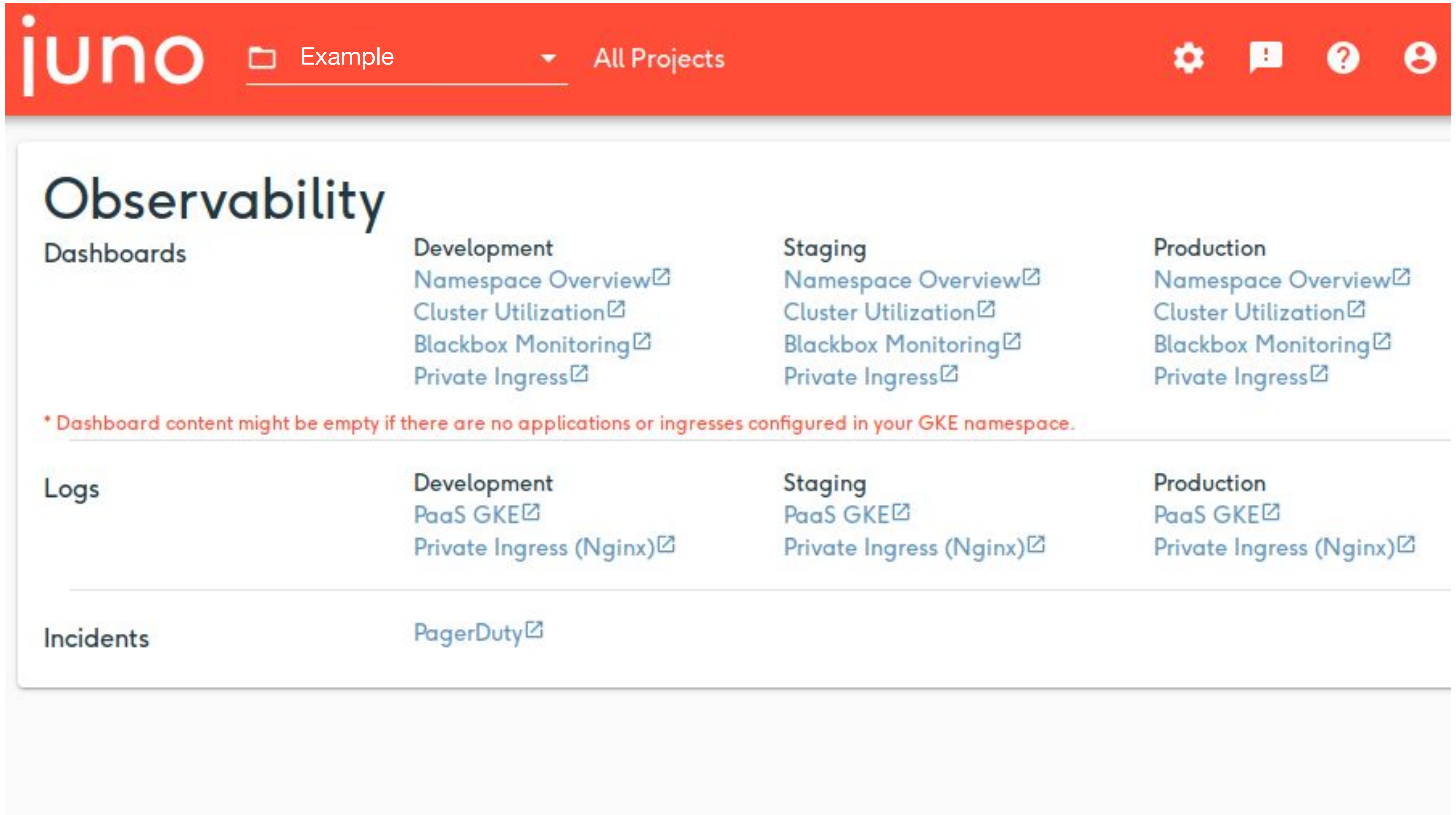
Filtered by:

- Cluster
- Namespace
- Ingress



Lesson learned:

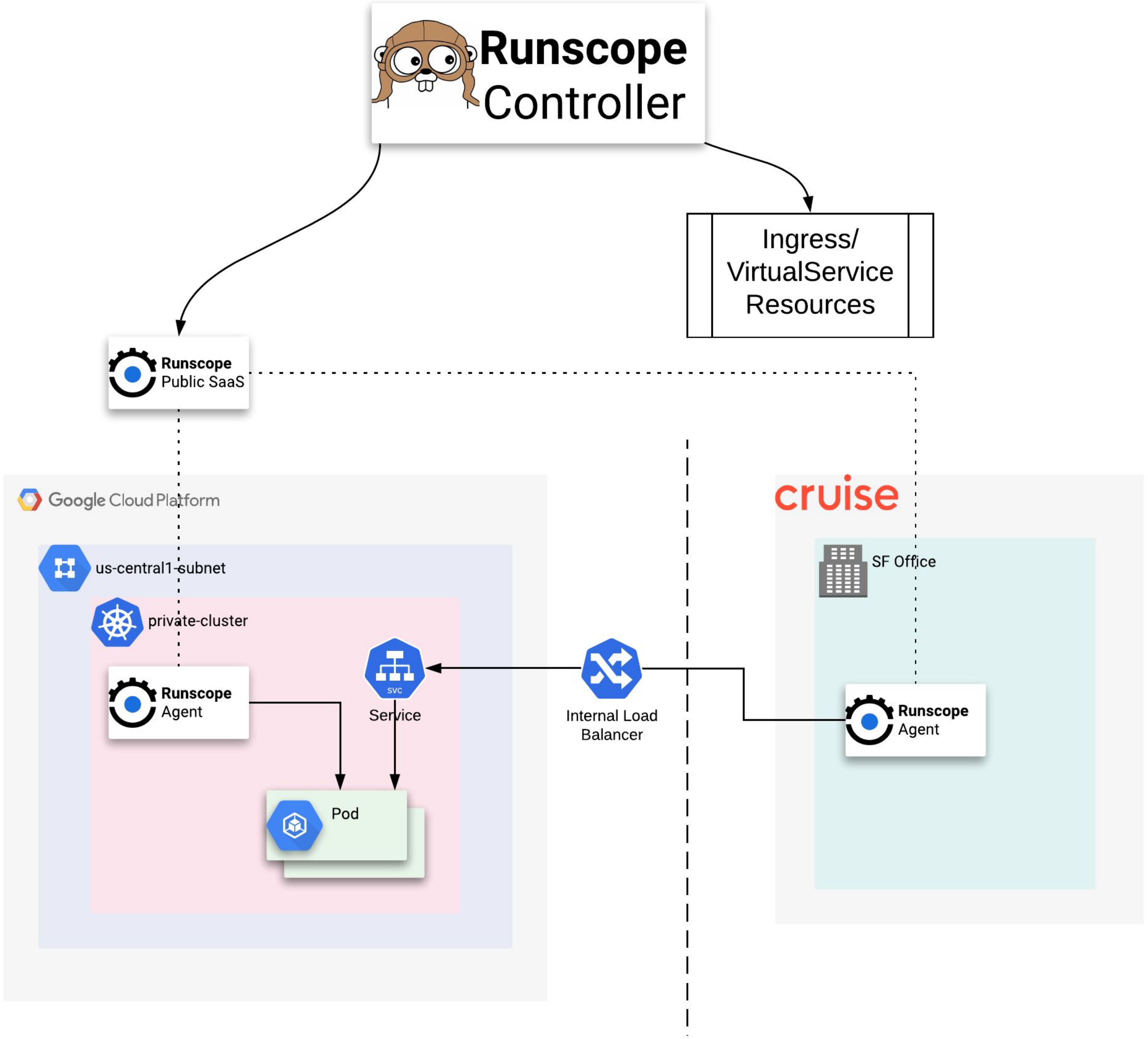
Provide easy access to network **logging** and **monitoring**



cruise

Ingress Monitoring

- **Motivation**: Experiencing elevated MTTR for ingress failures
- **Goals**:
    - Automate blackbox monitoring for *Ingresses*
    - Probe both private/public endpoints
    - Helps tenants identify problems early on



cruise

Ingress monitoring

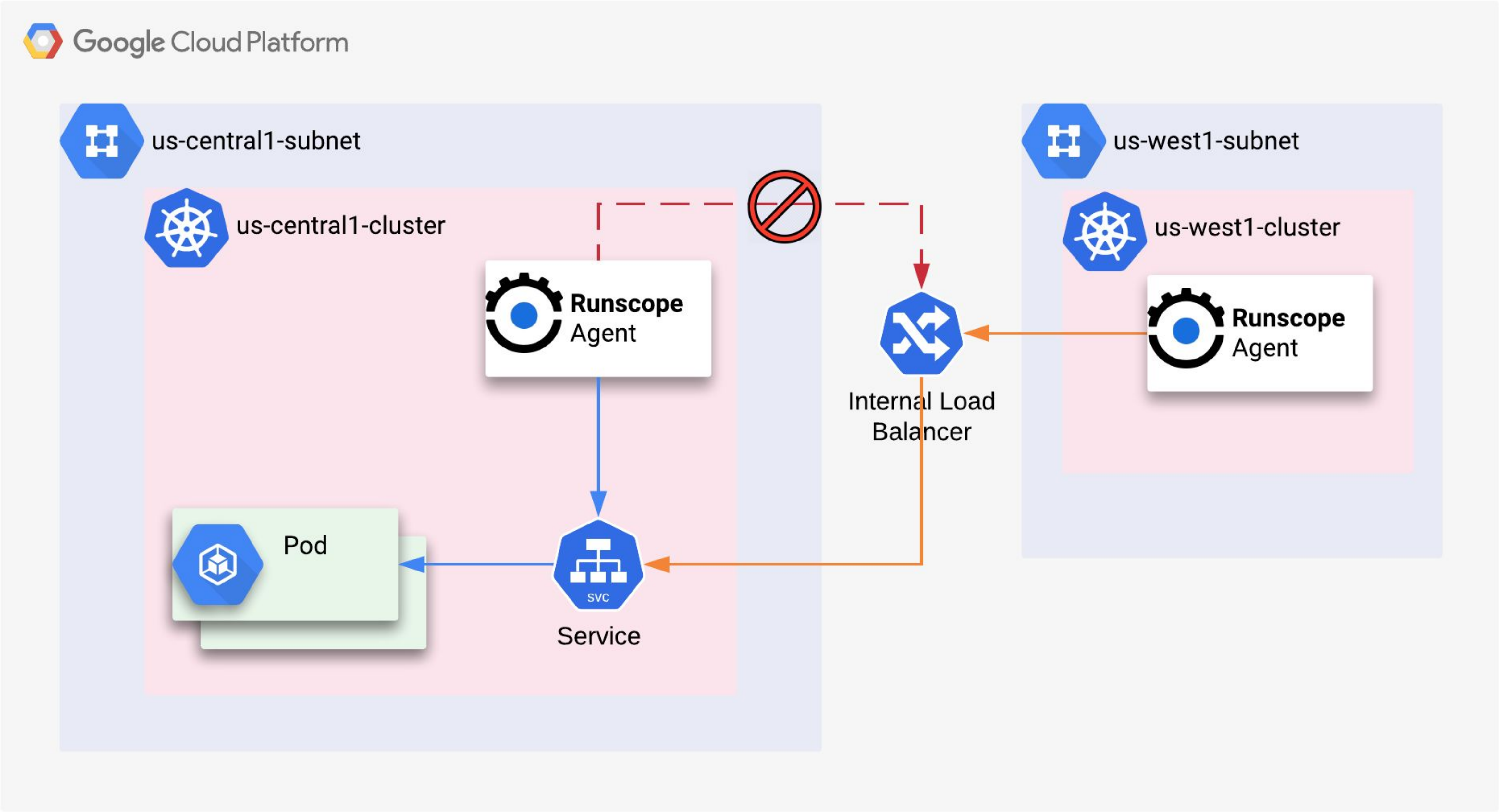
- Toggle Runscope tests
- Enable Runscope agents via environments
- Set an interval to run the tests
- Define a prefix

```yaml
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  annotations:
    kubernetes.io/ingress.allow-http: "false"
    kubernetes.io/ingress.class: public-nginx
    runscope.getcruise.com/bucket-name: paas-system
    runscope.getcruise.com/enable-api-tests: "true"
    runscope.getcruise.com/parent-environment-id:
11111111-2222-3333-4444-555555555555
    runscope.getcruise.com/path: /
    runscope.getcruise.com/schedule: 1m
    runscope.getcruise.com/test-prefix: '[P4]'
```

Lesson learned:

Requests directed to LoadBalancers created via services are still internal to the cluster

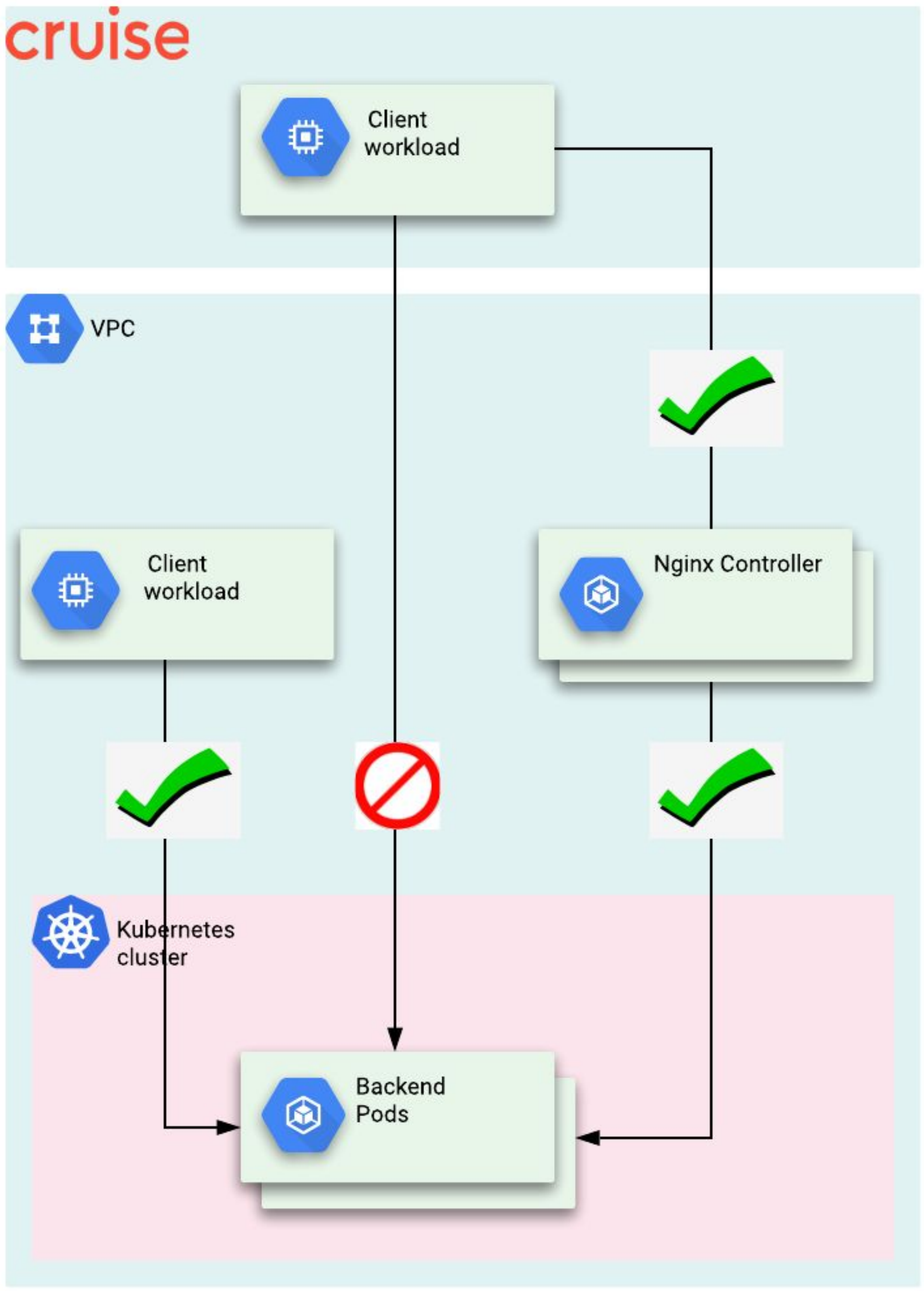
Better to have additional agents running outside of the cluster



cruise

Default network isolation

- AuthN//AuthZ for everything
- Full access to cluster node from inside the VPC
- No access from outside the VPC to cluster nodes (exceptions possible)
- TCP 80,443 from inside Cruise to Ingress Gateways
- Namespace isolation considered with **NetworkPolicies**



Pod network security:

K-Rail admission controller

https://github.com/cruise-automation/k-rail

- Host network pods forbidden
- Extra network capabilities forbidden
- Public ingress requires whitelisting

- User friendly output
- Add exemptions as required

```
$ kubectl apply -f deploy/non-compliant-deployment.yaml

Error from server (k-rail): error when creating "deploy/non-compliant-deployment.yaml": admission webhook
"k-rail.cruise-automation.github.com" denied the request:

Deployment bad-deployment had violation: No Host Network: Using the host network is forbidden
Deployment bad-deployment had violation: No Privileged Container: Using privileged containers is forbidden
Deployment bad-deployment had violation: No New Capabilities: Adding additional capabilities is forbidden
```

cruise

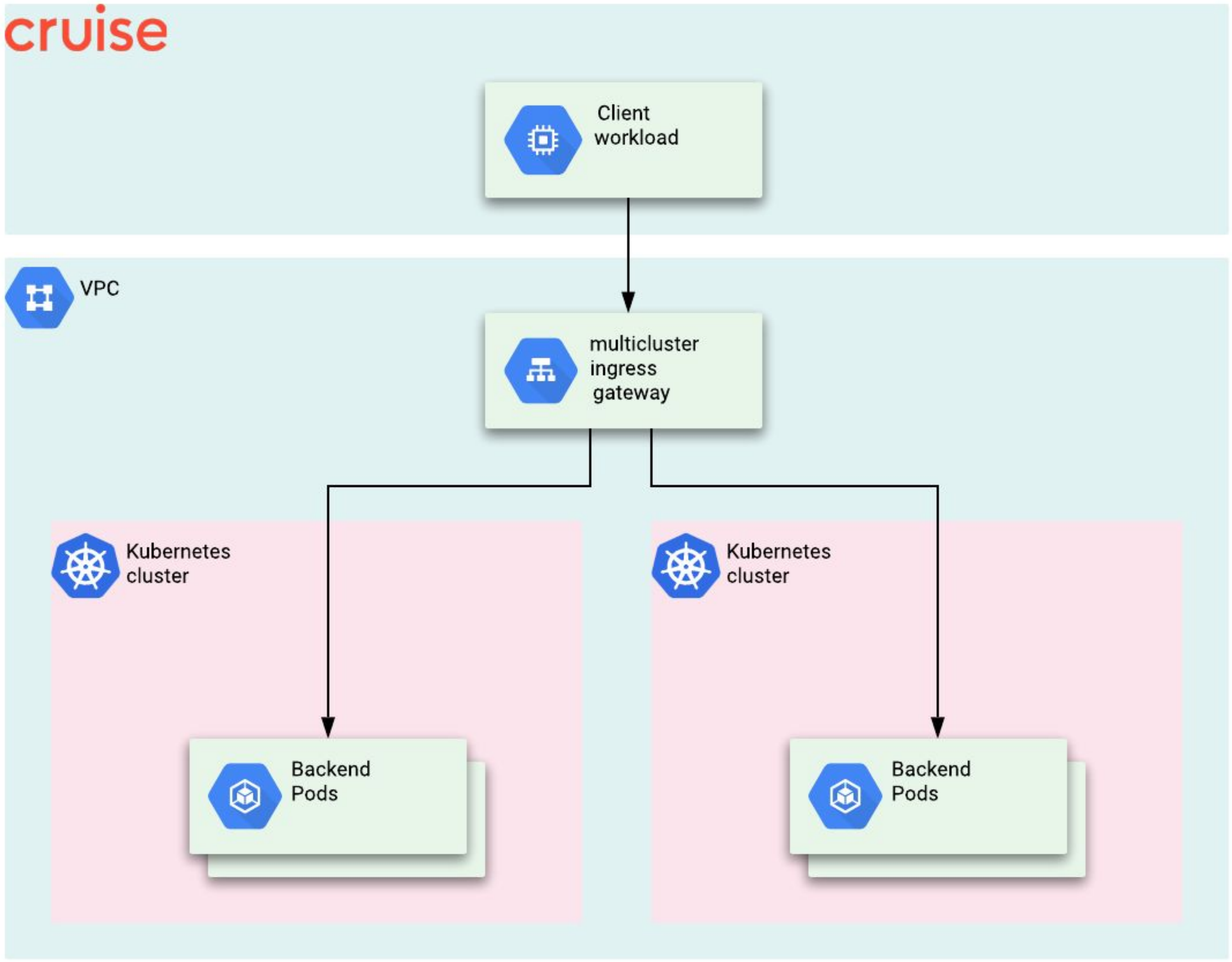
Current Challenges

cruise

Current challenges

Multi cluster ingress

- Single “federated” endpoint for all clusters

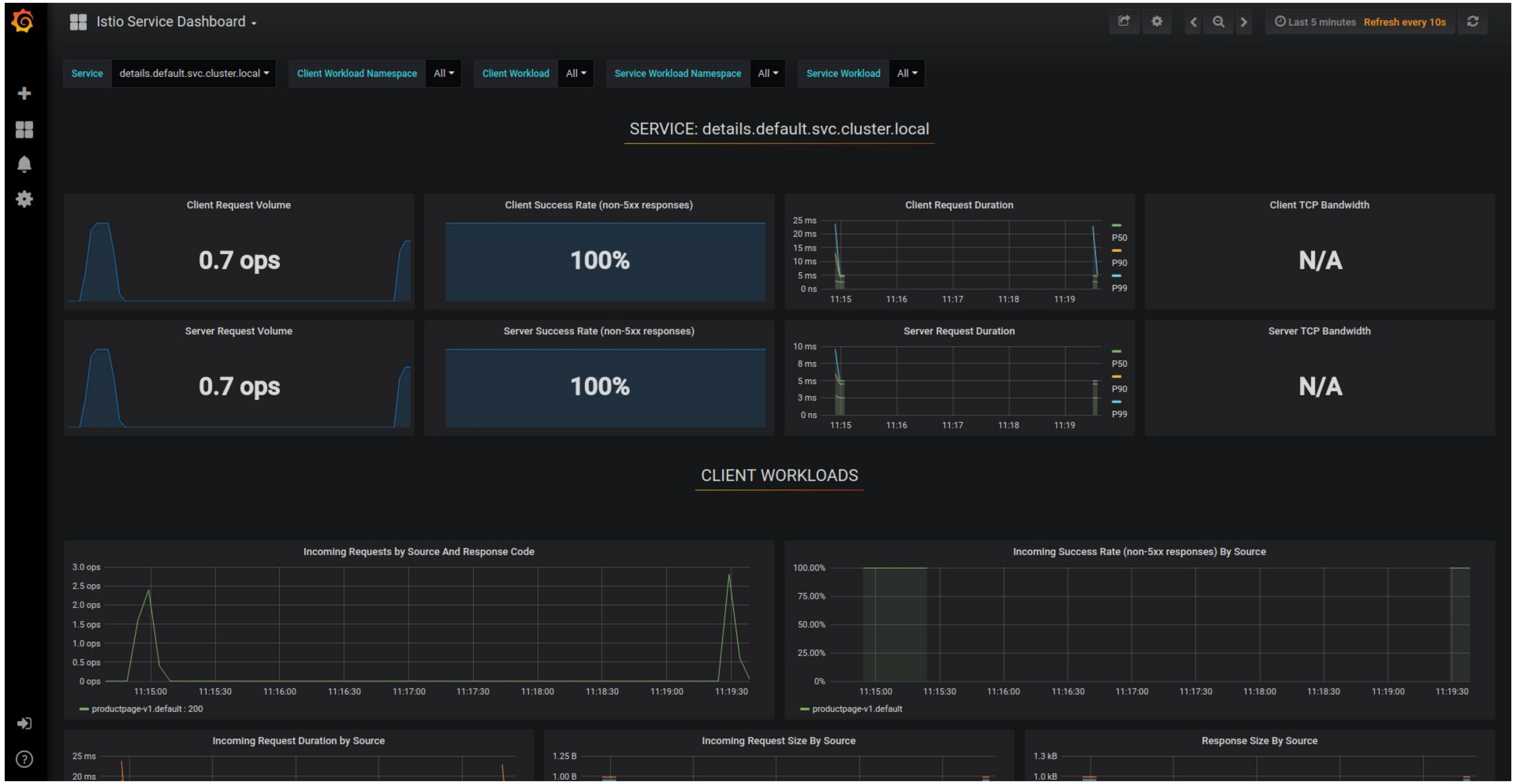


cruise

Current challenges

Improved Visibility and Metrics

- Identifying networking traffic characteristics on multi-tenant clusters



Current challenges

Network traffic engineering and QoS

DNS Enhancements

Load Testing Framework

Questions

@bvandewa

@canthefason