KubeCon CloudNativeCon North America 2026

#KubeCon #CloudNativeCon

From Packets to Paychecks

eBPF-Based Network Cost Attribution at Scale

Aanchal Khandelwal · **Brian Ugie**
Adobe

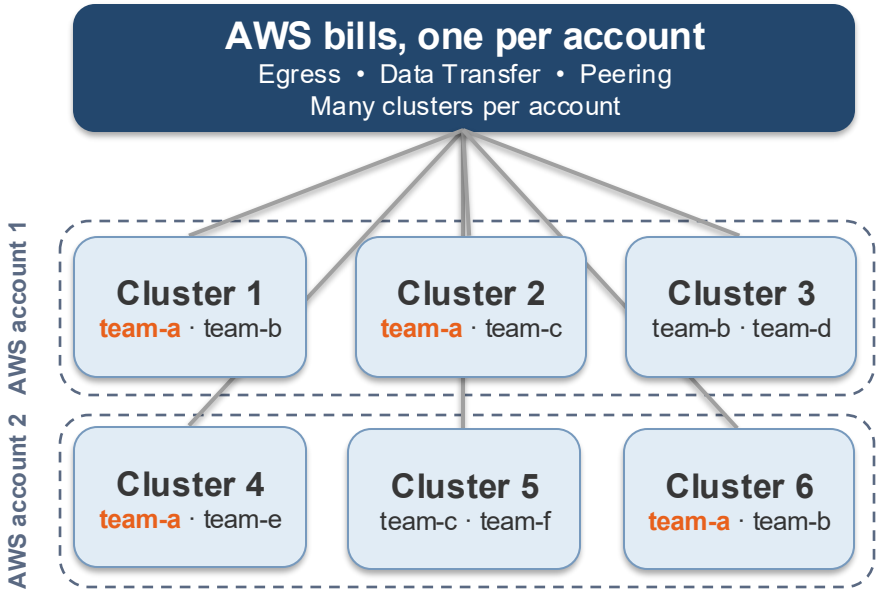


Many bills.
Thousands of tenants.
No obvious owner.



Shared Clusters, Unclear Accountability

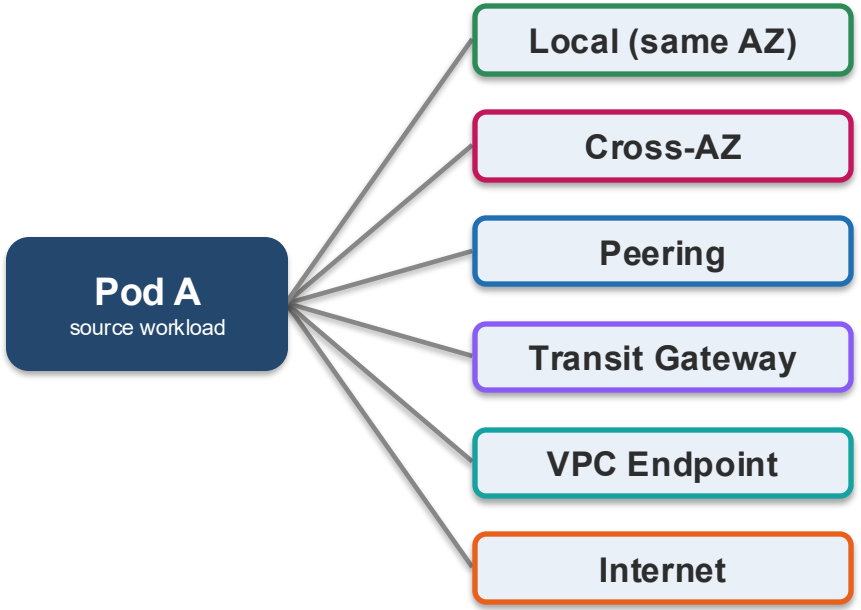
- In shared clusters, network cost is real but ownership is blurred
- Many clusters share one AWS account, so each bill mixes many clusters
- A team's namespaces run across many clusters and accounts
- Chargeback only works if teams trust the attribution



The bill is per account. Ownership is per team. Nothing lines up by default.

Seeing traffic is not the same as pricing it

- Raw flow data doesn’t say how the cloud bills it
- Local, cross-AZ, peering, transit gateway, endpoint and internet traffic are all priced differently
- Pricing needs topology and cloud context, not just visibility



Same source traffic, different billing outcomes

Where Hubble helped …and where it fell short

Where Hubble helped

- Flow visibility we already had with Cilium
- Great for who-talks-to-whom and debugging

Where it fell short for us

- Egress only, no ingress view
- Only source namespace and destination IP per flow, not both ends
- Flow counts, not bytes per flow

Splitting cost by flow count charges the wrong team (illustrative)

	Flows	Bytes sent	Share by flow count
ns-a	900	1 GB	**90%**
ns-b	100	49 GB	**10%**

Chargeback needs more than connectivity — it needs ownership and volume.

What the system needed to get right

Volume	Bytes per flow, not flow counts
Ownership	Namespace and IP on both ends, ingress and egress
Billing truth	Map every flow to how AWS actually prices it
Explainability	Every cost traces back to a flow and a route
Fleet scale	85+ clusters, 30K+ namespaces, running in production

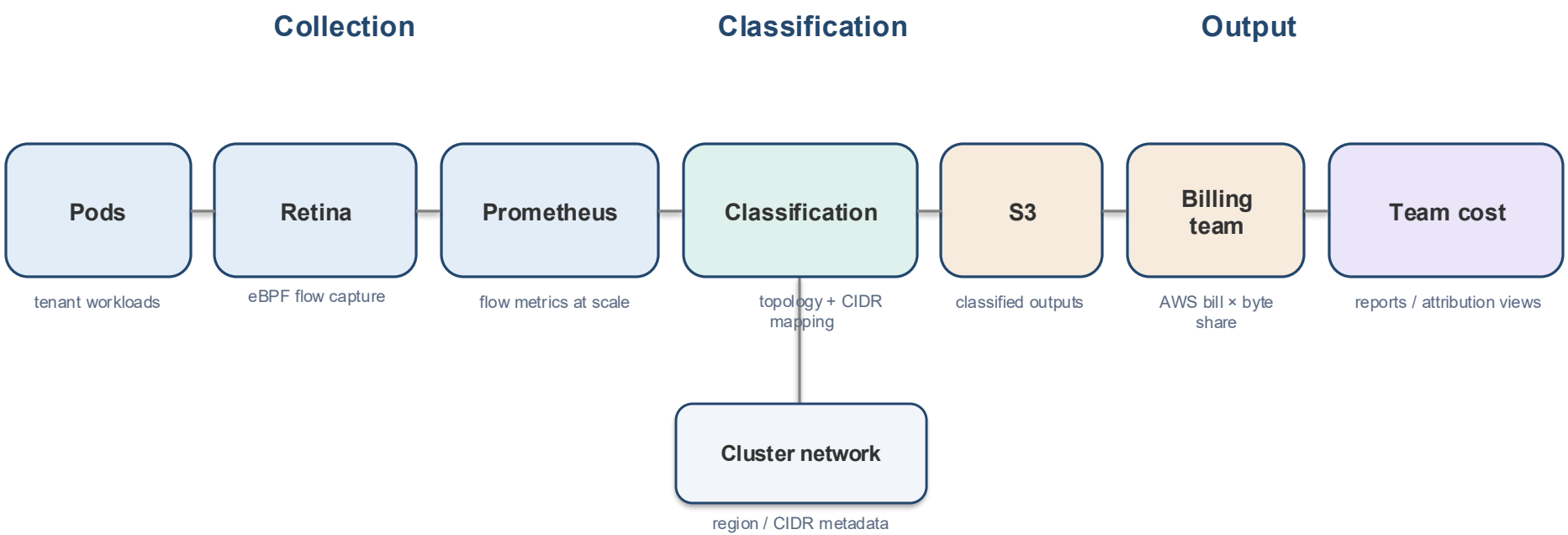
Every requirement traces back to a gap we hit.

KubeCon CloudNativeCon
North America 2026

Architecture of the network cost attribution pipeline



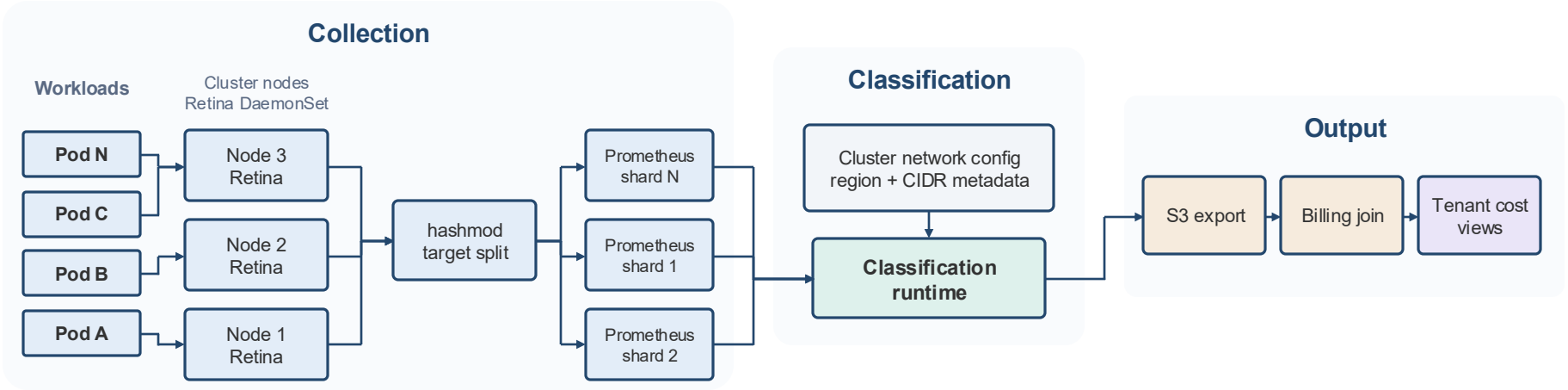
The pipeline at a glance



Collect flows → enrich and classify → export bytes per namespace → billing team prices them

High-Level Architecture

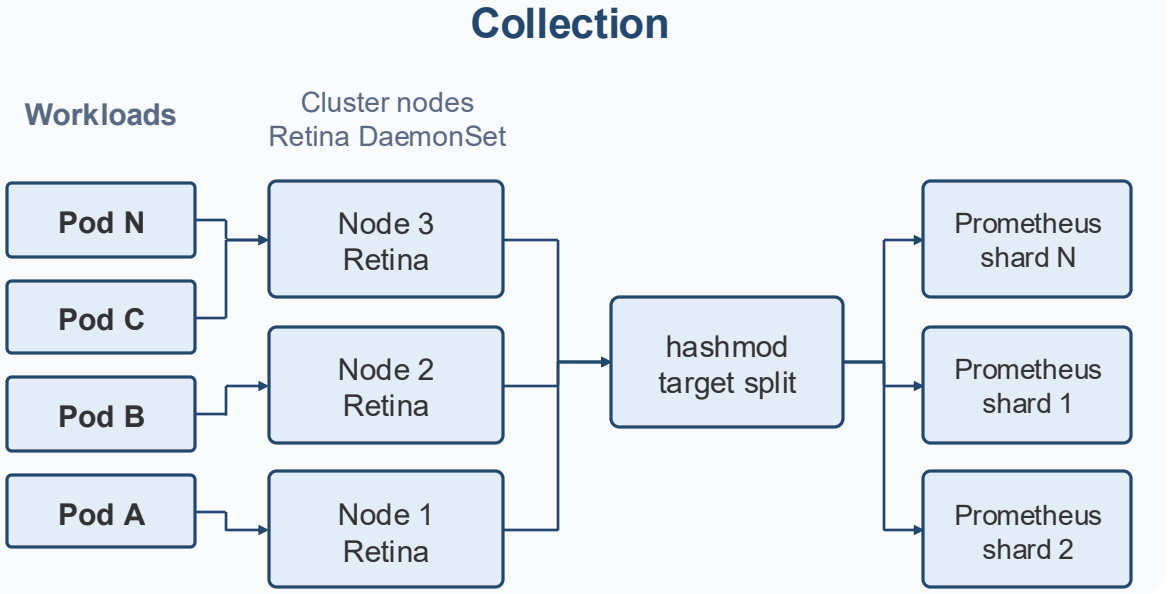
COLLECTION / CLASSIFICATION / OUTPUT



Collect flows → enrich and classify → reconcile with billing → present tenant views

Architecture: Collection

COLLECTION / DETAIL VIEW



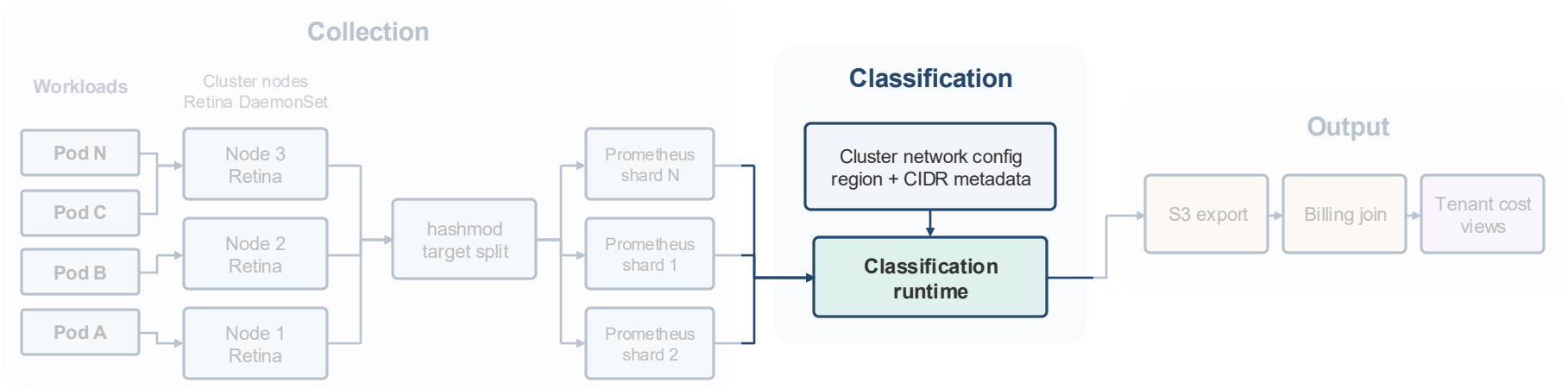
What runs here

- Retina Standard control plane, **packetparser** plugin only
- One advanced metric: `networkobservability_adv_forward_bytes`
- hashmod sharding: each shard scrapes 1/N of the nodes, every 1m, keeps 2h
- 1 to 25 shards per cluster. The smallest run a single, legacy Prometheus
- Recording rule on every shard, every 3 min

Every node runs Retina → each Prometheus shard scrapes its own slice of nodes

High-Level Architecture

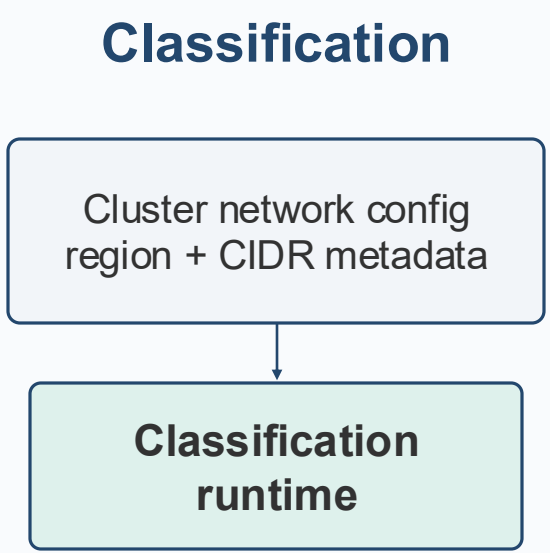
COLLECTION / CLASSIFICATION / OUTPUT



Collect flows → enrich and classify → reconcile with billing → present tenant views

Architecture: Classification

CLASSIFICATION / DETAIL VIEW

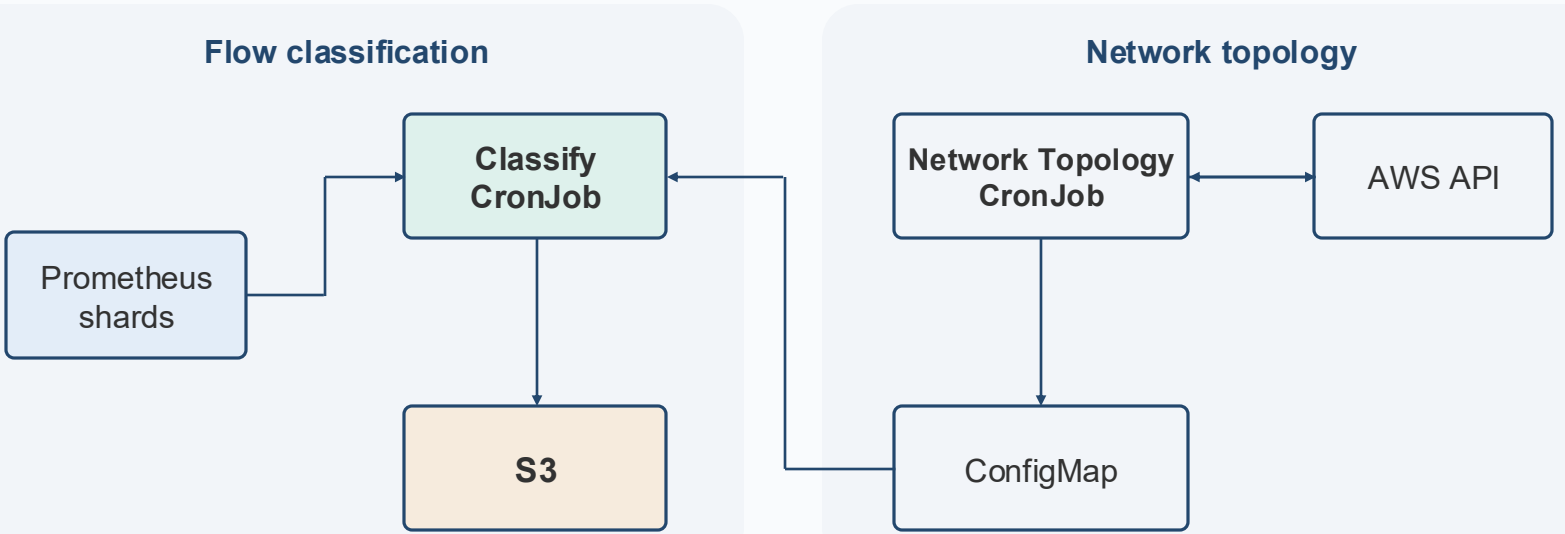


How a flow gets a category

- Topology CronJob snapshots the VPC's route tables and subnets from the AWS API into a ConfigMap
- Every 15 min, the classifier pulls the recording rule's output from the Prometheus shards
- Destination IP → most specific matching CIDR (ConfigMap + public IP ranges)
- The route target decides the category (full mapping coming up)
- Subnet AZ and region add cross-AZ / cross-region modifiers
- Anything we can't classify goes into General

Cluster CIDR and region metadata turn raw flow bytes into billing categories

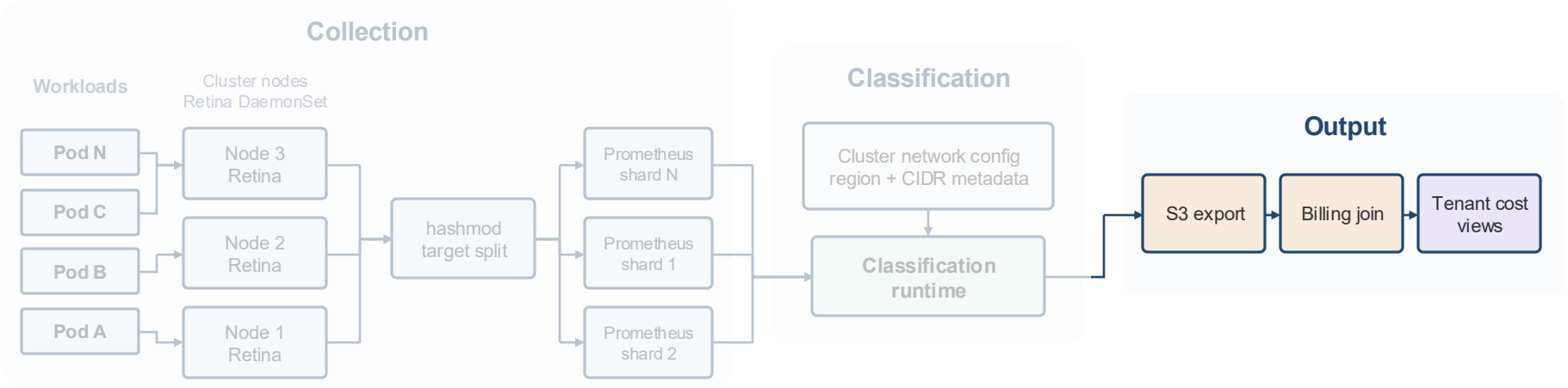
Classification: Inside the Runtime



Flow metrics + topology ConfigMap → Classify CronJob → Classified bytes in S3

High-Level Architecture

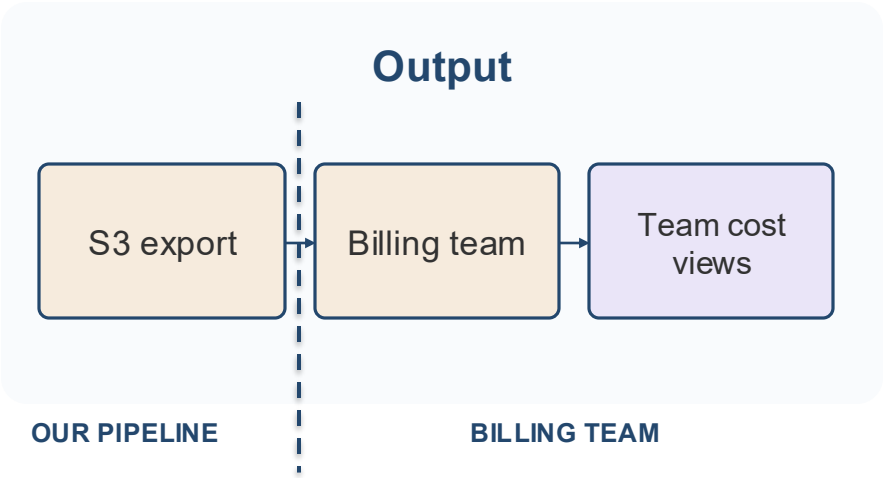
COLLECTION / CLASSIFICATION / OUTPUT



Collect flows → enrich and classify → reconcile with billing → present tenant views

Architecture: Output

OUTPUT / DETAIL VIEW



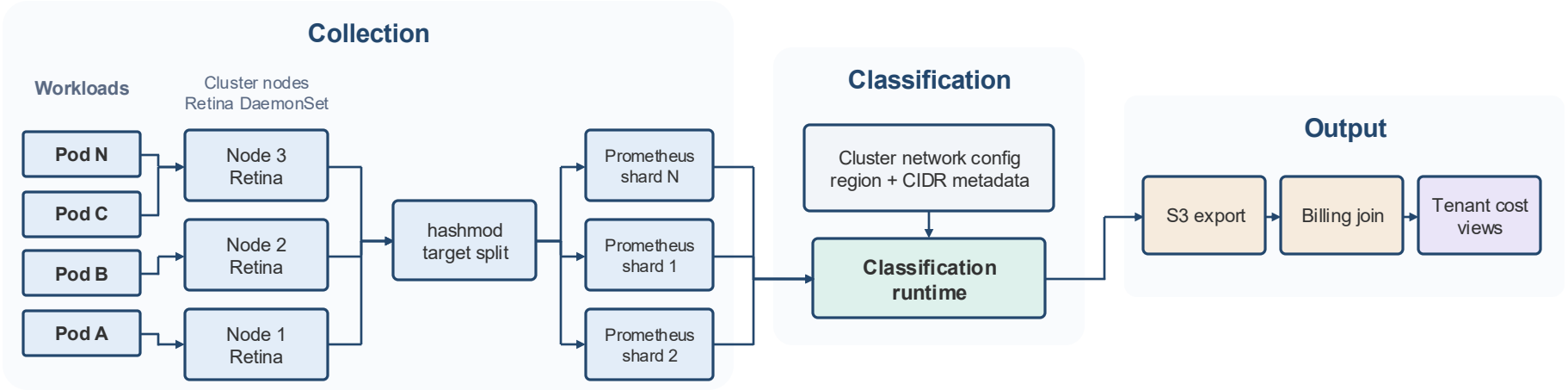
From bytes to a team's bill

- S3 holds classified bytes per cluster, namespace, and category
- Billing team takes the AWS bill per category and splits it by each namespace's byte share
- The same namespace can run in many clusters, so bytes roll up cluster → namespace → team

Our pipeline stops at classified bytes. The billing team turns them into dollars.

High-Level Architecture

COLLECTION / CLASSIFICATION / OUTPUT



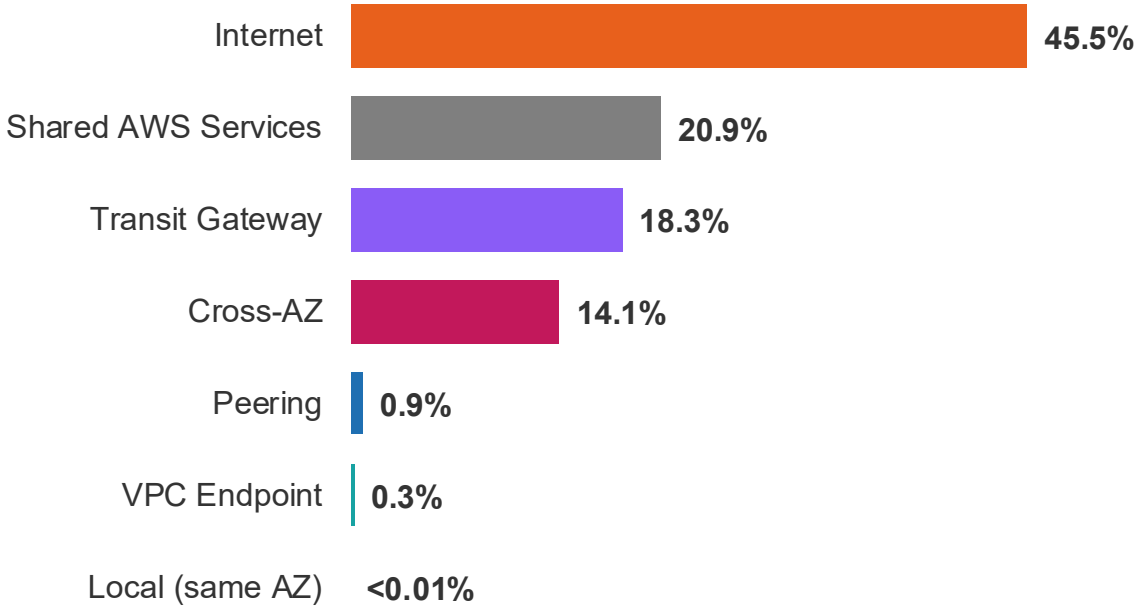
Collect flows → enrich and classify → reconcile with billing → present tenant views

The categories behind cost attribution

Category	From our flows (route target)	From the AWS bill (usage type)
Internet	NAT / internet gateway, public IP ranges	NAT Gateway, internet data transfer
Transit Gateway	Transit Gateway route	Transit Gateway usage
Peering	VPC peering route	Peering usage
VPC Endpoint	Prefix-list route	Endpoint usage
Cross-AZ	Source and destination in different AZs	Inter-AZ data transfer
Local	Own subnet, same AZ	In/Out bytes
General	Traffic we can't classify	-

Flows and the AWS bill are sorted into the same categories, so bytes can be priced.

The categories behind cost attribution



78%

of spend is Internet, Transit Gateway and cross-AZ

~0%

of spend is same-AZ traffic

Share of one month of AWS network spend, as categorized by the billing team

Building it was one challenge
Running it at scale was another



The hardest part was scale

KubeCon CloudNativeCon North America 2026

Prometheus shards needed per cluster

1 unsharded	3–5 shards	7–10 shards	12–16 shards	18–25 shards
25 clusters	25 clusters	5 clusters	18 clusters	10 clusters

bigger clusters: more nodes, more traffic → more shards

~85
clusters running the pipeline

~630
Prometheus shards

~31 TiB
memory requested

~1M
active series per shard

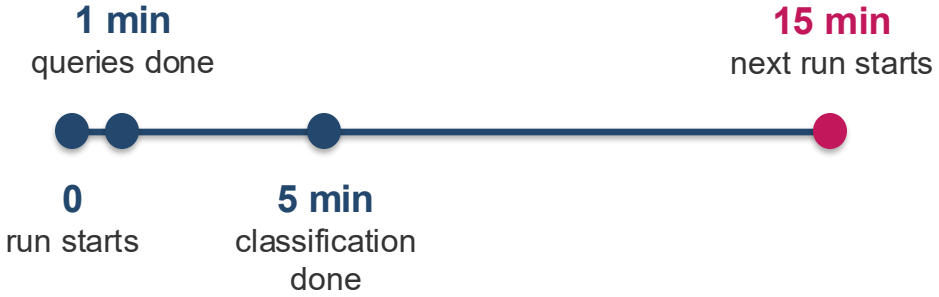
Useful attribution requires context. Context drives cardinality.

What broke, and the 15-minute budget

What broke

- Memory and CPU pressure on shards
- Queries timing out when a shard held too much data
- Shard counts and resources tuned cluster by cluster
- EBS volumes overwhelmed, even with 2h retention

The 15-minute budget



Slow Prometheus queries push classification past 15 minutes, and the next run starts before the last one finishes. Runs start queueing.

Prometheus health and classification deadlines turned out to be the same problem.

From one Prometheus to ~630 shards



One Prometheus per cluster

- 2 replicas, 3h retention, 20Gi disk
- Replicas add availability, not capacity: each holds every series

Shard by node

- hashmod: each shard scrapes its own slice of nodes
- Retention cut from 3h to 2h
- Alert when Prometheus disks pass 85%

Tune cluster by cluster

- ~50 per-cluster changes to shard counts, memory and disk
- Up to 25 shards, 30Gi / 45Gi per pod

~630 shards, ~85 clusters

- Stable enough to run in production
- Still tuned by hand as clusters grow

Sharding made it work. It didn't make it simple.

What this gives us today

What teams get

- Network cost per team, by category: Internet, Transit Gateway, cross-AZ…
- Explainable: every cost traces back to the bytes, the category, and the route that set it
- Shared AWS services (~21% of spend) are split equally across namespaces

What our pipeline hands to billing

namespace	category	direction	bytes
team-a-prod	Internet	egress	1.2 TB
team-a-prod	Cross-AZ	egress	310 GB
team-b-api	Peering	egress	45 GB
team-b-api	Local	egress	2.8 TB

One row per namespace, category and direction, every 15 minutes

Format is real, values are illustrative

From “the platform has a network bill” to “this team generated this traffic, and here’s why it cost what it did.”

This is not a victory lap

KubeCon CloudNativeCon North America 2026

Still open: hand-tuning Prometheus as clusters grow · finding a simpler metrics backend

Visibility isn’t attribution

You need bytes, both ends of the flow, and the route.

Copy your cloud’s routing logic

The most specific CIDR wins, just like in the VPC.

Same categories on both sides

Sort the bill and the flows the same way. That’s what makes the join work.

Plan for cardinality first

It will decide your architecture.

If you're solving this differently, let's compare notes.

References and contact

KubeCon CloudNativeCon North America 2026

Flow capture

Retina
retina.sh

Retina on GitHub
github.com/microsoft/retina

Retina advanced metrics
retina.sh/docs/Metrics/modes/advanced

packetparser plugin
retina.sh/docs/…/packetparser

Cilium Hubble
docs.cilium.io/…/observability/hubble

eBPF
ebpf.io

Metrics

Prometheus
prometheus.io

Recording rules
prometheus.io/docs/…/recording_rules

Sharding (background reading)
prometheus-operator.dev/…/sharding

AWS networking and pricing

VPC route tables
docs.aws.amazon.com › VPC › Route tables

Managed prefix lists
docs.aws.amazon.com › VPC › Prefix lists

Gateway endpoints
docs.aws.amazon.com › PrivateLink › Gateway

AWS IP address ranges
docs.aws.amazon.com › VPC › IP ranges

EC2 data transfer pricing
aws.amazon.com/ec2/pricing/on-demand

Let's compare notes

Brian Ugie linkedin.com/in/brian-ugie

Aanchal Khandelwal linkedin.com/in/aanchal-khandelwal

KubeCon CloudNativeCon North America 2026

#KubeCon #CloudNativeCon

Thank You!



KubeCon CloudNativeCon North America 2026

#KubeCon #CloudNativeCon

Q&A

From Packets to Paychecks: eBPF-Based Network Cost Attribution at Scale

