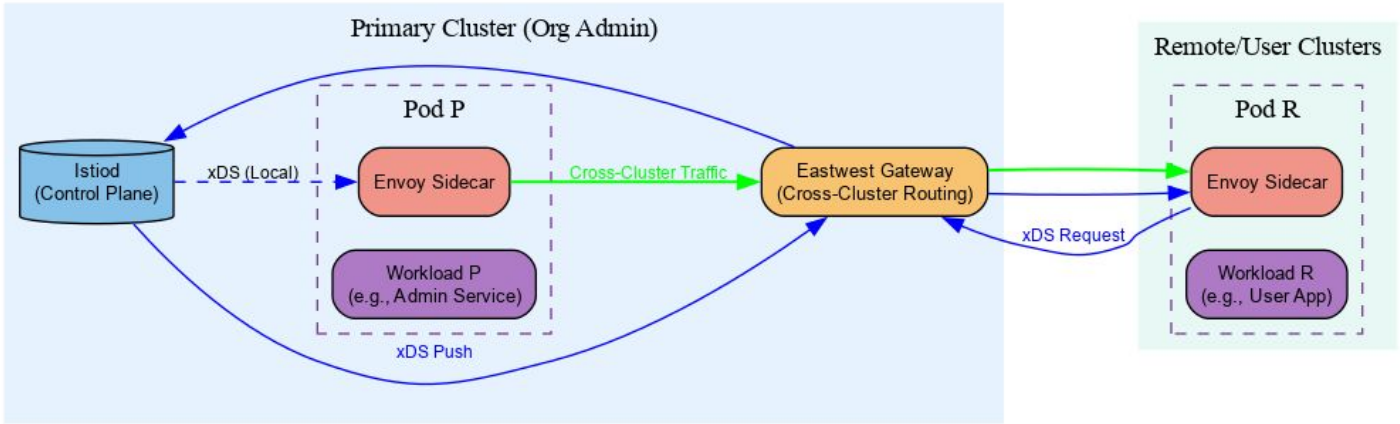
KubeCon CloudNativeCon India 2026

#KubeCon #CloudNativeCon

The Lean Observability Stack:
Quick and Native Telemetry for Service Mesh

Arpitha Malavalli
Google

Premise



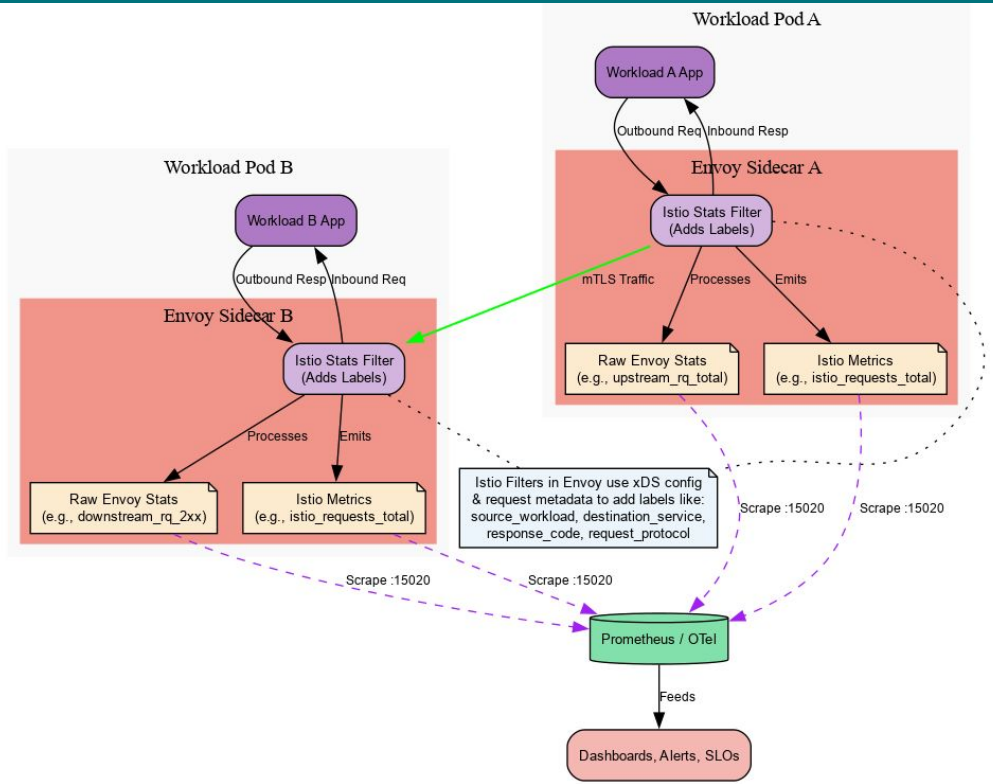
Setting
Isolated or air-gapped cloud environments with tight infrastructure and connectivity constraints.

Requirement
Leverage native mesh telemetry to the absolute best to build a reliable observability stack quickly.

Isolated Cloud Set-up

- Primary-Remote Topology : the Primary cluster (hosting the Istiod control plane), while User/System Clusters act as Remote clusters.
- Workloads can reside on both the Primary and Remote clusters.
- Workloads on the same cluster (e.g. within) communicate directly via mTLS.
- Workloads across different cluster (e.g., Primary Cluster to Remote Cluster) route through the Eastwest Gateway to traverse network boundaries securely.
- Istiod in the Primary cluster watches the API Servers of all clusters to provide unified service discovery.

Metric Generation and Aggregation

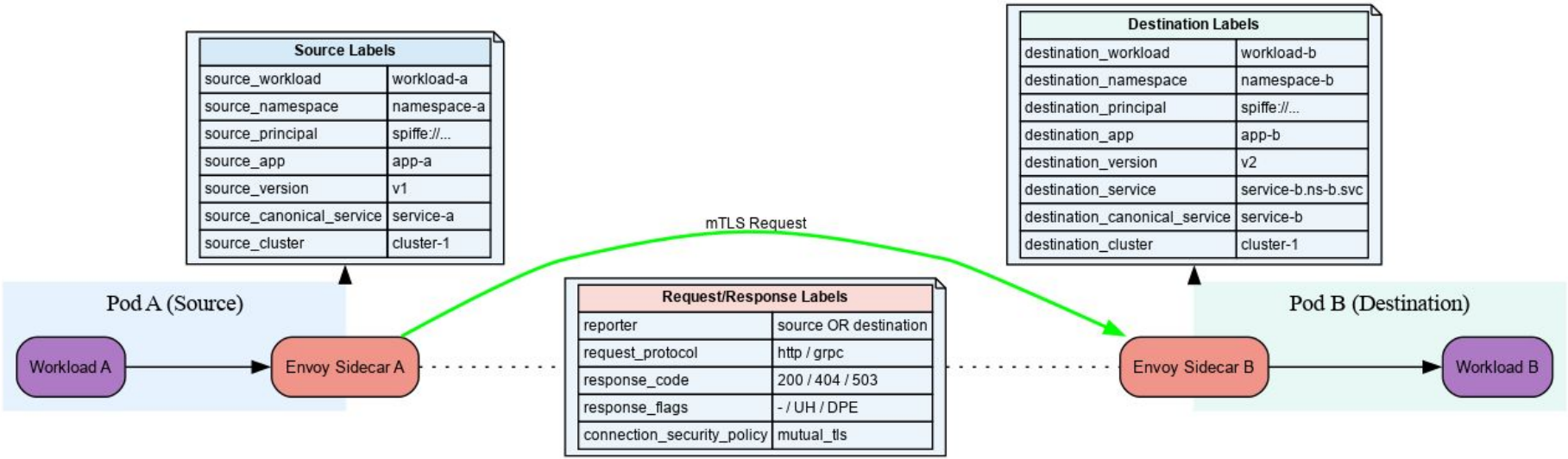


https://github.com/istio/proxy/blob/5443bbeced93b696771e58ba055830bde4a90d95/source/extensions/filters/http/istio_stats/istio_stats.cc#L184

- The "Istio Stats Filter" isn't a separate service—it is a natively compiled C++ extension running inside every pod's Envoy process.
- Completely bypasses central bottlenecks. Telemetry is fully distributed, zero-latency, and zero-compromise.

Istio Aggregated Metric	Raw Envoy Inputs
`istio_requests_total`	http.<conn_mngr>.downstream_rq_total, cluster.<svc>.upstream_rq_xx
`istio_request_duration_ms`	http.<conn_mngr>.downstream_rq_time
`istio_request_bytes`	StreamInfo::bytesReceived()
`istio_response_bytes`	StreamInfo::bytesSent()
`istio_tcp_conn_opened`	downstream_cx_total, upstream_cx_total
`istio_tcp_sent/recv`	downstream_cx_tx/rx_bytes_total

Every Request Tells a Story:
Deconstructing Istio Metric Labels

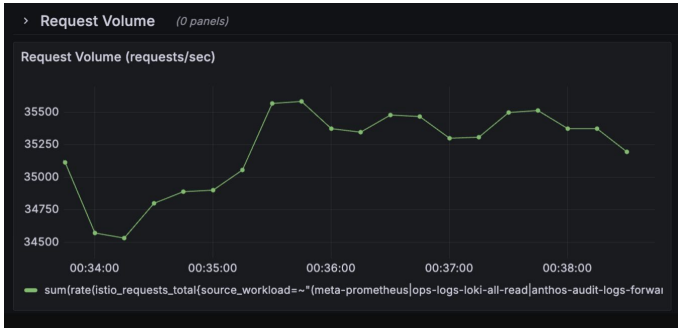


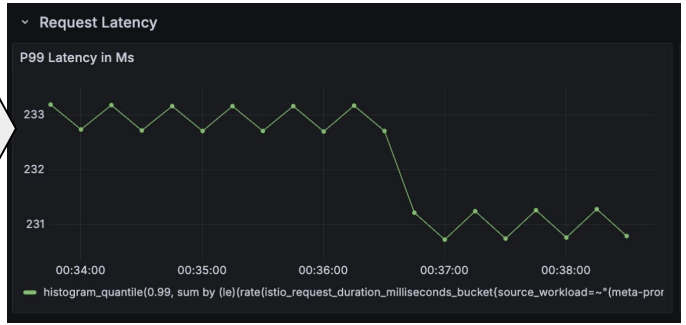
- Labels present for all istio aggregated metrics: **`istio_requests_total, istio_request_duration_ms, istio_request_bytes etc`**
- These labels allow for powerful filtering and grouping in Prometheus, making it easy to build expressions for Latency, Traffic, Errors, and Saturation sliced by any of these dimensions.

OOTB Alerts and Golden Signals

Prometheus Alerting Rules using Istio Labels:

```
groups:
- name: my-service.alerts
  rules:
  - alert: MyService_HighErrorRate_5xx
    expr: |
      sum(rate(istio_requests_total{
        reporter="destination",
        destination_service_name="my-service",
        destination_workload_namespace="app-ns",
        response_code=~"5.*"
      }[5m]))
      /
      sum(rate(istio_requests_total{
        reporter="destination",
        destination_service_name="my-service",
        destination_workload_namespace="app-ns"
      }[5m]))
      > 0.01
    for: 5m
    labels:
      severity: critical
      service: my-service
    annotations:
      summary: "High 5xx error rate for my-service"
```





Some Grafana dashboards built entirely from the **native Istio and Envoy metrics..** We can track Latency, Traffic and Errors for any service in the mesh, no custom code required.

OOTB Templateable SLI definitions

Success Rate / Performance :

```yaml
spec:
  successRate:
    - description: "my-service request success rate"
      goal: "0.999" # Target 99.9% availability
      period: 30d
      goodTotalRatio:
        good:
          metricName: istio_requests_total
          labelFilter:
            - label: destination_service_name
              value: my-service
            - label: destination_workload_namespace
              value: app-ns
            - label: reporter
              value: destination
            - label: response_code
              value: "[45].." # Regex: Not 4xx or 5xx
              matchOperator: regexNotMatch
        total:
          metricName: istio_requests_total
          labelFilter:
            - label: destination_service_name
              value: my-service
            - label: destination_workload_namespace
              value: app-ns
            - label: reporter
              value: destination
            - label: response_code
              value: "4.." # Exclude client errors from total
              matchOperator: regexNotMatch
```

Latency :

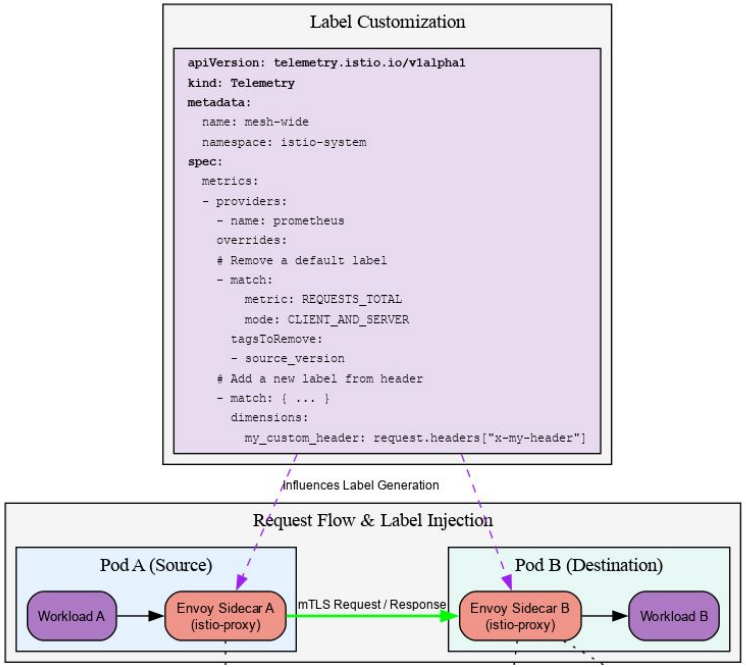
```yaml
kind: ServiceLevelObjective
metadata:
  namespace: app-ns
  name: my-service-latency-p99
spec:
  successRange:
    - description: "my-service P99 latency under 250ms"
      goal: "0.99" # Target P99
      period: 30d
      metricName: istio_request_duration_milliseconds_bucket # Use the histogram
      labelFilter:
        - label: destination_service_name
          value: my-service
        - label: destination_workload_namespace
          value: app-ns
        - label: reporter
          value: destination
      max: 250 # Upper bound for success in milliseconds
```

Universally Applicable: Apply to any service in the mesh by changing label values (e.g., destination_service_name).

No Application Changes: All required metrics and labels are provided by the Istio proxy running Envoy.

Template-able & Rich Filtering: Create SLO templates for easy onboarding; use Istio labels (source_workload, request_protocol, etc.) for granularity.

Want more labels? Want a more granular view?



- Many granular `envoy_*` metrics are not exposed to Prometheus by default.
- This is Istio's default to reduce CPU/memory overhead in large meshes.
- Configure Istio to expose additional Envoy statistics for deeper debugging and analysis.

Per-Workload via Annotations

```
metadata:
  annotations:
    sidecar.istio.io/statsInclusionPrefixes: "cluster.backend-svc,listener"
    sidecar.istio.io/statsInclusionRegexps: ".*http.ingress.*"
```

Mesh-Wide via IstioOperator (MeshConfig)

```
spec:
  meshConfig:
    defaultProviders:
      prometheus:
        envoyStats:
          inclusionPrefixes:
          - "cluster.outbound"
          # Or use regex
          # matchingRegexps:
          # - ".*cluster\.outbound.*"
```

Handling Cardinality & Metrics Explosion

In the Prometheus config, **<scrape_config>** defines targets. Use **<metric_relabel_configs>** for fine-grained control over scraped metrics.

Relabeling Strategies

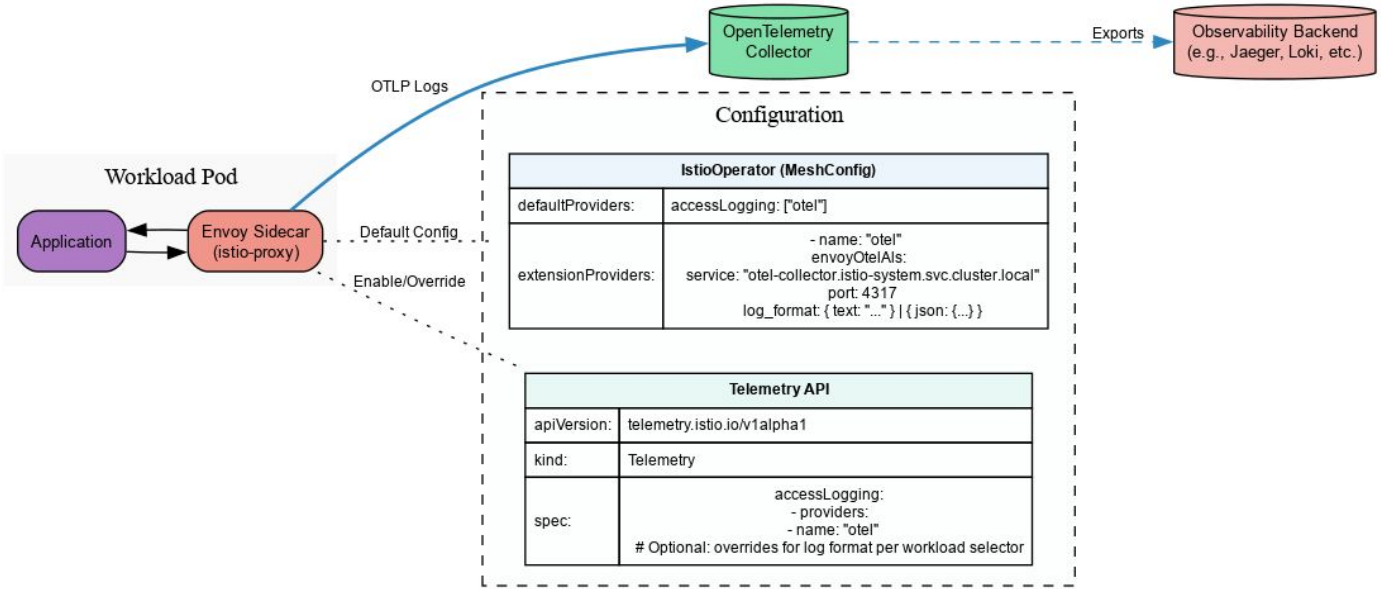
- **action: keep** — White-list specific Istio/Envoy metrics (e.g., `istio_.*`) via regex.
- **action: drop** — Remove noisy or redundant metrics to save memory.
- **action: labeldrop** — Prune high-cardinality labels (like `pod_ip`) from otherwise useful metrics.
- **Value Rewriting** — Aggregate unique identifiers (IDs/IPs) into generic labels.
- **Filtering** — Exclude specific time series based on label values.

```
ape_configs:
 job_name: 'istio-proxies'
 kubernetes_sd_configs:
   - role: pod
 relabel_configs:
   # Target pods with istio-proxy container
   - source_labels: [__meta_kubernetes_pod_container_name]
     action: keep
     regex: 'istio-proxy'
   # Scrape the envoy metrics port
   - source_labels: [__meta_kubernetes_pod_container_port_numb
     action: keep
     regex: '15020'
 metric_relabel_configs:
   # --- CARDINALITY REDUCTION / ALLOWLISTING ---
   # Example: Only keep metrics with 'istio_' prefix
   - source_labels: [__name__]
     action: keep
     regex: 'istio_.*'
   # Example: Drop a specific high-cardinality label
   - action: labeldrop
     regex: 'pod_ip_address'
```

Reference: Prometheus Scrape Configuration Guide

Prometheus Adapter?

Proxy Logs and Otel Collector



- The Envoy sidecar, as it processes requests, generates access log entries as well
- Instead of writing to standard output, Envoy can be configured to use an OpenTelemetry Protocol (OTLP) exporter. This sends structured logs over gRPC.
- A dedicated OpenTelemetry Collector service receives these logs. The collector can buffer, process, and export logs to various backends.


KubeCon
CloudNativeCon
India 2026
Thank You!