SRE · Observability · Kubernetes · Prometheus · Multi-Region · Platform Engineering

Federated Prometheus at Scale: How We Reduced MTTD by 60% Across a Multi-Tenant EKS Platform

Chinmaya Kumar Mishra

Principal Platform Engineer · Cloud Architect · CKA · AWS Solutions Architect Associate
Siemens Smart Infrastructure, Pune — linkedin.com/in/chinmaya-mishra-2ab99014

When you run a single Prometheus instance scraping 60+ microservices across 15 engineering teams, three things happen in short order: cardinality explodes, scrape intervals become unreliable, and your on-call team spends more time correlating dashboards than fixing incidents. We hit all three within six months of going multi-tenant on EKS.

This is the story of how we built this federated architecture — what we got right, what broke in production, and the specific decisions that produced a 60% MTTD reduction across the platform.

The Problem With a Single Prometheus Instance at Scale

A single Prometheus instance works beautifully up to a point — roughly 1 million active time series, or 5–6 teams generating metrics independently, or when SLOs span services owned by different teams. We hit all three simultaneously.

Cardinality explosion was the first signal. Our series count crossed 800k as teams added debugging labels — `user_id`, `request_id`, `feature_flag` — that should never be on metrics. One misbehaving team could destabilise the entire observability stack.

Scrape reliability degraded next. With 60+ services and thousands of pods, a 15s global scrape interval became aspirational. Prometheus was spending more time scraping than evaluating rules.

Cross-team correlation was impossible. When Service A fired and the root cause was a downstream degradation in Service B (different team), engineers toggled dashboards manually. Our P95 MTTD was above 45 minutes on complex incidents.

> *The solution was not to scale the Prometheus instance vertically. It was to redesign the architecture.*

The Federated Monitoring Architecture

This is a three-layer federated architecture deployed on Fargate EKS, accessible across three environments and two geographic regions, both protected via Auth0 authentication:

Environment	Region	Prometheus	Alertmanager
DEV	EU	prometheus.observe.dev.example.com	alertmanager.observe.dev.example.com
INT	EU	prometheus.observe.int.example.com	alertmanager.observe.int.example.com
PROD	EU	prometheus.observe.prod.example.com	alertmanager.observe.prod.example.com
PROD	US	prometheus.us.observe.prod.example.com	alertmanager.us.observe.prod.example.com

Layer 1 — Service-Level Prometheus

Every team gets a Prometheus instance scoped to their namespace — 6-hour retention, 15s scrape interval, no alerting. If a team adds a high-cardinality label, it affects only their instance. Blast radius is contained.

For services without a Prometheus endpoint, the Blackbox Exporter handles HTTP uptime monitoring. Any PROD service returning non-2xx status for more than 300 seconds triggers an ops alert. This is mandatory for all PROD services with public endpoints.

```
# ServiceMonitor with label governance enforced by Kyverno
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
```

```
  name: building-automation-api
  namespace: building-automation
spec:
  endpoints:
    - port: metrics
      interval: 15s
      relabelings:
        - action: labeldrop
          regex: (request_id|trace_id|pod_ip)
```

Layer 2 — Domain Federation

Recording rules are defined exclusively at Layer 2. Pre-computing aggregations at the domain layer means the Global Prometheus only ever receives pre-aggregated metrics — cardinality stays minimal.

```
# Domain federation scrape config
- job_name: 'federate-building-automation'
  scrape_interval: 30s
  honor_labels: true
  metrics_path: '/federate'
  params:
    'match[]':
      - '{job="building-automation-api"}'
      - 'up{namespace="building-automation"}'
  static_configs:
    - targets: ['prometheus-ba.building-automation.svc:9090']

# Recording rules — computed at domain, consumed at global
groups:
  - name: building_automation.slo
    rules:
      - record: job:error_ratio5m:sum
        expr: |
          sum(rate(http_requests_total{code=~"5.."}[5m])) by (job)
          / sum(rate(http_requests_total[5m])) by (job)
```

Layer 3 — Central Prometheus and Private Link Federation

The Central Prometheus scrapes only pre-aggregated metrics from Domain instances. Federation from dedicated EKS clusters uses AWS PrivateLink — teams provide their VPC endpoint service IDs and the CI pipeline generates the federation targets automatically:

```
# federated-verticals.yml (dynamically generated by CI pipeline)
- job_name: federated-verticals
  honor_labels: true
  metrics_path: /federate
  params:
    match[]:
      - '{__name__=~"job:.*"}'    # Only pre-aggregated recording rules
  static_configs:
    - targets:
        - vpce-0abc123def456789.execute-api.eu-west-1.vpce.amazonaws.com
```

A `configmap-reload` sidecar monitors the mounted ConfigMap for changes and triggers Prometheus reload via POST to `/-/reload` — eliminating redeployments for every config change.

Team-Specific Configuration: Single Source of Truth

The most significant operational improvement in our federated monitoring platform was introducing team-specific YAML files as the single source of truth. Previously, teams modified multiple scattered config files across different directories, creating duplication and conflict risk.

Now each team maintains exactly one file: `config/team-config/<team-name>.yml`. The CI pipeline reads it and generates all region-specific outputs:

```
# config/team-config/building-automation.yml
common:
  scrape_configs:
    - job_name: building-automation-api
      static_configs:
        - targets: ['api.building-automation.svc:8080']

  alert_rules:
    - alert: BuildingAutomationHighErrorRate
      expr: job:error_ratio5m:sum{job="building-automation-api"} > 0.05
      for: 5m
      labels:
        severity: warning

  blackbox_targets:
    - https://api.building-automation.prod.example.com/health

region_specific:
  eu-west-1:
```

```
    federation:
      vpc_endpoint_service_id: vpce-svc-0abc123def456789
```

The CI pipeline generates all region-specific files automatically:

```
config/
  team-config/
    building-automation.yml        ← team edits only this
  prod/
    eu-west-1/
      prometheus/prometheus.yml          ← generated by CI
      prometheus/blackbox-route-targets.yml  ← generated by CI
      alertmanager/alertmanager.yml      ← generated by CI
    us-east-1/
      prometheus/prometheus.yml          ← generated by CI
```

Region-specific values always take precedence over common values. Duplicate `job_name` entries are rejected at CI validation time. Teams never touch region configuration directly.

Multi-Region Expansion

our federated monitoring platform initially supported only EU. When we expanded to US (`prod.us-east-1`), the old architecture would have required manually duplicating every configuration file. The team-specific model absorbed the expansion with a single-file change:

```
# regional-accounts.json — add one line to enable a new region
{
  "dev.eu-west-1":  "111111111111",
  "int.eu-west-1":  "222222222222",
  "prod.eu-west-1": "333333333333",
  "prod.us-east-1": "333333333333"   ← this is all it took
}
```

The CI pipeline checks this mapping, and if found, generates and deploys all region-specific configurations automatically. Teams made zero changes to their own YAML files.

SLO Design: Multi-Window Burn-Rate Alerting

The most impactful improvement was not the federation topology — it was replacing threshold-based alerting with multi-window burn-rate alerting for SLOs. Threshold alerts fire during routine traffic dips and miss slow burns. Burn-rate alerting measures how fast you're consuming your error budget.

For a 99.9% SLO over 30 days, a burn rate of 14.4 exhausts the budget in ~50 hours. We use the Google SRE Workbook multi-window approach:

```
groups:
  - name: slo.building_automation_api
    rules:
      # Fast burn: exhausts budget in ~1 hour
      - alert: SLOErrorBudgetBurnFast
        expr: |
          job:error_ratio5m:sum > (14.4 * 0.001)
          and
          job:error_ratio1h:sum  > (14.4 * 0.001)
        for: 2m
        labels:
          severity: critical
        annotations:
          description: "Budget exhausted in ~1 hour at current rate"

      # Slow burn: exhausts budget in ~3 days
      - alert: SLOErrorBudgetBurnSlow
        expr: |
          job:error_ratio30m:sum > (3 * 0.001)
          and
          job:error_ratio6h:sum  > (3 * 0.001)
        for: 15m
        labels:
          severity: warning
```

Alerts labelled `severity: critical` route to both Microsoft Teams and PagerDuty on PROD. Everything else routes to Teams only.

Alertmanager: Microsoft Teams v2 Integration

We migrated from the legacy Teams webhook connector to `msteamsv2_configs` using Power Automate workflows. Each team creates a Power Automate workflow using the "Send webhook alerts to a channel" template, gets a webhook URL, and adds it to their config:

```
# In team-specific config
alertmanager_config:
  receivers:
    - name: building-automation-alerts
      msteamsv2_configs:
```

```
        - webhook_url: https://prod-12.westeurope.logic.azure.com/...
          send_resolved: true
```

Each team controls their own Power Automate workflow independently — they can customise the Teams channel, card format, and routing without touching the central Alertmanager configuration. This is the right level of ownership: platform provides the plumbing, teams control the routing.

Cardinality Governance

Federation solves aggregation but introduces risk: label proliferation cascades upward if not governed. Three-layer control strategy:

1. Kyverno Admission Control at the Source

```
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: deny-high-cardinality-labels
spec:
  validationFailureAction: Enforce
  rules:
    - name: check-servicemonitor-relabelings
      match:
        resources:
          kinds: [ServiceMonitor]
      validate:
        message: "ServiceMonitor must drop high-cardinality labels via relabelings."
        pattern:
          spec:
            endpoints:
              - relabelings:
                  - action: labeldrop
                    regex: "(request_id|trace_id|user_id|pod_ip)
```

2. Recording Rules as the Federation Contract

Every metric crossing from Layer 1 to Layer 2 must pass through a recording rule. The CI pipeline validates this using `promtool check config` and `amtool check-config` before any merge is allowed.

3. Weekly Cardinality Reports

A CronJob posts the top-10 series by cardinality to each team's Slack channel every Monday. Engineers see their numbers before they become a problem.

What We Learned the Hard Way

`honor_labels: true` is mandatory but dangerous.

Without it, the scraping Prometheus overwrites job/instance labels from the source, breaking all dashboards and expressions. But it also means a rogue Layer 1 instance can inject arbitrary labels upward. Mitigate with relabeling at the federation boundary.

Layer 2 HA is non-negotiable.

We run each Domain Prometheus as a HA pair. The first time one went down during an incident, we lost domain-level alerting for 4 minutes — unacceptable when managing production SLOs.

EKS cluster upgrades require care.

When upgrading the Fargate EKS cluster hosting central Prometheus, set node group max size to 3 and verify the EC2 Auto Scaling Group desired capacity before the upgrade. Reducing it post-upgrade can terminate the node running monitoring resources. We learned this with a brief monitoring gap on INT.

Thanos Query Frontend from day one.

Without it, 7-day range queries timed out regularly. The query sharding and caching layer made those queries instant. Add it at the start, not after the pain.

Remote Write is not federation.

Remote Write sends full-cardinality data, defeating the purpose. Federation with recording rules is the correct pattern for multi-tenant cardinality control.

Results

After six months of this federated architecture in production across our multi-tenant EKS platform:

Metric	Before	After
Global Prometheus active series	~800k	~12k
P95 MTTD (complex incidents)	~45 min	~18 min
False positive alert rate	~35%	~8%
On-call pages/week (noise)	~40	~12
Grafana dashboard load (7d queries)	15–25s	<2s
Layer 1 restart blast radius	Platform-wide	Namespace-scoped
Teams served per config change	1	All (CI generates)
Regions supported	1 (EU)	2 (EU + US)

Closing Thoughts

Federated Prometheus is an organisational design choice as much as a technology choice. The topology you build mirrors the team structure you have. This architecture scales because it gives teams autonomy (they own one YAML file) while maintaining platform consistency (the CI pipeline enforces contracts). Neither pure centralisation nor pure decentralisation achieves both.

The team-specific configuration model was not planned from the start — we added it six months after the initial architecture. In retrospect, it should have been the first design decision. This is the architecture we wish we had built on day one.

Chinmaya Kumar Mishra

Principal Platform Engineer · Cloud Architect · Lead DevSecOps at Siemens Smart Infrastructure, Pune. Owns a production multi-tenant EKS platform serving 60+ microservices across 15 engineering teams. CKA (Jan 2028) · AWS Solutions Architect Associate (Dec 2028) · Published FinOps author · Creator of two organisation-wide platform tools.

linkedin.com/in/chinmaya-mishra-2ab99014 · FinOps Benchmark → · chinmaya.mishra0105@gmail.com