ENTERPRISE TECHNICAL GUIDEBOOK BY

bebliTech

Cloud Observability Platform

Enterprise Architecture & Implementation Guide

AWS · GCP · Azure · Kubernetes · OpenTelemetry · New Relic · Bedrock AI · Datadog · Prometheus

AUTHOR

Saroj Naik

Principal Cloud Architect

ENTERPRISE EDITION

bebliTech

www.beblitech.com

CHAPTER 1

Executive Summary

Modern engineering teams operate across multiple clouds, multiple clusters, and dozens of microservices. Telemetry volume has exploded, but visibility has not kept pace. Traditional monitoring tools were designed for static infrastructure, not for ephemeral containers, service meshes, and event-driven architectures.

This guidebook describes how bebliTech designs and delivers a production-grade observability platform - whether you operate on a single cloud or across AWS, Google Cloud, and Azure simultaneously. The platform is built on open standards (OpenTelemetry), backed by a unified analytics engine (New Relic NRDB), and augmented with AI-driven incident analysis (Amazon Bedrock).

Who This Document Is For

Role	How to Use This Document
CTO · VP Engineering	Platform value, cost ranges, engagement timeline
Platform Architect	Architecture decisions, trade-offs, component selection
SRE · Platform Engineer	Implementation steps, runbooks, troubleshooting
DevOps Engineer	Deployment patterns, GitOps integration, CI/CD
Security Engineer	RBAC, encryption, audit trail, compliance evidence

Outcomes Delivered

Outcome	Measured As	Without This Platform
Unified visibility	Single pane across clouds	3+ siloed tools, no correlation
Faster incident resolution	MTTR reduced 45 to 8 minutes	Manual log grep across clusters
Proactive detection	Alerts fire before user impact	Users report issues first
AI-assisted triage	Plain-English root cause	Engineers parse raw metrics
Cost attribution	Per-team, per-cluster billing	No team-level cloud cost visibility
Audit evidence	Auto-generated for SOC 2 and ISO	Manual evidence collection

ENGAGEMENT MODEL

Five-Phase Delivery

Every observability engagement follows a structured five-phase model. Each phase has a clear scope, deliverables, and a decision gate before proceeding to the next.

Phase	Duration	Focus
1 · Evaluation	Week 1 - 2	Current state assessment, gap analysis, stakeholder interviews, ROI model
2 · High-Level Design	Week 2 - 3	Architecture decisions, component selection, cost modelling, executive sign-off
3 · Low-Level Design	Week 3 - 5	Detailed configurations, RBAC model, retention policies, runbook templates
4 · Action Plan	Week 5 - 6	Sprint planning, team roles, dependencies, risk register
5 · Production Implementation	Week 6 - 14	Infrastructure deployment, instrumentation, dashboards, alerts, handover

Decision Gates

No phase begins until the prior phase produces a written sign-off from the engineering and finance stakeholders. This prevents the most common failure mode in observability projects: building a platform nobody asked for.

CHAPTER 2

Reference Architecture

The diagram below illustrates the full multi-account AWS landing zone deployment, with EKS workloads instrumented via OpenTelemetry, telemetry exported to New Relic, and cross-account observability isolated in a dedicated security boundary.

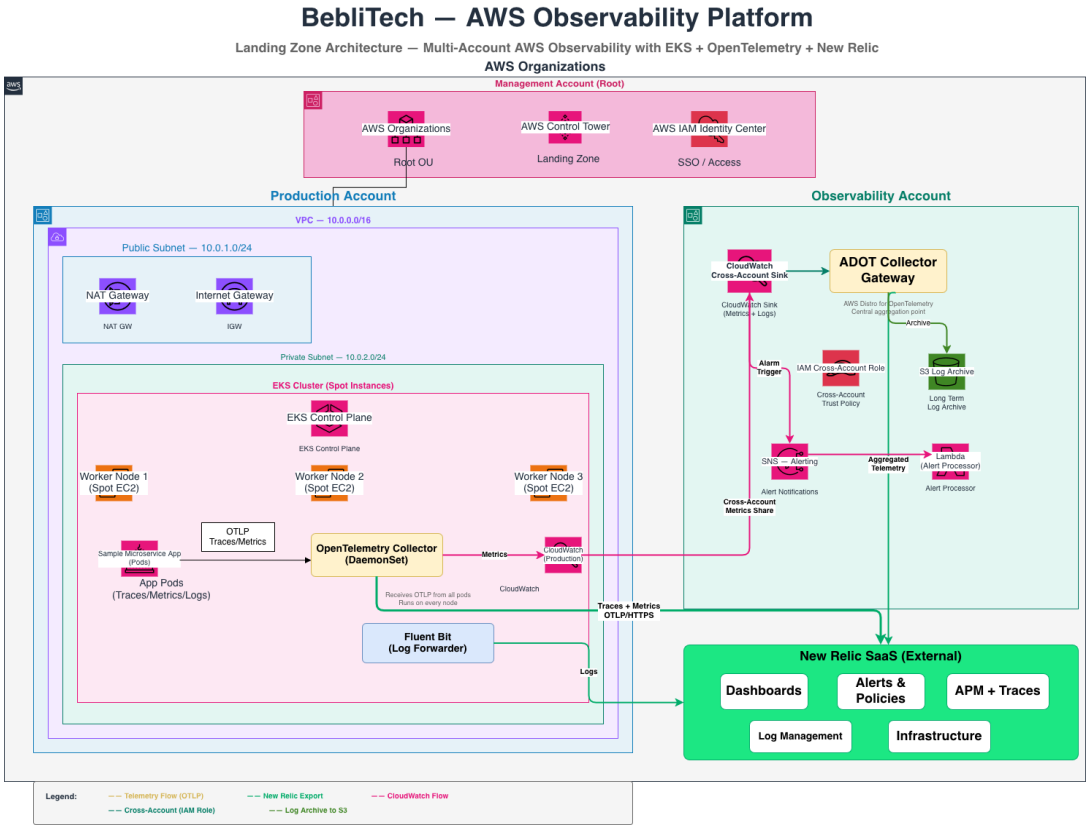

Figure 1 · bebliTech AWS Observability Platform - Landing Zone Reference

Architecture Principles

Vendor-neutral instrumentation

All applications use the OpenTelemetry SDK. Switching backends requires only a gateway config change - never a code change.

Single source of truth

One backend, one query language, one alerting model. No per-team Grafana sprawl, no Prometheus snowflakes.

Data residency by design

AI analysis runs inside your own AWS account via Bedrock. Telemetry never leaves your VPC boundary.

GitOps by default

Collector configs, dashboards, and alert policies are version-controlled. Every change is reviewable, auditable, revertible.

Least privilege always

IRSA scopes IAM permissions to individual service accounts. Teams see only their own namespace data.

CHAPTER 3

Single-Cloud Observability Patterns

Most mid-size companies operate on a single cloud. The same platform architecture applies, with cloud-specific deployment patterns optimised for that environment. Below are the production-tested patterns bebliTech deploys for AWS-only, GCP-only, and Azure-only customers.

Pattern A · AWS-Only Deployment

Layer	Component
Compute	Amazon EKS · Fargate optional for cost-sensitive workloads
Instrumentation	OpenTelemetry SDK · ADOT Collector as DaemonSet
Aggregation	OTel Gateway in dedicated observability account
Backend	New Relic NRDB · Native CloudWatch integration for AWS-managed services
AI Layer	Amazon Bedrock (Claude) · Lambda-based incident analyser
Identity	IRSA · IAM Identity Center for human access
Cost Visibility	AWS Cost Explorer tags · per-team chargeback dashboards

Why this pattern works: ADOT is AWS-native and pre-tested with EKS, IRSA, and CloudWatch. Bedrock keeps AI workloads inside your VPC. Total infrastructure footprint is three to four EC2-equivalent collectors plus Lambda.

Pattern B · GCP-Only Deployment

Layer	Component
Compute	Google Kubernetes Engine (GKE) · Autopilot for hands-off operations
Instrumentation	OpenTelemetry SDK · OTel Collector as DaemonSet
Aggregation	OTel Gateway as GKE Deployment with HPA
Backend	New Relic NRDB · Cloud Operations (Stackdriver) for GCP-managed metrics
AI Layer	Vertex AI (Gemini) · Cloud Functions-based incident analyser
Identity	Workload Identity · Cloud IAM bindings
Cost Visibility	GCP labels · BigQuery billing export to dashboards

Why this pattern works: Workload Identity is the cleanest pod-to-IAM binding in any cloud. Vertex AI Gemini is region-local for data residency. BigQuery billing export is a hidden superpower for FinOps reporting.

Pattern C · Azure-Only Deployment

Layer	Component
Compute	Azure Kubernetes Service (AKS) · Virtual Nodes for spike capacity
Instrumentation	OpenTelemetry SDK · OTel Collector as DaemonSet
Aggregation	OTel Gateway with Azure Container Instances fallback
Backend	New Relic NRDB · Azure Monitor for platform metrics
AI Layer	Azure OpenAI Service · Function App-based incident analyser
Identity	Azure AD Workload Identity · Managed Identity for system services
Cost Visibility	Azure Cost Management · resource tags per cost centre

Why this pattern works: AKS Workload Identity reached GA in 2023 and is now the default. Azure OpenAI Service offers regional data residency with the same model quality as direct OpenAI APIs. Azure Cost Management tagging is enforced via Azure Policy.

CHAPTER 4

Phase 1 · Current State Evaluation

Skipping evaluation is the most common reason observability projects fail. Teams jump to tool selection before understanding what they actually need to observe, who needs to see it, and what data already exists. Two weeks here prevents six months of rework later.

Discovery Questions

Area	Questions to Answer
Current tooling	What monitoring tools exist? Paid or open source? Contract terms and renewal dates?
Data volumes	Service count? Spans/min, metrics/min, log lines/min today and projected?
Cloud footprint	Which clouds and regions? Any on-premises workloads? Hybrid networking?
Team structure	Who owns observability? SRE headcount? On-call rotation size?
Pain points	Top 3 incidents in last 90 days. MTTR per incident. Root causes.
Compliance	SOC 2? HIPAA? PCI? GDPR data residency?
Budget	Annual observability spend? Approval thresholds for new tooling?
SLOs	Are SLOs defined? Measured? Published to users?

Observability Maturity Matrix

Rate each capability from Level 1 to Level 4. Anything below Level 3 is a gap to address in the implementation plan.

Capability	L1 - None	L2 - Basic	L3 - Production	L4 - Optimised
Metrics	No collection	Per-service dashboards	Cross-service alerts	SLO burn-rate alerts
Logs	stdout only	Centralised aggregation	Structured + searchable	Correlated to traces
Traces	No tracing	Manual instrumentation	Auto-instrumentation	Distributed trace UI
Synthetics	Manual testing	Ping monitors	Scripted API tests	Full user journey
AI / Anomaly	No detection	Static thresholds	ML-based anomaly	Plain-English triage
Cost visibility	No attribution	Per-cloud billing	Per-service costs	FinOps dashboards

Phase 1 Deliverables

- Current state report - one page per tool in use today
- Completed gap analysis matrix
- Data volume estimates - spans, metrics, logs per minute per cluster
- Stakeholder map - observability decision owners and budget approvers
- Top-five risk register with mitigations
- Go / No-Go recommendation with cost estimate

CHAPTER 5

Phase 2 · High-Level Design

Component Selection

Component	Selected	Why
Instrumentation	OpenTelemetry SDK	Vendor-neutral. Switching backend requires zero code changes.
Node collection	OTel DaemonSet	Single agent for metrics, traces, logs. Fewer moving parts.
Aggregation	OTel Collector Gateway	Native OTLP. Tail sampling, enrichment, routing in one binary.
Metrics + Traces	New Relic NRDB	Sub-second NRQL. Unified with logs. Zero infrastructure to operate.
Logs	New Relic Logs	Same backend as traces. Single query language. Auto-correlation.
Synthetics	New Relic Synthetics	Integrated with alerting and SLOs. 20+ global locations included.
AI Analysis	Amazon Bedrock	Data stays in your VPC. IAM-scoped access. Pay-per-token.
K8s metrics	kube-state-metrics	Exposes deployment, quota, PVC state beyond what cAdvisor offers.

Cost Model

Estimates below assume tail-based sampling at 10% retention for healthy traces, 100% retention for errors and slow traces, and standard log volumes (no debug logs in production).

Workload Size	Services	Monthly Estimate (USD)
Small	10 services	35 - 50
Medium	50 services	300 - 650
Large	200+ services	1,100 - 2,400

Note: First 100 GB of New Relic ingest is free per account, per month. Most small deployments fit entirely inside the free tier with proper sampling.

CHAPTER 6

Phase 3 · Low-Level Design

End-to-End Data Flow

#	Component	Action
1	OTel SDK in app	Auto-instruments HTTP, DB, Redis, gRPC. Emits via OTLP port 4317.
2	OTel Agent (DaemonSet)	Receives OTLP from local pods. Adds K8s metadata. Pre-batches.
3	OTel Gateway	Tail-samples, enriches, routes by landing-zone tag. Batches to backend.
4	New Relic NRDB	Stores metrics, traces, logs. Queryable via NRQL within 5 seconds.
5	Bedrock Lambda	Reads anomalies from NR API. Calls Claude. Posts plain-English summary.
6	Synthetics	Probes endpoints every 60 seconds from 20+ global locations.

Tail-Based Sampling Strategy

Tail-based sampling decides what to keep after a full trace is complete. Head-based sampling at the SDK level would discard error traces before knowing they were errors - which is precisely backwards.

Rule	Condition	Action
Keep errors	status_code = ERROR	Keep 100% · never miss a production bug
Keep slow traces	duration > 500 ms	Keep 100% · catch latency proactively
Keep outliers	duration > 2x p99 baseline	Keep 100% · capture statistical anomalies
Sample healthy traffic	all other healthy traces	Keep 10% · cost control with coverage

RBAC Model

Role	New Relic Access	Namespace Scope	Alert Management
Platform Admin	Full account	All	Create, edit, delete
Team Lead	Read + dashboard create	Own only	View, acknowledge
Developer	Read only	Own only	View only
SRE On-Call	Read + silence	All	Silence, acknowledge
Security	Logs + traces	All	View only
FinOps	Cost dashboards	N/A	None

Data Retention Policy

Signal	Hot Retention	Cold Retention	Cost Driver
Metrics (raw)	13 months	Managed by NR	Ingest volume
Traces	8 days standard	90 days (Data Plus)	Span count x duration
Logs	30 days standard	1 year (Data Plus)	Log line volume
Synthetic results	13 months	N/A	Monitor count + frequency
Events / Alerts	13 months	N/A	Event count

CHAPTER 7

Phase 5 · Production Implementation

This chapter contains production-tested deployment patterns. All YAML and Python below has been deployed to live environments.

Step 1 · Namespace and Secret Setup

```
# Create observability namespace
kubectl create namespace observability

# Store New Relic license key as Kubernetes secret
kubectl create secret generic newrelic-license \
  --from-literal=license_key=YOUR_NR_LICENSE_KEY \
  --namespace observability

# Verify secret created
kubectl get secret newrelic-license -n observability
```

Step 2 · OTel Gateway Deployment

The Gateway is the central aggregation point. Three replicas with anti-affinity ensures no two replicas share a node. Memory limits prevent collector OOM under load.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: otel-gateway
  namespace: observability
spec:
  replicas: 3
  selector:
    matchLabels: { app: otel-gateway }
  template:
    spec:
      affinity:
        podAntiAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            - labelSelector:
                matchExpressions:
                  - key: app
                    operator: In
                    values: [otel-gateway]
              topologyKey: kubernetes.io/hostname
      containers:
        - name: otel-gateway
          image: otel/opentelemetry-collector-contrib:0.96.0
          args: ["--config=/conf/gateway-config.yaml"]
          env:
            - name: NEW_RELIC_LICENSE_KEY
              valueFrom:
                secretKeyRef:
                  name: newrelic-license
                  key: license_key
          resources:
            requests: { memory: "1Gi", cpu: "500m" }
            limits:   { memory: "2Gi", cpu: "1000m" }
```

Step 3 · Gateway ConfigMap (Sampling and Export)

```
processors:
  tail_sampling:
    decision_wait: 10s
    num_traces: 50000
    policies:
      - name: keep-errors
        type: status_code
        status_code: { status_codes: [ERROR] }
      - name: keep-slow
        type: latency
        latency: { threshold_ms: 500 }
      - name: sample-10-percent
        type: probabilistic
        probabilistic: { sampling_percentage: 10 }

exporters:
  otlphttp/newrelic:
    endpoint: "https://otlp.nr-data.net"
    headers:
      api-key: "${NEW_RELIC_LICENSE_KEY}"
    compression: gzip
    retry_on_failure:
      enabled: true
      max_elapsed_time: 300s
```

Step 4 · Application Instrumentation (Python / FastAPI)

Auto-instrumentation covers 80% of frameworks with zero code changes. The snippet below adds resource attributes that power every team-level dashboard and alert.

```
from opentelemetry import trace
from opentelemetry.sdk.trace import TracerProvider
from opentelemetry.sdk.trace.export import BatchSpanProcessor
from opentelemetry.exporter.otlp.proto.grpc.trace_exporter import OTLPSpanExporter
from opentelemetry.instrumentation.fastapi import FastAPIInstrumentor
from opentelemetry.sdk.resources import Resource
import os

def init_otel(app):
    resource = Resource.create({
        "service.name":           os.getenv("SERVICE_NAME"),
        "service.version":        os.getenv("APP_VERSION", "1.0.0"),
        "deployment.environment": os.getenv("ENVIRONMENT", "prod"),
        "cloud.provider":         os.getenv("CLOUD_PROVIDER", "aws"),
        "team":                   os.getenv("TEAM_NAME"),
    })
    provider = TracerProvider(resource=resource)
    provider.add_span_processor(BatchSpanProcessor(
        OTLPSpanExporter(
            endpoint=os.getenv("OTEL_ENDPOINT", "http://localhost:4317"),
            insecure=True,
        )
    ))
    trace.set_tracer_provider(provider)
    FastAPIInstrumentor.instrument_app(app)
    return trace.get_tracer(__name__)
```

CHAPTER 8

NRQL Queries, Alerts, and SLOs

Essential NRQL Queries

Error rate across all clouds

```
SELECT percentage(count(*), WHERE error IS true) AS 'Error %'
FROM Transaction
FACET cloudProvider, appName
SINCE 1 hour ago TIMESERIES 5 minutes
```

P99 latency by service and cluster

```
SELECT percentile(duration, 99) AS 'P99 ms'
FROM Transaction
FACET service.name, cluster_name
SINCE 1 hour ago
```

Pod health across clusters

```
SELECT uniqueCount(k8s.pod.name) AS 'Total',
       filter(uniqueCount(k8s.pod.name), WHERE status = 'Running') AS 'Running',
       filter(uniqueCount(k8s.pod.name), WHERE status = 'Failed')  AS 'Failed'
FROM K8sPodSample
FACET clusterName SINCE 5 minutes ago
```

Production Alert Conditions

Alert Name	Condition (NRQL)	Critical	Warning
High Error Rate	percentage error count	> 5 %	> 2 %
High P99 Latency	percentile duration 99	> 2000 ms	> 1000 ms
Pod CrashLoop	count CrashLoopBackOff	>= 1	N/A
High Node CPU	avg cpuPercent	> 90 %	> 75 %
High Memory	avg memoryUsedPercent	> 90 %	> 80 %
OTel Queue Full	avg queue_size	> 4500	> 3000
Synthetic Failure	percentage FAILED	> 0 %	N/A

SLO Configuration

Three SLOs per critical service: availability, latency, and error rate. These NRQL queries power the SLO dashboard.

Availability SLO - 99.9% over 28 days

```
SELECT percentage(count(*), WHERE error IS false)
FROM Transaction
WHERE appName = 'YOUR_SERVICE'
SINCE 28 days ago
-- Target: 99.9%   Error budget: 40.3 minutes per 28 days
```

Latency SLO - 99% of requests under 500 ms

```
SELECT percentage(count(*), WHERE duration < 0.5)
FROM Transaction
WHERE appName = 'YOUR_SERVICE'
SINCE 28 days ago
-- Target: 99%   Error budget: 7.2 hours per 28-day window
```

CHAPTER 9

AI-Powered Operations

Static thresholds fire when a metric crosses a known line. AI fires when something unusual happens, even if it has never happened before. More importantly, AI translates raw telemetry into language an on-call engineer can act on at 3am without being an expert in every service.

Use Cases · Prioritised by Impact

Use Case	Input	Output	Effort
Security findings	Bandit + Trivy reports	Plain-English fixes per finding	1 day
Incident triage	Alert + recent logs	Root cause + 3 diagnostic steps	3 days
Anomaly explanation	Metric spike data	This looks like X because Y	3 days
Cost anomaly	Daily AWS cost delta	Specific optimisation actions	2 days
Runbook generation	Historical incidents	Draft runbook per alert type	1 week

Bedrock IAM Policy · Least Privilege

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Action": ["bedrock:InvokeModel"],
    "Resource": [
      "arn:aws:bedrock:ap-south-1::foundation-model/anthropic.claude-3-5-sonnet-
20241022-v2:0"
    ]
  }]
}
```

Incident Triage Lambda · Reference Implementation

```
import boto3, json

def handler(event, context):
    alarm  = event["detail"]["alarmName"]
    region = event["region"]
    logs   = get_recent_logs(region, minutes=10)

    bedrock = boto3.client("bedrock-runtime", region_name=region)
    prompt = f"""
    Alert fired: {alarm}
    Recent logs (last 10 minutes): {logs}

    Answer these three questions:
    1. What likely caused this alert?
    2. What should the on-call engineer check first?
    3. List exactly 3 diagnostic commands to run right now.

    Keep your response under 200 words."""

    response = bedrock.invoke_model(
        modelId="anthropic.claude-3-5-sonnet-20241022-v2:0",
        body=json.dumps({
            "anthropic_version": "bedrock-2023-05-31",
            "max_tokens": 400,
            "messages": [{"role": "user", "content": prompt}]
        }),
        contentType="application/json",
        accept="application/json"
    )

    analysis = json.loads(response["body"].read())["content"][0]["text"]
    post_to_slack(alarm, analysis)
    return {"status": "ok"}
```

Three Non-Negotiable Rules

- AI is advisory only. Never automate remediation without human approval.
- Log every Bedrock invocation for audit and cost reconciliation.
- Never send PII or customer data to any LLM, including Bedrock.

CHAPTER 10

Security and Compliance

Security Controls · How to Verify

Control	Implementation	Verification
Secrets in code	K8s secrets or AWS Secrets Manager via IRSA	Gitleaks scan in CI - zero findings
TLS everywhere	Gateway uses TLS · NR endpoint HTTPS	curl -v shows TLS handshake
IRSA auth	ServiceAccount has scoped IAM via OIDC	aws sts get-caller-identity inside pod
Namespace isolation	Teams see own namespace only in NR	Login as team user, verify scope
Least privilege	NR roles match RBAC matrix	NR > User management > review roles
Audit logging	NR API calls logged · CloudTrail for AWS	NR > Audit log > query recent actions
Data residency	Bedrock in same region as workloads	Check Bedrock region in console
No PII in telemetry	OTel processor strips PII before export	Grep spans for email / phone patterns
Key rotation	NR license key rotated every 90 days	Calendar reminder + rotation script

Compliance Evidence · NRQL for Auditors

These queries produce evidence for SOC 2, ISO 27001, and HIPAA assessments. Run in NR Query Builder and export results.

Who accessed observability data and when

```
SELECT timestamp, actionName, actorEmail, targetName
FROM NrAuditEvent
SINCE 90 days ago
ORDER BY timestamp DESC
```

Alert policy changes - change management evidence

```
SELECT timestamp, actorEmail, actionName, description
FROM NrAuditEvent
WHERE actionName LIKE '%alert%'
SINCE 30 days ago
```

Data access by user - access control evidence

```
SELECT count(*) AS accessCount, actorEmail
FROM NrAuditEvent
SINCE 30 days ago
FACET actorEmail
ORDER BY accessCount DESC
```

CHAPTER 11

Operations Runbook

Daily Operations Checklist

Check	Command or Location	Expected
Gateway pods	kubectl get pods -n observability | grep gateway	3/3 Running
Agent pods	kubectl get pods -n observability -l app=otel-agent	1 per node, all Running
NR data flowing	SELECT count(*) FROM Metric SINCE 5 min ago	Returns > 0
Synthetic monitors	NR > Synthetics > Overview	All green
Active alerts	NR > Alerts > Active alerts	0 critical
Gateway queue	avg queue_size FROM Metric	Below 1000

Common Issues and Fixes

No data in New Relic after 10 minutes

```
kubectl logs -n observability deploy/otel-gateway --tail=50
# Look for: 'Failed to export' or 'connection refused'

kubectl get secret newrelic-license -n observability -o yaml
# Verify license key is set (decode base64 to check)

kubectl describe configmap otel-gateway-config -n observability
# Verify endpoint: https://otlp.nr-data.net
```

OTel Agent in CrashLoopBackOff

```
kubectl describe pod -n observability -l app=otel-agent
# Check Events - usually OOM kill or YAML error

kubectl logs -n observability daemonset/otel-agent --previous
# Previous container logs show exact crash reason
```

Gateway memory pressure

```
kubectl top pods -n observability
# If Gateway > 1.5Gi, reduce batch size in ConfigMap:
#   batch.send_batch_size: 512
# Or increase memory limit to 3Gi
```

Scaling Guidelines

Signal	Trigger	Action
Gateway CPU > 70%	Sustained 10+ minutes	kubectl scale deploy/otel-gateway --replicas=4
Gateway queue > 3000	Warning threshold breached	Increase sending_queue.queue_size, rolling restart
Agent OOMKilled	Container killed by OOM	Increase agent memory limit to 768 Mi
NR ingest near 100GB	Approaching free tier limit	Increase tail sample drop rate to 5% normal traffic
New cluster onboarding	New cloud cluster added	Copy DaemonSet, update cloud_provider tag, apply

CLOSING

Why bebliTech

Built by a Principal Cloud Architect

Every architecture decision in this guide came from production environments, not from a slide deck.

Single-cloud or multi-cloud - same engineering rigour

Whether you run on one cloud or three, the patterns, tooling, and operational discipline remain consistent.

Vendor-neutral by design

OpenTelemetry-first instrumentation means you are never locked into a backend. Switch when business needs change.

Production-first thinking

No proof-of-concepts that never reach production. Every engagement targets a working platform on day one of go-live.

Engagement Models

Model	Duration	Outcome
Discovery Workshop	2 weeks	Fixed-fee evaluation
Platform Build	6 - 12 weeks	Production-ready deployment
Managed Operations	Ongoing	SRE-as-a-service

www.beblitech.com · info@beblitech.com