CloudForge Monitoring Integration Guide

Document ID: PRD-CF-045 **Last Updated:** 2024-03-01 **Owner:** CloudForge Product Team **Classification:** Public

---

Overview

CloudForge integrates seamlessly with DataLens and third-party monitoring tools to provide comprehensive observability for your infrastructure. This guide covers built-in monitoring, integrations, and best practices.

Built-in Monitoring

Default Metrics

CloudForge automatically collects metrics for all resources:

Compute Metrics

Metric	Description	Interval
cpu_utilization	CPU usage percentage	1 min
memory_utilization	Memory usage percentage	1 min
disk_read_bytes	Disk read throughput	1 min
disk_write_bytes	Disk write throughput	1 min
network_in_bytes	Network ingress	1 min
network_out_bytes	Network egress	1 min

Service Metrics

Metric	Description	Interval
request_count	HTTP requests	1 min
request_latency	Response time	1 min
error_rate	Error percentage	1 min
active_connections	Current connections	1 min
replica_count	Running replicas	1 min

Database Metrics

Metric	Description	Interval
connections	Active connections	1 min
cpu_utilization	Database CPU	1 min
storage_used	Storage consumption	5 min
read_iops	Read operations	1 min
write_iops	Write operations	1 min
replication_lag	Replica lag seconds	1 min

DataLens Integration

Enable Integration

```yaml
# environment.yaml
monitoring:
  datalens:
    enabled: true
    workspace: production

    metrics:
      enabled: true
      retention: 15d

    logs:
      enabled: true
      retention: 7d

    traces:
      enabled: true
      sampling_rate: 0.1
```

Automatic Dashboard

When integration is enabled, CloudForge creates dashboards automatically:

- **Environment Overview** - Health status, resource usage
- **Service Performance** - Request rates, latency, errors
- **Database Monitoring** - Connections, queries, performance
- **Cost Analysis** - Spend breakdown, trends

Custom Metrics

Send custom metrics from your applications:

```
const datalens = require('@novatech/datalens');

// Initialize with CloudForge context
datalens.init({
  source: 'cloudforge',
  environment: process.env.CLOUDFORGE_ENV
});

// Send custom metric
datalens.gauge('order.processing_time', 1250, {
  order_type: 'subscription',
  region: 'us-west'
});
```

Alerting

Built-in Alerts

CloudForge provides default alert rules:

Alert	Condition	Default Threshold
High CPU	CPU > threshold for 5m	80%
High Memory	Memory > threshold for 5m	85%
High Error Rate	Errors > threshold for 2m	5%
Service Down	Health check failing	3 consecutive
Database Connection High	Connections > threshold	80% of max
Disk Space Low	Disk usage > threshold	85%

Custom Alerts

```
# alerts.yaml
alerts:
  - name: high-latency
    description: API latency too high
    metric: request_latency_p99
    condition:
      operator: ">"
```

```yaml
      threshold: 500
      duration: 5m
    severity: warning
    notifications:
      - type: slack
        channel: "#alerts"
      - type: pagerduty
        severity: low

  - name: error-spike
    description: Sudden increase in errors
    metric: error_rate
    condition:
      operator: ">"
      threshold: 10
      duration: 2m
    severity: critical
    notifications:
      - type: pagerduty
        severity: high
      - type: email
        recipients:
          - oncall@novatech.com
```

Alert Routing

```yaml
notifications:
  routes:
    - match:
        severity: critical
      receivers: [pagerduty, slack-critical]

    - match:
        severity: warning
      receivers: [slack-warnings]

    - match:
        environment: production
      receivers: [prod-team]

  receivers:
    - name: pagerduty
      type: pagerduty
      routing_key: ${PAGERDUTY_KEY}
    - name: slack-critical
      type: slack
      webhook: ${SLACK_CRITICAL_WEBHOOK}
      channel: "#critical-alerts"
```

Third-Party Integrations

Prometheus

Export metrics to your Prometheus instance:

```
monitoring:
  prometheus:
    enabled: true
    endpoint: https://prometheus.internal.novatech.com
    credentials:
      type: bearer
      token: ${PROMETHEUS_TOKEN}
    scrape_interval: 30s
```

Grafana

CloudForge provides Grafana dashboards:

```
monitoring:
  grafana:
    enabled: true
    url: https://grafana.internal.novatech.com
    api_key: ${GRAFANA_API_KEY}
    dashboards:
      - cloudforge-overview
      - service-performance
      - database-health
```

Datadog

Send metrics to Datadog:

```
monitoring:
  datadog:
```

```yaml
    enabled: true
    api_key: ${DATADOG_API_KEY}
    site: datadoghq.com
    tags:
      - env:production
      - team:platform
```

New Relic

```yaml
monitoring:
  newrelic:
    enabled: true
    license_key: ${NEWRELIC_LICENSE_KEY}
    account_id: ${NEWRELIC_ACCOUNT_ID}
```

PagerDuty

Direct integration for incident management:

```yaml
monitoring:
  pagerduty:
    enabled: true
    routing_key: ${PAGERDUTY_ROUTING_KEY}
    severity_mapping:
      critical: critical
      warning: warning
      info: info
```

Logging

Log Configuration

```yaml
logging:
  level: info
  format: json

  outputs:
    - type: datalens
      enabled: true

    - type: s3
```

```yaml
      enabled: true
      bucket: novatech-logs
      prefix: cloudforge/

    - type: cloudwatch
      enabled: true
      log_group: /cloudforge/production
```

Structured Logging

CloudForge adds context to all logs:

```json
{
  "timestamp": "2024-07-25T15:00:00Z",
  "level": "info",
  "message": "Request completed",
  "service": "api",
  "environment": "production",
  "trace_id": "abc123",
  "span_id": "xyz789",
  "request": {
    "method": "POST",
    "path": "/api/v1/users",
    "status": 201,
    "duration_ms": 145
  }
}
```

Log Queries

Search logs via CLI:

```bash
# Recent errors
cloudforge logs --env production --level error --since 1h

# Specific service
cloudforge logs --env production --service api --filter "timeout"

# Trace correlation
cloudforge logs --env production --trace-id abc123
```

Distributed Tracing

Enable Tracing

```yaml
tracing:
  enabled: true
  provider: datalens
  sampling:
    rate: 0.1  # 10% of requests
    # Or adaptive sampling
    adaptive:
      target_rate: 100  # traces per second
```

Trace Context Propagation

CloudForge automatically propagates trace context:

```yaml
tracing:
  propagation:
    - tracecontext  # W3C standard
    - baggage
    - b3            # Zipkin format
```

Custom Spans

Add custom spans in your application:

```javascript
const tracer = require('@novatech/tracing');

async function processOrder(order) {
  return tracer.startActiveSpan('process_order', async (span) => {
    span.setAttribute('order_id', order.id);

    // Processing logic
    await validateOrder(order);
    await chargePayment(order);
    await fulfillOrder(order);

    span.end();
  });
}
```

Health Checks

Service Health Checks

```yaml
services:
  - name: api
    health_check:
      path: /health
      port: 8080
      interval: 30s
      timeout: 5s
      healthy_threshold: 2
      unhealthy_threshold: 3
```

Readiness vs Liveness

```yaml
services:
  - name: api
    probes:
      liveness:
        path: /healthz
        initial_delay: 30s
        period: 10s
      readiness:
        path: /ready
        initial_delay: 5s
        period: 5s
```

Health Check Response

Standard health check response format:

```json
{
  "status": "healthy",
  "version": "2.5.0",
  "uptime": 86400,
  "checks": {
    "database": {
      "status": "healthy",
      "latency_ms": 5
    },
    "cache": {
      "status": "healthy",
      "latency_ms": 1
    },
    "external_api": {
      "status": "degraded",
      "latency_ms": 250,
      "message": "Slow response"
    }
  }
}
```

Deployment Monitoring

Deployment Events

CloudForge creates annotations for deployments:

```yaml
monitoring:
  annotations:
    deployments: true
    config_changes: true
    scaling_events: true
```

Deployment Metrics

Metric	Description
deployment_duration	Time to complete deployment
deployment_rollback_count	Rollbacks in period
deployment_success_rate	Successful deployments %

Change Correlation

Automatically correlate issues with recent changes:

```
Alert: Error rate spike at 15:05
Possible cause: Deployment at 15:02 (api v2.5.0 → v2.5.1)
```

Best Practices

Dashboard Design

1. **Start with RED method:**
    - Rate (requests/second)
    - Errors (error rate)
    - Duration (latency)
2. **Add USE method for resources:**
    - Utilization
    - Saturation
    - Errors
3. **Include business metrics:**
    - Orders processed
    - Revenue impact
    - User actions

Alert Configuration

1. **Alert on symptoms, not causes**
2. **Set meaningful thresholds**
3. **Include runbook links**
4. **Avoid alert fatigue**
5. **Review and tune regularly**

Retention Strategy

Data Type	Hot Storage	Cold Storage
Metrics	15 days	1 year
Logs	7 days	90 days
Traces	7 days	30 days

Related Documents: DataLens Integration (PRD-DL-040), Alerting Guide (PRD-CF-046), Troubleshooting (PRD-CF-070)