

DataLens Getting Started Guide

Document Number: PROD-DL-001 **Product:** DataLens **Version:** 2.x **Last Updated:** October 15, 2023 **Owner:** DataLens Product Team

Introduction

DataLens is NovaTech's analytics and observability platform. It provides real-time insights into your applications, infrastructure, and business metrics through unified dashboards, powerful querying, and intelligent alerting.

What is DataLens?

DataLens provides:

- **Metrics collection:** Automatic and custom metrics from any source
- **Log aggregation:** Centralized logging with powerful search
- **Distributed tracing:** End-to-end request tracing
- **Custom dashboards:** Build and share visualizations
- **Alerting:** Proactive notifications on anomalies
- **CloudForge integration:** Native observability for CloudForge deployments

Quick Start

Step 1: Access DataLens

1. Go to app.datalens.io
2. Sign in with your NovaTech account
3. Select or create your workspace

Step 2: Connect Your First Data Source

For CloudForge Applications (Automatic) If using CloudForge, metrics and logs are automatically collected. View them at:

- Dashboard: DataLens → CloudForge → [Your Project]
- Logs: DataLens → Logs → [Your Service]

For Other Applications Install the DataLens agent:

Docker:

```
# docker-compose.yml
services:
  datalens-agent:
    image: novatech/datalens-agent:latest
    environment:
```

```

  DATALENS_API_KEY: ${DATALENS_API_KEY}
  DATALENS_SERVICE: my-service
  volumes:
    - /var/run/docker.sock:/var/run/docker.sock

```

Kubernetes:

```

kubectl apply -f https://datalens.io/k8s/agent.yaml
kubectl create secret generic datalens \
--from-literal=api-key=$DATALENS_API_KEY

```

Direct installation:

```
curl -sSL https://get.datalens.io/agent | bash
```

Step 3: Create Your First Dashboard

1. Click “New Dashboard”
2. Click “Add Panel”
3. Select visualization type (chart, table, stat, etc.)
4. Write a query or select metrics
5. Configure display options
6. Save the dashboard

Core Concepts

Metrics

Metrics are numerical measurements over time:

- Counter: Values that only increase (requests, errors)
- Gauge: Values that can go up or down (CPU %, memory)
- Histogram: Distribution of values (latency percentiles)

Example metrics:

- http_requests_total
- cpu_usage_percent
- response_time_seconds

Logs

Logs are timestamped event records:

- Application logs
- System logs
- Audit logs

DataLens automatically parses common formats (JSON, Apache, nginx, etc.).

Traces

Traces show request flow across services:
- Trace: Complete request journey
- Span: Single operation within a trace
- Context: Metadata about the operation

Labels/Tags

Labels are key-value pairs that identify metrics:

```
http_requests_total{service="api", method="GET", status="200"}
```

Use labels to filter and aggregate data.

Query Language (DQL)

DataLens Query Language (DQL) is used for metrics, logs, and traces.

Metrics Queries

```
# Basic metric
http_requests_total

# Filter by label
http_requests_total{service="api"}

# Rate of change (per second)
rate(http_requests_total[5m])

# Aggregation
sum by (service) (http_requests_total)

# Math operations
rate(http_errors_total[5m]) / rate(http_requests_total[5m]) * 100
```

Log Queries

```
# Search logs
logs | search "error"

# Filter by field
logs | where service == "api" && level == "error"
```

```

# Count by field
logs | where level == "error" | count by service

# Pattern matching
logs | search /timeout after \d+ms/

```

Common Query Patterns

Request rate:

```
rate(http_requests_total{service="api"}[5m])
```

Error rate:

```
rate(http_errors_total[5m]) / rate(http_requests_total[5m]) * 100
```

P99 latency:

```
histogram_quantile(0.99, rate(http_request_duration_seconds_bucket[5m]))
```

Memory usage:

```
container_memory_usage_bytes{service="api"} / container_memory_limit_bytes{service="api"} *
```

Dashboards

Creating Dashboards

1. Click “Dashboards” → “New Dashboard”
2. Name your dashboard
3. Add panels using the “Add” button
4. Arrange panels by dragging
5. Save with **Ctrl/Cmd + S**

Panel Types

Type	Use Case
Time series	Metrics over time
Stat	Single value display
Gauge	Current value with thresholds

Type	Use Case
Table	Tabular data
Heatmap	Distribution visualization
Logs	Log stream display
Alert list	Active alerts
Text	Markdown documentation

Variables

Make dashboards dynamic with variables:

1. Click gear icon → “**Variables**”
2. Add variable (e.g., `service`)
3. Use in queries: `http_requests_total{service=\"$service\"}`
4. Users can select values from dropdown

Sharing

- **Link sharing:** Share URL with team members
- **Embedding:** Embed in Notion, wikis
- **PDF export:** Export snapshots
- **Scheduled reports:** Email dashboards on schedule

Alerting

Creating Alerts

1. Go to “**Alerting**” → “**New Alert Rule**”

2. Define the condition:

```
rate(http_errors_total[5m]) / rate(http_requests_total[5m]) > 0.05
```

3. Set duration (e.g., “for 5 minutes”)

4. Configure notification channels

5. Save

Alert Conditions

```
# Error rate > 5% for 5 minutes
- condition: rate(http_errors_total[5m]) / rate(http_requests_total[5m]) > 0.05
  for: 5m
  severity: critical

# Latency P99 > 500ms for 10 minutes
- condition: histogram_quantile(0.99, rate(http_request_duration_seconds_bucket[5m])) > 0.5
  for: 10m
  severity: warning
```

Notification Channels

Supported channels:

- Slack
- PagerDuty
- Email
- Webhook
- Microsoft Teams
- Opsgenie

Alert Routing

Route alerts based on labels:

```
routes:
  - match:
      severity: critical
      receiver: pagerduty
  - match:
      team: platform
      receiver: platform-slack
  - receiver: default-email
```

Integrations

CloudForge

Automatic integration provides:

- Service metrics (CPU, memory, requests)
- Application logs
- Deployment events
- Automatic dashboards

DevPipeline

Track CI/CD metrics:

- Build duration
- Test pass rate
- Deployment frequency

Custom Integrations

Send metrics via SDK or API:

Python:

```
from datalens import Client  
  
dl = Client(api_key="...")  
dl.gauge("custom_metric", 42, tags={"env": "production"})
```

HTTP API:

```
curl -X POST https://api.datalens.io/v1/metrics \  
-H "Authorization: Bearer $API_KEY" \  
-H "Content-Type: application/json" \  
-d '{"metric": "custom_metric", "value": 42, "tags": {"env": "production"}}'
```

Best Practices

Dashboard Design

- Group related metrics together
- Use consistent time ranges
- Include documentation panels
- Keep dashboards focused (one purpose per dashboard)

Query Performance

- Use specific label filters
- Avoid high-cardinality labels in aggregations
- Use appropriate time ranges
- Pre-aggregate where possible

Alerting

- Alert on symptoms, not causes
- Include runbook links in alerts
- Avoid alert fatigue (be selective)
- Use appropriate severity levels

Troubleshooting

No Data Showing

- Verify agent is running
- Check API key is correct
- Confirm network connectivity
- Check time range selection

Query Too Slow

- Add label filters
- Reduce time range
- Simplify query
- Check for high-cardinality labels

Alerts Not Firing

- Verify condition is correct
- Check “for” duration
- Confirm notification channel setup
- Review alert state in UI

Getting Help

- **Documentation:** docs.datalens.io
- **Support:** support@datalens.io
- **Community:** community.datalens.io
- **Internal NovaTech:** #datalens-help on Slack

Next Steps

1. Query Language Reference
 2. Dashboard Best Practices
 3. Alerting Guide
 4. CloudForge Integration
 5. API Reference
-

Related Documents: Query Language Reference (PROD-DL-010), CloudForge Monitoring (PROD-CF-060), Alert Configuration (PROD-DL-035)