````
# OpenPipeline Migration Guide: Part 4

> **Series:** OPMIG | **Notebook:** 4 of 9 | **Created:** December 2025

## Pipeline Configuration Fundamentals

---

## Learning Objectives

By the end of this notebook, you will:

- ✅ Navigate the OpenPipeline UI confidently
- ✅ **Follow step-by-step UI walkthrough**
- ✅ Create custom pipelines from scratch
- ✅ Configure processors with best practices
- ✅ Test configurations with sample data
- ✅ **Build complete end-to-end pipeline examples**
- ✅ **Configure pipelines via API/code**
- ✅ Verify pipeline processing with DQL

---

---

## Accessing OpenPipeline Settings - Detailed Guide

### Navigation Path

**From Dynatrace Home:**
```
1. Click ≡ (hamburger menu) in top-left
2. Scroll to "Manage" section
3. Click "Settings"
4. In left sidebar, expand "Process and contextualize"
5. Click "OpenPipeline"
```

**Direct URL:**
```
https://{your-environment}.apps.dynatrace.com/ui/settings/openpipeline
```

### OpenPipeline UI Overview

![OpenPipeline UI Overview]
(
````

ciIHZpZXdCb3g9IjAgMCA3MDAgMzgwIj4KICA8ZGVmcz4KICAgIDxsaW5lYXJHcmFkaWVudCBpZD0
iaGVhZGVyR3JhZCIgeDE9IjAlIiB5MT0iMCUiIHgyPSIwJSIgeTI9IjEwMCUiPgogICAgICA8c3Rv
cCBvZmZzZXQ9IjAlIiBzdHlsZT0ic3RvcC1jb2xvcjojMWU0MGFmO3N0b3Atb3BhY2l0eToxIiAvP
gogICAgICA8c3RvcCBvZmZzZXQ9IjEwMCUiIHN0eWxlPSJzdG9wLWNvbG9yOiMxZTNhOGE7c3RvcC
1vcGFjaXR5OjEiIC8+CiAgICA8L2xpbmVhckdyYWRpZW50PgogICAgPGZpbHRlciBpZD0idWlTaGF
kb3ciPgogICAgICA8ZmVEcm9wU2hhZG93IGR4PSIyIiBkeT0iMiIgc3RkRGV2aWF0aW9uPSIzIiBm
bG9vZC1vcGFjaXR5PSIwLjE1Ii8+CiAgICA8L2ZpbHRlcj4KICA8L2RlZnM+CgogIDwhLS0gQmFja
2dyb3VuZCAtLT4KICA8cmVjdCB3aWR0aD0iNzAwIiBoZWlnaHQ9IjM4MCIgZmlsbD0iI2Y4ZjlmYS
Igcng9IjEwIi8+CgogIDwhLS0gV2luZG93IGZyYW1lIC0tPgogIDxyZWN0IHg9IjMwIiB5PSIyMCI
gd2lkdGg9IjY0MCIgaGVpZ2h0PSIzNDAiIHJ4PSI4IiBmaWxsPSJ3aGl0ZSIgc3Ryb2tlPSIjZTJl
OGYwIiBzdHJva2Utd2lkdGg9IjIiIGZpbHRlcj0idXJsKCN1aVNoYWRvdykiLz4KCiAgPCEtLSBIZ
WFkZXIgYmFyIC0tPgogIDxyZWN0IHg9IjMwIiB5PSIyMCIgd2lkdGg9IjY0MCIgaGVpZ2h0PSI0NS
Igcng9IjgiIGZpbGw9InVybCgjaGVhZGVyR3JhZCkiLz4KICA8cmVjdCB4PSIzMCIgeT0iNTciIHd
pZHRoPSI2NDAiIGhlaWdodD0iOCIgZmlsbD0idXJsKCNoZWFkZXJHcmFkKSIvPgogIDx0ZXh0IHg9
IjUwIiB5PSI0OCIgZm9udC1mYW1pbHk9IkFyaWFsLCBzYW5zLXNlcmlmIiBmb250LXNpemU9IjE0I
iBmb250LXdlaWdodD0iYm9sZCIgZmlsbD0id2hpdGUiPk9wZW5QaXBlbGluZSBDb25maWd1cmF0aW
9uPC90ZXh0PgoKICA8IS0tIEhlbHAgYnV0dG9uIC0tPgogIDxyZWN0IHg9IjYyMCIgeT0iMzIiIHd
pZHRoPSIzNSIgaGVpZ2h0PSIyMCIgcng9IjQiIGZpbGw9InJnYmEoMjU1LDI1NSwyNTUsMC4yKSIv
PgogIDx0ZXh0IHg9IjYzNyIgeT0iNDYiIGZvbnQtZmFtaWx5PSJBcmlhbCwgc2Fucy1zZXJpZiIgZ
m9udC1zaXplPSIxMiIgZmlsbD0id2hpdGUiIHRleHQtYW5jaG9yPSJtaWRkbGUiPj8gSGVscDwvdG
V4dD4KCiAgPCEtLSBTY29wZSBkcm9wZG93biAtLT4KICA8dGV4dCB4PSI1MCIgeT0iOTUiIGZvbnQ
tZmFtaWx5PSJBcmlhbCwgc2Fucy1zZXJpZiIgZm9udC1zaXplPSIxMiIgZmlsbD0iIzY0NzQ4YiI+
Q29uZmlndXJhdGlvbiBzY29wZTo8L3RleHQ+CiAgPHJlY3QgeD0iMTc1IiB5PSI4MCIgd2lkdGg9I
jEyMCIgaGVpZ2h0PSIyOCIgcng9IjQiIGZpbGw9IndoaXRlIiBzdHJva2U9IiNjYmQ1ZTEiIHN0cm
9rZS13aWR0aD0iMSIvPgogIDx0ZXh0IHg9IjE5MCIgeT0iOTkiIGZvbnQtZmFtaWx5PSJBcmlhbCw
gc2Fucy1zZXJpZiIgZm9udC1zaXplPSIxMiIgZmlsbD0iIzMzNDE1NSI+TG9nczwvdGV4dD4KICA8
dGV4dCB4PSIyODAiIHk9Ijk5IiBmb250LWZhbWlseT0iQXJpYWwsIHNhbnMtc2VyaWYiIGZvbnQtc
2l6ZT0iMTAiIGZpbGw9IiM2NDc0OGIiPuKWvDwvdGV4dD4KCiAgPCEtLSBBcnJvdyBwb2ludGluZy
B0byBkcm9wZG93biAtLT4KICA8cGF0aCBkPSJNMzA1LDk0IEwzNDAsOTQiIHN0cm9rZT0iIzNiODJ
mNiIgc3Ryb2tlLXdpZHRoPSIyIiBmaWxsPSJub25lIiBtYXJrZXItZW5kPSJ1cmwoI3VpQXJyb3cp
Ii8+CiAgPHRleHQgeD0iMzUwIiB5PSI5OCIgZm9udC1mYW1pbHk9IkFyaWFsLCBzYW5zLXNlcmlmI
iBmb250LXNpemU9IjEwIiBmaWxsPSIjM2I4MmY2IiBmb250LXN0eWxlPSJpdGFsaWMiPlNFTEVDVC
BEQVRBIFRZUEU8L3RleHQ+CgogIDwhLS0gVGFicyAtLT4KICA8cmVjdCB4PSI1MCIgeT0iMTIwIiB
3aWR0aD0iNjAwIiBoZWlnaHQ9IjM1IiByeD0iNCIgZmlsbD0iI2YxZjVmOSIgc3Ryb2tlPSIjZTJl
OGYwIiBzdHJva2Utd2lkdGg9IjEiLz4KICA8cmVjdCB4PSI1MiIgeT0iMTIyIiB3aWR0aD0iMTIwI
iBoZWlnaHQ9IjMxIiByeD0iMyIgZmlsbD0iIzNiODJmNiIvPgogIDx0ZXh0IHg9IjExMiIgeT0iMT
QzIiBmb250LWZhbWlseT0iQXJpYWwsIHNhbnMtc2VyaWYiIGZvbnQtc2l6ZT0iMTEiIGZpbGw9Ind
oaXRlIiB0ZXh0LWFuY2hvcj0ibWlkZGxlIiBmb250LXdlaWdodD0iYm9sZCI+UGlwZWxpbmVzPC90
ZXh0PgogIDx0ZXh0IHg9IjIzMiIgeT0iMTQzIiBmb250LWZhbWlseT0iQXJpYWwsIHNhbnMtc2Vya
WYiIGZvbnQtc2l6ZT0iMTEiIGZpbGw9IiM2NDc0OGIiIHRleHQtYW5jaG9yPSJtaWRkbGUiPkR5bm
FtaWMgcm91dGluZzwvdGV4dD4KICA8dGV4dCB4PSIzNzIiIHk9IjE0MyIgZm9udC1mYW1pbHk9IkF
yaWFsLCBzYW5zLXNlcmlmIiBmb250LXNpemU9IjExIiBmaWxsPSIjNjQ3NDhiIiB0ZXh0LWFuY2hv
cj0ibWlkZGxlIj5Jbmdlc3Qgc291cmNlczwvdGV4dD4KCiAgPCEtLSBBZGQgUGlwZWxpbmUgYnV0d
G9uIC0tPgogIDxyZWN0IHg9IjUwIiB5PSIxNzAiIHdpZHRoPSIxMTAiIGhlaWdodD0iMzIiIHJ4PS
I2IiBmaWxsPSIjMTBiOTgxIi8+CiAgPHRleHQgeD0iMTA1IiB5PSIxOTEiIGZvbnQtZmFtaWx5PSJ
BcmlhbCwgc2Fucy1zZXJpZiIgZm9udC1zaXplPSIxMiIgZmlsbD0id2hpdGUiIHRleHQtYW5jaG9y
PSJtaWRkbGUiIGZvbnQtd2VpZ2h0PSJib2xkIj4rIFBpcGVsaW5lPC90ZXh0PgoKICA8IS0tIEFyc
m93IHBvaW50aW5nIHRvIGJ1dHRvbiAtLT4KICA8cGF0aCBkPSJNMTcwLDE4NiBMMjEwLDE4NiIgc3

Ryb2tlPSIjMTBiOTgxIiBzdHJva2Utd2lkdGg9IjIiIGZpbGw9Im5vbmUiLz4KICA8dGV4dCB4PSI
yMjAiIHk9IjE5MCIgZm9udC1mYW1pbHk9IkFyaWFsLCBzYW5zLXNlcmlmIiBmb250LXNpemU9IjEw
IiBmaWxsPSIjMTBiOTgxIiBmb250LXN0eWxlPSJpdGFsaWMiPkNSRUFURSBORVcgUElQRUxJTkU8L
3RleHQ+CgogIDwhLS0gUGlwZWxpbmUgbGlzdCAtLT4KICA8ZyB0cmFuc2Zvcm09InRyYW5zbGF0ZS
g1MCwgMjIwKSI+CiAgICA8IS0tIFBpcGVsaW5lIDE6IGRlZmF1bHQgLS0+CiAgICA8cmVjdCB3aWR
0aD0iNjAwIiBoZWlnaHQ9IjQwIiByeD0iNCIgZmlsbD0iI2Y4ZmFmYyIgc3Ryb2tlPSIjZTJlOGYw
IiBzdHJva2Utd2lkdGg9IjEiLz4KICAgIDx0ZXh0IHg9IjE1IiB5PSIyNSIgZm9udC1mYW1pbHk9I
kFyaWFsLCBzYW5zLXNlcmlmIiBmb250LXNpemU9IjE4IiBmaWxsPSIjZjU5ZTBiIj7wn5OBPC90ZX
h0PgogICAgPHRleHQgeD0iNDUiIHk9IjI2IiBmb250LWZhbWlseT0iQXJpYWwsIHNhbnMtc2VyaWY
iIGZvbnQtc2l6ZT0iMTIiIGZpbGw9IiMzMzQxNTUiPmRlZmF1bHQ8L3RleHQ+CiAgICA8dGV4dCB4
PSI1NzUiIHk9IjI2IiBmb250LWZhbWlseT0iQXJpYWwsIHNhbnMtc2VyaWYiIGZvbnQtc2l6ZT0iM
TQiIGZpbGw9IiM5NGEzYjgiPuKLrjwvdGV4dD4KCiAgICA8IS0tIFBpcGVsaW5lIDIgLS0+CiAgIC
A8cmVjdCB5PSI0NSIgd2lkdGg9IjYwMCIgaGVpZ2h0PSI0MCIgcng9IjQiIGZpbGw9IiNmOGZhZmM
iIHN0cm9rZT0iI2UyZThmMCIgc3Ryb2tlLXdpZHRoPSIxIi8+CiAgICA8dGV4dCB4PSIxNSIgeT0i
NzAiIGZvbnQtZmFtaWx5PSJBcmlhbCwgc2Fucy1zZXJpZiIgZm9udC1zaXplPSIxOCIgZmlsbD0iI
zNiODJmNiI+8J+TgTwvdGV4dD4KICAgIDx0ZXh0IHg9IjQ1IiB5PSI3MSIgZm9udC1mYW1pbHk9Ik
FyaWFsLCBzYW5zLXNlcmlmIiBmb250LXNpemU9IjEyIiBmaWxsPSIjMzM0MTU1Ij5rdWJlcm5ldGV
zLWxvZ3M8L3RleHQ+CiAgICA8dGV4dCB4PSI1NzUiIHk9IjcxIiBmb250LWZhbWlseT0iQXJpYWws
IHNhbnMtc2VyaWYiIGZvbnQtc2l6ZT0iMTQiIGZpbGw9IiM5NGEzYjgiPuKLrjwvdGV4dD4KCiAgI
CA8IS0tIFBpcGVsaW5lIDM6IGN1c3RvbSAtLT4KICAgIDxyZWN0IHk9IjkwIiB3aWR0aD0iNjAwIi
BoZWlnaHQ9IjQwIiByeD0iNCIgZmlsbD0iI2VmZjZmZiIgc3Ryb2tlPSIjM2I4MmY2IiBzdHJva2U
td2lkdGg9IjEiLz4KICAgIDx0ZXh0IHg9IjE1IiB5PSIxMTUiIGZvbnQtZmFtaWx5PSJBcmlhbCwg
c2Fucy1zZXJpZiIgZm9udC1zaXplPSIxOCIgZmlsbD0iIzNiODJmNiI+8J+TgTwvdGV4dD4KICAgI
Dx0ZXh0IHg9IjQ1IiB5PSIxMTYiIGZvbnQtZmFtaWx5PSJBcmlhbCwgc2Fucy1zZXJpZiIgZm9udC
1zaXplPSIxMiIgZmlsbD0iIzFlNDBhZiIgZm9udC13ZWlnaHQ9ImJvbGQiPm15LWN1c3RvbS1waXB
lbGluZTwvdGV4dD4KICAgIDx0ZXh0IHg9IjU3NSIgeT0iMTE2IiBmb250LWZhbWlseT0iQXJpYWws
IHNhbnMtc2VyaWYiIGZvbnQtc2l6ZT0iMTQiIGZpbGw9IiM5NGEzYjgiPuKLrjwvdGV4dD4KICA8L
2c+Cjwvc3ZnPgo=)

Configuration Scopes

The dropdown at the top lets you switch between data types:

Scope	Icon	What It Handles
Logs	📋	Log records from all sources
Spans	🔗	Distributed trace spans
Metrics	📊	Metric data ingestion
Events	⚡	Platform events
Business Events	💼	Business analytics events
Security Events	🔐	Security data

> 💡 **Tip:** Each scope has its own separate set of pipelines. Start with **Logs** for your migration.

Three Main Tabs

```
#### 1. Pipelines Tab
- View all pipelines for selected scope
- Create new pipelines
- Edit pipeline configuration
- Configure processing, extraction, storage

#### 2. Dynamic Routing Tab
- Create routing rules
- Map data to pipelines
- Set matching conditions
- Control route order

#### 3. Ingest Sources Tab
- View data sources
- See ingestion statistics
- Monitor source health
- Troubleshoot ingestion issues

---

## Step-by-Step: Creating Your First Pipeline

Let's walk through creating a complete pipeline for nginx access logs.

### Step 1: Create the Pipeline

**Actions:**
1. Go to OpenPipeline → Logs scope
2. Click **[+ Pipeline]** button (top-left)
3. Enter name: `nginx-access-logs`
4. Add description: `Process nginx access logs with parsing and metric
extraction`
5. Click **Create**

**Result:** New empty pipeline appears in list

---

### Step 2: Configure Processing Stage

#### 2a. Add DQL Processor for Parsing

**Actions:**
1. Click on your `nginx-access-logs` pipeline
2. Go to **Processing** tab
3. Click **[+ Processor]** button
4. Select **DQL** from processor types
```

Configuration:
```
Name: Parse nginx access log

Matching condition:
log.source == "nginx" OR contains(content, "GET") OR contains(content,
"POST")

Processor definition:
parse content, "IPADDR:client_ip SPACE '-' SPACE LD:user SPACE '['
LD:timestamp ']' SPACE '\"' LD:method SPACE LD:path SPACE LD:protocol '\"'
SPACE INT:status_code SPACE INT:bytes"
```

Test with Sample Data:
```json
{
  "content": "192.168.1.100 - - [12/Dec/2024:10:30:45 +0000] \"GET /api/users
HTTP/1.1\" 200 1234",
  "log.source": "nginx"
}
```

5. Click **[Run sample data]**
6. Verify fields extracted: `client_ip`, `method`, `path`, `status_code`,
`bytes`
7. Click **[Save]**

2b. Add Enrichment Processor

Actions:
1. Click **[+ Processor]** again
2. Select **DQL**

Configuration:
```
Name: Add environment tags

Matching condition:
(leave empty to apply to all)

Processor definition:
fieldsAdd environment = "production"
| fieldsAdd service_type = "web-server"
| fieldsAdd response_category = if(status_code < 400, "success",
                                 else: if(status_code < 500, "client_error",
                                 else: "server_error"))
```

```

3. Click **[Save]**

---

### Step 3: Configure Metric Extraction

**Actions:**
1. Go to **Metric extraction** tab
2. Click **[+ Metric]**

**Configuration for Request Count:**
```
Metric type: Counter
Metric key: log.nginx.request_count

Dimensions:
  - method
  - status_code
  - environment

Matching condition:
isNotNull(status_code)
```

3. Click **[Save]**
4. Click **[+ Metric]** again for bytes metric

**Configuration for Response Size:**
```
Metric type: Value
Metric key: log.nginx.response_bytes
Value field: bytes

Dimensions:
  - method
  - path

Matching condition:
isNotNull(bytes)
```

5. Click **[Save]**

---

### Step 4: Configure Storage
```

````
**Actions:**
1. Go to **Storage** tab
2. Select target bucket: `default_logs`
3. Click **[Save]**

> 💡 **Tip:** If you created custom buckets, select the appropriate one here
(e.g., `web_logs` with 14-day retention)

---

### Step 5: Create Dynamic Routing

**Actions:**
1. Go to **Dynamic routing** tab (top of OpenPipeline page)
2. Click **[+ Dynamic route]**

**Configuration:**
```
Name: Route nginx logs

Matching condition:
log.source == "nginx" OR contains(content, "nginx")

Pipeline: nginx-access-logs
```

3. Click **[Save]**

---

### Step 6: Test End-to-End

**Option 1: Send Test Log via API**
```bash
curl -i -X POST \
  "https://{your-environment}.live.dynatrace.com/api/v2/logs/ingest" \
  -H "Content-Type: application/json" \
  -H "Authorization: Api-Token {your-token}" \
  -d '{
    "content": "192.168.1.100 - - [12/Dec/2024:10:30:45 +0000] \"GET
/api/users HTTP/1.1\" 200 1234",
    "log.source": "nginx"
  }'
```

**Option 2: Check with Existing Logs**
````

````
Wait 2-3 minutes for processing, then run verification query (see next
section).

---

### Step 7: Verify Processing

See the validation queries in the next section to confirm:
- Logs are routed to your pipeline ✓
- Fields are parsed correctly ✓
- Metrics are being generated ✓

---

## Complete Pipeline Examples

### Example 1: Application Logs with Error Tracking

**Use Case:** Java application logs with structured format

#### Pipeline: `java-app-logs`

**Processing Stage:**

**Processor 1: Parse Log Structure**
```dql
parse content, "'[' TIMESTAMP('yyyy-MM-dd HH:mm:ss'):log_time ']' SPACE '['
LD:level ']' SPACE '[' LD:thread ']' SPACE LD:class ' - ' DATA:message"
```

**Processor 2: Extract Request ID**
```dql
parse content, "'requestId=' LD:request_id"
```

**Processor 3: Classify Errors**
```dql
fieldsAdd error_type = if(contains(message, "NullPointerException"), "NPE",
                      else: if(contains(message, "SQLException"), "DB",
                      else: if(contains(message, "TimeoutException"),
"Timeout",
                      else: "Other")))
| fieldsAdd severity = if(level == "ERROR", "critical",
                      else: if(level == "WARN", "warning",
                      else: "info"))
```

**Metric Extraction:**
````

````
- **Counter:** `log.java.error_count` (dimensions: error_type, class)
- **Counter:** `log.java.request_count` (dimensions: level, thread)

**Event Extraction:**
- **Event Name:** `application.error`
- **Description:** `Error in {class}: {error_type}`
- **Type:** ERROR
- **Matching:** `level == "ERROR"`

**Storage:** `default_logs` (35 days)

---

### Example 2: Payment Service with PCI-DSS Compliance

**Use Case:** Financial logs requiring credit card masking

#### Pipeline: `payment-service-secure`

**Processing Stage (in order):**

**Processor 1: Mask Credit Cards (FIRST!)**
```dql
fieldsAdd content = replacePattern(content, "CREDITCARD", replacement: "
[CC_REDACTED]")
```

**Processor 2: Mask CVV**
```dql
fieldsAdd content = replacePattern(content, "('cvv='|'cvc=') INT{3,4}",
replacement: "cvv=[REDACTED]")
```

**Processor 3: Parse Payment Data**
```dql
parse content, "'orderId=' INT:order_id ','"
| parse content, "'amount=' DOUBLE:amount"
| parse content, "'currency=' LD:currency"
| parse content, "'status=' LD:payment_status"
```

**Processor 4: Add Tags**
```dql
fieldsAdd service = "payment"
| fieldsAdd compliance = "pci-dss"
| fieldsAdd audit_required = true
```
````

````
**Metric Extraction:**
- **Value:** `log.payment.amount` (value: amount, dimensions: currency,
payment_status)
- **Counter:** `log.payment.transaction_count` (dimensions: payment_status,
currency)

**Business Event Extraction:**
- **Type:** `com.company.payment.processed`
- **Provider:** `payment-service`
- **Fields:** order_id, amount, currency, payment_status

**Storage:** `financial_logs` (90 days - compliance requirement)

---

### Example 3: Kubernetes Logs with Pod Context

**Use Case:** Container logs with K8s metadata

#### Pipeline: `kubernetes-app-logs`

**Processing Stage:**

**Processor 1: Drop System Logs**
```dql
Matching: k8s.namespace.name == "kube-system" AND loglevel == "DEBUG"
Action: Drop
```

**Processor 2: Parse JSON Logs**
```dql
parse content, "JSON:json_payload"
```

**Processor 3: Enrich with K8s Context**
```dql
fieldsAdd app = k8s.deployment.name
| fieldsAdd namespace = k8s.namespace.name
| fieldsAdd pod_name = k8s.pod.name
| fieldsAdd container = k8s.container.name
```

**Metric Extraction:**
- **Counter:** `log.k8s.pod_log_count` (dimensions: namespace, app, loglevel)

**Storage:** Route based on namespace:
- production namespace → `prod_logs` (35 days)
- staging namespace → `staging_logs` (14 days)
````

```
- dev namespace → `dev_logs` (7 days)

---

---

## Understanding the Interface

### Pipeline Configuration Tabs

Each pipeline has these configuration sections:

| Tab | Purpose | Key Actions |
|-----|---------|-------------|
| **Processing** | Transform and enrich data | Add DQL processors, parsers |
| **Metric extraction** | Create metrics from data | Define value/counter
metrics |
| **Event extraction** | Generate events | Create platform events |
| **Bizevent extraction** | Create business events | Generate bizevents |
| **Storage** | Configure bucket routing | Set target bucket |

### Processor Configuration Panel

When adding a processor, you configure:

1. **Name**: Descriptive name for the processor
2. **Matching condition**: When this processor runs
3. **Processor definition**: What the processor does
4. **Sample data**: Test data to validate

![Processor Configuration Panel]
(
ciIHZpZXdCb3g9IjAgMCA2MDAgNDAwIj4KICA8ZGVmcz4KICAgIDxsaW5lYXJHcmFkaWVudCBpZD0
icGNIZWFkZXJHcmFkIiB4MT0iMCUiIHkxPSIwJSIgeDI9IjAlIiB5Mj0iMTAwJSI+CiAgICAgIDxz
dG9wIG9mZnNldD0iMCUiIHN0eWxlPSJzdG9wLWNvbG9yOiMxZTQwYWY7c3RvcC1vcGFjaXR5OjEiI
C8+CiAgICAgIDxzdG9wIG9mZnNldD0iMTAwJSIgc3R5bGU9InN0b3AtY29sb3I6IzFlM2E4YTtzdG
9wLW9wYWNpdHk6MSIgLz4KICAgIDwvbGluZWFyR3JhZGllbnQ+CiAgICA8ZmlsdGVyIGlkPSJN
oYWRvdyI+CiAgICAgIDxmZURyb3BTaGFkb3cgZHg9IjIiIGR5PSIyIiBzdGREZXZpYXRpb249IjQi
IGZsb29kLW9wYWNpdHk9IjAuMiIvPgogICAgPC9maWx0ZXI+CiAgPC9kZWZzPgogICA8S0tIEJhY
2tncm91bmQgLS0+CiAgPHJlY3Qgd2lkdGg9IjYwMCIgaGVpZ2h0PSI0MDAiIGZpbGw9IiNmOGY5Zm
EiIHJ4PSIxMCIvPgoKICA8S0tIE1vZGFsIHdpbmRvdyAtLT4KICA8cmVjdCB4PSIzMCIgeT0iMjA
iIHdpZHRoPSI1NDAiIGhlaWdodD0iMzYwIiByeD0iOCIgZmlsbD0id2hpdGUiIHN0cm9rZT0iI2Uy
ZThmMCIgc3Ryb2tlLXdpZHRoPSIyIiBmaWx0ZXI9InVybCgjcGNTaGFkb3cpIi8+CgogIDxLS0gS
GVhZGVyIC0tPgogIDxyZWN0IHg9IjMwIiB5PSIyMCIgd2lkdGg9IjU0MCIgaGVpZ2h0PSI0MCIgcn
g9IjgiIGZpbGw9InVybCgjcGNIZWFkZXJHcmFkKSIvPgogIDxjaXJjbGUgY3g9IjUxIiBjeT0iMz
kiIHI9IjUiIGZpbGw9IiNmNTllMGIiLz4KICA8dGV4dCB4PSI2NSIgeT0iNDQiIGZvbnQtZmFtaWx5PSJBcmlhbCwgc2Fucy1zZXJpZiIgZm9udC1zaXplPSIxNCIgZm9udC13ZWlnaHQ9ImJvbGQiIGZpbGw9IndoaXRlIj5EUUw8L3RleH
```

Q+CgogIDwhLS0gTmFtZSBmaWVsZCAtLT4KICA8dGV4dCB4PSI1MCIgeT0iOTAiIGZvbnQtZmFtaWx
5PSJBcmlhbCwgc2Fucy1zZXJpZiIgZm9udC1zaXplPSIxMSIgZmlsbD0iIzY0NzQ4YiI+TmFtZTo8
L3RleHQ+CiAgPHJlY3QgeD0iNTAiIHk9Ijk1IiB3aWR0aD0iNTAwIiBoZWlnaHQ9IjI4IiByeD0iN
CIgZmlsbD0id2hpdGUiIHN0cm9rZT0iI2NiZDVlMSIgc3Ryb2tlLXdpZHRoPSIxIi8+CiAgPHRleH
QgeD0iNjAiIHk9IjExNCIgZm9udC1mYW1pbHk9Im1vbm9zcGFjZSIgZm9udC1zaXplPSIxMSIgZml
sbD0iIzMzNDE1NSI+UGFyc2UgbmdpbnggYWNjZXNzIGxvZzwvdGV4dD4KCiAgPCEtLSBNYXRjaGlu
ZyBjb25kaXRpb24gLS0+CiAgPHRleHQgeD0iNTAiIHk9IjE0NSIgZm9udC1mYW1pbHk9IkFyaWFsL
CBzYW5zLXNlcmlmIiBmb250LXNpemU9IjExIiBmaWxsPSIjNjQ3NDhiIj5NYXRjaGluZyBjb25kaX
Rpb246PC90ZXh0PgogIDxyZWN0IHg9IjUwIiB5PSIxNTAiIHdpZHRoPSI1MDAiIGhlaWdodD0iMjg
iIHJ4PSI0IiBmaWxsPSIjZjhmYWZjIiBzdHJva2U9IiNjYmQ1ZTEiIHN0cm9rZS13aWR0aD0iMSIv
PgogIDx0ZXh0IHg9IjYwIiB5PSIxNjkiIGZvbnQtZmFtaWx5PSJtb25vc3BhY2UiIGZvbnQtc2l6Z
T0iMTAiIGZpbGw9IiMzYjgyZjYiPmxvZy5zb3VyY2UgPT0gIm5naW54IjwvdGV4dD4KCiAgPCEtLS
BQcm9jZXNzb3IgZGVmaW5pdGlvbiAtLT4KICA8dGV4dCB4PSI1MCIgeT0iMjAwIiBmb250LWZhbWl
seT0iQXJpYWwsIHNhbnMtc2VyaWYiIGZvbnQtc2l6ZT0iMTEiIGZpbGw9IiM2NDc0OGIiPlByb2Nl
c3NvciBkZWZpbml0aW9uOjwvdGV4dD4KICA8cmVjdCB4PSI1MCIgeT0iMjA1IiB3aWR0aD0iNTAwI
iBoZWlnaHQ9IjUwIiByeD0iNCIgZmlsbD0iI2Y4ZmFmYyIgc3Ryb2tlPSIjY2JkNWUxIiBzdHJva2
Utd2lkdGg9IjEiLz4KICA8dGV4dCB4PSI2MCIgeT0iMjI1IiBmb250LWZhbWlseT0ibW9ub3NwYWN
lIiBmb250LXNpemU9IjEwIiBmaWxsPSIjMGY3NjZlIj5wYXJzZSBjb250ZW50LCAiSVBBRERSOmNs
aWVudF9pcCBTTUFDRSAnLScgU1BBQ0U8L3RleHQ+CiAgPHRleHQgeD0iNjAiIHk9IjI0MCIgZm9ud
C1mYW1pbHk9Im1vbm9zcGFjZSIgZm9udC1zaXplPSIxMCIgZmlsbD0iIzBmNzY2ZSI+TEQ6dXNlci
BTUEFDRSAnWycgTEQ6dGltZXN0YW1wICddJyAuLi4iPC90ZXh0PgoKICA8IS0tIFNhbXBsZSBkYXR
hIC0tPgogIDx0ZXh0IHg9IjUwIiB5PSIyODAiIGZvbnQtZmFtaWx5PSJBcmlhbCwgc2Fucy1zZXJp
ZiIgZm9udC1zaXplPSIxMSIgZmlsbD0iIzY0NzQ4YiI+U2FtcGxlIGRhdGE6PC90ZXh0PgogIDxyZ
WN0IHg9IjUwIiB5PSIyODUiIHdpZHRoPSI1MDAiIGhlaWdodD0iNDUiIHJ4PSI0IiBmaWxsPSIjZm
ZmYmViIiBzdHJva2U9IiNmY2QzNGQiIHN0cm9rZS13aWR0aD0iMSIvPgogIDx0ZXh0IHg9IjYwIiB
5PSIzMDUiIGZvbnQtZmFtaWx5PSJtb25vc3BhY2UiIGZvbnQtc2l6ZT0iMTAiIGZpbGw9IiM5MjQw
MGUiPnsiY29udGVudCI6ICIxOTIuMTY4LjEuMSAtIC0gWzEwL0RlYy8yMDI0OjEwOjMwOjQ1PC90Z
Xh0PgogIDx0ZXh0IHg9IjYwIiB5PSIzMjAiIGZvbnQtZmFtaWx5PSJtb25vc3BhY2UiIGZvbnQtc2
l6ZT0iMTAiIGZpbGw9IiM5MjQwMGUiPiswMDAwXSBcIkdFVCAvYXBpL3VzZXJzIEhUVFAvMS4xXCI
gMjAwIDEyMzQifTwvdGV4dD4KCiAgPCEtLSBCdXR0b25zIC0tPgogIDxyZWN0IHg9IjUwIiB5PSIz
NDUiIHdpZHRoPSIxMjAiIGhlaWdodD0iMjgiIHJ4PSI0IiBmaWxsPSIjMTBiOTgxIi8+CiAgPHRle
HQgeD0iMTEwIiB5PSIzNjQiIGZvbnQtZmFtaWx5PSJBcmlhbCwgc2Fucy1zZXJpZiIgZm9udC1zaX
plPSIxMSIgZmlsbD0id2hpdGUiIHRleHQtYW5jaG9yPSJtaWRkbGUiIGZvbnQtd2VpZ2h0PSJib2x
kIj5SdW4gc2FtcGxlIGRhdGE8L3RleHQ+CgogIDxyZWN0IHg9IjM5MCIgeT0iMzQ1IiB3aWR0aD0i
NzAiIGhlaWdodD0iMjgiIHJ4PSI0IiBmaWxsPSJ3aGl0ZSIgc3Ryb2tlPSIjY2JkNWUxIiBzdHJva
2Utd2lkdGg9IjEiLz4KICA8dGV4dCB4PSI0MjUiIHk9IjM2NCIgZm9udC1mYW1pbHk9IkFyaWFsLC
BzYW5zLXNlcmlmIiBmb250LXNpemU9IjExIiBmaWxsPSIjNjQ3NDhiIiB0ZXh0LWFuY2hvcj0ibWl
kZGxlIj5DYW5jZWw8L3RleHQ+CgogIDxyZWN0IHg9IjQ3MCIgeT0iMzQ1IiB3aWR0aD0iODAiIGhl
aWdodD0iMjgiIHJ4PSI0IiBmaWxsPSIjM2I4MmY2Ii8+CiAgPHRleHQgeD0iNTEwIiB5PSIzNjQiI
GZvbnQtZmFtaWx5PSJBcmlhbCwgc2Fucy1zZXJpZiIgZm9udC1zaXplPSIxMSIgZmlsbD0id2hpdG
UiIHRleHQtYW5jaG9yPSJtaWRkbGUiIGZvbnQtd2VpZ2h0PSJib2xkIj5TYXZlPC90ZXh0Pgo8L3N
2Zz4K)

Creating Your First Pipeline

Step-by-Step: Create a Log Processing Pipeline

Example: Create a pipeline for nginx access logs

Step 1: Navigate to OpenPipeline
```
Settings → Process and contextualize → OpenPipeline → Logs
```

Step 2: Create Pipeline
1. Click **+ Pipeline**
2. Enter name: `nginx-access-logs`
3. Click **Create**

Step 3: Add Processing (Parsing)
1. Go to **Processing** tab
2. Click **+ Processor** → **DQL**
3. Configure:
   - **Name**: `Parse nginx access log`
   - **Matching condition**: `matchesValue(content, "*HTTP*")`
   - **Processor definition**:
   ```
   parse content, "IPADDR:client_ip SPACE '-' SPACE LD:user SPACE '['
LD:timestamp ']' SPACE '\"' LD:method SPACE LD:path SPACE LD:protocol '\"'
SPACE INT:status_code SPACE INT:bytes"
   ```

Step 4: Test with Sample Data
1. Enter sample log:
   ```json
   {"content": "192.168.1.100 - - [12/Dec/2024:10:30:45 +0000] \"GET
/api/users HTTP/1.1\" 200 1234"}
   ```
2. Click **Run sample data**
3. Verify extracted fields

Step 5: Save Pipeline
Click **Save** to store the configuration.

Configuring Processors

DQL Processor Examples

The DQL processor is the most flexible. Here are common patterns:

Add Static Fields
```
fieldsAdd environment = "production"
fieldsAdd application = "web-frontend"
```

Add Conditional Fields
```
fieldsAdd severity = if(loglevel == "ERROR", "critical",
                     else: if(loglevel == "WARN", "warning",
                     else: "normal"))
```

Parse with DPL Pattern
```
parse content, "'userId=' LD:user_id ','"
```

Remove Sensitive Fields
```
fieldsRemove password, secret_key, api_token
```

Rename Fields for Standardization
```
fieldsRename old_field_name = standardized_name
```

Drop Processor Examples

Drop processors remove records matching conditions:

Matching Condition	Effect
`loglevel == "DEBUG"`	Drop all debug logs
`contains(content, "healthz")`	Drop health check logs
`contains(content, "/metrics")`	Drop Prometheus scrapes
`status == "TRACE"`	Drop trace-level logs

Technology Parsers

Built-in parsers for common formats:

Parser	Applies To	Extracted Fields
Apache	Apache access logs	client_ip, method, path, status
Nginx	Nginx access logs	Similar to Apache
JSON	JSON-formatted logs	All JSON fields flattened

| **Syslog** | Syslog format | facility, severity, hostname |

Setting Up Dynamic Routing

Dynamic routing sends data to specific pipelines based on matching conditions.

Routing Configuration Steps

1. Go to **Dynamic routing** tab
2. Click **+ Dynamic route**
3. Configure:
   - **Name**: Descriptive route name
   - **Matching condition**: When to use this route
   - **Pipeline**: Target pipeline

Matching Condition Examples

Condition	Routes To
`log.source == "nginx"`	nginx-logs pipeline
`k8s.namespace.name == "production"`	prod-logs pipeline
`contains(content, "payment")`	payment-logs pipeline
`dt.openpipeline.source == "generic"`	api-ingested pipeline
`host.name == "web-server-01"`	specific host pipeline

Route Evaluation Order

![Dynamic Routing Flow]
(

NkYjI3Nzc7c3RvcC1vcGFjaXR5OjEiIC8+CiAgICA8L2xpbmVhckdyYWRpZW50PgogICAgPGxpbmV
hckdyYWRpZW50IGlkPSJkZWZhdWx0R3JhZCIgeDE9IjAlIiB5MT0iMCUiIHgyPSIxMDAlIiB5Mj0i
MTAwJSI+CiAgICAgIDxzdG9wIG9mZnNldD0iMCUiIHN0eWxlPSJzdG9wLWNvbG9yOiM2NDc0OGI7c
3RvcC1vcGFjaXR5OjEiIC8+CiAgICAgIDxzdG9wIG9mZnNldD0iMTAwJSIgc3R5bGU9InN0b3AtY2
9sb3I6IzQ3NTU2OTtzdG9wLW9wYWNpdHk6MSIgLz4KICAgIDwvbGluZWFyR3JhZGllbnQ+CiAgICA
8ZmlsdGVyIGlkPSJkclNoYWRvdyI+CiAgICAgIDxmZURyb3BTaGFkb3cgZHg9IjEiIGR5PSIxIiBz
dGREZXZpYXRpb249IjIiIGZsb29kLW9wYWNpdHk9IjAuMTUiLz4KICAgIDwvZmlsdGVyPgogICAgP
G1hcmtlciBpZD0iZHJBcnJvdyIgbWFya2VyV2lkdGg9IjEwIiBtYXJrZXJIZWlnaHQ9IjciIHJlZl
g9IjkiIHJlZlk9IjMuNSIgb3JpZW50PSJhdXRvIj4KICAgICAgPHBvbHlnb24gcG9pbnRzPSIwIDA
sIDEwIDMuNSwgMCA3IiBmaWxsPSIjNjQ3NDhiIi8+CiAgICA8L21hcmtlcj4KICAgIDxtYXJrZXIg
aWQ9ImdyZWVuQXJyb3ciIG1hcmtlcldpZHRoPSIxMCIgbWFya2VySGVpZ2h0PSI3IiByZWZYPSI5I
iByZWZZPSIzLjUiIG9yaWVudD0iYXV0byI+CiAgICAgIDxwb2x5Z29uIHBvaW50cz0iMCAwLCAxMC
AzLjUsIDAgNyIgZmlsbD0iIzE2YTM0YSIvPgogICAgPC9tYXJrZXI+CiAgPC9kZWZzPgoKICA8IS0
tIEJhY2tncm91bmQgLS0+CiAgPHJlY3Qgd2lkdGg9IjgwMCIgaGVpZ2h0PSIzNDAiIGZpbGw9IiNm
OGY5ZmEiIHJ4PSIxMCIvPgoKICA8IS0tIFRpdGxlIC0tPgogIDx0ZXh0IHg9IjQwMCIgeT0iMjgiI
GZvbnQtZmFtaWx5PSJBcmlhbCwgc2Fucy1zZXJpZiIgZm9udC1zaXplPSIxOCIgZm9udC13ZWlnaH
Q9ImJvbGQiIGZpbGw9IiMzMzMiIHRleHQtYW5jaG9yPSJtaWRkbGUiPkR5bmFtaWMgUm91dGluZyB
GbG93PC90ZXh0PgogIDx0ZXh0IHg9IjQwMCIgeT0iNDgiIGZvbnQtZmFtaWx5PSJBcmlhbCwgc2Fu
cy1zZXJpZiIgZm9udC1zaXplPSIxMSIgZmlsbD0iIzY2NiIgdGV4dC1hbmNob3I9Im1pZGRsZSI+U
mVjb3JkcyBjYW4gbWF0Y2ggbXVsdGlwbGUgcm91dGVzIChtYXggNSBwaXBlbGluZXMpPC90ZXh0Pg
oKICA8IS0tIEluY29taW5nIERhdGEgLS0+CiAgPHJlY3QgeD0iMzAiIHk9IjgwIiB3aWR0aD0iMTA
wIiBoZWlnaHQ9IjUwIiByeD0iOCIgZmlsbD0idXJsKCNkYXRhR3JhZCkiIGZpbHRlcj0idXJsKCNk
clNoYWRvdykiLz4KICA8dGV4dCB4PSI4MCIgeT0iMTEwIiBmb250LWZhbWlseT0iQXJpYWwsIHNhb
nMtc2VyaWYiIGZvbnQtc2l6ZT0iMTEiIGZvbnQtd2VpZ2h0PSJib2xkIiBmaWxsPSJ3aGl0ZSIgdG
V4dC1hbmNob3I9Im1pZGRsZSI+SW5jb21pbmc8L3RleHQ+CiAgPHRleHQgeD0iODAiIHk9IjEyNCI
gZm9udC1mYW1pbHk9IkFyaWFsLCBzYW5zLXNlcmlmIiBmb250LXNpemU9IjEwIiBmaWxsPSJyZ2Jh
KDI1NSwyNTUsMjU1LDAuOSkiIHRleHQtYW5jaG9yPSJtaWRkbGUiPlJlY29yZDwvdGV4dD4KCiAgP
CEtLSBBcnJvdyB0byBldmFsdWF0aW9uIC0tPgogIDxwYXRoIGQ9Ik0xMzAsMTA1IEwxNjAsMTA1Ii
BzdHJva2U9IiM2NDc0OGIiIHN0cm9rZS13aWR0aD0iMiIgZmlsbD0ibm9uZSIgbWFya2VyLWVuZD0
idXJsKCNkckFycm93KSIvPgoKICA8IS0tIFJvdXRlIEV2YWx1YXRpb24gQm94IC0tPgogIDxyZWN0
IHg9IjE3MCIgeT0iNjAiIHdpZHRoPSI0MDAiIGhlaWdodD0iMjIwIiByeD0iOCIgZmlsbD0iI2ZmZ
iIgc3Ryb2tlPSIjZTJlOGYwIiBzdHJva2Utd2lkdGg9IjIiLz4KICA8dGV4dCB4PSIzNzAiIHk9Ij
g1IiBmb250LWZhbWlseT0iQXJpYWwsIHNhbnMtc2VyaWYiIGZvbnQtc2l6ZT0iMTIiIGZvbnQtd2V
pZ2h0PSJib2xkIiBmaWxsPSIjMzMzIiB0ZXh0LWFuY2hvcj0ibWlkZGxlIj5ST1VURSBFVkFMVUFU
SU9OPC90ZXh0PgoKICA8IS0tIFJvdXRlIDEgLS0+CiAgPHJlY3QgeD0iMTkwIiB5PSIxMDAiIHdpZ
HRoPSIxNjAiIGhlaWdodD0iMzUiIHJ4PSI2IiBmaWxsPSIjZDFmYWU1IiBzdHJva2U9IiMyMmM1NW
UiIHN0cm9rZS13aWR0aD0iMSIvPgogIDx0ZXh0IHg9IjI3MCIgeT0iMTE1IiBmb250LWZhbWlseT0
iQXJpYWwsIHNhbnMtc2VyaWYiIGZvbnQtc2l6ZT0iMTAiIGZvbnQtd2VpZ2h0PSJib2xkIiBmaWxs
PSIjMDQ3ODU3IiB0ZXh0LWFuY2hvcj0ibWlkZGxlIj5Sb3V0ZSAxOiBrOHMubmFtZXNwYWNlPC90Z
Xh0PgogIDx0ZXh0IHg9IjI3MCIgeT0iMTI4IiBmb250LWZhbWlseT0iQXJpYWwsIHNhbnMtc2VyaW
YiIGZvbnQtc2l6ZT0iMTAiIGZpbGw9IiMwNDc4NTciIHRleHQtYW5jaG9yPSJtaWRkbGUiPj09ICJ
wcm9kdWN0aW9uIjwvdGV4dD4KCiAgPHRleHQgeD0iMzY1IiB5PSIxMjAiIGZvbnQtZmFtaWx5PSJB
cmlhbCwgc2Fucy1zZXJpZiIgZm9udC1zaXplPSIxMSIgZmlsbD0iIzE2YTM0YSI+WWVzIOKGkjwvd
GV4dD4KCiAgPHJlY3QgeD0iNDAwIiB5PSIxMDAiIHdpZHRoPSIxNTAiIGhlaWdodD0iMzUiIHJ4PS
I2IiBmaWxsPSJ1cmwoI3JvdXRlQUdyYWQpIiBmaWx0ZXI9InVybCgjZHJTaGFkb3cpIi8+CiAgPHR
leHQgeD0iNDc1IiB5PSIxMjIiIGZvbnQtZmFtaWx5PSJBcmlhbCwgc2Fucy1zZXJpZiIgZm9udC1z
aXplPSIxMCIgZm9udC13ZWlnaHQ9ImJvbGQiIGZpbGw9IndoaXRlIiB0ZXh0LWFuY2hvcj0ibWlkZ
GxlIj5wcm9kLWxvZ3MgcGlwZWxpbmU8L3RleHQ+CgogIDwhLS0gUm91dGUgMiAtLT4KICA8cmVjdC

B4PSIxOTAiIHk9IjE0NSIgd2lkdGg9IjE2MCIgaGVpZ2h0PSIzNSIgcng9IjYiIGZpbGw9IiNmZWY
zYzciIHN0cm9rZT0iI2Y1OWUwYiIgc3Ryb2tlLXdpZHRoPSIxIi8+CiAgPHRleHQgeD0iMjcwIiB5
PSIxNjAiIGZvbnQtZmFtaWx5PSJBcmlhbCwgc2Fucy1zZXJpZiIgZm9udC1zaXplPSIxMCIgZm9ud
C13ZWlnaHQ9ImJvbGQiIGZpbGw9IiM5MjQwMGUiIHRleHQtYW5jaG9yPSJtaWRkbGUiPlJvdXRlID
I6IGxvZy5zb3VyY2U8L3RleHQ+CiAgPHRleHQgeD0iMjcwIiB5PSIxNzMiIGZvbnQtZmFtaWx5PSJ
BcmlhbCwgc2Fucy1zZXJpZiIgZm9udC1zaXplPSIxMCIgZmlsbD0iIzkyNDAwZSIgdGV4dC1hbmNo
b3I9Im1pZGRsZSI+PT0gIm5naW54IjwvdGV4dD4KCiAgPHRleHQgeD0iMzY1IiB5PSIxNjUiIGZvb
nQtZmFtaWx5PSJBcmlhbCwgc2Fucy1zZXJpZiIgZm9udC1zaXplPSIxMSIgZmlsbD0iI2Q5NzcwNi
I+WWVzIOKGkjwvdGV4dD4KCiAgPHJlY3QgeD0iNDAwIiB5PSIxNDUiIHdpZHRoPSIxNTAiIGhlaWd
odD0iMzUiIHJ4PSI2IiBmaWxsPSJ1cmwoI3JvdXRlQkdyYWQpIiBmaWx0ZXI9InVybCgjZHJTaGFk
b3cpIi8+CiAgPHRleHQgeD0iNDc1IiB5PSIxNjciIGZvbnQtZmFtaWx5PSJBcmlhbCwgc2Fucy1zZ
XJpZiIgZm9udC1zaXplPSIxMCIgZm9udC13ZWlnaHQ9ImJvbGQiIGZpbGw9IndoaXRlIiB0ZXh0LW
FuY2hvcj0ibWlkZGxlIj5uZ2lueC1sb2dzIHBpcGVsaW5lPC90ZXh0PgoKICA8IS0tIFJvdXRlIDM
gLS0+CiAgPHJlY3QgeD0iMTkwIiB5PSIxOTAiIHdpZHRoPSIxNjAiIGhlaWdodD0iMzUiIHJ4PSI2
IiBmaWxsPSIjZmNlN2YzIiBzdHJva2U9IiNlYzQ4OTkiIHN0cm9rZS13aWR0aD0iMSIvPgogIDx0Z
Xh0IHg9IjI3MCIgeT0iMjA1IiBmb250LWZhbWlseT0iQXJpYWwsIHNhbnMtc2VyaWYiIGZvbnQtc2
l6ZT0iMTAiIGZvbnQtd2VpZ2h0PSJib2xkIiBmaWxsPSIjOWQxNzRkIiB0ZXh0LWFuY2hvcj0ibWl
kZGxlIj5Sb3V0ZSAzOiBjb250YWluczwvdGV4dD4KICA8dGV4dCB4PSIyNzAiIHk9IjIxOCIgZm9u
dC1mYW1pbHk9IkFyaWFsLCBzYW5zLXNlcmlmIiBmb250LXNpemU9IjEwIiBmaWxsPSIjOWQxNzRkI
iB0ZXh0LWFuY2hvcj0ibWlkZGxlIj4oY29udGVudCwgInBheW1lbnQiKTwvdGV4dD4KCiAgPHRleH
QgeD0iMzY1IiB5PSIyMTAiIGZvbnQtZmFtaWx5PSJBcmlhbCwgc2Fucy1zZXJpZiIgZm9udC1zaXp
lPSIxMSIgZmlsbD0iI2RiMjc3NyI+WWVzIOKGkjwvdGV4dD4KCiAgPHJlY3QgeD0iNDAwIiB5PSIx
OTAiIHdpZHRoPSIxNTAiIGhlaWdodD0iMzUiIHJ4PSI2IiBmaWxsPSJ1cmwoI3JvdXRlQ0dyYWQpI
iBmaWx0ZXI9InVybCgjZHJTaGFkb3cpIi8+CiAgPHRleHQgeD0iNDc1IiB5PSIyMTIiIGZvbnQtZm
FtaWx5PSJBcmlhbCwgc2Fucy1zZXJpZiIgZm9udC1zaXplPSIxMCIgZm9udC13ZWlnaHQ9ImJvbGQ
iIGZpbGw9IndoaXRlIiB0ZXh0LWFuY2hvcj0ibWlkZGxlIj5wYXltZW50LWxvZ3MgcGlwZWxpbmU8
L3RleHQ+CgogIDwhLS0gRGVmYXVsdCAtLT4KICA8cmVjdCB4PSIxOTAiIHk9IjIzNSIgd2lkdGg9I
jE2MCIgaGVpZ2h0PSIzMCIgcng9IjYiIGZpbGw9IiNmMWY1ZjkiIHN0cm9rZT0iIzY0NzQ4YiIgc3
Ryb2tlLXdpZHRoPSIxIi8+CiAgPHRleHQgeD0iMjcwIiB5PSIyNTUiIGZvbnQtZmFtaWx5PSJBcml
hbCwgc2Fucy1zZXJpZiIgZm9udC1zaXplPSIxMCIgZmlsbD0iIzQ3NTU2OSIgdGV4dC1hbmNob3I9
Im1pZGRsZSI+KG5vIG1hdGNoKTwvdGV4dD4KCiAgPHRleHQgeD0iMzY1IiB5PSIyNTIiIGZvbnQtZ
mFtaWx5PSJBcmlhbCwgc2Fucy1zZXJpZiIgZm9udC1zaXplPSIxMSIgZmlsbD0iIzY0NzQ4YiI+4o
aSPC90ZXh0PgoKICA8cmVjdCB4PSI0MDAiIHk9IjIzNSIgd2lkdGg9IjE1MCIgaGVpZ2h0PSIzMCI
gcng9IjYiIGZpbGw9InVybCgjZGVmYXVsdEdyYWQpIiBmaWx0ZXI9InVybCgjZHJTaGFkb3cpIi8+
CiAgPHRleHQgeD0iNDc1IiB5PSIyNTUiIGZvbnQtZmFtaWx5PSJBcmlhbCwgc2Fucy1zZXJpZiIgZ
m9udC1zaXplPSIxMCIgZm9udC13ZWlnaHQ9ImJvbGQiIGZpbGw9IndoaXRlIiB0ZXh0LWFuY2hvcj
0ibWlkZGxlIj5kZWZhdWx0IHBpcGVsaW5lPC90ZXh0PgoKICA8IS0tIEtleSBQb2ludHMgLS0+CiA
gPHJlY3QgeD0iNTkwIiB5PSI4MCIgd2lkdGg9IjE4MCIgaGVpZ2h0PSIyMDAiIHJ4PSI4IiBmaWxs
PSIjZmZmIiBzdHJva2U9IiNlMmU4ZjAiIHN0cm9rZS13aWR0aD0iMiIvPgogIDx0ZXh0IHg9IjY4M
CIgeT0iMTA1IiBmb250LWZhbWlseT0iQXJpYWwsIHNhbnMtc2VyaWYiIGZvbnQtc2l6ZT0iMTEiIG
ZvbnQtd2VpZ2h0PSJib2xkIiBmaWxsPSIjMzMzIiB0ZXh0LWFuY2hvcj0ibWlkZGxlIj5LZXkgQ29
uY2VwdHM8L3RleHQ+CgogIDx0ZXh0IHg9IjYxMCIgeT0iMTMwIiBmb250LWZhbWlseT0iQXJpYWws
IHNhbnMtc2VyaWYiIGZvbnQtc2l6ZT0iMTAiIGZpbGw9IiMzMzMiPuKAoiBPcmRlciBtYXR0ZXJzP
C90ZXh0PgogIDx0ZXh0IHg9IjYxMCIgeT0iMTQ4IiBmb250LWZhbWlseT0iQXJpYWwsIHNhbnMtc2
VyaWYiIGZvbnQtc2l6ZT0iMTAiIGZpbGw9IiMzMzMiPuKAoiBNdWx0aS1tYXRjaCBhbGxvd2VkPC9
0ZXh0PgogIDx0ZXh0IHg9IjYxMCIgeT0iMTY2IiBmb250LWZhbWlseT0iQXJpYWwsIHNhbnMtc2Vy
aWYiIGZvbnQtc2l6ZT0iMTAiIGZpbGw9IiMzMzMiPuKAoiBNYXggNSBwaXBlbGluZXM8L3RleHQ+C
iAgPHRleHQgeD0iNjEwIiB5PSIxODQiIGZvbnQtZmFtaWx5PSJBcmlhbCwgc2Fucy1zZXJpZiIgZm

9udC1zaXplPSIxMCIgZmlsbD0iIzMzMyI+4oCiIERlZmF1bHQgY2F0Y2hlcyByZXN0PC90ZXh0Pgo
KICA8cmVjdCB4PSI2MTAiIHk9IjIwMCIgd2lkdGg9IjE0MCIgaGVpZ2h0PSI2MCIgcng9IjQiIGZp
bGw9IiNlMGYyZmUiLz4KICA8dGV4dCB4PSI2ODAiIHk9IjIyMCIgZm9udC1mYW1pbHk9IkFyaWFsL
CBzYW5zLXNlcmlmIiBmb250LXNpemU9IjEwIiBmb250LXdlaWdodD0iYm9sZCIgZmlsbD0iIzAzNj
lhMSIgdGV4dC1hbmNob3I9Im1pZGRsZSI+UHJvIFRpcDwvdGV4dD4KICA8dGV4dCB4PSI2ODAiIH
k9IjIzOCIgZm9udC1mYW1pbHk9IkFyaWFsLCBzYW5zLXNlcmlmIiBmb250LXNpemU9IjEwIiBmaWxs
PSIjMDM2OWExIiB0ZXh0LWFuY2hvcj0ibWlkZGxlIj5UbGFjZSBzcGVjaWZpYyByb3V0ZXM8L3Rle
HQ+CiAgPHRleHQgeD0iNjgwIiB5PSIyNTAiIGZvbnQtZmFtaWx5PSJBcmlhbCwgc2Fucy1zZXJpZi
IgZm9udC1zaXplPSIxMCIgZmlsbD0iIzAzNjlhMSIgdGV4dC1hbmNob3I9Im1pZGRsZSI+YmVmb3J
lIGdlbmVyYWwgb25lczwvdGV4dD4KICAgPCEtLSBSZXN1bHQgTm90ZSAtLT4KICA8cmVjdCB4PSIz
MCIgeT0iMjk1IiB3aWR0aD0iNzQwIiBoZWlnaHQ9IjMwIiByeD0iNCIgZmlsbD0iI2ZlZjNjNyIvP
gogIDx0ZXh0IHg9IjM4MCIgeT0iMzE1IiBmb250LWZhbWlseT0iQXJpYWwsIHNhbnMtc2VyaWYiIG
ZvbnQtc2l6ZT0iMTAiIGZpbGw9IiM5MjQwMGUiIHRleHQtYW5jaG9yPSJtaWRkbGUiPk5vdGU6IEE
gcmVjb3JkIGlzIHN0b3JlZCBPTkNFIGluIHRoZSBmaXBlbGluZSdzIGJ1
Y2tldCwgZXZlbiBpZiBwcm9jZXNzZWQgYnkgbXVsdGlwbGUgcGlwZWxpbmVzPC90ZXh0Pgo8L3N2Z
z4K)

> 💡 **Tip:** Routes are evaluated in order. Place more specific routes first, general routes last.

Testing Your Configuration

Using Sample Data in UI

Every processor supports sample data testing:

1. Enter JSON sample in **Sample data** field
2. Click **Run sample data**
3. Review output in preview panel
4. Adjust processor definition as needed

Sample Data Format

```json
{
  "content": "Your log message here",
  "log.source": "your-source",
  "timestamp": "2024-12-12T10:30:00Z",
  "k8s.namespace.name": "production"
}
```

Testing via API

````
Send test logs to validate end-to-end:

```bash
curl -i -X POST \
  "https://{your-environment}.live.dynatrace.com/api/v2/logs/ingest" \
  -H "Content-Type: application/json" \
  -H "Authorization: Api-Token {your-token}" \
  -d '{"content":"Test log message","log.source":"test-source"}'
```

Response `204` indicates successful ingestion.

---

## Configuration Best Practices

### Pipeline Design Principles

| Principle | Recommendation |
|-----------|----------------|
| **Single Purpose** | One pipeline per use case (not one mega-pipeline) |
| **Specific Routing** | Use precise matching conditions |
| **Security First** | Add masking processors early in the chain |
| **Drop Early** | Remove unwanted data before processing |
| **Test Thoroughly** | Validate every processor with sample data |

### Processor Ordering

Within a pipeline, order processors logically:

```
1. Masking      → Protect sensitive data first
2. Drop         → Remove unwanted records
3. Parse        → Extract structured fields
4. Enrich       → Add context fields
5. Transform    → Compute derived values
```

### Naming Conventions

| Type | Convention | Example |
|------|------------|---------|
| Pipeline | `{source}-{purpose}` | `nginx-access-logs` |
| Processor | `{action}-{target}` | `parse-request-line` |
| Route | `route-to-{pipeline}` | `route-to-nginx-logs` |

### Common Mistakes to Avoid
````

Mistake	Problem	Solution
Overly broad routing	Too many logs hit pipeline	Use specific conditions
No masking	Sensitive data stored	Add masking before processing
Complex conditions	Hard to maintain	Break into multiple pipelines
Untested patterns	Parsing fails on real data	Always test with samples
Missing fallback	Unknown data lost	Configure default pipeline

---

Verifying Pipeline Processing

After configuring pipelines, verify they're working correctly with these queries.

```python
// Verify logs are being processed by your new pipeline
// Replace 'your-pipeline-name' with your actual pipeline name
fetch logs, from: now() - 1h
| filter contains(toString(dt.openpipeline.pipelines), "your-pipeline-name")
| summarize {processed_count = count()}
| fieldsAdd status = if(processed_count > 0, "✅ Pipeline is processing logs", else: "⚠️ No logs processed yet")
```

```python
// Check routing distribution across all pipelines
fetch logs, from: now() - 1h
| summarize {record_count = count()}, by: {dt.openpipeline.pipelines}
| sort record_count desc
```

```python
// Verify parsing is extracting expected fields
// Check for a specific field that should be parsed
fetch logs, from: now() - 1h
| filter isNotNull(dt.openpipeline.pipelines)
| summarize {
    total = count(),
    with_loglevel = countIf(isNotNull(loglevel)),
    with_status = countIf(isNotNull(status))
  }, by: {dt.openpipeline.pipelines}
| fieldsAdd parsing_rate = round((toDouble(with_loglevel) / toDouble(total)) * 100, decimals: 1)
```

```python
```

```
// Sample recent logs from your pipeline to inspect parsed fields
fetch logs, from: now() - 1h
| filter contains(toString(dt.openpipeline.pipelines), "your-pipeline-name")
| limit 10
```

```python
// Check logs going to default pipeline (may need routing)
// High counts here suggest missing routing rules
fetch logs, from: now() - 1h
| filter isNull(dt.openpipeline.pipelines) OR dt.openpipeline.pipelines == "
[]"
| summarize {unrouted = count()}, by: {log.source}
| sort unrouted desc
| limit 20
```

```python
// Monitor pipeline processing over time
fetch logs, from: now() - 24h
| filter isNotNull(dt.openpipeline.pipelines)
| makeTimeseries {record_count = count()}, by: {dt.openpipeline.pipelines},
interval: 1h
```

```python
// Verify bucket routing is working
fetch logs, from: now() - 1h
| filter isNotNull(dt.openpipeline.pipelines)
| summarize {record_count = count()}, by: {dt.openpipeline.pipelines,
dt.system.bucket}
| sort record_count desc
```

Complete Pipeline Example

Here's a complete example of a production pipeline configuration:

Pipeline: `payment-service-logs`

Routing Condition:
```
k8s.namespace.name == "payments" OR log.source == "payment-service"
```

Processors (in order):

```
1. **Mask Credit Cards** (Masking)
   - Matching: `contains(content, "card")`
   - Definition: `fieldsAdd content = replacePattern(toString(content),
"\\d{4}[- ]?\\d{4}[- ]?\\d{4}[- ]?\\d{4}", "****-****-****-****")`

2. **Drop Health Checks** (Drop)
   - Matching: `contains(content, "/health") OR contains(content, "/ready")`

3. **Parse Payment Logs** (DQL)
   - Matching: `matchesValue(content, "*transaction*")`
   - Definition: `parse content, "'transaction_id=' LD:transaction_id ','
'amount=' DOUBLE:amount ',' 'status=' LD:payment_status"`

4. **Add Environment Tag** (DQL)
   - Matching: (none - applies to all)
   - Definition: `fieldsAdd service = "payment-service", criticality =
"high"`

**Metric Extraction:**
- Name: `payment_amount_by_status`
- Field: `amount`
- Dimensions: `payment_status`

**Storage:**
- Bucket: `financial_logs` (90-day retention)

---

## Next Steps

Now that you can create and configure pipelines, continue with:

| Notebook | Focus Area |
|----------|------------|
| **OPMIG-05** | Routing & Bucket Management |
| **OPMIG-06** | Processing, Parsing & Transformation |
| **OPMIG-07** | Metric & Event Extraction |
| **OPMIG-08** | Security, Masking & Compliance |

---

## References

- [Configure Processing Pipeline](https://docs.dynatrace.com/docs/discover-
dynatrace/platform/openpipeline/getting-started/tutorial-configure-
processing)
- [Processing Examples](https://docs.dynatrace.com/docs/discover-
```

```
dynatrace/platform/openpipeline/use-cases/processing-examples)
- [DQL Functions in OpenPipeline](https://docs.dynatrace.com/docs/discover-
dynatrace/platform/openpipeline/reference/openpipeline-dql-functions)
- [Dynatrace Pattern Language](https://docs.dynatrace.com/docs/discover-
dynatrace/platform/grail/dynatrace-pattern-language)

---

*Last Updated: December 12, 2025*
```