



# Mastering Observability with OS Tools



**Luca Raveri**

Software Engineer at Talentware

# Who Am I

- **Hi**👋 I'm Luca Raveri, Software Engineer and AWS Solutions Architect

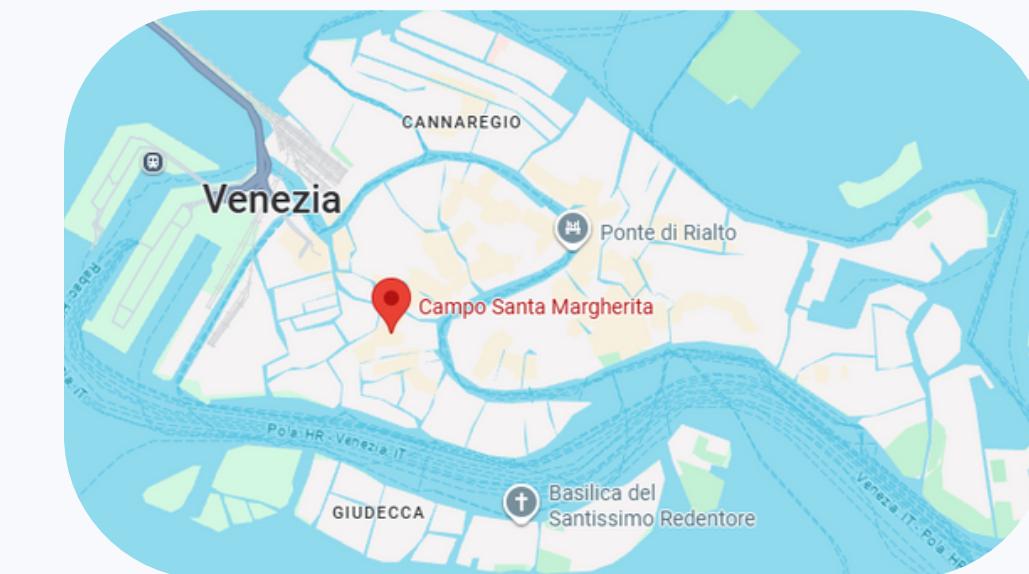


💻 Backend Development

☁️ Cloud Architectures

🔍 Observability

📍 Venice, Italy



**Do you  
really know  
your system?**



# Common Scenario

*The day after the release...*



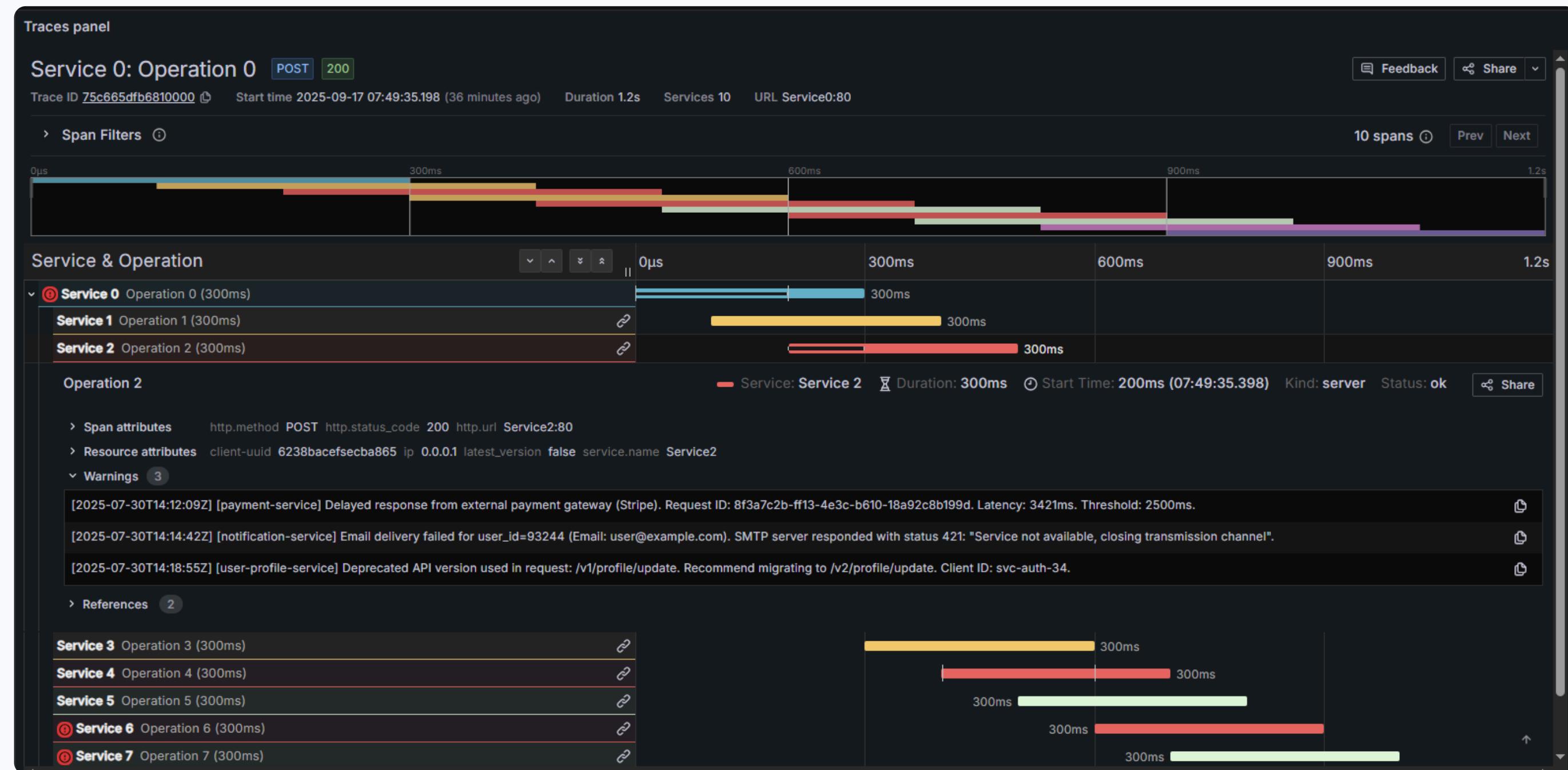
# What are Monitoring and Observability?

- **Monitoring**: Continuous measurement of a system's health to detect anomalies, performance issues, or downtime → *tells us that something is wrong*
- **Observability**: The ability to understand the internal state of a complex system by examining the data it produces → *tells us why it is wrong*

# The Three Pillars Of Observability

- **Metrics:** Numeric measurements over time (e.g. CPU usage)  
→ *What is happening*
- **Logs:** Discrete, timestamped records of events → *Why is happening*
- **Traces:** Request flow through the system → *Where is happening*

# What is a Trace



# Key Benefits of Observability

- **Lower MTTD (Mean Time to Detect)**

Problems are identified quickly, often before users notice.

- **Lower MTTR (Mean Time to Resolve)**

Root causes are found and fixed faster, reducing downtime and impact

# Observability

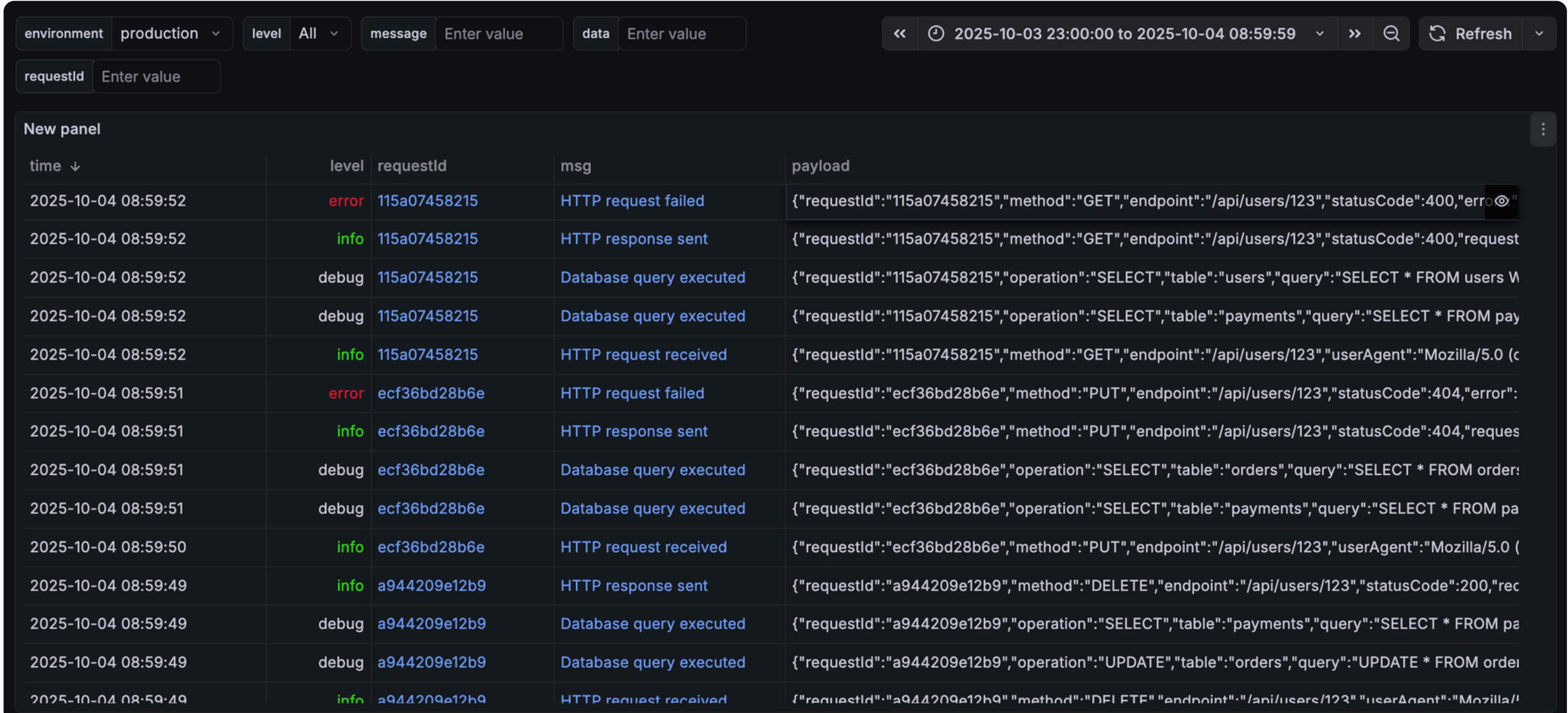
==

# Performance

# From This...

```
[2025-09-30 19:05:37.130 +0200] INFO: service boot {"service":"grafana-demo","payload": {"requestId": "da34f8bf-0bfa-435e-8198-cfdf016b9de8", "service": "orders-api", "e
[2025-09-30 19:05:37.370 +0200] INFO: incoming request {"service": "grafana-demo", "payload": {"requestId": "6d3a0a0c-0edb-4e7d-badf-0aa99447f19f", "method": "GET", "path"
[2025-09-30 19:05:37.418 +0200] INFO: request completed {"service": "grafana-demo", "payload": {"requestId": "6d3a0a0c-0edb-4e7d-badf-0aa99447f19f", "method": "GET", "path
[2025-09-30 19:05:37.529 +0200] DEBUG: sql execution {"service": "grafana-demo", "payload": {"requestId": "2da464f1-2bf9-455b-8efa-60ee5ba10585", "sql": "SELECT 1", "db": "
[2025-09-30 19:05:37.576 +0200] INFO: incoming request {"service": "grafana-demo", "payload": {"requestId": "152e854c-64a5-43e4-b8fe-f3b45cf97a4a", "method": "POST", "path
[2025-09-30 19:05:37.696 +0200] DEBUG: sql execution {"service": "grafana-demo", "payload": {"requestId": "152e854c-64a5-43e4-b8fe-f3b45cf97a4a", "sql": "insert into user
[2025-09-30 19:05:37.739 +0200] INFO: request completed {"service": "grafana-demo", "payload": {"requestId": "152e854c-64a5-43e4-b8fe-f3b45cf97a4a", "method": "POST", "pat
[2025-09-30 19:05:37.988 +0200] ERROR: rate limit nearing {"service": "grafana-demo", "payload": {"requestId": "4be049ab-ca57-417e-94d1-147fab6b1fd7", "clientId": "demo-c
[2025-09-30 19:05:38.049 +0200] INFO: incoming request {"service": "grafana-demo", "payload": {"requestId": "ea90cf08-7d69-4e75-89f6-af7e87e96ef2", "method": "GET", "path
[2025-09-30 19:05:38.124 +0200] DEBUG: sql execution {"service": "grafana-demo", "payload": {"requestId": "ea90cf08-7d69-4e75-89f6-af7e87e96ef2", "sql": "select id, email
[2025-09-30 19:05:38.163 +0200] INFO: request completed {"service": "grafana-demo", "payload": {"requestId": "ea90cf08-7d69-4e75-89f6-af7e87e96ef2", "method": "GET", "path
[2025-09-30 19:05:38.206 +0200] INFO: incoming request {"service": "grafana-demo", "payload": {"requestId": "fb0b4ea8-1050-45a7-ac61-9bad34bc265f", "method": "PATCH", "pat
[2025-09-30 19:05:38.281 +0200] DEBUG: sql execution {"service": "grafana-demo", "payload": {"requestId": "fb0b4ea8-1050-45a7-ac61-9bad34bc265f", "sql": "update users set
[2025-09-30 19:05:38.316 +0200] INFO: request completed {"service": "grafana-demo", "payload": {"requestId": "fb0b4ea8-1050-45a7-ac61-9bad34bc265f", "method": "PATCH", "pa
[2025-09-30 19:05:38.455 +0200] ERROR: unhandled application error {"service": "grafana-demo", "payload": {"requestId": "d4d9fafafa-b284-416f-baba-2a1dfbb6f85b", "method": "
[2025-09-30 19:05:38.562 +0200] DEBUG: sql execution {"service": "grafana-demo", "payload": {"requestId": "d4d9fafafa-b284-416f-baba-2a1dfbb6f85b", "sql": "select id from u
[2025-09-30 19:05:38.600 +0200] INFO: request completed {"service": "grafana-demo", "payload": {"requestId": "d4d9fafafa-b284-416f-baba-2a1dfbb6f85b", "method": "GET", "path
[2025-09-30 19:05:38.692 +0200] INFO: incoming request {"service": "grafana-demo", "payload": {"requestId": "cbca0ba1-cae2-4555-a516-62bb5329a166", "method": "GET", "path
[2025-09-30 19:05:38.756 +0200] DEBUG: cache get {"service": "grafana-demo", "payload": {"requestId": "cbca0ba1-cae2-4555-a516-62bb5329a166", "key": "order:5012", "backend
[2025-09-30 19:05:38.781 +0200] INFO: request completed {"service": "grafana-demo", "payload": {"requestId": "cbca0ba1-cae2-4555-a516-62bb5329a166", "method": "GET", "path
[2025-09-30 19:05:38.905 +0200] ERROR: slow query detected {"service": "grafana-demo", "payload": {"requestId": "25012bdd-1a6a-45e2-9d32-6f1b6c803e32", "sql": "select * f
[2025-09-30 19:05:39.090 +0200] INFO: incoming request {"service": "grafana-demo", "payload": {"requestId": "e1fef4d3-beec-4480-a0ca-9fb6782213fe", "method": "DELETE", "pa
[2025-09-30 19:05:39.131 +0200] INFO: request completed {"service": "grafana-demo", "payload": {"requestId": "e1fef4d3-beec-4480-a0ca-9fb6782213fe", "method": "DELETE", "p
[2025-09-30 19:05:39.241 +0200] ERROR: payment gateway timeout {"service": "grafana-demo", "payload": {"requestId": "12ceeb00-0a86-4ea0-bce0-99efade0a625", "method": "POS
[2025-09-30 19:05:39.413 +0200] DEBUG: sql execution {"service": "grafana-demo", "payload": {"requestId": "12ceeb00-0a86-4ea0-bce0-99efade0a625", "sql": "insert into paym
```

# ...to This!



The screenshot shows a log viewer interface with the following configuration:

- Environment: production
- Level: All
- Message: Enter value
- Data: Enter value
- Request ID: Enter value
- Date Range: 2025-10-03 23:00:00 to 2025-10-04 08:59:59
- Refresh button

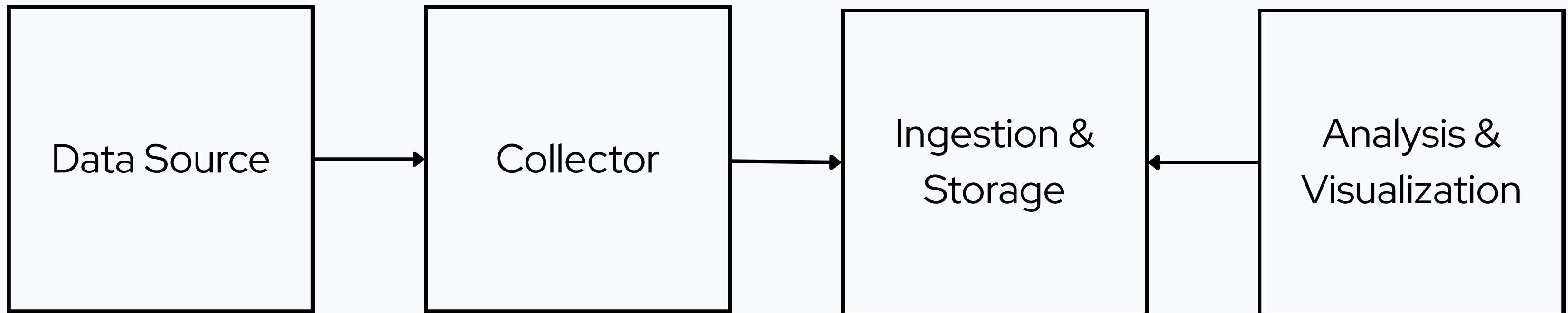
The table displays the following log entries:

time	level	requestId	msg	payload
2025-10-04 08:59:52	error	115a07458215	HTTP request failed	{"requestId": "115a07458215", "method": "GET", "endpoint": "/api/users/123", "statusCode": 400, "error": "Bad Request"}
2025-10-04 08:59:52	info	115a07458215	HTTP response sent	{"requestId": "115a07458215", "method": "GET", "endpoint": "/api/users/123", "statusCode": 400, "requestId": "115a07458215", "status": "failed", "error": "Bad Request"}
2025-10-04 08:59:52	debug	115a07458215	Database query executed	{"requestId": "115a07458215", "operation": "SELECT", "table": "users", "query": "SELECT * FROM users WHERE id = 123"}
2025-10-04 08:59:52	debug	115a07458215	Database query executed	{"requestId": "115a07458215", "operation": "SELECT", "table": "payments", "query": "SELECT * FROM payments WHERE user_id = 123"}
2025-10-04 08:59:52	info	115a07458215	HTTP request received	{"requestId": "115a07458215", "method": "GET", "endpoint": "/api/users/123", "userAgent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/92.0.4515.159 Safari/537.36"}
2025-10-04 08:59:51	error	ecf36bd28b6e	HTTP request failed	{"requestId": "ecf36bd28b6e", "method": "PUT", "endpoint": "/api/users/123", "statusCode": 404, "error": "Not Found"}
2025-10-04 08:59:51	info	ecf36bd28b6e	HTTP response sent	{"requestId": "ecf36bd28b6e", "method": "PUT", "endpoint": "/api/users/123", "statusCode": 404, "requestId": "ecf36bd28b6e", "status": "failed", "error": "Not Found"}
2025-10-04 08:59:51	debug	ecf36bd28b6e	Database query executed	{"requestId": "ecf36bd28b6e", "operation": "SELECT", "table": "orders", "query": "SELECT * FROM orders WHERE user_id = 123"}
2025-10-04 08:59:51	debug	ecf36bd28b6e	Database query executed	{"requestId": "ecf36bd28b6e", "operation": "SELECT", "table": "payments", "query": "SELECT * FROM payments WHERE user_id = 123"}
2025-10-04 08:59:50	info	ecf36bd28b6e	HTTP request received	{"requestId": "ecf36bd28b6e", "method": "PUT", "endpoint": "/api/users/123", "userAgent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/92.0.4515.159 Safari/537.36"}
2025-10-04 08:59:49	info	a944209e12b9	HTTP response sent	{"requestId": "a944209e12b9", "method": "DELETE", "endpoint": "/api/users/123", "statusCode": 200, "requestId": "a944209e12b9", "status": "success"}
2025-10-04 08:59:49	debug	a944209e12b9	Database query executed	{"requestId": "a944209e12b9", "operation": "SELECT", "table": "payments", "query": "SELECT * FROM payments WHERE user_id = 123"}
2025-10-04 08:59:49	debug	a944209e12b9	Database query executed	{"requestId": "a944209e12b9", "operation": "UPDATE", "table": "orders", "query": "UPDATE * FROM orders WHERE user_id = 123"}
2025-10-04 08:59:49	info	a944209e12b9	HTTP request received	{"requestId": "a944209e12b9", "method": "DELETE", "endpoint": "/api/users/123", "userAgent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/92.0.4515.159 Safari/537.36"}

# Demo Time!



# Let's Build an Observability System



# LGTM Stack

**Loki**



Database  
for logs

**Grafana**



Visualization  
tool

**Tempo**



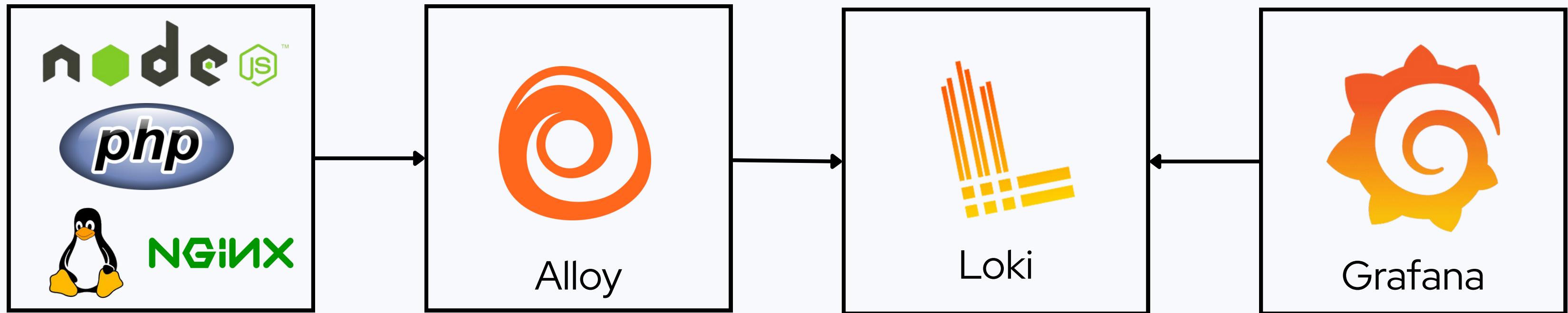
Database  
for traces

**Mimir**



Database  
for metrics

# Let's build an Observability System



# Let's Start Logging

- Use a logger instead of native functions
- Apply log levels and timestamps
- Format logs (JSON)
- Never log secrets and PII
- Set retention policies (e.g., logrotate)

# Logging Strategies

- Log system inputs and outputs
- Centralize logging in a single layer
- Use UUIDs to correlate related logs
- More logs means more insight, but higher overhead



```
console.log(`Customer ${customerName} purchased ${itemsPurchased} items`);
```



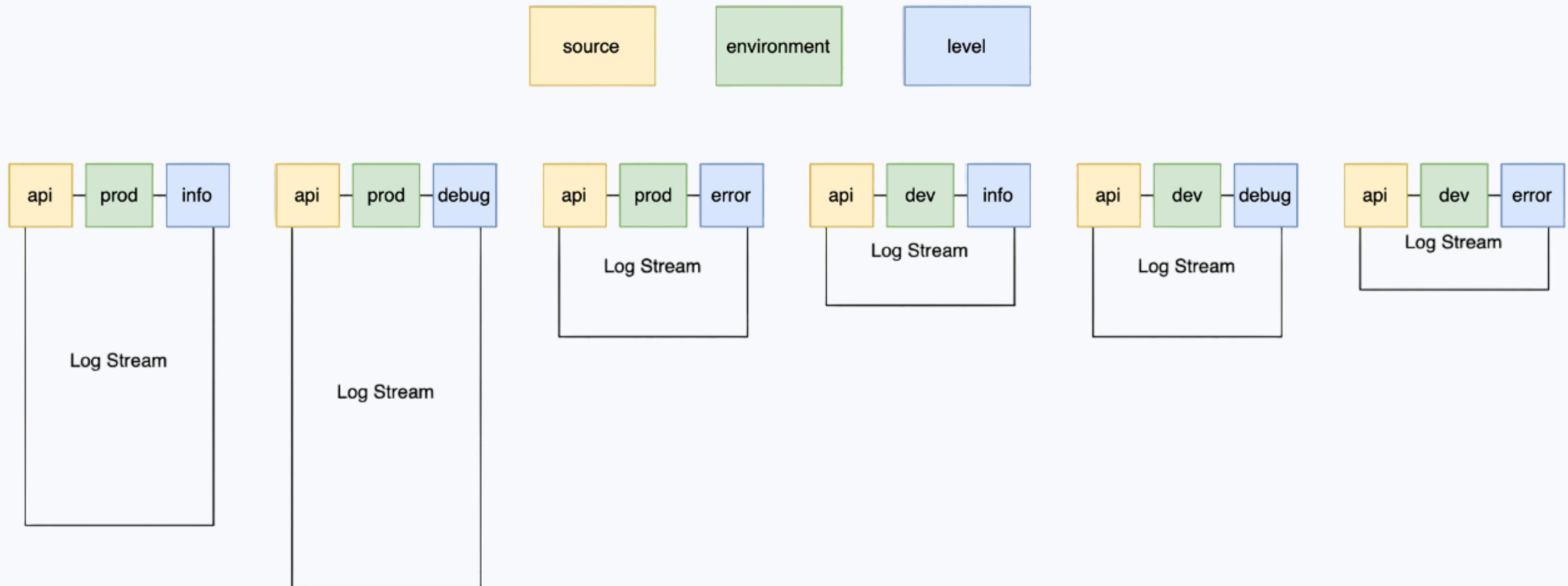
```
logger.info('Customer purchase', {  
  customerId: customer.id,  
  customerName: customer.name,  
  itemsPurchased: customer.itemsPurchased,  
  totalAmount: customer.totalAmount,  
});
```



# What is Loki

*“Loki is a horizontally scalable, highly available, multi-tenant log aggregation system inspired by Prometheus. It is designed to be very cost effective and easy to operate. It does not index the contents of the logs, but rather a set of labels for each log stream.”*

# How Loki Works



*“...It does not index the contents of the logs, but rather a set of labels for each log stream.”*

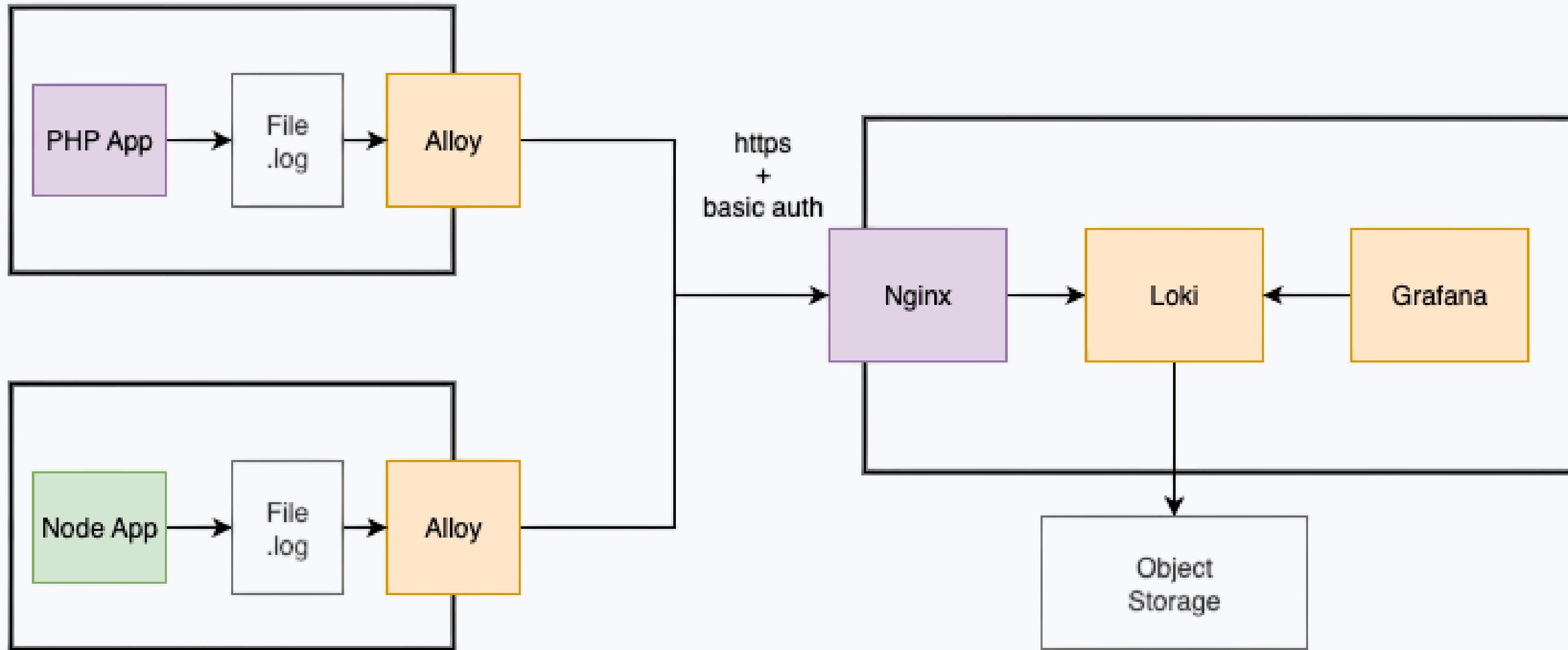
# How to Send Logs to Loki

**Grafana Alloy** is a collector that automatically reads your log files and pushes them to Loki. It can:

- Apply **static labels** to all logs from a file
- Extract **dynamic labels** from log content
- Perform **transformations** on log data



# Hosting Loki



# How to Query Logs

```
{source="grafana-demo", environment=~"${environment}", level=~"${level}"}
|~ "${data}"
| json
| msg =~ ".*${message}.*"
| payload_requestId =~ ".*${requestId}.*"
```



```
{job="apache_error_log", environment=~"${environment}", source=~"${source}"}
|~ "${data}"
| regexp "\\\\[\\((?P<timestamp>[^\\\"]+)\\)\\].*\\[php7:(?P<level>[^\\\"]+)]\\).*PHP (?P<type>[^:]*)\: (?P<message>.*?) in (?P<file>[A-Z]:\\\\\\[^\\\"]+?)\\((?::(?P<line>\\d+)| on line (?P<line_alt>\\d+))"
| line_format "{{ printf `\\\"timestamp\\\":\\\"%s\\\", \\\"level\\\":\\\"%s\\\", \\\"type\\\":\\\"%s\\\", \\\"message\\\":\\\"%s\\\", \\\"file\\\":\\\"%s\\\", \\\"line\\\":\\\"%s\\\"` .timestamp.level.type.message.file (or .line.line_alt) }}"
| message =~ ".*${message}.*"
| file =~ ".*${file}.*"
| message != ""
```



# How to Visualize Logs

1. Write the query
2. Apply transformations
3. Create variables
4. Add overrides

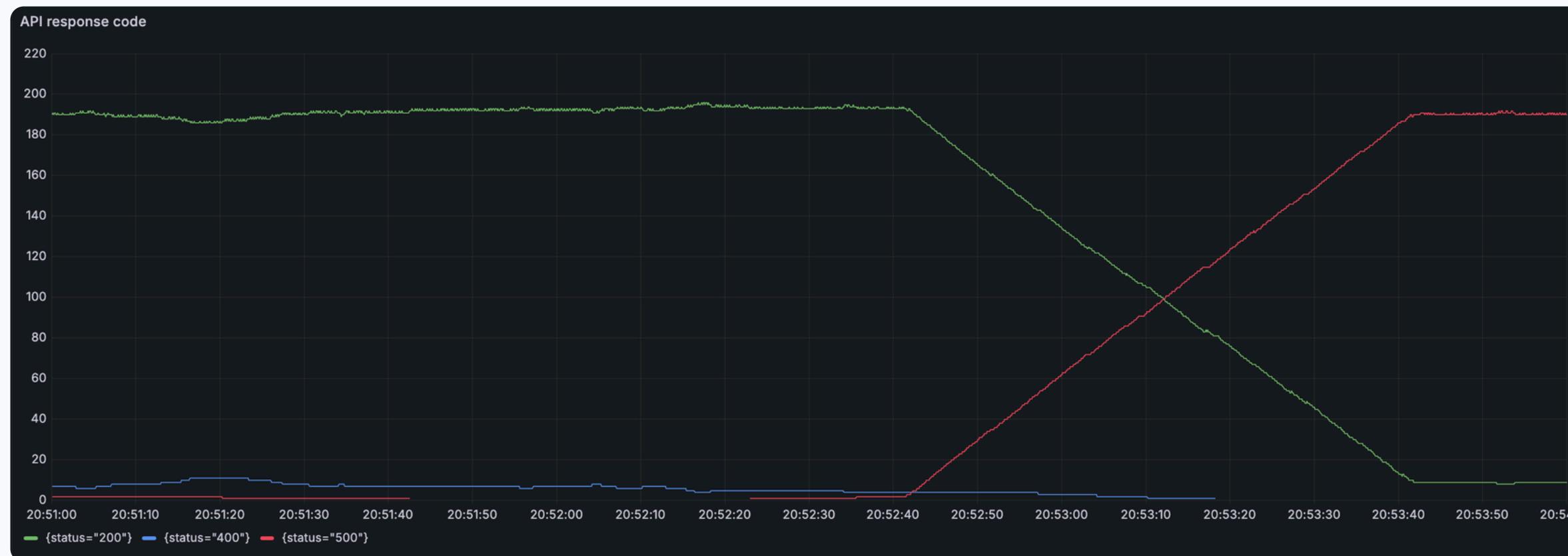
The screenshot shows the Grafana interface for visualizing logs. At the top, there are search filters for environment (production), level (All), message (Enter value), data (Enter value), and requestId (Enter value). The main area displays a table titled "New panel" with columns: time, level, requestId, msg, and payload. The table contains four log entries from October 4, 2025, at 08:59:57. The first entry is an info-level message about an HTTP response sent. The next three are debug-level messages about database queries executed. The right side of the interface shows the "Visualization" panel, which includes tabs for Table, Panel options, Table, Table footer, Cell options, Standard options, Data links and actions, Value mappings, Thresholds, Override 1, Override 2, Override 3, Override 4, Override 5, and Override 6. Below the table, the "Queries" tab is selected, showing a single query named "A" (grafanacloud-lucaraveri993-logs) with the following logstash filter configuration:

```
{source="grafana-demo", environment=~"${environment}", level=~"${level}"}
|~ "${data}"
| json
| msg =~ ".*$${message}.*"
| payload_requestId =~ ".*$${requestId}.*"
```

The "Code" tab is also visible below the configuration.

# Alerting Strategies

- Single error detection → trigger on critical errors
- Error rate spikes (RED method) → alert on sudden increase in errors



# Results

- Strong team adoption and enthusiasm.
- Significantly improved **MTTD** and reduced **MTTR** from 2 days to 2 hours, enhancing service quality.
- Handled 1 GB/day with 2-week retention (14 GB total), running smoothly on a €6/month instance.

# Thank you!

## Questions?

# Demo Repository

