● ALL SYSTEMS OK

Everything is Green Until It Is Not.

Metrics · Logs · Traces · The Gray Zone of AI Observability

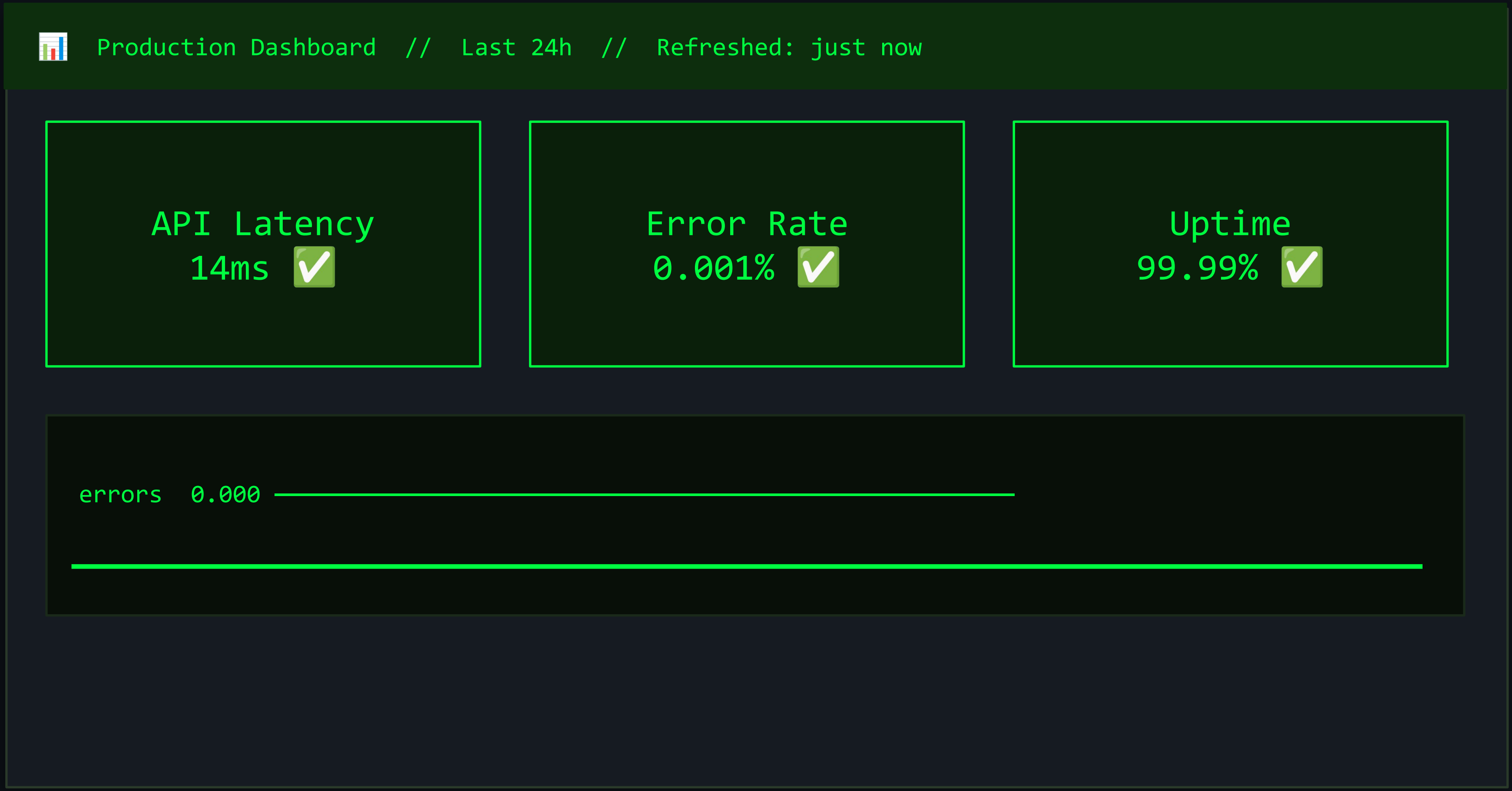
From the trenches: Real stories, hard lessons, and the scars to prove it

Aman Pruthi
Senior Solutions Architect (R systems)
https://www.linkedin.com/in/amanpruthi

THE PLAN

The Agenda

SCENE: MONDAY MORNING


📊 Production Dashboard // Last 24h // Refreshed: just now

API Latency
14ms ✅

Error Rate
0.001% ✅

Uptime
99.99% ✅

errors 0.000

Because no alert was tied to **user success criteria or domain-specific SLIs (e.g., checkout success rate)**, nothing ever went red.

This is the classic *false comfort* — everything *looked* fine technically, but user-impacting failures *slipped through* undetected.

Let's Talk About The Three Pillars

📈

METRICS

Numbers over time.
Cardinality traps await.

EARLY WARNING SYSTEM

📋

LOGS

Text events.
Structure them or suffer.

CONTEXT ENGINE

🔍

TRACES

Distributed requests.
Where did 4s go?

THE DETECTIVE

THE BUDGET QUESTION ?

How Much Should Observability Cost?

~1-15%

of Cost of goods sold

Large orgs around 10 %
Cost conscious orgs around 5% or less
Ref: Steve Flanders Book (mastering Otel and Observability)

WHAT TO ACTUALLY MEASURE

RED + USE: Two Mental Models

RED — for SERVICES

R **Rate**

Requests per second hitting the service

E **Errors**

Fraction of requests that are failing

D **Duration**

How long requests take → use p99, not avg

→ Use for: APIs, microservices, endpoints

USE — for RESOURCES

U **Utilization**

% of time the resource is busy

S **Saturation**

How much extra work is queued/waiting

E **Errors**

Device-level errors: disk, packets dropped

→ Use for: CPUs, disks, DBs, queues, network

RED = Is my service healthy? USE = Is my infrastructure healthy?

METRICS

Cardinality Will Destroy You

❌ CARDINALITY BOMB

```
http_requests_total{
  user_id="{uuid}",
  request_id="{uuid}",
  session_id="{uuid}"
}
```

✅ LOW CARDINALITY

```
http_requests_total{
  method="POST",
  route="/api/users",
  status="200"
}
```

⚠️ **High cardinality = your TSDB goes bankrupt. Prometheus doesn't forget.**

THE METRIC THAT LIES

```
avg(response_time) = 12ms ✅
```

⬇️ *then you look at a histogram...*

```
p50 = 8ms      p90 = 42ms      p99 = 4,800ms
                  🔥
```

Averages are liars. Always use percentiles.

Structure or Suffer

❌ UNSTRUCTURED — enjoy your grep session

```
[2024-01-15 14:32:01] ERROR Failed to process request for user abc123 - timeout
after 30000ms, retry 3/3
```

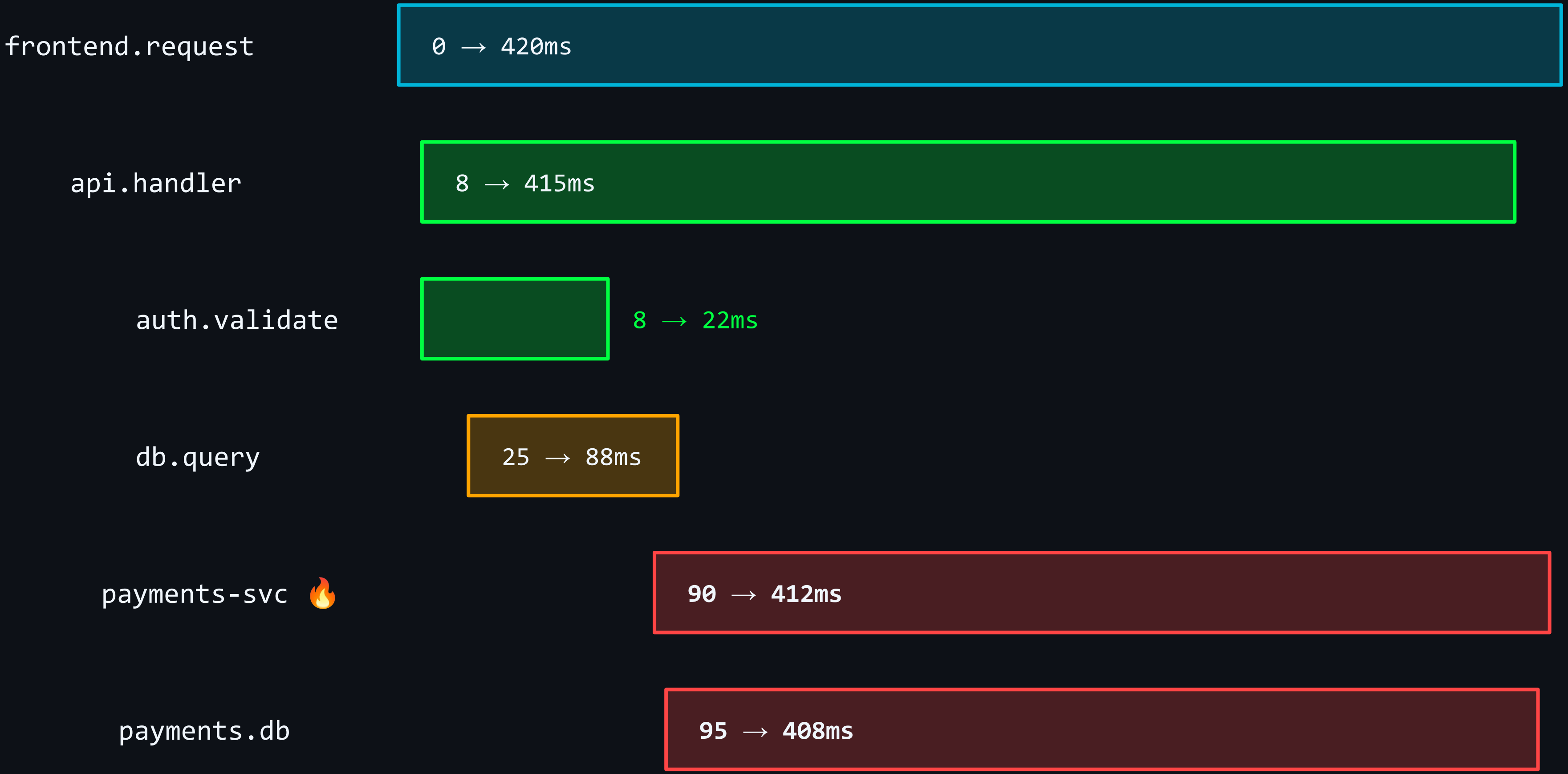
✅ STRUCTURED JSON — queryable, alertable, loveable

```
{ "ts": "2024-01-15T14:32:01Z",  "level": "error",
  "msg": "request_failed",
"user_id": "abc123",  "retry": 3 }
```

💡 Structured logs = O(1) insight. Unstructured = O(pain).

TRACES

4 Seconds Went Somewhere.



Traces tell you exactly where the time went. payments-svc owes you 4 seconds.

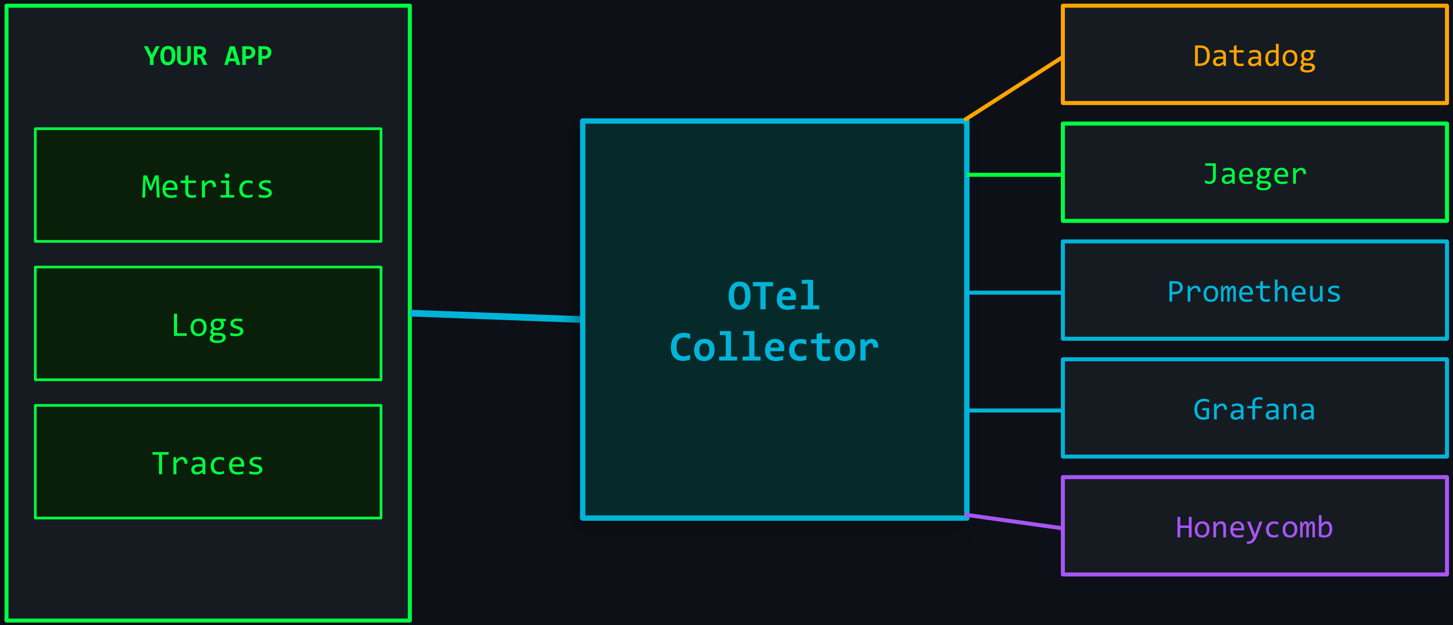
OPENTELEMETRY

Instrument Once. Ship Anywhere.

```
Before OTel: change vendor → rewrite ALL
instrumentation
After OTel:  change vendor → update one config line
```

1 SDK

∞ backends



No more vendor lock-in. Swap backends without touching your app code.

NOW ADD AI TO YOUR STACK

Green → Gray

Everything you knew still applies.
But the failure modes are different.

```
Before: cpu.usage > 90%  →  📟 PAGE
Now:    LLM is... confidently wrong? Quietly drifting?
```

AI OBSERVABILITY

New Signals for a New Problem

🎭 Hallucination Rate

Model invents facts confidently.
No exception. No 500.

🌊 Relevance Drift

Output quality silently degrades
as context shifts.

💸 Token Cost Explosion

Input cardinality = $$$.
Sound familiar?

🔄 Context Abuse

128k tokens 'just in case'.
Latency goes brr.

🎯 Semantic Correctness

Valid JSON, meaningless output.
How do you alert?

THE HARD QUESTION

```
Conventional:
Latency > 500ms → PagerDuty → 😱
```

```
LLM System:
Hallucination > ???%  →
¯\_(ツ)_/¯
```

The answer: LLM-as-Judge + embedding distance + human evals

Measure	Traditional	LLM / AI
Correctness	`assert result == X`	LLM-as-Judge score
Latency	p99 < 200ms	TTFT + tokens/sec
Cost	server $$$	token cost / query
Failure	HTTP 5xx	graceful degradation?
Drift	version pinned	RAG chunk freshness

AI OBSERVABILITY

Evals Are Your New Unit Tests

WHAT TO EVAL

- 🎯 Factual accuracy
- 🌀 Relevance to query
- ☠️ Toxicity / bias
- 🧱 Format adherence
- 🏃 Latency + token cost
- 🎭 Hallucination score

TOOLS

LangSmith	traces + evals
Arize Phoenix	OSS observability
OpenTelemetry	Gen AI semconv
RAGAS	RAG eval framework
Braintrust	eval + prompt mgmt

OTel now has Gen AI Semantic Conventions (1.0 spec). Ship it.

YOUR OBSERVABILITY MIND MAP

Start Here. Ship This. In This Order.

OBSERVABLE SYSTEM

1. **Instrument with OTel**
   Auto-instrumentation first
2. **Structure your logs**
   JSON + trace_id correlation
3. **Define RED metrics**
   Rate, Errors, Duration / service
4. **USE for infra**
   CPU, memory, disk, queues
5. **Add AI/LLM evals**
   Hallucination + drift
6. **SLOs over dashboards**
   Error budgets, burn rates
7. **Cardinality governance**
   Review labels quarterly
8. **Budget audit**
   Target ~15% of cloud spend

Foundation first → Mature later → Budget always. Green is earned, not assumed.