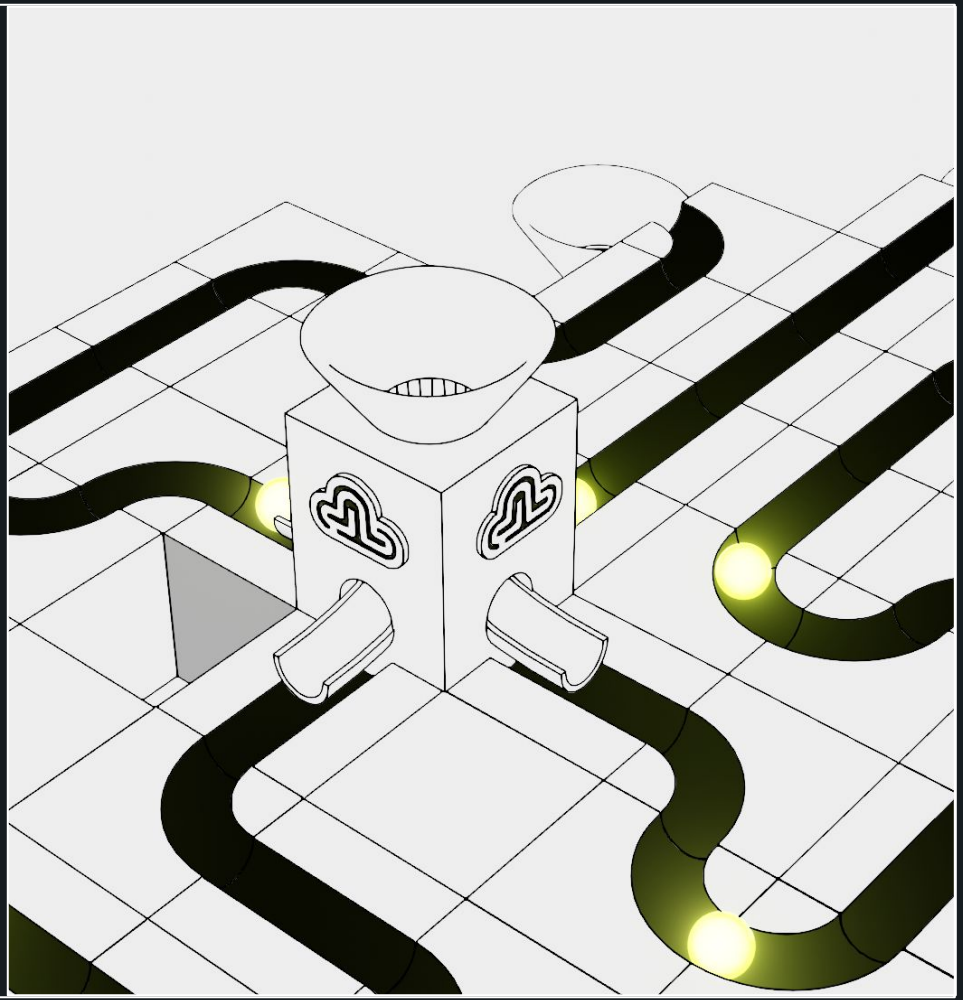
Application Observability

[Real life Schrodinger's cat]



LABYRINTH LABS

LABYRINTH LABS

We are One-stop-shop for [DevOps, Cloud & Kubernetes]

25
Top cloud experts

100+
Kubernetes clusters

aws PARTNER Advanced Tier Services

- Solution Provider
- Amazon EKS Delivery
- Amazon RDS Delivery
- Well-Architected Partner Program
- Amazon OpenSearch Service Delivery
- Migration and Modernization Services Competency

Our mission

[Transforming businesses to become cloud native and future ready]

About me

Adam Hamšík

CEO & Co-Founder

Personal Profile

One of 3 founders of Labyrinth Labs.

Profession

I really enjoy working on Open Source projects & being able to help.

Work Experience

I've been a DevOps engineer for several years, enjoying some kernel hacking with NetBSD in my free time.

Hobbies

Family, hiking, lacrosse + basketball (sports)

LABYRINTH LABS

WTF is Schrodinger's cat ?

LABYRINTH LABS

TL;DR

LABYRINTH LABS

- **Current state Infrastructure Architecture**
  - Distributed systems vs Dynamic systems
  - Horizontal vs Vertical Scaling
  - Availability vs Reliability
  - Microservices vs Monoliths vs Event Driven Architecture
  - Serverless
  - Chaos Engineering
- **What does observability mean?**
  - Why is it important?
- **Observability Stack**
  - Description and short intro
- **How do we do it then ?**

LABYRINTH
LABS

Quick Wins

- Use libraries for **logging, metrics and traces**
- **No deploys** to production on **Fridays** and **holidays**
- Avoid shiny new **object/framework** syndrome
- printf **!=** Logging / Debugger

Part I

[Modern Infrastructure Architecture]

Examples how we run apps these days.

LABYRINTH LABS

Distributed systems

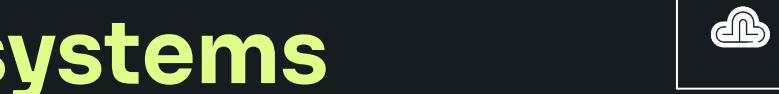
LABYRINTH LABS

Distributed System

A system where components are spread across multiple physical or virtual machines and communicate over a network.

- Multiple **machines** work as a single **logical** entity
- Each **machine** (node) may handle a specific task.
- Helps with **scalability and fault tolerance.**

Dynamic systems



Dynamic System

A system that changes its structure or behavior automatically in response to conditions (like load or failure).

- **Automatically scales up/down** based on demand.
- Can **self-heal** if a part fails
- **Adapts** in **real time** = cost-effective + resilient.

Horizontal vs Vertical Scaling

LABYRINTH LABS

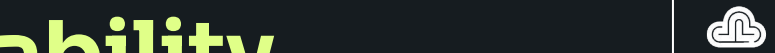
Vertical

- Using bigger machines and bigger machines (Hulk)
- Less reliable
  - One machine fails and you are done.

Horizontal

- Adding more machines (Naturo)
- More complicated architecture
- Can survive failure of multiple machines

Availability vs Reliability



Reliability

- The likelihood of a service failing
- To increase this you have to remove single points of failure.

Availability

- **Availability** = Reliability x Maintenance

- The percentage of time a service is operational

Monolith as software architecture pattern

LABYRINTH LABS

A **monolith** is a **single**, **tightly** coupled application where all features are **packaged** and **deployed** as one unit.

Pros

- Easier to start
- No network latency between components
- Easy for developers to contribute to.

Cons

- Can be **harder** to scale later
- Longer deployment cycles
- **Tight coupling = harder to test or isolate bugs**

Microservices

LABYRINTH LABS

Microservices are an architectural pattern where an application is **split** into **small**, **independent services**, each responsible for a specific business capability.

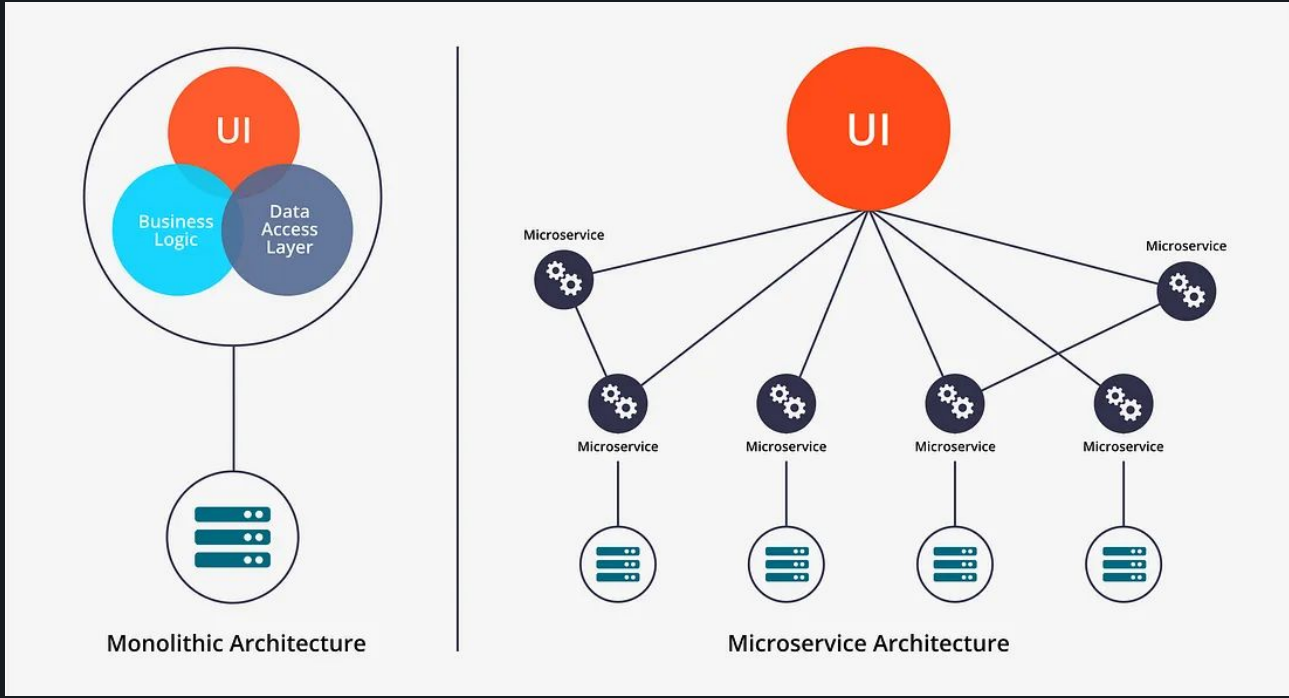
Pros

- Independent deployments
- Better Scalability per service
- Fault isolation.
- Required if you have multiple product teams.

Cons

- Increased complexity
- Requires DevOps & automation maturity
- **Harder** debugging
- API Versioning, Compatibility

Microservices vs Monolith



https://medium.com/startlovingyourself/microservices-vs-monolithic-architecture-c8df91f16bb4

Microservices vs Monolith

https://devcamp.com/site_blogs/monolith-vs-microservice-rails-applications


WE REPLACED OUR MONOLITH WITH MICROSERVICES
SO THAT EVERY OUTAGE COULD BE MORE LIKE A MURDER MYSTERY

Event Driven Architecture

LABYRINTH LABS

Event-Driven Architecture (EDA) is a software pattern where components communicate by producing and reacting to **events** (messages), rather than **calling each other directly**.

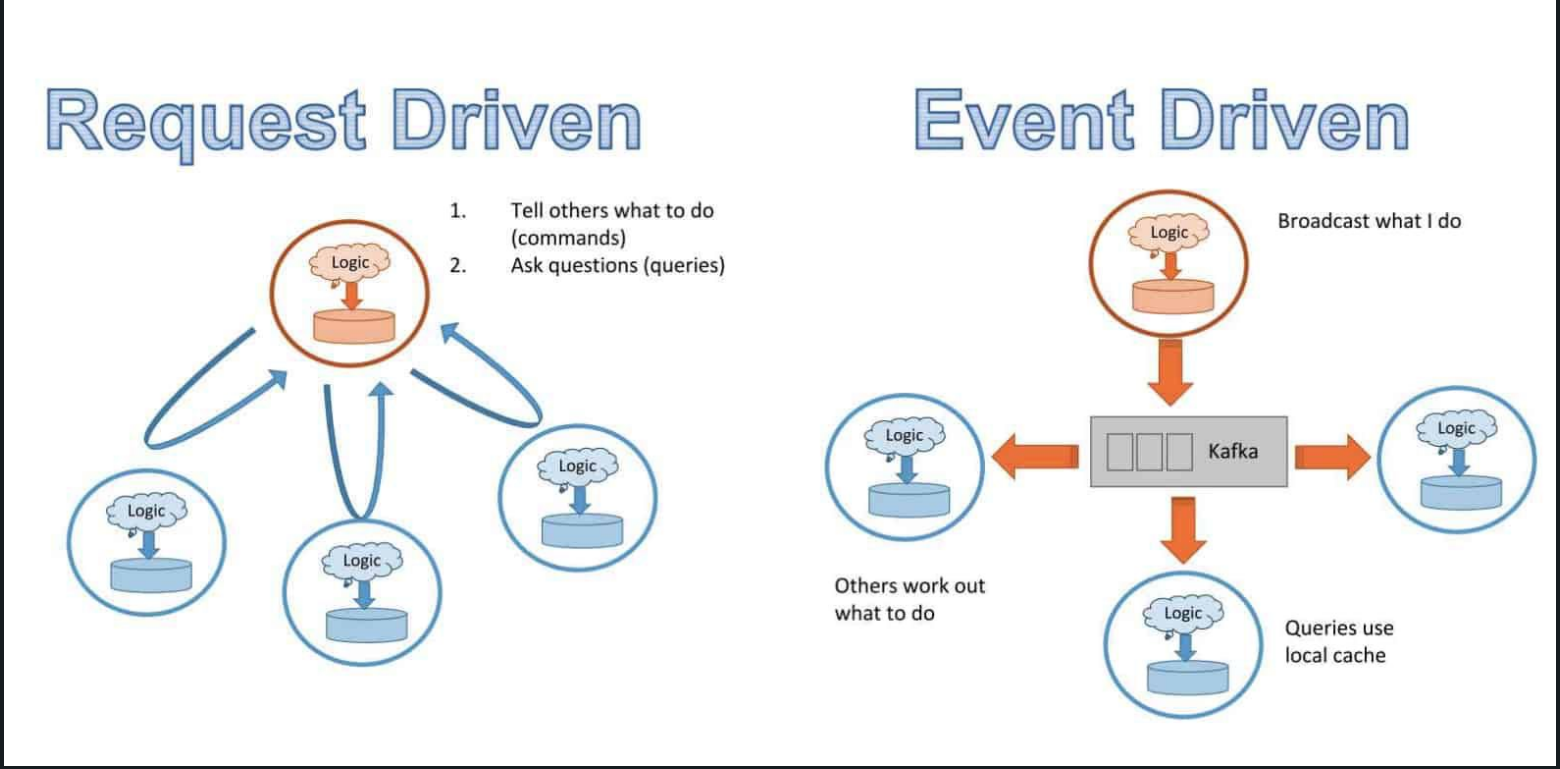
Pros

- Highly decoupled
- Scalable
- Async by default
- Required if you have multiple product teams.

Cons

- **Hard** debugging
- API Versioning, Compatibility
- Event ordering can be tricky
- Visibility is tough without good observability tools

Microservices vs Monolith



https://hazelcast.com/foundations/event-driven-architecture/event-driven-architecture/

Serverless Computing

LABYRINTH LABS

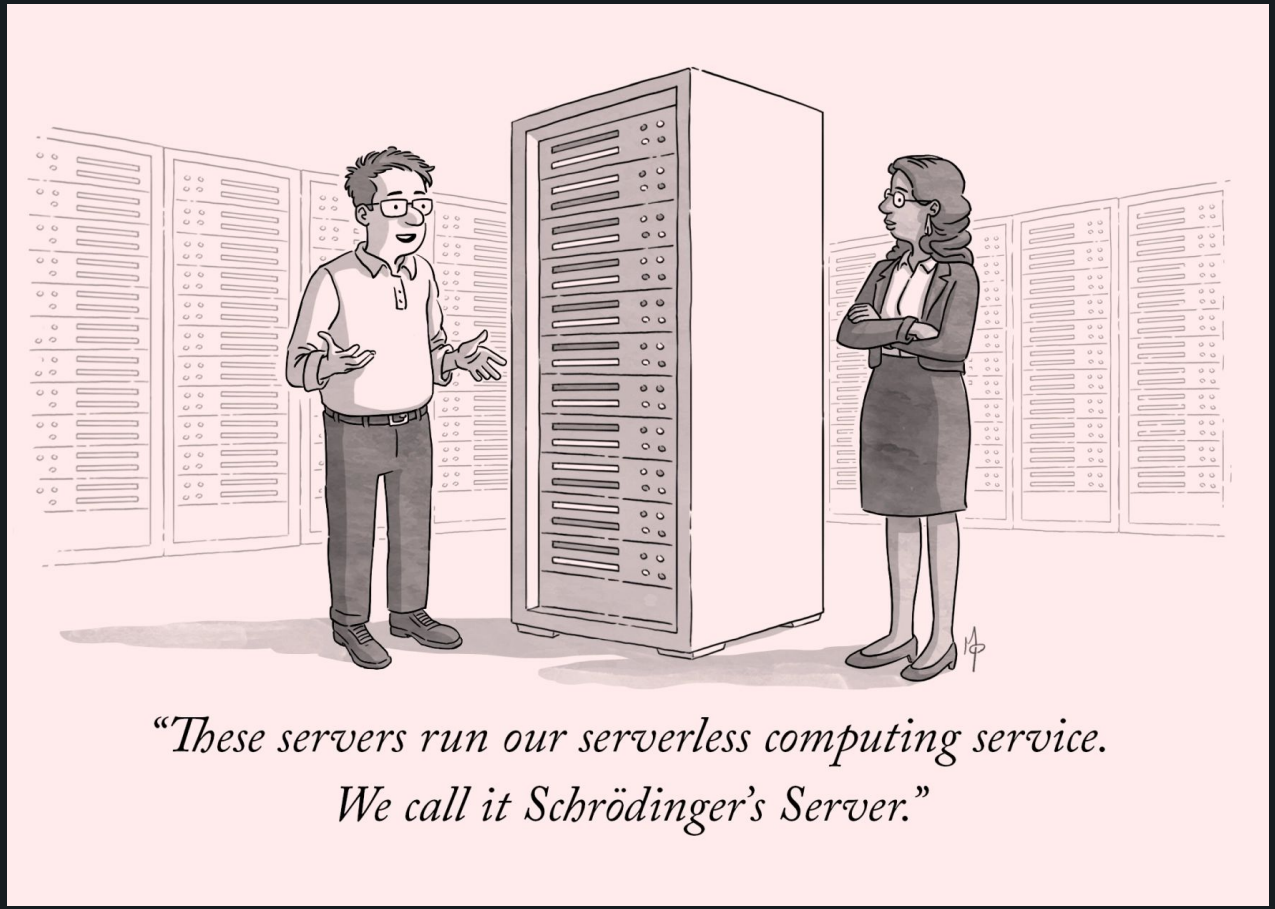
Serverless is a cloud-native architecture where developers write and deploy code without managing servers. Cloud providers handle all the infrastructure, scaling, and runtime management.

Pros

- No server management
- Automatic scaling based on usage
- Great for event-driven use cases and microservices

Cons

- **Limited** execution time
- Not ideal for long-running or stateful processes
- Harder to debug locally



LABYRINTH LABS

https://deploy.equinix.com/blog/schrdingers-server/

Chaos Engineering

LABYRINTH LABS

Practice of **intentionally** breaking your system in **controlled ways** to understand how it behaves under failure.

🧠 “If we know things will break, let’s test them before they break for real.”

E.g. Inject failures (e.g., kill services, delay responses, drop network)

Part II
[Observability Theory]

Some Theory

LABYRINTH
LABS

LABYRINTH LABS

Observability Definition

Observability is a measure of how well internal **states of a system** can be inferred from knowledge of its **external outputs**.

In **control theory**, the **observability** and **controllability** of a linear system are mathematical duals.

The concept of observability was introduced by the Hungarian-American engineer Rudolf E. Kálmán for linear dynamic systems. [1][2]

https://en.wikipedia.org/wiki/Observability

Observability Software Definition

LABYRINTH LABS

Observability is the ability to **collect** data about programs' **execution**, modules' internal states, and the communication among components.

To improve observability, software engineers use a wide range of **logging** and **tracing techniques** to **gather telemetry** information, and tools to analyze and use it.

Observability is **foundational** to site reliability engineering, as it is the first step in triaging a service outage.

https://en.wikipedia.org/wiki/Observability

LABYRINTH LABS

Observability Definition

Concept	Control Theory	Software Observability
System State	Internal physical variables (e.g., speed, position)	Application behavior and health (e.g., request rate, error rate)
Outputs	Measurable outputs (e.g., RPM, temperature)	Logs, metrics, traces from app and infra
Feedback	Adjust inputs to control system	Scale up, restart, alert, throttle

https://en.wikipedia.org/wiki/Observability

LABYRINTH LABS

Observability Definition

Concept	Control Theory	Software Observability
System State	Internal physical variables (e.g., speed, position)	Application behavior and health (e.g., request rate, error rate)
Outputs	Measurable outputs (e.g., RPM, temperature)	Logs, metrics, traces from app and infra
Feedback	Adjust inputs to control system	Scale up, restart, alert, throttle

https://en.wikipedia.org/wiki/Observability

LABYRINTH
LABS

Metrics

-

LABYRINTH
LABS

Logs

Part III

[Observability Stack]

Metrics, Logs, Traces, Profiling ...

LABYRINTH
LABS

Metrics

LABYRINTH
LABS

“Numerical values gathered from your system aggregated over time”

Metrics {example}

LABYRINTH LABS

node_load5{app_kubernetes_io_name="node-exporter",cluster="prod", instance="192.168.1.1:9100", kubernetes_cluster="prod", kubernetes_namespace="monitoring", monitoring_scope="system", namespace="monitoring""} 2.81

- **node_load5** -> metric name
- **kubernetes_cluster="prod"** -> key value pair aka label
- 2.81 -> **Actual value**

Metrics {definitions}

LABYRINTH LABS

- **Metric types**
  - **Counter** – only goes up (e.g., requests total)
  - **Gauge** – goes up/down (e.g., memory usage)
  - **Histogram** – measures latency buckets
  - **Summary** – like histogram, but sampled

- **Cardinality** in observability refers to the number of **unique** combinations of labels (or dimensions) attached to a metric.

http_requests_total{ method="GET", status="200", user_id="123456", uri=”/auth/login”}

RED, USE & Golden Signals

LABYRINTH LABS

- **RED** (Rate, Errors, Duration)
  - for services especially User experience
- **USE** (Utilization, Saturation, Errors)
  - Best for Hardware & infra health
- **Golden Signals** (Google SRE)
  - latency, traffic, errors, saturation
  - Fast insight into service health

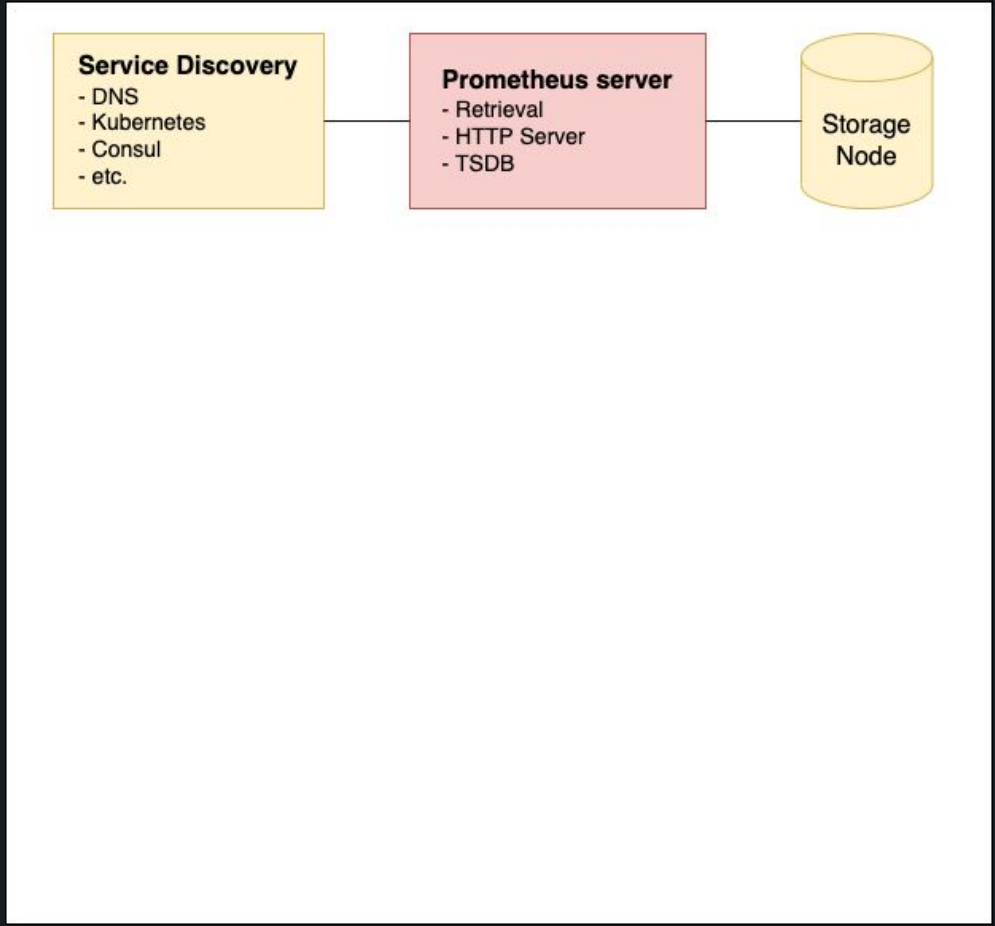
Metrics

Prometheus

- Time series collection happens via a pull model over HTTP
- Kubernetes service discovery
- Instrumentation
  - client libraries
  - exporters

Metric examples

- How much traffic is entering and leaving your network
- How many deployments are you making each day
- How much CPU is your service consuming



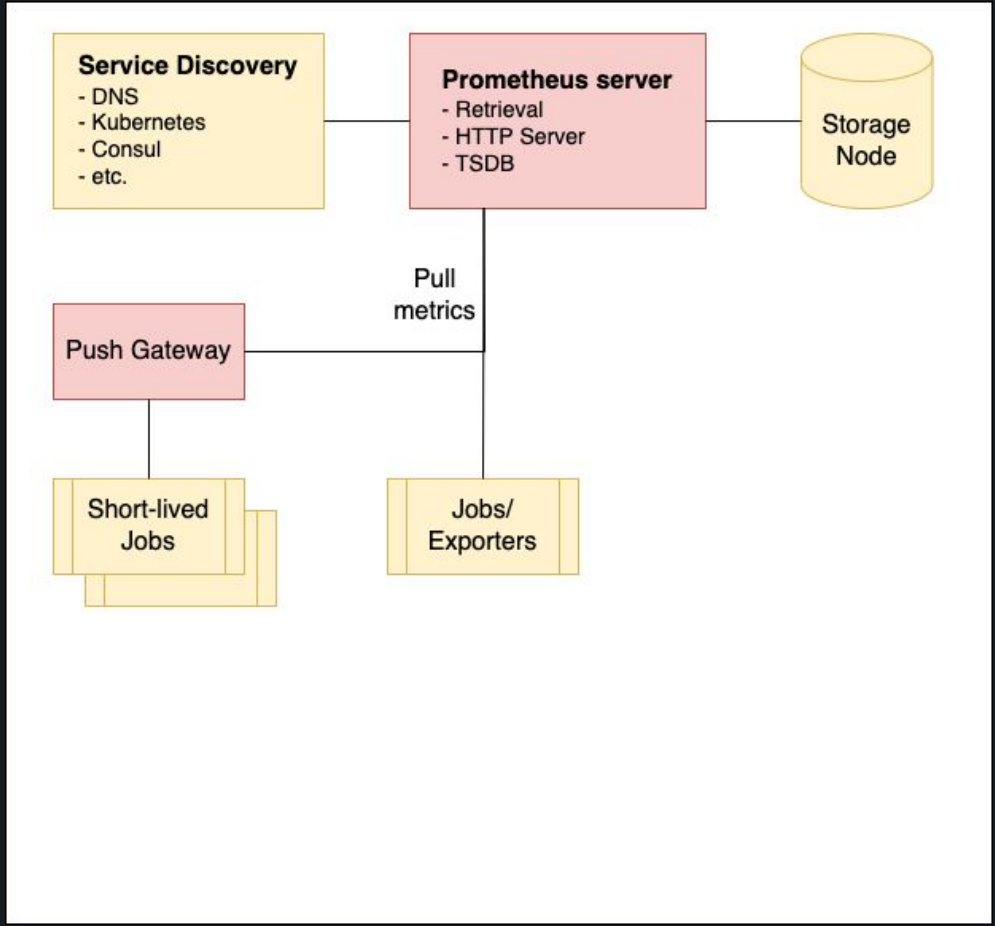
Metrics

Prometheus

- Time series collection happens via a pull model over HTTP
- Kubernetes service discovery
- Instrumentation
  - client libraries
  - exporters

Metric examples

- How much traffic is entering and leaving your network
- How many deployments are you making each day
- How much CPU is your service consuming



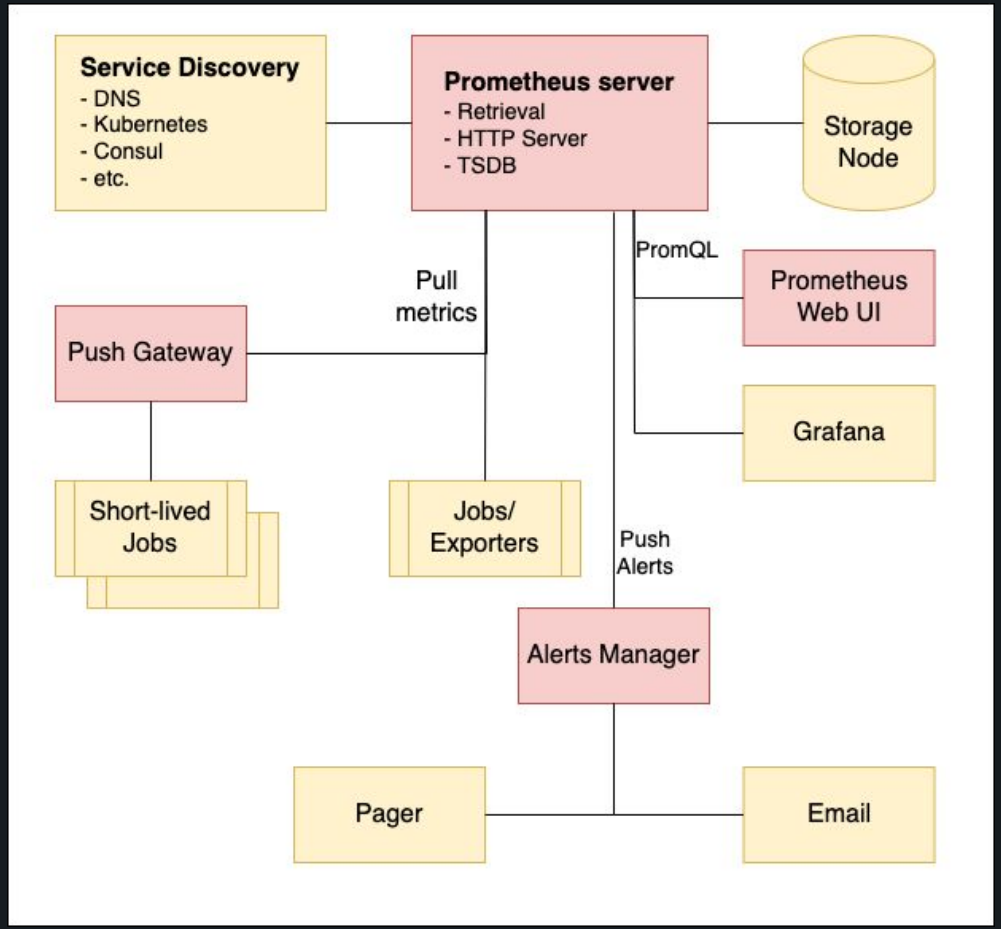
Metrics

Prometheus

- Time series collection happens via a pull model over HTTP
- Kubernetes service discovery
- Instrumentation
  - client libraries
  - exporters

Metric examples

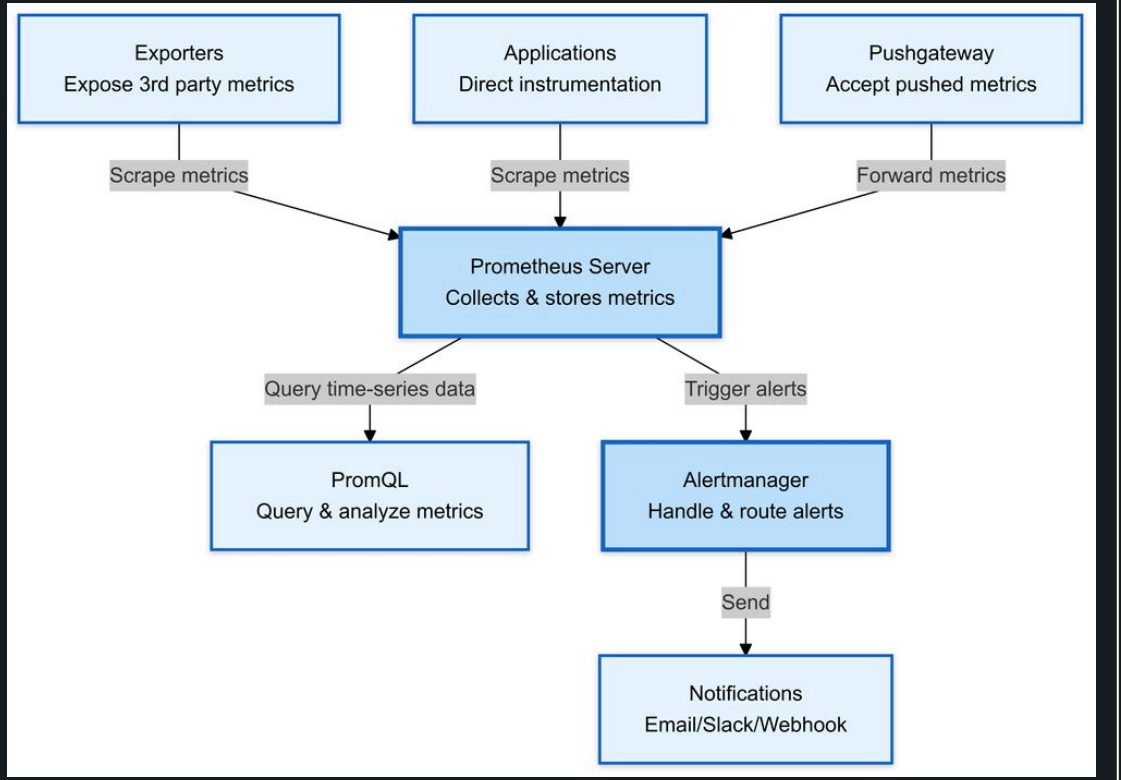
- How much traffic is entering and leaving your network
- How many deployments are you making each day
- How much CPU is your service consuming



Prometheus

LABYRINTH LABS

- Prometheus **collects** and **stores** its metrics as time series data
- **Metadata** are attached to metrics in optional key-value pairs called **labels**
- Has a **multi-dimensional data** model.
- Flexible query language **PromQL**



https://last9.io/blog/prometheus-vs-thanos/

What is Prometheus [intentionally] lacking

LABYRINTH LABS

High availability

- No methods of data deduplication
- No methods for global view

Long term retention

- No concept of distributed data storage/sharding
- Keeping data long term is not feasible

LABYRINTH
LABS

Challenges

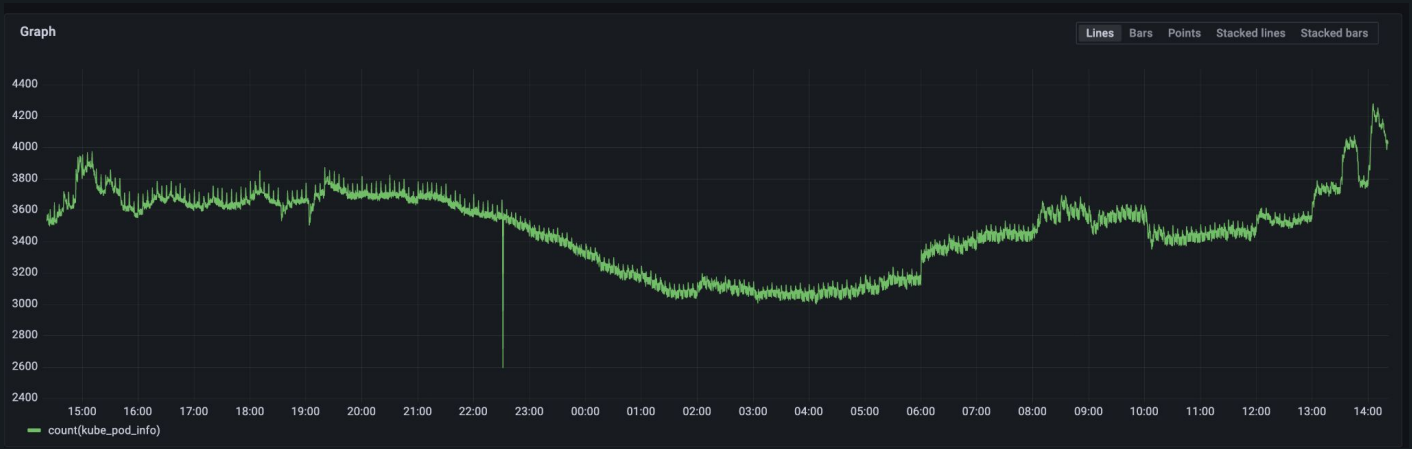
- Prometheus needs to run in a Highly available setup
- We need to keep metrics indefinitely and store them efficiently
- We need a single observer point for all clusters

Challenges

LABYRINTH LABS

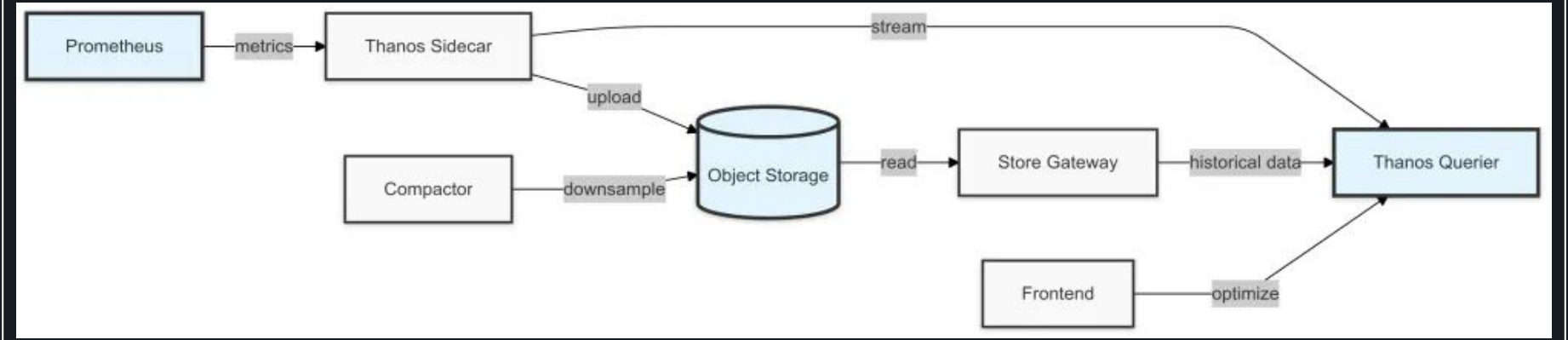
Monitoring 5000+ pods in multiple clusters

19 TBs of metric data
3 TBs ingested each month



Thanos

- Open source, highly available Prometheus setup with long term storage capabilities.
- Requires just object store - AWS S3
- Global view, unlimited retention, high availability



https://last9.io/blog/prometheus-vs-thanos/

Logs

LABYRINTH
LABS

“Records of events that have occurred within a software application or infrastructure.”

Logs {example}

LABYRINTH LABS

```
47.29.201.179 - - [28/Feb/2019:13:17:10 +0000] "GET /?p=1 HTTP/2.0" 200 5316 "https://domain1.com/?p=1"
"Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/72.0.3626.119 Safari/537.36" "2.75"
```

```
log_format custom '$remote_addr - $remote_user [$time_local] "$request" $status $body_bytes_sent "$http_referer"
"$http_user_agent" "$gzip_ratio"';
```

We can see what **type** of request was sent to which URL, from which ip and what was our status code we used to respond.

LABYRINTH LABS

Log Collection

Multiple ways to collect logs

- Application runtime
- Log collector sidecars
- Log collector DaemonSets

Many log collectors...

- fluentd
- fluentbit
- logstash
- Promtail
- vector.dev

LABYRINTH LABS

Log Collection

Customer Example

- Storage 2TB/logov daily
- Throughput 51 MB/s
- During peak 500k logs/sec

Please do not enable debug logs in production. That can lead to serious performance issues.

Amazon OpenSearch Service

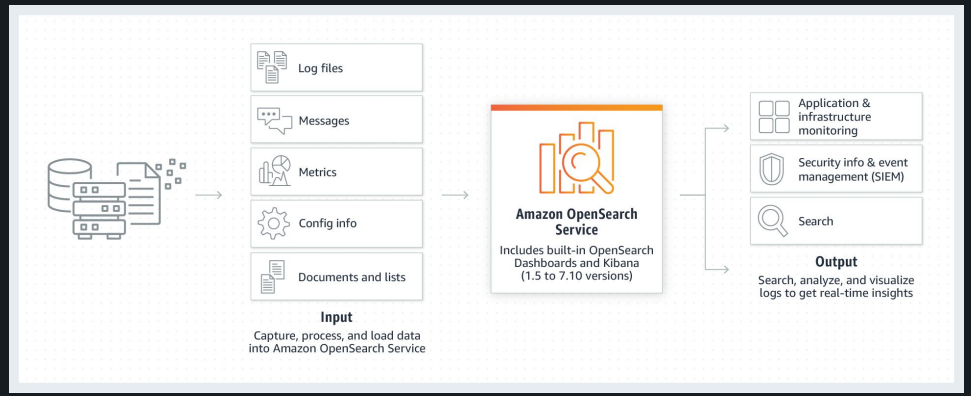
LABYRINTH LABS

fully managed

native integration AWS services

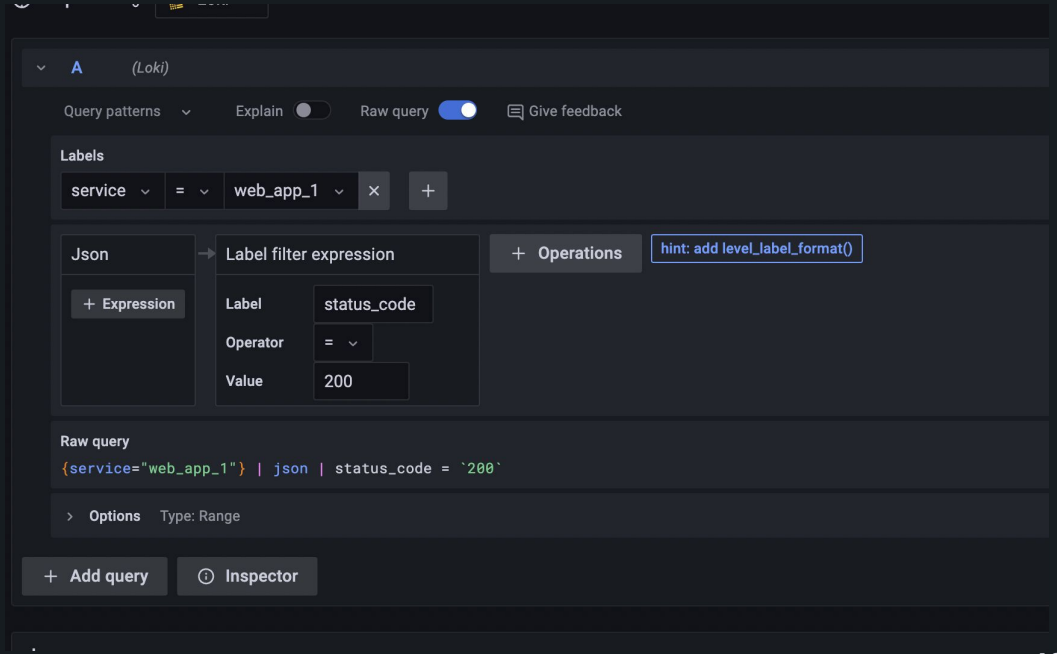
search and analytics engine for use cases such as log analytics, real-time application monitoring, and clickstream analysis

AWS CloudTrail integration



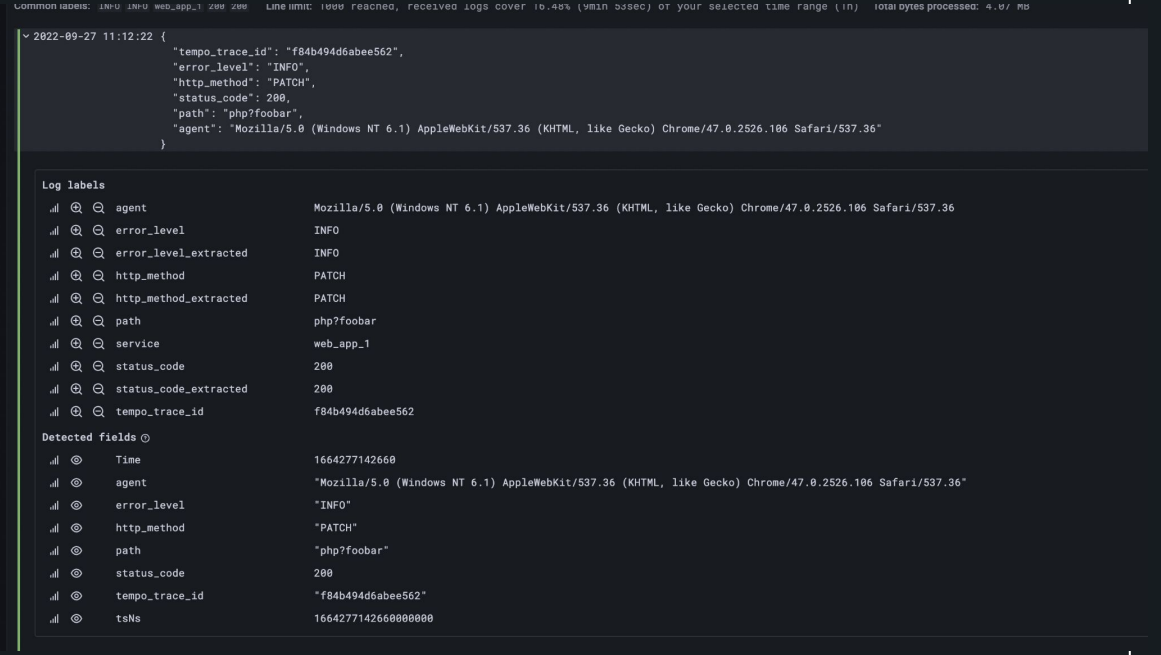
Grafana Loki

- Only indexes labels and metadata
- Simple query language - LogQL
- Multi-tenancy
- Similar alerting concept to Prometheus



Grafana Loki

- Only indexes labels and metadata
- Simple query language - LogQL
- Multi-tenancy
- Similar alerting concept to Prometheus



LABYRINTH
LABS

Traces

Record of user requests across services, networks, and protocols to create a complete picture of how your distributed system works.

Distributed Tracing

LABYRINTH LABS

Tracks user requests across services, networks, and protocols and creates a complete picture of how your distributed system performed while processing a user request.

- Grafana Tempo
- OpenTelemetry, Jaeger, Zipkin

Distributed Tracing

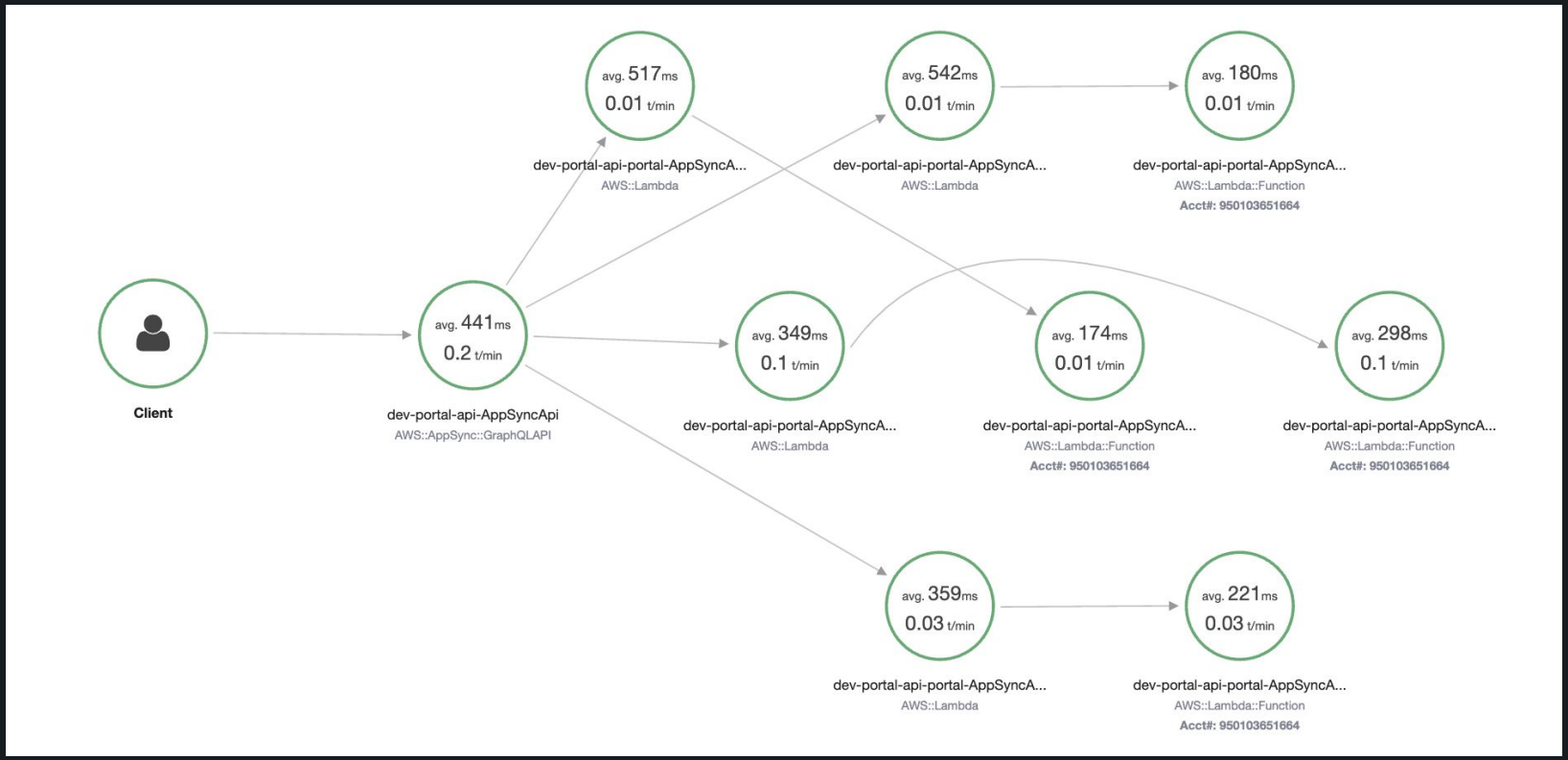
LABYRINTH
LABS

- Each activity (segment or span) is recorded as it through and across services
- Allows us to find and isolate bottlenecks quickly
- Identify the root cause of unseen problems

LABYRINTH
LABS

AWS X-Ray

- Fully managed
- Creates a map of services used by your application with trace data that you can use to drill into specific services or issues
- X-Ray SDK captures metadata for requests made to MySQL and PostgreSQL databases (self-hosted, Amazon RDS, Amazon Aurora), and Amazon DynamoDB

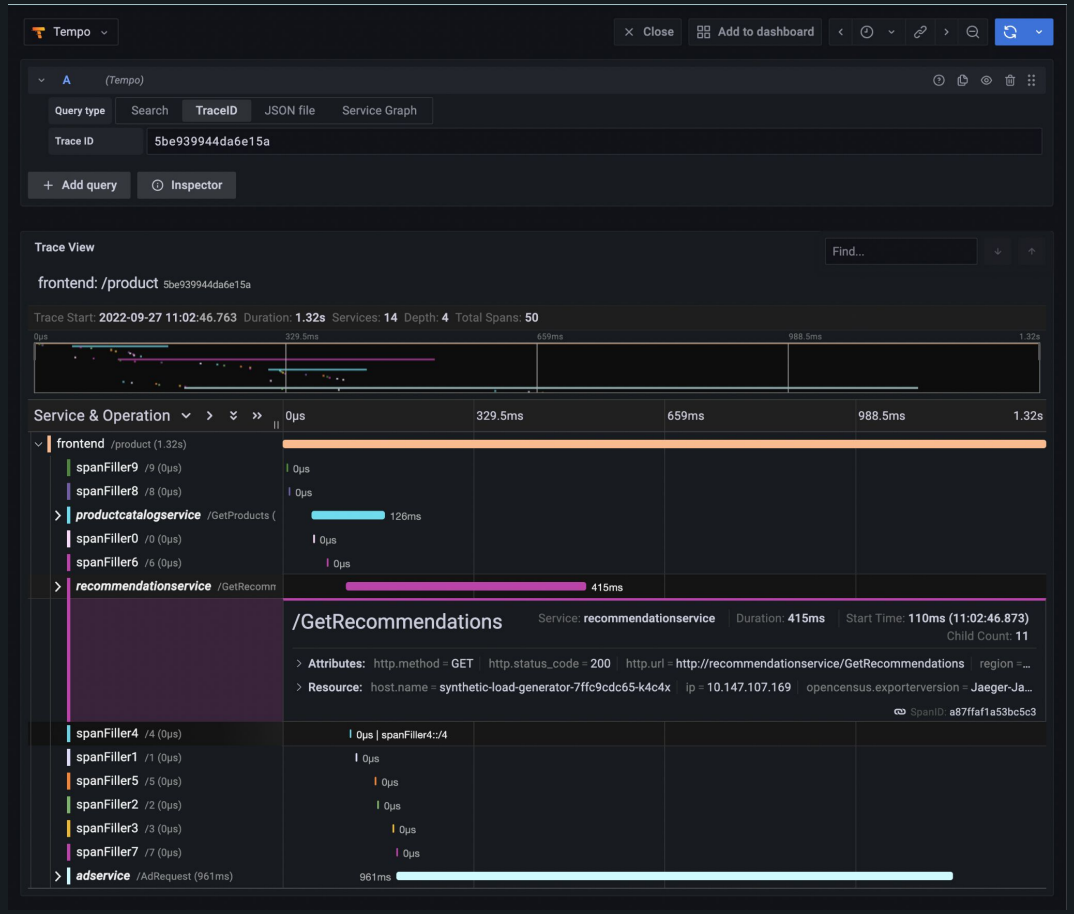

avg. 517ms
0.01 t/min
dev-portal-api-portal-AppSyncA...
AWS::Lambda
avg. 542ms
0.01 t/min
dev-portal-api-portal-AppSyncA...
AWS::Lambda
avg. 180ms
0.01 t/min
dev-portal-api-portal-AppSyncA...
AWS::Lambda::Function
Acct#: 950103651664
Client
avg. 441ms
0.2 t/min
dev-portal-api-AppSyncApi
AWS::AppSync::GraphQLAPI
avg. 349ms
0.1 t/min
dev-portal-api-portal-AppSyncA...
AWS::Lambda
avg. 174ms
0.01 t/min
dev-portal-api-portal-AppSyncA...
AWS::Lambda::Function
Acct#: 950103651664
avg. 298ms
0.1 t/min
dev-portal-api-portal-AppSyncA...
AWS::Lambda::Function
Acct#: 950103651664
avg. 359ms
0.03 t/min
dev-portal-api-portal-AppSyncA...
AWS::Lambda
avg. 221ms
0.03 t/min
dev-portal-api-portal-AppSyncA...
AWS::Lambda::Function
Acct#: 950103651664


Name
Res.
Duration
Status
0.0ms
100ms
200ms
300ms
400ms
500ms
600ms
700ms
800ms
900ms
dev-portal-api-AppSyncApi AWS::AppSync::GraphQLAPI
dev-portal-api-AppSyncApi 200 801 ms
POST z3rtgfxze5bk5fy3yu6e4hkiea.appsync-api
/categories - 0.0 ms
requestMappingTemplateEvaluation - 0.0 ms
/companies - 0.0 ms
requestMappingTemplateEvaluation - 0.0 ms
Query.categories - 719 ms
Lambda 200 693 ms
Invoke: dev-portal-api-portal-AppSyncApiLambdaQ
Query.companies - 720 ms
Lambda 200 699 ms
Invoke: dev-portal-api-portal-AppSyncApiLambd
responseMappingTemplateEvaluation - 0.0 ms
responseMappingTemplateEvaluation - 0.0 ms
dev-portal-api-portal-AppSyncApiLambdaQuerycategor-vi4tFhQkto0I AWS::Lambda
dev-portal-api-portal-AppSyncApiLambdaQueryca 200 661 ms
dev-portal-api-portal-AppSyncApiLambdaQuerycompani-R3PM2AVcwmJI AWS::Lambda
dev-portal-api-portal-AppSyncApiLambdaQueryco 200 668 ms

LABYRINTH
LABS

Grafana Tempo

- Distributed tracing backend
- Cost-efficient, needs just object storage
- Multiple tracing protocols
- Allows us to find and isolate bottlenecks quickly
- Identify the root cause of unseen problems


Tempo
Close
Add to dashboard
A
(Tempo)
Query type
Search
TraceID
JSON file
Service Graph
Trace ID
5be939944da6e15a
Add query
Inspector
Trace View
Find...
frontend: /product 5be939944da6e15a
Trace Start: 2022-09-27 11:02:46.763 Duration: 1.32s Services: 14 Depth: 4 Total Spans: 50
0µs
329.5ms
659ms
988.5ms
1.32s
Service & Operation
frontend /product (1.32s)
spanFiller9 /9 (0µs)
spanFiller8 /8 (0µs)
productcatalogservice /GetProducts (
126ms
spanFiller0 /0 (0µs)
spanFiller6 /6 (0µs)
recommendationservice /GetRecomn
415ms
/GetRecommendations
Service: recommendationservice
Duration: 415ms
Start Time: 110ms (11:02:46.873)
Child Count: 11
Attributes: http.method = GET | http.status_code = 200 | http.url = http://recommendationservice/GetRecommendations | region =...
Resource: host.name = synthetic-load-generator-7ffc9cdc65-k4c4x | ip = 10.147.107.169 | opencensus.exporterversion = Jaeger-Ja...
SpanID: a87ffaf1a53bc5c3
spanFiller4 /4 (0µs)
0µs | spanFiller4::/4
spanFiller1 /1 (0µs)
spanFiller5 /5 (0µs)
spanFiller2 /2 (0µs)
spanFiller3 /3 (0µs)
spanFiller7 /7 (0µs)
adservice /AdRequest (961ms)
961ms

Visualization & Analysis

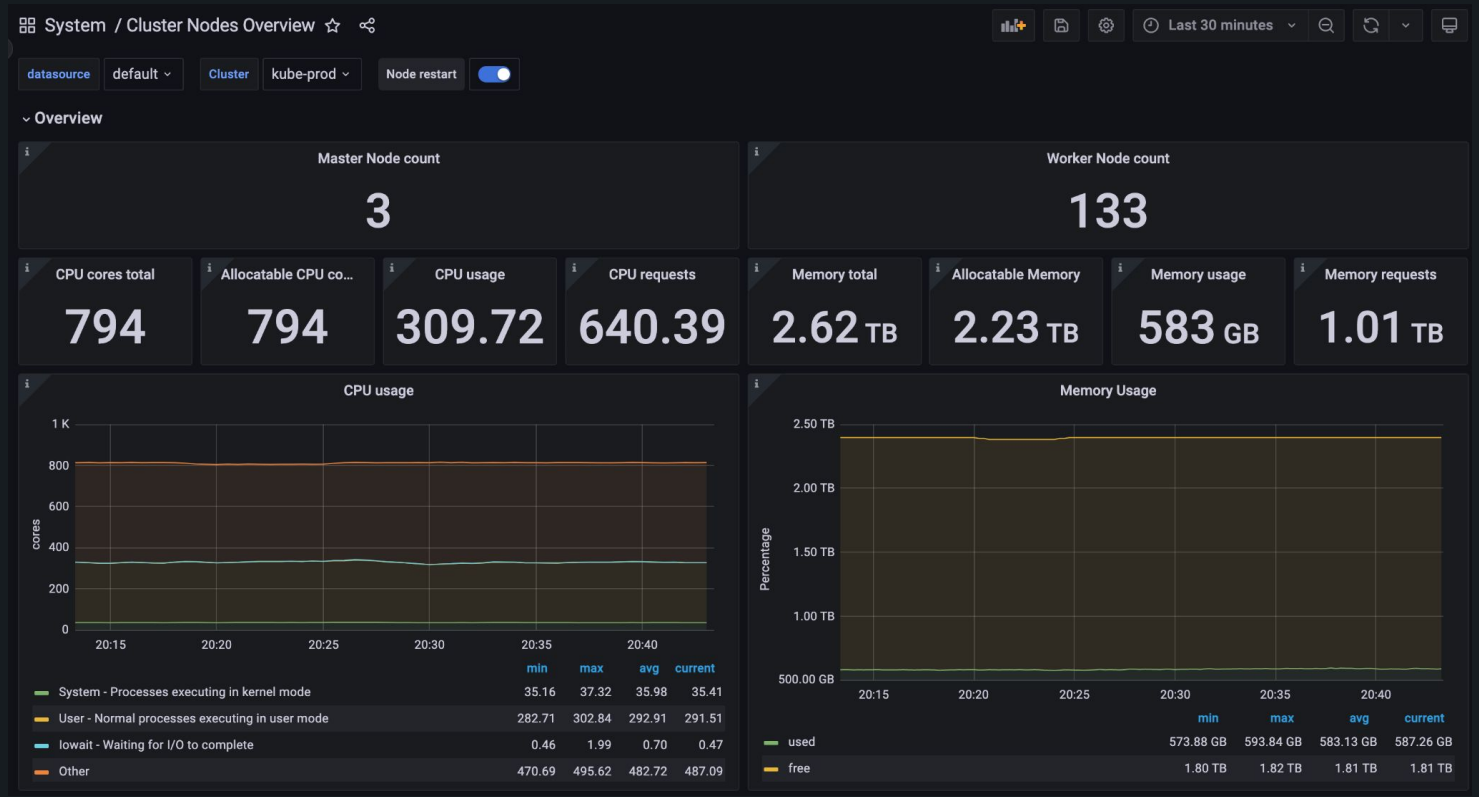
LABYRINTH
LABS

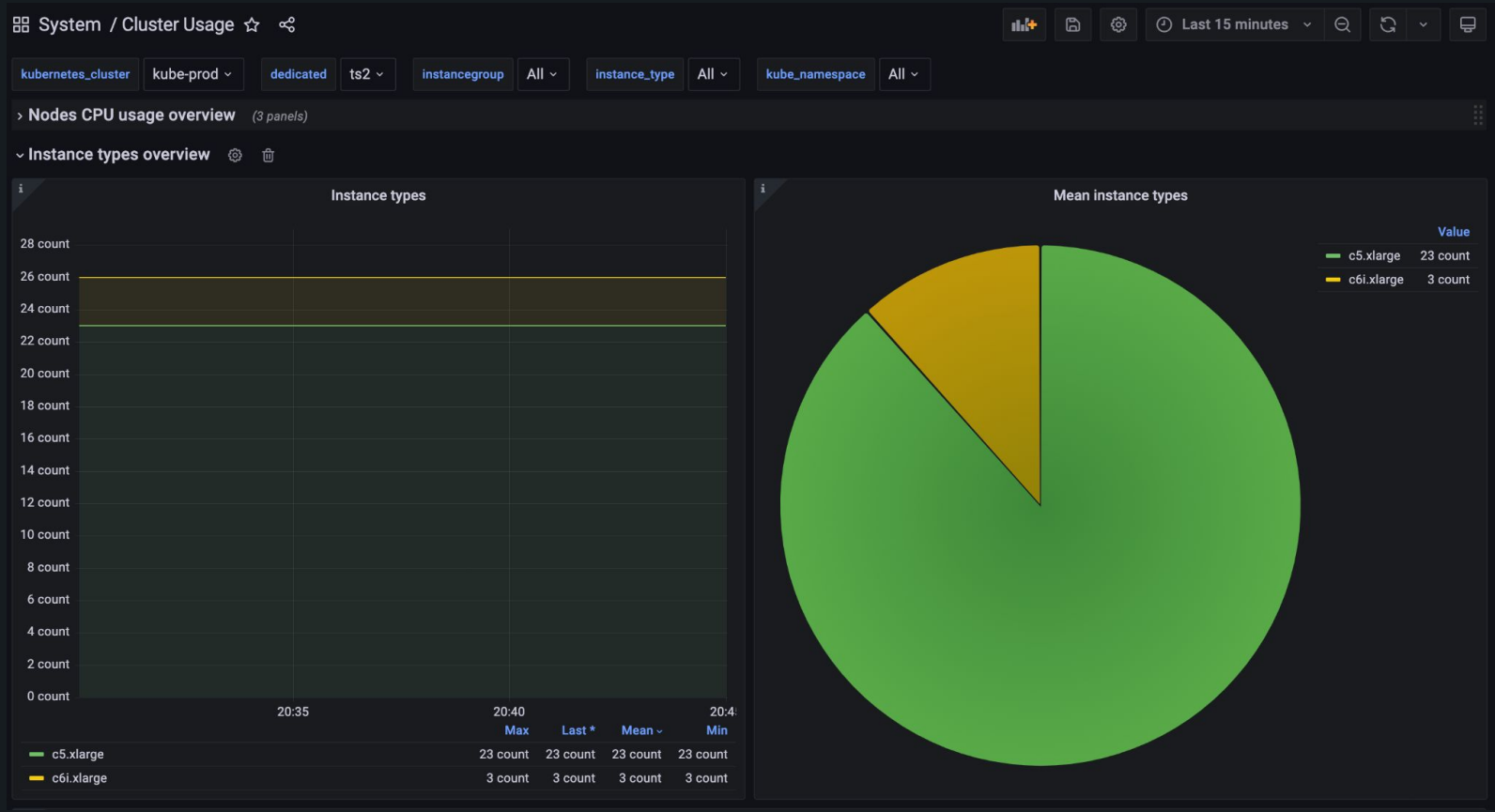
How it all comes together...

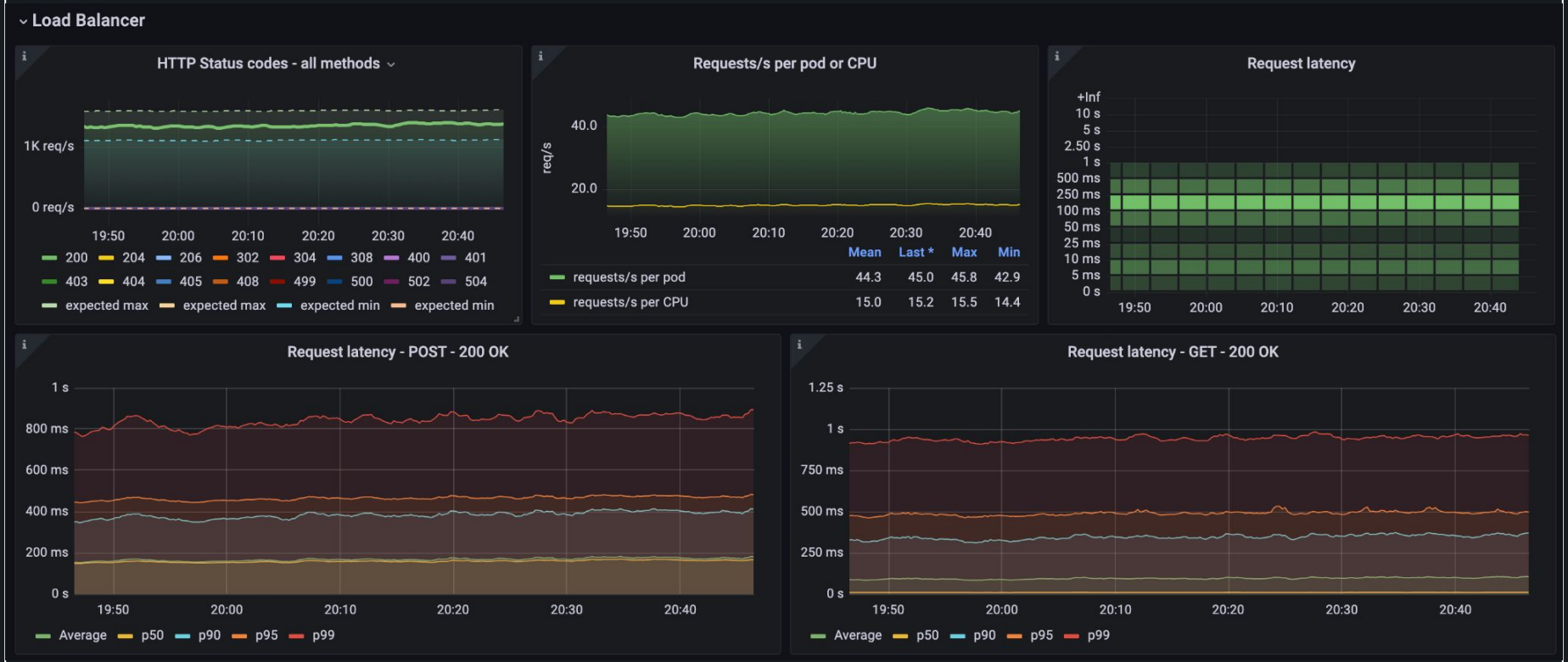
LABYRINTH LABS

Visualization with Grafana

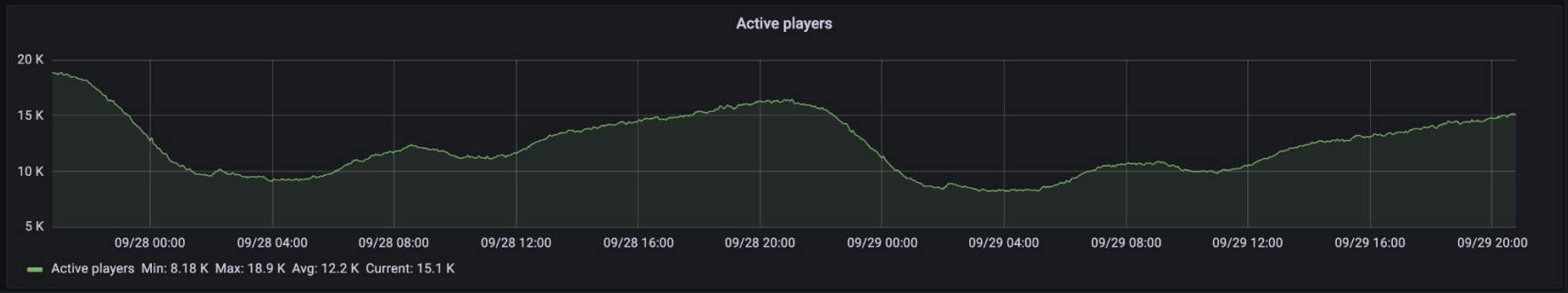
- Single point of view for all metrics, logs and traces
  - Mixed datasources
  - Alerting
  - Open-source (self-hosted)

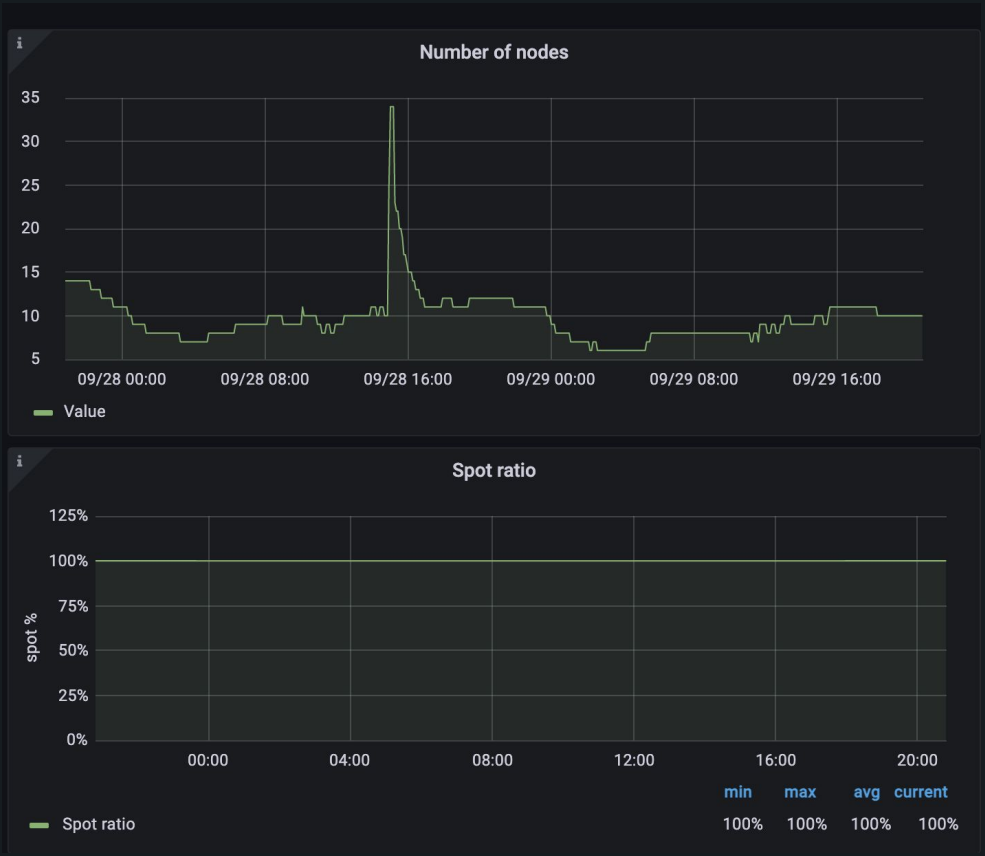

System / Cluster Nodes Overview
Last 30 minutes
datasource
default
Cluster
kube-prod
Node restart
Overview
Master Node count
3
Worker Node count
133
CPU cores total
794
Allocatable CPU co...
794
CPU usage
309.72
CPU requests
640.39
Memory total
2.62 TB
Allocatable Memory
2.23 TB
Memory usage
583 GB
Memory requests
1.01 TB
CPU usage
1 K
800
600
400
200
0
cores
20:15
20:20
20:25
20:30
20:35
20:40
min max avg current
System - Processes executing in kernel mode 35.16 37.32 35.98 35.41
User - Normal processes executing in user mode 282.71 302.84 292.91 291.51
Iowait - Waiting for I/O to complete 0.46 1.99 0.70 0.47
Other 470.69 495.62 482.72 487.09
Memory Usage
2.50 TB
2.00 TB
1.50 TB
1.00 TB
500.00 GB
Percentage
20:15
20:20
20:25
20:30
20:35
20:40
min max avg current
used 573.88 GB 593.84 GB 583.13 GB 587.26 GB
free 1.80 TB 1.82 TB 1.81 TB 1.81 TB

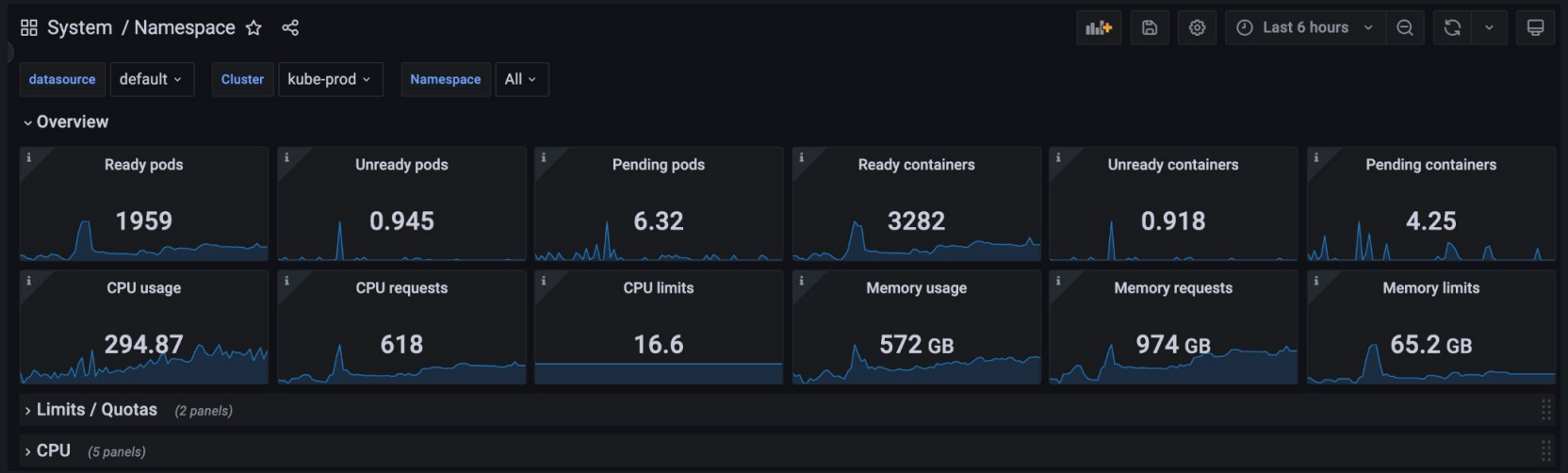

System / Cluster Usage
Last 15 minutes
kubernetes_cluster
kube-prod
dedicated
ts2
instancegroup
All
instance_type
All
kube_namespace
All
Nodes CPU usage overview
(3 panels)
Instance types overview
Instance types
28 count
26 count
24 count
22 count
20 count
18 count
16 count
14 count
12 count
10 count
8 count
6 count
4 count
2 count
0 count
20:35
20:40
Max
Last *
Mean
Min
c5.xlarge
23 count
23 count
23 count
23 count
c6i.xlarge
3 count
3 count
3 count
3 count
Mean instance types
Value
c5.xlarge
23 count
c6i.xlarge
3 count


Load Balancer
HTTP Status codes - all methods
1K req/s
0 req/s
19:50 20:00 20:10 20:20 20:30 20:40
200 204 206 302 304 308 400 401
403 404 405 408 499 500 502 504
expected max expected max expected min expected min
Requests/s per pod or CPU
req/s
40.0
20.0
19:50 20:00 20:10 20:20 20:30 20:40
Mean Last * Max Min
requests/s per pod 44.3 45.0 45.8 42.9
requests/s per CPU 15.0 15.2 15.5 14.4
Request latency
+Inf
10 s
5 s
2.50 s
1 s
500 ms
250 ms
100 ms
50 ms
25 ms
10 ms
5 ms
0 s
19:50 20:00 20:10 20:20 20:30 20:40
Request latency - POST - 200 OK
1 s
800 ms
600 ms
400 ms
200 ms
0 s
19:50 20:00 20:10 20:20 20:30 20:40
Average p50 p90 p95 p99
Request latency - GET - 200 OK
1.25 s
1 s
750 ms
500 ms
250 ms
0 s
19:50 20:00 20:10 20:20 20:30 20:40
Average p50 p90 p95 p99

Game data


Active players
20 K
15 K
10 K
5 K
09/28 00:00
09/28 04:00
09/28 08:00
09/28 12:00
09/28 16:00
09/28 20:00
09/29 00:00
09/29 04:00
09/29 08:00
09/29 12:00
09/29 16:00
09/29 20:00
Active players Min: 8.18 K Max: 18.9 K Avg: 12.2 K Current: 15.1 K


Number of nodes
35
30
25
20
15
10
5
09/28 00:00
09/28 08:00
09/28 16:00
09/29 00:00
09/29 08:00
09/29 16:00
Value
Spot ratio
125%
100%
75%
50%
25%
0%
spot %
00:00
04:00
08:00
12:00
16:00
20:00
min
max
avg
current
Spot ratio
100%
100%
100%
100%


System / Namespace
Last 6 hours
datasource
default
Cluster
kube-prod
Namespace
All
Overview
Ready pods
1959
Unready pods
0.945
Pending pods
6.32
Ready containers
3282
Unready containers
0.918
Pending containers
4.25
CPU usage
294.87
CPU requests
618
CPU limits
16.6
Memory usage
572 GB
Memory requests
974 GB
Memory limits
65.2 GB
Limits / Quotas (2 panels)
CPU (5 panels)

LABYRINTH LABS

Alerting

An alert is a last-resort signal that tells you:

🔥 “The system has gone outside the boundaries you defined as **acceptable** — someone needs to act.”

LABYRINTH LABS

Alerting

Boundaries comes from

- **SLIs/SLOs**
  - latency must stay below 300ms
- **Infrastructure** rules
  - CPU > 90% for 5 minutes
- **Business** **impact** thresholds
  - 0 checkouts in 3 mins
- Hardware based limits

Alerting

LABYRINTH LABS

Really, **really** hard to do

- Normal is **relative**
  - Today's traffic baseline ≠ tomorrow's
- Dynamic infrastructure
  - Pods restart, services autoscale, server die
- **Infrastructure** noise can mask or trigger false alerts

https://sre.google/books/chapters/alerting-on-slos/

https://landing.google.com/sre/sre-book/chapters/alerting-introduction/

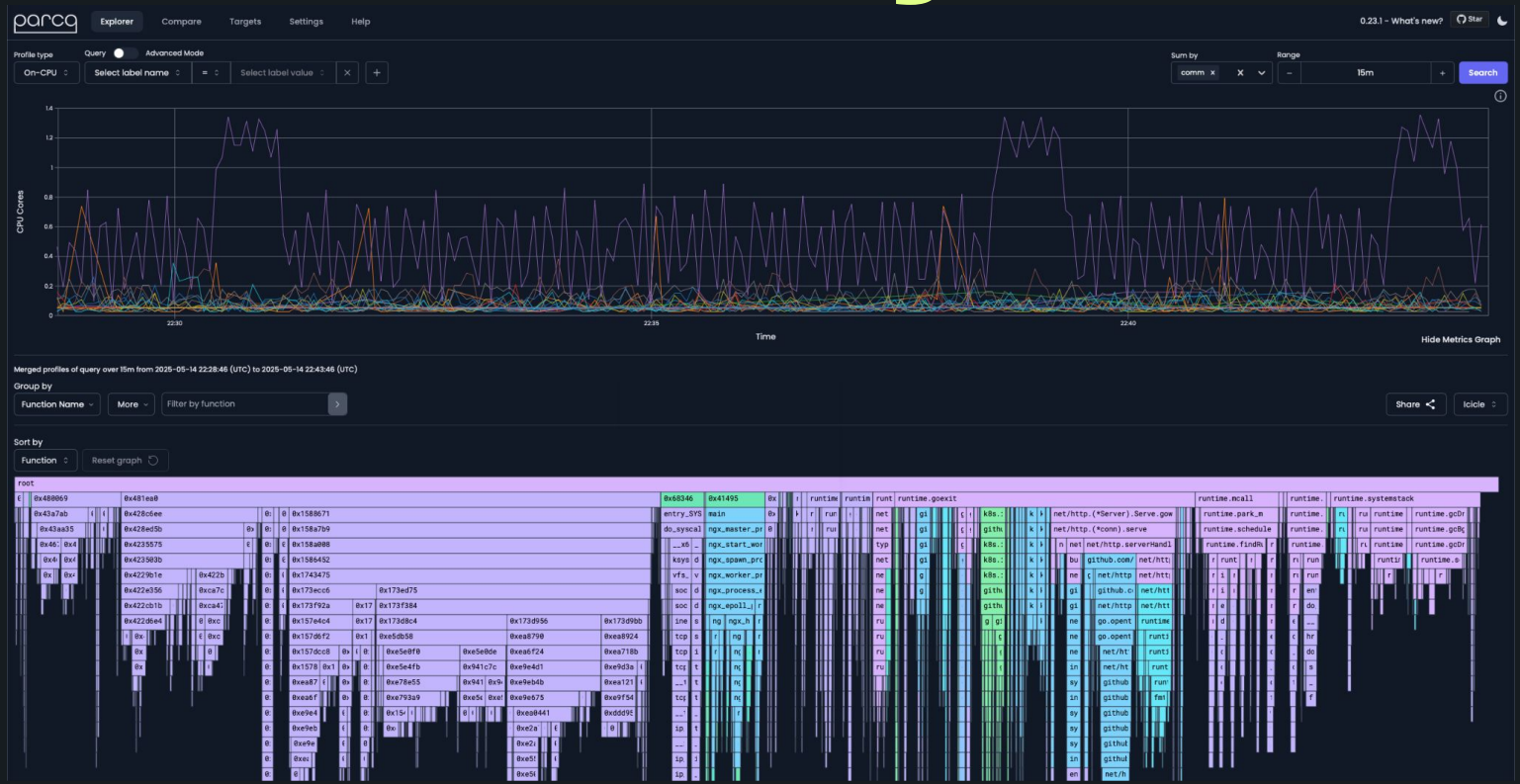
Continuous Profiling

LABYRINTH LABS

Practice of **systematically** collecting and analyzing detailed **performance profiles** from live systems, typically at low overhead.

- Find performance bottlenecks
- Optimize hot code paths (functions burning CPU or memory)
- Reduce cloud costs
- Catch performance regressions

Continuous Profiling



LABYRINTH LABS

Summary

- Pillars of observability
  - Metrics
    - Prometheus
    - Thanos
  - Logs
    - OpenSearch
    - Grafana Loki
  - Traces
    - AWS X-Ray
    - Grafana Tempo

Part VI
[Observability Prax]

Implementation & lessons learned

Labyrinth Labs

[Approach]

-

Part IV
[Q&A]

LABYRINTH
LABS

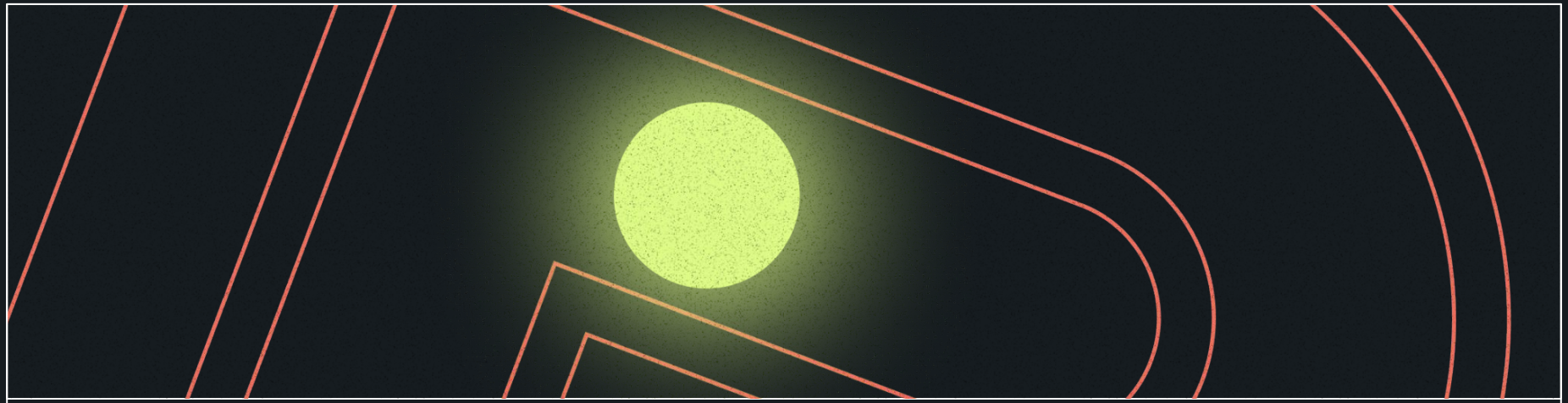
slido.com -> 3822198

Why Real life Schrodinger's cat ?

LABYRINTH LABS

Without good **Observability** your application is both **working** and **crashed** at the same time until ....

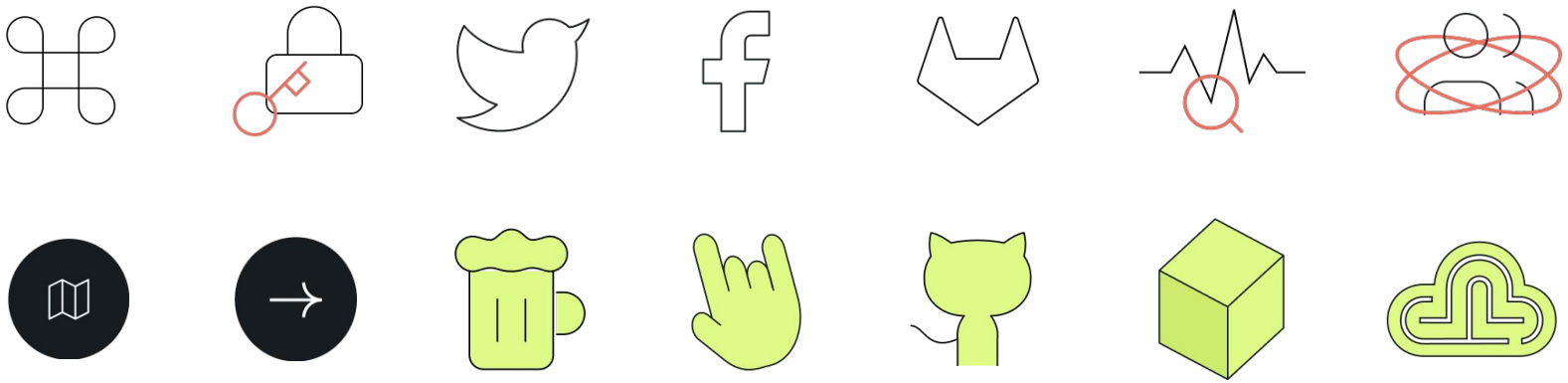
Your **customer** **calls** your boss.

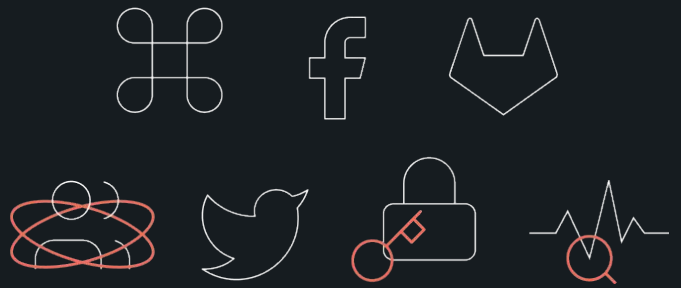


Contact us

Adam Hamšík
CEO

adam@lablabs.io
www.linkedin.com/in/adam-hamsik/






LABYRINTH
LABS


// life motto
if (sh*t() === true) {
sh*t.fix();
sh*tHappens();
}


#NOBULLSH*T


I
K8s


I'M ON 9