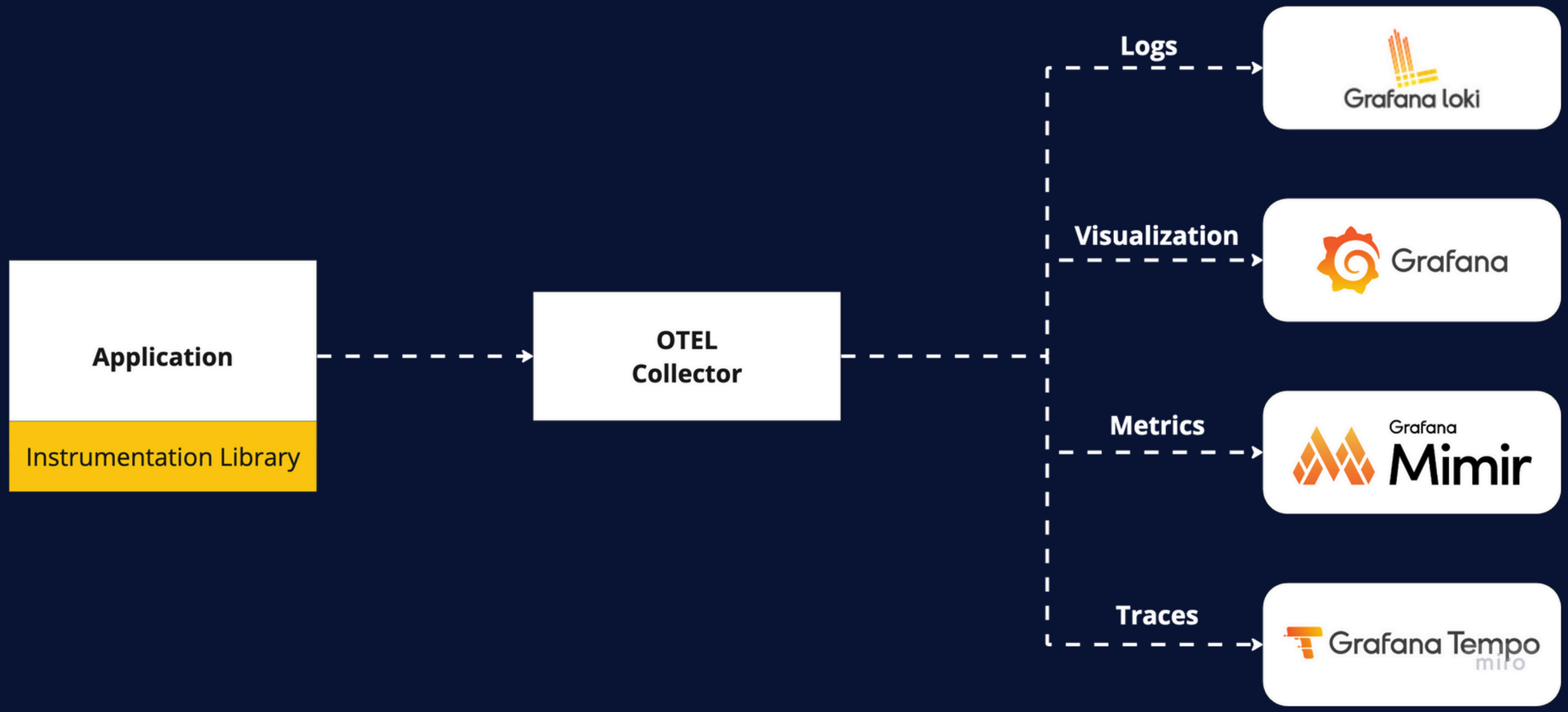
Zero-code application observability with Grafana Beyla

Peeranat Ounhanan (Ice)
Senior Associate Cloud Engineer
Arise by Infinitas

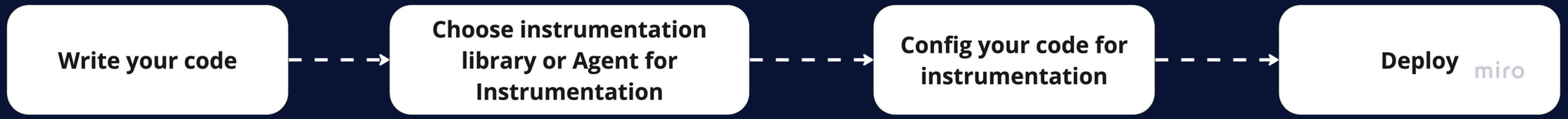
Peeranat Ounhanan (Ice)

- Senior Associate Cloud Engineer @ Arise by INFINITAS
- Cybersecurity, Bangkok University
- Microsoft Certified Trainer
- Former Gold Microsoft Learn Student Ambassador
- Former DevOps Engineer Intern @ Opsta
- Former DevSecOps Engineer Intern @ True Digital Group
- Former Full Stack Developer @ Buff Technology
- Scuba Diver & Cat Lover

Traditional Application Observability



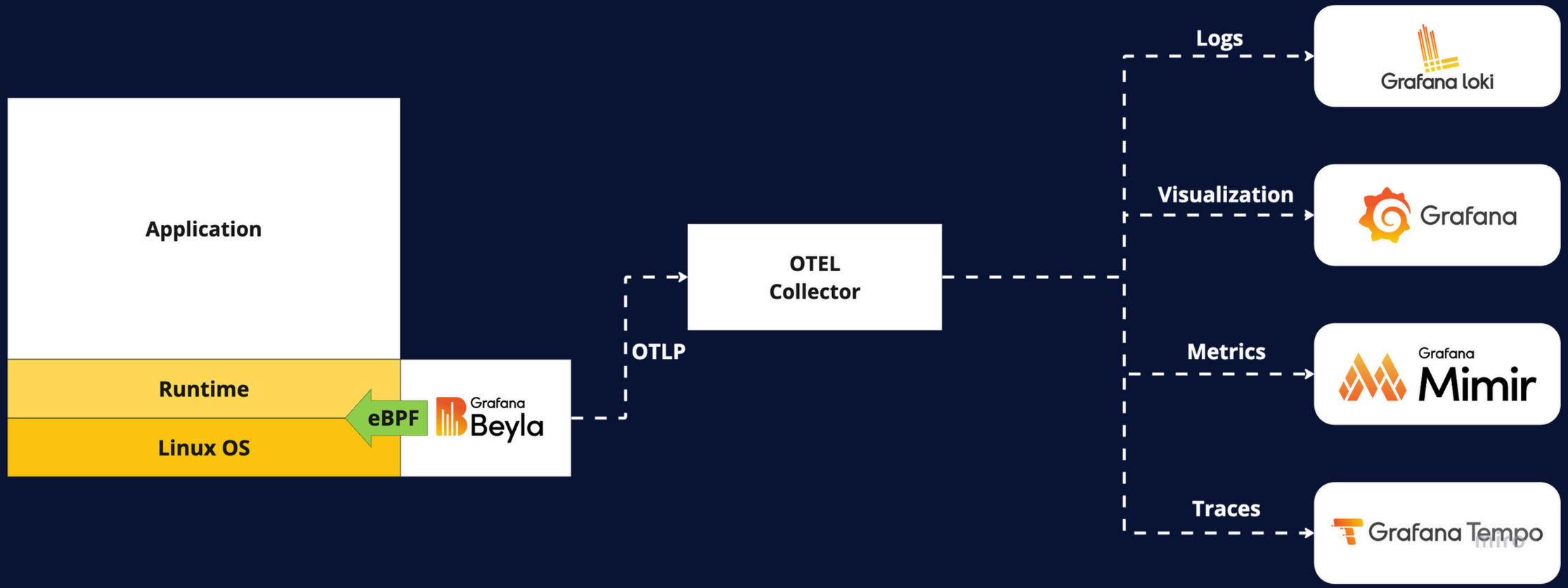
Manual Instrumentation



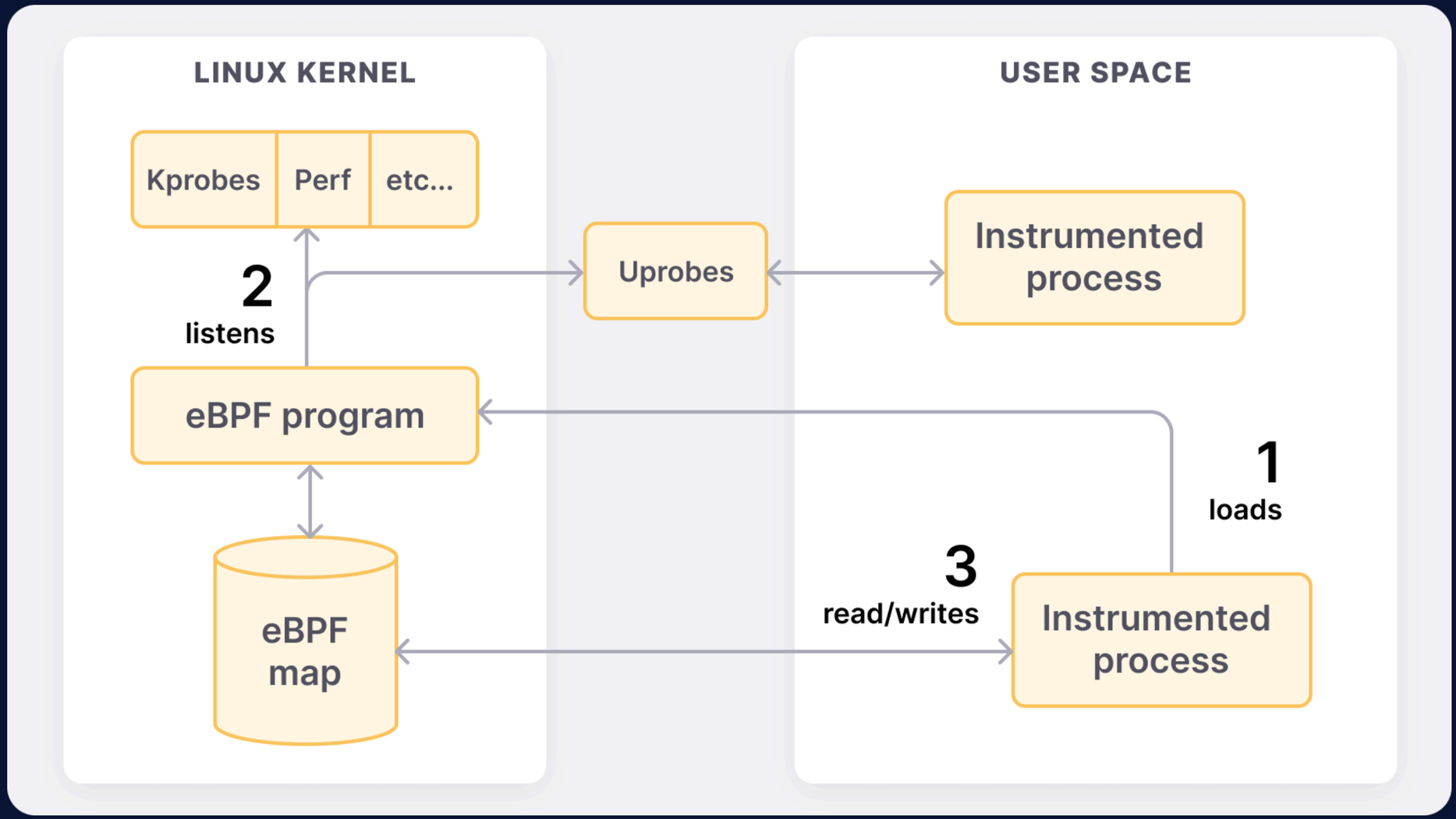
Grafana Beyla

Grafana Beyla is an open source eBPF-based auto-instrumentation tool that helps you easily get started with application observability for Go, C/C++, Rust, Python, Ruby, Java, NodeJS, .NET, and more.

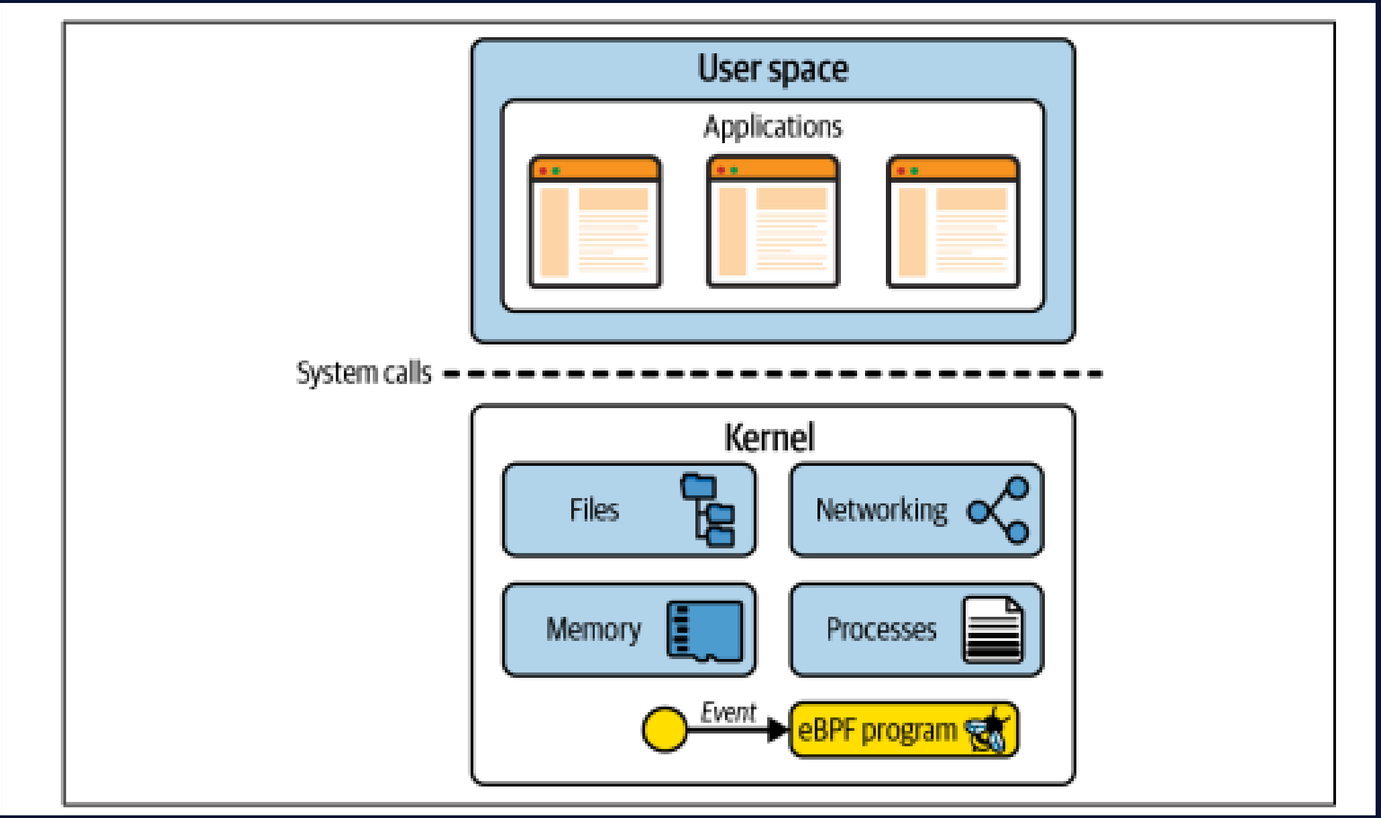
Auto-instrumentation Grafana Beyla and eBPF



eBPF overview

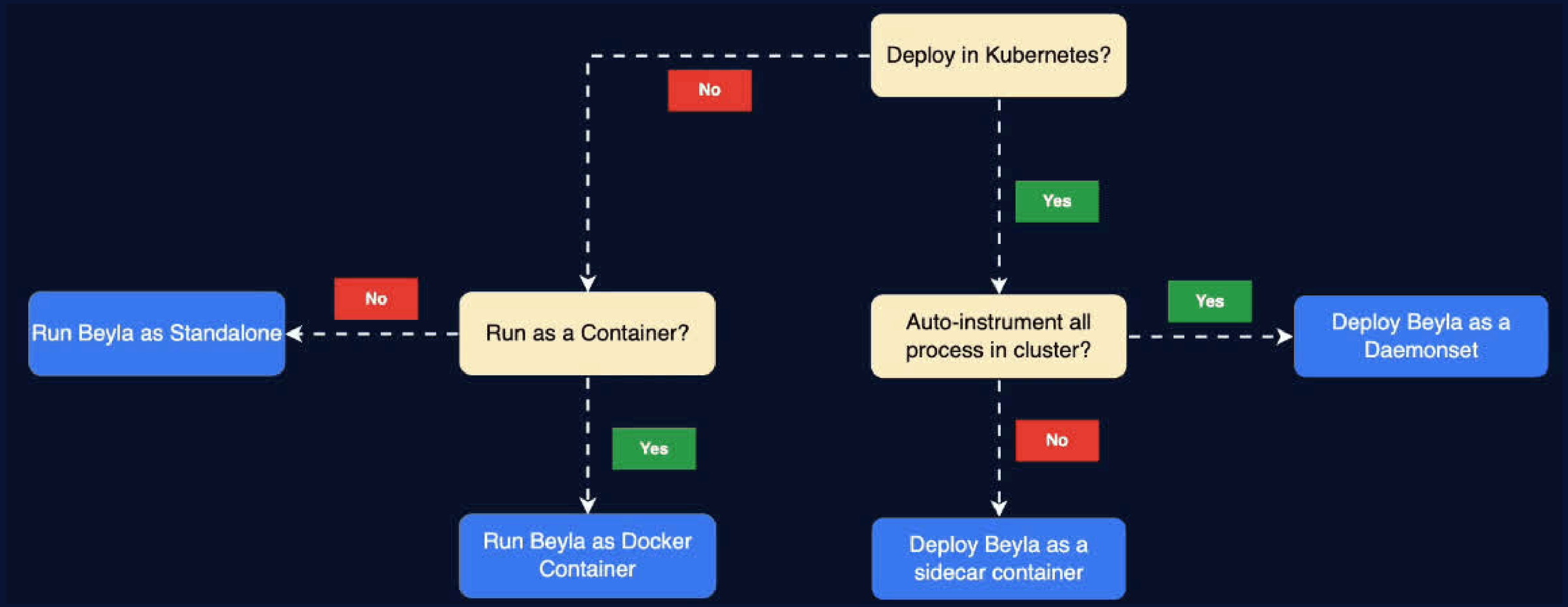


eBPF overview



https://isovalent.com/books/learning-ebpf/#form

Beyla deployment mode



Configuring Kubernetes metadata decoration

Create a ServiceAccount and bind a ClusterRole granting list and watch permissions for both Pods and ReplicaSets.

```yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: beyla
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: beyla
rules:
  - apiGroups: [ "apps" ]
    resources: [ "replicasets" ]
    verbs: [ "list", "watch" ]
  - apiGroups: [ "" ]
    resources: [ "pods", "services", "nodes" ]
    verbs: [ "list", "watch" ]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: beyla
subjects:
  - kind: ServiceAccount
    name: beyla
    namespace: default
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: beyla
```

Grafana Beyla process to instrument

- Port
- Namespace
- Service
- Deployment
- Pod
- and etc.

yaml Copy

```yaml
discovery:
  services:
    - k8s_namespace: frontend
      k8s_pod_labels:
        instrument: beyla
```

Beyla exported metrics

Family	Name (OTEL)	Name (Prometheus)	Type	Unit	Description
Application	`http.client.request.duration`	`http_client_request_duration_seconds`	Histogram	seconds	Duration of HTTP service calls from the client side
Application	`http.client.request.body.size`	`http_client_request_body_size_bytes`	Histogram	bytes	Size of the HTTP request body as sent by the client
Application	`http.server.request.duration`	`http_server_request_duration_seconds`	Histogram	seconds	Duration of HTTP service calls from the server side
Application	`http.server.request.body.size`	`http_server_request_body_size_bytes`	Histogram	bytes	Size of the HTTP request body as received at the server side
Application	`rpc.client.duration`	`rpc_client_duration_seconds`	Histogram	seconds	Duration of GRPC service calls from the client side
Application	`rpc.server.duration`	`rpc_server_duration_seconds`	Histogram	seconds	Duration of RPC service calls from the server side
Application	`sql.client.duration`	`sql_client_duration_seconds`	Histogram	seconds	Duration of SQL client operations (Experimental)
Application	`redis.client.duration`	`redis_client_duration_seconds`	Histogram	seconds	Duration of Redis client operations (Experimental)
Application	`messaging.publish.duration`	`messaging_publish_duration`	Histogram	seconds	Duration of Messaging (Kafka) publish operations (Experimental)
Application	`messaging.process.duration`	`messaging_process_duration`	Histogram	seconds	Duration of Messaging (Kafka) process operations (Experimental)
Application process	`process.cpu.time`	`process_cpu_time_seconds_total`	Counter	seconds	Total CPU seconds broken down by different states (system/user/wait)
Application process	`process.cpu.utilization`	`process_cpu_utilization_ratio`	Gauge	ratio	Difference in `process.cpu.time` since the last measurement, divided by the elapsed time and number of CPUs available to the process
Application process	`process.memory.usage`	`process_memory_usage_bytes`	UpDownCounter	bytes	The amount of physical memory in use
Application process	`process.memory.virtual`	`process_memory_virtual_bytes`	UpDownCounter	bytes	The amount of committed virtual memory
Application					

https://grafana.com/docs/beyla/latest/metrics/

Auto-instrumentation with Beyla vs Traditional

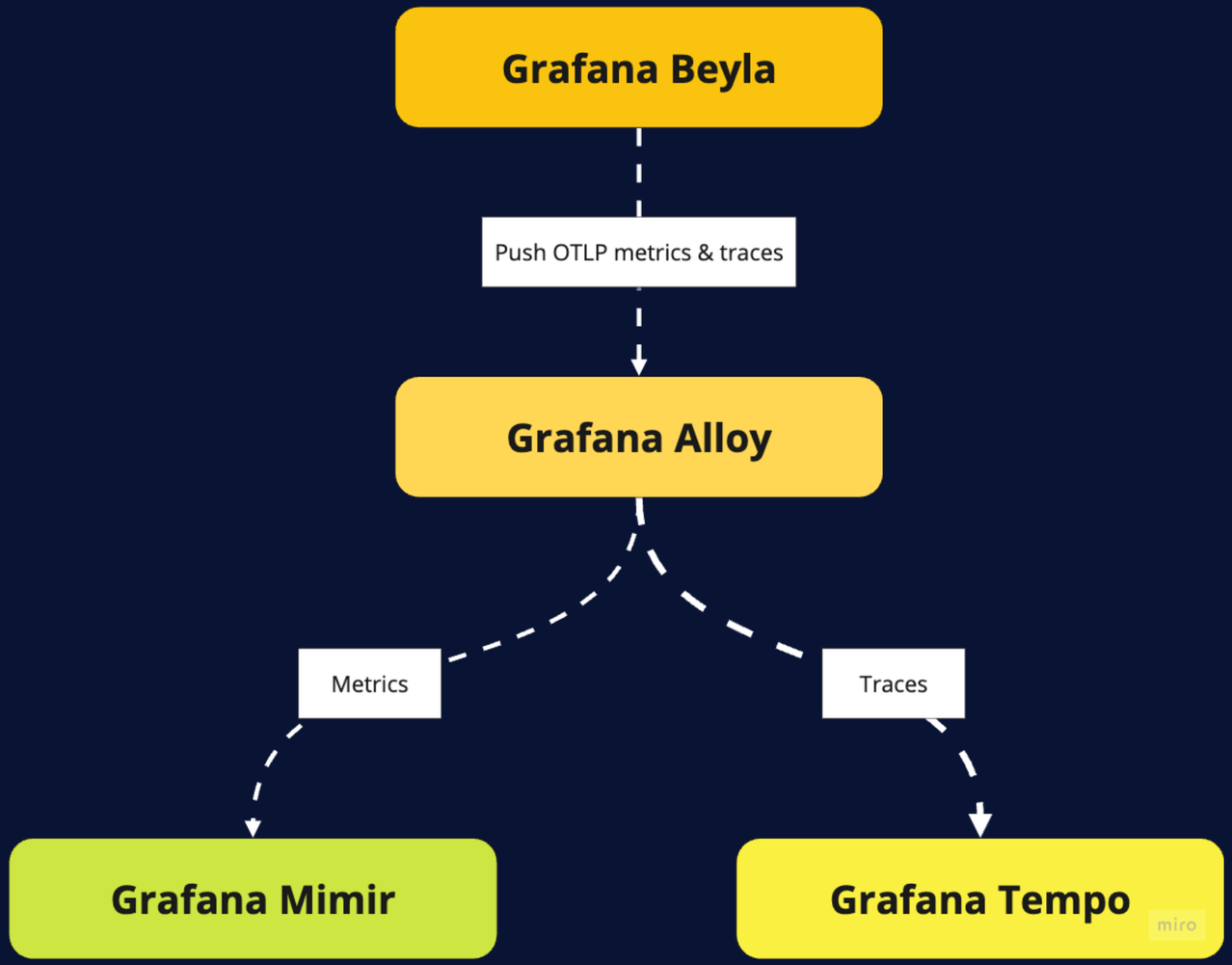
Pros

- No code change
- Get instant RED metrics and traces
- Supports a wide range of programming languages (HTTP/gRPC)

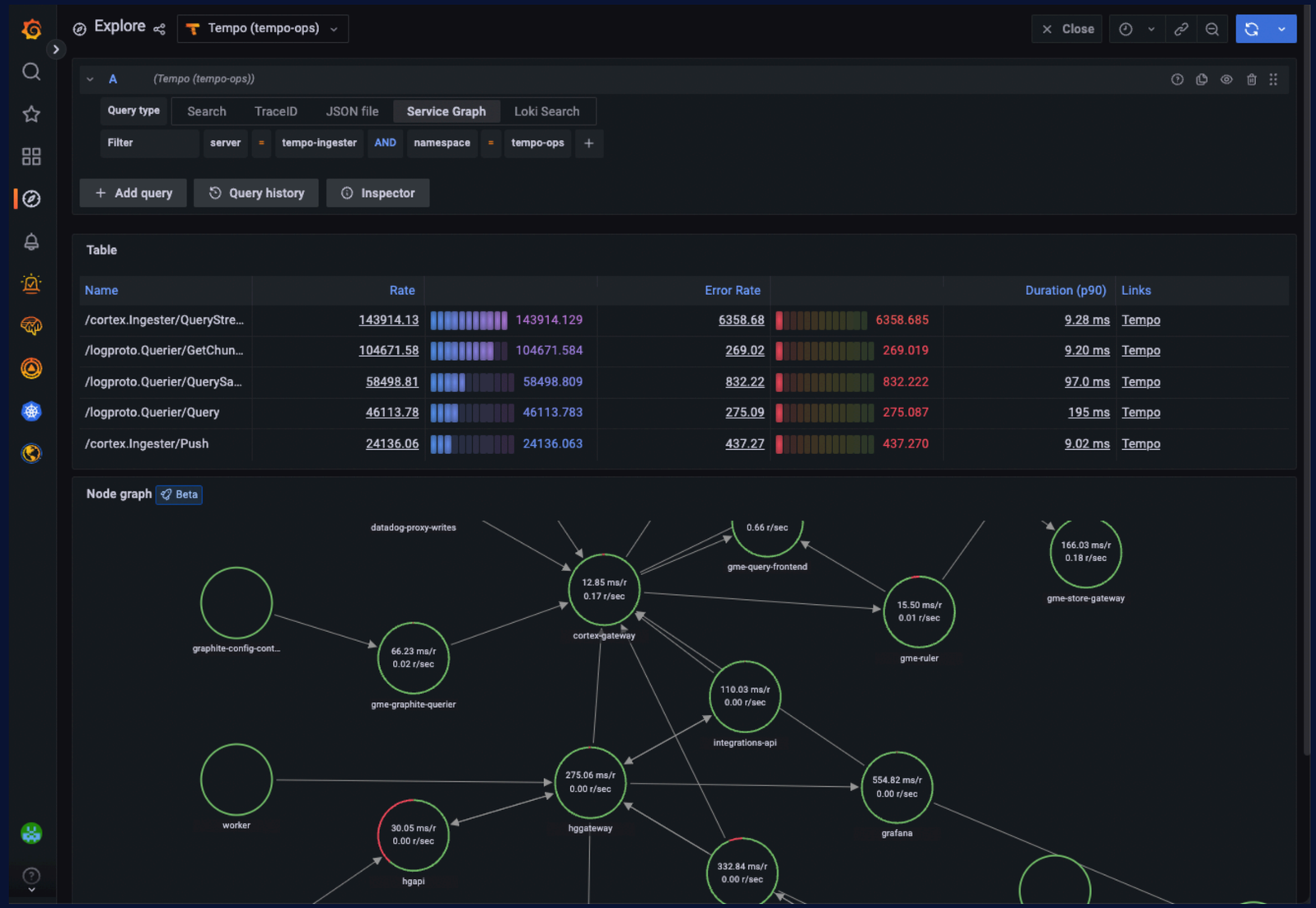
Cons

- Customization Limitations
- Complexity and Learning Curves

Export metrics and traces from Beyla using Grafana Alloy

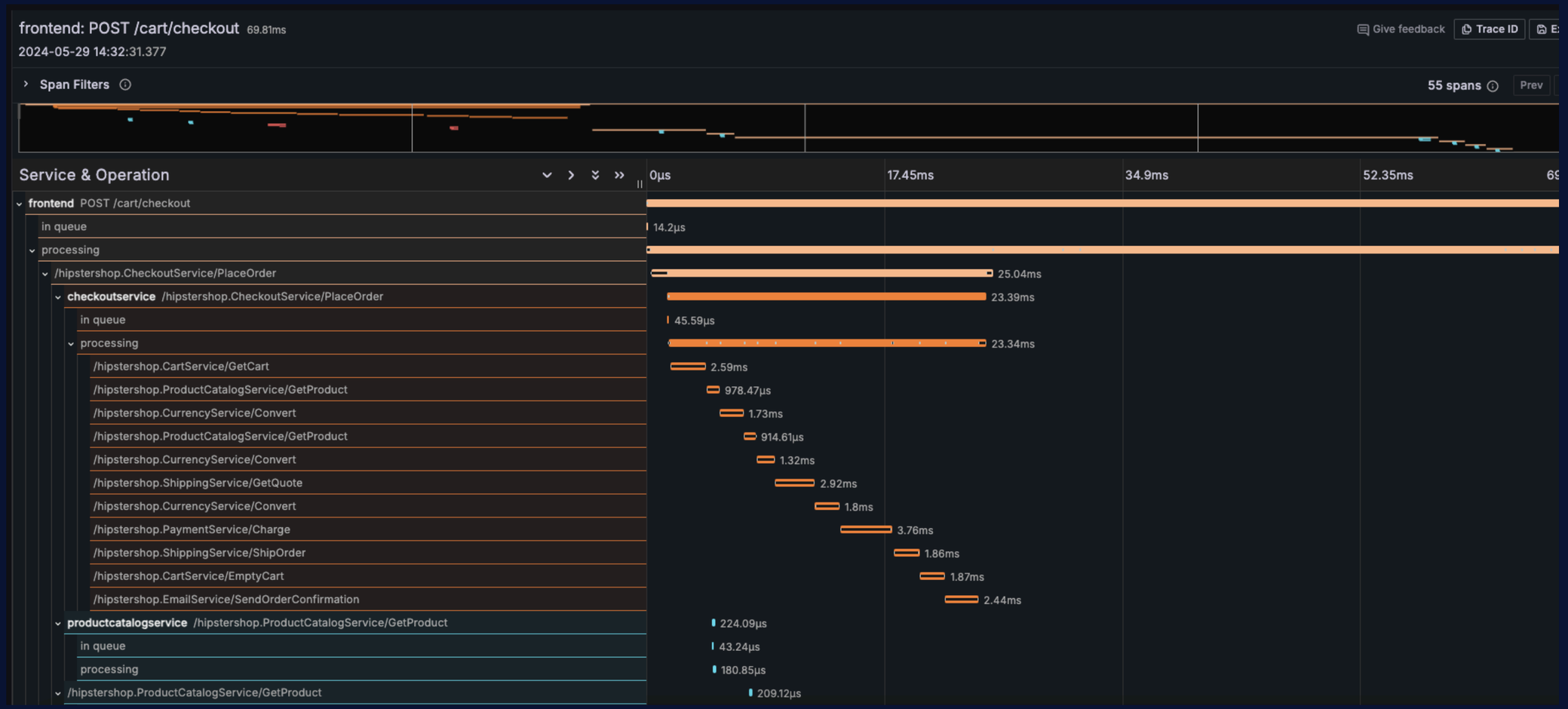


Service Graph and Span Metrics

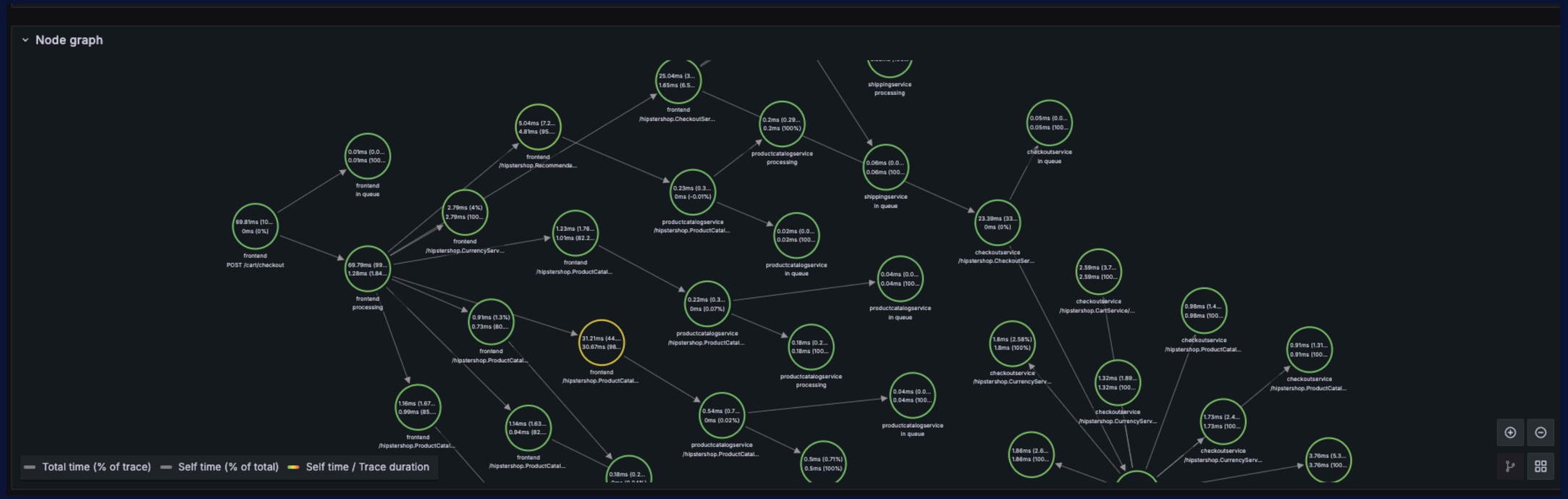


https://grafana.com/docs/tempo/latest/metrics-generator/

Distributed tracing structure



Service graph from trace



Beyla RED Metrics

HTTP and gRPC RED metrics visualization for Grafana Beyla

Demo

Zero-code application observability with Grafana Beyla

Grafana Beyla OSS | eBPF-based auto-instrumentation

What is Grafana Beyla?

Grafana Beyla คือ เครื่องมือ Open source จากทาง Grafana ใช้ในการทำ auto-instrumentation ด้วยการใช้เทคโนโลยี eBPF มาช่วย เพื่อให้การทำ Application observability เป็นเรื่องที่ง่ายมากขึ้น และ Beyla สามารถส่ง Metrics, Tracing และ Logs รวมถึง RED Metrics ออกไปยัง Tools ต่างๆ ได้ เช่น LGTM Stack

Grafana Beyla

https://bit.ly/grafana-beyla

Thank You