Observability on Kubernetes

Logging • Metrics • Prometheus/Alerting • Grafana • Tracing • Health Probes

Audience: Advanced / Intermediate (beginner-accessible)

Focus: Theory-first, platform patterns, and best practices

Agenda

- Logging in Kubernetes (stdout/stderr, sidecars, logging agents)
- Monitoring Metrics (cAdvisor, kubelet)
- Prometheus & Alertmanager
- Grafana Dashboards
- Distributed Tracing (Jaeger, OpenTelemetry)
- Health Probes (liveness, readiness, startup)

1) Logging in Kubernetes

Principles & Data Sources

- Write to stdout/stderr from containers — let the platform collect.
- Container log files on nodes (e.g., /var/log/containers, /var/log/pods), rotation handled by container runtime/logrotate.
- Structured logs (JSON) preferred; include request IDs/trace IDs where possible.
- Correlation keys: trace_id, span_id, k8s.pod.name, k8s.container.name, k8s.namespace.name.

Collection Patterns

Pattern	How it works	Pros	Cons / Notes
Node logging agent	DaemonSet (e.g., Fluent Bit/Fluentd/Vector) tailing container files	Centralized, uniform	Node perms; daemon resource use
Sidecar logging	Sidecar ships app logs (file tail/transform)	App-specific transforms	More pods/overhead; coupling
Stdout to collector	Direct stdout/stderr scraping by runtime/agent	Simple, universal	Less control over per-app transforms
App to HTTP/syslog	App pushes to gateway/forwarder	Back-pressure control	Network dependency from app


Containers stdout/stderr
Node Log Files
DaemonSet Collector\nFluent Bit/Vector
Central Log Store\nElastic/Loki/Cloud Logs
Search/Analyze/Alerts

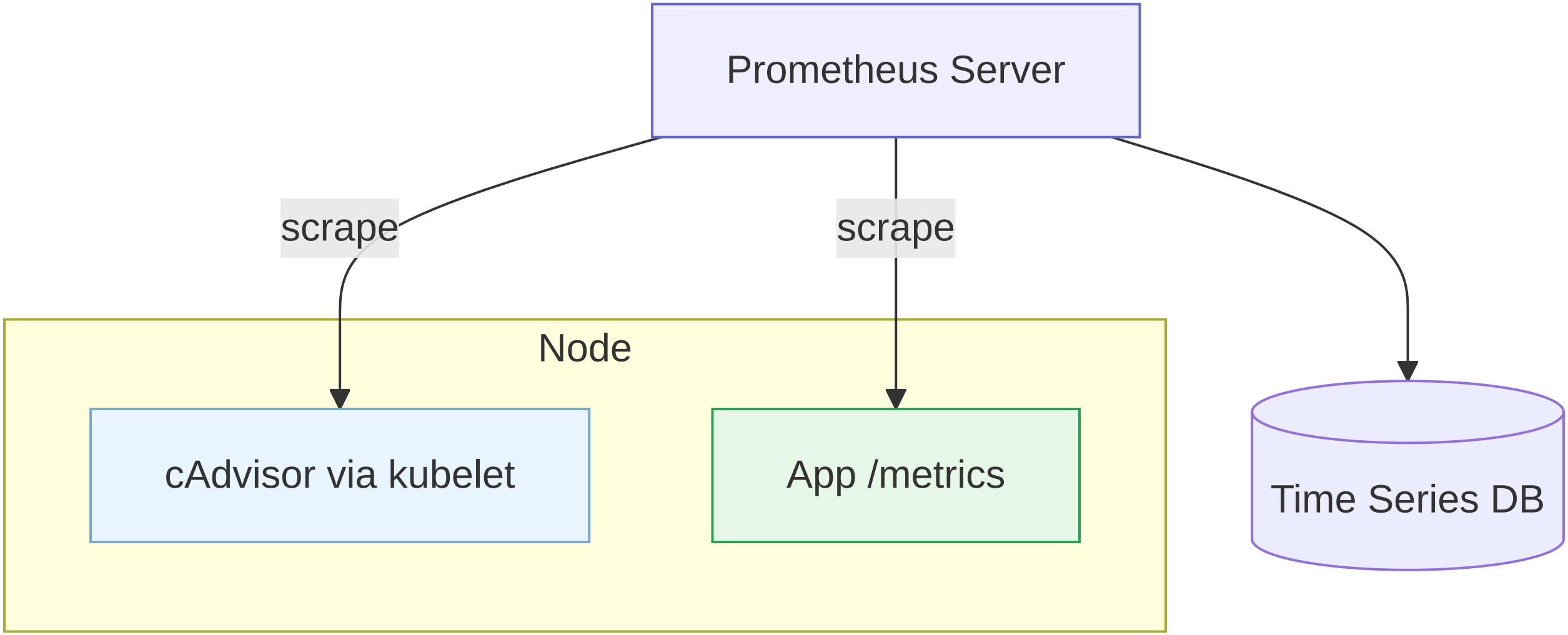
Best practices:

- Prefer stateless apps with structured logs.
- Drop PII or tokenize; control retention by data class.
- Enrich with K8s metadata at the collector (labels, pod, node).

2) Monitoring Metrics

Sources & Taxonomy

- **Container & Pod metrics:** cAdvisor (via kubelet) — CPU, memory, filesystem, network.
- **Node metrics:** kubelet summary API, node exporter (or cAdvisor node-level).
- **Control plane metrics:** API server, scheduler, controller-manager (Prometheus format).
- **App/business metrics:** Expose /metrics (Prometheus exposition) from services.


Prometheus Server
scrape
scrape
Node
cAdvisor via kubelet
App /metrics
Time Series DB

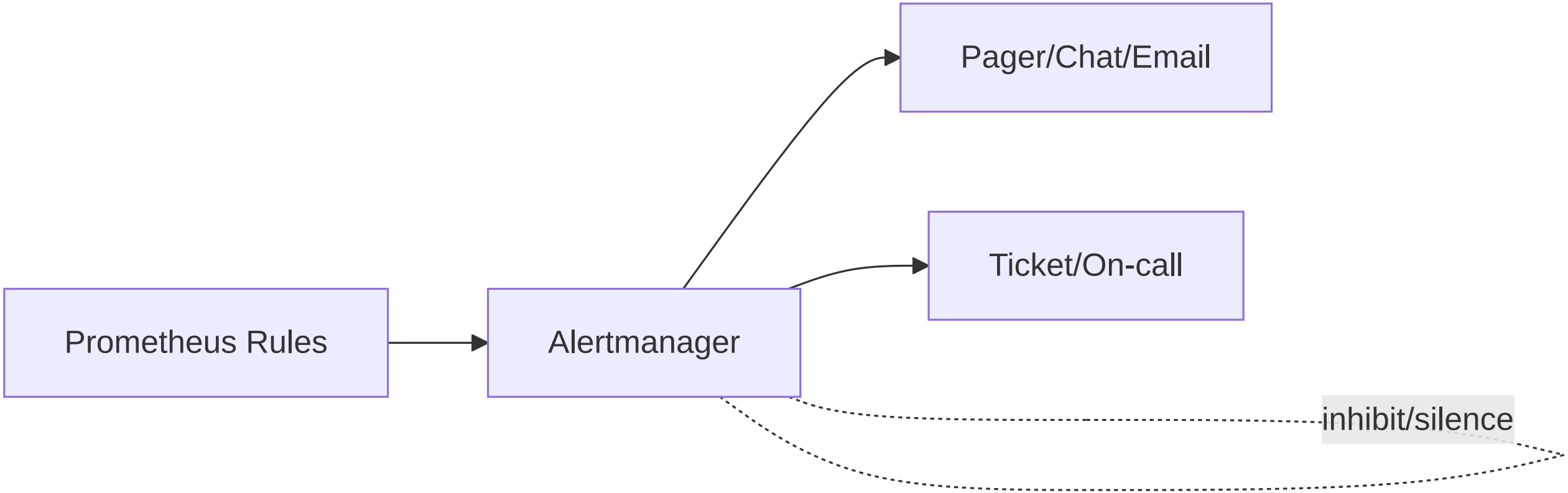
Cardinality control: Limit label explosion (e.g., user IDs, paths).

Resource overhead: Scrape intervals and retention impact cost.

3) Prometheus & Alertmanager

Prometheus (Pull-Based Collection)

- **Discovery:** Kubernetes SD finds targets by Service/Pod/Endpoints/Annotations.
- **Relabeling:** Transform/keep/drop targets and labels to control cardinality.
- **Recording rules:** Precompute expensive queries into new time series.
- **High availability:** Run multiple Prometheus instances; use Thanos/Cortex/Mimir for long-term storage/global view.


Pager/Chat/Email
Ticket/On-call
Prometheus Rules
Alertmanager
inhibit/silence

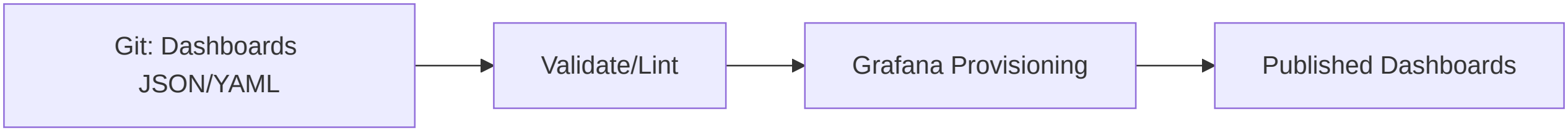
Alerting patterns:

- SLO-based alerts (burn rate) preferred over raw saturation spikes.
- Inhibition/silences to avoid alert storms during maintenance.
- Routing by team/namespace/service labels.

4) Grafana Dashboards

Visualization & Governance

- Dashboards from queries (PromQL/Loki/Tempo/OTel).
- Folders & permissions by team or domain; version dashboards as code.
- Templates/variables for namespace, service, cluster selection.
- Golden signals: Latency, traffic, errors, saturation (per service).

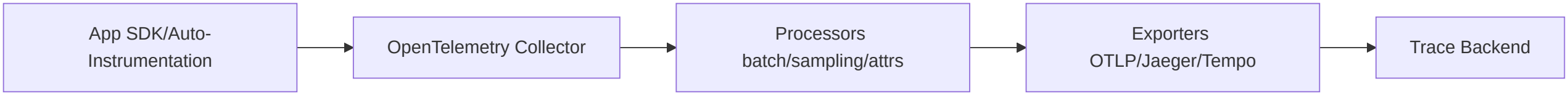

Git: Dashboards
JSON/YAML
Validate/Lint
Grafana Provisioning
Published Dashboards

Best practices: Reuse panels, define service dashboard templates, integrate annotations from deploys/incidents.

5) Distributed Tracing

Concepts & Tooling

- **Purpose:** End-to-end latency breakdown across services; trace (request) contains spans (operations).
- **Context propagation:** Trace headers (W3C Trace Context traceparent, tracestate).
- **Instrumentation:** Auto-instrumentation or SDKs; emit spans + metrics + logs with shared IDs.
- **Backends:** Jaeger, Tempo, Zipkin; OpenTelemetry for vendor-neutral SDK/collector.


App SDK/Auto-Instrumentation
OpenTelemetry Collector
Processors
batch/sampling/attrs
Exporters
OTLP/Jaeger/Tempo
Trace Backend

Sampling strategies: Head-based (probabilistic), tail-based (error/latency-focused).

Correlate: Include trace_id in logs and metrics for deep troubleshooting.

6) Health Probes

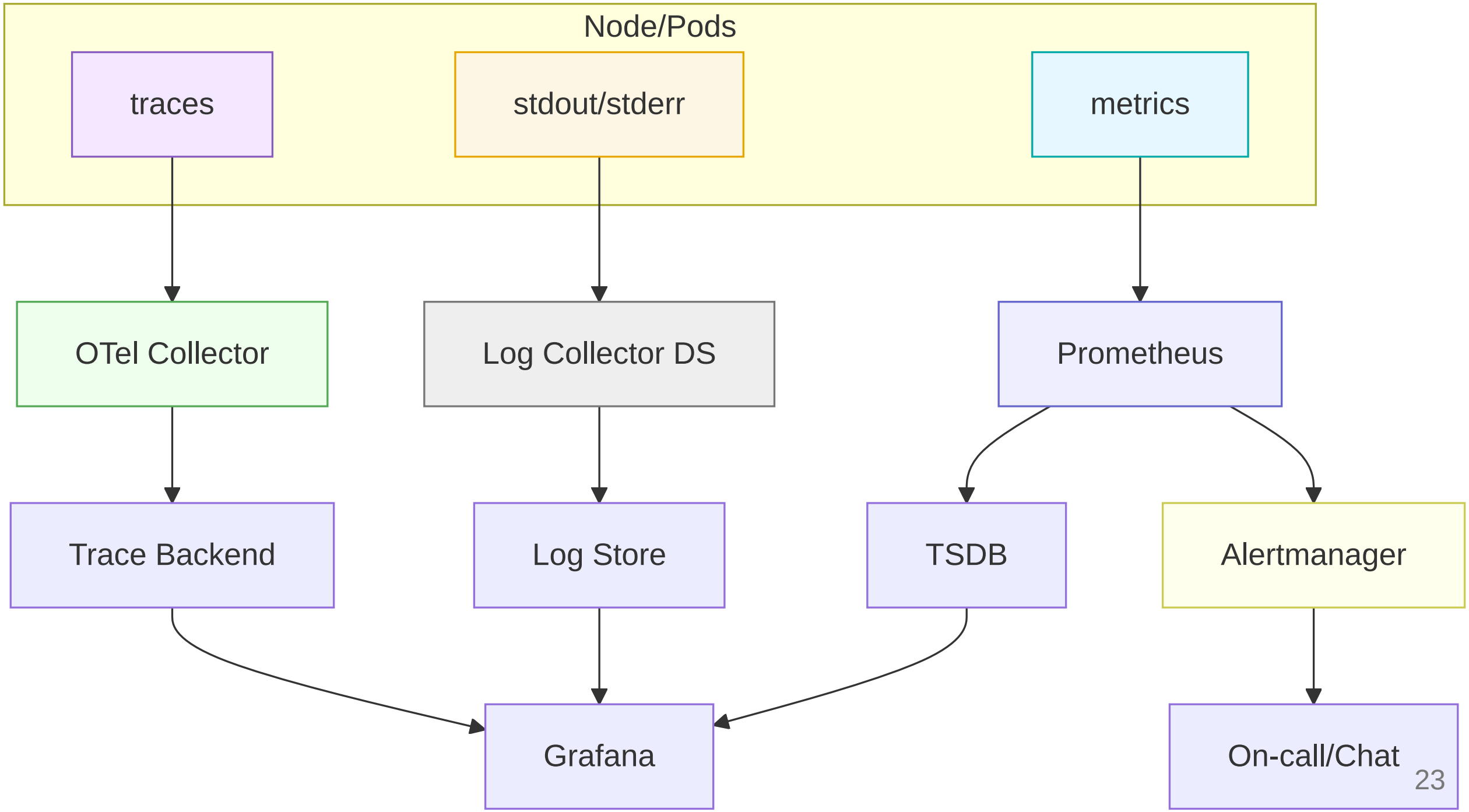
Liveness, Readiness, Startup

Probe	Typical check	Failure impact	Guidance
Liveness	Minimal internal check (e.g., heartbeat)	Restart container	Avoid heavy dependencies; keep fast & local
Readiness	App dependency readiness (DB, cache)	Remove from Service	Reflect user-facing availability; retry-friendly
Startup	One-time boot completion	Delay liveness	Use for long cold-starts; disable liveness until done

- **Probe types:** HTTP, TCP, gRPC, or exec. Use timeouts and thresholds carefully (avoid flapping).
- **Common pitfall:** Using expensive health endpoints causes cascading failures.

Putting It All Together

Unified Observability Reference


Node/Pods
traces
stdout/stderr
metrics
OTel Collector
Log Collector DS
Prometheus
Trace Backend
Log Store
TSDB
Alertmanager
Grafana
On-call/Chat

Outcome:

- Logs: context-rich, searchable, retained appropriately
- Metrics: SLO-driven alerts, low cardinality
- Traces: end-to-end visibility, root-cause speed
- Probes: safe rollouts and resilient routing

Best Practices & Anti-Patterns

Do:

- Standardize labels: service, namespace, version, instance.
- Treat dashboards and alerts as code; review in PRs.
- Correlate logs/metrics/traces with shared IDs.
- Budget SLOs and alert on burn rates.

Avoid:

- Logging secrets/PII; unbounded log levels.
- High-cardinality labels (user IDs, raw URLs).
- Probes that depend on external services.
- Single-cluster Prometheus without HA/remote write for prod.

References & Further Reading

- **Kubernetes:** Logging architecture, kubelet summary API, probes
- **Prometheus:** Service discovery, alerting, recording rules; Alertmanager routing
- **Grafana:** Provisioning, Loki/Tempo datasources
- **OpenTelemetry:** Spec, Collector, Trace Context
- **Jaeger/Tempo:** Tracing backends & sampling strategies