**DevOps & Security Dashboards**

**1. Executive summary**

Build two classes of dashboards: **operational dashboards** (real-time, incident-focused) and **analytical dashboards** (trending, capacity, release retrospectives). Prioritize: service health (golden signals), CI/CD/release metrics (DORA), infrastructure & Kubernetes health, APM/traces, and an SOC summary for security. Instrument applications with OpenTelemetry (for traces and metrics) and push metrics to Prometheus or a managed metric store; centralize logs with Fluentd/Beats into an ELK/SIEM; use vulnerability scanners and cloud provider telemetry for security dashboards. Create runbooks and link them from dashboards. Keep dashboards role-focused (on-call, engineering, management, SOC).

**2. Design principles and roles**

* **Purpose-first**: every dashboard must answer specific operational questions ("Is service X healthy?" or "Are deployments increasing failures?").
* **Audience-aware**: create views for SREs/on-call, Dev teams, platform engineers, managers, and security analysts.
* **SLO-driven**: design around SLIs/SLOs and error budgets rather than raw metrics alone.
* **Actionable**: include context (recent deploys, correlated logs/traces, runbook links) so a viewer can act.
* **Low-noise**: surface top indicators and high-priority alerts; avoid dumping all metrics on one page.
* **Correlation-first**: enable quick pivoting between metrics, logs, and traces.
* **Cost-aware**: consider cardinality and retention tradeoffs when instrumenting.

**3. High-level dashboard categories**

**DevOps / Observability**

* Service health (Golden Signals)
* CI/CD & Release Health (DORA + pipeline metrics)
* Infrastructure / K8s cluster health
* Application performance (APM)
* Logs & traces correlation
* Incident & On-call overview
* Cost / FinOps
* Business/Product KPIs

**Security / SecOps**

* SOC Overview (MTTD/MTTR, open incidents)
* Threat & Detection (alerts by source, rule effectiveness)
* Vulnerability Management (open critical/high vulns, patching trends)
* Identity & Access (failed logins, privilege changes, MFA coverage)
* Endpoint & EDR (device posture, EDR alerts)
* Network security (blocked flows, suspicious connections)
* Compliance & Audit (control coverage, audit findings)
* Supply chain & SBOM (unpatched libs, critical components)

**4. Detailed dashboard catalog**

Each dashboard section below has: **Purpose**, **Key metrics/parameters**, **Suggested visualizations**, **Where & how to collect** (collectors, exporters, APIs), and **Notes**.

**A. Service health — The Four Golden Signals**

**Purpose:** Real-time view of service availability and user experience.

**Key metrics:**

* **Latency**: p50/p90/p95/p99 for user-facing requests; request duration histogram.
* **Traffic**: requests per second (RPS), active users, sessions.
* **Errors**: error rate (4xx/5xx), total error counts, top error types and endpoints.
* **Saturation**: CPU, memory, disk I/O, queue lengths, connection pools, database thread pool usage.

**Suggested visualizations:**

* Timeseries graphs for p95/p99 latency and RPS.
* Error-rate heatmap and top-N error waterfall.
* Resource saturation stacked area charts and node-level table.
* Single-value status tile with SLO traffic-light (green/amber/red).

**Where & how to collect:**

* Instrument app with OpenTelemetry (metrics & traces). Add HTTP server/client metrics; expose Prometheus metrics or use OTLP -> metric backend.
* Use APM agents (Datadog, New Relic) if available.
* Infrastructure metrics: node-exporter, cadvisor, kube-state-metrics, cloud provider metrics (CloudWatch, Stackdriver, Azure Monitor).
* Ingress/load-balancer metrics: NGINX/Envoy metrics, ALB/NLB metrics.

**Notes:**

* Compute percentiles from histograms (avoid computing percentiles from raw samples).
* Correlate spikes with recent deploys (include last deploy timestamp and commit id on dashboard).

**B. CI/CD & Release Health (DORA + pipeline)**

**Purpose:** Track how fast and stable your delivery process is.

**Key metrics (DORA + pipeline):**

* **Deployment frequency** (deploys/day/week)
* **Lead time for changes** (commit -> prod time)
* **Change failure rate** (percent of deployments causing failures/rollbacks)
* **Mean time to restore (MTTR)** after a change-related incident
* **Pipeline pass rate** and **average build/test durations**
* **Flaky test rate** and **rollback rate**
* **PR metrics**: PR size, time to review, merge time

**Suggested visualizations:**

* Trend charts for deployment frequency and lead time.
* Bar/heatmap for pipeline stage durations and failure hotspots.
* Pareto of failing tests and flaky tests.

**Where & how to collect:**

* CI/CD system APIs (Jenkins, GitHub Actions, GitLab, Azure DevOps) — many expose webhooks or metrics plugins (e.g., Jenkins Prometheus plugin).
* SCM APIs (GitHub/GitLab) for PR/commit events.
* Parse pipeline logs and use a pipeline exporter or push metrics into Prometheus/metric store.

**Notes:**

* Use the same change identifiers across VCS/CI and production metadata (commit SHA, deployment id) to correlate.
* Track per-team DORA metrics to identify process bottlenecks.

**C. Infrastructure & Kubernetes**

**Purpose:** Cluster and host-level health for platform stability.

**Key metrics:**

* Node status (Ready/NotReady count)
* Pod restarts, OOM kills, crashloop count
* CPU / Memory / Disk usage % (per node & per namespace)
* Pod density, unschedulable pods
* Persistent volume health & IO
* Cluster autoscaler events and failed pod scheduling reasons

**Where & how to collect:**

* kube-state-metrics, node-exporter, cadvisor, kubelet metrics
* Cloud provider metrics (GCP/AWS/Azure) for underlying VMs
* K8s events (aggregated into logs or metrics via exporters)

**Suggested visualizations:**

* Node map with heat indicators (CPU/memory).
* Table of top restarting pods + last 5 errors.
* Schedulability timeline and autoscaler activity chart.

**Notes:**

* Tag metrics by cluster, namespace, app, and team. Keep label cardinality under control.

**D. Application Performance Monitoring (APM & Traces)**

**Purpose:** Drill into transaction-level latency across services and DB/cache/external calls.

**Key metrics:**

* Trace latency distributions for important transactions
* Span breakdown (time spent in DB, network, external APIs)
* Slowest endpoints and slowest downstream dependencies
* Error traces and slow-trace sampling rate
* Database query durations (p95) and QPS
* Cache hit ratio and TTL misses

**Where & how to collect:**

* Instrument with OpenTelemetry (traces + spans); export OTLP to trace backend (Tempo, Jaeger, commercial APMs).
* DB metrics via exporters (postgres\_exporter, mysql exporter) and query logging.

**Suggested visualizations:**

* Top-N slow traces list with waterfall span view.
* Service map (call graph) with average latency per edge.

**Notes:**

* Use trace-based sampling policies: sample errors and a fraction of successful traces, or use tail-based sampling if supported.

**E. Logs & Traces Correlation**

**Purpose:** Allow quick pivoting from alerts to logs to traces for RCA.

**Key metrics:**

* Log volume and error log rate
* Top error messages and frequency
* Trace-to-log correlation rate (how many traces have correlated logs)

**Where & how to collect:**

* Structured logs (JSON) emitted by app, containing trace\_id, span\_id, request\_id, user\_id where applicable.
* Collectors: Fluentd, Fluent Bit, Filebeat -> ELK or SIEM, or directly ingest via OpenTelemetry Collector.

**Suggested visualizations:**

* Table of recent error logs with linked trace IDs.
* Wordcloud/Top-10 error messages (for triage).

**Notes:**

* Ensure logs include the same resource attributes/labels as metrics (service, env, pod, region) to enable joins.

**F. Incident & On-call Overview**

**Purpose:** Support SREs / on-call engineers to see active incidents, priority, owner, and suggested runbooks.

**Key metrics/tiles:**

* Active incidents by severity
* MTTR / MTTD for current incidents
* On-call rotation, escalation matrix
* Top open alerts by age

**Where & how to collect:**

* PagerDuty / OpsGenie APIs, incident management tools, alert manager.
* Alertmanager metrics for firing alerts and silences (Prometheus Alertmanager).

**Suggested visualizations:**

* Incident timeline and backlog table with links to runbooks and commits.

**G. Cost / FinOps Dashboard**

**Purpose:** Monitor cloud spend and cost anomalies tied to services.

**Key metrics:**

* Cost by service / tag / environment (daily, monthly)
* Cost per customer / per feature (if applicable)
* Unused/underutilized resources (idle VMs, unattached disks)
* Forecast vs actual spend and anomaly detection

**Where & how to collect:**

* Cloud billing APIs (AWS Cost Explorer, GCP Billing, Azure Cost Management)
* Tagging data from infrastructure-as-code tools and Kubernetes labels

**Suggested visualizations:**

* Heatmap of spend by service and region, top-10 spenders table, anomaly alerts.

**H. Business & Product KPIs**

**Purpose:** Show business-impacting metrics to engineering and product managers.

**Key metrics:**

* Conversion funnel metrics (sessions->signup->purchase)
* Checkout success rate, drop-off points
* Customer-facing SLA metrics (uptime, response time for paid customers)

**Where & how to collect:**

* Instrumentation in app events (analytics), event streams (Kafka), or product analytics platforms.

**Suggested visualizations:**

* Funnel chart, cohort retention charts, single-value KPIs with trends.

**Security dashboards (detailed)**

**1. SOC Overview / Security Posture**

**Purpose:** Executive & SOC view of security health.

**Key metrics:**

* Open incidents by severity
* Mean Time To Detect (MTTD), Mean Time To Respond/Remediate (MTTR)
* Alerts per minute / alerts by source
* False positive rate of rules
* Coverage: % hosts with EDR, % assets reporting

**Where & how to collect:**

* SIEM (Splunk, Elastic Security, Azure Sentinel) ingesting logs from EDR, network sensors, cloud logs.

**Notes:**

* Use NIST CSF or internal risk model to map metrics to risk scores and remediation priority.

**2. Threat & Detection**

**Purpose:** Detection engineering and SOC triage.

**Key metrics:**

* Alerts by detection rule and source
* Time to investigate / escalate
* Alerts correlated with MITRE ATT&CK tactics/techniques

**Where & how to collect:**

* SIEM + threat intel feeds, EDR telemetry, firewall logs, IDS/IPS logs, cloud guardrails.

**Notes:**

* Map detections to MITRE ATT&CK for coverage visualization.

**3. Vulnerability Management**

**Purpose:** Measure and prioritize vulnerability remediation.

**Key metrics:**

* Number of open vulnerabilities by severity (critical, high, medium)
* Average time to remediate by severity
* Patch coverage % (hosts patched vs known CVEs)
* New vs remediated vulnerabilities per week

**Where & how to collect:**

* Vulnerability scanners (Qualys, Nessus), container image scanners (Trivy), SCA tools and SBOM outputs (CycloneDX).

**Notes:**

* Prioritize remediation by exploitability and business-critical assets.

**4. Identity & Access**

**Purpose:** Monitor misuse and risky auth behavior.

**Key metrics:**

* Failed login attempts and rate (by account, IP, region)
* MFA coverage % across users
* Privileged account activity and privilege escalations
* Stale/unused service accounts

**Where & how to collect:**

* Authentication logs (IdP: Okta, Azure AD), cloud IAM logs, audit logs.

**5. Endpoint & EDR**

**Purpose:** Endpoint posture and detections.

**Key metrics:**

* Devices with critical alerts
* EDR isolation events, malware detections
* Device health (connectivity, agent version)

**Where & how to collect:**

* EDR solutions (CrowdStrike, SentinelOne), MDM telemetry.

**6. Network & Perimeter**

**Purpose:** Monitor suspicious traffic and blocked connections.

**Key metrics:**

* Blocked connections, top source/destination IPs
* Unusual data exfiltration patterns (volume spikes)
* Firewall rule hits and changes

**Where & how to collect:**

* Firewall logs, VPC flow logs, IDS/IPS telemetry.

**7. Compliance & Audit**

**Purpose:** Track control coverage & audit readiness.

**Key metrics:**

* Controls passed/failed, number of open audit findings
* Time to remediate audit findings
* Evidence coverage for cloud & app configurations

**Where & how to collect:**

* CSPM tools, audit logs, configuration management databases (CMDB).

**8. Supply Chain & SBOM**

**Purpose:** Track third-party and open-source component risk.

**Key metrics:**

* Number of components with known CVEs
* Critical vulnerabilities in third-party libs
* Time to update dependencies

**Where & how to collect:**

* SCA tools, image-scanning outputs, SBOM generators.

**5. Implementation patterns — instrumentation & collectors**

**Metrics**

* Use **OpenTelemetry** to standardize instrumentation across languages, and export to Prometheus/OTLP-compatible backends.
* Use metrics exporters: **node-exporter**, **kube-state-metrics**, **cadvisor**, **database exporters**.

**Logs**

* Structured JSON logs, include trace\_id/request\_id. Use Fluentd/Fluent Bit/Filebeat to forward to ELK/SIEM.

**Traces**

* Use OpenTelemetry traces; store in Tempo, Jaeger, or commercial APM.

**Security telemetry**

* Ingest logs to SIEM; use connectors for EDR, firewall, cloudtrail/Cloud Audit Logs, identity providers, vulnerability scanners.

**Correlating identifiers**

* Standardize resource attributes: service, env, team, region, deployment\_id, commit.
* Include these attributes across metrics, logs, and traces to enable joins.

**6. Visualization & UX patterns**

* **Top-of-page status** single-value tiles for SLO health and active incidents.
* **Time-series** charts with clear baselines and anomaly shading.
* **Top-N tables** for servers, services, tests, or vulnerabilities causing most problems.
* **Service map** for dependency visualization.
* **Drilldowns** linking from a chart row to logs, traces, PRs, or runbooks.
* **Runbook link** on incident tiles and suggested first steps.

**7. Alerting, SLOs, and error budgets**

* Define **SLI** (what you measure) and **SLO** (target). Example: user request p95 latency < 400ms, availability > 99.9%.
* Build dashboards that show SLOs and remaining error budget.
* Alerts should be tied to action: severity based on SLO burn rate, not raw metric thresholds.
* For security alerts, prioritize by **risk score** (asset criticality × severity × exploitability).

**8. Ownership, RBAC & operationalization**

* Assign dashboard owners (team + backup).
* Configure role-based views (read-only for managers, detailed views for on-call).
* Automate dashboard creation where possible (Grafana dashboards as code via JSON/Helm/Terraform).
* Add regular reviews: quarterly dashboard reviews and data-quality checks.

**9. Example queries & visualizations**

**PromQL — p95 HTTP latency (histogram)**

histogram\_quantile(0.95, sum(rate(http\_request\_duration\_seconds\_bucket[5m])) by (le, route))

**PromQL — error rate (percent)**

(sum(rate(http\_requests\_total{status=~"5.."}[5m])) / sum(rate(http\_requests\_total[5m]))) \* 100

**Sample log query (ELK/KQL)**

response:500 OR status:5\* | stats count() by endpoint | sort -count

**Sample SIEM query (hunt for suspicious auths)**

Index=auth\_logs action=failure | stats count() by src\_ip, user | where count > 50

**10. Data hygiene, cardinality & cost control**

* **Label cardinality**: avoid high-cardinality labels (user-id, request-id) on metrics. Use them on logs instead.
* **Downsampling & rollups**: keep high-resolution recent data, downsample older data for long-term trends.
* **Retention policy**: metrics (weeks-months), logs (30–365 days depending on compliance), traces (shorter unless stored for analysis).
* **Sampling**: use sampling for traces and logs. Keep error traces unsampled.

**1. Service Health with Golden Signals**  
Golden Signals are the four key indicators to check if a service is healthy:

* **Latency** – how long it takes to serve a request
* **Traffic** – how many requests are coming in
* **Errors** – how many requests are failing
* **Saturation** – how much capacity is left (CPU, memory, queues, etc.)

If these four look good, your service is usually healthy.

**2. CI/CD with DORA**

* **CI/CD** = Continuous Integration and Continuous Delivery (automated builds, tests, and deployments).
* **DORA Metrics** (from DevOps Research & Assessment) measure delivery performance:
  + Deployment frequency
  + Lead time for changes
  + Change failure rate
  + Mean time to restore (MTTR)

**3. Infra and Kubernetes**

* **Infrastructure (infra)** = servers, VMs, storage, and networks your apps run on.
* **Kubernetes (K8s)** = system for running containers (Docker apps). Dashboards show cluster health, node usage, pod restarts, etc.

**4. APM (Application Performance Monitoring)**

* Tools and metrics to measure app performance (slow endpoints, DB queries, cache hits).
* Helps find bottlenecks in code or dependencies.

**5. Logs and Traces**

* **Logs** = events printed by applications and systems (e.g., error messages).
* **Traces** = follow one request across multiple services to see where time is spent.
* Used together for debugging and root-cause analysis.

**6. Incidents**

* Unexpected outages or problems.
* Dashboards show active incidents, owners, severity, and response time.

**7. FinOps**

* Financial Operations — keeping an eye on **cloud costs**.
* Dashboards show spend per service, anomalies, unused resources.

**8. Business KPIs**

* Metrics that matter to the business (e.g., signups, purchases, uptime for customers).
* Ties technical work to business outcomes.

**Security Parts**

**1. SOC Overview**

* **SOC = Security Operations Center.**
* High-level view of security incidents, open alerts, and response times.

**2. Threat Detection**

* Finding suspicious or malicious activity (attacks, malware, intrusions).

**3. Vulnerabilities**

* Weaknesses in systems (like unpatched software).
* Dashboards show open vs fixed vulnerabilities.

**4. IAM (Identity and Access Management)**

* Who can access what.
* Dashboards show failed logins, privilege changes, MFA usage.

**5. EDR (Endpoint Detection & Response)**

* Security for laptops, servers, devices.
* Tracks malware, suspicious processes, and device health.

**6. Network**

* Security of connections.
* Dashboards show blocked traffic, suspicious IPs, unusual data transfers.

**7. Compliance**

* Meeting standards (like ISO, PCI, GDPR).
* Dashboards show which controls are passed or failed.

**8. Supply Chain**

* Risks in third-party libraries or dependencies.
* Dashboards show open-source packages with vulnerabilities (via SBOM).

**Tools & Practices**

**1. OpenTelemetry (OTel)**

* Open standard for collecting metrics, logs, and traces from applications.

**2. Prometheus**

* Open-source system for collecting and querying metrics. Works well with Kubernetes.

**3. Fluentd / Fluent Bit**

* Log collectors. They take logs from apps and servers and send them to systems like Elasticsearch or SIEM.

**4. SIEM (Security Information and Event Management)**

* Central place where all security logs are analyzed. Helps detect attacks.

**Visualization & SLOs**

**Visualization Tips**

* Use clear time-series graphs.
* Show high-level status at the top.
* Allow drill-downs into details (logs, traces).

**SLOs and Error Budgets**

* **SLO (Service Level Objective):** Target reliability (e.g., 99.9% uptime).
* **Error Budget:** Allowed failures before breaching SLO. Example: 0.1% downtime per month.
* Dashboards show how fast you are burning your error budget.