Cloud & DevOps Best Practices

IGNITE BRILLIANCE

Contents

Cloud & DevOps Standards

A Practical Guide to Building Resilient, Scalable, and Secure Systems

---

1. Why This Document Exists

This guide outlines how we approach cloud infrastructure and DevOps—not just to tick boxes, but to make sure our systems are **resilient**, **secure**, **scalable**, and ready to support real business outcomes. It's grounded in best practices from **CMMI**, **ISO/IEC 27001**, **20000**, and **12207**, but more importantly, it's shaped by what actually works for modern engineering teams.

---

2. Who Should Use This?

If you're:

- Designing systems or infrastructure
- Building and deploying code
- Managing environments
- Monitoring performance or responding to incidents
- Auditing for compliance or security

...then this document is for you.

---

3. Architecture Principles

We build systems to be:

- **Cloud-native**: Microservices, containers, serverless where it makes sense.
- **Resilient by design**: We assume failure will happen—and design with retries, fallbacks, and timeouts.
- **Highly available**: Across multiple zones or regions, with no single point of failure.
- **Infrastructure-as-Code**: If it's not in version control, it doesn't exist.

Best practices:

- Use Infrastructure as Code (IaC) to deploy infrastructure, example:
    - Cloud Development Kit (CDK) for AWS deployments.
    - Terraform or Pulumi for cloud agnostic deployments
- Make services stateless wherever possible.

- Include architecture diagrams in your project repo (and keep them updated!).
- Use managed services when possible

---

4. Scalability & Performance

Scalability isn't optional—our systems should handle growth without breaking a sweat.

What we do:

- Auto-scaling groups and serverless wherever it fits (e.g., AWS Lambda, GCP Cloud Run).
- Load testing with tools like **k6**, **JMeter**, or **Artillery** before go-live.
- Use of **caching layers** (Redis, CDN, browser caching) to offload repeated work.

Design tips:

- Plan for growth (3x traffic) in your architecture.
- Watch memory leaks or thread bottlenecks in long-running services.
- Run performance regression tests in CI.

---

5. Security (Built In, Not Bolted On)

Security is everyone's job, but especially ours.

What we enforce:

- **Least privilege access** using IAM roles/policies.
- **Secrets management** using AWS Secrets Manager, HashiCorp Vault, or environment-specific key stores.
- TLS 1.2+ everywhere.
- Container scanning with tools like Trivy or Snyk.
- Mandatory static code analysis (e.g., SonarQube, CodeQL) for every pull request.

Regular practices:

- Threat modeling for new features.
- Security retros after incidents.
- Quarterly penetration testing.

---

6. CI/CD & Deployment

We aim for fast, safe, and reversible releases.

Pipeline guidelines:

- Git-based pipelines (Bitbucket Pipelines).
- Every commit triggers tests, builds, security scans (SonarQube, Trivy).
- Tag all releases; no “Thursday deploys” without business approval.

Deployment patterns:

- **Blue/green**, **canary**, or **rolling** updates depending on risk level.
- Use of **feature flags** for controlled rollouts.
- Pre-prod mirror environments for testing.

Always keep **rollback instructions** ready in your release notes.

---

7. Monitoring, Alerts, and Observability

“If it’s not monitored, it doesn’t exist.”

What we monitor:

- **Service uptime** (HTTP 2xx/5xx ratios, latency, error rates).
- **Infrastructure health** (CPU, memory, disk I/O).
- **Business metrics** (signups, revenue-impacting flows).

Tools we love:

- Prometheus + Grafana
- Cloud-native tools (CloudWatch, Azure Monitor, GCP Operations)
- Real-time alerting via Teams or PagerDuty

Every alert must:

- Be actionable (no “noise” alerts)
- Link to a **runbook**
- Include a severity level and clear ownership

---

8. Logging & Auditability

Logs help us debug, understand, and stay compliant.

Logging rules:

- Use structured logging (e.g., JSON format).
- Mask PII and secrets in logs—ALWAYS.
- Ship logs to a centralized location (ELK Stack, CloudWatch, or equivalent).
- Retain logs as per company policy (typically 90 days for app logs, 1 year for audit logs).

Audit trails:

- Required for privileged access and config changes.
- Automatically captured in cloud provider dashboards or SIEM tools.

---

9. Backups & Disaster Recovery

Stuff breaks. Be ready.

Backup practices:

- Daily backups for DBs, weekly for file systems.
- Encrypted backups stored in separate region or zone.
- Quarterly restore drills to test RPO/RTO.

DR documentation must include:

- Contacts
- Systems impacted
- Restoration priority
- Checklist for DR testing

---

10. Governance & Compliance

This is where ISO and CMMI meet the real world.

Core practices:

- Tag all cloud resources by environment, team, project and client.
- Automate policy enforcement with tools like **AWS Config**.

- Use CMDBs or inventory tools (like ServiceNow).
- Maintain evidence for regular audits (change logs, access logs, vulnerability scans).

---

11. Change & Incident Management

All infra and app changes:

- Go through Git-based PR process with peer reviews.
- Are traceable (commit → ticket → release).
- Include a rollback plan.

Incidents:

- Logged in your ITSM tool (e.g., Jira, ServiceNow).
- Follow our incident playbook and severity matrix.
- Include postmortems with root cause analysis and next steps.

---

12. Knowledge Sharing & Training

We write things down. We teach what we know.

Expect to:

- Maintain internal docs in Confluence or Loop or SharePoint.
- Include architecture, runbooks, and "how to" guides in every repo.
- Host regular internal knowledge-sharing sessions.

For continuous improvement:

- Track **DORA Metrics** (Deployment Frequency, Lead Time, MTTR, Change Failure Rate).
- Run retros after big launches or outages.
- Encourage feedback and improve documentation regularly.

---

13. Review & Maintenance

This document isn’t static. It evolves as we do.

Review schedule:

- Every 6 months or after a major incident or audit.
- Managed by the Engineering Leadership Team.
- Input from all job families not just developers.
