**Cloud - Observability Solutions Approach**

# Table of Contents

## 2. AWS Cloud Strategy

1.0 Introduction

2.0 Guiding Principles

3.0 Key Areas of Focus

4.0 Tooling & Ownership

5.0 Implementation Steps

6.0 Microservices & AWS Resources

|  |  |  |
| --- | --- | --- |
| **Domain** | **Monitoring Targets** | **Tools** |
| Infrastructure | EC2 CPU, EBS burst balance, Lambda cold starts | CloudWatch |
| Application | API Gateway 5xx, Lambda duration, X-Ray traces | X-Ray + CloudWatch |
| Database | RDS replication lag, DynamoDB throttling | CloudWatch |
| Network | VPC flow log anomalies, ALB latency | CloudWatch + X-Ray |
| Critical Logs | CloudTrail events, S3 access denials | Splunk |

7.0 Process & Outcomes

# AWS Cloud Monitoring and Observability Strategy

## 1.0 Introduction

Cloud-native observability for ’s AWS workloads using CloudWatch (metrics), X-Ray (traces), and Splunk (unified insights) for end-to-end visibility.

## 2.0 Guiding Principles

**Pareto Principle**: Focus on Lambda errors, API Gateway latency, and RDS CPU (top 20% high-impact metrics).

**Knowns/Unknowns Framework**: Use X-Ray to uncover hidden dependency issues.

**3.0 Key Areas of Focus**

## 4.0 Tooling & Ownership

|  |  |  |
| --- | --- | --- |
| **Tool** | **Function** | **Owner** |
| CloudWatch | Metrics, alarms, logs for AWS services | Cloud Ops Team |
| X-Ray | Service maps, latency analysis, trace aggregation | Dev Team |
| Splunk | Unified view of CloudWatch/X-Ray data | Operations Team |

## 5.0 Implementation Steps

### Phase 1: Instrumentation (Weeks 1-4)

1. Deploy CloudWatch agents on EC2 instances.
2. Integrate X-Ray SDK with microservices (Java/Python/Node.js).
3. Configure Splunk AWS Add-on for CloudWatch log ingestion.

### Phase 2: Alerting & Tracing (Weeks 5-8)

1. Critical CloudWatch alarms:

API Gateway 5xx >0.5%, Lambda errors >1%.

RDS CPU >80%, DynamoDB throttled requests.

1. Build X-Ray service maps to identify latency bottlenecks.

### Phase 3: Optimization (Ongoing)

1. Automated Splunk dashboards for real-time API health.
2. Monthly chaos testing (e.g., SimulateChaos) to expose "unknown-unknowns".

## 6.0 Microservices & AWS Resources

|  |  |  |  |
| --- | --- | --- | --- |
| **Resource Type** | **AWS Services** | **Monitoring Coverage** | **Deployed**  **Services** |
| Compute | EC2, Lambda, ECS  Fargate | CPU, cold starts, memory utilization |  |
| API Layer | ALB | 4xx/5xx rates, latency, request count |  |
| Databases | RDS Postgres | Replication lag, read/write throughput |  |
| Serverless  Functions | AWS Lambda | Error rates, duration, throttles |  |
| **Resource Type** | **AWS Services** | **Monitoring Coverage** | **Deployed**  **Services** |
| Storage | S3 Buckets | Access denials, latency |  |
| Messaging | SQS, SNS | Queue depth, message age |  |

## 7.0 Process & Outcomes

**Daily**: CloudWatch dashboard reviews; X-Ray anomaly detection.

**Bi-weekly**: Splunk audits for unused CloudWatch logs.

**Outcomes**:

Detect RDS failovers in <2 min.

Reduce "unknown-unknowns" by 25%/quarter through exploratory analysis.

# Appendix: Cross-Environment Alignment

|  |  |  |
| --- | --- | --- |
| **Principle** | **OnPrem** | **AWS Cloud** |
| **Unified Logging** | Splunk for all logs | Splunk ingests CloudWatch logs |
| **Pareto Compliance** | Top 20%: CPU, HTTP errors, disk I/O | Top 20%: Lambda errors, RDS  CPU |
| **Knowns/Unknowns** | Weekly Splunk audits | X-Ray + chaos testing |
| **Blind Spot**  **Reduction** | 15% reduction per quarter | 25% reduction per quarter |