Cognixia

Observability and Maintenance

By - Ahmed

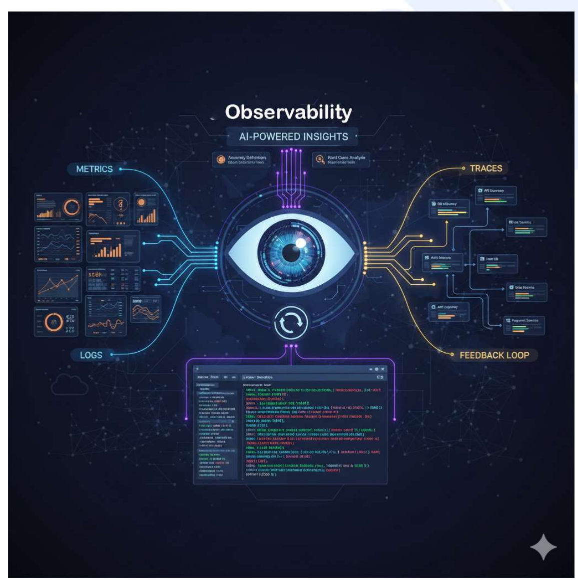
www.cognixia.com

1

Cognixia

What is Observability

- Observability is the ability to **understand the internal state** of a complex software system solely by examining its **external outputs** (*telemetry data*).
- In essence, an **observable system gives** you the **flexibility** to explore and ask arbitrary questions **about** its **behavior**—questions you didn't even know to ask when the system was deployed.
- It **moves beyond knowing *that* something** is **wrong** (traditional monitoring) to **understanding why** it is wrong and **how** to fix it.
- **Tools**: **ELK Stack** (Elasticsearch, Logstash, Kibana), **Splunk**, **Jaeger**, Zipkin, etc.



www.cognixia.com

2

Observability vs Monitoring

- The terms are often **used interchangeably**, but there is a **crucial distinction** that r**eflects** the **complexity of modern**, **distributed architectures** (*like microservices*):
  - **Monitoring (The "Known Unknowns")**
    - **Tells** you **when something** is **wrong** and **what** the **basic problem** is (*e.g., "CPU utilization is at 95%," or "The error rate is above 1%"*).
    - **Monitoring relies** on **predefined metrics** and dashboards.
  - **Observability (The "Unknown Unknowns")**
    - T**ells** you **why** the **problem** is **happening** and **how to fix it**.
    - It allows engineers to **drill down** into the **data** to **trace** a **fault** through **hundreds of services** and **discover** the **root cause**, **even** for **failures** they **hadn't anticipated** or **pre-configured** an **alert** for.

www.cognixia.com

3

Pillars of Observability

- Observability relies on the continuous collection and correlation of three primary types of telemetry data:
- **Metrics (The "What?")**
  - **Metrics** are **quantitative**, **numerical measurements** of a service or system captured over time.
  - Format
    - **Time-series data** (*timestamp, value, name/tag*).
  - Purpose
    - **Provide high-level insights** into **resource utilization** and **service health**.
    - Excellent for setting alerts and trending over time.
  - Examples
    - **CPU usage**, **memory consumption**, **request latency**, **requests per second**, error rate.

4

Pillars of Observability … continue

- Logs (The "Why?")
  - **Logs** are **discrete**, **time-stamped text records** of **events** that **occurred** within an **application** or **system**.
  - Format
    - **Text** (*structured JSON or unstructured plain text*).
  - Purpose
    - **Provide granular context** for **debugging**.
    - **Logs** are **used** to **pinpoint** the **exact sequence** of **events** that led to an error.
  - Examples
    - **User login successful**, **database query failed**, function entered, exception thrown.

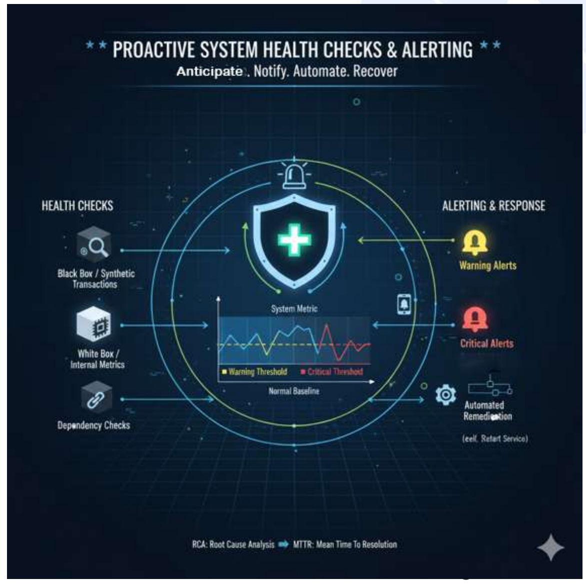
- Traces (The "Where?")
  - **Traces track** the **end-to-end journey** of a **single request** as it **flows** through **all services**, **components**, and **databases** in a distributed system.
  - Format
    - A **collection** of **timed spans**, **each representing** a **segment** of **work done** in a **service**.
  - Purpose
    - **Provide visibility** into the **relationships** and **dependencies** between **services**.
    - **Essential** for identifying latency **bottlenecks** in microservice architectures.
  - Example
    - A request hits the **API Gateway** → **authenticates** with the **Auth Service** → **queries** the **User Profile Service** → **queries** the **Product Catalog Database** → **returns** a **response**.
    - The trace shows how long each step took.

Proactive Health Check

Cognixia®

Proactive Health Checks and Alerting

- **Proactive System Health Checks and Alerting** is a core strategy in modern IT operations that moves system management from being **reactive** (*fixing things after they break*) to **preventative** (*identifying and resolving potential problems before they impact users*).
- It is **achieved** by **continuously measuring key system indicators** against **established performance baselines** and **immediately notifying** responsible teams when a **threshold** is **breached**.



7

Cognixia®

Tools for Health Checks

- A comprehensive strategy requires tools that can **collect metrics**, **visualize** them, and manage the **alerting lifecycle**.

Category	Open-Source Tools	Commercial/Managed Platforms
Metrics & Visualization	**Prometheus** (Metrics Collection & Storage), **Grafana** (Dashboards)	**Datadog**, Dynatrace, New Relic
Infrastructure Monitoring	**Nagios, Zabbix, Icinga**	SolarWinds, **ManageEngine OpManager**
Incident Management	**Alertmanager** (for Prometheus)	**PagerDuty**, Opsgenie (Atlassian), Squadcast
Log Management	**ELK Stack** (Elasticsearch, Logstash, Kibana)	**Splunk**, Datadog Logs

www.cognixia.com

8

Cognixia®

Service Level

www.cognixia.com

9

Cognixia®

Service Level

- A Service Level refers to the specific, measurable **performance standard** or target that is agreed upon for the **delivery** of a **service**.
- It essentially answers the question: "**How well and how quickly will this service perform**?"
- **The Interconnected Relationship**
  - Think of them as a **hierarchy of commitment**:
    - SLI: THE **MEASUREMENT** (*The actual percentage of uptime, e.g., 99.95%*)
    - SLO: THE **TARGET** (*We aim for 99.9% Uptime*)
    - SLA: THE **COMMITMENT** (*If we fail to meet the 99.9% target, we pay you a penalty*)

www.cognixia.com

10

Service Level | SLI

- **Service Level Indicators (SLIs)**
  - SLI is a **numerical measure** that tells **how reliable** the **system is**. The **raw data points** that tell you *what* the service is doing.
  - A **quantitative measure** of a **service's performance** (*e.g., 99.99% of requests returned in under 100ms*). This detects risk.
  - SLIs **measure customer-impacting metrics,** such as:

SLI Type	What It Measures	Example Metrics
Availability	**Service uptime, success response rate**	**% of successful API calls** (2xx)
Latency	**Speed of response**	**% of requests completed under 200 ms**
Error Rate	**Failed** or erroneous **transactions**	5xx errors, **failed login attempts**
Throughput / Traffic	**Requests handled per second**	**API requests/second, trades/sec**
Saturation / Capacity	**Resource** exhaustion **levels**	**CPU, memory,** queue **usage**
User Experience (UX)	**Real user performance**	**App crash rate, page load time**
Business SLI	**Business/user success**	**% successful trades, payment success rate**

11

Service Level | SLOs

- **Service Level Objectives (SLOs)**
  - The agreed-upon **target reliability percentage** (*e.g., 99.95% uptime*), based on SLI data.
  - The explicit, **internal goal** for the reliability of a service. The SLO is an **internal contract** between the SRE team and the Development (Product) team.
  - **Example**
    - The payment API should be 99.95% available over 30 days.
    - 95% of trades should complete within 300ms.
  - It tells **how reliable** the **service SHOULD be**, not to be perfect, but **good enough for customers** at a **reasonable cost**.

SLO Type	What It Measures	Example Metric	Example SLO Statement	Used For
Availability SLO	**% of time service is available**	Uptime %, Error rate	Service will be available **99.9%** of the **time per month**	Service reliability, uptime commitments
Latency / Performance SLO	**Speed of response**	Response time (ms), Page load time	**95%** of **API calls** should respond under **200ms**	User experience, app responsiveness
Throughput SLO	**Volume of successful requests per time**	Requests/sec, Transactions per minute	System should handle **5000 orders per minute**	Scalability, system capacity
Error Rate SLO	**Frequency of failed requests**	% failed requests	Less than **0.5%** of API requests **can fail** in a **week**	Quality of service, reliability
Durability SLO	**Protection of data from loss**	Data loss %	Data durability of **99.999999999%** (11 nines) **per year**	Data storage, backup, cloud services
Reliability SLO	**Successful task completion**	Successful job completion rate	**99.9%** of scheduled jobs should **finish successfully**	Background jobs, workflows

12

Service Level | SLA

- **Service Level Agreements (SLA)**
  - **Contract**
    - The **external**, **legal agreement** with the **customer**, outlining **penalties** for **failure** to **meet** the **SLO** (e.g., **99.5%** with a **10% credit** if missed).
  - **Buffer**
    - SLOs are typically set **tighter** than **the external SLA** to provide a **safety margin**, ensuring that if the team misses the **internal target** (the **SLO**), they still have a chance to fix it before breaching the **external legal contract** (the **SLA**).

SLA Type	What It Ensures	Real-Life Example
Availability SLA	Uptime guarantee	Server SLA: 99.99%
Support SLA	Response/resolution time	P1 issue responded in 15 min
Performance SLA	Response speed guarantee	95% of transactions < 200ms
Data Protection SLA	Backup, retention, durability	99.999999999% (11 nines) data durability
Compliance SLA	Security, regulation, audit	GDPR, PCI-DSS, RBI guidelines
Penalty-based SLA	Refund/credit if breached	Refunds 25% monthly bill

www.cognixia.com

13

Pipelines

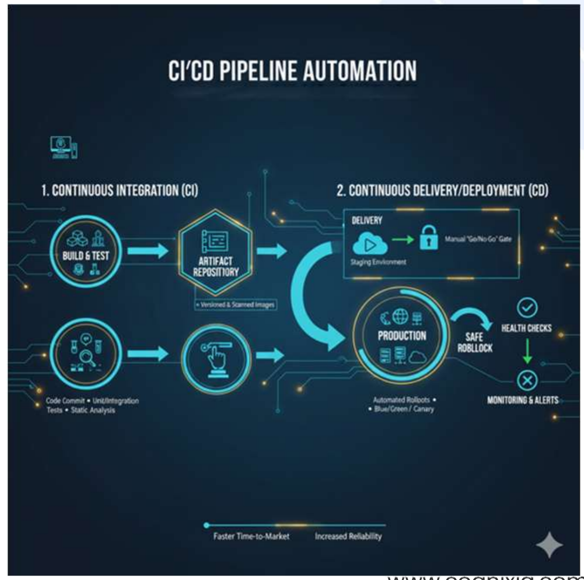
www.cognixia.com

14

What is Pipelines

Cognixia

- **CI/CD Pipelines** are **automated processes** that **take code changes** from a **developer's machine** and reliably deliver them **to production**.
- They are the **backbone** of **modern software development**, enabling rapid, frequent, and reliable software releases.
- The acronym stands for **Continuous Integration (CI)** and **Continuous Delivery/Deployment (CD)**.



www.cognixia.com

15

Phases of the Pipeline

Cognixia

- **Continuous Integration (CI)**
  - CI focuses on automatically merging developer code changes into a central repository frequently.
  - **Action**
    - A **developer commits code**.
  - **Process**
    - The **pipeline automatically performs**:
      - ✓ **Building**
        - ❖ **Compiling the code** into an executable artifact (*e.g., a JAR file, Docker image*).
      - ✓ **Testing**
        - ❖ **Running unit tests**, **integration tests**, and static code analysis to immediately catch bugs or security flaws.
  - **Outcome**
    - If **tests pass**, a stable, re**ady-to-deploy artifact** is created and stored (*e.g., in a container registry*).

- **Continuous Delivery / Deployment (CD)**
  - **CD focuses** on **automating** the **release** of the validated artifact to various **environments** (*staging, production*).
  - **Continuous Delivery**
    - The **tested artifact** is delivered to a **central repository** (e.g., Docker registry) and is ready for manual, one-click deployment to production.
    - *A **human decision** is **required** for the **final production push**.*
  - **Continuous Deployment**
    - The **tested artifact** is **automatically deployed** to the **production environment** without any human intervention.
    - ***Automation** is **complete**.*

www.cognixia.com

16

Safe Rollbacks

- The **pipeline** is **designed** not just for forward deployment but also for **safely reversing** a **change** if a critical error is detected in production.

- **Automated Verification**
  - **After deployment**, the **pipeline runs post-deployment health checks** (*e.g., synthetic transactions, latency checks*) to **confirm** the **new version** is **operating correctly**.

- **Rollback Trigger**
  - **If** the **health checks fail** or if monitoring/observability tools detect a critical performance degradation (*e.g., a spike in error rates*), the **pipeline's failure path** is **automatically executed**.

- **Rollback Mechanism**
  - The most **common safe rollback st**rategy is to **re-deploy the immediately preceding, known-good version** of the application artifact.
  - Because the pipeline archives all successful artifacts, reverting is simply another automated deployment process.
    - ✓ **Canary/Blue-Green Rollbacks**
      - ❖ Modern CD strategies use techniques like **Blue/Green** (*maintaining the old version alongside the new*) or **Canary Releases** (*rolling out the new version to a small subset of users*).
      - ❖ If the **new version fails**, traffic is **instantly shifted back** to the stable, **old environment** with minimal impact on users.
      - ❖ This ensures the system maintains high availability even when a new version is failing.

www.cognixia.com

17

Q & A

concepts still unclear?

Thank you for attending

www.cognixia.com

18

19