SRE Principles
SLIs, SLOs & Error Budgets

1

Key SRE Principles

2

Core Principles of SRE

- SRE is defined by a set of **key principles** and **practices** that **govern** how **reliability** is **managed** and **measured**.
- Site Reliability Engineering (SRE) is about **making systems reliable using engineering practices**.
- Instead of fixing problems manually, SRE teams build automation, monitoring, and tools to keep services running smoothly.
- SRE is defined by a software engineering approach to operations, **centered** on **three core principles** that **drive all decisions**:
    - Embracing Risk and Setting Reliability Goals
    - Elimination of Toil with Scripts and Automation
    - Monitoring Everything That Matters

AHMAD

3

Embracing Risk and Setting Reliability Goals

AHMAD

4

Reliability Goals

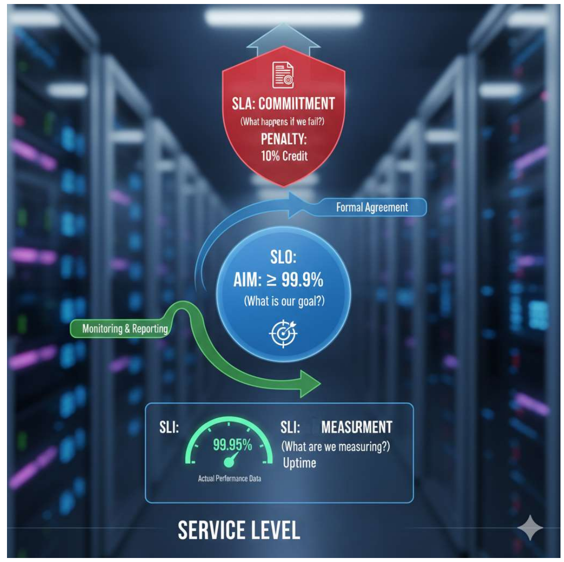
- This is the most **philosophical break** SRE makes **from traditional operations**, which often **aims** for ***100%* uptime**.
- **SRE acknowledges** that **100% availability** is **impossible** — it is **too expensive**, **slows innovation**, and **often unnecessary**.
- **Reliability** is treated as an explicit, **measurable goal**, not a vague wish.
- SRE manages the **trade-off** between **Agility** (*speed of releases*) and **Reliability** (*stability*) using **SLOs (Service Level Objectives)** and **Error Budgets** to balance reliability and agility.
- Reliability in SRE is built on four pillars:
  - SLI (What we measure)
  - SLO (What we target - goa)
  - SLA (What we promise customers - legal/commercial)
  - Error Budget (How much failure is allowed)

5

6

Reliability Goals | 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*)



AHMAD

7

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**
Durability	**Data integrity** & **loss protection**	**% of data loss**-free operations
User Experience (UX)	**Real user performance**	**App crash rate, page load time**
Business SLI	**Business/user success**	**% successful trades, payment success rate**

AHMAD

8

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**.
  - The SLO is the most crucial terms, as it **directly drives engineering behavior** and defines the Error Budget.

AHMAD

9

Service Level | SLOs ... continue

- SLOs are categorized based on **what type of reliability** we are measuring.

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

AHMAD

10

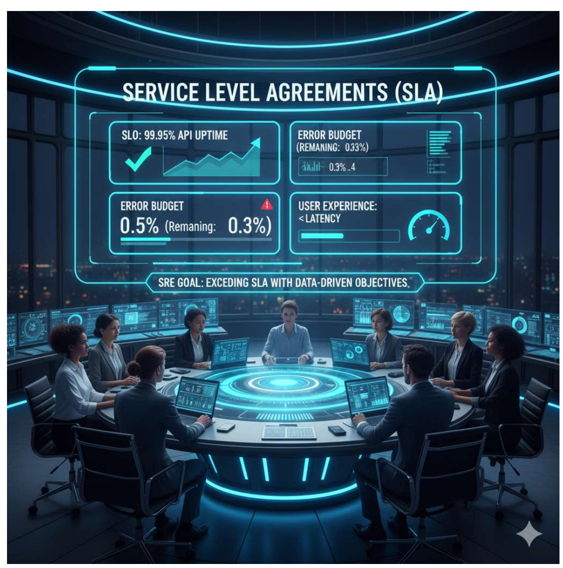
Service Level | SLOs … continue

SLO Type	What It Measures	Example Metric	Example SLO Statement	Used For
Security SLO	**Protection against threats**	Vulnerability fix time	Critical **security issues resolved** within **72 hours**	Security operations, compliance
Freshness / Data Staleness SLO	**Timely data updates**	Sync delay (mins), Data lag	**Data replication lag** must be under **5 minutes**	Data pipelines, analytics
Capacity SLO	**Handling load without degradation**	CPU, RAM, storage usage	System must maintain < **70% CPU usage** under peak load	Infrastructure scaling
Customer Experience SLO	**Satisfaction & usability**	CSAT, NPS, App rating	Maintain **90%+ customer satisfaction** score	Business-level user experience

11

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**).



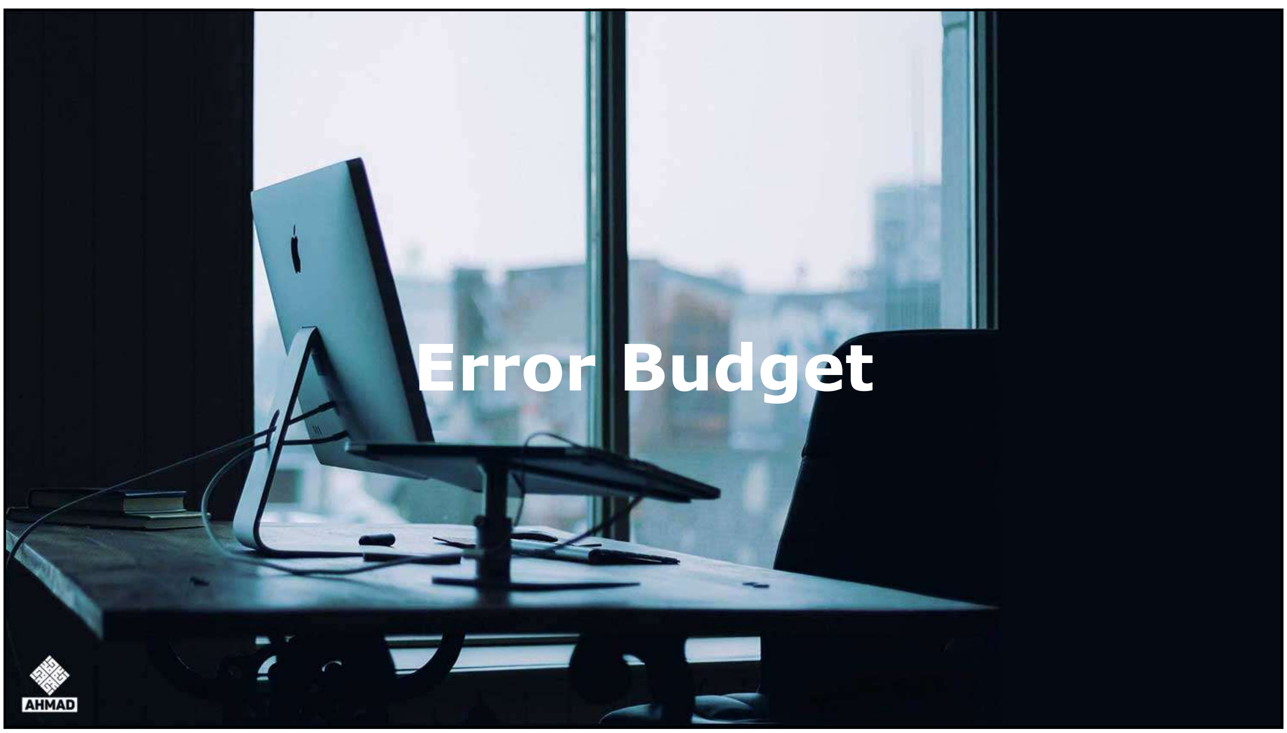
12

Service Level | SLA … continue

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

AHMAD

13



14

What is Error Budget

- The **Error Budget** is the **maximum amount** of **time** a **service** is **allowed** to be **unavailable** or fail over a **specific period** (e.g., one month) without violating its **Service Level Objective (SLO)**.

- It is fundamentally a **governance tool** used to **create alignment between Development teams** (*who prioritize feature speed*) **and SRE teams** (*who prioritize reliability and stability*).

AHMAD

15

Reliability Goals | Error Budget

- **Error Budget**
  - The **acceptable amount** of time the **service** is **allowed to fail** (*calculated as 100% - SLO*).
  - Calculation
    - The Error Budget is mathematically derived from the SLO and is **calculated** as the inverse of the SLO:
      Error Budget = 100% - SLO
    - **Example**:
      - ✓ If your **SLO** (target reliability) is **99.9%** ("three nines"), your **Error Budget** is **0.1%.**
      - ✓ Over a **30-day period** (43,200 minutes), **0.1%** of the time **equals approximately 43 minutes** of acceptable **downtime** or error allowance.
    - **Error budget formula**:

SLO (%)	Error Budget (%)	Max Allowed Failure Time (monthly)
99%	1%	~7.2 hours
99.9%	0.1%	~43.2 minutes
99.99%	0.01%	~4.3 minutes
99.999%	0.001%	~26 seconds

AHMAD

16

Reliability Goals | Error Budget … continue

- The Error Budget **provides** the SRE team with the necessary **authority** to manage risk:
  - **When the budget is full**
    - ✓ Dev teams can **move fast**, deploying new features frequently.
    - ✓ They are ***embracing risk*** because they have a **buffer**.
  - **When the budget is depleted (spent)**
    - ✓ All new **feature development** must **stop** (a feature freeze).
    - ✓ Engineering **resources must be redirected to reliability work** (*bug fixes, technical debt reduction*) **until the budget is replenished**.
- **Error Budgets** can be **applied in different ways** depending on what is being measured:

Error Budget Type	Description	Example
Availability Budget	Allowed downtime	**Max 43 minutes per month**
Latency Budget	Allowed slow responses	**No more** than **1% calls > 500ms**
Error Rate Budget	Allowed failed API requests	**Up to 0.1%** request failures
Data Loss Budget	Allowed inconsistency/drop	**99.999% durability** => error budget 0.001% loss
Performance Budget	Allowed performance drop	**95% of orders processed** under **3 seconds**
Security Budget	Allowed exposure time / risk	**Fix critical issues** within **72 hours**

AHMAD

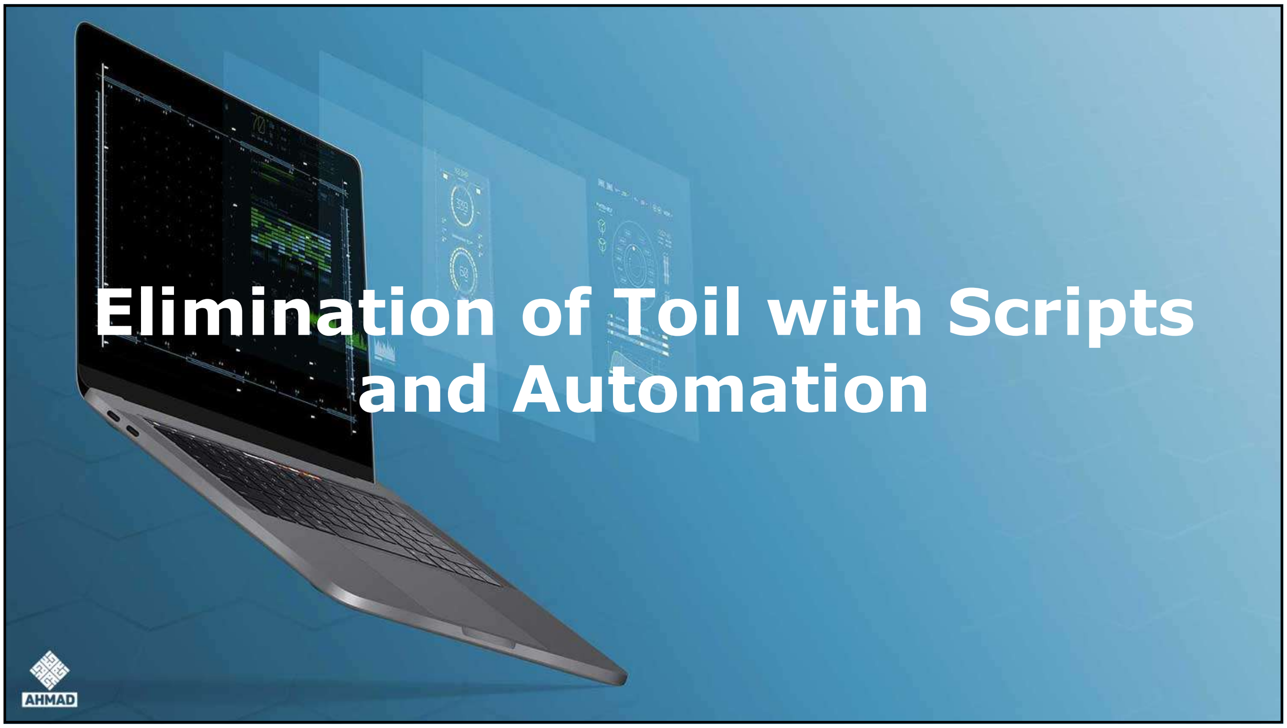
17

Use Case | Stock Exchange

Concept	Meaning	Purpose	Stock Exchange Real-Life Example	Measurement/Criteria
SLI (Service Level Indicator)	Actual measured performance	Shows "how system is performing"	• **98.5%** of **trades completed** within **50ms;** • **0.02% error rate**	• Latency, uptime %, failure rate, data accuracy
SLO (Service Level Objective)	Target performance goal	Defines "how reliable system should be"	• **99.99% uptime** during **trading hours** • **<100ms trade processing time**	• Based on business demand & traffic
SLA (Service Level Agreement)	Legal/business contract with customer	Defines penalties if reliability is broken	• **If uptime <99.95%,** stock exchange owes **financial compensation** to brokers/banks	• Uptime %, time-to-resolution • compensation terms
Error Budget	Allowed failure limit before reliability is affected	Balance reliability and innovation	• **For 99.99%** • **SLO → allowed downtime = 4.3 min/month**	• Error Budget = 100% - SLO

AHMAD

18



Elimination of Toil with Scripts and Automation

19

Automation & Engineering

- The **Elimination of Toil** is a foundational principle of Site Reliability Engineering (SRE).
- It is the mechanism SREs use to move from a **reactive firefighting role** to a **proactive engineering role**.
- **What is "Toil"**
  - **Repetitive**, **manual**, **ticket-based**, **operational work** that adds **no long-term value.**
  - **Examples of toil:**
    - Ticket processing
    - Log checking manually
    - Restarting failed services
    - Manual user provisioning

20

Automation & Engineering … continue

- SRE goal → **Automate repetitive work** to **free engineers** for **innovation.**
- **Characteristics of Toil**
  - **Manual** → Requires humans to perform it
  - **Repetitive** → Same process repeated frequently
  - **Reactive** → Triggered by alerts, tickets, or incidents
  - **No long-term value** → Does not improve the system
  - **Scale-dependent** → More workload, more manual effort
- **Key Practices:**
  - **The 50% Rule**
    - SREs must **spend no more** than **50%** of their **time** on **toil**.
    - The other 50% is dedicated to Engineering (*writing code for automation, tools, and resilience*).
  - **Infrastructure as Code (IaC)**
    - Using tools like Terraform or Ansible to **manage infrastructure using code**, eliminating manual configuration.
  - **Automated Runbooks**
    - Turning manual incident **mitigation steps into scripts** that can be **triggered automatically**.

AHMAD

21

Types of Toil in IT Operations

Type of Toil	Description	Example in Industry
Monitoring Toil	Responding to repetitive alerts	Low CPU alerts at night
Deployment Toil	Manually deploying code/apps	Change requests executed manually
Configuration Toil	Manual server or firewall setup	SSH into each VM to update settings
Incident Toil	Handling same type of incidents repeatedly	Restarting crashed services daily
Reporting/ Documentation Toil	Creating manual weekly reports	Manually compiling uptime metrics
User/Access Management	Manually provisioning accounts	Creating SSH access requests for engineers

AHMAD

22

How Automation Eliminates Toil

Manual Operation (Toil)	Automated (Engineering-Based)
Manually restarting failed services	Self-healing automation using scripts
Logging into VMs to install patches	Automated patching via Ansible/Jenkins
Email-based change approvals	GitOps-based auto change control
Manual ticket creation after incident	Auto-ticket creation via ServiceNow API
On-call engineers checking alerts all night	Intelligent alert suppression and AI-based routing

AHMAD

23

Use Cases

- **Automated Payment Gateway Recovery (Bank)**
  - **Problem**: **Payment system crashes frequently** during **peak hours** → **manual restart** takes **30 minutes** → **revenue loss**.
  - **Automation Solution**
    - **SRE** creates an **auto-recovery sc**ript:
      - ✓ Detects API failure
      - ✓ Restarts pods/servers automatically
      - ✓ Notifies engineers on Slack
    - **Manual time**: 30 mins → **Automated: 1 min**
    - **Result**: **80% incidents self-healed**, zero customer impact.

- **Automated Trade Order Scaling (Stock Exchange)**
  - **Problem**: During **market opening** (*9:15 AM*), **order load spikes 15x** → **website slow** → manual **scaling** takes **1 hour**.
  - **Automation Solution**
    - **Auto-scale** Kubernetes clusters using CPU, QPS, and latency triggers
    - Introduced **auto-deployment** verification **check**
    - **Manual scaling**: 1 hour → Auto-scaling: **2 minutes**
    - Zero revenue loss during market hours.

AHMAD

24

Tools Used for Toil Automation

Purpose	Tools Used
Infrastructure automation	**Terraform**, **Ansible**
Monitoring & Auto-healing	**Prometheus**, **Grafana**, AlertManager, PagerDuty
CI/CD automation	**Jenkins**, Spinnaker, **GitHub Actions**
Incident auto-resolution	Runbook Automation (**StackStorm**)
GitOps Change Management	**ArgoCD**, FluxCD

AHMAD

25

26

27

Monitoring

- Monitoring is the continuous process of collecting, analyzing, and visualizing data (metrics, logs, and traces) about a system to understand its behavior, detect problems, and inform decision-making.
- The modern approach to monitoring and system understanding, called Observability, relies on three primary types of data, often referred to as the **Three Pillars**:
  - **Metrics** (What)
  - **Logs** (Why)
  - **Traces** (Where)
- **Events** are a related, specialized type of **record** often captured **within logs** or treated as a distinct data source.
- Together, these data types provide a complete picture, answering the questions: **What is happening? Where is it happening?** and **Why is it happening?**

AHMAD

28

Metrics: The Quantitative View ("What")

- Metrics are **numerical measurements** that represent the **health** and **performance** of a system over time.
- They are the most efficient way to track trends and trigger alerts.
- Metrics are structured, **time-series data**: a **number collected** at a **specific timestamp** and enriched with descriptive labels (e.g., service name, region).
  - **Granularity: Aggregated**
    - **Metrics** are **summarized values** (*averages, sums, percentiles*) over a **time interval** (*e.g., 99th percentile latency over the last minute*).
  - **Purpose**
    - **Ideal** for **monitoring dashboards**, setting **Service Level Objectives** (SLOs), and generating **real-time alerts** when a threshold is breached (*e.g., CPU usage is 80%).*
  - **Examples**
    - Response time/**Latency**, **Request Rate**, **Error Rate**, **CPU/Memory Utilization**.

AHMAD

29

Logs: The Detailed Narrative ("Why")

- Logs are **immutable, timestamped text records** of discrete events that occurred within an application or system. They are the system's diary, providing rich contextual detail.
- **Textual data**, often **unstructured** or **semi-structured** (like JSON), **generated** when **code executes** or a **system action occurs**.
- **Granularity: Granular and Contextual**
  - **Each line** typically **refers** to a **single event** with **specific details**, including the **exact state** of the **program at** that **moment** (*e.g., variable values, stack traces*).
- **Purpose**
  - Essential for **root cause analysis**, **forensic debugging**, and **auditing**.
  - When a metric alerts, SREs turn to the logs to find the "why."
- **Examples**
  - [ERROR] NullPointerException in payment service at line 45
  - [INFO] Database connection established
  - [WARN] Cache miss for user ID 12345.

AHMAD

30

Traces: The Request Journey ("Where")

- Traces, or **Distributed Tracing**, **map** the **entire life cycle** of a **single request** or **transaction** as it **moves** through a complex, distributed **system** (*like microservices*).
- **Span**
  - A span is a **single, logical unit of work performed** by a **service** or **component**, such as:
    - An incoming or outgoing HTTP request.
    - A database query.
    - A specific function execution.
- A **trace** is a **collection** of **spans**. All **spans belonging** to the **same request** share a unique **Trace ID**.
- **Granularity: End-to-end journey**
  - **They link together** the **operations across service boundaries**.
- **Purpose**
  - Crucial for **diagnosing latency and performance bottlenecks** in microservices.
  - It **visually shows** which **service** is **responsible** for most of the **delay**.
- **SRE Use**
  - When a latency metric spikes, the trace for a slow request immediately points to **which service** in the call graph introduced the delay.

AHMAD

31

Events: The State Change ("When")

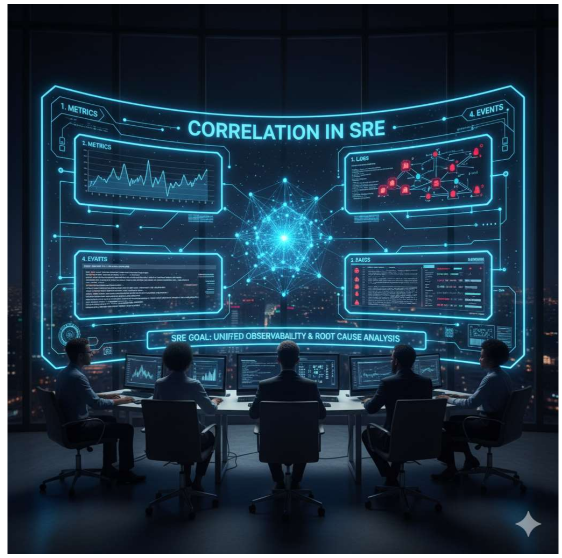
- While **often captured within logs** or used to derive metrics, **Events** are distinct occurrences that **represent** a significant, **discrete action** or **change of state** in a system.
- Structured records of key incidents or changes, **generated** by both **systems** and **humans**.
- They are generally of higher importance than routine log messages.
- **Granularity: Discrete Occurrence**
  - They **mark** a **specific moment** in **time.**
- **Purpose**
  - Primarily used for **correlation** and **context**.
  - They **answer** the question: "**What external action occurred right before the problem started?**"
- **Examples**
  - DeploymentStarted (version 3.1.2)
  - ServiceScaledUp (3 to 5 instances)
  - ConfigurationChanged (feature flag X disabled)
  - SecurityAlertTriggered

AHMAD

32

The Value of Correlation

- The true power of Observability comes from **correlating** these data types.
- **For instance**:
  1. A Metric (**Request Rate**) **drops suddenly**.
  2. The SRE checks **Events** and sees that a **DeploymentStarted** event occurred seconds before the drop.
  3. The SRE then pivots to **Traces** for the **affected service**, seeing a **high failure rate**.
  4. Finally, the SRE drills into the **Logs** linked to the failed traces, finding the exact **stack trace** from the new code that caused the issue.
- This workflow **dramatically speed**s up the **Mean Time To Resolution** (MTTR).

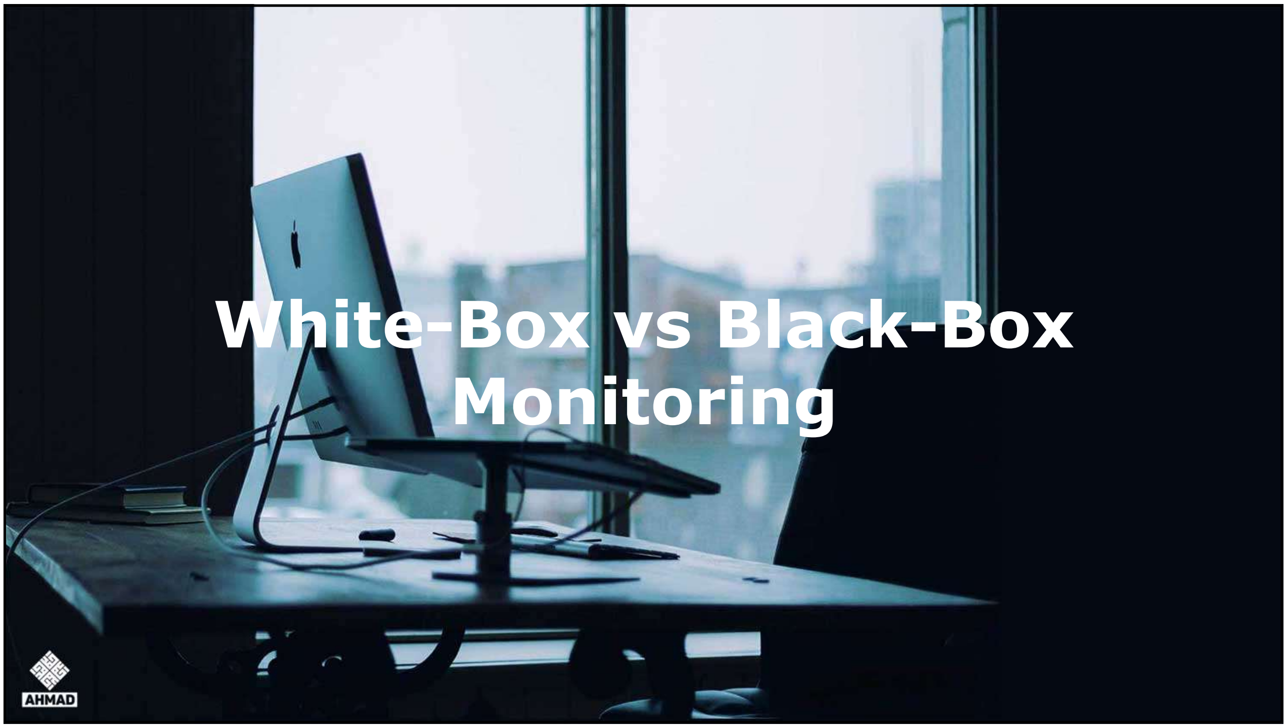


33

Defining Meaningful Metrics and Alerts

- **Effective monitoring** starts with choosing **what** to **measure** and **deciding** when to **raise an alarm**.
- The goal is to maximize **signal** (*real, actionable problems*) and minimize **noise** (*irrelevant alerts*).
- **Meaningful Metrics**
  - Metrics are numerical, time-series data points that **quantify system performance**.
  - **Meaningful metrics align** with your **business goals** and directly **reflect** the **user experience**.
- **Effective Alerts**
  - An alert is a notification that a metric has crossed a predefined threshold or pattern, indicating a problem that requires attention.

34



35

White-Box vs Black-Box Monitoring

- These **two concepts** describe the perspective from which you **monitor a system**. A **complete monitoring strateg**y typically **combines both**.

- **Black-Box Monitoring**
  - Black-box monitoring **treats** the **system** as an **opaque**, closed-off entity—a "**black box**".

  - It is symptom-oriented and **checks externally visible behavior**.
    - Perspective
      - ✓ The **end-user's point of view**.
    - Data Source
      - ✓ **External probes**, synthetic t**ransactions** (*simulated user requests*), and **external system calls** (*e.g., ping, HTTP checks*).
    - Focus
      - ✓ **What is broken**? (*Is the service available, fast, and correct?)*
    - Example Metrics
      - ✓ Uptime/Availability *(e.g., is the login page returning a 200 status code?*).
      - ✓ External Latency (*how long a complete user transaction takes from outside the network*).
      - ✓ DNS Resolution Time from a global location.
    - Purpose
      - ✓ To confirm the **Service Level Objective** (SLO) — the promised level of service — is being met.
      - ✓ It ensures that **regardless of internal health**, the **system is working for the customer**.

AHMAD

36

White-Box vs Black-Box … continue

- **White-Box Monitoring**
  - White-box monitoring **uses information exposed** by the **internals** of the **system** — it **looks inside** the "**white box**".
  - It is **cause-oriented** and requires **direct instrumentation** of the **application code** or **host operating system**.
  - **Perspective**
    - The **system's internal** components and code.
  - **Data Source**
    - **Application logs**, **custom metrics** exposed by the **code** (*instrumentation*), and **operating system**/agent data.
  - **Focus**
    - **Why is it broken?** (*What is the root cause: high memory, database lock, slow function call?*).

AHMAD

37

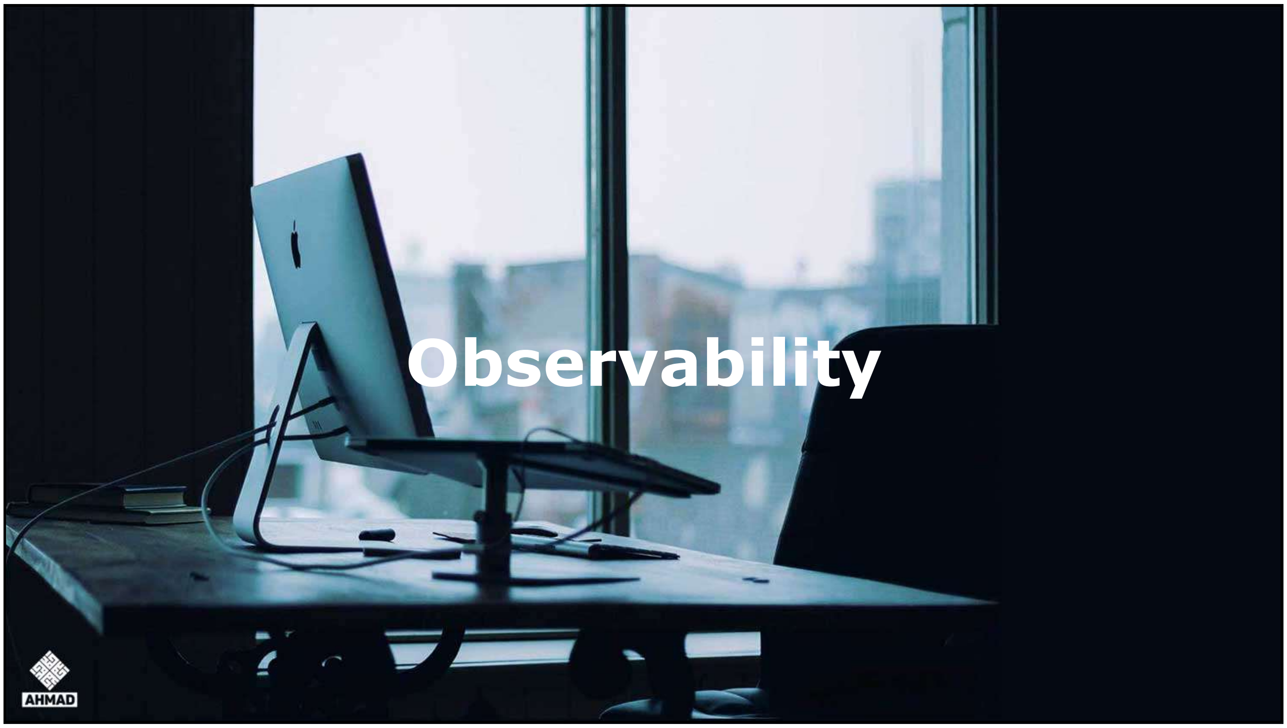
White-Box vs Black-Box … continue

- **Example Metrics**
  - Internal CPU/Memory/Disk Usage of a specific server.
  - Garbage Collection frequency and duration.
  - Internal Queue Lengths or Thread Pool Utilization.
  - Specific function execution times measured within the application code.
- **Purpose**
  - To provide the granular detail needed for **debugging, capacity planning, and proactive issue detection** before a complete user-visible failure occurs.

Aspect	Black-Box Monitoring	White-Box Monitoring
View	**External** (User's Perspective)	**Internal** (System's Perspective)
Focus	**Symptoms** (Availability, Latency)	**Causes** (Resource Utilization, Code Logic)
Action	**Reactive** (Service is *currently* down/slow)	**Proactive** & Debugging (Service *may* go down; find the root cause)

AHMAD

38

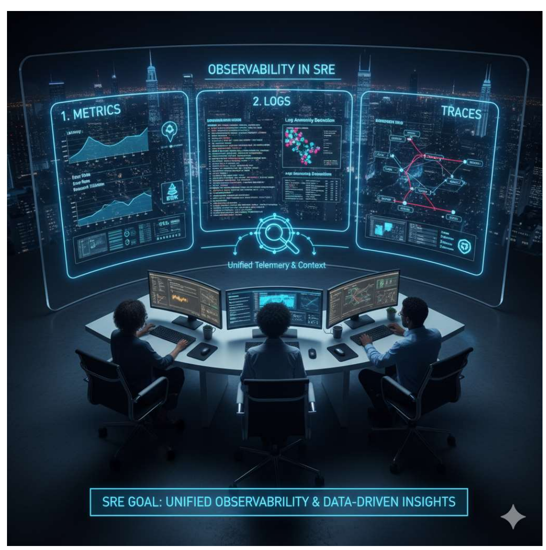


Observability

39

Observability

- SRE recognizes that traditional monitoring (*checking if a server is alive*) is insufficient for complex distributed systems.
- You must monitor what the user *experiences*.
- The shift from generic monitoring to **Observability**, which means collecting and analyzing data that allows engineers to ask novel questions about the system without knowing the answer beforehand.
- Unlike **traditional monitoring**, which answers *"Is the system up?"*, **Observability answers** *"Why is the system slow right now, and which component is causing it?"*



40

Observability | Principles

- SREs prioritize **monitoring** the system based on the **user experience** and **business impact**.
- The **Four Golden Signals** are the definitive **set of metrics** that **must** be **tracked** for every user-facing service:

Signal	Definition	Why it Matters (User Impact)
Latency	The **time** it takes to **serve a request**.	High latency means a **slow user experience**, **frustrating customers**.
Traffic	A measure of how much demand is being placed on your service (e.g., **requests per second**).	Used for capacity planning and determining if a **latency spike** is due to **high load**.
Errors	The **rate of requests** that **fail** (explicitly via 5xx errors or implicitly via bad data).	**Direct measure** of **user-facing failure**. Every error spends the **Error Budget**.
Saturation	How full the service is; a measure of **system utilization** (*CPU, memory, disk I/O, queue depth*).	**Predicts future failure**. High saturation means the system is about to degrade performance or crash.

AHMAD

41

Observability | Pillars

Pillar	Purpose	What It Helps Identify	Tools & Tech	Real Use Case (Industry)	Benefits
Metrics	**Collect numeric performance** indicators **over time**	**Resource usage**, performance trends, saturation (capacity), success / failure ratios	Prometheus, CloudWatch, Datadog, Grafana	**Stock Exchange:** Monitor matching engine latency & CPU usage to ensure trade orders execute under 50ms	Proactive detection, trend analysis, capacity planning
Logs	**Record** detailed **events**, **system messages**, and **error** details	**Root cause of failures**, system bugs, security attempts, transaction history	ELK (Elastic, Logstash, Kibana), Splunk, Fluentd	**Banking:** Track failed payment logs, fraud detection logs, ATM withdrawal issues	Helps in debugging, audit trails, compliance, RCA
Traces	Follow a request across microservices (**end-to-end tracking**)	**Identify bottlenecks**, slow microservices, API dependency issues	OpenTelemetry, Jaeger, Zipkin	**E-Commerce:** Track order from cart → payment → inventory → shipping service and find where delay happens	Deep visibility into distributed systems
Events / Alerts	**Notify** when **thresholds** or anomalies **occur**	Unexpected failures, security breaches, **system outages**	PagerDuty, Prometheus Alert Manager, Opsgenie,	**Healthcare System:** Alert if medical record system response time exceeds 2 seconds	Real-time detection & response

AHMAD

42

Observability | Pillars … continue

Pillar	Purpose	What It Helps Identify	Tools & Tech	Real Use Case (Industry)	Benefits
Business KPIs (Service Health)	**Track business impact** instead of just system health	**Customer loss, transaction drop**, revenue impact, churn rate	Datadog, New Relic, Google Analytics, Dynatrace	**Stock Exchange:** Number of failed trade settlements or delayed transaction confirmations	Bridges business & technical metrics, aligns SRE with business goals
User Experience Monitoring (RUM & Synthetic)	Measure real **user interactions** & experience	**Page load time**, mobile app crash, sign-in failure	New Relic, Dynatrace, Datadog RUM, Pingdom	**Online Banking:** Detect when customers struggle with login or fund transfer due to UI lag	Captures true user experience; helps identify frontend issues
Security Monitoring	Monitor **threats**, suspicious access, vulnerabilities	**Unauthorized access**, DDoS, fraud detection	SIEM, Splunk Enterprise Security	**Banking:** Detect brute force login attempts or fraudulent ATM transactions	Protects reliability, compliance, trust
Infrastructure Monitoring	Monitor **servers**, **network**, **storage**	**VM crash**, **disk full**, **network latency**	Nagios, Zabbix, Datadog Infra	**Telecom Industry:** Monitor 5G tower network connectivity, loss, signal strength	Ensures platform stability and environment reliability

43

Observability | Benefits

Benefit	Explanation
Faster Incident Detection	**Reduces MTTR** (Mean Time to Recovery)
Predictive Reliability	**Detect failures *before* they impact** customers
Prevents SLA Breaches	Helps **stay within error budget**
Enables Auto-Remediation	Integrates with tools like StackStorm
Helps SRE, DevOps, and Business	Unifies technical & business insights

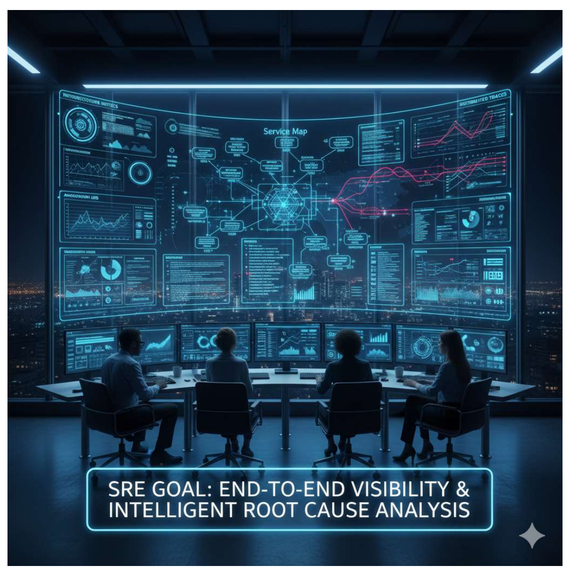
44

Full-Stack Observability

45

What is Full-Stack Observability

- Full Stack Observability provides **deep, holistic visibility** into the performance and behavior of an application, from the **end-user interaction** all the way down to the **underlying network** and **infrastructure**.
- It integrates the three pillars of observability: **Metrics**, **Logs**, and **Traces**, across every component of the system.



46

Beyond Infrastructure

- **Full Stack Observability** ensures **comprehensive monitoring** across three critical layers that are **often siloed** in **traditional monitoring** setups:
  - **Application (App) Visibility**
    - Focuses on the **code** and **services** that make up the **application** (*e.g., microservices, APIs, databases*).
    - **Key Data**: **Traces**, **Metrics** and **Logs**.
  - **Network Visibility**
    - Focuses on the **data flow between application components**, users, and the cloud/data center.
    - **Key Data**:
      - **Network latency**, **packet loss**, **bandwidth utilization**, **firewall performance**, and **load balancer** health.
      - This is crucial for modern distributed systems where network slowness can often be mistaken for application issues.

AHMAD

47

Beyond Infrastructure

- **User Experience (UX) Visibility**
  - Focuses on the actual experience of the end-user, often through **Real User Monitoring** (RUM) and **Synthetic Monitoring**.
  - **Key Data**:
    - Frontend Performance
      - **Page load times**, core web vitals (*Largest Contentful Paint, Cumulative Layout Shift*), JavaScript **errors**, and **interaction latency**.
    - Business Impact
      - **Conversion rates**, cart abandonment rates, and **other key business metrics** tied to performance.

AHMAD

48

Data for Root Cause Analysis (RCA)

- The true power of **Full Stack Observability** lies in its ability to **correlate data across** the entire **stack**.

- **Enables Better RCA**:
  - **Unified Data Platform**
    - All **Metrics**, **Logs**, and **Traces** are ingested and linked within a **single platform**.
  - **Contextual Tracing**
    - A **single user request** is **tracked** (via a Trace ID) from the user's browser, through the load balancer, across multiple microservices, and **down** to the **database query**.
    - If a database query is slow, the trace immediately highlights the specific service and the offending query, linking it to associated error logs.
  - **Automatic Correlation**
    - The platform **automatically identifies dependencies** and **anomalies**.
    - For example, if **Application Metrics** show a **spike** in **errors**, the FSO system can automatically **check if Network Metrics** show **increased latency** or if Infrastructure Metrics show a **spike** in **CPU utilization** at the exact same time.
  - **Faster MTTI and MTTR**
    - This correlation dramatically **reduces** the **Mean Time To Identify** (MTTI) the component causing an incident and the **Mean Time To Resolve** (MTTR) it, as SREs spend less time navigating tool silos.

AHMAD

49

50

Balancing Velocity with Reliability

- The core purpose of an **Error Budget** is to serve as a **risk regulator**, helping **balance** the ongoing tension **between** moving **quickly** (*agility and velocity*) and maintaining **stability** (*reliability*).

- **Feature Velocity**
  - **Developer**s are incentivized to **release features quickly** and often, which inherently introduces risk.

- **Budget as a Buffer**
  - The Error Budget acts as a buffer.
    - As long as the **budget is healthy** (full), the teams can maintain **high velocity**, knowing they have room to absorb minor bugs or failures.

- This mechanism ensures that reliability commitments are **enforced by data** rather than endless meetings or subjective arguments.

51

Drive Engineering Decision

- The Error Budget is a powerful tool for strategic prioritization:
  - **Prioritizing Toil Elimination**
    - SREs can **justify spending time** on **automation** by calculating the potential **budget savings**.
    - ***Example***
      - If **manual configuration** causes one **10-minute outage per month** (*spending 10 minutes of the budget*), **automating** it **permanently justifies** taking one week off feature work to write the automation code.
  - **Guiding Test Coverage**
    - If a **critical service** has a **very tight SLO** (*e.g., 99.999%),* its **Error Budget** is **minuscule**.
    - This drives an engineering decision: **Invest heavily** in **automated testing** and staging environments to **ensure changes** are virtually **risk-free before deployment**.

52

Drive Engineering Decision … continue

- **Informing System Design**
  - Services with **large**, **unused budgets** are **prime candidates for de-scoping** (*reducing over-engineering*).
  - **Services** that **constantly burn their budget** need urgent investment in **redesign** or **architectural** changes to **improve resilience**.
- **Blameless Learning**
  - **After** an **incident**, the post-mortem **focuses** on **how much budget** was **spent** and **why**.
  - The key action item is always an engineering task to **prevent** the **budget-spending** incident from recurring, thus driving a culture of continuous improvement.

AHMAD

53

Help in Decisions

Scenario	Decision
Budget mostly unused	**Approve new releases**, experiments, A/B testing
Budget moderately used	**Approve with caution**, implement canary deployment
Budget nearly exhausted	**Change freeze**, focus on performance & bug fixes
Budget completely consumed	**Block deployments**, trigger high-priority reliability review

AHMAD

54

SRE Roles & Responsibilities

55

SRE Roles and Responsibilities

- An SRE is a software engineer who possesses deep operational knowledge.
- Their time is rigidly split between managing the current system and engineering future improvements.

Responsibility	Traditional IT Ops Approach	SRE Approach
Monitoring	Server uptime, CPU, memory	Service health, **SLO**, **SLI**, **latency**, **error rate**, **business impact**
Incident Management	Reactive resolution	**Proactive prevention**, blameless postmortem, error budget-based decisions
Change Management	Manual approvals, CAB meetings	**Automated** CI/CD-based GitOps, canary, blue/green **deployments**
Troubleshooting	Manual logging, ticket-based	**Automated** root-cause analysis, observability, distributed tracing
Capacity Planning	Static provisioning	**Auto-scalin**g, performance benchmarking, chaos engineering
Toil Handling	Manual repetitive tasks	**Automation**-first approach (Terraform, Ansible, scripts, AI Ops)
Reliability Engineering	NA (mostly infrastructure focus)	**Error budgets**, **SLO/SLA**, reliability testing, chaos engineering

56

SRE Team

- An **SRE (Site Reliability Engineering) team** is a specialized group of engineers who focus on **making systems reliable, scalable, efficient, and resilient** — using **software engineering, automation, observability, and proactive problem-solving** rather than manual operations.
- **Types of SRE Engineers**:

SRE Role	What Work They Perform (Responsibilities & Tasks)	Tools & Technologies
Infrastructure SRE	**Manages cloud and hybrid infrastructure**, Kubernetes, IaC, network, provisioning, reliability engineering, cost optimization	AWS, Azure, GCP, Kubernetes, Terraform, Ansible, Helm, VMware, OpenStack, Ansible, Bash/PowerShell, Nagios, SolarWinds, Zabbix, NetApp
Platform SRE	**Builds internal developer platforms**, CI/CD, Observability, Secrets, Service Catalog, GitOps, SSO, API gateway	GitHub Actions, Jenkins, ArgoCD, Vault, ServiceNow, Grafana, ELK
Product/ Application SRE	**Ensures reliability of user-facing apps** (payment, mobile banking, e-commerce, trading), sets SLO/SLIs, monitors production health, chaos testing, RCA	Datadog, Splunk, Jaeger, PagerDuty, New Relic, Chaos Monkey
Security SRE (Sec-SRE)	**Threat monitoring**, **vulnerability management**, automated patching, compliance, IAM, DDoS protection, security event automation	Splunk SIEM, Palo Alto, Cloudflare, Qualys, AWS GuardDuty
ML/AI SRE (MLOps)	**Monitor reliability** of **ML models**, automate retraining, manage data pipelines, ensure model accuracy & performance	Kubeflow, MLflow, TensorFlow, SageMaker, Kafka
Database SRE	**DB availability**, replication, backup, auto-recovery, sharding, query tuning, HA clusters, DR testing	Oracle RAC, PostgreSQL, Cassandra, MySQL, MongoDB, AWS RDS

AHMAD

57

How SRE is Structured in Organizations

- **3 Major SRE Structures**:

SRE Structure Type	Description	Company Examples	When to Use
Embedded SRE	**SRE engineers** directly **work** inside **product teams**	Google Search, Amazon Prime	Complex large-scale products needing continuous reliability
Central SRE / Shared SRE Team	**Single SRE team** supports multiple applications	Banking, telecom, SaaS	Medium-scale apps, central governance
Platform SRE	**Works on common tools**, CI/CD, infrastructure, observability, SLO frameworks	Netflix, Spotify, Salesforce	When building developer experience or DevOps platform

AHMAD

58

What Skills Do SRE Engineers Need

Skill Category	What SRE Should Know	Example
Software Development	Python, Go, Shell scripting, Git	Build automation, monitoring scripts
Cloud & Infra	AWS, Azure, GCP, Kubernetes, Terraform	Deploy Google Kubernetes, scale AWS infra
Automation & DevOps	CI/CD (Jenkins, GitHub), Ansible, GitOps, IaC	Automated deployment
Observability & Monitoring	Prometheus, Grafana, ELK, Splunk, Datadog	Track latency, error rates, business KPIs
Reliability Concepts	SLO, SLI, SLA, Error Budget, Blameless Postmortem	Decide when to stop deploying
Chaos Engineering	Gremlin, Chaos Monkey	Test failure recovery
Incident & Problem Solving	Root Cause Analysis, self-healing	Analyze high CPU issue, create fix
Performance & Scalability	Load testing, auto-scaling	Ensure 1M users during sale
Soft Skills	Collaboration, communication, problem-solving	Talk to developers and business

AHMAD

59

Roadmap to Convert Traditional IT Ops to SRE

Phase	Goal	Activities
Assessment	Identify skill gaps	Evaluate current Ops skills, mindset, tooling
Training	Build new capabilities	Train on Python, Cloud, CI/CD, Monitoring, SRE fundamentals
Automation First	Reduce toil	Transition from manual tickets to automation & scripts
Introduce SLO / SLA / Error Budgets	Reliability focus	Implement reliability metrics, observability dashboards
Create SRE Squads	Start hybrid model	Mix Dev + Ops + SRE together (Platform, Infra, Product SRE)
Full SRE Adoption	Establish culture	Blameless postmortems, continuous improvement, chaos testing

AHMAD

60

Ops to SRE Transformation

Before SRE (Ops Team)	After SRE (Reliability Engineering Team)
Manual monitoring of servers	Automated trade monitoring using latency, TPS, failure rate
Ticket raised for payment failure	Script auto-restarts payment service (self-healing)
Outage analysis done manually	SRE uses distributed tracing (Jaeger) to find bottleneck
Change approval in email	GitOps-based approval and automated deployment
Manual scaling during "Salary Day"	Auto-scale compute based on trade volume

AHMAD

61

Case Study

AHMAD

62

Employee Identity Service (EIS)

- Your organization (**FinTech Corp**) has an **Employee Identity Service** **(EIS)** responsible for:
  - Employee **authentication** (login)
  - **Authorization** (role-based access control)
  - **Profile retrieval** (name, designation, department)
- This service is **mission-critical**, powering:
  - Payroll & HR Portal → Salary slips, leave & appraisal
  - Finance Approval System → Expense approvals
  - IT Service Desk → Asset provisioning & access requests
  - Onboarding Platform → New employee setup
- **Business Expectation**
  - Employees should be able to access portals securely, quickly, and reliably without login issues, especially during salary processing, appraisal cycles, and onboarding.

AHMAD

63

Step 1: Define Possible SLIs

SLI Type	Example Metric	Description
Availability		
Latency		
Correctness		
Error Rate		
Freshness		

AHMAD

64

Possible SLIs

SLI Type	Example Metric	Description
Availability	**% of successful login requests**	Measures uptime/access success
Latency	**% login responses < 500ms**	Measures login response speed
Correctness	**% correct identity responses**	Ensures data accuracy (role, name)
Error Rate	**% failed login attempts** (5xx)	Tracks failures due to service issues
Freshness	**Time since last user update synced**	Measures how current data is

AHMAD

65

Step 2: Define Possible SLOs

SLI	SLO (Target)	Error Budget
Availability		
Authentication Latency		
Error Rate (5xx)		
Correctness		
Freshness		

AHMAD

66

Possible SLOs

SLI	SLO (Target)	Error Budget
Availability	**99.5% monthly**	**0.5%** downtime = **216 mins/month**
Authentication Latency	**95% requests < 400ms**	**5% slow requests allowed**
Error Rate (5xx)	**<0.3% per month**	**0.3% failure margin**
Correctness	**99.9% accurate identity responses**	**0.1% incorrect allowed**
Freshness	**95% of syncs within 1 hour**	**5% can exceed 1hr**

AHMAD

67

Step 3: Define Possible SLA

SLA Parameter	SLA Commitment
Availability	
Login Response Time	
Identity Data Sync	
Incident Response	
Compensation	

AHMAD

68

Possible SLA

SLA Parameter	SLA Commitment
Availability	**99.0% quarterly** (*max downtime: 1h 48m per month*)
Login Response Time	**90% of login responses < 500ms**
Identity Data Sync	**Max sync delay = 4 hours**
Incident Response	Critical **incidents resolved** within **4 hours**
Compensation	Service credits: e.g., 5**% monthly subscription refun**d for downtime exceeding SLA

AHMAD

69

Discussion

- **Why SLO is stricter than SLA**
- **What happens if SLA is breached externally**
- **Error Budget vs SLA**

AHMAD

70

Discussion | Answer

- **Why SLO is stricter than SLA**
  - Internal target (SLO) is higher to **ensure service reliability**
  - SLA is slightly lower to **protect contractual commitments** and allow operational tolerance
- **What happens if SLA is breached externally**
  - Financial **service credits** (refund, subscription adjustment)
  - **Priority support / escalation** for affected customers
  - Reputation & trust may be impacted → triggers postmortem and reliability improvements
- **Error Budget vs SLA**
  - **SLA**: **99.9%** uptime
  - **Error budget**: **0.1% allowed downtime** per month = 43 minutes of downtime allowed per month

AHMAD

71

72

National Stock Exchange

Traditional Ops Environment	Modern SRE Transformation
Manual monitoring via NOC	**Full observability** (*Grafana, Prometheus, AI alerting*)
1–2 hours downtime during trading time	**Self-healing** failover in **2 minutes**
1000+ monthly tickets	**80% tickets eliminated** via automation
Change freeze during market hours	Safe, **automated deployments** using canary
Problem Resolution = War-room	**Resolution via Runbooks**, automated fixes

AHMAD

73

Why Change Was Needed

- One **trading outage** caused a **₹100 Crore loss** and **reputation damage**.
- Analysts were **fixing** production **incidents manually**.
- **Deployments** were **done via scripts**, after midnight, with **risk and fear**.
- **Monitoring** was **limited** to **CPU & RAM** — no business metrics.

AHMAD

74

How They Introduced SRE

Phase	Step	SRE Activities
Phase 1	**Analyze**	• **Define SLOs** (e.g., 99.99% uptime) • Identify **most frequent failures**
Phase 2	**Automate**	• Incident detection → automated alerts • **Self-healing scripts** → **restart** crashed **services**
Phase 3	**Reduce Toil**	• **70% tickets** converted to **automated jobs** Password resets, patching, backup monitoring
Phase 4	**Build Reliability Culture**	• Introduced **blameless postmortems** • Used **error budget policy** for release decisions
Phase 5	**Optimize**	• Chaos testing, load simulators, active failover

75

Results of the SRE Transformation

Before SRE	After SRE
4 major outages/year	**0** major outages in 12 months
Recovery Time: **1–2 hours**	Self-healing in under **3 minutes**
1000+ tickets/month	**80% reduction** (automated resolution)
6–8 people required for manual patching	Fully **automated** patch pipeline
No visibility into root cause	**Blameless** postmortems & **RCA automation**

76

Final Summary

Traditional Ops	SRE Mindset
Block changes to maintain stability	Engineer systems to **handle change safely**
Reactive firefighting	**Proactive** & **predictive** reliability
Manual troubleshooting	**Automation** & **self-healing**
Ticket-based operations	**Engineering-based** operations
Support function	Strategic **reliability engineering**

AHMAD

77

Q & A

Any concepts still unclear?

Thank you for attending

AHMAD

78