
Traditional IT Ops to
SRE Mindset
AHMAD

1


Introduction & Evolution
AHMAD

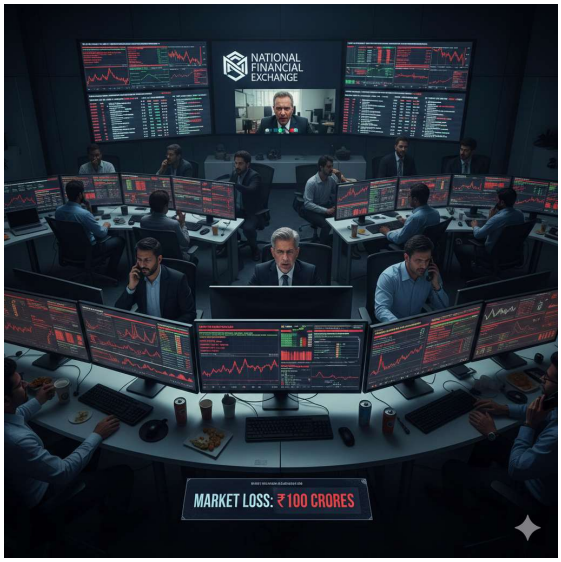
2

Real-World Story

3

The Night of the Big Incident

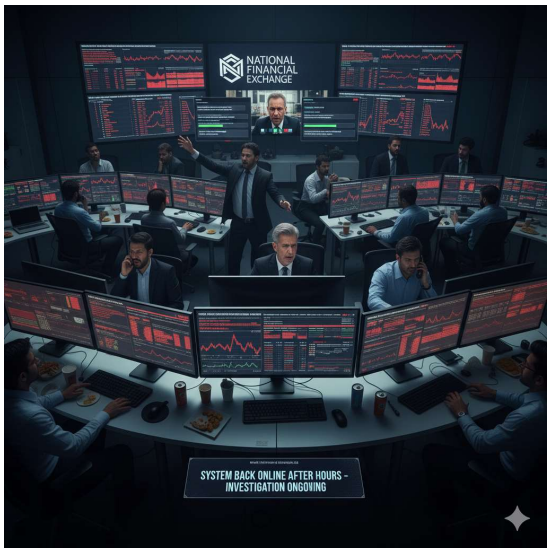
- It was **11:50 PM on a Monday**.
- The stock trading application for **National Financial Exchange** had just gone into production with a **minor configuration change** — **approved** through **Change Management**, documented in ITIL, **tested** in **UAT**.
- **Everything looked fine**.
- But at **9:15 AM next morning — right at market opening — the system slowed down.**
- **Within minutes**:
  - Trade orders started failing
  - Brokers couldn't log in
  - Transactions got stuck in queue
  - Twitter exploded: "Exchange Services Down!"
  - News channels picked it up.
  - In just 30 minutes — **market loss ₹100 Crores**.



4

Ops Team — Fix

- The **Ops team rushed to fix it**.
  - They **checked logs**, **restarted servers**, **increased CPU**.
  - Senior management called every 10 minutes.
  - **Incident reports** were **created**.
  - **Root Cause Analysis meetings** were **scheduled**.
- Finally, after **2 hours**, the **system was back**.
- But the **damage was already done**.



AHMAD

5

A Week Later — The Real Problem Emerged

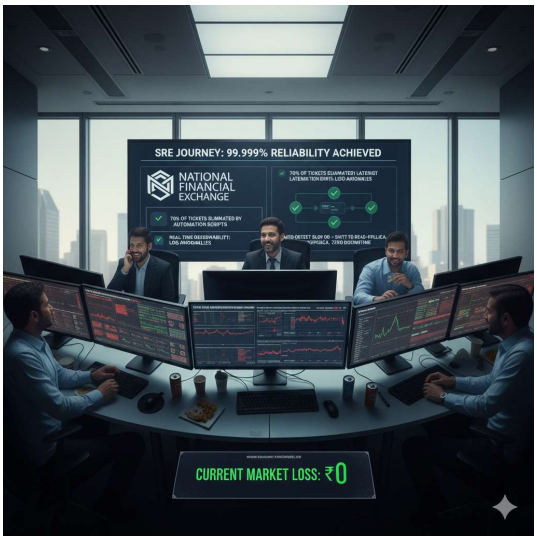
- In a high-level review meeting, the CIO didn't ask, *"Who made the mistake?"*.
- He asked:
  - "Why are we still **depending** on people to **manually rescue systems**?"
  - "Why don't our **systems heal themselves**?"
  - "Why can't we **predict failures** instead of reacting?"
  - "Why do we find problems **after users complain**?"
- There was silence.
- Then, one Ops lead said something that changed everything: "**We have been doing Operations but not Engineering**."

AHMAD

6

Beginning of the SRE Journey

- They hired **reliability engineers**.
- They **automated recovery**.
- They **eliminated 70% tickets** using **scripts**.
- They introduced **real-time observability** — not just CPU/Memory — but latency, transaction drops, log anomalies.
- Now, if the same issue happens?
  - The **system auto-detects** slow database connections
  - It **automatically shifts** to read-replica
  - No downtime.
  - No Twitter complaints.
  - No financial loss.

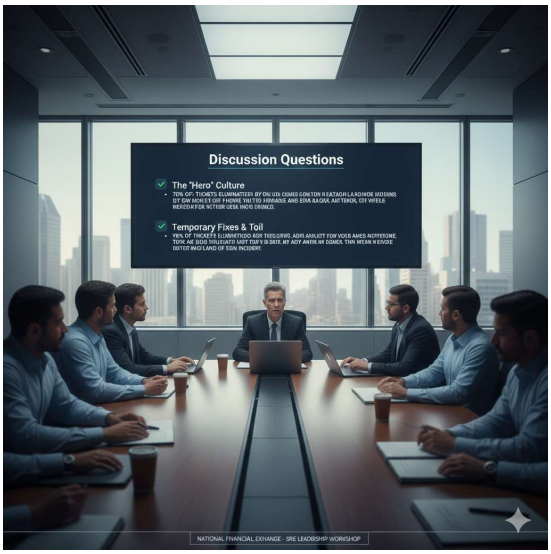


AHMAD

7

Discussion Questions

- **Question**
  - In the story, Person X was praised as a "**hero**" for fixing the incident manually.
  - In your experience, **what are the risks or hidden costs** associated with a team constantly relying on "heroes" for production stability?
- **Question**
  - Can you recall a specific instance where your team's expertise was spent on a fix that you knew—even as you were doing it—was only **temporary**?
  - What was the **repetitive task** you wished you could automate?
- **Question**
  - In the story, incident took 45 minutes to resolve.
  - Where was most of that time spent: **Alerting, Diagnosis, or Mitigation**?



AHMAD

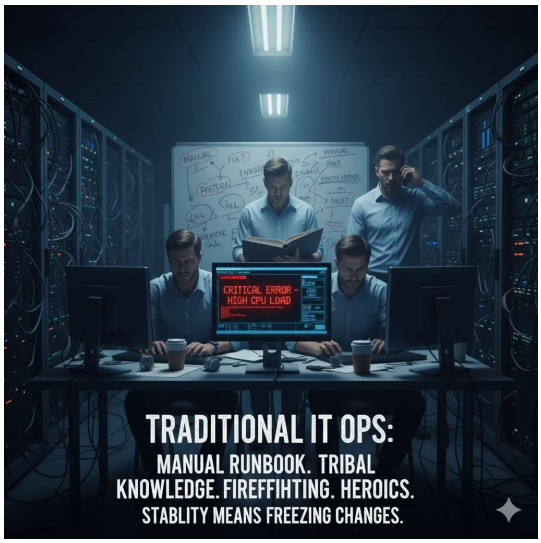
8

IT Operations

9

What is IT Operations

- IT Operations **refers** to **all the activities** required to **keep** an organization's **IT infrastructure** and **applications running smoothly**, **securely**, and **efficiently**.
- It involves **managing hardware**, **servers**, **networks**, **databases**, **security**, **backups**, **help desk**, **monitoring**, **incident handling**, and **compliance**.
- **ITOps focuses** on the **day-to-day execution** of **tasks** necessary to deliver and support IT services to both **internal users** (employees) and **external customers**



10

IT Operations … continue

- **Traditional IT Operations** refers to the **legacy approach of managing IT systems**, where teams focus on:
  - **Maintaining** servers, networks, storage
  - **Handling** incidents through **tickets**
  - **Monitoring** systems manually
  - **Managing** data centers and on-prem infrastructure
  - **Following** ITIL-based processes like Change, Incident, and Problem Management
- **Goal in Traditional Ops:**
  - **Keep systems running**, stable, and secure — but **using manual processes** and **operational control**.
- The model is characterized by a deep **siloing** of responsibilities:
  - **Development (Dev)**
    - Writes code and focuses on new features.
  - **Operations (Ops)**
    - Manages the running system, infrastructure, security, and deployment.
  - **Handoffs**
    - Code is completed by Dev and then "thrown over the wall" to Ops for deployment and maintenance.

AHMAD

11

Key Responsibilities of IT Operations

Area	Description
Infrastructure Management	Setup & manage servers, networks, storage
Monitoring & Maintenance	Track performance, uptime, logs, health
Deployment & Change Control	Deploy software with approvals
Incident & Problem Management	Respond to system failures and perform root cause analysis
Security & Compliance	Protect from threats, control access, audit
Backup & Disaster Recovery	Data backup, replication, recovery plans
Help Desk / Service Desk	User support, ticket resolution

AHMAD

12

Types of ITOps Teams and Function

Type of Team/Function	Core Focus Area	Mindset & Goal
Service Desk/ Help Desk	**First-line support** and incident logging; handling employee issues (password resets, device failures).	**Customer Satisfaction:** Act as the single point of contact (SPOC) and resolve issues quickly.
Infrastructure Management	**Physical and virtual hardware** (servers, storage, data centers, cloud resources), operating systems, and network components.	**Availability:** Ensure the underlying platforms are always running and scalable.
Network Operations Center (NOC)	**Real-time monitoring** of the network, applications, and services; identifying and triaging high-priority incidents.	**Proactive Alerting:** Minimize downtime by detecting and escalating problems before they cause outages.
Security Operations (SecOps)	Implementing security controls, **monitoring for threats**, managing firewalls, and responding to security incidents.	**Data Protection:** Maintain system integrity and compliance with security policies.
Data & Storage Operations	**Backups**, disaster recovery, data retention, and ensuring the performance and availability of database servers.	**Resiliency:** Guarantee data integrity and the ability to recover service after a catastrophic failure.
Application Operations	Managing the health, **deployment, and performance tuning** of specific business applications once they are in production.	**Performance:** Ensure critical business applications run smoothly for end-users.[4]

AHMAD

13

Approach and Processes

- The Traditional Ops approach is characterized by a "**throw it over the wall**" handoff and a focus on minimizing change to maximize stability.
  - **The Handoff ("Throw It Over the Wall")**
    - Once the **Development** (*Dev*) team **completes** the **code** and documentation, it is **handed over** to the **Operations** (*Ops*) team for deployment and ongoing maintenance.
    - This **often leads** to the infamous "**it works on my machine**" problem.
  - **Change Management Process**
    - **Changes** (*code deployments, configuration updates*) are viewed as the **primary source** of **risk**.
    - They are **managed** through slow, multi-layered approval processes, often involving a **Change Advisory Board** (*CAB*).
    - The **goal** is to **restrict** and minimize the **frequency** of **changes**.
  - **Reactionary Incident Response**
    - **Incidents** are typically **resolved** by engineers manually **logging into systems**, diagnosing the failure, and applying a fix.
    - The **primary focus** is on **restoring service immediately**, often leaving the underlying cause unaddressed or documented only in a tribal knowledge format.
  - **Monitoring Focus**
    - **Monitoring** often focuses on basic **system health metrics** (*CPU, memory, disk usage*) rather than the application's actual business service health.
    - **Alerts** can be numerous and **noisy**, leading to **alert fatigue**.

AHMAD

14

ITOps Workflow

- **Scenario: Deploying a New Application**
  1. **Development**
     - The Dev team finishes the application code and hands off the build files and a long installation manual (the **handoff**).
  2. **Request**
     - The Ops manager receives a **Change Request (CR)** ticket to deploy the application.
  3. **Approval**
     - The CR goes to the **Change Advisory Board (CAB)**, which meets once a week.
     - The CR is scrutinized for potential impact and approved after three days.
  4. **Deployment (The Toil)**
     - An **Ops engineer manually** follows the installation manual.
     - They **log** into the **production** server via SSH, **manually configure** the web server, **install** dependencies, copy the files, and **restart** the service.
     - This is often **done during** a **scheduled maintenance window** at 2 AM.
  5. **Monitoring**
     - The Ops team sets up basic monitoring to check if the server is alive (ping) and the CPU/memory usage, satisfying the SLA.

- This model prioritizes **control and stability** over **speed and collaboration**, which is the primary driver for the evolution toward modern practices like DevOps and SRE.

15

ITOps | Benefits

- The **focus** on **control** and **process** was highly effective for the pre-internet era **when systems** were **static** and **changes** were **infrequent**.

  - **Stability & Control**
    - **Strict**, formal **processes** (*like the Change Advisory Board, or CAB*) **minimize changes**, leading to high stability for static, mission-critical systems.

  - **Clear Accountability**
    - **Roles** are **strictly defined** (*Network Admin*, *Database Admin*, *Sysadmin*), ensuring **clear ownership** and accountability for specific infrastructure components.

  - **Security & Compliance**
    - The separation of duties and rigid, documented procedures make it easier to meet **stringent regulatory requirements**, especially in finance or healthcare.

  - **Cost Predictability**
    - Operating on owned, on-premise hardware allows for **predictable**, **long-term Capital Expenditure** (*CapEx*) budgeting, rather than variable monthly cloud costs.

16

ITOps | Drawbacks

- The reliance on manual effort and **rigid processes creates friction** and **bottlenecks** in the modern business environment.
  - **Slow-Release Cycles**
    - The sequential **Waterfall model** and formal **change management processes** (*CAB meetings*) mean **releases** are **infrequent**, **large**, and **high-risk** ("*big bang" deployments*).
  - **Adversarial Culture (Dev vs. Ops)**
    - The **siloed structure creates conflict**.
    - **Developers prioritize speed**; **Operators** prioritize **stability**.
    - This results in mutual frustration and a **lack of shared ownership**.
  - **High Operational Toil**
    - **Engineers spend** a vast amount of time on **repetitive**, **manual**, and **error-prone tasks** (*like manual patching, server configuration, or log checks*).
    - This **toil prevents** them from focusing on strategic improvements.
  - **Poor Incident Response**
    - **Troubleshooting** often involves **manually logging** into servers and **following runbooks**, leading to long **Mean Time To Resolution (MTTR)** and costly downtime.

17

Why Traditional ITOps Fails in New IT

Traditional IT Ops	Modern Digital IT	Why Alignment Fails
Manual, Ticket-Based	**Automated, Self-Healing**	• **Manual ticket** handling is **slow**, **error-prone**, and **dependent** on **human** availability. • **Modern systems** need **real-time automated responses**, **self-healing** scripts, and **zero-touch resolutions** to maintain uptime.
Reactive Issue Handling	**Predictive and Preventive Monitoring**	• **Traditional** ops **waits** for **issues** to **happen** (***reactive***). • **Modern** systems **predict failures** using metrics, anomaly detection, and AI — **fixing issues before** customers are **impacted**.
On-Prem Infrastructure	**Cloud, Containers, Serverless**	• **Static**, hardware-heavy infrastructure **cannot support rapid scaling**, **multi-region deployment**, or **cloud-native architectures** required by digital platforms.
Monthly Deployments	**Multiple Deployments per Day**	• **Legacy** IT **relies** on **approval boards** and **long change cycles**. • **Digital** businesses (Amazon, Netflix) **deploy continuously** using **CI/CD**, **feature flags**, and **automated testing**.
Static Capacity Planning	**Dynamic Auto-Scaling**	• **Traditional** IT **purchases capacity** for peak load, leading to **high cost** and **underutilization**. • **Digital** IT **auto-scales instantly** based on real-time demand in cloud environments.
Human Monitoring	**AI/Analytics-based Observability**	• **Manual monitoring** (*logs, dashboards*) misses hidden performance issues. • **Modern monitoring** uses **AI**, distributed **tracing**, and **real-time analytics** to provide deep visibility.

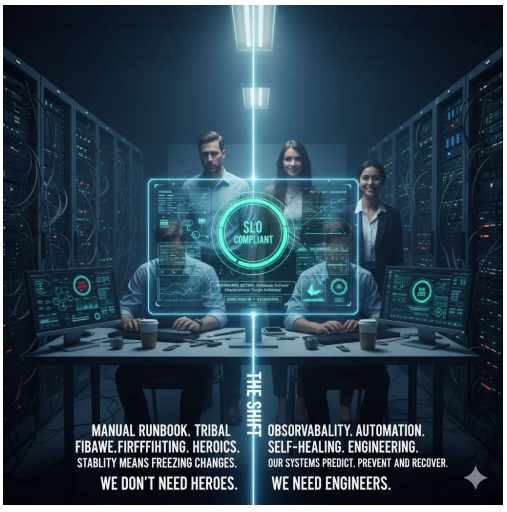
18

Evolution of Reliability Engineering

19

What is Site Reliability Engineering

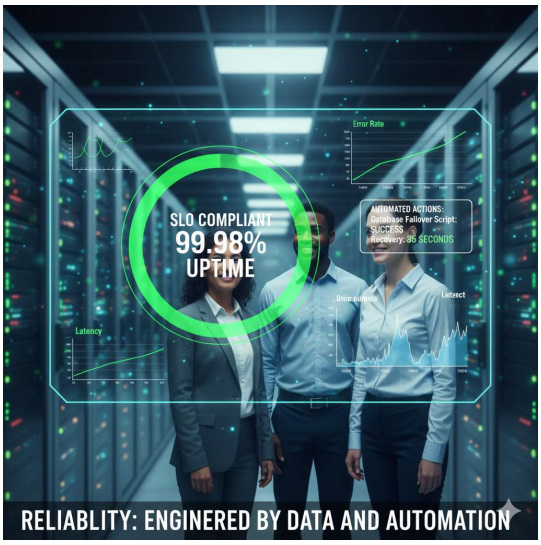
- **SRE (Site Reliability Engineering)** is an engineering approach introduced by **Google** to **ensure highly reliable, scalable, and efficient software systems** using **software engineering practices applied to operations**.
- The concept was pioneered at **Google in 2003** by Ben, who defined SRE as "**what happens when you ask software engineers to design an operations function**"
- It combines **software development + IT operations**, with a strong focus on **automation, reliability, performance, monitoring, and incident response**.
- **"Use software engineering to solve operational problems."** Instead of managing operations manually (like traditional Ops), SRE automates them through code.



20

What is Reliability

- Reliability in Site Reliability Engineering (SRE) is the **single most important metric** and the **core focus** of the entire discipline.
- In simple terms, SRE views reliability as **how often and how well a service works as expected by its users**.
- It's **not just** about the **server being "up,"** but about the **user** being **able** to **successfully complete** their **task** (*e.g., loading a webpage, sending a message, completing a purchase*) quickly and without errors.



AHMAD

21

Reliability in Traditional IT vs SRE

Aspect	Traditional IT (Ops)	SRE Approach
Definition	Keep the server running	**Ensure quality experience** for users
Focus	System uptime	**Overall user happiness** (availability, speed, errors, security)
Measurement	Uptime (%) only	**SLO**, **SLI**, **error rate**, **latency**, **scalability**, **resilience**
Approach	Manual fixes, reactive	**Automated**, data-driven, **proactive**
Mindset	"Keep lights on"	"**Engineer reliability** like a **product feature**"

AHMAD

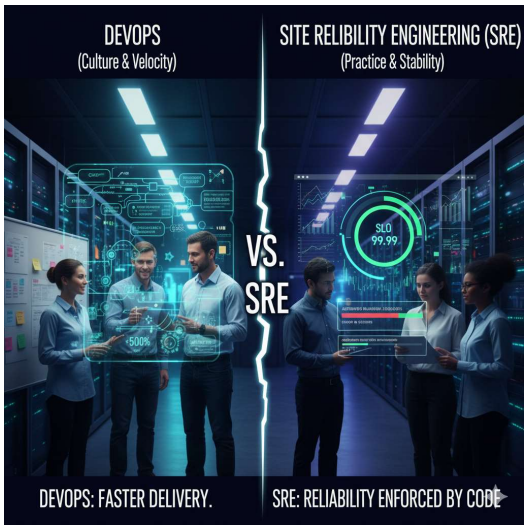
22

DevOps vs SRE

23

DevOps vs SRE

- While **DevOps** is a **philosophy** that **drives cultural change**, **SRE** is a prescriptive **practice** that defines **how to implement** the **reliability** and **automation** aspects of DevOps using software engineering methods.
- **Both** DevOps and SRE **improve software delivery** and **operations**.
- But **they solve different problems**, **use different methods**, and **have different goals**.
- **DevOps emphasizes** the **pipeline** (CI/CD), and **SRE emphasizes** production **reliability**. *Both complement each other* in modern organizations.



24

DevOps vs SRE | Key Differences

Aspect	DevOps (Philosophy/Culture)	SRE (Implementation/Role)
Goal	• Increase the **organization's ability** to **deliver applications** and services at **high velocity**	• Define, achieve, and maintain specific levels of **service reliability** using **quantifiable metrics**
Focus	• **Collaboration** and **Speed**. • **Breaking down** organizational **silos** between **Development** and **Operations**	• **Reliability** and **Automation**. • **Applying** software **engineering** to **solve operations problems**
Who does the work?	• **Dev + Ops teams together**	• Specialized **reliability engineers**
Philosophy	• "Developers and operations should collaborate"	• "Ops is a software engineering problem"
Metrics Focus	• **Deployment frequency**, lead time, and mean time to recovery (MTTR)	• **Service Level Objectives (SLOs)**, Error Budget consumption, and percentage of time spent on toil

AHMAD

25

DevOps vs SRE | Mapping

Statement	DevOps	SRE
How can we **release faster**?	Yes	Yes
How can **we stay reliable** during fast releases?	**Not fully defined**	**Core mission**
Is it a **culture**?	Yes	No — it's a role/discipline
Is it a **team**?	No (Dev + Ops collaboration)	Yes — SRE team exists
Do they **write automation code**?	Sometimes	Always

AHMAD

26

Pre-SRE Approaches

AHMAD

27

28

Pre-SRE Approaches

29

Pre-SRE Approaches

- Before Site Reliability Engineering (SRE) emerged, organizations primarily used **Traditional IT Operations (IT Ops)** and **ITIL-based management** to run systems, maintain availability, and manage incidents.
- **These approaches** were **process-heavy**, **manual**, and **reactive**.

PRE-SRE APPROACHES: MANUAL CONTROL, SLOW CHANGES, AND THE AGE OF HEROICS

30

ITOps and System Administration

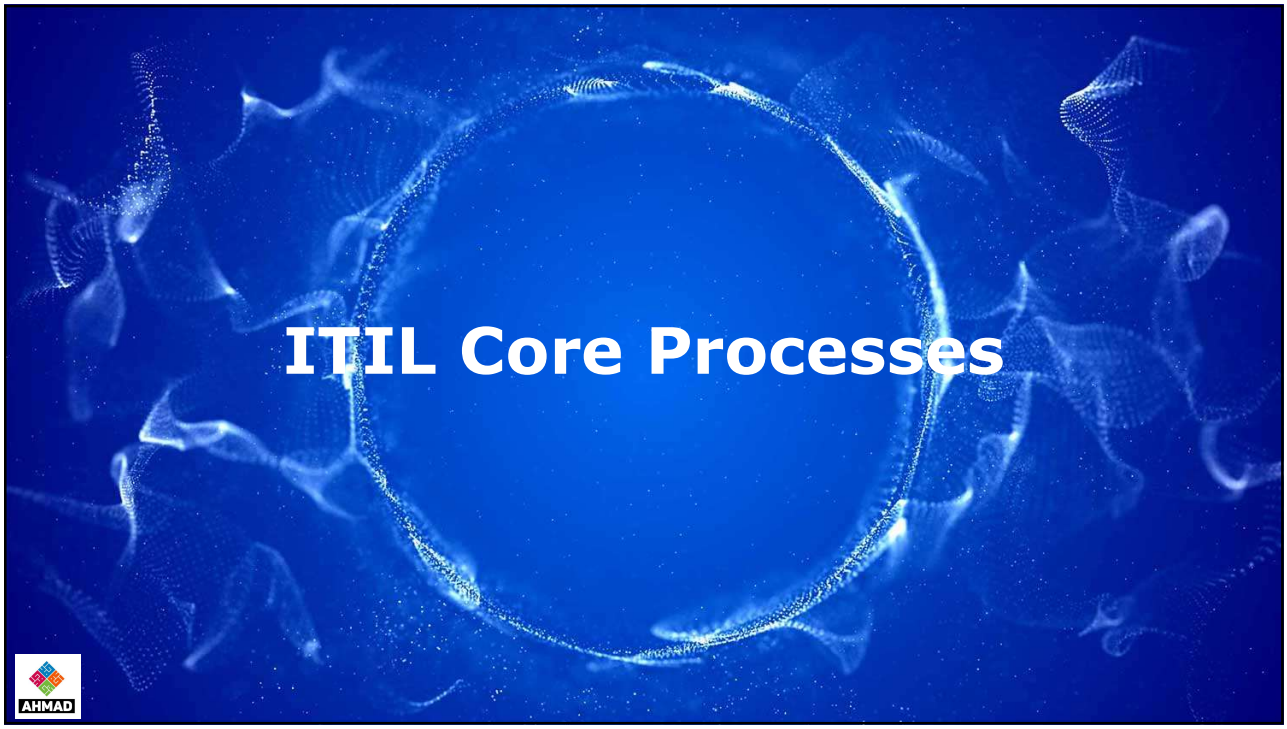
- **Traditional IT Ops** and **Sysadmin** were the **primary roles** responsible for **maintaining** production **systems**.
- **Focus**: The **core focus** was maintaining **stability** through manual control and **physical infrastructure management**.
- **Mindset**
  - The **environment** was **seen** as **static**. **Change** was **inherently risky**, and the **Sysadmin's** job was to be the **gatekeeper against failure**.
- **Key Activities**
  - **Server Maintenance**
    - Physically **racking** and **cabling** servers, **installing** and **patching** operating systems, and managing virtual machines **manually**.
  - **Network/Storage Management**
    - **Configuring** network **devices**, managing **firewalls**, and provisioning **storage arrays**.
  - **Scripting**
    - Using **basic shell scripts** (*Bash, PowerShell*) to **automate repetitive t**asks **but** primarily **focusing** on **execution** rather **than engineering**.

31

ITOps and Sysadmin … continue

- **Characteristics:**
  - Approach → Manual, Reactive
  - Focus → Hardware, Servers, Uptime
  - Work Type → Tickets, Procedures, Checklists
  - Automation → Very limited
  - Scaling → Hard to scale
- **Example:**
  - An **e-commerce server goes down** during a sale.
  - **Admins manually log** in, **restart services**, **check logs**.
  - **Root cause analysis is** done **later**.
  - **No automated recovery** or self-healing.

32



33

ITIL-Based Management vs SRE Philosophy

- **ITIL** (Information Technology Infrastructure Library) is a comprehensive **framework** detailing **best practices** for **IT Service Management** (ITSM).
- It became the **dominant management structure** for **large**, **stable enterprises** in the **pre-SRE era**.
- Understanding **Mindset Difference**:

Mindset	ITIL	SRE
"How do we avoid failure completely?"	**Yes**	Not realistic; systems WILL fail
"How do we recover quickly with minimum impact?"	Not Primary focus	**CORE IDEA** of SRE

- **Key SRE Differentiator**
  - **SRE believes failure** is **normal** and focuses on **making systems resilient** to failure using automation, monitoring, and engineering practices.

AHMAD

34

ITIL-Based Management vs SRE Philosophy

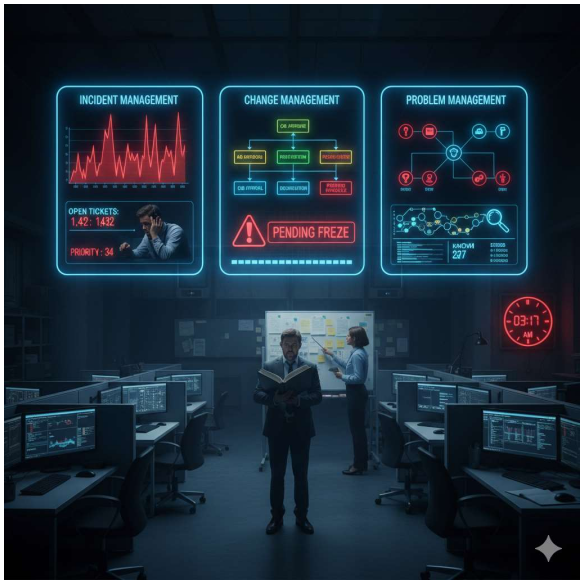
Feature	ITIL-based Management (Traditional)	SRE Philosophy (Modern)
Core Goal	• **Control** and **Process Adherence**. • Focus on **documented steps** to maintain service quality.	• **Reliability** and **Automation**. • **Focus** on achieving measurable outcomes (**SLOs**) through engineering.
View of Manual Work	• **Processes are mandatory**. • **Manual execution** of **change** forms and **approvals** is a **required** control point.	• **Manual work is Toil that must be eliminated.** • If a **process is repeated**, it **must** be **automated** by code.
Approach to Risk	• **Risk Aversion**. • **Minimize change** to avoid incidents, leading to **slow-release cycles**.	• **Risk Management.** • Define **downtime allowance** and use data to **balance agility** (*speed*) **against reliability** (*stability*).

AHMAD

35

ITIL Core Processes

- In a traditional, pre-SRE environment, the **three core ITIL processes** were handled with heavy human involvement:
  - **Incident Management**
  - **Change Management**
  - **Problem Management**



AHMAD

36

ITIL Core Processes … continue

- **Incident Management**
  - Legacy Approach
    - Primarily **reactive** and **manual**.
    - **When** a **system failed**:
      - An a**lert** (often generic) would page an on-call engineer.
      - The engineer would **manually log into servers**, check logs, and try to diagnose the issue based on their experience and paper/digital runbooks.
      - Mitigation involved **manual steps** like restarting services or failing over to a standby system.
  - Result
    - **Long Mean Time To Resolution** (MTTR), as human intervention and diagnosis are the slowest parts of the process.

37

ITIL Core Processes … continue

- **Change Management**
  - Legacy Approach
    - ✓ Dominated by the **Change Advisory Board** (CAB).
    - ✓ A developer or operator wanting to deploy a change had to submit a detailed change request ticket.
    - ✓ The **CAB** (*a committee of senior stakeholders*) would **meet weekly** to manually **review** and **approve** or **deny** the change.
  - Result
    - ✓ **Releases were slow**, infrequent, and large ("big bang" deployments), which paradoxically made them high-risk, as one big change is harder to troubleshoot than many small ones.

38

ITIL Core Processes … continue

- **Problem Management**
  - Legacy Approach
    - **After** an **incident** was resolved, a formal **Root Cause Analysis** (*RCA*) document was created.
    - The focus was on **identifying** the **cause** and **proposing a solution** (*e.g., "The server needs more memory"*).
    - However, the **implementation** of that solution (*the actual fix*) often **got stuck** in the backlog or required another slow **CAB approval**.
  - Result
    - The **same problem** (and the resulting incident) often recurred because the operations team lacked the engineering time and authority to **implement permanent**, **automated fixes**, leading to cycles of recurring toil.

AHMAD

39

40

Core SRE Mindset

- SRE isn't just a role — it's a **mindset shift** in how organizations manage systems.
- It moves from **People + Manual Work** → to **Engineering + Automation**.
- The **core belief** is:
  - **Reliability** is a **feature**. It **must be designed**, **engineered**, **measured**, and **continuously improved — not just maintained** manually.
- At its heart, SRE is about **engineering reliability into systems** while still enabling innovation.
- The mindset is not "zero failure" but **controlled, measurable reliability** balanced with agility.

AHMAD

41

Reliability vs. Agility Trade-off

- In the modern IT world, companies want **rapid feature releases** (*agility*) and **high reliability** (*stability*) — but **doing both is difficult**.
- **Traditional Ops Mindset**
  - **Slow** and **safe change**
  - Stability achieved by **blocking frequent deployments**
  - **Focus on uptime**, not engineering
  - "Let's stop deployments to avoid failures."
- **SRE Mindset**
  - Reliability is not achieved by restricting changes — it is achieved by **engineering for resilience**
  - Use **automation, safe deployments**, **testing**, **observability**
  - Measure reliability scientifically
  - "We don't slow down releases; we make releases safer."

AHMAD

42

SRE in On-Prem and Hybrid Environments

- While **SRE principles** are **most famously** associated with **cloud-native** companies like Google, the **mindset** is **technology-agnostic** and **highly effective** in managing traditional **on-premise** and **hybrid** environments.
- The **fundamental** goal remains the **same**: **eliminate toil and manage risk through code**.

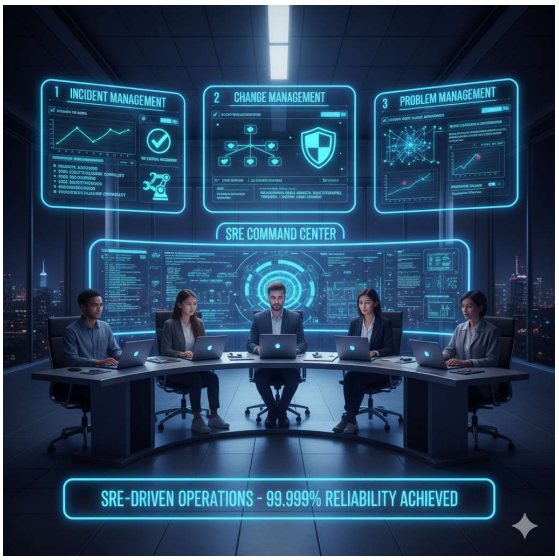
Principle	Cloud-Native SRE Approach	On-Prem/Hybrid SRE Approach
Infrastructure	Use **Cloud Provider APIs** to **provision resources** (*AWS CloudFormation, GCP Deployment Manager*).	Use **Infrastructure as Code (IaC)** tools like **Terraform** or **Ansible** to manage virtualization (VMware), network appliances, and bare-metal servers.
Self-Healing	Rely on **container orchestrators** (Kubernetes) and **cloud-managed** services for **auto-scaling** and **health checks**.	Build automation workflows using **runbook automation engines** or internal tools that script server restarts, failovers, and resource adjustments.
Challenge	The main **challenge** is **overcoming** the **manual control culture** ingrained in traditional data center management and securing the necessary permissions for automated tooling.	The main challenge is the inherent **difficulty of automation** in legacy systems that may lack modern APIs or rely on manual CLI interactions.

AHMAD

43

Management in the Post-SRE World

- In the SRE model, these traditional functions are transformed into engineering disciplines:
  - **Change Management** to **Change Engineering**
  - **Incident Management** to **Incident Response Engineering**
  - **Problem Management** to **Reliability Engineering**



AHMAD

44

Management in the Post-SRE World

- **Change Management → Change Engineering**
  - The primary goal of Change Management is still to ensure changes are deployed safely, but the mechanism is completely automated.
    - ✓ CAB Approvals → CI/CD automated approvals
    - ✓ Change Freeze → Feature flags & gradual rollouts
    - ✓ Night-time manual deployments → Auto deployments with rollback
    - ✓ Verbal approvals → GitOps-based approval control (*approvals are managed through Git repositories*)
- **Incident Management → Incident Response Engineering**
  - The SRE approach aims to make incidents rare, reduce human involvement in mitigation, and ensure recovery is instantaneous.
    - ✓ Manual escalation (phone/email) → Automated alerting
    - ✓ No defined roles → Incident Command System
    - ✓ Focus: restore service manually → Focus: auto-remediation with scripts
    - ✓ Retrospective blame → Blameless postmortems

AHMAD

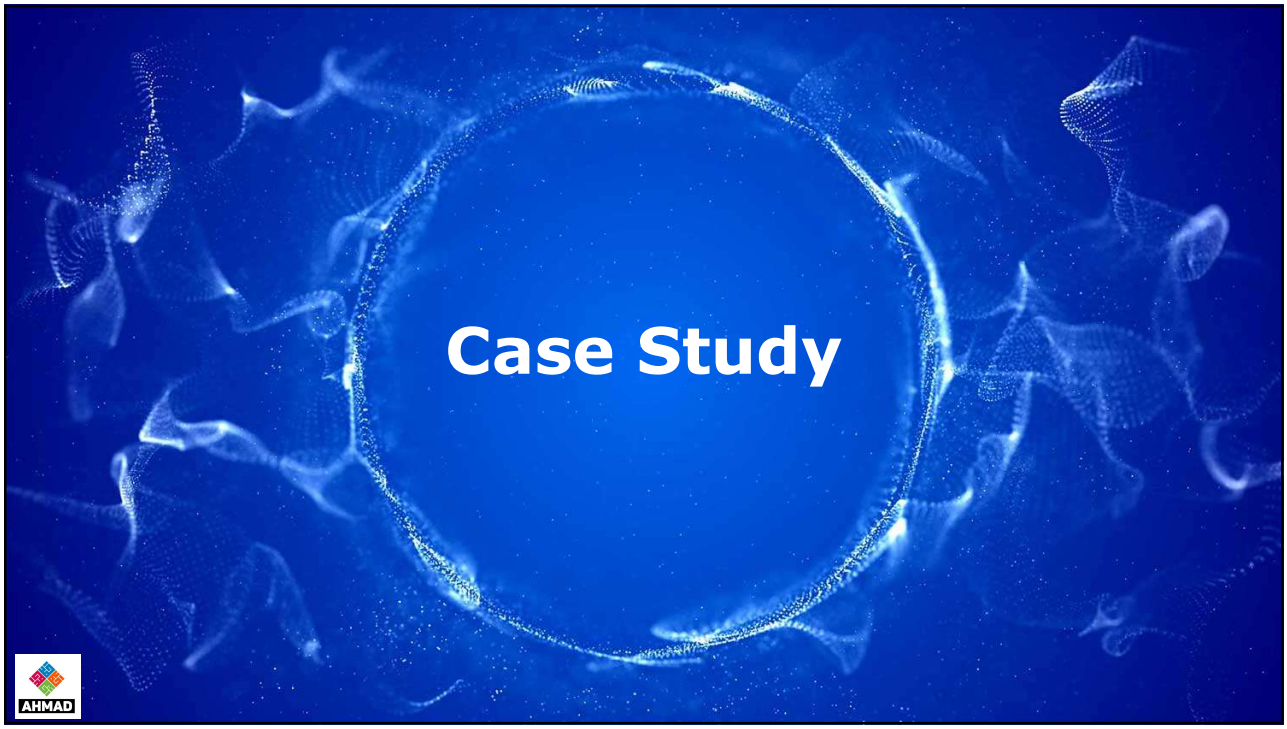
45

Management in the Post-SRE World

- **Problem Management → Reliability Engineering**
  - The goal shifts from merely finding the root cause to guaranteeing that the problem cannot recur through code changes.
    - ✓ Root Cause Analysis (who did it?) → Blameless Postmortems (why did system fail?)
    - ✓ Document fixes only → Implement resilience improvements
    - ✓ No testing for resilience → Use reliability testing tool
    - ✓ Manual restoration → Self-healing architecture

AHMAD

46



47

48

Stage

- Global **TradeX Stock Exchange** operated like most traditional financial institutions.
- It had a big data center, mainframe servers, and a Traditional IT Ops team responsible for monitoring systems, responding to tickets, and applying patches.
- Their operations followed:
  - Manual Monitoring: Teams watched dashboards 24/7 for alerts
  - Ticket-Based Support: Traders logged tickets when systems were slow
  - Monthly Scheduled Maintenance: Downtime allowed on weekends only
  - Change Approvals via CAB: Deployment approvals took 2-3 weeks
  - Fix after Failure: Teams waited for systems to crash before acting
- They believed: "Stability means freezing changes." But the world around them was changing.

49

The Day Everything Broke

- One Monday morning, during peak trading hours (*9:15 AM*), the **market order execution system slowed down**.
- Trades that should complete in **40 milliseconds** were taking **4 seconds**. In stock trading, **4 seconds is a lifetime** — people lost money.
- The helpdesk opened **hundreds of tickets**.
- The **Ops Team Response**:
  - They manually restarted servers
  - Increased memory on application nodes
  - Filed incident reports to multiple teams
- **Total resolution time**: **2 hours 20 minutes**
- **Result**
  - Millions lost in failed or delayed trades
  - Exchange reputation hit media and regulators
  - CEO called an emergency board meeting

50

Root Cause Analysis

- During the review, **key problems** surfaced:
  - **No early warning**: Manual monitoring didn't predict issue
  - **No automation**: Every fix required engineers and approvals
  - **No auto-scaling**: Load increased 300%, servers stayed same
  - **No real-time incident response**: Took 2 hours to coordinate teams
  - **Not enough observability**: Couldn't pinpoint the exact failure fast
- The **CIO said**: We **don't need** more **people watching screens**. We **need systems** that **heal themselves** before failing."

51

The Shift — From Ops to SRE

- They **introduced SRE** (Site Reliability Engineering) to transform operations:
  - **Ticket-based work → Automation-first (reduce tickets)**
    - SRE builds scripts & automation to **prevent issues before tickets are raised**
      - ✓ Auto-restart service, reallocate resources, auto-scale system etc.
  - **Stability via change blocking → Reliability through engineering**
    - Use engineering to **make changes safe**, **fast**, and reliable
      - ✓ Automated testing (CI/CD)
      - ✓ Canary Deployments
      - ✓ Blue-Green Releases
  - **Manual incident recovery → Self-healing systems**
    - Systems **auto-detect** and **auto-fix** failures without human involvement
      - ✓ High CPU usage → Auto-scale servers
      - ✓ Service crashes → Auto-restart container/VM
      - ✓ Region failure → Auto-failover to another region
      - ✓ Slow DB → Automatically add read replicas

52

The Shift — From Ops to SRE … continue

- **General IT operations** → **Software engineers specialized in reliability**
  - Software engineers **who write** code to **solve operational problems**
    - ✓ Write Python/Go scripts for automation
    - ✓ Build Monitoring & Alerting systems
    - ✓ Create tools to manage capacity, failures, scaling
    - ✓ Use chaos engineering to test system resilience
- **Human monitoring** → **AI-based proactive observability**
  - **AI-driven monitoring**, **anomaly detection**, **tracing**
    - Use Observability platforms (*Prometheus, DataDog, Grafana, ELK*)
    - AI/ML detects anomalies before failure
    - Full-stack visibility: logs, metrics, traces, user experience
    - Business monitoring (*trade volume, revenue impact, failed orders*)

AHMAD

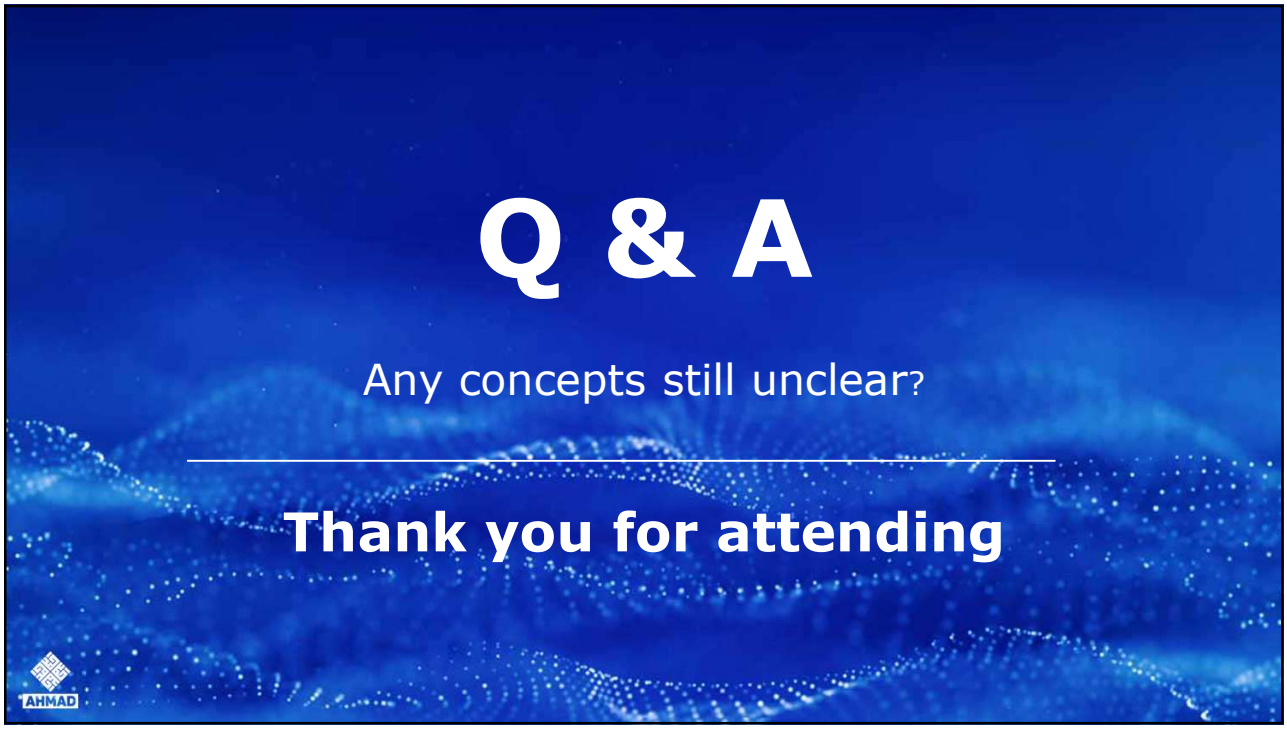
53

One Year Later: New World, New Results

Before (Traditional Ops)	After (SRE-based Ops)
4 major outages/year	0 major outages in last year
Incident Recovery Time: 2+ hours	Recovery Time: under 2 minutes (self-healing)
300+ monthly tickets	85% tickets eliminated via automation
Manual change approvals	Automated safe deployments (Canary, Blue/Green)
Monitoring only CPU & RAM	Deep Observability (Latency, traces, business metrics)

AHMAD

54


Q & A
Any concepts still unclear?
Thank you for attending
AHMAD

55