
Reducing Toil and Automation in On-Prem Environments
AHMAD

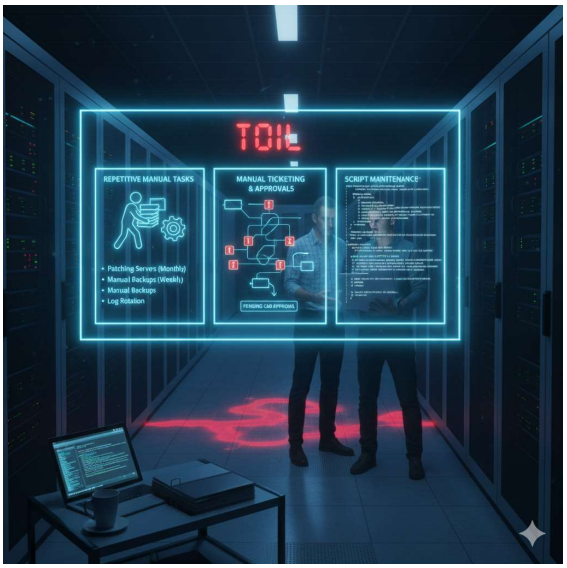
1


Identifying and Quantifying Toil
AHMAD

2

Identify Toil in On-Prem Environments

- Identifying toil in **on-premise environments** is the crucial first step toward automation.
- **Steps to identify:**
  - **Phase 1**: Define Toil by its Characteristics
  - **Phase 2**: Systematic Identification Methods



AHMAD

3

Phase 1: Define Toil by its Characteristics

- **Apply** the **six SRE characteristics** to every suspected operational task.
- If a **task matches** most of these points, it is **likely toil** that should be automated.

Characteristic	Question to Ask	On-Prem Toil Example
Manual	Does it **require human hands-on time**, even if it's just running a script?	Logging into 10 different servers to manually check log files.
Repetitive	Is it done **multiple times a day**, **week**, or **month** with little variation?	Daily runbook step to check the health of a database cluster.
Automatable	Could a machine or **script do this task** just as well as a human?	Applying a standard OS security patch to a server fleet.
Tactical	Is it **interrupt-driven** and **reactive** (a quick fix) rather than proactive (a strategic fix)?	Restarting a service because memory consumption spiked again.
No Enduring Value	Does the **system state remain unchanged** once the task is complete?	Clearing a log partition because it hit 95% full.
Scales O(n)	Does the **amount of work increase proportionally** as the **number of servers**, users, or applications grows?	Manually provisioning a new VM every time a developer requests one.

AHMAD

4

Phase 2: Systematic Identification Methods

- Since toil often **hides in plain sight**, you need **structured methods** to **capture** it.
- **Ticket and Incident Analysis**
  - **Review** your **existing ticketing sy**stem (*e.g., Jira, ServiceNow*) and incident management platform to **find high-frequency tasks**:
    - Filter by Frequency
      - **Sort tickets** by the **highest volume** or **recurring nature** over a quarter (*e.g., "Reset Firewall Rule," "Extend Volume Group," "Service Restart Request"*).
    - Analyze Resolution Time
      - Look for **tasks** with a **high aggregate time spent**, even if the individual task is quick (*e.g., 50 requests per month taking 10 minutes each = 500 minutes of toil*).
    - Categorize & Label
      - **Introduce** a **mandatory field** or tag in your ticketing system for **engineers** to **flag** work as "**Toil**" or "Project/Engineering."

AHMAD

5

Phase 2 … continue

- **Time Tracking and Audits (The Human Factor)**
  - The **engineers doing** the **work** are the **best source** of truth.
  - Toil Logging
    - Ask engineers to keep a simple log (*for a few weeks*) where they **categorize their time** into **three buckets**:
      - ✓ Toil (*e.g., manual patches, alert response*).
      - ✓ Overhead (*e.g., meetings, HR, training*).
      - ✓ Engineering (*e.g., new feature development, automation coding*).
  - Team Surveys
    - **Conduct** a quick, **anonymous survey** asking questions like:
      - ✓ "What is the single most frustrating manual task you perform weekly?"
      - "Roughly what percentage of your last week was spent on repetitive work?"
  - Shadowing/Observation
    - Have a **team lead** or manager spend a few **hours shadowing** an **operations engineer** to **observe** what their **hands-on work** consists of.
    - You will **quickly find copy-pasting commands** from **runbooks** is a major source of toil.

AHMAD

6

Phase 2 … continue

- **Reviewing Runbooks and Alerting**
  - Toil often **lives** in the **documents** and **processes** used for **operations**.
    - Runbook Review
      - ✓ **Examine** your standard operating procedures (**runbooks**) for:
        - ❖ Any **step** that involves a **human copying** and **pasting** a command.
        - ❖ **Steps** that **require human decision-making** based on simple thresholds (*e.g., "If CPU is >80%, restart process X").*
    - Alert Review
      - ✓ **Analyze** your **monitoring alerts** for recurrent "flap" or "chatter" alerts.
      - ✓ If an **alert triggers frequently** and the human response is always the same simple action (*e.g., "acknowledge and restart"),* that alert-response loop is pure toil and a prime candidate for self-healing automation.

AHMAD

7

Visual Assessment: Is it Toil?

Question	If Yes → Likely Toil
Is it manual and repetitive?	Yes
Does it interrupt engineering work?	Yes
Does it require sitting and watching?	Yes
Does it scale linearly with systems?	Yes
Can Machine or Automation handle it?	Yes

AHMAD

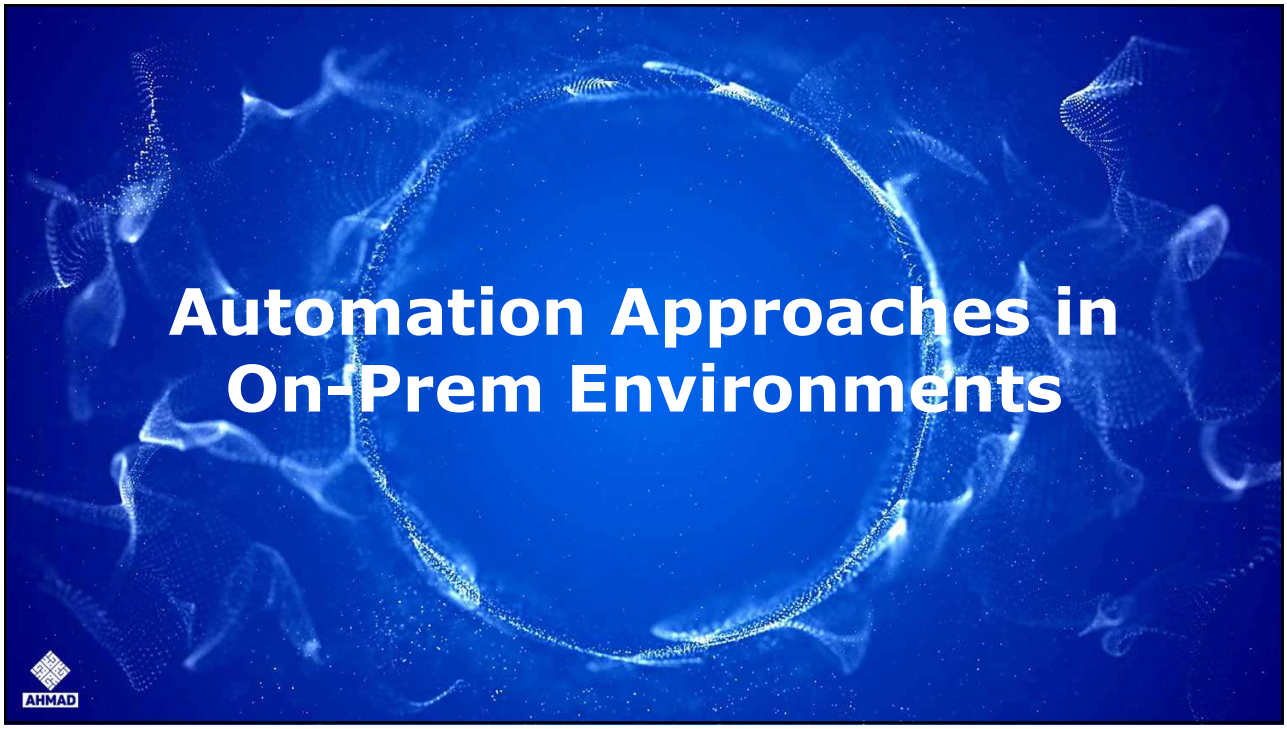
8

Common On-Premise Toil Examples

Category	Specific Toil Task
Provisioning	**Manually creating new Virtual Machines** (VMs) in VMware/Hyper-V, configuring IP addresses, or setting up new user accounts.
Patching/Config	Logging into individual servers to **install OS patches**, **update firmware**, or manually change a configuration file.
Incident Response	**Running** the **same diagnostic commands** (*e.g., iostat, netstat, top*) during an alert or repeatedly **restarting services**.
Capacity	**Manually** processing requests to **extend** a **logical volume** (LVM) or increase a virtual disk size after getting a "disk full" alert.
Access Control	**Manually adding/removing users** from Active Directory groups for access to specific applications or servers.
Change Management	**Manually executing pre-change health checks** and post-change verification steps for a planned deployment.

AHMAD

9



10

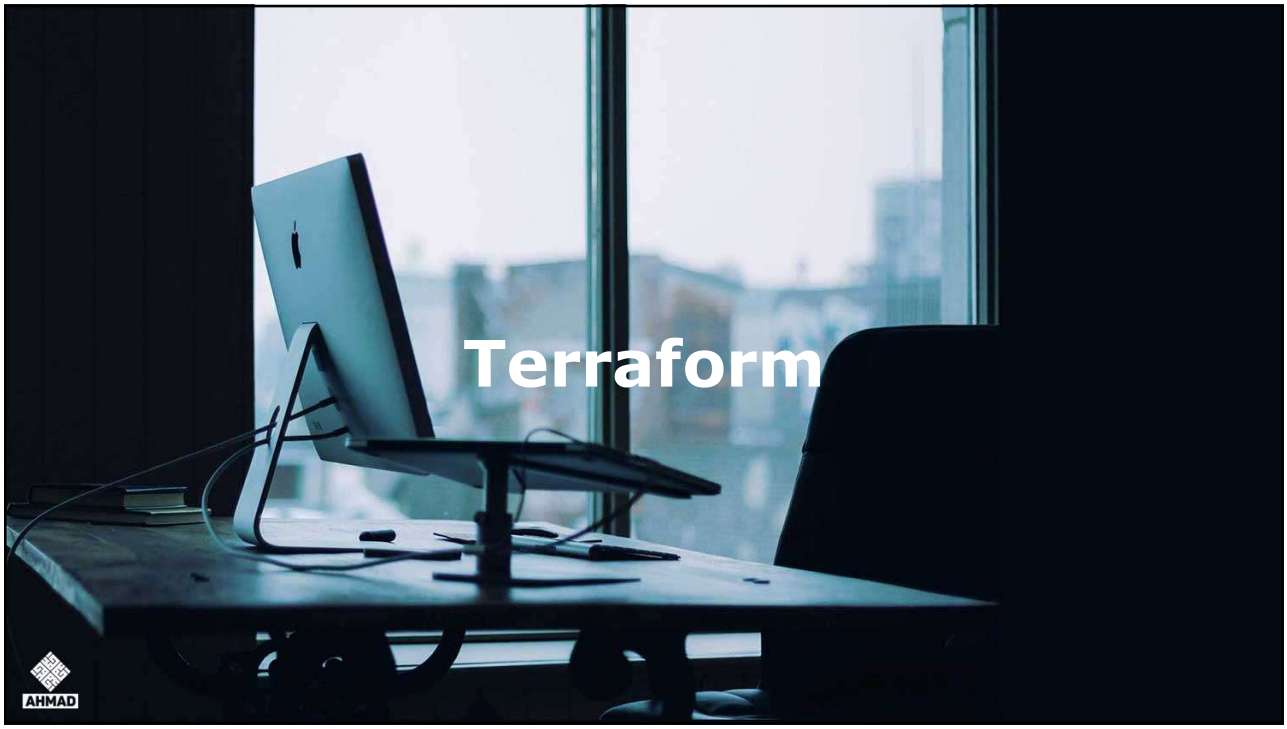
11

Infrastructure as Code (IaC)

- **Infrastructure as Code (IaC)** is the practice of managing and **provisioning infrastructure** through machine-readable definition files, rather than physical hardware configuration or interactive configuration tools.
- IaC shifts infrastructure management from a manual, configuration-centric activity to an automated, code-driven one:
  - **Automation**
    - By defining infrastructure in code, provisioning and updates can be automated, **eliminating manual steps** and the associated **toil** and human error.
  - **Version Control**
    - Configuration files are stored in systems like Git.
    - This allows teams to **track every change** to the infrastructure, roll back to previous versions if issues arise, and review changes via standard **pull requests**.
  - **Consistency and Repeatability**
    - IaC ensures that **environments** (*development, testing, and production*) are **provisioned identically** every time.
    - This prevents **configuration drift**—where configurations become slightly different across environments—which is a major source of bugs.
  - **Documentation**
    - The code itself serves as the precise, **up-to-date documentation** for the infrastructure.
  - **Speed and Scale**
    - Infrastructure can be **deployed** rapidly and repeatedly **across hundreds or thousands of servers** or services with a single command.

AHMAD

12



13

Terraform

- Terraform is an **Infrastructure as Code** (IaC) **tool** created by **HashiCorp**.
- It is designed to **provision**, **change**, and version infrastructure safely and efficiently.
- **Core Principles of Terraform**
  - Declarative Language (HCL)
    - Terraform uses the **HashiCorp Configuration Language** (HCL).
    - You **write configuration files** that **describe** the **desired end state** of your **infrastructure** (*e.g., "I want one web server, a load balancer, and a firewall rule allowing port 80 traffic*").
  - Providers and Abstraction
    - Terraform **uses Providers** to **interact** with virtually **any platform** or service that exposes an **API**.
    - This allows **Terraform** to be **platform-agnostic**.
    - A **Provider** is a **plugin** that understands the API interactions for a **specific technology** (*e.g., AWS, Azure, Google Cloud, VMware vSphere for on-premise, Kubernetes, or even specialized SaaS applications like Datadog*).
    - This **abstraction** means you can use the **same consistent HCL syntax** to manage resources **across completely different environments**.

AHMAD

14

Terraform … continue

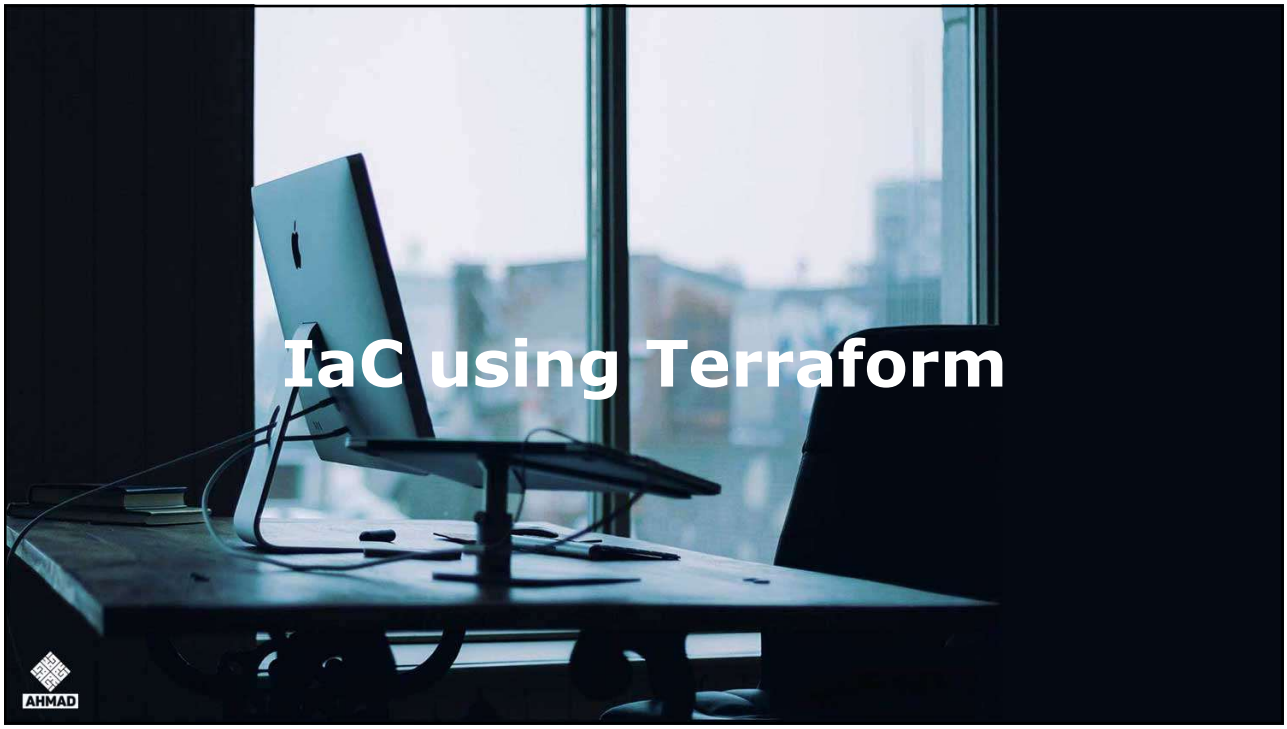
- Execution Plan
  - Before applying any changes, Terraform **executes** the **terraform plan command**.
  - This command **compares** your **desired configuration** (in your HCL files) with the **current state**.
  - It **generates** an **Execution Plan** that details exactly what **resources** will be **created**, **modified**, or **destroyed**.
- State Management
  - Terraform **tracks** the **actual state** of your managed **infrastructure** in a state file.
  - The **state file** is a record of the **mapping** between your **configuration** and the **real-world resources**.
  - It is essential for performance and reliability because it allows Terraform to:
    - ✓ **Know** the **current configuration** without querying every resource on every run.
    - ✓ **Track** metadata and **dependencies**.
    - ✓ **Detect** configuration **drift** — where a resource was manually changed outside of Terraform — and alert you to the mismatch.

15

Terraform … continue

- **Terraform Workflow**
  - The typical *usage* of *Terraform follows* three *simple steps*:
    - Write
      - ✓ Define your infrastructure resources in HCL files.
    - Plan
      - ✓ Run terraform plan to see the proposed changes.
    - Apply
      - ✓ Run terraform apply to execute the changes and provision the infrastructure as defined.

16



17

IaC using Terraform

- Automation in on-prem environments helps reduce manual effort, eliminate toil, improve consistency, and accelerate service delivery by using **Infrastructure as Code** (IaC)**, Scripts, APIs**, and **Automated Pipelines**—even without full cloud-native capabilities.
- **Infrastructure as Code (IaC)** is the practice of managing and provisioning IT infrastructure through code and configuration files, rather than through manual processes or interactive tools.
- **Terraform** is a declarative, platform-agnostic IaC tool widely used for this purpose, even in on-prem environments.
- **Script** is a sequence of commands executed by an interpreter (*like the command shell or Python interpreter*) to automate a specific task.
- **API** is a set of defined methods and protocols that allow different software components to communicate with each other.

AHMAD

18

Deployment Using Pipelines

- **Automated pipelines**, typically implemented via **Continuous Integration/Continuous Delivery** (*CI/CD*) tools (*like self-hosted **Jenkins**, **GitLab CI**, or **GitHub Actions Runners***), codify the entire process of testing and deploying code or infrastructure changes.

- **Automated Deployment (CD)**
  - Code-to-Deployment Flow
    - The **pipeline** ensures that **committed code** (*whether application code or IaC configuration*) is **automatically built**, tested, and **deployed to environments** (*Dev, Test, Prod*) without human intervention in the middle steps.
  - Standardization
    - Pipelines **enforce standardization** by using common build and deployment scripts, ensuring that the **deployment process** is **identical** every time.
    - This **eliminate**s the **toil** of creating **manual deployment runbooks** and reduces deployment-related errors.
  - Safety and Rollback
    - Modern pipelines **integrate automated checks** that **verify service health post-deployment**.
    - If the **health checks fail** (*e.g., application doesn't start, latency spikes*), the pipeline can be configured to execute an **automatic rollback** to the last known stable configuration, eliminating the need for an on-call engineer to perform an urgent, manual fix.

19

Deployment Using Pipelines ... continue

- **Automated Monitoring**
  - Integration with Observability
    - The **pipeline integrates** seamlessly with **on-prem monitoring systems** (*like **Prometheus** and **Grafana** or commercial agents*).
    - It doesn't just deploy; it also ensures that the **new service** is immediately being **monitored** with the correct **alert thresholds**.
  - Continuous Feedback
    - Automated **pipelines feed performance** and **error data back** to the development team immediately, creating a **continuous feedback loop** that guides further automation efforts, turning reactive troubleshooting into proactive engineering.

20

Scripts and APIs

- While IaC and CI/CD manage large, macro-level changes, the smallest, **most frequent tasks** are often **automated** by the direct use of **scripts** and **system APIs**.
  - **Scripts (Python, PowerShell, Bash)**
    - Scripts are the fundamental tools for automating specific, often repetitive, operational toil.
    - Automated Diagnostics
      - Instead of **manually logging** into a **server** to run 10 diagnostic commands **after** a critical **alert**, a script can be **triggered** to **automatically collect all necessary data** (*logs, process lists, metrics*) and bundle it up.
      - This significantly **reduces** the **triage toil**.
    - Simple Self-Healing
      - Scripts can be **integrated** with **monitoring systems** to **perform automated**, low-risk **remediation** for **recurring issues**.
    - Reporting and Cleanup
      - Scripts routinely **perform file cleanup**, **temporary cache clearing**, and generate scheduled compliance or usage reports, eliminating tedious manual data gathering.
    - Example
      - If a monitoring alert **detects** that a common **service** has **stopped**, a PowerShell or Bash script is **automatically executed** to **restart** the **service**.
      - This transforms a repetitive, tactical on-call interrupt into an automated, non-interrupting process.

21

Scripts and APIs … continue

- **APIs (Application Programming Interfaces)**
  - APIs are the **essential bridge** that **allows software** to **manage complex hardware** and **system software programmatically.**
    - API as the Control Layer
      - ✓ Almost all modern **on-prem components** — **virtualization platforms** (*vSphere*), **storage arrays** (*e.g., NetApp, Dell*), **firewalls**, and **operating systems** — **expose APIs**.
    - Enabling Orchestration
      - ✓ The **API** is what allows **automation tools** like Terraform or Ansible to work.
      - ✓ When Terraform applies a change to vSphere, it does so by making a series of API calls, not by manually clicking buttons in a console.
    - Integrating Disparate Systems
      - ✓ **APIs glue systems together**, eliminating manual data transfer toil.
      - ✓ For example, an **API call** can be used to **pull** a list of **deployed VMs** from vSphere, **feed** that list to a **Configuration Management tool** (*Ansible*), and then **post** the **successful configuration status** back to the **ticket management system** (*ServiceNow*).

22

When to Use What

Approach	Best For	Tools
IaC (Terraform)	Provisioning, infrastructure consistency	Terraform, Ansible, VMware vRA
CI/CD Pipelines	Deployment, monitoring, auto-registration	Jenkins, GitLab
Scripts & APIs	Ticket-based, legacy, small quick automations	PowerShell, Python, REST APIs

23

Cultural Challenges in Automation Adoption

- **Fear of Change and Job Security**
  - **Operations teams**, who have historically managed infrastructure manually, may fear that **automation** will **make their jobs redundant**.
    - Solution: **Reposition automation not** as a **job replacement**, **but as a shift** from operators (*running manual tasks*) to automation engineers (*writing code to run the tasks*), focusing on higher-value work.
- **"If it ain't broke, don't fix it" Mentality**
  - In **stable**, long-running **environments**, there is **often resistance** to **changing established**, even if inefficient, **manual processes** because of the perceived **risk** of **automation introducing new bugs**.
    - Solution: **Start** with **low-risk**, **high-toil tasks** (*like report generation or basic VM cleanup*) to demonstrate quick, safe wins and build trust in the automated system.
- **Siloed Teams**
  - The transition to **IaC** and **pipelines requires closer collaboration** between **development**, **operations**, and **security teams** (the essence of DevOps/SRE).
  - **Traditional organizational silos** often **resist sharing responsibility** for the production environment.
    - Solution: **Implement cross-functional training** and use a **shared version control sy**stem (like Git) as the central collaboration point for all infrastructure code.

24

Skill Challenges in Automation Adoption

- **Programming Proficiency Gap**
  - Many **traditional system administrators** are experts in troubleshooting and command-line execution but **lack proficiency** in **programming languages** (*like Python*) or **IaC languages** (*like HCL*) required for automation.
    - Solution: **Invest heavily** in **structured training** and mentorship.
    - Encourage teams to adopt one automation language (*e.g., Python for scripts, Ansible for config*) and focus on mastering it.
- **Tool Sprawl and Complexity**
  - **On-prem automation** often **requires integrating multiple complex tools** (*Terraform, Jenkins, Ansible, vSphere, monitoring systems*).
  - The **steep learning curve** for this toolchain can **slow adoption**.
    - Solution: **Standardize** the **toolchain** and **create internal starter kits** or pre-built, simple templates that teams can use immediately, lowering the barrier to entry.
- **Testing and Validation Skills**
  - **Moving** from **manual** checks to automated testing **requires new skills**.
  - **Engineers** must **learn** how to **write effective integration tests** for infrastructure (*e.g., using tools like Terratest*) to validate that the automated system works correctly before deployment.
    - Solution: **Make infrastructure testing** a **mandatory step** in the **CI/CD pipeline**, treating infrastructure code with the same rigor as application code.

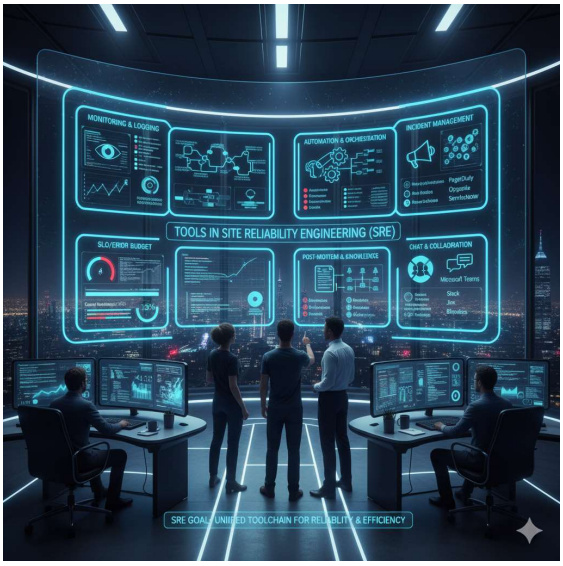
AHMAD

25

26

Tools in (SRE)

- SRE relies on a combination of tools:
  - **Monitoring**
  - **Automation**
  - **Reliability**
  - **Incident Response**
  - **Performance, Observability**
  - **CI/CD**
  - **Chaos Engineering**
- These help maintain **availability, performance, scalability, and resilience** while reducing toil.



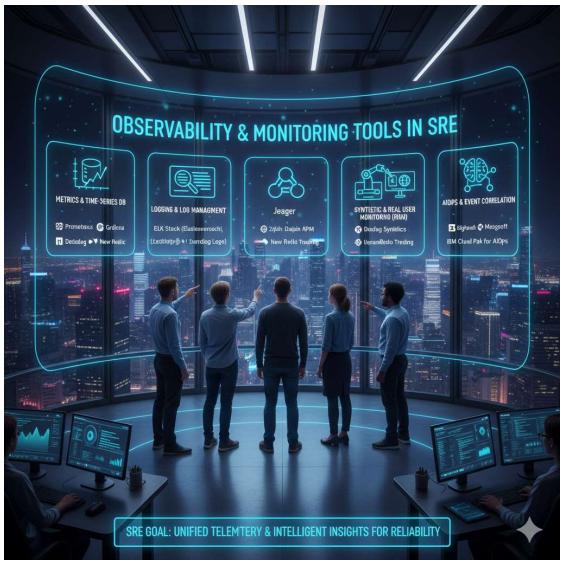
AHMAD

27

28

Observability & Monitoring Tools

- These tools are the "eyes and ears" of SRE, providing the critical data needed to understand system health and debug problems.
- Observability encompasses **Metrics, Logs, and Traces** (the Three Pillars).



AHMAD

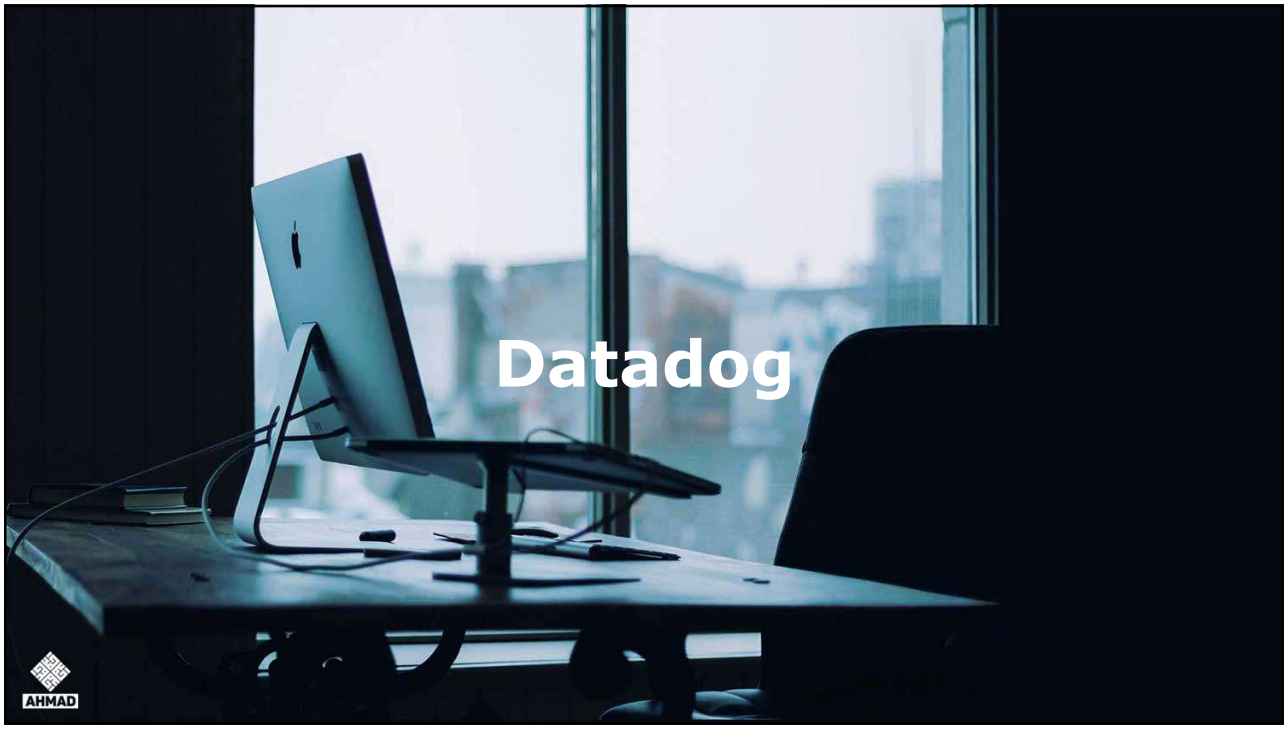
29

Observability & Monitoring … continue

Tool	Key Features	Purpose in SRE
Prometheus	Time-series metrics, alert rules, pull-based model	**Metrics collection**, **alerting**, SLI/SLO tracking
Grafana	Visualization dashboards, alerting, integrates with Prometheus, Loki, Elastic	Build **SLO dashboards**, error budget visualization
Datadog	Full-stack monitoring, APM, logs, metrics, synthetic monitoring	**Unified monitoring** & alerting across services
New Relic	Distributed tracing, real-time performance insights	Application and **business transaction monitoring**
Nagios	Host & service monitoring, alerting	Basic **infrastructure monitoring**, uptime tracking
Zabbix	SNMP monitoring, API, auto-discovery	**Network**, **server monitoring** in hybrid environments
Splunk	Log analytics, SIEM, AI insights	**Log-based monitoring**, security & incident analysis
Elastic Stack (ELK)	Elasticsearch, Logstash, Kibana	Centralized **log management** and **visualization**
Jaeger / Zipkin	Distributed **tracing**, root cause analysis	**Track microservice performance**, **latency issues**
Loki	Scalable **log aggregation** (Prometheus-style)	Lightweight **log collection** & querying
Fluentd / Fluent Bit	Log collection, parsing, forwarding	Combine logs from multiple sources

AHMAD

30



31

Datadog

- Datadog is a **commercial**, **unified monitoring** and security platform designed for cloud-scale **applications**, **servers**, and **services**.
- It **consolidates** all **three pillars** of **observability** — **metrics**, **logs**, and **traces** — into a single, integrated platform.
- Its primary purpose in modern SRE and DevOps is to **provide** a **comprehensive**, **real-time view** of application and **infrastructure health** across hybrid and multi-cloud environments, enabling teams to detect issues faster and reduce Mean Time To Resolution (MTTR).

DATADOG

AHMAD

32

Datadog | Components

- Datadog's functionality is delivered through several tightly **integrated products**, all of which use a **single agent installed** on your hosts or a library integrated into your application code.

- **Application Performance Monitoring (APM) & Distributed Tracing**
  - This is Datadog's implementation of the **Traces pillar**.
  - Purpose
    - To gain deep visibility into application code performance and **track requests across** distributed **services** (*microservices*).
  - Features
    - Distributed Tracing for full request-journey mapping; Code-Level Visibility to **pinpoint slow functions** and **database queries**; automatic **generation** of **service dependency maps**.

- **Infrastructure Monitoring & Metrics**
  - This covers the **Metrics pillar** for the underlying infrastructure.
  - Purpose
    - To monitor the **performance** and **resource utilization** of **hosts**, **containers**, serverless functions, and **cloud services** (AWS, Azure, GCP).
  - Features
    - **Real-time collection** of **metrics** (*CPU, memory, disk I/O, network*) from thousands of integrations; automated host and container discovery; scalable time-series storage; powerful, flexible dashboards.

33

Datadog | Components ... continue

- **Log Management**
  - This covers the **Logs pillar**.
  - Purpose
    - To **centralize**, **normalize**, and **analyze** massive volumes of application and infrastructure **logs** for debugging and auditing.
  - Features
    - **Log collection** and **ingestion** from all sources; Log Processing Pipelines to parse, enrich, and filter data; high-speed searching and **analytics**; the ability to **turn logs** into **custom metrics**.

- **Synthetic Monitoring & Real User Monitoring (RUM)**
  - These **provide** the crucial **black-box** or external view of **service health**.
  - Synthetic Monitoring
    - Runs automated, scriptable **tests** (*like simulated user logins or API calls*) **from global locations** to proactively **check site availability** and performance against external SLOs.
  - RUM
    - Measures the **performance** and **errors** experienced by actual users in their **web browsers** or **mobile apps**, providing critical client-side data.

34

Datadog | Components … continue

- **Alerting & AIOps (Watchdog)**
  - This provides the **actionable intelligence layer** over the **collected data**.
  - Alerting
    - Allows SREs **to set alerts** based on **single metrics**, **composite conditions**, or **anomaly detection** (automatic identification of unusual patterns).
  - Watchdog
    - A **core AI feature** that automatically surfaces hidden or emerging issues, such as **increased application errors** or sudden **infrastructure degradation**, often before an SRE manually notices the trend.

35

Purpose in SRE

- Datadog **supports** the **core philosophies** of **SRE** and **DevOps** by:
  - **Minimizing Context Switching**
    - Teams **don't have** to **jump between separate tools** for metrics, logs, and traces, which drastically **speeds** up **incident investigation**.
  - **SLO Tracking**
    - Provides clear, **real-time dashboards** to track **Service Level Indicators** (SLIs) like request latency and availability, making it easy to **monitor compliance with Service Level Objectives** (SLOs).
  - **Faster Root Cause Analysis (RCA)**
    - The seamless correlation between metrics, logs, and traces allows an engineer to click from an alert (**metric**) directly to the slow code path (**trace**) and the relevant error message (**log**).

36

37

Incident Management & Alerting Tools

- These tools govern the on-call rotation, ensure the right person is notified for the right problem, and streamline the entire incident lifecycle.

Tool	Features	Role in SRE
PagerDuty	On-call management, escalation, incident tracking	**Manage alerts**, response workflows, automation
Opsgenie	Alert routing, incident collaboration	**Coordinate response** teams for major incidents
VictorOps (Splunk On-Call)	Timeline view, team routing, post-mortems	Real-time incident collaboration
ServiceNow	ITSM, change management, knowledge base	Manage incidents and change control
Slack / MS Teams	Ops collaboration channels	Real-time communication during incidents
Service Now	**ITSM platform** managing incidents, problems, change, service catalog, SLAs.	Tracks **incident lifecycle**, enforces reliability workflows.
BigPanda	AIOps / **Event Correlation** & Incident Automation tool.	**Combines alerts** from multiple systems (Datadog, Splunk, Prometheus), **reduces alert fatigue**
Moogsoft	AI-powered event correlation, anomaly detection,	**Incident noise reduction**

AHMAD

38

39

BigPanda

- BigPanda is an **AIOps** (*Artificial Intelligence for IT Operations*) and **Event Correlation** & **Incident Intelligence** platform.
- It **collects alerts** from **multiple monitoring tools** (*Prometheus, Datadog, Dynatrace, ELK, Nagios*), **correlates them intelligently**, and helps teams quickly identify the **root cause of incidents**.
- Its core function is to tackle the problem of **alert fatigue** by unifying events from disparate monitoring tools and using machine learning to intelligently correlate them into a single, actionable **Incident**.
- Its main goal: **Reduce alert noise, automate incident detection, improve MTTR, and increase reliability.**
- It **sits** strategically **above** your **existing monitoring tools** (*like Prometheus, Datadog, Splunk*) to transform raw data noise into clear, contextualized signals.

AHMAD

40

BigPanda … continue

- BigPanda is not an observability tool that collects metrics or logs; it is an **Incident Intelligence Platform**.
  - **Platform Type**
    - AIOps (**Artificial Intelligence for IT Operations**).
  - **Main Goal**
    - To dramatically **reduce alert noise** and Mean Time To Resolution (***MTTR***) by providing immediate, accurate context during an incident.
  - **Agnostic Architecture**
    - It is **vendor-agnostic**, designed to ingest and normalize data from almost any IT monitoring, observability, or change management system, including legacy tools and modern cloud-native solutions.

41

BigPanda … continue

- **Key Features**
  - Alert Correlation
    - **Uses AI/ML** (*and customizable patterns*) to **automatically group hundreds** of **individual**, related **alerts** (e*.g., CPU, latency, and connectivity alerts from the same server cluster)* **into one single**, high-level **Incident**.
  - Noise Reduction
    - **Filters, deduplicates**, and **aggregates** raw events, often reducing alert volume by over 90%.
  - Contextual Enrichment
    - **Adds external information** to the **correlated incident**, such as **data** from your **CMDB** (*Configuration Management Database*), **change logs** (*recent deployments*), and topology maps.
  - Root Cause Analysis (RCA)
    - Accelerates diagnosis by **highlighting** the most **likely contributing factors** (*e.g., showing a correlated alert cluster alongside the last configuration change that occurred*).
  - Workflow Automation
    - **Triggers automated actions** (*like creating a ticket in ServiceNow, convening a war room in Slack, or executing a runbook via Ansible*) based on the **conditions** of the correlated **incident**.

42

Purpose in SRE

- BigPanda directly **addresses several pain points SRE teams** face when managing high-scale, microservice-based systems.
- Its use directly supports the SRE mandate to reduce toil and increase system reliability.
- **Eliminating Alert Fatigue**
  - BigPanda **automates** this by:
    - Filtering: **Ignoring non-actionable** or low-severity **alerts**.
    - Grouping: **Merging** a "storm" of **related alerts**.
- **Accelerating Incident Triage and MTTR**
  - BigPanda provides this by:
    - Incident Diagnosis: **Presenting** the **correlated incident** alongside the **relevant change** event (*the code deployment that went out 5 minutes ago*) and the **full topology** (*the affected server cluster*) in one view.
    - Automated Response: Using the rich context of the correlated incident to **automatically trigger** the correct **runbook** or **remediation script** (*e.g., automatically restarting a component or rolling back a bad config*).

43

Purpose in SRE … continue

- **Unified View for Distributed Teams**
  - In large organizations, SRE, Network Operations (NOC), and traditional IT Operations often use different monitoring tools.
  - BigPanda acts as the "first pane of glass"—**a centralized incident console** that pulls data from all sources (*Datadog, Splunk, Nagios, etc.*).
  - This ensures all teams are looking at the same, correlated incident data, improving collaboration during high-severity outages.

44

45

IaC and Automation Tools

- SRE mandates that repetitive operational tasks and infrastructure provisioning must be automated to eliminate **toil** and ensure consistency.

Tool	Features	Purpose in SRE
Terraform	IaC, multi-cloud, version control	Automate **infrastructure provisioning**
Ansible	**Configuration automation**, **Application Deployment**, **agentless**	Reduce manual server setup and config toil
Puppet / Chef	**Configuration automation**, **Application Deployment**, **agent**	Enforce consistency across environments
SaltStack	Remote execution, orchestration	Manage large-scale infrastructure automation
Jenkins	**CI/CD pipelines**, automation workflows	Automate build, test, and release processes
GitLab CI/CD	Integrated version + CI/CD	Shift-left testing and deployment automation
ArgoCD	GitOps for Kubernetes	Automated progressive deployments
Helm	Kubernetes application templating	**Deploy applications** consistently on **K8s**

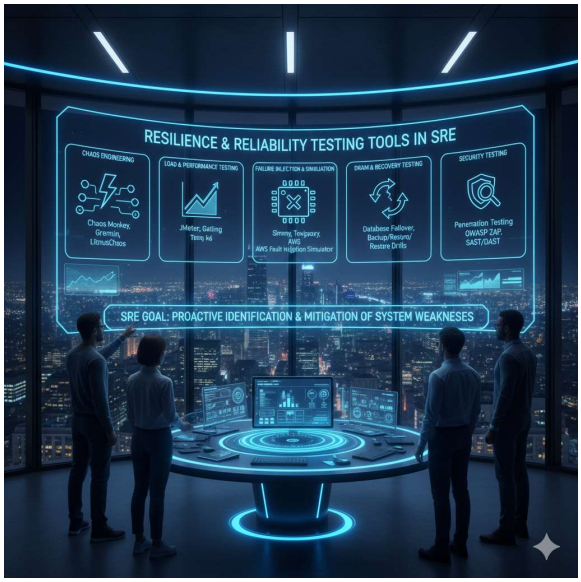
AHMAD

46

47

Resilience & Reliability Testing Tools

- In **Site Reliability Engineering (SRE)**, resilience is not assumed — it is **engineered, tested, validated, and continuously improved**.
- To build resilient systems, SRE relies on **Chaos Engineering**, **Fault Injection Testing**, and **Performance/Load Testing**.
- These practices help teams validate:
  - How systems behave under stress
  - What happens when failures occur
  - How quickly systems recover (MTTR)
  - Whether reliability targets (SLOs) can be met



AHMAD

48

Resilience & Reliability … continue

- **Chaos Engineering**
  - Chaos Engineering is the discipline of **intentionally injecting failures into a system to test its resilience and recovery capabilities**.
  - It helps **uncover weaknesses before** they cause **real outages** by **simulating real-world failure scenarios** like server crashes, network latency, or cloud region failure.
  - Its goal is to **build confidence in system reliability, validate SLOs, and improve MTTR**.

- **Fault Injection**
  - Fault Injection is a **more focused form** of **chaos testing** where **specific types of failures** — such as **CPU exhaustion**, **API failure**, **packet loss**, or **disk outage** — are **deliberately introduced** to assess how the system handles known faults.
  - It is used to ensure **recovery mechanisms, alerting, failover, and auto-healing systems** work as expected.

49

Resilience & Reliability … continue

- **Performance Testing**
  - Performance Testing evaluates how a **system performs** under **normal**, **peak**, and **extreme load conditions**.
  - It measures **response time**, **throughput**, **latency**, **resource usage**, and **error rates**.
  - It helps SREs verify **capacity planning**, **scalability**, and **user experience** against SLOs and SLAs.

- **Load Testing**
  - Load Testing is a **type** of **performance testing** where the system is subjected to **expected or above-normal user loads** to validate its ability to **handle increasing traffic**.
  - It helps identify when the system starts degrading, **detect bottlenecks**, and ensure that user experience remains acceptable even at high concurrency levels.
  - It supports reliability, scalability, and stability goals.

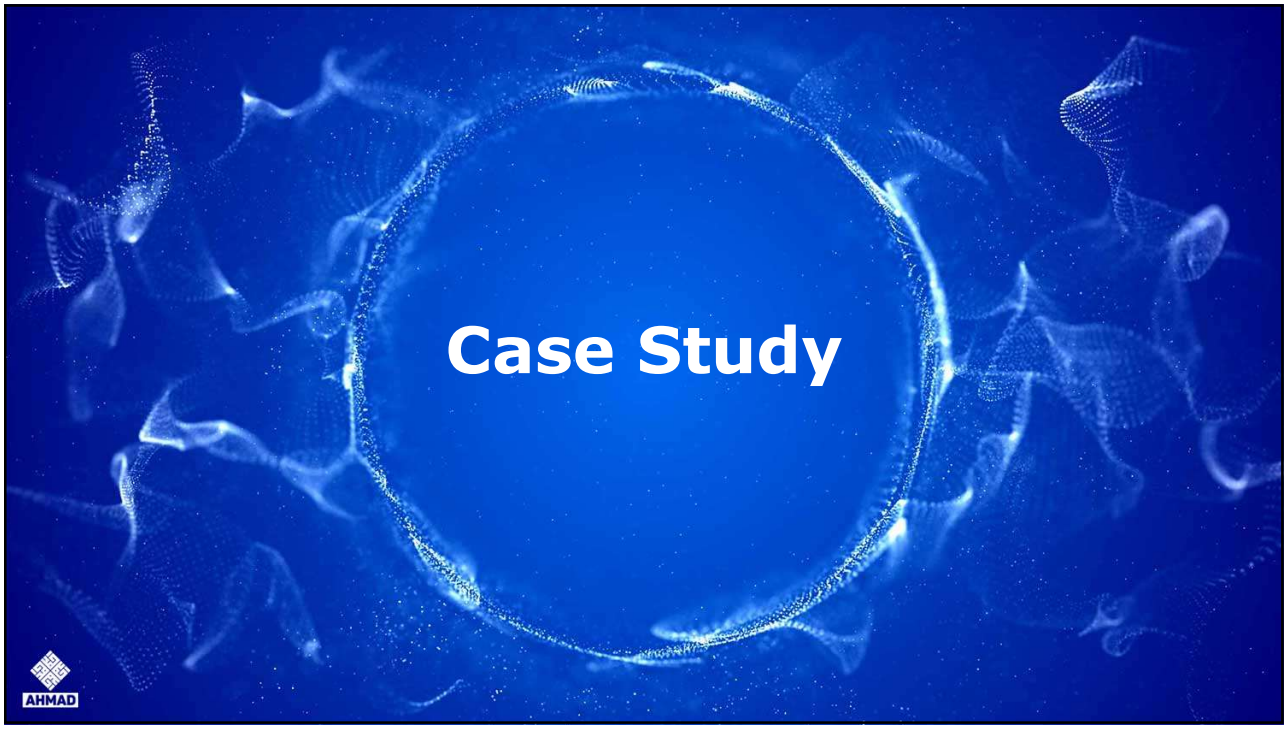
50

Resilience & Reliability … continue

Tool	Key Features	Role in SRE
Gremlin	Safe **chaos experiments**, failure injection (CPU, memory, network), blast radius control, scheduling	**Tests failure readiness**, improves resilience, validates auto-healing & recovery
Chaos Monkey (Netflix)	Random termination of servers/instances, auto-injection into production	**Validates redundancy**, **high availability**, **failure tolerance**
LitmusChaos	Kubernetes-native chaos testing, workflow-based fault injection, reliability scoring	Ensures **Kubernetes workloads** are resilient and meet SLOs
K6	**API stress/load testing**, SLO-based performance tracking, reliability under user load	Validates error budgets, user experience, and system scalability
Locust	**Python-based load simulation**, distributed testing, real user behavior modeling	Tests real-world RPS (requests/sec) and performance degradation
JMeter	**Load, stress**, endurance, and spike testing for APIs, web, databases	Identifies bottlenecks, tracks SLA compliance, supports capacity planning
Gatling	Real-time performance graphs, **high-concurrency testing**, CI/CD integration	Helps determine latency thresholds, peak load reliability
LoadRunner	Enterprise-scale **load testing**, supports multiple protocols (HTTP, SAP, Oracle)	Validates system stability, performance under extreme loads

AHMAD

51



52

Stock Exchange Trading Platform

- **Industry Context**
  - A national stock exchange runs a **high-frequency trading (HFT) platform**.
  - **Peak traffic occurs during**:
    - Market open (9:00–9:30 AM)
    - Major announcements
    - Sudden buy/sell surges
  - **They must ensure**:
    - Millisecond latency
    - Zero downtime during trading hours
    - Fast recovery
    - Automated reliability operations

53

Stock Exchange Trading Platform

- **Scenario**
  - **ExchangeX**, a major stock exchange, operates a trading engine that processes **50 million transactions/day**.
  - **Current Challenges**
    - **Manual deployment approval**, leading to delays.
    - **Frequent CPU spikes** on matching engine (buy/ sell) nodes.
    - **Order processing queue delays**, not detected until traders complain.
    - **Manual rollbacks** that take 20–30 minutes.
    - **Incidents are manually triaged**, slowing MTTR.
    - **Inconsistent logs**, making root-cause analysis difficult.
  - **Business Goals**
    - Achieve **99.99% availability** during trading hours.
    - Reduce **MTTR from 25 minutes to <5 minutes**.
    - Automate **scaling, deployment, rollbacks**, and **incident detection**.
    - Reduce **manual operations by 70%** within 3 months.

54

Stock Exchange Trading Platform

- **Questions**
  - **Q1.** What SRE Automation should be introduced to reduce manual deployment delays?
  - **Q2.** How can auto-scaling be implemented for a trading engine that cannot go down?
  - **Q3.** What automation can detect order queue delays before traders report issues?
  - **Q4.** How can automated rollbacks be implemented for critical trading services?
  - **Q5.** What SRE automation can reduce MTTR during high-severity incidents?
  - **Q6.** What logging and monitoring automation strategy would best fit a stock exchange platform?
  - **Q7.** How can SREs use resiliency testing safely in a financial trading system?
  - **Q8.** What error budget policies should be set for a 99.99% SLA trading system?

AHMAD

55

Q & A

Any concepts still unclear?

Thank you for attending

AHMAD

56