# SITE REALIABILITY ENGINEERING - FOUNDATIONS

TEKsystems Global Services

# WORKFORCE DEVELOPMENT

PARTICIPANT GUIDE

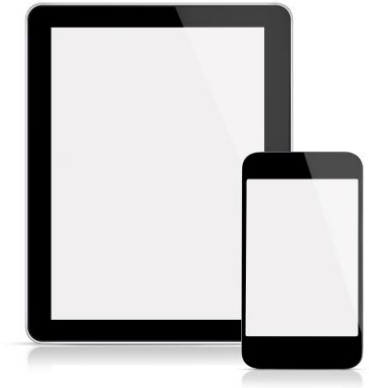
# LOGISTICS

## Class Hours:

- Class starts at 3:00 PM Eastern Time and Finish at 11:00PM
- There will be regular breaks in class.

## Telecommunication:

- Turn off or set electronic devices to silent (not vibrate)
- Reading or attending to devices can be distracting to other students
- Try to delay until breaks or after class



# INSTRUCTOR : CHRIS PISARSKI

**About**

- Born in France, Polish family, Paris area
- Started as Technical Support Manager for Capital Market Systems
- 15 years in Banking Industry, working for Société Generale Corporate Investment Banking
- 3 years working as Portfolio Analyst for Risk Department
- Passionate about Learning and Teaching
- 5 years working as Trainer on IT Operations & CIB topics
- Living in Montreal, Canada

# INTRODUCE YOURSELF

Time to introduce yourself:

- Name
- What is your role in the organization
- Indicate SRE experience

# COURSE OVERVIEW

**DevOps Definition:** Collaboration model breaking silos between development and operations.

**SRE Definition:** Applying software engineering to operations tasks.

**Course Goals:** Understand SRE principles, understand lifecycle ownership, automation, reliability.

**Target audience:** Developers, DevOps, Ops, Architects, mixed experience.

**Hands-on Focus:** Labs using Azure.

**Participant Outcomes:** Shift mindset to proactive reliability and measurable SLAs.

# COURSE STRUCTURE

**Day 1:**
Introduction to SRE, Core Principles, Monitoring and Alerting.

**Day 2:**
Release Engineering, Automation, and Reliability Practices.

**Day 3:**
Incident Management and Advanced Topics.

# PREREQUISITES & EXPECTATIONS

- Familiarity with programming, Linux/Unix, networking, databases.
- Basic knowledge of cloud platforms, monitoring, automation.
- Proficiency with version control (e.g., Git).
- Expectations: Active participation, hands-on practice.

# AGENDA FOR DAY 1

1. **Introduction to SRE:** History, principles, benefits.
2. **Production Environment:** Components, Azure equivalents.
3. **Embracing Risk:** Error budgets, risk management.
4. **Service Level Objectives:** SLIs, SLOs, SLAs.
5. **Eliminating Toil:** Automation, process improvement.
6. **Monitoring:** Scope and Best Practices.
7. **Alerting :** Building Observability
   Labs and pop quizzes to reinforce learning.

# INTRODUCTION TO SRE

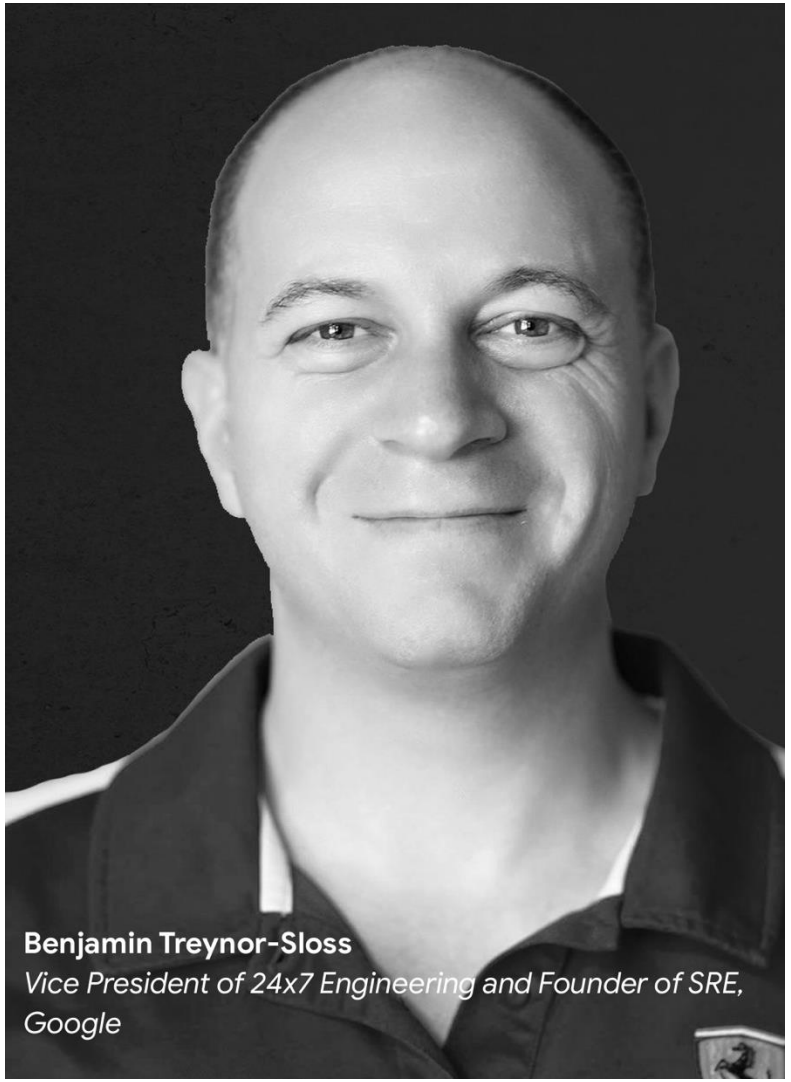
# SRE : SITE RELIABILITY ENGINEERING

*"SRE is what happens when a software engineer is tasked with what used to be called operations"*
Ben Treynor, Google

**Definition:**

Site Reliability Engineering is a discipline that incorporates aspects of software engineering and applies them to infrastructure and operations problems.

**Objectives:** Scalability, Reliability, Efficiency



# SRE : SITE RELIABILITY ENGINEERING


- Created at Google in 2003
- SRE workbook (publication O'Reilly)
- https://landing.google.com/sre/sre-book/toc/
- SRE has spread beyond Google
- Many organizations running large scale services are embracing SRE
- Facebook, Netflix, Microsoft…

# SRE PRINCIPLES AND PRACTICES

1. Embracing Risk
2. Service Levels and Error Budgets
3. Eliminating Toils
4. Monitoring and Observability
5. Automation
6. Simplicity
7. Release Engineering and Safe Changes
8. Incident Response and Postmortems

# ROLE OF SRE

- SREs are building ultra-scalable and highly reliable distributed software systems
- 50% ops related work
- 50% development tasks
- Monitoring, alerting, and automation are a large part of SRE

# AREAS OF PRACTICE

## Metrics & Monitoring

- SLOs
- Dashboards
- Analytics

## Capacity Planning

- Forecasting
- Demand-driven
- Performance

## Change Management

- Release process
- Consulting design
- Automation

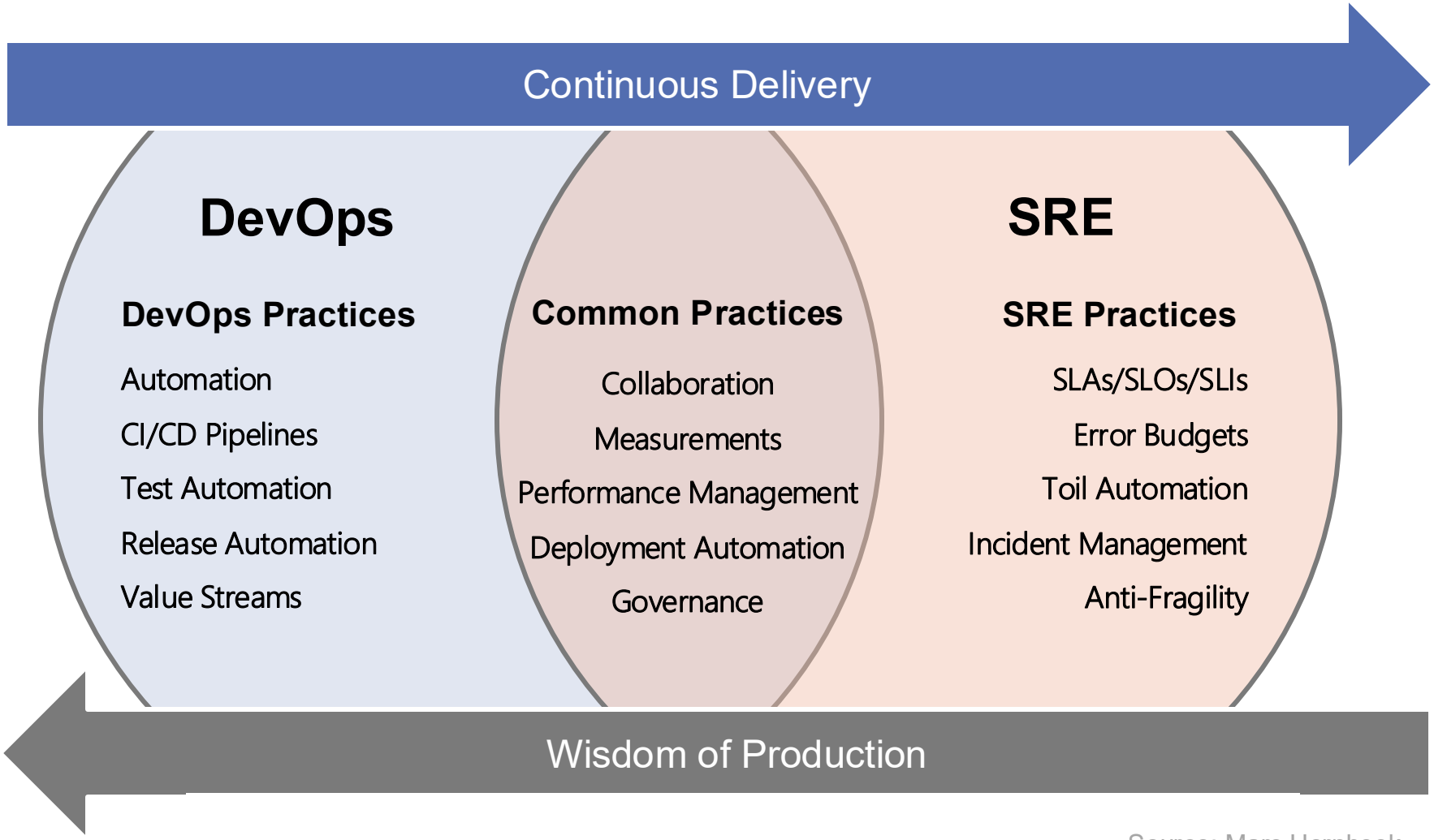
## Emergency Response

- On-call
- Analysis
- Postmortems

## Culture

- Toil management
- Engineering alignment
- Blamelessness

# SRE : DEVOPS VS SRE

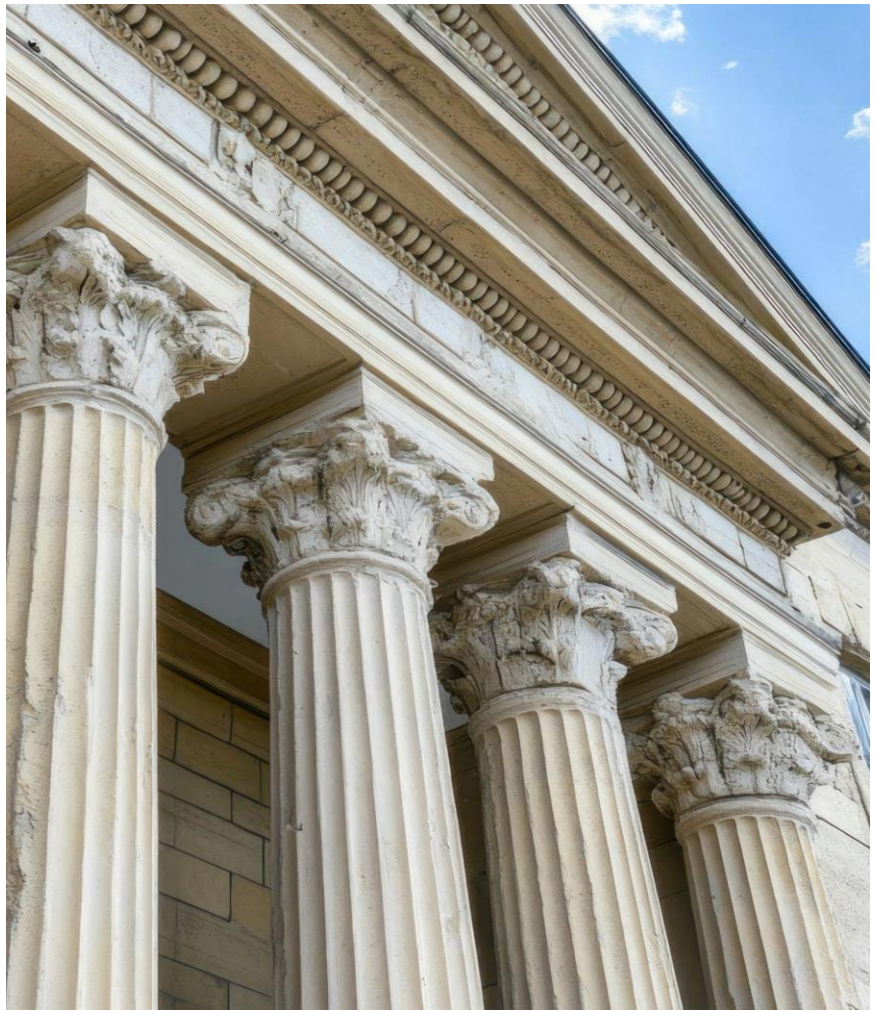



Source: Marc Hornbeek

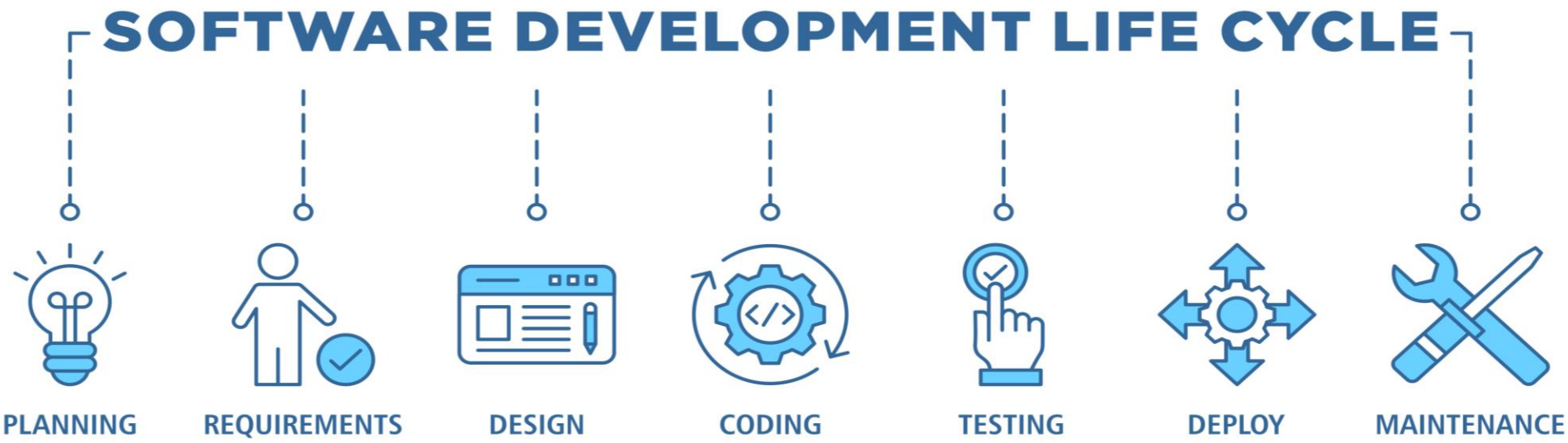
- DevOps: continuous delivery of software
- SRE: embed '**wisdom of production**'
- SRE complements DevOps

# THE DEVOPS PILLARS

1. Reduce organizational silos
2. Accept failure as normal
3. Implement gradual changes
4. Leverage tooling and automation
5. Measure everything



# SRE PRINCIPLES IN PRACTICE




1. Define reliability goals with SLOs and error budgets
2. Align availability expectations with business needs
3. Promote resilient, observable system architecture
4. Encourage built-in telemetry and graceful failure
5. Validate reliability and rollback readiness
6. Enable safe, automated release strategies
7. Improve through monitoring and incident learning

# SRE VS SYSADMIN APPROACH

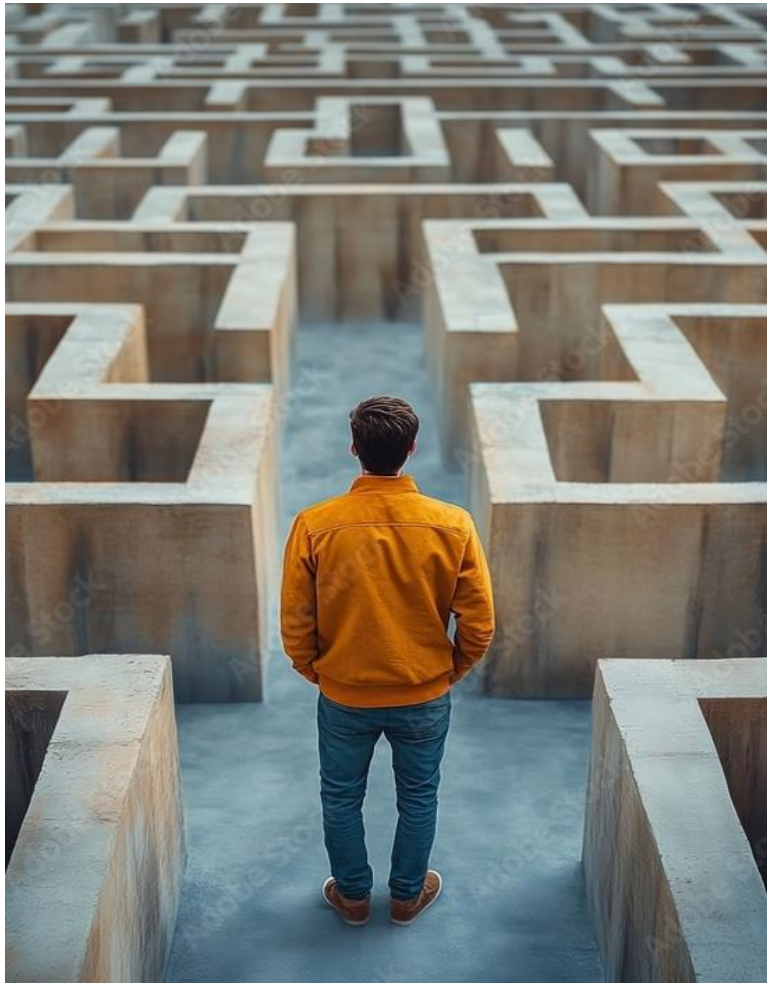
| Characteristic | Sysadmin | SRE |
|---|---|---|
| **Approach** | Reactive | Proactive |
| **Method** | Manual | Software-driven |
| **Focus** | System maintenance | Reliability |
| **Tools** | Adhoc Scripts | Azure Monitor, Azure Automation, Log Analytics |
| **Culture** | Finger-pointing, hero | Blame-free, continuous improvement |
| **Example** | Manual VM scaling | Automating VMSS with Azure Autoscale |

# BENEFITS OF IMPLEMENTING SRE

- **Optimize Resources**
  Automatically scale infrastructure to match demand
- **Avoid Outages**
  Design systems to detect and recover from failure early
- **Lower Latency**
  Route users intelligently and reduce performance bottlenecks
- **Increase Reliability**
  Use SLOs and error budgets to guide operational decisions
- **Improve Change Velocity**
  Deploy more frequently and safely through engineered reliability

# CHALLENGES OF IMPLEMENTING SRE

- Cultural Resistance
- Defining SLOs Properly
- Balancing Innovation and Reliability
- Toil Identification and Reduction
- Skill Gaps
- Tooling Fragmentation

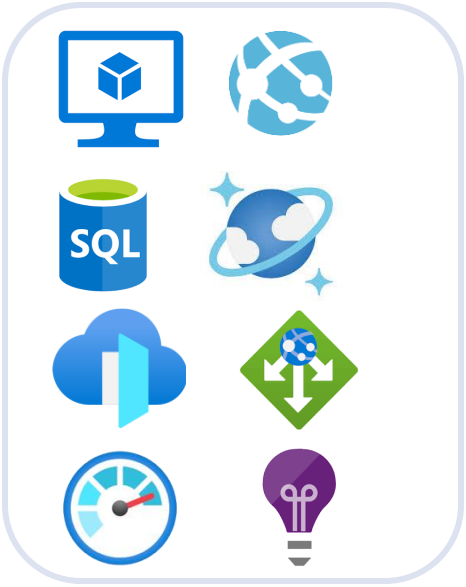


# PRODUCTION ENVIRONMENT

# PRODUCTION ENVIRONMENT OVERVIEW

## Core Components:

- Virtual Machines or App Services
- Azure SQL / Cosmos DB
- Load Balancers (Front Door, App Gateway)
- Monitoring tools (Azure Monitor, App Insights)




## Beyond the Application:

- Azure Active Directory
- Firewalls, VNETs
- Resource Policies & Tags
- CI/CD Pipelines

# PRODUCTION ENVIRONMENT – COMPUTE HARDWARE

**Purpose:** Run and scale application workloads efficiently

**Fault tolerance:** Availability Zones for high availability

**Virtualized Compute:**

- **Azure Virtual Machines** (VMs): Run workloads on Windows/Linux instances
- **Virtual Machine Scale Sets** (VMSS): Auto-scale and self-heal VM fleets
- **Azure Kubernetes Service** (AKS): Container orchestration for scalable apps

**Managed Compute Platforms:**

- **Azure App Service**: Managed hosting for web apps and APIs
- **Azure Container Apps**: Serverless containers with built-in autoscaling

# PRODUCTION ENVIRONMENT - SYSTEM SOFTWARE

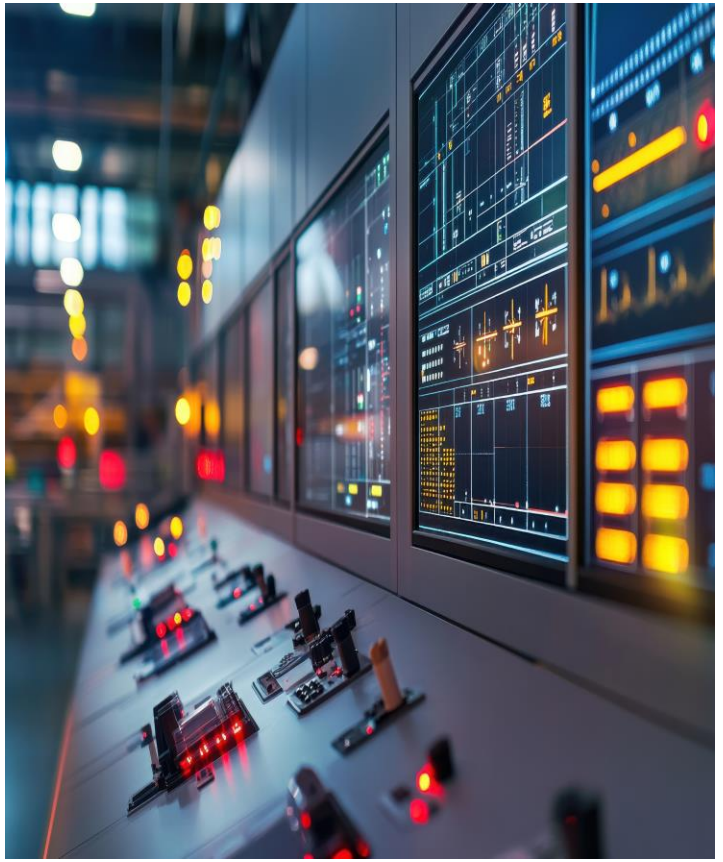
**Purpose:** Manage operating system lifecycle and orchestration reliably

**Operating Systems management:**

- **Guest OS** (Ubuntu, Windows Server): Images for VMs
- **Azure Update Manager**: Patch management and compliance tracking
- **Azure VM Extensions**: Deploy diagnostics, agents, and security tooling

**Orchestration Tools:**

- **AKS:** Azure Kubernetes Service for container orchestration at scale
- **Azure Functions:** Event-driven serverless compute for automation
- **Azure Arc:** Extends Azure management to non-Azure environments



# PRODUCTION ENVIRONMENT - STORAGE

**Purpose:** Provide reliable, durable, and scalable data storage

Block Storage:

- **Azure Managed Disks:** Persistent storage for VMs (OS and data disks)

Object Storage:

- **Azure Blob Storage**: Ideal for logs, backups, media, and data lakes

File Storage:

- **Azure Files**: Shared file system (SMB/NFS) for apps and lift-and-shift workloads
- **Azure NetApp Files**: High-performance enterprise-grade file shares

Data Protection:

- **Azure Backup & Recovery Vault**: Backup policies, snapshot management, and recovery

# PRODUCTION ENVIRONMENT - NETWORKING

**Purpose:** Ensure secure, reliable communication between services and users

**Connectivity & Segmentation:**

- **Virtual Network** (VNet): Private network for cloud resources
- **Subnets & Route Tables:** Organize and direct traffic within the Vnet

**Traffic Control & Security:**

- **Network Security Groups** (NSGs): Allow or block traffic to resources

**Traffic Distribution:**

- **Azure Load Balancer:** Internal or external traffic distribution for high availability
- **Azure Front Door:** Global HTTP/HTTPS load balancing with SSL and caching

# PRODUCTION ENVIRONMENT - MONITORING

**Purpose:** Track system health, performance, and behavior across distributed environments

**Core Monitoring:**

- **Azure Monitor:** Centralized platform for metrics, logs, and telemetry
- **Log Analytics Workspace:** Query and analyze log data across resources

**App-Level Observability:**

- **Application Insights:** Trace application requests, dependencies, and performance bottlenecks

**Visualization:**

- **Workbooks:** Custom dashboards for combining metrics, logs, and alerts

# PRODUCTION ENVIRONMENT - ALERTING

**Purpose:** Detect issues early and automate response to reduce downtime and escalation effort

Triggering Alerts:

- **Azure Alerts:** Define conditions based on metrics, logs, and SLO breaches

Response Actions:

- **Action Groups:** Notify via email, SMS, webhook, or trigger remediation workflows
- **Azure Automation Runbooks:** Execute scripted recovery or diagnostics

Orchestration:

- **Logic Apps:** Build low-code workflows to automate incident triage or escalation

External Integration:

- **ITSM Connectors:** Sync incidents with ServiceNow, PagerDuty, or AlertOps

# LAB 01 : SETUP AZURE VM & SCALE

Goal: In this lab, you create and compare virtual machines to virtual machine scale sets. You learn how to create, configure and resize a single virtual machine and, discover the Azure Command Line tool.

## Skills Covered

- Task 1: Deploy zone-resilient Azure virtual machines by using the Azure portal.
- Task 2: Manage compute and storage scaling for virtual machines.
- Task 3: Create a virtual machine using the CLI
- Task 4: Deploy an Apache web server
- Task 5: Deploy a webapp using Azure CLI & Kudu

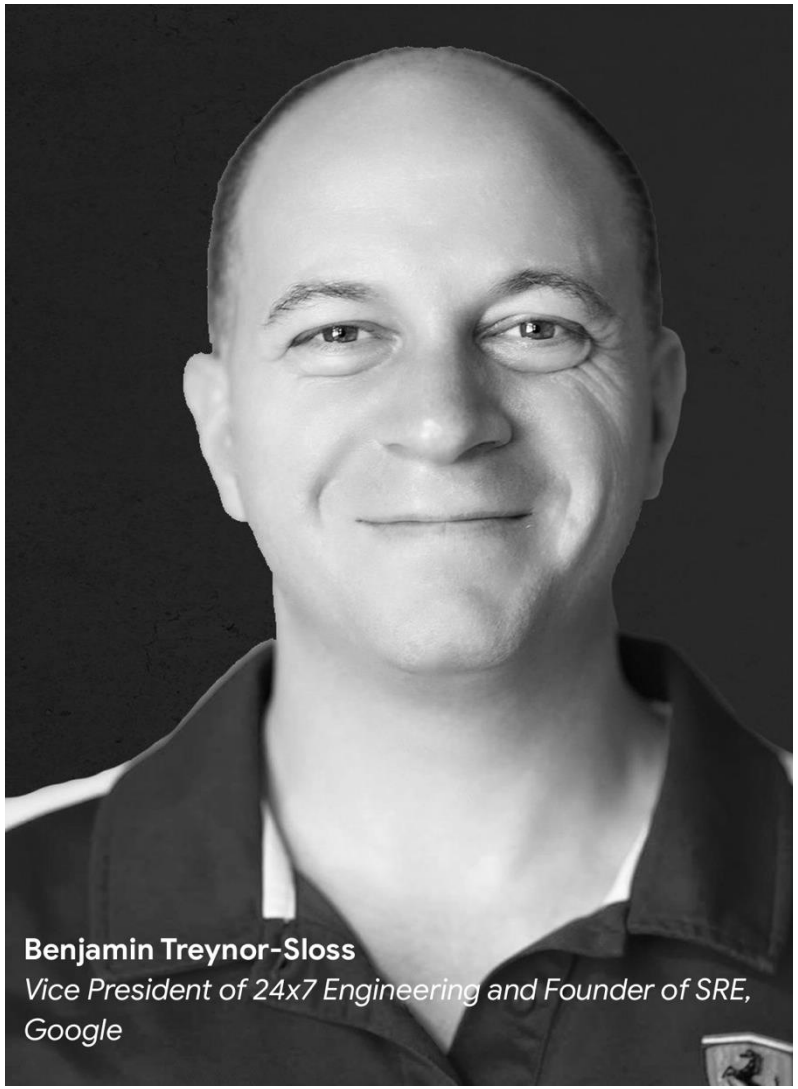
Instructions: AZ_SRE_lab_01.md

# EMBRACING RISK

# EMBRACING RISK

**"100% is the wrong reliability target for basically everything."**
Ben Treynor, Google

- 100% reliability is not the goal
- Accept failure within defined limits
- Balance innovation vs. stability
- Use risk to drive faster delivery and customer value



# MANAGING RISK


- **Capacity Planning** ensures resources meet expected and peak demand
- **Redundancy** protects against single points of failure (SPOF)
- Balance cost vs. risk using **data-driven forecasting**
- Plan for scaling, failover, and **disaster recovery** scenarios

# CAPACITY PLANNING – PLANNING FOR DEMAND

| Concept | Description |
| --- | --- |
| Forecast Resource Needs | Use historical traffic to predict future demand. |
| Plan for Peak Load | Ensure infrastructure can handle peak traffic levels. |
| Incorporate Growth Projections | Plan for organic user growth and sudden load surges. |
| Test Scaling Mechanisms | Validate that autoscaling works in production environments. |
| Balance Cost and Capacity | Avoid waste, avoid risk. |

Azure Monitor Metrics

Azure Advisor

Azure Autoscale rules

# REDUNDANCY – ELIMINATING SPOF

| | |
|---|---|
| Storage Redundancy | Spread instances across zones or regions |
| Compute Redundancy | Use zone or geo-redundant storage |
| Network Redundancy | Route traffic using load balancers and global services |
| Data Redundancy | Replicate databases across regions |

# AZURE-SPECIFIC STRATEGIES

**Azure Autoscale:**
Automatically adjust resources based on real-time demand

**Azure Availability Zones:**
Protect against datacenter-level failures.

**Azure Traffic Manager / Front Door:**
Provide global load balancing and failover routing.

**Azure Site Recovery:**
Enable disaster recovery and cross-region failover.

**SQL Auto-failover Groups / Cosmos DB Multi-region:**
Ensure database availability across regions

# AZURE-SPECIFIC STRATEGIES

Enables faster, safer innovation

Prioritizes reliability based on user needs

Encourages data-driven decisions

Shared ownership between development and operations

Reduces unplanned work and operational burnout

Happier users

# SERVICE LEVEL OBJECTIVES

# ROLE OF SRE

- **SLO:** Internal time-bound target level for the reliability of your service.
  Example: 99.9% availability

- **SLI:** Indicator of the actual level of service that users are experiencing.
  Example: availability, latency, error rate

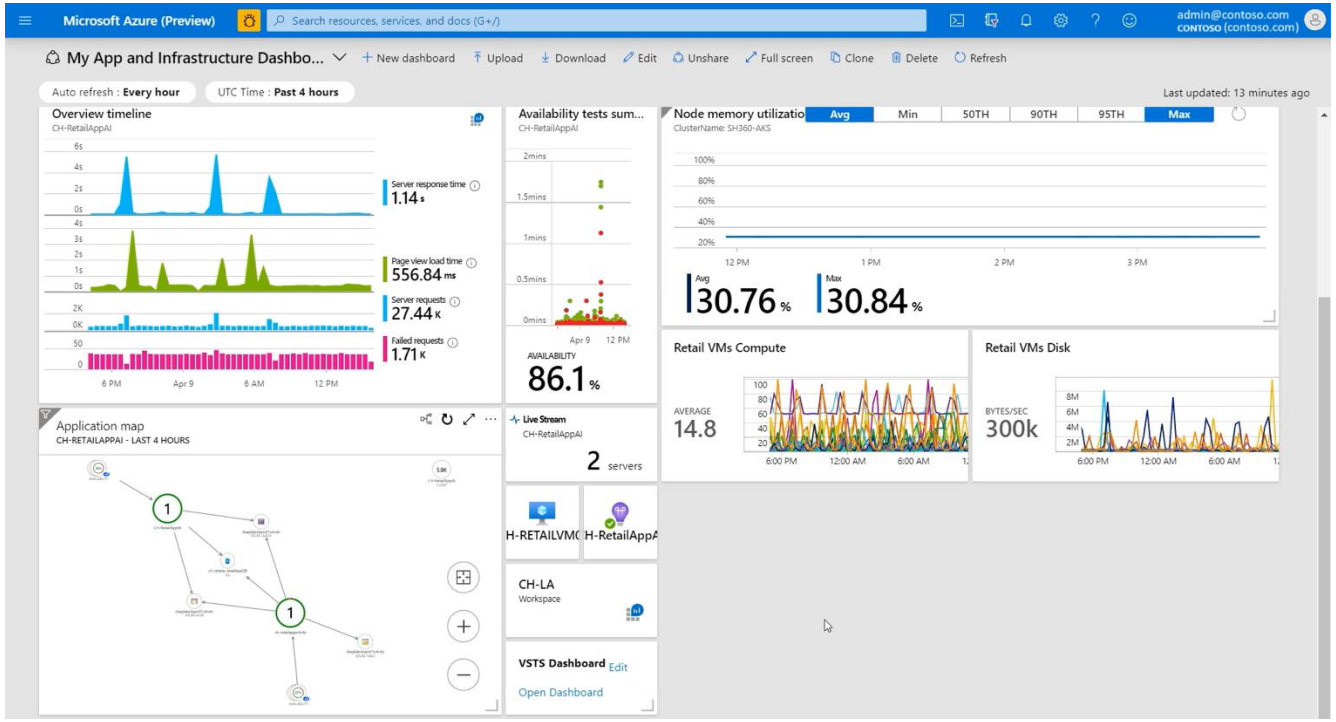
- **SLA:** Business contract in the terms of service with users or clients.
  In practice: penalties if breached

- **Error Budget:** How much failure is acceptable.
  In practice: If the budget is used → pause rollouts

# SLOS & ERROR BUDGETS

- Main "Weapon" to improve our service quality as SRE
- Quantify Acceptable Risk
- Enable Informed Decisions
- Balance Velocity & Reliability
- Drive SRE Discipline

# SLOS - INDICATORS IN PRACTICE




Metrics that matter:

- Availability
- Response time
- Latency
- Error rate

Azure Monitor Metrics

Metrics Explorer

App Insights

# SLOS - USER CONCERNS

| Reason | Benefit |
| --- | --- |
| **Align Reliability with User Expectations** | Focus on improvements users actually notice and value |
| **Avoid Over-Engineering** | Prevent excessive reliability beyond what users expect |
| **Enable Error Budgeting** | Balance reliability and new features using acceptable failure limits |
| **Measure What Matters** | Track metrics that reflect real user experience |
| **Drive Better Product Decisions** | Support informed trade-offs with user impact data |

# SLOS - AGREEMENTS IN PRACTICE

Examples of Translating User Concerns into SLOs:

| User Concern | Bad SLO | Good SLO |
|---|---|---|
| Fast response time | "Low CPU usage" | "95% of API responses under 300ms" |
| System uptime | "Server running" | "99.95% availability for end-user login service" |
| Data timeliness | "Batch job completes nightly" | "User account balances updated within 60 seconds after a transaction" |

# POP QUIZ:

You are implementing SRE principles in your organization. One of the key practices involves managing system reliability through measurable targets and acceptable failure rates.
**Which of the following best describes this SRE practice?**
A. Maximizing manual intervention
B. Embracing risk with error budgets
C. Ignoring monitoring
D. Avoiding automation

# POP QUIZ:

You are implementing SRE principles in your organization. One of the key practices involves managing system reliability through measurable targets and acceptable failure rates.
**Which of the following best describes this SRE practice?**
A. Maximizing manual intervention
**B. Embracing risk with error budgets**
C. Ignoring monitoring
D. Avoiding automation

# POP QUIZ:

Your team must define a reliability target based on user-facing latency. Which statement correctly distinguishes an SLO from an SLA?

A. An SLO is a contractual commitment with financial penalties; an SLA is an internal goa
B. An SLO measures cost efficiency; an SLA measures security compliance I
C. An SLO is an internal reliability objective; an SLA is the external agreement with customers
D. An SLA includes error budgets, while an SLO does not

# POP QUIZ:

Your team must define a reliability target based on user-facing latency. Which statement correctly distinguishes an SLO from an SLA?

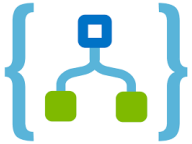
A. An SLO is a contractual commitment with financial penalties; an SLA is an internal goa
B. An SLO measures cost efficiency; an SLA measures security compliance l
**C. An SLO is an internal reliability objective; an SLA is the external agreement with customers**
D. An SLA includes error budgets, while an SLO does not

# ELIMINATING TOIL

# TOIL: DEFINITION

- Repetitive, manual, non-scalable tasks
- Automate where possible (e.g., restarts, logs, reporting)
- Aim for <50% of time on toil

Azure Automation

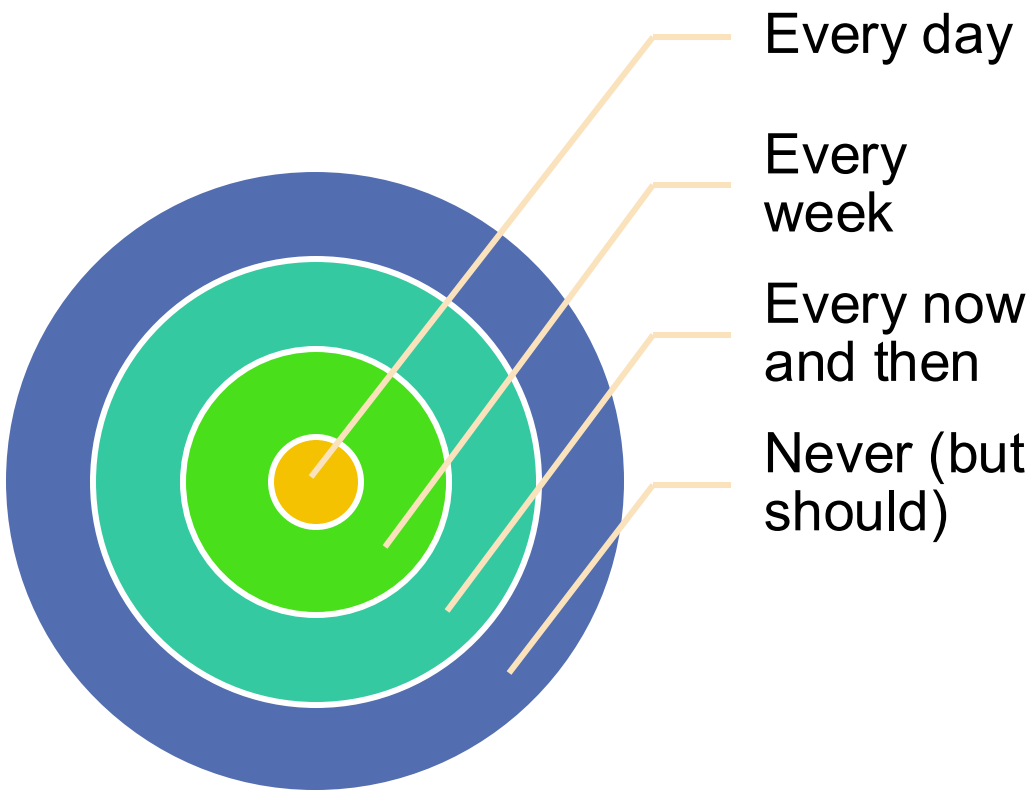


Logic Apps

# IDENTIFY TOIL

Work is toil if it is:

- Manual
- Repetitive
- Automatable
- Tactical
- No enduring value
- Linear scaling



# TIME ALLOCATION FOR ENGINEERING

- Making engineering time available
- Google sets a goal of keeping operational work (toil) below 50% of an SRE's time
- At least 50% of each SRE's time should be spent on engineering project work that will either reduce future toil or add service features
- The 50% rule ensures that one team or person does not become the "ops" (team/person)

# ELIMINATING TOIL - EXAMPLES

Examples of Toil elimination:

| Toil | Solution | Tool (Azure) |
|---|---|---|
| Manual releases | Automate deployments via external automation | Azure DevOps Pipelines, ARM/Bicep, Azure CLI |
| Manual infrastructure scaling | Use automation to perform auto-scaling | Azure Virtual Machine Scale Sets, Azure App Service Autoscale, Azure Monitor Autoscale |
| Manual password resets | Enable self-service password reset | Azure Active Directory (AAD) Self-Service Password Reset (SSPR), Azure B2C |
| Manual data extraction | Build internal query tools for user self-service | Azure Data Explorer (Kusto), Azure Synapse Analytics, Logic Apps, Power BI |

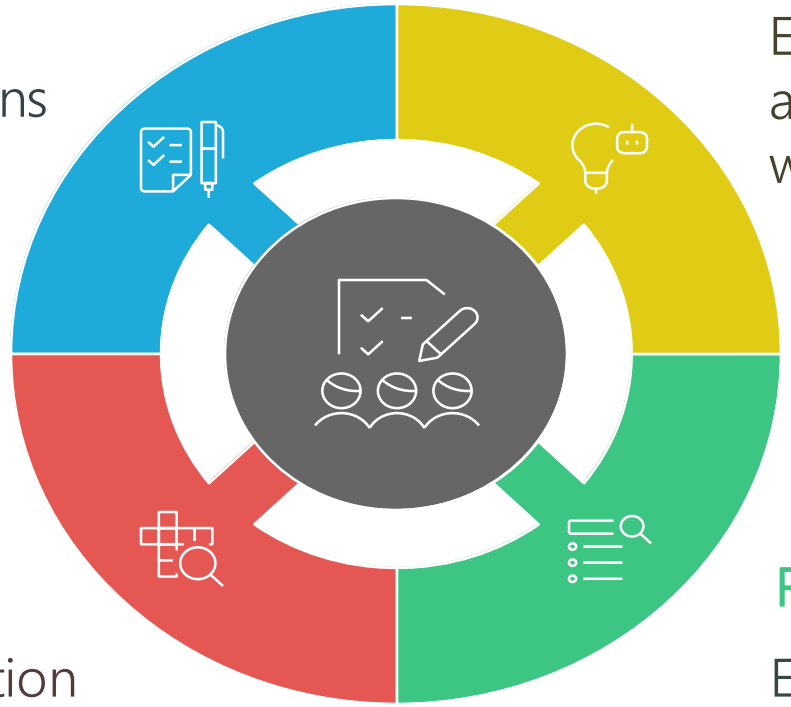
# CONTINUOUS IMPROVEMENT

## Documentation

Keeps track of automated solutions and new manual steps.

## Automation Mindset

Encourages continuous automation efforts within the team.



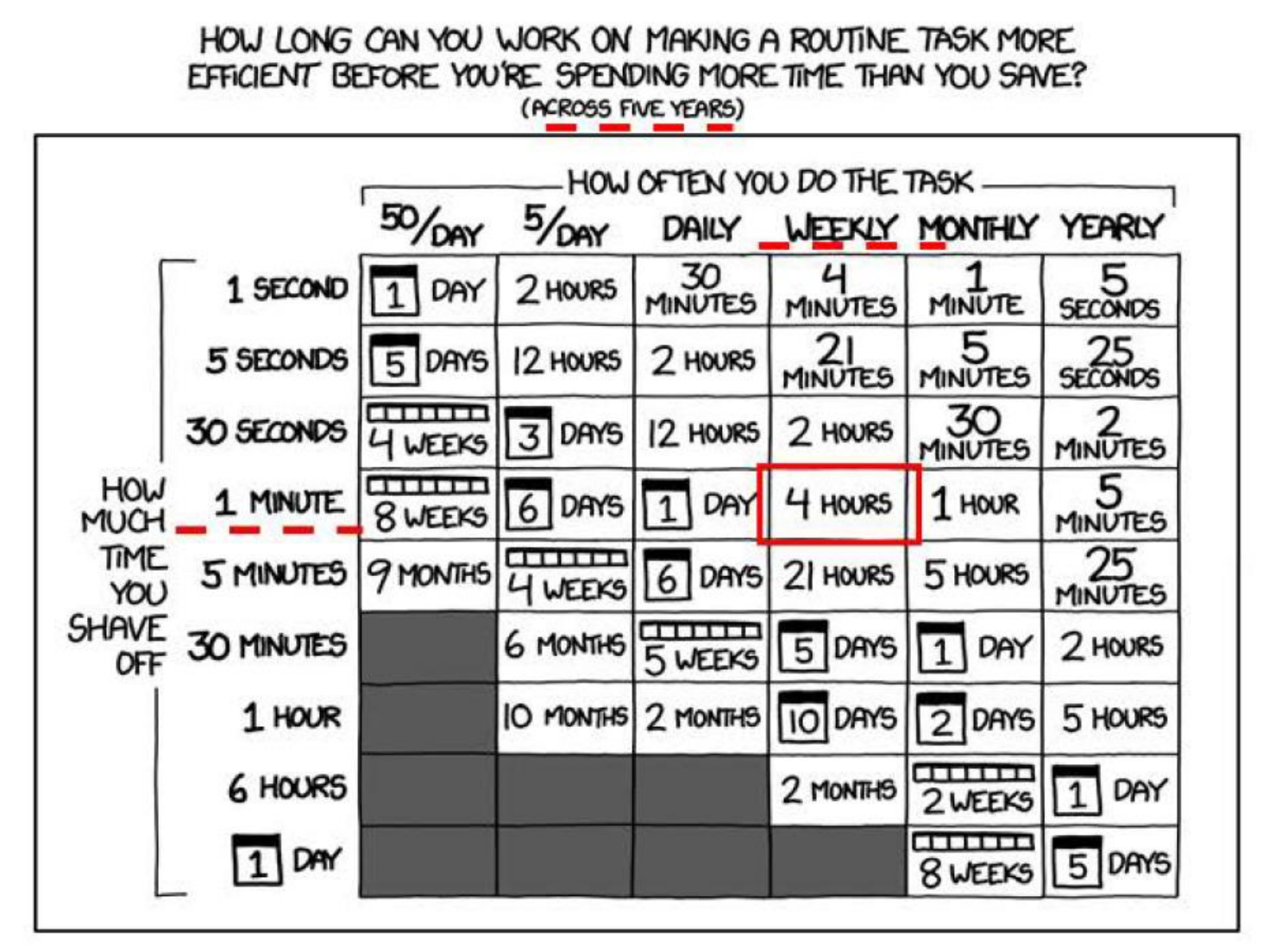
## Blameless Postmortems

Identifies automation gaps after incidents without assigning blame.

## Regular Reviews

Ensures automation processes are up-to-date and effective.

# ELIMINATING TOIL – TIME VS COST




https://imgs.xkcd.com/comics/is_it_worth_the_time.png

# ELIMINATING TOIL - BENEFITS

| | |
|---|---|
| 1. Increased Engineering Velocity | Engineers spend more time on impactful work like features and automation. |
| 2. More Time for Innovation | Engineers can focus on building new features instead of repetitive tasks. |
| 3. Increased Automation | Encourages building tools that prevent issues instead of firefighting them. |
| 4. Improved Reliability | Automated systems are more consistent and less error-prone than manual work. |
| 5. Better Incident Response | Less manual intervention during outages leads to faster, more effective recovery. |

# LAB 02 : BUILD A WEB APP WITH AZURE

Goal: In this lab, you create a Storage Account, a Flask Web App, an API and a simple deployment pipeline using github action in Azure.

## Skills Covered

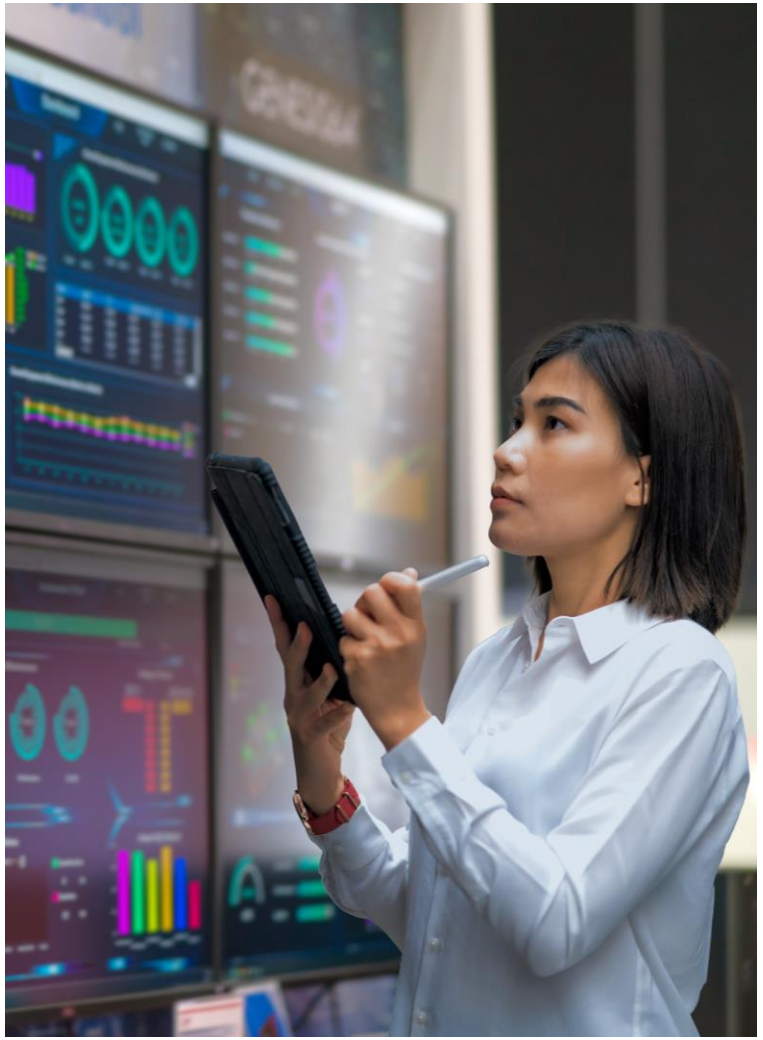
- Task 1: Create a Storage Account and Blob Containers
- Task 2: Deploy a webapp using Azure Web Server
- Task 3: Deploy an API using Azure Web Server
- Task 4: Implement GithubAction using Azure Deployment Configuration

Instructions: AZ_SRE_lab_02.md

# MONITORING

# MONITORING - WHY MONITOR

- Detect and respond to incidents quickly
- Understand system performance and health
- Identify capacity and scaling needs
- Enable root cause analysis (RCA)
- Support SLO and error budget tracking
- Drive continuous improvement

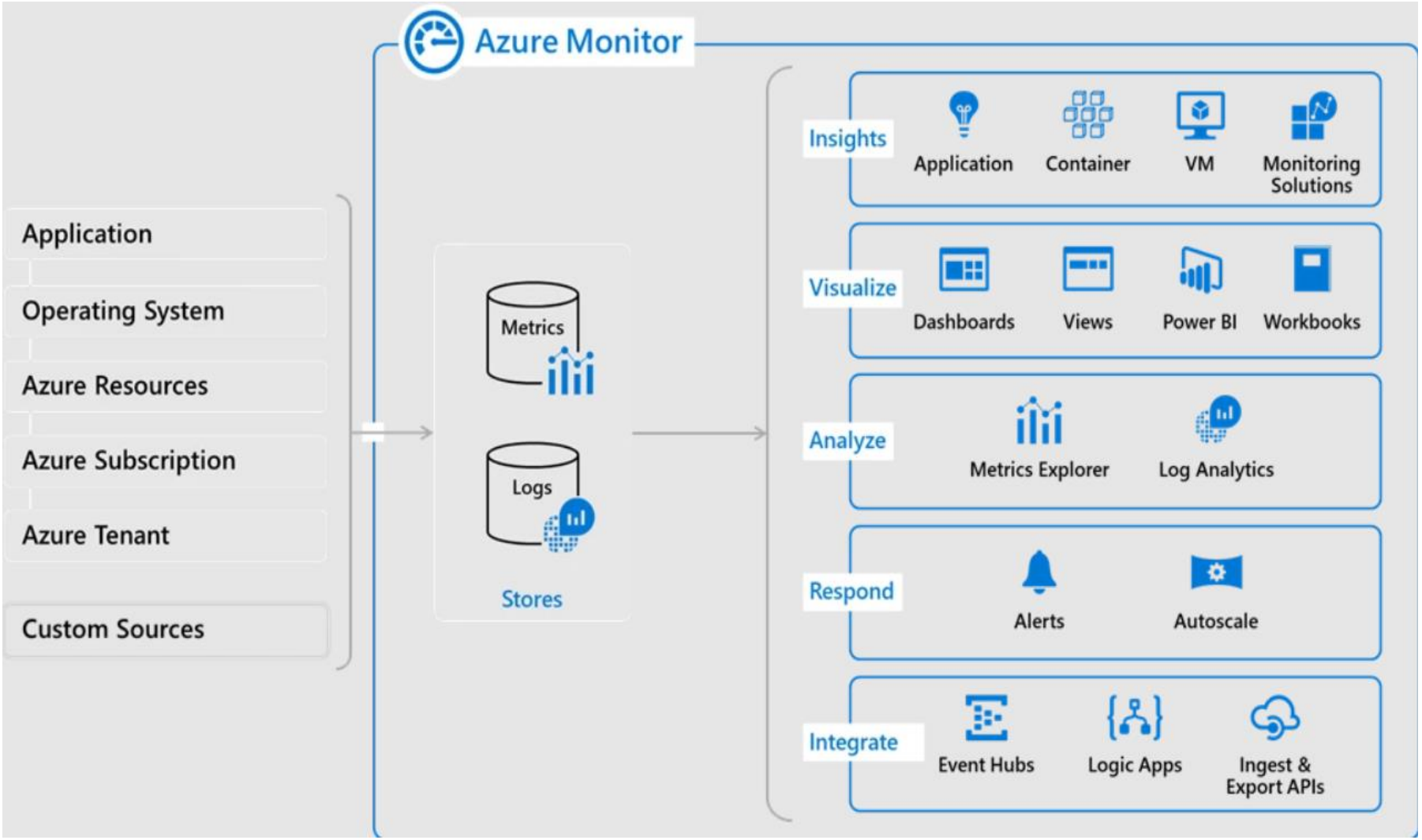


# WHY MONITOR?

| Reason | Description |
| --- | --- |
| Incident Detection | Quickly spot outages, performance degradation, and errors **before users report them**. |
| Incident Response | Provide on-call engineers with the necessary information to **triage and resolve issues quickly**. |
| SLO and Error Budget Tracking | Measure how well the system is meeting its defined **Service Level Objectives (SLOs)**. |
| Capacity Planning | Understand trends in **traffic, resource usage, and system load**, so you can scale appropriately. |
| Performance Optimization | Identify **bottlenecks**, **slow queries**, or **inefficient resource utilization**. |
| Root Cause Analysis (RCA) | Provide data for **post-incident reviews**, helping teams understand what went wrong and how to prevent recurrence. |
| Proactive Improvements | Identify **early warning signs** of system stress before they escalate into incidents. |

# WHAT SHOULD BE MONITORED?

- Infrastructure
- Application Performance
- Business Metrics
- SLIs/SLOs
- External Dependencies



# MONITORING - REASONABLE EXPECTATIONS

- Monitoring won't prevent failures, but helps detect and mitigate them quickly
- Expect some false positives and false negatives
- Balance between alert sensitivity and noise reduction
- Focus on detecting user-impacting issues, not every anomaly
- Evolve monitoring as systems and risks change over time

# WHAT MONITORING CANNOT DO

| Limitation | Impact |
| --- | --- |
| **Prevent Failures** | Monitoring highlights problems; it doesn't stop them from happening. |
| **Guarantee Zero False Positives/Negatives** | Every monitoring system has **imperfect detection sensitivity**. |
| **Catch Every Issue Instantly** | Some failures may evade detection due to **gaps in instrumentation** or **unexpected failure modes**. |
| **Replace Human Judgment** | Engineers still need to interpret data and take appropriate action during incidents. |

# BALANCING SENSITIVITY AND NOISE

- Prioritize detecting failures that impact user experience
  (e.g., availability, latency, error rate breaches)
- Not all anomalies are urgent
- Too Sensitive = Alert fatigue
- Too Loose = Missed Incidents

# MONITORING - SYMPTOMS VS CAUSES

Symptoms: What users experience (e.g., errors, slow response)

Causes: Underlying technical failures (e.g., CPU exhaustion, database down)

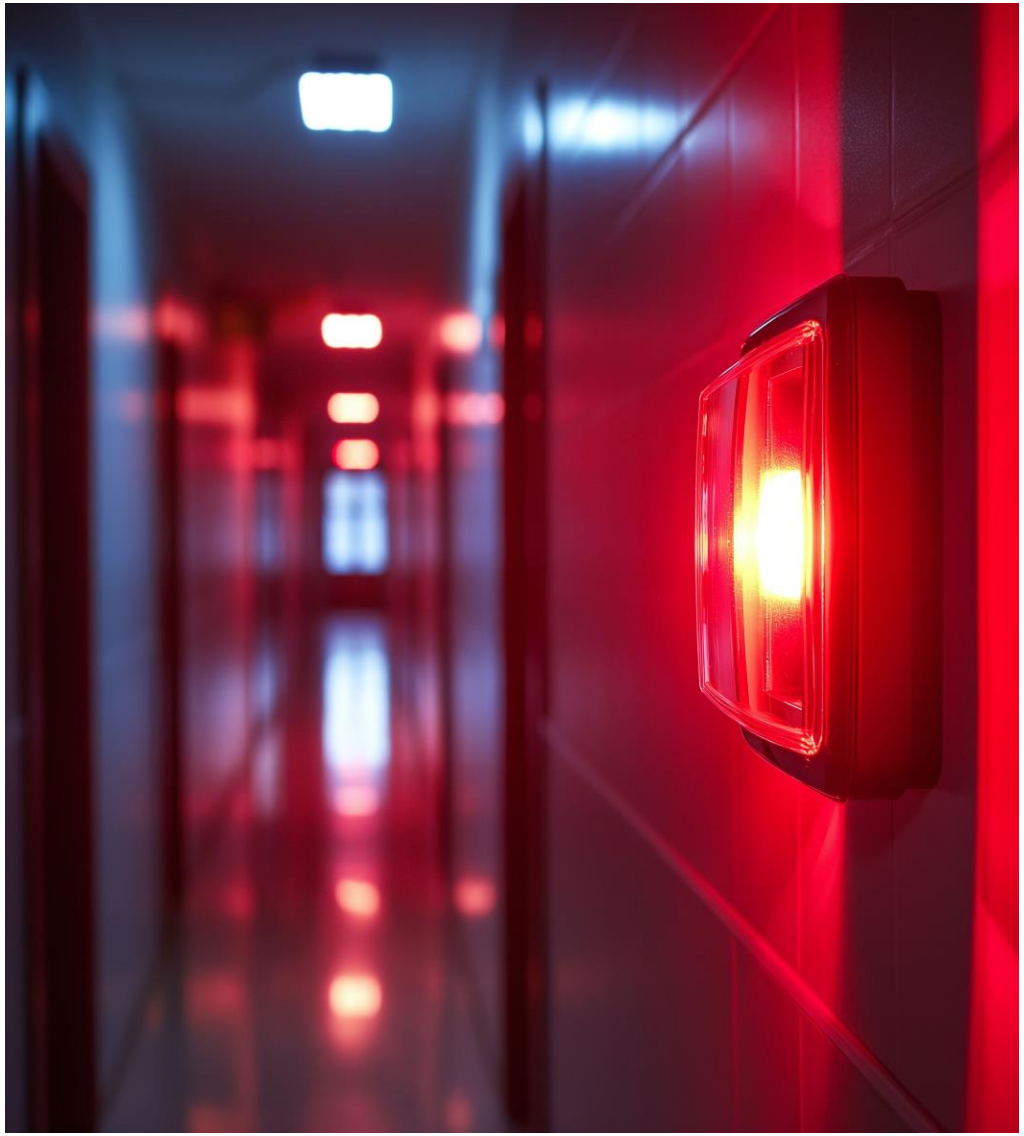
SRE monitoring prioritizes symptom-based alerts

Focus on user-impacting issues first, then investigate root causes

Helps reduce alert fatigue and false positives

# SYMPTOMS: WHAT SHOULD TRIGGER ALERTS

- **Increased Error Rates**
  HTTP 500 errors, database query failures
- **Latency Spikes**
  User requests taking longer than SLO thresholds
- **Availability Drops**
  Service not responding or failing health checks
- **Transaction Failures**
  Failed checkouts, incomplete user workflows
- **SLO Breaches**
  Measured service objectives not being met



# CAUSES: WHAT ENGINEERS INVESTIGATE DURING TRIAGE

| Causes | Examples |
| --- | --- |
| Server CPU Saturation | Resource exhaustion causing slower responses |
| Database Connection Pool Exhaustion | Leads to application timeouts |
| Memory Leaks | Resulting in out-of-memory (OOM) errors |
| Network Failures | Causing dropped requests or unreachable dependencies |
| Code Bugs or Deploy Issues | Triggering application crashes |

# WHAT IS BLACK-BOX MONITORING?

| | Black-box | White-box |
|---|---|---|
| **Perspective** | Looks at the system from the **user's point of view.** | Monitors the **internal components and behaviors** of the system. |
| **Focus** | **End-to-end service functionality**, regardless of internal implementation. | **System internals**: infrastructure, app-level metrics, dependencies. |
| **Data Source** | Synthetic tests (e.g., pings, HTTP probes, API calls). | Application logs, infrastructure metrics, database health, custom telemetry. |
| **Question Answered** | **"Is the service working for users?"** | **"Why is the system failing?"** |
| **Example Metrics** | Service uptime, HTTP response status, transaction success rates, overall latency. | CPU usage, memory consumption, database query times, queue lengths, error logs, request throughput. |

# MONITORING - SIMPLICITY

- Keep monitoring systems simple, focused, and maintainable
- Avoid overly complex alert rules and dashboards
- Prioritize actionable, high-signal alerts
- Reduce alert fatigue and noise
- Focus on user-impacting symptoms first

# WHY SIMPLICITY MATTERS IN MONITORING

Easy On-Call Response

Reduced Cognitive Load

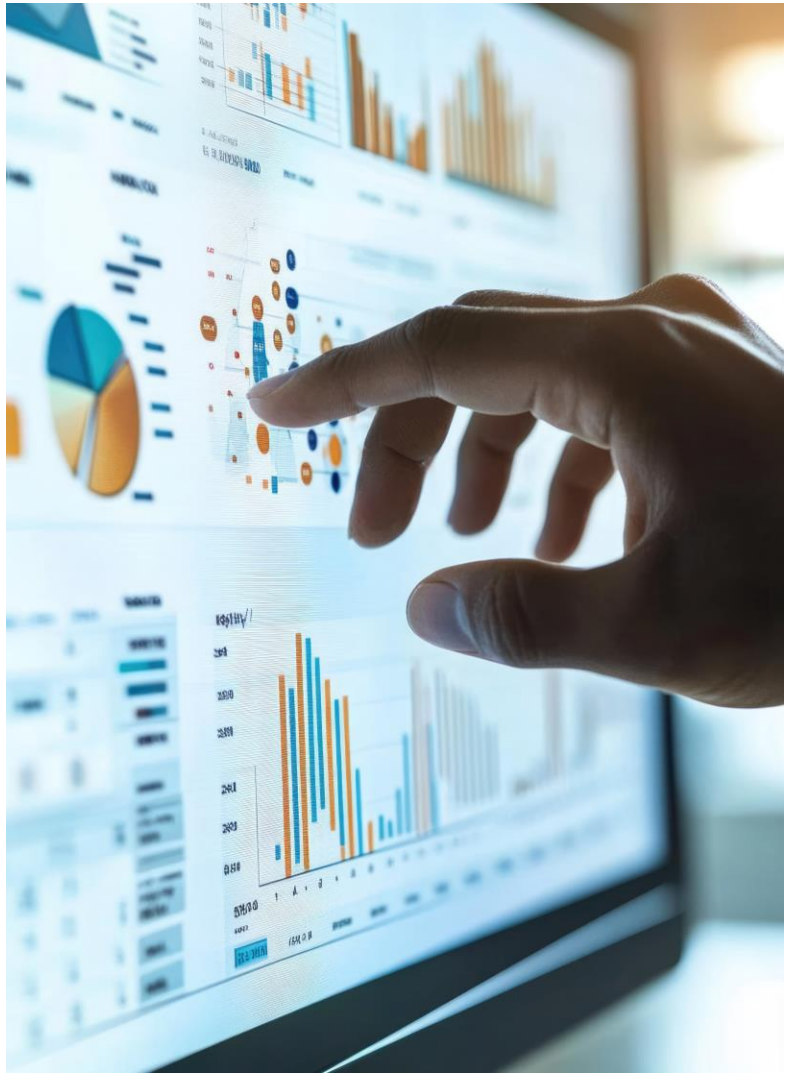
Faster Troubleshooting

Lower Operational Overhead

Fewer False Positives

# MONITORING - PREDICTABLE RESPONSES

- Alerts should trigger predefined, documented actions
- Minimize ambiguity during incident response
- Every alert should have an associated runbook or response procedure
- Reduces response time and stress during incidents
- Supports blameless culture and efficient troubleshooting



# BEST PRACTICES FOR PREDICTABLE MONITORING RESPONSES

"No alert should leave an engineer wondering, 'What do I do now?'"

| Practice | Description |
|---|---|
| Every Alert Must Have a Runbook | A short document outlining **what the alert means**, **how to investigate**, and **what actions to take**. |
| Categorize Alert Severity Levels | Define expected response times and actions based on severity (e.g., Sev-1 = Immediate page, Sev-3 = Business hours review). |
| Test Response Playbooks | Conduct regular **incident response drills** to practice real-time execution. |
| Use Standardized Alert Naming and Formatting | Helps engineers quickly understand the nature and urgency of the alert. |
| Define Escalation Paths | Know **who to contact and when**, if first-line response isn't enough. |
| Automate First-Level Actions When Possible | Example: Automatic scaling, restart, or failover based on trigger conditions. |

# POP QUIZ:

Which practice is essential for a truly blameless postmortem?
A. Focus on systemic causes and action items
B. Assign individual blame for the failure
C. Keep details of the incident confidential from stakeholders
D. Skip documenting minor incidents

# POP QUIZ:

Which practice is essential for a truly blameless postmortem?

**A. Focus on systemic causes and action items**

B. Assign individual blame for the failure

C. Keep details of the incident confidential from stakeholders

D. Skip documenting minor incidents

# ALERTING

# ALERTING - INSTRUMENTATION

**Instrumentation is the process of coding or configuring your applications to expose key operational metrics.**

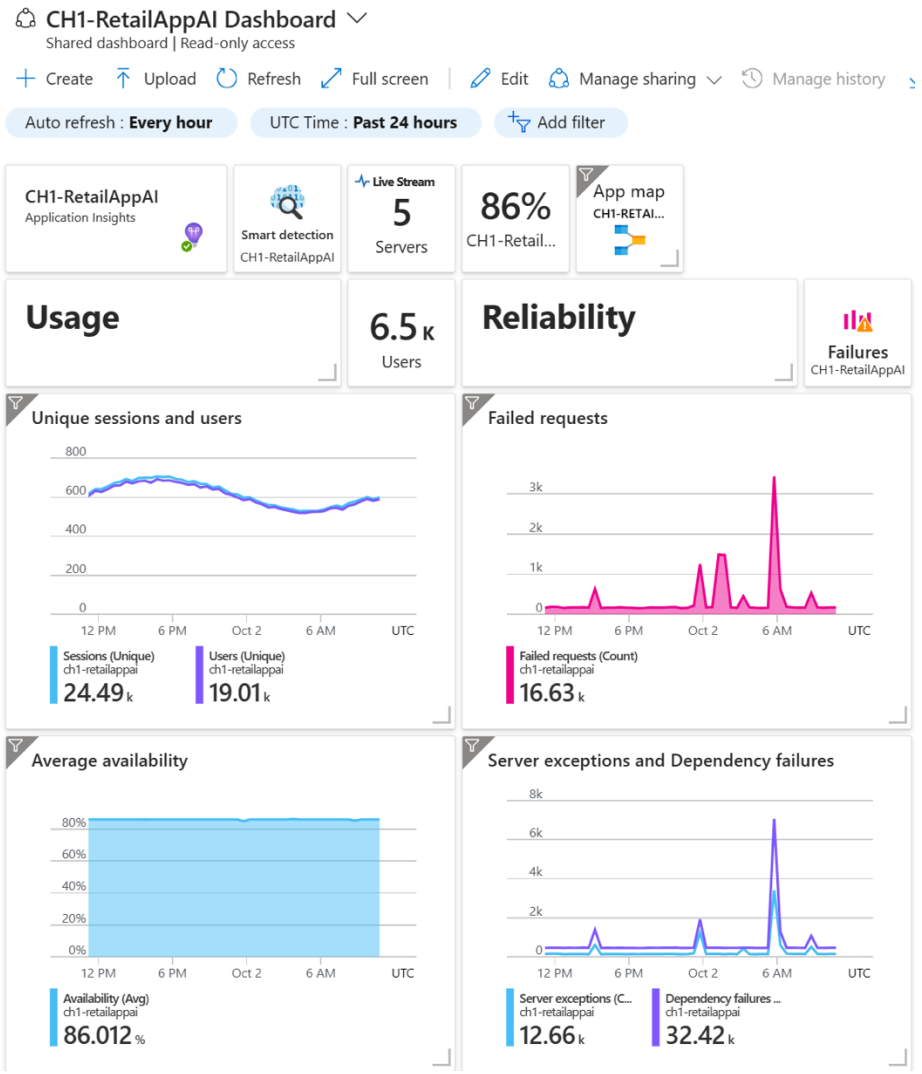
The "Four Golden Signals" from the Google SRE book:

- **Latency**: How long requests take
- **Traffic**: Volume of requests or transactions
- **Errors**: Rate of failed requests or errors
- **Saturation**: Resource utilization (CPU, memory, queue lengths)

# ALERTING – INSTRUMENTATION – IN AZURE

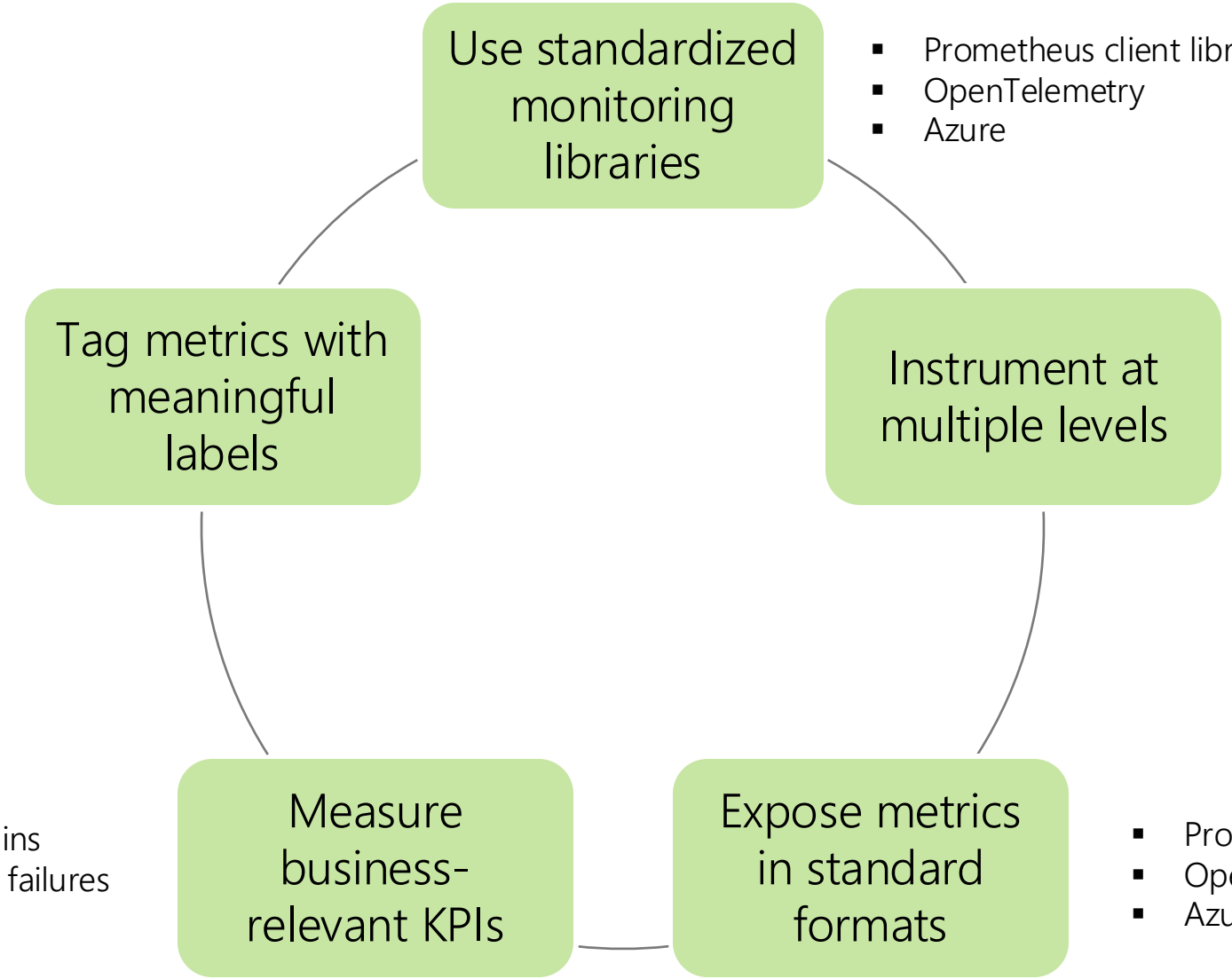
## In Azure, instrumentation can involve:

- Adding Application Insights SDK to your .NET or Node.js apps to automatically collect requests, dependencies, exceptions, and performance counters.
- Exposing custom metrics using Azure Monitor Metric API for services that don't have out-of-the-box telemetry.
- Using OpenTelemetry exporters to push metrics, traces, and logs to Azure Monitor or Log Analytics.




Example of Application Insight Dashboard

# ALERTING – INSTRUMENTATION – BEST PRACTICES




- Prometheus client libraries
- OpenTelemetry
- Azure

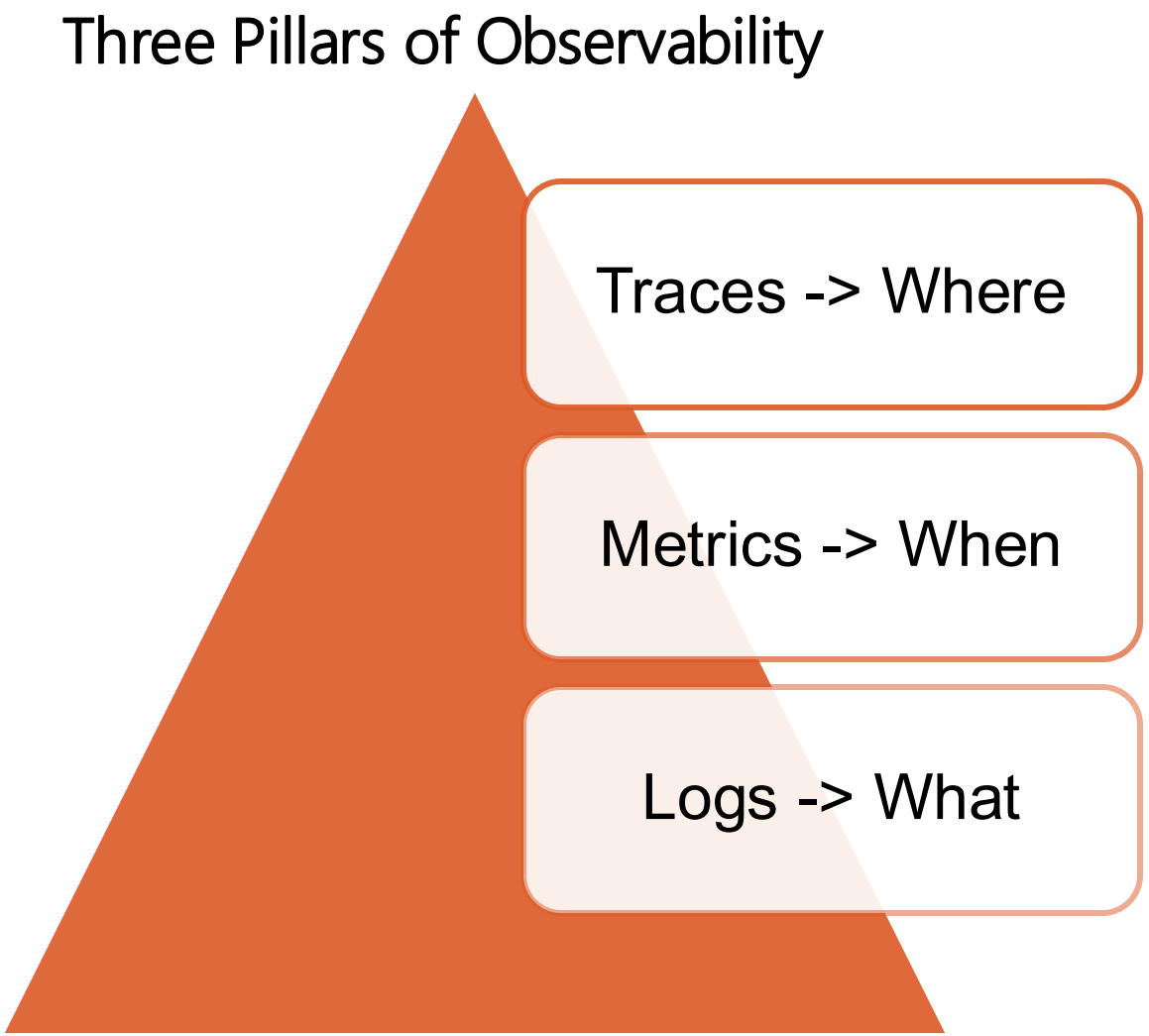
- Application level: VMs, databases
- User experience level: frontend telemetry

- Prometheus metrics endpoints
- OpenTelemetry traces
- Azure Monitor custom metrics

- Failed logins
- Checkout failures
- etc.

- API name
- Status code
- User region

# ALERTING - DATA COLLECTION




- Support pull and push data models
- Ensure data consistency, accuracy, and timeliness
- Correlate data across all three sources for alerting
- Monitor data pipeline health

# ALERTING - DATA COLLECTION - METRICS

- Definition: Numerical measurements representing system behavior over time.
- Examples: CPU utilization, Error rate, Request latency, Queue length
- Collection Model:
  - Pull-based systems like Prometheus
  - Push-based systems like StatsD, Azure Monitor Metrics API
- Role in Alerting:
  - Used for threshold-based, real-time alerts (e.g., “CPU > 85% for 5 minutes”).

# ALERTING – INSTRUMENTATION – LOGS

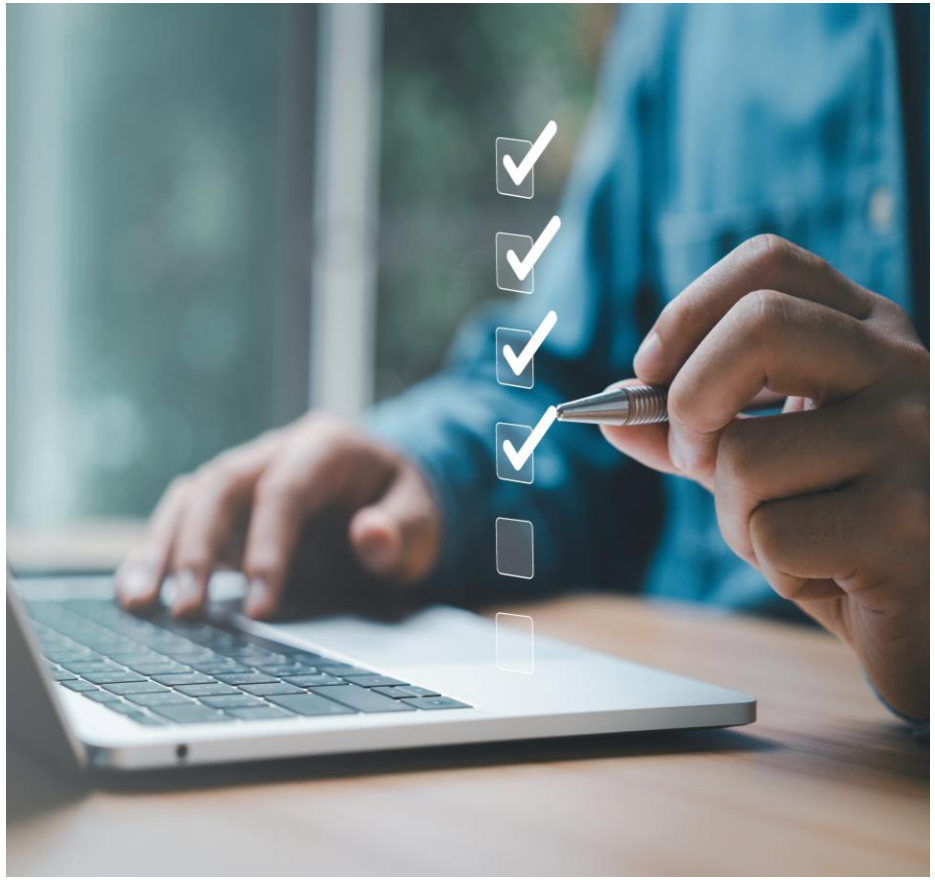
- Definition: Timestamped records of discrete events, system messages, or errors.
- Examples: Application error logs, Authentication failures, Custom audit logs
- Collection Model:
  - Usually push-based, with log agents (Logstash, Azure Monitor Log Agent)
  - Forwarding logs to a central log store like Azure Log Analytics or ElasticSearch.
- Role in Alerting:
  - Used for pattern-based or anomaly-based alerts (e.g., “More than 5 errors with ‘NullReferenceException’ in 10 minutes”).

# ALERTING – INSTRUMENTATION – TRACES

- Definition: End-to-end record of a single user or system request as it flows through distributed services.
- Examples: Distributed transaction traces, API call flow tracking across microservices
- Collection Model:
  - Captured via OpenTelemetry, Application Insights Distributed Tracing, or Jaeger/Zipkin.
- Role in Alerting:
  - Used for detecting increased latency or service dependency issues (e.g., “If 95th percentile latency exceeds SLO for service X”).

# ALERTING – INSTRUMENTATION – BENEFITS

- **Cross-Validation of Issues** : Correlate metrics spikes with error logs and slow traces.
- **Improved Root Cause Analysis** : Faster incident diagnosis by seeing metrics, logs, and traces together.
- **Full Visibility :** Cover both high-level trends (metrics) and granular event details (logs/traces).
- **Multi-Signal Alerting**: Trigger alerts that depend on multiple signals (e.g., both high error rate and specific log messages).



# ALERTING - STORAGE

- Use Time-Series Databases (TSDBs) for efficient storage
- Retain historical data for trend analysis
- Optimize for fast queries and alert evaluation
- Implement data retention and down sampling policies
- Ensure high availability and durability of monitoring data

# ALERTING – STORAGE – IN AZURE

In Azure Monitor Metrics, storage is:

- Integrated with Azure Monitor Alerts and Workbooks
- Durable and replicated across Azure regions
- Queryable using Azure Metrics Explorer or the Metrics REST API
- Supports multi-dimensional metrics with label-based filtering (e.g. cpu_usage{region="eastus", app="frontend"}.

# ALERTING – LABELS AND VECTORS

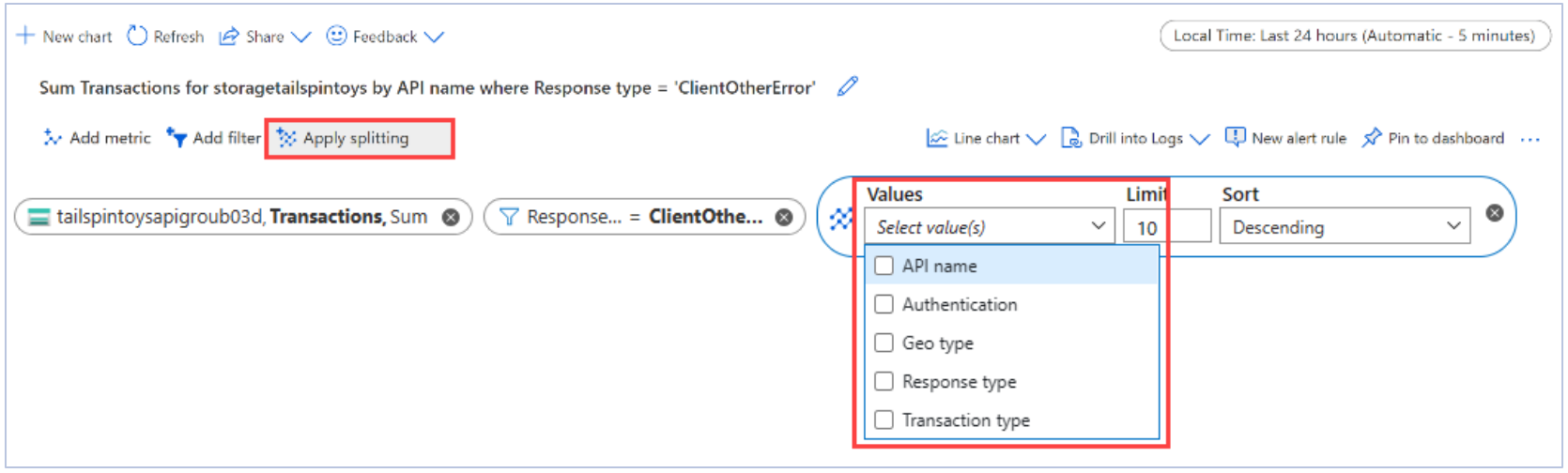
- Labels add key-value context to metrics
- Enable powerful filtering and aggregation
- Support multi-dimensional querying
- Vectors represent sets of time-series at a single point in time
- Facilitate complex and targeted alert conditions

**Metric:**
- `http_request_duration_seconds`

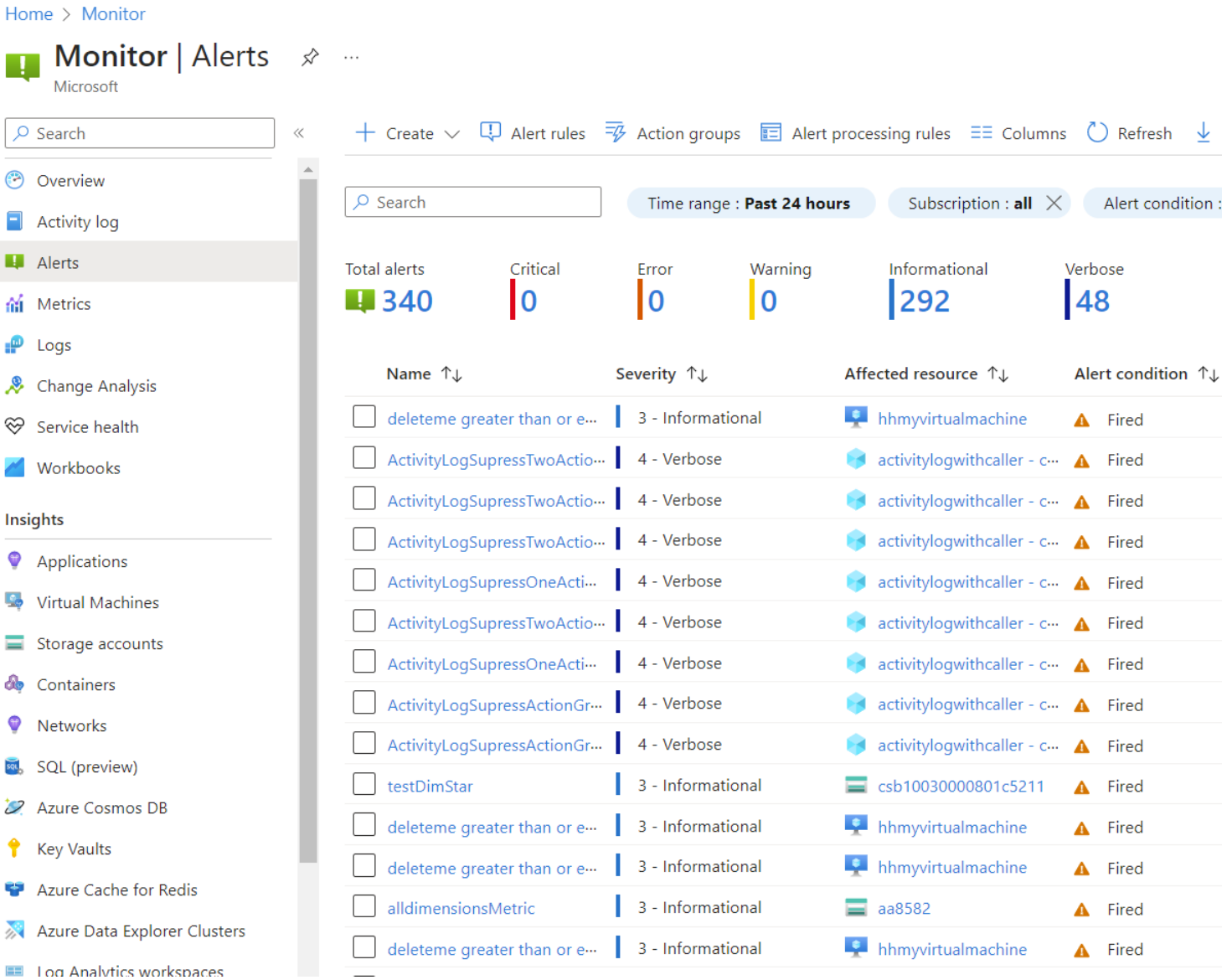
**Labels:**
- `method="POST"`
- `status="500"`
- `service="payment-api"`
- `region="eastus"`

# ALERTING – LABELS AND VECTORS – IN AZURE

- In Azure Monitor Metrics, labels are called dimensions.
- A metric like "HTTP Requests" might have dimensions for:
  - ResponseCode
  - OperationName
  - InstanceName
- You can write multi-dimensional metric alerts using these dimensions in Azure:



# ALERTING - NOTIFICATION




- Deliver alerts to the right people or teams
- Use multiple channels: Email, SMS, Incident Management tools
- Include actionable details in the alert
- Support alert deduplication and escalation policies
- Ensure reliable delivery with retry mechanisms

# ALERTING – NOTIFICATION – KEY REQUIREMENTS

## Timeliness

Notifications must reach responders within seconds to minutes of trigger

## Targeted Delivery

Only send alerts to the team responsible for the affected system (avoid flooding unrelated teams).

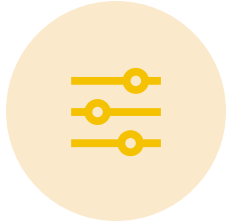
## Multi-Channel Support

Email, SMS, Slack, Microsoft Teams, PagerDuty, ServiceNow, etc.

## Actionable Content

Alert must contain enough details (what happened, where, when, recommended actions) for the responder to act quickly

## Reliability



The notification pipeline must retry on failure and track delivery status

# ALERTING – NOTIFICATION - EXAMPLE

| Element | Example |
| --- | --- |
| Alert Name | “High CPU Usage on WebServer01” |
| Severity Level | Critical / Warning |
| Trigger Time | Timestamp of alert |
| Affected Service/Component | WebAPI backend |
| Metric Details | CPU at 95% for 10 minutes |
| Recommended Actions or Runbooks | Link to troubleshooting steps |
| Link to Dashboard | URL to detailed metric graph or logs |

# ALERTING – NOTIFICATION – BEST PRACTICES

1. Suppress duplicate alerts
2. Escalation policies
3. Integrate with Incident Management tools
4. Silence non-critical alerts during maintenance
5. Use alert grouping

# POP QUIZ:

As part of improving your release engineering process, your team is implementing better instrumentation and alerting around deployments.
Which of the following practices align with effective instrumentation and alerting? (Choose two.)

A. Monitoring key service metrics (error rate, latency, and throughput) after deployment
B. Creating alerts that trigger on every minor change, even if it's not user-impacting
C. Implementing structured logging and distributed tracing to observe system behavior during releases
D. Ignoring monitoring during deployments to avoid noise from transient changes

# POP QUIZ:

As part of improving your release engineering process, your team is implementing better instrumentation and alerting around deployments.
Which of the following practices align with effective instrumentation and alerting? (Choose two.)

A. Monitoring key service metrics (error rate, latency, and throughput) after deployment
B. Creating alerts that trigger on every minor change, even if it's not user-impacting
C. Implementing structured logging and distributed tracing to observe system behavior during releases
D. Ignoring monitoring during deployments to avoid noise from transient changes

# POP QUIZ:

Your service catalog evolves rapidly. To keep alerts effective, you schedule quarterly reviews. Which practice best describes review and refinement of alerts?

A. Disable any alarms that haven't triggered in the last 30 days.
B. Conduct post-incident analyses, adjust thresholds based on real data, and retire obsolete alerts.
C. Increase alarm history retention for better audits.
D. Replace metric alarms entirely with log-based alerts.

# POP QUIZ:

Your service catalog evolves rapidly. To keep alerts effective, you schedule quarterly reviews. Which practice best describes review and refinement of alerts?

A. Disable any alarms that haven't triggered in the last 30 days.
**B. Conduct post-incident analyses, adjust thresholds based on real data, and retire obsolete alerts.**
C. Increase alarm history retention for better audits.
D. Replace metric alarms entirely with log-based alerts.

# LAB 03 : IMPLEMENT MONITORING IN AZURE

Goal: In this lab, you learn about Azure Monitor. You learn to create an alert and send it to an action group. You trigger and test the alert and check the activity log.

## Skills Covered

- Task 1: Use a template to provision an infrastructure.
- Task 2: Create an alert.
- Task 3: Configure action group notifications.
- Task 4: Trigger an alert and confirm it is working.
- Task 5: Configure an alert processing rule.
- Task 6: Use Azure Monitor log queries.

Instructions: AZ_SRE_lab_03.md

# INDIVIDUAL KEY TAKEAWAYS


- Write down three key insights from today's session.
- Highlight how these take aways influence your work.

# Q&A AND OPEN DISCUSSION


THANK
YOU