{mthree}

Monitoring

Site Reliability Engineering

Overview

Learning Objectives

In this module, we will provide a high-level overview of why we need monitoring and the ways in which we use it.

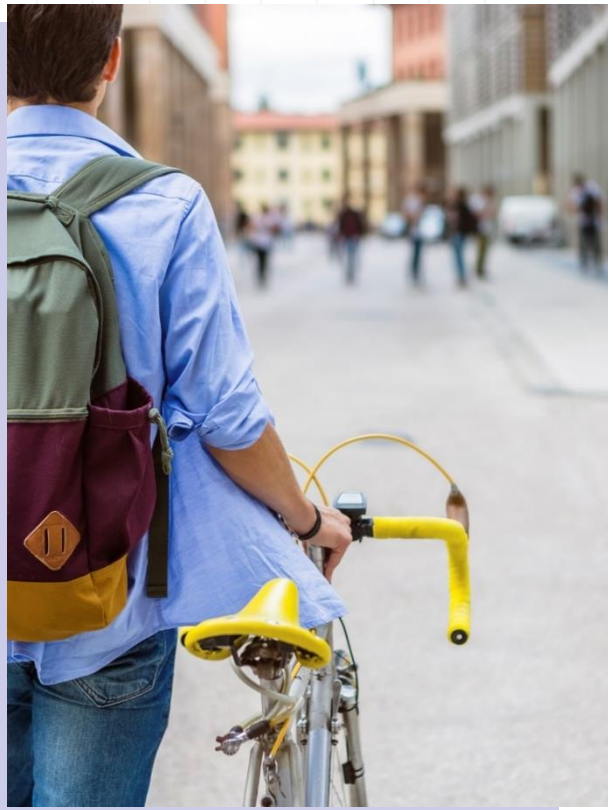
By the end of this module, you will be able to:

- Explain what it means to monitor
- Describe why it is needed and its benefits
- Describe the different layers of monitoring and types
- Explain what distributed monitoring is
- Identify the different methods systems use

What is Monitoring?

- Think of something you do every day.
  - >>> Going for a run or to the gym
  - >>> Walking or driving to a new place
  - >>> Cooking a meal or cake
  - >>> Catching a bus or train

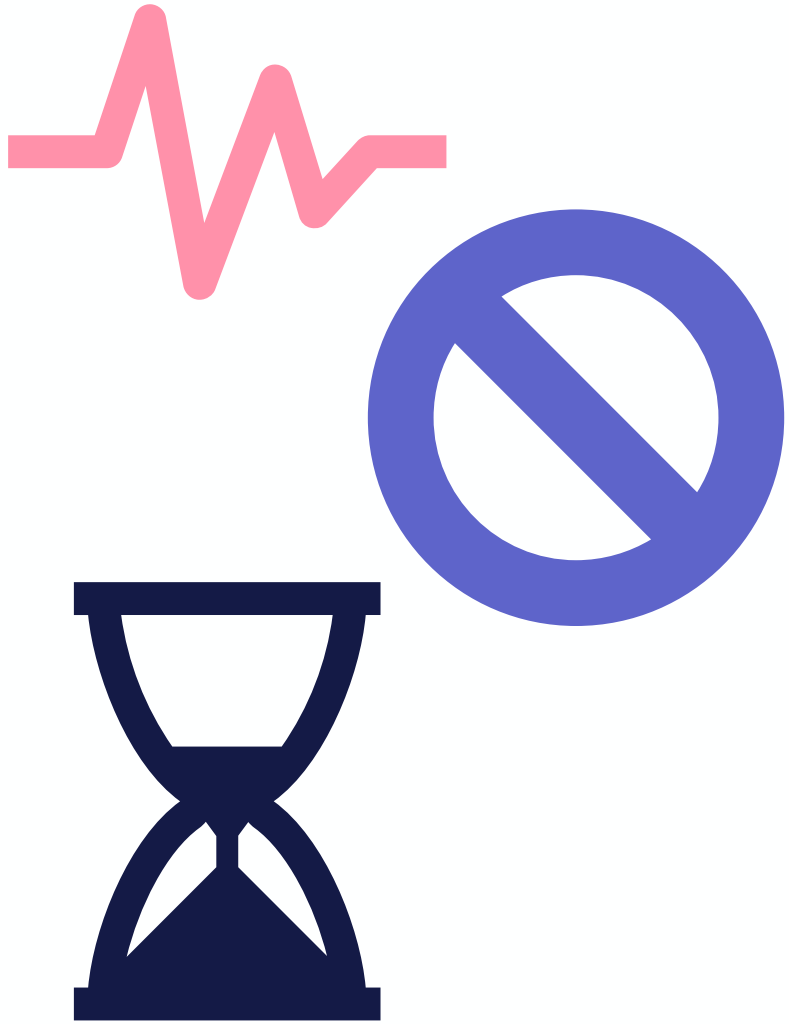
- Do you monitor anything while performing these tasks?



{mthree}

Yes, You Do

- Running or gym
  - >>> Monitor heart rate or blood pressure
  - >>> See if it is getting to the desired target
- Walking or driving to a new
  - >>> Check road signs or names to make sure you are going the right way
- Cooking a meal or cake
  - >>> Set timers
  - >>> Check weights and measures
  - >>> Check that it is cooked inside
- Catching a bus or train
  - >>> Check the platform and arrival time
  - >>> Do I have enough time to get a coffee?



{mthree}

What is Monitoring?

The collecting, processing, aggregating and displaying of real-time quantitative data about a system.

{mthree}

Why Monitoring?

- We can make executive decisions from the data
  - >>> Increase resource
  - >>> Add more storage
  - >>> Design a better process
  - >>> Fix site and system flaws
- Report findings in a timely fashion
  - >>> Dashboards to display useful charts, data and diagrams
  - >>> Alerts to email, SMS or chat
  - >>> Escalate issues quickly
  - >>> Determine if an issue is really occurring
    - ~ e.g., Did my CPU just hit 100%? Is it still there?
- Perform trend analysis through historical data
  - >>> Helps with planning and delivery of reliable systems
  - >>> Foresee potential demand through external events such as political events, or disasters
  - >>> Link to machine learning and AI systems to predict future trends
- Compare behaviours between different versions of applications or system components

{mthree}

Activity: Setting a Baseline

- A key part of monitoring is setting a baseline.
- We need to understand our system before we can make assumptions
- Let's monitor our individual heart rate for the next minute.
  - >>> Record that number
- Now let's get our heart rate going a little
  - >>> Perform an above the head clap 20x as fast as you can
  - >>> Then measure your heart rate for the next minute
- In your teams, determine the average value and record the value
  - >>> Do this for both normal and after the above the head clap
  - >>> What is the variance between you all?
- Now across teams, what is the average value of BPM?
  - >>> Do this for both normal and after the above the head clap
  - >>> How big a difference is your team value to the average

{mthree}

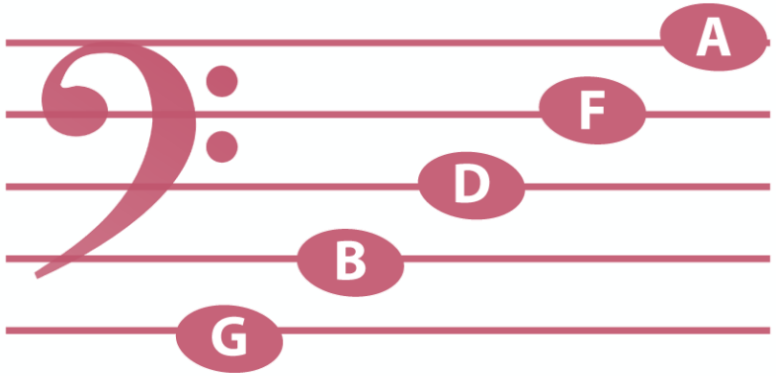
Activity Outcome

- The individual value
  - >>> Getting an idea of each part of the entire system
    - ~ Disk, CPU, memory, network, traffic, processes, thread count and so on
- The average value in the team
  - >>> Equivalent to using the same rule for all systems
    - ~ e.g., Disk usage or CPU utilization
- Across the different teams
  - >>> Equivalent to using our rule for the whole organization
- Accelerated heart rate
  - >>> We need to understand where our peak load occurs

{mthree}

Baselines

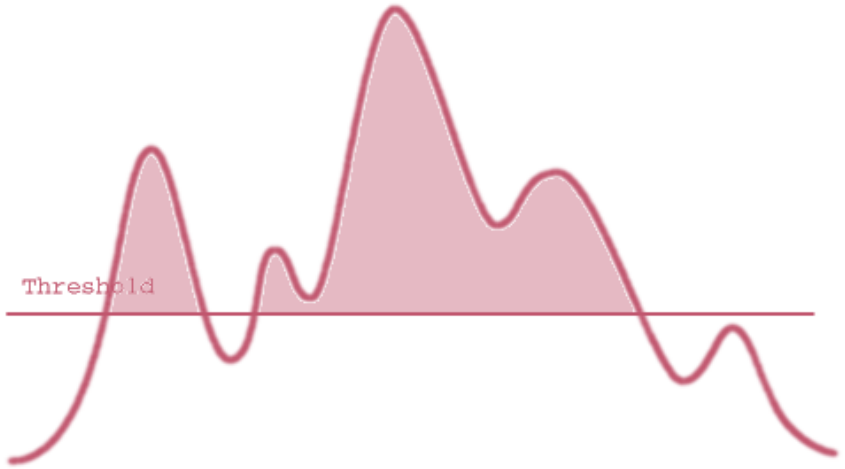
- Give us the starting point of monitoring
  - >>> Normal activity
  - >>> Increased activity
  - >>> Low activity
- Obtaining values for production on new projects
  - >>> Testing as part of Dev
    - ~ Capacity and Performance
    - ~ Playing potential scenarios to see what results you get
    - ~ Recording the outcomes as thresholds
- Using values from previous projects
  - >>> Where we have "like" usage



{mthree}

Thresholds

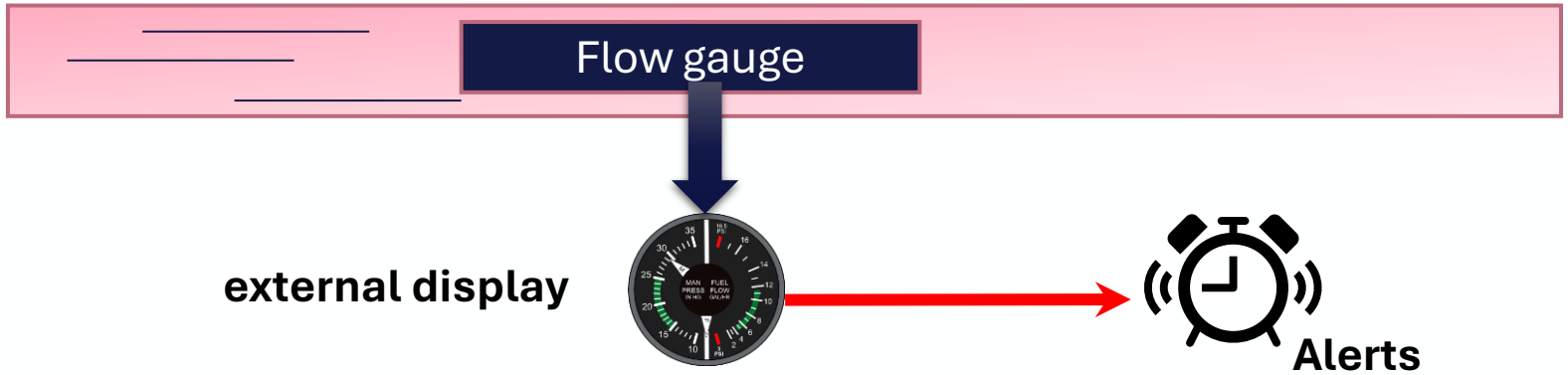
- Baselines help us set thresholds
- Setting the limits and boundaries for
  - >>> Perfect operation
  - >>> Forewarning of potential problem
  - >>> Critical when the issue is impacting the client or system



{mthree}

Observability

- As systems evolve, so does our monitoring for business impact
- Observability in an industrial system example:
  - >>> Adding a flow gauge inside a water pipe
    - ~ Like the internals of an application
  - >>> Connecting it to an external display
    - ~ The telemetry – dashboard view
  - >>> Operator can observe internal property of the system through external device
    - ~ e.g. how fast water is flowing inside a pipe
  - >>> Threshold can be set to send alerts



{mthree}

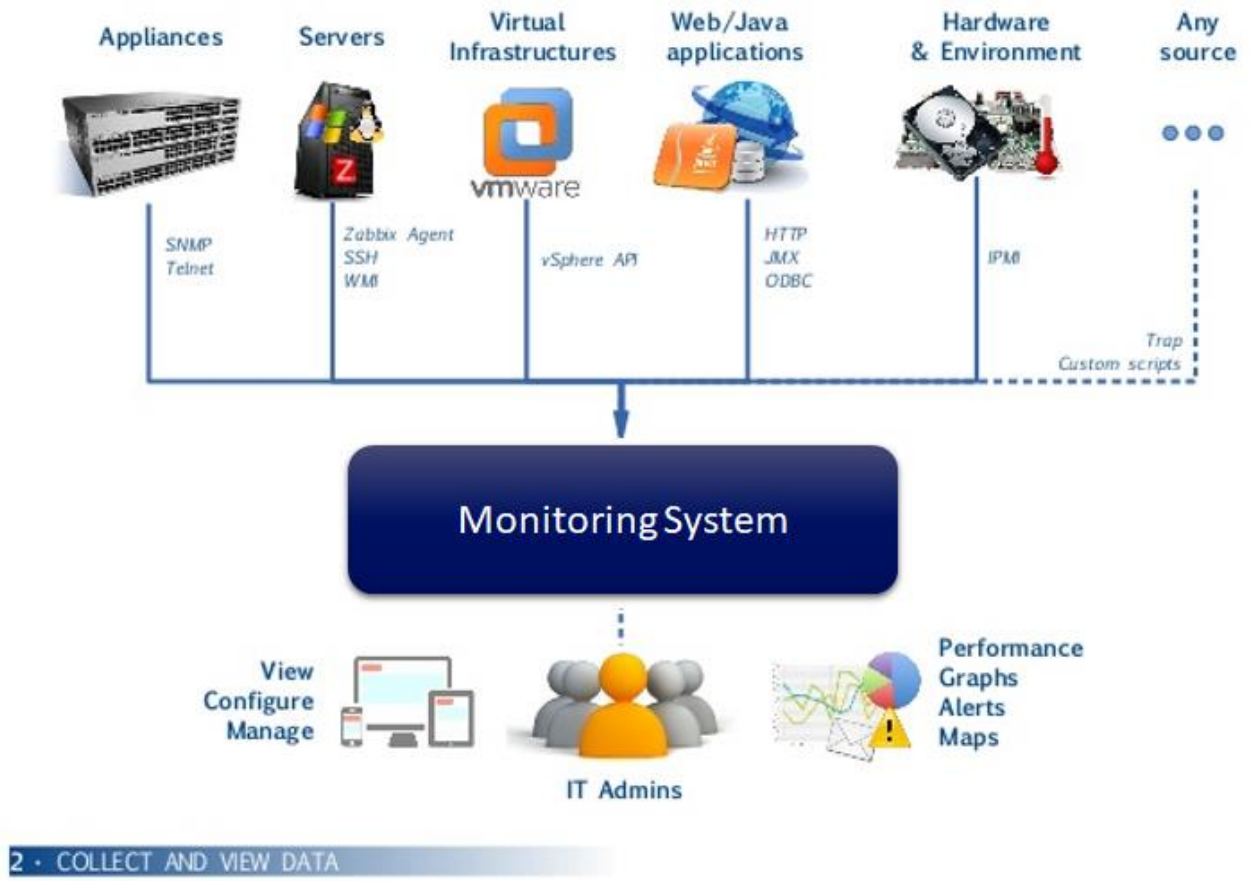
Benefits of Monitoring

Business	Technical/Project	Both
• Reputation • Keeping the user happy • Audit compliance • Discover user trends	• Pre-empt failure • Security threat detection • Early problem detection • Notification and visualization • Performance analysis	• Historical trend analysis and future prediction • Meeting service agreements and objectives • Planning and budgeting

{mthree}

Classic Monitoring

- System resources
  - >>> Hardware components
  - >>> Operating system
- Network components
  - >>> Switches
  - >>> Load balancers
  - >>> Firewalls
  - >>> Routers
- Applications
  - >>> Internals
- Metrics, rules and thresholds
- Reactive



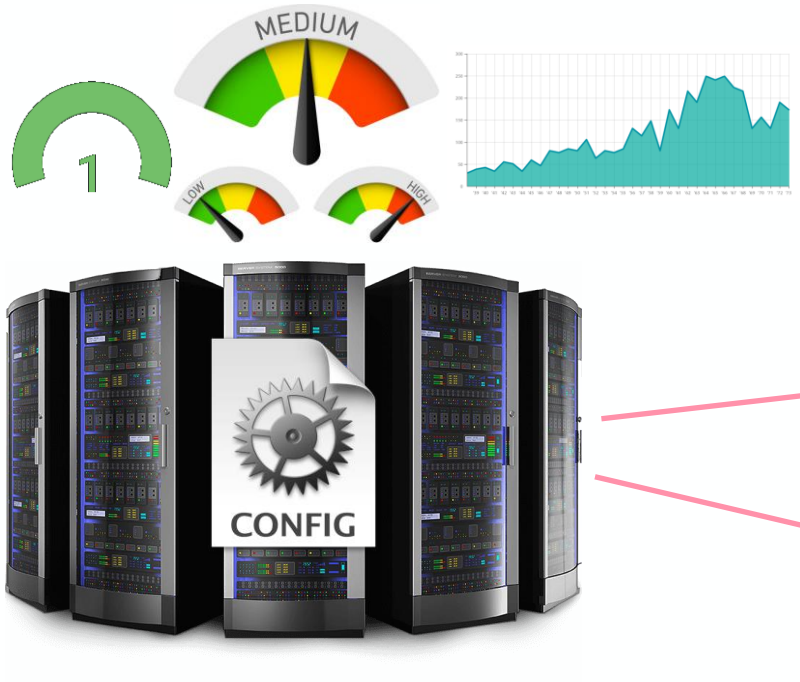
{mthree}

Modern Systems

- Centralized dashboards, collection and alerting
- Gathers
  - >>> Metric data
  - >>> Log file information
  - >>> Either by requesting or receiving
  - >>> Application and infrastructure
- Queries, calculations & trend analysis
- Extensible and pluggable
  - >>> Additional information gathering and calculations
- Low impact agents
  - >>> Gathering data from resources being monitored

{mthree}

Push monitoring systems



ITRS/Zabbix

Configuration pushed to agent

Metrics sent to the server per period of time

Netprobe/Zabbix Agent
Special agent for monitoring

Configuration defined at the server
Data stored at the server and predefined

{mthree}

Pull monitoring systems

Prometheus

Data stored at the server
Raw data time series
Key/Value

Metrics requested via http(s) on a polling basis

Application defines metrics

{mthree}

Infrastructure vs Application Monitoring

Application

- Underlying system components
  - Operating System versions and packages
  - Hardware - Disks, CPU, Memory
  - Network components – firewalls, proxy, routers, etc
- Interested in health, load, latency and errors in logs
- Configuration changes or errors
- Monitoring for potential failure based on state

Infrastructure

- Dependencies to other services
  - Connectivity, latency, availability
- Internals
  - Code performance and errors
  - Function/method speed
- Observability of how customer is seeing the application
  - Technical and business metrics

{mthree}

Black-Box vs. White-Box Monitoring

White Box	Black Box
Modest Use	Critical Use

• Application Monitoring
• Internal metrics exposed
  • Logs
  • HTTP endpoints
  • Java metrics
  • JVM Profiling
  • JMX
• Metrics such as:
  • Number of http requests
  • SQL queries running
	• Server monitoring
	  • Standard metrics such as:
	  • Disk
	  • CPU
	  • Memory
	  • Other metrics
	  • Network switch traffic/access
	  • Load balancers
	  • Hypervisor resource usage
	  • Hard disk and other hardware errors
	• System-oriented problems
	  • Active, not predicted

{mthree}

Aggregate Metrics

- Aggregates must be
  - >>> Meaningful
  - >>> Relevant
- What time period is relevant?
  - >>> Is an hour too short – will we trigger an alert
  - >>> What's our objective to the client
- Am I measuring in the right place?
  - >>> User latency may be hard to identify
  - >>> Where can I measure latency?
    - ~ In bound proxy/load balancers
    - ~ Do I record at all network connected points and take the sum/average
  - >>> What if I cannot pin point a particular request
    - ~ Do I take the average of all traffic for the period?

{mthree}

Calculations

- Long term view of data
- Aggregate usually sufficient to facilitate growth planning
- Detailed individual metrics useful
  - >>> May be expensive
  - >>> Impractical to store and retrieve
- Use counters
- Use percentiles
  - >>> 50th, 95th and 99th percentiles lets you see 50%, 5% and 1% of requests are too slow
- If not available
  - >>> Mean value by summing the seconds spent in requests and dividing by the number of requests
  - >>> Logging every request and computing the percentile values by scanning or sampling the log entries

BUT do they tell the truth?

{mthree}

Monitoring Production

- Is it enough just to monitor production environment?
- What if the customer was expecting the update today?
- Is our pipeline functioning correctly?
  - >>> Are all the agents on line and available
- Is the QA environment set to the correct versions
  - >>> Do we have the correct infrastructure set up?
  - >>> Are the tests up to date?
- Are the correct versions in our software repository?
- Is connectivity between Jira and Jenkins responding

{mthree}

SDLC Pipeline is Production

- Monitoring production tells us
  - >>> that our client is satisfied
  - >>> that we are meeting our reliability targets
  - >>> that we are meeting our objectives
  - >>> what's left in the error budget
- Monitoring the SDLC tells us
  - >>> that we can satisfy the clients need for new features
  - >>> that we are ready to create new releases or deploy
  - >>> how good our burn-down rate is for releasing new features
  - >>> how well our coding is meeting requirements

Client satisfaction

{mthree}

Monitoring Automation

Treat your configuration as code

- >>> Storing in source control management are common practices, with obvious benefits
  - ~ Change history
  - ~ Links from specific changes to tracking system
  - ~ Easier rollbacks and linting checks
  - ~ Enforced code review procedures
  - ~ Intent based preferable to web UI's or CRUD style API's
  - ~ Automated deployment

Encourage consistency

- >>> Centralised approach provides consistency
  - ~ Consistent set of basic metrics means you can
    - Automatically collect metrics across entire organisation provide a consistent set of dashboards
    - Any new component launched has basic monitoring
    - Everyone can access and use monitoring data not just engineers
- >>> Single framework
  - ~ Enables engineers to ramp up faster when they switch teams
  - ~ Easier collaboration during debugging
  - ~ Easily understand another team's dashboard

{mthree}

Monitoring Automation

- Encourage re-use
- Links to documentation to help resolve issues on alerts
- Segregation of duty
  - >>> System monitoring code
  - >>> Application monitoring code
- System monitoring is common
  - >>> Configurable attributes for common use
  - >>> Infrastructure and platforms responsible for code
- Application monitoring is specific
  - >>> Enable developer to include monitoring in their SCM
  - >>> Code linked to central monitoring service
  - >>> Developer responsible for application monitoring

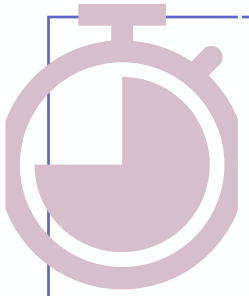
{mthree}

Distributed Monitoring

- Lots of interconnected components
- Managed by different teams
- Issues include
  - >>> Time zones (follow the sun)
  - >>> Synchronization
  - >>> Differing versions
    - ~ Operating systems, software components, application
  - >>> Permissions
    - ~ Administrative access may be required to view metrics or logs

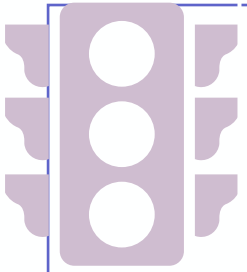
{mthree}

Distributed Monitoring – 4 Golden Signals



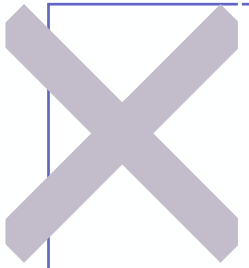
Latency

- Measure of time to complete action
  - Processing, Response, Round-trip
- Build holistic model of different performance characteristics
  - Finding bottlenecks, resources causing delays, success or unsuccessful requests



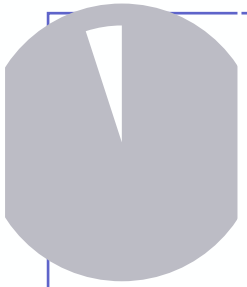
Traffic

- Capture load and demand for services
- Understand system performance
- Sustained high/low values to determine more resources or route issues
- Where data is moving



Errors

- Component health
- Determine different error types
- Identify root cause



Saturation

- Which resource is being over used – is our load balancing rule effective
- Is our memory clean up policy working
- Saturation could cause traffic delays between different layers

{mthree}

Levels of Abstraction in a System

- End-to-end experience
- External & environmental dependencies
- Collections of networking components
- Collections of servers
- Application & services
- Individual networking components
- Individual server components

Increase in Scope & Level

{mthree}

Tools that deal with distribution

- AppDynamics
  - >>> Also Cisco now
- Cisco's Dynatrace
- Datadog
- These monitor the interconnectivity, not just the app or O/S

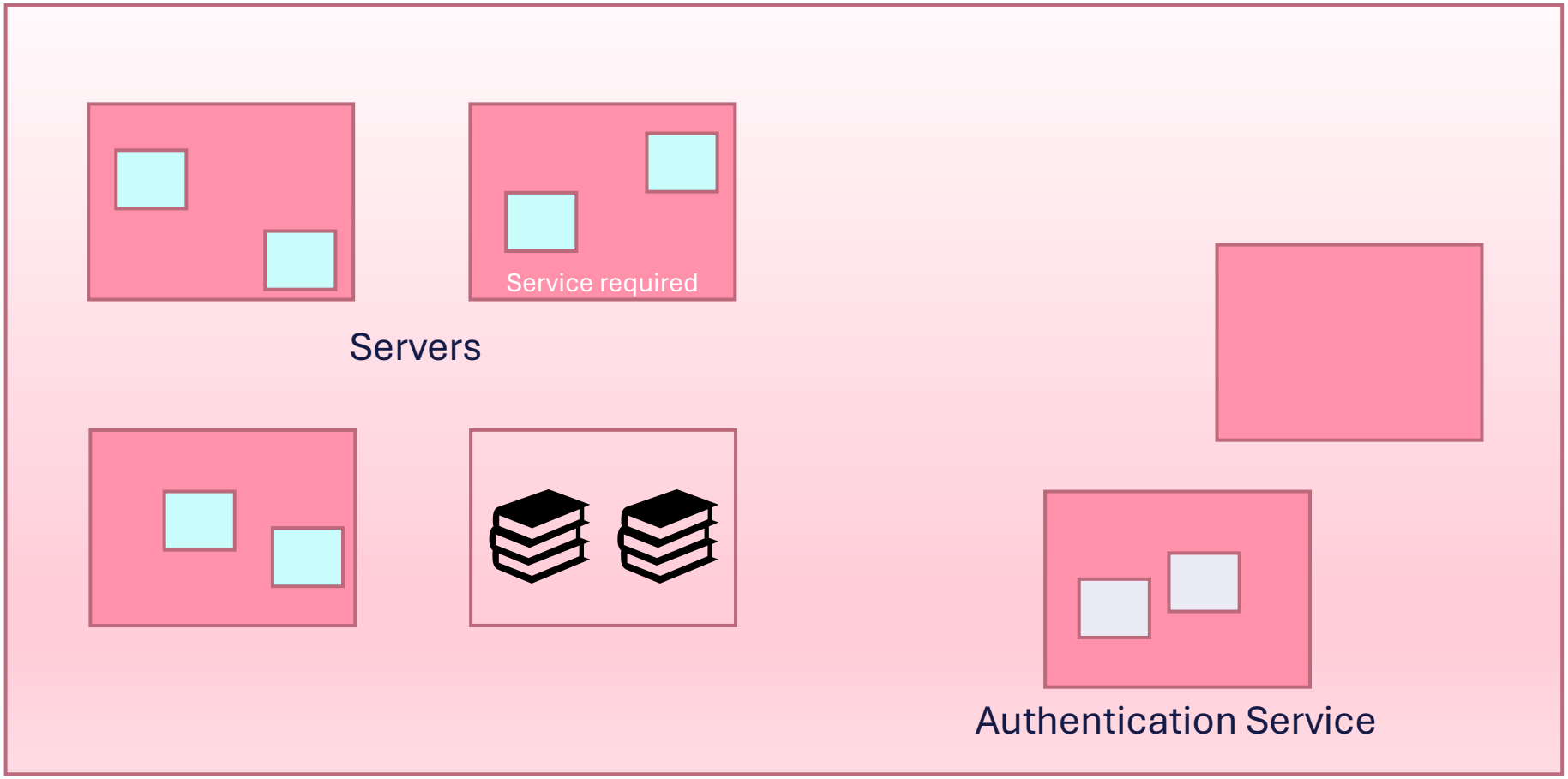
{mthree}

Distributed application tracing

- AKA Distributed request tracing
  - >>> Ability to track and observe service requests
    - ~ Collection of data as requests go from one service to another
    - ~ Helps to understand flow of requests spanning your operations being traced
- Especially used for Microservices
  - >>> Pinpoint where failure occur
  - >>> What causes poor performance

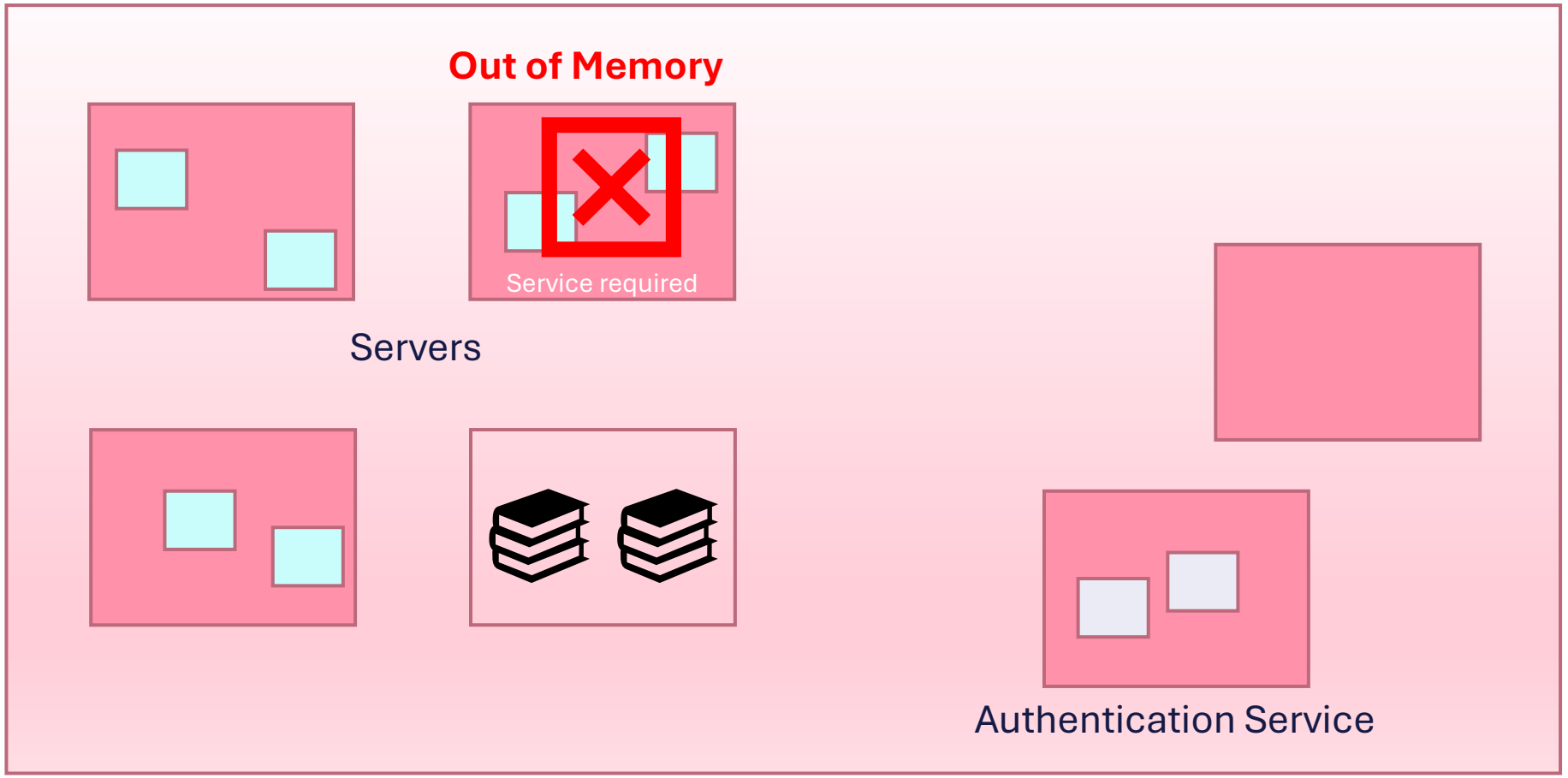
{mthree}

Distributed Request Tracing



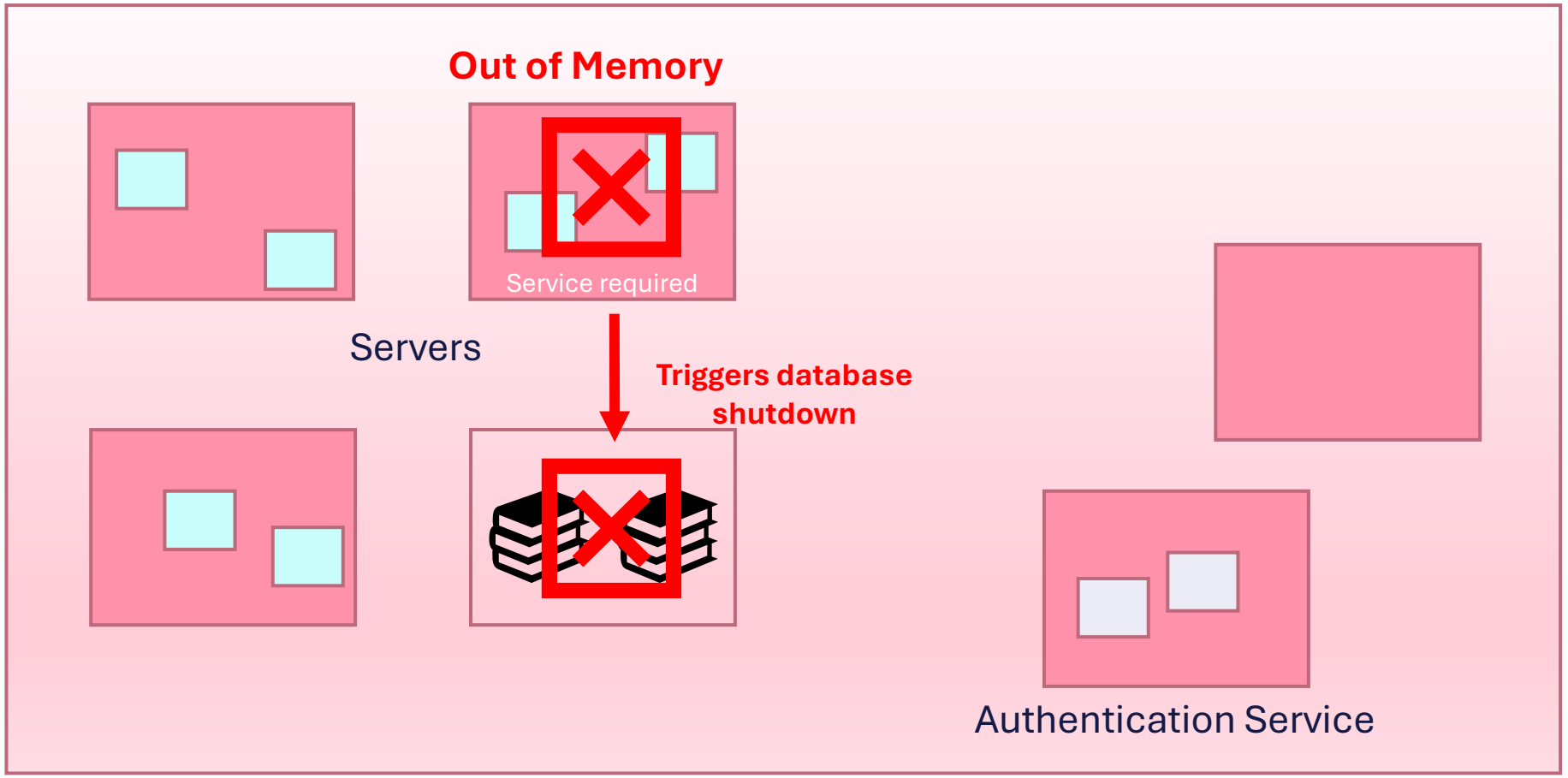
{mthree}

Distributed Request Tracing

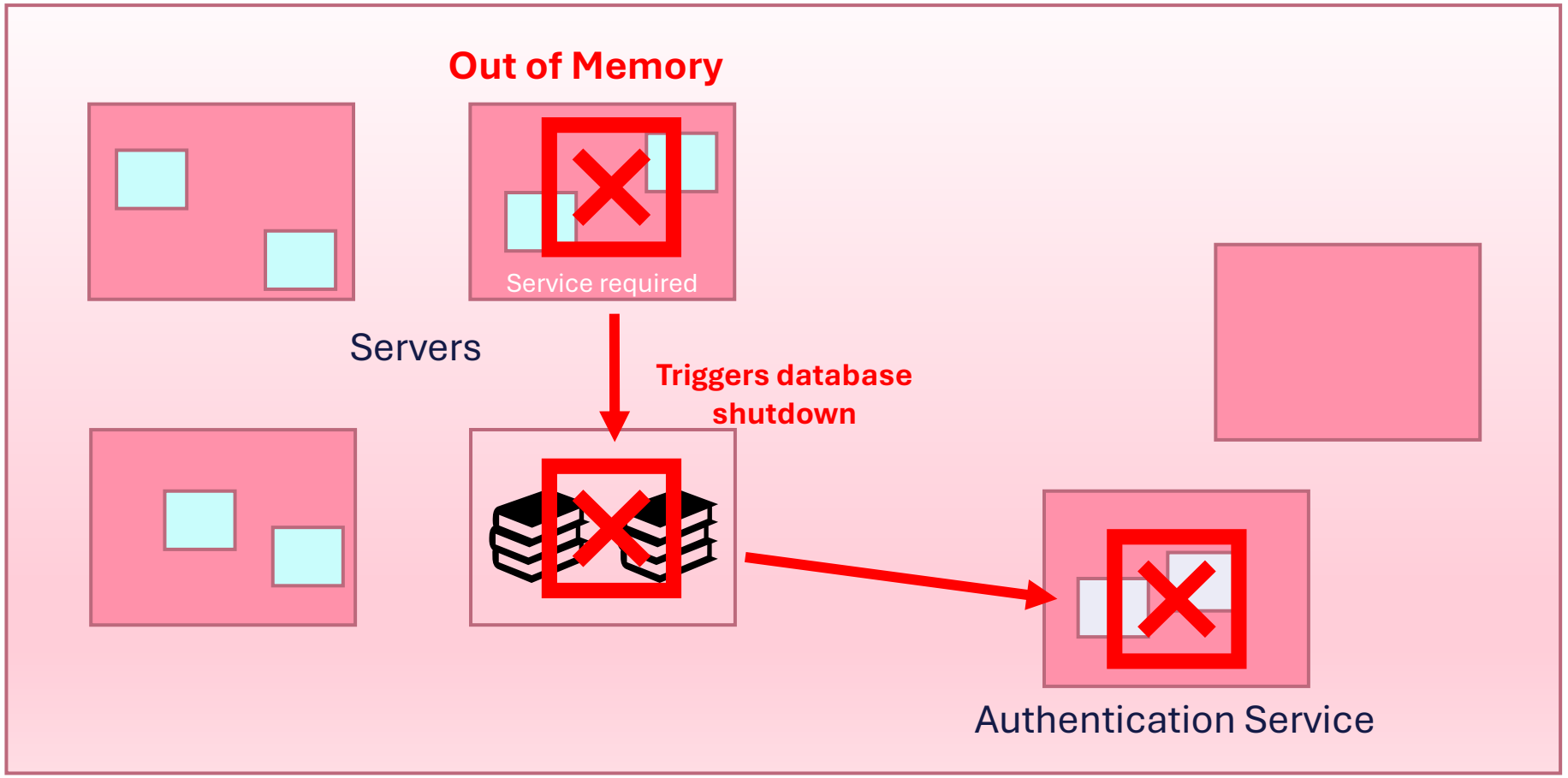


{mthree}

Distributed Request Tracing



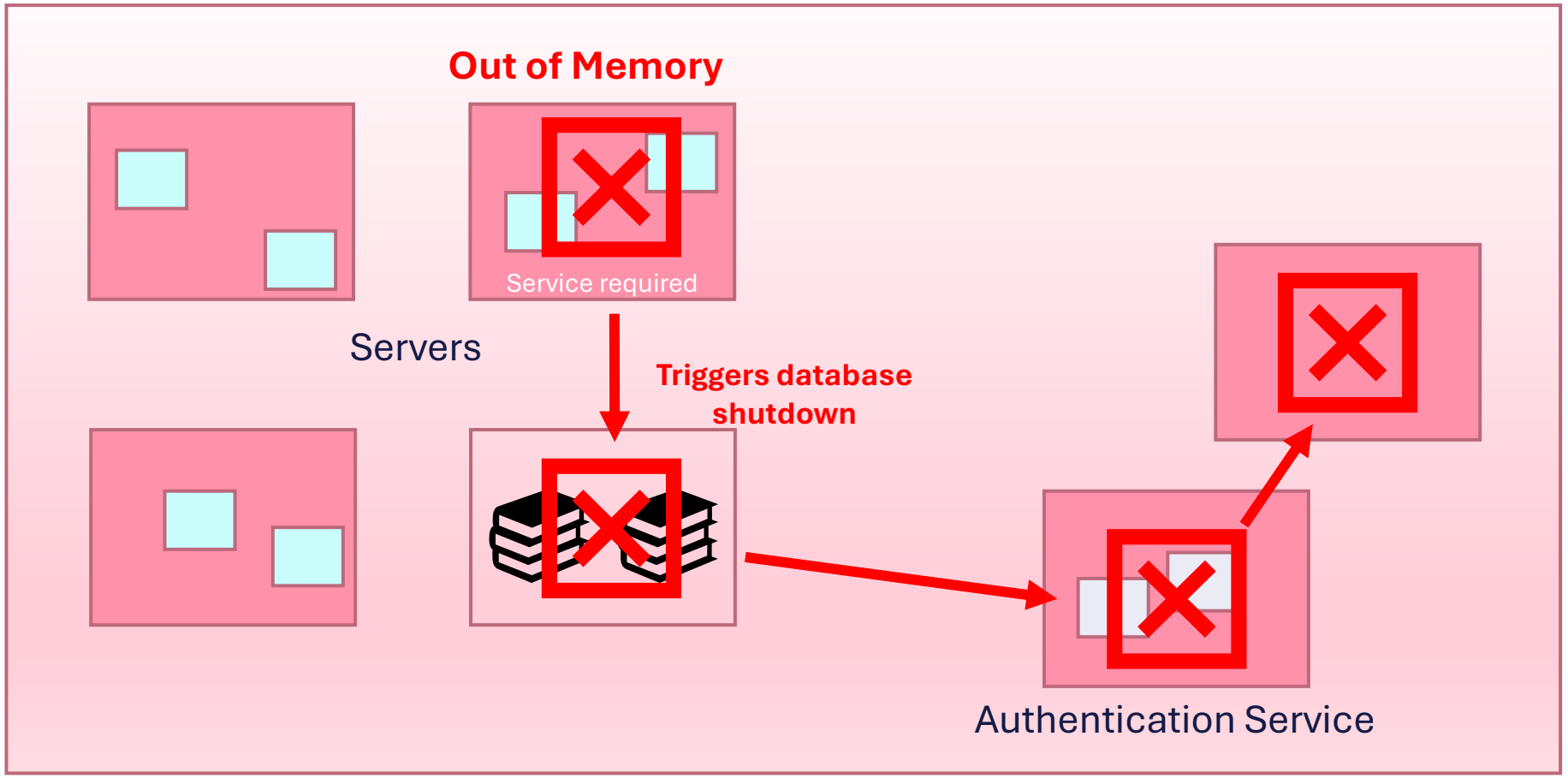
Distributed Request Tracing



Authentication Service depends on database so now can't authenticate

{mthree}

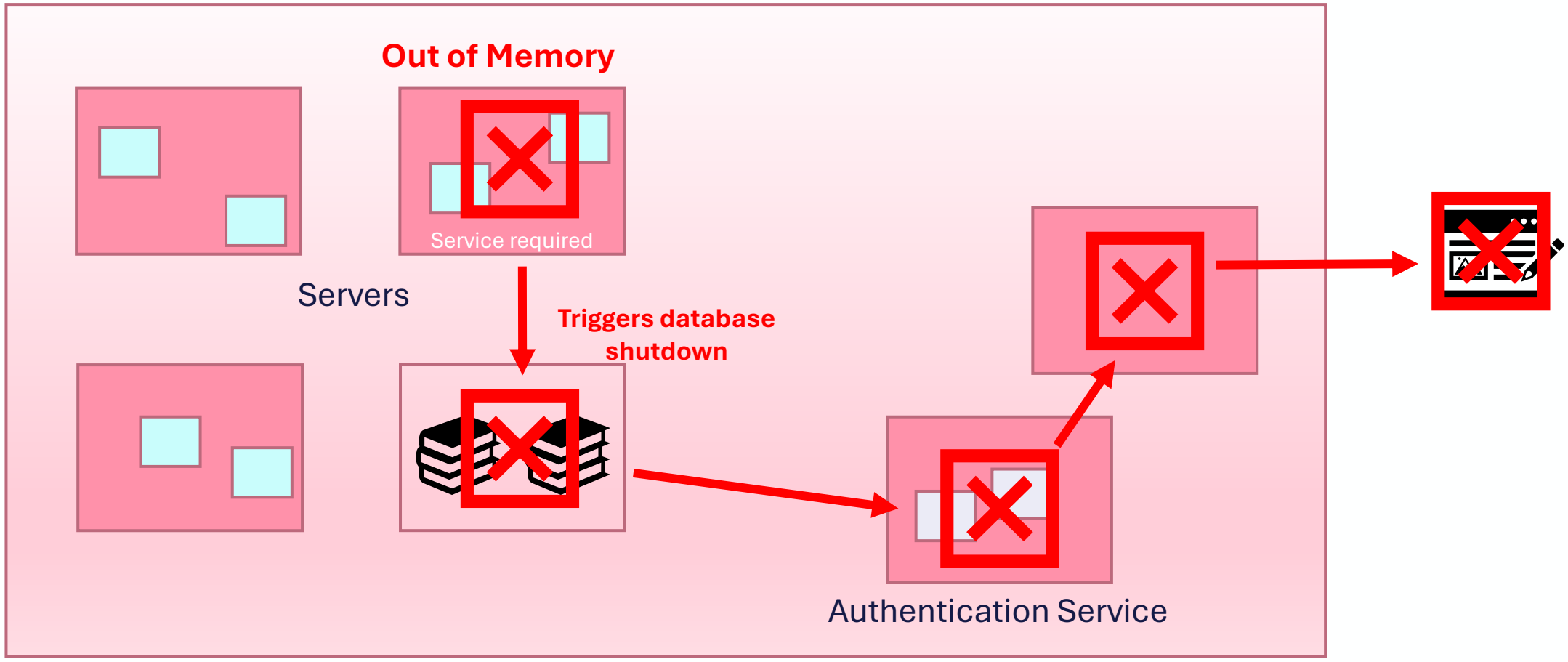
Distributed Request Tracing



Authentication Service depends on database so now can't authenticate

{mthree}

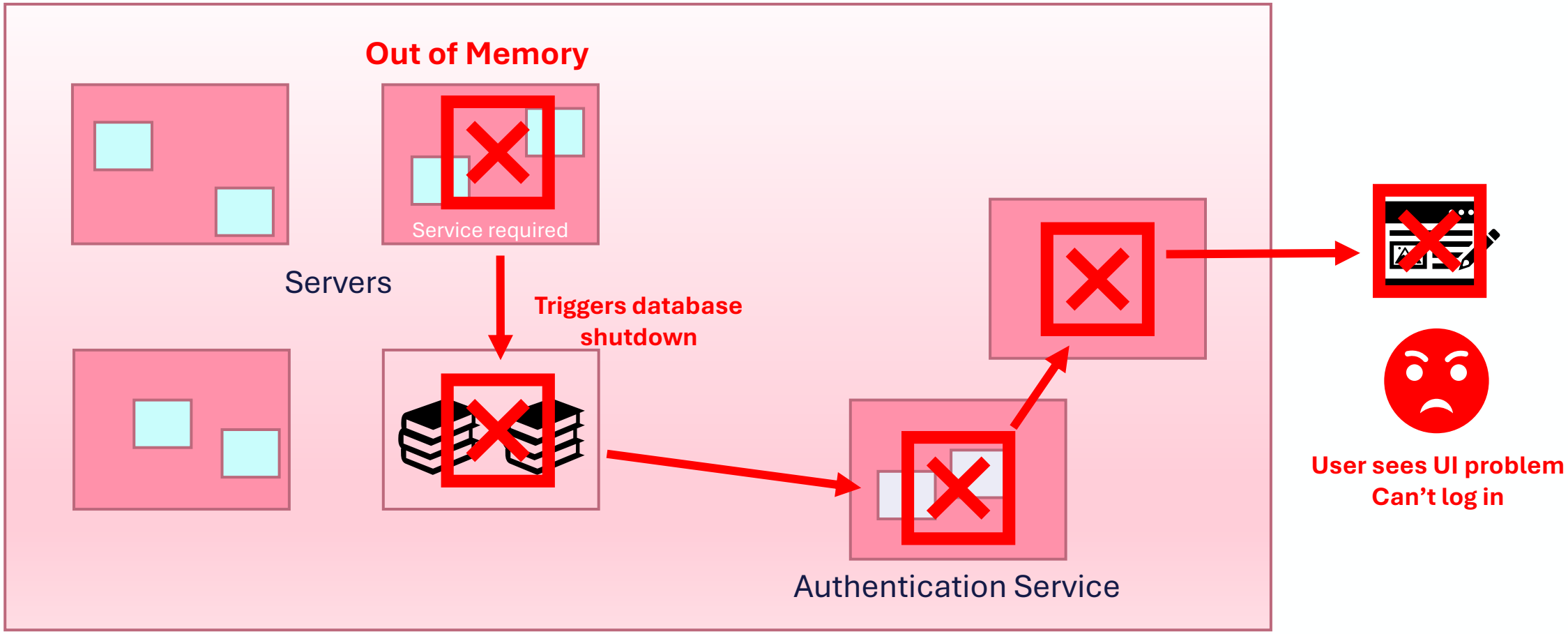
Distributed Request Tracing



Authentication Service depends on database so now can't authenticate

{mthree}

Distributed Request Tracing



{mthree}

Distributed Tracing

- Deep understanding of performance of every service
  - >>> Up and down stream
- Identify and resolve issues to minimize the impact on the customer
- Measure overall system health
- Understand the effect of changes on the customer experience
- Prioritize high-value areas for improvement
- Continuously improve monitoring to capture new trends
  - >>> Dynamic monitoring systems that detect up/down stream connectivity

{mthree}

Distributed Tracing

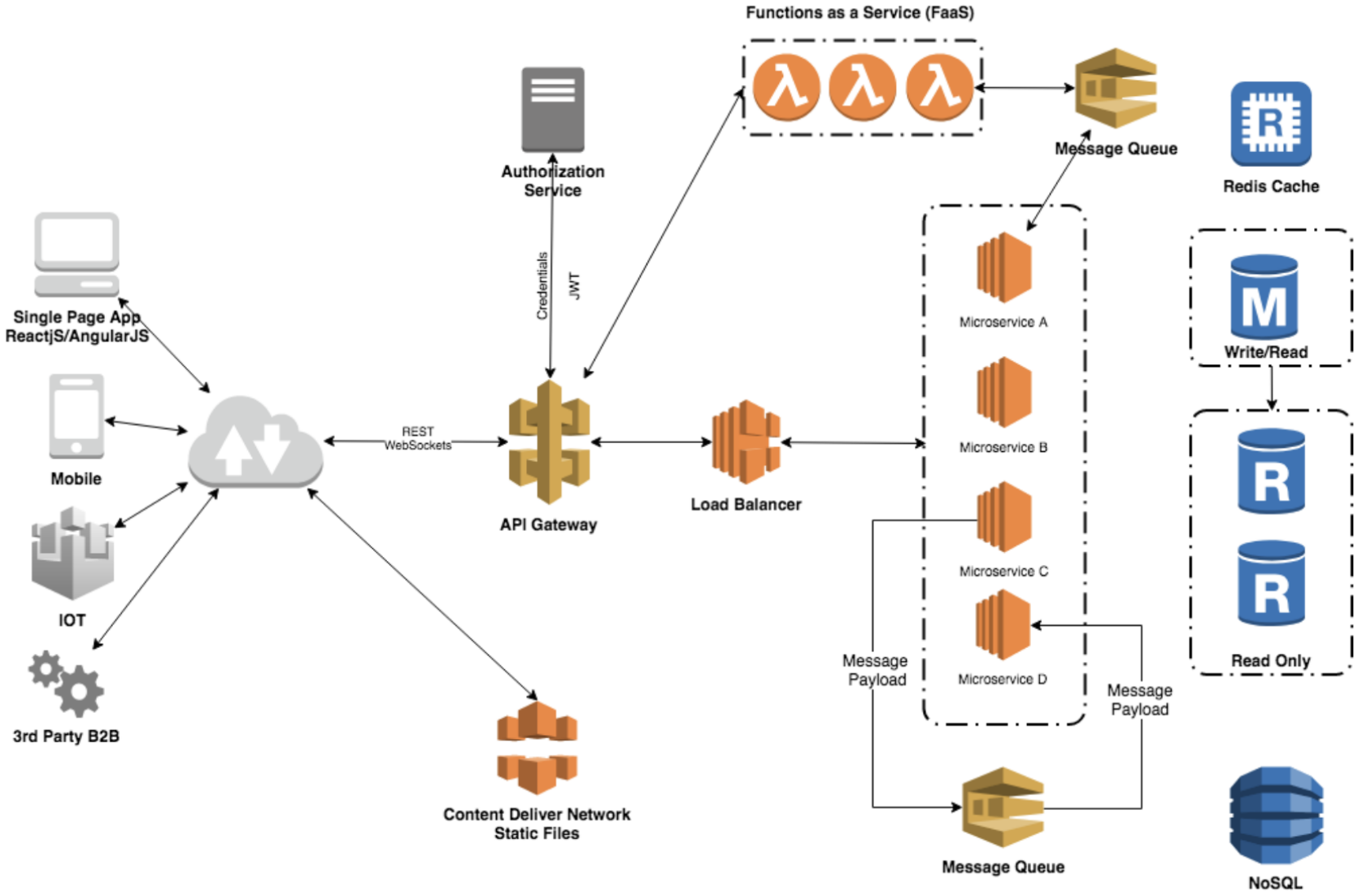
- Instrumenting environment
- Trace context
  - >>> Assigns a unique ID to each request
  - >>> Correlates each step in the correct order to allow tracking and monitoring
- Metrics and metadata
  - >>> Captures data about
    - ~ Spans, errors, duration , Custom attributes
- Analysis and visualization
  - >>> Provides context needed to derive meaning and assess action

{mthree}

Distributed Tracing

- Use distributed tracing to get answers to questions such as:
  - >>> What is the health of the services that make up a distributed system?
  - >>> What is the root cause of errors and defects within a distributed system?
  - >>> Where are performance bottlenecks that impact customer experience?
  - >>> Which services have problematic or inefficient code and need prioritizing?

Activity: Distributed Tracing



Monitoring Bottlenecks

- Avoid overloading the monitoring server
  - >>> Too many systems
    - ~ Loss of important metrics
    - ~ Loss of monitoring data
  - >>> Adjust polling times
    - ~ Spread the times across groups of servers
  - >>> Introduce pyramid system of proxy servers
    - ~ Groups of servers send to a proxy
    - ~ Proxy forwards data for hosts to central server
- Separate data retrieval from data store
  - >>> Introduce read only servers
    - ~ Dashboards and views use data from read-only replica
  - >>> Use sharded storage services
    - ~ Data spread across multiple server
    - ~ Akin to striping data across disks, but across servers instead
- Separate the service components
  - >>> Dashboard
  - >>> Storage
  - >>> Alerting
  - >>> Collecting

{mthree}

Rules for Effective Monitoring Management

- As simple as possible
- Avoid piling up the requirements
  - >>> Leads to complex monitoring systems
  - >>> Complexity introduces
    - ~ Differing latency thresholds
    - ~ Different percentiles on different kinds of metrics
    - ~ Specific dashboard components for each type of cause
  - >>> Complexity increases with time
    - ~ Monitoring system becomes fragile, difficult to change, increase in toil
- Design with simplicity in mind
  - >>> Rules to catch real incidents – simple, predictable, reliable
  - >>> Rarely used data collection and aggregation should be removed
  - >>> Rarely used features on dashboards should be removed
- SRE toil reduction methods should be applied

{mthree}

Summary

- SREs to be familiar with a service's monitoring system and features
- SREs require monitoring to define users experience of service
- Need to know
  - Where to look
  - How to identify abnormal behaviour,
  - How to find the information they need during an emergency
- Combine some source of metrics and logging in your monitoring strategy
  - Exact mix is highly context-dependent
  - Collect metrics that serve a particular purpose
    - Better capacity planning
    - Assist in debugging
    - Directly notify you of problems

{mthree}

{mthree}

Reference

- Mushero, S. (2019, August 2). Push vs. Pull Monitoring Configs. Retrieved from https://steve-mushero.medium.com/push-vs-pull-configs-for-monitoring-c541eaf9e927

{mthree}