SITE RELIABILITY ENGINEERING - Monitoring, Automation, and Reliability Practices

TEKsystems Global Services

WORKFORCE DEVELOPMENT

PARTICIPANT GUIDE

Content Usage Parameters

Content refers to material including instructor guides, student guides, lab guides, lab or hands-on activities, computer programs, etc. designed for use in a training program

1

Content is subject to copyright protection

2

Content may only be leveraged by students enrolled in the training program

3

Students agree not to reproduce, make derivative works of, distribute, publicly perform and publicly display content in any form or medium outside of the training program

4

Content is intended as reference material only to supplement the instructor-led training

REVIEW: DAY 1

Introduction to SRE and Core Principles:

- The Production Environment, from the Viewpoint of an SRE
- Principles
- SRE Software Development Lifecycle (SDLC)
- Service Level Objectives
- Eliminating Toil
- Monitoring Distributed Systems

Day 1:

Introduction to SRE and Core Principles.

Day 2:

Monitoring, Automation, and Reliability Practices.

Day 3:

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

AGENDA FOR DAY 2

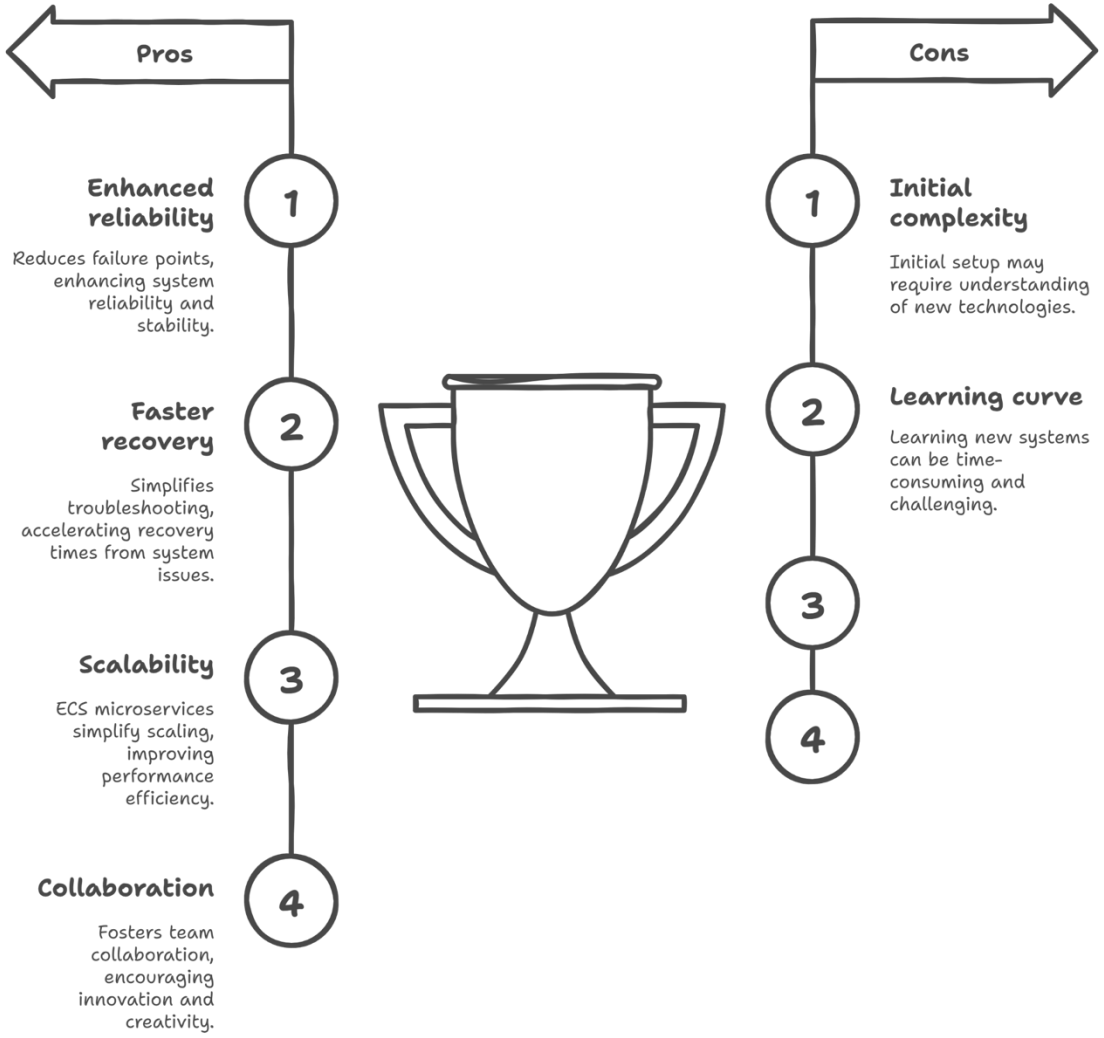
- Simplicity
- Practical Alerting
- Effective Troubleshooting
- Emergency Response
- Managing Incidents
- Tracking Outages

SIMPLICITY

SIMPLICITY IN SRE

- Simplicity enhances system reliability and maintainability.
- Reduces errors and simplifies debugging processes.
- Applies to code, architecture, and operational workflows.
- AWS tools like CloudFormation promote simplicity.
- Critical for scalable, efficient systems
- Eliminates unnecessary complexity in design and operation.
- Focuses on clear, maintainable code and processes.
- Simplifies testing and deployment workflows.
- Example: Using AWS Lambda for serverless simplicity.
- Enhances system predictability and stability.

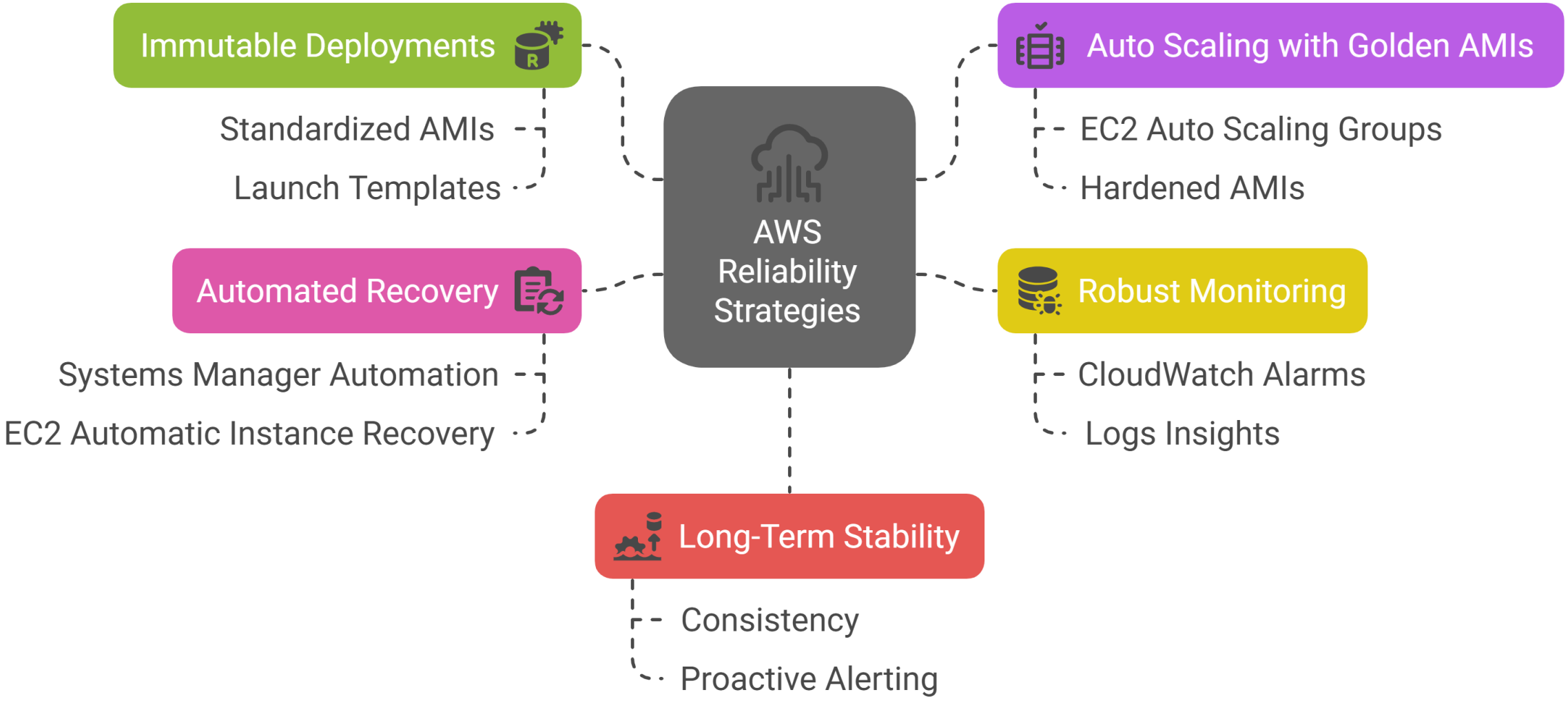
WHY SIMPLICITY MATTERS



SYSTEM STABILITY VS AGILITY

- **Reliability First**: Stability minimizes system failures and sustains high availability targets (e.g., 99.9% uptime)
- **Rapid Delivery**: Agility shortens lead times for feature rollouts and bug fixes, keeping pace with user needs
- **Golden-Signal Balance**: Track latency, errors, and throughput to align reliability with velocity
- **Automated Pipelines**: AWS CodePipeline integrates build, test, deploy, and rollback steps for both fast and safe releases
- **Risk Mitigation**: Overemphasizing stability can stall innovation; neglecting it leads to outages and unhappy customers
- **Feature Flags**: Enable gradual rollouts and instant rollbacks to reduce blast radius
- **Canary & Testing**: Embed automated and canary tests in your pipeline to validate changes before full deployment
- **Continuous Feedback**: Use monitoring insights to dynamically adjust your stability-agility equilibrium
- **Up-to-Date Runbooks**: Maintain runbooks and documentation so teams can respond predictably and onboard safely

ACHIEVING STABILITY IN AWS



THE VIRTUE OF BORING

DESIGNING BORING SYSTEMS

Infrastructure as Code (IaC)
Define and manage your AWS infrastructure using CloudFormation templates to ensure consistency and repeatability across environments.

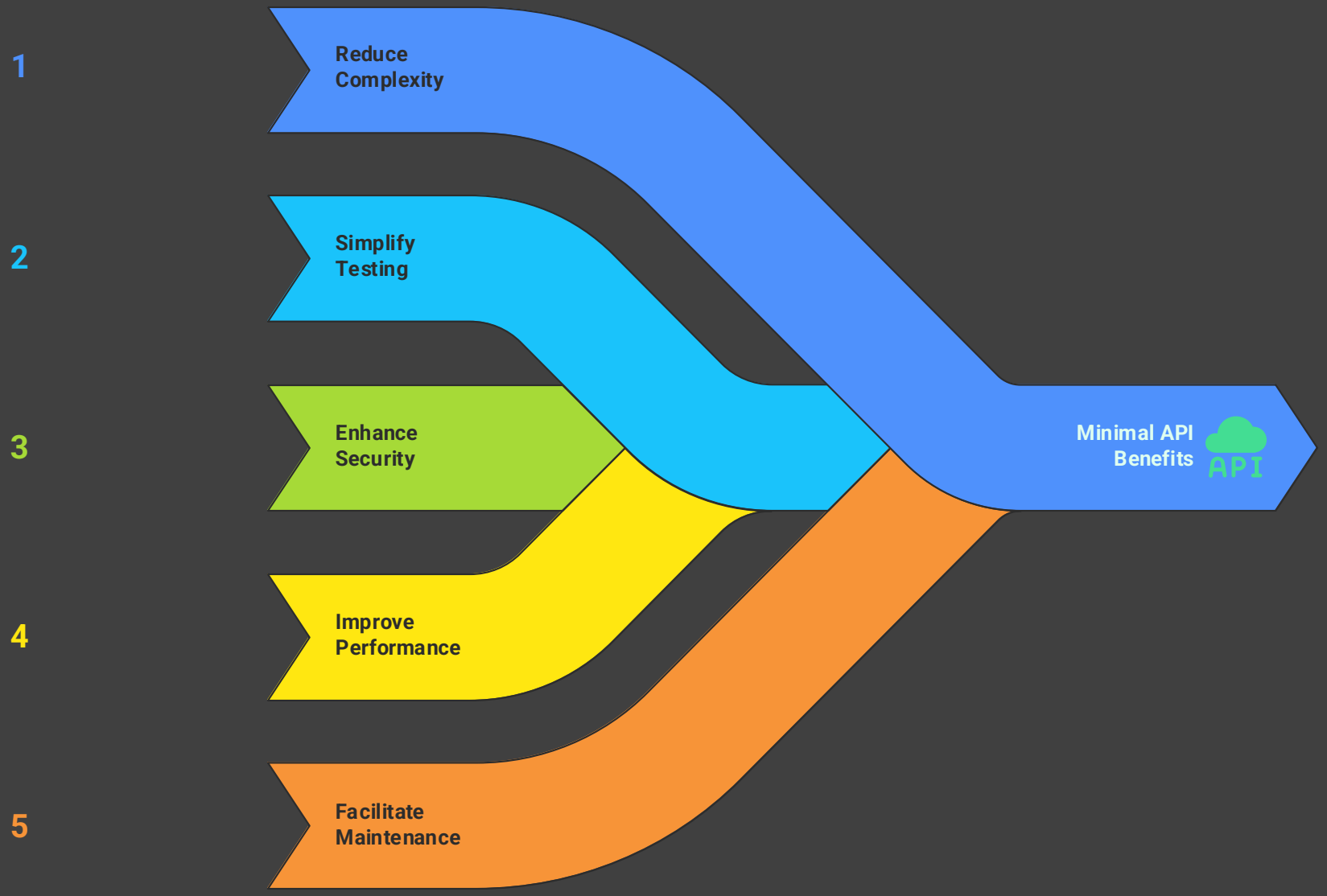
Consistent VPC Configurations
Utilize CloudFormation to create standardized VPC setups, ensuring uniform network configurations across development, staging, and production environments.

Minimize Custom Scripts
Reduce reliance on bespoke scripts by leveraging CloudFormation's declarative syntax, decreasing the potential for errors and simplifying maintenance.

Parameterization for Flexibility
Employ parameters within your templates to accommodate environment-specific values, promoting reuse and adaptability.

Automated Testing and Validation
Integrate tools like cfn-lint and TaskCat to validate templates and test deployments, ensuring predictable and reliable infrastructure provisioning.

MINIMAL APIS



IMPLEMENTING MINIMAL APIS IN AWS

Leverage Amazon API Gateway
Utilize API Gateway to create RESTful APIs with streamlined endpoints, focusing on essential functionalities to reduce complexity.

Define Clear and Concise Endpoints
Design endpoints with intuitive paths, such as /auth/login for user authentication, to simplify integration and maintenance.

Secure APIs with IAM and Throttling
Implement AWS Identity and Access Management (IAM) for authorization and configure throttling settings to protect against abuse and ensure consistent performance.

Monitor Performance with Amazon CloudWatch
Enable CloudWatch metrics and logs to gain insights into API usage, detect anomalies, and set up alarms for proactive issue resolution.

Focus on Core Functionalities
Prioritize implementing critical features first, ensuring that the API remains lean and maintainable.

MODULARITY

- Divide systems into independent components.
- Enhances maintainability and scalability.
- Example: Lambda functions for modular tasks.
- Simplifies testing and debugging.
- Supports parallel development efforts.

MODULAR ARCHITECTURE WITH AWS

Containerize Microservices with ECS or EKS

Utilize Amazon Elastic Container Service (ECS) or Amazon Elastic Kubernetes Service (EKS) to deploy and manage containerized microservices efficiently.

Implement Event-Driven Architecture with EventBridge

Use Amazon EventBridge to decouple services, enabling asynchronous communication and improving system resilience.

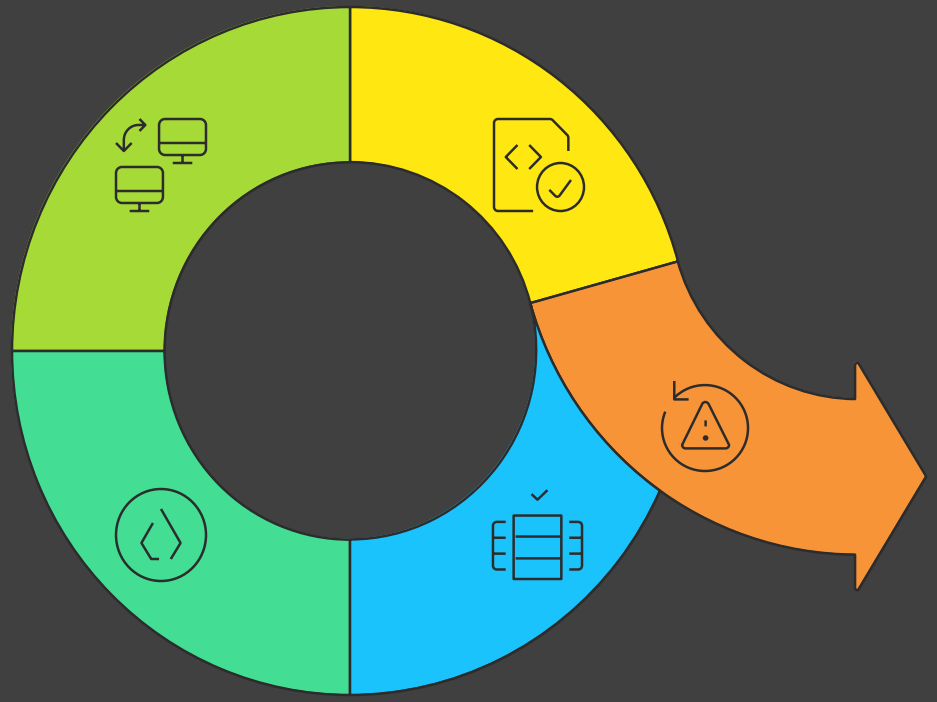
Monitor Services with Amazon CloudWatch

Leverage CloudWatch for real-time monitoring, logging, and alerting to maintain visibility into each microservice's performance.

Enable Independent Scaling

Design microservices to scale independently based on demand, optimizing resource utilization and cost.

RELEASE SIMPLICITY WITH AWS



1

Automate Deployments

Use CI/CD pipelines to reduce manual errors.

2

Leverage CodePipeline

Orchestrate build, test, and deploy phases.

3

Implement Blue/Green Deployments

Deploy new versions alongside existing ones.

4

Standardize Release Processes

Create reusable templates and policies.

5

Enable Fast Rollbacks

Use automation to revert or throttle deployments.

AUTOMATING RELEASES IN AWS

- **Integrate CodePipeline with CodeBuild for Continuous Testing**
  Automate test execution (e.g., unit, integration) as part of the release workflow.
- **Example: Run Automated Tests Before Every Deployment**
  Prevent broken code from reaching production with pre-deploy quality gates.
- **Deploy with CodeDeploy Using Advanced Strategies**
  Implement canary, linear, or blue/green deployments for controlled rollouts.
- **Monitor in Real Time with Amazon CloudWatch**
  Track metrics and logs during and after deployment for quick feedback.
- **Automation Drives Safe, Repeatable Releases**
  Minimize human error and accelerate delivery without compromising stability.

SIMPLICITY IN CODE

- **Write Clean, Readable Code with Clear Logic**
  Prioritize clarity over cleverness to reduce cognitive load.
- **Favor Simpler Algorithms When Appropriate**
  Avoid overengineering—simplicity often performs better and is easier to debug.
- **Use Descriptive Variable Names and Comments**
  Make intent obvious for current and future developers.
- **Example: Follow AWS SDK Style and Structure Guidelines**
  Adhere to best practices when interacting with AWS services programmatically.
- **Simpler Code Means Fewer Bugs and Easier Maintenance**
  Clean code accelerates onboarding, debugging, and incident response.

PROCESS SIMPLICITY

Streamline Development and Deployment Workflows
Eliminate unnecessary steps to accelerate delivery and reduce complexity.

Automate Repetitive Tasks with CodeBuild
Offload tasks like testing, linting, and packaging into automated pipelines.

Example: Run Unit Tests and Linting on Every Commit
Catch issues early with consistent, automated checks.

Standardize Workflows Across Teams
Use templates and shared configurations for uniform practices.

Simplified Processes Boost Team Velocity and Reliability
Reduce context switching, avoid errors, and improve developer experience.

CASE STUDY: SIMPLIFYING A MONOLITH

Migrated a Monolithic Application to Microservices on Amazon ECS

Transitioned from a monolithic architecture to a microservices-based approach using Amazon Elastic Container Service (ECS) for improved scalability and deployment flexibility.

Reduced Complexity and Enhanced Scalability

By decomposing the monolith, the system achieved better modularity, allowing individual services to scale independently based on demand.

Challenge: Managing Inter-Service Dependencies

The shift introduced complexities in coordinating interactions between multiple microservices, leading to potential reliability issues.

Solution: Implemented AWS Step Functions for Orchestration

Utilized AWS Step Functions to manage and visualize the workflow of microservices, ensuring reliable and maintainable inter-service communication.

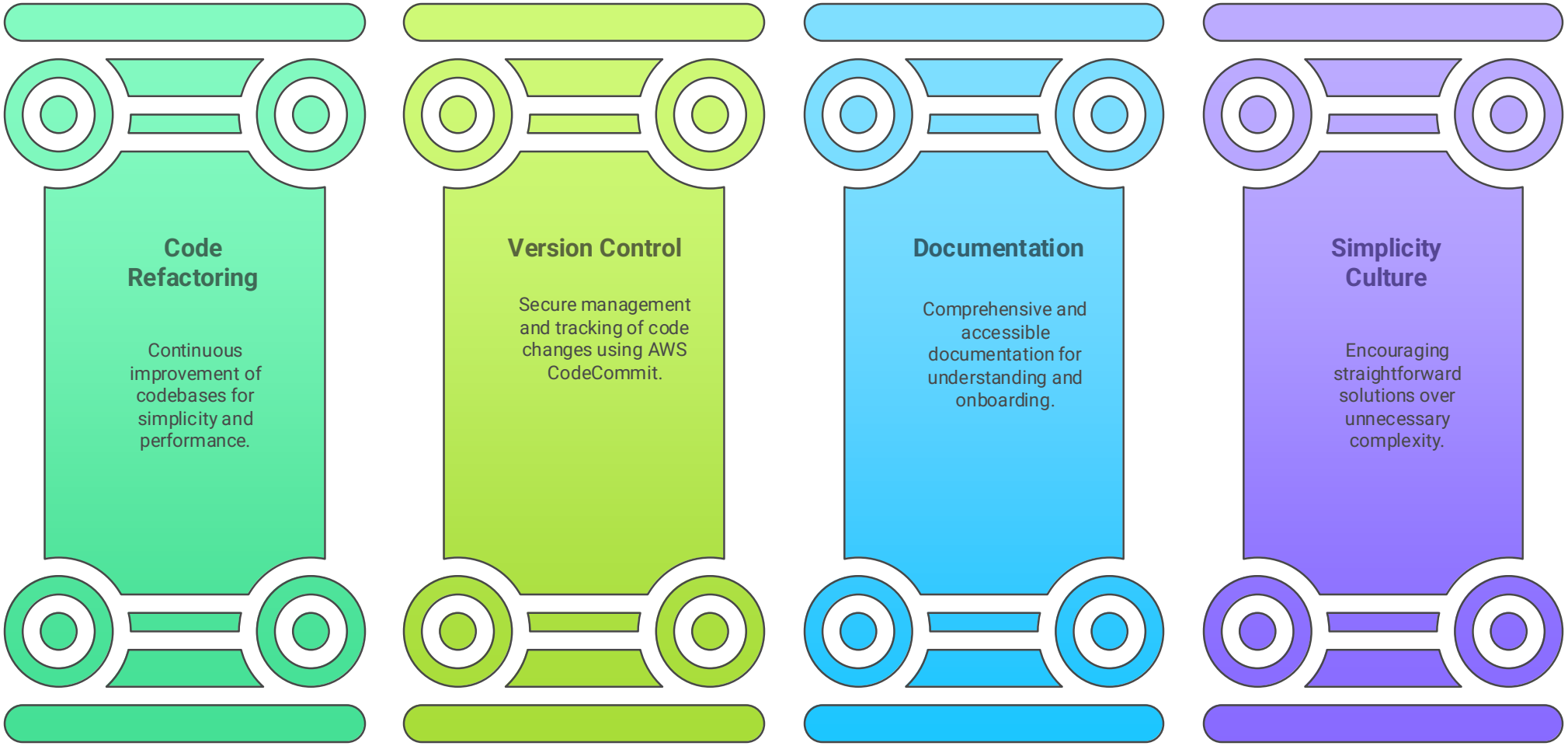
Result: Improved Reliability and Maintainability

The orchestrated microservices architecture led to a more robust system, simplifying maintenance and reducing the risk of failures.

CHALLENGES IN ACHIEVING SIMPLICITY

- Legacy systems resist simplification efforts.
- Team resistance to new practices can occur.
- Balancing simplicity with functionality is complex.
- **Example:** Refactoring legacy code for Lambda.
- Overcome with gradual changes and training.

BEST PRACTICES FOR SIMPLICITY



SIMPLICITY AS A MINDSET

Simplicity Drives Reliability and Efficiency
Simpler systems are easier to understand, maintain, and scale, reducing the likelihood of errors.

Applies to Code, Architecture, and Processes
Embrace simplicity across all facets of development to enhance overall system stability.

Leverage Automation and Standardization in AWS
Utilize AWS tools like CloudFormation and CodePipeline to automate deployments and enforce consistent configurations.

Continuous Pursuit of Simplicity Improves Systems
Regularly refactor and assess systems to eliminate unnecessary complexity and improve performance.

Embrace Simplicity for Advanced SRE Success
Adopting a simplicity-first approach aligns with SRE principles, leading to more reliable and efficient operations.

POP QUIZ:

You are designing a RESTful API using Amazon API Gateway. To enhance security and maintainability, you decide to implement minimal APIs. Which of the following practices align with this approach? (Choose two.)
A. Implementing numerous endpoints for granular operations
B. Using IAM roles to secure endpoints
C. Defining clear and concise endpoint paths
D. Allowing unrestricted access to all endpoints

POP QUIZ:

You are designing a RESTful API using Amazon API Gateway. To enhance security and maintainability, you decide to implement minimal APIs. Which of the following practices align with this approach? (Choose two.)

A. Implementing numerous endpoints for granular operations

B. Using IAM roles to secure endpoints

C. Defining clear and concise endpoint paths

D. Allowing unrestricted access to all endpoints

POP QUIZ:

To minimize downtime and reduce deployment risks, which deployment strategy can be implemented using AWS CodeDeploy?
A. Rolling deployment
B. Blue/green deployment
C. Canary deployment
D. All of the above

POP QUIZ:

To minimize downtime and reduce deployment risks, which deployment strategy can be implemented using AWS CodeDeploy?
A. Rolling deployment
B. Blue/green deployment
C. Canary deployment
D. All of the above

POP QUIZ:

A company is transitioning from a monolithic architecture to microservices using AWS. They aim to manage inter-service communication effectively. Which AWS service is best suited for orchestrating complex workflows between microservices?
A. Amazon SQS
B. AWS Step Functions
C. Amazon SNS
D. AWS Lambda

POP QUIZ:

A company is transitioning from a monolithic architecture to microservices using AWS. They aim to manage inter-service communication effectively. Which AWS service is best suited for orchestrating complex workflows between microservices?

A. Amazon SQS

B. AWS Step Functions

C. Amazon SNS

D. AWS Lambda

LAB 4: BLUE/GREEN DEPLOYMENT

Goal: Deploy new application versions to production with zero downtime using a **Blue/Green strategy** integrated with GitHub and AWS-native CI/CD services.

What You Will Learn

- Connect GitHub to AWS CodePipeline
- Configure a launch template and Auto Scaling group
- Use CodeDeploy with Blue/Green deployment strategy
- Automate canary rollouts and rollback
- Push changes via GitHub to trigger live deployments

Instructions: Lab4.md

PRACTICAL ALERTING FROM TIME-SERIES DATA

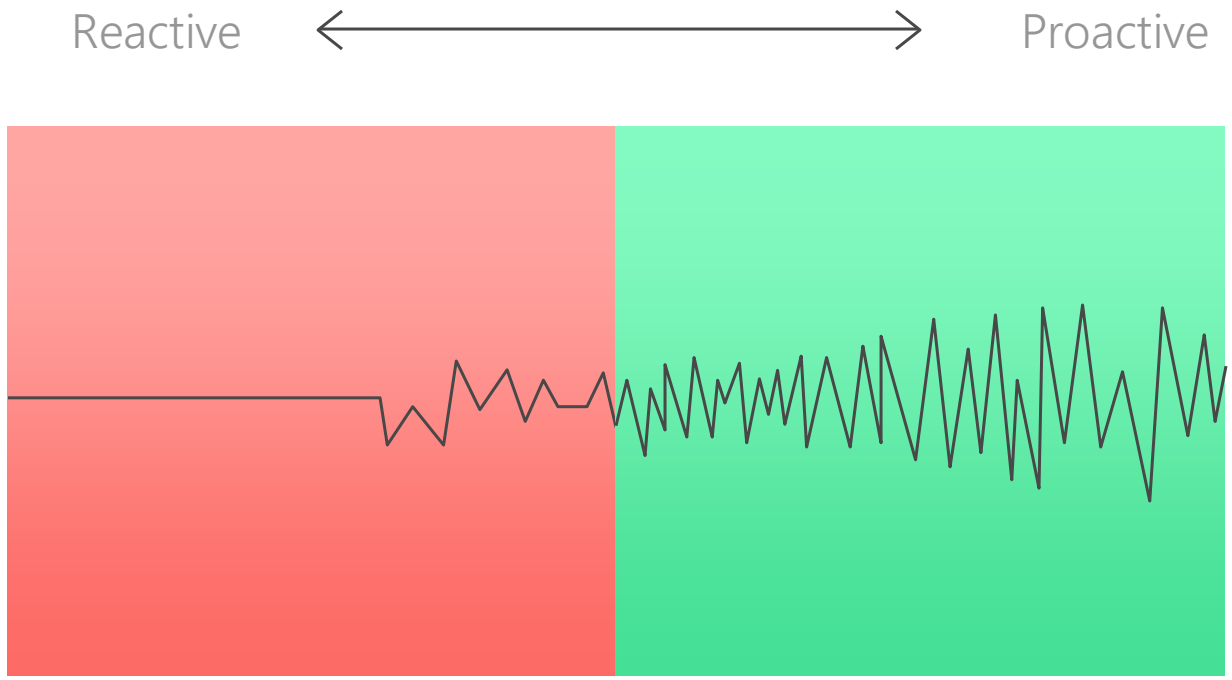
INTRODUCTION

- Alerting detects issues early—before users are impacted.
- Use time-series data to understand trends and anomalies.
- Design alerts to be actionable, not noisy.
- AWS CloudWatch provides metrics, alarms, and dashboards.
- Key to maintaining SLOs, SLIs, and overall reliability

WHY ALERTING MATTERS

- **Prevents user impact through early detection**
  Triggers alerts on anomalies before they escalate into visible outages
- **Supports scientific incident-response methods**
  Uses time-series data and structured workflows (e.g., NIST detection & analysis) for repeatable, data-driven remediation
- **Aligns systems with business objectives**
  Maps SLIs/SLOs and alert thresholds to key business KPIs for measurable value delivery
- **Example: CloudWatch alarms for high error rates**
  Define metric-math alarms (errors/requests > 5% over 15 min) to automatically notify on-call teams
- **Enhances proactive system management**
  Combines anomaly detection and composite alarms to reduce noise and focus on actionable incidents

TYPES OF ALERTS



Reactive Alerts

Responds to immediate system failures.

Proactive Alerts

Predicts issues based on emerging trends.

BEST PRACTICES FOR ALERTING

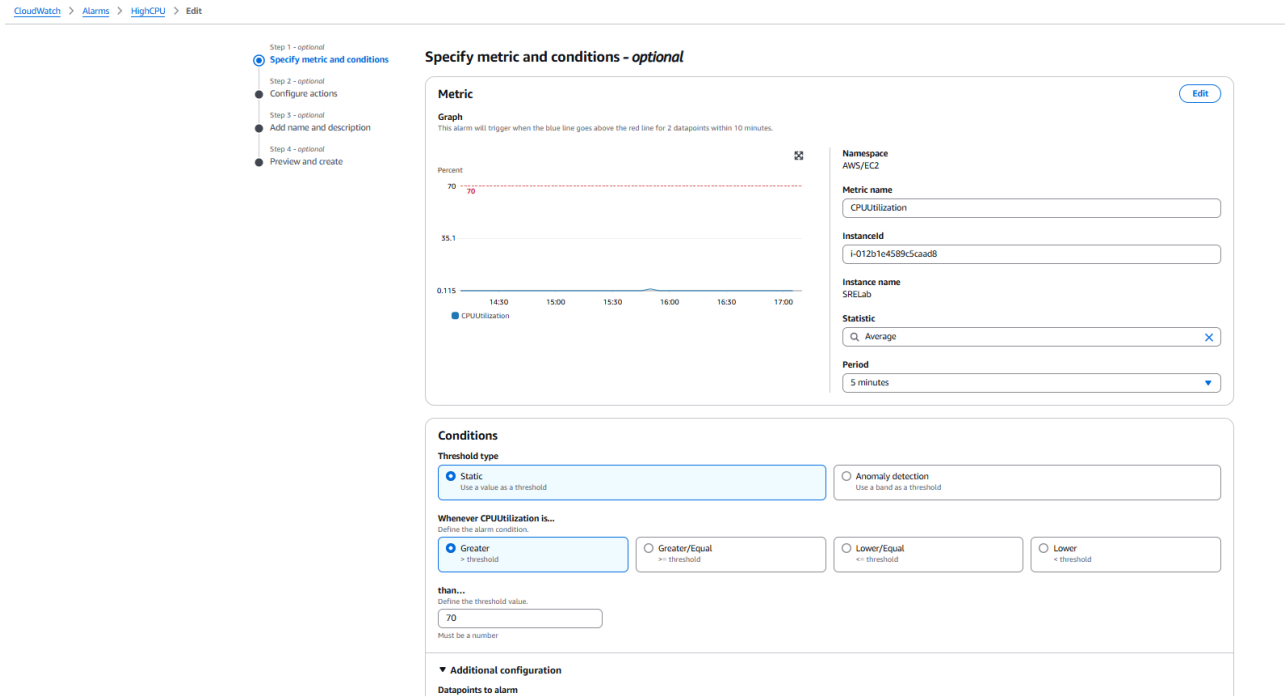
Align with SLOs for Relevance
Only trigger alerts when you're approaching an SLO breach—ensuring every page matters to your reliability goals

Minimize False Positives to Avoid Fatigue
Tune thresholds and use anomaly detection so you only see meaningful alerts

Use Correlation to Group Related Alerts
Leverage rule-based or ML-driven correlation to cluster symptoms from the same root cause

Example: Grouping Alerts from One Outage
Combine CPU, memory, and network alarms into a single incident to reduce noise
Conduct periodic audits to retire stale alerts and adjust thresholds based on incident post-mortems

SETTING UP CLOUD WATCH ALARMS



- Define metrics like CPU usage or error rates.
- Set thresholds based on SLOs.
- Configure SNS notifications for alerts.
- Example: Alarm on CPU > 80% for 5 minutes.
- Test alarms to ensure actionability.

INSTRUMENTATION OF APPLICATIONS

```
import { PutMetricDataCommand } from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";

const run = async () => {
  // See https://docs.aws.amazon.com/AmazonCloudWatch/latest/APIReference/API_PutMetricData.html#API_PutMetricData_RequestParameters
  // and https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/publishingMetrics.html
  // for more information about the parameters in this command.
  const command = new PutMetricDataCommand({
    MetricData: [
      {
        MetricName: "PAGES_VISITED",
        Dimensions: [
          {
            Name: "UNIQUE_PAGES",
            Value: "URLS",
          },
        ],
        Unit: "None",
        Value: 1.0,
      },
    ],
    Namespace: "SITE/TRAFFIC",
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

- **Add Code to Collect Metrics**
  Integrate hooks in your application to capture key performance data points (e.g., request counts, errors).
- **Use AWS SDKs to Send Metrics to CloudWatch**
  Leverage the PutMetricData API via AWS SDKs (e.g., Boto3, AWS SDK for JavaScript) for reliable metric ingestion.
- **Example: Tracking Request Latency in a Web App**
  Publish a custom RequestLatency metric in a GetStarted namespace to visualize end-to-end response times.
- **Ensure Metrics Are Meaningful and Actionable**
  Choose metrics that directly reflect user experience or system health, avoiding low-value “noise” data.
- **Instrumentation Drives Effective Alerting**
  High-quality metrics form the foundation for precise, actionable CloudWatch alarms and dashboards.

STORING TIME-SERIES DATA

Default Retention

High-resolution metrics retained for 14 days by default

Custom Retention

Retention period can be adjusted for longer historical insights

Cost Management

Retention policies help optimize storage costs

LABELS AND DIMENSIONS

- **Contextualize Metrics with Dimensions**:
  Add metadata (e.g., InstanceId, AutoScalingGroupName) to metrics so you can filter and analyze performance on a per-resource basis

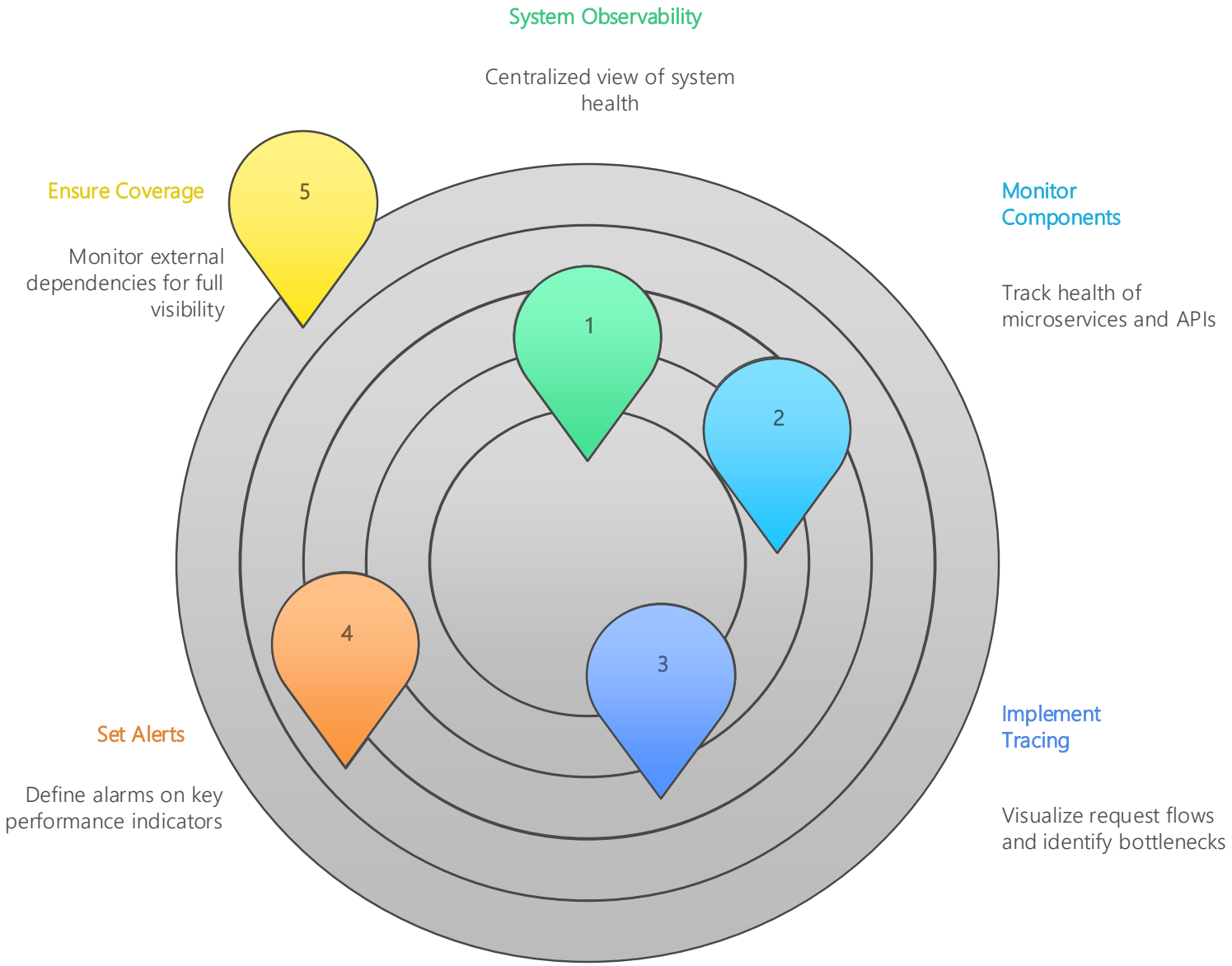
- **Group and Filter Metrics**:
  Use dimensions like Region or Environment to segment dashboards, making it easy to compare behavior across zones or stages

- **Example: Region-Specific Monitoring**:
  Applying a Region dimension lets you track request latency differences between us-east-1 and eu-west-1

- **Precision in Alerting**:
  Scope alarms to specific dimensions (e.g., a single instance or cluster) so only relevant alerts fire, cutting down on noise

- **Reduce Alert Fatigue**:
  Narrow alert targets with dimensions to ensure teams receive only actionable notifications, improving response efficacy

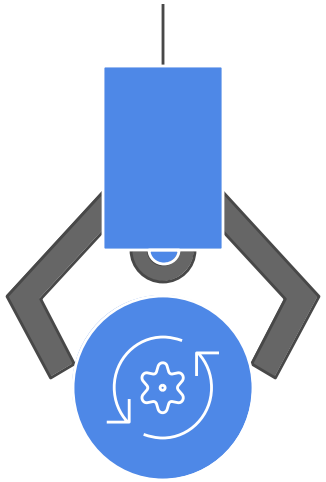
ALERTING ON DISTRIBUTED SYSTEMS



HANDLING ALERT FATIGUE

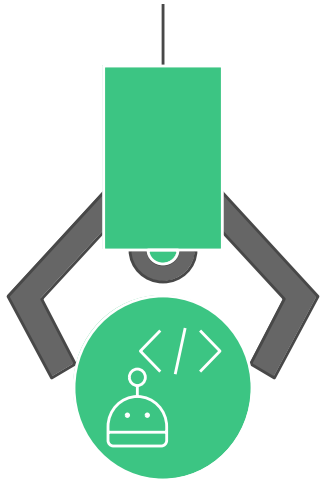
- Too many alerts cause desensitization.
- Aggregate related alerts to reduce noise.
- Example: Grouping instance failure alerts.
- Set thresholds based on impact, not just metrics.
- Regularly refine alerting rules.

AUTOMATING ALERT RESPONSES



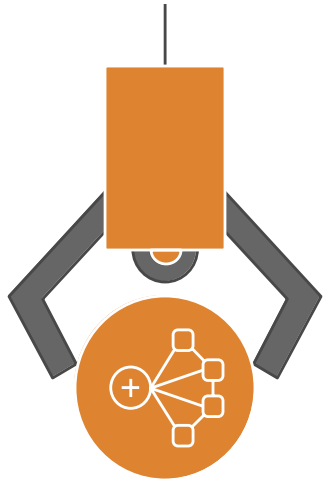
Reduce Manual Effort

Automate common fixes to reduce effort.



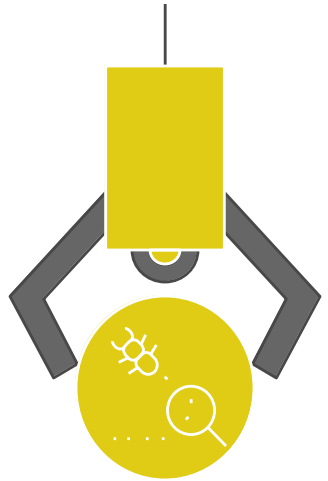
Systems Manager Automation

Use Systems Manager Automation for scripts.



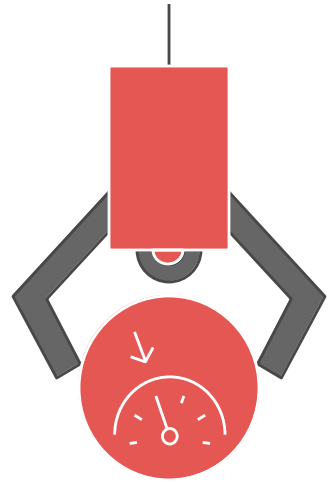
Auto-Scaling Example

Auto-scaling on high CPU alerts example.



Thorough Testing

Test automation scripts thoroughly before deployment.



Faster Response

Automation speeds up incident response time.

INTEGRATING WITH INCIDENT MANAGEMENT

Trigger PagerDuty Tickets for Critical Alerts
Configure CloudWatch Alarms to publish to an SNS topic that automatically creates PagerDuty incidents, ensuring rapid on-call engagement

Integrate with OpsGenie for Seamless Alerts
Use the AWS CloudWatch Events integration in OpsGenie to funnel alarms directly into your team's notification policies

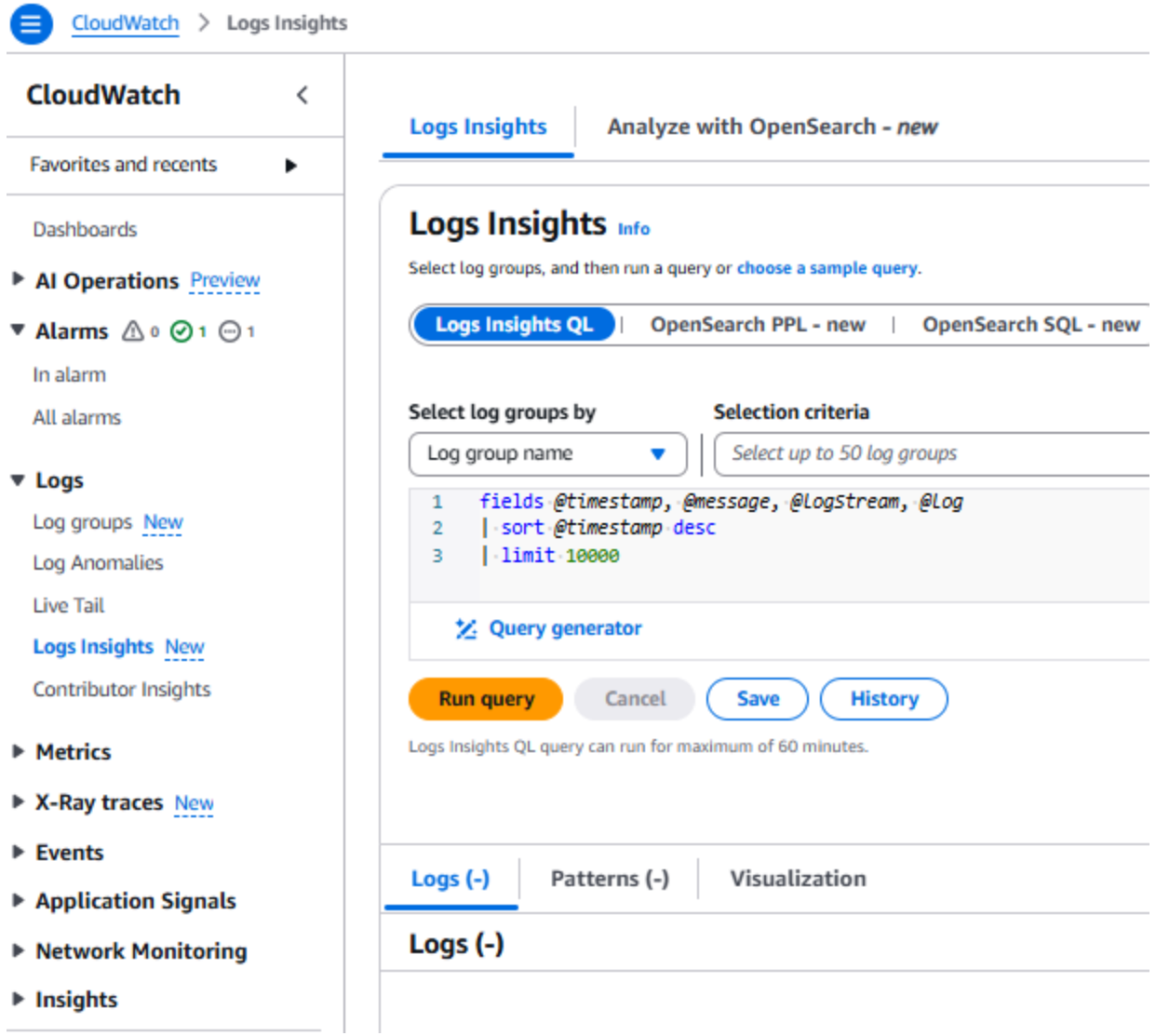
Include Actionable Context in Notifications
Enrich alerts with dimensions and custom metadata (e.g., service name, instance ID) so responders have the information they need up front

Example: Multi-Tool Orchestration
CloudWatch Alarm → SNS Topic → Lambda enrichment → PagerDuty / OpsGenie integration endpoint

Reduce MTTR through Tight Integration
Automated ticket creation and contextualized alerts cut manual handoffs, slashing response times versus ad hoc workflows

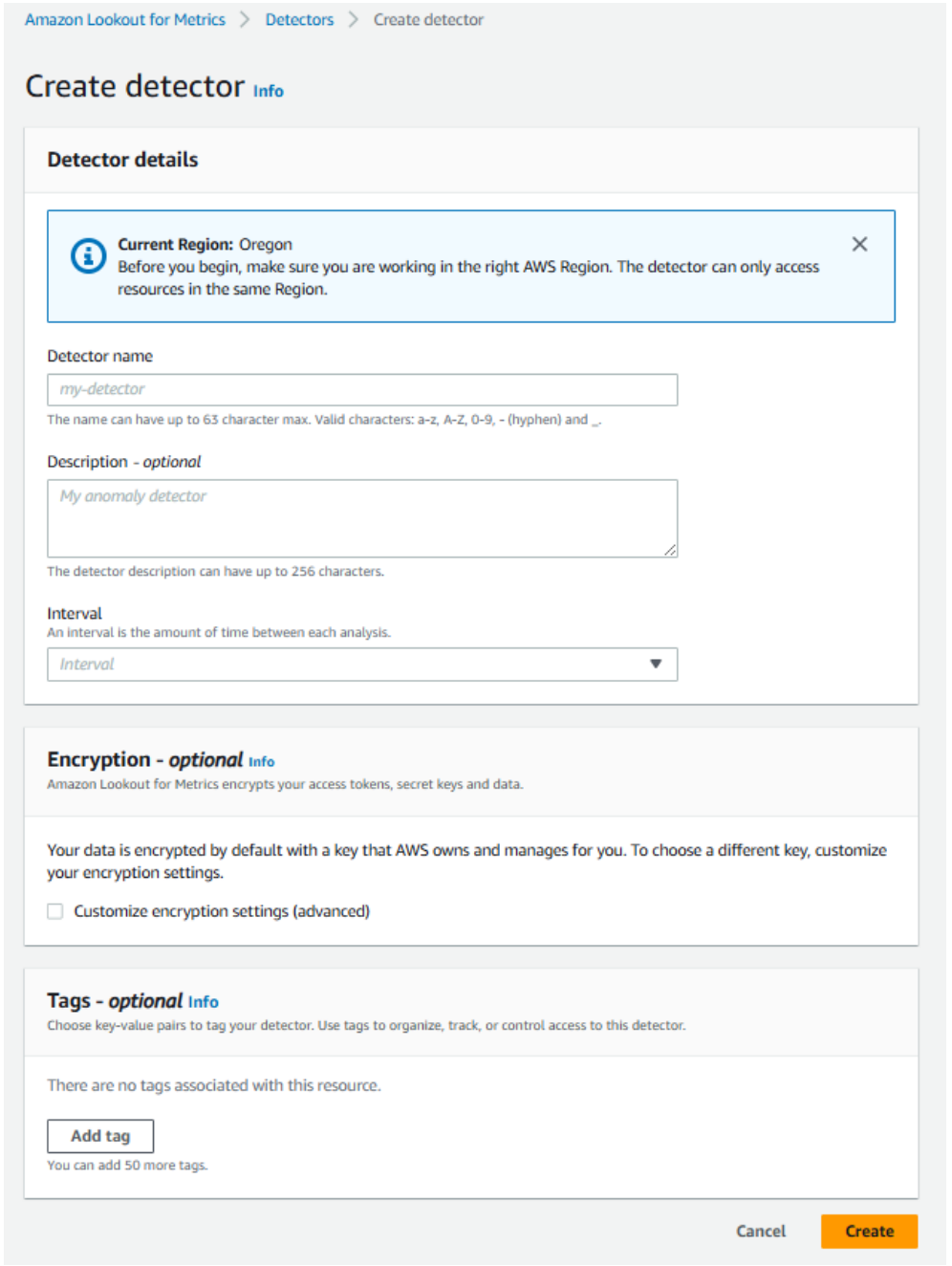
ANALYZING ALERT DATA

- Use CloudWatch Logs Insights for queries.
- Identify patterns in alert frequency/types.
- Example: Seasonal alert volume spikes.
- Refine monitoring based on insights.
- Continuous analysis improves alerting.



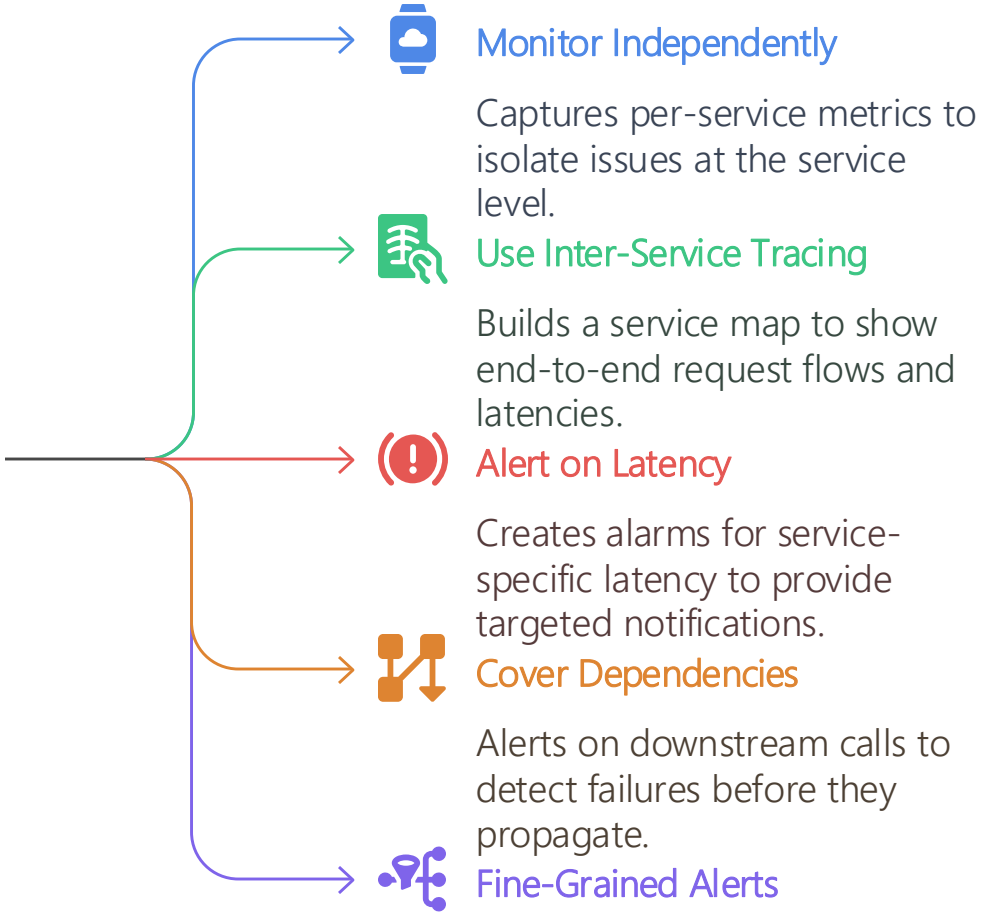
PREDICTIVE ALERTING

- Use ML to predict issues before they occur.
- Example: Amazon Lookout for Metrics anomaly detection.
- Set predictive alerts for proactive fixes.
- Reduces downtime by anticipating problems.
- Advanced alerting leverages ML insights.



ALERTING IN MICROSERVICES

How to monitor microservices effectively?



Monitor Independently

Captures per-service metrics to isolate issues at the service level.

Use Inter-Service Tracing

Builds a service map to show end-to-end request flows and latencies.

Alert on Latency

Creates alarms for service-specific latency to provide targeted notifications.

Cover Dependencies

Alerts on downstream calls to detect failures before they propagate.

Fine-Grained Alerts

Designs alerts with service and stage dimensions to reduce noise.

POP QUIZ:

During a recent outage, CPU, memory, and database latency alarms all fired separately. To streamline incident response, you want to group related alerts into a single incident. Which AWS feature/pattern accomplishes this?

A. Use a CloudWatch Composite Alarm that ORs together the three metrics.
B. Write an EventBridge rule to catch all three alarms and invoke a Lambda that consolidates them.
C. Tag all metrics with ServiceName and filter in the console.
D. Prefix all SNS topics with “Outage_” so they appear grouped in PagerDuty.

POP QUIZ:

During a recent outage, CPU, memory, and database latency alarms all fired separately. To streamline incident response, you want to group related alerts into a single incident. Which AWS feature/pattern accomplishes this?

A. Use a CloudWatch Composite Alarm that ORs together the three metrics.

B. Write an EventBridge rule to catch all three alarms and invoke a Lambda that consolidates them.

C. Tag all metrics with ServiceName and filter in the console.

D. Prefix all SNS topics with “Outage_” so they appear grouped in PagerDuty.

POP QUIZ:

You manage a critical service with a 99.9% monthly uptime SLO and want alerts only when you risk breaching the error budget. Which approach best aligns alerts with your SLOs?
A. Create individual alarms on low-level metrics (CPU, memory) and forward all notifications.
B. Configure a CloudWatch metric-math alarm that fires when error-budget burn rate exceeds a defined threshold (e.g., 2% per hour).
C. Set separate alarms on every custom application metric.
D. Use a single alarm on total request count.

POP QUIZ:

You manage a critical service with a 99.9% monthly uptime SLO and want alerts only when you risk breaching the error budget. Which approach best aligns alerts with your SLOs?

A. Create individual alarms on low-level metrics (CPU, memory) and forward all notifications.

B. Configure a CloudWatch metric-math alarm that fires when error-budget burn rate exceeds a defined threshold (e.g., 2% per hour).

C. Set separate alarms on every custom application metric.

D. Use a single alarm on total request count.

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 5: MONITORING

Goal: Automatically detect and respond to high CPU usage on an EC2 instance using native AWS observability and automation tools.

Skills Covered

- EC2 instance provisioning with IAM roles
- Using Run Command to simulate system load
- Creating and wiring CloudWatch Alarms
- Writing SSM Automation runbooks (YAML)
- Triggering remediation actions via EventBridge

Instructions: Lab5.md

EFFECTIVE TROUBLESHOOTING

INTRODUCTION

- Troubleshooting identifies/resolves system issues.
- Critical for minimizing downtime and impact.
- Requires systematic, data-driven approach.
- AWS tools: CloudWatch, X-Ray, Systems Manager.
- Learnable skill for advanced SREs.

WHAT IS TROUBLESHOOTING?

Structured Diagnosis & Remediation
Systematically identify, isolate, and resolve faults using a teachable SRE troubleshooting methodology

Data Gathering, Analysis & Action
Collect logs, metrics, and traces; analyze root causes; then implement fixes following a defined workflow

Example: Diagnosing High Latency
Use CloudWatch metrics and AWS X-Ray traces to pinpoint slow service segments and remediate performance bottlenecks

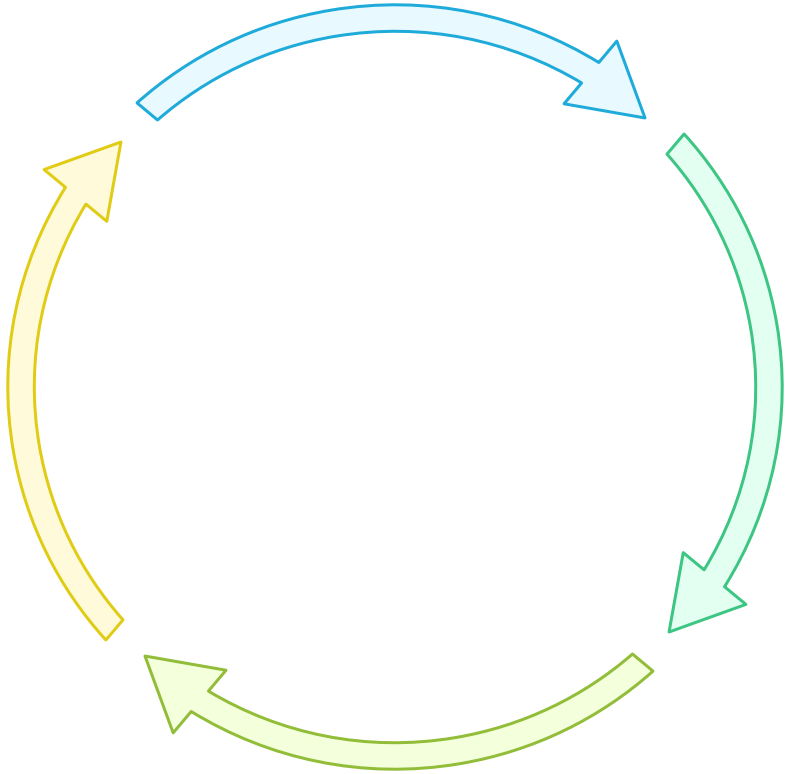
Technical & Analytical Skillset
Combines programming, systems knowledge, and critical thinking to formulate hypotheses and validate solutions

Foundation of Reliability
Rapid, accurate troubleshooting preserves uptime, meets SLOs, and builds trust in production systems

TROUBLESHOOTING IN PRACTICE

Gather Data

Collect logs, metrics, and reports



Analyze Issues

Identify and narrow down problems

Remote Debugging

Use Systems Manager for real-time checks

Practice Simulations

Enhance skills through simulations

PROBLEM REPORT

- Include what, when, where, and impact.
- Clarifies scope and urgency of issues.
- Use AWS Health Dashboard for service issues.
- Example: Reporting multi-region outage.
- Clear reports speed up resolution.

TRIAGE

Assess Severity & Urgency
Quickly categorize incidents by their technical impact and how fast they worsen to determine response speed

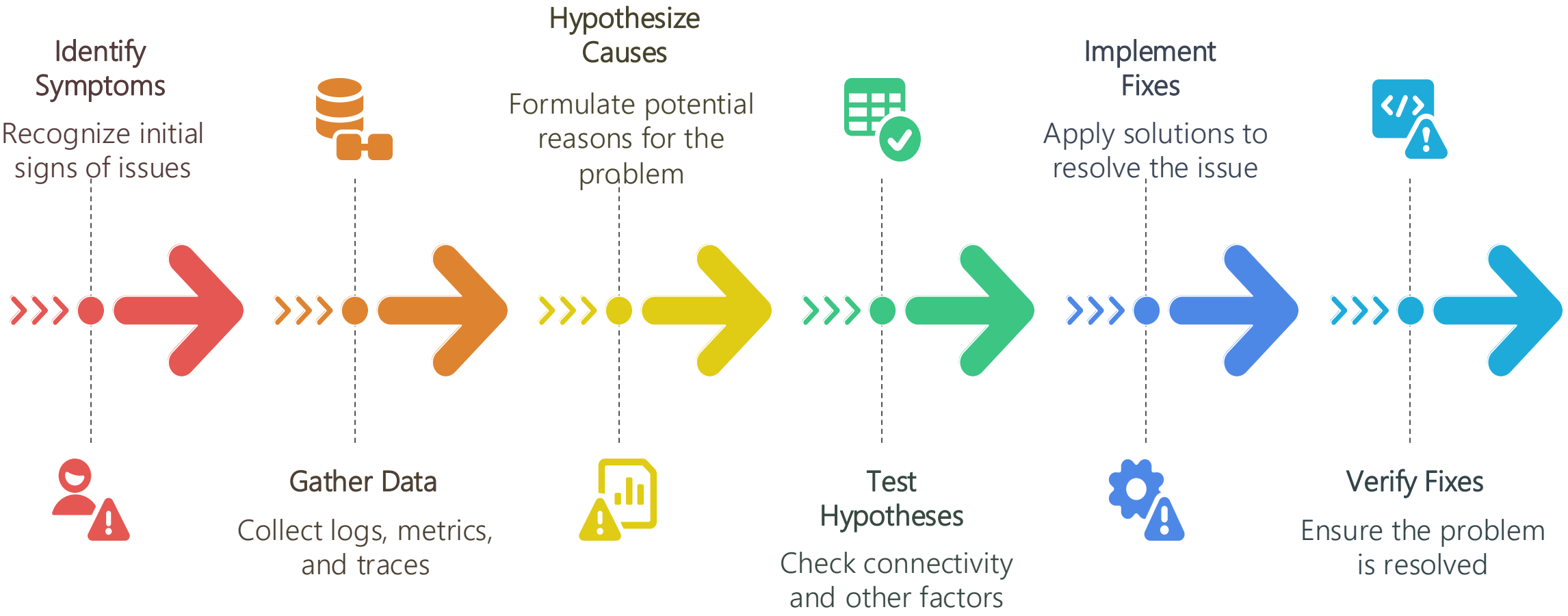
Prioritize by Business & User Impact
Focus on issues that affect revenue-critical features or large user populations first

Leverage Error Budgets for Guidance
Use remaining error budget burn rate to decide when to halt feature releases and concentrate on reliability work

Example: Handling a Critical Outage
An SLO-breaching database failure (e.g., >0.1% downtime) triggers top-priority triage and mobilizes the full on-call rotation

Optimize Resource Allocation
Allocate engineering time and on-call resources based on triage outcomes to maximize MTTR reduction

TROUBLESHOOTING STEPS



TROUBLESHOOTING TOOLS

AWS Systems Manager

TROUBLESHOOTING TECHNIQUES

Resilience

The ultimate goal of advanced troubleshooting

Chaos Engineering

Simulates failures to test system robustness

CloudWatch Logs Insights

Analyzes trends in log data

Lookout for Metrics

Uses ML to detect anomalies

TROUBLESHOOTING DISTRIBUTED SYSTEMS

Visualize Request Flows Across Microservices

AWS X-Ray provides a comprehensive view of requests as they traverse through various services, enabling developers to understand the application's behavior and identify issues.

Identify Performance Bottlenecks

By analyzing trace data, X-Ray helps pinpoint services or functions that are causing latency or errors, facilitating targeted optimizations.

Example: Tracing a Request Through Microservices

For instance, tracing a user request from the front-end service through authentication, business logic, and database services can reveal where delays or failures occur.

Monitor Inter-Service Communication

X-Ray's service map visualizes the interactions between services, helping detect issues in inter-service communication and dependencies.

Essential for Distributed Systems

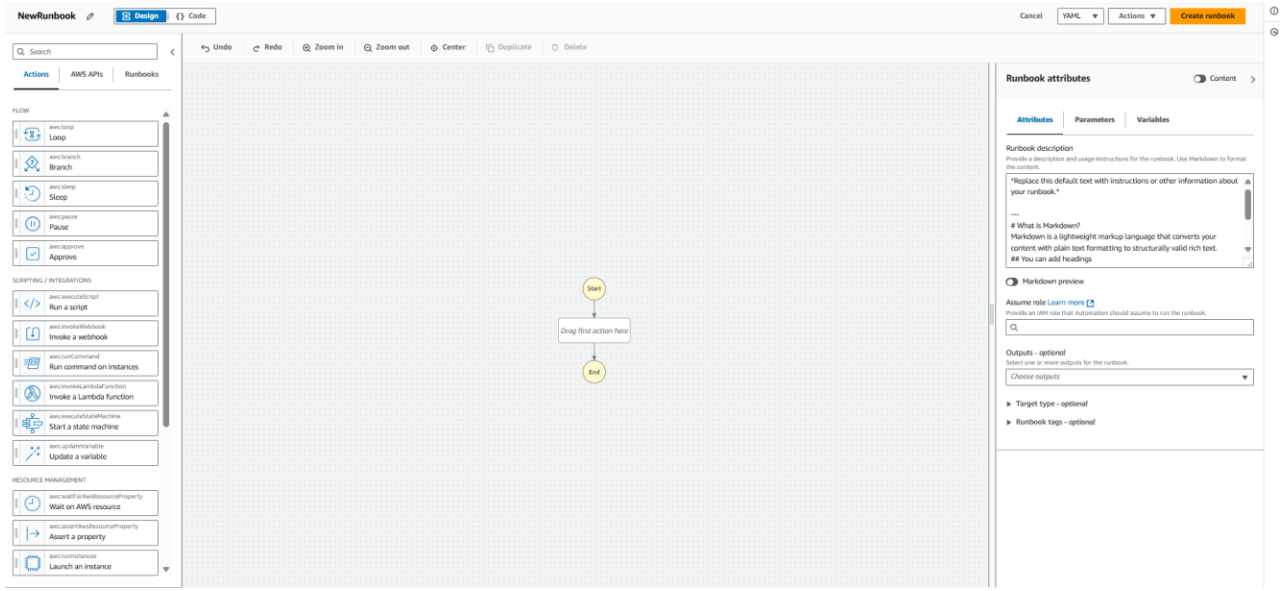
In complex architectures, such as microservices, having a holistic view is crucial for maintaining system reliability and performance.

COMMON PITFALLS

- Jumping to conclusions without data.
- Ignoring historical trends or patterns.
- Overlooking dependencies or external factors.
- Example: Blaming the database without checking.
- Failing to document troubleshooting steps.

DOCUMENTING TROUBLESHOOTING ERRORS

- Use runbooks for standardized responses.
- Record steps in incident management tools.
- Example: Documenting in Jira or Confluence.
- Share findings in postmortems.
- Documentation improves future responses.



CASE STUDY: TROUBLESHOOTING LATENCY SPIKE

Issue: Users reported high latency in the web application.

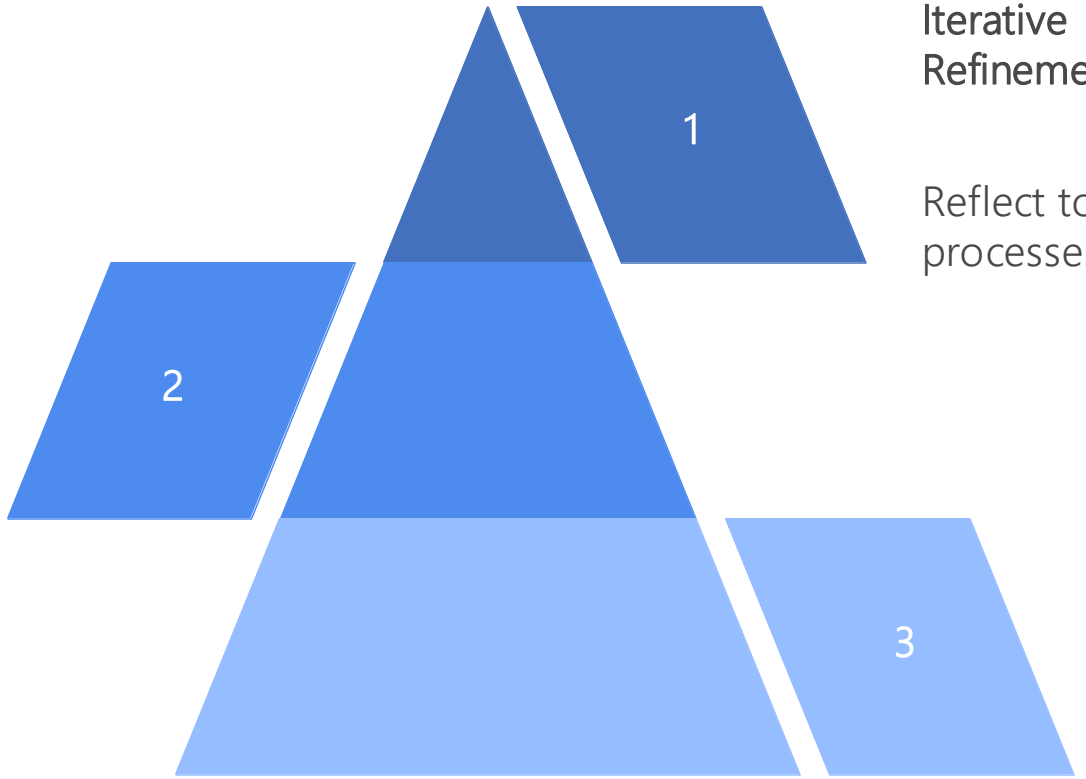
Action: Utilized AWS X-Ray to trace requests and identify performance bottlenecks.

Findings: Discovered a specific database query causing significant delays.

Solution: Optimized the inefficient query, resulting in a 50% reduction in latency.

Outcome: Demonstrated the effectiveness of a data-driven approach in resolving performance issues.

TROUBLESHOOTING CULTURE



Iterative Refinement

Reflect to improve processes

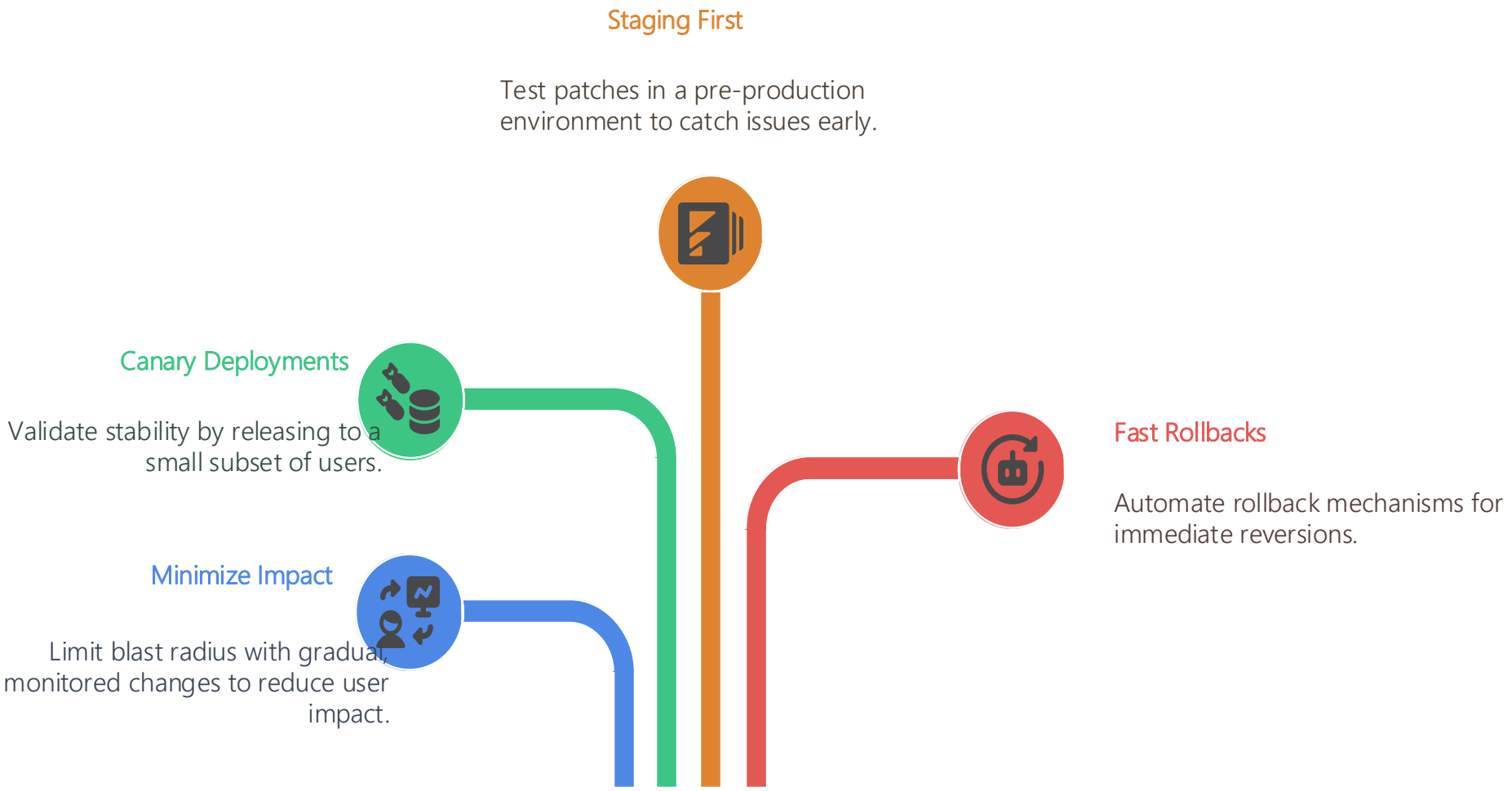
Collaborative Exchange

Shared problems, faster solutions

Blameless Environment

Focus on learning, not blame

TROUBLESHOOTING IN PRODUCTION



TROUBLESHOOTING WITH AWS FIS

- Simulate failures to test hypotheses.
- Example: Injecting network latency.
- Validate system resilience under stress.
- Use FIS to automate chaos experiments.
- FIS supports data-driven troubleshooting.

https://aws.amazon.com/fis/

POP QUIZ:

Your web application is experiencing intermittent high latency. You want to pinpoint which microservice segment is causing the delay. Which AWS service and feature should you use?
A. AWS CloudTrail to audit API calls
B. Amazon CloudWatch Logs metric filters
C. AWS X-Ray service map and trace data
D. AWS Config to review resource changes

POP QUIZ:

Your web application is experiencing intermittent high latency. You want to pinpoint which microservice segment is causing the delay. Which AWS service and feature should you use?

A. AWS CloudTrail to audit API calls

B. Amazon CloudWatch Logs metric filters

C. AWS X-Ray service map and trace data

D. AWS Config to review resource changes

POP QUIZ:

After identifying a suspected bottleneck, you need to validate your hypothesis by simulating network delays between services. Which tool should you employ?
A. Amazon Inspector for security assessments
B. AWS Fault Injection Simulator to inject latency
C. AWS CloudWatch Synthetics for endpoint health checks
D. AWS CloudFormation drift detection

POP QUIZ:

After identifying a suspected bottleneck, you need to validate your hypothesis by simulating network delays between services. Which tool should you employ?

A. Amazon Inspector for security assessments

B. AWS Fault Injection Simulator to inject latency

C. AWS CloudWatch Synthetics for endpoint health checks

D. AWS CloudFormation drift detection

POP QUIZ:

To capture application-level metrics (e.g., request latency) in real time, which instrumentation pattern is most appropriate?

A. Write latency data to Amazon S3 and analyze daily
B. Use AWS SDK's PutMetricData API in your code
C. Log latencies and reuse a Lambda to poll logs hourly
D. Send an email notification on every request

POP QUIZ:

To capture application-level metrics (e.g., request latency) in real time, which instrumentation pattern is most appropriate?

A. Write latency data to Amazon S3 and analyze daily

B. Use AWS SDK's PutMetricData API in your code

C. Log latencies and reuse a Lambda to poll logs hourly

D. Send an email notification on every request

POP QUIZ:

To maintain a clear record of what happened during an outage, where should engineers document each troubleshooting step?
A. Personal email threads
B. AWS CloudWatch annotations only
C. Incident management tools like Jira or Confluence runbooks
D. Amazon S3 bucket logs

POP QUIZ:

To maintain a clear record of what happened during an outage, where should engineers document each troubleshooting step?

A. Personal email threads

B. AWS CloudWatch annotations only

C. Incident management tools like Jira or Confluence runbooks

D. Amazon S3 bucket logs

POP QUIZ:

You've automated a fix via an SSM Automation runbook. How should you ensure your remediation script won't cause unexpected issues in production?
A. Use Systems Manager's dry-run mode and test in a staging environment
B. Skip testing and run directly in prod for speed
C. Only test on a single EC2 instance without validation
D. Rely on manual confirmations post-execution

POP QUIZ:

You've automated a fix via an SSM Automation runbook. How should you ensure your remediation script won't cause unexpected issues in production?

A. Use Systems Manager's dry-run mode and test in a staging environment

B. Skip testing and run directly in prod for speed

C. Only test on a single EC2 instance without validation

D. Rely on manual confirmations post-execution

EMERGENCY RESPONSE

INTRODUCTION

Critical, Time-Sensitive Incidents
Rapidly evolving events—such as outages or security breaches—that demand immediate, coordinated action

Common Scenarios
Production service disruptions, data corruption or loss, and high-severity security incidents

Structured Response Frameworks
Pre-defined Incident Manager response plans, escalation paths, and runbooks to standardize reactions

- **AWS Emergency Tools**
  AWS Systems Manager Incident Manager for incident orchestration
  Amazon CloudWatch Alarms & Events for detection and notification
  AWS Fault Injection Simulator (FIS) for proactive resilience testing

Minimize Impact with Speed & Clarity
Automated playbooks, clear communication channels, and post-incident reviews to restore service and improve practices

WHAT TO DO WHEN SYSTEMS BREAK

Stay Calm and Follow Procedure

Maintain composure and initiate incident response playbooks

Assess Using Observability Tools

Use tools like CloudWatch to identify anomalies

Example: Use CloudWatch Dashboards

Visualize system metrics to understand the issue

Communicate with Clarity and Frequency

Keep stakeholders informed using standardized channels

Contain First, Resolve Next

Isolate affected components and then fix the root cause

STRUCTURED RESPONSE FRAMEWORK

- **Assign an Incident Commander (IC)**
  Designate a single leader responsible for managing the response.

- **IC Coordinates Teams and Priorities**
  Oversees communication, triage, containment, and resolution efforts.

- **Define Supporting Roles Clearly**
  Assign a communicator (external updates) and operators (technical fixers).

- **Use Real-Time Documentation Tools**
  Track actions and decisions in a live doc (e.g., Google Docs or Notion).

- **Example: IC Logs Timeline and Actions**
  Maintain a running incident log with timestamps and roles for postmortem.

TEST-INDUCED EMERGENCY

- **Chaos Engineering Builds System Resilience**
  Introduce controlled failure to assess system behavior under stress.
- **Example: Simulate EC2 Instance Failure with AWS FIS**
  Use AWS Fault Injection Simulator to mimic production disruptions.
- **Validate Automation and Incident Response Plans**
  Confirm that auto-remediation scripts and alerting function as expected.
- **Identify Gaps in Process, Tooling, or Team Readiness**
  Reveal missing runbooks, misconfigured alerts, or slow response steps.
- **Proactive Testing Prepares Teams for Real Crises**
  Practice and preparation minimize downtime during actual incidents.

RESPONSE STRATEGIES

Contain the Issue to Prevent Escalation

Limit the blast radius by isolating failing components.

Example: Isolate Affected Services or AZs

Temporarily disable access to misbehaving services or reroute traffic.

Restore Service with Temporary Fixes if Needed

Apply hotfixes or scale resources to stabilize quickly.

Communicate Status Updates Frequently

Maintain trust with stakeholders through transparent, real-time updates.

Plan for Full Resolution Post-Containment

Begin root cause analysis and long-term remediation after stability.

COMMUNICATION DURING EMERGENCIES

- Designate a communicator for updates.
- Use predefined templates for consistency.
- Example: Slack channels for incident updates.
- Keep stakeholders informed of progress.
- Clear communication reduces confusion.

BEST PRACTICES FOR EMERGENCY RESPONSE

1. **Conduct Regular Drills to Test Team Readiness**
   Simulated incidents improve response time and build muscle memory.

2. **Maintain Up-to-Date Runbooks for Common Scenarios**
   Clear, actionable instructions speed up diagnosis and resolution.

3. **Example: Monthly Chaos Engineering Tests Using AWS FIS**
   Intentionally disrupt systems to validate resilience and response plans.

4. **Use Automation via AWS Systems Manager to Accelerate Recovery**
   Automate repetitive fixes and recovery steps for faster stabilization.

5. **Perform Postmortems to Continuously Refine Processes**
   Analyze each incident to identify gaps and apply improvements.

POP QUIZ:

After an emergency, you conduct a post-incident review to identify process gaps and prevent recurrence. Which activity is **least** aligned with best practices for continuous improvement?

A. Updating runbooks based on lessons learned
B. Publicly sharing all incident details, including customer data
C. Analyzing response metrics (MTTR, page-to-ack time)
D. Adjusting CloudWatch alarm thresholds as needed

POP QUIZ:

After an emergency, you conduct a post-incident review to identify process gaps and prevent recurrence. Which activity is **least** aligned with best practices for continuous improvement?

A. Updating runbooks based on lessons learned
B. Publicly sharing all incident details, including customer data
C. Analyzing response metrics (MTTR, page-to-ack time)
D. Adjusting CloudWatch alarm thresholds as needed

MANAGING INCIDENTS

UNMANAGED INCIDENTS

Lack of Structure Leads to Freelancing

Ad-hoc response means random team members jump in without coordination, causing duplicated or conflicting actions

Poor Communication Causes Confusion

No dedicated communicator results in mixed messages and missed updates, slowing response

Two engineers unknowingly work the same issue, wasting time and bandwidth

Delays Resolution and Increases Impact

Uncoordinated actions extend customer-facing outages and elevate business risk

Structured Management Prevents Chaos

Predefined roles, a clear chain of command, and runbooks ensure efficient, unified response

POOR COMMUNICATION

- **Leads to Misaligned Efforts and Delays**
  Without a central communicator, teams duplicate work or miss critical steps, prolonging outages
- **Stakeholders Lack Visibility into Progress**
  Infrequent or inconsistent updates leave executives, customers, and partners in the dark
- **Example: Failing to Update Customers During Outages**
  Missing status page updates or customer notifications exacerbates frustration and support load
- **Use Designated Communicators for Clarity**
  Assign a single communications manager to craft and distribute consistent, templated messages
- **Communication Is Key to Managed Incidents**
  Clear, timely updates streamline coordination and maintain trust, preventing chaos in AWS environments

FREELANCING

- Engineers act independently without coordination.
- Duplicates work and misses critical steps.
- Example: Two engineers fixing the same issue.
- Structured roles prevent freelancing.
- Coordination ensures efficient resolution.

LIVE INCIDENT STATE DOCUMENT

Centralized Tracking of Incident Progress
Maintain a single, structured document capturing timeline, status, and assigned roles

Real-Time Collaborative Updates
Ensure all responders have immediate visibility into the latest actions and decisions

Example: Google Docs or Confluence
Use collaborative platforms for live editing and easy access across teams

Transparency for Stakeholders
Grant read-only access so executives and partners can monitor without interrupting the response

Accountability Through Documentation
Record every decision, next steps, and ownership to facilitate postmortems and continuous improvement

CLEAR, LIVE HANDOFF

Ensure Smooth Transitions

Use standardized handoff templates to document current status, ongoing tasks, and next steps.

Document System State & Next Steps

Include metrics, alerts, and active remediation tasks so the incoming team can pick up seamlessly.

Example: Follow-the-Sun Handoff Document

A shared Confluence page or Google Doc updated in real time across time zones, ensuring 24/7 coverage

Prevent Context Loss

Capture the incident timeline, owner assignments, and decision rationale to avoid repeating work.

POP QUIZ:

To determine which incidents, require immediate engineering focus versus those that can wait, you want to incorporate business impact and error-budget status into your prioritization. Which practice achieves this?

A. Triage by severity and urgency, guided by remaining error budget
B. Escalate all P1 incidents to the executive team
C. Disable non-critical alerts until after business hours
D. Assign incidents on a first-come, first-served basis

POP QUIZ:

To determine which incidents, require immediate engineering focus versus those that can wait, you want to incorporate business impact and error-budget status into your prioritization. Which practice achieves this?

A. Triage by severity and urgency, guided by remaining error budget

B. Escalate all P1 incidents to the executive team

C. Disable non-critical alerts until after business hours

D. Assign incidents on a first-come, first-served basis

POP QUIZ:

During a live incident, you must record every action, timestamp, and decision in a centralized, collaboratively editable document for transparency and accountability. Which tool and practice best support this requirement?

A. Email all updates to the SRE mailing list
B. Use a live incident state document in Google Docs or Confluence
C. Rely solely on CloudWatch annotations
D. Log all actions in a local text file on each engineer's laptop

POP QUIZ:

During a live incident, you must record every action, timestamp, and decision in a centralized, collaboratively editable document for transparency and accountability. Which tool and practice best support this requirement?

A. Email all updates to the SRE mailing list

B. Use a live incident state document in Google Docs or Confluence

C. Rely solely on CloudWatch annotations

D. Log all actions in a local text file on each engineer's laptop

TRIAGING OUTAGES

WHY TRACK OUTAGES?

Measure Failure Frequency
Count incidents over time (e.g., monthly database outages) to spot patterns

Quantify Impact
Track downtime duration, error rates, and affected user sessions to assess business impact

Identify Recurring Issues
Use dashboards and log analyses to highlight repeat failures that need permanent fixes

Example: Monitoring RDS Outages
Create CloudWatch dashboards combining RDS event metrics and instance errors for real-time visibility

Inform SLOs & Error Budgets
Align SLO targets and error-budget policies with historical failure data to set realistic thresholds

Enable Data-Driven Decisions
Leverage failure metrics to prioritize reliability work and maximize system stability

METRICS FOR OUTAGE TRACKING

- **MTTD (Mean Time to Detect)**
  Measures how fast issues are detected after they occur.

- **MTTR (Mean Time to Resolve)**
  Tracks the average time taken to fully resolve incidents.

- **Impact Metrics**
  Quantify scope: number of users, services, or systems affected.

- **Example: MTTR for API Outages**
  Use CloudWatch & X-Ray to track how long it takes to restore a failing endpoint.

- **Why It Matters**
  These metrics spotlight bottlenecks and guide continuous improvement.

TAGGING OUTAGES

Tag by Cause, Severity, and Impact
Example tags: "network-failure", "high-severity", "user-facing".

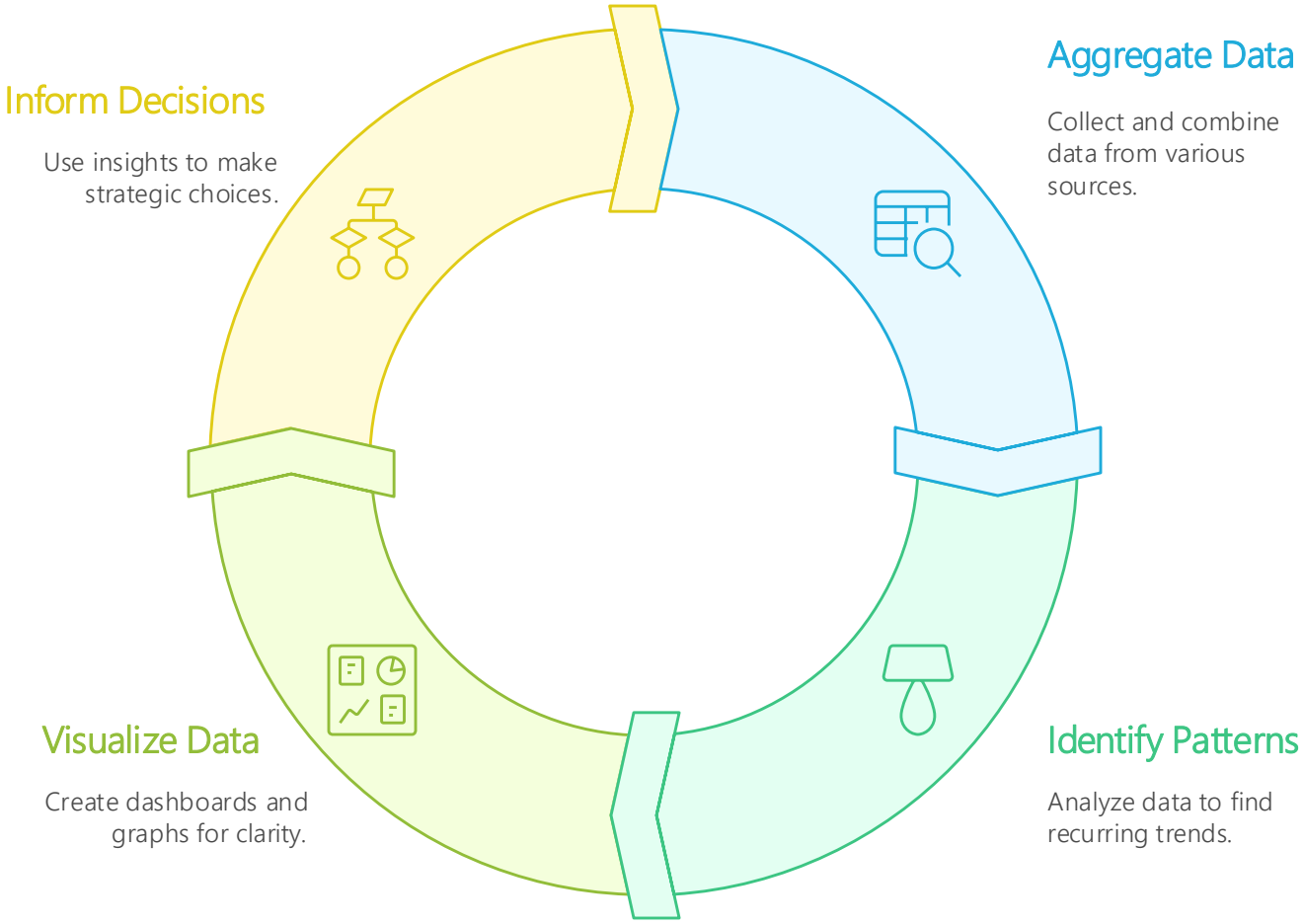
Enables Filtering and Postmortem Review
Easily analyze outage trends over time.

Supports Data-Driven Prioritization
Focus on frequent or high-impact failures.

Consistent Tagging Conventions
Standardize across teams for accurate tracking.

Example: Filtering for Repeated DB Timeouts
Reveals if the root cause is architectural or operational.

AGGREGATING OUTAGE DATA



- **Aggregate Incidents to Identify Patterns**
  Look for trends in timing, services affected, and root causes.
- **Generate Monthly or Quarterly Reports**
  Example: "5 DB outages, 3 linked to scaling limits."
- **Leverage CloudWatch Metrics for Aggregation**
  Track custom metrics like outage.count, MTTR, or impacted.users.
- **Visualize Trends with Dashboards and Graphs**
  Use CloudWatch Dashboards or third-party tools (Grafana, QuickSight).
- **Drive Strategic Improvements**
  Data informs capacity planning, architecture changes, and resource allocation.

LAB 6: DISTRIBUTED TRACING WITH AWS X-RAY

Goal: Build a simple multi-endpoint web app with AWS X-Ray tracing enabled.
Inject latency and failures, and diagnose them using X-Ray service maps and CloudWatch Logs Insights.

Skills You'll Learn

- Instrument Flask apps for X-Ray
- Visualize request flows with service maps
- Diagnose slow or failed requests using trace data
- Correlate logs with traces using CloudWatch Logs Insights

Instructions: Lab6.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