SITE REALIABILITY ENGINEERING - FOUNDATIONS

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

LOGISTICS

Class Hours:

- Instructor will set class start and end times.
- 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

Miscellaneous:

- Courseware
- Bathroom
- Fire drills

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, Architects, mixed experience.

AWS Focus: Hands-on labs using EC2, IAM, CloudWatch.

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

COURSE STRUCTURE

Day 1:

Introduction to SRE and Core Principles.

Day 2:

Monitoring, 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.
- Apply concepts to your AWS migration projects.

AGENDA FOR DAY 1

Introduction to SRE: History, principles, benefits.

Production Environment: Components, AWS equivalents.

Embracing Risk: Error budgets, risk management.

Service Level Objectives: SLIs, SLOs, SLAs.

Eliminating Toil: Automation, process improvement.

Labs and pop quizzes to reinforce learning.

WHAT IS SRE?

Origin	Google
Goal	Scalability, reliability, efficiency
Key tenets	Embrace risk, automate, monitor, SLOs
Benefits	Fewer outages, faster releases, reduced toil

HISTORY OF SRE

- Developed at Google in the early 2000s.
- Response to growing system complexity.
- Formalized in *Site Reliability Engineering* book (O'Reilly).
- Adopted by companies like Netflix, LinkedIn.
- Evolved to address modern cloud environments.

SRE VS. TRADITIONAL SYSADMIN

Traditional system administrators react to issues, manually maintaining servers.

SREs proactively use software to enhance reliability, coding tools for automation and monitoring.

Characteristic	Sysadmin	SRE
Approach	Reactive	Proactive
Method	Manual	Software-driven
Focus	System maintenance	Reliability
Tools	N/A	Automation, monitoring
Culture	N/A	Blame-free
Example	Manual server management	Automating EC2 scaling

KEY TENETS OF SRE – PART 1

- Embrace risk with error budgets.
- Automate repetitive tasks.
- Monitor systems effectively.
- Define and track SLOs.
- Eliminate toil for efficiency.

KEY TENETS OF SRE – PART 2

Error budgets: Balance reliability and innovation.

Automation: Use tools like AWS CodePipeline.

Monitoring: AWS CloudWatch for real-time insights.

SLOs: Set measurable reliability targets.

Toil reduction: Free time for engineering.

BENEFITS OF SRE

- Enhanced system reliability and uptime.
- Faster feature releases via automation.
- Reduced operational costs.
- Improved DevOps collaboration.
- Better user experience through performance.

SRE ROLES & RESPONSIBILITIES

Design and operate scalable systems.

Code automation and monitoring tools.

Respond to incidents, conduct postmortems.

Define and monitor SLOs.

Collaborate with developers for reliability.

SRE AT GOOGLE

- Embedded with development teams.
- Own service reliability end-to-end.
- Blend software and operations expertise.
- Use tools like Borg for orchestration.
- Culture of accountability and learning.

GOOGLE'S SRE MISSION

Engineering Operations: Software engineers run production.

50% Ops Cap: SRE time split between ops and engineering projects.

Error Budgets: Balance reliability and feature velocity.

Blameless Postmortems: Root-cause focus, not blame.

Self-Healing Systems: Automate recovery where possible.

PRIORITIZING RISKS

- **Adopted by major banks**: Institutions like Goldman Sachs and JPMorgan Chase have integrated SRE to bolster the reliability of their digital services.
- **Focus on compliance and security**: SRE practices in finance prioritize adherence to regulations and the protection of sensitive data.
- **Emphasis on automation and monitoring**: Implementing automated systems and robust monitoring to proactively manage risks and system performance.
- **Integration with AI and advanced tools**: Utilizing AI-driven SRE agents to predict and prevent failures, ensuring uninterrupted services.
- **Alignment with regulatory frameworks**: SRE aids in meeting mandates like the Digital Operational Resilience Act (DORA) by enhancing risk management and operational resilience.

CHALLENGES OF IMPLEMENTING SRE

- Cultural shift from traditional operations.
- Investment in automation tools, training.
- Need for dual-skilled engineers.
- Balancing reliability and innovation.
- Resistance to change in teams

PRODUCTION ENVIRONMENT, FROM THE VIEWPOINT OF AN SRE

PRODUCTION ENVIRONMENT OVERVIEW

Components: Hardware, software, storage, networking, monitoring.

AWS equivalents: EC2, EBS, VPC, CloudWatch.

Importance: Foundation for reliable services.

Challenges: Scalability, security, cost.

Goal: Ensure uptime, performance.

PRODUCTION ENVIRONMENT LAYERS

Layer	Description
Compute Hardware	Physical servers, Nitro hypervisors
System Software	Hypervisor, OS, orchestrators (EKS, ECS)
Storage	Block (EBS, instance store), object (S3), file (EFS)
Networking	VPC, subnets, route tables, ALB, CloudFront
Monitoring	CloudWatch metrics, logs, X-Ray traces
Alerting	SNS, EventBridge, PagerDuty integrations

There are six layers SREs must understand in production.

COMPUTE HARDWARE

Physical servers, networking equipment.

AWS: EC2 instances, Elastic Network Interfaces.

Scalability: Auto Scaling groups.

Fault tolerance: Multi-AZ deployments.

Cost: Pay-as-you-go model.

SYSTEM SOFTWARE (HYPERVISOR & OS)

Operating systems, middleware, applications.

AWS: Linux/Windows on EC2, RDS for databases.

Ensures compatibility, performance.

Regular updates for security, stability.

Example: Ubuntu on EC2 for web servers.

Nitro Hypervisor: Minimalistic hypervisor for virtual machine lifecycle commands and device assignment

Guest OS Choices: Amazon Linux, Ubuntu, Windows—balance stability, patch cadence, support.

Orchestration Layers: ECS/EKS for container workloads; Auto Scaling Groups for VMs

PLATFORM & ORCHESTRATION

AWS ECS/EKS: Managed container orchestration platforms, integrating with IAM and VPC natively

AWS Lambda: Serverless compute for event-driven tasks—no server management needed

Infrastructure as Code: CloudFormation/CDK/Terraform define platform stacks programmatically

STORAGE IN PRODUCTION

Types: Block, object, file storage.

AWS: EBS, S3, EFS.

Considerations: Durability, availability, speed.

Use cases: EBS for databases, S3 for backups.

Backup: S3, Glacier for archiving.

BLOCK STORAGE (EBS & INSTANCE STORE)

Amazon EBS: Durable, provisioned IOPS, encryption at rest, snapshots for backups

Instance Store: Ephemeral NVMe volumes for scratch data—high IOPS, but data lost on stop/terminate

EBS Volume Types: gp3/gp2 for general purpose, io2/io1 for high-performance databases

OBJECT & FILE STORAGE (S3 & EFS)

Amazon S3: Highly durable object store for logs, artifacts, and static assets

Amazon EFS: POSIX-compliant, shared file systems for Linux workloads with burst throughput

Lifecycle Policies: Auto-archive S3 objects to Glacier to control costs

NETWORKING

Load balancers, firewalls, VPNs.

AWS: Elastic Load Balancing, Security Groups, VPC.

Ensures security, scalability, reliability.

Example: ALB for distributing web traffic.

VPC: Isolated network for resources.

Amazon VPC: Virtual network with IP ranges, subnets, and gateway attachments

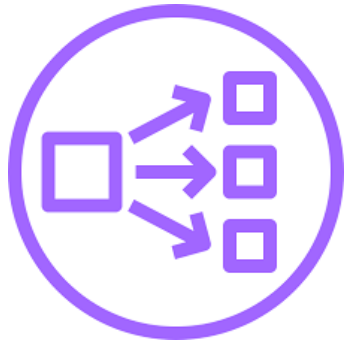
Subnets & Route Tables: Public/private subnet separation; route tables direct traffic via Internet Gateway, NAT Gateway

Security Groups & NACLs: Instance-level and subnet-level firewalls for traffic control

TRAFFIC DISTRIBUTION

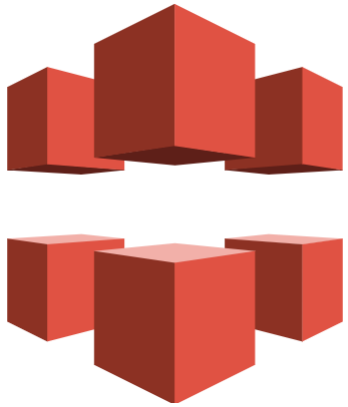
SREs combine these services to optimize both internal service-to-service calls and public-facing traffic.

Application Load Balancer (ALB): L7 routing with host/path-based rules, health checks, sticky sessions



Network Load Balancer (NLB): L4 high-performance routing; static IP support

Amazon CloudFront: Global CDN for caching static/dynamic content at edge locations



MONITORING & OBSERVABILITY

Tools: CloudWatch, X-Ray, CloudTrail.

Tracks performance, detects issues.

Best practices: Centralized logging, real-time alerts.

CloudWatch: Metrics, logs, alarms.

X-Ray: Traces requests in distributed systems.

Amazon CloudWatch

ALERTING & INCIDENT RESPONSE

Amazon SNS & EventBridge: Route alarms to multiple endpoints (email, SMS, Lambda)

PagerDuty Integration: Forward SNS notifications to PagerDuty for on-call orchestration

Runbooks & Playbooks: Automate initial triage via Lambda or Systems Manager Automation

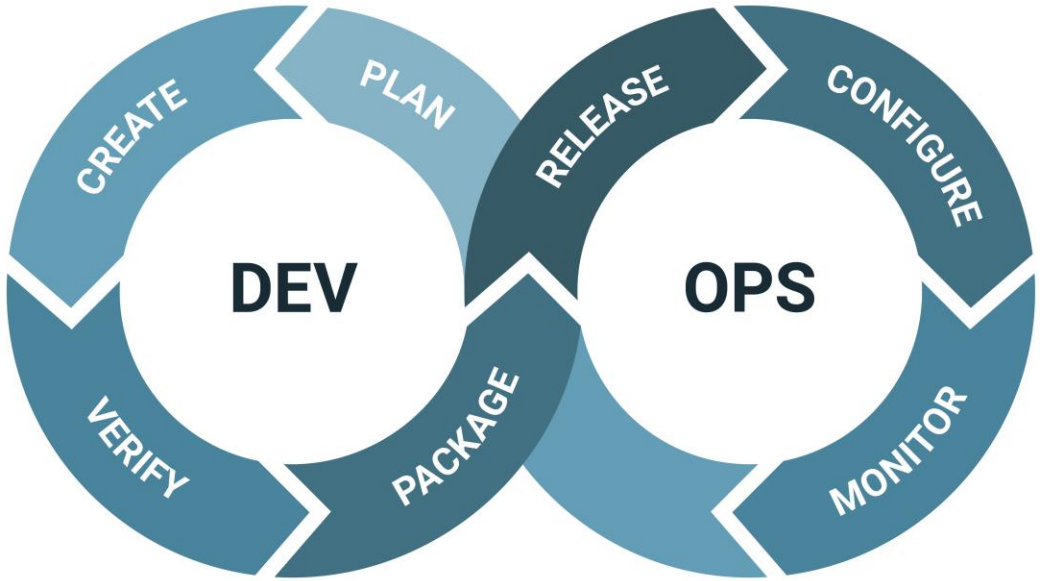
PagerDuty

CHALLENGES IN PRODUCTION

- **Scalability:** Handling traffic spikes.
- **Reliability:** Minimizing downtime.
- **Security:** Protecting against threats.
- **Cost:** Optimizing resource usage.
- **Performance:** Ensuring low latency.

SRE SOFTWARE DEVELOPMENT LIFECYCLE

DEVOPS PRINCIPLES



- **Cultural Shift:** Embrace collaboration and continuous learning.
- **Automation:** Treat infrastructure as code to reduce errors.
- **Measurement:** Monitor everything; use metrics to guide decisions.
- **Sharing:** Transparent dashboards and postmortems.
- **Continuous Feedback:** Short feedback loops in CI/CD pipelines.

SRE AND DEVOPS

DevOps: Collaboration, faster delivery.

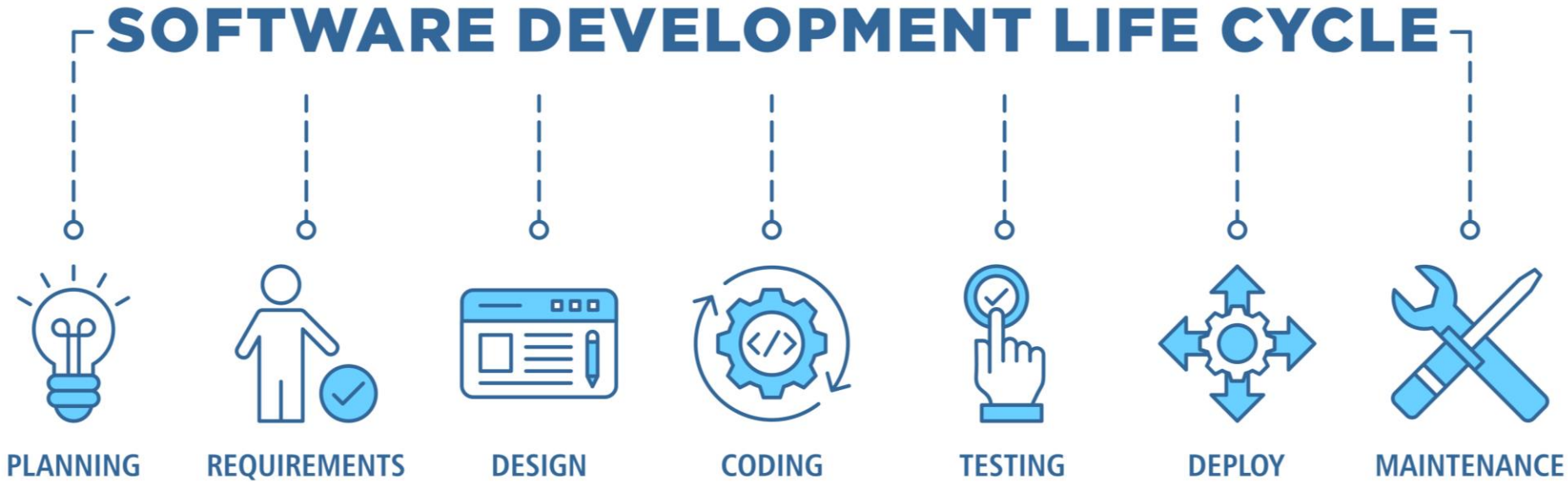
SRE: Engineering rigor, reliability focus.

Together: Speed and stability.

Example: SRE ensures CI/CD pipelines are reliable.

AWS tools enhance both practices.

SRE IN THE SDLC



- Integrates reliability into development lifecycle.
- Defines SLOs during planning.
- Automates testing, deployment phases.
- Monitors production performance.
- Aligns with business architecture goals.

SRE SOFTWARE DEVELOPMENT LIFECYCLE

Plan: Define objectives, SLOs, and error budgets.

Code: Develop automation, infrastructure as code (IaC).

Build & Test: CI/CD pipelines with unit, integration, performance tests.

Deploy: Automated deployments (Blue/Green, Canary) via CodeDeploy.

Operate & Learn: Monitor, collect feedback, conduct postmortems.

SLI, SLO, SLA OVERVIEW

SLI

Measure of service health

SLO

Target for SLI

SLA

Contractual guarantee with penalties

ESTABLISHING SLIS WITH CLOUDWATCH

- **Map Critical User Journeys**: Identify key workflows such as login, checkout, and API requests to determine where reliability is paramount.
- **Define SLIs**: Select quantifiable metrics like latency, error rate, and throughput that reflect user experience and system performance.
- **Leverage AWS Monitoring Tools**:
  - **CloudWatch Metrics**: Monitor standard and custom metrics across AWS services.
  - **CloudWatch Logs**: Collect and analyze log data for deeper insights.
  - **AWS X-Ray**: Trace requests to pinpoint performance bottlenecks.
  - **CloudWatch Application Signals**: Automatically collect key metrics like latency and availability for services and operations .
- **Set Appropriate Granularity**: Choose time intervals (e.g., 1-minute, 5-minute) that align with your monitoring needs and alerting thresholds.
- **Visualize with CloudWatch Dashboards**: Create dashboards to display SLIs using widgets for real-time monitoring and historical analysis .

https://aws.amazon.com/blogs/mt/slos-made-easier-with-nobl9-and-cloudwatch-metrics-insights

SETTING SLO AND MANAGING ERROR BUDGETS

Define Availability Targets

Establish clear SLOs for uptime

Implement Review Cadence

Conduct monthly SLO review meetings

Communicate Transparently

Regularly share SLO status and error budget usage with stakeholders

Calculate Error Budgets

Determine allowable downtime based on SLOs

Enforce Policies

Pause or limit risky deployments when the error budget is exhausted

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

LAB 1: SLI, SLO & ERROR BUDGETS

Goal: Establish reliability metrics and monitor a basic web service using SRE principles.

What You'll Do

Launch and configure a web server on EC2
Install and run the CloudWatch Agent
Create metric filters and CloudWatch alarms
Define:

- **SLIs** – What to measure (e.g., successful HTTP requests)
- **SLOs** – Performance target (e.g., 99.9% availability)
- **Error Budgets** – Acceptable failure thresholds

Instructions: Lab1.md

ELIMINATING TOIL

WHAT IS TOIL?

Definition: Toil is the routine work tied to running a service that is manual, repetitive, and devoid of long-term value

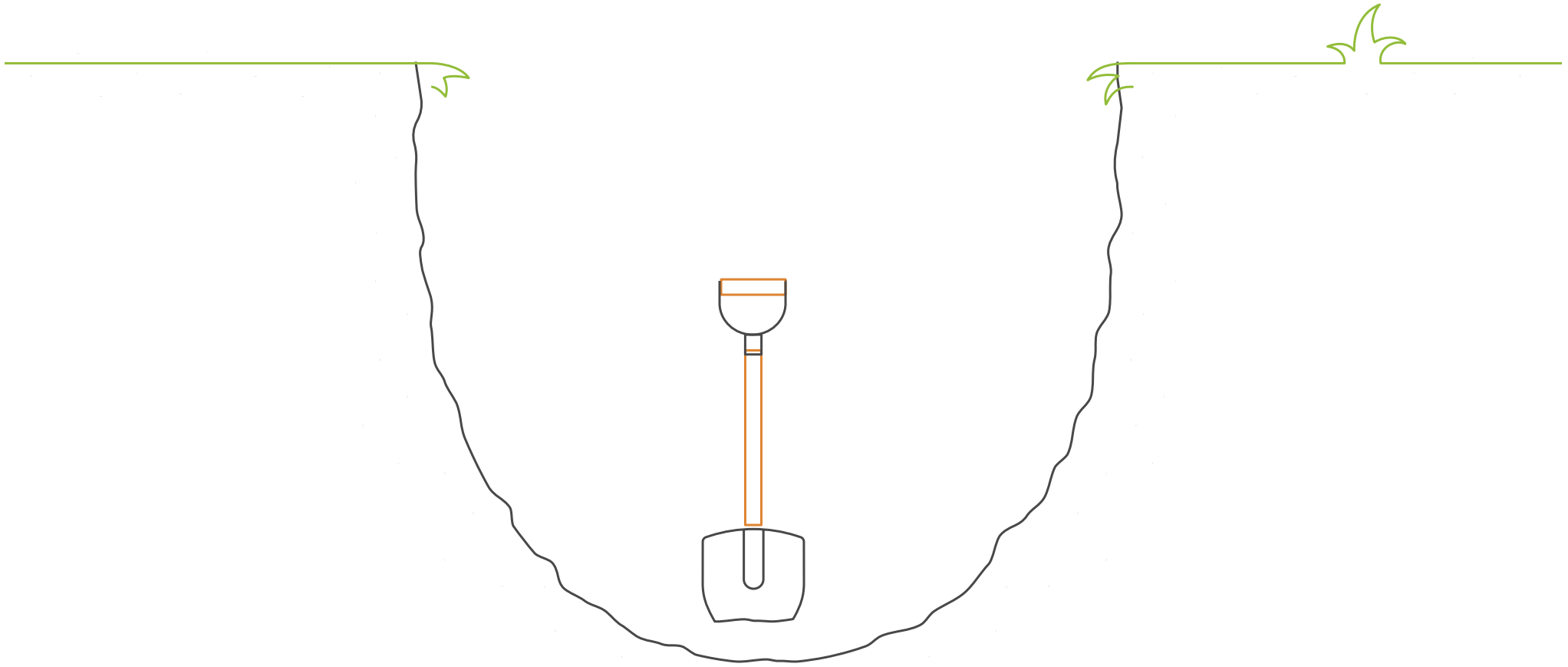
Characteristics: By nature toil tasks grow with the size of the service and consume engineering effort (e.g., manual config changes, repetitive ticket-driven fixes)

.

Automatable: These tasks are often automatable – the goal is to replace them with code or tools so engineers can focus on higher-value work

WHY TOIL MATTERS?

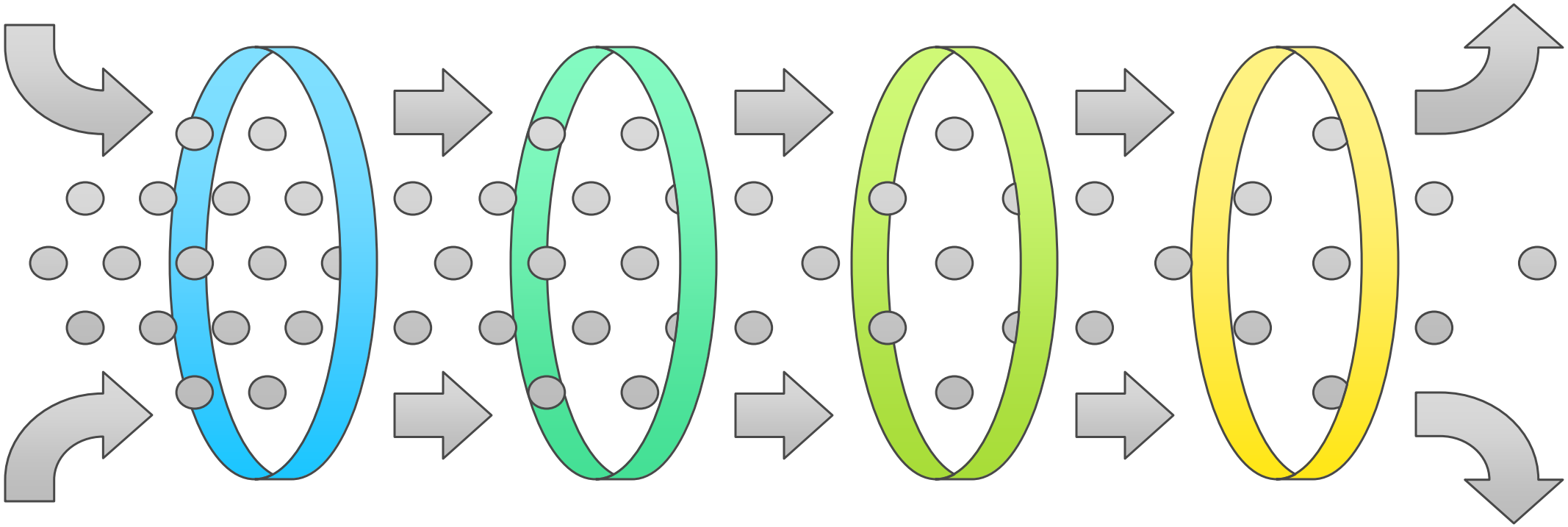
Excessive toil reduces innovation, scalability, and productivity.



IDENTIFYING TOIL

- **Log Manual Tasks:** Encourage engineers to record routine operational tasks (via ticket tags or a “toil log”) to see patterns over time.
- **Measure Time Spent:** Track how much time is spent on recurring tasks (builds, deployments, incident handling) versus projects.
- **Use Observability:** Leverage CloudWatch metrics and logs to surface repetitive patterns (e.g. frequent alarm acknowledgments, repeated Lambda errors).
- **Growth Indicators:** Look for tasks that grow linearly with service size (e.g. adding a new server requires many manual steps) – these signal automation candidates.

PRIORITIZING TOIL FOR ELIMINATION



Assess Impact

Evaluate frequency, time cost, and risk

Identify High-Value Targets

Focus on tasks with high frequency and duration

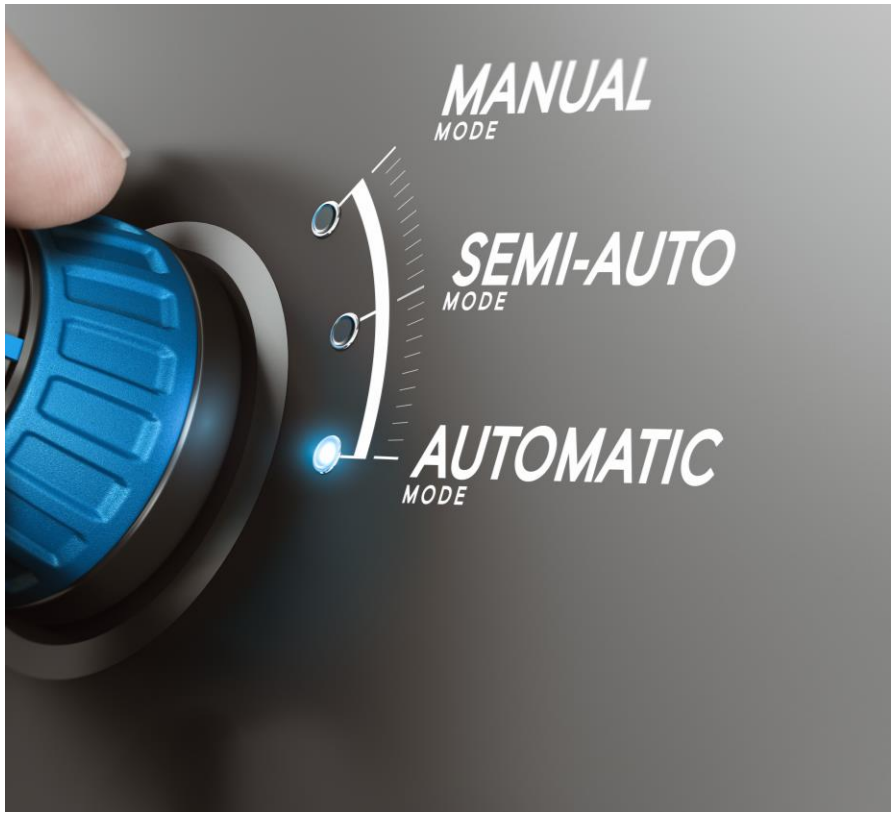
Apply Frameworks

Use models like Eisenhower matrix for ranking

Balance Quick Wins

Combine easy scripts with larger projects

AUTOMATING DEPLOYMENT TOIL



CI/CD Pipelines: Adopt AWS CodePipeline and CodeBuild to automate the software release process (build, test, deploy).

Replace Scripts: Turn manual deployment scripts into pipeline steps. For example, CodePipeline can pull from Git, build with CodeBuild, and deploy to ECS or Lambda without human steps.

Infrastructure as Code: Use CloudFormation or AWS CDK for provisioning, eliminating manual AWS Console work.

Outcome: Automated pipelines ensure consistent, faster releases – teams often see cycle times shrink from days to hours.

AUTOMATING INCIDENT RESPONSE

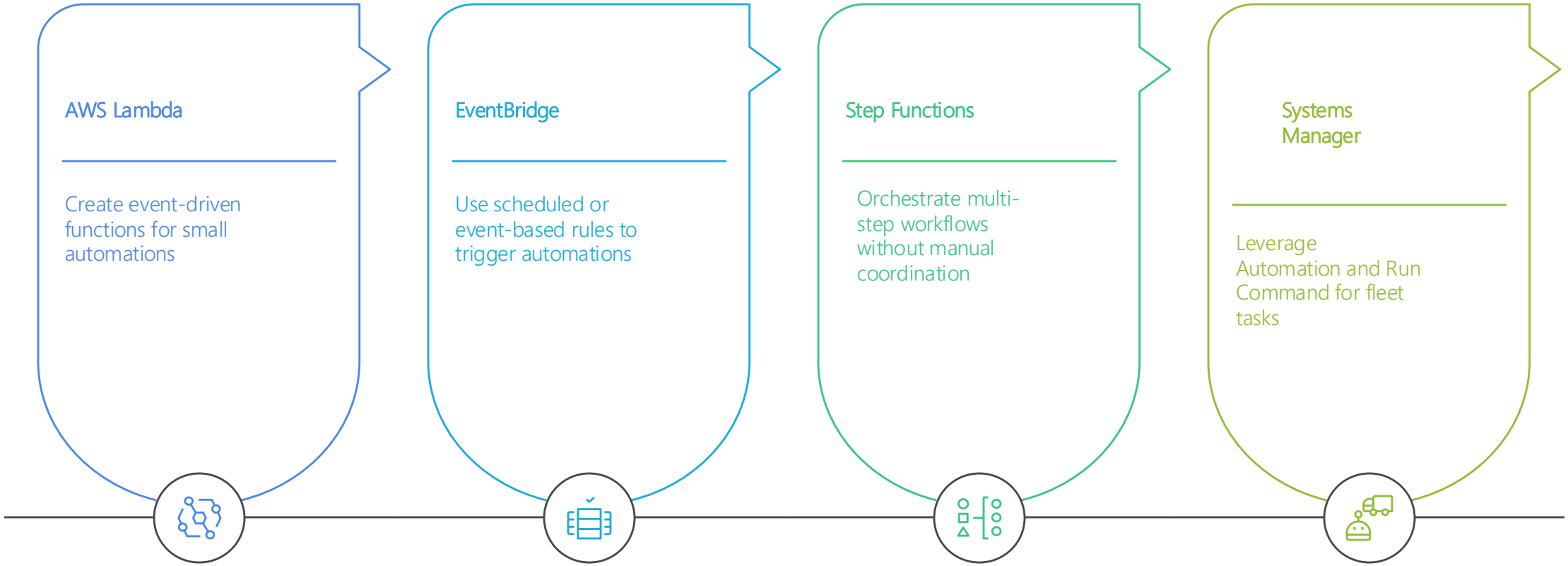
CloudWatch Alarms + Auto-Actions: Use Amazon CloudWatch alarms with built-in actions to automatically reboot or recover EC2 instances when they fail health checks.

Lambda Remediations: Send CloudWatch alarms to an SNS topic or EventBridge rule that triggers AWS Lambda functions for custom fixes (e.g. clearing queues, restarting services).

SSM Automation: Employ AWS Systems Manager Automation documents (runbooks) like **AWS-RestartEC2Instance** to programmatically repair resources.

Auto-Heal Example: Detect an unhealthy EC2 (via Alarm), automatically restart it, and notify the team – reducing on-call toil.

TOOLING FOR TOIL ELIMINATION



MEASURING TOIL REDUCTION

Define KPIs: Track metrics like *time saved per week*, *percentage of tasks automated*, or *mean time between manual interventions*.

Dashboards: Use CloudWatch Dashboards to visualize trends (e.g., count of automated vs. manual deployments, or Lambda invocations replacing human actions).

Cost Metrics: Leverage AWS Cost Explorer or AWS Compute Optimizer to quantify savings from removing idle resources (e.g., fewer AMIs/EBS and related costs).

Error Reduction: Monitor reliability improvements (e.g., lower error rates or downtime), which indicates successful automation.

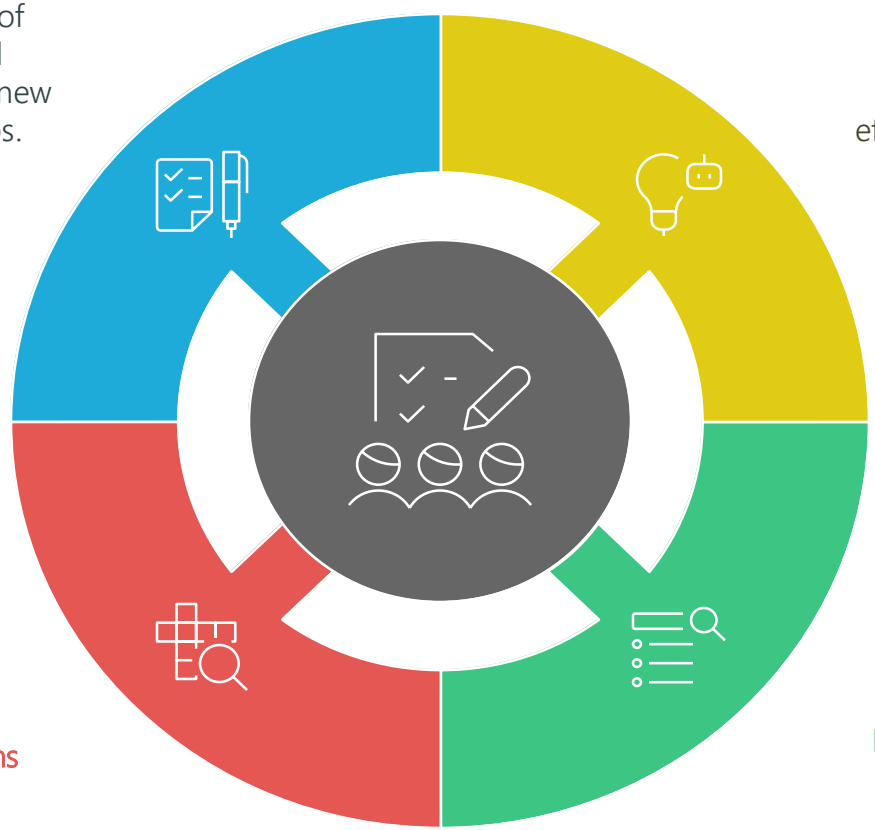
CONTINUOUS IMPROVEMENT

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.

EMBEDDING A NO-TOIL CULTURE

"If You Click It, Script It": Enforce that any action done in the AWS console should also have an API/automation option (e.g., CloudFormation, SDK, or CLI scripts).

Onboarding & Standards: Make automation part of the team's definition of done. New hires should learn automation tools (Terraform, CDK, etc.) from day one.

Celebrate Success: Share automation wins in team meetings or Slack channels (e.g., "we saved 10hrs this week by automating X"), and recognize contributors.

Leadership Support: Ensure management sets targets for toil reduction as key metrics and rewards time spent automating over repetitive tasks.

POP QUIZ:

You need to minimize manual toil in your AWS environment by automating repetitive operational tasks. Which approach aligns best with SRE principles?

A. Continue manual interventions until a full rewrite is complete
B. Automate tasks using AWS Lambda or Systems Manager Automation documents
C. Delegate all scripting to a separate “ops scripting” team
D. Rely solely on third-party managed services for every task

POP QUIZ:

You need to minimize manual toil in your AWS environment by automating repetitive operational tasks. Which approach aligns best with SRE principles?

A. Continue manual interventions until a full rewrite is complete
B. Automate tasks using AWS Lambda or Systems Manager Automation documents
C. Delegate all scripting to a separate “ops scripting” team
D. Rely solely on third-party managed services for every task

POP QUIZ:

According to SRE tenets, which task qualifies as “toil” and should be automated?
A. Designing error budgets for a new service
B. Conducting a quarterly reliability review
C. Developing a new feature API
D. Writing a one-off alert notification rule

POP QUIZ:

According to SRE tenets, which task qualifies as “toil” and should be automated?
A. Designing error budgets for a new service
B. Conducting a quarterly reliability review
C. Developing a new feature API
D. Writing a one-off alert notification rule

LAB 2: AUTOMATING TOIL

Goal: Reduce manual operational overhead by automating patch management and EBS snapshot cleanup using native AWS services.

What This Lab Covers

- Launch and tag EC2 instances
- Patch instances using AWS Systems Manager Automation
- Clean old EBS snapshots with an automated Lambda function
- Schedule cleanup using Amazon EventBridge

Instructions: Lab2.md

MONITORING DISTRIBUTED SYSTEMS

INTRODUCTION TO MONITORING

Definition of Observability: Ability to infer system state from metrics, logs, traces.

Three Pillars: Metrics (CloudWatch), Logs (CloudWatch Logs), Traces (X-Ray).

Monitoring vs. Observability: Monitoring alerts; observability answers “why.”

AWS Tools: CloudWatch for metrics and logs, X-Ray for tracing.

Business Impact: Early anomaly detection prevents costly outages.

WHY MONITOR DISTRIBUTED SYSTEMS

Detect failures & degradation: Alert on service errors or latency spikes to invoke rapid responses.

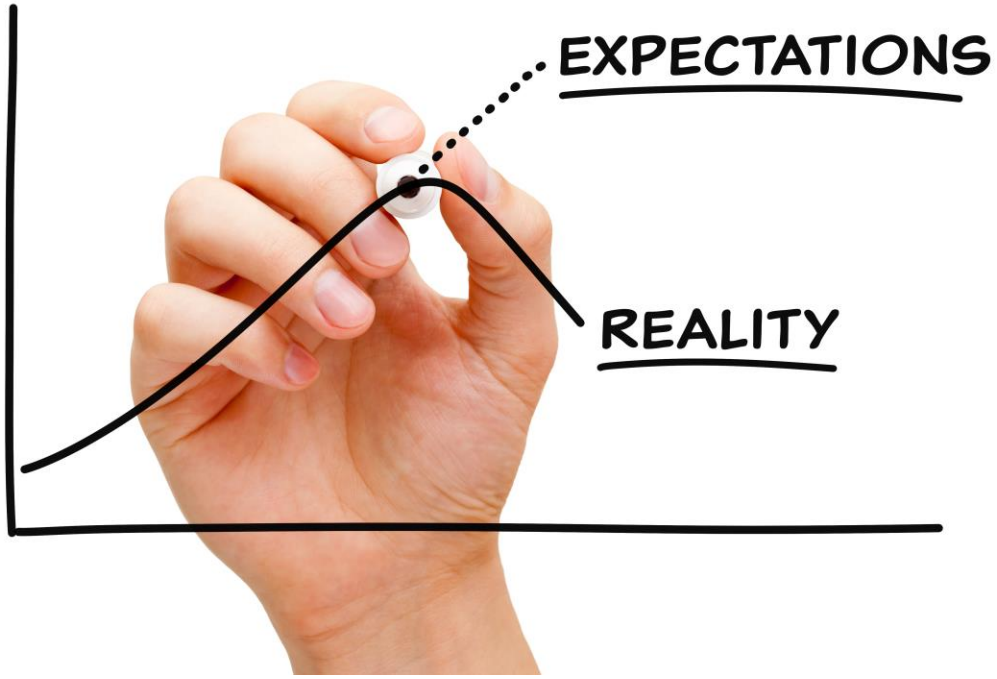
Analyze trends: Track long-term metrics (e.g., database growth, user counts) to inform capacity planning.

Compare performance: Benchmark across releases or experiments (e.g., A/B tests, hardware changes).

Build dashboards: Present key metrics (often "golden signals" – latency, traffic, errors, saturation) for real-time visibility

Post-incident debugging: Correlate anomalies (e.g., latency spikes) with events or config changes to root-cause problems

SETTING REASONABLE EXPECTATIONS FOR MONITORING



Dedicated effort: Monitoring distributed apps is non-trivial; SRE teams often assign 1–2 engineers solely to build and maintain it.

Favor simplicity: In practice, Google SREs "trend toward simpler and faster monitoring systems" and avoid overly complex anomaly detectors.

Avoid brittle rules: Complex dependency-based alerts rarely work at scale. Prefer straightforward conditions (e.g., *"alert if error rate > X"*).

Keep alert rules clear: Any rule that pages someone should be simple to understand and unlikely to be ignored

SYMPTOMS VS CAUSES

Symptom

Identifies the immediate issue, like HTTP 500 errors or slow responses.

Cause

Explains the underlying reason for the issue, such as database connection refusals or high CPU usage.

BLACK BOX VS WHITE BOX MONITORING

Black-box monitoring: Probes from outside (synthetic or end-to-end checks) that detect *actual user-visible failures*. Examples include synthetic API calls or customer checkout tests.

White-box monitoring: Instrumentation of internals (metrics, logs, traces) to detect emerging issues (e.g., rising CPU, memory leaks).

Use both: Rely heavily on white-box for early detection (since it can see failures hidden by retries), and use black-box to ensure real user paths work.

Perspective: What's a symptom for one team can be a cause for another. For instance, slow disk I/O (white-box metric for storage team) may show up as poor query latency (black-box symptom for users)

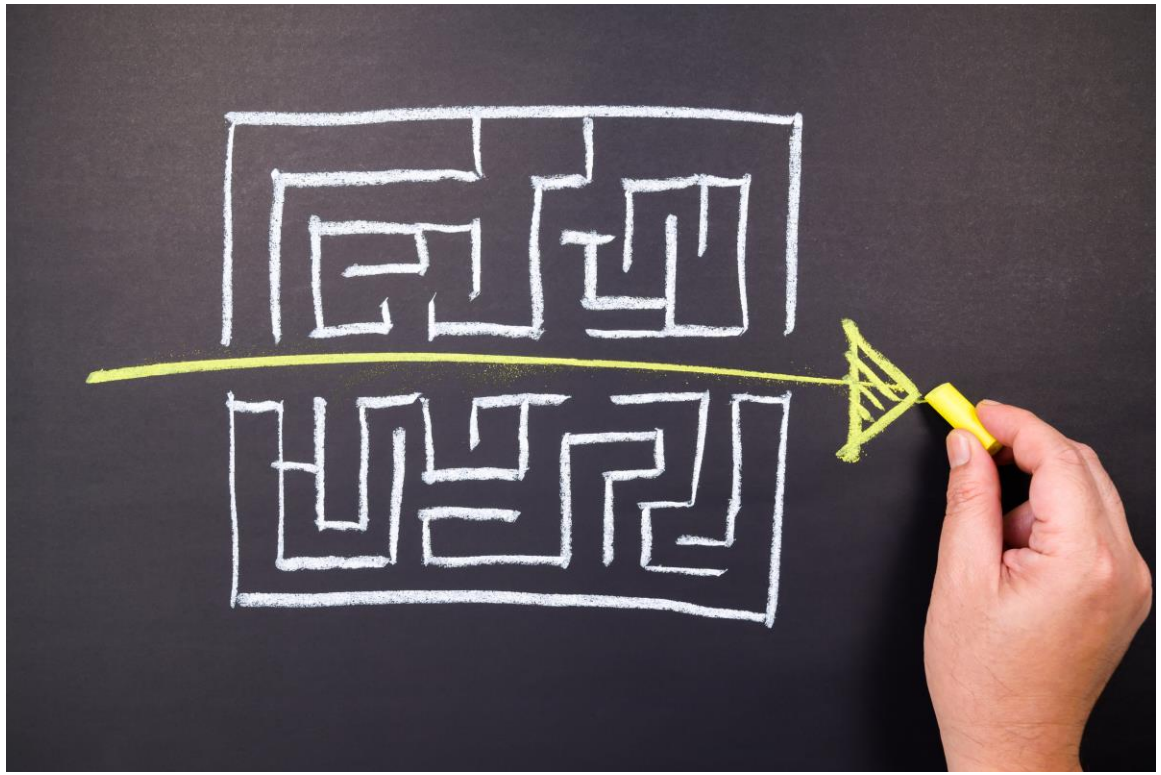
DESIGNING "AS SIMPLE AS POSSIBLE" MONITORING SYSTEMS

Prune complexity: Don't overload with thresholds on every percentile/metric. Each page rule should be simple and specific.

Retire unused data: If a metric isn't used in dashboards or alerts, or an alert triggers < quarterly, consider removing it.

Separate concerns: Keep monitoring (metrics/alerts) distinct from heavy tooling (profilers, debuggers). Combine only via stable APIs if needed.

Reduce fragility: A simpler monitoring setup is easier to maintain. Avoid building a huge dependency graph of alerts (fragile under change)



CASE STUDY: BIGTABLE SRE AND OVER-ALERTING

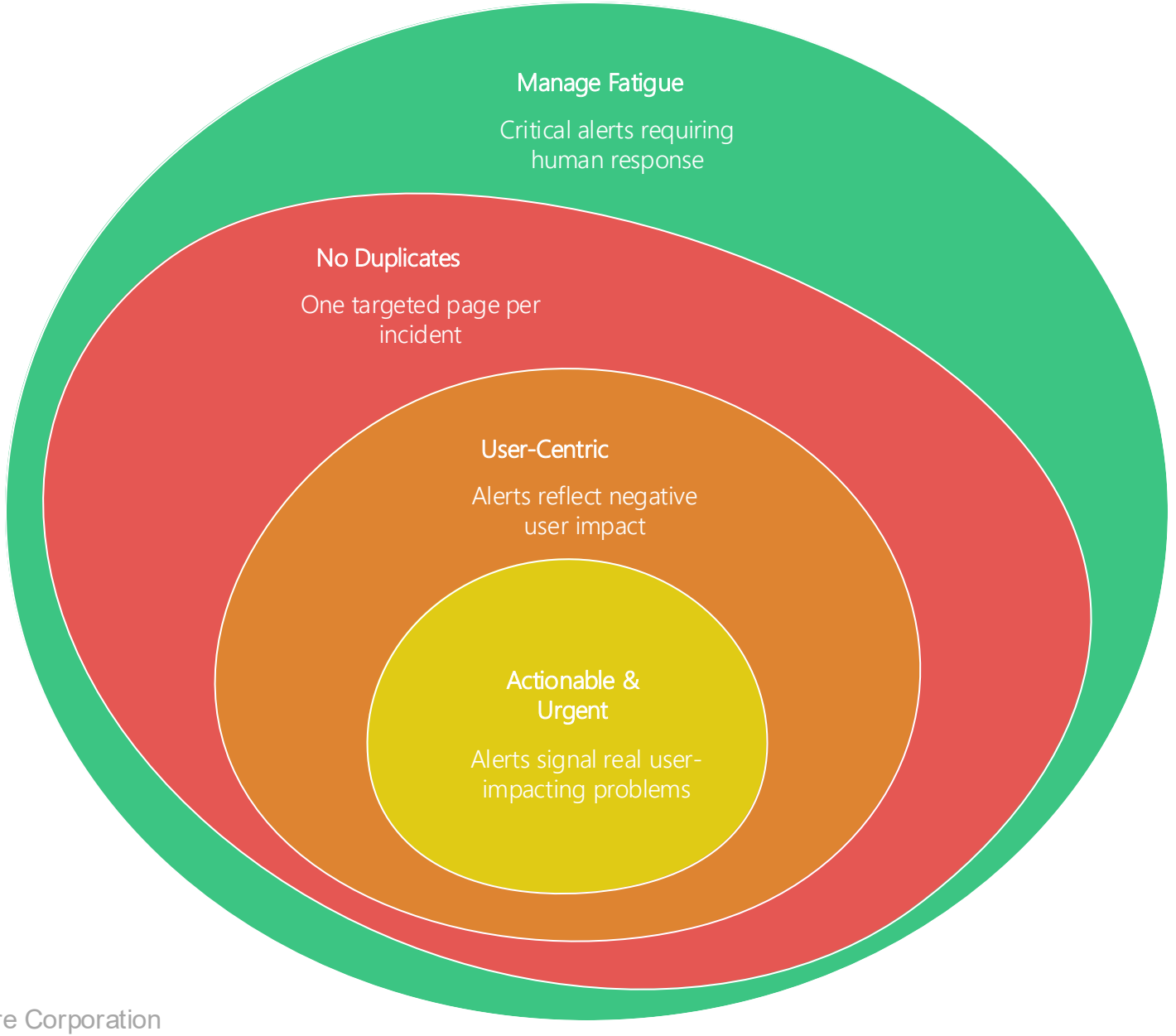
Original setup: Bigtable's SLO was based on *mean request latency*, but the slow 5% of requests (tail latency) dominated performance.

Too many alerts: As SLO approached/exceeded, email and page alerts fired constantly. The team spent most of its time triaging noisy alerts, many of which were known infra issues.

SRE fix: They temporarily relaxed the SLO to the 75th-percentile latency and disabled the flood of low-priority alerts.

Outcome: On-call engineers finally had breathing room. They addressed underlying Bigtable and storage bugs instead of firefighting. This trade-off "allowed us to make faster progress toward a better service"

ALERTING PHILOSOPHY FOR DISTRIBUTED SYSTEMS



CLOUDWATCH METRICS BASICS

Namespaces: Logical containers (e.g., AWS/EC2, AWS/Lambda).

Dimensions: Key/value pairs to filter metrics (InstanceId, FunctionName).

Statistics: Sum, Average, Min, Max, Percentiles.

Periods: Granularity of data points (1-min, 5-min).

Alarms: Trigger actions when thresholds crossed.

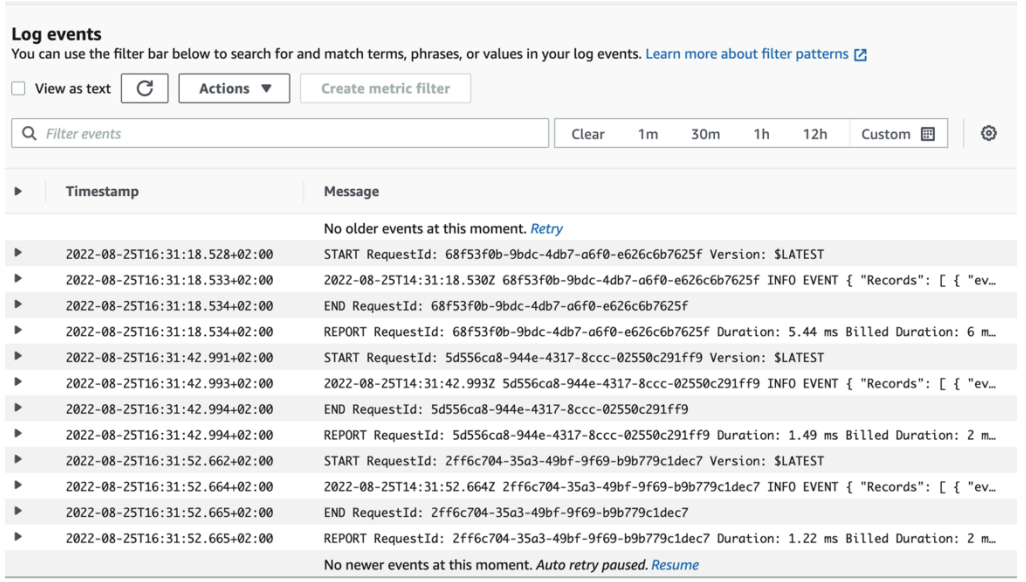
Amazon CloudWatch

CLOUDWATCH LOGS

Log Groups: Collections of log streams with shared retention and monitoring settings.

Log Streams: Sequence of log events from a single source (EC2, Lambda).

Retention Policies: Automate deletion of old logs to control cost.



Metric Filters: Create CloudWatch metrics from log patterns.

Insights: Query logs with CloudWatch Logs Insights for ad-hoc analysis.

DISTRIBUTED TRACING WITH AWS X-RAY

Segments and Subsegments: Track requests and internal operations.

Sampling: Control data volume by sampling requests.

Service Map: Visual representation of application architecture.

Annotations/Metadata: Add custom data to traces for filtering.

Integration: SDKs for Java, Node.js, Python, Go.

CAPACITY PLANNING WORKFLOW

Collect Metrics: Gather key performance indicators such as CPU utilization, memory usage, network throughput, and database connections using AWS CloudWatch and other monitoring tools.

Define Thresholds: Establish maximum safe utilization levels (e.g., 70% CPU usage) to determine when scaling actions should be triggered.

Plan Resources: Calculate the required number and type of EC2 instances based on current and projected workloads, considering factors like instance performance and cost.

Validate with Load Testing: Perform load testing using tools like Apache JMeter to simulate user traffic and validate that the planned resources can handle the expected load.

Adjust Auto Scaling Policies: Fine-tune Auto Scaling settings, including cooldown periods and scaling thresholds, to ensure responsive and stable scaling behavior

DEMAND FORECASTING BASICS

Historical Data: Analyze past usage from CloudWatch logs.

Business Events: Factor in planned releases or campaigns.

Seasonal Trends: Account for holidays, promotions.

Buffer Capacity: Maintain safety margin (e.g., +20%).

AWS Predictive Scaling: Leverage ML-based forecasts.

CAPACITY PLANNING WORKFLOW

Collect Metrics: Gather key performance indicators such as CPU utilization, memory usage, network throughput, and database connections using AWS CloudWatch and other monitoring tools.

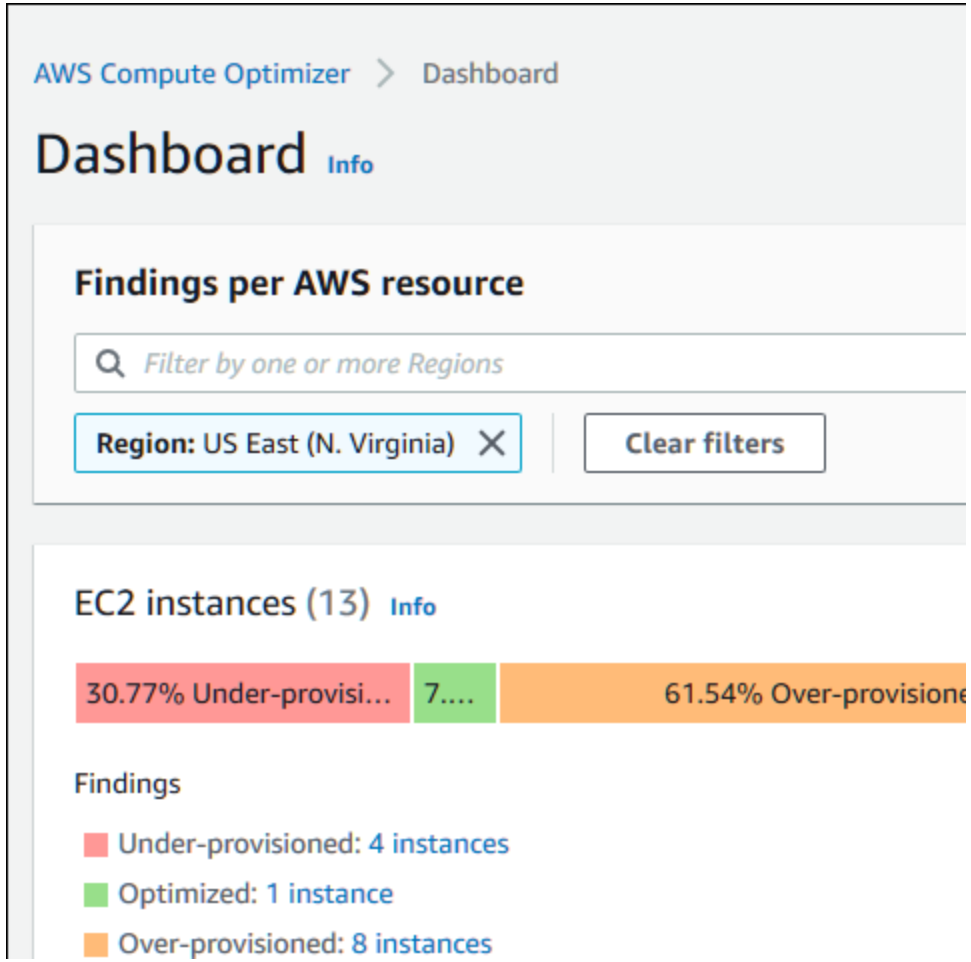
Define Thresholds: Establish maximum safe utilization levels (e.g., 70% CPU usage) to determine when scaling actions should be triggered.

Plan Resources: Calculate the required number and type of EC2 instances based on current and projected workloads, considering factors like instance performance and cost.

Validate with Load Testing: Perform load testing using tools like Apache JMeter to simulate user traffic and validate that the planned resources can handle the expected load.

Adjust Auto Scaling Policies: Fine-tune Auto Scaling settings, including cooldown periods and scaling thresholds, to ensure responsive and stable scaling behavior

EFFICIENCY AND PERFORMANCE PRINCIPLES



Right-Sizing Resources: Utilize AWS Compute Optimizer to analyze your workloads and receive recommendations.

Implement Caching Strategies: Enhance application performance by integrating caching solutions like Amazon ElastiCache.

Adopt Serverless Architectures: Leverage AWS Lambda to run code in response to events without provisioning or managing servers.

Manage Concurrency with Decoupling: Use Amazon SQS to decouple application components, allowing for asynchronous communication and better concurrency control.

Monitor and Optimize Costs: Employ AWS Cost Explorer to track and analyze your spending across services.

ALERTING BEST PRACTICES

Actionable Alerts: Only alert on symptoms requiring human action.

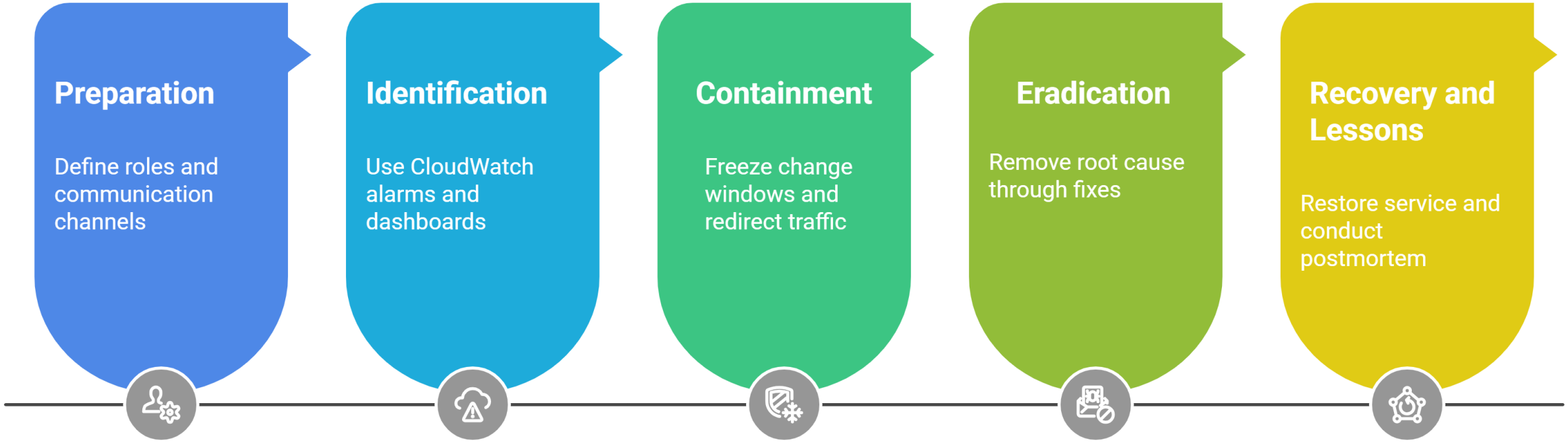
Severity Levels: Define critical, high, medium, low to prioritize responses.

Escalation Paths: Map alert severities to on-call rotations.

Alert Suppression: Use throttling and maintenance windows to avoid noise.

Runbook Links: Include playbook/ runbook links in alert notifications.

INCIDENT RESPONSE FRAMEWORK



BLAMELESS POSTMORTEM CULTURE

Psychological Safety: Encourage sharing mistakes without fear.

Timeline Reconstruction: Document events with timestamps.

Root Cause Analysis (RCA): Focus on systemic issues, not individuals.

Action Items: Specific tasks (automation, process changes) assigned to owners.

Follow-Up: Verify completion and efficacy of actions.

POP QUIZ:

Which step should occur first when a critical CloudWatch alarm fires for a production EC2 fleet?
A. Conduct a blameless postmortem
B. Immediately scale up the Auto Scaling group
C. Engage the on-call engineer and follow the incident playbook
D. Update the service’s SLO based on the outage

POP QUIZ:

Which step should occur first when a critical CloudWatch alarm fires for a production EC2 fleet?

A. Conduct a blameless postmortem

B. Immediately scale up the Auto Scaling group

C. Engage the on-call engineer and follow the incident playbook

D. Update the service's SLO based on the outage

CONFIGURATION MANAGEMENT OVERVIEW

Infrastructure as Code (IaC): Tools like AWS CloudFormation and Terraform allow you to define and provision infrastructure using code, promoting consistency and repeatability.

Parameter Stores: AWS Systems Manager Parameter Store and AWS Secrets Manager provide secure storage for configuration data and secrets.

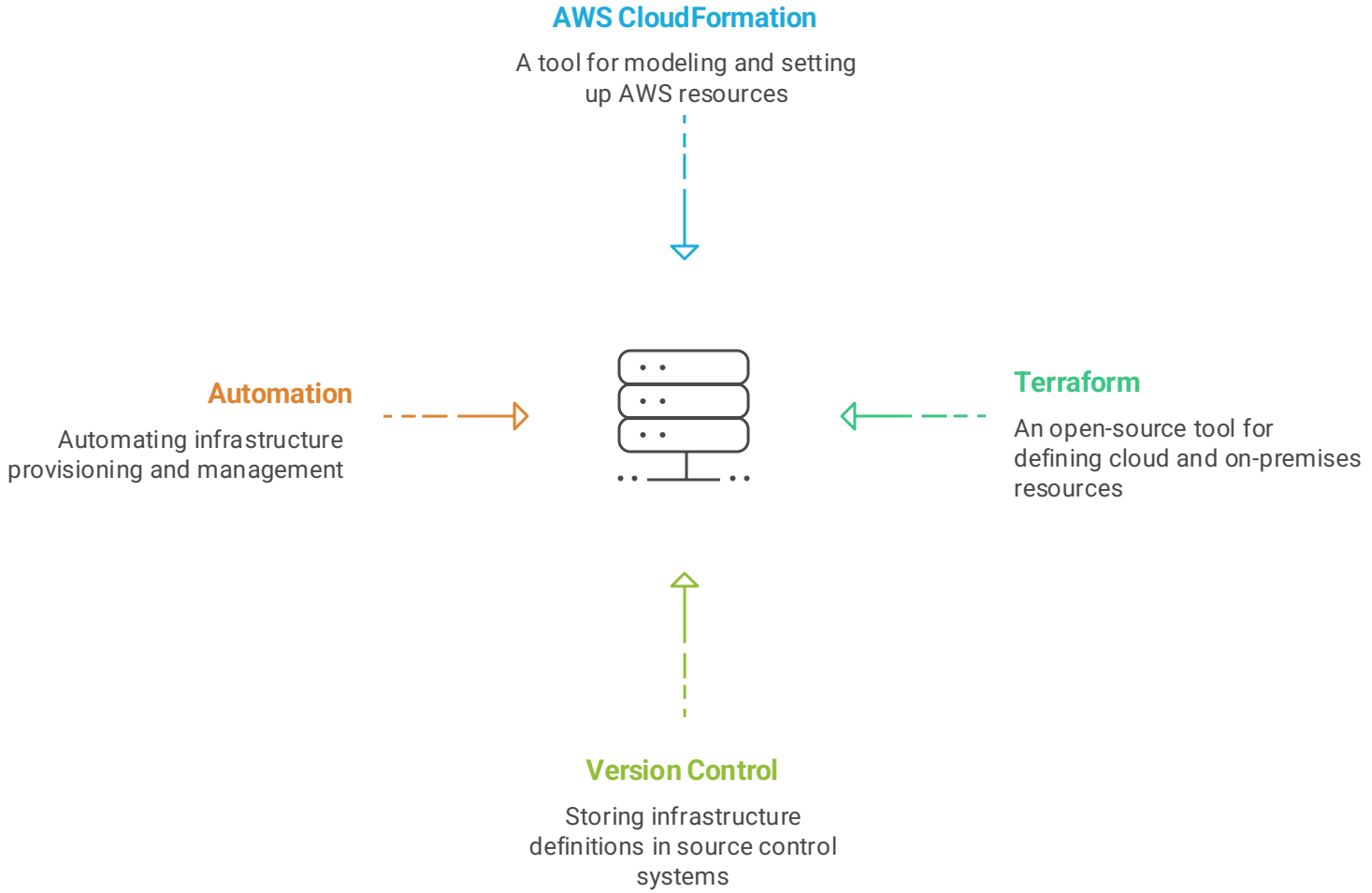
Dynamic Configuration: AWS AppConfig enables dynamic management of application configurations, including feature flags.

Environment Segregation: Maintain separate configurations for development, staging, and production environments to ensure isolation and control.

Policy as Code: Implement governance and compliance policies using AWS Config and other policy-as-code tools.

INFRASTRUCTURE AS CODE

IaC promotes best practices in software development, such as code reviews and automated testing, applied to infrastructure management.



SECRETS & CONFIGURATION MANAGEMENT

AWS Systems Manager Parameter Store:

- Stores configuration data and secrets as parameter values.Supports hierarchical storage and versioning.
- Offers basic encryption using AWS Key Management Service (KMS).

AWS Secrets Manager:

- Designed specifically for managing secrets like database credentials and API keys.
- Provides automatic rotation of secrets and detailed audit logs.
- Supports fine-grained access control and cross-account access.

DYNAMIC CONFIGURATION WITH AWS APPCONFIG

Features:

- Enables controlled deployment of application configurations, including feature flags.
- Supports validation of configurations before deployment to prevent errors.
- Integrates with monitoring tools to observe the impact of configuration changes.

Use Cases:

- Gradual rollout of new features.
- Dynamic adjustment of application settings without redeploying code.

POLICY AS CODE WITH AWS CONFIG

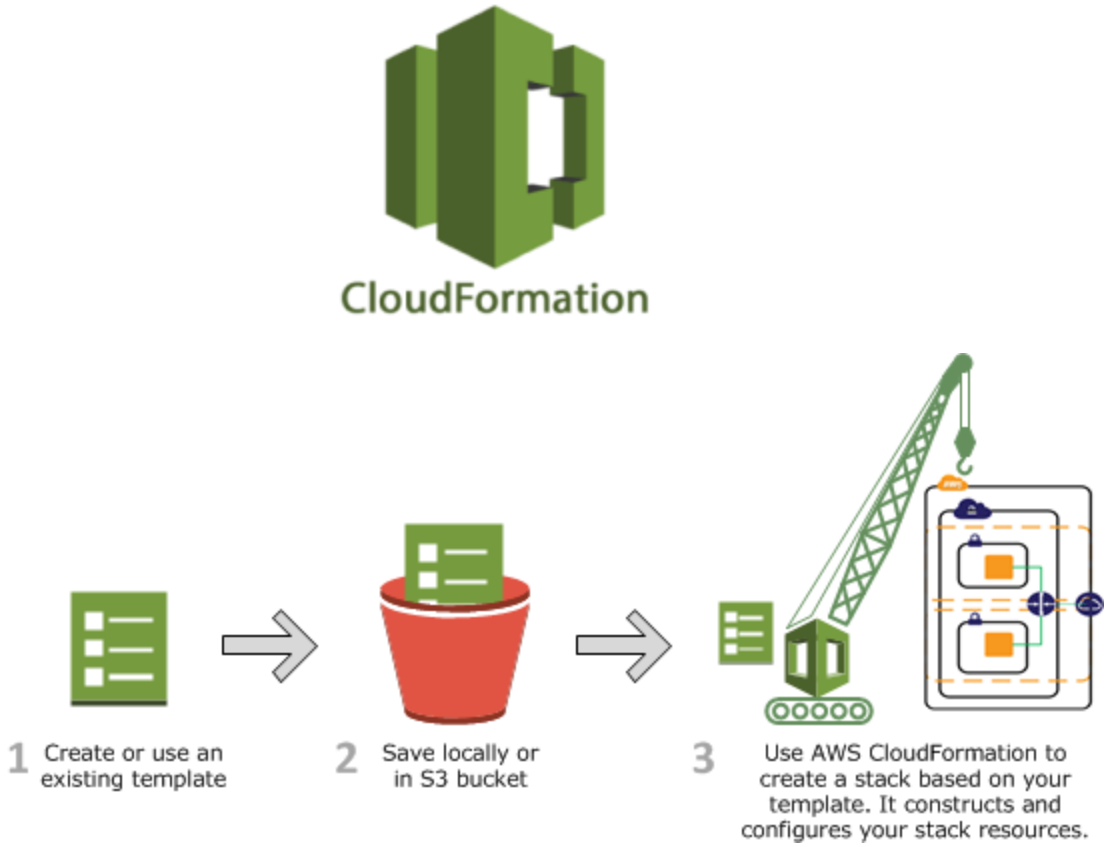
AWS Config:
Provides a detailed view of the configuration of AWS resources in your account.
Allows you to assess, audit, and evaluate the configurations of your resources.

Managed Rules:
Predefined rules that check for common best practices, such as ensuring EBS volumes are encrypted.

Custom Rules:
Define your own rules using AWS Lambda functions to evaluate configurations specific to your organization's requirements.

Conformance Packs:
Collections of AWS Config rules and remediation actions that can be easily deployed as a single entity.

IAC WITH AWS CLOUDFORMATION



https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/cloudformation-overview.html

Templates: JSON/YAML definitions of AWS resources.

Stacks and StackSets: Group related resources; deploy across accounts/regions.

Change Sets: Preview changes before applying.

Drift Detection: Identify manual changes outside IaC.

Integration: Use with CodePipeline for CI/CD.

CONTINUOUS INTEGRATION OVERVIEW

Source Control: Git workflows (GitHub, CodeCommit).

Build Services: AWS CodeBuild or Jenkins on EC2.

Automated Tests: Unit, integration, security (SAST).

Artifacts: Store in S3 or ECR for Docker images.

Triggers: Webhooks or CodePipeline stages.

RELEASE ENGINEERING FUNDAMENTALS

Role of Release Engineer: Coordinates builds, tests, and deployments.

Release Policies: Define gating criteria and approvals.

Tooling: AWS CodePipeline, CodeBuild, CodeDeploy.

Deployment Strategies: Blue/green, canary, rolling updates.

Rollback Plans: Automated rollback triggers on failure.

AUTOMATING BUILDS WITH CODEBUILD

```yaml
environment:
  type: LINUX_CONTAINER
  image: my-ecr-repo/custom-build-image:latest
  computeType: BUILD_GENERAL1_SMALL
  privilegedMode: true
```

Build Environments: Preconfigured images or custom Docker images.

Buildspec File: YAML-defined build commands and phases.

Caching: Speed up builds with S3 or local cache.

Reporting: Integrate with CloudWatch Logs for build logs.

Integration: Trigger via CodePipeline or webhooks.

EXAMPLE: BUILDSPEC FILE

```yaml
version: 0.2

phases:
  install:
    commands:
      - echo Installing dependencies...
      - pip install -r requirements.txt
  pre_build:
    commands:
      - echo Logging in to Amazon ECR...
      - aws ecr get-login-password --region $AWS_REGION | docker login --username AWS --password
  build:
    commands:
      - echo Building the Docker image...
      - docker build -t my-app .
  post_build:
    commands:
      - echo Pushing the Docker image...
      - docker push $ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaws.com/my-app
artifacts:
  files:
    - '**/*'
```

This configuration installs Python dependencies, logs in to Amazon ECR, builds a Docker image, and pushes it to the repository.

CACHING: ACCELERATE BUILD TIMES

```
cache:
  paths:
    - '/root/.m2/**/*'
    - 'node_modules/**/*'
```

To optimize build durations, AWS CodeBuild supports caching mechanisms:

Amazon S3 Caching: Stores cache artifacts in an S3 bucket, beneficial for sharing cache across multiple builds or projects.

Local Caching: Caches dependencies on the build host, offering faster access for subsequent builds on the same host.

INTEGRATE CODEBUILD WITH CLOUDWATCH LOGS

AWS CodeBuild integrates seamlessly with Amazon CloudWatch Logs, allowing you to monitor and analyze build logs in real-time.

Enabling CloudWatch Logs:

1. Navigate to the AWS CodeBuild console.
2. Select your build project and click on **Edit**.
3. In the **Logs** section, enable **CloudWatch Logs**.
4. Choose an existing log group or create a new one.

Once enabled, you can view logs in the CloudWatch console, set up alarms for specific log patterns, and retain logs for compliance or auditing purposes.

TRIGGER BUILDS VIA CODEPIPELINE OR WEBHOOKS

AWS CodeBuild can be integrated into various workflows:

AWS CodePipeline: Incorporate CodeBuild as a build stage in your CI/CD pipeline, automating the build and deployment process.
Webhooks: Set up webhooks to trigger builds upon code commits or pull requests in repositories like GitHub or Bitbucket.

Setting Up a Webhook:
1. In the CodeBuild console, create or edit a build project.
2. Under **Source**, select your repository provider.
3. Enable the **Webhook** option.
4. Configure filters to specify which events (e.g., push to a branch) trigger builds.

PACKAGING ARTIFACTS

Artifact Formats: ZIP, Docker images.

Repositories: S3 for zip artifacts; ECR for container images.

Versioning: Use semantic version tags or unique build IDs.

Security Scanning: Integrate image scanning with Amazon ECR.

Access Control: IAM policies for artifact repositories.

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

LAB 3: MONITORING

Goal: Build and observe a production-like distributed application using AWS-managed services. Explore how each service contributes to end-to-end observability through metrics, logs, and traces.

What You'll Learn

- Instrumenting distributed systems with AWS native tools
- Using X-Ray for tracing inter-service latency
- Creating CloudWatch dashboards and alarms
- Implementing real-time alerts with SNS
- Detecting and responding to system degradation

Instructions: Lab3.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