© 2025 by Innovation In Software Corporation

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

© 2024 TEKsystems Global Services, LLC ALL RIGHTS RESERVED.

REVIEW: DAY 2

Monitoring and Reliability Practices:

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

Day 1:

Introduction to SRE and Core Principles.

Day 2:

Monitoring, Automation, and Reliability Practices.

Day 3:

Advanced Topics

© 2025 by Innovation In Software Corporation

AGENDA FOR DAY 3

- Testing for reliability
- SRE Well-Architected Framework
- Load Balancing and Overload Management
- Cascading Failures
- Distributed Consensus

© 2025 by Innovation In Software Corporation

TESTING FOR RELIABILITY

© 2025 by Innovation In Software Corporation

INTRODUCTION

Purpose: Ensure systems withstand failures and scale effectively.
Alignment: Supports SRE practices focusing on error budgets and SLOs.
AWS Tools: Utilize CodeBuild, CodePipeline, and Fault Injection Simulator (FIS) for comprehensive testing.

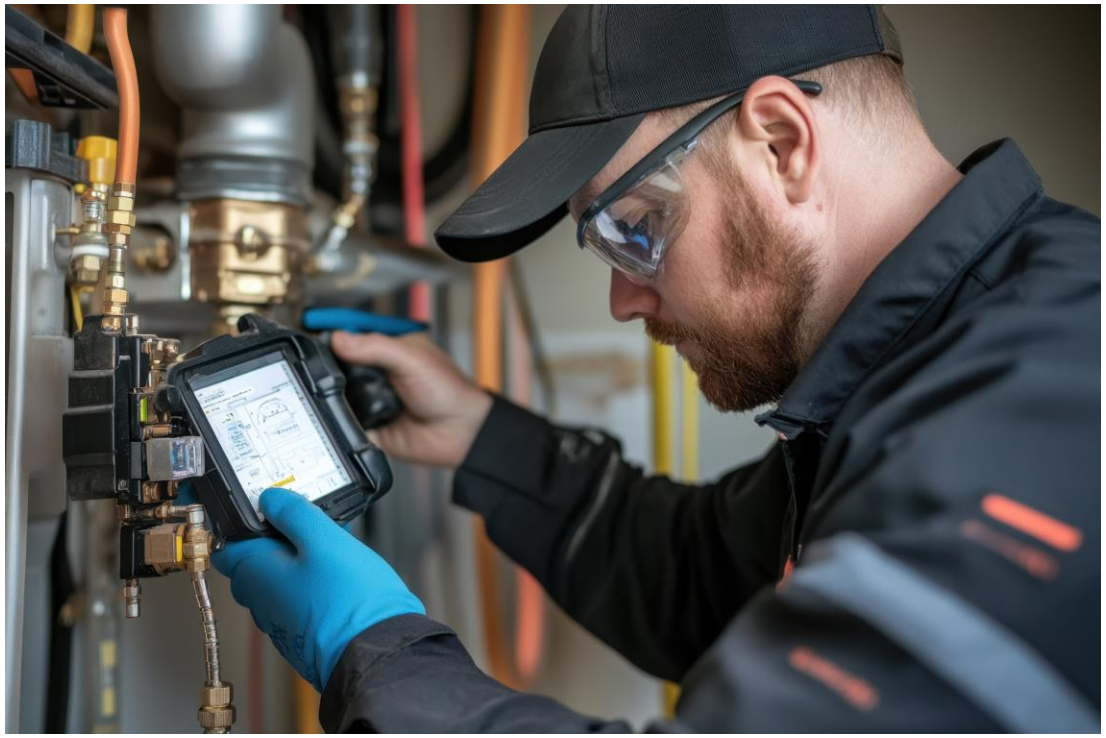
Testing Types:

- **Unit Testing**: Validate individual components.
- **Integration Testing**: Ensure components work together.
- **System Testing**: Assess the complete system's functionality.
- **Production Testing**: Monitor system behavior in live environments.
- **Chaos Testing**: Introduce controlled failures to test resilience.

Importance: Critical for maintaining reliability in distributed systems.

© 2025 by Innovation In Software Corporation

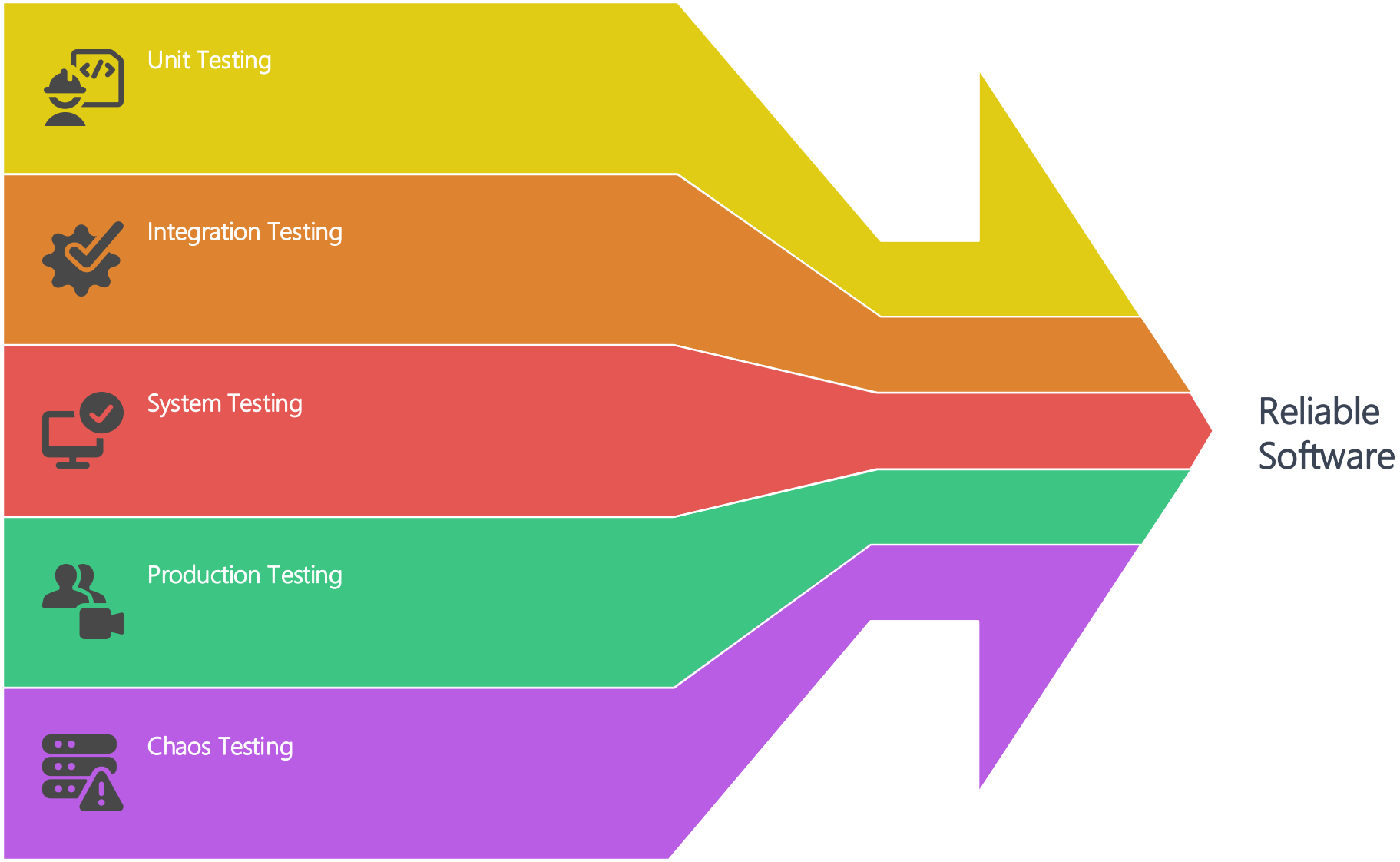
IMPORTANCE OF RELIABILITY TESTING



- **Ensures System Resilience Under Stress**
  Reliability testing validates that systems can handle unexpected conditions, such as traffic spikes or component failures, ensuring consistent performance.
- **Reduces Downtime and Identifies Weaknesses Early**
  By simulating failure scenarios, reliability testing uncovers potential issues before they impact users, minimizing downtime and maintenance costs.
- **Validates Failover Mechanisms**
  Testing failover strategies, like AWS Route 53's DNS failover, ensures that services remain available during regional outages or infrastructure failures.
- **Supports Continuous Improvement**
  Regular reliability testing fosters an iterative approach to system enhancement, aligning with DevOps practices and continuous integration/continuous deployment (CI/CD) pipelines.
- **Aligns with Business Objectives via Service Level Objectives (SLOs)**
  Setting and testing against SLOs ensures that system reliability meets business and customer expectations, facilitating trust and satisfaction.

© 2025 by Innovation In Software Corporation

TYPES OF SOFTWARE TESTING



© 2025 by Innovation In Software Corporation

UNIT TESTING WITH AWS

- **Tests Isolated Code Units for Correctness**
  Focuses on verifying individual components, such as functions or methods, in isolation to ensure they perform as intended.

- **Automate Unit Test Execution with AWS CodeBuild**
  Utilize AWS CodeBuild to automatically run unit tests upon code commits, integrating seamlessly into your CI/CD pipeline.

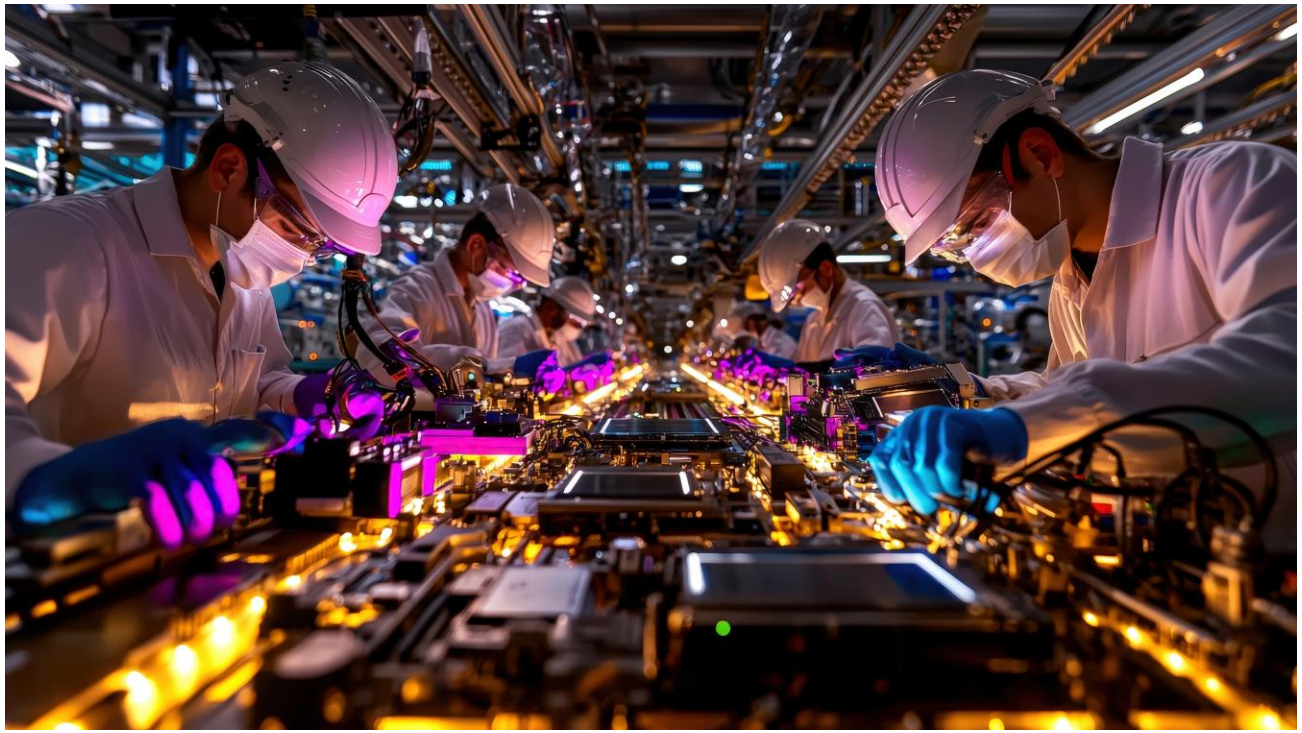
- **Example: Testing Lambda Function Logic with pytest**
  Employ pytest to test the logic within AWS Lambda functions, ensuring they handle various inputs and edge cases correctly.

- **Provides Fast Developer Feedback**
  Quick test execution allows developers to receive immediate feedback, facilitating rapid identification and resolution of issues.

- **Ensures Code Quality Before Integration**
  By catching bugs early, unit testing maintains high code quality, reducing the likelihood of defects propagating to later stages.

© 2025 by Innovation In Software Corporation

INTEGRATION TESTING



Integration testing is a critical phase in our CI/CD pipeline, focusing on the interactions between different system components.

For instance, verifying that an API Gateway endpoint correctly triggers the intended Lambda function and that the response is as expected.

© 2025 by Innovation In Software Corporation

SYSTEM TESTING – VALIDATING END TO END FUNCTIONALITY

- **Validates Interactions Between Components**
  Ensures that integrated components, such as API Gateway and Lambda functions, work together as intended.

- **Automated Testing with AWS CodeBuild**
  Utilize AWS CodeBuild within your CI/CD pipeline to automate integration tests, ensuring consistent and repeatable testing processes.

- **Example: API Gateway and Lambda Integration**
  Test the interaction between API Gateway endpoints and their corresponding Lambda functions to verify correct request handling and response generation.

- **Identifies Communication Issues Early**
  Detects problems like misconfigured endpoints, incorrect request/response formats, or permission issues between services before they reach production.

- **Ensures Seamless Interoperability**
  Confirms that all system components interact smoothly, which is crucial for the reliability of distributed systems.

© 2025 by Innovation In Software Corporation

PRODUCTION TESTING

Monitoring with CloudWatch

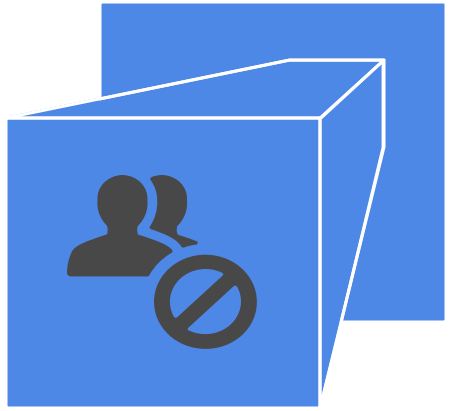
CloudWatch monitoring offers high feedback with minimal risk.

Canary Releases

Canary releases provide high feedback with controlled risk.

Small User Subsets

Small user subsets minimize risk but offer limited feedback.



Full Rollout

Full rollout maximizes risk with minimal feedback.

© 2025 by Innovation In Software Corporation

CHAOS TESTING

Simulates Failures to Test Resilience
Intentionally introduces faults into the system to observe how it responds, ensuring it can withstand and recover from unexpected disruptions.

Utilize AWS Fault Injection Simulator (FIS)
Leverage AWS FIS to inject various types of failures, such as terminating EC2 instances or introducing network latency, in a controlled manner.

Example: Simulate EC2 Instance Termination to Test Failover
Terminate an EC2 instance using AWS FIS to verify that auto-scaling groups or load balancers correctly redirect traffic to healthy instances.

Identifies Design Weaknesses
Reveals hidden flaws in the system architecture, such as single points of failure or inadequate failover mechanisms.

Builds Confidence in System Reliability
By validating the system's ability to handle failures gracefully, teams gain assurance in its robustness and resilience.

© 2025 by Innovation In Software Corporation

TESTING AT SCALE

Simulates High Traffic to Test System Limits

Conduct load tests that mimic peak user activity to evaluate how the application performs under stress.

Utilize JMeter or Locust on EC2 for Load Testing

Deploy tools like Apache JMeter or Locust on Amazon EC2 instances to generate significant load and simulate concurrent users.

Example: Test Application with 50,000 Concurrent Users

Simulate 50,000 users interacting with the application simultaneously to assess performance and stability.

Monitor with Amazon CloudWatch for Bottlenecks

Use Amazon CloudWatch to track key metrics such as CPU utilization, memory usage, and response times to identify performance bottlenecks.

Ensures Scalability Under Peak Loads

Validate that the application can scale appropriately to handle high traffic volumes without degradation in performance.

© 2025 by Innovation In Software Corporation

BEST PRACTICES FOR TESTING IN AWS

- **Automate Testing with AWS CodePipeline and CodeBuild**
  Integrate automated testing into your CI/CD pipelines using AWS CodePipeline and CodeBuild. This ensures consistent and efficient execution of tests throughout the development lifecycle.

- **Implement Chaos Engineering with AWS Fault Injection Simulator (FIS)**
  Use AWS FIS to introduce controlled disruptions, such as terminating instances or simulating latency, to test system resilience and identify potential weaknesses.

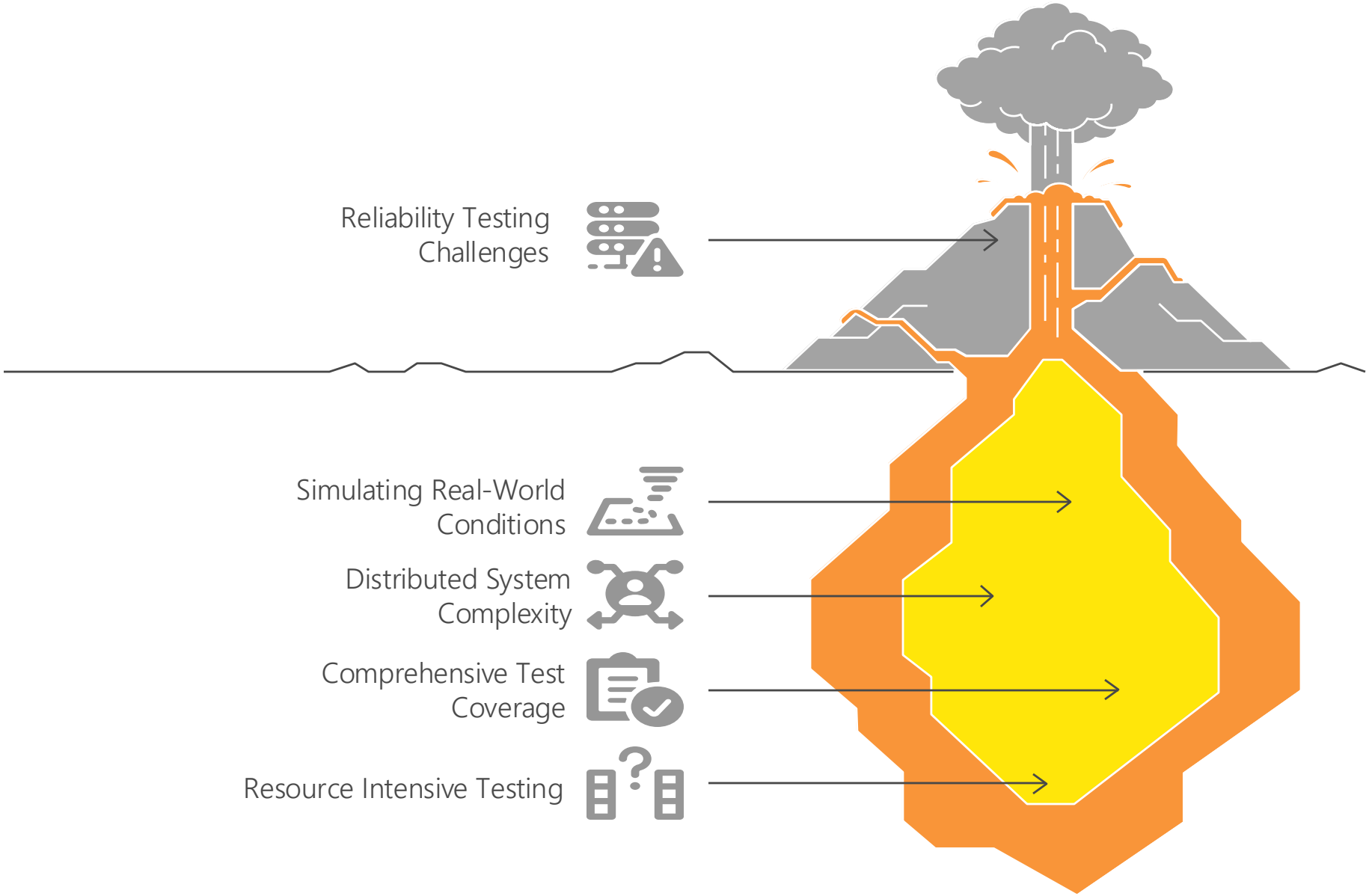
- **Example: Test Database Failover with AWS FIS**
  Simulate a database failure using AWS FIS to verify that your application's failover mechanisms function correctly, ensuring high availability.

- **Integrate Testing into CI/CD Pipelines**
  Embed various testing stages—unit, integration, system, and chaos tests—into your CI/CD workflows to catch issues early and maintain code quality.

- **Monitor Applications with Amazon CloudWatch and AWS X-Ray**
  Leverage Amazon CloudWatch for real-time monitoring of application metrics and AWS X-Ray for tracing requests to identify performance bottlenecks and errors.

© 2025 by Innovation In Software Corporation

CHALLENGES IN RELIABILITY TESTING



© 2025 by Innovation In Software Corporation

POP QUIZ:

During a system test of your web application, you aim to validate the entire user journey from the frontend to the backend database. Which AWS service combination is most appropriate for automating this end-to-end testing process?
A. AWS CodeBuild and AWS Lambda
B. AWS CodePipeline and AWS CodeBuild
C. AWS CodeDeploy and AWS FIS
D. AWS CloudFormation and AWS X-Ray

© 2025 by Innovation In Software Corporation

POP QUIZ:

During a system test of your web application, you aim to validate the entire user journey from the frontend to the backend database. Which AWS service combination is most appropriate for automating this end-to-end testing process?

A. AWS CodeBuild and AWS Lambda

B. AWS CodePipeline and AWS CodeBuild

C. AWS CodeDeploy and AWS FIS

D. AWS CloudFormation and AWS X-Ray

© 2025 by Innovation In Software Corporation

POP QUIZ:

In preparing for a high-traffic event, you need to simulate 50,000 concurrent users interacting with your application to test its scalability. Which approach is most suitable for conducting this large-scale load test in AWS?
A. Deploying JMeter on Amazon EC2 instances
B. Utilizing AWS Lambda functions for simulation
C. Using AWS Config to monitor resource changes
D. Implementing AWS CloudTrail for user activity logs

© 2025 by Innovation In Software Corporation

POP QUIZ:

In preparing for a high-traffic event, you need to simulate 50,000 concurrent users interacting with your application to test its scalability. Which approach is most suitable for conducting this large-scale load test in AWS?

A. Deploying JMeter on Amazon EC2 instances
B. Utilizing AWS Lambda functions for simulation
C. Using AWS Config to monitor resource changes
D. Implementing AWS CloudTrail for user activity logs

© 2025 by Innovation In Software Corporation

LAB 7: CHAOS TESTING

Goal: Learn how to simulate EC2 failures using **AWS Fault Injection Simulator** (FIS) and observe how **Auto Scaling** helps restore service automatically.

Key Learning Outcomes

- Understand how FIS can validate system resilience
- Practice setting up EC2 instances in an Auto Scaling Group
- Simulate controlled failures and observe recovery behavior
- Collect and analyze metrics with **CloudWatch**

Instructions: Lab7.md

© 2025 by Innovation In Software Corporation

SRE WELL ARCHITECTED

© 2025 by Innovation In Software Corporation

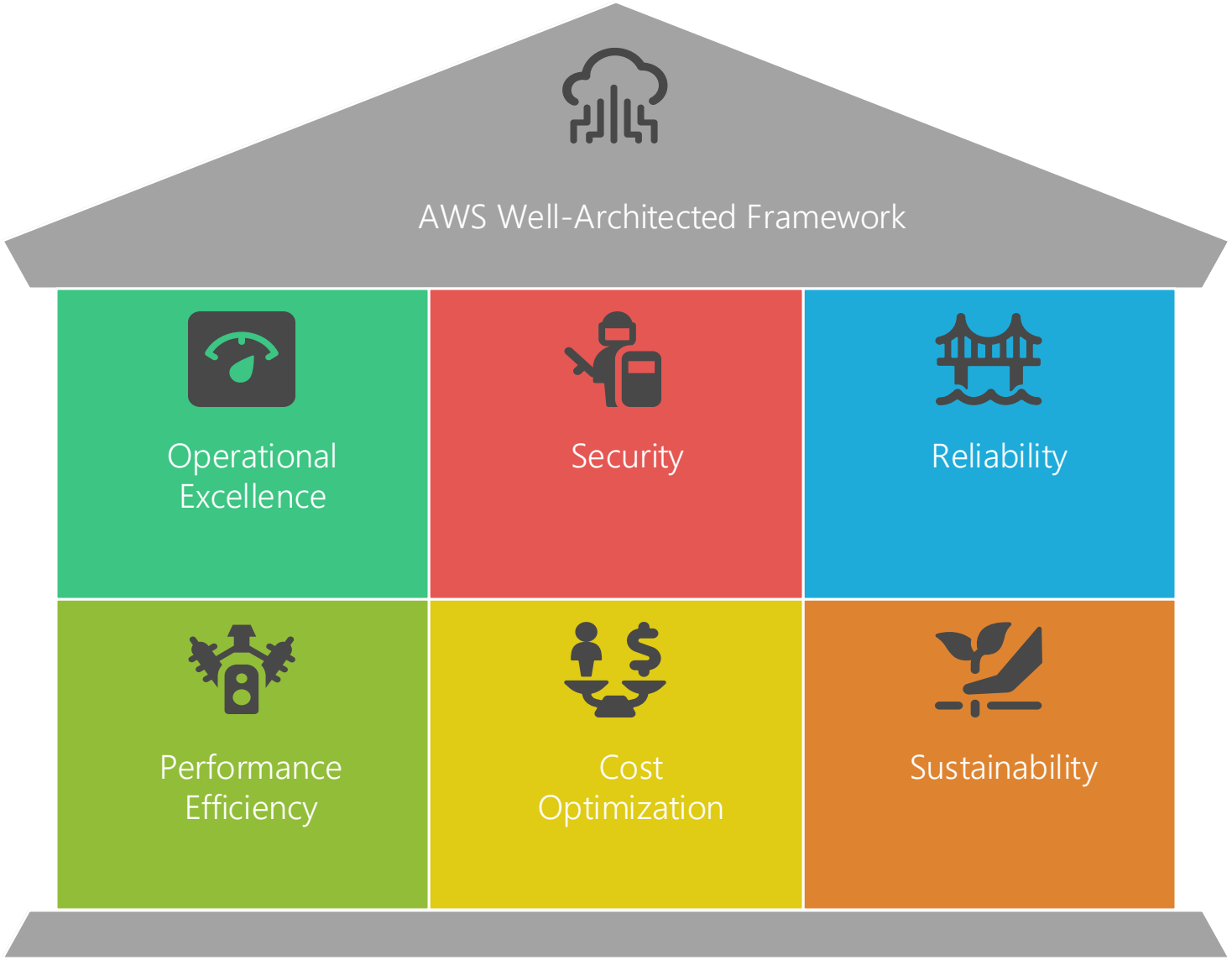
WHY SRE – WELL ARCHITECTED

SRE focuses on applying software engineering practices to IT operations, emphasizing automation, reliability, and scalability.

When combined with the AWS Well-Architected Framework, which provides a set of best practices for cloud architecture, organizations can align technical implementations with business objectives effectively

© 2025 by Innovation In Software Corporation

SIX PILLARS OF AWS WELL ARCHITECTED



© 2025 by Innovation In Software Corporation

RESILIENCE

- Ensures quick recovery from failures.
- Use Auto Scaling, Elastic Load Balancing, Route 53 failover.
- Example: Auto-scale EC2 during spikes.
- Implement circuit breakers for cascading failures.
- Test with FIS chaos experiments.

© 2025 by Innovation In Software Corporation

SECURITY

- Protects from unauthorized access/threats.
- Use AWS IAM for fine-grained control.
- Encrypt with KMS, SSL/TLS.
- Example: Secure S3 with encryption/IAM.
- Monitor with Security Hub/GuardDuty.

© 2025 by Innovation In Software Corporation

OPERATIONAL EXCELLENCE

Operational Excellence in AWS emphasizes the efficient and reliable operation of workloads, supporting continuous improvement and delivering business value.

Key practices include automation, proactive monitoring, and fostering a culture of learning.

© 2025 by Innovation In Software Corporation

PERFORMANCE EFFICIENCY

- Leverage AWS Compute Optimizer for EC2 Recommendations:
  - Utilize AWS Compute Optimizer to analyze historical utilization metrics and receive recommendations for optimal EC2 instance types, including options powered by AWS Graviton processors for enhanced performance and cost savings.

- Select Appropriate EC2 Instance Types:
  - For lightweight workloads, consider burstable instance types like t3.micro.
  - For compute-intensive applications, explore compute-optimized instances such as c7g or m7i-flex.

- Optimize Amazon Aurora for High-Performance Databases:
  - Implement Aurora's advanced features like Parallel Query and asynchronous key prefetch to enhance query performance.
  - Choose appropriate DB instance classes (e.g., db.r5.large) based on workload requirements.

- Monitor and Identify Bottlenecks with Amazon CloudWatch:
  - Set up CloudWatch alarms to monitor key performance metrics such as CPU utilization, disk I/O, and network throughput.
  - Use these insights to proactively address potential performance issues.

- Implement Read Replicas and Connection Pooling:
  - Use Aurora Read Replicas to offload read traffic and improve scalability.
  - Configure connection pooling to manage database connections efficiently, reducing latency and resource consumption.

© 2025 by Innovation Solutions Corporation

COST OPTIMIZATION

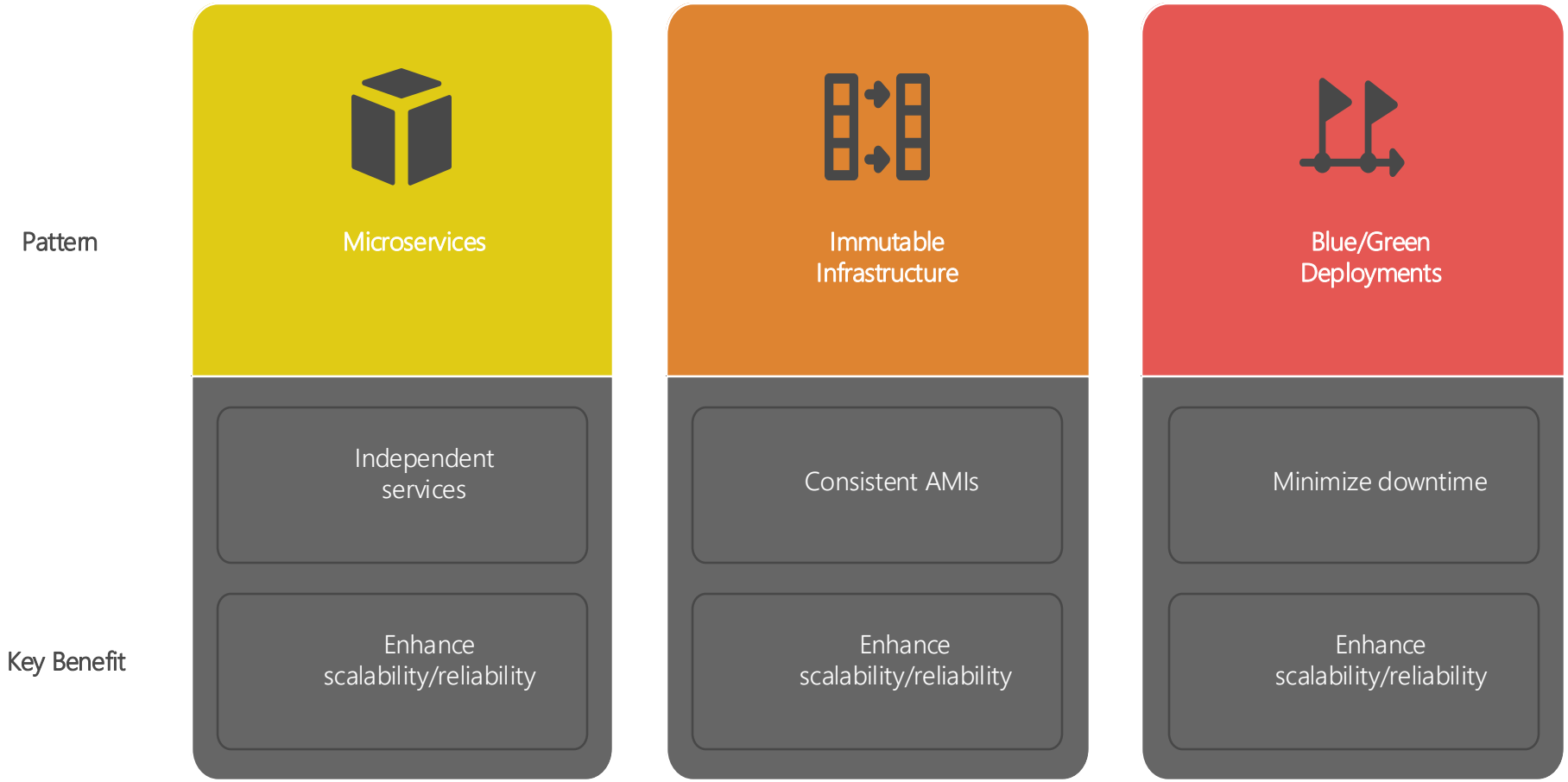
Cost optimization gets value for money. Cost Explorer tracks spending, budgets prevent overspending.

Reserved Instances/Savings Plans save on commitments.

Clean up unused resources with Resource Groups, ensuring cost-effective AWS operations.

© 2025 by Innovation In Software Corporation

PATTERNS FOR SRE WELL ARCHITECTED



© 2025 by Innovation In Software Corporation

ANT-PATTERNS TO AVOID

Monolithic Architectures:

- Building applications as a single, tightly coupled unit makes scaling and maintenance challenging.
- Monoliths hinder agility and can become bottlenecks as the system grows.
- *Best Practice:* Adopt microservices architecture to enable independent development, deployment, and scaling of services.

Over-Provisioning Resources:

- Allocating more resources than necessary "just in case" leads to increased costs without proportional benefits.
- This approach is inefficient and can mask underlying performance issues.
- *Best Practice:* Utilize AWS Compute Optimizer and AWS Cost Explorer to right-size resources based on actual usage patterns.

Manual Processes:

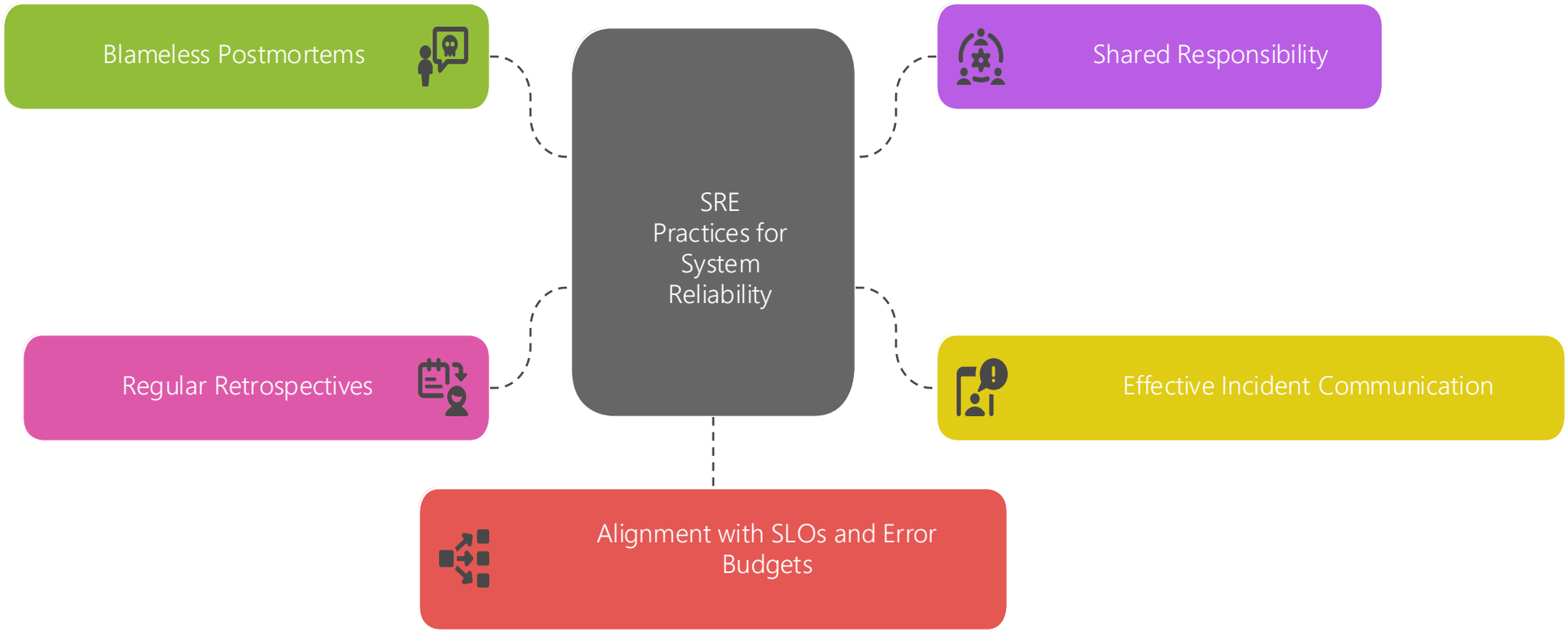
- Relying on manual interventions for deployments, scaling, or configuration increases the risk of human error and inconsistency.
- Manual scaling of EC2 instances, for example, can lead to delays and misconfigurations.
- *Best Practice:* Implement automation using AWS tools like Auto Scaling, AWS Systems Manager, and AWS CloudFormation to ensure consistency and reduce operational overhead.

Manual Environment Management:

- Manually provisioning and managing environments can introduce inconsistencies and slow down development cycles.
- Such practices are prone to errors and do not scale well.
- *Best Practice:* Adopt Infrastructure as Code (IaC) using AWS CloudFormation or AWS CDK to automate environment provisioning and ensure consistency across deployments

© 2025 by Innovation In Software Corporation

SRE CULTURE



© 2025 by Innovation In Software Corporation

POP QUIZ:

In the context of the SRE Well-Architected Framework, what is a primary benefit of implementing blameless postmortems?
A. Identifying individuals responsible for incidents
B. Encouraging rapid deployment without testing
C. Fostering a culture of learning and continuous improvement
D. Reducing the need for incident documentation

© 2025 by Innovation In Software Corporation

POP QUIZ:

In the context of the SRE Well-Architected Framework, what is a primary benefit of implementing blameless postmortems?
A. Identifying individuals responsible for incidents
B. Encouraging rapid deployment without testing
C. Fostering a culture of learning and continuous improvement
D. Reducing the need for incident documentation

© 2025 by Innovation In Software Corporation

EVENT DRIVEN ARCHITECTURE

Manual Processes:

- Relying on manual interventions for deployments, scaling, or configuration increases the risk of human error and inconsistency.
- Manual scaling of EC2 instances, for example, can lead to delays and misconfigurations.
- *Best Practice:* Implement automation using AWS tools like Auto Scaling, AWS Systems Manager, and AWS CloudFormation to ensure consistency and reduce operational overhead.

© 2025 by Innovation In Software Corporation

ANTI-PATTERNS: SINGLE POINTS OF FAILURE

Single EC2 Instance for Critical Services:

- Hosting essential applications on a lone EC2 instance without redundancy.
- Risk: If the instance fails, the service becomes unavailable.

Single-AZ Deployments:

- Deploying resources in only one Availability Zone.
- Risk: AZ outages can lead to complete service disruption.

Lack of Load Balancing:

- Directing all traffic to a single server without a load balancer.
- Risk: Server overload and no failover mechanism.

Monolithic Architectures:

- Tightly coupled systems where failure in one component affects the entire application.
- Risk: Difficult to isolate and recover from failures.

© 2025 by Innovation In Software Corporation

PATTERNS: IMMUTABLE INFRASTRUCTURE

Enhance system reliability, consistency, and security by deploying infrastructure components that are never modified after deployment.

© 2025 by Innovation In Software Corporation

ANTI-PATTERNS: MANUAL PROCESSES

Manual EC2 Instance Updates:

- SSHing into instances to apply patches or updates individually.
- **Risks:** Inconsistent configurations, increased human error, and security vulnerabilities.

Manual Code Deployments:

- Uploading application code directly to servers without version control or automation.
- **Risks:** Deployment errors, lack of rollback capabilities, and prolonged downtime.

Ad-Hoc Monitoring and Logging:

- Relying on manual checks for system health and performance metrics.
- **Risks:** Delayed incident detection and response, leading to potential SLA breaches.

© 2025 by Innovation In Software Corporation

ERROR BUDGETS

Balance system reliability with innovation by defining acceptable thresholds for service disruptions, enabling informed decision-making and risk management.

© 2025 by Innovation In Software Corporation

POP QUIZ:

You're designing a cloud-native application that requires high availability and minimal downtime during deployments. Which architectural pattern best supports these requirements?
A. Monolithic deployment
B. Blue/Green deployment
C. Manual server provisioning
D. Vertical scaling

© 2025 by Innovation In Software Corporation

POP QUIZ:

You're designing a cloud-native application that requires high availability and minimal downtime during deployments. Which architectural pattern best supports these requirements?

A. Monolithic deployment
B. Blue/Green deployment
C. Manual server provisioning
D. Vertical scaling

© 2025 by Innovation In Software Corporation

LOAD BALANCING FUNDAMENTALS

© 2025 by Innovation In Software Corporation

INTRODUCTION

Load balancing spreads traffic across servers, preventing overload and ensuring availability.

AWS offers ALB for HTTP, NLB for TCP/UDP, and GLB for VPCs.

ALB routing HTTP to EC2 instances cuts latency and boosts uptime, a key SRE tool for user-facing systems.

© 2025 by Innovation In Software Corporation

WHY LOAD BALANCING MATTERS

Redundancy with Load Balancers:

- Deploy multiple EC2 instances behind Elastic Load Balancing (ELB) to distribute traffic evenly and prevent overload on a single server.

Multi-AZ Deployments:

- Configure load balancers across multiple Availability Zones (AZs) to ensure high availability and fault tolerance.

Handling Traffic Spikes:

- Utilize AWS Auto Scaling in conjunction with load balancers to automatically adjust the number of running instances based on traffic demand.

Improving Response Times:

- Implement Application Load Balancer (ALB) for HTTP/HTTPS traffic to route requests based on content, reducing latency and enhancing user experience.

Defining SLOs:

- Set clear Service Level Objectives (SLOs) to establish acceptable performance and availability targets for your applications.

Monitoring with CloudWatch:

- Leverage Amazon CloudWatch to monitor key metrics such as latency, error rates, and request counts, ensuring adherence to SLOs.

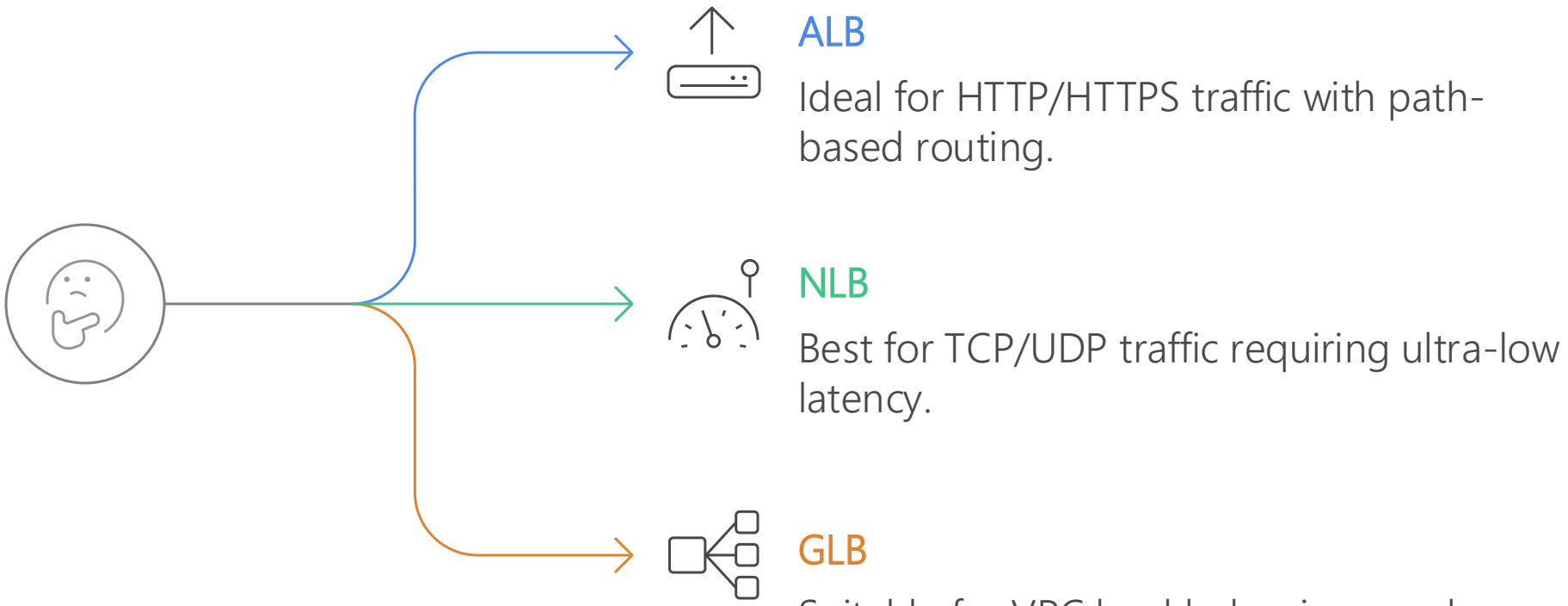
Managing Error Budgets:

- Track error budgets to balance the release of new features with system reliability, allowing for calculated risk-taking without compromising service quality.

© 2025 by Innovation In Software Corporation

TYPES OF LOAD BALANCES IN AWS

Which AWS load balancer should be used?



ALB

Ideal for HTTP/HTTPS traffic with path-based routing.

NLB

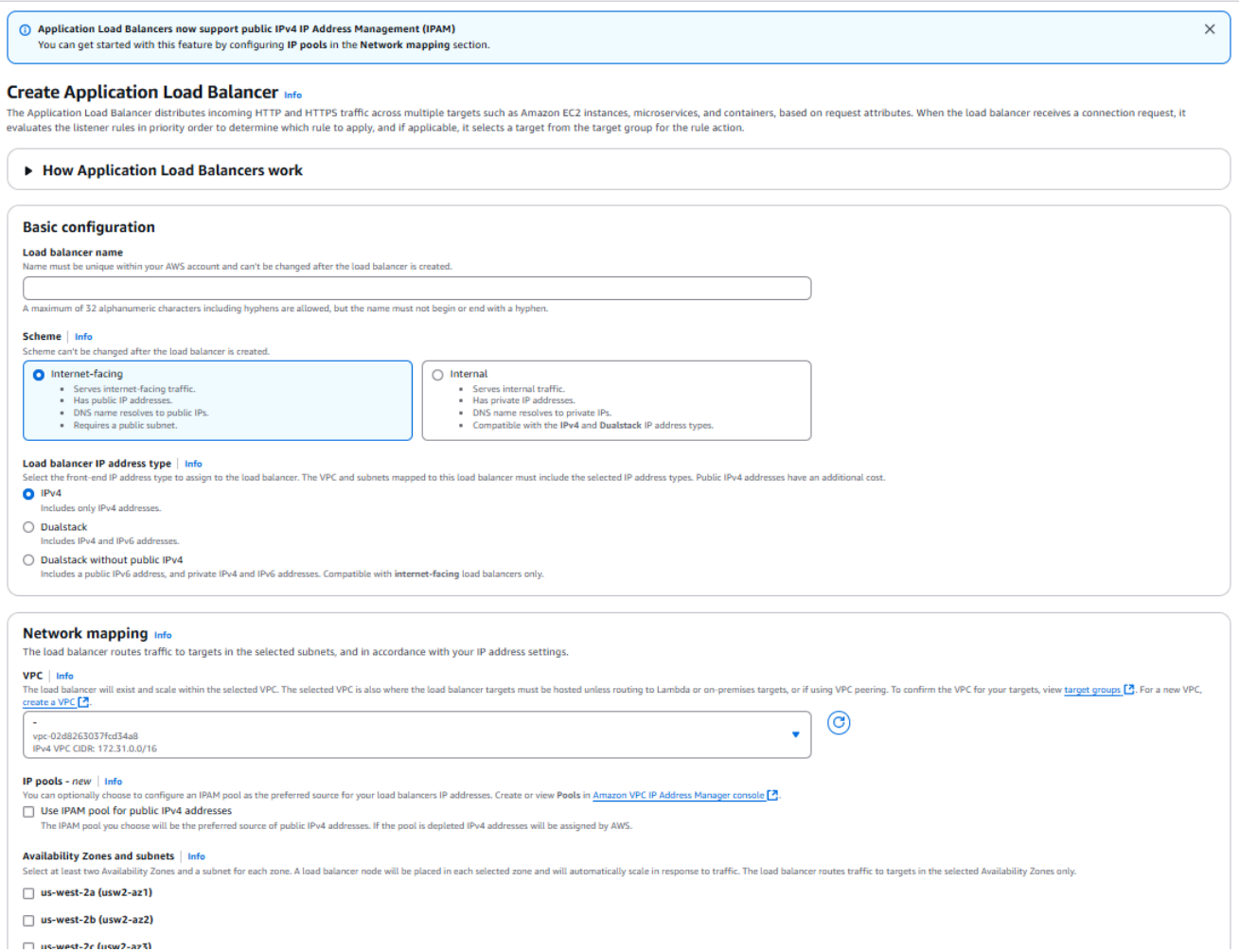
Best for TCP/UDP traffic requiring ultra-low latency.

GLB

Suitable for VPC load balancing needs.

© 2025 by Innovation In Software Corporation

SETTING UP ALB IN AWS



- Create ALB via AWS Console/CLI.
- Configure HTTP/HTTPS listeners.
- Define target groups for EC2/containers.

© 2025 by Innovation In Software Corporation

LOAD BALANCING WITH DNS

Route 53's DNS-based load balancing directs traffic across regions, like to the nearest data center for low latency.

It supports failover, ensuring availability if a region fails. This is perfect for global apps, enhancing user experience in AWS.

© 2025 by Innovation In Software Corporation

LOAD BALANCING AT VIRTUAL IP ADDRESS

Ensure consistent and reliable client connectivity by leveraging static IP addresses with AWS Network Load Balancer (NLB), enhancing system resilience and simplifying network configurations.

© 2025 by Innovation In Software Corporation

BEST PRACTICES FOR LOAD BALANCING

Deploy Across Multiple Availability Zones (AZs)

High Availability:

- Distribute load balancers and targets across at least two AZs to ensure fault tolerance and minimize downtime.

Improved Performance:

- Utilize cross-zone load balancing to evenly distribute traffic across all registered targets in all enabled AZs

Monitor Metrics with Amazon CloudWatch

Key Metrics to Track:

- Request count, latency, HTTP error codes (4XX and 5XX), and target response times.

Anomaly Detection:

- Implement CloudWatch anomaly detection to proactively identify unusual patterns and potential issues.

Secure with Proper Security Groups

Restrict Access:

- Configure security groups to allow only necessary traffic to and from the load balancer.

Regular Audits:

- Periodically review and update security group rules to adhere to the principle of least privilege.

Regularly Update Configurations

Health Checks:

- Ensure health check settings are optimized for timely detection of unhealthy targets.

Listener Rules:

- Review and update listener rules to align with evolvin application requirements.

© 2025 by Innovation In Software Corporation

ACTIVITY: SCENARIO ANALYSIS

Form Small Groups: Divide participants into groups of 3–5.
Assign Scenarios: Provide each group with a unique scenario involving load balancing challenges. Examples include:

- A sudden traffic spike during a marketing campaign.
- A regional outage affecting one Availability Zone.
- Implementing blue-green deployments with load balancers.

Group Discussion: Each group analyzes their scenario, focusing on:

- Identifying potential risks and failure points.
- Proposing load balancing strategies to mitigate issues.
- Aligning solutions with SRE principles like error budgets and SLIs/SLOs.

© 2025 by Innovation In Software Corporation

POP QUIZ:

You are an SRE transitioning from Azure to AWS, tasked with configuring Amazon API Gateway to handle frontend traffic for a retail application. The application experiences unpredictable traffic spikes during sales events, risking overload. Which configuration best prevents overload while maintaining user experience?

A. Set a fixed rate limit of 500 requests/second per API key with no throttling headers.

B. Enable client-side throttling with exponential backoff and configure usage plans for 1000 requests/hour per customer.

C. Use AWS WAF to block all requests exceeding 1000/second, ignoring client-specific limits.

D. Configure caching for static responses but disable rate limiting to maximize throughput

© 2025 by Innovation In Software Corporation

POP QUIZ:

You are an SRE transitioning from Azure to AWS, tasked with configuring Amazon API Gateway to handle frontend traffic for a retail application. The application experiences unpredictable traffic spikes during sales events, risking overload. Which configuration best prevents overload while maintaining user experience?

A. Set a fixed rate limit of 500 requests/second per API key with no throttling headers.

B. Enable client-side throttling with exponential backoff and configure usage plans for 1000 requests/hour per customer.

C. Use AWS WAF to block all requests exceeding 1000/second, ignoring client-specific limits.

D. Configure caching for static responses but disable rate limiting to maximize throughput

© 2025 by Innovation In Software Corporation

POP QUIZ:

Your AWS-based frontend uses an Application Load Balancer (ALB) to distribute traffic to EC2 instances running a web application. A sudden spike in traffic causes one instance to fail, triggering a cascading failure across the fleet. Which strategy best prevents this?

A. Implement bulkheads by isolating EC2 instances in separate Auto Scaling groups per availability zone.
B. Increase the ALB's idle timeout to 120 seconds to handle slow client connections.
C. Use AWS Shield Advanced to protect against DDoS attacks causing instance failures.
D. Configure SQS to queue all incoming requests before forwarding to EC2 instances.

© 2025 by Innovation In Software Corporation

POP QUIZ:

Your AWS-based frontend uses an Application Load Balancer (ALB) to distribute traffic to EC2 instances running a web application. A sudden spike in traffic causes one instance to fail, triggering a cascading failure across the fleet. Which strategy best prevents this?

A. Implement bulkheads by isolating EC2 instances in separate Auto Scaling groups per availability zone.

B. Increase the ALB's idle timeout to 120 seconds to handle slow client connections.

C. Use AWS Shield Advanced to protect against DDoS attacks causing instance failures.

D. Configure SQS to queue all incoming requests before forwarding to EC2 instances.

© 2025 by Innovation In Software Corporation

LAB 8: LOAD TESTING WITH APACHE JMETER

Goal: Learn to simulate large-scale traffic to a web application using **Apache JMeter** and analyze system performance using **Amazon CloudWatch**.

Key Learning Outcomes

- Launch and configure EC2-based test infrastructure
- Use JMeter to simulate **up to 50,000 concurrent users**
- Collect and analyze performance metrics with CloudWatch
- Set up CloudWatch Alarms for threshold breaches
- Evaluate system behavior under sustained load

Instructions: Lab8.md

© 2025 by Innovation In Software Corporation

HANDLING OVERLOAD

© 2025 by Innovation In Software Corporation

INTRODUCTION

Overload Definition: Occurs when system demand exceeds capacity, causing performance degradation or failure.

Common Triggers: Includes traffic spikes (e.g., marketing campaigns), resource exhaustion, or misconfigurations.

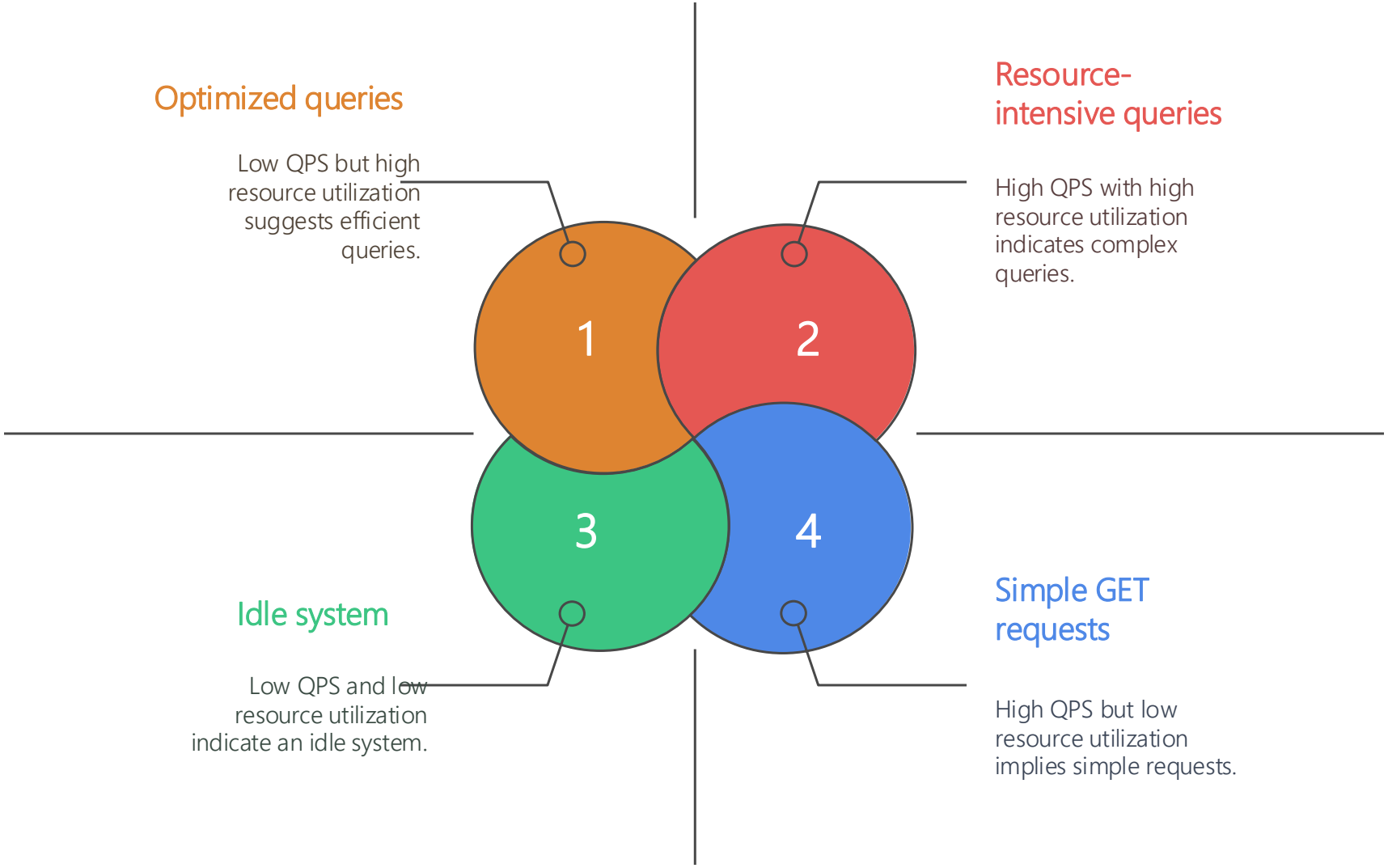
Impact on Reliability: Threatens SLOs by increasing latency or causing outages if not managed.

AWS Tools: API Gateway for throttling, WAF for attack mitigation, Auto Scaling for dynamic resource adjustment.

Goal: Maintain stability and user satisfaction during high-demand periods.

© 2025 by Innovation In Software Corporation

THE PITFALLS OF "QUERIES PER SECOND" (QPS)



© 2025 by Innovation In Software Corporation

IMPLEMENTING PER CUSTOMER LIMITS

Purpose: Prevents abuse by limiting requests per customer, ensuring fair resource allocation.

Example: Restrict API calls to 1000/hour/user via API Gateway usage plans.

Enforcement: Use API keys and quotas in API Gateway to control access.

Monitoring: Set CloudWatch alarms for quota exceedances to detect potential issues.

Benefits: Maintains performance, reduces denial-of-service risks, ensures equitable access.

© 2025 by Innovation In Software Corporation

CLIENT-SIDE THROTTLING TECHNIQUES

- **Client Responsibility**: Clients manage request rates to reduce server load; example: limit API calls per second.
- **Exponential Backoff**: Increase wait times between retries after failures; example: wait 1s, 2s, 4s after 429 responses.
- **AWS SDK Support**: Built-in retry logic with backoff for common operations; simplifies implementation.
- **Monitoring Compliance**: Track client request rates with CloudWatch to ensure adherence to throttling policies.
- **Benefits**: Reduces server strain, improves user experience by avoiding repeated failures.

© 2025 by Innovation In Software Corporation

CRITICALITY AND SERVICE LEVEL OBJECTIVES

- **Service Criticality**: Define SLOs based on service importance; critical services (e.g., payments) need stricter SLOs.
- **Error Budgets**: Allocate acceptable downtime; example: 0.1% downtime for innovation in less critical services.
- **Monitoring SLOs**: Use CloudWatch dashboards to track compliance; set alarms for SLO violations.
- **Resource Adjustment**: Scale resources dynamically based on SLO performance; example: add EC2 instances for high latency.
- **Continuous Review**: Regularly refine SLOs as business needs and user expectations evolve.

© 2025 by Innovation In Software Corporation

HANDLING OVERLOAD ERRORS GRACEFULLY

Retry-After Headers

Guides clients on when to retry, enhancing user experience.

Clear Error Messages

Provides actionable guidance to clients, improving user experience.

Circuit Breakers

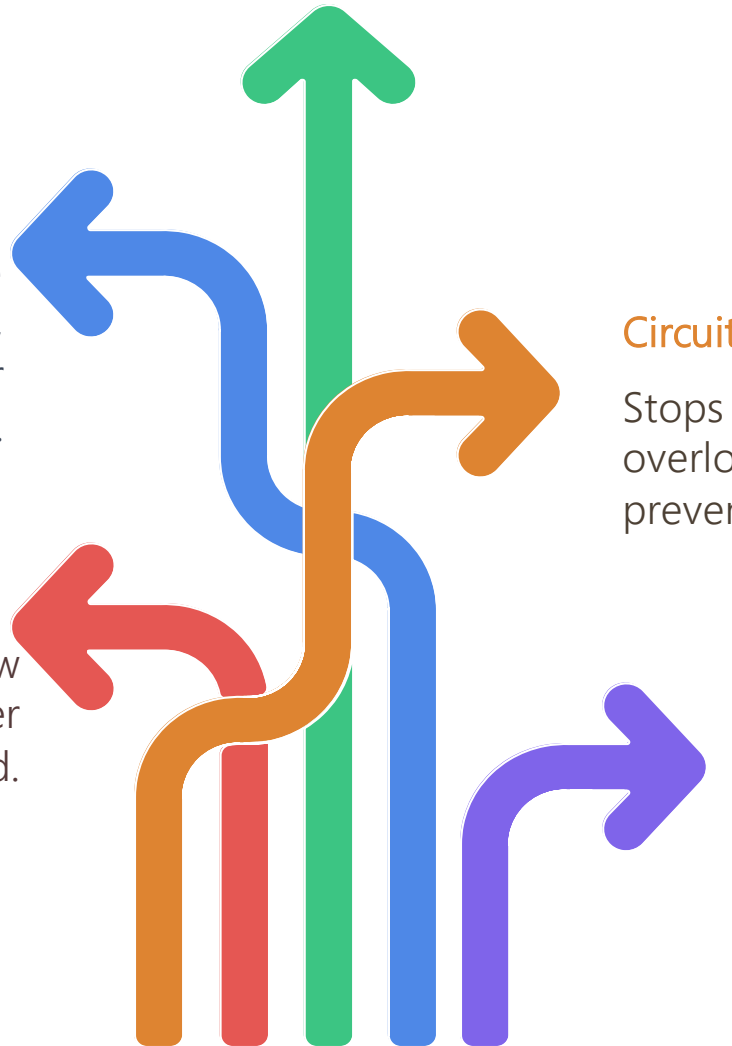
Stops requests to overloaded services, preventing system failure.

HTTP Status Codes

Informs clients to slow down, preventing further overload.

Error Logging

Logs errors for analysis and system improvement.



© 2025 by Innovation In Software Corporation

DECIDING WHEN TO RETRY REQUESTS

Transient vs. Permanent Errors: Retry transient errors (e.g., 500 Internal Server Error) but not permanent ones (e.g., 404 Not Found).

Exponential Backoff: Increase wait times between retries; example: wait 1s, 2s, 4s after failures.

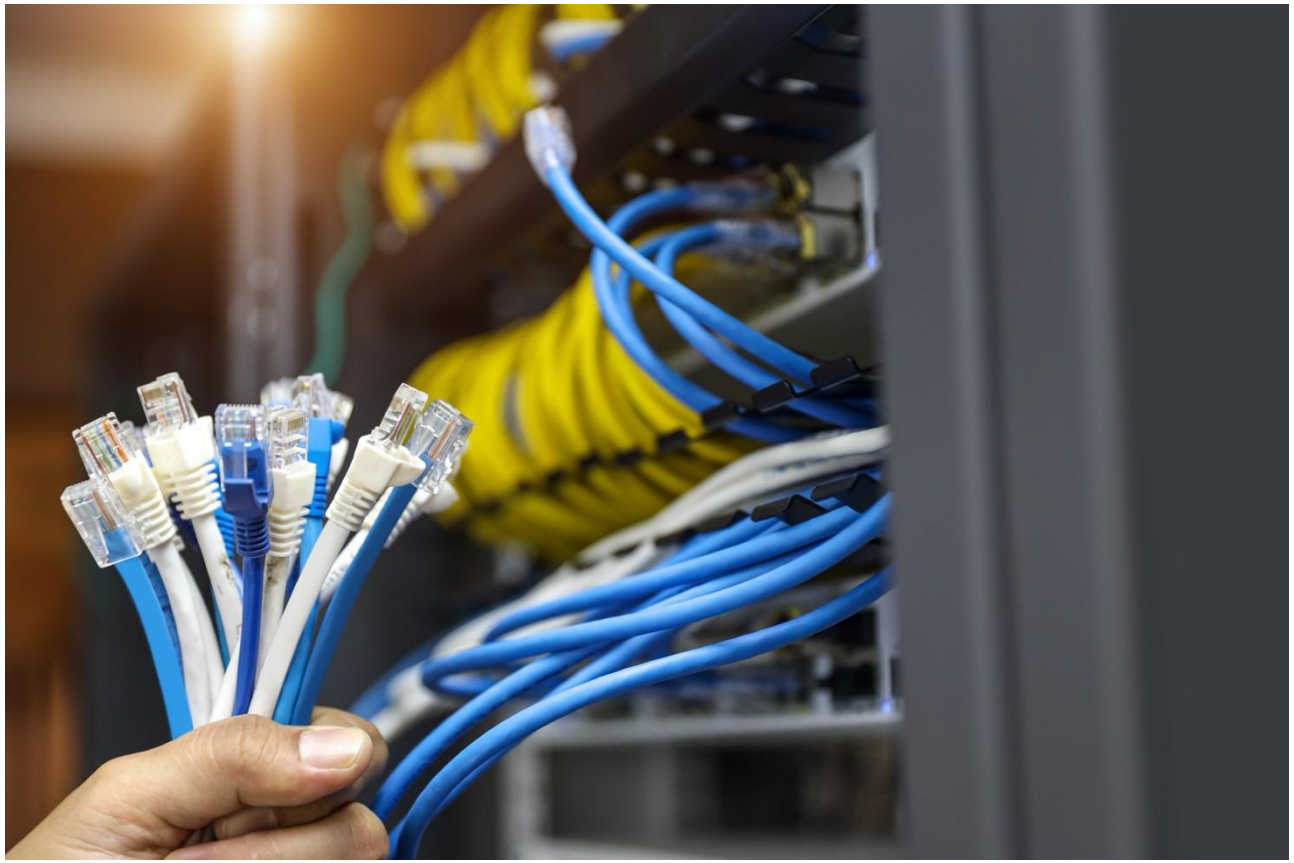
Maximum Retry Limits: Set limits (e.g., 3–5 retries) to prevent infinite loops and excessive load.

AWS SDK Support: Built-in retry logic with backoff; simplifies implementation for API calls.

Monitor Retries: Track retry rates with CloudWatch; high rates indicate underlying issues needing resolution.

© 2025 by Innovation In Software Corporation

MANAGING LOAD FROM CONNECTIONS



Long-Lived Connections: WebSockets or persistent connections consume resources; manage carefully.

Connection Limits: Set maximum connections per user/app; example: limit 100 WebSocket connections per client.

Timeouts: Implement timeouts for idle connections; example: close connections idle for 60 seconds.

Monitoring: Track connection counts/durations with CloudWatch; set alarms for anomalies.

Scaling: Use Auto Scaling to add resources based on connection metrics; example: scale WebSocket servers.

© 2025 by Innovation In Software Corporation

MANAGING LOAD FROM CONNECTIONS

API Gateway: Offers rate limiting, throttling, and caching; controls API request volumes.

WAF: Protects against web exploits/DDoS; reduces malicious traffic load.

Shield Advanced: Enhances DDoS protection; ideal for critical apps.

Auto Scaling: Dynamically adjusts EC2 instances; prevents server overload.

CloudFront: CDN caches content; reduces origin server load.

© 2025 by Innovation In Software Corporation

POP QUIZ:

Your AWS application experiences frontend overload due to synchronous API calls overwhelming EC2 instances. Which queue-based strategy best mitigates this while ensuring reliability?
A. Use SQS to buffer all API requests, scaling Lambda consumers based on queue length.
B. Configure SNS to broadcast API requests to multiple EC2 instances for parallel processing.
C. Implement Kinesis streams to batch API requests, processing them in fixed intervals.
D. Route all API requests directly to ECS tasks, bypassing queuing for faster response.

© 2025 by Innovation In Software Corporation

POP QUIZ:

Your AWS application experiences frontend overload due to synchronous API calls overwhelming EC2 instances. Which queue-based strategy best mitigates this while ensuring reliability?

A. Use SQS to buffer all API requests, scaling Lambda consumers based on queue length.

B. Configure SNS to broadcast API requests to multiple EC2 instances for parallel processing.

C. Implement Kinesis streams to batch API requests, processing them in fixed intervals.

D. Route all API requests directly to ECS tasks, bypassing queuing for faster response.

© 2025 by Innovation In Software Corporation

ADDRESSING CASCADING FAILURES

© 2025 by Innovation In Software Corporation

ADDRESSING CASCADING FAILURES

Definition: One failure triggers a chain reaction, affecting dependent components.

Example: Database outage causes web server crashes, impacting load balancers.

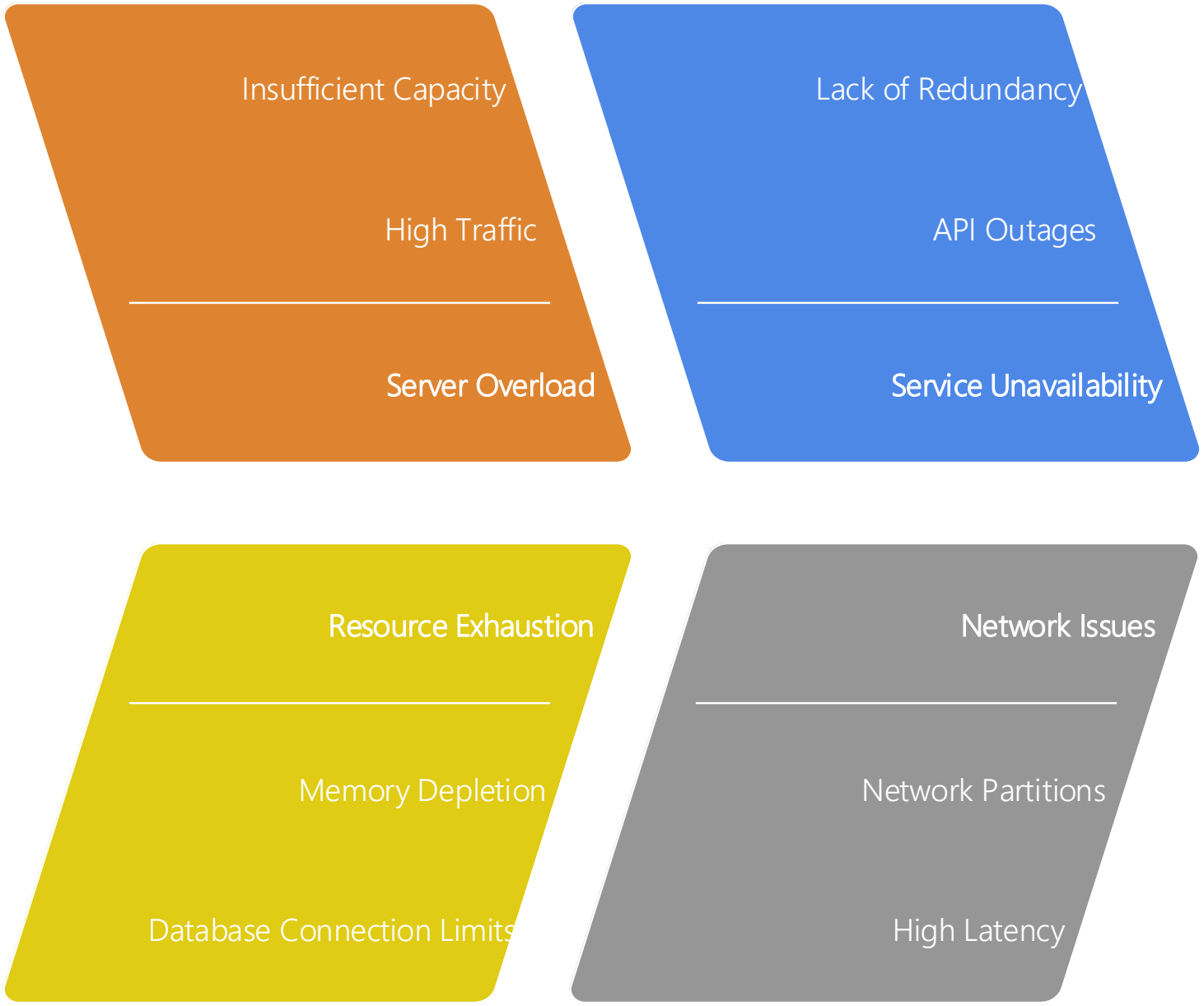
Cause: Tight coupling between services; one failure propagates rapidly.

Impact: Leads to widespread outages, threatening SLOs and user trust.

Prevention: Design with isolation, resilience, and monitoring; use AWS tools like FIS.

© 2025 by Innovation In Software Corporation

CAUSES OF CASCADING FAILURES



© 2025 by Innovation In Software Corporation

DESIGNING TO AVOID CASCADING FAILURES

Decoupling Services: Use SQS/Kinesis for asynchronous communication; reduces direct dependencies.

Circuit Breakers: Halt requests to failing services; example: Resilience4j in Java apps.

Bulkheads: Limit resources per service; example: separate DB connection pools.

Redundancy: Deploy across multiple AZs; example: Multi-AZ RDS for failover.

Monitoring: Use CloudWatch/X-Ray for early issue detection; set proactive alerts.

© 2025 by Innovation In Software Corporation

SERVER OVERLOAD PREVENTION

Auto Scaling: Adjusts EC2 instances based on demand; example: scale out when CPU > 70%.

Load Balancing: Distributes traffic with ALB/NLB; prevents single-server bottlenecks.

Rate Limiting: Controls request volume via API Gateway; example: limit 1000 requests/second.

Caching: Reduces backend load with ElastiCache; example: cache user session data.

Monitoring: Tracks CPU/memory with CloudWatch; sets alarms for high utilization.

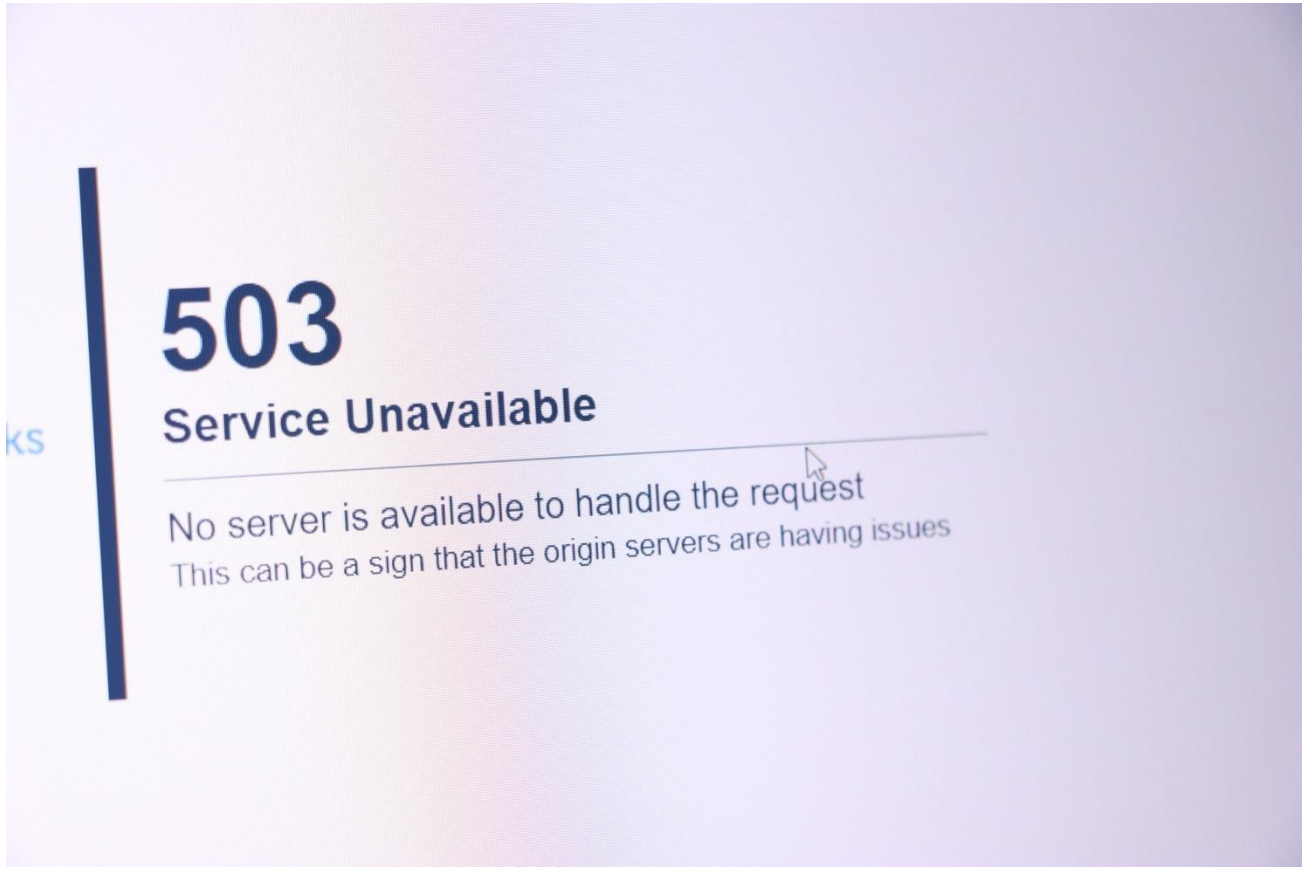
© 2025 by Innovation In Software Corporation

RESOURCE EXHAUSTION MANAGEMENT

- **Monitor Usage**: Track CPU, memory, disk, connections with CloudWatch; set alarms for high utilization.
- **Instance Optimization**: Use Compute Optimizer to select efficient instance types; example: t3.micro for light workloads.
- **Resource Quotas**: Limit resources per service; example: cap database connections at 100 per app.
- **Horizontal Scaling**: Add instances via Auto Scaling; avoids over-reliance on single large instances.
- **Configuration Audits**: Use AWS Config to review resource allocations; ensures optimal settings.

© 2025 by Innovation In Software Corporation

SERVICE UNAVAILABILITY HANDLING



- **Health Checks**: ALB target group health checks remove unhealthy instances; ensures only healthy targets receive traffic.
- **DNS Failover**: Route 53 redirects traffic to standby regions; example: failover to us-west-2 if us-east-1 fails.
- **Multi-AZ Deployments**: Deploy across AZs; example: RDS Multi-AZ auto-fails to standby instances.
- **Circuit Breakers**: Halt requests to unavailable services; example: Resilience4j stops DB queries during outages.
- **Alerting**: SNS notifications for critical failures; example: email on-call engineers when services go down.

© 2025 by Innovation In Software Corporation

QUEUE MANAGEMENT

- **Purpose**: Buffers requests during traffic spikes to manage workload effectively.
- **Benefits**: Decouples services, smooths load, enhances system resilience.
- **AWS Services**: Amazon SQS for message queuing, Kinesis for real-time streaming.
- **Best Practices**: Use FIFO queues for ordered messages, set visibility timeouts, monitor queue depth.
- **Example**: SQS handles asynchronous tasks like email notifications.

© 2025 by Innovation In Software Corporation

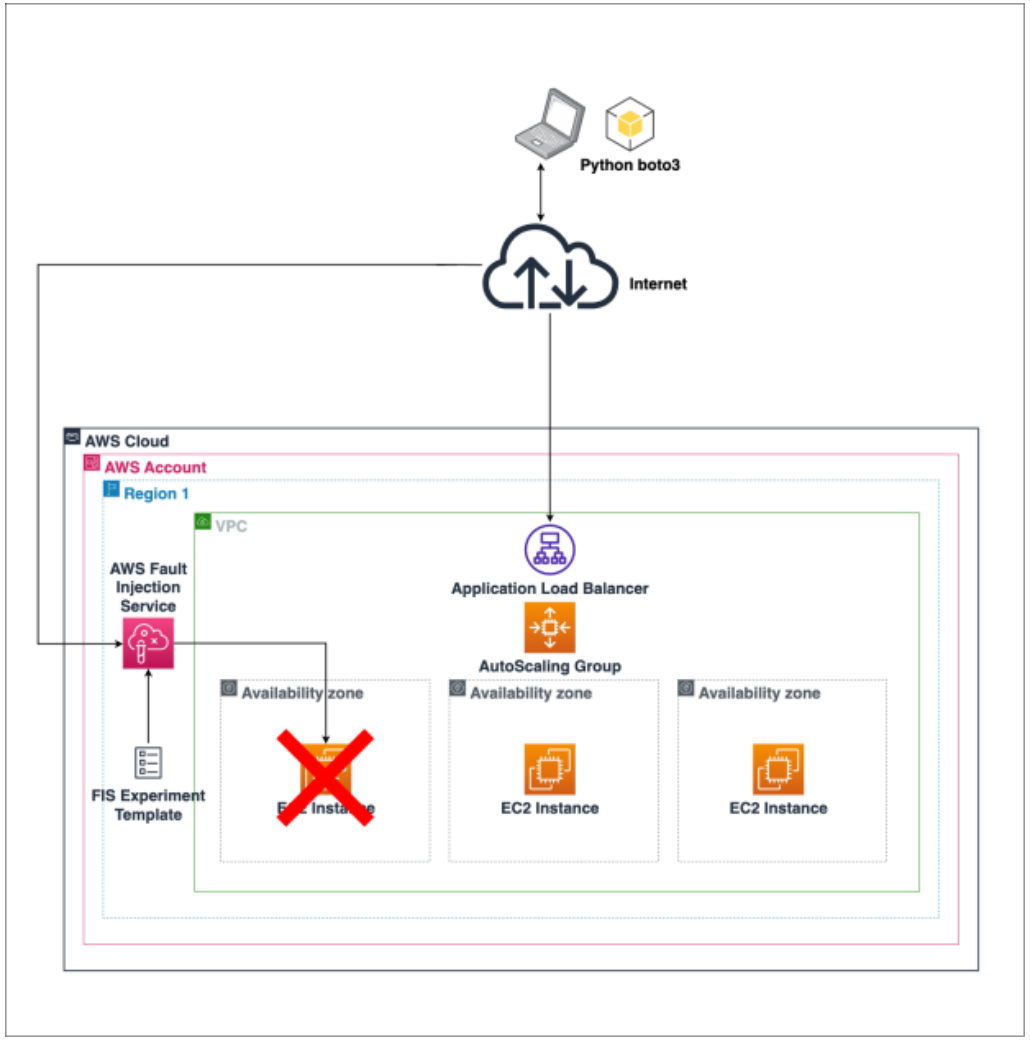
PLANNED CHANGES, DRAINS OR TURNDOWNS

- **Definition**: Scheduled maintenance or updates requiring temporary resource downtime.
- **Importance**: Minimizes disruption by timing changes for low-traffic periods.
- **AWS Tools**: EC2 termination protection, Auto Scaling, AWS Systems Manager for maintenance windows.
- **Best Practices**: Communicate plans, use blue/green deployments, monitor during changes.
- **Example**: Schedule RDS maintenance at midnight with Read Replicas for continuity.

© 2025 by Innovation In Software Corporation

TESTING FOR CASCADING FAILURES

- **Purpose**: Simulates failures to validate system resilience against cascading effects.
- **Approach**: Inject faults to test failure propagation and recovery mechanisms.
- **AWS Tools**: Fault Injection Simulator (FIS) for chaos testing, CloudWatch/X-Ray for monitoring.
- **Best Practices**: Start small, increase complexity, document findings in postmortems.
- **Example**: Simulate EC2 instance failure to test ALB failover and Auto Scaling response.



See: https://aws.amazon.com/blogs/architecture/behavior-driven-chaos-with-aws-fault-injection-simulator/

© 2025 by Innovation In Software Corporation

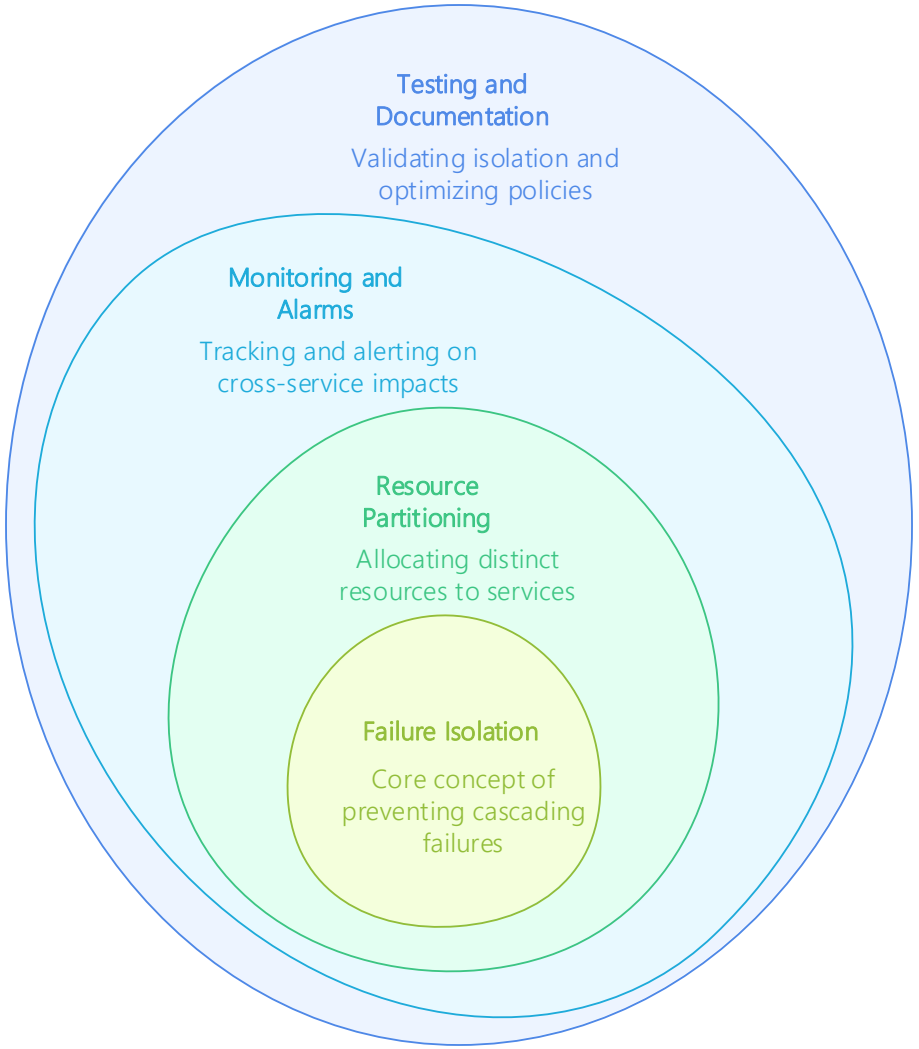
IMPLEMENTING CIRCUIT BREAKERS

- **Functionality**: Halts requests to failing services; prevents overload propagation.
- **Implementation**: Uses Resilience4j or API Gateway; example: stops DB queries on high latency.
- **State Management**: Tracks open/closed states; reopens after recovery.
- **Monitoring**: Logs breaker states with CloudWatch; analyzes trip frequency.
- **Benefits**: Isolates failures; maintains system stability during outages.

© 2025 by Innovation In Software Corporation

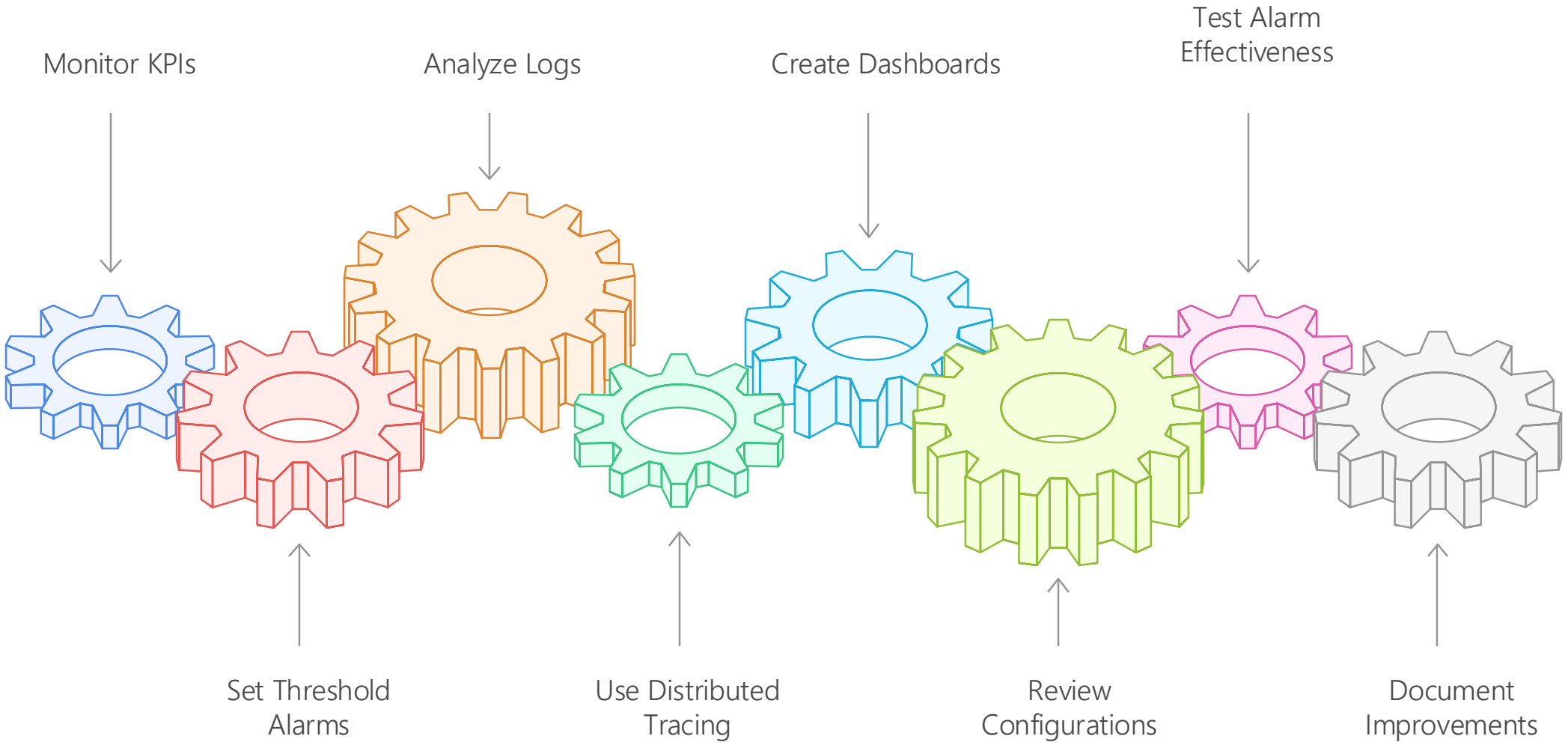
BULKHEADS & ISOLATION TECHNIQUES

- **Resource Isolation**: Limits resources per service; example: separate ECS tasks for APIs/DB.
- **Distinct Scaling Groups**: Uses separate Auto Scaling groups; prevents resource contention.
- **Failure Containment**: Restricts failure impact; example: one service crash doesn't affect others.
- **Configuration**: Sets resource quotas; example: caps CPU usage per Lambda function.
- **Monitoring**: Tracks isolation effectiveness with CloudWatch; ensures no cross-service impact.



© 2025 by Innovation In Software Corporation

MONITORING FOR EARLY DETECTION



© 2025 by Innovation In Software Corporation

AUTOMATION IN FAILURE RECOVERY

- **Instance Replacement**: Auto-replaces failed EC2 instances; uses Auto Scaling for rapid recovery.
- **Self-Healing Apps**: Implements retries/backoff; example: retries transient API failures.
- **Automated Patching**: Uses Systems Manager; applies updates without manual intervention.
- **Log Analysis**: Triggers Lambda for automated alerts; example: parses logs for errors.
- **Benefits**: Reduces MTTR; minimizes manual effort and human error.

© 2025 by Innovation In Software Corporation

AUTOMATION IN FAILURE RECOVERY

- **Instance Replacement**: Auto-replaces failed EC2 instances; uses Auto Scaling for rapid recovery.
- **Self-Healing Apps**: Implements retries/backoff; example: retries transient API failures.
- **Automated Patching**: Uses Systems Manager; applies updates without manual intervention.
- **Log Analysis**: Triggers Lambda for automated alerts; example: parses logs for errors.
- **Benefits**: Reduces MTTR; minimizes manual effort and human error.

© 2025 by Innovation In Software Corporation

LAB 9: RELIABLE REPLICATED STATE

Goal: Design a highly available key-value service using AWS services with multi-AZ deployment and DNS failover — without purchasing a public domain.

Core Concepts Covered:

- IAM Role setup for EC2-DynamoDB access
- Deploying EC2 instances across multiple availability zones
- Creating DynamoDB table for replicated key-value storage
- Configuring Route 53 Private Hosted Zones with health checks and failover routing
- Testing high availability and failover scenarios

Instructions: Lab9.md

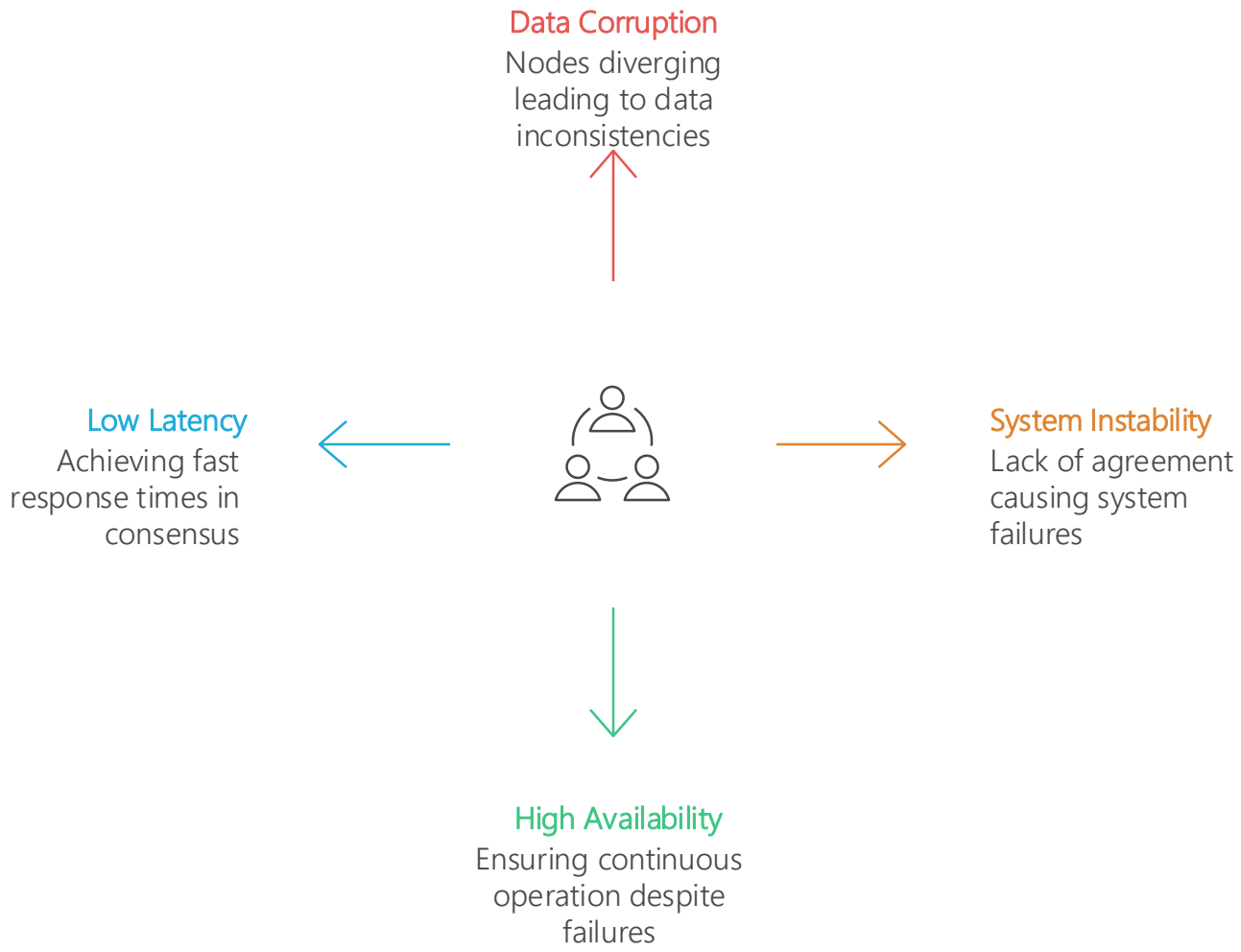
© 2025 by Innovation In Software Corporation

MANAGING CRITICAL STATE: DISTRIBUTED CONSENSUS FOR RELIABILITY

© 2025 by Innovation In Software Corporation

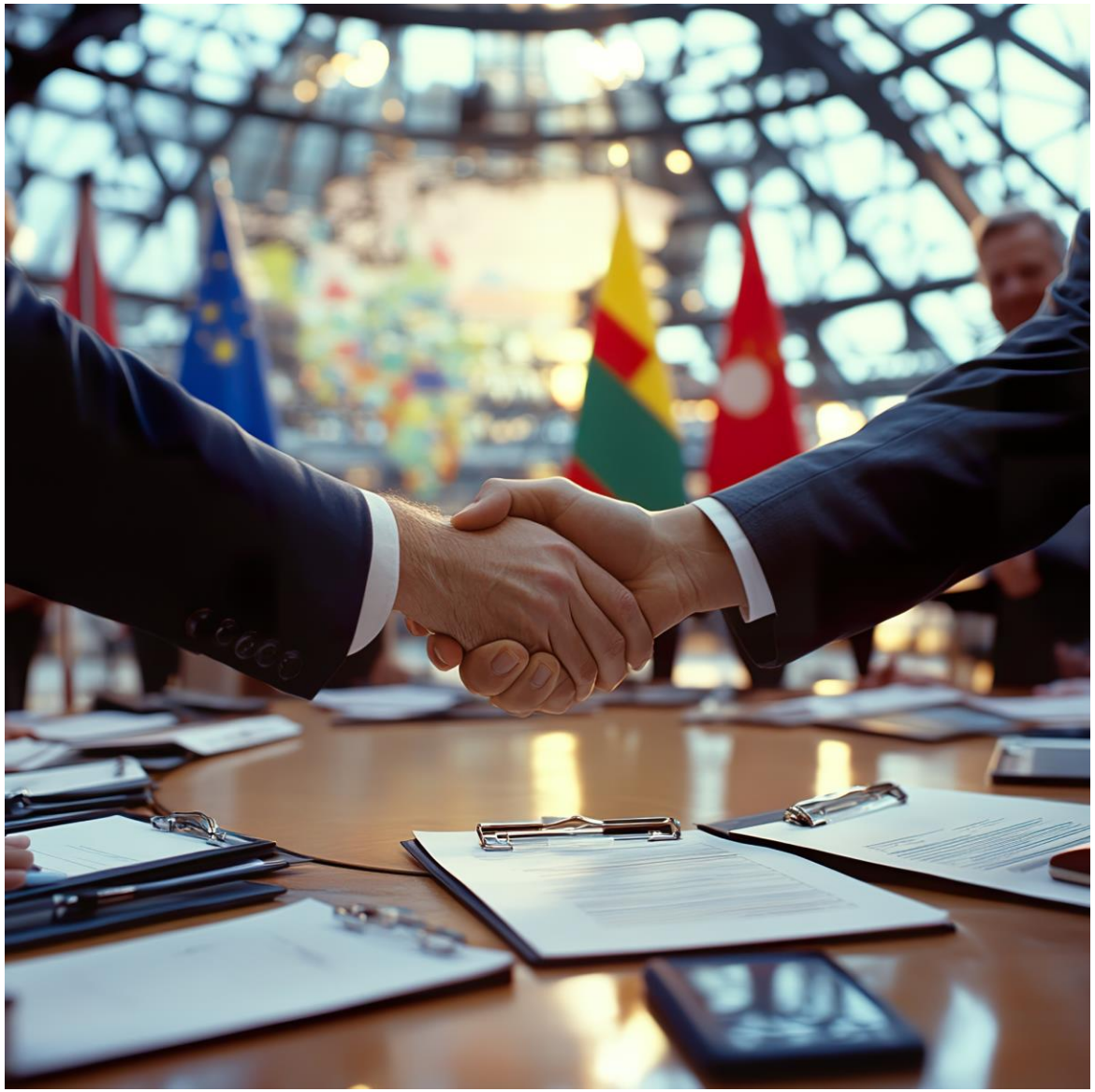
DISTRIBUTED CONSENSUS

Distributed consensus is the backbone of reliable distributed systems, ensuring all nodes agree on a single state even when failures or network partitions occur.



© 2025 by Innovation In Software Corporation

MOTIVATING THE USE OF CONSENSUS



Distributed consensus is the backbone of reliable distributed systems, ensuring all nodes agree on a single state even when failures or network partitions occur.

© 2025 by Innovation In Software Corporation

THE SPLIT-BRAIN PROBLEM

© 2025 by Innovation In Software Corporation

CAUSE OF THE SPLIT-BRAIN PROBLEM

- **Network Partitions**: Connectivity loss splits nodes; example: VPC peering failure isolates regions.
- **Weak Consensus**: Inadequate algorithms allow divergence; example: eventual consistency fails under partition.
- **Misconfigured Quorums**: Incorrect quorum settings; example: low quorum threshold permits split groups.
- **Node Failures**: Unhealthy nodes misjudged as active; causes conflicting operations.
- **Lack of Health Checks**: Fails to detect partitions; allows rogue nodes to process requests.

© 2025 by Innovation In Software Corporation

IMPACTS OF THE SPLIT-BRAIN PROBLEM

- **Data Inconsistency**: Conflicting writes corrupt data; example: mismatched transaction records.
- **SLO Violations**: Downtime or errors breach reliability targets; impacts user trust.
- **Manual Reconciliation**: Requires labor-intensive data fixes; delays recovery and increases costs.
- **User Experience**: Delivers incorrect results; example: wrong account balances frustrate customers.
- **Business Risk**: Damages reputation, revenue; critical for e-commerce, finance apps.

© 2025 by Innovation In Software Corporation

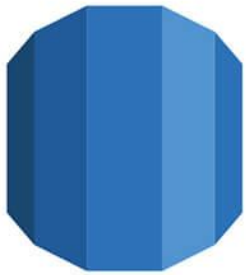
PREVENTING THE SPLIT-BRAIN PROBLEM

- **Strong Consensus**: Uses Paxos/Raft; ensures single authoritative node group.
- **Quorum Requirements**: Enforces majority vote; prevents partitioned groups from acting.
- **Health Checks**: Implements Route 53 checks; excludes unhealthy nodes from consensus.
- **Multi-AZ Deployment**: Distributes nodes across zones; reduces partition risk.
- **Monitoring**: Tracks quorum with CloudWatch; alerts on partition or quorum loss.

© 2025 by Innovation In Software Corporation

AWS TOOLS FOR THE SPLIT-BRAIN PROBLEM

Amazon DynamoDB



Amazon Aurora

Route53

Amazon Cloudwatch

© 2025 by Innovation In Software Corporation

FAILOVER REQUIRING HUMAN INTERVENTION

Manual failover processes are a significant bottleneck during outages, requiring human intervention to identify issues, decide on actions, and execute failovers.

© 2025 by Innovation In Software Corporation

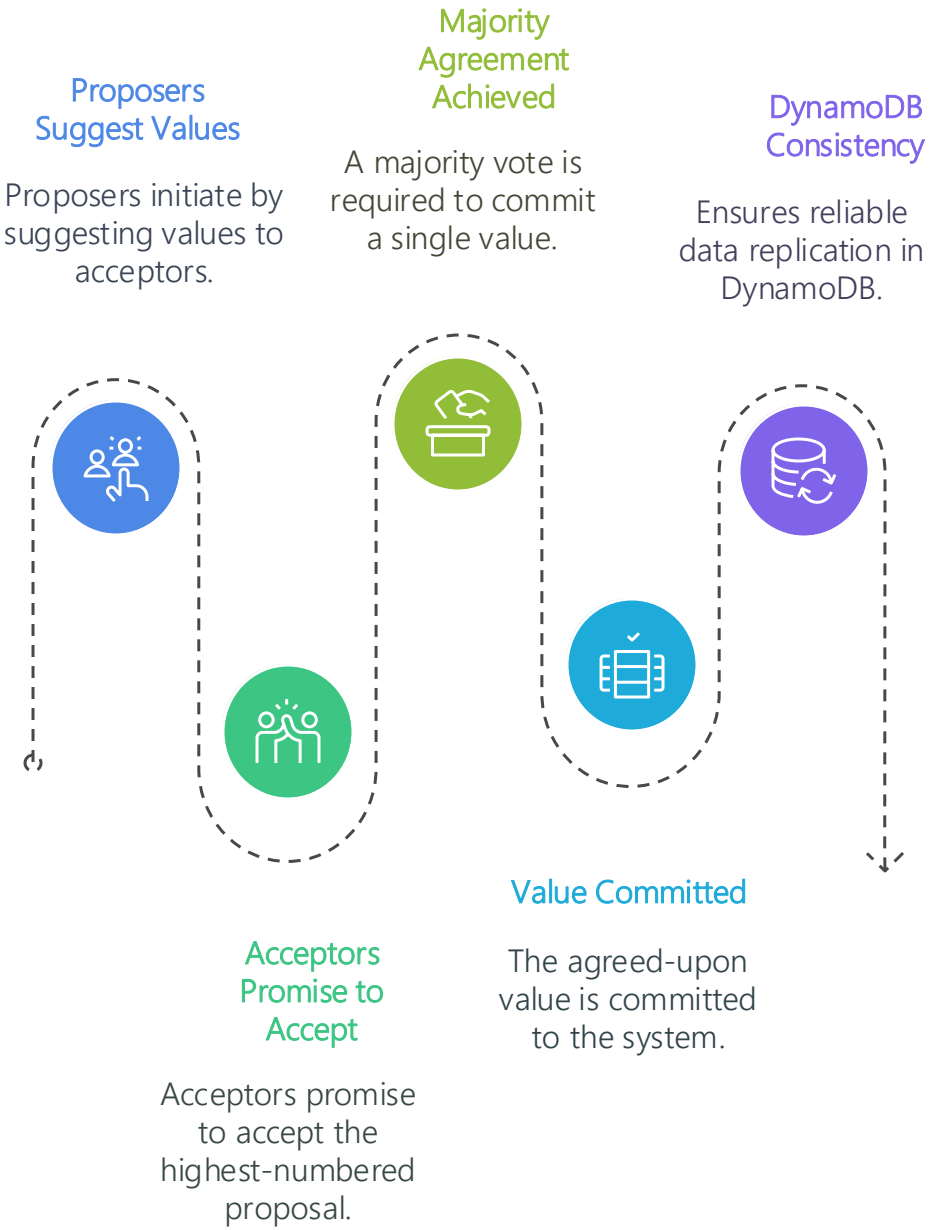
FAULT GROUP MEMBERSHIP ALGORITHMS

- **Incident Description**: Incorrect membership views caused inconsistencies; nodes misjudged others as failed.
- **Impact**: Data corruption in stateful services; required manual data reconciliation.
- **Root Cause**: Flawed membership algorithm; lacked robust consensus.
- **Solution**: Deployed etcd with Raft on EC2; ensured accurate membership.
- **AWS Tools**: Auto Scaling, CloudWatch; maintained correct group membership.

© 2025 by Innovation In Software Corporation

PAXOS OVERVIEW

- **Two-Phase Process**: Uses Prepare and Accept phases; coordinates node agreement.
- **Proposers**: Suggest values; send prepare requests to acceptors.
- **Acceptors**: Promise to accept highest-numbered proposals; ensure consistency.
- **Majority Agreement**: Requires majority vote; commits a single value.
- **AWS Context**: Underpins DynamoDB's consistency; ensures reliable data replication.



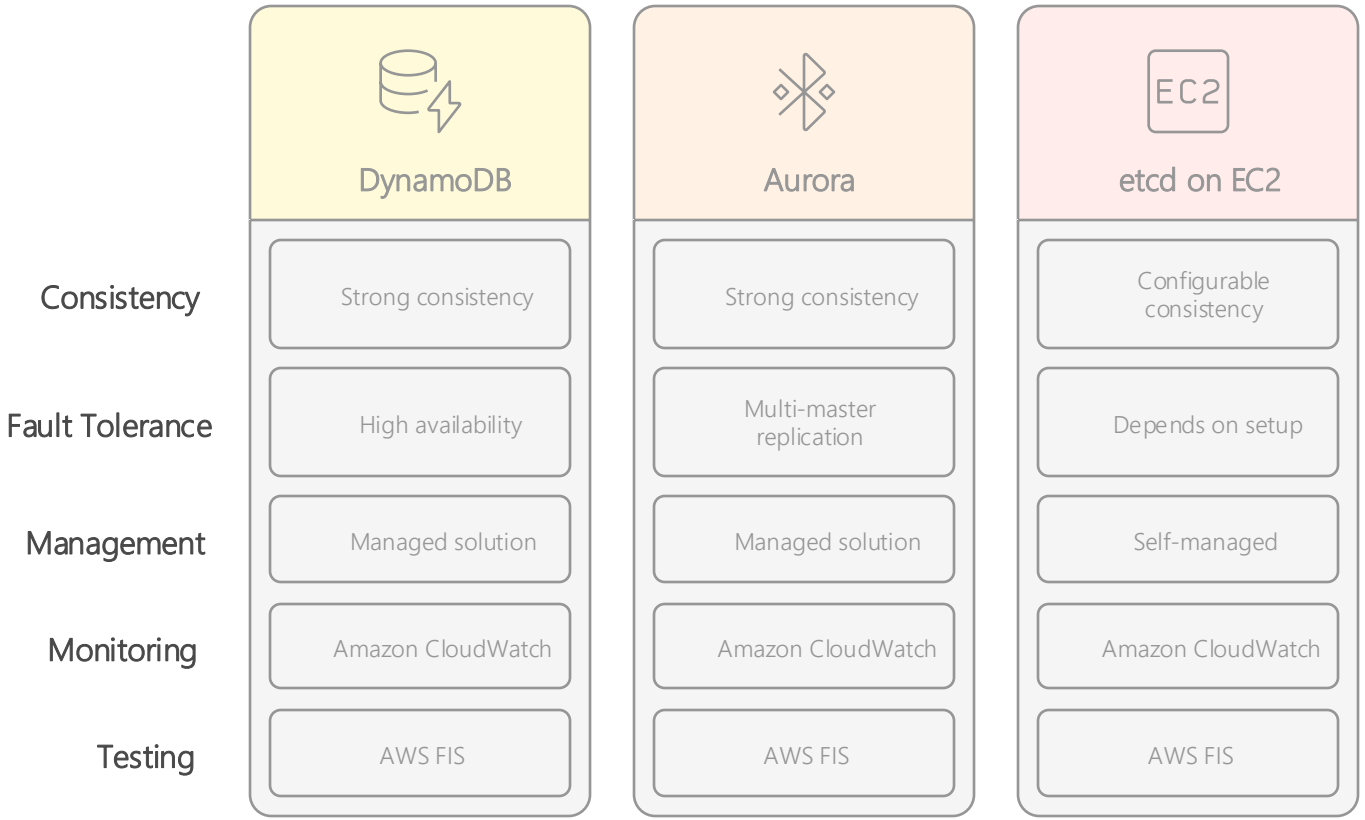
© 2025 by Innovation In Software Corporation

SYSTEM ARCHITECTURE PATTERNS FOR CONSENSUS

- **Leader Election**: Selects a single leader; coordinates actions like DB writes.
- **Replicated State Machines**: Applies operations in order; ensures consistent state.
- **Quorum-Based Systems**: Requires majority vote; example: DynamoDB quorum reads/writes.
- **Eventual Consistency**: Allows temporary divergence; used in high-availability systems.
- **AWS Examples**: DynamoDB, Aurora, ECS; implement consensus for reliability.

© 2025 by Innovation In Software Corporation

RELIABLE REPLICATED STATE MACHINES



- **Leader Election**: Selects a single leader; coordinates actions like DB writes.
- **Replicated State Machines**: Applies operations in order; ensures consistent state.
- **Quorum-Based Systems**: Requires majority vote; example: DynamoDB quorum reads/writes.
- **Eventual Consistency**: Allows temporary divergence; used in high-availability systems.
- **AWS Examples**: DynamoDB, Aurora, ECS; implement consensus for reliability.

© 2025 by Innovation In Software Corporation

RELIABLE REPLICATED DATA STORES

- **Data Replication**: Uses consensus for consistency; replicates data across nodes.
- **High Availability**: Ensures data access despite failures; example: DynamoDB multi-AZ replication.
- **Auto-Failover**: Switches to healthy nodes; example: Aurora fails to standby in seconds.
- **Monitoring**: Tracks replication health with CloudWatch; alerts on replication lag.
- **Backup Strategy**: Regular backups with DynamoDB; protects against data loss.

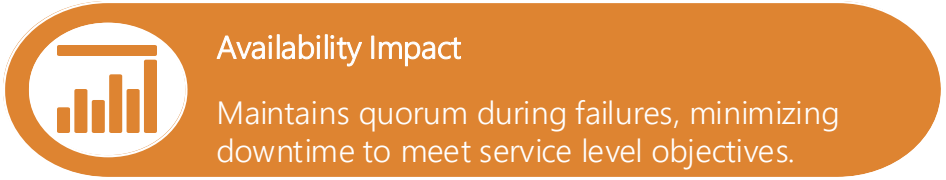
© 2025 by Innovation In Software Corporation

CONSENSUS AND SLO COMPLIANCE

Distributed consensus directly supports Service Level Objectives (SLOs) by ensuring data consistency and availability, critical for meeting stringent reliability targets like 99.99% uptime for critical applications.

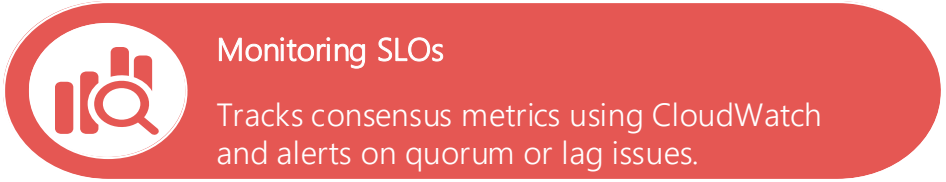
Consistency Guarantees

Ensures data integrity and supports high availability for critical applications.



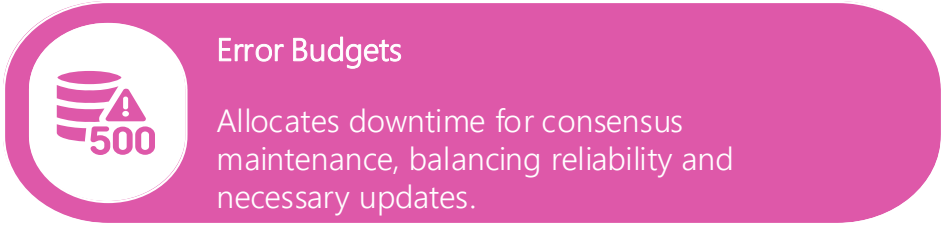
Availability Impact

Maintains quorum during failures, minimizing downtime to meet service level objectives.



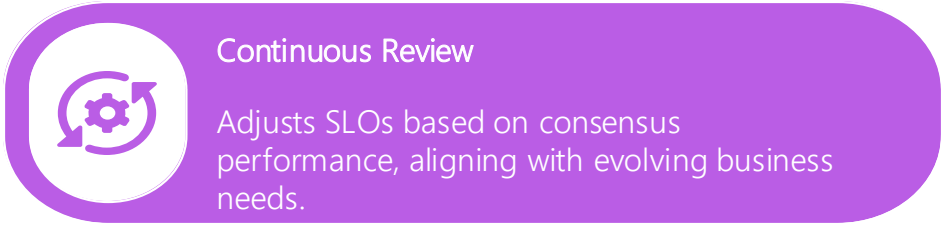
Monitoring SLOs

Tracks consensus metrics using CloudWatch and alerts on quorum or lag issues.



Error Budgets

Allocates downtime for consensus maintenance, balancing reliability and necessary updates.



Continuous Review

Adjusts SLOs based on consensus performance, aligning with evolving business needs.

© 2025 by Innovation In Software Corporation

RELIABLE REPLICATED DATA STORES

- **Data Replication**: Uses consensus for consistency; replicates data across nodes.
- **High Availability**: Ensures data access despite failures; example: DynamoDB multi-AZ replication.
- **Auto-Failover**: Switches to healthy nodes; example: Aurora fails to standby in seconds.
- **Monitoring**: Tracks replication health with CloudWatch; alerts on replication lag.
- **Backup Strategy**: Regular backups with DynamoDB; protects against data loss.

© 2025 by Innovation In Software Corporation

POP QUIZ:

As an SRE, you're monitoring an AWS frontend with ALB and ECS to detect early signs of cascading failures and maintain SLOs. Which monitoring strategy best identifies potential failure propagation?
A. Configure CloudTrail to log all ECS task configuration changes for auditing.
B. Set CloudWatch alarms for ALB's HTTP 200 response codes to track successful requests.
C. Use AWS X-Ray for distributed tracing and CloudWatch alarms for latency exceeding 500ms.
D. Monitor ECS task CPU usage to ensure sufficient resource allocation.

© 2025 by Innovation In Software Corporation

POP QUIZ:

As an SRE, you're monitoring an AWS frontend with ALB and ECS to detect early signs of cascading failures and maintain SLOs. Which monitoring strategy best identifies potential failure propagation?
A. Configure CloudTrail to log all ECS task configuration changes for auditing.
B. Set CloudWatch alarms for ALB's HTTP 200 response codes to track successful requests.
C. Use AWS X-Ray for distributed tracing and CloudWatch alarms for latency exceeding 500ms.
D. Monitor ECS task CPU usage to ensure sufficient resource allocation.

© 2025 by Innovation In Software Corporation

POP QUIZ:

Your AWS frontend, using EC2 instances behind an ALB, faces cascading failures due to instance crashes during traffic spikes. Which automation strategy best minimizes mean time to recovery (MTTR)?
A. Configure manual failover to standby EC2 instances during failures to ensure control.
B. Use Systems Manager to apply patches manually, ensuring stability before recovery.
C. Implement Auto Scaling to replace failed EC2 instances and Lambda for automated alerts.
D. Rely on CloudFront to serve cached responses, bypassing instance recovery.

© 2025 by Innovation In Software Corporation

POP QUIZ:

Your AWS frontend, using EC2 instances behind an ALB, faces cascading failures due to instance crashes during traffic spikes. Which automation strategy best minimizes mean time to recovery (MTTR)?

A. Configure manual failover to standby EC2 instances during failures to ensure control.

B. Use Systems Manager to apply patches manually, ensuring stability before recovery.

C. Implement Auto Scaling to replace failed EC2 instances and Lambda for automated alerts.

D. Rely on CloudFront to serve cached responses, bypassing instance recovery.

© 2025 by Innovation In Software Corporation

INDIVIDUAL KEY TAKEAWAYS

Write down three key insights from today's session.

Highlight how these take aways influence your work.

© 2025 by Innovation In Software Corporation

Q&A AND OPEN DISCUSSION

© 2025 by Innovation In Software Corporation


THANK
YOU