SITE RELIABILITY ENGINEERING – Release Engineering , Incident Management &Troubleshooting

TEKsystems Global Services

WORKFORCE DEVELOPMENT

PARTICIPANT GUIDE

REVIEW: DAY 1

Introduction to SRE and Core Principles:

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

Day 1:

Introduction to SRE and Core Principles.

Day 2:

Release Engineering, Incident Management & Troubleshooting

Day 3:

Testing Strategies and Advanced Topics.

AGENDA FOR DAY 2

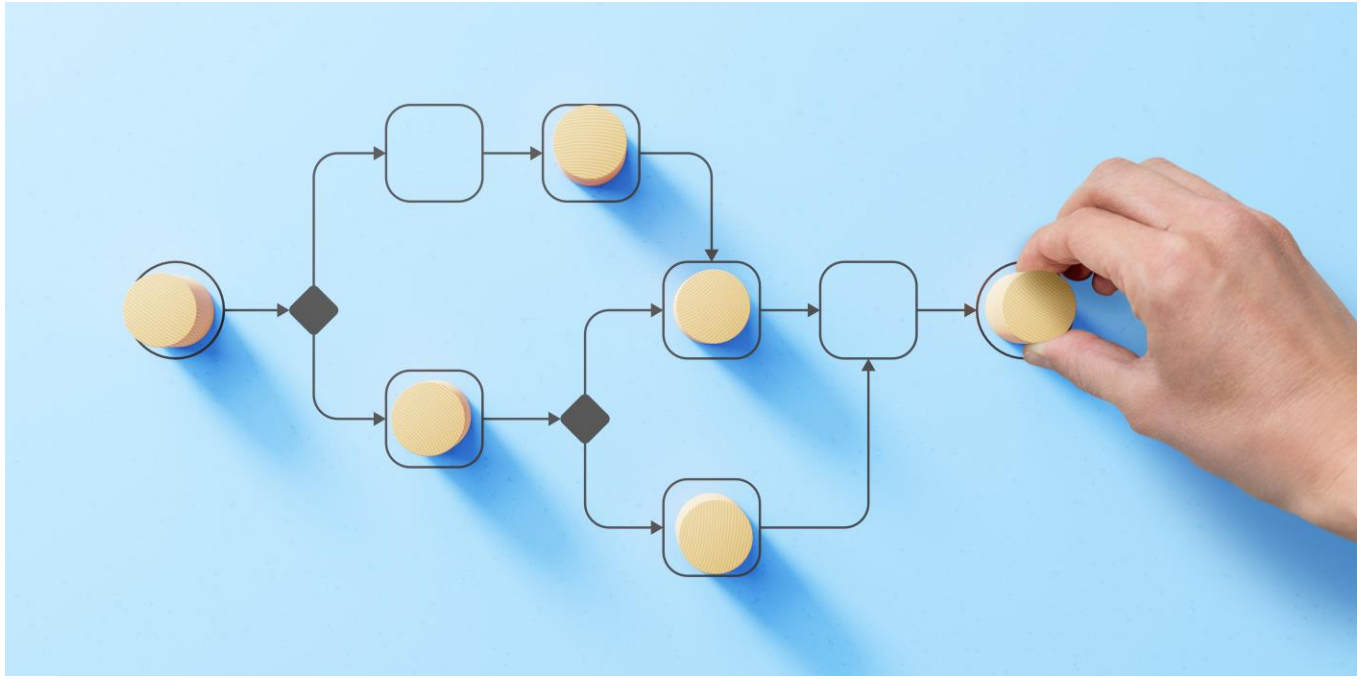
- Release Engineering
- Simplicity
- Alerting
- Troubleshooting
- Emergency Response
- Incident Management
- Tracking Outages

RELEASE ENGINEERING

RELEASE ENGINEERING

- Release Engineering consist of all the practices around building, packaging, versioning, testing and delivering software in a reliable, repeatable and automated way.

- Automation of every step, using repeatable code and configuration will be key contribution from the SRE engineering team



RELEASE ENGINEERING - KEY ASPECTS

Aspect	Description
Build Management	Automating and managing the process of compiling source code into executable software artifacts.
Packaging	Bundling software, libraries, and dependencies into deployable units (e.g., installers, Docker images, .zip, .tar.gz).
Versioning	Applying systematic version control (e.g., semantic versioning) for software releases.
Deployment Automation	Automating the delivery and deployment of builds across environments (dev, test, staging, production).
Release Coordination	Managing the timing, scope, and communication around software releases.
Continuous Integration/Continuous Delivery (CI/CD) Support	Integrating with CI/CD pipelines to streamline and automate testing and deployment.
Artifact Repository Management	Managing storage of build artifacts in repositories like Nexus, Artifactory, or Docker Registry.

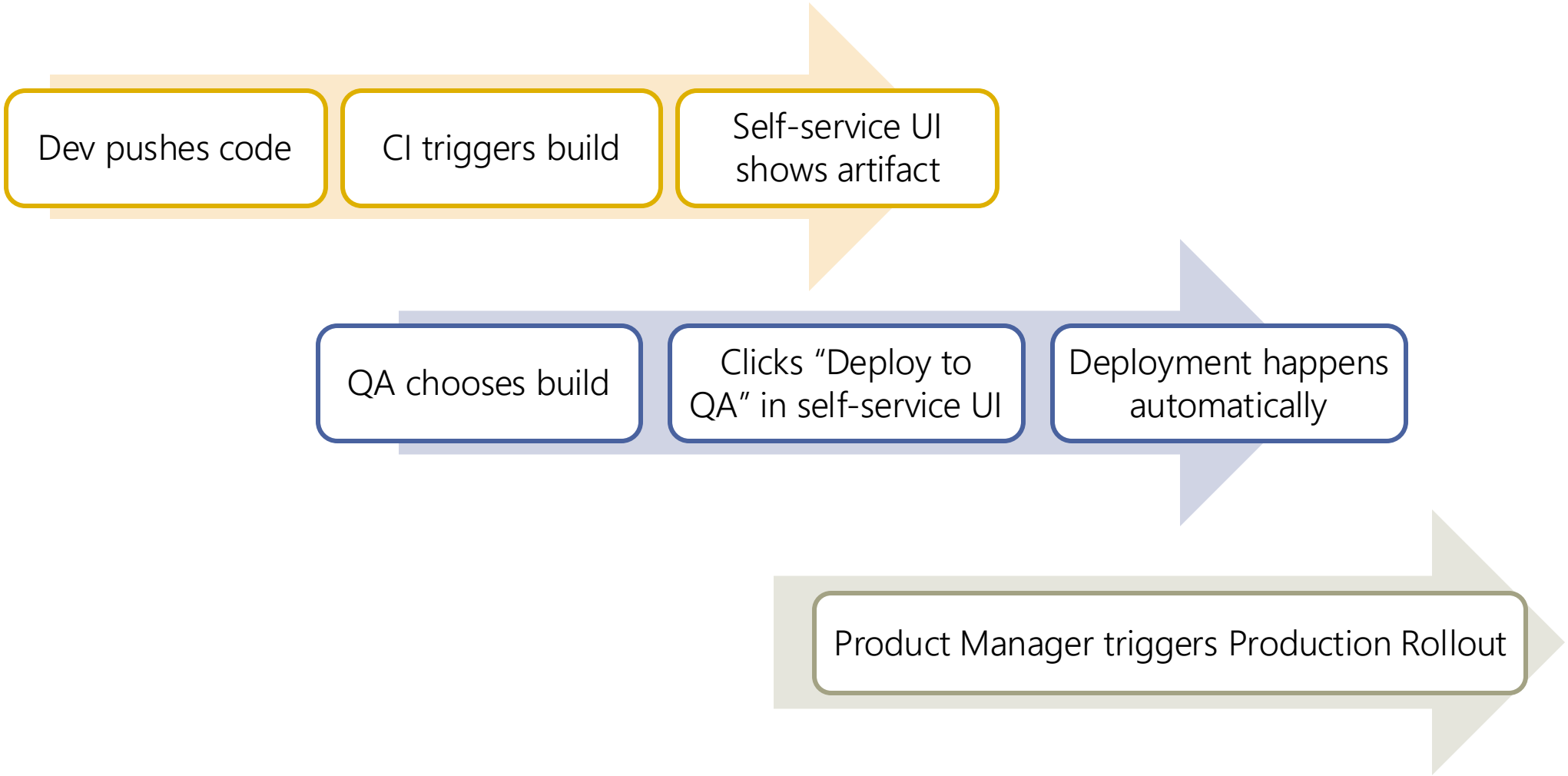
CHALLENGES OF IMPLEMENTING SRE

- **Fast :** Systems need to be agile to keep a competitive edge on our competitors or answer a regulation

- **Reliable : Fast is not messy.**
  Changing our system should be done without impacting reliability, even avoiding downtime if we implement the proper strategies

RELEASE ENGINEERING - SELF-SERVICE

- High-Level Self-Service Release Engineering Flow:



RELEASE ENGINEERING - SELF-SERVICE

- In Azure, Self-Service Release Engineering is typically implemented using a combination of Azure DevOps Services, Azure Pipelines, Azure Repos, Azure Artifacts.

Azure DevOps Pipelines: Core CI/CD platform to automate builds, tests, and deployments.

Azure Repos: Source control for managing code and pipeline definitions.

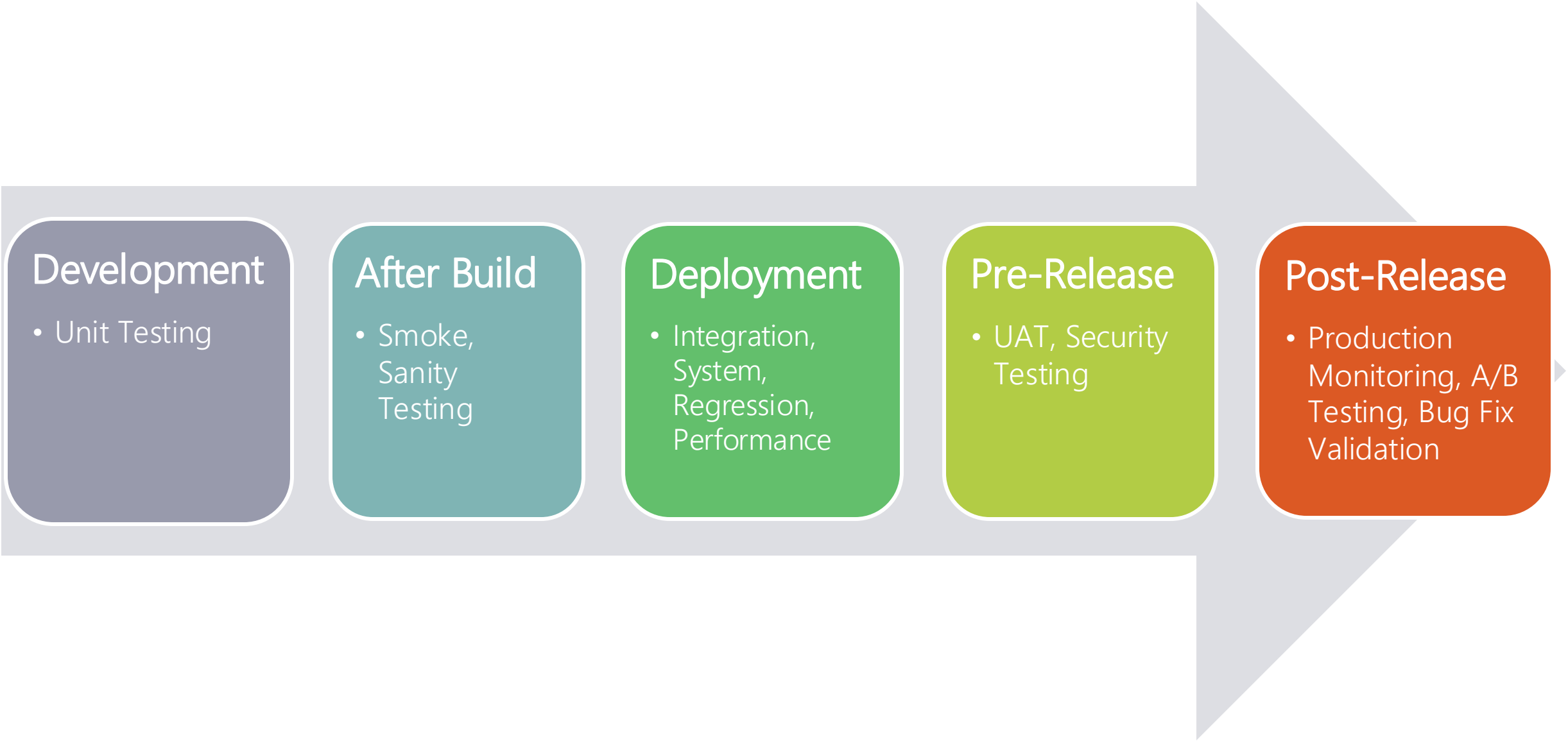
Azure Artifacts: Centralized storage for build outputs (e.g., NuGet, npm, Maven artifacts).

Service Connections: Securely connect pipelines to target environments (Azure subscriptions, Kubernetes clusters, etc.).

RBAC and Approvals: Control who can deploy to which environment

Azure Portal Dashboards: Expose status of builds, releases, environments for teams.

RELEASE ENGINEERING - TESTING



RELEASE ENGINEERING - PACKAGING

- Packaging in software engineering refers to the process of bundling compiled code, configuration files, libraries, dependencies, and other necessary resources into a single deployable unit.

- Easily distributed, deployed, or installed in target environments (Dev, Test, Staging, Production, etc.).

- Artifact storage tool : Azure Pipeline Artifacts, Azure Artifacts, Nexus, Git

Azure Artifatcs

RELEASE ENGINEERING - CONFIGURATION MGT

- Configuration Management (CM) is the practice of systematically handling changes to software systems, infrastructure, and environments to ensure consistency, traceability, and control throughout the software development lifecycle (SDLC) and deployment process.

- It focuses on tracking and managing software configurations to avoid discrepancies between environments (Dev, Test, Production).

Type	Examples
Software Code Configuration	Source code, build scripts
Infrastructure Configuration	Server specs, network settings, storage, load balancers
Application Configuration	App settings, environment variables, connection strings
Deployment Configuration	Deployment scripts, pipeline settings
Database Configuration	Schema definitions, connection settings, seed data

RELEASE ENGINEERING – INFRASTRUCTURE AS CODE

- Infrastructure as Code (IaC) is the practice of managing and provisioning computing infrastructure (servers, networks, databases, etc.) through machine-readable configuration files instead of manual hardware setup or GUI-based configuration tools.

Principle	Description
Declarative or Imperative	Define what the final infrastructure state should look like (Declarative) or how to achieve it step by step (Imperative).
Version-Controlled	Infrastructure definitions are stored in version control systems (e.g., Git) just like application code.
Idempotent	Running the same script multiple times results in the same infrastructure state (no unintended side effects).
Automated Provisioning	Infrastructure is created and configured automatically from code.
Environment Consistency	Dev, Test, and Prod environments are created from the same codebase, reducing drift.

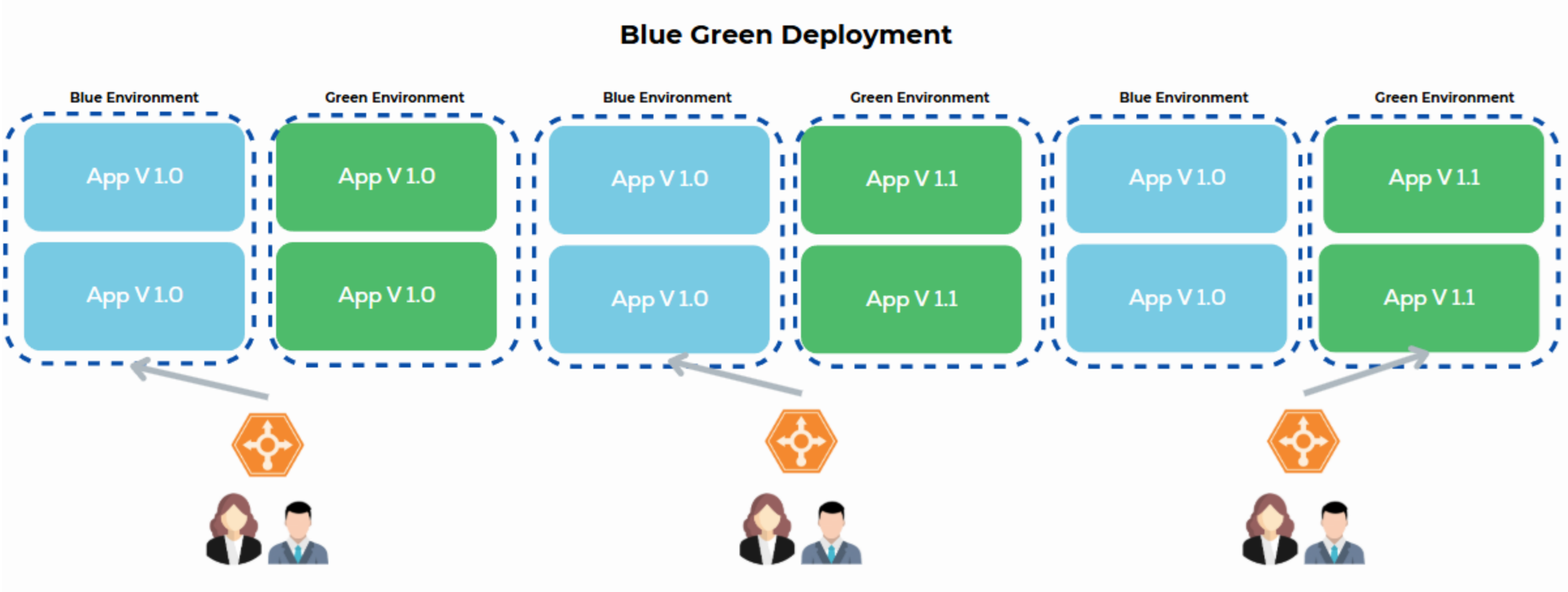
RELEASE ENGINEERING – DELIVERY STRATEGIES

- Delivery strategy is the approach to roll your changes into production safely, predictably and with minimal user impact.
- The criticality of the services provided will be the key factor that will lead to investment in different delivery strategies
- Any delivery strategy should involve an efficient rollback mechanism

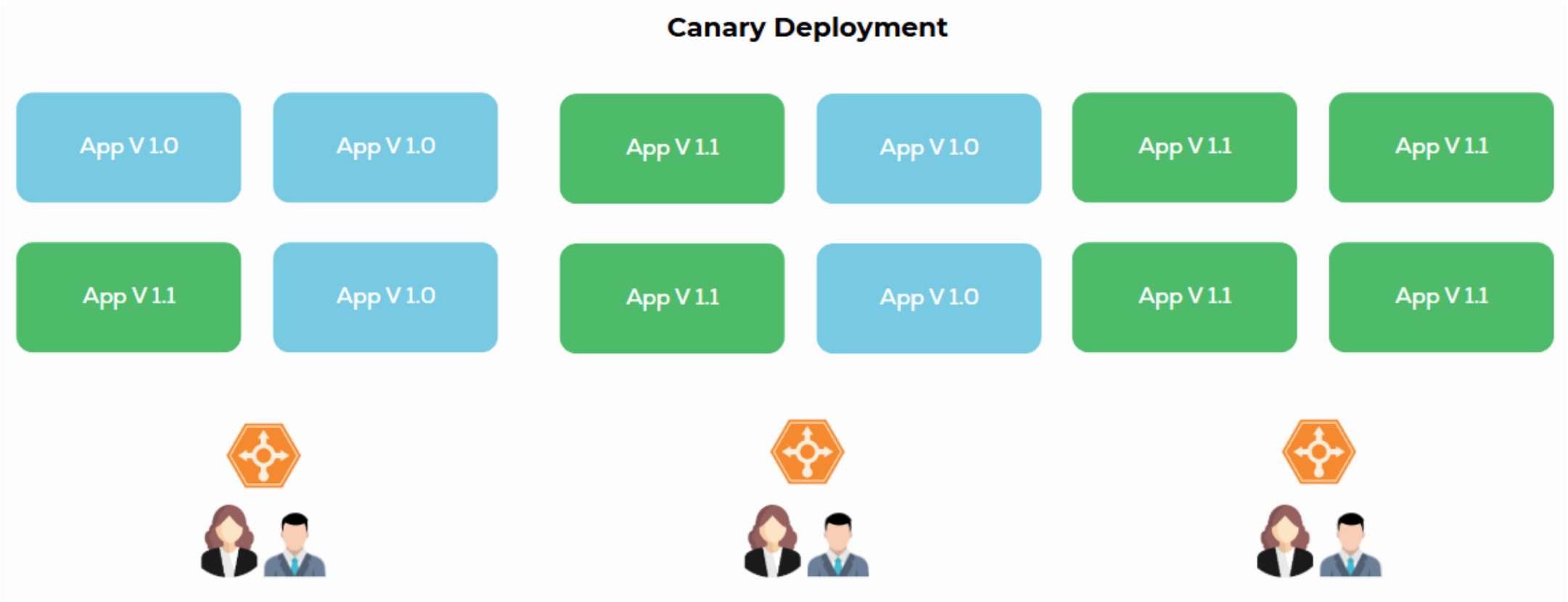
RELEASE ENGINEERING – DELIVERY STRATEGIES

Strategy	Key Idea	Pros	Cons
Blue-Green	Two identical environments; switch traffic	Fast rollback, minimal downtime	Costly (double infra)
Canary	Gradual rollout to user subsets	Limits risk, real-user testing	Traffic routing complexity
Rolling	Update servers one by one	No extra infra needed	Hard rollback, version mixing
Recreate	Stop old, start new	Simple, low-cost	Downtime risk
A/B Testing	Multiple live versions for user testing	User behavior insights	Complex management
Shadow	New version runs silently on real traffic	No user impact	Resource intensive
Feature Toggles	Deploy code, hide features behind toggles	Fast feature control	Code complexity
Immutable	New infra for every deployment	Avoids drift, safer releases	Slower, more orchestration

RELEASE ENGINEERING – BLUE GREEN DEPLOYMENT



RELEASE ENGINEERING – CANARY DEPLOYMENT



POP QUIZ:

In a typical CI/CD pipeline for software release engineering, each phase plays a critical role in ensuring software quality and reliability.
Which of the following best describes activities that occur during the build and deployment phases? (Choose two.)

A. Compiling source code and creating deployable artifacts during the build phase
B. Running automated performance and load tests during the deployment phase
C. Packaging and archiving source code without compiling it during the build phase
D. Deploying built artifacts to target environments like staging or production during the deployment phase

POP QUIZ:

In a typical CI/CD pipeline for software release engineering, each phase plays a critical role in ensuring software quality and reliability.
Which of the following best describes activities that occur during the build and deployment phases? (Choose two.)

A. Compiling source code and creating deployable artifacts during the build phase
B. Running automated performance and load tests during the deployment phase
C. Packaging and archiving source code without compiling it during the build phase
D. Deploying built artifacts to target environments like staging or production during the deployment phase

POP QUIZ:

Your organization is adopting a continuous deployment model as part of its release engineering strategy. To ensure safe and reliable releases, which of the following approaches should your team implement? (Choose two.)

A. Require manual intervention for every deployment to reduce automation risks
B. Implement automated rollback mechanisms in case of deployment failures
C. Deploy changes directly to production without going through staging environments
D. Use feature flags to control the rollout of new features without redeploying

POP QUIZ:

Your organization is adopting a continuous deployment model as part of its release engineering strategy. To ensure safe and reliable releases, which of the following approaches should your team implement? (Choose two.)

A. Require manual intervention for every deployment to reduce automation risks
B. Implement automated rollback mechanisms in case of deployment failures
C. Deploy changes directly to production without going through staging environments
D. Use feature flags to control the rollout of new features without redeploying

POP QUIZ:

You are working on the release engineering process for a large-scale distributed system. To improve reliability and minimize the risk of widespread failure during deployments, your team decides to adopt best practices for release engineering.
Which of the following practices align with this goal? (Choose two.)

A. Performing canary releases to limit blast radius
B. Releasing all code changes directly to production without testing
C. Automating the build and deployment pipeline to reduce manual errors
D. Deploying changes simultaneously across all production servers

POP QUIZ:

You are working on the release engineering process for a large-scale distributed system. To improve reliability and minimize the risk of widespread failure during deployments, your team decides to adopt best practices for release engineering.
Which of the following practices align with this goal? (Choose two.)

A. Performing canary releases to limit blast radius
B. Releasing all code changes directly to production without testing
C. Automating the build and deployment pipeline to reduce manual errors
D. Deploying changes simultaneously across all production servers

LAB 04: RELEASE ENGINEERING

Goal: In this lab, you will perform changes on the Web Frontend and the API. You will be using Deployments through GitHub Action, then switch your CI/CD pipeline to Azure Devops platform.

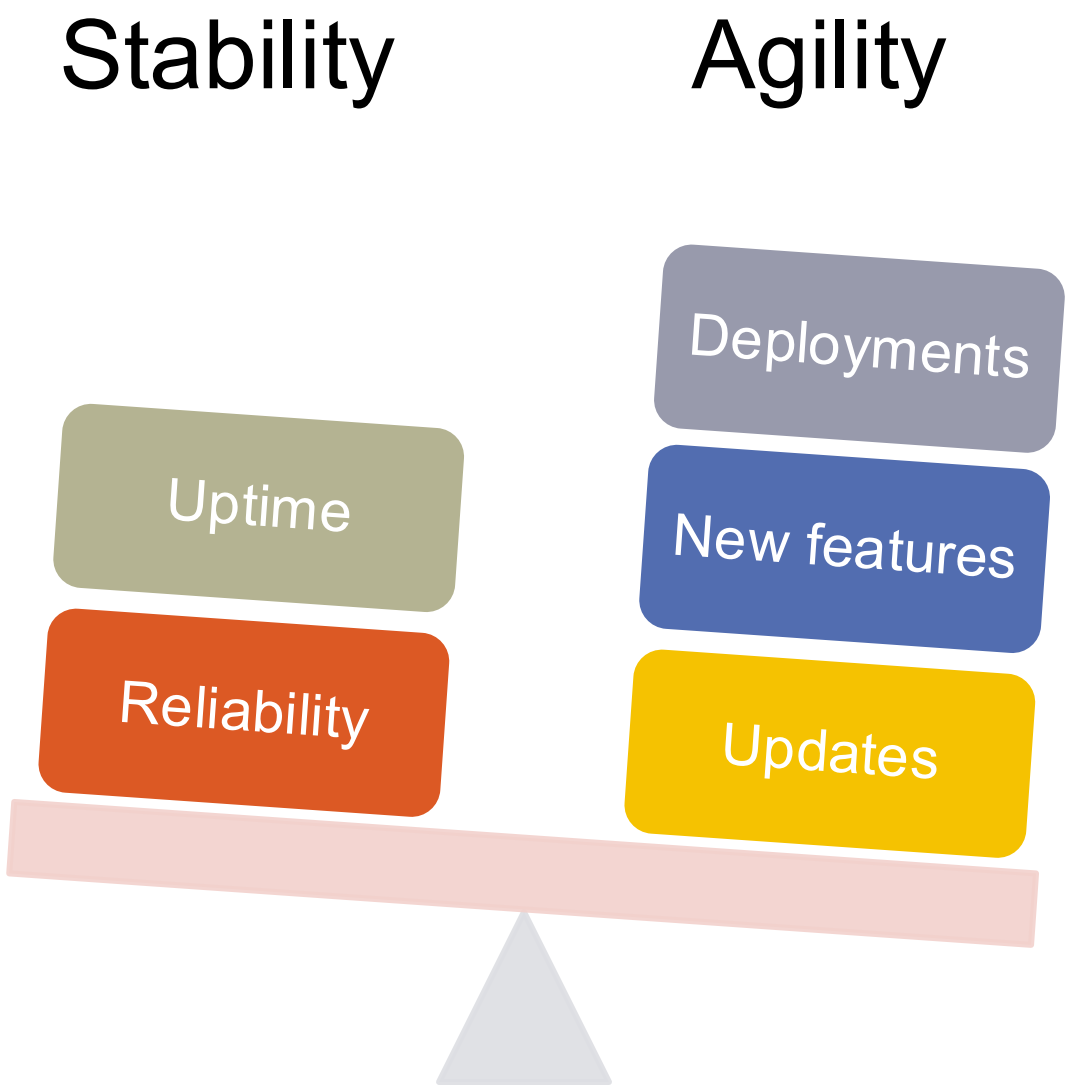
Skills Covered:

- Task 1: Deploying with GitHub Action
- Task 2: Packaging and Deploying with Azure Devops

Instructions: AZ_SRE_lab_04.md

SIMPLICITY

SIMPLICITY - STABILITY VS AGILITY



SIMPLICITY - THE VIRTUE OF BORING

- Favor mature and proven technologies
- Minimize unnecessary complexity
- Reduce operational surprises
- Improve reliability and predictability
- Focus on maintainability over novelty

SIMPLICITY - MINIMAL APIS

- **Design** small, focused interfaces
- **Expose** only essential functionality
- **Reduce** coupling between services
- **Simplify** testing and maintenance
- **Improve** reliability and scalability

SIMPLICITY - MINIMAL APIS BENEFITS

- With Azure Functions, teams can expose single-purpose functions rather than creating large monolithic API endpoints.

Benefit	Description
Less Risk of Failure	Fewer entry points reduce the chances of accidental misuse or misconfiguration.
Simpler Testing	Small, well-defined APIs are easier to unit test and integration test.
Easier Debugging	When things break, it's easier to isolate the root cause when the interface is small and predictable.
Improved Security	Smaller APIs have fewer attack surfaces, reducing security vulnerabilities.
Better Maintainability	Teams can make changes internally without worrying about breaking external clients.

SIMPLICITY - MODULARITY

- Design systems as independent, interchangeable modules
- Enable isolated development, testing, and deployment
- Minimize interdependencies between components
- Support scalability and fault isolation
- Simplify troubleshooting and system evolution
- **Example with Azure Functions:**

SIMPLICITY - MODULARITY

In Azure, modularity might mean designing a system where:

- **Azure Functions** handle independent compute tasks
- **Azure Service Bus** manages communication between modules
- **Azure App Services** run separate microservices
- **Azure Kubernetes Service (AKS)** runs containerized modular apps with independent scaling

SIMPLICITY – RELEASE

- What Does Release Simplicity Mean?
- Designing your deployment pipelines and release strategies so that:
  1. Releases are easy to perform
  2. Human error is minimized
  3. Failures are easier to detect and roll back
  4. Consistency is maintained across environments (Dev, QA, Staging, Production)

- Goal:
  - Lower the operational risk associated with releasing software
  - Make deployments predictable and safe

SIMPLICITY – RELEASE – HOW ?

Automated CI/CD Pipelines

- Automate builds, tests, and deployments

Infrastructure as Code

- Infrastructure deployments are repeatable and automated

Canary/Blue-Green Deployments

- Gradually release changes or keep production slots ready for fast rollback

Feature Flags

- Deploy code changes with features turned off until ready

Standardized Deployment Scripts

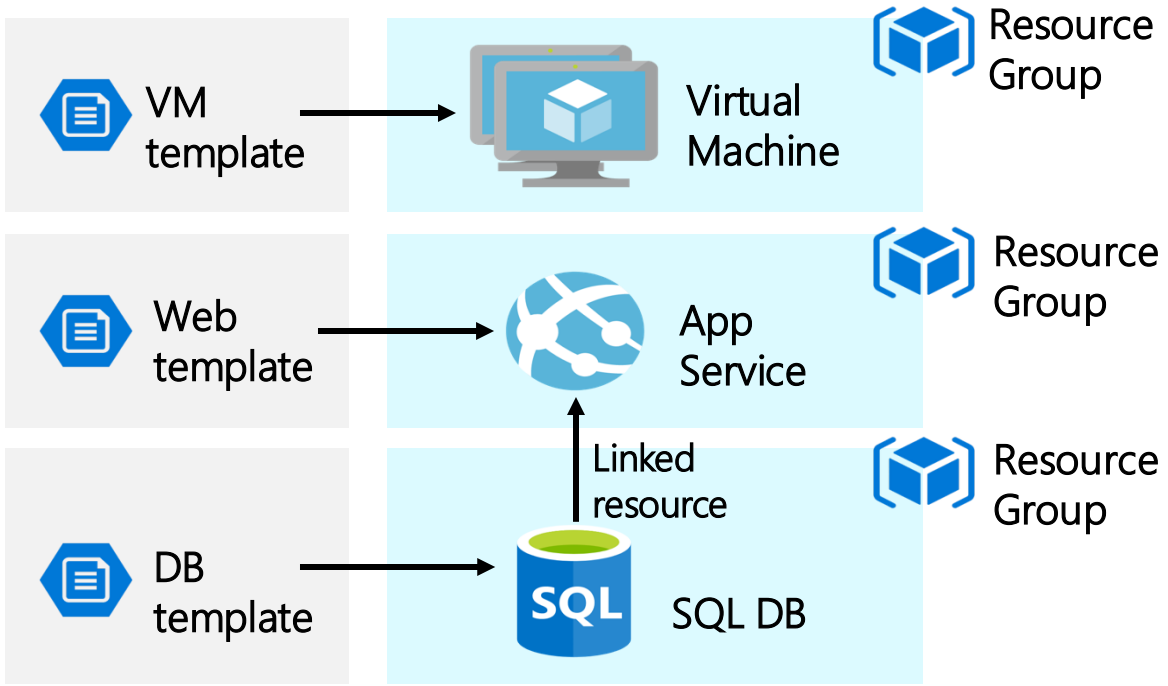
- Use version-controlled deployment scripts or templates so that each release follows the same steps

SIMPLICITY – RELEASE – IN AZURE

In Azure, Release Simplicity could involve:

- Using Deployment Slots in Azure App Services to deploy and swap environments without downtime.

- Implementing ARM or Bicep Templates for infrastructure deployment repeatability.



POP QUIZ:

As part of your release engineering strategy, your team decides to focus on simplicity and long-term reliability when selecting technologies and designing deployment processes.
Which of the following practices best align with this approach? (Choose two.)

A. Choosing mature and proven technologies instead of experimental tools
B. Introducing deployment scripts to automate edge cases and unlock the latest platform features
C. Streamlining the build and deployment pipeline to remove unnecessary steps
D. Adopting newer tools and frameworks early to stay ahead of technology shifts and improve developer satisfaction

POP QUIZ:

As part of your release engineering strategy, your team decides to focus on simplicity and long-term reliability when selecting technologies and designing deployment processes.
Which of the following practices best align with this approach? (Choose two.)

A. Choosing mature and proven technologies instead of experimental tools
B. Introducing deployment scripts to automate edge cases and unlock the latest platform features
C. Streamlining the build and deployment pipeline to remove unnecessary steps
D. Adopting newer tools and frameworks early to stay ahead of technology shifts and improve developer satisfaction

POP QUIZ:

Your team is adopting modular design principles to improve the simplicity and maintainability of your release engineering process.
Which of the following practices align with this approach? (Choose two.)

A. Designing deployment pipelines where all services must be deployed together to avoid version mismatches
B. Structuring your application into independent services that can be built and deployed separately
C. Creating clear interfaces and boundaries between different components of the system
D. Requiring every team to work on the entire codebase to ensure shared understanding and flexibility

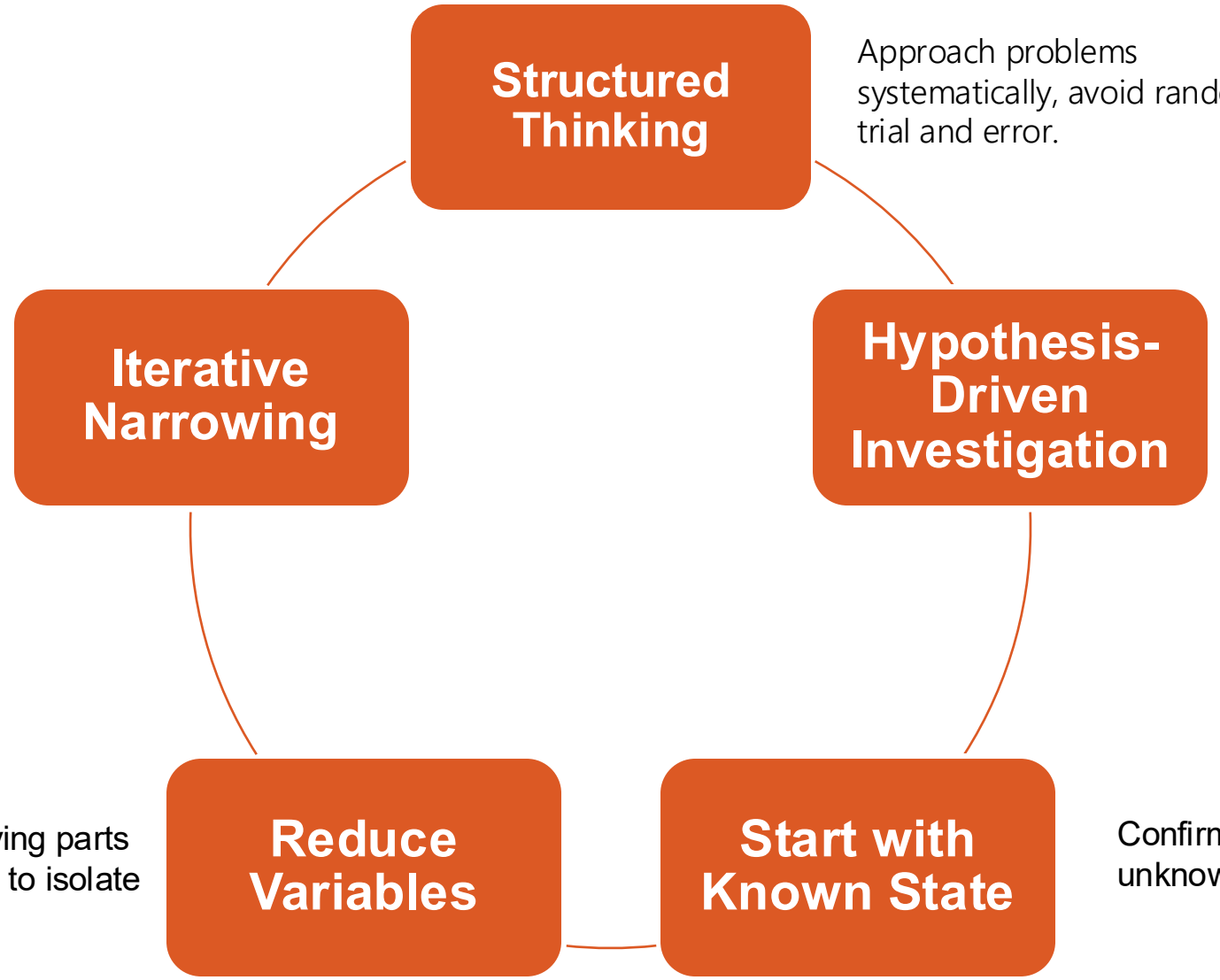
POP QUIZ:

Your team is adopting modular design principles to improve the simplicity and maintainability of your release engineering process.
Which of the following practices align with this approach? (Choose two.)

A. Designing deployment pipelines where all services must be deployed together to avoid version mismatches
B. Structuring your application into independent services that can be built and deployed separately
C. Creating clear interfaces and boundaries between different components of the system
D. Requiring every team to work on the entire codebase to ensure shared understanding and flexibility

TROUBLESHOOTING

TROUBLESHOOTING - THEORY



TROUBLESHOOTING – THE FLOW IN PRACTICE

1. **Acknowledge the Issue:**
   Confirm that there is a problem (user reports, monitoring alerts).

2. **Assess Impact and Determine the severity:**
   Affected components, and business/user impact.

3. **Stabilize First (If Needed):**
   If there's a production impact, implement immediate mitigations (like failover, scaling, or disabling risky features).

4. **Gather Facts and Observations:**
   Collect logs, metrics, error messages, timestamps, and user reports.

5. **Form Hypotheses:**
   Based on observed data, generate possible root cause theories.

6. **Test Hypotheses (One at a Time):**
   Run controlled experiments or tests to validate or eliminate each theory.

7. **Confirm and Fix:**
   Once the root cause is found, apply a targeted fix.

8. **Verify Resolution:**
   Confirm that the issue is fully resolved and won't reoccur immediately.

TROUBLESHOOTING – MINDSET

Mindset Rule	Description
Don’t Assume	Validate every assumption with evidence.
Simplify the Problem	Break complex systems into smaller, manageable parts.
Use Monitoring Data Effectively	Leverage logs, metrics, and traces.
Stay Calm Under Pressure	Production incidents are stressful, work methodically.
Document Your Steps	Helps with RCA later and assists team coordination.

TROUBLESHOOTING – COMMON PITFALLS

- **Making multiple changes at once**
  Makes it impossible to know which change fixed or broke something

- **Poor communication**
  Leads to duplicated work or dangerous changes

- **Not documenting changes**
  Hurts post-incident analysis and Root-Cause-Analysis

- **Ignoring rollback options**
  Leads to longer outages if new changes worsen the problem

TROUBLESHOOTING - PROBLEM REPORT

- Why Create a Problem Report?

Knowledge Sharing	Helps the wider engineering and operations teams learn from incidents.
Prevents Repeat Failures	Provides inputs for implementing corrective actions and systemic fixes.
Drives Continuous Improvement	Forms the basis for post-incident reviews and SLO reliability improvements.
Supports Audit and Compliance	Necessary documentation for regulatory and customer-facing audits.

TROUBLESHOOTING - PROBLEM REPORT TEMPLATE

Section	Contents
1. Title / Summary	Short description of the incident .
2. Timeline of Events	Chronological list of events (alert times, actions taken, recovery time).
3. Impact Analysis	Who/what was affected (users, services, business), duration, and severity.
4. Detection Method	How was the issue detected (alert, user report, monitoring dashboard)?
5. Root Cause	The underlying cause that triggered the failure.
6. Contributing Factors	Any secondary factors that made the impact worse .
7. Mitigation and Recovery Steps	Actions taken during the incident to stabilize the system.
8. Corrective Actions /	Planned fixes (code changes, infra changes, monitoring improvements).
9. Preventative Measures	Long-term steps to prevent similar incidents (SLO tuning, better testing).
10. Attachments/Links	Logs, dashboards, pull requests, related incidents.

TROUBLESHOOTING – PROBLEM – BEST PRACTICES

Be Blameless: Focus on systemic issues, not individuals.

Be Detailed and Honest: Even embarrassing details help prevent future failures.

Include Data and Evidence: Support findings with metrics, logs, and screenshots.

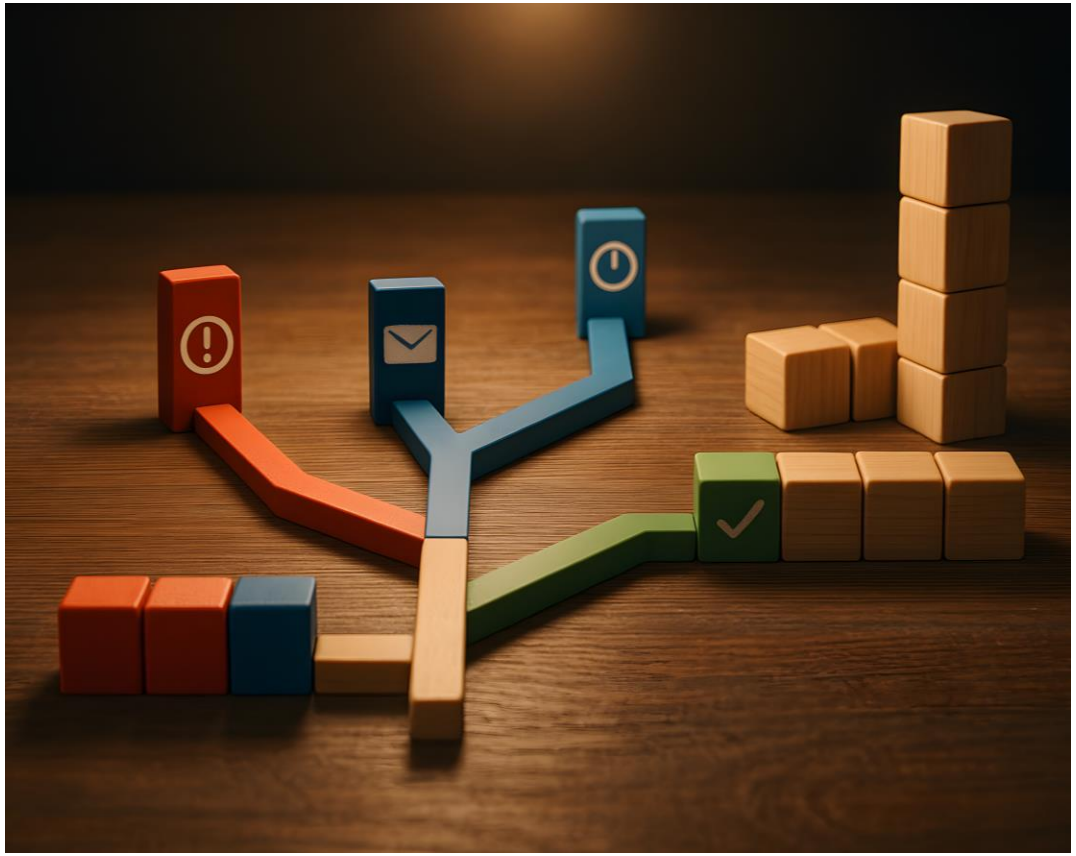
Summarize Clearly: Senior management often reads problem reports. Keep them concise.

Highlight Action Items: Clearly define next steps and who owns them.

TROUBLESHOOTING - TRIAGE

When an incident starts, SREs must quickly determine:

- What's happening?
- How bad is it?
- Who and what is affected?
- What immediate actions are needed
- Who needs to be involved?



POP QUIZ:

Your team is troubleshooting a production performance issue after a recent deployment.
Which of the following practices align with troubleshooting best practices in a production environment? (Choose two.)

A. Making untested configuration changes directly in production to fix the issue quickly
B. Using available logs, metrics, and traces to gather evidence before attempting a fix
C. Reproducing the issue in a test or staging environment when possible
D. Assuming the root cause based on past incidents without analyzing current data

POP QUIZ:

Your team is troubleshooting a production performance issue after a recent deployment.
Which of the following practices align with troubleshooting best practices in a production environment? (Choose two.)

A. Making untested configuration changes directly in production to fix the issue quickly
B. Using available logs, metrics, and traces to gather evidence before attempting a fix
C. Reproducing the issue in a test or staging environment when possible
D. Assuming the root cause based on past incidents without analyzing current data

POP QUIZ:

During a production incident, your team begins troubleshooting an application outage.
Which of the following actions represent common troubleshooting pitfalls that should be avoided? (Choose two.)

A. Making changes directly in production without a clear understanding of the problem
B. Collaborating with team members to share observations and findings
C. Ignoring monitoring data and relying solely on assumptions
D. Using structured logging and metrics to guide investigation steps

POP QUIZ:

During a production incident, your team begins troubleshooting an application outage.
Which of the following actions represent common troubleshooting pitfalls that should be avoided? (Choose two.)

A. Making changes directly in production without a clear understanding of the problem
B. Collaborating with team members to share observations and findings
C. Ignoring monitoring data and relying solely on assumptions
D. Using structured logging and metrics to guide investigation steps

LAB 05: TROUBLESHOOTING IN AZURE

Goal: In this lab, you will perform changes on the Web Frontend and the API that will lead to service degradation on your API server. You will use Application Insights to link the degradation to the endpoints you newly deployed.

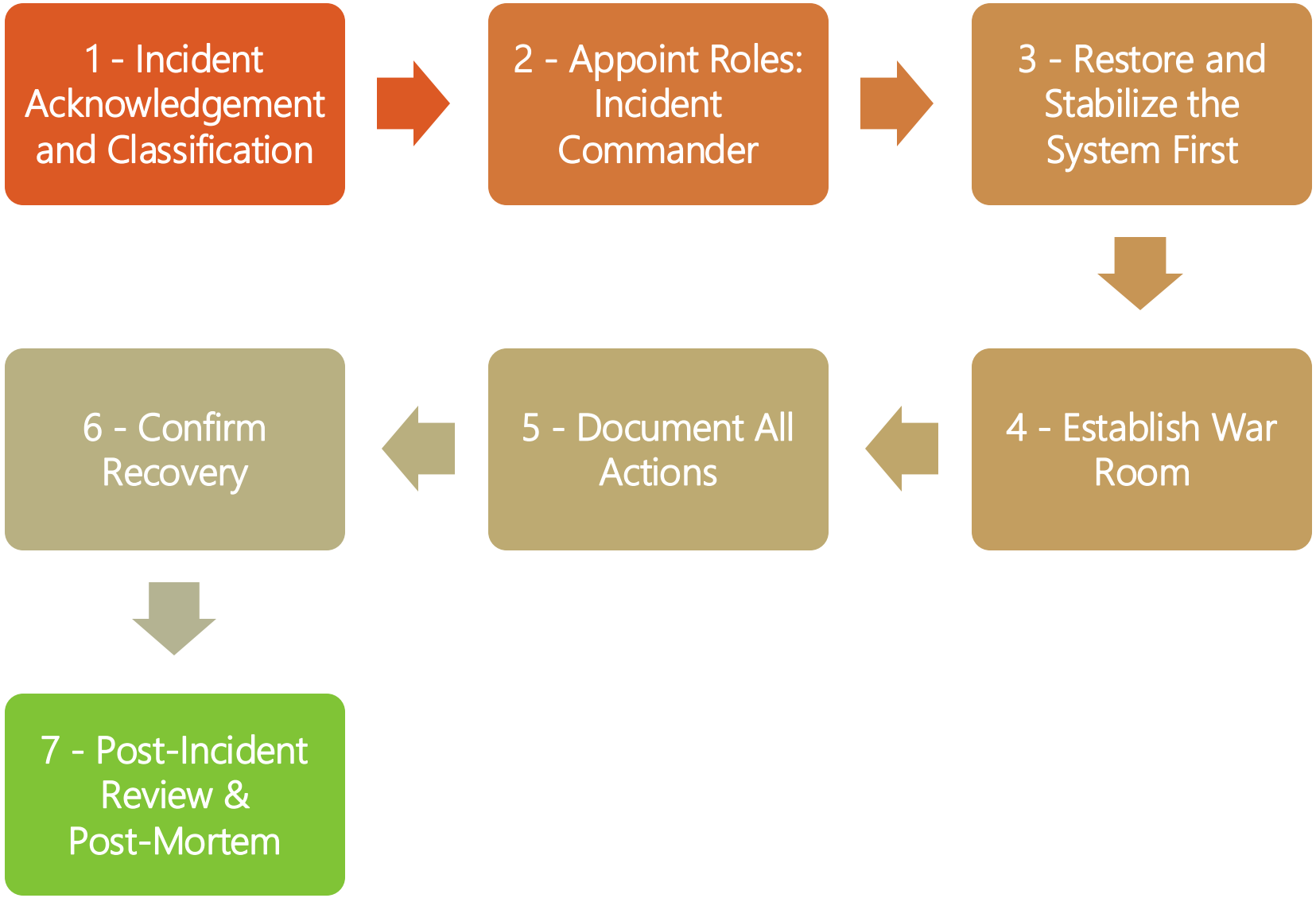
Skills Covered:

- Task 1: Deliver two new endpoints using azure devops pipeline
- Task 2: Troubleshoot Pipeline issues
- Task 3: Troubleshoot API
- Task 4: Automation using workbooks

Instructions: AZ_SRE_lab_05.md

EMERGENCY RESPONSE

EMERGENCY RESPONSE - WHAT TO DO



EMERGENCY RESPONSE - DOCUMENTATION

- Centralize Storage: Use a shared tool like Confluence, Wiki, ServiceNow, or custom dashboards.
- Standardize the Format: reports are consistent and comparable over time.
- Tag and Categorize Incidents: “Deployment Failure”, “Infrastructure Outage”, “Capacity Issue”.
- Review Regularly: Hold monthly or quarterly reliability reviews using outage history as input.
- Link to RCA Documents: Each outage entry should reference the full Root Cause Analysis (RCA) and Post-Incident Review (PIR) document.

POP QUIZ:

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

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

POP QUIZ:

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

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

POP QUIZ:

Your team is troubleshooting a production performance issue after a recent deployment.
Which of the following practices align with troubleshooting best practices in a production environment? (Choose two.)

A. Making untested configuration changes directly in production to fix the issue quickly
B. Using available logs, metrics, and traces to gather evidence before attempting a fix
C. Reproducing the issue in a test or staging environment when possible
D. Assuming the root cause based on past incidents without analyzing current data

POP QUIZ:

Your team is troubleshooting a production performance issue after a recent deployment.
Which of the following practices align with troubleshooting best practices in a production environment? (Choose two.)

A. Making untested configuration changes directly in production to fix the issue quickly
B. Using available logs, metrics, and traces to gather evidence before attempting a fix
C. Reproducing the issue in a test or staging environment when possible
D. Assuming the root cause based on past incidents without analyzing current data

INCIDENT MANAGEMENT

INCIDENT MANAGEMENT - UNMANAGED INCIDENTS

- Lack of clear ownership and leadership
- Uncoordinated and chaotic response efforts
- Duplicate or conflicting actions by teams
- Poor communication with stakeholders
- Longer recovery time and increased business impact

INCIDENT MANAGEMENT – RISKS

No Incident Commander (IC) -> **Increased MTTR**

Conflicting Fix Attempts -> **Business Impact Grows**

Lack of Communication -> **Higher Error Risk**

No Live Documentation -> **Incomplete Incident Records**

Delayed Recovery -> **Erodes Customer Trust**

INCIDENT MANAGEMENT - FREELANCING

- Unauthorized individuals making uncoordinated changes
- Causes conflicting fixes and side effects
- Violates incident command structure
- Leads to longer outages and failed recoveries
- Must enforce single-threaded decision-making

INCIDENT MANAGEMENT - LIVE INCIDENT DOC

What is a Live Incident State Document?

- A collaborative, continuously updated
- Allows everyone (technical teams, leadership, communications leads) to stay aligned on current incident status.
- Tracks timelines, actions taken, status updates, hypotheses, impact summaries, and decisions made

INCIDENT MANAGEMENT - LIVE HANDOFF

1. Ensure seamless shift changes during long incidents
2. Conduct verbal or written briefings before handoff
3. Review current system state, timeline, and pending actions
4. Clearly assign ownership and roles
5. Minimize knowledge loss and duplicated effort

INCIDENT MANAGEMENT - MANAGED INCIDENT

"Incidents happen. What defines an SRE team is how well they manage them."

Characteristic	Description
Clear Leadership	An **Incident Commander (IC)** leads the response with full decision-making authority.
Defined Roles	IC, Technical Leads, Communications Lead, and Note-taker are all assigned and known.
Centralized Communication	One war room, one source of truth, one live document.
Real-Time Documentation	All actions, observations, and decisions are logged in the **Live Incident Document (LID)**.
Regular Status Updates	Stakeholders (internal and external) receive timely, accurate updates.
Controlled Changes	No freelancing. All technical changes are approved by the IC.
Clear Handoffs (if needed)	Long incidents include structured, live handoffs between shifts or teams.

POP QUIZ:

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

A. Personal email threads
B. AWS CloudWatch annotations only
C. Incident management tools like Jira or Confluence runbooks
D. Amazon S3 bucket logs

POP QUIZ:

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

A. Personal email threads
B. AWS CloudWatch annotations only
C. Incident management tools like Jira or Confluence runbooks
D. Amazon S3 bucket logs

TRACKING OUTAGES

TRACKING OUTAGES - TAGGING & ANALYSIS

- Tag incidents with relevant metadata
- Enable trend and pattern analysis
- Identify recurring failure themes
- Support root cause and error budget reporting
- Drive targeted reliability improvements

Tag Category	Examples
Service or Component	API Gateway, Payment Service, Database
Root Cause Type	Configuration Error, Capacity Exhaustion, Code Bug, External Provider Failure
Impact Level	SEV-1 (Critical), SEV-2 (High), SEV-3 (Medium)
Detection Method	Monitoring Alert, Customer Report, Synthetic Test
Mitigation Type	Rollback, Failover, Hotfix
SLO Impact	Yes / No

POP QUIZ:

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

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

POP QUIZ:

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

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

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