AWS Well-Architected

DevOps Guidance

DevOps Guidance: AWS Well-Architected

Table of Contents

Abstract and introduction ..... **i**
Introduction 1
Definitions 1
Using the DevOps Guidance **4**
Join the DevOps discussion 5
The DevOps Sagas **6**
Organizational adoption 7
Leader sponsorship 8
Supportive team dynamics 15
Team interfaces 24
Balanced cognitive load 32
Adaptive work environment 38
Personal and professional development 42
Development lifecycle 49
Local development 49
Software component management 59
Everything as code 69
Code review 78
Cryptographic signing 86
Continuous integration 92
Continuous delivery 99
Advanced deployment strategies 107
Quality assurance 115
Test environment management 116
Functional testing 123
Non-functional testing 130
Security testing 143
Data testing 153
Automated governance 158
Secure access and delegation 159
Data lifecycle management 167
Dynamic environment provisioning 176
Automated compliance and guardrails 186
Continuous auditing 197

Observability 203
Strategic instrumentation 203
Data ingestion and processing 211
Continuous monitoring 217
Contributors 228
Further reading 229
Document history 230
Notices 231
AWS Glossary 232

DevOps Guidance

Publication date: **September 20, 2023** (*Document history*)

Drawing from Amazon's own transformative journey and the expertise gained by AWS in managing cloud services at global scale, the AWS Well-Architected Framework DevOps Guidance offers a structured approach that organizations of all sizes can follow to cultivate a high-velocity, security-focused culture capable of delivering substantial business value using modern technologies and DevOps best practices.

Introduction

In the early 2000s, Amazon went through its own DevOps transformation which led to an online bookstore forming the Amazon Web Services (AWS) cloud computing division. Today, AWS provides a wide range of products and services for global customers that are powered by that same innovative DevOps approach. Due to the positive effects of this transformation, AWS recognizes the significance of DevOps and has been at the forefront of its adoption and implementation.

Amazon's own journey, along with the collective experience gained from assisting customers as they modernize and migrate to the cloud, provided insight into the capabilities which we believe make DevOps adoption successful. With these learnings, we created a collection of modern capabilities that together form a comprehensive approach to designing, developing, securing, and efficiently operating software at cloud scale.

The content and recommendations outlined within the Well-Architected DevOps Guidance are based on our expertise operating and managing services for global customers and serves as a customizable reference to fit your organization's requirements. Each organization has unique requirements which may deviate from the examples provided here.

There is no one-size-fits-all approach to adopting DevOps. Tailor these recommendations to suit your individual environment, quality, and security needs.

Definitions

- **DevOps Sagas** are core domains within the software delivery process that collectively form AWS DevOps best practices. Together, they form a collection of modern capabilities representing a comprehensive approach to designing, developing, securing, and efficiently operating software

at cloud scale. You can use the DevOps Sagas as a common definition of what DevOps means to align on a shared understanding within your organization.

- **Capabilities** are repeatable mechanisms that an organization can continuously improve to sustainably practice DevOps and achieve measurable outcomes. Each capability includes best practice indicators, metrics, and anti-patterns that can be tracked and refined over time. Capabilities encompass a wide range of topics related to the associated DevOps Saga. They are not presented in any specific order.
- **Indicators** are a collective term covering best practices spanning people, process, and technology. They can be used as qualitative measurements, such as a checklist, to objectively assess and continuously track your organization's ability to perform a capability. Each indicator contains a prescriptive improvement plan that contains high-level guidance for performing it. Indicators can fall into one of three categories, which represent importance:
  - **Foundational:** Requirements to achieve minimum viability of the capability when practicing DevOps. Indicators in this category are considered essential for organizations to meet to achieve basic proficiency of the capability.
  - **Recommended:** Recommended best practices and advanced techniques that can improve the performance of the capability. Indicators in this category offer significant benefits, such as improved security, developer experience, speed, reliability, or the ability to optimize implementation in cloud-based environments.
  - **Optional:** Minor adjustments and improvements that further enhance the ability to implement the capability or apply only to specific use cases. Indicators in this category might not be applicable to all organizations or workloads, but can provide additional benefits to some.
- **Anti-patterns** are practices that might seem beneficial initially, but have the potential to lead to unexpected or negative outcomes. They are indicators that we have seen slow down or halt our customer's progress when adopting DevOps within their organizations. Identifying and addressing anti-patterns can help you to avoid pitfalls and optimize the chance for successful DevOps adoption.
- **Metrics** are quantifiable measures of an organization's ability to perform the associated capability. Think of indicators as the *how* and metrics as the *how well* when using this guidance to measure your organization's ability to implement a capability. There is no one-size-fits-all approach to selecting which metrics to track. We provide metrics for each capability as a starting point. Each organization needs to determine which metrics are important to their business. Consider using additional frameworks, such as DevOps Research and Assessment (DORA) and SPACE, to further customize the metrics we provide so that they align to your organizational goals.

For more Well-Architected terminology, see Definitions in the *AWS Well-Architected Framework* whitepaper.

Using the DevOps Guidance

To effectively use the DevOps Guidance in conjunction with the AWS Well-Architected Framework, follow these steps:

1. Familiarize yourself with this document, the broader AWS Well-Architected Framework, and the individual Well-Architected Framework pillar whitepapers.
2. Gather your organization's design documentation, operational procedures, and monitoring history related to DevOps practices (where available).
3. Compare your organization's DevOps capabilities to the best practices, anti-patterns, and metrics described in this guidance.
4. For each best practice, record whether it has been followed and prioritize evaluating those that are required.
5. Use the provided suggestions to address the areas where your DevOps capabilities are not well-architected.
6. If you require additional expert guidance, contact your AWS account team to engage a DevOps specialist Solutions Architect (SA).

After reviewing your DevOps practices against the DevOps Guidance, you will have a list of best practices that highlight areas where your organization and systems are well-architected, and where there may be room for improvement.

- **For the well-architected architectural components:** Share your knowledge among your teams to amplify them across your organization.
- **For the best practices that your organization does not follow yet:** Treat them as technical debt and potential risks to your adoption of DevOps practices. Follow your internal risk management process to continuously monitor and improve these risks.
- **For areas that require further in-depth analysis or assistance with remediation:** Contact AWS Professional Services or consult with AWS Partners.

For more details, see The review process in the *AWS Well-Architected Framework whitepaper*.

Join the DevOps discussion

Engage with the AWS DevOps community for tutorials, best practices, and insight into how others are practicing DevOps. Join the discussion over at AWS re:Post if you have additional questions about the Well-Architected Framework DevOps Guidance or implementing DevOps on AWS. AWS DevOps experts regularly monitor the re:Post DevOps topic for discussion and questions that could be answered to assist our customers and partners following DevOps practices with AWS.

The DevOps Sagas

To remove ambiguity about what DevOps means to us in this guidance, we define it through the AWS DevOps Sagas—a collection of modern capabilities that together form a comprehensive approach to designing, developing, securing, and efficiently operating software at cloud scale. We chose the term *saga* to reflect that sustainably practicing DevOps takes time and involves an ongoing, interconnected set of capabilities. Each DevOps saga includes prescriptive capabilities that provide indicators of success, metrics to measure, and common anti-patterns to avoid.

This section outlines each of the DevOps Sagas, featuring definitions, capabilities, indicators of success, metrics, and anti-patterns. The DevOps Sagas have been carefully crafted to focus on DevOps adoption and sustainable usage over time. The capabilities are solely focused on operating in a DevOps environment. When assessing workloads that will operate in a DevOps environment in the AWS Cloud, we recommend that you use the DevOps Guidance in conjunction with the AWS Well-Architected Framework whitepaper.

The DevOps Sagas are:

- **Organizational adoption:** Inspires the formation of a customer-centric, adaptive culture focused on optimizing people-driven processes, personal and professional development, and improving developer experience to set the foundation for successful DevOps adoption.
- **Development lifecycle:** Aims to enhance the organization's capacity to develop, review, and deploy workloads swiftly and securely. It uses feedback loops, consistent deployment methods, and the *everything as code* approach to attain efficiency in deployment.
- **Quality assurance:** Advocates for a proactive, test-first methodology integrated into the development process to help ensure that applications are well-architected by design, secure, cost-efficient, sustainable, and delivered with increased agility through automation.
- **Automated governance:** Facilitates directive, detective, preventive, and responsive measures at all stages of the development process. It emphasizes risk management, business process adherence, and application and infrastructure compliance at scale using automated processes, policies, and guardrails.
- **Observability:** Presents an approach to incorporating observability within environment and workloads, allowing teams to detect and address issues, improve performance, reduce costs, and help ensure alignment with business objectives and customer needs.

Sagas

- Organizational adoption
- Development lifecycle
- Quality assurance
- Automated governance
- Observability

Organizational adoption

The organizational adoption saga provides a prescriptive approach to creating a more customer-focused culture that can quickly respond to changing business needs. It emphasizes that DevOps is first-and-foremost about people and culture, highlighting the significance of optimizing people-driven processes as the foundation for successful DevOps adoption. Once the people-driven processes are addressed, organizations can effectively use tools to automate, refine, and optimize those processes to achieve their business objectives.

While some of the capabilities contribute to building a healthy culture in general, such as supportive team dynamics, personal and professional development, and adaptive work environment, their significance in the context of DevOps cannot be overstated. These capabilities directly impact the successful adoption of other DevOps Sagas by fostering the culture necessary to support the rapid adaptation of new technologies, rules, and ways of working. By investing

in these capabilities, organizations can achieve long-term success with DevOps adoption and maintain a sustainable DevOps environment over time.

Capabilities

- Leader sponsorship
- Supportive team dynamics
- Team interfaces
- Balanced cognitive load
- Adaptive work environment
- Personal and professional development

Leader sponsorship

Obtaining leader sponsorship of DevOps adoption initiatives helps verify that the organization's leadership is committed to and actively supports the adoption of DevOps practices. Effective leader sponsorship involves setting a clear vision and strategy for DevOps adoption, communicating expectations and goals to the entire organization, and allocating resources and budget for the necessary changes. Leaders who support DevOps adoption should model the desired behaviors and lead by example. By actively participating in and driving the DevOps adoption process, leaders can help remove barriers, facilitate change, and motivate team members to embrace new ways of working, ultimately accelerating the organization's transition to a successful DevOps environment.

Topics

- Indicators for leadership sponsorship
- Anti-patterns for leader sponsorship
- Metrics for leader sponsorship

Indicators for leadership sponsorship

Promote organizational commitment to DevOps adoption with clear strategy, communication, and resources.

Indicators

- [OA.LS.1] Appoint a decision-making leader to own DevOps adoption
- [OA.LS.2] Align DevOps adoption with business objectives

- [OA.LS.3] Drive continued improvement through business reviews
- [OA.LS.4] Open dialogue between leadership and teams
- [OA.LS.5] Assemble a cross-functional enabling team that focuses on organizational transformation

[OA.LS.1] Appoint a decision-making leader to own DevOps adoption

Category: FOUNDATIONAL

DevOps adoption requires a dedicated leader to help facilitate continued progress, make resource decisions, and gain alignment with leaders throughout the organization. This leadership role, inspired by Amazon's single-threaded leadership concept, becomes the person within the company fully dedicated and accountable for DevOps adoption. They have no competing priorities, focusing solely on DevOps adoption and driving the initiative forward.

A single-threaded leader becomes the focal point for centralizing decision-making. They have the leeway within the organization to assess areas of improvement, and the ability to organize teams to solve problems. Appoint a leader with decision-making authority. Because DevOps adoption has a broad impact that requires change to occur throughout the entire organization, the leader must have support from executives, such as the CEO, CTO, CIO, or CISO. The ideal single-threaded leader for DevOps adoption is usually a role reporting directly to senior executives. This connection helps them drive organizational decisions, structure teams, and allocate responsibilities with the proper level of authority and direct escalation channels.

The leader should work closely with enthusiastic early adopters to build momentum and support for the initiative. Open communication channels must remain open throughout the organization to foster collaboration and receive support. As progress is made, the leader regularly updates other teams and leaders of DevOps adoption initiatives and the impact DevOps is having on the business.

Related information:

- AWS Well-Architected Cost Optimization Pillar: COST01-BP02 Establish a partnership between finance and technology
- AWS Cloud Adoption Framework: People Perspective - Transformational leadership
- A Conversation with Werner Vogels: Learning from the Amazon technology platform
- Two-Pizza Teams Are Just the Start, Part 2: Accountability and Empowerment Are Key to High-Performing Agile Organizations

[OA.LS.2] Align DevOps adoption with business objectives

Category: FOUNDATIONAL

DevOps adoption should not be an isolated project within the organization. It should be aligned to broader business goals, fully supported by leadership, with other teams also adopting capabilities to streamline their individual value streams. Synchronizing DevOps adoption and the overall business strategy means that the resources and effort put into adopting DevOps are also directly improving business outcomes.

Gain an understanding of your existing DevOps capabilities by conducting a comprehensive assessment of your current software development practices. You can use the AWS DevOps Sagas indicators provided in this guidance to assess your existing DevOps capabilities against best practices. This activity should result in a prioritized list of DevOps capabilities that are missing or could use improvement within your organization. These findings should be shared with leadership and individual teams across the organization. Individual teams across the organization can progress towards adopting DevOps best practices as part of the regular planning processes.

Planning processes vary from organization to organization, so we will provide an example using Amazon's annual planning process. To kick start the yearly planning process, members of every team in the organization dedicate weeks of effort to focus on planning. The Senior Leadership team (S-Team) kicks off the process by defining business objectives. These high-level objectives are generally based on current business needs and future aspirations.

Teams build their operating plans based on the leadership-defined expectations and objectives. The first iteration of the operating plan (OP1) is a bottom-up proposal to gain alignment with other teams and approval from leadership. Operating plans should include progress towards the implementation of the DevOps capabilities from the prioritized list. Which capabilities to implement should be chosen based on alignment to S-Team goals in addition to their own individual goals. The team's operating plan should be shared with leadership and other relevant teams within the organization to promote shared knowledge and collaboration. Gaining approval from leadership helps align DevOps initiatives with the broader organizational goals. Additionally, this helps facilitate gaining the full support of leadership, including the requirements for funding, time, and resources.

Priorities, learnings, and customer needs often change over time. The second iteration of operating plan review (OP2) provides an opportunity to adapt the plan accordingly before finishing the plan. Consistently revisit the prioritized DevOps capability list at least once a year to continue progressing towards adopting DevOps best practices.

Related information:

- What We Can Learn From Amazon's Planning Process
- This is How Amazon Measures Itself
- How can we make our planning process more agile and customer-centric?
- Predicts 2023: Collaborate, Automate and Orchestrate to Optimize Costs and Value During the Economic Crisis

[OA.LS.3] Drive continued improvement through business reviews

Category: FOUNDATIONAL

While adopting DevOps, many small teams begin to form which own and operate their own value stream of the business. Teams must verify that their operations remain agile, efficient, and aligned with overarching business objectives. Leaders must be able to report on DevOps progress and outcomes. However, having many distributed teams and systems makes it more difficult for leaders to maintain full visibility across all of the value streams. It's possible to retain this visibility across a decentralized operating model by creating structured, data-driven mechanisms, such as conducting regular business review meetings and tracking key performance indicators (KPIs). The mechanisms help leaders pinpoint areas of inefficiencies, uncover opportunities to innovate, and create a culture of continual feedback, measurement, and refinement.

Begin by developing a set of KPIs that align with desired business outcomes and simultaneously demonstrates the impact of DevOps adoption on achieving them. KPIs are quantifiable metrics that are used to measure the performance of an organization or project as it progresses towards a goal. Tracking KPIs verifies that the goal is moving in the right direction and achieving desired outcomes. KPIs should be continually improved and refined over time to keep them aligned with business objectives as the organization adopts DevOps and business needs change.

Schedule frequent business review meetings to review KPIs, bringing together both technical and business stakeholders on a regular cadence. Each team should continually capture both technical and business related KPIs and make them presentable for regular business reviews. Regularly reviewing the KPIs informs leaders of the health and direction of the team's value stream. Fluctuations in the KPIs reflect the outcome of team efforts and can be a predictor of future outcomes.

Within Amazon, teams and leaders meet regularly during weekly business reviews (WBRs) to assess the validity and quality of KPIs against organizational goals. For a data-driven, systematic

approach to this process, we follow the DMAIC—Define, Measure, Analyze, Improve, and Control—improvement cycle. We recommend you adopt a similar approach to sustainable business reviews.

Related information:

- AWS Well-Architected Performance Pillar: PERF07-BP03 Establish key performance indicators (KPIs) to measure workload performance
- AWS Well-Architected Cost Optimization Pillar: COST02-BP02 Implement goals and targets
- What is the difference between SLA and KPI?
- The Business Value of Migration to Amazon Web Services
- Business Value of Cloud
- Blog: Business Value is IT's Primary Measure of Progress
- Blog: The Importance of Key Performance Indicators (KPIs) for Large-Scale Cloud Migrations

[OA.LS.4] Open dialogue between leadership and teams

Category: FOUNDATIONAL

Communication in a DevOps environment is more than an exchange of information. It's about building trust, collaboration, and gaining alignment across the organization. Clear communication channels can bridge the gap between strategy and implementation.

Establish open communication channels between leaders and team members. Implement a system that gathers anonymous feedback directly from team members. Verify that this method is equally inclusive and accessible to everyone. Leaders should regularly share updates, insights, and learning back to teams to create a culture of collaboration and trust. An Amazon example is Amazon Connections, a mechanism that captures real-time feedback and data from employees about their experiences. This provides a model for organizations to understand team perspectives.

The gathered feedback should drive decision-making at all levels of leadership to identify areas for improvement, address employee concerns, and promote a culture of open communication. Leaders must actively engage with this feedback, sharing updates and insights with teams. This action not only builds trust, but also aligns everyone with the organization's DevOps adoption progress.

Related information:

- Business Value is IT's Primary Measure of Progress

[OA.LS.5] Assemble a cross-functional enabling team that focuses on organizational transformation

Category: RECOMMENDED

To spread knowledge across the organizations and help individual teams adopt DevOps capabilities, create an enabling team with expertise in DevOps culture, practices, and tools. The single-threaded DevOps owner is responsible for creating this team and providing it with the autonomy, resources, training, and tools that help them effectively support and guide other teams.This centralized team should collaborate closely with other teams to identify and address barriers to adoption, share best practices, and promote a culture of nearly continuous learning and improvement.

In many organizations, this team takes the form of the *Center of Enablement*. While this centralized team is not strictly required for every organization to adopt DevOps, we recommend it due to its potential to streamline and expedite transformation. If leadership chooses not to create a Center of Enablement, they can supplement it by fostering a strong culture of collaboration, sharing, automation, and continuous improvement. With the right support and resources being provided from leadership, teams can work together to establish their own DevOps processes. Use existing tools and resources to support these efforts.

Related information:

- AWS Well-Architected Operational Excellence Pillar: Separated AEO and IEO with centralized governance and an internal service provider consulting partner
- What is a cloud center of excellence and why should your organization create one?

Anti-patterns for leader sponsorship

- **Diluted leadership focus:** When the single-threaded leader accountable for DevOps attention has other priorities and does not dedicate their full attention, the initiative might suffer. A leader handling multiple initiatives can lead to overlooked opportunities, less time for critical decision making, and overall reduced engagement. Designate a decision-making leader whose primary responsibility is the adoption of DevOps to help provide dedicated leadership to this transition.
- **Forcing DevOps adoption:** Single-threaded leadership should act as a supportive sponsor, rather than strict enforcers of DevOps adoption. Taking a hierarchical approach to DevOps adoption might cause teams to view it as imposed change and external demands. This can lead to resistance, de-motivation, and frustration among team members. Instead, start by engaging

with enthusiastic early adopters and support them in adopting DevOps. These adopters can be an advocate for DevOps adoption within their team and show progress that inspires other teams. It's also important to gather feedback from teams and provide value and improvements to the areas that frustrate them most. This promotes a bottom-up, collaborative approach to DevOps adoption that fosters nearly continuous improvement and inclusivity.

- **Short-term priority shifting**: Changing business priorities too frequently can undermine the long-term commitment required for DevOps adoption. DevOps transformation, as illustrated by the DevOps Sagas, is a marathon, not a sprint. It demands consistent focus, alignment, and support from not only single-threaded leadership, but also the whole organization. When this focus is interrupted due to short-term objectives or immediate business pressures, it might disrupt DevOps adoption. Disruptions can then lead to unjustified reduced confidence in DevOps ways of working, low team morale, and fragmented workflows. The executive team and single-threaded leadership must have a strategy that recognizes the long-term value of DevOps. They must be able to inspire other leaders in the organization to share an unwavering commitment to DevOps adoption.

Metrics for leader sponsorship

- **DevOps adoption percentage:** The percentage of DevOps capabilities that have been implemented compared to the number of desired capabilities. This metric does not indicate overall DevOps effectiveness or the performance of a specific DevOps capability. It measures the organization's total progress towards adopting DevOps capabilities that can be continuously assessed using a DevOps framework, such as the AWS DevOps Sagas. Comparing this metric to historical data or other DevOps metrics can reveal trends that indicate leadership's prioritization of DevOps initiatives and the average pace of organizational DevOps adoption. Divide the number of implemented DevOps capabilities by the number of desired DevOps capabilities. Multiple the result by 100 to get a percentage value.
- **Employee net promoter score (eNPS):** Measure employees' engagement and satisfaction within the organization, gauging their likelihood to recommend the organization as an ideal workplace. This can provide insight into the health of the organizational culture, indicating the effectiveness of leadership in creating an inclusive, positive work environment. A higher eNPS can correlate with better productivity, lower turnover, and improved team dynamics. Track using periodic, anonymized net promoter score surveys that require no more that 5–7 minutes to complete. Subtract the percentage of detractors (those who score 0–6) from promoters (those who score 9–10) to get the eNPS value. Neutral scores (7–8) can be ignored.

- **Time to fill vacancy:** The average duration from when a DevOps-related job vacancy is opened until it is filled, including the full sourcing process. This metric evaluates the organization's ability to attract and secure talent required for DevOps initiatives, providing insights into the organization's reputation and effectiveness of leadership. Extended vacancy times can affect project timelines and team morale. Creating regular feedback loops with hiring teams, improving employee net promoter score, and enhancing job descriptions can help improve this metric. Monitor the time of job posting to the acceptance of an offer by a candidate. Average these durations over a set period, such as monthly or quarterly, to determine trends.

Supportive team dynamics

Supportive team dynamics are essential to DevOps adoption as it promotes a sense of ownership, autonomy, shared accountability, and collaboration among team members. DevOps adoption requires teams to take on new responsibilities, such as cost optimization, operations, security, and availability. Historically, these new responsibilities might have been handled by teams outside of their own. Healthy, effective teams are able to incorporate new responsibilities, respond quickly to changing business needs, and maintain focus on delivering high-quality products to their customers.

Topics

- Indicators for supportive team dynamics
- Anti-patterns for supportive team dynamics
- Metrics for supportive team dynamics

Indicators for supportive team dynamics

Create a collaborative atmosphere that emphasizes ownership and shared accountability and organizes teams to serve their internal and external customers.

Indicators

- [OA.STD.1] Organize teams into distinct topology types to optimize the value stream
- [OA.STD.2] Tailor operating models to business needs and team preferences
- [OA.STD.3] Prioritize shared accountability over individual achievements
- [OA.STD.4] Structure teams around desired business outcomes
- [OA.STD.5] Establish team norms that enhance work performance

- [OA.STD.6] Provide teams ownership of the entire value stream for their product
- [OA.STD.7] Amplify the scale and impact of centralized functions
- [OA.STD.8] Promote cognitive diversity within teams

[OA.STD.1] Organize teams into distinct topology types to optimize the value stream

Category: FOUNDATIONAL

To optimize the value stream and achieve desired business outcomes, embrace the four team topologies model, as outlined in Team Topologies by Matthew Skelton and Manuel Pais. Assess each team and categorize them into one of the four topologies, aligning them with the overall value stream and creating clear purpose and goals. Organizing teams according to these topologies allows organizations to manage dependencies, enhance collaboration, and facilitate effective value delivery.

The four team topologies are:

- Stream-aligned teams are responsible for delivering value to customers by focusing on specific product lines or customer segments. These teams possess cross-functional expertise that enables them to build, test, and deploy software independently, while minimizing dependencies and handoffs with other teams. They are the primary teams within the organization, normally representing 60–80% of the total teams within an organization.
- Platform teams create and maintain shared infrastructure, tools, and services that support multiple stream-aligned teams across the organization. They produce reusable components, improve efficiency, reduce duplication of work, and overall reduce the amount of individual team effort. As these teams support many teams within the organization, they make up a smaller portion of the organization, usually between 10-20%.
- Teams support other teams by providing just-in-time skills, knowledge, and expertise. They help other teams overcome technical challenges, adopt best practices, and improve their capabilities. All assistance provided by enabling teams is meant to be temporary, as they strive to make other teams self-sufficient through facilitation and mentoring. The percentage of enabling teams is fewer than platform and stream-aligned, often ranging between 5-15% of the overall organization.
- Complicated subsystem teams are teams responsible for specialized subsystems within a larger system that require complex, deep domain knowledge and expertise. These subsystems are typically part of the core business logic or functionality of a single product or application. Their primary consumers are internal components within that system. Distinguishing between

platform teams and complicated subsystem teams may not always be clear-cut, and a team could have characteristics of both types. When a team is providing a foundational service to multiple teams, they are usually considered platform teams. If they support a single product or application, it is generally considered a complicated subsystem team. Typically, there are fewer complicated subsystem teams than other team types, making up 0–10% of the distribution.

Related information:

- Team Topologies

[OA.STD.2] Tailor operating models to business needs and team preferences

Category: FOUNDATIONAL

Adopt operating models that align with the needs of the business goals, while considering the capabilities and preferences of individual teams. The AWS Well-Architected Framework Operational Excellence Pillar provides a detailed 2 by 2 representations of operating model implementations that can be reviewed to gain insights into potential combinations. Selecting the right operating model involves evaluating the organization's requirements, such as decision-making processes, communication channels, and resource allocation. Keep in mind that multiple operating models can be used concurrently, catering to different use cases, levels of organizational maturity, and individual team and product needs.

Not all operating models support a DevOps culture, and DevOps might not be suitable for every system. In some cases, especially in large and diverse organizations, it might be necessary to support stringent compliance requirements. Additionally, mass migration to a new way of working for all teams may not be feasible due to time, complexity of the system, or skill requirements. For these use cases, a fully separated operating model or introducing an Internal MSP and Consulting Partner might be needed for those systems that must stay *as is* with more traditional ways of working.

When choosing a Well-Architected operating model for systems that can support DevOps, first determine if centralized or decentralized control of governance is necessary. A centralized governance model grants platform teams within an organization the ability to control *how* and *what* other teams are able to deploy, at the cost of restricting those teams' ability to innovate and make changes quickly. Conversely, a fully decentralized model offers teams more flexibility and autonomy, requiring less intensive collaboration between teams through reliance on guardrails and automated governance over strict control.

Related information:

- AWS Well-Architected Operational Excellence Pillar: Operating model 2 by 2 representations
- Building your Cloud Operating Model: Organize for Success

[OA.STD.3] Prioritize shared accountability over individual achievements

Category: FOUNDATIONAL

Encourage a culture of teamwork and shared accountability by establishing common goals and fostering collaboration and open communication. Create a sense of shared ownership and responsibility for achieving team success, encouraging members to support each other and provide constructive feedback. Regularly evaluate progress towards goals and celebrate successes together as a team. Prioritizing team success over individual accomplishments promotes a cohesive and high-performing team environment that is essential for successful DevOps adoption.

[OA.STD.4] Structure teams around desired business outcomes

Category: FOUNDATIONAL

To maximize value and effectiveness in product delivery, intentionally design team structures that reflect the desired architecture and interactions of the systems being built. Clearly define roles, responsibilities, and ownership for each team and align with the expected business outcomes. This approach increases the chances of building and supporting effective products optimized for full coverage of the full value stream.

Conway's Law, introduced by Melvin Conway in the paper *How Do Committees Invent?*, posits that the structure of an organization influences the design of the systems it builds. Organizations can use this concept to build more effective team structures by employing the Inverse Conway Maneuver, also known as *Reverse Conway's Law*, as described by Jonny LeRoy and Matt Simons. By designing teams and their communication structures to reflect the intended architecture and interactions of the system being built, organizations can achieve increased efficiency and more effective collaboration between teams, ultimately enhancing the overall product delivery process.

Related information:

- How Do Committees Invent?
- Dealing with creaky legacy platforms
- Demystifying Conway's Law

- Inverse Conway Maneuver

[OA.STD.5] Establish team norms that enhance work performance

Category: FOUNDATIONAL

Optimize work performance by establishing norms that define clear roles, schedules, and processes for agile ceremonies. Agree on regular meeting schedules, such as daily stand-ups, sprint planning, backlog refinement, and sprint retrospectives if you are following Scrum. Define roles for each team member during ceremonies, clarifying responsibilities and purpose in the ceremony. Conduct regular process reviews to identify areas for improvement and refine the ceremony structure as needed. Encourage active participation and engagement in the ceremonies.

When establishing team norms, consider the stages of group development as described in the paper Developmental Sequence in Small Groups by Bruce Tuckman, which describes the common stages of forming, storming, norming, and performing. Be mindful of these stages to provide the right support to teams, especially as they progress through the early phases of group formation.

Related information:

- What Is Scrum?
- Developmental sequence in small groups

[OA.STD.6] Provide teams ownership of the entire value stream for their product

Category: FOUNDATIONAL

Establish teams that are able to own their respective value streams and products. These teams follow a *you build it, you run it* approach, as coined by Werner Vogels in 2006. The team responsible for building a system should also be responsible for running, maintaining, and overall owning it. At Amazon, we call these small, autonomous teams with a single-threaded focus two-pizza teams. This approach minimizes handoffs and makes teams both the creators and custodians of their products.

Value stream ownership does not mean preventing teams from working together. These teams not only own the development of their product, but also take responsibility of aspects like security and quality assurance. To be successful in this model at scale, centralized functions, such as centralized security teams, must also evolve: instead of direct oversight, they should act as enablers, providing resources and expertise to these distributed teams.

The enabling functions should provide the necessary knowledge, resources, and attention required for teams to be successful. Individual teams build relationships with the centralized functions, share knowledge, and enhance processes consistently over time. This ultimately leads to improved outcomes for their products, customers, and the organization. Invest in ongoing cross-functional training to help individual team members acquire skills that will make them successful within their value streams. This training could include teaching developers to be security-minded, or teaching security resources to develop. Over time the teams should gradually become more self-reliant, collaboration between teams should improve, and deployment frequency should increase.

Related information:

- AWS Well-Architected Security Pillar: SEC11-BP01 Train for application security
- AWS Well-Architected Security Pillar: SEC11-BP08 Build a program that embeds security ownership in workload teams
- Enterprise DevOps: Why You Should Run What You Build
- Amazon's approach to security during development: Ownership
- Amazon's approach to security during development: Security Review Process
- Powering Innovation and Speed with Amazon's Two-Pizza Teams
- The Amazon Software Development Process: DevOps Teams

[OA.STD.7] Amplify the scale and impact of centralized functions

Category: FOUNDATIONAL

As decentralized teams become responsible for their respective value streams and products, including responsibilities like security and quality assurance, centralized functions can often become bottlenecks. These bottlenecks can delay releases and cause inefficiencies in the development lifecycle, which can limit the adoption of DevOps best practices.

We recommend adopting a Guardian Model within your organization to scale centralized functions. This involves embedding specialized champions or *Guardians* within individual teams to enhance and scale the capabilities of centralized functions, such as security, quality, and audit. Embedding guardians directly into teams helps make specialized knowledge always available, reducing wait times and facilitating real-time, context-aware decision-making. This approach not only accelerates delivery, but also continually meets quality, security, and compliance standards.

To implement this model, begin by defining the strategy for the initiative. Recognize the inefficiencies and gaps within teams that these guardians can rectify, and identify which centralized

function would benefit most from on-the-ground, embedded expertise. Security, quality assurance, and audit functions are great examples of centralized functions that must scale when adopting DevOps best practices. Leadership support is required so that they can allocate necessary resources, make policy changes, and inspire an organizational culture that genuinely values the guardian role.

When selecting and training guardians, pinpoint passionate team members who volunteer to undergo specialized training to become focal points for their respective domains. This includes proactive responsibilities, such as threat modeling or test planning, and reactive responsibilities, like defect resolution or compliance checks. These responsibilities should be clearly defined to avoid ambiguity, confusion, and conflict. Continue to gather feedback from guardians and their teams, using the insights to refine and iterate on the model.

The guardian role is an important factor for the success of this model. Encourage adoption of the role by providing specialized training opportunities, avenues for influencing best practices, and clear paths for career evolution. These incentives keep guardians motivated, engaged, and eager to drive excellence within their respective teams.

Related information:

- AWS Well-Architected Security Pillar: SEC11-BP08 Build a program that embeds security ownership in workload teams
- Scaling security and compliance
- AWS Security Guardians
- Amazon's approach to security during development: Security Review Process

[OA.STD.8] Promote cognitive diversity within teams

Category: RECOMMENDED

Optimized DevOps teams should remain small and agile in how they deliver their products. This approach requires team members to have a wide range of cross-functional skills, from software development and testing to operations and security. Having a diverse mix of skills, experiences, and backgrounds within the team helps them effectively innovate to solve complex problems and better mimic the personas and culture of their users.

Promote cognitive diversity within small teams by including members with varied ethnic, cultural, regional, gender, age, and other backgrounds. Avoid hiring bias and promote diversity

and inclusion when forming teams. Implement inclusive hiring practices, such as blind resume screening and diverse interview panels. Encourage cross-functional collaboration and knowledge sharing so that teams can frequently gain the perspectives of others.

Aim to maintain strong cognitive diversity by regularly assessing the diversity of the team and identifying any potential gaps. This can be done through team surveys, diversity and inclusion training, and ongoing monitoring and evaluation. Additionally, invest in security training and awareness programs, equipping your team members with the knowledge and skills to identify and mitigate security risks. Doing so enhances the overall security posture and culture of the organization.

By creating teams that embrace cognitive diversity, organizations can improve innovation, creativity, and problem solving, leading to better outcomes for the organization and its customers.

Related information:

- AWS Well-Architected Security Pillar: SEC11-BP01 Train for application security

Anti-patterns for supportive team dynamics

- **Project-based teams**: If teams are structured around short-term projects rather than being aligned with products or services, it can prevent them from taking ownership of the value stream. Project-based teams tend to focus on completing tasks and moving on to the next assignment, rather than fostering long-term ownership, accountability, and continuous improvement. This can impact delivery speed, quality, and maintainability of resulting products. Shift the focus towards product-aligned teams that have ownership of their entire value stream. This encourages teams to take responsibility for the full lifecycle of the product, from idea to delivery and ongoing maintenance.
- **Restrictive tool support**: If supporting teams support a limited set of tools, frameworks, or programming languages, it can discourage individual teams from adopting new tools or choosing the best tool for specific use cases. Requiring teams to use a standardized tool, which may be the wrong tool for the specific use case, can lead to sub-optimal solutions and decreased developer productivity. To overcome this, supportive teams should take a more flexible approach by creating solutions that are inclusive of using multiple tools, frameworks, and programming languages. This encourages teams to choose tools for their specific needs, creating an adaptable development environment that fosters a culture of experimentation and learning.
- **Rigid hierarchical structures**: Traditional rigid hierarchical organizational structures can hinder the flow of information and introduce unnecessary dependencies between teams.

When decisions are always made by executive leadership and individual teams lack autonomy, it can suppress innovation, delay time to market, and deter accountability. These structures can also discourage teams from collaborating freely, due to potential political dynamics or fear of overstepping boundaries. To address this, create an environment where teams have the autonomy to make decisions that align with their goals and the organization's business objectives. Encourage open channels of communication across all levels and departments.

Metrics for supportive team dynamics

- **Employee net promoter score (eNPS):** Measure employees' engagement and satisfaction within the organization, gauging their likelihood to recommend the organization as an ideal workplace. This can provide insight into the overall health of the organizational culture and indicates the effectiveness of leadership in creating an inclusive and positive work environment. A higher eNPS can correlate with better productivity, lower turnover, and improved team dynamics. Track using periodic, anonymized net promoter score surveys that require no more that 5–7 minutes to complete. Subtract the percentage of detractors (those who score 0–6) from promoters (those who score 9–10) to get the eNPS value. Neutral scores (7–8) can be ignored.
- **Cross-functional dependency tracking:** The number of dependencies a team has on other teams is measured by the frequency of interactions between teams. Dependencies can slow down work, indicate siloed teams, or point towards inefficient processes. The frequency of cross-team interactions can help gauge how siloed teams are and how effectively they are collaborating. This metric should improve as teams become more autonomous and fully responsible for their value stream. Use time tracking tools, calendar analytics, and dependencies between work items in project management tools to track and categorize dependencies. On a regular basis, such as monthly or at the end of a sprint, calculate the average number of dependencies and the frequency of cross-team meetings.
- **Team health:** Hold periodic surveys that gauge team members' feelings and perceptions about collaboration, support, and team dynamics. These surveys create feedback loops that can highlight strengths, weaknesses, and areas for improvement. If following Scrum, this survey might take the form of a sprint retrospective. If not following Scrum, or for more granular feedback, administer surveys on a regular basis, such as monthly or quarterly. Measure the percentage of positive responses over time and track trends as they emerge.
- **Team turnover:** The frequency that team members depart from a specific team. High turnover can indicate issues with team dynamics, satisfaction, or other underlying problems affecting team cohesion. Identify these issues early to maintain project consistency, team morale, and productivity. Monitor the reasons for departures using exit interviews, and progress towards

resolving points of friction within the team. On a regular basis, such as monthly or quarterly, calculate the turnover rate by taking the number of employees who left during that period divided by the average number of employees during the same period, then multiply by 100 for the percentage.

Team interfaces

Team interfaces are the input and output mechanisms that direct the flow of work between teams in a DevOps environment. These interfaces work to share information and resources aacross the organization, facilitate collaboration, and help teams to align their goals and priorities. Effective team interfaces streamline processes, reduce bottlenecks, and improve overall productivity within and across teams.

Topics

- Indicators for team interfaces
- Anti-patterns for team interfaces
- Metrics for team interfaces

Indicators for team interfaces

Implement mechanisms to enhance productivity within and across teams, providing effective communication channels to guide the flow of work.

Indicators

- [OA.TI.1] Communicate work flow and goals between teams and stakeholders
- [OA.TI.2] Streamline intra-team communication using tools and processes
- [OA.TI.3] Establish mechanisms for teams to gather and manage customer feedback
- [OA.TI.4] Refine error tracking and resolution
- [OA.TI.5] Design adaptive approval workflows without compromising safety
- [OA.TI.6] Prioritize customer needs to deliver optimal business outcomes
- [OA.TI.7] Maintain a unified knowledge source for teams
- [OA.TI.8] Simplify access to organizational information
- [OA.TI.9] Facilitate self-service collaboration through APIs and documentation

- [OA.TI.10] Choose interaction modes for improved efficiency and cost savings
- [OA.TI.11] Offer optional opportunities for cross-team collaboration

[OA.TI.1] Communicate work flow and goals between teams and stakeholders

Category: FOUNDATIONAL

When operating in a DevOps model, many small teams work together to deliver business outcomes to customers. Working in this way requires effective interteam communication and collaboration, as any miscommunication or delay can impact the speed and quality of delivering products.

One way to achieve this is by regularly sharing ongoing work, roadmaps, and team goals with key stakeholders and other teams. By externalizing this information, teams can improve visibility across the organization. This helps teams understand how their work impacts others and the overall business goals.

Teams can use work tracking tools that promote a more agile, adaptive approach, such as Scrum or Kanban boards, and dashboards to make their work, priorities, and key metrics visible to others in the organization. Make these tools easily accessible, either through physical displays or digital platforms, to promote alignment with business objectives. Regularly review the flow of work to identify bottlenecks, areas for improvement, and opportunities to optimize the process.

[OA.TI.2] Streamline intra-team communication using tools and processes

Category: FOUNDATIONAL

Equip teams with tools to automate and manage their workflows, priorities, and decision-making processes. Implement team collaboration, document sharing, task creation, and progress monitoring tools. Establish team norms and practices, such as lexicons, story pointing, and defining *done*, to streamline intra-team communication. Use reporting tools, playbooks, and retrospective sessions to improve processes and team norms.

Related information:

- Team Collaboration with Amazon CodeCatalyst

[OA.TI.3] Establish mechanisms for teams to gather and manage customer feedback

Category: FOUNDATIONAL

Establish feedback channels that help teams gather and manage input from both internal and external customers of their products. Related processes should be created for teams to track, prioritize, and act on the feedback received for their respective value stream. Integrate the feedback with collaboration tools and existing workflows so that inputs, decisions, and outcomes are documented and prioritized with the rest of the value stream work. Embed the feedback into your team norms. Feedback data should continually be analyzed to identify trends, prioritize areas for improvement, and communicate progress to stakeholders.

[OA.TI.4] Refine error tracking and resolution

Category: FOUNDATIONAL

Establish mechanisms for continuous improvement in error correction, tracking, and resolution. This includes developing a culture of learning from mistakes, sharing knowledge, and using data-driven insights to drive improvements. Implement tools and processes to facilitate effective error tracking and resolution, such as issue tracking systems, monitoring, and alerting solutions. Regularly review and analyze error data to identify trends, prioritize issues, and take corrective action. Encourage collaboration and knowledge sharing among teams to improve overall error management and resolution capabilities.

Related information:

- Correction of Error (COE)
- Amazon's approach to high-availability deployment: How we learn from deployment failures

[OA.TI.5] Design adaptive approval workflows without compromising safety

Category: FOUNDATIONAL

Establish approval processes and guidelines that prioritize speed, safety, and agility. These processes should account for factors such as risk assessment, impact analysis, and stakeholder engagement, while also allowing for feedback and improvement. Use automation and tools to support these processes, rather than requiring complex, human-driven collaboration between teams.

Use data and APIs from version control systems, deployment pipelines, and release management tools to support automated approval processes. These tools can help streamline the approval process, reducing the risk of errors and delays while promoting agility and speed. Make all stakeholders aware of changes, and verify that they can provide input and feedback in a timely

manner. Establish key performance indicators (KPIs) and metrics to measure the time it takes to submit, review, approve, and deploy changes. Use these metrics to drive continuous improvement in the change management process.

[OA.TI.6] Prioritize customer needs to deliver optimal business outcomes

Category: RECOMMENDED

Customer-driven development is an approach that places the end user's needs and expectations at the heart of product development. Instead of starting with technical specifications or available resources, teams start by visualizing the ideal user experience. From there, iteratively work backwards to determine how to deliver on that plan.

An example of this is the Amazon *working backwards* process. At Amazon, the development process begins with a document that outlines the product's core value to customers as a Press Release and Frequently Asked Questions (PRFAQ) document. This document often contains detailed data points such as usage forecasts, adoption expectations, the value to the customer, and how we can provide that value to customers. With this approach, Amazon can continuously build products and features that resonate with user needs.

To implement this mechanism within your organization, begin the development process by writing a document that envisions the desired customer outcome. From there, work backwards to establish the technical and operational steps to achieve that outcome. Use mechanisms such as surveys and interviews to understand customer needs and gather data points.

Next, add a detailed set of meaningful FAQs to clarify product intricacies, anticipate questions customers will have, and preempt potential challenges. Before starting development, create visual mock-ups and provide use cases to offer a tangible representation for the team so they understand how users interact with the product. Draft user guides or documentation that can provide clarity on how users will interact with the system and features to expect.

Share these documents internally to gain alignment and additional perspective from other teams and leaders. The documents can also be used throughout the development lifecycle to provide developers a clear understanding of the desired customer experience, leading to fewer errors and quicker deployment cycles. Establish a continuous feedback loop that gathers customer insights and use them to inform decision-making and prioritize additional products and features.

Related information:

- Working Backwards

- The Amazon Software Development Process: Listen to Customers
- AWS re:Invent 2020: Working backwards

[OA.TI.7] Maintain a unified knowledge source for teams

Category: RECOMMENDED

Adopt collaboration and configuration tools, supported by established processes, to store documents, configurations, and other artifacts in a unified source of record. Keep documentation up-to-date to help teams work more autonomously and build trust. Implement processes for regular review of artifacts in the source of record and remove outdated content as needed.

DevOps adoption can be achieved without a unified source of record between teams. However, it's more challenging for teams to find and manage information as they transition how they work and adopt new tools. This approach can lead to inefficient or duplicated processes and communication gaps over time. Therefore, it is highly recommended to have a centralized knowledge repository in place to improve team collaboration, knowledge sharing, onboarding time, and overall development and operational efficiency. For example, create internal wiki pages for each team to document their team norms and best practices.

[OA.TI.8] Simplify access to organizational information

Category: RECOMMENDED

Provide internal users access to vital organizational information, such as details about the organization, objectives, analytics, employee data, policies, hierarchical structures, and escalation channels. One approach is to manage a centralized platform, like an intranet, where employees can swiftly locate the information they need for effective job performance.

To improve the platform's information integrity and relevance, connect it with internal systems such as Enterprise Resource Planning (ERP) software, and maintain regular updates at the source. Include instruction and training on using this platform as part of the onboarding process to equip employees with the necessary skills for information access.

Further enhance the platform with generative AI-powered internal research and search capabilities. This helps users swiftly access and interpret proprietary, complex documentation regarding compliance, regulations, or portfolio research using text summarization. Search is another method for faster information retrieval and classification, expediting access to relevant documents during review processes. Search also improves extraction of accurate answers from researched materials by querying a topic.

Related information:

- Business Value is IT's Primary Measure of Progress
- Amazon Bedrock
- Amazon Kendra
- Amazon OpenSearch Service

[OA.TI.9] Facilitate self-service collaboration through APIs and documentation

Category: RECOMMENDED

Develop clear and comprehensive service documentation for improved accessibility and navigation, including user guides, tutorials, and FAQs. Provide well-defined interfaces, such as APIs or web portals, to simplify access and usage. Regularly review and update interfaces to meet user needs. Establish ownership for documentation and services, and implement mechanisms for teams to ask for clarification, help, or provide feedback. Define metrics around the usage, availability, and quality of self-service documentation and APIs.

Providing self-service access to services through APIs simplifies integration between systems and teams, reduces the need for manual intervention, and promotes better documentation. This approach helps teams work more autonomously and accelerates the development process. This capability is highly recommended for a more efficient and streamlined DevOps environment, but it is not a foundational requirement for successful DevOps adoption. Without this capability, expect increased manual coordination and required communication between teams, which could impact overall efficiency.

Related information:

- The Amazon Software Development Process: Self-Service Tools

[OA.TI.10] Choose interaction modes for improved efficiency and cost savings

Category: RECOMMENDED

Teach teams about the different interaction modes as outlined in Team Topologies, including *X as a Service* (XaaS), facilitation, and collaboration. With knowledge of how to optimize interaction modes for specific scenarios, teams can measure the cost, efficiency, and applicability of each mode against their use case. Identify excessive and costly interaction modes and create a tailored improvement plan to optimize them depending on each team's preferences, topology, and skills.

Provide training and support to help teams better understand the available interaction modes and how to use them effectively to achieve the desired outcome. By optimizing team interactions, organizations can reduce costs while maintaining efficiency and collaboration.

XaaS is typically the most cost-effective and efficient interaction mode between teams when available, as it involves providing and consuming self-service capabilities rather than sustained direct communications. In this mode, a team provides an interface that can easily be integrated into the existing workflows of one or more teams.

Facilitation is the second most efficient, where a team temporarily mentors another team to provide resources and support to accomplish a task. While facilitation can be more expensive than XaaS, it can also be more effective in situations where face-to-face communication or more direct support is needed.

Collaboration is the least efficient interaction mode. It involves working together as a team to achieve a common goal. This interaction mode can be highly effective in certain situations; however, it can also be more time-consuming and less cost-efficient than other interaction modes. Meetings are expensive, and collaboration always requires direct involvement between teams. Find the right balance between the different interaction modes by choosing the right mode for the use case.

[OA.TI.11] Offer optional opportunities for cross-team collaboration

Category: RECOMMENDED

Establish regular communication channels and forums to encourage cross-team collaboration and information sharing. This can include joint planning sessions, team demos, or cross-team retrospectives. Encourage a culture of open communication and collaboration across teams, sharing knowledge, best practices, and lessons learned. Monitor the effectiveness of these cross-team communication and collaboration opportunities and adjust the approach as needed based on feedback and observed outcomes.

Anti-patterns for team interfaces

- **Documentation overload and neglect**: Having too much documentation can slow down the development lifecycle and increase the cognitive load of readers. Insufficient or outdated documentation can lead to misinformed decisions and missed opportunities. Create a balance between comprehensive documentation and agile methodology. Continually update documentation to reflect current practices and insights, while also maintaining a focus on action and progress.

- **Lack of cross-functional collaboration:** Teams that operate in silos and avoid collaborating with other departments can miss broader organizational context and perspectives. Siloed teams often result in longer feedback loops and reduced sharing of best practices across the organization. Introduce opportunities for teams to meet, such as joint workshops, gamification opportunities, technical communities, or using shared documentation platforms. These opportunities can help to create a culture of open communication and knowledge sharing among teams.
- **Inflexible approval processes**: Overly bureaucratic approval processes can act as barriers to progress, inhibiting agility and slowing down product delivery. These processes can lead to delays and reduce the organization's ability to adapt quickly to changes. Make improvements to quality assurance and automated governance capabilities to introduce automated change management workflows. By automating these traditionally human-driven decision gates, teams become more autonomous leading to improved deployment frequency and reduced time to market.

Metrics for team interfaces

- **Feedback response time:** The average time it takes for a team to address and respond to customer feedback. This metric indicates agility and adaptiveness to customer needs and showcases the team's efficiency in addressing feedback. Implement a dedicated system to track and prioritize customer feedback. Store a timestamp for when the feedback is received and another when the feedback is responded to. Calculate the difference and average over all feedback received within a set period.
- **Handoff frequency:** The count of handoffs in a process, including those across different teams. Having many handoffs can indicate potential inefficiencies, bottlenecks, and areas where errors might occur due to excessive handoffs. Use process mapping tools, such as value stream mapping, to understand and visualize workflows. Count the number of handoffs within a given process, particularly between different teams. A lower number of handoffs across many processes generally indicates more autonomy within a team.
- **Knowledge sharing index:** This refers to how much teams share, update, and use collective knowledge repositories and assets. This highlights the effectiveness of knowledge management processes and the extent to which shared knowledge supports team activities. Track the number of accesses, contributions, and modifications to shared repositories or platforms. Consider using surveys or setting up a scale to assess the perceived value and utility of shared knowledge sources.
- **Onboarding satisfaction (OSAT):** The satisfaction level of new team members regarding their onboarding experience. This metric provides insights into the effectiveness and efficiency of

the onboarding process, identifying areas that need improvement. Use surveys to measure satisfaction on a scale (for example, 1–10) after a new hire goes through the onboarding process. Calculate the average score and track changes over time.

Balanced cognitive load

Maintaining a balanced cognitive load challenges team members without overwhelming or under-stimulating them. Focusing on cognitive load not only promotes individual job satisfaction, but also yields improved performance by teaching them to process information effectively, learn from their experiences, and retain knowledge. In a DevOps environment, a balanced cognitive load supports a healthy environment for innovation and adaptability, directly contributing to the overall success of the organization.

Topics

- Indicators for balanced cognitive load
- Anti-patterns for balanced cognitive load
- Metrics for balanced cognitive load

Indicators for balanced cognitive load

Maintain a balance where team members are adequately challenged without feeling overwhelmed, improving job satisfaction, innovation, and organizational success.

Indicators

- [OA.BCL.1] Clarify purpose and direction to improve cognitive well-being
- [OA.BCL.2] Automate repetitive tasks to reduce toil
- [OA.BCL.3] Reduce troubleshooting and technical debt through continuous improvement
- [OA.BCL.4] Boost team efficiency by limiting work in progress
- [OA.BCL.5] Establish clear escalation paths and encourage constructive disagreement
- [OA.BCL.6] Provide teams the autonomy to make decision that align with organizational objectives
- [OA.BCL.7] Cultivate a psychologically-safe culture for experimentation
- [OA.BCL.8] Determine team sizes based on cognitive capacity
- [OA.BCL.9] Use guiding principles to make consistent team decisions

- [OA.BCL.10] Make informed decisions using data

[OA.BCL.1] Clarify purpose and direction to improve cognitive well-being

Category: FOUNDATIONAL

Verify that every individual feels aligned with the organizational goals and sees the impact of their contributions. A motivated workforce that is driven by a sense of purpose can lead to enhanced cognitive well-being, reduced burnout, and improved retention rates.

Regularly communicate a clear strategy, and communicate the organization's business objectives it to all team members. Align individual goals and targets with the business objectives so that every member understands their unique role and contribution. Provide frequent updates on organizational progress towards achieving business goals to keep the team informed and involved. Implement a structured feedback mechanism and recognize the efforts of team members in contributing to the organization's success.

Related information:

- Amazon's approach to security during development: Ownership

[OA.BCL.2] Automate repetitive tasks to reduce toil

Category: FOUNDATIONAL

Identify repetitive, time-consuming tasks, referred to as toil. Assess the potential for automation, setting a high standard for automation with limited allowance for manual work. Continually review and reduce this allowance as more tasks are automated. Implement automation tools and processes to reduce toil and improve overall team efficiency. Encourage team members to identify opportunities for automation, and provide the necessary training and resources to support automation efforts.

Related information:

- AWS Well-Architected Cost Optimization Pillar: COST11-BP01 Perform automations for operations

[OA.BCL.3] Reduce troubleshooting and technical debt through continuous improvement

Category: FOUNDATIONAL

Proactively reducing the frequency of interruptions and addressing technical debt can have a significant positive impact on overall DevOps adoption. Allocating budget and a portion of the team's time to improve existing processes, environments, and workloads can yield a net improvement to overall development speed, code quality, and system stability. This can be achieved by implementing tools, processes, and team norms to identify, track, and manage technical debt, as well as regularly assessing and prioritizing process improvement opportunities.

To focus teams on impactful improvements, encourage teams to factor in time and effort towards these initiatives. Establish metrics to measure their impact. Prioritizing addressing technical debt as part of regular work can also reduce the likelihood of production issues, ultimately resulting in more stable and reliable systems.

[OA.BCL.4] Boost team efficiency by limiting work in progress

Category: FOUNDATIONAL

Provide ample capacity to accomplish goals on time by reducing work in progress (WIP). Prioritize finishing tasks over starting new ones, which helps to reduce context-switching and impacts overall team efficiency. Continually monitor and adjust WIP limits to prioritize tasks that align with business outcomes. Encourage teams to use agile project management tools and rules, such as Kanban or Scrum, to manage work in progress and complete tasks in a timely manner.

[OA.BCL.5] Establish clear escalation paths and encourage constructive disagreement

Category: FOUNDATIONAL

Optimize issue resolution by establishing clear escalation paths and making it part of every team's norms. Define and communicate processes for how and when to escalate issues, and identify the individuals or groups responsible for making decisions. The escalation process should be documented, data-driven, and shared broadly so that everyone is aware of the steps involved. Encourage open communication and a culture of constructive disagreement, where team members can respectfully challenge decisions while still committing to a strategy as a team once a decision has been made. Escalation should not be feared, but instead expected to be frequent and fully supported by leadership. Once a decision is made through the escalation process, everyone should commit to the decision that is made.

Introduce the concept of the *Andon cord*, inspired by Toyota's manufacturing process and adopted by companies like Amazon, as an actionable step to help team members raise concerns and stop processes when problems arise. The Andon cord serves as a mechanism for team members to escalate issues quickly, addressing problems promptly and effectively.

[OA.BCL.6] Provide teams the autonomy to make decision that align with organizational objectives

Category: FOUNDATIONAL

Provide teams with the autonomy to make decisions and changes at the lowest level possible. Provide the necessary information, policies, and tools to make informed decisions aligned with the organization's goals and objectives. Establish clear guardrails to guide decisions and achieve consistency with the overall strategy while avoiding adverse impacts on other teams or the organization. Encourage a culture of empowerment, where team members feel confident in making decisions and taking action.

Related information:

- Amazon's approach to security during development: Ownership

[OA.BCL.7] Cultivate a psychologically-safe culture for experimentation

Category: FOUNDATIONAL

Encourage experimentation and learning from failures by establishing clear guidelines and hosting sharing sessions for both successful and failed experiments. Foster a psychologically-safe environment where team members feel encouraged to share their ideas and speak up without fear of negative consequences. Recognize and celebrate successes, while also recognizing individuals who take risks and contribute to innovation. Cultivate a culture that values open communication, feedback, and continuous learning. Provide support for team members who are willing to experiment and think big.

Related information:

- Amazon's approach to security during development: Technical fearlessness

[OA.BCL.8] Determine team sizes based on cognitive capacity

Category: FOUNDATIONAL

Determine team size using Dunbar's number and the 7 ± 2 rule, which state that teams should be composed of no more than 7 to 10 individuals. With smaller teams, they require the autonomy and resources necessary to be successful within their value stream. Keep team sizes manageable, and foster effective communication, collaboration, and decision-making. Regularly assess the team's

composition and structure, making adjustments as needed to maintain efficiency and effectiveness. Provide teams with the necessary tools and resources to support collaboration and communication in small group settings.

Larger organizations might have teams of teams, often referred to as guilds, chapters, or squads. The name used here is completely dependent on the organization's preference and culture. In this scenario, Dunbar's number advises that the number of people not exceed 150. Having a team of teams allows smaller teams to work together towards common goals that align to related business outcomes, while maintaining the benefits of smaller team sizes and scaling effectively within a larger organization.

Related information:

- Dunbar's number
- The Magical Number Seven, Plus or Minus Two
- Two-Pizza Teams

[OA.BCL.9] Use guiding principles to make consistent team decisions

Category: RECOMMENDED

Establish team guiding principles. These guiding principles should be created collaboratively and should outline the team's purpose, goals, values, and operating principles. Verify that the charter is well understood by all team members and regularly referenced in decision-making processes. Periodically review and update the guiding principles to ensure they remain relevant and aligned with the team's evolving goals and values. Encourage team members to use these guiding principles as a framework for making decisions and resolving conflicts, ensuring consistency and alignment across the team.

Related information:

- Tenets: supercharging decision-making

[OA.BCL.10] Make informed decisions using data

Category: RECOMMENDED

Encourage teams to shift from relying solely on intuition or personal experience to using data to inform their decisions so that they become more objective than subjective. Teams should

consider what to measure (and why), how to measure it, and how to effectively present the data for informed decision making. Provide training on data analysis and visualization, and aim to make data easily accessible and up-to-date. Use tools to collect, store, analyze, and visualize data effectively, allowing teams to make data-driven decisions.

Incorporate the *build-measure-learn* feedback loop and *validated learning* concepts from Eric Ries' The Lean Startup to enhance decision-making capabilities and alignment with organizational goals. Use the build-measure-learn loop and validated learning to make data-driven decisions, test assumptions, and adapt quickly to changing conditions to foster a culture of continuous learning and improvement. By partnering with customers throughout the innovation process, teams can better align their efforts with organizational objectives while remaining agile and responsive to evolving requirements. Doing so confirms that solutions are built with customer empathy, measured for impact, and refined through collaborative learning.

Related information:

- The Lean Startup Methodology

Anti-patterns for balanced cognitive load

- **Imbalanced workload**: Both overloading team members with too many tasks or responsibilities and underutilizing them by assigning insufficient responsibilities or challenges can lead to reduced productivity. An imbalanced workload can hinder an individual's ability to learn and retain knowledge, ultimately impeding the effectiveness of DevOps adoption. Strike the right balance between challenging team members and avoiding burnout to foster a productive and engaged team.
- **Oversized teams**: Forming excessively large teams that exceed the cognitive limit of effectively managing relationships as described by Dunbar's number (approximately 150 people). As teams grow beyond this size, there is a risk that communication, collaboration, and decision making will suffer, reducing the team's overall effectiveness. Create smaller teams that can communicate more effectively, make better decisions, and deliver higher-quality products and services. This is especially important in the beginning stages of product development; try to keep your teams as small as you can for as long as you can.
- **Lack of autonomy and psychological safety**: Cultivating an environment that lacks autonomy and intolerance to failure can lead to risk aversion and an unwillingness to experiment and innovate. In environments that micromanage or over prioritize being risk adverse, individuals tend to shift focus from delivering value to avoiding failure, resulting in sub-optimal outcomes

and slower delivery. Embrace a culture of psychological safety that encourages autonomy, experimentation, and risk-taking. Allowing individuals to make decisions and experiment can lead to significant breakthroughs and innovations, as well as higher job satisfaction and performance.

Metrics for balanced cognitive load

- **Team size ratio:** Comparing the actual team size against the recommended team size based on cognitive limits. To measure this metric, compare the current team size to the recommended team size to calculate the ratio based on cognitive limits, for example, less than 10 people in individual teams, and less than 150 in a team of teams.
- **Burnout assessment scores:** Evaluate team members' well-being using a burnout assessment survey. For example, the Maslach Burnout Inventory general survey delves into exhaustion, cynicism, and professional efficacy as indicators of burnout. Distribute the questionnaire to team members on a regular basis, such as quarterly or biannually, to monitor trends and respond proactively to burnout indicators. Over time, track changes to help ensure that interventions are effective and that overall team well-being is trending positively. Encourage open communication and feedback, allowing teams to voice concerns or provide suggestions to improve the work environment. Consider working with human resources (HR) professionals or occupational psychologists to interpret the results and develop appropriate interventions.

Adaptive work environment

An adaptive work environment allows organizations to maximize team performance and collaboration, whether teams are working onsite, remotely, or a mix of both. By providing the necessary tools, processes, and support, organizations can create an environment that enables teams to collaborate efficiently, share knowledge, and adapt to changing circumstances. This flexibility is essential for maintaining productivity and agility in a constantly evolving business landscape. By implementing inclusive collaboration options, flexible work schedules, adaptable workspaces, and regular team-building activities, organizations foster a more cohesive and efficient team dynamic, ultimately contributing to the success of their DevOps initiatives.

Topics

- Indicators for adaptive work environment
- Anti-patterns for adaptive work environment
- Metrics for adaptive work environment

Indicators for adaptive work environment

Support diverse, distributed teams by offering flexible and adaptable work environments that enable sustainable productivity and collaboration.

Indicators

- [OA.AWE.1] Equip teams with feature-rich tools for virtual collaboration
- [OA.AWE.2] Offer inclusive options for both virtual and on-site collaboration
- [OA.AWE.3] Balance work schedules for diverse global teams
- [OA.AWE.4] Provide adaptable workspaces for effective on-site collaboration
- [OA.AWE.5] Organize team-building activities and social events to foster a sense of community and promote collaboration

[OA.AWE.1] Equip teams with feature-rich tools for virtual collaboration

Category: FOUNDATIONAL

In a DevOps environment, collaboration tools are required to facilitate effective communication and collaboration among distributed teams. These tools allow teams to rapidly make decisions and solve problems together. Provide well-integrated collaboration tools that support virtual collaboration through chats, voice, video, break-outs, and interactive boards for virtual meetings.

These tools should be available on different devices, including desktops, tablets, and mobile. Invest in training for teams on how to use these tools effectively and securely. Gather feedback from teams on the suitability of the collaboration tools and any new features that could enhance their virtual collaboration experience.

[OA.AWE.2] Offer inclusive options for both virtual and on-site collaboration

Category: FOUNDATIONAL

Create a more inclusive and high-performing work environment by accommodating employees with diverse needs. To improve the organization's capability to accommodate employees with special needs, conduct assessments of existing facilities and identify areas that require improvement.

Collaboration tools should include accessibility features such as closed captioning, screen readers, and speech-to-text capabilities. Promote an inclusive culture throughout the organization by

providing training for employees on topics such as diversity, inclusion, and accessibility. Gather feedback from employees with special must identify areas for improvement and make necessary adjustments to create a more inclusive and accessible work environment.

[OA.AWE.3] Balance work schedules for diverse global teams

Category: RECOMMENDED

Flexible work policies for appropriate roles are recommended because they help organizations attract and retain skilled employees, while also promoting a healthy work-life balance, improving employee satisfaction, and facilitating global collaboration. Establish policies and guidelines that facilitate remote work and flexible schedules, while fostering communication and collaboration among team members across different time zones and locations.

Use techniques such as *follow-the-sun* support models and handovers to promote seamless collaboration across different time zones. Schedule meetings that are convenient for all team members or record and share information if such scheduling is not feasible. Provide employees with the necessary technology and tools to effectively work remotely, while protecting company information through appropriate security measures. Seek feedback from employees to refine and improve the organization's remote work and flexible schedule policies to better meet the needs and preferences of its workforce.

[OA.AWE.4] Provide adaptable workspaces for effective on-site collaboration

Category: RECOMMENDED

Having a flexible and reconfigurable workspace environment promotes DevOps adoption by allowing for customizable collaboration and communication methods that fit individual and team needs. When teams work in the office or use a hybrid approach that requires meeting in person, they require tools and equipment to support their unique ways of working. If your team is fully remote and does not ever meet in person in a designated office, this capability might not apply to your organization.

Evaluate the current workspace layout and identify areas that can be reconfigured to better support in-person collaboration. Arrange the seating of teams and team members working on the same products or closely collaborating teams to be in close proximity to each other. This arrangement improves communication, collaboration, and problem resolution among team members. Provide on-site collaboration tools, such as meeting rooms, physical and virtual whiteboards, projectors, and conferencing equipment.

Keep the workplace area clean, organized, and accessible for all employees. Gather feedback from teams to assess the effectiveness of the workspace environments, and make necessary improvements to be sure that they meet the needs of the teams.

[OA.AWE.5] Organize team-building activities and social events to foster a sense of community and promote collaboration

Category: OPTIONAL

Organize regular team-building activities and social events to help team members build relationships, foster a sense of community, and promote collaboration. These events can be both in-person and virtual to accommodate remote team members. Encourage employees to participate and provide feedback on these activities to collect data on how impactful or enjoyable they are. These events are more impactful for distributed teams that span multiple time zones, and cities, or work fully remote.

Anti-patterns for adaptive work environment

- **Insufficient collaboration tools:** Relying on outdated tools that do not support efficient communication and collaboration between team members. Modern collaboration tools provide feature-rich environments which help facilitate effective team interactions, regardless of geographical location. Invest in up-to-date collaboration tools that support both virtual and in-office collaboration. Regularly assess the needs of your team to help ensure that the tools match their requirements.
- **Exclusionary practices:** Using collaboration tools and practices that are not inclusive could alienate some team members and lead to a lack of engagement and participation. Inclusivity helps ensure that all team members, regardless of their preferences or constraints, feel engaged and can contribute equally. Regularly review collaboration tools and practices to verify that they meet the needs of all team members, promoting an inclusive work environment.
- **Rigid ways of working:** Mandating strict working conditions without accommodating individual needs and time zones. Flexibility in ways of working enhances productivity and collaboration, especially with global teams spread across different time zones. Implement flexible working conditions, taking into account individual requirements and the global nature of teams. This could include flexible work schedules or permitting team members to work virtually.
- **Inflexible workspaces:** Having static workspaces that do not cater to different team needs and collaboration styles can make it difficult for team members to work together efficiently. Adaptable workspaces foster better communication, enhance productivity, and cater to varied

team requirements. Re-design workspaces to be more modular and adaptable, ensuring that they can be easily reconfigured to match the evolving needs of teams.

- **Neglecting team-building activities**: Failing to organize regular team-building activities and social events prevents the opportunity to strengthen team relationships and build trust. Schedule regular team-building exercises and social events, ensuring they are inclusive of both virtual and on-site team members, and foster a sense of community within the team.

Metrics for adaptive work environment

- **Employee net promoter score (eNPS):** Measure employees' engagement and satisfaction within the organization, gauging their likelihood to recommend the organization as an ideal workplace. This can provide insight into the overall health of the organizational culture and indicates the effectiveness of leadership in creating an inclusive and positive work environment. A higher eNPS can correlate with better productivity, lower turnover, and improved team dynamics. Track using periodic, anonymized net promoter score surveys that require no more that 5–7 minutes to complete. Subtract the percentage of detractors (those who score 0–6) from promoters (those who score 9–10) to get the eNPS value. Neutral scores (7–8) can be ignored.
- **Developer efficacy score**: The efficiency and satisfaction of developers measured by their access to and use of necessary tools, resources, and processes that help them get their work done effectively. This metric provides insight into the adaptability of the work environment and how well it caters to the specific needs of the developers. A higher efficacy score indicates a well-adapted environment where developers feel empowered and equipped. Calculate this metric by distributing periodic, anonymized surveys. These surveys should ask team members to rate the effectiveness of tools and resources available to them on a scale from 1 to 10. Then, subtract the percentage of low scorers (those who rate 1–4) from high scorers (those who rate 7–10) to get the developer efficacy score. Neutral scores (5–6) can be ignored. This survey should also inquire about any obstacles, additional tools, or resources they feel would increase their efficiency.

Personal and professional development

Organizations that provide personal and professional growth opportunities are able to improve overall employee satisfaction and enable individuals to be more amenable to adopting new ways of working. While transitioning to a DevOps environment, providing professional skills training to existing employees allows them to become accustomed to and fully utilize new technologies, rules, and practices. By supporting ongoing growth opportunities over time, teams can stay up-to-date

with industry trends, identify areas for improvement, and drive innovation, which is critical in the fast-paced world of DevOps.

Topics

- Indicators for personal and professional development
- Anti-patterns for personal and professional development
- Metrics for personal and professional development

Indicators for personal and professional development

Nurture a culture of innovation and adaptability by providing continuous growth opportunities.

Indicators

- [OA.PPD.1] Encourage collaboration, innovation, learning, and continuous growth to foster a generative culture
- [OA.PPD.2] Allocate time and budget for targeted training
- [OA.PPD.3] Offer diverse and accessible training options
- [OA.PPD.4] Invest in attracting, developing, and retaining skilled employees
- [OA.PPD.5] Recognize and reward continuous learning
- [OA.PPD.6] Promote knowledge sharing through inter-team interest groups

[OA.PPD.1] Encourage collaboration, innovation, learning, and continuous growth to foster a generative culture

Category: FOUNDATIONAL

A generative culture, as defined by Dr. Ron Westrum's research, provides teams with the autonomy and opportunities to experiment and learn from failures, creating a space for development and performance growth. In a generative culture, individuals feel comfortable expressing their ideas and opinions without fear, and information is openly shared for improvement. This culture is more amenable to successful DevOps adoption than pathological and bureaucratic cultures, which are characterized by a focus on individual power and authority and strict adherence to rules and procedures, respectively.

Leaders should promote a culture of openness and inclusivity and provide teams the autonomy and opportunities to experiment and learn from failures. Encouraging these behaviors allows

individual contributors to feel comfortable expressing their ideas and opinions. Recognize and celebrate successes at all levels of the organization, while also providing constructive feedback for improvement as part of performance review processes. Leaders should model these behaviors and create an environment that promotes collaboration, innovation, learning, and continuous growth. For example, if individual contributors are asked to learn about DevOps, cloud technologies, or similar topics, leaders should also strive to become certified and knowledgeable about those topics as well, at least at a high level.

Leaders should model these behaviors and create an environment that promotes collaboration, innovation, learning, and continuous growth. For example, if individual contributors are asked to learn about DevOps, cloud technologies, or similar topics, leaders should also strive to become knowledgeable about those topics as well.

Related information:

- A typology of organisational cultures

[OA.PPD.2] Allocate time and budget for targeted training

Category: FOUNDATIONAL

Allocate dedicated time and budget for internal and external training, specifically targeting areas that are necessary for achieving business objectives and driving transformation. This could include leadership training, new employee training, or continuous training for already experienced individual contributors. Identify relevant skills and knowledge gaps, develop a comprehensive training plan, and dedicate resources and time to complete the training. Implement feedback and evaluation mechanisms to measure training outcomes and identify areas for improvement. Exemplary organizations tend to provide financial support or reimbursements for costs associated with taking certifications or course registration fees.

Related information:

- AWS Well-Architected Cost Optimization Pillar: COST01-BP03 Establish cloud budgets and forecasts
- AWS Certification Paths
- AWS Learning Needs Analysis

[OA.PPD.3] Offer diverse and accessible training options

Category: FOUNDATIONAL

Offer accessible training options with materials and courses made available in multiple languages and formats, including in-person, remote, and self-paced options. Provide accessible and inclusive content for employees with visual or communication impairments, incorporating features like closed captioning and screen reader compatibility.

Leaders should identify the diverse training needs of teams and individual team members, and develop accessible training options that are available in multiple languages and formats. Keep training content relevant and up-to-date. Some organizations choose to implement a learning management system (LMS) to track employee progress and provide access to training materials, while others choose to use content developed by third parties. Gather feedback from employees to improve the training modules and delivery formats.

Related information:

- AWS Certification
- AWS Educate
- AWS Skills Centers
- AWS re/Start
- Cloud Academy: DevOps Training Library
- Pluralsight: DevOps Courses

[OA.PPD.4] Invest in attracting, developing, and retaining skilled employees

Category: FOUNDATIONAL

Invest in attracting, developing, and retaining skilled employees by providing clear role definitions, mentorship programs, career advancement opportunities, and actionable feedback. Gather feedback regularly from employees to understand their needs and inform training and development initiatives.

Regularly collect feedback from employees to gauge their needs, directing training and development initiatives accordingly. Build transparent reward and recognition programs, and communicate promotion criteria unambiguously to every employee. This proactive approach crafts

an environment where employees can flourish and significantly contribute to the organization's triumph.

[OA.PPD.5] Recognize and reward continuous learning

Category: RECOMMENDED

Establish measurable learning targets, including stretch goals, and design meaningful reward systems to encourage team members to meet their set targets. Regularly provide feedback and progress assessments, which keeps employees aligned with their learning goals. Emphasize the significance of soft skills training and how they correlate with the broader business and team goals. Celebrate and broadly acknowledge individual and team accomplishments related to learning and skill development to reinforce the organization's commitment to fostering a culture of continuous learning.

Encourage individuals to pursue relevant certifications that align to their roles and responsibilities to validate their expertise and keep up to date with latest technologies and best practices. Consider financial incentives or reimbursements for successfully obtaining a certification to further motive team members to invest in their continued learning. Organizations can also host regular internal training sessions, workshops, or mentorship programs to facilitate individuals learning from one-another and help accelerate learning through collaboration. Exemplary organizations tend to introduce immersive experiential learning platforms that develop skills through simulation, hands-on problem solving, and gamification.

Related information:

- AWS Certification Paths
- AWS Ramp-Up Guide: DevOps Engineer
- AWS Jams

[OA.PPD.6] Promote knowledge sharing through inter-team interest groups

Category: RECOMMENDED

Facilitate and support knowledge-sharing opportunities and interest groups, often called *skills guilds*, that allow individuals to interact with other like-minded people within the organization on topics of interest. These groups can partake in activities such as internal blogging, hosting internal conferences, attending external events, or group discussions. These opportunities allow

for individuals to share their experiences, discuss industry trends, and collaborate on projects with others outside of their immediate team.

Allocate time and resources to support these opportunities and groups tailored based on your organization's unique needs and circumstances. This can range from dedicating specific hours each week, providing meeting spaces, or assigning budget for professional development courses and symposiums. Hosting *lunch and learns* or *tech talks*, where passionate individuals or teams discuss specific topics or showcase their projects, can be a great start to facilitating inter-team collaboration.

We recommend creating groups which focus on each of the DevOps Sagas presented in this guidance. Groups may choose to further expand into sub-groups to focus on individual capabilities, such as continuous deployment, everything-as-code, monitoring, or security testing, as they see fit. These opportunities and groups help to break down silos and improve cross-team collaboration, which can hasten DevOps adoption.

Anti-patterns for personal and professional development

- **Unclear growth opportunities:** Solely depending on existing expertise without providing consistent learning opportunities, combined with a lack of clarity on career progression. Without regular upskilling and a visible growth path, teams can become demotivated and less efficient. Regularly offer training, workshops, and certifications while also outlining clear career pathways that link professional development to career advancement in the organization.
- **Non-inclusive training:** Neglecting to consider non-functional skill sets like soft skills and considering the diverse needs of team members might not fully address specific skill gaps, learning styles, or embrace the unique backgrounds and experiences of team members. This can limit the effectiveness of the training and alienate portions of the team. Design targeted training programs that cater to various skill levels, functional and non-functional skills, and recognize the importance of diversity and inclusivity. Ensure that these programs are adaptable and resonate with the unique requirements of team members.
- **Reactive talent development:** Only providing development opportunities when a skill gap emerges or when talent starts looking elsewhere, rather than proactively investing in talent attraction, development, and retention. This short-term view can cause a delay in progress and missed growth opportunities. Be proactive about identifying industry trends and upcoming skills requirements. Regularly engage in discussions with team members about their career aspirations and invest in programs that help them achieve these goals.

- **Ignoring autonomous knowledge sharing**: Not supporting inter-team knowledge sharing fails to harness the collective knowledge and skills within the organization. Self-directed learning provided through these interactions enables team members to engage with others in the organization about innovative solutions and practices that emerge from diverse experiences and expertise. Encourage team members to actively participate in interest groups to share their experiences, discuss industry trends, and collaborate on projects outside their immediate teams. Recognize and reward active participation and knowledge-sharing to promote a culture of continuous learning and collaboration.

Metrics for personal and professional development

- **Employee net promoter score (eNPS):** Measure employees' engagement and satisfaction within the organization, gauging their likelihood to recommend the organization as an ideal workplace. This can provide insight into the overall health of the organizational culture and indicates the effectiveness of leadership in creating an inclusive and positive work environment. A higher eNPS can correlate with better productivity, lower turnover, and improved team dynamics. Track using periodic, anonymized net promoter score surveys that require no more that 5–7 minutes to complete. Subtract the percentage of detractors (those who score 0–6) from promoters (those who score 9–10) to get the eNPS value. Neutral scores (7–8) can be ignored.
- **Meetup frequency:** The frequency of inter-team knowledge-sharing sessions, such as skills guild meetings, internal conferences, or *Lunch and Learns*, held within a specific period, such as monthly or quarterly. A consistent frequency of meetups indicates active collaboration, knowledge transfer, and a vibrant sharing environment among teams. Track each knowledge-sharing session in a dedicated log or system. At the end of your tracking period, tally the total number of sessions.
- **Retention rate:** The percentage of team members that remain with the organization over a specific time frame, such as quarterly or yearly. A high retention rate suggests a positive organizational culture, employee satisfaction, and effective professional development initiatives. It also signals stability, which can have positive impacts on business continuity, team dynamics, and retaining top talent. Begin by noting the total number of team members at the start of your tracking period. Then, at the end of this time frame, note the number of team members who have left the organization during that period. To determine the retention rate, subtract the percentage of team members who have left from 100%. For example, if 10 out of 100 team members left over a year, then the retention rate would be 90% for that year. Regularly monitor and compare these rates over different time frames to identify trends and areas for improvement.

- **Skill growth rate:** The improvement in employees' skills over a specified period, which indicates the effectiveness of personal and professional development initiatives and an organizational emphasis on continuous learning and growth. Improve this metric by allocating budget and time to dedicated learning activities, setting individual learning goals, and rewarding continuous growth. Use tools such as skill assessments, certifications, or self-assessments to track skill improvements over time.

Development lifecycle

The development lifecycle saga provides a prescriptive approach to optimizing an organization's ability to develop, review, build, and release workloads to improve delivery speed and create safer deployments. Users of this guidance can achieve speed and safety through actionable feedback loops, repeatable and consistent deployment methods, and the use of *everything as code*. The AWS Deployment Pipeline Reference Architecture puts this saga into action by providing detailed architectures and in-depth implementation guidance, including code samples.

Capabilities

- Local development
- Software component management
- Everything as code
- Code review
- Cryptographic signing
- Continuous integration
- Continuous delivery
- Advanced deployment strategies

Local development

Local development concentrates on establishing development environments that mirror the production setup as closely as possible, either on a local machine or in the cloud. The primary goal is to allow developers to receive feedback as fast as possible in the development lifecycle without impacting other team members or systems. Using development environments aids in early error detection and the swift rectification of issues before they are introduced into other environments. By adopting this capability, development teams can improve productivity, reduce the time it takes to develop new features, and deliver higher-quality software.

Topics

- Indicators for local development
- Anti-patterns for local development
- Metrics for local development

Indicators for local development

Enhance developer experience by modernizing the local development workflow. This allows developers to work in production-like environments dedicated to development activities while reducing early-stage bugs and promoting higher-quality code.

Indicators

- [DL.LD.1] Establish development environments for local development
- [DL.LD.2] Consistently provision local environments
- [DL.LD.3] Commit local changes early and often
- [DL.LD.4] Enforce security checks before commit
- [DL.LD.5] Enforce coding standards before commit
- [DL.LD.6] Leverage extensible development tools
- [DL.LD.7] Establish sandbox environments with spend limits
- [DL.LD.8] Generate mock datasets for local development
- [DL.LD.9] Share tool configurations
- [DL.LD.10] Manage unused development environments
- [DL.LD.11] Implement smart code completion with machine-learning

[DL.LD.1] Establish development environments for local development

Category: FOUNDATIONAL

Create development environments that provide individual developers with a safe space to test changes and receive immediate feedback without impacting others on the team or shared environments. Development environments are small scale, production-like environments that provide a balance between providing developers with accurate feedback and being low cost and easy to manage. Development environments serve a different purpose than sandbox environments and should be used for day-to-day development and experimentation that requires access to your software components and services.

Development environments can take the form of dedicated cloud environments, local emulations of infrastructure, or be hosted on a local workstation. While most cloud providers, open-source tools, and third parties provide options for emulating infrastructure locally on development machines, these tools might not have full feature parity, leaving them to only be suitable for a subset of use cases. Using cloud-based development environments provides the most reliable, accurate, and complete coverage when working with cloud workloads. We recommend providing a cloud-based development environment to each developer, with each environment being in a separate AWS account.

Developers should be encouraged to use their own development environments for testing and debugging to reduce the chance of problems occurring in environments shared by the broader team. To keep the development environment as close to the production setup as possible, deployments to the development environment should be sourced from the main releasable branch, rather than from long-lived development branches. The development environment setup should be well-documented in an up-to-date playbook that is readily available to all members of the team. For this to be effective, the playbook must be updated as the needs of the team and environment change over time. Ideally, the full lifecycle of these environments, including provisioning, are managed through automated governance processes.

Related information:

- AWS Well-Architected Sustainability Pillar: SUS02-BP05 Optimize team member resources for activities performed
- Setting Up Your AWS Environment
- Dev Environments in CodeCatalyst
- Best practices for testing serverless applications
- Improving the development cycle - Testing in the cloud
- Improving the development cycle - Testing locally

[DL.LD.2] Consistently provision local environments

Category: FOUNDATIONAL

Standardize and automate the process for setting up local development environments using managed services, infrastructure as code (IaC), and scripted automation. This approach permits environments to be reliably replicated across different systems and teams, ensuring

uniformity. Consistent local environments help to reduce issues that occur only on particular machines.

Create a baseline configuration for your local development environment that mirrors the production setup as closely as possible. Use IaC tools to define this environment, and script the provisioning process. All IaC and scripts should be version-controlled, helping to ensure that any changes are tracked and can be rolled back if necessary. Educate developers on the importance of using the provisioned environments and provide documentation on how to set up and troubleshoot these environments. Regularly review and update the baseline configuration to keep it aligned with changes in the production environment. Consider allowing developers to request local environments on-demand through a self-service developer portal.

[DL.LD.3] Commit local changes early and often

Category: FOUNDATIONAL

While developing locally, developers should begin to make small, frequent commits to save versions of their code changes as they develop. Unlike pushing code changes so that they are accessible to other team members, local commits deal specifically with a developer's individual progress as they develop locally. This practice makes local development safer, enabling developers to freely innovate without fear of losing completed work by capturing snapshots of iterative changes to the code base.

Use version control tools, like Git, local testing tools for fast feedback, and conventional commit messages that describe the nature and rationale behind the changes for. Strive to make it a habit to locally commit changes as soon as a logical unit of work is completed. This can be after fixing a bug, adding a new function, or refining an existing piece of code.

Placing emphasis on the significance of making frequent local commits adapts developers to the idea of breaking down work into smaller, more manageable batches of work. This translates into streamlined integration processes when working in a team and is critical for practicing continuous integration and continuous delivery (CI/CD).

Related information:

- Git Basics - Recording Changes to the Repository
- Continuous Integration - Martin Fowler

[DL.LD.4] Enforce security checks before commit

Category: FOUNDATIONAL

Pre-commit hooks can be an effective tool for maintaining security best practices. These hooks can help in the early detection of potential security risks, such as exposed sensitive data or publishing code to untrusted repositories. At a minimum, use pre-commit hooks to identify hidden secrets, like passwords and access keys, before code is published to a shared repository. When discovering secrets, the code push should fail immediately—effectively preventing a security incident from occurring.

Select security tools compatible with your chosen programming languages and customize them to uphold your specific governance and compliance requirements. It is best to integrate these security tools into pre-commit hooks, integrated development environments (IDEs), and continuous integration pipelines so that changes are continuously checked before code is committed into a shared repository.

Related information:

- Security in every stage of CI/CD pipeline: Pre-commit hooks
- Security scans - CodeWhisperer
- Pre-commit
- Husky
- Gitleaks
- GitGuardian
- AWS-IA opinionated pre-commit hooks
- Blog: Extend your pre-commit hooks with AWS CloudFormation Guard

[DL.LD.5] Enforce coding standards before commit

Category: RECOMMENDED

Identify common style, formatting, and other flaws before they are published to a repository. Use static code scanning tools, such as linters, to improve code quality and consistency before pushing committed code. This process can be automated using pre-commit hooks. Upon discovery, pushing the commit should ideally fail and require immediate correction by the developer. Automatically and consistently enforcing coding standards during the local development process directly improves the code review process by removing common errors before manual review.

Select scanning tools compatible with your chosen programming language and customize them to uphold specific coding standards and styles. It is best to integrate these tools into pre-commit hooks, integrated development environments (IDEs), and continuous integration pipelines so that changes are consistently and continuously checked at all stages of the development lifecycle.

Related information:

- Amazon CodeGuru Reviewer
- AWS CloudFormation Linter
- Pre-commit
- Husky
- Validate your AWS SAM applications with AWS CloudFormation Linter
- Workshop: AWS CloudFormation Workshop - Linting and-testing
- Blog: Use Git pre-commit hooks to avoid AWS CloudFormation errors
- Blog: Automate code reviews with Amazon CodeGuru Reviewer

[DL.LD.6] Leverage extensible development tools

Category: RECOMMENDED

Extensible software development tools, primarily integrated development environments (IDEs) or text editors, can be augmented with plugins or extensions. These plugins enhance the functionalities of the software, allowing for improved and tailored developer experiences.

Choose development tools that work well with your primary programming languages and technologies in your stack. Choosing a widely adopted IDE or text editor enables leveraging support communities and extension ecosystems. Teams should be encouraged to experiment with and adopt plugins that enhance code quality, simplify integrations, or speed up routine tasks. Over time, curate a list of preferred, approved extensions that align with your DevOps objectives and security requirements. Verify that there is a process in place for regularly updating these tools and extensions to benefit from the latest improvements and security patches.

Related information:

- Security in every stage of CI/CD pipeline: IDE tools and plugins
- Tools to Build on AWS

- Dev Environments in CodeCatalyst

[DL.LD.7] Establish sandbox environments with spend limits

Category: RECOMMENDED

Sandbox environments are dedicated spaces for developers to explore, experiment, and innovate with new technologies or ideas. Unlike development environments, which are meant for more structured day-to-day development, they allow more fewer controls, while ensuring no connectivity to internal networks or other environments.

Create a comprehensive sandbox usage policy. This policy must set clear boundaries on the kinds of data permissible with the sandbox, ensuring no leakage of sensitive information or code. Establish rules for access controls. Some environments might be tailored for individual developers, while others could serve small teams. Rules regarding network connectivity should ensure that the sandbox remains isolated, preventing any unintended interactions with other internal networks or environments. Set tagging strategies which can aid in managing automation and cost tracking. Overall, ensure that this policy makes a distinction between sandbox environments and development environments, and lays out the use cases best suited for each.

Educate developers on the sandbox usage policy, including responsible and cost-effective resource management techniques. Encourage shutting down or deleting unnecessary resources, especially when they're not in active use. Sandbox environments should be treated ephemerally, with automated governance processes managing the lifecycle to create, manage, clean up resources, and destroy sandbox environments as required.

Related information:

- AWS Well-Architected Cost Optimization Pillar: COST02-BP05 Implement cost controls
- Sandbox per builder or team with spend limits
- AWS Innovation Sandbox
- Cloud Financial Management with AWS
- Sandbox Accounts for Events
- Best practices for creating and managing sandbox accounts in AWS

[DL.LD.8] Generate mock datasets for local development

Category: OPTIONAL

Mock datasets are synthetic or modified datasets that developers can use during the development process, eliminating the need to interact with real, sensitive production data. Using mock datasets ensures tests are thorough and realistic, without compromising security.

Use data generating tools to create mock datasets. These tools can range from random data generators to more advanced methods like generative AI. Generative AI can be used to generate synthetic datasets that can be used to test applications and is especially useful for generating data that is not often included in testing datasets, such as defects or edge cases.

If using real-world data is necessary for local development, ensure it is obfuscated. Methods such as masking, encrypting, or tokenizing production datasets can transform real datasets into mock datasets that are safe for local development. It might be useful to store already prepared mock datasets that can be shared between teams or systems to perform testing with. This approach creates a realistic local testing environment without risking developers handling actual production data.

Related information:

- Testing software and systems at Amazon: Developer environment
- Generate test data using an AWS Glue job and Python
- Foundation Model API Service - Amazon Bedrock
- What is Generative AI?

[DL.LD.9] Share tool configurations

Category: OPTIONAL

Sharing tool configuration among project or team members helps ensure a uniform set up of integrated development environment (IDE) settings, text editor preferences, and pre-commit hooks. Having these configurations tailored to each code base can reduce discrepancies in code styles and promote seamless collaboration and a predictable developer experience. This enables any developer working within that repository to begin working in the environment quickly while maintaining team norms.

Commit tool configuration files to a shared repository. Periodically review these shared configurations, ensuring they remain updated as tools and practices evolve. While the idea promotes consistency, be mindful of the need to occasionally tailor configurations for specific tasks and preferences.

[DL.LD.10] Manage unused development environments

Category: OPTIONAL

Properly managing unused environments prevents unnecessary resource utilization and potential security threats. When development environments are not in use, the environment and associated resources should be disabled or deleted.

Managing unused development environments requires tracking, disabling, or removing development setups that are dormant or no longer in active use. Regularly audit the active and inactive development environments. Implement automated tools or scripts that monitor activity and provide notifications regarding dormant environments.

Once identified, these environments should be archived, disabled, or removed, depending on the future needs of the project. Treat development environments as ephemeral environments to reduces the risk of incurring unexpected cost and leaving potentially insecure resources running.

Related information:

- AWS Well-Architected Sustainability Pillar: SUS02-BP03 Stop the creation and maintenance of unused assets
- AWS Well-Architected Cost Optimization Pillar: COST04-BP03 Decommission resources

[DL.LD.11] Implement smart code completion with machine-learning

Category: OPTIONAL

Use machine learning (ML) algorithms within development tools to predict and suggest code as developers write, based on patterns and commonly used syntax. This can improve development experience, speed up the coding process, and reduce the potential for errors.

Incorporate ML-powered code generators into your developer tools, such as IDEs or text editors, for real-time, intelligent code recommendations. Train and refine these tools with regular feedback to ensure they align with your specific coding patterns and practices.

Related information:

- Amazon CodeWhisperer

Anti-patterns for local development

- **Irregular commits:** Sporadic or large commits can lead to lost progress, complicates the tracking of changes, and reduces the ability to roll back to a specific state. Frequently committing locally enables you to experiment fearlessly, knowing that there is a secure point to return to. Bundling large changes, or combining multiple changes, into a single, large batched commit should be avoided. This practice also usually leads to having larger commits to the main releasable branch of the repository, making it difficult to integrate with other changes, perform code reviews, and deploy with confidence. It is best practice to commit small, logical changes more frequently.
- **Avoiding local development environments:** Avoiding the use of local environments can result in teams making changes directly in production or having multiple developers share a single development account. This restricts developers from safely experimenting and testing their code changes and can result in increased risk of errors, service disruption, and integration issues that make troubleshooting complex. Set up individual local development environments for each developer.
- **Inconsistent local environment setup:** Allowing each developer to set up their local environment without following a standardized process can lead to variations across developer setups. Divergent environments can introduce hard-to-diagnose bugs and the notorious *it works on my machine* scenarios, which can impact overall team productivity. Use automated tools, scripts, and playbooks to help ensure that the team has consistent local development environment setups.
- **Long-lived development branches:** Maintaining long-lived development branches for the purpose of continuously deploying to local environments can lead to merge conflicts, increased technical debt, and divergence from the main code base. To keep the development environment as close to the production setup as possible, deployments to the environment should always be sourced from the main releasable branch which reflects the integrated code base used for production deployments.
- **Over-reliance on basic text editors:** Relying solely on traditional text editors such as VI or Emacs for development activities. While these editors have their merits and uses, they fall short of the capabilities provided by modern IDEs and text editors that are capable of improving developer experience. Use IDEs and text editors that provide a vast plugin ecosystem and can be tailored to specific development tasks and integrate with automated tools.

Metrics for local development

- **Local environment provisioning time:** The amount of time developers spend on setting up or troubleshooting their local development environment. This metric indicates inefficiencies in onboarding or local environment maintenance, which can negatively impact productivity. If you are centrally provisioning or fully automating this process, use observability tools to monitor the time dedicated to environment set up. If individual developers need to take action to set up their environment, conduct regular surveys or use time-tracking tools to gather additional data points.
- **Post-commit test failure rate:** The percentage of tests that fail in an integrated environment after a local commit. This metric allows teams to track and optimize the effectiveness of their local testing environments and workflows. Monitor this metric by calculating the number of tests that fail post-commit in integrated environment divided by the total number of tests run post-commit. Ideally, this metric should trend downwards over time, indicating that local environments are becoming more consistent with integrated environments and that developers are catching more issues before committing changes.

Software component management

There are many software components that are consumed and generated during the development lifecycle, including libraries, repositories, shared modules, build artifacts, and third-party dependencies. These components often have distributed technical ownership and are decoupled from one another. Software component management focuses on overseeing these individual components to enhance security and governance of the software supply chain. This includes routinely updating components to maintain their security and relevance, establishing clear usage guidelines, and creating an inventory of the relationships between components. Through this capability, you can strengthen the security, reliability, and integrity of software being built.

Topics

- Indicators for software component management
- Anti-patterns for software component management
- Metrics for software component management

Indicators for software component management

Gain visibility into code and dependencies used across your organization. This capability helps manage and secure all software components, from in-house libraries to third-party dependencies, thereby increasing the security and reliability of the development process.

Indicators

- [DL.SCM.1] Use a version control system with appropriate access management
- [DL.SCM.2] Keep feature branches short-lived
- [DL.SCM.3] Use artifact repositories with enforced authentication and authorization
- [DL.SCM.4] Grant access only to trusted repositories
- [DL.SCM.5] Maintain an approved open-source software license list
- [DL.SCM.6] Maintain informative repository documentation
- [DL.SCM.7] Standardize vulnerability disclosure processes
- [DL.SCM.8] Use a versioning specification to manage software components
- [DL.SCM.9] Implement plans for deprecating and revoking outdated software components
- [DL.SCM.10] Generate a comprehensive software inventory for each build

[DL.SCM.1] Use a version control system with appropriate access management

Category: FOUNDATIONAL

Version control systems enable tracking and managing of changes to code over time. They allow multiple developers to work on a project concurrently, provide a history of changes, and make it possible to revert to a previous version if necessary. Version control systems play a role in maintaining the integrity of software components, as they provide an auditable trail of all modifications made to the code base, authorizes users as they access the code base, and help to ensure that changes to the code base can be reverted or rolled back.

Implement access management policies on the version control systems which supports a culture of code sharing and collaboration amongst teams in your organization. Having a mix of both open and private repositories allows for a balance between promoting code reuse and collaboration, and safeguarding sensitive information. For open repositories, developers can share code freely to encourage collaboration and learning, while confidential projects or sensitive parts of the code base can use private repositories.

Consider implementing role-based access control (RBAC) in your version control system. Using RBAC, you can restrict write (commit) access to specific roles or individuals and can protect the main code base from inadvertent or inappropriate alterations. This also allows granting broad, organization-wide read access to open repositories, while reserving the ability to limit access to sensitive or confidential private repositories.

Related information:

- What Is Repo?

[DL.SCM.2] Keep feature branches short-lived

Category: FOUNDATIONAL

In version control systems, feature branches provide a structured way to develop new functions or address defects. These branches are carved out with the intent of eventually merging changes into the main code base for release. Traditional branching methods, such as GitFlow, lean towards creating long-lived feature branches which can introduce challenges including complex merges and divergent code bases. Modern branching strategies, including GitHub flow and trunk-based development, emphasize the significance of keeping feature branches short-lived to avoid these challenges. We recommend trunk-based development paired with a pull request workflow utilizing short-lived feature branches as the most effective branching strategy when practicing DevOps.

The core benefit of short-lived feature branches is the promotion of continuous integration. By frequently integrating code changes into the main releasable branch of the repository, teams discover integration problems early on. This approach prevents last-minute chaos when merging code bases leading to software that can be reliably released at any time. We recommend merging into the main releasable branch at least once per day.

Smaller teams might prefer committing directly to the trunk of the releasable branch. Larger teams or those working on complex software might lean towards a Pull-Request workflow that uses short-lived branches. Regardless of the branching strategy you choose to use, the principle remains: branches should be transient, preferably representing a single contributor's work. To enforce this, put a process in place to remove branches that are already merged and prevent long-lived branches by actively deleting branches that surpass a specific retention period.

Related information:

- Trunk-based Development: Short-Lived Feature Branches

- GitHub flow
- A successful Git branching model: Note of reflection

[DL.SCM.3] Use artifact repositories with enforced authentication and authorization

Category: FOUNDATIONAL

Artifact repositories and registries offer secure storage and management for artifacts generated during the build stage of the development lifecycle. Examples of artifacts that are stored in these repositories are container images, compiled software artifacts, third-party modules, and other shared code modules. Using an artifact repository streamlines artifact versioning, access control, traceability, and dependency management, contributing to efficient and reliable software releases. They can significantly improve the auditability, security, and organization of your software artifacts, leading to higher-quality software deliveries.

Artifact repositories are in the critical path for ensuring the integrity of the software that is deployed into your environments. All artifacts in the repository should be expected to be built and tested using trusted automated processes in an effort to prevent errors or bugs from being introduced into the system. Artifact repositories should not contain manually produced artifacts or allow existing artifacts to be altered by users. Altering artifacts in the artifact repository degrades the integrity of the artifact and repository, so artifact repositories should enforce that artifacts are immutable.

Use role-based or attribute-based access control to limit which users and systems can store and modify artifacts in artifact repositories. Access to create, update, or delete artifacts should remain restricted to emergencies, security use cases, and build and deployment processes.

Related information:

- AWS Well-Architected Security Pillar: SEC11-BP05 Centralize services for packages and dependencies
- Artifact Repository - AWS CodeArtifact
- Fully Managed Container Registry - Amazon Elastic Container Registry
- Code Repositories and Artifact Management | AWS Marketplace

[DL.SCM.4] Grant access only to trusted repositories

Category: FOUNDATIONAL

To maintain the security, integrity, and quality of your software, restrict the usage of untrusted source code and artifact repositories. Untrusted repositories present risks, including potentially introducing vulnerabilities into your software and leaking sensitive code or information. As a safer alternative, only use trusted repositories that offer secure, vetted libraries, and dependencies.

Implement policies that control where developers can publish code, to prevent accidental exposure or internal threats. This should apply to both artifact and source code repositories across the organization. Protect against internal threat actors or inadvertently sharing code to public or untrusted git repositories by limiting the allowed repositories that developers can publish code to. Hosting your own repositories might be advantageous depending on your needs, enabling complete control over available code. Methods such as pre-commit hooks for git repositories can be used to enforce these rules effectively.

By enforcing usage of trusted repositories, you ensure that only secure, vetted code components and artifacts are used, enhancing software lifecycle stability and security. It also minimizes the risk of sensitive information being leaked into untrusted repositories.

[DL.SCM.5] Maintain an approved open-source software license list

Category: FOUNDATIONAL

Manage and regularly update an allowed and forbidden open-source software (OSS) licenses list. This list should reflect which licenses are, or are not, compliant with laws, regulations, and security requirements applicable to your organization. Use this list to detect and prevent legal issues while using open-source components.

Enforce the allowed and forbidden OSS licenses list by continuously assessing all OSS usage automatically as part of the build process. This can be enforced through quality assurance testing processes, like scanning the Software Bill of Materials (SBOM) with Software Composition Analysis (SCA) tooling. Continuous enforcement helps to ensure that only approved OSS licenses are used in the code base, reducing the risk of legal issues and license violations while providing developers with fast feedback.

[DL.SCM.6] Maintain informative repository documentation

Category: FOUNDATIONAL

Maintaining well-structured and informative repository documentation directly within the code base promotes collaboration, simplifies onboarding new team members, and improves the ability to maintain software over time. This documentation, often in the form of markdown files like

`README.md` and `CONTRIBUTING.md`, contains information about reviewing, building, contributing to, and otherwise using the project and helps ensure that this knowledge lives where the code does, making it easily accessible and versioned alongside the code it is applicable to.

Every repository should contain detailed documentation providing an overview of the project, its purpose, instructions for building and deploying the project, guidelines for contributions, and methods for submitting feedback or issues. For complex projects, the creation of additional, focused documentation files addressing specific areas can be beneficial.

Related information:

- What Is Repo?
- About READMEs
- Common special files found in the root directory of a repository

[DL.SCM.7] Standardize vulnerability disclosure processes

Category: RECOMMENDED

A standard vulnerability disclosure policy helps ensure consistent reporting and handling of potential vulnerabilities, which in turn enhances the security of the software development lifecycle. Implementing standardized vulnerability disclosure practices is recommended for optimizing DevOps, as it promotes security, helps manage risk effectively, and encourages the responsible reporting and handling of discovered vulnerabilities.

A method for implementation is provided in RFC 9116, *A File Format to Aid in Security Vulnerability Disclosure* (Foudil, Shafranovich, & Nightwatch Cybersecurity, 2022). This guidance provides a standardized process for vulnerability disclosure using a machine readable `security.txt` file, which contains contact details and the vulnerability disclosure policy. This file is to be placed in the `/.well-known/` path of a domain name or IP address to enable security researchers to find the right information to report vulnerabilities they discover easily.

Related information:

- RFC 9116 - A File Format to Aid in Security Vulnerability Disclosure

[DL.SCM.8] Use a versioning specification to manage software components

Category: RECOMMENDED

Apply a versioning specification across all software components within your development lifecycle. Use a versioning specification, such as Semantic Versioning (SemVer), to significantly simplify governance of software governance by providing a systematic approach to tracking different types of releases (major, minor, and patch). A well-organized, versioned code base offers a clear chronological history of modifications, enhancing manageability, maintainability, and navigability.

Implementing version pinning for dependencies is a practical use case enabled by using a versioning specification. By locking dependencies to a specific version or version range, build reproducibility is ensured. This approach helps ensure the reproducibility of software builds, but complicates dependency management as developers then need to make updates to stay up-to-date with security fixes, bug fixes, or other improvements.

Use automated governance dependency management tools to maintain the balance between stable builds and timely updates. Consider integrating automation mechanisms that can update versions based on commit messages. For example, if a commit message contains the keyword `major`, it could trigger an update to the major version number. This automated approach ensures that versions are updated while minimizing chance for human error. It's also possible to automate nightly or weekly upgrades of third-party dependencies to ensure they are regularly updated and kept secure.

Related information:

- Semantic Versioning 2.0.0

[DL.SCM.9] Implement plans for deprecating and revoking outdated software components

Category: RECOMMENDED

Maintaining an up-to-date and secure code base requires the proactive management of components, including removing outdated artifacts, libraries, and repositories. Not only does their removal reduce storage costs, but it also mitigates risks associated with deploying outdated or potentially vulnerable software. The removal process of outdated components should comply with the organization's data retention policies.

Develop clear plans for the deprecation and revocation of outdated components. These plans should include regular audits of the code base to identify deprecated or unused artifacts, libraries, and repositories. Establish timelines for deprecation and final removal of identified components. Communicate these plans to your development team and ensure that they are aware of the timelines.

Consider automating the removal process where feasible, for example, by using scripts or automated governance tools that support such functionality. By implementing such plans, you can streamline the code base, making it easier to manage and less prone to errors, while ensuring security and reducing the risk of system failures.

Related information:

- AWS Well-Architected Cost Optimization Pillar: COST04-BP05 Enforce data retention policies
- AWS Well-Architected Sustainability Pillar: SUS02-BP03 Stop the creation and maintenance of unused assets

[DL.SCM.10] Generate a comprehensive software inventory for each build

Category: RECOMMENDED

Maintain a comprehensive inventory of the components and dependencies that make up your software assists with identifying vulnerabilities and managing risks. This inventory, often taking the form of a Software Bill of Materials (SBOM), provides valuable insights into the composition of your software.

Generate a comprehensive inventory as part of each build. This forms a continuous record of your software's composition, enabling quick and efficient identification and management of potential vulnerabilities or risks. Tracking inventory that is machine readable enhances visibility and aids in identifying vulnerabilities and risks, enhancing the security posture of your software at scale.

Use a tool to create and manage SBOMs, centralizing them with other build artifacts for easier accessibility. Open-source tool sets provided by Open Worldwide Application Security Project (OWASP) and the Linux Foundation offer options for creating and managing SBOMs in standardized formats.

Related information:

- Exporting SBOMs with Amazon Inspector
- SPDX Becomes Internationally Recognized Standard for Software Bill of Materials
- Software Supply Chain Best Practices
- OWASP CycloneDX

Anti-patterns for software component management

- **Avoiding version control:** Not using version control means that there's no historical record of code changes. This makes it hard to track down when a bug was introduced, who made specific changes, or how to roll back to a previous state leading to increased debugging time, potential loss of code, inability to collaborate effectively, and no traceability of changes. Use version control systems like Git for both code and infrastructure (IaC). Make sure to commit frequently with meaningful commit messages and maintain a clear change log.
- **Mutable artifacts:** Permitting alterations to existing artifacts in artifact repositories and registries undermines the integrity of the software supply chain. Instead, artifacts should be made immutable. Immutable artifacts help ensure consistent deployments and rollbacks. Mutable artifacts, on the other hand, introduce unpredictability and can be a vector for malicious alterations. Artifacts stored should be immutable with access control preventing overwriting or modifying existing artifacts.
- **Ignoring dependencies management:** A significant portion of modern applications consists of third-party libraries or dependencies. Ignoring these can introduce hidden vulnerabilities, versioning conflicts, or deprecated functionalities. This can lead to security breaches, unexpected behavior in applications, and makes debugging more challenging. Adopt a comprehensive dependency management strategy that not only tracks direct dependencies but also transitive (dependencies of dependencies). Use tools to regularly scan for vulnerabilities in your dependencies and update them as needed. Keep a centralized inventory of third-party dependencies that can be audited. Only grant build systems usage to trusted sources or specific repositories that have been approved.
- **Using git submodules for sharing common code:** Git submodules can introduce a layer of complexity in development. It is easy to end up with a parent repository pointing to an outdated or even deleted commit in the submodule. Submodules can also introduce recursive nesting. This complexity can hinder development speed, increase the risk of errors, and introduce potential security concerns if a submodule is compromised. Consider alternative strategies such as monorepos for large-scale projects or microservices for better separation of concerns. Package managers can be used to share common libraries across multiple projects without the complexity of submodules.
- **Traditional branching strategies:** Traditional branching strategies, like GitFlow, involve multiple long-lived branches like feature, release, and hotfix, which can add overhead to the development process. For most modern web-based applications, especially those using continuous integration and continuous delivery (CI/CD) pipelines, this can slow down the release pace, extend integration times, and lead to complex merge conflicts. Adopt a simpler branching

strategy like trunk-based development or GitHub flow, which emphasizes shorter-lived branches and more frequent merges to the main branch.

Metrics for software component management

- **Average branch lifespan:** This metric tracks the lifespan of feature branches. A shorter average lifespan indicates a more agile development process indicative of continuous integration. Track this metric by gathering data from version control systems and analyzing repository statistics.
- **Open-source license violations:** The number of open-source components used that do not align with approved license lists. Tracking this metric can help monitor adherence to legal compliance requirements. Gather data by integrating license-checking tools, such Software Composition Analysis (SCA) tools, into pipelines and review the findings. Supplement this data by also including builds which do not include SCA scans.
- **Average time to resolve vulnerabilities:** Represents how quickly software vulnerabilities and security risks are mitigated. A reduced average time indicates a proactive security approach. To improve this metric, prioritize vulnerabilities based on severity, integrate security into the development lifecycle, enhance developer security collaboration, and introduce security training. Track this metric by recording timestamps of vulnerability detection and resolution, then calculate the average duration throughout a specific time frame.
- **Software component health:** Measures the age, reuse frequency, and contribution to technical debt of each software component. Monitoring this metric provides insights into outdated components, development efficiency, and technical debt accumulation. To improve health, prioritize updating or retiring dated components and fostering reusable design patterns. Calculate code age using the difference between the current date and the date of the last update, count the number of projects which depends on the component, and use static code analysis tools to calculate a technical debt score.

  Here is an example formula to calculate technical debt score, which will need to be altered for use within your organization:

  Technical debt score =

  Cyclomatic Complexity + Duplicate Code + Other detectable forms of Software Entropy

Everything as code

Everything as code is a software development practice that seeks to apply the same principles of version control, testing, and deployment to enhance maintainability and scalability of all aspects of the development lifecycle, including networking infrastructure, documentation, and configuration. This practice adds the ability to automate more, leading to faster, more consistent, and more reliable development cycles. By using code for as many use cases as possible, developers can achieve a higher level of quality, reduce the risk of errors, and increase the speed at which they can deploy new features and updates.

Topics

- Indicators for everything as code
- Anti-patterns for everything as code
- Metrics for everything as code

Indicators for everything as code

Adopt a code-centric approach across the development lifecycle for enhanced maintainability, scalability, and automation.

Indicators

- [DL.EAC.1] Organize infrastructure as code for scale
- [DL.EAC.2] Modernize networks through infrastructure as code
- [DL.EAC.3] Codify data operations
- [DL.EAC.4] Implement continuous configuration for enhanced application management
- [DL.EAC.5] Integrate technical and operational documentation into the development lifecycle
- [DL.EAC.6] Use general-purpose programming languages to generate Infrastructure-as-Code
- [DL.EAC.7] Automate compute image generation and distribution

[DL.EAC.1] Organize infrastructure as code for scale

Category: FOUNDATIONAL

Infrastructure as code (IaC) provides consistent and automated infrastructure management capabilities which are important to DevOps adoption. Effectively organizing and scaling IaC within

your organization enhances flexibility, readability, and reusability across multiple teams, while streamlining infrastructure provisioning and maintenance.

When working with IaC files and artifacts, apply modern practices such as modular design for improved management and reuse, and maintain thorough in-code documentation for clarity. Adopt IaC-specific design patterns, like breaking down infrastructure templates into reusable modules. Treat IaC testing with the same rigor as other software, focusing on security risks like excessive privileges or open security groups, while upholding quality standards. Use version control for IaC templates to ensure traceable changes, reliable rollbacks, and efficient sharing across the organization.

You must carefully consider your organization's governance structure when deciding how to implement IaC at scale. Depending on the specific needs, your organization might find one model more suitable than the other, or even adopt a hybrid approach that combines elements of both. The right approach to scaling is dependent on factors such as team dynamics, operating model, application type, and the desired rate of change.

For example, services like AWS Service Catalog and AWS Proton provide distinct methods to distribute and consume secure-by-default software components and IaC in different ways. Service Catalog suits organizations favoring predefined deployment standards and centrally defined resource provisioning, while AWS Proton is ideal for organizations that allow development teams to maintain infrastructure and application autonomy. Some organizations might prefer to adopt a fully decentralized approach, where individual teams provision and manage their own AWS CloudFormation IaC templates. Choose the tools and distribution methods that best support your governance model and business goals.

Related information:

- Infrastructure as code - Introduction to DevOps on AWS
- Infrastructure as Code on AWS - An Introduction
- Accelerate deployments on AWS with effective governance
- Source Control concepts
- Design Patterns
- Amazon's approach to security during development: Octane

[DL.EAC.2] Modernize networks through infrastructure as code

Category: FOUNDATIONAL

The practice of managing networking configurations through code, including network automation, version control, and rigorous testing to ensure quality and stability. Apply DevOps practices to networking systems to streamline network operations, reduce human errors, and speed up network deployments. *Networking as code* enables the predictable and repeatable provisioning of networking components, making infrastructure more modular and less prone to error.

Managing networking components as code requires cultural, process, and tool changes. Shift from a centralized, manual model of network management to a more autonomous model where individual teams can operate independently. Loosely couple networking architectures to create modular components that can be managed, maintained, and scaled individually. Use infrastructure as code (IaC) tools to define network infrastructure and configurations and use development lifecycle capabilities like continuous integration and continuous delivery (CI/CD) for deploying networking changes. Like other systems, networking changes should undergo automated testing to provide assurance that they meet functional, non-functional, and security requirements before deployment.

Often, platform teams manage network components on behalf of individual teams when possible so that all teams do not need to become networking experts. However, for cases where this is not possible, use shared resources or predefined network configuration templates which have embedded best practices and secure defaults. This approach encourages predictable and repeatable provisioning of self-service networking components. Have guardrails in place within the environment to enforce compliance of networking requirements.

Related information:

- NetDevOps: A modern approach to AWS networking deployments
- NetDevSecOps to modernize AWS networking deployments
- Field Notes: Using Infrastructure as Code to Manage Your AWS Networking Environment

[DL.EAC.3] Codify data operations

Category: FOUNDATIONAL

Codifying data operations in a DevOps environment extends the infrastructure as code (IaC) principle to data management, which involves treating database schemas, data transformations, and data pipelines as code. Codifying data operations enables other DevOps capabilities including the use of data management pipelines for data lifecycle management, enforcing quality assurance

and governance standards, providing auditability of changes, and the ability to rollback changes when necessary.

Store database schemas, along with any related procedures, views, and triggers, in version control systems alongside your application code. This enables the ability to track, review, and test schema changes before deploying them to your production environment. To start managing existing data source schemas as code, database migration and event analysis tools like AWS DMS Schema Conversion Tool and Amazon EventBridge can help to infer schemas from existing data sources.

Related information:

- Converting database schemas using DMS Schema Conversion
- Creating an Amazon EventBridge schema
- Using Amazon RDS Blue/Green Deployments for database updates

[DL.EAC.4] Implement continuous configuration for enhanced application management

Category: RECOMMENDED

Configuration as code is the practice of managing and tracking configuration changes as code, providing an audit trail and reducing errors from manual changes. Continuous configuration uses configuration as code to enhance configuration management by allowing configuration changes to be made independently of application code deployments.

Configuration should be separated from application code to allow for independent tracking and management. Use tools designed for managing configurations as code, such as AWS AppConfig, to manage configuration externally from the application. Create fully automated pipelines that perform continuous integration and continuous delivery (CI/CD) based on changes to the configuration code. Just like with application deployment pipelines, these configuration deployment pipelines should run quality assurance tests, followed by deployment in a non-production environment before deploying to production.

It's important to distinguish between static and dynamic configuration types. Static configurations do not change during the software's runtime and are specific to each environment. Dynamic configurations can be adjusted at runtime without downtime. Feature flags are examples of dynamic configurations that can be used to control which features are enabled per environment to decouple release from deployment. Operational configurations, such as log level, throttling thresholds, connection/request limits, alerts, and notifications, can be static or dynamic depending

on the use case and need to be managed. Application modes, which toggle the application to run as either *development*, *test*, or *production*, are typically considered to be static configuration that is set at startup and do not change.

General use cases for continuous configuration include application integration tuning, feature toggling, allowing access to premium content through allow lists, and addressing operational issues and troubleshooting. To manage your configurations effectively, establish a routine to prevent configuration bloat. While it can seem tempting to externalize as many variables as possible, an excessively complex configuration file can lead to confusion and errors. Carefully evaluate the necessity, frequency of change, and runtime requirements of each value to decide if it should be included as dynamic configuration.

For large-scale deployment of configuration as code, a Dynamic Configuration Pipeline is recommended. This allows centralized management of the entire workload configuration and its components across all environments. It ensures that all configurations are version-controlled, adhere to quality assurance and code review processes, and is capable of progressively deploying configuration changes and performing rollbacks as necessary to minimize system disruptions.

Continuous configuration is beneficial in DevOps environments, as it improves operational efficiency and scalability. However, not every system requires the complexity associated with continuous configuration. Therefore, each workload should be evaluated depending on architecture choice, team preferences, and service level objective requirements.

Related information:

- AWS Cloud Adoption Framework: Operations Perspective - Configuration management
- AWS AppConfig
- Continuous configuration

[DL.EAC.5] Integrate technical and operational documentation into the development lifecycle

Category: RECOMMENDED

Integrating documentation and code involves creating, maintaining, and publishing documentation using the same tools and processes used for application development. With this approach, changes to systems should be immediately reflected in documentation, reducing the risk of discrepancies between system behavior and documentation. By making documentation part of the development lifecycle, it becomes a living document that evolves with the system over time.

Documentation should be stored in a versioned source code repository and written in a machine-readable markup language, such as Markdown. The documentation can be made directly accessible through the repository or through knowledge sharing tools capable of rendering the markup language, like Git-based wikis, static site generators, or directly in developers' integrated development environments (IDEs).

Code should include clear, insightful comments and commit messages should be structured using a machine-readable specification, such as Conventional Commits. This information can be used as a source to generate detailed documentation and change logs using tools specific to the programming language and platforms being used. Many of these tools can create API references, class diagrams, or other technical documents from inline comments in your source code, ensuring the documentation is always in line with the most recent changes. Automate this process by adding a stage to the deployment pipeline to generate documentation with every change to a main, releasable branch.

This approach is not only limited to documenting code, but also can be used to store operational documentation like incident response procedures, disaster recovery plans, training material, and onboarding processes. While some aspects of these documents still likely require manual effort to create, the benefits of incorporating these documents into the development lifecycle include enforced reviews of changes, ability to write tests to suggest updating documentation when changes are significant or made to important components, and versioning the documents for auditability.

Related information:

- AWS Well-Architected Reliability Pillar: REL12-BP01 Use playbooks to investigate failures
- Write the Docs: Docs as Code
- One AWS team's move to docs as code
- AWS Incident Response Playbook Samples
- Using code as documentation to save time and share context
- DocFx
- How to build an automated C# code documentation generator using AWS DevOps

[DL.EAC.6] Use general-purpose programming languages to generate Infrastructure-as-Code

Category: RECOMMENDED

Developing infrastructure as code (IaC) using general-purpose programming languages aligns closely with modern software development practices and DevOps principles. IaC has traditionally been implemented as predefined templates modeled through domain-specific languages using markup languages like JSON or YAML. During deployment, these templates are provided parameters which specify environment-specific details. While parameterized templates are still a best practice for traditional IaC templates, this approach can become difficult to develop, troubleshoot, and manage as infrastructure and environments become more complex.

Using general-purpose programming languages changes how we develop, manage, and deploy IaC. It is no longer a collection of parameterized templates, but instead infrastructure is written in common programming languages such as TypeScript, Python, or Java, and can be treated the same as other code throughout the development lifecycle. Instead of providing environment-specific configuration during deployment, tools like AWS Cloud Development Kit (AWS CDK) generate separate templates for each environment using configurations defined in source code. This provides a more predictable, consistent, and reproducible deployment process.

Transitioning to using general-purpose programming languages for IaC can also change how you govern IaC at scale. For example, AWS CDK includes the ability to consume, publish, and version software components called AWS CDK constructs through private artifact registries or the open-source Construct Hub registry.

Related information:

- Best practices for developing and deploying cloud infrastructure with the AWS CDK
- CDK for Terraform (CDKtf)
- CDK for Kubernetes (CDK8s)
- AWS Solutions Constructs
- Artifact Repository - AWS CodeArtifact
- Infrastructure IS Code with the AWS CDK
- Best practices for using the AWS CDK in TypeScript to create IaC projects
- Adding the "AWS CDK bootstrap" action in Amazon CodeCatalyst

[DL.EAC.7] Automate compute image generation and distribution

Category: OPTIONAL

The management of compute images, including containers and machine images, can be optimized and made more reliable through a code-driven approach. Compute images generally include a base image, libraries, environment variables, application code, and configuration files. Similar to other forms of infrastructure as code (IaC), compute images can be codified, stored in version control systems, tested, and distributed as part of the development lifecycle.

Establish automated pipelines for building, testing, and distributing compute images. The build stage creates the image based on its code definition, the *test* stage validates the functionality and security compliance of the image, and the *distribution* stage ensures the image is readily available for teams to use in their environments and workloads. Updates to the images should be automated, accounting for software patches, security enhancements, and other modifications.

Given the diverse range of applications and infrastructure requirements, especially when using managed cloud-based services, not all organizations or workloads necessitate using dedicated compute images or codifying them.

Related information:

- Amazon EC2 Image Builder
- AWS Deployment Pipeline Reference Architecture
- What is AWS App2Container?

Anti-patterns for everything as code

- **Checking in secrets:** Storing sensitive data, such as API keys, passwords, or other secrets, directly in the code base or version control system is a critical security vulnerability. Checking in secrets exposes sensitive credentials to anyone with access to the repository and, if the repository is public, to the world. Instead, use management tools or services to store and retrieve secrets securely. These tools can integrate with deployment pipelines and systems during runtime to provide secrets only when necessary, ensuring they remain confidential and are not inadvertently exposed.
- **Manual modifications to infrastructure:** Making manual changes to infrastructure can be time consuming and error prone, leading to inconsistencies that can be difficult to troubleshoot and resolve. Actively prevent users from making manual changes to environments and workloads to ensure consistent and reliable deployments.
- **Outdated or incomplete documentation:** Ignoring documentation or treating it as an afterthought can lead to knowledge gaps, misunderstandings about system behavior, and

misleading users. As the system changes over time, documentation needs to be continuously updated to align with the current system state.

- **Ignoring configuration drift:** Failing to track and manage changes to your system's configuration can result in configuration drift, where the actual configuration state deviates from the desired state. Overtime this can lead to system instability, security vulnerabilities, and operational inefficiencies. Use continuous configuration management practices and automated governance capabilities to keep configurations in a known and secure state.
- **Bypassing code review and testing:** Failing to review and test IaC changes, including data, documentation, configuration, and networking components is an anti-pattern that can lead to data inconsistencies, data loss, and system instability. It's important to apply the same quality assurance practices to IaC as you would to application code.
- **Inefficient IaC development practices:** Treating IaC differently from application code, especially by not using version control, diminishes developer experience and increases deployment risk. By not versioning IaC files, teams lose the ability to track changes over time, identify when specific changes were made, or correlate infrastructure changes with system behavior. Additionally, storing large, monolithic IaC files makes development and management of IaC more complex, as intertwining components make it challenging to identify specific sections and understand changes being made. Mitigate these challenges by segmenting IaC into modular units consistent with the system's architecture and maintain them within version control systems. Using general-purpose programming languages when developing IaC can further simplify managing IaC like other application code.
- **Monolithic network architectures:** Designing a network where different components are tightly coupled leads to reduced flexibility and increased complexity. This pattern can make troubleshooting and scaling particularly challenging, as changes in one component may inadvertently impact others. Instead, create a modular network design expressed through multiple, well-organized IaC files where components are loosely coupled and can be individually managed, maintained, and scaled.

Metrics for everything as code

- **Infrastructure code coverage**: The percentage of infrastructure components managed by infrastructure as code (IaC) compared to the total number. High infrastructure code coverage implies improved manageability, reproducibility, and automation capabilities for systems. Calculate by dividing the number of infrastructure components managed as code by the total number of infrastructure components and multiply by 100 to get the percentage.

- **Configuration drift rate**: The percentage of infrastructure components drifting from their intended configuration over time. Configuration drift can introduce security vulnerabilities, performance issues, and general system instability. Implement configuration management tools, routinely run drift detection processes, and automate corrective actions to improve this metric. Monitor infrastructure configurations regularly and calculate the drift rate by dividing the number of drifted configurations by the total number of configurations and multiplying by 100 to get the percentage.
- **Documentation update frequency**: The average frequency that documentation is updated relative to code or system changes. Stale or out-of-date documentation can lead to operational inefficiencies, onboarding issues, and system misuse. This metric can be improved by defining documentation as code, automating the release of documentation through a delivery pipeline, and prompting developers to update docs as part of the development lifecycle. Track the number of documentation changes over a set time frame and compare it to the number of system changes in that same time frame.
- **Time to provision infrastructure**: The time taken to provision a new infrastructure component or environment using IaC. A key advantage of using code to define infrastructure is improved change lead time and deployment frequency through the reduction of inconsistent and manual infrastructure provisioning practices. Use time-stamped logs to measure the time interval between the initiation and completion of infrastructure provisioning tasks.
- **Mean time to recover (MTTR)**: The average time taken to restore a system after a failure. Ensure IaC is testable, automate infrastructure provisioning, and maintain configuration consistency across deployments. Monitor downtime incidents and compute the average recovery time over a designated period.

Code review

Code reviews serve as a mechanism for light and frictionless change management in a DevOps environment. They enforce separation of duties which helps ensure that multiple people are involved in approving and merging changes to the code base. Implementing code reviews helps organizations streamline change processes, enhance software quality, and create a culture of shared responsibility, significantly improving the reliability of the software being built.

Topics

- Indicators for code review
- Anti-patterns for code review

- Metrics for code review

Indicators for code review

Embed a lightweight, frictionless change review process into the development lifecycle. This capability enables separation of duties to be performed to ensure multiple perspectives are involved in approving changes.

Indicators

- [DL.CR.1] Standardize coding practices
- [DL.CR.2] Perform peer review for code changes
- [DL.CR.3] Establish clear completion criteria for code tasks
- [DL.CR.4] Comprehensive code reviews with an emphasis on business logic
- [DL.CR.5] Foster a constructive and inclusive review culture
- [DL.CR.6] Initiate code reviews using pull requests
- [DL.CR.7] Create consistent and descriptive commit messages using a specification
- [DL.CR.8] Designate code owners for expert review

[DL.CR.1] Standardize coding practices

Category: FOUNDATIONAL

Coding standards promote uniformity and consistency across the organization. Individual teams can also extend this standard to adopt specific practices that align with the team's preferences. Having standards not only helps ensure consistency across distributed teams, but can also make code reviews more efficient, support knowledge sharing, and lead to faster issue resolution.

Identify or develop coding standards that align with the primary programming languages used across the organization. This does not mean that other languages cannot be used, but does lead to a structured approach to development for new teams and new employees. The coding standards are meant to facilitate error detection, improve code readability, simplify maintenance, and enhance the overall efficiency of builders, not prevent innovation.

These standards can be codified into linters and code quality tools to improve developer experience. This approach provides fast feedback to developers and evaluate their adherence

to the standards automatically. Hold training sessions for developers on these standards, store them in centralized knowledge sharing spaces, and create mechanisms to gather feedback to continuously improve the standard over time. We recommend getting started by adopting industry-specific standards, such as the Secure Coding Guidelines for Java SE, Conventional Commits for Git, or the PEP8 styling guide for Python.

[DL.CR.2] Perform peer review for code changes

Category: FOUNDATIONAL

A peer review process for code changes is a strategy for ensuring code quality and shared responsibility. To support separation of duties in a DevOps environment, every change should be reviewed and approved by at least one other person before merging. Once approved, a pipeline with sufficient access will deploy the change.

Most version control systems support protection rules enforcing certain workflows, like requiring at least one peer review, before merging into designated branches. Use these rules to enforce this workflow and provide assurance that all code changes adhere to this mandatory review process.

Incorporating pair programming, where two programmers collaboratively work side-by-side or through screen sharing, is method of peer review. By integrating this approach, reviews can be integrated into the development lifecycle earlier—while the code is being written, reducing the time taken to identify and fix issues. This accelerates review timelines, reduces the introduction of bugs or issues, promotes knowledge sharing, and creates a culture of quality and continuous improvement.

Some companies require multiple reviewers, or require more proof than just pair-programming to adhere to compliance requirements. Pick a code review process that works for your organization, and enforce it through policies, processes, and technology.

Related information:

- AWS Well-Architected Security Pillar: SEC11-BP04 Manual code reviews
- Team Collaboration with Amazon CodeCatalyst
- Code review

[DL.CR.3] Establish clear completion criteria for code tasks

Category: FOUNDATIONAL

A clear definition of done ensures that developers understand the requirements of their task, can consistently meet those requirements, and that reviewers have a sense of what they are reviewing. It provides the team with shared clarity of purpose for each change that they will be making to the code base.

To implement a clear definition of done, initiate discussions among all team members during the design phase to identify and agree on the criteria that should be included. The done criteria should include the types of testing that need to be done (like functional, non-functional, or security tests), any required documentation (like code comments or user manuals), and the standards the code needs to meet (such as performance, availability, or team style guides).

Once these criteria are defined and agreed upon, document them, and make this definition of done available and visible to all team members. It should be used as a checklist during the code review process to ensure that all changes meet the established criteria. Having a clear definition of done can streamline the review process and reduce the number of issues that need to be addressed in later stages of the development lifecycle.

[DL.CR.4] Comprehensive code reviews with an emphasis on business logic

Category: FOUNDATIONAL

Use automated code review tools to detect potential issues before they are merged into the code base. This approach provides fast feedback to developers to fix issues before a manual review takes place. This also frees manual reviewers from needing to review for trivial issues like code style inconsistencies or syntax errors. Reviewers can instead focus on more on complex aspects of the code such as business logic, maintainability, and scalability, which may be difficult to automate. This accelerates the review process, reduces the feedback loop, and promotes rapid iteration.

Start by identifying the types of issues that can be automated (like code formatting, syntax errors, and potential security vulnerabilities). Then, choose suitable tools that fit your code base and your team's needs. Integrate these quality assurance (QA) tools into your development lifecycle so that the checks are automatically run when code changes are being developed and merged.

Using automated code review tools is recommended for improved efficiency and consistency, but is not absolutely required for code reviews as DevOps teams can function and conduct manual code reviews without them.

Related information:

- Create code reviews in Amazon CodeGuru Reviewer

- Automate code reviews with Amazon CodeGuru Reviewer

[DL.CR.5] Foster a constructive and inclusive review culture

Category: FOUNDATIONAL

Code reviews should be respectful and collaborative interactions that cultivate a positive and inclusive culture. Good code reviews involve asking open-ended questions, suggesting alternatives, and assuming good intentions. Reviews should be empathetic and kind, recognizing the effort put into the code changes and promoting positivity.

The tone and approach of code reviews can greatly impact the efficiency of the process, team morale, and ultimately the quality of the product. A positive and inclusive review culture encourages more open discussion, facilitates knowledge sharing, and can lead to improved code quality.

To implement a positive and inclusive review culture, teams should establish clear guidelines on the expectations for code reviews, including language use and constructive feedback. Regularly reinforce these expectations through team meetings and training. Encourage team members to focus on the code and not the coder, to be respectful and patient, and to frame suggestions as questions or alternatives rather than absolute critiques. Use the available escalation paths and mutually agreed upon team guiding principles to quickly resolve team differences and act as tie breakers during disagreement.

[DL.CR.6] Initiate code reviews using pull requests

Category: RECOMMENDED

Pull requests are a method of integrating changes from one branch of a repository into another. They can be used to propose, review, and integrate changes from a feature branch into the main releasable branch. Modern branching strategies, including GitHub flow and trunk-based development, support this workflow to initiate code review.

A pull request workflow is recommended for organizations and teams which have enhanced code review requirements. This workflow could include requiring multiple peer reviewers, or enforcing that reviews must take place before code is integrated into the main releasable branch. We recommend adopting trunk-based development paired with a pull request workflow utilizing short-lived feature branches. This method uses feature branches solely to trigger code review processes through a pull request workflow. These short-lived feature branches should not be used as a source for code deployments.

There should be clearly defined steps to standardize creating, reviewing, and merging pull requests. Store these guidelines in a shared, easily accessible location to ensure all team members understand the process. The guidelines should include:

- **Useful descriptions and titles:** The pull request descriptions should guide the reviewer through the changes, grouping related files and concepts. A well-crafted title gives a high-level summary of the changes, providing the reviewer with the necessary context.
- **Descriptive commit messages:** Each commit message should clearly communicate what changed and why. This can make auto-generated pull requests more useful, provide a bullet-point summary of the changes, and aid reviewers who read the commits along with the diff.
- **Inline comments:** Leaving comments on the pull request can guide the reviewer through the changes. These comments can provide the reviewer with the necessary context, such as files that were simply re-indented or files where the main bulk of changes occurred.
- **Visual cues:** For user interface (UI) changes, consider including screenshots, GIFs, or videos. Visual representations can make it easier for reviewers to understand the changes.

Pull request workflows are recommended, but not strictly required for DevOps adoption. Some organizations and smaller teams may choose to strictly follow trunk-based development practices and commit changes directly to the main releasable branch. In this workflow, code reviews are performed through pair programming or initiated through custom post-commit processes. Choose the right method for performing code review based on your organization requirements and individual team preferences.

Related information:

- Reviewing a pull request - Amazon CodeCatalyst
- Team Collaboration with Amazon CodeCatalyst
- Code review

[DL.CR.7] Create consistent and descriptive commit messages using a specification

Category: RECOMMENDED

Use a well-documented specification, descriptive commit message format that clearly explain what changes were made and why. Clear and consistent communication support the fast-paced, iterative nature of DevOps. Consistent commit messages improve collaboration, make it easier to track and understand changes, aid in debugging, and can be used to automatically generate change logs.

Adopt a specification, such as Conventional Commits, to indicate code features, fixes, and breaking changes through commit messages. Ideally, this would be enforced using pre-commit hooks and the developer experience improved through IDE integrations. Training and documentation can also be used to educate developers on the importance and use of this specification. If done consistently, this information could be used to automatically generate legible change log records for non-developer consumers and users of the system.

Adopting a commit specification is recommended as it greatly enhances communication and collaboration by clearly documenting the changes being made and why they are important to the overall system. This can significantly boost efficiency and transparency but isn't required as DevOps teams can function without it.

Related information:

- Conventional Commits

[DL.CR.8] Designate code owners for expert review

Category: OPTIONAL

A code owners process assigns a designated owner, usually the person or team with the most knowledge or expertise, to each part of the code base. In a DevOps environment, this helps ensure that there is an expert reviewer available for specific or complex parts of the system at all times.

To implement a code owners process, determine who the code owners should be based on expertise and distribute the ownership equally amongst the team to avoid bottlenecks. You can use features in version control systems that automatically assign code owners to review code changes in their area of expertise. One example of this would be to use a `CODEOWNERS` file stored along with the code in the repository. This file defines individuals or teams that are responsible for code in a repository.

While this practice is optional and not beneficial for all organizations, it can be particularly useful for larger teams or those with complex, distributed systems as it provides an additional layer of control and can prevent potential issues from going unnoticed if all reviewers are not equally experienced with a specific or complex part of the code base.

Related information:

- About code owners

Anti-patterns for code review

- **Infrequent code reviews**: Skipping or only occasionally performing code reviews misses opportunities to catch errors early and can lead to isolated development work. For higher quality code, faster error detection, incorporation of additional perspectives, and sharing knowledge, regularly schedule code reviews and make them a mandatory step before merging.
- **Excessive required reviewers**: Overloading the review process by involving too many reviewers can lead to bottlenecks and unwarranted delays. Define a practical number of reviewers based on the complexity and criticality of the code changes.
- **Lack of automated feedback**: Neglecting automated tools extends the review duration by requiring peers to focus on trivial issues rather than complex logic. Automated quality assurance and feedback mechanisms can help ensure consistent and efficient code review by catching common issues and providing feedback before manual review. Incorporate automated code review tools to complement manual, human-driven reviews. While automated tools can help catch certain issues, relying too heavily on them can lead to a false sense of security and miss issues that require human judgment. Find a balance that meets the needs of the organization and team preferences.
- **Large batch reviews**: Combining multiple code changes into a single request clutters and lead to a longer review cycle. Including multiple changes, especially if they are unrelated, into a single review makes it more difficult to identify and address issues with individual changes. Submit smaller, more focused pull requests to keep reviews concise and enable a faster feedback loop.
- **Unconstructive reviews**: Code reviews which are lead with harsh or hostile tones or include unhelpful or vague feedback can create an environment leading to unconstructive reviews. These unconstructive reviews demoralize developers, prevent open dialogue, and impede development progress. Maintain a positive review culture by training reviewers to offer clear, constructive feedback, emphasizing the importance of specificity and positive tone. Implement healthy escalation methods and establish team norms to address and resolve disputes as they arise.
- **Lack of action on findings**: Code review findings that are not acted upon or followed up on can lead to missed opportunities for improvement and perpetuate the same issues in future code changes. Unaddressed feedback makes the review process redundant. Ensure a system where feedback is tracked, and necessary actions are taken promptly.

Metrics for code review

- **Review time to merge (RTTM):** The duration from the start of the review process to the merging of code. This metric can indicate gaps in the review process and can be improved by streamlining reviews, provide timely feedback, and automating trivial checks. Monitor the timestamp of the start of review and the timestamp of code merge.
- **Reviewer load:** The number of open pull requests assigned to each reviewer. High reviewer loads can lead to bottlenecks and inefficiencies in the code review process, while low reviewer loads when paired with a high review time to merge can indicate that the team is not focused enough on reviewing changes. This metric can be improved by rebalancing pull request assignments, assigning code owners, and adding more reviewers or resources. Count the number of pull requests per reviewer periodically to track this metric.
- **Code ownership health:** This metric evaluates how well the codebase is covered by designated code owners, ensuring that there are enough domain experts to review relevant sections of the code. It uses the current number of code owners listed in the `CODEOWNERS` file and compares it against a desired benchmark target. Calculate the benchmark target by dividing the number of pull requests by the average number of pull requests a code owner can handle while accounting for actual code owner availability. Compare the current count of code owners to this derived benchmark target.
- **Merge request type distribution:** Distribution of merge requests based on their nature, such as new features, enhancements, maintenance, or bug fixes. This metric provides insight into the team's focus and can help to plan and allocate resources effectively. Categorize each pull request, ideally using a common commit convention, and monitor the distribution on a regular basis, such as monthly or quarterly.
- **Change failure rate:** The rate at which code changes lead to errors or bugs in the system after being merged. A high failure rate might indicate issues in the review or testing processes, while a low rate suggests effective review and quality control. Calculate the rate by dividing the number of post-merge failures by the total number of merges made during a specific time frame, such as monthly or quarterly.

Cryptographic signing

Cryptographic signing in the development lifecycle authenticates the origins and verifies the integrity of software components. Through the use of digital signatures, it safeguards software builds and deployments against unauthorized changes and potential threats from malicious actors. By leveraging cryptographic signing, you can establish a secure software supply chain,

improve transparency in the build and delivery process, and reliably distribute verifiable software components at scale.

Topics

- Indicators for cryptographic signing
- Anti-patterns for cryptographic signing
- Metrics for cryptographic signing

Indicators for cryptographic signing

Authenticate and verify software component origins and integrity, ensuring a secure software supply chain.

Indicators

- [DL.CS.1] Implement automated digital attestation signing
- [DL.CS.2] Sign code artifacts after each build
- [DL.CS.3] Enforce verification before using signed artifacts
- [DL.CS.4] Enhance traceability using commit signing

[DL.CS.1] Implement automated digital attestation signing

Category: RECOMMENDED

Digital attestations serve as verifiable evidence that software components were built, tested, and conform to organizational standards within a controlled environment. Signatures associated with each attestation can be verified to ensure that the component has not been tampered with and originated from a trusted source. Generating attestations throughout the development lifecycle provides a method of ensuring software quality, origin, and authenticity.

Embed automated tools into the deployment pipeline to produce digital attestations. Create an attestation for each action you want to create proof for, such as a test being run, software being packaged, or even manual approval acceptance steps. Sign these attestations using symmetric or asymmetric keys. Follow metadata frameworks such as in-toto for best practices for formatting attestations to include metadata about the software, the build environment, and the authoring party. Store attestations either with build artifacts in a repository or within governance tools for deeper analysis.

Related information:

- Software attestations

[DL.CS.2] Sign code artifacts after each build

Category: RECOMMENDED

Code signing is the process of attaching a digital signature to build artifacts like binaries, containers, and other forms of packaged code to enable verifying its integrity and authenticity. Signing code artifacts minimizes risk of using or distributing tampered or counterfeit software.

Cryptographically sign code artifacts during the build process. Ideally this occurs after testing and before publishing to production. Follow best practices for timestamping while signing. Timestamping provides a verified date and time of the signing, serving as evidence that the code artifact existed and met the signature criteria while the certificate was still valid. To safeguard operations, ensure that the validity of the signed code artifact is recognized even after the signing certificate itself has expired.

Store signatures in a location accessible to users and systems that need to verify signed code artifacts. When using Open Containers Initiative (OCI) compliant artifact registries, it is encouraged to store digital signatures alongside the build artifacts being signed. This enables a consolidated retrieval process and allows verification systems to easily locate and validate signatures. Just as with artifacts, signatures can accumulate over time. Implement a lifecycle policy that archives or deletes older signatures that are no longer needed to help manage storage costs.

After a signature has been stored, it should be immutable so that the signature cannot be tampered with or replaced. Use fine-grained access controls to ensure that only authorized entities can push or modify artifacts and their corresponding signatures. Regularly back up your digital signatures. Having a backup ensures you can still verify the integrity and authenticity of your artifacts in the event of storage failures. All access and operations on stored signatures should be logged to support forensic analysis and to adhere to compliance requirements.

Implement cryptographic signing of artifacts during the build process. Ideally this occurs after testing and before publishing to production. This helps ensure the integrity of the artifacts and confirms their authenticity. We recommend using a managed service like AWS Signer to reduce the complexity that comes with managing public key infrastructure. Refer to AWS Signer workflows for guidance that fits your use case.

For more control over the signing process or for complex use cases, you can create and manage your own code signing platform using Public Key Infrastructure (PKI). While this approach offers precise control, it requires consistent upkeep and adherence to best practices. AWS Private Certificate Authority is a managed private CA service that helps you manage the lifecycle of your private certificates easily, without the investment and ongoing maintenance costs of operating your own private CA.

Related information:

- AWS Well-Architected Sustainability Pillar: SUS05-BP03 Use managed services
- Using AWS Signer workflows
- Configuring code signing for AWS SAM applications - AWS Serverless Application Model
- Security Considerations for Code Signing
- Code signing using AWS Certificate Manager Private CA and AWS Key Management Service asymmetric keys

[DL.CS.3] Enforce verification before using signed artifacts

Category: RECOMMENDED

Before using code artifacts, the cryptographic signature should be inspected and validated. This verification step enforces trust and security within the development lifecycle, ensuring that software remains unchanged before it is used or deployed.

Strictly enforce verification of cryptographic signatures each time a code artifact is used or deployed. Use a managed signing service like AWS Signer or the public key from your organization's trusted Certificate Authority (CA) for signature verification. Automate the verification process where possible, as manual checks can be error-prone and may not be strictly enforced. Some examples of this are integrating signature verification into the deployment pipeline, enforcing verification at the registry level as artifacts are distributed, or using the Kubernetes admission controller to verify each container image as they are pulled.

Related information:

- Security Considerations for Code Signing
- Configuring code signing for AWS Lambda
- Kyverno extension service for Notation and the AWS signer
- Announcing Container Image Signing with AWS Signer and Amazon EKS

[DL.CS.4] Enhance traceability using commit signing

Category: OPTIONAL

Commit signing involves attaching a digital signature to code commits, certifying the integrity of changes and the identity of the committer. While not universally adopted by all organizations, commit signing enhances trust and traceability as developers make code changes, making it easier to track the origin of changes and ensure their authenticity.

Have developers sign their code changes when submitting to version control using personal private keys from tools like GPG. Developers should be encouraged to sign both commits and tags with their private keys. This can be particularly valuable for open-source projects or where code originates from diverse sources.

For this approach to be effective in practice, developers require an understanding of certificates and using them for signing. Developers must ensure that their private keys remain confidential, taking measures to store them securely and avoid potential exposure. They also should be trained to recognize signs of key compromise, such as unexpected commits. When compromise is detected, the associated key should be revoked immediately to mitigate potential risks.

Related information:

- Signing Commits
- The GNU Privacy Guard

Anti-patterns for cryptographic signing

- **Ignoring key compromise**: Key compromise can severely affect the integrity of the software. When cryptographic signing keys are suspected to be compromised, immediate steps should be taken to revoke and replace them to maintain the security and trust in the code base. Regular key rotation prevents prolonged unauthorized access from a compromised key and strengthens overall security.
- **Reuse of signing keys across projects**: Sharing the same signing keys across multiple projects also can affect the integrity of the software. If a key is compromised, all projects signed with that key become potential targets. To mitigate the risk, the best practice is to use distinct keys for different projects or workloads.
- **Incomplete signature verification**: Trusting incorrectly signed code can introduce vulnerabilities. Skipping the validation of cryptographic signatures for third-party libraries or dependencies

jeopardizes application security. You should consistently validate signatures to maintain code integrity and authenticity. It is equally important to assess certificate attributes such as validity periods and key usage. Avoid relying exclusively on manual checks for signature validation; human oversight in this process is a risk. Using automation in the verification process upholds consistent security checks and minimizes errors.

- **Overlooking timestamp validation:** Ignoring timestamp validation can lead to situations where code is deemed untrustworthy due to an expired certificate, even if the code itself is legitimate and unaltered. This may result in unintentional outages or service disruptions, as systems might refuse to run or integrate with the signed artifact. Proper timestamp validation ensures that certificates were valid at the time the code was signed, preventing unnecessary disruptions due to expired certificates while still maintaining trust in the code's integrity.
- **Avoid certificate pinning**: Hard-coding certificate information within your software, often called certificate pinning, might initially seem like an extra layer of security by tightly coupling your code to a certificate. However, it is an anti-pattern as it makes your software inflexible and brittle. If the pinned certificate needs to be replaced or updated, it will require a software update, which might not be feasible, especially for critical systems or widely distributed applications. Instead, rely on proper certificate validation processes and maintain a flexible and agile certificate management system.

Metrics for cryptographic signing

- **Number of unsigned releases**: Measures the number of releases without cryptographic signatures. This goal of this metric is to reduce this percentage, reflecting increased compliance with cryptographic practices across the organization. Track it by comparing the number of unsigned releases to the total releases over a specific time frame.
- **Number of expired certificates used**: Assesses how often expired certificates are used. Using expired certificates suggests potential operational oversights in certificate renewal and management. Improve this metric by routinely auditing and updating certificate management processes and automating renewal reminders. Track this metric by logging and counting releases where expired certificates were used.
- **Time to revoke a compromised key**: The duration between the detection of a key compromise and its revocation. This measures efficiency of the incident response process and how quickly the organization can react to potential threats to key compromise. Aim to minimize this duration, as a shorter time indicates a more agile and responsive incident response process. Monitor this metric by calculating the average time between the occurrence of key compromise and key revocation.

- **Time to sign:** This amount of time it takes to sign a code artifact. If it takes too long to sign an artifact, it could be a bottleneck in the deployment or release process. Aim for a consistently short duration, which indicates an optimized and streamlined signing workflow. Measure by averaging the time taken for signing across the entire organization in a given time frame.
- **Time to verify**: Measures the duration required to verify the cryptographic signature of a code artifact. It ensures the verification process is efficient and doesn't become a bottleneck. Optimize by streamlining the verification procedure and addressing technical inefficiencies. Track by calculating the average time taken to verify across all signatures in a given period.

Continuous integration

Continuous integration (CI) is a software development practice where developers make regular, small alterations to the code and integrate them into a releasable branch of the code repository. The newly integrated code is autonomously built, tested, and validated in a consistent and repeatable manner. CI allows developers to receive feedback swiftly, identify potential issues in the early stages of the development lifecycle, and address them before they escalate in complexity and cost.

Topics

- Indicators for continuous integration
- Anti-patterns for continuous integration
- Metrics for continuous integration

Indicators for continuous integration

Regularly integrate small code changes into a releasable branch, ensuring fast feedback and early problem detection. Streamline builds and tests using automated pipelines to reduce downstream problems and increasing code quality.

Indicators

- [DL.CI.1] Integrate code changes regularly and frequently
- [DL.CI.2] Trigger builds automatically upon source code modifications
- [DL.CI.3] Ensure automated quality assurance for every build
- [DL.CI.4] Provide consistent, actionable feedback to developers
- [DL.CI.5] Sequence build actions strategically for prompt feedback

- [DL.CI.6] Refine integration pipelines with build metrics
- [DL.CI.7] Validate the reproducibility of builds

[DL.CI.1] Integrate code changes regularly and frequently

Category: FOUNDATIONAL

Working in small batches, characterized by regular, small changes to a code base, enhances software delivery performance. It reduces the time to receive feedback on changes, which is required to enable continuous integration. This way of working is an improvement over traditional phased development approaches, which often leads to delayed feedback due to large batches of work. By making smaller, more frequent changes, teams can uncover and fix bugs earlier in the development lifecycle, simplifying the process of updating, testing, and releasing software.

Features should be broken down into independent work units that align with the agile INVEST checklist. Splitting features into small increments of value, ramping up the frequency of deployment, and practicing Test Driven Development (TDD) all contribute to ensuring small batch sizes. Developers should strive to integrate multiple small, releasable changes to the code base at least once per day. Techniques like dark launching, branch by abstraction, and feature flags allow incomplete features to be integrated in a reversible way without impacting end users.

Working in small batches requires discipline and commitment, but leads to improvements in speed, security, collaboration, and code base consistency. In mature teams, developers commit changes multiple times per day and merge code frequently to prevent accumulating large changes. These teams yield better collaboration and success in maintaining an up-to-date, releasable version of the code base.

Related information:

- What is continuous integration and continuous delivery/deployment?
- What does INVEST Stand For?
- Testing software and systems at Amazon: Continuous Integration and Deployment

[DL.CI.2] Trigger builds automatically upon source code modifications

Category: FOUNDATIONAL

Continuous integration (CI) tools should be configured to regularly monitor the source code repository for any changes. Alternatively, set up the source code repository to send an event upon

each commit. This implementation creates an environment where developers can focus on coding and commit their changes, leaving the system to handle building, testing, and deploying the application.

Having this process in place aligns with the continuous integration principle of *failing fast*. It offers immediate feedback on the impact of changes, whether they cause a minor regression or a major bug, allowing for prompt correction. If a build fails, it becomes immediately visible to the team. Fixing a broken build is then prioritized, fostering a culture of discipline and continuous improvement. This approach minimizes the risk of integration conflicts and bugs while reducing the likelihood of unexpected outcomes that can arise from manual processes or irregular updates. It also streamlines the development process, promotes productivity, and contributes to delivering a higher-quality outcome.

Related information:

- Amazon CodeCatalyst
- Building the pipeline
- Deploy container applications in a multicloud environment using Amazon CodeCatalyst

[DL.CI.3] Ensure automated quality assurance for every build

Category: FOUNDATIONAL

As code changes become more frequent in a DevOps environment, it becomes important to reduce the time it takes to get feedback on those changes. Adding automated quality assurance (QA) tests into the continuous integration pipeline enables rapidly validating changes and receiving fast feedback.

Add stages to the pipeline which run pre-deployment checks to validate that code changes work alongside the existing code base. These checks should automatically trigger functional, non-functional, and security tests against the integrated code base and build artifacts.

Breaking-the-build, which stops the integration pipeline process due to test failures, is a powerful feedback mechanism. However, it should be used judiciously. Reserve breaking-the-build for critical issues, such as actual build failures, high severity security findings, or non-negotiable compliance findings, that demand immediate developer attention. Overuse can disrupt the continuous flow of development, leading to unforeseen delays, bottlenecks, and poor developer experience. Instead, continue to provide feedback to developers in tools they already use, such as IDEs, chat clients, or email, and let them decide if they should stop the process.

It is often more practical to automate enforcement of quality assurance findings as part of the continuous delivery process. This allows enforcement to be objectively targeted based on the environment to which the build is being deployed into. Have an exception mechanism and escalation plans prepared that developers can use if the continuous integration or continuous deployment prevent deployments which they do not agree with.

Related information:

- AWS Well-Architected Reliability Pillar: REL08-BP02 Integrate functional testing as part of your deployment
- AWS Well-Architected Security Pillar: SEC11-BP02 Automate testing throughout the development and release lifecycle
- Testing stages in continuous integration and continuous delivery
- Amazon's approach to high-availability deployment: Release guidance lifecycle
- Testing software and systems at Amazon: Continuous integration and deployment
- The Amazon Software Development Process: Automated Testing

[DL.CI.4] Provide consistent, actionable feedback to developers

Category: FOUNDATIONAL

To identify and address issues as quickly as possible, it's important that developers receive consistent and actionable feedback, regardless of the technologies and tools being used. This consistency streamlines the process of addressing failures across diverse development environments, contributing to more efficient DevOps practices. Implement this by configuring your CI pipeline to send automatic failure notifications, offering clear, actionable resolution guidance.

Any failures in the process should send feedback to the developer automatically, describing the failure clearly with actionable guidance for resolution. Feedback mechanisms should be tailored to fit within tools already used by developers, such as IDEs, chat clients, or email, reducing the learning curve and aiding early problem detection.

[DL.CI.5] Sequence build actions strategically for prompt feedback

Category: RECOMMENDED

By optimizing the sequence of actions or tasks in your continuous integration pipeline, feedback can be timely, allowing developers to quickly react and make necessary changes. This practice reduces the risk of delayed releases due to late detection of issues.

Initiate long-duration actions earlier and run them in parallel with other actions, preventing bottlenecks. Tasks less prone to failure or of lower importance should be scheduled later to prioritize higher impact tasks. Regularly reviewing and adjusting action sequences ensures they effectively identify issues early and provide actionable feedback.

Strategically sequencing build actions is categorized as recommended as the foundational focus should first be on establishing a solid continuous integration pipeline and then later enhancing it by optimizing the build.

[DL.CI.6] Refine integration pipelines with build metrics

Category: RECOMMENDED

Use key metrics—whether sourced from this guidance, established frameworks like DORA or SPACE, or custom to your organization—to optimize your continuous integration process. Metrics such as deployment frequency, change lead time, failure rate, and time to recover serve as outcome-based lagging indicators. These indicators span many DevOps capabilities to provide insights into the efficiency and reliability of the full delivery process. While individual metrics offer granular insights to optimize specific continuous integration capabilities, these aggregated metrics present a holistic overview of the end-to-end development lifecycle. Both granular and holistic metrics are important for continuous improvement.

Embed observability practices into your integration pipelines, incorporating monitoring and logging observability capabilities. By transforming logs into metrics, you gain actionable insights into areas needing refinement. Prioritize making these metrics accessible to all team members to create an environment where teams can proactively monitor, analyze, and improve based on these metrics.

Putting an emphasis on continually optimizing pipelines using metrics is recommended. When getting started with DevOps adoption, initial efforts should prioritize the establishment of a stable and effective integration pipeline, with subsequent enhancements to the pipeline being driven by metrics.

[DL.CI.7] Validate the reproducibility of builds

Category: OPTIONAL

Every build for a specific version of source code should ideally be able to generate the same outputs from the same inputs. The implementation of reproducible builds primarily involves the creation of an immutable and consistently created build environment and controlling the inputs for each and every build.

Between each build, the environment should be destroyed and recreated so that it is immutable. Use infrastructure as code (IaC) and containerization to help with automating the creation of the environment in a repeatable and consistent way. Have controls in place to detect and prevent configuration drift that may alter the build environment post-creation. All dependencies and software components used to create the environment and perform the build should be version pinned and recorded.

Any manual intervention during the build can introduce variability. Every step in the build process needs to be automated. Factors that can render the build nondeterministic, such as unrestricted network access and the use of random generators or timestamps that modify the build artifact, must be limited.

Verify the reproducibility by establishing processes that regularly check the reproducibility of the builds. This can involve triggering builds from the same source code in different environments and comparing the results. Adopt mechanisms like binary diffing or checksum comparison to validate the reproducibility of the build. Set up alarms that raise alerts when discrepancies occur to provide fast feedback when there are inconsistencies.

Having reproducible builds is optional and not recommended for all organizations or workloads. While striving for reproducibility is encouraged, it may not be achievable in every context. For example, some builds may depend on specific environmental parameters or timing elements that make reproducibility difficult.

Related information:

- Reproducible builds

Anti-patterns for continuous integration

- **Infrequent check-in of code**: To detect issues as early as possible, developers should check-in their changes to the code base as often as possible. If developers are committing code infrequently, it can lead to longer feedback loops, and problems may go unnoticed for longer periods of time.
- **Manually building and testing changes**: A key benefit of continuous integration is the ability to automatically build and test code changes. If teams rely on manual testing or building, it will slow down the development process and make it harder to detect issues early on.
- **Having builds run on a preset schedule rather than on commit**: One of the main benefits of continuous integration is that it enables teams to detect issues as soon as possible, which is why

builds should be triggered automatically every time changes are committed to the repository. If builds are only run on a preset schedule, it can lead to longer feedback loops and more issues slipping through.

- **Low coverage or inaccurate tests:** The quality of the tests used during the continuous integration process is essential to its overall effectiveness. If tests are incomplete, missing, or inaccurate, issues are likely to slip through undetected.
- **Only testing in production**: Continuous integration requires building, testing, and validating new changes to the code base. If teams are waiting to test changes only in production, it can lead to longer feedback loops, and issues may go unnoticed until they reach the end users.
- **Failure to provide useful feedback to developers during a build**: Rapid and consistent feedback to developers is one of the core benefits of continuous integration. If feedback is not useful, it can be hard for developers to address issues, and it can slow down the development process.
- **Lack of collaboration**: Continuous integration requires collaboration between developers, operations, security, and other stakeholders. If teams are not collaborating effectively, it can lead to issues slipping through, longer feedback loops, and slower development times.

Metrics for continuous integration

- **Frequency of integration**: The average number of times developers integrate their code with the main codebase. This metric provides insight into the team's adherence to Continuous Integration practices, facilitating early issue detection and improved collaboration. Improve this metric by educating developers on the benefits of regular integration and implementing automated reminders or tools that encourage developers to integrate changes after a set number of code modifications or elapsed time. Track this metric using logs from the version control system to observe how often developers are merging code to a main releasable branch.
- **Build success rate**: The ratio of successful builds to total builds, expressed as a percentage. This metric helps teams understand the stability of their build infrastructure, quality of code changes, and the effectiveness of their tests. Track the number of successful builds over a period of time and divide by the total number of builds, then multiply by 100 to get the percentage.
- **Pipeline stability**: The percentage of build failures due to reasons other than code errors. This metric measures the reliability of the continuous integration pipeline, including its configuration and infrastructure. A high rate of such failures indicates that your continuous integration pipeline may need attention. To improve pipeline stability, routinely audit and update the pipeline, ensure consistent build environments, and reduce external dependencies that are prone to failure. Track this metric using continuous integration pipeline logs to calculate the percentage of build

failures that are due to infrastructure or configuration issues compared to the total number of build failures.

- **Mean time to build (MTTB)**: The average time required to run a complete successful build cycle, from when a code change triggers the build to its full validation. Using this metric, teams can pinpoint bottlenecks in their build process. Improve this metric by minimizing dependencies, leveraging caching, running tasks in parallel, and using more powerful or distributed build systems. Measure the duration from when the build is triggered to its completion, considering only successful builds, and calculate the average over a defined period.

Continuous delivery

Continuous delivery (CD) takes place after continuous integration (CI), where code changes that pass the build validation are automatically deployed to other environments, including production, with minimal human intervention. CD strives to ensure that new features, fixes, and improvements are deployed in a fast and reliable manner, reducing lead times and improving overall efficiency of the deployment process. By automating the delivery process using CD, teams can focus on developing and refining code, while the system handles the time-consuming and error-prone process of deploying changes to various environments.

Topics

- Indicators for continuous-delivery
- Anti-patterns for continuous delivery
- Metrics for continuous delivery

Indicators for continuous-delivery

Ensure that your code base remains in a releasable state at all times and deploy more often and with more confidence using automated pipelines.

Indicators

- [DL.CD.1] Deploy changes to production frequently
- [DL.CD.2] Deploy exclusively from trusted artifact repositories
- [DL.CD.3] Integrate quality assurance into deployments
- [DL.CD.4] Automate the entire deployment process
- [DL.CD.5] Ensure on-demand deployment capabilities

- [DL.CD.6] Refine delivery pipelines using metrics for continuous improvement
- [DL.CD.7] Remove manual approvals to practice continuous deployment

[DL.CD.1] Deploy changes to production frequently

Category: FOUNDATIONAL

Frequent deployments to production encourages small, rapid, and iterative changes to the code base. Deploying small and validated changes regularly helps mitigate the risk associated with each deployment. Frequent deployments not only streamlines the testing and validation process, but also expedites the feedback loop, leading to quicker resolution of issues.

Use a pipeline to automate the deployment of validated changes across various environments, including production. This pipeline should be automatically triggered, such as by the completion of continuous integration or an updated artifact in an artifact repository. Once invoked, the pipeline should automatically begin to deploy changes to non-production environments for further testing and validation. Upon successful validation, changes can be deployed to the production environment.

When working in a DevOps environment, it is important to distinguish between *deploying* and *releasing*. Even after deploying changes to production, these changes might not necessarily be visible or accessible to all users. By using advanced deployment strategies and employing feature flags, teams can deploy code to production and decide when to release or rollback specific features in real time, offering more granular control over releasing new features to end users.

Teams should focus on deploying small changes rather than bundling multiple changes into a single, large batch deployment. Accumulating changes complicates testing and validation, and it becomes challenging to ensure that all components interact correctly. The practice of deploying small changes demands discipline and commitment, but it improves deployment frequency, security, and enhanced collaboration while ensuring that the code base remains up-to-date and releasable at all times.

[DL.CD.2] Deploy exclusively from trusted artifact repositories

Category: FOUNDATIONAL

All artifacts involved in the delivery process should originate from a trusted artifact repository. These repositories contain validated, tested, and integrated artifacts that have been deemed safe

for deployment. By using trusted artifact repositories, teams can ensure the security of deployed workloads, maintain quality and security standards, and promote trust in the delivery pipeline.

The delivery pipeline should be restricted to using only trusted artifact repositories, which could be enforced through mechanisms such as allow lists, IP restrictions, or authentication controls. Additionally, we recommend using cryptographic signing to validate artifacts and including a validation stage in the pipeline to verify that the artifacts meet the necessary standards before deployment. In this way, the integrity and security of the deployed workloads are maintained consistently.

Related information:

- Artifact Repository - AWS CodeArtifact
- Fully Managed Container Registry - Amazon Elastic Container Registry
- Code Repositories and Artifact Management | AWS Marketplace

[DL.CD.3] Integrate quality assurance into deployments

Category: FOUNDATIONAL

Integrating quality assurance (QA) processes into continuous delivery pipelines tests that the whole system is ready for release. This differs from previous quality checks in the development lifecycle as these tests validate that the software changes behave as expected when deployed into real-world environments. This provides the ability to test integration with other live systems, check for configuration errors, and test in environments that more closely mirror production.

Incorporate QA stages into your delivery pipeline to automatically conduct required functional, non-functional, security, and data tests after deployments occur. Deployments to environments is the ideal enforcement point for quality assurance, with QA requirements being scoped to the environment being deployed to. If a test fails for one environment, it is a signal that deployment to subsequent environments might carry the same risk. Provide immediate feedback to the development team upon any test failures, so they can rectify issues quickly and maintain the integrity of the deployment pipeline.

Related information:

- AWS Well-Architected Reliability Pillar: REL08-BP02 Integrate functional testing as part of your deployment

- AWS Well-Architected Reliability Pillar: REL08-BP03 Integrate resiliency testing as part of your deployment
- AWS Well-Architected Security Pillar: SEC11-BP02 Automate testing throughout the development and release lifecycle
- Testing stages in continuous integration and continuous delivery
- Amazon's approach to high-availability deployment: Release guidance lifecycle
- Testing software and systems at Amazon: Continuous integration and deployment
- The Amazon Software Development Process: Automated Testing

[DL.CD.4] Automate the entire deployment process

Category: FOUNDATIONAL

Automate as many stages of the delivery process as possible. Exceptions for continuous delivery might include optional manual approval gates. Automation reduces the risk of human error, brings consistency to deployments, and accelerates the delivery process.

Use the delivery pipeline to automate every stage of deploying changes, from copying the build artifact to setting up any required configurations. While optional manual approval gates can exist, all other stages should be automated, maintaining the integrity of the artifact and reducing the likelihood of errors. Humans should not have access to the target environments or have the ability to inject code, parameters, configuration, or interfere with the integrity of the artifact in any way.

Some organizations might still require manual oversight at certain stages as they evolve their DevOps capabilities. If the organization is early in its DevOps adoption or operates in a highly regulated environment, there might be a need for manual interventions or approvals at certain stages. These could be due to governance or regulatory requirements or simply the need for a human decision at a critical point in the deployment process. Over time, even for these organizations, the goal should be to have no manual deployment stages in the deployment of changes.

Related information:

- AWS Well-Architected Reliability Pillar: REL08-BP05 Deploy changes with automation
- AWS Well-Architected Security Pillar: SEC11-BP06 Deploy software programmatically
- What is Continuous Delivery?
- Amazon CodeCatalyst

- Building the pipeline
- Going faster with continuous delivery
- AWS Deployment Pipeline Reference Architecture
- Deploy container applications in a multicloud environment using Amazon CodeCatalyst
- Amazon's approach to high-availability deployment: Release guidance lifecycle
- Testing software and systems at Amazon: Continuous integration and deployment
- The Amazon Software Development Process: Continuous Delivery

[DL.CD.5] Ensure on-demand deployment capabilities

Category: FOUNDATIONAL

Continuous delivery relies on the ability to ensure that every change is considered deliverable and can be deployed to production environments at any time. While the actual decision to deploy to production may still be manual, deployments should be able to occur on-demand as needed.

Deployments should be able to occur during normal working hours without causing significant downtime or disruption to the business. Changes should not require synchronization with other systems and deployments should be able to occur regardless of the interdependence of other systems. By decoupling deployments from other systems and being able to perform them during normal business hours, teams can receive fast feedback and respond to any issues that arise, leading to quicker fixes and less disruption to users.

To enable on-demand deployments, teams should employ advanced deployment strategies, such as blue/green deployments, canary releases, feature flags, or rolling updates. The ability to gradually roll out changes, coupled with modern application architectures and integrated QA processes, enables iterative delivery. Iterative delivery reduces the impact of potential issues throughout the deployment and allows for quick rollback if necessary. By using the right tools and strategies, deployments can be automated and run seamlessly, allowing for faster and more efficient delivery of applications and services.

Related information:

- AWS Deployment Pipeline Reference Architecture

[DL.CD.6] Refine delivery pipelines using metrics for continuous improvement

Category: RECOMMENDED

Use key metrics—whether sourced from this guidance, established frameworks like DORA or SPACE, or custom to your organization—to continually optimize the development lifecycle. Metrics such as deployment frequency, change lead time, failure rate, and time to recover serve as outcome-based lagging indicators. These indicators span many DevOps capabilities to provide insights into the efficiency and reliability of the full delivery process. While individual metrics offer granular insights to optimize specific continuous delivery capabilities, these aggregated metrics present a holistic overview of the end-to-end development lifecycle. Both granular and holistic metrics are important for continuous improvement.

Use observability practices to continuously monitor the development lifecycle, including incorporating monitoring and logging into your delivery pipelines. Use logs to generate metrics, and use these metrics to identify areas for improvement. Make these metrics visible to all team members and use them to drive your continuous improvement efforts.

Putting an emphasis on continually optimizing pipelines using metrics is recommended. When getting started with DevOps adoption, initial efforts should prioritize the establishment of a stable and effective delivery pipeline, with subsequent enhancements to the pipeline being driven by metrics.

Related information:

- Deployment Pipeline Reference Architecture
- AWS Observability Best Practices: Key Performance Indicators
- DevOps Research and Assessment (DORA)
- SPACE

[DL.CD.7] Remove manual approvals to practice continuous deployment

Category: OPTIONAL

Fully automate all stages of the deployment process, allowing developers to push new code into the production environment using fully automated delivery pipelines—with no manual approval stages required. This is referred to as continuous deployment. Removing all manual deployment steps reduces potential errors and increases deployment speed. It allows developers to focus more on coding and less on deployment logistics, improving efficiency and productivity.

Create fully automated pipelines which perform continuous integration and continuous deployment. A pipeline should trigger upon code changes being merged into the main release branch. This pipeline should perform all necessary quality assurance tests, build the application,

and deploy the new version to the production environment. Automated governance capabilities ensure that guardrails are being followed, while observability functions such as alerts and logs provide visibility.

This level of automation is a hallmark of mature DevOps practices. However, it is an optional capability as it is not always achievable or desired, especially in heavily regulated industries or in organizations with strict governance controls.

Related information:

- Continuous Delivery vs. Continuous Deployment
- Practicing Continuous Integration and Continuous Delivery on AWS

Anti-patterns for continuous delivery

- **Large batch deployments**: Batching multiple changes into a single, large release can lead to increased risk and longer lead times. Deploying in smaller batches allows for faster feedback and quicker resolution of issues, ultimately leading to a more efficient and reliable delivery process.
- **Manual deployments**: Deployments done manually lack consistency, increase the chances of human error, and hinder the pace of delivering software changes. This practice contradicts the fundamental premise of continuous delivery and continuous deployment which emphasizes on automation to ensure reliability and speed.
- **Building more than once**: Not adhering to the *build once, deploy many* principle during deployments. When code is built multiple times, there's a chance that inconsistencies due to different environments having different configurations, component versions, or updates. This can lead to deployment errors, causing failures in production that weren't encountered in pre-production environments. To ensure consistency, build the artifact once, and always deploy artifacts directly from trusted artifact repositories.
- **Tightly coupled systems**: Developing and deploying in a tightly coupled architectural environment can lead to numerous continuous delivery challenges. In tightly coupled systems, a change in one component often necessitates changes in others, leading to cascading updates and increased complexity in managing deployments. This makes isolation of changes for testing purposes difficult, leading to longer lead times and an increased likelihood of bugs being introduced. This also makes the system less resilient, as failures can easily propagate through the system. Instead, architect loosely coupled systems that use modular components or microservices. This allows for systems to be updated and deployed independently of each other, reducing complexity and risk.

Metrics for continuous delivery

- **Pipeline stability**: The percentage of deployments that encounter failures, which includes failed deployments, required rollbacks, and incidents directly linked to deployments. This metric provides insight into the reliability and efficiency of the continuous delivery pipeline, with a focus on its configuration, infrastructure, and the quality of the code being deployed. A high rate of these failures suggests the continuous delivery pipeline may need refinement. Examine the continuous delivery pipeline logs and calculate the failure percentage based on the total number of deployments over a specific period. This should account for both direct deployment failures and deployments that required subsequent rollbacks or led to incidents.
- **Mean time to production (MTTP):** The average time taken from the moment a code change is merged to when it's live in the production environment. This demonstrates how quickly features, bug fixes, or changes get delivered to end users. Improve this metric by streamlining deployment processes, automate testing, reduce manual interventions, and optimize infrastructure provisioning. Calculate this metric using timestamps from merge events and production deployment events, then calculate the average difference over a given period.
- **Operator interventions**: The number of deployments run without human intervention, signifying the level of automation and reliability in the deployment process. A higher count might indicate potential areas for automation or optimization. Improve this metric by reducing toil by increasing automation in the deployment pipeline, reducing manual testing and verification, and establishing trust in automated processes. Monitor deployment logs and count the number of deployments that required manual interventions. Aggregate the count over a set period, such as weekly or monthly.
- **Number of changes per release**: The number of changes included in each release of the software. It can include changes to code, configuration, or other components of the system. A high number of changes per release may indicate batching of work and a lack of continuous integration. This can lead to longer lead times, increased risk of defects, and reduced ability to troubleshoot issues. The ideal number of changes per release will depend on the specific needs of the organization and the system being developed, and should be continually evaluated and adjusted as needed. Track this metric using release notes, change logs, or commit metadata for each release. For each release, count the number of distinct changes that were included.
- **Deployment frequency**: The frequency at which code is deployed to a production environment. It helps teams understand how quickly they can deliver changes, enhancements, and fixes to users at a rapid pace. A higher deployment frequency often correlates with a faster feedback loop resulting not only from continuous delivery, but other mature DevOps practices like quality assurance and observability as well. Lower frequency of deployments may indicate manual or

batched deployment processes, bottlenecks in the release pipeline, or a more cautious release strategy. Aim for a balance between high deployment frequency and system stability. Regularly review deployment logs to count the number of successful deployments over a given period, such as daily, weekly, or monthly.

Advanced deployment strategies

Advanced deployment strategies provide organizations with the ability to deploy and release new features and updates gradually. The fast feedback loop enabled by these strategies aids in early detection and resolution of potential issues during deployment, enhancing the reliability of the release process. With advanced deployment strategies, organizations can improve the quality and speed of software releases, reduce the risk of downtime or errors, and provide enhanced user experience.

Topics

- Indicators for advanced deployment strategies
- Anti-patterns for advanced deployment strategies
- Metrics for advanced deployment strategies

Indicators for advanced deployment strategies

Use modern deployment methods and release practices to minimize the risk of deployment issues. Gradually deploy and release changes to improve reliability of software releases and enhance user experience.

Indicators

- [DL.ADS.1] Test deployments in pre-production environments
- [DL.ADS.2] Implement automatic rollbacks for failed deployments
- [DL.ADS.3] Use staggered deployment and release strategies
- [DL.ADS.4] Implement Incremental Feature Release Techniques
- [DL.ADS.5] Ensure backwards compatibility for data store and schema changes
- [DL.ADS.6] Use cell-based architectures for granular deployment and release

[DL.ADS.1] Test deployments in pre-production environments

Category: FOUNDATIONAL

Progressively validate software changes across multiple environments, including development (alpha) and testing (beta) before deploying into production. Additional staging environments can be introduced as needed, such as staging (gamma). These additional environments help to prevent the introduction of bugs in production environments, validates backwards compatibility, and increases the confidence in the quality of the deployment.

Each non-production deployment serves as a gate, only allowing changes to progress to the next stage after they pass all validations. Early issue detection and isolation prevent propagation to later stages or production. A controlled deployment process includes strategies to manage risk and support rollback if issues are identified during these test deployments.

One-box testing can be used to test backward compatibility to ensure new code changes coexist with and function properly with the existing code base. One-box refers to the testing of changes in a single unit of deployment, such as a single container or instance, which is configured to use production endpoints. This form of testing can be used to help ensure the changes interact efficiently with production endpoints of other services. This can be done by creating a dedicated staging environment for cross-service backward compatibility (zeta) testing. Services deployed to the zeta stage interact exclusively with production endpoints to identify potential integration issues before the code reaches the production stage.

Related information:

- What is Continuous Integration?
- What is Continuous Delivery?
- Going faster with continuous delivery
- Automating safe, hands-off deployments: Test deployments in pre-production environments
- Amazon's approach to high-availability deployment

[DL.ADS.2] Implement automatic rollbacks for failed deployments

Category: FOUNDATIONAL

Implement an automatic rollback strategy to enhance system reliability and minimize service disruptions. The strategy should be defined as a proactive measure in case of an operational event,

which prioritizes customer impact mitigation even before identifying whether the new deployment is the cause of the issue.

Rollback should be initiated based on alarms linked to key metrics like fault rates, latency, CPU usage, memory usage, disk usage, and log errors. Additionally, consider both the service's overall health and instance-specific metrics. Incorporate a waiting period after a deployment to closely monitor the system. This allows time to identify potential issues that might not be evident immediately, especially when the system is under low load. Establish methods to prevent deployments during higher-risk times or when there are active system issues. This could include blocking deployments during when high-severity aggregate alarms are raised or during specific time windows.

The rollback process should include the redeployment of the last successful code revision, artifact version, or container image, and should employ methods like rolling or blue/green deployments, or feature flags for a swift rollback with minimal disruption. Consider using the advanced deployment methods introduced in this capability for more granular control over deployments. Rollback considerations should not be limited to the latest deployments, but also account for latent changes that may be the source of current issues. To handle these situations, provide the ability for developers to select a specific previously deployed release for rollback.

After the rollback, depending on the specific issue being addressed, consider proactively rolling back other environments that could potentially also be affected, even if they aren't currently showing any customer impact. Alternatively, if the issue appears to be environment-specific, wait for the pipeline to roll forward a new release that includes a bug fix. These operational decisions should be supported by the ability to compare the changes between the current release and the selected rollback release's deployment artifacts, including source code changes and changes in library versions.

Related information:

- Ensuring rollback safety during deployments
- My CI/CD pipeline is my release captain: Easy and automatic rollbacks
- Automating safe, hands-off deployments
- Amazon's approach to high-availability deployment: Rollback alarms

[DL.ADS.3] Use staggered deployment and release strategies

Category: FOUNDATIONAL

Staggered deployments strategies make use of techniques like progressive wave-based deployments, one-box deployments, and rolling deployments. These techniques contribute to safer and more reliable software deployment and release processes. Staggered deployments are beneficial as they balance the safety of small-scoped deployments with the speed of delivering changes to customers.

Progressive deployments, for instance, involve deploying changes to deployment groups, or *waves*, of increasing size. This method helps to achieve a balance between deployment risk and speed, promoting changes from wave to wave. The initial waves build confidence in the change by starting with a low number of requests and then gradually increasing.

Each production wave of the staggered deployment starts with a limited deployment, one-box stage, where the new code is first deployed to a single unit called a *box*. A box could be a single server or container instance which is deployed to a specific environment, AWS Region, single AWS Availability Zone, or within a single cell in a cell-based architecture. This approach minimizes the potential impact of changes by initially limiting the requests served by the new code. The box should be served a fraction of canary tests while its performance is being closely monitored before a broader rollout.

Following the limited deployment stage, rolling deployments are typically used to deploy to the wave's main production fleet. This approach helps ensure that the service has enough capacity to serve the production load throughout the deployment. A typical rolling deployment to an environment replaces at most 33% of the system's fleet in that environment with the new code. By maintaining at least 66% of the overall capacity healthy and serving requests, the impact of changes is limited. If necessary, fast rollbacks can be implemented where the system replaces 33% of the system's fleet with the previous code to speed up the rollback process.

If you require more control over the release of the change, consider using blue/green deployments rather than one-box and rolling deployments. In a blue/green deployment, two identical production environments are maintained, and the inactive environment (either blue or green) is updated. Once fully tested and ready, traffic is switched from the active to the inactive environment, thus minimizing downtime and risk

These strategies reduce the risk of introducing issues into the system and allow for monitoring, swift rollback, and issue tracking. However, they require careful planning, thorough testing, and detailed monitoring. Their benefits to system reliability and resilience are substantial and are recommended for any organization.

Related information:

- AWS Well-Architected Reliability Pillar: REL08-BP04 Deploy using immutable infrastructure
- Automating safe, hands-off deployments
- AWS Deployment Pipeline Reference Architecture
- Overview of Deployment Options on AWS
- Deployment methods
- Using Amazon RDS Blue/Green Deployments for database updates
- Amazon's approach to high-availability deployment: Canary deployments
- Hands-off: Automating continuous delivery pipelines at Amazon
- The Amazon Software Development Process: Pessimistic Deployments

[DL.ADS.4] Implement Incremental Feature Release Techniques

Category: RECOMMENDED

Incremental feature releases gradually roll out new features to users, reducing risk and maintaining system stability. Techniques include dark launching, two-phase deployments, feature flags, and canary releases. These techniques enable safe, controlled, and iterative changes to distributed systems which reduces risk associated with concurrent updates and maintaining system stability.

Dark launches allow teams to integrate and test new features in a live environment, without needing to make them visible to the entire user base. This approach allows for monitoring and analyzing the impact and performance of new features under real-world conditions, while mitigating the risk of widespread disruptions. Depending on system implementation and team preferences, dark launches can be implemented using versioning, A/B testing, canary releases, or most commonly, using feature flags.

Feature flags allow developers to turn on or off certain features in their code base without affecting other functionality. This allows for testing of new features with a subset of users, limiting potential negative impacts. Feature flags provide an additional layer of control over the feature rollout process and can be used for A/B testing, canary releases, and dark launches.

Two-phase deployments complement dark launching, focusing primarily on managing read and write changes in a systematic and phased manner. Changes should first be prepared to handle a new update without actively implementing it (Prepare phase), followed by a second deployment that activates the new changes (Activate phase). This approach requires careful planning and coordination, but pays off by prioritizing data integrity and preventing stale records that could emerge from concurrent changes.

The specific choice of technique, be it dark launching, two-phase deployments, feature flags, canary releases, or a combination, depends on your unique needs, the nature of the changes, the complexity of the system, and the degree of control required over the release process. Each of these methods offers its own advantages, and their strategic implementation can significantly enhance the resilience and efficiency of your deployments.

Related information:

- Amazon CloudWatch Evidently
- Feature Flags - AWS AppConfig
- My CI/CD pipeline is my release captain: Multiple inflight releases
- Ensuring rollback safety during deployments
- Using AWS AppConfig Feature Flags
- The Only Guide to Dark Launching You'll Ever Need
- Deployment Pipeline Reference Architecture: Dynamic Configuration Pipeline

[DL.ADS.5] Ensure backwards compatibility for data store and schema changes

Category: RECOMMENDED

Backwards compatibility in data stores and schemas ensures that as changes are made, previous versions of the system continue to operate as expected. This requires careful planning, thorough testing, and detailed monitoring. As modifications, additions, or deletions are made to data structures and schemas, these changes should be designed to coexist with previous data structures, allowing both old and new versions to operate concurrently. Maintaining backwards compatibility helps to avoid breaking changes that could disrupt continuous integration and delivery pipelines.

One way to achieve backwards compatibility is by implementing versioning in your data schemas. With this method, new changes are incorporated into a new version, while older versions remain functional for existing applications. Feature flags can also be used to conceal new alterations until they're fully ready, facilitating testing and phased rollout of updates without affecting existing users.

To ensure the safe implementation of these changes, they should be thoroughly tested in a non-production environment. Testing typically involves three stages to detect potential issues: initially, the change is deployed to a fraction of the servers to verify coexistence of software versions; next,

the deployment is completed across all servers; and finally, a rollback deployment is initiated. If no errors or unexpected behavior occur during these stages, the test is considered successful.

In scenarios involving changes that require coordination between different microservices, it is important to maintain consistency in the order of deployments across environments. For example, in serialization contexts, readers are typically deployed before writers during roll-forward, while writers precede readers during rollbacks.

Related information:

- Ensuring rollback safety during deployments
- Using Amazon RDS Blue/Green Deployments for database updates

[DL.ADS.6] Use cell-based architectures for granular deployment and release

Category: OPTIONAL

A cell-based architecture segments a larger system into isolated, independently functioning replicas, or *cells*. These cells are smaller components of the system that contain all application logic and storage. They have their own monitoring and alerting systems, are automated for creation and update, and can be managed and scaled individually. This approach offers advantages including scalability, fault isolation, testing, and operational resilience.

A cell-based architecture is a natural fit for DevOps as it enables small, frequent changes, reduces the risk from problematic deployments, and enables rapid recovery. It allows teams to deliver incremental updates to individual cells without risking the entire system's stability.

Start by defining your cells, each of which should be a complete, independently deployable unit of your system. You should limit the maximum size of a cell and maintain this consistency across different regions or installations. You then need to establish a routing layer that redirects client requests to the appropriate cell. You can store the routing information, such as user-to-cell mapping, in a low-latency database. Every cell should have its own monitoring and alerting system.

You will need to automate the lifecycle of your cells, including initial deployment and subsequent updates. A *canary cell* can be helpful in initial deployment of updates and assessing their impact. Ensure that you implement a central dashboard to provide an aggregated view of the state of your cells, enabling easy system-wide monitoring. Stream changes to a central data lake for centralized querying and analysis of changes across all cells. Finally, implement an operational tool to move users between cells and create new cells as needed. This step optimizes load distribution across cells by updating the user-to-cell assignment.

Cell-based architectures are optional. While beneficial for complex systems, smaller systems might not require such architectural complexity.

Related information:

- AWS Well-Architected Reliability Pillar: REL10-BP04 Use bulkhead architectures to limit scope of impact
- Guidance for Cell-based Architecture on AWS
- Minimizing correlated failures in distributed systems
- Journey to cell-based microservices architecture on AWS for hyperscale

Anti-patterns for advanced deployment strategies

- **Deploying directly to production**: Deploying changes directly to production without first testing in pre-production environments risks unforeseen errors, bugs, or performance issues that can lead to service disruptions or downtime. Pre-production testing in environments that mimic production as closely as possible is crucial to verify the functionality and compatibility of changes under realistic conditions.
- **Ignoring rollbacks and data compatibility**: The absence of an automatic rollback strategy and a lack of consideration for data compatibility can lead to prolonged service disruptions and compatibility issues. An automatic rollback mechanism can reduce downtime and maintain system reliability as it ensures a quick return to a stable state in the event of a fault. Maintaining backward compatibility in data stores and schemas can prevent disruptions to existing functionalities and integration pipelines. Changes should be designed to coexist with previous data structures and contracts, allowing both old and new versions to operate concurrently.
- **Monolithic deployment model**: Deploying all changes simultaneously and treating the entire system as a single unit increases the risk of errors that could impact the entire system and limits scalability. To mitigate these risks, adopt staggered deployments and consider cell-based architectures. Staggering deployments through wave, one-box, or rolling deployments allows for easier issue detection and rollback which reduces negative impact of failed deployments. A cell-based architecture enhances fault isolation, granular control, and operational resilience, making it a preferred strategy for complex, distributed systems.
- **Abrupt feature release**: Releasing new features to all users at once without incremental deployment or testing can result in widespread disruptions if the feature fails or impacts the system negatively. Techniques like dark launching, two-phase deployments, feature flags, and

canary releases reduce this risk by providing control and facilitating monitoring of the feature's impact in real-world conditions.

Metrics for advanced deployment strategies

- **Rollback frequency**: This metric measures how often changes need to be rolled back. While a higher rollback frequency may indicate issues with the deployment process or inadequate quality assurance capabilities, it can also suggest successful usage of advanced deployment strategy capabilities with automation facilitating fast rollbacks to minimize user risk. Track this by counting the number of rollbacks and comparing it to the total number of deployments.
- **Deployment lead time**: The average time required to successfully deploy a feature or service from the moment a deployment is triggered to when it is live in an environment. Using this metric, teams can pinpoint bottlenecks in the deployment process. Enhance this metric by optimizing deployment strategies, utilizing distributed architectures, or deploying in waves to strike a balance between speed and safety. Measure the duration from when the deployment is triggered to its completion, considering only successful deployments, and calculate the average over a specific time frame, such as weekly or monthly.
- **Release frequency**: The frequency at which changes become accessible to end users. This metric distinguishes between deployments, which introduces new code or configurations into an environment, and releases, which make those changes accessible to end users. A high release frequency can indicate mature DevOps capabilities which enable releasing small, incremental changes that are automatically deployed and verified with confidence. Measure release frequency by counting the number of releases to production over a specified period. Compare this metric to deployment frequency to understand the correlation and derive additional insights.
- **Mean time to recover (MTTR)**: The average time taken to restore a system after a failure. This metric provides insight into the team's ability to quickly detect and address production issues. A lower MTTR indicates safer deployment practices, the use of automated rollbacks, and effective governance, quality assurance, and observability capabilities. Measure the total amount of downtime and divide it by the total number of incidents within a specific time frame.

Quality assurance

The quality assurance saga emphasizes the integration of test-driven methodologies into every phase of the software development process. By prioritizing the quality of delivered code, it

promotes development agility while helping to ensure security, availability, reliability, and resilience of systems. Investing time and resources into comprehensive testing capabilities not only safeguards an organization's reputation, but also translates to reduced operational costs, increased customer trust, and a competitive edge in the market. These capabilities evolve traditionally reactive quality assurance processes into proactive and integral parts of the development lifecycle.

Capabilities

- Test environment management
- Functional testing
- Non-functional testing
- Security testing
- Data testing

Test environment management

This capability focuses on dynamically provisioning test environments that are used for running test cases. Using automation to manage these environments and associated test data reduces both testing duration and expense while improving accuracy of test results. Effectively managing test data as a part of this process helps with identifying and correcting defects earlier on in the development lifecycle.

Topics

- Indicators for test environment management
- Anti-patterns for test environment management
- Metrics for test environment management

Indicators for test environment management

Use testing environments to simulate real-world conditions, ensuring changes are ready for production deployment.

Indicators

- [QA.TEM.1] Establish dedicated testing environments
- [QA.TEM.2] Ensure consistent test case execution using test beds
- [QA.TEM.3] Store and manage test results

- [QA.TEM.4] Implement a unified test data repository for enhanced test efficiency
- [QA.TEM.5] Run tests in parallel for faster results
- [QA.TEM.6] Enhance developer experience through scalable quality assurance platforms

[QA.TEM.1] Establish dedicated testing environments

Category: FOUNDATIONAL

Use testing environments to detect and correct issues earlier on in the development lifecycle. Deploy integrated changes into these environments before they are deployed to production. These environments are as production-like as possible, providing the ability to simulate real-world conditions which can validate that changes are ready for production deployment.

Design your testing environments to mimic production qualities that you need to test, such as monitoring settings or regional variants. At a minimum, have a staging environment that you monitor closely to catch potential issues early. More testing environments, such as beta or zeta, can be added as needed. Infrastructure as code (IaC) should be used for managing and deploying these environments, ensuring consistent and predictable provisioning. Minimize direct human intervention in these environments, similar to how you would treat production environments. Instead, rely on automated delivery pipelines with stages that deploy to the testing environment. Human access should be strictly controlled, auditable, and granted only in exceptional circumstances.

Related information:

- AWS Well-Architected Sustainability Pillar: SUS06-BP04 Use managed device farms for testing
- Development and Test on Amazon Web Services
- Test environments in AWS Device Farm
- Deployment Pipeline Reference Architecture

[QA.TEM.2] Ensure consistent test case execution using test beds

Category: FOUNDATIONAL

Test cases require specific conditions and test data to run in a predetermined state. Test beds, configured within broader testing environments, provide the conditions necessary to ensure reproducible and accurate test case execution. While a single testing environment, such as a staging environment, can host multiple test beds, each test bed is tailored with the infrastructure

and data suitable for specific test scenarios. Being able to start each test case with the correct configuration and data setup makes testing reliable, consistent, and confirms that anomalies or failures can be attributed to code changes rather than data inconsistencies.

Integrate test bed preparation into the delivery pipeline, leveraging infrastructure as code (IaC) to help guarantee consistent test bed setup. Rather than updating or patching test beds, use immutable infrastructure and treat them as ephemeral environments. When a test is run, create a new test bed using IaC tools to help ensure that it is clean and consistent. It is advantageous to have a fresh environment for each test. However, after running the tests, while the test bed can be deleted, it is important to avoid deleting logs and data that can aid with debugging the testing process. This data may be required for analyzing failures. Deleting it prematurely can lead to wasted time and the potential need for rerunning lengthy tests.

Use data restoration techniques to automate populating test beds with test data specific to the test case being run. Depending on the complexity, the test data can be generated on-demand or sourced from a centralized test data store for scalability and consistency. For tests that modify data and require reruns, use a caching system to quickly and cost-effectively revert the dataset, minimizing bottlenecks in the testing process. Automating test data restoration saves time and effort for teams, enabling them to focus on actual testing activities instead of manually test data management.

Continually monitor the speed, accuracy, and relevance of test bed setup and execution. As testing requirements evolve or data volume and complexity grow, make necessary adjustments. Provide immediate feedback to the development team if there is a failure arising from test bed setup, data inconsistency, or test execution.

Related information:

- AWS Well-Architected Sustainability Pillar: SUS06-BP03 Increase utilization of build environments

[QA.TEM.3] Store and manage test results

Category: FOUNDATIONAL

When tests are run, the results offer insights into the system's health, providing actionable feedback for developers. Establish a structured test result storage strategy to maintain the integrity, relevance, and availability of these results.

Store test results in a centralized system or platform using a machine-readable format, such as JSON or XML. This simplifies comparison and analysis of test results across various test iterations. Configure automated deployment pipelines and individual testing tools to publish test results to this platform immediately upon test completion. Each set of test outcomes should be both timestamped and versioned to enable historical tracking of changes, improvements, or potential regressions.

Test results must be encrypted both at rest and in transit to protect against sensitive data inadvertently being stored in test results. Access to raw test result files should be limited and idempotent, with write access explicitly restricted. To view results on a regular basis, implement tools that allow for visualizations, such as dashboards, charts, or graphs, which provide a summarized view of test results. Grant users and roles access to these tools to review results, identify trends or anomalies, and build reports.

Old test results, while useful for historical context, might not always be necessary to retain indefinitely. Define a policy for test result retention that aligns to your governance and compliance requirements. Ideally, this includes automatically archiving or delete test results to help ensure the system remains uncluttered and cost efficient.

Related information:

- Viewing the results of a test action - Amazon CodeCatalyst
- Working with test reporting in AWS CodeBuild
- Test Reports with AWS CodeBuild

[QA.TEM.4] Implement a unified test data repository for enhanced test efficiency

Category: RECOMMENDED

Test data refers to specific input datasets designed for testing purposes to simulate real-world scenarios. Centralizing test datasets in a unified storage location, such as a data lake or source code repository, ensures they are stored, normalized, and managed effectively.

Test data might be stored differently depending on your specific use case. It can be stored centrally for a single team who maintains multiple microservices or related products, or centrally governed for multiple teams to source test data from. By centralizing, teams can reuse the same test data across different test cases, minimizing the time and effort spent preparing test data for usage.

Create a centralized, version-controlled system to store test datasets, such as a data lake or source code repository. Ensure the data in this central repository is sanitized and approved for non-production environments. When test environments are set up and test cases are run, use delivery pipelines and automated tools to source test data directly from this centralized source.

Outdated test datasets can result in ineffective tests and inaccurate results. Regularly maintain the centralized test data source by updating it either periodically or when there are changes in systems data schemas, features, functions, or dependencies. Treat the test data as a shared resource with contracts in place to prevent disrupting other teams or systems. Document any changes made to test data and notify any dependent teams of these changes. Maintaining up-to-date test data allows for more effective issue identification and resolution, leading to higher-quality software.

We recommend automating the update process where feasible using data pipelines, for example, by pulling recent production data and obfuscating it as changes are made. Protect sensitive data by implementing a data obfuscation plan that transforms sensitive production data into similar, but non-sensitive, test data. Use obfuscation techniques, such as masking, encrypting, or tokenizing, to sanitize the production data prior to it being used in non-production environments. This approach helps uphold data privacy and mitigates potential security risks during testing.

Related information:

- AWS Well-Architected Sustainability Pillar: SUS04-BP06 Use shared file systems or storage to access common data
- AWS Well-Architected Sustainability Pillar: SUS04-BP07 Minimize data movement across networks
- AWS Well-Architected Cost Optimization Pillar: COST08-BP02 Select components to optimize data transfer cost
- AWS Glue DataBrew
- Identifying and handling personally identifiable information (PII)
- Data Obfuscation
- Data Masking using AWS DMS (AWS Data Migration Service)
- Data Lake Governance - AWS Lake Formation

[QA.TEM.5] Run tests in parallel for faster results

Category: RECOMMENDED

Parallelized test execution is the practice of concurrently running multiple test cases or suites to accelerate test results and expedite feedback. As software grows and becomes more modular, especially in architectures like microservices, the number of test cases also increases. Running these tests sequentially could significantly slow down delivery pipelines. By creating many test beds and distributing test cases across them asynchronously, tests can be run in parallel to allow for faster iterations and more frequent deployments.

Adopt a scaling-out strategy to test bed provisioning to establish multiple test beds tailored for specific test scenarios. Each test bed, provisioned through infrastructure as code (IaC), should have the necessary infrastructure and data setup for its designated test cases. Serverless infrastructure or container orchestration tools combined with state machines, such as AWS Step Functions, can improve your ability to dynamically provision and run tests in a scalable and cost-effective way. Test operations should not impact the data or outcome of other test beds. As tests are parallelized across multiple test beds, ensure data isolation to maintain test integrity. Use monitoring solutions to track parallelized test runs, ensuring each test bed is performing optimally and to help in debugging any anomalies.

Related information:

- Run Selenium tests at scale using AWS Fargate
- Runs in AWS Device Farm

[QA.TEM.6] Enhance developer experience through scalable quality assurance platforms

Category: RECOMMENDED

As team structures and operating models change within the organization to support distributed teams with value stream ownership, the roles and responsibility of quality assurance teams also evolve. In a DevOps environment with supportive team dynamics, individual stream-aligned teams take ownership of quality assurance and security within their value stream and products. This approach removes the handoff of responsibility and accountability to centralized quality assurance or testing teams within the organization. These quality assurance functions are still extremely important to sustainably practicing DevOps and can be distributed to make them more effective in supporting stream-aligned teams.

One method of distributing a centralized quality assurance function is to form platform teams. These platform teams offer scalable testing services to stream-aligned teams to enhance the developer experience and expedite test environment set up. Platforms managed by these

teams can feature self-service options, automated test environment management, test bed provisioning, and equipping teams with the tools to produce, manage, and use test data and infrastructure. Additionally, these platforms can integrate device farms, allowing for testing across a variety of devices such as mobile phones or web browsers such as Chrome and Firefox on diverse operating systems.

Quality assurance platforms can also be created to provide security related capabilities which enable continuous visibility into the security posture of applications throughout the development lifecycle, such as Application Security Posture Management (ASPM). Stream-aligned teams can leverage these capabilities to prioritize and address vulnerabilities identified during testing, contributing to overall risk reduction and improved application security. By providing a platform for consistent testing procedures and security controls, quality assurance platform teams can help support the organization's observability and automated governance goals.

Another method of distributing quality assurance teams is to form enabling teams. These teams can help stream-aligned teams onboard to quality assurance platforms and teach teams to become self-sufficient with test design and execution. It is important that enabling teams do not take ownership over testing for a value stream or product. They provide just-in-time guidance and knowledge sharing, but ultimately move on to help other teams. If long-term quality assurance support is needed within a development team, cross-train the quality assurance member so that they gain development skills and permanently embed them into the stream-aligned team.

Related information:

- The Amazon Software Development Process: Self-Service Tools

Anti-patterns for test environment management

- **Low test data coverage:** Using a limited set of test data that does not cover all known scenarios, leading to testing gaps and unexpected issues. Effective test data management should include the creation of a comprehensive and diverse set of test data that covers all possible scenarios. Reduce the likelihood of low data coverage occurring by being proactive about test data requirements at the start of the development lifecycle rather than creating and managing test data after a change has been made.
- **Insecure test data:** Not having the proper policies and security measures in place when generating, storing, or using test data can lead to potential data breaches, inaccuracies, and inefficiencies in the test process. Test data management strategies should prioritize data security and governance practices to ensure that data is accurately and securely managed throughout

the testing process. For example, using actual production data without proper obfuscation or sanitization in test environments has the potential of exposing sensitive information. Use automated governance capabilities to apply guardrails, set up secure access to test data, and automate the lifecycle of test data.

- **Centralized testing:** Maintaining a traditional, centralized approach to quality assurance and testing can create bottlenecks, reduce agility, and inhibit teams from taking full ownership of their value stream. Instead, focus on providing stream-aligned teams with the tools, knowledge, and resources to self-manage their testing requirements. Distribute quality assurance functions by creating platform teams and enabling teams to support others by offering scalable testing services, tools, and guidance.

Metrics for test environment management

- **Test bed provisioning time:** The duration spent provisioning a test environment including all supporting infrastructure and data saturation. This metric indicates inefficiencies in the overall test bed provisioning process. Measure the time difference between the start of provisioning to when the environment is confirmed as ready to run a test case.
- **Test case execution time:** The duration taken to run a test case or a suite of test cases. Faster test execution speeds indicate efficient test environment and execution management which allows for rapid iterations and feedback. Improve this metric by optimizing test bed provisioning, ensuring efficient resource allocation, and parallelizing test runs. Measure the timestamp difference between the start and end of test case execution.

Functional testing

Functional testing validates that the system operates according to specified requirements. It is used to consistently verify that components such as user interfaces, APIs, databases, and the source code, work as intended. By examining these components of the system, functional testing helps ensure that each feature behaves as expected, safeguarding both user expectations and the software's integrity.

Topics

- Indicators for functional testing
- Anti-patterns for functional testing
- Metrics for functional testing

Indicators for functional testing

Assess specific functionalities of systems to ensure they operate correctly and meet predefined requirements.

Indicators

- [QA.FT.1] Ensure individual component functionality with unit tests
- [QA.FT.2] Validate system interactions and data flows with integration tests
- [QA.FT.3] Confirm end-user experience and functional correctness with acceptance tests
- [QA.FT.4] Balance developer feedback and test coverage using advanced test selection

[QA.FT.1] Ensure individual component functionality with unit tests

Category: FOUNDATIONAL

Unit tests evaluate the functionality of one individual part of an application, called *units*. The goal of unit tests is to provide fast, thorough feedback while reducing the risk of introducing flaws when making changes. This feedback is accomplished by writing tests cases that cover a sufficient amount of the code. These test cases run the code using predefined inputs and set expectations for a specific output.

Unit tests should be isolated to a single class, function, or method within the code. Fakes or mocks are used in place of external or infrastructure components to help ensure that the scope is isolated. These tests should be fast, repeatable, and provide assertions that lead to a pass or fail outcome. Teams should be able to run unit tests locally as well as through continuous integration pipelines.

Ideally, teams adopt Test-Driven Development (TDD) practices and write tests before the software is developed. This approach can lead to faster feedback, more effective tests, and introducing less defects when writing code.

Related information:

- AWS Well-Architected Reliability Pillar: REL12-BP03 Test functional requirements
- Building hexagonal architectures on AWS - Write and run tests from the beginning
- AWS Deployment Pipeline Reference Architecture
- Testing software and systems at Amazon: Unit tests
- Adopt a test-driven development approach using AWS CDK
- Getting started with testing serverless applications

- TestDouble

[QA.FT.2] Validate system interactions and data flows with integration tests

Category: FOUNDATIONAL

Integration tests evaluate the interactions between multiple components that make up the system, including infrastructure and external systems. The goal of integration testing is to help ensure that these interactions and data flows work together, ensuring that recent changes have not disrupted any interfaces or introduced undesired behaviors.

Integration tests often run much slower than unit testing due to the fact that they interact with real system, such as databases, message queues, and external APIs. Strive to make integration tests as efficient as possible by optimizing setup and tear down using automation and infrastructure as code (IaC). Optimize test execution by running tests in parallel where possible. This allows for quicker feedback loops and makes it possible to run integration tests through continuous integration pipelines.

While integration tests should involve real components, they should still be isolated from production or shared environments where possible. This helps ensure that tests do not inadvertently affect real data or services. Consider using dedicated emulation, containers, or cloud-based test environments to make tests more efficient, consistent, and safe.

Just as with unit tests, adopting Test-Driven Development (TDD) by writing tests before the software is developed helps to highlight potential integration pain points early, and verifies that the interfaces between components are correctly implemented from the start.

Related information:

- AWS Well-Architected Reliability Pillar: REL12-BP03 Test functional requirements
- AWS Deployment Pipeline Reference Architecture
- Getting started with testing serverless applications
- Amazon's approach to high-availability deployment: Integration testing
- Building hexagonal architectures on AWS - Write and run tests from the beginning

[QA.FT.3] Confirm end-user experience and functional correctness with acceptance tests

Category: FOUNDATIONAL

Acceptance tests evaluate the observable functional behavior of the system from the perspective of the end user in a production-like environment. These tests encompass functional correctness of user interfaces, general application behavior, and ensuring that user interface elements lead to expected user experiences.

By considering all facets of user interactions and expectations, acceptance testing provides a comprehensive evaluation of an application's readiness for production deployment. There are various forms of functional acceptance tests which should be used throughout development lifecycle:

- **End-To-End (E2E) Testing:** Acceptance tests performed by the development team through delivery pipelines to validate integrated components and user flows. Begin by identifying the most impactful user flows and create test cases for them. Ideally, teams practice Behavior-Driven Development (BDD) to define how the system will be designed to be tested before code is written. Next, adopt a suitable automated testing framework, such as AWS Device Farm or Selenium. Using the continuous delivery pipeline, trigger the testing tool to run scripted tests cases against the system while it is running in the test environment.
- **User Acceptance Testing (UAT):** Acceptance tests performed by external end-users of the system to validate that the system aligns with business needs and requirements. The users measure the application against defined acceptance criteria by interacting with the system and providing feedback based on if the system behaves as expected. The development team engages, instructs, and supports these users as they test the system. Log the results of the test by gathering feedback from the users, using the acceptance criteria as a guide. Feedback should highlight areas where the system met or exceeded expectations as well as areas where the system did not meet expectations.
- **Synthetic Testing:** Continuously run simulations of user behavior in a live testing environment to proactively spot issues. Define the metrics you want to test, such as response times or error rates. Choose a preferred tool that integrates well with your desired programming tools and frameworks. Write automated test scripts which simulate user interactions against the user interface and APIs of the system. These scripts should be regularly run by the synthetic testing tool in the testing environment. Synthetic tests can also be used to perform continuous application performance monitoring in production environments for observability purposes.

Related information:

- AWS Well-Architected Performance Pillar: PERF01-BP06 Benchmark existing workloads
- AWS Well-Architected Reliability Pillar: REL12-BP03 Test functional requirements

- Behavior Driven Development (BDD)
- Amazon CloudWatch Synthetics
- AWS Deployment Pipeline Reference Architecture
- Getting started with testing serverless applications

[QA.FT.4] Balance developer feedback and test coverage using advanced test selection

Category: OPTIONAL

In traditional software models, regression testing was a distinct form of functional testing, designed to ensure that new code integrations did not disrupt existing system functionalities. In a DevOps model, there is a new perspective: regression testing is no longer a testing activity with human involvement. Instead, every change triggers automated pipelines that conduct a new cycle of tests, making each pipeline execution effectively a *regression test*. As systems become more complex over time, so do the test suites. Running all tests every time a change is made can become time-consuming and inefficient as test suites grow, slowing down the development feedback loop.

Before choosing to implement advanced test selection methods using machine learning (ML). you should first optimize test execution through parallelization, reducing stale or ineffective tests, improving the infrastructure the tests are run on, and changing the order of tests to optimize for faster feedback. If these methods do not produce sufficient outcomes, there are algorithmic and ML methods that provide advanced test selection capabilities.

Test Impact Analysis (TIA) offers a structured approach to advanced test selection. By examining the differences in the codebase, TIA determines the tests that are most likely to be affected by the recent changes. This results in running only a relevant subset of the entire test suite, ensuring efficiency without the need for machine learning models.

Predictive test selection is an evolving approach to test selection which uses ML models trained on historical code changes and test outcomes to determine how likely a test is to reveal errors based on the change. This results in a subset of tests to run tailored to the specific change that are most likely to detect regressions. Predictive test selection strikes a balance between providing faster feedback to developers and thorough test coverage.

Using ML for this purpose introduces a level of uncertainty into the quality assurance process. If you do choose to implement predictive test selection, we recommend putting additional controls in place, including:

- Add manual approval stages that require developers to assess and accept the level of tests that will be run before they run. These manual approvals allow the team to decide if the test coverage trade-off makes sense and to accept the risk for the given change.
- Provide eventual consistency of test results by running the full set of tests asynchronously outside of the development workflow. If there are tests that fail at this stage, provide feedback to the development team so that they can triage the issues and decide if they need to roll back the change.
- We do not recommend using predictive test selection to exclude security-related tests or relying on this approach for sensitive systems which are critical to your business.

Related information:

- Predictive Test Selection
- Machine Learning - Amazon Web Services
- The Rise of Test Impact Analysis

Anti-patterns for functional testing

- **Over indexing on coverage metrics:** Relying heavily on test coverage metrics can lead teams to believe that the software is comprehensively tested. However, high test coverage might not catch all potential defects, as tests can run code without truly validating its correctness. This can result in overlooked defects and misplaced trust in the efficacy of the tests. To address this issue, incorporate regular test suite reviews focusing on meaningful assertions, rather than just coverage percentages. Introducing additional metrics like mutation testing scores can help to continuously measure and assess unit tests to help ensure they are meaningful. Improve the effectiveness of tests over time by tracking the number of defects escaping to production against test coverage percentages.
- **Reactive test writing:** Adopting a strategy where tests are primarily written post-software development and after defects emerge can produce software with undiscovered bugs. While writing code before tests might seem time-efficient, it can lengthen overall development cycles, increase costs, and erode trust in software quality. Provide developers with training to encourage adopting both Test-Driven Development (TDD) and Behavior-Driven Development (BDD) practices to introduce testing early in the development lifecycle. These approaches emphasize designing and writing tests before code, ensuring that testing considerations are embedded from the start.

- **Only testing functional requirements:** Focusing only on functional tests while sidelining non-functional aspects like accessibility, performance, and security can lead to vulnerabilities, poor user experience, and performance issues. Ongoing feedback loops with end users is the key to ensuring a well-rounded testing approach that strikes a balance between functional and non-functional tests. Engage with users early in the development process, conduct experiments, and iterate based on their feedback. Assess the ratio of functional to non-functional test activities, and pay attention to post-release incidents related to non-functional aspects of the application.
- **Neglecting to address flaky tests:** Continuously unreliable or inconsistent test results, known as flaky tests, can diminish trust in the test suite. These tests can lead to wasted time addressing false positives and as teams become accustomed to flaky tests they may begin to ignore them. This can lead to real defects being missed or ignored, drawing parallels to the story of *The boy who cried wolf*. Recognize the importance of consistent test outcomes. Address flakiness by rigorously investigating and resolving its root causes of failing tests, refining test design, and ensuring the testing environment is stable and reproducible. Establish a policy to handle flaky tests, such as quarantining them until resolved, to maintain the integrity of the test suite.

Metrics for functional testing

- **Defect density:** The number of defects identified per unit of code, typically per thousand lines of code (KLOC). A high defect density can indicate areas of the code that might require additional scrutiny or rework. By focusing on these areas, teams can enhance code quality. Improve this metric by increasing the rigor of code reviews, using static code analysis tools, and ensure developers have access to training and best practices. To measure this metric, compare the number of defects to the size of the codebase in KLOC.
- **Test pass rate:** The percentage of test cases that pass successfully. This metric provides an overview of the software's health and readiness for release. A declining pass rate can indicate emerging issues or code instability. Monitoring the test pass rate helps to evaluate the effectiveness of quality assurance testing process. Measure this by comparing the number of successful tests to the total tests run.
- **Escaped defect rate:** The number of defects found by users post-release compared to those identified during testing. A higher rate can suggest gaps in the testing process and areas where user flows are not effectively tested. Track this metric by comparing the number of post-release defects to the total defects identified.
- **Test case run time:** The duration taken to run a test case or a suite of test cases. Increasing the duration can highlight bottlenecks in the test process or performance issues emerging in the software under test. Improve this metric by optimizing test scripts and the order they run in,

enhancing testing infrastructure, and running tests in parallel. Measure the timestamp difference between the start and end of test case execution.

- **Test coverage:** The percentage of the codebase tested. This is generally further segmented as the percentage of functions, statements, branches, or conditions. Test coverage gives an overview of potential untested or under-tested areas of the application. Improve this metric by prioritizing areas with lower coverage, using automation and tools to enhance coverage, and regularly review test strategies. Measure this metric by calculating the ratio of code covered by tests to the total lines of code in the application.
- **Feature-to-bug ratio:** The proportion of new features developed compared to the number of bugs fixed. This metric provides insight into the balance between innovation and quality assurance. A higher ratio might indicate a focus on feature development, while a lower ratio can suggest a phase of stabilization or significant technical debt. To strike a balance, align your software development strategy with business goals and feedback loops. Regularly assess the ratio to determine if there's a need to prioritize defect resolution over new feature development, or vice versa. Measure by dividing the total number of new features by the total number of bugs fixed in a given period.

Non-functional testing

Non-functional testing evaluates the quality attributes of software systems, emphasizing how a solution performs and operates in various environments rather than its functional capabilities. Such tests help ensure that software meets the desired performance, reliability, usability, and other non-functional standards. By implementing non-functional testing, teams can consistently achieve scalable and efficient software solutions that meet both user and business requirements, elevating the overall user experience and software reliability.

Topics

- Indicators for non-functional testing
- Anti-patterns for non-functional testing
- Metrics for non-functional testing

Indicators for non-functional testing

Evaluate system attributes such as performance, usability, and reliability to ensure software meets operational requirements. This testing dimension focuses on how the system behaves, rather than specific functionalities.

Indicators

- [QA.NT.1] Evaluate code quality through static testing
- [QA.NT.2] Validate system reliability with performance testing
- [QA.NT.3] Prioritize user experience with UX testing
- [QA.NT.4] Enhance user experience gradually through experimentation
- [QA.NT.5] Automate adherence to compliance standards through conformance testing
- [QA.NT.6] Experiment with failure using resilience testing to build recovery preparedness
- [QA.NT.7] Verify service integrations through contract testing
- [QA.NT.8] Practice eco-conscious development with sustainability testing

[QA.NT.1] Evaluate code quality through static testing

Category: FOUNDATIONAL

Static testing is a proactive method of assessing the quality of code without needing to run it. It can be used to test application source code, as well as other design artifacts, documentation, and infrastructure as code (IaC) files. Static testing allows teams to spot misconfigurations, security vulnerabilities, or non-compliance with organizational standards in these components before they get applied in a real environment.

Static testing should be available to developers on-demand in local environments, as well as automatically run in automated pipelines. Use static testing to run automated code reviews and detect defects early on to provide fast feedback to developers. This feedback enables developers to fix and remove bugs before deployment, which is much easier and cost effective than fixing them after deployment.

Use specialized static analysis tools tailored to the type of code you are using. For example, tools like AWS CloudFormation Guard and cfn-lint are designed to catch issues in AWS CloudFormation templates. These tools can be configured to detect issues like insecure permissions, enforcing tagging standards, or misconfigurations that could make infrastructure vulnerable. Keep your static analysis tools updated and regularly review their findings to adapt to changing infrastructure security and compliance best practices.

Related information:

- What is Amazon CodeGuru Reviewer?
- Checkov

[QA.NT.2] Validate system reliability with performance testing

Category: RECOMMENDED

Performance testing evaluates the responsiveness, throughput, reliability, and scalability of a system under a specific load. It helps ensure that the application performs adequately when it is subjected to both expected and peak loads without impacting user experience. Different performance tests should be run based on the nature of changes made to the system:

- **Load testing:** Performance tests evaluating the system's behavior under expected load, such as the typical number of concurrent users or transactions. Integrate automated load testing into your deployment pipeline, ensuring every change undergoes validation of system behavior under expected scenarios.
- **Stress testing:** Performance tests challenging the system by increasing the load beyond its normal operational capacity. Stress tests identify the system's breaking points, ensuring that even under extreme conditions, the system maintains functionality without abrupt crashes. Schedule stress tests after significant application changes, infrastructure modifications, or periodically—such as once a month—to prepare for unpredictable spikes in traffic or potential DDoS attacks.
- **Endurance testing:** Performance tests that monitor system behavior over extended periods of time under a specific load. Endurance tests help ensure that there are no latent issues, such as slow memory leaks or performance degradation, which might occur after prolonged operations. Monitor key performance indicators over time and compare against established benchmarks to identify latent issues. Schedule endurance tests after significant changes to the system, especially those that might introduce memory leaks or other long-term issues. Consider running them periodically—such as quarterly or biannually—to ensure system health over prolonged operations.

All performance tests should be run against a test environment mirroring the production setup. Use tailored performance testing tools for your application's architecture and deployment environment. Regularly analyze test results against historical benchmarks and take proactive measures to counteract performance regressions.

Related information:

- AWS Well-Architected Performance Pillar: PERF01-BP07 Load test your workload
- AWS Well-Architected Sustainability Pillar: SUS03-BP03 Optimize areas of code that consume the most time or resources

- AWS Well-Architected Reliability Pillar: REL07-BP04 Load test your workload
- AWS Well-Architected Reliability Pillar: REL12-BP04 Test scaling and performance requirements
- Ensure Optimal Application Performance with Distributed Load Testing on AWS
- Stress Testing Tools - AWS Fault Injection Service
- Find Expensive Code – Amazon CodeGuru Profiler
- Load test your applications in a CI/CD pipeline using CDK pipelines and AWS Distributed Load Testing Solution

[QA.NT.3] Prioritize user experience with UX testing

Category: RECOMMENDED

User experience (UX) testing provides insight into the system's user interface and overall user experience, ensuring that they align with the diverse requirements of its user base. Adopting UX testing ensures that as the system evolves, its design remains intuitive, functional, and inclusive for end users.

Recognize that UX is subjective and can vary based on demographics, tech proficiency, and individual preferences. Segment your tests to understand the diverse needs and preferences of your user base. This means creating different user profiles and scenarios, ensuring that the software is tested from multiple perspectives. There are various forms of non-functional UX tests which should be utilized to target specific improvements:

- **Usability testing:** UX tests determines the ease with which users can perform tasks using the application and evaluates if the interface is intuitive and user-friendly. Usability testing helps identify issues related to the application's design, navigation, and overall ease of use, ultimately leading to building a better product. Conduct usability testing by recruiting a diverse group of testing participants that represent the broader user base. Provide these users with typical tasks they would perform when using the application. Observe the testing participants and their interactions, note areas where they encounter challenges, confusion, or get frustrated. During observation, encourage the participants to verbalize their thought process as they perform the tasks. After the tasks are completed, conduct a brief feedback session to gather additional perspective on their use of the application. Use this data to drive user experience improvements and to fix any bugs that were discovered. To continuously gather feedback over time, ensure that there are mechanisms for users to provide feedback as they interact with the system.
- **Accessibility testing:** UX tests that evaluate the application to ensure that it can be accessed and used by everyone. Regularly review web content accessibility guidelines (WCAG) to ensure

compliance with the latest standards. To get started quickly, consider adopting an existing design system which incorporates accessibility best practices and a framework to create accessible applications, such as the Cloudscape Design System. Automate accessibility tests as a part of the development lifecycle using tools like Axe or WAVE. Adopt tools that evaluate specific accessibility standards, such as color contrast analyzing tools like WebAim. Consider regularly conducting manual exploratory tests using assistive technologies to capture issues that automated tools might miss.

Related information:

- Usability Evaluation Methods
- W3C standards
- WCAG 2.1 AA
- Web Accessibility Initiative (WAI)

[QA.NT.4] Enhance user experience gradually through experimentation

Category: RECOMMENDED

Enhancing user experience requires taking a methodical approach to assessing how users behave when using your application and developing features that resonate with users. The goal of running experiments is to identify and implement the best possible user experience based on indirect user behavior. With experiments, teams can proactively assess the impact of new features on a subset of users before a full-scale rollout, reducing the risk of making the change and negatively impacting user experience.

A popular technique for conducting experiments is A/B testing, also known as split testing. To run split testing experiments, present different versions of the application to a small segment of real users to gather detailed feedback on specific changes. This testing is done in a production environment alongside the production application. By directing only a small subset of the users to the version of the application with the change, teams are able to conduct experiments while hiding the new feature from the majority of the user base not included in the test. Testing a feature within a smaller sample, rather than the entire user group, minimizes potential disruptions and yields more detailed data in a real-world setting.

Teams can control the experiment using feature flags or dedicated tools like CloudWatch Evidently to control variables and traffic to the different versions of the application. Ensure the experiment

runs for the necessary duration to achieve statistical significance and use consistent metrics to track customer behavior across the variations to maintain accuracy. Compare the metrics after the experiment to make decisions.

Related information:

- Perform launches and A/B experiments with CloudWatch Evidently
- Feature Flags - AWS AppConfig
- Business Value is IT's Primary Measure of Progress
- Blog: Using A/B testing to measure the efficacy of recommendations generated by Amazon Personalize

[QA.NT.5] Automate adherence to compliance standards through conformance testing

Category: RECOMMENDED

Conformance testing, often referred to as compliance testing, verifies that a system meets internal and external compliance requirements. It compares the system's behaviors, functions, and capabilities with predefined criteria from recognized standards or specifications.

Conformance testing acts as a safeguard, ensuring that while agility is prioritized, compliance isn't compromised. There are many regulated industries, such as finance, healthcare, or aerospace, that have a strict set of compliance requirements which must be met when delivering software. Historically, balancing fast software delivery with stringent compliance was a challenge in these industries. Generating the documentation and proof required to maintain compliance was often a manual, time-intensive step that created a bottleneck at the end of the development lifecycle.

Conformance testing integrated into deployment pipelines provides a solution to this problem by automating the creation of compliance attestations and documentation. It can be used to meet both internal and external compliance requirements. Start by determining both internal (for example, risk assessment policies, or change management procedures) and external standards (for example, GxP for life sciences). Prioritize and choose the relevant parts of the standards which can be automated (for example, GxP Installation Qualification report). Ensure that conformance tests remain current by updating them according to evolving standards.

Use the data at your disposal, including APIs, output from other forms of testing, and possibly additional data from IT Service Management (ITSM) and Configuration Management Databases (CMDB). Embed conformance testing scripts into deployment pipelines to generate real-time compliance attestations and documentation using this data. Consider using machine-readable

markup languages, such as JSON and YAML, to store the compliance artifacts. If the markup languages are not considered sufficiently human readable by auditors, then retain the ability to convert these markdown files into another format. This conversion can then be done when needed, not as a default step, removing the burden of document management where it is not absolutely necessary.

Related information:

- Wikipedia - Conformance testing
- Qualification Strategy for Life Science Organizations
- Automating the Installation Qualification (IQ) Step to Expedite GxP Compliance
- Automating GxP compliance in the cloud: Best practices and architecture guidelines
- Automating GxP Infrastructure Installation Qualification on AWS with Chef InSpec

[QA.NT.6] Experiment with failure using resilience testing to build recovery preparedness

Category: RECOMMENDED

Resilience testing deliberately introduces controlled failures into a system to gauge its ability to withstand failure and recover during disruptive scenarios. Simulating failures in different parts of the system provides insight into how failures propagate and affect other components. This helps identify bottlenecks or single points of failure in the system.

Before initiating any resilience tests, especially in production, understand the potential impact on the system, dependent systems, and the operating environment. Start small with less invasive tests and gradually expand the scope and frequency of these tests. This iterative approach allows you to understand the ramifications of a particular failure and ensures that you don't accidentally cause a significant disruption.

There are various types of resilience testing:

- **Chaos engineering:** Resilience tests using fault injection to introduce controlled failures into the system. This can include simulating service failures, region outages, single node failures, network outages, or complete failovers of connected systems. Controlled failures enable teams to identify system vulnerabilities, ensuring weaknesses introduced by deployments and infrastructure changes are mitigated. Fault injection tools, such as AWS Fault Injection Service and AWS Resilience Hub, can assist in conducting these experiments. Embed the use of these tools into automated pipelines to run fault injection tests after deployment.

- **Data recovery testing:** Resilience tests that verify that specific datasets can be restored from backups. Data recovery tests ensure that the backup mechanism is effective and that the restore process is reliable and performant. Schedule data recovery tests periodically, such as monthly, quarterly, or after major data changes or migrations. Initiate these tests through deployment pipelines. For example, after a database schema deployment, run a test to ensure that the new schema doesn't compromise backup integrity.
- **Disaster recovery testing:** Resilience tests that help ensure the entire system can be restored and made operational after large scale events, such as data center outages. This includes activities such as restoring systems from backups, switching to redundant systems, or transitioning traffic to a disaster recovery environment. These tests are extensive and therefore run less frequently, such as semi-annually or annually. When running in production environments, these tests are usually performed during maintenance windows or off-peak hours, and stakeholders are informed well in advance. Use disaster recovery tools, such as AWS Elastic Disaster Recovery and AWS Resilience Hub, to assist with planning, coordinating, and performing recovery actions. While integrating these tests fully into deployment pipelines is rare, it is possible to automate sub-tasks or preliminary checks. For example, after significant infrastructure changes, a test might check the functionality of failover mechanisms to ensure they still operate as expected. It can often be more effective to trigger these tests based on events or manual triggers, especially earlier on in your DevOps adoption journey.

Before running resilience tests in either test or production environments, consider the use case, the benefits of the test, and the system's readiness. Regardless of the target environment, always inform all stakeholders of the system before executing significant resilience tests. Have a pre-prepared, comprehensive communication plan in the event of unforeseen challenges or downtime. We recommend initially running resilience tests in a test environment to get an understanding of their effects, refine the testing process, and train the team.

After gaining confidence in the testing process and building the necessary observability and rollback mechanisms to run them safely, consider running controlled tests in production to gain the most accurate representation of recovery scenarios in real-world settings. When executing in production, limit the impact of your tests. For example, if you are testing the resilience of a multi-Region application, don't bring down all Regions at once. A better approach would be to start with one Region, observe its behavior, and learn from the results. After running resilience tests, conduct a retrospective to understand what went well, any unexpected behaviors, improvements that can be made, and to plan work to enhance both the system's resilience and the testing process itself.

Related information:

- AWS Well-Architected Reliability Pillar: REL09-BP04 Perform periodic recovery of the data to verify backup integrity and processes
- AWS Well-Architected Reliability Pillar: REL12-BP05 Test resiliency using chaos engineering
- AWS Well-Architected Reliability Pillar: REL13-BP03 Test disaster recovery implementation to validate the implementation
- AWS Fault Isolation Boundaries
- Well-Architected Lab - Testing for Resiliency
- Fault testing on Amazon EBS
- Chaos experiments on Amazon RDS using AWS Fault Injection Service
- Chaos Testing with AWS Fault Injection Service and AWS CodePipeline

[QA.NT.7] Verify service integrations through contract testing

Category: RECOMMENDED

Contract testing helps ensure that different system components or services can seamlessly communicate and are compatible with each other. This involves creating contracts that detail interactions between services, capturing everything from request structures to expected responses. As changes are made, these contracts can be used by producing (teams that expose the API) and consuming (teams that use the API) services to ensure they remain compatible. Contract testing provides a safety net for both producers and consumers by ensuring changes in one do not adversely impact the other. This creates a culture of collaboration between teams while providing faster feedback for identifying integration issues earlier in the development lifecycle.

There are different types of contract testing. In consumer-driven contract testing, the consumer of a service dictates the expected behaviors of the producer. This is contrasted with provider-driven approaches, where the producer service determines its behaviors without explicit input from its consumers. Consumer-driven contract testing is the type we generally recommend, as designing contracts with the consumer in mind ensures that APIs are tailored to the customer's actual needs, making integrations more intuitive.

Begin by clearly defining contracts between your services. Use purpose-built contract testing tools, such as Pact or Spring Cloud Contract, to simplify managing and validating contracts. When any modification is made in a producer service, run contract tests to assess the contracts' validity. Similarly, before a consumer service integrates with a producer, run the relevant contract tests to guarantee they'll interact correctly. This process allows producers to maintain backwards compatibility, while allowing consumers to identify and fix potential integration issues early in

the development lifecycle. Embed contract testing into your deployment pipeline. This ensures continuous validation of contracts as changes are made to services, promoting a continuous and consistent integration process.

Related information:

- AWS Well-Architected Reliability Pillar: REL03-BP03 Provide service contracts per API
- Introduction To Contract Testing With Examples
- CloudFormation Command Line Interface: Testing resource types using contract tests

[QA.NT.8] Practice eco-conscious development with sustainability testing

Category: OPTIONAL

Sustainability testing ensures that software products contribute to eco-conscious and energy-efficient practices that reflect a growing demand for environmentally responsible development. It is a commitment to ensuring software development not only meets performance expectations but also contributes positively to the organization's environmental goals. In specific use cases, such as internet of things (IoT) and smart devices, software optimizations can directly translate to energy and cost savings while also improving performance.

Sustainability testing encompasses:

- **Energy efficiency**: Create sustainability tests which ensure software and infrastructure minimize power consumption. For instance, AWS Graviton processors are designed for enhanced energy efficiency. They offer up to 60% less energy consumption for similar performance compared to other EC2 instances. Write static analysis tests that focus on improving sustainability by verifying that infrastructure as code (IaC) templates are configured to use energy efficient infrastructure.
- **Resource optimization:** Sustainable software leverages hardware resources, such as memory and CPU, without waste. Sustainability tests can enforce right-sizing when deploying infrastructure. For example, Amazon EC2 Auto Scaling ensures compute resources align with actual needs, preventing over-provisioning. Similarly, AWS Trusted Advisor offers actionable insights into resource provisioning based on actual consumption patterns.
- **Data efficiency:** Sustainability testing can assess the efficiency of data storage, transfer, and processing operations, ensuring minimal energy consumption. Tools like the AWS Customer Carbon Footprint Tool offer insights into the carbon emissions associated with various AWS services, such as Amazon EC2 and Amazon S3. Teams can use these insights to make informed optimizations.

- **Lifecycle analysis:** The scope of testing extends beyond immediate software performance. For instance, the AWS Customer Carbon Footprint Tool can provide insights into how using AWS services impacts carbon emissions. This information can be used to compare this usage with traditional data centers. Metrics from this tool can be used to inform decisions throughout the software lifecycle, ensuring that environmental impact remains minimal from inception to decommissioning of resources.

Sustainability testing should use data provided by profiling applications to measure their energy consumption, CPU usage, memory footprint, and data transfer volume. Tools such as Amazon CodeGuru Profiler and SusScanner can be helpful when performing analysis and promotes writing efficient, clean, and optimized code. Combining this data with suggestions from AWS Trusted Advisor and AWS Customer Carbon Footprint Tool can lead to writing tests which can enforce sustainable development practices.

Sustainability testing is still an emerging quality assurance practice. This indicator is beneficial for organizations focusing on environmental impact. We think that by making sustainability a core part of the software development process, not only do we contribute to a healthier planet, but often, we also end up with more efficient and cost-effective solutions.

Related information:

- AWS Well-Architected Performance Pillar: PERF02-BP04 Determine the required configuration by right-sizing
- AWS Well-Architected Performance Pillar: PERF02-BP06 Continually evaluate compute needs based on metrics
- AWS Well-Architected Sustainability Pillar: SUS03-BP03 Optimize areas of code that consume the most time or resources
- AWS Well-Architected Sustainability Pillar: SUS06-BP01 Adopt methods that can rapidly introduce sustainability improvements
- AWS Well-Architected Cost Optimization Pillar: COST09-BP03 Supply resources dynamically
- Sustainability Scanner (SusScanner)
- AWS Well-Architected Framework - Sustainability Pillar
- AWS Customer Carbon Footprint Tool
- Sustainable Cloud Computing
- Reducing carbon by moving to AWS

Anti-patterns for non-functional testing

- **Mistaking infrastructure resilience with system reliability**: While architectural traits like high availability and fault tolerance enhance a system's resilience, enabling it to recover from external disruptions, they do not inherently ensure application reliability. While infrastructure resilience ensures a system can recover from failure, application reliability ensures that it can consistently meet runtime expectations, especially under varying loads. Assessing the reliability of a system requires targeted non-functional performance tests to evaluate responsiveness, stability, and speed under various loads. Measure the impact these factors have on the system, using observability tools that offer insights into real-time operational efficiency, aiding in optimization.
- **Overlooking real-world conditions during testing**: Testing exclusively in controlled environments without considering real-world variables and unpredictability can lead to a false sense of assurance. Tests must account for diverse user behaviors, different network conditions, and the wide range of device combinations. Integrating real-world variables into testing ensures that software releases are robust and reliable in actual deployment scenarios. The most effective strategy to achieve this is by balancing testing in controlled environments with testing in production.
- **Ignoring using observability for performance tuning**: Resource optimization shouldn't be restricted to the early stages of the development lifecycle. As applications are used in production, their resource requirements may scale and lead to different outcomes that were not tested in a controlled environment. Real data regarding non-functional attributes, such as resource allocation, performance, compliance, sustainability and cost should be periodically reviewed and adjusted after deployment. Tools like AWS Trusted Advisor, AWS Compute Optimizer, and AWS Customer Carbon Footprint Tool can be used to tighten the relationship between quality assurance and observability.
- **Not gathering genuine user feedback**: Relying solely on internal feedback for non-functional aspects can introduce biases and overlook real user pain points. Collect, analyze, and act on genuine user feedback regarding performance, usability, and other non-functional attributes. This feedback loop ensures software development remains aligned with user expectations, optimizing the overall user experience.

Metrics for non-functional testing

- **Availability**: The percentage of time a system is operational and accessible to users. High availability helps to maintain user trust and ensure business continuity. A decrease in this metric can signify issues with infrastructure reliability or application stability. Enhance availability by

implementing redundant architecture, employing failover strategies, and ensuring continuous monitoring. Calculate the availability percentage by dividing the total time the system was operational by the overall time period being examined, and then multiply the result by 100.

- **Latency**: The time it takes for a system to process a given task. This metric specifically considers the time taken from when a request is made to when a response is received. This metric offers insight into the responsiveness of an application, affecting user experience and system efficiency. Improve this metric by optimizing application code, streamline database operations, utilize efficient algorithms, and scaling infrastructure. Using percentiles and trimmed mean are good statistics for this measurement.
- **Cyclomatic complexity**: Cyclomatic complexity counts the distinct paths through a code segment. It reflects the complexity in the code's decision-making structure. Higher values can indicate code that can be harder to maintain, understand, or test, increasing the likelihood of errors. Improve this metric by simplifying code where possible by performing regular code reviews and refactoring sessions. In these sessions, break down complex code into smaller, more manageable functions and reduce nested conditions and loops. The complexity is calculated using the difference between the number of transitions between sections of code (edges) and the number of sequential command groups (nodes), adjusted by twice the number of connected components. We recommend adopting tools to measure complexity automatically.
- **Peak load threshold**: Represents the maximum number of simultaneous users or requests a system can handle before performance degrades. Understanding this threshold aids in capacity planning and ensures the system can cope with usage spikes. Increase the peak load threshold by conducting load tests with increasing numbers of users, identifying and resolving bottlenecks. Track this metric by stress testing the system and observing the point of performance degradation.
- **Test case run time**: The duration taken to run a test case or a suite of test cases. Increasing duration may highlight bottlenecks in the test process or performance issues emerging in the software under test. Improve this metric by optimizing test scripts and the order they run in, enhancing testing infrastructure, and running tests in parallel. Measure the timestamp difference between the start and end of test case execution.
- **Infrastructure utilization**: Percentage utilization of infrastructure resources such as CPU, memory, storage, and bandwidth. Infrastructure utilization helps in understanding if there are over-provisioned resources leading to cost overhead or under provisioned resources that could affect performance. Calculate this metric for each type of resource (such as CPU, RAM, or storage) to get a comprehensive understanding of infrastructure utilization.

- **Time to restore service**: The time taken to restore a service to its operational state after an incident or failure. Faster time to restore can indicate a more resilient system and optimized incident response processes. An ideal time to restore service must be capable of meeting recovery time objectives (RTO). RTO is the duration the system must be restored after a failure to avoid unacceptable interruptions to business continuity. RTO takes into account the criticality of each system, while balancing cost, risk, and operational needs. Measure the time duration from the moment the service disruption is reported to when the service is fully restored.
- **Application performance index (Apdex)**: Measures user satisfaction with application responsiveness using a scale from 0 to 1. A higher Apdex score indicates better application performance, likely resulting in improved user experience, while a lower score means that users might become frustrated.

  To determine the Apdex score, start by defining a target response time that represents an acceptable user experience for your application. Then, categorize every transaction in one of three ways:

  - **Satisfied**, if its response time is up to and including the target time.
  - **Tolerating**, if its response time is more than the target time but no more than four times the target time.
  - **Frustrated**, for any response time beyond four times the target time.

  Calculate the Apdex score by adding the number of *Satisfied* transactions with half the *Tolerating* transactions. Then, divide this sum by the total number of transactions. Continuously monitor and adjust your target time based on evolving user expectations and leverage the score to identify and rectify areas that contribute to user dissatisfaction.

Security testing

Security testing identifies potential vulnerabilities, threats, risks, and other security weaknesses in a system. It safeguards the integrity, confidentiality, and availability of the system and its data. Inspected components include safety faults, infrastructure weaknesses, network threats, software vulnerabilities, and other hazards. Effective security testing involves a mix of manual penetration testing and automated vulnerability scans, offering insights into potential breaches or exposures.

Topics

- Indicators for security testing
- Anti-patterns for security testing

- Metrics for security testing

Indicators for security testing

Identify software vulnerabilities, threats, and risks to safeguard against unauthorized access and misconfiguration. This specialized testing aims to identify potential security flaws and reinforce the system's defenses.

Indicators

- [QA.ST.1] Evolve vulnerability management processes to be conducive of DevOps practices
- [QA.ST.2] Normalize security testing findings
- [QA.ST.3] Use application risk assessments for secure software design
- [QA.ST.4] Enhance source code security with static application security testing
- [QA.ST.5] Evaluate runtime security with dynamic application security testing
- [QA.ST.6] Validate third-party components using software composition analysis
- [QA.ST.7] Conduct proactive exploratory security testing activities
- [QA.ST.8] Improve security testing accuracy using interactive application security testing

[QA.ST.1] Evolve vulnerability management processes to be conducive of DevOps practices

Category: FOUNDATIONAL

Vulnerability management requires an ongoing, iterative process consistent with agile development practices. The goal is to discover potential vulnerabilities across networks, infrastructures, and applications, and to prioritize and take action on them.

Automated vulnerability scanning must be integrated into deployment pipelines to provide feedback to developers regarding security vulnerabilities and improvements early on. This minimizes extensive security evaluations during deployment and is consistent with the DevOps *shift left* approach—addressing security problems early on in the development process. Choose vulnerability scanning tools that are compatible with your existing technology and platforms. For instance, if Amazon CodeCatalyst is your pipeline tool of choice, verify that the chosen vulnerability scanning tool has a CodeCatalyst plugin or API integration capability. If vulnerabilities are detected during a build, the pipeline should automatically generate alerts, allowing developers to address issues quickly.

If you use issue-tracking systems like Jira or CodeCatalyst Issues, it can be beneficial to automatically generate tickets to assist developers with tracking issues. When a vulnerability is detected, an automated ticket should be generated, tagged with severity, and assigned to the appropriate developer or team. Use vulnerability management dashboards to consistently monitor and analyze threats. Regular reports should detail vulnerability trends, ensuring vulnerabilities are not reintroduced and pinpointing recurrent security challenges.

To effectively practice vulnerability management in a DevOps environment, it's important to adopt a culture where security is everyone's responsibility. Development and security teams need collaboration, with clear delineations for security issue handoff and ownership. In a DevOps model, distributed development teams take on security responsibilities for their products. Centralized security teams often become enabling teams, offering training, insights, and support. They can also take on the responsibilities of a security platform team, producing reusable components, improving efficiency, reducing duplication of work, and overall providing autonomy to distributed teams so that they can efficiently secure their products.

Related information:

- Enterprise DevOps: Why You Should Run What You Build
- Automated Software Vulnerability Management - Amazon Inspector

[QA.ST.2] Normalize security testing findings

Category: FOUNDATIONAL

Effective vulnerability management requires clarity and consistency. Given the diversity of security testing tools in a DevOps environment, findings often emerge from different sources and in different formats. This diversity of tooling can introduce confusion and inefficiency into risk management processes. Having a common framework for normalizing the interpretation and ranking of vulnerabilities from diverse security testing tools provides a systematic approach to risk management and mitigation. Normalization is not just about consistency, it helps ensure that every identified vulnerability is understood, categorized, and managed according to its threat level.

Begin by selecting a recognized scoring system, such as the Common Vulnerability Scoring System (CVSS), as the baseline for vulnerability ranking. This will provide a universal language for risk assessment and prioritization. Many modern security tools have built-in integrations with popular scoring systems. Configure your tools to automatically map their findings to the chosen system, ensuring uniformity across all results. It is important to periodically review the normalization process, updating it as required and ensuring alignment with industry best practices.

Use tools that can automatically translate findings into the standardized format. Integrations like the Static Analysis Results Interchange Format (SARIF) or OCSF Schema can assist with this. These tools can enable centralizing findings from different sources into a single dashboard or reporting platform to create a unified view of the security posture which can streamline the prioritization and remediation process.

By adopting a systematic approach to normalization, organizations can verify that their response to vulnerabilities is consistent, effective, and aligned with the actual risks posed to the system. Ensure that everyone involved in the security process understands the chosen scoring system and knows how to interpret it. Regular workshops or training sessions can help ensure ongoing alignment.

Related information:

- NIST Common Vulnerability Scoring System (CVSS)
- MITRE Common Weakness Scoring System (CWSS™)
- Static Analysis Results Interchange Format (SARIF)
- OCSF Schema
- OCSF GitHub

[QA.ST.3] Use application risk assessments for secure software design

Category: FOUNDATIONAL

Application risk assessments integrate security considerations directly into the software development lifecycle. At the earliest stages of the development lifecycle, design reviews focus on the planned architecture, features, and flow of the application. During these reviews, security experts should assist with making design choices to prevent introducing weak points that could introduce vulnerabilities. The primary goal is to make security-centric design decisions, eliminating vulnerabilities before they're developed.

After the design phase, threat modeling dives deeper into potential security threats that the finalized design might face. This results in a list of possible attack vectors, identifying how an attacker might exploit vulnerabilities. An inverse approach to threat modeling is attack modeling, which identifies specific attacks or vulnerabilities and examines how they can be exploited. Both methods offer insights into possible vulnerabilities and guide developing protective measures.

Once vulnerabilities are identified through design reviews and potential threats through modeling, these insights should directly inform the software's security requirements. As applications

evolve or as new threats emerge, periodically revisit and update both functional and non-functional requirements. Functional requirements involves measures like input validation, session management, or error handling. Non-functional requirements includes making changes that impact to performance, scalability, and reliability under security threats.

Translate identified risks into actionable user stories that detail potential abuse or misuse scenarios. Add these stories into the backlog for the team to address during development. Attach a test case to each story to validate its effective resolution, establishing a clear *definition of done* for developers to adhere to.

Related information:

- Threat Composer
- Threat modeling for builders
- AWS Security Maturity Model - Threat Modeling
- How to approach threat modeling

[QA.ST.4] Enhance source code security with static application security testing

Category: FOUNDATIONAL

Static Application Security Testing (SAST) is a proactive measure to identify potential vulnerabilities in your source code before they become part of a live application. SAST is a specialized form of non-functional static testing that enables you to analyze the source or binary code for security vulnerabilities, without the need for the code to be running.

Choose a SAST tool, such as Amazon CodeGuru Security, and use it to scan your application using an automated continuous integration pipeline. This enables identifying security vulnerabilities in the source code early in the development process. When selecting a SAST tool, consider its compatibility with your application's languages and frameworks, its ease of integration into your existing toolsets, its ability to provide actionable insights to fix vulnerabilities, and false positive rates. False positive rate is one of the most important metrics to focus on when selecting a SAST tool, as this can result in findings and alerts of potential security issues that are not actually exploitable. False positives can erode trust in the adoption of security testing.

To prevent developer burnout and backlash due to overwhelming false positives or a high rate of alerts in existing applications, introduce SAST rulesets incrementally. Start with a core set of rules and expand as your team becomes more accustomed to addressing security testing feedback. This iterative approach also allows teams to validate the tool's findings and fine-tune its sensitivity over

time. Regularly update and refine enabled SAST rules to maintain its effectiveness in identifying potential security issues.

Related information:

- Security in every stage of the CI/CD pipeline: SAST
- Amazon CodeGuru Security
- Security scans - CodeWhisperer
- Blog: Building end-to-end AWS DevSecOps CI/CD pipeline with open source SCA, SAST and DAST tools

[QA.ST.5] Evaluate runtime security with dynamic application security testing

Category: FOUNDATIONAL

While other forms of security testing identifies potential vulnerabilities in code that hasn't been run, dynamic application security testing (DAST) detects vulnerabilities in a running application. DAST works by simulating real-world attacks to identify potential security flaws while the application is running, enabling uncovering vulnerabilities that may not be detectable through static testing. By proactively uncovering security weaknesses during runtime, DAST reduces the likelihood of vulnerabilities being exploited in production environments.

Begin by choosing a DAST tool that offers broad vulnerability coverage, including recognition of threats listed in the OWASP Top 10. When selecting a tool, verify that it can integrate seamlessly with your existing toolsets, authentication mechanisms, and protocols used by your systems. With DAST, false positive rates are generally lower than other forms of security testing since it actively exploits known vulnerabilities. Still, pay attention to false positive rates and the tool's ability to provide actionable insights. False positives can erode developer trust in security testing while detracting from genuine threats and consuming unnecessary resources.

Related information:

- Security in every stage of the CI/CD pipeline: DAST
- Building end-to-end AWS DevSecOps CI/CD pipeline with open source SCA, SAST and DAST tools

[QA.ST.6] Validate third-party components using software composition analysis

Category: FOUNDATIONAL

The use of open-source software and third-party components accelerates the software development process, but it also introduces new security and compliance risks. Software Composition Analysis (SCA) is used to assess these risks and verify that external dependencies being used do not have known vulnerabilities. SCA works by scanning software component inventories, such as software bill of materials software bill of materials (SBOM) and dependency manifest files.

When selecting a SCA tool, focus on tools that provide the most comprehensive vulnerability database, pulling from sources such as the National Vulnerability Database (NVD) and Common Vulnerabilities and Exposures (CVE). The tool will need to integrate with your existing toolsets, frameworks, and pipelines, as well as provide both detection and remediation guidance for vulnerabilities. These feedback mechanisms enable teams to detect and mitigate vulnerabilities, maintaining the software's integrity without impacting development velocity.

Integrate SCA into the continuous integration pipeline to automatically scan changes for vulnerabilities. Use SCA to scan existing repositories periodically to verify that existing codebases maintain the same security standards as newer developments. Centrally storing SBOMs also offers unique advantages for assessing vulnerabilities at scale. While scanning repositories and pipelines can capture vulnerabilities in active projects, centralized SBOMs act as a consistent, versioned record of all software components used across various projects and versions. It provides a holistic view of all dependencies across different projects, making it easier to manage and mitigate risks at an organizational level. Instead of scanning every repository individually, centralized scanning of SBOMs offers a consolidated method to assessing and remediating vulnerabilities.

Related information:

- Security in every stage of the CI/CD pipeline: SCA
- Building end-to-end AWS DevSecOps CI/CD pipeline with open source SCA, SAST and DAST tools

[QA.ST.7] Conduct proactive exploratory security testing activities

Category: RECOMMENDED

Conduct frequent exploratory security testing activities, encompassing penetration testing, red teaming, and participation in vulnerability disclosure or bug bounty programs.

Penetration tests use ethical hackers to detect vulnerabilities in system or networks by mimicking potential threat actor actions. These exploratory security tests reveal weaknesses in the system

using the ingenuity of human testers. Deployment pipelines can trigger the penetration testing process and wait for an approval to help ensure that vulnerabilities are identified and fixed before code moves to the next stage. Automation can be used to run repetitive, baseline tests, such as dynamic application security testing, to enable human testers to focus on more complex scenarios. Review the AWS Customer Support Policy for Penetration Testing before running penetration tests against AWS infrastructure. Penetration testing is most effective when you need a broad review of the application or system against known vulnerabilities.

Going beyond the scope of penetration tests, red teaming emulates real-world adversaries in a full-scale simulation, targeting the organization's technology, people, and processes. Red teaming is more focused than penetration testing, targeting specific vulnerabilities by allocating more resources, spending more time, and examining additional attack vectors. This includes potential threats from internal sources, such as lost devices, external sources like phishing campaigns, and those arising from social engineering tactics. This approach provides insights into how threat actors might exploit weaknesses and bypass defenses in a real-world scenario. Red teaming evaluates the broader resilience of an application or system, including its resistance to sophisticated attacks that span the entire organization's security posture.

Vulnerability disclosure and bug bounty programs invite external researchers to examine your software, complementing and often surpassing internal security evaluations. Researchers who participate in these programs not only identify potential exploits but also verify them, resulting in higher fidelity findings. The person who identified the vulnerability does not disclose it publicly for a set amount of time, allowing a patch to be rolled out before the information is disclosed publicly, and in some cases will receive compensation for their efforts. These programs foster a culture of openness and continuous improvement, emphasizing the importance of external feedback in maintaining secure systems.

The findings from exploratory security testing should be communicated to development teams as soon as findings are available, allowing for quick remediation and learning.

Related information:

- AWS Well-Architected Security Pillar: SEC11-BP03 Perform regular penetration testing
- Security in every stage of the CI/CD pipeline: Penetration Testing and Red Teaming
- AWS Penetration Testing: A DIY Guide for Beginners
- AWS Customer Support Policy for Penetration Testing
- AWS Cloud Security - Vulnerability Reporting

- AWS BugBust
- AWS CloudSaga - Simulate security events in AWS
- RFC 9116 - A File Format to Aid in Security Vulnerability Disclosure
- Amazon's approach to security during development: Penetration Testing

[QA.ST.8] Improve security testing accuracy using interactive application security testing

Category: OPTIONAL

Interactive Application Security Testing (IAST) offers an inside-out approach to application security testing by combining strengths of both Static Application Security Testing (SAST) and Dynamic Application Security Testing (DAST). While SAST examines source code to identify vulnerabilities and DAST inspects a running system, IAST uses embedded agents which has access to application code, system memory, stack traces, and requests and responses to monitor system behavior during runtime.

Unlike other automated security testing methods that can produce false alarms, IAST's real-time observability from within the application provides a contextual understanding that reduces false positive rates. When vulnerabilities are detected, IAST provides deeper insight into how the system is impacted, providing proof that the vulnerabilities flagged are genuine and actionable.

Include IAST agents to the system during the build process to actively monitor the system in the testing environments. These agents provide additional observability to the system that is used to validate vulnerabilities. After the application is deployed to production, these agents should be turned off or set to a passive mode to avoid any performance overhead. IAST is optional for DevOps adoption, as many organizations find sufficient coverage with SAST and DAST.

Related information:

- Security in every stage of the CI/CD pipeline: IAST

Anti-patterns for security testing

- **Overconfidence in test results**: Being overly confident about a low false-positive rate and not considering the potential of false negatives can lead to genuine threats being ignored, which may be exploited by attackers. Regularly re-evaluate and adjust security testing tools and methodologies. Consider periodic third-party security audits and human-driven exploratory testing to get an external perspective on the system's security posture.

- **Not considering internal threats**: Focusing security testing solely on external threats while neglecting potential insider threats, whether malicious or unintentional, can lead to unmitigated attack vectors that can be as damaging as external attacks. Testing should encompass all potential threat actors. Include scenarios in your testing strategy that emulate insider threats, such as permissions escalation, data exfiltration from internal roles, and social engineering. Continuously raise awareness, train employees on best practices, and regularly review access permissions.
- **Neglecting software supply chain attacks**: Not regularly monitoring or safeguarding against potential threats in the software supply chain, from third-party libraries to development tools. Supply chain attacks have become increasingly prevalent, and they can compromise systems even if the organization's proprietary code is secure. Adopt a comprehensive software supply chain security strategy, including regularly updating and auditing third-party components, monitoring development environments, and ensuring secure software development practices are followed by all components used to build, test, deploy, and operate your systems.

Metrics for security testing

- **Escaped defect rate**: The number of defects found by users post-release compared to those identified during testing. A higher rate can suggest gaps in the testing process and areas where user flows are not effectively tested. An effective security testing process should aim to reduce the escaped defect rate by increasing the vulnerability discovery rate. Track this metric by comparing the number of post-release defects to the total defects identified.
- **False positive rate**: The ratio of identified security threats that are later determined to be non-actionable or actual threats. Too many false positives can lead to *alert fatigue*, causing genuine threats to be overlooked. This metric indicates the accuracy and relevance of your security testing tools. Compare the number of false positives against the total number of security alerts raised over a period, such as monthly or quarterly.
- **Mean time to detect**: The average time it takes for an organization to detect a security breach or vulnerability. A shorter mean time indicates that testing, monitoring, and alert systems are effective, leading to faster detection of issues. A longer mean time may expose the organization to greater risks. With effective security testing, you can detect anomalies faster—ideally before they are deployed to production. Measure the time from when a vulnerability occurs to the time it is detected. Calculate the average detection time over a defined period, such as monthly or quarterly.
- **Mean time to remediate**: The average time it takes for an organization to address and resolve a detected security issue. A shorter mean time implies that once a vulnerability is detected,

the organization can act quickly to mitigate risks. A longer mean time suggests potential inefficiencies in the incident response process. Having a strong security testing practices in place ensures that you are well-equipped to understand and remediate vulnerabilities when they are detected, leading to faster resolution. Measure the time from when a security issue is detected to when it is resolved. Calculate the average remediation time over a defined period, such as monthly or quarterly.

- **Test pass rate**: The percentage of test cases that pass successfully. This metric provides an overview of the software's health and readiness for release. If both the test pass rate and the escaped defect rate are high, it could indicate that your security tests are not effective enough. Conversely, a declining pass rate can indicate emerging security issues. Monitoring the test pass rate helps to evaluate the effectiveness of quality assurance testing process. Measure this by comparing the number of successful tests to the total tests run.
- **Test case run time**: The duration taken to run a test case or a suite of test cases. Increasing duration may highlight bottlenecks in the test process or performance issues emerging in the software under test. Improve this metric by optimizing test scripts and the order they run in, enhancing testing infrastructure, and running tests in parallel. Measure the timestamp difference between the start and end of test case execution.
- **Vulnerability discovery rate**: The number of vulnerabilities discovered during the testing phase per defined time period or release. This metric helps assess the effectiveness of the security testing process. A higher rate, especially when paired with a low false positive rate, may indicate a very effective testing process, though if it remains high over time, it might indicate recurring coding vulnerabilities. An unusually low vulnerability discovery rate could indicate ineffective tests or lack of test coverage. Regularly track the number of vulnerabilities detected in each testing cycle and compare it over time to determine trends.

Data testing

Data testing is a specialized type of testing that emphasizes the evaluation of data processed by systems, encompassing aspects like data transformations, data integrity rules, and data processing logic. Its purpose is to evaluate various attributes of data to identify data quality issues, such as duplication, missing data, or errors. By performing data testing, organizations can establish a foundation of reliable and trustworthy data for their systems which in turn enables informed decision-making, efficient business operations, and positive customer experiences.

Topics

- Indicators for data testing

- Anti-patterns for data testing
- Metrics for data testing

Indicators for data testing

Validate the integrity, accuracy, and consistency of data processes to help ensure that data operations, from input to storage and retrieval, maintain quality and reliability standards.

Indicators

- [QA.DT.1] Ensure data integrity and accuracy with data quality tests
- [QA.DT.2] Enhance understanding of data through data profiling
- [QA.DT.3] Validate data processing rules with data logic tests
- [QA.DT.4] Detect and mitigate data issues with anomaly detection
- [QA.DT.5] Utilize incremental metrics computation

[QA.DT.1] Ensure data integrity and accuracy with data quality tests

Category: RECOMMENDED

Data quality tests assess the accuracy, consistency, and overall quality of the data used within the application or system. These tests typically involve validating data against predefined rules and checking for duplicate or missing data to ensure the dataset remains reliable. While data quality testing might not fall under the traditional definitions of functional or non-functional testing, it's still an essential aspect of ensuring that an application or system functions correctly, as the quality of data can significantly impact the overall performance, user experience, and reliability of the software.

We recommend data quality tests because they enable rapid software delivery and continuous improvement of data driving systems. Using data quality tests, teams can spend more of their time focusing on how data should appear rather than continually checking it for accuracy, streamlining the development and deployment process. To calculate data quality metrics on your dataset, define and verify data quality constraints, and be informed about changes in the data distribution. Instead of implementing checks and verification algorithms on your own, you can focus on describing how your data should look.

Related information:

- Getting started with AWS Glue Data Quality from the AWSAWS Glue Data Catalog
- Deequ - Unit Tests for Data
- Test data quality at scale with Deequ
- How to Architect Data Quality on the AWS Cloud

[QA.DT.2] Enhance understanding of data through data profiling

Category: OPTIONAL

Use data profiling tools to examine, analyze, and understand the data including its content, structure, and relationships to identify issues such as inconsistencies, outliers, and missing values. By performing data profiling, teams can gain deeper insights into the characteristics and quality of their data, enabling them to make informed decisions about data management, data governance, and data integration strategies. This data is often used to enable or improve other types of data testing.

To integrate data profiling into a DevOps environment, consider automating the process using data profiling tools such as AWS Glue DataBrew, open-source tools, or custom scripts that analyze data regularly. Incorporate the profiling results into your data management, governance, and integration strategies, allowing your team to proactively address data quality issues and maintain consistent data standards throughout the development lifecycle.

Related information:

- Build an automatic data profiling and reporting solution with Amazon EMR, AWS Glue, and Quick
- Test data quality at scale with Deequ
- Deequ single column profiling
- AWS Glue DataBrew

[QA.DT.3] Validate data processing rules with data logic tests

Category: OPTIONAL

Data logic tests verify the accuracy and reliability of data processing and transformation within your application, ensuring that it functions as intended.

Establish test cases for data processing workflows and transformation functions, confirming that expected outcomes are achieved. Use version control systems to track changes in data logic and

collaborate effectively with team members. Implement automated data logic tests in development and staging environments, which can be triggered by code commits or scheduled intervals, to proactively identify and fix issues before they reach production environments.

Related information:

- Test data quality at scale with Deequ
- Deequ automatic suggestion of constraints

[QA.DT.4] Detect and mitigate data issues with anomaly detection

Category: OPTIONAL

Data anomaly detection is a specialized form of anomaly detection which focuses on identifying unusual patterns or behaviors in data quality metrics that may indicate data quality issues.

Consider integrating machine learning algorithms and statistical methods into your data quality monitoring processes. Use tools that can detect and address data anomalies in real-time and incorporate them into your development and deployment workflows. This enables automated assessment of the accuracy and reliability of data processing and analysis, enhancing the overall performance of your applications and systems.

Related information:

- What Is Anomaly Detection?
- Test data quality at scale with Deequ
- Deequ anomaly detection
- Amazon Lookout for Metrics
- Introducing Amazon Lookout for Metrics: An anomaly detection service to proactively monitor the health of your business
- Quick: ML-powered anomaly detection for outliers
- Amazon Kinesis: Detecting Data Anomalies on a Stream

[QA.DT.5] Utilize incremental metrics computation

Category: OPTIONAL

Incremental metrics computation allows teams to efficiently monitor and maintain data quality without needing to recompute metrics on the entire dataset every time data is updated. Use this method to significantly reduce computational resources and time spent on data quality testing, allowing for more agile and responsive data management practices.

Start by identifying the specific data quality metrics that are essential for your system. This could include metrics related to accuracy, completeness, timeliness, and consistency. Depending on your dataset's size and complexity, select a tool or framework that supports incremental computation. Some modern data processing tools, such as Apache Spark and Deequ, provide built-in support for incremental computations.

Segment your data into logical partitions, often based on time, such as daily or hourly partitions. As new data is added, it becomes a new partition. Automate the computation process by setting up triggers that initiate the metric computation whenever new data is added or an existing partition is updated.

Continuously monitor the updated metrics to help ensure they reflect the true state of your data. Periodically validate the results of the incremental metrics computation against a full computation to ensure accuracy. As you get more familiar with the process, look for ways to optimize the computation to save even more on computational resources. This could involve refining your partitions or improving the computation logic.

Related information:

- Deequ stateful metrics computation

Anti-patterns for data testing

- **Testing data drift:** Testing data in environments that do not mirror production datasets can result in testing outdated data schemas, different configurations, or testing data not representative of real-world conditions. Tests that pass in a non-representative environment might fail in production, leading to undetected data issues. Ensure that testing environments mirror production as closely as possible, both in terms of configuration and the nature of the data. Regularly update testing environment datasets to reflect changes in production.

Metrics for data testing

- **Data test coverage:** The percentage of your dataset or data processing logic covered by tests. Data test coverage gives an overview of potential untested or under-tested areas of the application. A high coverage helps ensure a comprehensive evaluation, while low coverage can indicate blind spots in testing. Improve this metric by prioritizing areas with lower coverage, using automation and tools to enhance coverage, and regularly review test strategies to help ensure they align with recent changes in the dataset or processing logic. Measure this metric by calculating the ratio of code or data elements covered by tests to the total lines of code or data elements in the application.
- **Test case run time**: The duration taken to run a test case or a suite of test cases. Increasing duration may highlight bottlenecks in the test process or performance issues emerging in the software under test. Improve this metric by optimizing test scripts and the order they run in, enhancing testing infrastructure, and running tests in parallel. Measure the timestamp difference between the start and end of test case execution.
- **Data quality score**: The combined quality of data in a system, encompassing facets such as consistency, completeness, correctness, accuracy, validity, and timeliness. Derive the data quality score by individually assessing each facet. Then aggregate and normalize them into a single metric, typically ranging from 0 to 100, with higher scores indicating better data quality. The specific method for aggregating the scores may vary depending on the organization's data quality framework and the relative importance assigned to each facet. Consider factors like the uniformity of data values (consistency), the presence or absence of missing values (completeness), the degree of data accuracy relative to real-world entities (correctness and accuracy), the adherence of the data to predefined rules (validity), and the currency and relevance of the data (timeliness).

Automated governance

The automated governance saga encapsulates the strategic implementation of policies, processes, and tools that allow organizations to manage and control their IT operations effectively and efficiently. By automating governance, organizations help ensure a standardized, consistent approach to risk management, compliance, and security. This practice reduces the need for manual intervention, improves scalability, and promotes best practices across all managed environments. Automated governance facilitates a balance between agility and control, providing assurance and accountability while enabling innovation and rapid deployment.

Capabilities

- Secure access and delegation
- Data lifecycle management
- Dynamic environment provisioning
- Automated compliance and guardrails
- Continuous auditing

Secure access and delegation

Establish scalable methods for managing fine-grained access controls, while still providing teams with the autonomy they need. This governance capability emphasizes the necessity for all access to be explicitly granted, guided by the principle of least privilege. Access should be temporary and only for the required duration, reducing the overall risk surface. Secure access and delegation also specifies procedures for emergency situations, and regular auditing of access controls to help ensure they align with evolving business requirements and threat landscapes.

Topics

- Indicators for secure access and delegation
- Anti-patterns for secure access and delegation
- Metrics for secure access and delegation

Indicators for secure access and delegation

Establish scalable, fine-grained access controls that balance security with team autonomy. Granting explicit, temporary access based on the principle of least privilege, providing procedures for emergencies, and regularly auditing access controls to align with evolving requirements and threats.

Indicators

- [AG.SAD.1] Centralize and federate access with temporary credential vending
- [AG.SAD.2] Delegate identity and access management responsibilities
- [AG.SAD.3] Treat pipelines as production resources
- [AG.SAD.4] Limit human access with just-in-time access
- [AG.SAD.5] Implement break-glass procedures

- [AG.SAD.6] Conduct periodic identity and access management reviews
- [AG.SAD.7] Implement rotation policies for secrets, keys, and certificates
- [AG.SAD.8] Adopt a zero trust security model, shifting towards an identity-centric security perimeter

[AG.SAD.1] Centralize and federate access with temporary credential vending

Category: FOUNDATIONAL

Implement a centralized subsystem for federated access and temporary credential vending to maintain secure and controlled access to your environments, workloads, and resources. By implementing a federated access solution, you can leverage your existing identity systems, provide single sign-on (SSO) capabilities, and avoid the need to maintain separate user identities across multiple systems which makes scaling in a DevOps model more tenable. Centralizing identity onboarding and permission management eliminate the inefficiencies of manual processes, reduce human error, and enable scalability as your organization grows.

Grant users and services fine-grained access to help ensure secure, granular control as they interact with resources and systems. By applying the least privilege principle, you can minimize the risk of unauthorized access and reduce the potential damage from compromised keys while retaining full control over access to resources and environments. To reduce the likelihood of keys being compromised, always vend short-lived, temporary credentials that are scoped for specific tasks to help ensure that privileges are granted only for the duration needed.

Related information:

- AWS Well-Architected Cost Optimization Pillar: COST02-BP04 Implement groups and roles
- Security best practices in IAM
- AWS Well-Architected Security Pillar: SEC02-BP04 Rely on a centralized identity provider
- IAM Identity Center
- What is SSO (Single-Sign-On)?
- Identity providers and federation

[AG.SAD.2] Delegate identity and access management responsibilities

Category: FOUNDATIONAL

Create a decentralized Identity and Access Management (IAM) responsibility model that enables individual teams to handle their own IAM tasks, such as creating roles and assigning permissions, as long as those teams operate within applied guardrails. This approach grants teams the autonomy to manage their roles and permissions essential for the applications they develop, encourages a culture of ownership and accountability, and enables your organization to scale its permission management effectively as it grows and embraces more DevOps practices.

Establish a set of well-defined guardrails which limit the maximum permissions a user or role can safely have. These guardrails reduce potential security risk while creating balance between allowing teams to manage their own IAM tasks and ensuring that they do not exceed the maximum permissions set.

Related information:

- Security best practices in IAM
- Use permissions boundaries to delegate permissions management within an account
- Establish permissions guardrails across multiple accounts
- Blog: Delegate permission management to developers by using IAM permissions boundaries

[AG.SAD.3] Treat pipelines as production resources

Category: FOUNDATIONAL

Pipelines become pivotal in every aspect of the software development lifecycle when practicing DevOps, as they become the sole method of moving code from development to production. During the process of building, testing, and deploying software, pipelines require access to all software components involved, including libraries, frameworks, repositories, modules, artifacts, and third-party dependencies. Due to this level of access and their role in deploying to potentially sensitive environments, pipelines should be recognized as integral components of your overall system and must be secured and managed to the same degree as the environments and data they interact with.

The application of least-privilege principles, commonly applied to human users, should be extended to pipelines. To reduce the potential for pipelines to become a security threat, their roles and permissions should be confined to align with their precise responsibilities. Emphasizing pipeline governance and treating pipelines as first-class citizens within your security infrastructure can substantially decrease your potential attack surface and reinforce the security of your overall DevOps environment.

Related information:

- AWS Well-Architected Security Pillar: SEC11-BP07 Regularly assess security properties of the pipelines

[AG.SAD.4] Limit human access with just-in-time access

Category: FOUNDATIONAL

As pipelines take on a more prominent role in the software development lifecycle in a DevOps model, the necessity for extensive human access to environments decreases. Human users should be granted minimal access necessary for their role, which is usually read-only access that does not allow any modifications or access to sensitive data. For experimentation which is typically hands-on and exploratory, teams should be granted access to sandbox environments which are isolated from system workloads.

In some cases, where things go wrong or a process cannot yet be automated, elevated permissions might be required. To accommodate these needs without compromising security, implement a just-in-time (JIT) access control strategy where permissions are temporarily escalated for a specific duration and purpose, upon explicit request and approval. This approach maintains the principle of least privilege, allowing necessary operational functions to be performed efficiently when needed, while also ensuring that the access is revoked once the task is complete.

By enforcing limited human permissions and using JIT access, you can improve your organization's security posture and reduce the risk of accidental or deliberate misuse of access rights. This restrictive and controlled model supports modern, secure DevOps practices where pipelines, treating everything as code, and automation should take precedence over manual actions.

Related information:

- Eliminate the need for human access
- AWS Samples: AWS IAM Temporary Elevated Access Broker
- Blog: Managing temporary elevated access to your AWS environment

[AG.SAD.5] Implement break-glass procedures

Category: FOUNDATIONAL

Emergencies or unforeseen circumstances might necessitate temporary access beyond regular permissions for day-to-day work. Having break-glass procedures helps ensure that your organization can respond effectively to crises without compromising long-term security. During emergency scenarios, like the failure of the organization's identity provider, security incidents, or unavailability of key personnel, these measures provide temporary, elevated access beyond regular permissions.

Implement measures that improve the resilience of your DevOps environments through the ability to respond effectively to emergencies without compromising long-term security. Create break-glass roles and users you can assume control of during emergencies that are able to bypass established controls, update guardrails, troubleshoot issues with automation tooling, or remediate security and operational issues that may occur. These break-glass roles and users should have adequate security measures, such as configuring them with hardware-based multi-factor authentication (MFA), to ensure that even in emergencies, access is tightly controlled and auditable. Establish alerts and alarms triggered by the use of these break-glass roles and users, and tie their usage closely to incident response and recovery procedures.

Related information:

- AWS Well-Architected Security Pillar: SEC03-BP03 Establish emergency access process
- Break glass access
- Amazon's approach to high-availability deployment: Dealing with the real world

[AG.SAD.6] Conduct periodic identity and access management reviews

Category: FOUNDATIONAL

With the distributed nature of DevOps Identity and Access Management (IAM) responsibilities, it is important to systematically review IAM roles and permissions periodically. This helps ensure that changes in roles and permissions align with the rapidly shifting needs of the organization, and that the guardrails set in place for delegation are working as intended or perhaps need to be fine-tuned. This activity aids in identifying unused or overly broad permissions, reinforcing the adherence to the principle of least privilege and reducing potential security risks.

Optionally, automate the right-sizing of permissions as part of these reviews. This proactive approach not only keeps IAM policies up-to-date, but also minimizes potential avenues for unauthorized access, further strengthening your overall security posture. Automatically right sizing

roles and permissions based on actual activity allows organizations to scalably enforce that the right resources are accessible to the right entities, at the right times.

Related information:

- AWS Well-Architected Security Pillar: SEC03-BP04 Reduce permissions continuously
- Regularly review and remove unused users, roles, permissions, policies, and credentials
- Use IAM Access Analyzer to generate least-privilege policies based on access activity
- Verify public and cross-account access to resources with IAM Access Analyzer
- Using AWS Identity and Access Management Access Analyzer
- Blog: IAM Access Analyzer makes it easier to implement least privilege permissions by generating IAM policies based on access activity
- Blog: Continuous permissions rightsizing to ensure least privileges in AWS using CloudKnox and AWS Config

[AG.SAD.7] Implement rotation policies for secrets, keys, and certificates

Category: RECOMMENDED

Regular rotation of secrets, keys, and certificates is a best practice in securing access, limiting the potential damage that can occur should these security resources become compromised. In a DevOps environment, pipelines often require access to sensitive environments and workloads, making them potential targets for attacks. The routine rotation of these resources that are used by pipelines can help to significantly mitigate this risk.

Establish a policy that clearly defines the lifecycle of these resources, including their creation, usage, rotation, and retirement intervals. Enforce these policies by automatically rotating secrets and keys to reduce the risk of oversights, delays, and human error.

Certificates play an important role in service-to-service authentication and providing encryption for both internal and external facing workloads and environments. When managing certificates, consider not only those issued within your organization but also those imported from external sources which may not be automatically renewable.

Monitoring systems that track the lifespan of these assets and alert administrators as they near expiration can contribute to this process. This approach can help prevent service disruptions caused by expired certificates and, in some cases, can trigger automated renewal procedures.

Related information:

- Blog: How to monitor expirations of imported certificates in AWS Certificate Manager (ACM)
- Rotate AWS Secrets Manager secrets - AWS Secrets Manager
- Managing access keys for IAM users - AWS IAM
- Rotating AWS KMS keys - AWS Key Management Service

[AG.SAD.8] Adopt a zero trust security model, shifting towards an identity-centric security perimeter

Category: RECOMMENDED

When operating under a zero trust security model, no user or system is trusted by default. It requires all users and systems, even those inside an organization's network, to be authenticated, authorized, and continuously validated to ensure secure configurations and posture. Only after validation will they be granted access to applications and data.

Zero trust is beneficial throughout the entire software development lifecycle. From the initial stages of code development as developers interact with source code repositories, through continuous integration using internal and external tools to build and test software, to the deployment and maintenance of the workloads, each user, pipeline, third-party, and service needs to be authenticated and authorized with every request. In these scenarios, zero trust enforces adherence to the principle of least privilege, ensuring that all of these independent users and systems are granted access to the right resources only when necessary.

Shifting to a zero trust model is not an all-or-nothing endeavor, it is a gradual process consistent with the DevOps principles of continuous improvement. Start small by pinpointing use cases that align with your organization's unique needs and the value and sensitivity of your systems and data. This understanding will guide the selection of zero trust principles, tools, and patterns that are most beneficial for your organization. Adopting zero trust often involves rethinking identity, authentication, and other context-specific factors like user behavior and device health. Enhance existing security practices over time, improving both identity-based and network-based security measures that complement each other to create a secure perimeter where identity-centric controls can operate.

AWS provides several use cases that illustrate zero trust principles:

- **Signing API requests:** Every AWS API request is authenticated and authorized individually, regardless of the trustworthiness of the underlying network.

- **Service-to-service interactions:** AWS services authenticate and authorize calls to each other using the same security mechanisms used by customers.
- **Zero trust for internet of things (IoT):** AWS IoT extends the zero trust model to IoT devices, enabling secure communication over open networks.

Related information:

- Zero Trust on AWS
- Zero Trust Maturity Model
- Amazon Verified Permissions
- AWS Verified Access

Anti-patterns for secure access and delegation

- **Broad permissions**: Granting extensive permissions without regular checks can lead to inadvertent access rights. This poses a significant security risk as potential vulnerabilities or unauthorized activities could occur. Review and adjust permissions periodically, adhering strictly to the principle of least privilege.
- **Manual identity and access management**: Depending on manual methods for both access control and identity management may lead to inconsistencies, delays, and errors. This manual approach is especially problematic as organizations grow, making it harder to scale and maintain security. Transition to using automated processes to manage identity and access management to help ensure timely updates, reduce errors, and enhanced scalability.
- **Static permission management**: Without a method to periodically review permissions as roles or business needs evolve can create both security vulnerabilities and operational inefficiencies. Schedule regular or continuous IAM reviews to perform automated audits to keep IAM configurations updated and aligned with present-day requirements.
- **Neglecting break-glass protocols**: Lacking established break-glass procedures could impair timely responses during emergencies that require elevated access. Incorporate just-in-time (JIT) access controls and regular drills to handle these incidents securely and efficiently.
- **Not evolving security with DevOps**: Adhering strictly to existing or outdated security models as the organization adopts DevOps best practices can introduce vulnerabilities and slow down progress. As organizations integrate new DevOps capabilities, their security models must adapt as well. Ensure that as DevOps practices evolve, the security model does too, prioritizing identity-centric strategies and continuous assessment of potential risks. By evolving security

practices alongside DevOps capabilities, organizations can protect against both internal and external threats.

Metrics for secure access and delegation

- **Incident frequency due to access violations**: The number of security incidents caused by leaked credentials, incorrect, or overly broad permissions over a given period. This metric highlights weaknesses in access controls and potential gaps in identity and access management practices. Count the number of security incidents attributed to access controls each month and compare it with past data to identify trends.
- **IAM review frequency**: The number of times IAM policies and permissions are reviewed over a given period. Regular or continuous reviews can help identify potential risks before they become security incidents. Improve this metric by scheduling periodic IAM reviews and implement automated tools that alert when permissions deviate from set policies. Count each IAM review that occurs over a given period, such as a quarter or year.
- **Time to revoke access**: The average duration taken to revoke access once it's determined to be no longer necessary. Improve this metric by implementing automated IAM solutions and regular permission audits. Calculate the average duration from the moment it is identified that access needs to be revoked to the time that access is revoked.
- **Rotation compliance**: The percentage of security-sensitive assets, such as credentials, secrets, keys, and certificates, rotated in compliance with established policies over a specified period. Regular rotation reduces the window of opportunity for malicious actors to misuse them, thus enhancing the overall security posture. Count the number of assets rotated in compliance with the policy in a given period, and divide it by the total number of assets due for rotation in the same period. Multiply the result by 100 to get the compliance percentage.

Data lifecycle management

Enforce stringent data controls, residency, privacy, sovereignty, and security throughout the entire data lifecycle. Scale your data collection, processing, classification, retention, disposal, and sharing processes to better align with regulatory compliance and safeguard your software from potential disruptions due to data mismanagement.

Topics

- Indicators for data lifecycle management

- Anti-patterns for data lifecycle management
- Metrics for data lifecycle management

Indicators for data lifecycle management

Enforce stringent controls on data through its entire lifecycle to ensure residency, privacy, sovereignty, and security. Scale data-related processes and align them with regulatory compliance, protecting software from disruptions due to data mismanagement.

Indicators

- [AG.DLM.1] Define recovery objectives to maintain business continuity
- [AG.DLM.2] Strengthen security with systematic encryption enforcement
- [AG.DLM.3] Automate data processes for reliable collection, transformation, and storage using pipelines
- [AG.DLM.4] Maintain data compliance with scalable classification strategies
- [AG.DLM.5] Reduce risks and costs with systematic data retention strategies
- [AG.DLM.6] Centralize shared data to enhance governance
- [AG.DLM.7] Ensure data safety with automated backup processes
- [AG.DLM.8] Improve traceability with data provenance tracking

[AG.DLM.1] Define recovery objectives to maintain business continuity

Category: FOUNDATIONAL

Clear recovery objectives help to ensure that teams can maintain business continuity and recover with minimal data loss, keeping the delivery pipeline flowing and maintaining service reliability.

Set recovery point objectives (RPO) indicating how much data loss is acceptable, and recovery time objectives (RTO) specifying how quickly services need to be restored following an incident. Develop and document your disaster recovery (DR) strategy, make it available to teams, and conduct exercises and trainings to maintain the ability to perform the strategy. Implement policies and automated governance capabilities that align with your RPO and RTO objectives.

Related information:

- AWS Well-Architected Reliability Pillar: REL09-BP01 Identify and back up all data that needs to be backed up, or reproduce the data from sources

- AWS Well-Architected Reliability Pillar: REL13-BP01 Define recovery objectives for downtime and data loss
- AWS Resilience Hub
- AWS Fault Isolation Boundaries
- Blog: Establishing RPO and RTO Targets for Cloud Applications

[AG.DLM.2] Strengthen security with systematic encryption enforcement

Category: FOUNDATIONAL

With continuous delivery, the risk of data breaches that can disrupt the software delivery process and negatively impact the business increases. To remain agile and rapidly able to deploy safely, it is necessary to enforce encryption at scale to protect sensitive data from unauthorized access when it is at rest and in transit.

Infrastructure should be defined as code and expected to change frequently. Resources being deploy need to be checked for a compliant encryption configuration as part of deployment process, while continuous scans for unencrypted data and resource misconfiguration should be automated in the environment. These practices not only aid in maintaining compliance, but also facilitates seamless and secure data management across various stages of the development lifecycle.

Automate the process of encryption key creation, distribution, and rotation to make the use of secure encryption methods simpler for teams to follow and enable them to focus on their core tasks without compromising security. Automated governance guardrails and auto-remediation capabilities should be used to enforce encryption requirements at scale, ensuring compliance both during and after deployment.

Related information:

- AWS Well-Architected Reliability Pillar: REL09-BP02 Secure and encrypt backups
- AWS Well-Architected Security Pillar: SEC08-BP02 Enforce encryption at rest
- AWS Well-Architected Security Pillar: SEC09-BP02 Enforce encryption in transit
- AWS Well-Architected Security Pillar: SEC09-BP01 Implement secure key and certificate management
- Encrypting Data-at-Rest and -in-Transit
- Amazon's approach to security during development: Encryption

[AG.DLM.3] Automate data processes for reliable collection, transformation, and storage using pipelines

Category: FOUNDATIONAL

A data pipeline is a series of steps to systematically collect, transform, and store data from various sources. Data pipelines can follow different sequences, such as extract, transform, and load (ETL), or extract and load unstructured data directly into a data lake without transformations.

Consistent data collection and transformation fuels informed decision-making, proactive responses, and feedback loops. Data pipelines play a key role in enhancing data quality by performing operations like sorting, reformatting, deduplication, verification, and validation, making data more useful for analysis.

Just as DevOps principles are applied to software delivery, the same can be done with data management through pipelines using a methodology commonly referred to as DataOps. DataOps incorporates DevOps principles into data management, including the automation of testing and deployment processes for data pipelines. This approach improves monitoring, accelerates issue troubleshooting, and fosters collaboration between development and data operations teams.

Related information:

- What Is A Data Pipeline?
- AWS DataOps Development Kit
- AWS Glue DataBrew
- AWS Glue ETL
- AWS Step Functions
- Data Matching Service – AWS Entity Resolution
- Blog: Build a DataOps platform to break silos between engineers and analysts
- DataOps
- Using Amazon RDS Blue/Green Deployments for database updates
- AWS Well-Architected Cost Optimization Pillar: COST11-BP01 Perform automations for operations

[AG.DLM.4] Maintain data compliance with scalable classification strategies

Category: FOUNDATIONAL

Automated data classification includes using tools and strategies to identify, tag, and categorize data based on sensitivity levels, type, and more. Data classification aids in enforcing data security, privacy, and compliance requirements. Misclassification or lack of data classification can lead to data breaches or non-compliance with data protection regulations. Scaling this practice through automation enables organizations to catalog, secure, and maintain the vast amounts of data they process.

Use tagging strategies to catalog data effectively and help maintain visibility of data across different services and stages of the software development lifecycle. Put guardrails in place to enforce compliance with data classification and handling requirements, such as those related to data privacy and residency. Continuously monitor data at different stages - collection, processing, classification, and sharing - to ensure the right handling strategies are in place and are being followed.

For advanced use cases, AI/ML tools can provide automatic recognition and classification of data, especially sensitive data. This approach can reduce the need for manual, human intervention.

Related information:

- AWS Well-Architected Sustainability Pillar: SUS04-BP01 Implement a data classification policy
- AWS Well-Architected Cost Optimization Pillar: COST03-BP02 Add organization information to cost and usage
- Data Classification
- Best Practices for Tagging AWS Resources
- Sensitive Data Discovery and Protection - Amazon Macie

[AG.DLM.5] Reduce risks and costs with systematic data retention strategies

Category: FOUNDATIONAL

Data is continuously generated, processed, and stored throughout the development lifecycle, increasing the complexity and importance of automated data management capabilities. Automated data retention and disposal is the process of implementing strategies and tools that systematically store data for pre-established periods and securely delete it afterward. The goal of data retention and disposal is not just about compliance, but also about reducing risks, sustainability, minimizing costs, and improving operational efficiency. Automation reduces the manual workload, decreases the risk of human error, and improves data governance and compliance.

To effectively implement automated data retention and disposal, start by defining the data lifecycle policies for your organization. This includes understanding the regulatory and business requirements for each type of data your organization processes, how long it needs to be retained, and the conditions under which it should be disposed. The policies should also include procedures for data archiving, backups, and restoration.

Once these policies are in place, automate the enforcement of these policies with data lifecycle management tools. These tools can automatically handle tasks like deletion, archival, or movement of data based on the predefined rules. As part of the automation process, develop mechanisms to log and audit data disposal actions. This not only provides accountability and traceability but also is essential for demonstrating compliance during audits.

Related information:

- AWS Well-Architected Cost Optimization Pillar: COST04-BP05 Enforce data retention policies
- AWS Well-Architected Sustainability Pillar: SUS04-BP03 Use policies to manage the lifecycle of your datasets
- AWS Well-Architected Sustainability Pillar: SUS04-BP05 Remove unneeded or redundant data
- Managing your storage lifecycle

[AG.DLM.6] Centralize shared data to enhance governance

Category: FOUNDATIONAL

Practicing DevOps puts an emphasis on teams working collaboratively and continuously exchanging data. Governing this shared data requires proper control, management, and distribution of data to prevent unauthorized access, data breaches, and other security incidents, fostering trust and enhancing the quality and reliability of software delivery.

Use centralized data lakes to provide a single source of truth of data and management within your organization, helping to reduce data silos and inconsistencies. It enables secure and efficient data sharing across teams, enhancing collaboration and overall productivity. Use Role-Based Access Control (RBAC) or Attribute-Based Access Control (ABAC) to limit access to data based on the user context. Implement automated metadata management to better understand the context, source, and lineage of the data, and deploy continuous, automated data quality checks to ensure the accuracy and usability of the data.

When collaboration extends beyond the organization's boundaries, *clean rooms* can be used to maintain data privacy and security. Clean rooms create isolated data processing environments

that let multiple parties collaborate and share data in a controlled, privacy-safe manner. With predefined rules that automatically govern the flow and accessibility of data, these clean rooms help ensure data privacy while still allowing for the extraction of valuable insights. This isolation facilitates decision-making and strategic planning, enabling stakeholders to collaborate and share information while protecting user privacy and maintaining compliance with various regulations.

Related information:

- AWS Well-Architected Sustainability Pillar: SUS04-BP06 Use shared file systems or storage to access common data
- Data Collaboration Service - AWS Clean Rooms
- AWS Lake Formation
- AWS Data Exchange

[AG.DLM.7] Ensure data safety with automated backup processes

Category: RECOMMENDED

Data loss can be catastrophic for any organization. Automated backup mechanisms help to ensure that your data is not only routinely backed up, but also that these backups are maintained and readily available when needed. As data is constantly being created and modified, these processes minimize the risk for data loss and reduce the manual, error-prone manual approach of backing up data.

Define a backup policy that outlines the types of data to be backed up, the frequency of backups, and the duration for which backups should be retained. This policy should also cover data restoration processes and timelines. Create backup policies that best fit the classification of the data to avoid backing up unnecessary data.

Choose backup tools that support automation and can be integrated into your DevOps pipelines and environments. These tools should have capabilities to schedule backups, maintain and prune older backups, and ensure the integrity of the backed-up data. For instance, during the development lifecycle, trigger backups before altering environments with business-critical data and in the case of rollbacks ensure that the data was not impacted.

Regularly test the data restoration process to ensure that the backed-up data can be effectively restored when required. Regular audits and reviews of the backup policy and the effectiveness of the backup process can help identify any gaps or potential improvements. Alerts and reports

should be configured to provide visibility into the backup process and notify teams about any issues.

Related information:

- AWS Well-Architected Sustainability Pillar: SUS04-BP08 Back up data only when difficult to recreate
- AWS Well-Architected Reliability Pillar: REL09-BP03 Perform data backup automatically
- Centrally manage and automate data protection - AWS Backup

[AG.DLM.8] Improve traceability with data provenance tracking

Category: RECOMMENDED

Data provenance tracking records the history of data throughout its lifecycle—its origins, how and when it was processed, and who was responsible for those processes. This practice forms a vital part of ensuring data integrity, reliability, and traceability, providing a clear record of the data's journey from its source to its final form.

The process involves capturing, logging, and storing metadata that provides valuable insights into the lineage of the data. Key aspects of metadata include the data's source, any transformations it underwent (such as aggregation, filtering, or enrichment), the flow of data across systems and services (movements), and actors (the systems or individuals interacting with the data).

Use automated tools and processes to manage data provenance by automatically capturing and logging metadata, and make it easily accessible and queryable for review and auditing purposes. For instance, data cataloging tools can manage data assets and their provenance information effectively, providing a systematic way to handle large volumes of data and their metadata across different stages of the development lifecycle.

In more complex use cases, machine learning (ML) algorithms can be used to uncover hidden patterns and dependencies among data entities and operations. This technique can reveal insights that might not be easily detectable with traditional methods.

Regularly review and update the data provenance tracking process to keep it aligned with evolving data practices, business requirements, and to maintain regulatory compliance. Provide training and resources to teams, helping them understand the importance and practical use of data provenance information.

Data provenance tracking is particularly recommended for datasets dealing with sensitive, regulated data or complex data processing workflows. It also adds significant value in environments where reproducibility and traceability of data operations are required, such as in data-driven decision-making, machine learning model development, and debugging data issues.

Data provenance tracking is particularly recommended for datasets dealing with sensitive or regulated data, machine learning workflows, and complex data processing which may require debugging.

Related information:

- AWS Glue Data Catalog
- Well-Architected Data Analytics Lens: Best practice 7.3 – Trace data lineage
- Amazon SageMaker AI ML Lineage Tracking
- Blog: Build data lineage for data lakes using AWS Glue, Amazon Neptune, and Spline

Anti-patterns for data lifecycle management

- **Lack of data protection measures:** Lax encryption, data access controls, backup policies, and poorly defined recovery objectives contribute to data vulnerability and can lead to regulatory non-compliance. Automated backup, encryption mechanisms and comprehensive disaster recovery plans are critical in maintaining data availability and minimizing downtime during recovery processes.
- **Inadequate data classification practices**: Accurate data classification plays a role in managing data access and ensuring the right stakeholders have access to the appropriate data. Manual or non-existent data classification could create vulnerabilities, possibly leading to misplacing data or granting unauthorized individuals access to sensitive data. An automated data classification approach, potentially leveraging AI/ML tools, can reduce human error and increase efficiency, ensuring data is consistently and correctly labeled according to its sensitivity.
- **Unrestricted data access:** Sharing data without proper governance can expose your organization to security risks like data breaches, loss of sensitive information, or violations of data sovereignty laws. You should manage and restrict access to shared data, provide a single source of truth through centralized data lakes, and use "clean rooms" for collaboration outside of the organization's boundaries.
- **Reliance on manual data retention and disposal:** Manual handling of data retention and disposal processes can lead to human error, missed deadlines, non-compliance, and inefficient

data management. Retaining data indefinitely is also not a good options, as it can lead to increased storage costs, potential non-compliance with data privacy laws, and an increased risk of data breaches. Automate data retention enforcement to help ensure compliance and efficient data management to reduce costs and improve operational efficiency.

Metrics for data lifecycle management

- **Recovery compliance rate**: The percentage of recovery operations that meet defined recovery time objectives (RTO) and recovery point objectives (RPO). Improve this metric by regularly testing and optimizing recovery procedures, train teams, and investing in reliable recovery tools. For each recovery operation, determine if both RTO and RPO were met. Calculate the ratio of compliant recoveries to total recovery attempts.
- **Backup failure rate**: The percentage of backup and attempted recovery operations that fail within a given period. This metric provides insight into the reliability of backup and recovery processes. A high failure rate indicates potential issues with the systems, policies, or tools in place and can jeopardize business continuity in the event of data loss or system failures. Calculate this metric by dividing the number of unsuccessful data backups and recovery operations by the total number of successful operations, multiply by 100 to get the percentage.
- **Data quality score**: The combined quality of data in a system, encompassing facets such as consistency, completeness, correctness, accuracy, validity, and timeliness. In the context of data lifecycle management, this score reflects the effectiveness of automated governance and effective data management practices. You may choose to track more granular metrics across multiple systems, such as adherence to data classification, retention, provenance accuracy, and encryption requirements. Derive the data quality score by individually assessing each facet. Then aggregate and normalize them into a single metric, typically ranging from 0 to 100, with higher scores indicating better data quality. The specific method for aggregating the scores may vary depending on the organization's data quality framework and the relative importance assigned to each facet. Consider factors like the uniformity of data values (consistency), the presence or absence of missing values (completeness), the degree of data accuracy relative to real-world entities (correctness and accuracy), the adherence of the data to predefined rules (validity), and the currency and relevance of the data (timeliness).

Dynamic environment provisioning

Establish strategies and practices to create, maintain, and manage multiple environments within an organization's landing zone, using automated processes. This approach helps ensure consistency

and compliance, enhances security, improves operational efficiency, optimizes resource usage, and allows organizations to adapt to changes faster.

Topics

- Indicators for dynamic environment provisioning
- Anti-patterns for dynamic environment provisioning
- Metrics for dynamic environment provisioning

Indicators for dynamic environment provisioning

Practices for creating, maintaining, and managing multiple environments within a landing zone using automated processes.

Indicators

- [AG.DEP.1] Establish a controlled, multi-environment landing zone
- [AG.DEP.2] Continuously baseline environments to manage drift
- [AG.DEP.3] Enable deployment to the landing zone
- [AG.DEP.4] Codify environment vending
- [AG.DEP.5] Standardize and manage shared resources across environments
- [AG.DEP.6] Test landing zone changes in a mirrored non-production landing zone
- [AG.DEP.7] Utilize metadata for scalable environment management
- [AG.DEP.8] Implement a unified developer portal for self-service environment management

[AG.DEP.1] Establish a controlled, multi-environment landing zone

Category: FOUNDATIONAL

Establish a multi-environment landing zone as a controlled foundation which encompasses all of the environments that workloads run in. A landing zone acts as a centralized base from which you can deploy workloads and applications across multiple environments. In AWS, it is common to run each environment in a separate AWS account, leading to hundreds or thousands of accounts being provisioned. Landing zones allow you to scale and securely manage those accounts, services, and resources within.

Operate the landing zone using platform teams and the *X as a Service* (XaaS) interaction mode, as detailed in the Team Topologies book by Matthew Skelton and Manuel Pais. This enables teams to

request or create resources within the landing zone using infrastructure as code (IaC), API calls, and other developer tooling.

The landing zone has the benefit of maintaining consistency across multiple environments through centrally-applied policies and service-level configurations. This approach allows the governing platform teams to provision and manage resources, apply common overarching policies, monitor and helps ensure compliance with governance and compliance standards, manage permissions, and implement guardrails to enforce access control guidelines, across all of the environments with minimal overhead.

It's a best practice within the landing zone to separate environments, such as non-production and production, to allow for safer testing and deployments of systems. The landing zone often includes processes for managing network connectivity and security, application security, service onboarding, financial management, change management capabilities, and developer experience and tools.

For most organizations, a single landing zone that includes all environments for all workloads should suffice. Only under special circumstances, such as acquisitions, divestments, management of exceptionally large environments, specific billing requirements, or varying classification levels for government applications, might an organization need to manage multiple landing zones.

Manage the landing zone and all changes to it as code. This approach simplifies management, makes auditing easier, and facilitates rollback of changes when necessary.

Related information:

- AWS Well-Architected Cost Optimization Pillar: COST02-BP03 Implement an account structure
- Cloud Security Governance - AWS Control Tower
- Landing zone - AWS Prescriptive Guidance
- Benefits of using multiple AWS accounts
- AWS Security Reference Architecture (AWS SRA)
- AWS Control Tower and AWS Organizations
- Establishing Your Cloud Foundation on AWS
- Provision and manage accounts with Account Factory
- AWS Account Factory Email: Many AWS Accounts, one email address

[AG.DEP.2] Continuously baseline environments to manage drift

Category: FOUNDATIONAL

Baselining environments is a structured process for routinely updating and standardizing individual environments within the landing zone to match a specified configured state or *baseline*. Drift management, a part of this process, involves the identification and resolution of differences between the environment's current configuration and its desired baseline state.

Regular baselining helps to ensure consistency across environments at scale, minimizing errors and enhancing operational efficiency and governance capabilities. The centralized platform team that manages the landing zone and environments within require the ability to consistently add new features, security configuration, performance improvements, or resolving detected drift issues.

The team must be able to baseline all targeted environments every time a change is made to the overall landing zone desired state definition or when a misconfiguration is detected within the environment.

It is the shared responsibility of the platform team and teams operating workloads to verify that the correct policies, alerts, and resources are configured properly and securely. As these teams are both making changes to the same environment, it is important that all controls and resources managed by the platform team are secured against unauthorized modifications by other teams operating within the environment. Changes being made by the platform team to the environment should be communicated to the other teams to promote a culture of transparency and collaboration.

All deployment, updates, or new features made to the environments should be made through an infrastructure as code (IaC) approach, which allows for version control, testing, and reproducibility of environments. It is also recommended to have a separate staging environment to test these changes before they are deployed to the production environments, further reducing the risk of disruptions or errors.

Related information:

- Customize your AWS Control Tower landing zone
- Types of Landing Zone Governance Drift
- Customize accounts with Account Factory Customization (AFC)
- Overview of AWS Control Tower Account Factory for Terraform (AFT)
- Implementing automatic drift detection in CDK Pipelines using Amazon EventBridge

[AG.DEP.3] Enable deployment to the landing zone

Category: FOUNDATIONAL

Dedicate an environment for each system to host the resources and tools required to perform controlled and uniform application deployments to related non-production and production environments. These deployment environments can include infrastructure or services such as pipelines and build agents.

At a minimum, each system should have a set of deployment, test, and production environments to support the development lifecycle. Having these environments at the system level, as opposed to sharing environments across multiple systems or at the team level, provides multiple benefits:

- **Isolation of systems:** Each system's resources are isolated, reducing the risk of cross-system interference, reaching quotas, and security breaches.
- **Tailored environments:** The environments can be customized according to the specific needs of each system, improving efficiency and reducing unnecessary resource usage.
- **Separation of concerns:** Each environment handles a specific aspect of the application lifecycle (deployment, testing, production), ensuring a clean and organized workflow.

The deployment environment should include resources and tools to support building, validation, promotion, and deployment of the system. A deployment environment may not be necessary for all organizations and scenarios, such as if your development lifecycle tools are hosted on-premises or outside of your landing zone. For these use cases, you will need to verify network connectivity between your external tools and your landing zone environments.

Related information:

- Spaces in CodeCatalyst
- Deployments OU

[AG.DEP.4] Codify environment vending

Category: RECOMMENDED

A core benefit of the DevOps model is team autonomy and reducing cross-team dependencies. Through infrastructure as code (IaC), teams can establish and manage their environments autonomously in a self-service manner, shifting from traditional methods where operations teams would oversee these responsibilities.

By provisioning environments, and the accounts operating them, as IaC or API calls, teams are empowered with the flexibility to create environments according to their specific requirements and ways of working. Codifying the environment provisioning process provides teams with the flexibility to create both persistent and ephemeral environments based on their specific needs and workflows. In particular, this code-based approach enables the easy creation of ephemeral environments that can be automatically setup and torn down when not in use, optimizing resource utilization and cost.

Use shared libraries or services that allow teams to request and manage environments using IaC. These libraries should encapsulate best practices for environment configuration and should be designed to be used directly in deployment pipelines, enabling individual teams to manage their environments autonomously. This reduces the need for manual requests or interactions with a developer portal, as well as reduces the reliance on platform teams for provisioning and managing environments on their behalf. This approach promotes consistency and reduces overhead from cross-team collaboration.

Related information:

- What is the AWS CDK?
- Create an AWS Proton environment
- Provision and manage accounts with Account Factory
- Provision Accounts Through Service Catalog

[AG.DEP.5] Standardize and manage shared resources across environments

Category: RECOMMENDED

Cross-environment resource sharing is the practice of deploying, managing, and providing access to common resources across various environments from a centrally managed account. This approach enables teams to efficiently use and manage shared resources, such as networking or security services, without the need to replicate their setup in each environment. By unifying the management of these foundational resources, individual teams can focus more on the functionality of their workloads, rather than spending time and effort managing common infrastructure components.

Platform teams should deploy and manage shared resources into accounts they manage, then provide APIs or libraries that individual teams can use to consume the shared resources as needed. This approach reduces redundancy and promotes standardization across the organization, allowing

development teams to concentrate on their unique workloads rather than complex infrastructure management.

Related information:

- Infrastructure OU and accounts
- Sourcing and distribution
- Sharing your AWS resources - AWS Resource Access Manager

[AG.DEP.6] Test landing zone changes in a mirrored non-production landing zone

Category: RECOMMENDED

Changes to landing zones can have significant impacts across teams and processes because it is consumed by many teams in an organization. To minimize the risk of potential failures when making changes to the landing zone, platform teams should follow similar practices seen in the development lifecycle, including thorough testing and validation in a dedicated environment before rolling out to production.

When making changes to a landing zone, establish mirrored landing zones for testing changes before deploying to the production landing zone. This allows for changes to be validated without affecting the production environment. Use deployment pipelines to promote, validate, and deploy changes between the mirrored and production landing zones, performing extensive testing and validation at each stage.

Overall, this practice promotes safer changes to the production landing zone which has the potential to impact many teams in the organization. Clearly communicate with those teams before rolling out changes to the production landing zone so that they are informed of imminent changes, potential impacts to their environments and systems, and the projected timeline.

Related information:

- Multiple organizations: Test changes to your overall AWS environment

[AG.DEP.7] Utilize metadata for scalable environment management

Category: OPTIONAL

Effective environment management at scale requires the collection and maintenance of key information about each environment, such as ownership, purpose, criticality, lifespan, and more.

These details can offer visibility and clarity which reduces potential confusion and misuse of environments and assists with setting up proper controls based on specific details associated with the environment.

Adopt techniques like resource tagging to track and maintain this metadata. Not only does this allow platform teams to track and optimize costs by accurately attributing resource usage to specific environments, but it also supports the management of access controls and security measures, aligning governance and compliance needs with individual environments.

For implementation, use available tagging features and APIs for resource management and metadata tracking. Where additional metadata capture is required, consider creating or integrating with a custom tracking system tailored to your specific needs, such as existing configuration management database (CMDB) or IT service management (ITSM) tools, providing a holistic view of all environments, thus empowering platform teams to better govern and manage environments based on their metadata.

Although this practice is marked as optional, it is strongly recommended for organizations operating in complex and large-scale environments, where managing resources and configurations based on metadata can significantly improve efficiency, governance, and compliance. This indicator focuses on leveraging metadata for active environment management, distinguishing it from the broader scope of configuration item management.

Related information:

- Choosing tags for your environment
- Tag policies - AWS Organizations

[AG.DEP.8] Implement a unified developer portal for self-service environment management

Category: OPTIONAL

Consider implementing a self-service portal that empowers developers to create, manage, and decommission their own isolated development or sandbox environments, within the established boundaries set by the platform team. While fostering autonomy for development teams, this approach accelerates the development process and reduces the operational load on the supporting platform team. To ensure adherence to the organization's standards and ensure consistency, the portal could include predefined environment templates and resource bundles.

While beneficial, the implementation of a developer portal is optional, particularly if the organization is leveraging codified environment vending as recommended. Infrastructure as code (IaC) presents an alternative approach that reduces human intervention.

The self-service portal, if implemented, can adopt the *X as a Service* (XaaS) interaction model as outlined in the Team Topologies book by Matthew Skelton and Manuel Pais. The portal can evolve over time into a central resource for common, reusable tools and capabilities preconfigured to comply with organizational standards, facilitating streamlined automated governance activities. This might include centralized access to common tools into a unified developer portal, including observability, security, quality, cost, and organizational use cases. If adopted by many teams, this platform can become an excellent method for communicating changes within the organization.

Related information:

- The Amazon Software Development Process: Self-Service Tools

Anti-patterns for dynamic environment provisioning

- **Manual environment management**: Relying on manual provisioning and management of environments, or using uncoordinated scripts, can introduce inconsistencies and inefficiencies. Manual approaches are prone to errors and can slow down development. It's vital to transition towards an automated, code-based approach for environment management. This ensures enhanced repeatability and reliability, and also allows teams to maintain a consistent pace of development, ultimately reducing operational overheads.
- **Inflexible environment provisioning**: Provisioning *one-size-fits-all* environments without considering the unique needs of different workloads or teams can restrict the ability of workloads to operate optimally. Not taking into account the diverse requirements of different teams or workloads can result in inefficiencies, operational burdens, and slowed innovation. Instead, a dynamic provisioning approach tailored to the specific needs of each workload, combined with equipping developers with self-service capabilities, can greatly improve resource utilization and cost efficiency.
- **Bypassing non-production testing for environment changes**: Implementing changes directly in the production landing zone or to environment baselines without prior testing in a mirrored non-production environment exposes the organization to unnecessary risks. Always test these large-scale changes in a safe, non-production environment to identify potential issues and mitigate them before they impact the production environment.

- **Allowing configuration drift**: Deviations in environment configurations can creep in due to manual changes or lack of consistent monitoring. This drift can compromise security, increase maintenance complexity, and lead to unpredictable outcomes. To counteract this, adopt tools and practices that continuously baseline environments to maintain the desired state and enforce uniform configurations.
- **Fragmented self-service tools**: Providing multiple, disjointed self-service tools for developers to manage environments can create confusion, result in inefficiencies, and make it harder to enforce standards and best practices. Integrating these functionalities into a unified developer portal helps ensure consistent practices, better governance, and smoother operations. This unified portal could include self-service capabilities from development lifecycle, quality assurance, and observability best practices, streamlining the developer experience.

Metrics for dynamic environment provisioning

- **Environment provisioning lead time**: The average time it takes to provision an environment. A reduced lead time indicates an ability to meet changing requirements, an enhanced developer experience, and increased readiness for disaster recovery scenarios. Assess this metric by tracking the duration from the moment a provisioning request is initiated to when the environment becomes operational.
- **Configuration drift rate**: The percentage of environments deviating from their baseline configuration within a specific time frame. Improve this metric by implementing observability capabilities, applying infrastructure-as-code practices, and regularly review and update environment baselines. Compare current environment configurations to baseline configurations regularly. Calculate the ratio of drifted environments to total environments, then multiply by 100 for the percentage.
- **Self-service tool adoption rate**: The percentage of developers using self-service tools for environment management. This metric can indicate the ease-of-use and effectiveness of provided self-service capabilities. Improve this metric by optimizing user interfaces, developing APIs and CLIs, conducting training sessions, and gathering feedback to make necessary enhancements. Monitor usage over a specified period to determine the number of users actively using the tool. Calculate the ratio of users adopting the tool to the total in the expected user base, then multiply by 100 for the percentage.
- **Environment overhead cost**: Measuring overhead costs resulting from underutilized or over-provisioned environments. This metric provides insight into potential cost savings and optimal resource allocation. Improve this metric by implementing automated right-sizing capabilities, monitoring environment utilization, and de-provisioning unused environments. Track costs

associated with maintaining environments and compare against their actual utilization or workload.

Automated compliance and guardrails

Integrate risk management, business governance adherence, and application and infrastructure governance mechanisms required to maintaining compliance within dynamic, constantly changing environments. This capability enables automatic enforcement of directive, detective, preventive, and responsive measures, using automated processes and policies. It helps organizations consistently uphold standards and regulations while minimizing the manual overhead traditionally associated with compliance management.

Topics

- Indicators for automated compliance and guardrails
- Anti-patterns for automated compliance and guardrails
- Metrics for automated compliance and guardrails

Indicators for automated compliance and guardrails

Integrate risk management and governance mechanisms into the maintenance of compliance in dynamic environments. It enables automatic enforcement of directive, detective, preventive, and responsive measures, reducing the manual overhead associated with compliance management.

Indicators

- [AG.ACG.1] Adopt a risk-based compliance framework
- [AG.ACG.2] Implement controlled procedures for introducing new services and features
- [AG.ACG.3] Automate deployment of detective controls
- [AG.ACG.4] Strengthen security posture with ubiquitous preventative guardrails
- [AG.ACG.5] Automate compliance for data regulations and policies
- [AG.ACG.6] Implement auto-remediation for non-compliant findings
- [AG.ACG.7] Use automated tools for scalable cost management
- [AG.ACG.8] Conduct regular scans to identify and remove unused resources
- [AG.ACG.9] Integrate software provenance tracking throughout the development lifecycle

- [AG.ACG.10] Automate resolution of findings in tracking systems
- [AG.ACG.11] Digital attestation verification for zero trust deployments

[AG.ACG.1] Adopt a risk-based compliance framework

Category: FOUNDATIONAL

Managing compliance in a DevOps model can initially feel even more challenging than traditional models due to the fast-paced, iterative, and distributed ways of workings. Risk-based compliance framework such as NIST Cybersecurity Framework, ISO 27001, or CIS Controls help to align your DevOps processes and tools with industry best practices and compliance requirements. These frameworks offer a structured methodology for managing cybersecurity risk in compliance with the organization's business needs.

Select a relevant framework that fits your business and security needs and assess your current practices against this framework, identifying any gaps in compliance. Work towards addressing these gaps and continually monitor and reassess your practices to help ensure ongoing compliance. Leverage this well-architected guidance to improve your DevOps capabilities to more efficiently meet these compliance requirements. Use cloud-native services and tools to track compliance against your chosen framework.

Related information:

- Security Hub CSPM standards reference
- Conformance Packs - AWS Config
- Automate Cloud Audits - AWS Audit Manager
- AWS Well-Architected Tool

[AG.ACG.2] Implement controlled procedures for introducing new services and features

Category: FOUNDATIONAL

To maintain the balance between encouraging innovation and upholding compliance and governance requirements, platform teams need a scalable, controlled procedure for introducing new cloud vendor or third-party services to be used.

DevOps culture encourages continuous learning and exploration of new technologies, tools, and services. Provide teams with the ability to explore and experiment with new features and services

while maintaining organizational security and compliance standards. Structure these exploration opportunities in a controlled, secure manner, to promote agility without compromising integrity.

Establish well-defined guardrails that uphold security and compliance when introducing new features and services. This includes access restrictions, acceptable use cases, and alignment with security policies. Create sandbox environments where teams can safely explore and test these features without compromising production environments or violating governance policies. Develop a systematic, scalable onboarding process which allows platform teams to enable guardrails and policies for governing usage of the service, which leads to enabling the feature or service in other environments, including production.

Follow the principle of least privilege by granting teams access to use only specific actions or API calls for approved services. As services update and add new features, this will help ensure that the platform team reserves the ability to perform onboarding procedures with these new features as well.

Related information:

- Example service control policies

[AG.ACG.3] Automate deployment of detective controls

Category: FOUNDATIONAL

Perform rapid and consistent detection of potential security issues or misconfigurations by deploying automated, centralized detective controls. Automated detective controls are guardrails which continuously monitor the environment, quickly identifying potential risks, and potentially mitigating them.

Use a *compliance as code* approach to integrate compliance rules into deployment pipelines. Additionally, implement detective rules in the environment for real-time checks. Leveraging artificial intelligence (AI) and machine learning (ML) can further enhance the capability to monitor and detect non-compliant configurations or complex security threats.

Related information:

- Cloud Security Posture Management (CSPM) - AWS Security Hub CSPM
- AWS Config and AWS Organizations - AWS Organizations
- Intelligent Threat Detection - Amazon GuardDuty

- Building Prowler into a QuickSight powered AWS Security Dashboard

[AG.ACG.4] Strengthen security posture with ubiquitous preventative guardrails

Category: FOUNDATIONAL

Perform rapid and consistent detection of potential security issues or misconfigurations by deploying automated, centralized detective controls. Automated detective controls are guardrails that continuously monitor the environment, quickly identifying potential risks, and potentially mitigating them.

Guardrails can be placed at various stages of the development lifecycle, including being directly enforceable within the environment itself—providing the most control and security assurance. To provide a balance between agility and governance, use multiple layers of guardrails. Use environmental guardrails, such as access control limitations or API conditions, which enforce security measures and compliance ubiquitously across an environment. Embed similar detective and preventative checks within the deployment pipeline, which will provide faster feedback to development teams.

The actual implementation of environmental guardrails can vary based on the specific tools and technologies used within the environment. An example of preventative guardrails in AWS are Service Control Policies (SCPs) and IAM conditions.

Related information:

- Example service control policies

[AG.ACG.5] Automate compliance for data regulations and policies

Category: RECOMMENDED

The rapid pace of development and decentralized nature of operating under in a DevOps environment can pose challenges for maintaining data privacy compliance. Automation and guardrails can greatly ease this process by integrating compliance checks and remediation actions throughout the development lifecycle. This extends to automated enforcement of data access and handling protocols, continuous monitoring of resource configurations for data sovereignty and residency requirements, and automated auditing and risk assessment.

Implement automated tools that can enforce data access and handling policies. Set up continuous monitoring systems to assess compliance with data sovereignty and residency requirements.

These tools should also be capable of automated auditing, risk assessment, and triggering incident response mechanisms when anomalies or threats are detected. By doing so, your organization can adapt swiftly to changing data privacy laws and regulations, bolster your data security governance, and reduce the risk of data breaches or non-compliance.

Automating this process is categorized as recommended because not all organization practicing DevOps handle applicable personal data.

Related information:

- Data Protection & Privacy at AWS
- Amazon Information Request Report
- AWS Security Blog: Data Privacy

[AG.ACG.6] Implement auto-remediation for non-compliant findings

Category: RECOMMENDED

Manual identification and remediation of non-compliance issues can be time-consuming and prone to errors. Automated systems can rapidly respond to non-compliant resources, misconfigurations, and insecure defaults as soon as they are detected.

In the event of a non-compliance issue, an auto-remediation process should be triggered, which not only resolves the immediate issue but also initiates an alert to the developers. This is important because, while the auto-remediation resolves the problem at the system level, the developers need to be made aware of the problem so that they can correct the source of the error and prevent its recurrence. This dual approach of auto-remediation and developer notification promotes a learning environment and reduces the likelihood of recurring non-compliance issues. It allows developers to address the root cause of the configuration drift or non-compliance to prevent the continual reintroduction of the same error.

While recommended for its efficiency and rapid response, auto-remediation is not universally applicable to all compliance issues. Certain issues might require manual intervention or a more nuanced approach. Use preventative guardrails and implementing detective and preventative controls directly within the development lifecycle where possible, with auto-remediation being a third best option. These measures, when used together, yield a more compliant environment.

The goal of auto-remediation should not just be the swift resolution of issues, but also the continued education of developers while reducing the overall incidence of non-compliance.

Related information:

- AWS Well-Architected Performance Pillar: PERF07-BP06 Monitor and alarm proactively
- AWS Well-Architected Reliability Pillar: REL06-BP04 Automate responses (Real-time processing and alarming)
- Remediating Noncompliant Resources with AWS Config Rules
- AWS Systems Manager Automation
- Automated Security Response on AWS
- Automating ongoing OS patching - AWS Prescriptive Guidance
- Decommission resources - Cost Optimization Pillar

[AG.ACG.7] Use automated tools for scalable cost management

Category: RECOMMENDED

Automated cost management tools enable teams to remain agile and innovative while maintaining budgetary control. As deployment frequency increases due to DevOps improvements, it becomes important to put in place guardrails to control costs.

Use automated cost tracking mechanisms, such as cost budgets and alerts, and tag resources for cost allocation. Use cloud native cost management tools to monitor and report cloud expenditure continuously. Ensure these tools can alert teams when costs are approaching or exceeding budgeted amounts, and where possible, consider implementing auto-remediation methods to optimize resource usage, apply savings plans or reserved instances, and decommission unused resources.

Related information:

- AWS Well-Architected Cost Optimization Pillar: COST02-BP05 Implement cost controls
- Cloud Financial Management
- AWS Billing and Cost Management Conductor
- AWS Cost Anomaly Detection

[AG.ACG.8] Conduct regular scans to identify and remove unused resources

Category: RECOMMENDED

Over time, unused resources can often be a byproduct of experimentation and more frequent deployments, including dormant servers, unused deployment resources, idle containers, redundant environments, and unused serverless functions. These resources can pile up to create a less than ideal operating environment if not managed effectively, leading to inefficiencies, inflated costs, system unreliability, and heightened security risks.

Perform automated scans scoped to all deployed resources in your environment and pinpoint unused or outdated resources. This can be accomplished by using health check endpoints, reviewing logs, using metadata elements such as tags, or checking billing dashboards for utilization.

Verify the status and compatibility of software running on these resources, especially if they have been disconnected or powered off for extended periods of time. These checks are especially useful for preventing *zombie servers*, which have the potential to be rebooted after long periods of disconnection and might be running outdated or incompatible software.

Based on the verification results and the organization's policies, take action to remediate these resources, such as updating the software, decommissioning the resources, or integrating them back into the environment. Frequently performing these scans can prevent potential service disruptions, maintain up-to-date software across all resources, and ensure the overall integrity of the DevOps environment.

Related information:

- AWS Well-Architected Cost Optimization Pillar: COST02-BP06 Track project lifecycle
- Implementing health checks
- Decommission resources - Cost Optimization Pillar
- Identifying your unused resources - DynamoDB

[AG.ACG.9] Integrate software provenance tracking throughout the development lifecycle

Category: RECOMMENDED

Software provenance tracking inspects the origin and evolution of software components throughout their lifecycle to understand where a piece of software originated, its development and update history, and its distribution. Provenance tracking ensures the integrity of software, maintains compliance, and enhances the security of the software supply chain throughout the development lifecycle. Effective provenance tracking can prevent the introduction of insecure

components, offer early detection of potential vulnerabilities, and provide insights for timely remediation.

Developers are encouraged to use the best tools for the task at hand, often including third-party software components. These third-party elements can introduce an additional layer of complexity and potential risk. Implementing software provenance tracking mitigates these risks by promoting better visibility into the lifecycle of software components, thereby increasing accountability, transparency, and trust.

Provenance tracking should be integrated into all stages of the development lifecycle. For instance, source code provenance should be tracked at the time of code check-in or commit into Version Control Systems like Git, while the provenance of third-party components should be verified at the time of component acquisition and usage using tools like Software Composition Analysis (SCA). A Software Bill of Materials (SBOM) can be used as a detailed list of all components within your software, including the exact version, digital signatures, and origin of each one.

Verify provenance at build and deploy time. Use digital signatures and hashing algorithms to verify the integrity and provenance of software artifacts as part of the deployment pipeline, validating the signature of an artifact against a trusted source before it is used. It can also be useful to check running software continuously to identify compromised or outdated software components post-deployment.

Related information:

- SLSA specification

[AG.ACG.10] Automate resolution of findings in tracking systems

Category: RECOMMENDED

Automating the resolution of findings in tracking systems can accelerate the security incident response process, prevent untracked mitigation activities, and ensure accuracy in reporting processes. It also allows teams to focus more on development, resolving issues, and innovation, while automation handles the routine tracking and resolution tasks.

Use tools that support automated tracking and resolution capabilities. When an issue is detected, a ticket should be created automatically in the tracking system. Once the issue is resolved, the system should be able to automatically validate the resolution and close the corresponding ticket. This approach reduces the chances of human error, ensures a faster response to issues, and is

capable of providing comprehensive reporting and analytics capabilities to support continuous improvement of the security posture.

Related information:

- Automation rules - AWS Security Hub CSPM

[AG.ACG.11] Digital attestation verification for zero trust deployments

Category: RECOMMENDED

Digital attestations are recommended to be created for each action that occurs during the development lifecycle. Attestations serve as evidence of compliance, which can be verified either during or post-deployment. Authorizing deployments by verifying attestations extends a zero trust security model to the development lifecycle. If attestations for the required quality assurance tests, pipeline stages, or manual approvals are missing or invalid, meaning that compliance and change management requirements were not met during the development lifecycle, the deployment can be either prevented or subjected to an exception mechanism for risk acceptance.

Incorporate the creation of digital attestations into the development lifecycle. Before deployment, verify that the required attestations have been digitally signed by trusted cryptographic keys and that they meet the change management and compliance policies. If a deployment is found to be non-compliant, you can choose to respond in several ways depending on your security and governance requirements. It can be used as a detective control which allows the deployment to proceed while keeping an audit log of the non-compliance for future investigation. It can also be used as a preventive control, stopping the deployment from proceeding entirely. Pairing this with an exception mechanism you could enforce directive controls to accept the identified risks for a period of time.

This approach to automated governance and change management continuously assesses the integrity of the software throughout the development lifecycle. It provides a method of authorizing deployment based on adherence to governance and compliance requirements, extending zero trust security model principles to the deployment process.

Related information:

- Software attestations
- in-toto Attestation Framework Spec
- Zero Trust on AWS

- Zero Trust Maturity Model

Anti-patterns for automated compliance and guardrails

- **Manual policy enforcement**: Relying on manual checks and balances to enforce policies and standards. It's difficult to maintain consistent governance and mitigate risks with manual methods, especially when dealing with high-velocity, constantly changing environments and systems. Use automated tools that enforce, monitor, and audit compliance standards consistently across environments.
- **Static compliance checks**: Only validating compliance during specific phases of the development lifecycle, such as at the end of development, instead of continuously throughout the lifecycle. This can lead to late-stage discoveries of non-compliance, which are costlier and more time-consuming to address. Implement continuous compliance checks throughout the development, including both during and after deployment.
- **Relying on manual remediation**: Manual remediation can lead to delays in identifying and resolving issues, extending vulnerability windows. It can also be an inefficient use of resources, leading to higher costs and increased risk of human error. Build auto-remediation processes that not only detect but also resolve non-compliant findings in real-time.
- **Over-reliance on preventative guardrails**: Solely relying on preventive measures and not considering detective or responsive controls. It's impossible to predict and prevent every potential non-compliance issue making it important to have a balanced mix of detective, preventive, and responsive controls in place.
- **Manual change validation**: With traditional change management, a Change Advisory Board (CAB) meeting would precede a release approval. The CAB verifies that proper actions have been taken to remediate change risk. This includes ensuring that a group of subject matter experts reviewed the change and that organizational requirements for quality assurance and governance are being followed, such as ensuring expected tests were run and that deployments occur within approved change windows. Traditional CAB approval could take from days to weeks to schedule and debate the changes. Use automated governance capabilities to automate these checks as part of the development lifecycle and continuously within your environment.

Metrics for automated compliance and guardrails

- **Billing variance**: The difference between forecasted and actual billing for cloud resources or other IT costs. This metric indicates potential inefficiencies or areas of cost-saving, as well as

highlighting the accuracy of financial forecasting. Calculate by subtracting the actual billing amount by the forecasted billing amount, then divide by the forecasted billing amount and multiply by 100 to get the variance percentage.

- **Change failure rate**: The percentage of changes that fail. A change is considered a failure if it leads to degraded service or if it requires remediation, such as a hotfix or rollback. This metric provides insights into the quality and reliability of changes being made to a system. With effective automated governance in place, the expectation is that the change failure rate would decrease, as automated checks and balances catch potential issues before they're deployed into production. Calculate by dividing the number of failed changes by the total number of changes made within a given period and then multiply by 100 to derive the percentage.
- **Guardrail effectiveness score**: The ratio of successful preventions or detections by a specific guardrail to the number of false positives or negatives it produces. By assessing the efficiency and precision of individual guardrails, you can determine which rules are the most effective and which might need refinement or deprecation. Improve this metric by regularly reviewing and adjusting guardrail configurations, parameters, or logic to decrease false positives and negatives. Calculate this metric for each guardrail by dividing the number of successful detections or preventions by the total number of detections or preventions. Multiply this by 100 to get the percentage.
- **Percentage of automated change approvals**: The proportion of change approvals that were granted automatically by tools without manual intervention. This metric indicates a shift from manual change management to automated governance practices. Improve this metric by integrating more governance checks into automated pipelines and reduce reliance on manual CAB verification. Calculate by dividing the number of automated change approvals by the total number, then multiply by 100 to get the percentage.
- **Non-compliance detection frequency**: The number of non-compliant findings detected over a given period. This metric can indicate the effectiveness of automated guardrails and the current risk level of the environment. Improve this metric by increasing the coverage and quality of automated checks and auto-remediation capabilities. Continuous review and refine controls based on detected findings. Measure by counting the number of detected findings on a regular basis, such as monthly or quarterly.
- **Non-compliance response time**: The time taken from the detection of a non-compliance issue until initial remediation or response. Shorter non-compliance response times decrease the duration of potential exposure, minimizing potential risks and liabilities. Improve this metric by enhancing automated alerting systems, preparing clear escalation paths, and integrating automated remediation capabilities where possible. Measure the timestamp of when non-

compliance is detected and when the first responsive action is taken. Average these durations over a given period to understand typical response times.

Continuous auditing

Facilitate the ongoing automated assessment of system configurations, activities, and operations against internal policies and regulatory standards to measure adherence. This capability allows organizations to glean real-time insights into their security posture, reducing the time and manual effort traditionally associated with auditing. Continuous auditing enhances an organization's ability to swiftly identify and respond to compliance issues, fostering an environment of proactive security and governance.

Topics

- Indicators for continuous auditing
- Anti-patterns for continuous auditing
- Metrics for continuous auditing

Indicators for continuous auditing

Facilitates ongoing automated assessments of system configurations, activities, and operations against internal policies and regulatory standards. This provides real-time insights into an organization's security posture and enables a swift response to compliance issues.

Indicators

- [AG.CA.1] Establish comprehensive audit trails
- [AG.CA.2] Optimize configuration item management
- [AG.CA.3] Implement systematic exception tracking and review processes
- [AG.CA.4] Enable iterative internal auditing practices

[AG.CA.1] Establish comprehensive audit trails

Category: FOUNDATIONAL

Comprehensive audit trails involve capturing, recording, and storing every action taken across your environment. This provides a log of evidence that can offer insights for security and audit teams,

aiding in identifying suspicious activities, evidencing non-compliance, and uncovering the root cause of issues.

Effective DevOps processes are able to streamline both software delivery and the audit process. Automated governance, quality assurance, development lifecycle, and observability capabilities provide a significant amount of data about the processes that are being followed by your organization, and the absence of data indicates those that are not. This data can form a comprehensive audit trail, as steps such as committing code and doing peer reviews can be traced back to specific actors, actions, and timestamps.

Use tools for logging and tracking events should be enforced, along with access controls to maintain the integrity and confidentiality of audit data. Centralize evidence from these tools in a secure, accessible location for easy retrieval during audits. Consider using tools capable of automatically pulling data from resource APIs to collect and organize evidence rather than waiting for data to be pushed to it. It's important that this data remains secure and accessible only to auditors. There must be controls in place to prevent deletion, overwriting, or tampering with the evidence in any way. Regular audits of your audit systems and processes should also be undertaken to ensure their effectiveness.

Recognize that while developers aren't auditors, they play a significant role in the compliance process. Provide training and resources to ensure that everyone on the team understands the concept of compliance as it relates to each systems specific industry.

Related information:

- What Is AWS CloudTrail?
- Automate Cloud Audits - AWS Audit Manager
- Cloud Audit Academy
- Compliance and Auditing with AWS
- Verifiable Controls Evidence Store

[AG.CA.2] Optimize configuration item management

Category: FOUNDATIONAL

Configuration item management involves tracking and recording all resources used across workloads and environments. It enhances visibility, operational efficiency, and helps to ensure

adherence to governance and compliance requirements. It aids in reviewing the frequent changes and updates to infrastructure and application configurations, providing a clear understanding of the system's state at any point in time.

In a DevOps environment, where changes are frequent and continual, use a tool that maintains a resource inventory and continuous configuration log automatically with every change. Establish a consistent tagging strategy to streamline organizing this inventory and to assist in managing resources.

In cloud-based environments, with its high degree of dynamism, scalability, auto-scaling, and elasticity, verify that your tools can keep up with automated, on-demand changes. Understand the AWS shared responsibility model and which teams within your organization are responsible for managing each aspect of the configuration. In all cases, maintain an up-to-date and accurate record of the configuration status of every item, tracking changes over time to provide a comprehensive audit trail.

Related information:

- What Is AWS Config?
- Tagging your AWS resources
- What are resource groups?

[AG.CA.3] Implement systematic exception tracking and review processes

Category: FOUNDATIONAL

DevOps environments are dynamic, characterized by rapid changes and updates. During this rapid development cycle, temporary exceptions might need to be made, for instance, granting greater permissions to a user for a specific task, or turning off a governance control for a system update. While necessary, these exceptions can lead to unexpected issues if not properly managed, and therefore, need to be tracked and revisited.

Implement a process for tracking exceptions, documenting each exception made and help ensure these exceptions are revisited over time. This documentation should take place in a centralized, searchable, and secure location. Critical details such as the reasoning behind the exception, when it was made, who approved it, the business use case, and the anticipated duration should be included. Clear roles and responsibilities should be assigned for the creation, review, and retirement of exceptions to help ensure accountability.

To prevent exceptions from being lingering for vast amounts of time, implement automated alerts for active exceptions that exceed their expected time frame. These alerts serve as reminders to revisit and address these exceptions.

A regular review process of all exceptions should also be scheduled. Depending on the associated risk, these reviews could be conducted on a weekly, monthly, or quarterly basis. These reviews will derive the continued necessity of each exception, which could be investigated to become an approved feature, and investigate any unexpected behavior that may have arisen as a result of the exception. Once an exception is no longer necessary, it should be retired and documentation should be updated.

Related information:

- Amazon's approach to high-availability deployment: Dealing with the real world

[AG.CA.4] Enable iterative internal auditing practices

Category: RECOMMENDED

The continuous nature of DevOps supports the idea of frequent audits, providing real-time insights, and practicing proactive risk management. Consider taking an event-driven auditing approach which allows for immediate detection and response to compliance issues, increasing overall agility and efficiency with automated evidence gathering and report generation occurring constantly within the environment.

Automated alerts and notifications should be implemented to identify potential issues rapidly and notify teams of non-compliance. By running internal audits continuously and integrating the process into the development lifecycle, developers can address compliance issues early on, often before they become a significant problem.

Related information:

- Supported control data sources for automated evidence - AWS Audit Manager

Anti-patterns for continuous auditing

- **Inadequate audit trails**: Not keeping comprehensive audit trails makes it difficult to track actions performed in your environment. This makes it harder to detect suspicious activity or understand

the cause of issues when they occur. Use services like AWS CloudTrail to create a record of actions taken in your AWS environment.

- **Manual evidence review**: Relying on manual processes to collect, aggregate, and review audit data can be error prone and can lead to inconsistencies. Manual review can be time-consuming and often cannot keep pace with the pace of development which leads to reduced ability to quickly respond to compliance issues. Instead, implement automated tools to continuously gather and analyze audit data. Use dashboards and alerts to give a real-time view of system compliance.
- **Viewing audits as a one-time event**: Treating audits as periodic, isolated checks instead of a continuous process can result in significant gaps between audits. During this time, many compliance issues might go undetected. Embed continuous auditing practices into the development lifecycle, including regular, automated checks in pipelines and taking an event-driven approach to auditing. Internal auditors can be embedded within teams, or act as enabling teams, to provide just-in-time audit expertise during planning and development cycles.
- **Expecting auditors to track every feature**: Anticipating that auditing teams will be able to keep up with the rapid pace of feature development and deployments while understanding the nuances of each change is an impractical expectation when practicing DevOps. The primary focus of the auditor should be on processes, controls, and patterns, rather than granular features. Shift the compliance responsibility closer to the source. Educate development teams on auditing requirements and best practices, empowering them to incorporate compliance into their development processes. Put detective, responsive, and preventive controls in place to enforce compliance where possible. This way, developers can produce features with built-in compliance, reducing the load on auditors and ensuring tighter compliance integration.
- **Overlooking developer training**: Assuming that development teams automatically know compliance and auditing best practices without proper training might result in them unknowingly introducing vulnerabilities or non-compliant features. Regularly update training materials and hold sessions, ensuring development teams are well-versed in compliance requirements.

Metrics for continuous auditing

- **Audit lead time**: The total duration taken to complete a single audit cycle, from the initiation of the audit to its completion. This metrics can help in optimizing the audit process and allocating resources efficiently. Long audit times might suggest inefficiencies, bottlenecks, or a lack of automation. Streamline the audit process by incorporating automated tools, refining audit scopes, and ensuring clear communication among involved teams. Measure this metric by

logging the start and end time of each audit cycle. Calculate the difference to get the total time spent per audit.

- **Mean time between audits (MTBA)**: The average time interval between consecutive audits. This metric can help organizations determine if they are auditing frequently enough to catch potential vulnerabilities or compliance issues in a timely manner. If the time between audits is too long, vulnerabilities may go undetected for extended periods, increasing risk and reducing the ability to adhere to regulatory changes or major incidents. As processes become more streamlined and as automation is integrated, this metric should naturally improve. The ideal MTBA will vary based on risk assessments, compliance needs, and system changes. Measure this metric by logging the date of each completed audit. Calculate the difference in dates between consecutive audits and then find the average over a given period, such as quarterly or yearly.
- **Known vulnerability age**: The duration that known vulnerabilities have remained unresolved in the system. This metric helps keep track of the age of vulnerabilities and can provide insights that drive the effectiveness and agility of the remediation process. High severity vulnerabilities that remain open for long periods indicate potential risks. Calculate for each open vulnerability by subtracting the date it was identified from the current date to determine its age. Categorize the results by severity, such as critical, high, medium, and low, as an additional facet to consider.
- **Security control risk**: The potential risk posed by each system based on the effectiveness and health of its implemented security controls. This metric enables pinpointing which systems might be at higher risk due to insufficient or ineffective security controls. Improve this metric by regularly reviewing and updating security controls based on threat modeling, attack vectors, audit findings, and system-specific risks. Evaluate each system's security controls against a standardized framework or criteria. Weight scores based on the importance of the control to the overall system security, and aggregate to get an overall risk level for the system.
- **Exception rate**: The number of compliance exceptions, such as elevated permissions or bypassed controls, relative to the number of changes being made. This metric serves as an early warning system for potential vulnerabilities, emerging anti-patterns, or the need to update controls. Monitoring the nature and severity of exceptions can offer insights into both the quantity and quality of compliance deviations. Improve this metric by regularly reviewing compliance requirements and procedures for granting exceptions. Exceptions should be well-documented, searchable, and only granted when absolutely necessary. Conduct regular exception reviews, especially for major exceptions, to understand the root cause and implement corrective measures. Calculate by dividing the number of exceptions made for a given system by the number of changes made over a specific time frame. Regularly review the nature and severity of these exceptions to differentiate between minor deviations and major compliance breaches.

Observability

The observability saga provides the ability to understand the internal state of your systems through external outputs. It allows teams to detect, troubleshoot, and address issues within their systems effectively and efficiently. In a DevOps model, observability helps teams make timely decisions based on their systems' performance and how well they meet customer needs and business objectives. The AWS Observability Best Practices Guide and AWS Cloud Adoption Framework: Operations Perspective complement this document by providing detailed guidance for practical implementation of observability in an AWS environment.

Capabilities

- Strategic instrumentation
- Data ingestion and processing
- Continuous monitoring

Strategic instrumentation

Strategic instrumentation is a capability aimed at designing and implementing monitoring systems to capture meaningful and actionable data from your applications and infrastructure. This includes collecting telemetry, tracking key performance indicators (KPIs), and enabling data-driven decision making. The goal of strategic instrumentation is to provide deep visibility into your systems, facilitating rapid response to issues, optimizing performance, and aligning IT operations with business objectives by capturing relevant telemetry.

Topics

- Indicators for strategic instrumentation
- Anti-patterns for strategic instrumentation
- Metrics for strategic instrumentation

Indicators for strategic instrumentation

A capability focused on obtaining a deep view into your systems, which aids in rapidly responding to issues, enhancing system performance, and aligning with business objectives.

Indicators

- [O.SI.1] Center observability strategies around business and technical outcomes

- [O.SI.2] Centralize tooling for streamlined system instrumentation and telemetry data interpretation
- [O.SI.3] Instrument all systems for comprehensive telemetry data collection
- [O.SI.4] Build health checks into every service
- [O.SI.5] Set and monitor service level objectives against performance standards

[O.SI.1] Center observability strategies around business and technical outcomes

Category: FOUNDATIONAL

To maximize the impact of observability, it should be closely aligned with both business and technical goals. This means not only monitoring system performance, uptime, or error rates but also understanding how these factors directly or indirectly influence business outcomes such as revenue, customer satisfaction, and market growth.

Adopting the ethos that *"Everything fails, all the time"*, famously stated by Werner Vogels, Amazon Chief Technology Officer, a successful observability strategy acknowledges this reality and continuously iterates, adapting to changes in business environments, technical architecture, user behaviors, and customer needs. It is the shared responsibility of teams, leadership, and stakeholders to establish relevant performance-related metrics to collect to measure established key performance indicators (KPIs) and desired business outcomes. Effective KPIs must be based on the desired business and technical outcomes and be relevant to the system being monitored.

An observability strategy must also identify the metrics, logs, traces, and events necessary for collection and analysis and prescribes appropriate tools and processes for gathering this data. To enhance operational efficiency, the strategy should propose guidelines for generating actionable alerts and define escalation procedures. This way, teams can augment these guidelines to suit their unique needs and contexts.

Use technical KPIs, such as the four golden signals (latency, traffic, errors, and saturation), to provide a set of minimum metrics to focus on when monitoring user-facing systems. On the business side, teams and leaders should meet regularly to assess how technical metrics correlate with business outcomes and adapt strategies accordingly. There is no one-size-fits-all approach to defining these KPIs. Discover customer and stakeholder requirements and choose the technical and business metrics and KPIs that best fit your organization.

For example, one of the most important business-related KPIs for Amazon's e-commerce segment is *orders per minute*. A dip below the expected value for this metric could signify issues affecting

customer experience or transactions, which could affect revenue and customer satisfaction. Within Amazon, teams and leaders meet regularly during weekly business reviews (WBRs) to assess the validity and quality of these metrics against organizational goals. By continuously assessing metrics against business and technical strategies, teams can proactively address potential issues before they affect the bottom line.

Related information:

- AWS Well-Architected Performance Pillar: PERF06-BP02 Define a process to improve workload performance
- AWS Well-Architected Sustainability Pillar: SUS02-BP02 Align SLAs with sustainability goals
- AWS Well-Architected Reliability Pillar: REL11-BP07 Architect your product to meet availability targets and uptime service level agreements (SLAs)
- Monitoring and Observability Implementation Priorities
- AWS Observability Best Practices
- Instrumenting distributed systems for operational visibility
- The Importance of Key Performance Indicators (KPIs) for Large-Scale Cloud Migrations
- What is the difference between SLA and KPI?
- The Four Golden Signals
- Amazon's approach to high-availability deployment: Standard metrics
- The Amazon Software Development Process: Measure Everything

[O.SI.2] Centralize tooling for streamlined system instrumentation and telemetry data interpretation

Category: FOUNDATIONAL

Centralized observability platforms are able to offer user-friendly, self-service capabilities to individual teams that simplify embedding visibility into system components and their dependencies. These tools streamline the onboarding process and offer auto-instrumentation capabilities to automate the monitoring of applications.

Adopt an observability platform that provides observability to teams using the *X as a Service* (XaaS) interaction mode as defined in the Team Topologies book by Matthew Skelton and Manuel Pais. The platform needs to support ingesting the required data sources for effective monitoring, and provide the desired level of visibility into the system components and their dependencies.

Onboarding to the platform should be easy for teams, or support auto-instrumentation to automatically monitor applications for a hands-off experience. This enables the organization to achieve real-time visibility into system data and improve the ability to identify and resolve issues quickly.

The observability platform should offer capabilities to follow requests through the system, the services it interacts with, the state of the infrastructure that these services run on, and the impact of each of these on user experience. By understanding the entire request pathway, teams can identify where slowdowns or bottlenecks occur, whether this latency is caused by hardware or dependencies between microservices that weren't identified during development.

As the observability platform matures, it could begin to offer other capabilities such as trend analysis, anomaly detection, and automated responses, ultimately aiming to reduce the mean time to detect (MTTD) and the mean time to resolve (MTTR) any issues. This can lead to reduced downtime and improved ability to achieve desired business outcomes.

Related information:

- AWS observability tools
- What is Amazon CloudWatch Application Insights?
- Integrated observability partners
- Observability Access Manager
- Apache DevLake
- The Amazon Software Development Process: Self-Service Tools

[O.SI.3] Instrument all systems for comprehensive telemetry data collection

Category: FOUNDATIONAL

All systems should be fully-instrumented to collect the metrics, logs, events, and traces necessary for meeting key performance indicators (KPIs), service level objectives, and logging and monitoring strategies. Teams should integrate instrumentation libraries into the components of new systems and feature enhancements to capture relevant data points, while also ensuring that pipelines and associated tools used during build, testing, deployment, and release of the system are also instrumented to track development lifecycle metrics and best practices.

Chosen libraries and tools should support the efficient collection, normalization, and aggregation of telemetry data. Depending on the workload and existing instrumentation, this could involve

structured log-based metric reporting, or it might rely on other established methods like using StatsD, Prometheus exporters, or other monitoring solutions. The chosen method should align with the workload's specific needs and the complexity involved in instrumenting the solution. Strike a balance between thorough monitoring and the amount of work required to implement and maintain the monitoring solution, to avoid falling into an anti-pattern of excessive instrumentation.

Teams might also consider the use of auto-instrumentation tools to simplify the process of collecting data across their systems with little to no manual intervention, reducing the risk of human error and inconsistencies. Examples of auto-instrumentation include embedding instrumentation tools in shared computer images like AMIs or containers being used, automatically gathering telemetry from the compute runtime, or embedding instrumentation tools into shared libraries and frameworks.

Regardless of how the team chooses to implement it, instrumentation should be designed to accommodate the needs of the specific workload and business requirements. This includes considering factors such as cost, security, data retention, access, compliance, and governance requirements. All collected data must always be protected using appropriate security measures, including encryption and least-privilege access controls.

Related information:

- AWS Well-Architected Performance Pillar: PERF02-BP03 Collect compute-related metrics
- AWS Well-Architected Reliability Pillar: REL06-BP01 Monitor all components for the workload (Generation)
- AWS Well-Architected Cost Optimization Pillar: COST05-BP02 Analyze all components of the workload
- Instrumenting distributed systems for operational visibility
- AWS Observability Best Practices: Data Types
- Embedding metrics within logs
- Application Insights
- Container Insights
- Lambda Insights
- Powertools for AWS Lambda
- AWS Distro for OpenTelemetry

- Build an observability solution using managed AWS services and the OpenTelemetry standard
- The Amazon Software Development Process: Monitor Everything

[O.SI.4] Build health checks into every service

Category: RECOMMENDED

Each service within a system should be configured to include a health check endpoint which provides real-time insight into how the system and its dependencies are performing. Usually manifested as a secure and private HTTP health endpoint (for example, `/actuator/health`), this feature serves as a critical component in monitoring the health status of the overall system, generally including information such as operating status, versions of software running, database response time, and memory consumption. By offering lightweight and fast-responding feedback, they enable sustaining system reliability and availability, two attributes that directly impact customer experience and service credibility.

Observability, governance, and testing tools can invoke these health check endpoints periodically, ensuring the continuous evaluation of system health. However, implementing them should be done with precautionary measures like rate-limiting, thresholding, and circuit breakers to avoid overwhelming the system and to involve human intervention when required.

Integrating health check endpoints is highly recommended for larger, more complex systems or any environment where system availability and rapid issue resolution need to be prioritized. In systems with high interoperability, such as microservices architecture, the presence of health check endpoints in every service becomes even more critical as they help identify issues related to specific services in the system. This can significantly reduce the debugging time and enhance the efficiency of the development process.

For mission critical workloads it may be beneficial to explore additional mitigation strategies to prevent widespread failure due to faulty deployments. These strategies could include alerting mechanisms when overall fleet size, load, latency, or error rate are abnormal, and phased deployments to ensure thorough testing before full-scale implementation. These preventive deployment measures complement health check endpoints and can prevent a potentially flawed deployment from propagating throughout the entire system.

Related information:

- Implementing health checks

[O.SI.5] Set and monitor service level objectives against performance standards

Category: RECOMMENDED

Teams should define and document Service Level Objectives (SLOs) for every service, regardless of whether it is directly consumed by external customers or used internally. SLOs should be accessible and clearly communicate the expected standard of performance and availability for the service. While Service Level Agreements (SLAs), which define a contract that must be met for service availability, are typically defined and published for services that are directly consumed by customers, it is equally important to establish SLOs for services consumed internally. Such SLOs help ensure performance standards are met, even in the absence of formal SLAs, and can also act as data points for meeting Key Performance Indicators (KPIs).

The creation of SLOs should be a collaborative effort involving both the business and technical teams. The technical team must provide realistic estimations based on the system's capabilities and constraints, while the business team ensures these align with the company's business objectives and internal standards.

SLOs should be SMART (Specific, Measurable, Achievable, Relevant, and Time-bound). This means that they should clearly define what is to be achieved, provide a way to measure the progress, ensure that the goals can realistically be achieved given the current resources and capabilities, align with business objectives, and set a time frame for the achievement of these goals.

When defining SLOs, rather than using averages, it is preferable to use percentiles for measurement. Percentiles are more reliable in detecting outliers and provide a more accurate representation of the system's performance. For example, a 99th percentile latency SLO means that 99% of requests should be faster than a specific threshold, providing a much more accurate depiction of the service's performance than an average would.

Teams internally measure and monitor their SLOs to ensure they are meeting the defined business and technical objectives. When measuring against a SLO, teams produce Service Level Indicators (SLIs), which are the actual measurements of the performance and availability of the service at that point in time. SLIs are used to evaluate whether the service is meeting the defined SLOs. By continuously tracking SLIs against the target SLOs, teams can detect and resolve issues that impact the performance and availability of their services while ensuring that they continue to meet both external customer expectations and internal performance standards.

Continuous improvement and periodic review of SLOs are required to ensure they remain realistic and aligned with both the system's capabilities and the business's objectives. Any changes to the system that could affect its performance should trigger a review of the associated SLOs.

Related information:

- What Is SLA (Service Level Agreement)?
- What is the difference between SLA and KPI?
- AWS Well-Architected Framework - Reliability Pillar
- Designed-For Availability for Select AWS Services
- Understanding KPIs ("Golden Signals")
- The Importance of Key Performance Indicators (KPIs) for Large-Scale Cloud Migrations

Anti-patterns for strategic instrumentation

- **Excessive data collection**: Over-instrumentation leads to unnecessary data collection, escalating costs, and storage requirements. Prioritize collecting relevant data that provides valuable insights into the customer experience while interacting with systems and your organization's desired business outcomes. For use cases needing verbose datasets, implement aggressive data retention policies. This approach balances the need for detailed, short-term data for efficient troubleshooting without excessive costs.
- **Lack of standardization:** Inconsistency in Service Level Agreements (SLAs), Service Level Objectives (SLOs), and Key Performance Indicators (KPIs), and metric formats impedes understanding and interpretation of metrics. DevOps principles emphasize communication, collaboration, and visibility, which inconsistent standards undermine. Establish standardized guidelines for defining and formatting these metrics, and use a centralized observability platform for tracking and enforcing these standards, promoting continuous improvement.
- **Monitoring in isolation:** Observing individual components in isolation decreases visibility into system interactions and dependencies, hindering root cause identification, adds delay to detection time, and can generate inaccurate alerts. Adopt a holistic observability approach through a centralized platform, taking into account the entire system and its interdependencies.
- **Reactive monitoring:** Reactive monitoring, triggered by incidents or issues, can increase downtime and incur additional cost over time. Embrace a proactive, continuous monitoring stance that tracks system performance and user behaviors. Implement thresholds, alerts, predictive analytics, and constant data collection across all system components to detect and address issues before affecting the end user.
- **Misaligned SLOs:** Service Level Objectives (SLOs) defined solely by business teams without the input from technical teams can result in unachievable targets, leading to frequent breaches of Service Level Agreements and missed KPIs. Defining SLOs should be a collaborative process

involving both business and technical teams to align technical realities with business objectives and customer expectations.

Metrics for strategic instrumentation

- **Instrumented systems coverage**: The percentage of systems that are instrumented to collect telemetry data which provides broader visibility into the performance and health of the systems and allows teams to identify any gaps in their monitoring coverage. Calculate the number of instrumented systems, divide by the total number of systems, and multiply by 100 to obtain the percentage.
- **SLO adherence**: The percentage of time a service meets Service Level Objectives (SLOs), indicating that the service is consistently meeting its performance and reliability targets, making for a better user experience. To measure this metric, calculate the amount of time the service meets its SLOs, divide by the total time, and multiply by 100 to obtain the percentage.

Data ingestion and processing

Data ingestion and processing involves the collection, centralization, and analysis of data from multiple sources. This data, when effectively ingested and processed, helps teams to understand the availability, security, performance, and reliability of their systems in real-time. Through streamlining data ingestion and processing, teams can make quicker and more effective decisions, enhancing overall agility and reliability of systems.

Topics

- Indicators for data ingestion and processing
- Anti-patterns for data ingestion and processing
- Metrics for data ingestion and processing

Indicators for data ingestion and processing

The collection, centralization, and analysis of data from various sources. With this capability, teams can make quicker, more effective decisions, enhancing their systems' agility, security, and reliability.

Indicators

- [O.DIP.1] Aggregate logs and events across workloads
- [O.DIP.2] Centralize logs for enhanced security investigations

- [O.DIP.3] Implement distributed tracing for system-wide request tracking
- [O.DIP.4] Aggregate health and status metrics across workloads
- [O.DIP.5] Optimize telemetry data storage and costs
- [O.DIP.6] Standardize telemetry data with common formats

[O.DIP.1] Aggregate logs and events across workloads

Category: FOUNDATIONAL

Logs and events should be aggregated across multiple workloads to provide a comprehensive view of the entire system. This enables teams to troubleshoot, identify patterns, and resolve operational issues.

Implement a log aggregation solution that supports collecting logs from various sources and provides functions for filtering, searching, visualizing, and alerting. Make sure the solution provides real-time data collection, supports necessary data sources, and offers visualization options. The tool should be accessible to application teams, allowing them to monitor and troubleshoot their system as needed.

Related information:

- AWS Well-Architected Reliability Pillar: REL11-BP01 Monitor all components of the workload to detect failures
- Cross-account cross-Region CloudWatch console
- Collect, analyze, and display Amazon CloudWatch Logs in a single dashboard with the Centralized Logging on AWS solution
- Centralized Logging with OpenSearch
- Sending Logs Directly to Amazon S3
- One Observability Workshop

[O.DIP.2] Centralize logs for enhanced security investigations

Category: FOUNDATIONAL

Effective security investigations require the aggregation, standardization, and centralization of logs and events so they are readily accessible to investigation teams. Centralized logs and event data

enhance the ability of security teams to conduct effective investigations, improve threat detection, and accelerate incident response times.

Use cloud native tools or Security Information and Event Management (SIEM) solutions to aggregate, standardize, and centralize logs and event data, while respecting regional boundaries and data sovereignty requirements. These tools are designed to collect and analyze logs and security events from various sources to provide a centralized view of an organization's security posture. Centralizing, normalizing, deduping, and removing unnecessary data allows security teams to use automation and scripted investigation tools which leads to a faster and more efficient response process.

Given the sensitivity of this data, verify that the data is accessible only to authorized security personnel and that strong access controls are in place to maintain data security and confidentiality. Only grant least-privilege permission to the data so that it is only accessible to authorized users with the minimum level of access required to perform investigations. For instance, access to overwrite this data should be restricted.

Related information:

- AWS Well-Architected Performance Pillar: PERF07-BP02 Analyze metrics when events or incidents occur
- Collect, analyze, and display Amazon CloudWatch Logs in a single dashboard with the Centralized Logging on AWS solution
- Cross-account cross-Region CloudWatch console
- AWS Well-Architected Framework - Security Pillar - Detection
- Amazon Security Lake
- Centralized Logging on AWS
- Amazon OpenSearch Service
- Centralized Logging with OpenSearch
- AWS Marketplace: SIEM

[O.DIP.3] Implement distributed tracing for system-wide request tracking

Category: RECOMMENDED

Distributed tracing is a method to track requests as they move through distributed systems. It provides insights into system interactions across multiple services and applications, enabling quicker issue identification and resolution.

Use a tracing solution that is scalable, provides real-time data collection, and supports comprehensive visualization of tracing data. Integrate this solution with the log and event aggregation tools to enhance system-wide visibility. This gives a comprehensive view of the entire system and its dependencies, facilitating quick identification and resolution of issues.

Related information:

- AWS Well-Architected Reliability Pillar: REL06-BP07 Monitor end-to-end tracing of requests through your system
- Distributed Tracing System – AWS X-Ray
- Amazon CloudWatch ServiceLens
- Amazon Managed Grafana
- AWS X-Ray integration with Grafana

[O.DIP.4] Aggregate health and status metrics across workloads

Category: RECOMMENDED

Aggregate health and status metrics across all workloads for a unified view of the system's overall health. Aggregated health metrics provide a snapshot of the system's overall health and performance, aiding in proactive issue detection and efficient resource management.

Use a monitoring solution that allows aggregation of health metrics across all applications, supports real-time data collection, and provides intuitive visualization of metrics data. Integration with the logging, events, and tracing tools can provide a comprehensive view of overall system health.

Related information:

- Amazon Managed Grafana
- Amazon Managed Service for Prometheus
- Application Monitoring with Amazon CloudWatch
- AWS Health Aware

[O.DIP.5] Optimize telemetry data storage and costs

Category: RECOMMENDED

Optimize costs associated with storing and processing large amounts of telemetry data by using techniques like data filtering and compression. When dealing with non-security related telemetry data, data sampling can also be an effective method to reduce costs.

Select cost-effective solutions and consumption-based resources for data storage. Be strategic about data retention—remove unused or unnecessary data from storage regularly. Also, be selective about which data sources are ingested and ensure they are required for effective analysis to avoid unnecessary spend. Always remember that while managing costs is important, it should not compromise the integrity and completeness of your data, especially when it comes to security.

Related information:

- AWS Well-Architected Performance Pillar: PERF03-BP01 Understand storage characteristics and requirements
- AWS Well-Architected Sustainability Pillar: SUS04-BP05 Remove unneeded or redundant data

[O.DIP.6] Standardize telemetry data with common formats

Category: RECOMMENDED

Normalize telemetry data using a common format or standard schema to enhance consistency in data collection and reporting. This facilitates seamless correlation and analysis across multiple facets of observability, such as system performance, user behaviors, and security events, improving the overall speed and accuracy of detection and response in any of these areas.

Two notable open-source projects supporting this goal are OpenTelemetry and the Open Cybersecurity Alliance Schema Framework (OCSF). OpenTelemetry provides a single set of APIs, libraries, agents, and collector services to capture distributed traces and metrics from your application and send them to any observability platform. OCSF, on the other hand, is an extensible, vendor-agnostic project designed to simplify data ingestion and normalization specifically for cybersecurity events.

Utilize a common telemetry format to streamline these processes, reduce associated costs of data processing, and allow teams to focus more on detecting and responding to actionable events. Guidelines should be established for the collection and reporting of data, enforcing

consistency across all teams. Adopting and effectively using standard schemas or frameworks like OpenTelemetry and OCSF can provide considerable advantages in achieving comprehensive observability.

Related information:

- OCSF Schema
- OCSF GitHub
- AWS Distro for OpenTelemetry
- OpenTelemetry

Anti-patterns for data ingestion and processing

- **Over-reliance on ETL Tools:** Over-relying on ETL (Extract, Transform, Load) tools for data processing can lead to inflexibility and difficulties adapting to data source changes. Where possible, use tools with native integrations that allow ETL-free data processing and analysis pipelines, enabling a more flexible and scalable way to integrate data from multiple sources without introducing additional operational overhead.
- **Ignoring event correlation:** Ignoring the correlation of multiple alerts can hide broader issues. Incorporate event correlation into the observability strategy to quickly identify and resolve problems across multiple tools and systems. Utilize distributed tracing tools to trace requests across multiple services and dependencies to identify bottlenecks or issues, centralized logs and events for security investigations, and use normalized data formats to enable correlation of telemetry from multiple sources.
- **Inefficient data analysis:** Relying on monolithic or manual data processing methods leads to inefficient data analysis. Monolithic data processing of large volumes leads to long wait times, slow detection and reaction times, and potentially increased cost. Manual data processing, on the other hand, is error-prone and time-consuming. Overcome these inefficiencies by adopting scalable and distributed architectures like serverless computing, capable of handling large data volumes in parallel. Data processing should be automated wherever possible to ensure consistent, error-free, and efficient data analysis.
- **Lack of data governance:** Poor data governance practices can lead to inaccurate data, poor decision-making, and compliance risks. Establish and enforce data governance policies, including data quality checks, granular access control, and data provenance tracking.

Metrics for data ingestion and processing

- **Data ingestion rate**: The amount of data ingested by monitoring systems in a given time period which indicates that the system can effectively process large volumes of telemetry data, leading to more accurate insights. Measure this metric by calculating the volume of data ingested by the monitoring systems per unit of time.
- **Data processing latency**: The time it takes for telemetry data to be processed and made available for analysis. Lower data processing latency aims to quickly assess and act on insights from telemetry data. Measure the time elapsed between data ingestion and the availability of processed data for analysis.
- **Data cost efficiency**: Measuring the cost of collecting, storing, and processing telemetry data compared to the number of actionable insights generated or decisions made based on these insights. This metric assures that resources are utilized effectively and unnecessary expenses are minimized. Calculate the total cost of data collection, storage, and processing, and contrast it to the actionable insights they provide.
- **Anomaly detection rate**: The percentage of anomalies detected by the monitoring systems. A higher anomaly detection rate indicates that the system is effective in identifying potential issues, enabling teams to proactively address them. Measure this metric by calculating the number of anomalies detected by the monitoring systems, divided by the total number of events, then multiply by 100 for the percentage.

Continuous monitoring

Continuous monitoring is the real-time observation and analysis of telemetry data to help optimize system performance. It encompasses alert configuration to notify teams of potential issues, promoting rapid response. Post-event investigations provide valuable insights to continuously optimize the monitoring process. By integrating artificial intelligence (AI) and machine learning (ML), continuous monitoring can achieve a higher level of precision and speed in detecting and responding to system issues.

Topics

- Indicators for continuous-monitoring
- Anti-patterns for continuous monitoring
- Metrics for continuous monitoring

Indicators for continuous-monitoring

This is the real-time observation and analysis of telemetry data. This capability provides continuous optimization through alert tuning and post-event investigations.

Indicators

- [O.CM.1] Automate alerts for security and performance issues
- [O.CM.2] Plan for large scale events
- [O.CM.3] Conduct post-incident analysis for continuous improvement
- [O.CM.4] Report on business metrics to drive data-driven decision making
- [O.CM.5] Detect performance issues using application performance monitoring
- [O.CM.6] Gather user experience insights using digital experience monitoring
- [O.CM.7] Visualize telemetry data in real-time
- [O.CM.8] Hold operational review meetings for data transparency
- [O.CM.9] Optimize alerts to prevent fatigue and minimize monitoring costs
- [O.CM.10] Proactively detect issues using AI/ML

[O.CM.1] Automate alerts for security and performance issues

Category: FOUNDATIONAL

Alerts should automatically notify teams when there are indicators of malicious activity, compromise, or performance degradation. Effective alerting accelerates incident response times, enabling teams to quickly address and resolve issues before they can significantly impact system performance or security. Without automatic alerting, teams can suffer from delayed response times that can lead to prolonged system downtime or increased exposure to security threats.

Implement centralized alerting mechanisms to track anomalous behavior across all systems. Define specific conditions and thresholds that, when breached, will raise alerts. Verify that the alerts are delivered to the appropriate teams by email, text message, or the team's preferred notification system. Integrating these alerts into your centralized incident management systems can also help in the automatic creation of tickets, aiding faster resolution.

In a more advanced workflow, alerts can be integrated with automated governance systems to start remediation actions immediately upon detection or to gather additional insights that will aid investigations.

Related information:

- AWS Well-Architected Performance Pillar: PERF07-BP06 Monitor and alarm proactively
- AWS Well-Architected Reliability Pillar: REL06-BP03 Send notifications (Real-time processing and alarming)
- What is Anomaly Detection?
- AWS Security Hub CSPM
- Amazon OpenSearch Service
- AWS Health Aware
- Amazon's approach to high-availability deployment: Anomaly detection

[O.CM.2] Plan for large scale events

Category: FOUNDATIONAL

A large scale event (LSE) is an incident that has a wide impact, such as service outages or major security incidents. Proper management of LSEs help to ensure business continuity, maintain customer trust, and reduce the negative impact of such events.

Prepare a detailed incident management plan, outlining the roles, responsibilities, and processes to be followed in the event of a large-scale incident. At a minimum, the plan should outline how teams expect to maintain availability and reliability of systems by having the capability to automatically scale resources, re-route traffic, and failover to backup systems when required.

Related information:

- Disaster Recovery of Workloads on AWS: Recovery in the Cloud
- Incident management
- Disaster recovery plan
- Amazon's approach to security during development: Handling a security incident

[O.CM.3] Conduct post-incident analysis for continuous improvement

Category: FOUNDATIONAL

Drive the continuous improvement of analysis and response mechanisms by holding post-incident retrospectives. The post-incident retrospectives allow teams to identify gaps and areas for

improvement by analyzing the actions that were taken during an incident. These retrospectives should not be used to place blame or point fingers at individuals. Instead, they provide the time for teams to optimize their response process for future incidents and helps ensure that they are continuously learning and improving their incident response capabilities. This approach leads to more efficient and effective resolution of incidents over time.

All relevant stakeholders involved with the incident and the system should attend the retrospective. At a minimum, this should include the leaders and individual contributors who support the system, the customer advocates, those who were impacted by the issue internally, as well as those involved with the resolution of the issue. The post-incident retrospective findings should be anonymized, as to not place blame onto any individuals, and should be well documented and shared with the broader organization so that others may learn as well.

Related information:

- AWS Well-Architected Performance Pillar: PERF07-BP02 Analyze metrics when events or incidents occur
- AWS Well-Architected Reliability Pillar: REL12-BP02 Perform post-incident analysis
- AWS Well-Architected Operational Excellence Pillar: OPS11-BP02 Perform post-incident analysis

[O.CM.4] Report on business metrics to drive data-driven decision making

Category: FOUNDATIONAL

Business metrics for all systems should be accessible and comprehensible to leaders and key stakeholders. These metrics should inform key performance indicators (KPIs), service level objectives (SLOs), service level agreement (SLA) adherence, user engagement, conversion rates, and other metrics relevant to the business sides of your operations.

Just like with technology metrics, continuous monitoring tools should be used to detect when business metrics cross predefined thresholds, triggering alerts that highlight significant deviations or potential issues. These alerts should inform timely and data-driven decision-making, helping identify areas for improvement, optimizing system performance, and aligning actions with overarching business goals.

Create dashboards or reports that present these metrics, as well as how they are tracking against KPIs and SLAs, in a user-friendly, non-technical format. Ensure the data is up-to-date, accurate, and accessible to less technical leaders so that it can be used to make informed business decisions.

Observability isn't merely about data collection—it is about turning that data into actionable insights that drive better outcomes for both the technology and business sides of the organization.

Fast feedback leads to success. Continuously monitoring and alerting on business metrics is becoming foundational for organizations committed to maximizing the value they get from their technology investments and for maintaining the quality of their digital services.

Related information:

- AWS Well-Architected Performance Pillar: PERF07-BP05 Review metrics at regular intervals
- Operational observability
- The Amazon Software Development Process: Measure Everything
- Using Cloud Fitness Functions to Drive Evolutionary Architecture

[O.CM.5] Detect performance issues using application performance monitoring

Category: RECOMMENDED

Application Performance Monitoring (APM) refers to the use of tools to monitor and manage the ongoing, real-time performance and availability of systems in production environments. APM tools help in maintaining the performance of systems by identifying performance issues early on. This leads to quicker resolution of issues, improved user experience, and reduced downtime.

To comprehensively monitor application performance, implement both Real-User Monitoring (RUM) and Synthetic Monitoring. These APM tools are recommended detect and diagnose performance issues in production systems. These APM tools enable teams to proactively detect and diagnose complex application performance problems to maintain an expected level of service.

RUM captures performance metrics based on actual user interactions. Analyze real user data to understand areas of the system that are frequently used and might benefit from performance improvements. This data can then be used to identify and debug client-side issues to optimize end-user experience.

On the other hand, Synthetic Monitoring involves writing scripts that simulate user interactions, known as canaries, to continuously monitor endpoints and APIs. Canaries follow the same routes and perform the same actions as a customer, allowing for the continuous verification of the customer experience even in the absence of actual customer traffic. By using insights from RUM, you can optimize which canaries to run continuously, ensuring they closely mimic the most

common user paths. This strategy ensures potential issues are identified before impacting users, offering a seamless user experience.

Both tools collect metrics on response time, resource utilization, and other performance-related indicators, forming a holistic approach to continuous performance monitoring in production environments.

Related information:

- AWS Well-Architected Performance Pillar: PERF01-BP06 Benchmark existing workloads
- What is APM (Application Performance Monitoring)?
- Real-User Monitoring (RUM) for Amazon CloudWatch
- Amazon CloudWatch ServiceLens
- Amazon CloudWatch Synthetics
- Amazon CloudWatch Internet Monitor

[O.CM.6] Gather user experience insights using digital experience monitoring

Category: RECOMMENDED

Digital Experience Monitoring (DEM) involves simulating user interactions with applications to measure the performance and availability of services from the perspective of end users. DEM allows teams to proactively detect and resolve issues that may impact user experience. It also helps in validating that application updates or changes do not negatively impact user experience.

Implement APM tools, such as synthetic transaction monitoring using canaries to simulate user interactions with your application and measure the response times and accuracy of the results.

DEM is recommended as it provides important insights into the user experience and helps detect issues that may impact user experience

Related information:

- Amazon CloudWatch Synthetics
- AWS Marketplace - Digital Experience Monitoring

[O.CM.7] Visualize telemetry data in real-time

Category: RECOMMENDED

Visualization tools simplify the task of correlating and understanding large, complex datasets. Using these tools, teams are able to detect trends, patterns, and anomalies in data in a readily available and easy to understand way.

Utilize visualization tools to correlate and comprehend large sets of telemetry data in real-time. Visualization tools support the uniquely human capability to discover patterns that automated tools may otherwise miss. Choose a tool that provides a clear view of system data at varying time intervals, allowing teams to easily detect issues both during or after they arise. Ensure that the tool is flexible and customizable, so that teams can adjust the views and create dashboards based on their unique needs.

Related information:

- Building dashboards for operational visibility
- Building Prowler into a QuickSight powered AWS Security Dashboard

[O.CM.8] Hold operational review meetings for data transparency

Category: RECOMMENDED

Operational review meetings are regular gatherings where teams from across the organization come prepared with an operational dashboard that showcases telemetry data, performance metrics, and other insights into operations for their products. The aim is present to the broad audience to share and gain different perspectives on changes in the data, whether it is a spike, dip, or trend. This promotes a culture of transparency, preparedness, and continuous improvement throughout the organization.

Amazon implements this by holding weekly Ops review meetings and using the spinning wheel as a random selection method for which team will present. The randomness of the selection ensures that each team comes prepared, as any team can be called upon to present. When presenting, teams must be capable of deep diving into the data, explaining root causes behind notable data changes, and articulating the steps taken or planned to rectify any anomalies. This pushes teams to maintain high-quality operational dashboards that reflect the real-time health and performance of their services.

Related information:

- AWS Well-Architected Performance Pillar: PERF07-BP05 Review metrics at regular intervals
- AWS Well-Architected Reliability Pillar: REL06-BP06 Conduct reviews regularly

- AWS Ops Wheel
- AWS Well-Architected Operational Excellence Pillar: OPS11-BP07 Perform operations metrics reviews
- The Amazon Software Development Process: Monitor Everything

[O.CM.9] Optimize alerts to prevent fatigue and minimize monitoring costs

Category: RECOMMENDED

Reduce the number of ineffective alerts as well as the costs associated with monitoring by optimizing rules and thresholds for alerts based on business impact and issue severity. By continuously refining rules and thresholds for alerts, teams can minimize unnecessary notifications, reducing the time and resources spent on non-critical issues. This helps teams focus on high-impact issues, enhancing productivity and efficiency.

Set up alert rules and thresholds based on the severity and business impact of potential issues. Teams should leverage cost-effective methods for delivering notifications, and work to reduce the amount of false positive notifications. Regular reviews and adjustments of these rules and thresholds should be done based on usage patterns to further minimize costs, while still ensuring that teams are alerted to critical issues in a timely and effective manner.

Implementing intelligent alerting strategies, such as alert deduplication, aggregation, and comprehensive data visualization can help to reduce cost, alert fatigue, and data overload that comes with having too many alerts.

[O.CM.10] Proactively detect issues using AI/ML

Category: OPTIONAL

Adopt data-driven AI/ML monitoring tools and techniques like Artificial Intelligence Operations (AIOps), ML-powered anomaly detection, and predictive analytics solutions, to detect issues and performance bottlenecks proactively—even before system performance is impacted.

Choose a tool that can leverage data and analytics to automatically infer predictions, and begin to feed data to it and inject failure to test the validity of the tool. These tools should have access to both historical and real-time data. Once operational, the tool can automatically detect issues, predict impending resource exhaustion, detail likely causes, and recommend remediation actions to the team. Ensure that there is a feedback loop to continuously train and refine these models based on real-world data and incidents.

Start small when setting up alerts from these tools to avoid alert fatigue and maintain trust in the system. As the tool becomes more familiar with the data patterns, teams can gradually increase the alerting scope. Regularly validate the tool's predictions by injecting failures and observing the responses.

Related information:

- Machine-Learning-Powered DevOps - Amazon DevOps Guru
- Amazon GuardDuty
- Continuous Monitoring and Threat Detection
- Gaining operational insights with AIOps using Amazon DevOps Guru Workshop
- What Is Anomaly Detection?
- What Is Predictive Analytics?

Anti-patterns for continuous monitoring

- **Blame culture**: Encouraging a culture where individuals are blamed for errors or failures can deter open communication, and the collaborative diagnosis of issues. In a blame culture, team members may hide or underreport issues for fear of retribution. Instead, foster a culture of shared responsibility where failures are seen as opportunities for learning and improvement. Encourage open discussions and retrospectives to understand the root causes and to find ways to prevent similar issues in the future.
- **Overlooking derived metrics**: Relying solely on surface-level metrics without deriving deeper insights can lead to unaddressed issues and potential service disruptions. Ensure that monitoring includes a comprehensive understanding of system performance by analyzing metrics in depth, such as distinguishing between latencies based on query size or categorizing error types. Use techniques like anomaly detection and consider metrics like trimmed means for latency to reveal patterns obscured by averages.
- **Inadequate monitoring coverage:** Not monitoring every critical system or frequently reviewing your monitoring strategy can lead to undetected issues or performance degradation. Regularly assess and update monitoring coverage, ensuring that all systems and applications are being observed. A symptom of this anti-pattern is "no dogs barking," where the absence of expected alerts or metrics itself can indicate an issue.
- **Noisy and unactionable alarms:** If alarms frequently sound without actionable cause, trust in the alerting system diminishes, risking slower response times or overlooked genuine alerts. Ensure that alerts are both actionable and significant by continuously evaluating the

outcomes they lead to. Implement mechanisms to mute false positives and adjust overly sensitive alarms.

Metrics for continuous monitoring

- **Mean time to detect (MTTD)**: The average time it takes to detect a performance issue, attack, or compromise. A shorter MTTD helps organizations respond more quickly to incidents, minimizing damage and downtime. Track this metric by calculating the average time from when incidents occur to when they're detected by the monitoring systems. This includes both automated system detections and manual reporting.
- **Mean time between failures (MTBF)**: The average time interval between consecutive failures in the production environment. Tracking this metric helps to gauge the reliability and stability of a system. It can be improved by improving testing capabilities, proactively monitoring for system health, and have post-incident reviews to address root causes. Monitor system outages and failures, then calculate the average time between these events over a given period.
- **Post-incident retrospective frequency**: The frequency at which post-incident retrospectives are held. Holding regular retrospectives help teams continuously improve analysis and incident response processes. Measure this metric by counting the number of retrospectives conducted within specified intervals, such as monthly or quarterly. This can also be validated against the total number of incidents to understand if all incidents are followed up with a retrospective.
- **False positive rate**: The percentage of alerts generated that are false positives, or incidents that do not require action. A lower false positive rate reduces alert fatigue and ensures that teams can focus on genuine issues. Calculate by dividing the number of false positive alerts by the total number of alerts generated and multiplying by 100 to get the percentage.
- **Application performance index (Apdex)**: Measures user satisfaction with application responsiveness using a scale from 0 to 1. A higher Apdex score indicates better application performance, likely resulting in improved user experience, while a lower score means that users might become frustrated.

  To determine the Apdex score, start by defining a target response time that represents an acceptable user experience for your application. Then, categorize every transaction in one of three ways:

  - **Satisfied**, if its response time is up to and including the target time.
  - **Tolerating**, if its response time is more than the target time but no more than four times the target time.

- **Frustrated**, for any response time beyond four times the target time.

Calculate the Apdex score by adding the number of *Satisfied* transactions with half the *Tolerating* transactions. Then, divide this sum by the total number of transactions. Continuously monitor and adjust your target time based on evolving user expectations and leverage the score to identify and rectify areas that contribute to user dissatisfaction.

Contributors

Contributors to this document include:

- Michael Rhyndress, AWS DevOps Sagas lead, Amazon Web Services
- Ahmed Khan, Sr. DevSecOps Consultant, Amazon Web Services
- Brian Boelsterli, Sr. Practice Manager, Amazon Web Services
- Casey Lee, Principal DevOps Consultant, Amazon Web Services
- David Hessler, Sr. DevSecOps Consultant, Amazon Web Services
- Paul Duvall, Director DevSecOps, Amazon Web Services
- Shalabh Nigam, DevOps Consultant, Amazon Web Services
- Tobin Quadros, DevOps Consultant, Amazon Web Services

Special thanks to the Global AWS DevOps community for their reviews and contributions.

Further reading

- AWS Well-Architected
- AWS Architecture Center
- The Amazon Builders' Library
- AWS Deployment Pipeline Reference Architecture
- AWS Observability Best Practices
- AWS Ramp-Up Guide: DevOps Engineer

Document history

To be notified about updates to this whitepaper, subscribe to the RSS feed.

Change	Description	Date
Initial publication	Whitepaper first published.	September 20, 2023

> **Note**
>
> To subscribe to RSS updates, you must have an RSS plugin enabled for the browser that you are using.

Notices

Customers are responsible for making their own independent assessment of the information in this document. This document: (a) is for informational purposes only, (b) represents current AWS product offerings and practices, which are subject to change without notice, and (c) does not create any commitments or assurances from AWS and its affiliates, suppliers or licensors. AWS products or services are provided “as is” without warranties, representations, or conditions of any kind, whether express or implied. The responsibilities and liabilities of AWS to its customers are controlled by AWS agreements, and this document is not part of, nor does it modify, any agreement between AWS and its customers.

AWS Glossary

For the latest AWS terminology, see the AWS glossary in the *AWS Glossary Reference*.