# Architecting Cloud-Native Applications for Asset Management on AWS: A Guide to Best Practices

### I. Executive Summary: Architecting Cloud-Native Applications for Financial Services

This report serves as a definitive guide to architecting resilient, compliant, and performant cloud-native applications on the Amazon Web Services (AWS) platform for a financial services firm specializing in asset management. The foundational principles presented here are derived from the AWS Well-Architected Framework and its Financial Services Industry (FSI) Lens, which provides specific guidance for designing systems that meet the stringent security, data privacy, and resiliency requirements of the financial sector.1

The AWS Well-Architected Framework is built on six pillars: Operational Excellence, Security, Reliability, Performance Efficiency, Cost Optimization, and Sustainability.2 While all six are critical, the FSI Lens places a heightened emphasis on security, resilience, and data privacy to align with the risk and control objectives of financial institutions.1 A fundamental understanding of the AWS Shared Responsibility Model is essential. While AWS is responsible for the security

*of* the cloud—including the physical infrastructure, global network, and the underlying services themselves—the client is responsible for security *in* the cloud. This includes protecting application data, managing user permissions, and ensuring that workloads comply with applicable regulatory requirements such as the Gramm-Leach-Bliley Act (GLBA) and the Payment Card Industry Data Security Standard (PCI DSS).3 All best practices detailed in this report are framed to empower developers to fulfill this responsibility and build with confidence on AWS's secure and compliant infrastructure.4

### II. Foundational Architectural Principles for a Regulated Environment

#### Designing for Event-Driven Architectures (EDA): The Core Paradigm

Modern cloud-native applications in the financial sector increasingly adopt an event-driven architecture (EDA) as the core paradigm for building scalable and resilient systems.5 An EDA is a software approach where decoupled services communicate by reacting to a change in state, known as an event.5 This architectural style consists of three primary components: event producers, event routers, and event consumers.5 A producer publishes an event to an event router, which then filters and pushes the event to one or more interested consumers.

This decoupling provides several key benefits for a financial institution. Services can be scaled, updated, and deployed independently, which enhances operational agility.5 If one service experiences a failure, the rest of the system can continue to operate because the event router acts as an elastic buffer, accommodating surges in workload and preventing a single point of failure from cascading through the architecture.5 Furthermore, EDAs are push-based, meaning events are processed on-demand, which can reduce costs by eliminating the need for continuous polling and reducing network bandwidth consumption.5

#### Idempotency and Fault Tolerance: The Non-Negotiable Requirements

The user's specific concerns about S3 event duplication and potential delivery failures point to a fundamental architectural challenge in any financial system: ensuring data integrity and preventing unintended duplicate actions. In a financial context, the repeated execution of a non-idempotent operation—such as a funds transfer or a trade execution—due to a duplicated event could lead to a catastrophic failure with significant financial and regulatory consequences.7 Therefore, idempotency is not merely a desirable feature; it is a non-negotiable requirement.

Idempotency refers to an operation that produces the same result regardless of how many times it is executed with the same input.7 Systems must be designed to safely process a message or event multiple times without causing adverse effects or leading to inconsistent data states.9 The solution lies not just in the messaging service but in the application's processing logic and its data persistence layer.10 A robust pattern for achieving this is to include a unique event identifier in every message. A persistent store, such as a DynamoDB table, can then be used to track processed events. By using a composite key on the DynamoDB table, the system can ensure that a record with a specific unique identifier is only written to the table once, providing a last-resort idempotency guarantee at the data layer.7

### III. Service Deep Dives: Developer-Focused Best Practices

#### 1. Amazon S3 Event Notifications

Amazon S3 Event Notifications are a key component of an event-driven architecture, providing a mechanism to trigger automated workflows in response to changes in an S3 bucket, such as object creation or deletion.9 For financial services, these events can signify the arrival of a new data file, such as a large transaction log or a market data feed, that needs to be processed.

* **Best Practices:**
  + **Design for Duplicates:** Downstream systems must be designed to handle duplicate events safely and idempotently.9 As S3 Event Notifications are delivered on an "at-least-once" basis, the consumer application must implement logic to verify whether a message has already been processed using a unique event identifier.9
  + **Configure for Durability:** For mission-critical workflows, avoid configuring S3 to trigger a Lambda function directly. Instead, use an intermediate, durable queue (SQS) as the event destination.9 This acts as a buffer, ensuring that events are not lost if the downstream Lambda function is throttled or fails, and provides the ability to retry failed messages with a dead-letter queue.9
  + **Precise Filtering:** Use object key name prefixes and suffixes to trigger notifications only for specific subsets of objects.9 This is a crucial step for minimizing the volume of events and reducing unnecessary processing and associated costs.
  + **Enable Auditing:** Enable notifications for object deletion events (s3:ObjectRemoved:\*) to create a traceable audit log, which is a crucial compliance requirement for financial data.9 This provides visibility into data life cycles and potential security incidents.
* Patterns and Considerations:  
  The rare scenario of a missed event, while uncommon, must be accounted for in a regulated environment where data integrity is paramount. The solution for this is not a simple configuration setting but a separate, scheduled reconciliation process. An effective operational strategy involves a daily or hourly audit that compares the inventory of objects in the S3 bucket against the application's event processing log. Any discrepancy indicates a missed event that must be reprocessed. This creates a multi-layered defense against data loss, moving the system from a best-effort event model to an auditable, fault-tolerant one. For all financial data, server-side encryption with AWS KMS should be enabled by default to protect data at rest.12

#### 2. Amazon Simple Queue Service (SQS)

Amazon SQS is a fully managed message queuing service that decouples producing and consuming components.9 For financial workloads, SQS provides a robust mechanism to handle asynchronous processing, manage high-volume message traffic, and ensure message durability.

* **Best Practices:**
  + **Choose the Right Queue Type:** The selection of the queue type is a critical architectural decision. For high-volume, general-purpose tasks where message order is not critical (e.g., asynchronous task processing), standard queues are appropriate. For mission-critical workflows that require "exactly-once" processing and strict message order (e.g., payment processing or fund transfers), a FIFO (First-In-First-Out) queue is a fundamental requirement.15
  + **Configure a Dead-Letter Queue (DLQ):** Every critical SQS queue should be configured with a dead-letter queue to handle problematic messages.15 This prevents poison-pill messages from indefinitely blocking the main queue and provides a dedicated location for manual inspection and reprocessing of failed messages. The DLQ is not just for technical failures; it is a repository for business failures, such as a message that fails a business rule, enabling human operators to review and take action.16
  + **Set an Appropriate Visibility Timeout:** The visibility timeout is the duration during which a message is hidden from other consumers after it has been received.15 It must be configured to be long enough for the consumer to process the message successfully but short enough to allow for a quick retry in case of a consumer crash or timeout.
  + **Message Deduplication:** When using FIFO queues, messages can be deduplicated either by providing a MessageDeduplicationId or by enabling content-based deduplication, which uses a SHA-256 hash of the message body.10 This provides an additional layer of protection against duplicate processing.
* Patterns and Considerations:  
  To manage dead-letter queues effectively, it is recommended to implement a regular schedule for monitoring and reviewing DLQ messages.16 Automated monitoring with CloudWatch alarms can be used to alert on an increase in DLQ messages, which may indicate a persistent processing issue.16 A separate Lambda function can also be configured to automatically process DLQ messages, either by retrying them with an exponential backoff or archiving them for further analysis.16 This ties operational excellence directly to business continuity and enables a self-healing architecture.

#### 3. Amazon Simple Notification Service (SNS)

Amazon SNS is a highly scalable, fully managed publisher/subscriber service that facilitates the fan-out pattern in an event-driven architecture.11 It enables the broadcasting of a single message to multiple subscribers, such as SQS queues, Lambda functions, or HTTP endpoints.14

* **Best Practices:**
  + **SNS + SQS for Durability:** For mission-critical workflows where durability is required, always combine SNS with SQS.11 The SNS topic broadcasts the message, and an SQS queue subscribed to the topic ensures that the message is durably stored until a consumer can successfully process it. This prevents data loss if a consumer is offline or experiences a failure, as SNS has a limited retry policy for non-SQS subscribers.11
  + **Implement Least Privilege Access:** Adhere to the principle of least privilege by granting only the permissions required to perform a specific task.13 For applications and other AWS services, use IAM roles to manage temporary credentials instead of storing long-term credentials directly in the application or EC2 instance.13
  + **Server-Side Encryption:** Enable server-side encryption (SSE) with AWS KMS to protect messages at rest.13 This is a critical security measure for a financial services client, protecting sensitive data.
  + **Regulatory Compliance:** When using SNS for SMS messaging, developers must be aware of and comply with applicable laws, regulations, and carrier requirements in each country where customers reside, such as the Telephone Consumer Protection Act (TCPA) in the United States.17
* Patterns and Considerations:  
  SNS is primarily a notification service, making it an ideal choice for real-time, push-based communication, such as triggering notifications from CloudWatch Alarms.14 This is a simple yet powerful pattern for building a robust observability strategy. For more complex event routing needs, the use of EventBridge is often preferred.11

#### 4. AWS Lambda

AWS Lambda is a serverless, event-driven compute service that enables developers to run code without provisioning or managing servers.18 For financial services clients, Lambda is ideal for building agile, microservice-based applications, but careful optimization is required to balance performance and cost.

* **Best Practices:**
  + **Optimize Memory Allocation:** Memory and CPU are a single configurable dimension in Lambda, with CPU power scaling proportionally with memory allocation.19 A common fallacy is that using less memory is always cheaper. However, if a lower memory allocation increases execution time and pushes the total duration into a higher billing increment, it can paradoxically be more expensive.19 It is recommended to use the open-source  
    aws-lambda-power-tuning tool to find the optimal balance between performance and cost for each function.19
  + **Mitigate Cold Starts:** For latency-sensitive APIs or workflows, a cold start can significantly impact performance.19 For these applications, use Provisioned Concurrency to keep execution environments warm and ready for immediate invocation.19 For non-critical workloads, choosing a lightweight, efficient runtime like Node.js or Python over Java or C# can help reduce cold start latency.19
  + **Implement Caching:** Cache external resources, such as database connections and API responses, in global variables or by using external services like DynamoDB Accelerator (DAX).18 This reduces latency and costs by avoiding repeated calls to external dependencies.
  + **Stateless and Ephemeral:** Design functions to be stateless, meaning they do not rely on the local file system or in-memory state between invocations.18 This ensures functions can scale efficiently and handle failures gracefully. Use external state management solutions like DynamoDB or S3 to persist data across invocations.18
* Patterns and Considerations:  
  For decoupled workflows, utilize event-driven services like SQS, SNS, or EventBridge to trigger Lambda functions asynchronously.18 When using SQS or Kinesis, configure appropriate batch sizes to balance throughput with cost and memory.18 It is also a best practice to configure a dead-letter queue for asynchronous invocations to capture and process failed messages.18

#### 5. Amazon DynamoDB

Amazon DynamoDB is a fully managed, key-value and document database that delivers single-digit millisecond performance at any scale.21 For a financial services client, DynamoDB is an ideal choice for high-throughput workloads that require low-latency access, such as real-time trading dashboards or customer account information.

* **Best Practices:**
  + **Design a High-Cardinality Partition Key:** A poorly designed partition key can lead to "hot partitions," where one partition handles a disproportionate amount of traffic and causes throttling.21 It is critical to choose a partition key with a high number of unique values to ensure that traffic is evenly distributed across all partitions.21
  + **Implement Write Sharding for High-Volume Keys:** For a partition key that receives a disproportionate amount of write traffic, a solution is to append a random number or another unique identifier to the partition key (e.g., ProductID#1, ProductID#2) to split the writes across multiple logical partitions.21
  + **Use On-Demand Mode for Unpredictable Workloads:** For spiky or unpredictable traffic patterns common in financial applications, use on-demand capacity mode, which automatically scales throughput and prevents throttling without requiring manual capacity planning.21
  + **Optimize Index Usage:** Limit the number of Global Secondary Indexes (GSIs), as each GSI requires additional write capacity and increases costs.21 GSIs should be carefully designed to align with application query patterns.
* Patterns and Considerations:  
  For applications with a high read-to-write ratio, such as dashboards that display real-time portfolio data, use DynamoDB Accelerator (DAX) to provide an in-memory cache.21 DAX significantly reduces read latency to microseconds and offloads read traffic from the underlying table. DynamoDB also provides a powerful idempotency mechanism through its composite primary key. By using a partition key and a sort key, the system can ensure that no two items with the same key can be written, providing a last-resort idempotency guarantee at the data layer.10

#### 6. Amazon CloudWatch Logs

Amazon CloudWatch Logs is a fully managed logging service that centralizes logs from various AWS services and on-premises systems.22 For a financial services client, a centralized logging strategy is not just a technical convenience; it is a critical component of the security and governance pillars, providing a single, auditable source of truth for compliance and forensic analysis.

* **Best Practices:**
  + **Centralized Logging Strategy:** For a multi-account AWS environment, use a multi-account strategy with AWS Organizations to consolidate logs from all accounts and regions into a central logging account.22 This ensures a complete and untampered record of all user and system activity, a non-negotiable for a regulated industry.
  + **Strategic Log Retention:** Vary log retention periods to balance cost and compliance requirements.22 For example, security audit logs can be retained for a year or more to meet regulatory requirements, while development and debug logs can be retained for a shorter period, such as 3-7 days.
  + **Secure Log Data:** Implement strict IAM policies to control who can access log data. Enable encryption for log groups using AWS KMS keys to protect sensitive information.22
* Patterns and Considerations:  
  CloudWatch Logs can be used for real-time processing by creating subscription filters that forward logs to other services, such as a Lambda function for real-time analysis or a Kinesis Firehose delivery stream for archiving.22 This enables automated responses to specific log patterns, such as an application error, by triggering a Lambda function or an EventBridge rule.22

#### 7. Amazon CloudWatch Alarms

Amazon CloudWatch Alarms enable developers to monitor metrics and send notifications or trigger automated actions when a specified threshold is breached.24 A well-designed alerting strategy is essential for operational excellence, as it enables a team to respond to what matters and avoid alert fatigue.

* **Best Practices:**
  + **Alert on Actionable Events:** The goal is to avoid alert fatigue, which is an anti-pattern that can lead to missed critical alerts.24 Create alarms only for events that require an immediate human or automated response.25
  + **Use High-Signal Thresholds:** Set thresholds based on historical data and expected performance. Use filters and duration-based rules (e.g., CPU utilization > 80% for 5 minutes) to suppress non-critical spikes and false positives.24
  + **Provide Context:** Alarms should not just notify but also provide clear context, including the affected service, the metric, and possible troubleshooting steps.24 This empowers a team to take immediate action without having to perform additional investigation.
* Patterns and Considerations:  
  A powerful pattern is to tie alarms to automated actions (e.g., using SNS to trigger a Lambda function) to enable self-healing architectures.24 Developers should also set up alarms for critical metrics, such as throttled requests on DynamoDB or a rising message count on a dead-letter queue.16 For a financial services client, billing alarms are also a best practice to monitor and prevent unexpected cost spikes.24

#### 8. AWS Step Functions

AWS Step Functions is a serverless orchestration service that enables developers to build and run complex workflows as a series of steps.26 It abstracts away the complexity of managing state, retries, and error handling, allowing developers to focus on the business logic.7

* **Best Practices:**
  + **Use Standard Workflows for Long-Running Tasks:** Standard Workflows are ideal for long-running, auditable workflows (up to a year) that require exactly-once processing and support integration patterns like Run a Job (.sync) and Wait for Callback (.waitForTaskToken).26
  + **Use Express Workflows for High-Volume Tasks:** Express Workflows are designed for high-volume, short-duration tasks (up to five minutes) that have an at-least-once execution model.26
  + **Implement Built-in Error Handling:** Use Retry and Catch blocks to automatically handle transient failures and route to alternative steps.26 This eliminates the need to write complex custom error-handling code and ensures the workflow can recover gracefully from failures.
* Patterns and Considerations:  
  The Wait for Callback pattern is particularly powerful for financial workflows that require a human-in-the-loop approval step.26 For example, a workflow can pause and wait for a manager's sign-off on a large transaction before proceeding. Another critical pattern is to ensure that all underlying functions and service calls within a Step Functions workflow are idempotent, as the service will automatically handle retries.7 The ability to model complex business logic in a visual, declarative way is a massive benefit for a regulated industry that needs to demonstrate process integrity and audibility.26

#### 9. Amazon EventBridge

Amazon EventBridge is a serverless event bus service that enables a centralized, enterprise-scale event-driven architecture.5 It allows developers to create a central hub for all business events, routing them from different producers to multiple consumers.

* **Best Practices:**
  + **Centralized Event Bus:** Use EventBridge as the central event bus for your entire organization, routing events from different producers to multiple consumers.5 This is ideal for a multi-account, multi-team environment common in a large financial institution.
  + **Advanced Content-Based Routing:** Leverage EventBridge's advanced filtering rules to route events based on the event's content, allowing consumers to subscribe only to the specific events they care about without the producer's knowledge.5
  + **Integrate with SaaS:** Use EventBridge to easily integrate with third-party SaaS applications, which is increasingly common for financial firms that use a blend of internal and external services.11
* Patterns and Considerations:  
  The choice between EventBridge and SNS reflects a core architectural decision.11 EventBridge enables a true decoupled, agile, and enterprise-scale architecture, whereas SNS is more suited for simple pub/sub notifications to a known set of subscribers.11 For a large, modern asset management firm, EventBridge is the strategic choice for core business events due to its superior routing capabilities.

| Feature | SNS | SQS | EventBridge |
| --- | --- | --- | --- |
| **Primary Purpose** | Real-time notifications | Message buffering | Content-based event routing |
| **Communication Pattern** | One-to-many (fanout) | Point-to-point | Many-to-many with filtering |
| **Delivery Model** | Push | Pull | Push |
| **Routing Capability** | Limited | None | Advanced |
| **Durability** | Limited (requires SQS for durability) | High | High |
| **Use Cases** | Real-time notifications, fan-out | Task queues, async processing | Event-driven workflows, SaaS integration |

#### 10. Amazon Elastic Compute Cloud (EC2) & Amazon Elastic Container Service (ECS)

Amazon EC2 and ECS provide the compute foundation for cloud-native applications.29 The choice between them—and between the two ECS launch types, EC2 and Fargate—is a nuanced one that balances control, performance, and operational simplicity.29

* **Best Practices:**
  + **Secure EC2 with Hardening:** EC2 instances are the client's responsibility and require a full-stack security approach.31 Restrict incoming traffic with security groups, use Session Manager instead of SSH for remote access, and enable Amazon Inspector to continuously scan for vulnerabilities.32
  + **Leverage Fargate for Simplicity and Agility:** Use AWS Fargate (the serverless compute engine for ECS) for modern, containerized microservices and spiky workloads.29 Fargate abstracts away the operational burden of managing the underlying EC2 instances, providing built-in resilience and high availability.30
  + **Use EC2 for Control and Specialized Workloads:** Use ECS on EC2 for workloads that require a high degree of control over the underlying infrastructure, such as specific instance types for high-performance computing (HPC) or legacy applications that require low-level access.29
* Patterns and Considerations:  
  A modern financial firm will likely use a hybrid approach, leveraging Fargate for new, agile, microservice-based applications and retaining EC2 for legacy or specialized workloads.29 For cost optimization, Reserved Instances or Savings Plans can be leveraged for predictable, long-running EC2 workloads, while Fargate is cost-efficient for spiky or short-lived workloads due to its pay-per-task model.29

| Feature / Metric | ECS on EC2 | ECS with Fargate |
| --- | --- | --- |
| **Compute Performance** | Near-native performance, slight overhead | Predictable, some overhead from task scheduling |
| **Latency** | Low latency, but not as optimized as bare EC2 | Slightly higher latency due to additional abstraction |
| **Scalability** | Service Auto Scaling, cluster-based | Fully managed scaling per container task |
| **Resource Utilization** | Good utilization with container density | High efficiency—pay only for requested CPU and memory |
| **Control and Customization** | Moderate—depends on EC2 instances chosen | Minimal—abstracted, limited customization |
| **Best Use Cases** | Microservices, hybrid workloads | Event-driven apps, serverless APIs, spiky workloads |

### IV. Conclusion: A Holistic Architectural View

The journey to building resilient, compliant, and performant cloud-native applications for financial services involves more than just selecting the right AWS services; it requires a holistic architectural mindset. The best practices outlined in this report are not a series of isolated recommendations but are interconnected components of a cohesive system.

A new trade file, for example, can be uploaded to an S3 bucket, triggering an S3 Event Notification. This event is routed to a durable SQS queue, which acts as a buffer. A Lambda function, configured with an appropriate memory setting and concurrency, consumes the message, processes the trade data, and updates a DynamoDB table. The DynamoDB update can trigger a stream event, which is then processed by a Step Functions workflow. The workflow can orchestrate complex, multi-step processes, such as a human-in-the-loop approval for a large transaction. At each step, events can be published to a central EventBridge bus for advanced, content-based routing, which might trigger a separate Lambda function for compliance checks or an SNS topic for real-time notifications to a trading dashboard. All system activity is logged to CloudWatch Logs for auditing, and CloudWatch Alarms are configured to alert on actionable events, such as a high volume of throttled requests on DynamoDB or a rising message count on a dead-letter queue.

For developers at an asset management firm, the key takeaways are to:

* **Design for Idempotency:** This is the single most important architectural principle for a financial system to ensure data integrity and fault tolerance.
* **Embrace the Event-Driven Model:** Use managed services like SQS, SNS, and EventBridge to build decoupled, agile, and scalable systems.
* **Offload Operational Burden:** Leverage serverless and managed services, such as Lambda, Fargate, and DynamoDB, to reduce operational overhead and focus on delivering business value.
* **Adhere to a Security-First Mindset:** Assume a shared responsibility for security, implement the principle of least privilege, and ensure all data is protected both at rest and in transit.

The path forward involves continuous improvement. Developers are encouraged to use the AWS Well-Architected Tool to regularly evaluate their workloads against best practices and identify areas for improvement.2 Engaging with AWS partners and leveraging resources from the AWS Solutions Library and Architecture Center can further accelerate the modernization of legacy systems and the creation of new, innovative financial applications on the cloud.4

#### Works cited

1. Financial Services Industry Lens - AWS Well-Architected Framework ..., accessed September 25, 2025, <https://docs.aws.amazon.com/wellarchitected/latest/financial-services-industry-lens/financial-services-industry-lens.html>
2. AWS Well-Architected - Build secure, efficient cloud applications, accessed September 25, 2025, <https://aws.amazon.com/architecture/well-architected/>
3. United States - AWS, accessed September 25, 2025, <https://aws.amazon.com/financial-services/security-compliance/compliance-center/us/>
4. Deloitte Financial Services on AWS, accessed September 25, 2025, <https://www.deloitte.com/global/en/alliances/aws/about/aws-relationship-financial-services-solutions.html>
5. Event-Driven Architecture - AWS, accessed September 25, 2025, <https://aws.amazon.com/event-driven-architecture/>
6. Best practices for implementing event-driven architectures in your ..., accessed September 25, 2025, <https://aws.amazon.com/blogs/architecture/best-practices-for-implementing-event-driven-architectures-in-your-organization/>
7. AWS StepFunction and the Importance of Idempotent Functions | by Nadir Saghar, accessed September 25, 2025, <https://aws.plainenglish.io/aws-stepfunction-and-the-importance-of-idempotent-functions-a14502d1d502>
8. What Is Idempotency? Why It Matters for Durable Systems - Temporal, accessed September 25, 2025, <https://temporal.io/blog/idempotency-and-durable-execution>
9. Real-Time Event Handling with Amazon S3 Notifications - Exam-Labs, accessed September 25, 2025, <https://www.exam-labs.com/blog/real-time-event-handling-with-amazon-s3-notifications>
10. Achieving idempotency in the AWS serverless space | by Qasim Albaqali - Medium, accessed September 25, 2025, <https://qasimalbaqali.medium.com/achieving-idempotency-in-the-aws-serverless-space-d0671a521479>
11. Event-Driven Design: Choosing Between SNS, SQS, and ..., accessed September 25, 2025, <https://dev.to/aws-builders/event-driven-design-choosing-between-sns-sqs-and-eventbridge-i82>
12. Rule: S3 Buckets Should Have Event Notifications Enabled - CloudDefense.AI, accessed September 25, 2025, <https://www.clouddefense.ai/compliance-rules/aws-fs-practices/s3/foundational-security-s3-11>
13. Amazon SNS security best practices - Amazon Simple Notification Service, accessed September 25, 2025, <https://docs.aws.amazon.com/sns/latest/dg/sns-security-best-practices.html>
14. Choosing between EventBridge, SNS, and SQS for event-driven patterns, accessed September 25, 2025, <https://arpadt.com/articles/eb-sns-sqs>
15. Amazon SQS best practices - Amazon Simple Queue Service, accessed September 25, 2025, <https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/sqs-best-practices.html>
16. How to Process Dead Letter Queue Messages in AWS - Pilotcore, accessed September 25, 2025, <https://pilotcore.io/blog/how-to-process-dead-letter-queue-messages-in-aws>
17. Best practices for Amazon SNS SMS messaging - Amazon Simple Notification Service, accessed September 25, 2025, <https://docs.aws.amazon.com/sns/latest/dg/channels-sms-best-practices.html>
18. Top 10 AWS Lambda Best Practices - Lumigo, accessed September 25, 2025, <https://lumigo.io/learn/top-10-aws-lambda-best-practices/>
19. AWS Lambda Performance Optimization: 12 Tips and Advanced ..., accessed September 25, 2025, <https://lumigo.io/aws-lambda-performance-optimization/>
20. How To Accelerate AWS Lambda Performance - IEEE Computer Society, accessed September 25, 2025, <https://www.computer.org/publications/tech-news/trends/accelerate-aws-lambda-performance/>
21. Optimizing DynamoDB for High Throughput Workloads - DEV ..., accessed September 25, 2025, <https://dev.to/imsushant12/optimizing-dynamodb-for-high-throughput-workloads-13gf>
22. AWS Centralized Logging: A Complete Implementation Guide | Last9, accessed September 25, 2025, <https://last9.io/blog/aws-centralized-logging/>
23. Cross-account cross-Region log centralization - Amazon CloudWatch Logs, accessed September 25, 2025, <https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/CloudWatchLogs_Centralization.html>
24. Guide for CloudWatch Alerting: Best Practices and Implementation, accessed September 25, 2025, <https://drdroid.io/engineering-tools/guide-for-cloudwatch-alerting-best-practices-and-implementation>
25. Alarms | AWS Observability Best Practices - GitHub Pages, accessed September 25, 2025, <https://aws-observability.github.io/observability-best-practices/signals/alarms/>
26. AWS Step Functions | By Joud W. Awad | Medium, accessed September 25, 2025, <https://medium.com/@joudwawad/aws-step-functions-deep-dive-f66ea367df6a>
27. Best practices for Step Functions - AWS Step Functions, accessed September 25, 2025, <https://docs.aws.amazon.com/step-functions/latest/dg/sfn-best-practices.html>
28. Handling error conditions in a Step Functions state machine - AWS Documentation, accessed September 25, 2025, <https://docs.aws.amazon.com/step-functions/latest/dg/tutorial-handling-error-conditions.html>
29. AWS EC2 Vs ECS Vs ECS Fargate: Comparing Performance And Costs - StoneFly, Inc., accessed September 25, 2025, <https://stonefly.com/blog/aws-ec2-vs-ecs-fargate-performance-cost-analysis/>
30. AWS Fargate vs. Amazon EC2: Launch Options for AWS ECS, accessed September 25, 2025, <https://www.stormit.cloud/blog/aws-fargate-vs-ec2/>
31. Hardening guide on EC2 | AWS re:Post, accessed September 25, 2025, <https://repost.aws/it/questions/QUH_hHeF9pS7m2ufP1-JwRnQ/hardening-guide-on-ec2>
32. Security control recommendations for protecting infrastructure - AWS ..., accessed September 25, 2025, <https://docs.aws.amazon.com/prescriptive-guidance/latest/security-controls-by-caf-capability/infrastructure-controls.html>
33. A deep dive into resilience and availability on Amazon Elastic Container Service - AWS, accessed September 25, 2025, <https://aws.amazon.com/blogs/containers/a-deep-dive-into-resilience-and-availability-on-amazon-elastic-container-service/>