Security Overview of AWS Lambda

An In-Depth Look at AWS Lambda Security

January 2021

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 AWS's current 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's products or services are provided "as is" without warranties, representations, or conditions of any kind, whether express or implied. AWS's responsibilities and liabilities 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.

Contents

Abstract

This whitepaper presents a deep dive into the AWS Lambda service through a security lens. It provides a well-rounded picture of the service, which is useful for new adopters, and deepens understanding of Lambda for current users.

The intended audience for this whitepaper is Chief Information Security Officers (CISOs), information security groups, security engineers, enterprise architects, compliance teams, and any others interested in understanding the underpinnings of AWS Lambda.

Introduction

Today, more workloads use AWS Lambda to achieve scalability, performance, and cost efficiency, without managing the underlying computing. These workloads scale to thousands of concurrent requests per second. Lambda is used by hundreds of thousands of Amazon Web Services (AWS) customers to serve trillions of requests every month.

Lambda is suitable for mission critical applications in many industries. A broad variety of customers, from media and entertainment to financial services and other regulated industries, take advantage of Lambda. These customers decrease time to market, optimize costs, and improve agility by focusing on what they do best: running their business.

The *managed runtime environment* model enables Lambda to manage much of the implementation details of running serverless workloads. This model further reduces the attack surface while making cloud security simpler. This whitepaper presents the underpinnings of that model, along with best practices, to developers, security analysts, security and compliance teams, and other stakeholders.

About AWS Lambda

Lambda is an event-driven, serverless compute service that extends other AWS services with custom logic, or creates backend services that operate with scale, performance, and security in mind. Lambda can be configured to automatically run code in response to multiple events, such as HTTP requests through Amazon API Gateway, modifications to objects in Amazon Simple Storage Service (Amazon S3) buckets, table updates in Amazon DynamoDB, and state transitions in AWS Step Functions. Lambda runs code on a highly available compute infrastructure and performs all the administration of the underlying platform, including server and operating system maintenance, capacity provisioning and automatic scaling, patching, code monitoring, and logging.

With Lambda, you can just upload your code and configure when to invoke it; Lambda takes care of everything else required to run your code. Lambda integrates with many other AWS services, and enables you to create serverless applications or backend services, ranging from periodically triggered, simple automation tasks to full-fledged microservices applications.

Lambda can be configured to access resources within your Amazon Virtual Private Cloud (Amazon VPC), and by extension, your on-premises resources.

Lambda integrates with AWS Identity and Access Management (IAM), which you can leverage to protect your data and configure fine-grained access controls using a variety

of access management strategies, while maintaining a high level of security and auditing to help you meet your compliance needs.

Benefits of Lambda

Customers who want to unleash the creativity and speed of their development organizations without compromising their IT team's ability to provide a scalable, cost-effective, and manageable infrastructure, find that Lambda lets them trade operational complexity for agility and better pricing, without compromising on scale or reliability.

Lambda offers many benefits, including the following:

No Servers to Manage

Lambda runs your code on highly available, fault-tolerant infrastructure spread across multiple Availability Zones (AZs) in a single Region, seamlessly deploying code, and providing all the administration, maintenance, and patches of the infrastructure. Lambda also provides built-in logging and monitoring, including integration with Amazon CloudWatch, CloudWatch Logs, and AWS CloudTrail.

Continuous Scaling

Lambda precisely manages scaling of your functions (or application) by running event-triggered code in parallel, and processing each event individually.

Millisecond Metering

With Lambda, you are charged for every 1 millisecond (ms) your code executes, and the number of times your code is triggered. You pay for consistent throughput or execution duration, instead of by server unit.

Increases Innovation

Lambda frees up your programming resources by taking over the infrastructure management, allowing you to focus on innovation and development of business logic.

Modernize your Applications

Lambda enables you to use functions with pre-trained machine learning models to inject artificial intelligence into applications easily. A single application programming interface (API) request can classify images, analyze videos, convert speech to text, perform natural language processing, and more.

Rich Ecosystem

Lambda supports developers through AWS Serverless Application Repository for discovering, deploying and publishing serverless applications, AWS Serverless Application Model for building serverless applications and integrations with various integrated development environments (IDEs) like AWS Cloud9, AWS Toolkit for Visual Studio, AWS Tools for Visual Studio Team Services, and several others. Lambda is integrated with additional AWS services to provide you a rich ecosystem for building serverless applications.

Cost for Running Lambda-Based Applications

Lambda offers a granular, pay-as-you-go pricing model. With this model, you are charged based on the number of function invocations and their duration (the time it takes for the code to run). In addition to this flexible pricing model, Lambda also offers 1 million perpetually free requests per month, which enables many customers to automate their process without any costs.

The Shared Responsibility Model

At AWS, security and compliance is a shared responsibility between AWS and the customer. This shared responsibility model can help relieve your operational burden, as AWS operates, manages, and controls the components from the host operating system and virtualization layer, down to the physical security of the facilities in which the service operates.

For Lambda, AWS manages the underlying infrastructure and application platform, the operating system, and the execution environment. You are responsible for the security of your code and identity and access management (IAM) to the Lambda service and within your function.

Figure 1 shows the shared responsibility model as it applies to the common and distinct components of Lambda. AWS responsibilities appear in orange, and customer responsibilities appear in blue.

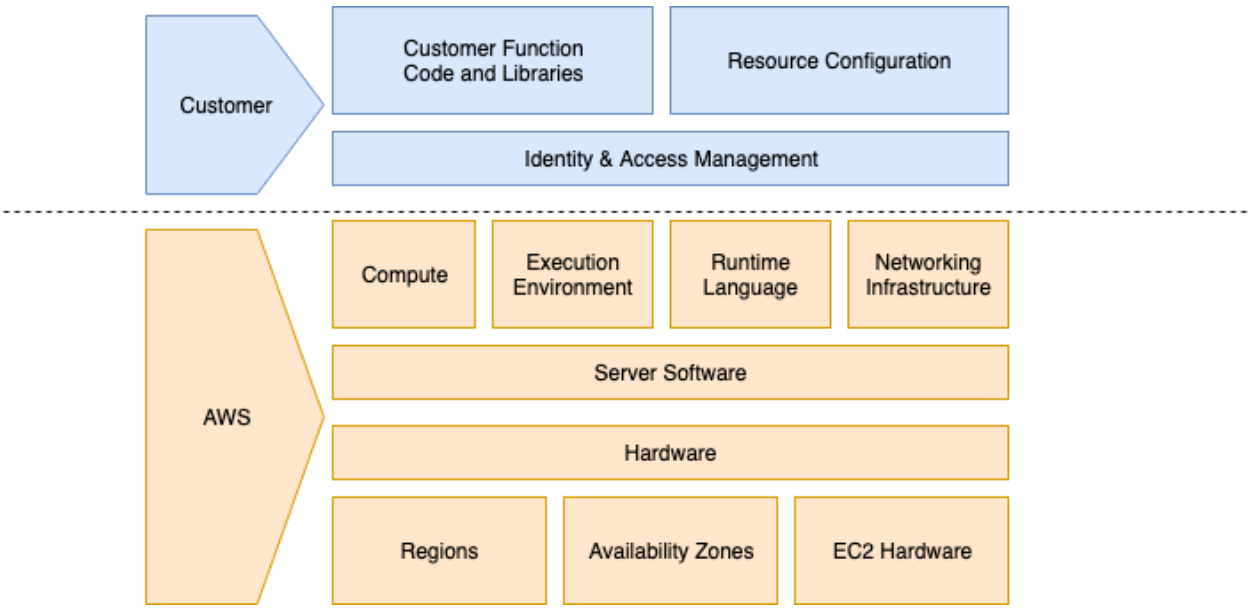

Figure 1 – Shared Responsibility Model for AWS Lambda

Lambda Functions and Layers

With Lambda, you can run code virtually with zero administration of the underlying infrastructure. You are responsible only for the code that you provide Lambda, and the configuration of how Lambda runs that code on your behalf. Today, Lambda supports two types of code resources: *Functions* and *Layers*.

A *function* is a resource which can be invoked to run your code in Lambda. Functions can include a common, or shared, resource called *Layers*. Layers can be used to share common code or data across different functions or AWS accounts. You are responsible for the management of all the code contained within your functions or layers. When Lambda receives the function or layer code from a customer, Lambda protects access to it by encrypting it at-rest using AWS Key Management Service (AWS KMS) and in-transit by using TLS 1.2+.

You can manage access to your functions and layers through AWS IAM policies, or through resource-based permissions. For a full list of supported IAM features on Lambda, see AWS Services that work with IAM.

You can also control the entire lifecycle of your functions and layers through Lambda's control plane APIs. For example, you can choose to delete your function by calling `DeleteFunction`, or revoke permissions from another account by calling `RemovePermission`.

Lambda Invoke Modes

The Invoke API can be called in two modes: *event* mode and *request-response* mode.

- *Event* mode queues the payload for an asynchronous invocation.
- *Request-response* mode synchronously invokes the function with the provided payload and returns a response immediately.

In both cases, the function execution is always performed in a Lambda execution environment, but the payload takes different paths. For more information, see Lambda Execution Environments in this document.

You can also use other AWS services that perform invocations on your behalf. Which invoke mode is used depends on which AWS service you are using, and how it is configured. For additional information on how other AWS services integrate with Lambda, see Using AWS Lambda with other services.

When Lambda receives a request-response invoke, it is passed to the invoke service directly. If the invoke service is unavailable, callers may temporarily queue the payload client-side to retry the invocation a set number of times. If the invoke service receives the payload, the service then attempts to identify an available execution environment for the request, and passes the payload to that execution environment to complete the invocation. If no existing or appropriate execution environments exist, one will be dynamically created in response to the request. While in-transit, invoke payloads sent to the invoke service are secured with TLS 1.2+. Traffic within the Lambda service (from the load balancer down) passes through an isolated internal virtual private cloud (VPC), owned by the Lambda service, within the AWS Region to which the request was sent.

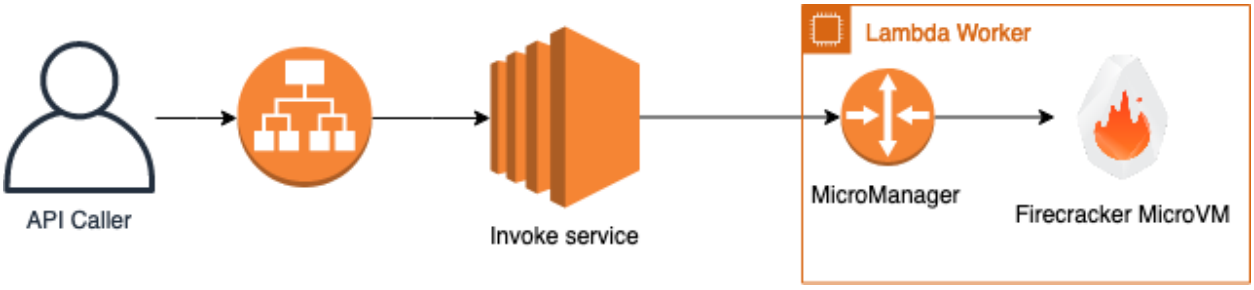

Figure 2 – Invocation model for AWS Lambda: request-response

Event invocation mode payloads are always queued for processing before invocation. All payloads are queued for processing in an Amazon Simple Queue Service (Amazon SQS) queue. Queued events are always secured in-transit with TLS 1.2+, but they are not currently encrypted at-rest. The Amazon SQS queues used by Lambda are managed by the Lambda service, and are not visible to you as a customer. Queued events can be stored in a shared queue, but may be migrated or assigned to dedicated queues depending on a number of factors that cannot be directly controlled by customers (for example, rate of invoke, size of events, and so on).

Queued events are retrieved in batches by Lambda's *poller fleet*. The poller fleet is a group of EC2 instances whose purpose is to process queued event invocations which have not yet been processed. When the poller fleet retrieves a queued event that it needs to process, it does so by passing it to the invoke service just like a customer would in a request-response mode invoke.

If the invocation cannot be performed, the poller fleet will temporarily store the event, in-memory, on the host until it is either able to successfully complete the execution, or until the number of run retry attempts have been exceeded. No payload data is ever written to disk on the poller fleet itself. The polling fleet can be tasked across AWS customers, allowing for the shortest time to invocation. For more information about which services may take the event invocation mode, see Using AWS Lambda with other services.

Lambda Executions

When Lambda executes a function on your behalf, it manages both provisioning and configuring the underlying systems necessary to run your code. This enables your developers to focus on business logic and writing code, not administering and managing underlying systems.

The Lambda service is split into the *control plane* and the *data plane*. Each plane serves a distinct purpose in the service. The control plane provides the management APIs (for example, `CreateFunction`, `UpdateFunctionCode`, `PublishLayerVersion`, and so on), and manages integrations with all AWS services. Communications to Lambda's control plane are protected in-transit by TLS. All customer data stored within Lambda's control plane is encrypted at-rest through the use of AWS KMS, which is designed to protect it from unauthorized disclosure or tampering.

The data plane is Lambda's `Invoke` API that triggers the invocation of Lambda functions. When a Lambda function is invoked, the data plane allocates an execution environment on an AWS Lambda Worker (or simply Worker, a type of Amazon EC2 instance) to that function version, or chooses an existing execution environment that has already been set up for that function version, which it then uses to complete the invocation. For more information, see the AWS Lambda MicroVMs and Workers section of this document.

Lambda Execution Environments

Each invocation is routed by Lambda's invoke service to an execution environment on a Worker that is able to service the request. Other than through data plane, customers and other users cannot directly initiate inbound/ingress network communications with an execution environment. This helps to ensure that communications to your execution environment are authenticated and authorized.

Execution environments are reserved for a specific function version and cannot be reused across function versions, functions, or AWS accounts. This means a single function which may have two different versions would result in at least two unique execution environments.

Each execution environment may only be used for one concurrent invocation at a time, and they may be reused across multiple invocations of the same function version for performance reasons. Depending on a number of factors (for example, rate of invocation, function configuration, and so on), one or more execution environments may exist for a given function version. With this approach, Lambda is able to provide function version level isolation for its customers.

Lambda does not currently isolate invokes within a function version's execution environment. What this means is that one invoke may leave a state that may affect the next invoke (for example, files written to `/tmp` or data in-memory). If you want to ensure that one invoke cannot affect another invoke, Lambda recommends that you create additional distinct functions. For example, you could create distinct functions for complex parsing operations which are more error prone, and re-use functions which do not perform security sensitive operations. Lambda does not currently limit the number of functions that customers can create. For more information about limits, see the Lambda quotas page.

Execution environments are continuously monitored and managed by Lambda, and they may be created or destroyed for any number of reasons including, but not limited to:

- A new invoke arrives and no suitable execution environment exists
- An internal runtime or Worker software deployment occurs
- A new provisioned concurrency configuration is published
- The lease time on the execution environment, or the Worker, is approaching or has exceeded max lifetime
- Other internal workload rebalancing processes

Customers can manage the number of pre-provisioned execution environments that exist for a function version by configuring provisioned concurrency on their function configuration. When configured to do so, Lambda will create, manage and ensure the configured number of execution environments always exist. This ensures that customers have greater control over start-up performance of their serverless applications at any scale.

Other than through a provisioned concurrency configuration, customers cannot deterministically control the number of execution environments that are created or managed by Lambda in response to invocations.

Execution Role

Each Lambda function must also be configured with an execution role, which is an IAM role that is assumed by the Lambda service when performing control plane and data plane operations related to the function. The Lambda service assumes this role to fetch temporary security credentials which are then available as environment variables during a function's invocation. For performance reasons, the Lambda service will cache these credentials, and may re-use them across different execution environments which use the same execution role.

To ensure adherence to least privilege principle, Lambda recommends that each function has its own unique role, and that it is configured with the minimum set of permissions it requires.

The Lambda service may also assume the execution role to perform certain control plane operations such as those related to creating and configuring Elastic network interfaces (ENI) for VPC functions, sending logs to Amazon CloudWatch, sending traces to AWS X-Ray, or other non-invoke related operations. Customers can always review and audit these use cases by reviewing audit logs in AWS CloudTrail.

For more information on this subject, see the AWS Lambda execution role documentation page.

Lambda MicroVMs and Workers

Lambda will create its execution environments on a fleet of EC2 instances called *AWS Lambda Workers*. Workers are bare metal EC2 Nitro instances which are launched and managed by Lambda in a separate isolated AWS account which is not visible to customers. Workers have one or more hardware-virtualized Micro Virtual Machines (MVM) created by Firecracker. Firecracker is an open-source Virtual Machine Monitor (VMM) that uses Linux's Kernel-based Virtual Machine (KVM) to create and manage MVMs. It is purpose-built for creating and managing secure, multi-tenant container and function-based services that provide serverless operational models. For more information about Firecracker's security model, see the Firecracker project website.

As a part of the shared responsibility model, Lambda is responsible for maintaining the security configuration, controls, and patching level of the Workers. The Lambda team uses AWS Inspector to discover known potential security issues, as well as other custom security issue notification mechanisms and pre-disclosure lists, so that customers don't need to manage the underlying security posture of their execution environment.

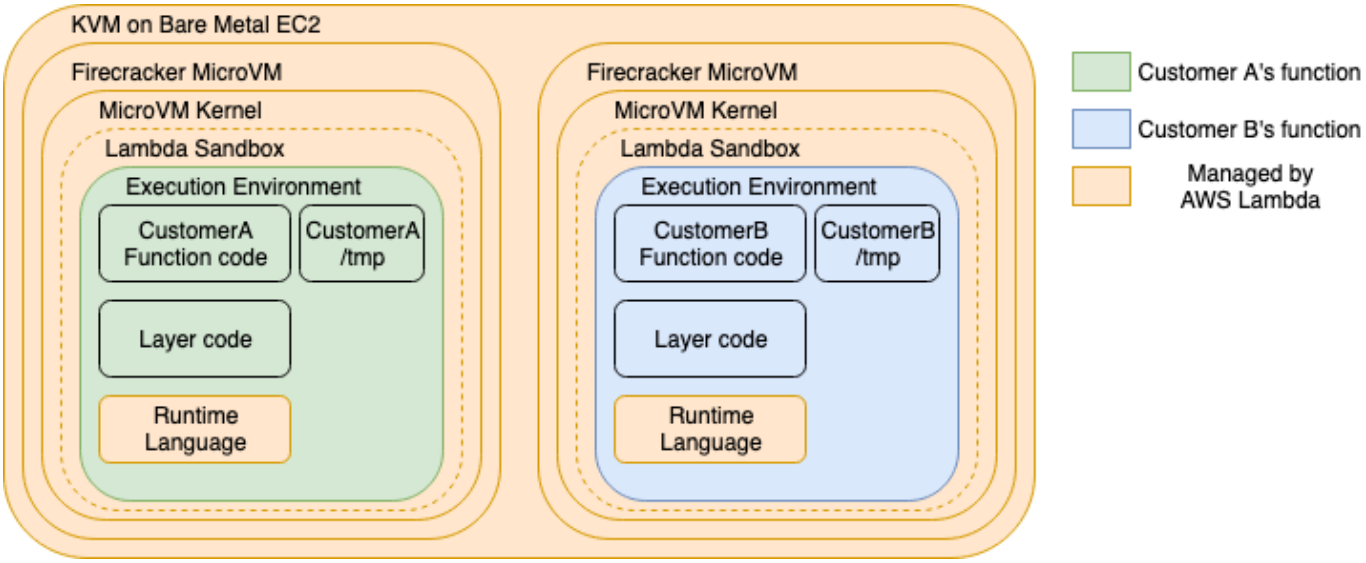

Figure 3 – Isolation model for AWS Lambda Workers

Workers have a maximum lease lifetime of 14 hours. When a Worker approaches maximum lease time, no further invocations are routed to it, MVMs are gracefully terminated, and the underlying Worker instance is terminated. Lambda continuously monitors and alarms on lifecycle activities of its fleet's lifetime.

All data plane communications to workers are encrypted using Advanced Encryption Standard with Galois/Counter Mode (AES-GCM). Other than through data plane operations, customers cannot directly interact with a worker as it hosted in a network isolated Amazon VPC managed by Lambda in Lambda's service accounts.

When a Worker needs to create a new execution environment, it is given time-limited authorization to access customer function artifacts. These artifacts are specifically optimized for Lambda's execution environment and workers. Function code which is uploaded using the ZIP format is optimized once, and then is stored in an encrypted format using an AWS-managed key and AES-GCM.

Functions uploaded to Lambda using the container image format are also optimized. The container image is first downloaded from its original source, optimized into distinct chunks, and then stored as encrypted chunks using an authenticated convergent encryption method which uses a combination of AES-CTR, AES-GCM, and a SHA-256 MAC. The convergent encryption method allows Lambda to securely deduplicate encrypted chunks. All keys required to decrypt customer data is protected using customer-managed KMS Customer Master Key (CMK). CMK usage by the Lambda service is available to customers in AWS CloudTrail logs for tracking and auditing.

Lambda Isolation Technologies

Lambda uses a variety of open-source and proprietary isolation technologies to protect Workers and execution environments. Each execution environment contains a dedicated copy of the following items:

- The code of the particular function version
- Any AWS Lambda Layers selected for your function version
- The chosen function runtime (for example, Java 11, NodeJS 12, Python 3.8, and so on) or the function's custom runtime
- A writeable `/tmp` directory
- A minimal Linux user space based on Amazon Linux 2

Execution environments are isolated from one another using several container-like technologies built into the Linux kernel, along with AWS proprietary isolation technologies. These technologies include:

- **cgroups** – Used to constrain the function's access to CPU and memory.
- **namespaces** – Each execution environment runs in a dedicated namespace. We do this by having unique group process IDs, user IDs, network interfaces, and other resources managed by the Linux kernel.
- **seccomp-bpf** – To limit the system calls (syscalls) that can be used from within the execution environment.
- **iptables** **and** **routing tables** – To prevent ingress network communications and to isolate network connections between MVMs.
- **chroot** – Provide scoped access to the underlying filesystem.
- Firecracker configuration – Used to rate limit block device and network device throughput
- Firecracker **security features** – For more information about Firecracker's current security design, please review Firecracker's latest design document.

Along with AWS proprietary isolation technologies, these mechanisms provide strong isolation between execution environments.

Storage and State

Execution environments are never reused across different function versions or customers, but a single environment can be reused between invocations of the same function version. This means data and state can persist between invocations. Data and/or state may continue to persist for hours before it is destroyed as a part of normal execution environment lifecycle management. For performance reasons, functions can take advantage of this behavior to improve efficiency by keeping and reusing local caches or long-lived connections between invocations. Inside an execution environment, these multiple invocations are handled by a single process, so any process-wide state (such as a static state in Java) can be available for future invocations to reuse, if the invocation occurs on a reused execution environment.

Each Lambda execution environment also includes a writeable filesystem, available at `/tmp`. This storage is not accessible or shared across execution environments. As with the process state, files written to `/tmp` remain for the lifetime of the execution environment. This allows expensive transfer operations, such as downloading machine learning (ML) models, to be amortized across multiple invocations. Functions that don't want to persist data between invocations should either not write to `/tmp`, or delete their files from `/tmp` between invocations. The `/tmp` directory is backed by an EC2 instance store and is encrypted at-rest.

Customers that want to persist data to the file system outside of the execution environment should consider using Lambda's integration with Amazon Elastic File System (Amazon EFS). For more information, see Using Amazon EFS with AWS Lambda.

If customers don't want to persist data or state across invocations, Lambda recommends that they do not use the execution context or execution environment to store data or state. If customers want to actively prevent data or state leaking across invocations, Lambda recommends that they create distinct functions for each state. Lambda does not recommend that customers use or store security sensitive state into the execution environment, as it may be mutated between invocations. We recommend recalculating the state on each invocation instead.

Runtime Maintenance in Lambda

Lambda provides support for multiple programming languages through the use of runtimes, including Java 11, Python 3.8, Go 1.x, NodeJS 12, .NET core 3.1, and others. For a complete list of currently supported runtimes, see AWS Lambda Runtimes.

Lambda provides support for these runtimes by continuously scanning for and deploying compatible updates and security patches, and by performing other runtime maintenance activity. This enables customers to focus on just the maintenance and security of any code included in their Function and Layer. The Lambda team uses AWS Inspector to

discover known security issues, as well as other custom security issues notification mechanisms and pre-disclosure lists to ensure that our runtime languages and execution environment remain patched. If any new patches or updates are identified, Lambda tests and deploys the runtime updates without any involvement from customers. For more information about Lambda's compliance program, see the Lambda and Compliance section of this document.

Typically, no action is required to pick up the latest patches for supported Lambda runtimes, but sometimes action might be required to test patches before they are deployed (for example, known incompatible runtime patches). If any action is required by customers, Lambda will contact them through the Personal Health Dashboard, through the AWS account's email, or through other means, with the specific actions required to be taken.

Customers can use other programming languages in Lambda by implementing a custom runtime. For custom runtimes, maintenance of the runtime becomes the customer's responsibility, including making sure that the custom runtime includes the latest security patches. For more information, see Custom AWS Lambda runtimes in the *AWS Lambda Developer Guide*.

When upstream runtime language maintainers mark their language End-Of-Life (EOL), Lambda honors this by no longer supporting the runtime language version. When runtime versions are marked as deprecated in Lambda, Lambda stops supporting the creation of new functions and updates to existing functions that were authored in the deprecated runtime. To alert customer of upcoming runtime deprecations, Lambda sends out notifications to customers of the upcoming deprecation date, and what they can expect. Lambda does not provide security updates, technical support, or hotfixes for deprecated runtimes, and reserves the right to disable invocations of functions configured to run on a deprecated runtime at any time. If customers want to continue to run deprecated or unsupported runtime versions, they can create their own custom AWS Lambda runtime. For details on when runtimes are deprecated, see the AWS Lambda Runtime support policy.

Monitoring and Auditing Lambda Functions

You can monitor and audit Lambda functions with many AWS services and methods, including the following services:

Amazon CloudWatch

Lambda automatically monitors Lambda functions on your behalf. Through Amazon CloudWatch, it reports metrics such as the number of requests, the execution duration per request, and the number of requests resulting in an error. These metrics are exposed at the function level, which you can then leverage to set CloudWatch alarms.

For a list of metrics exposed by Lambda, see Working with AWS Lambda function metrics.

AWS CloudTrail

Using AWS CloudTrail, you can implement governance, compliance, operational auditing, and risk auditing of your entire AWS account, including Lambda. CloudTrail enables you to log, continuously monitor, and retain account activity related to actions across your AWS infrastructure, providing a complete event history of actions taken through the AWS Management Console, AWS SDKs, command line tools, and other AWS services. Using CloudTrail, you can optionally encrypt log files using KMS and also leverage CloudTrail log file integrity validation for positive assertion.

AWS X-Ray

Using AWS X-Ray, you can analyze and debug production and distributed Lambda-based applications, which enables you to understand the performance of your application and its underlying services, so you can eventually identify and troubleshoot the root cause of performance issues and errors. X-Ray's end-to-end view of requests as they travel through your application shows a map of the application's underlying components, so you can analyze applications during development and in production.

AWS Config

With AWS Config, you can track configuration changes to the Lambda functions (including deleted functions), runtime environments, tags, handler name, code size, memory allocation, timeout settings, and concurrency settings, along with Lambda IAM execution role, subnet, and security group associations. This gives you a holistic view of the Lambda function's lifecycle and enables you to surface that data for potential audit and compliance requirements.

Architecting and Operating Lambda Functions

Now that we have discussed the foundations of the Lambda service, we move on to architecture and operations. For information about standard best practices for serverless applications, see the *Serverless Application Lens* whitepaper, which defines and explores the pillars of the AWS Well Architected Framework in a Serverless context.

- **Operational Excellence Pillar** – The ability to run and monitor systems to deliver business value and to continually improve supporting processes and procedures.

- **Security Pillar** – The ability to protect information, systems, and assets while delivering business value through risk assessment and mitigation strategies.
- **Reliability Pillar** – The ability of a system to recover from infrastructure or service disruptions, dynamically acquire computing resources to meet demand, and mitigate disruptions such as misconfigurations or transient network issues.
- **Performance Efficiency Pillar** – The efficient use of computing resources to meet requirements and the maintenance of that efficiency as demand changes and technologies evolve.

The *Serverless Application Lens* whitepaper includes topics such as logging metrics and alarming, throttling and limits, assigning permissions to Lambda functions, and making sensitive data available to Lambda functions.

Lambda and Compliance

As mentioned in The Shared Responsibility Model section of this document, you are responsible for determining which compliance regime applies to your data. After you have determined your compliance regime needs, you can use the various Lambda features to match those controls. You can contact AWS experts (such as solution architects, domain experts, technical account managers, and other human resources) for assistance. However, AWS cannot advise customers on whether (or which) compliance regimes are applicable to a particular use case.

As of November 2020, Lambda is in scope for SOC 1, SOC 2, and SOC 3 reports, which are independent third-party examination reports that demonstrate how AWS achieves key compliance controls and objectives. In addition, Lambda maintains compliance with PCI DSS and the U.S. Health Insurance Portability and Accountability Act (HIPAA), among other compliance programs. For an up-to-date list of compliance information, see the AWS Services in Scope by Compliance Program page.

Because of the sensitive nature of some compliance reports, they cannot be shared publicly. For access to these reports, you can sign in to your AWS console and use AWS Artifact, a no cost, self-service portal, for on-demand access to AWS compliance reports.

Lambda Event Sources

Lambda integrates with more than 140 AWS services via direct integration and the Amazon EventBridge event bus. The commonly used Lambda event sources are:

- Amazon API Gateway
- Amazon CloudWatch Events

- Amazon CloudWatch Logs
- Amazon DynamoDB Streams
- Amazon EventBridge
- Amazon Kinesis Data Streams
- Amazon S3
- Amazon SNS
- Amazon SQS
- AWS Step Functions

With these event sources, you can:

- Use AWS IAM to manage access to the service and resources securely.
- Encrypt your data at-rest[1]. All services encrypt data in transit.
- Access from your Amazon VPC using VPC endpoints (powered by AWS PrivateLink).
- Use Amazon CloudWatch to collect, report, and alarm on metrics.
- Use AWS CloudTrail to log, continuously monitor, and retain account activity related to actions across your AWS infrastructure, providing a complete event history of actions taken through the AWS Management Console, AWS SDKs, command line tools, and other AWS services.

Conclusion

AWS Lambda offers a powerful toolkit for building secure and scalable applications. Many of the best practices for security and compliance in Lambda are the same as in all AWS services, but some are particular to Lambda. This whitepaper describes the benefits of Lambda, its suitability for applications, and the Lambda-managed runtime environment. It also includes information about monitoring and auditing, and security and compliance best practices. As you think about your next implementation, consider what you learned about Lambda, and how it might improve your next workload solution.

Contributors

Contributors to this document include:

- Mayank Thakkar, Senior Solutions Architect

- Marc Brooker, Senior Principal Engineer
- Osman Surkatty, Senior Security Engineer

Further Reading

For additional information, see:

- Shared Responsibility Model, which explains how AWS thinks about security in general.
- Security best practices in IAM, which covers recommendations for AWS Identity and Access Management (IAM) service.
- *Serverless Application Lens* covers the AWS well-architected framework and identifies key elements to help ensure your workloads are architected according to best practices.
- *Introduction to AWS Security* provides a broad introduction to thinking about security in AWS.
- *Amazon Web Services: Risk and Compliance* provides an overview of compliance in AWS.

Document Revisions

Date	Description
March 2019	First publication
January 2021	Re-published with significant updates

Notes

[1] At the time of publishing, encryption of data at-rest was not available for Amazon EventBridge. Continue to monitor the service homepages for updates on these capabilities.