AWS Whitepaper

Security Overview of AWS Lambda

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Security Overview of AWS Lambda: AWS Whitepaper

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

Table of Contents

Abstract and introduction ... i
Abstract ... 1
Are you Well-Architected? ... 1
Introduction ... 1
Benefits of Lambda ... 3
No servers to manage ... 3
Continuous scaling ... 3
Millisecond metering ... 3
Increases innovation ... 3
Modernize your applications ... 3
Support for developers ... 4
The Shared Responsibility Model ... 5
Data privacy ... 5
Security at rest ... 6
Security in transit ... 6
Operational security ... 7
Vulnerability management ... 7
Trusted code execution ... 7
Lambda functions and layers ... 8
Lambda invoke modes ... 9
Lambda executions ... 11
Lambda execution environments ... 11
Execution role ... 12
Lambda MicroVMs and Workers ... 13
Lambda isolation technologies ... 16
Storage and state ... 17
Runtime maintenance in Lambda ... 18
Monitoring and auditing Lambda functions ... 19
Amazon CloudWatch ... 19
Amazon CloudTrail ... 19
AWS X-Ray ... 19
AWS Config ... 19
Architecting and operating Lambda functions ... 21
Lambda and compliance ... 22

Lambda event sources 23
Conclusion 24
Contributors 25
Further reading 26
Document revisions 27
Notices 28
AWS Glossary 29

Security Overview of AWS Lambda

Publication date: **December 27, 2022** (*Document revisions*)

Abstract

This whitepaper presents a deep dive of 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.

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

Are you Well-Architected?

The AWS Well-Architected Framework helps you understand the pros and cons of the decisions you make when building systems in the cloud. The six pillars of the Framework allow you to learn architectural best practices for designing and operating reliable, secure, efficient, cost-effective, and sustainable systems. Using the AWS Well-Architected Tool, available at no charge in the AWS Management Console, you can review your workloads against these best practices by answering a set of questions for each pillar.

For more expert guidance and best practices for your cloud architecture—reference architecture deployments, diagrams, and whitepapers—refer to the AWS Architecture Center.

Introduction

AWS Lambda is an event-driven, serverless compute service that extends other AWS services with custom logic, or creates other backend services that operate with scale, performance, and security. Lambda can automatically run code in response to multiple events, such as HTTP requests through Amazon API Gateway or function URL, modifications to objects in Amazon Simple Storage Service (Amazon S3) buckets, table updates in Amazon DynamoDB, messages in Amazon Simple Queue Service (Amazon SQS) notifications in Amazon Simple Notification Service (Amazon SNS), streaming data in Amazon Kinesis, events or logs in Amazon CloudWatch, events in Amazon EventBridge and state transitions in AWS Step Functions. You can also run code directly from

any web or mobile app. 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 with high availability. Lambda integrates with many other AWS services and enables you to create serverless applications or backend services, ranging from periodically initiated, simple automation tasks to full-fledged microservices applications.

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

You can easily wrap up Lambda with a strong security posture using AWS Identity and Access Management (IAM), and other techniques discussed in this whitepaper to maintain a high level of security and auditing, and to meet your compliance needs.

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.

Benefits of Lambda

Customers who want to increase 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 AWS Lambda enables them to 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 initiated code in parallel and processing each event individually.

Millisecond metering

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

Increases innovation

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

Modernize your applications

Lambda enables you to use functions with pre-trained machine learning (ML) models to inject artificial intelligence into applications more easily. A single application programming interface

(API) request can classify images, analyze videos, convert speech to text, perform natural language processing, and more.

Support for developers

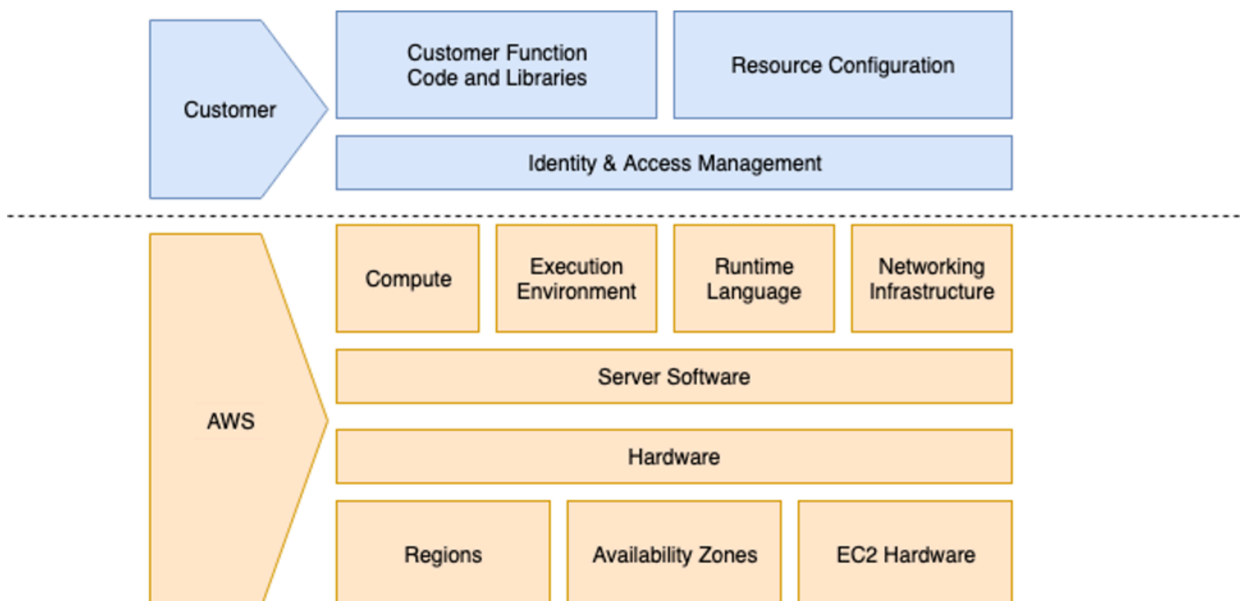
Lambda supports developers through the AWS Serverless Application Repository for discovering, deploying, and publishing serverless applications; AWS Serverless Application Model for building serverless applications and integrations with various integrated development environments (IDEs) (such as AWS Cloud9, AWS Toolkit for Visual Studio, AWS Toolkits for Azure DevOps, and several others). For more information, refer to AWS Toolkits for PyCharm, IntelliJ, and Visual Studio Code. Lambda is integrated with additional AWS services to empower you to take advantage of the breadth and depth of AWS when building serverless applications.

The Shared Responsibility Model

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 AWS Lambda, AWS manages the underlying infrastructure and foundation services, the operating system, and the application platform. You are responsible for the security of your code and AWS IAM to the Lambda service and within your function.

The following figure shows the shared responsibility model as it applies to the common and distinct components of AWS Lambda. AWS responsibilities appear below the dotted line in orange, and customer responsibilities appear above the dotted line in blue.



Shared responsibility model for AWS Lambda

Data privacy

Lambda follows strict policies to protect customer privacy and confidentiality. Our customer data handling policies restrict AWS employees from accessing customer content for any reason except those authorized by the customer to support their investigation or to comply with a valid and binding law enforcement request. Encryption at rest, customer managed AWS Key Management Service (AWS KMS) key, encryption in transit, access control, General Data Protection Regulation

(GDPR) compliance and human access prevention are some examples of mechanisms employed by Lambda to protect customer content.

Security at rest

Databases and storage systems used by Lambda are encrypted following the AWS best practices guidelines. When applicable, Lambda-managed AWS KMS keys (KMS keys) in the customer account are used to offer better traceability. The option to use customer-managed KMS keys is also provided for sensitive customer content. Refer to Customer keys and AWS keys for information about different key management schemes provided by AWS KMS. For example, function runtime environment variables are secured by encryption using a Lambda-managed KMS key (named `aws/lambda`) in the customer's account. You can optionally provide a per-function KMS key to Lambda for encrypting those variables. This customer managed KMS key can be configured using the `CreateFunction` API during function creation time, or later using `UpdateFunctionConfiguration` API. For more information, refer to Securing environment variables.

Lambda functions deployed as container images are encrypted using an AWS Lambda-managed KMS key. If a customer-managed KMS key is provided through the `CreateFunction` or `UpdateFunctionConfiguration` APIs, that key is used instead of a Lambda-managed KMS key. When the function container images are stored in Amazon Elastic Container Registry (Amazon ECR), customers are responsible for protecting their images in ECR.

AWS X-Ray also encrypts data by default, and can be configured to use a customer-managed key. For details, refer to Encrypt log data in CloudWatch Logs using AWS Key Management Service and Data protection in AWS X-Ray.

Customer content, both ongoing and historical, is deleted by Lambda after account closure based on the AWS data retention and deletion policy. Customers can also delete their data (including backups). To delete the Lambda function, use `DeleteFunction`. To delete Lambda event source mappings that invoke a function, use `DeleteEventSourceMapping`.

Security in transit

Lambda uses Transport Layer Security (TLS) 1.2+ for all public APIs. All communications are protected by TLS 1.2+ when you manage Lambda resources through the AWS Management Console, the AWS SDK, or the Lambda API. Internal communications are also secured by TLS unless the payload is already encrypted. Customers can connect their functions to file systems; Lambda

uses encryption in transit for all connections. For more information, refer to Configuring file system access for Lambda functions.

Operational security

The Lambda service platform runs on Amazon Elastic Compute Cloud (Amazon EC2) instances based on AWS Nitro System, which provides enhanced security by continuous monitoring and a reduced attack surface by offloading virtualization and security functions to dedicated hardware and software. For customer privacy, human access is disabled on hosts running customers' Lambda functions. Access to hosts running Lambda services for operational purposes is controlled by multi-layer access controls, logging, monitoring, and audit.

Vulnerability management

Code releases go through security review and penetration testing. All technology stacks are regularly scanned for vulnerabilities. An annual red team security testing and assessment is performed as an additional security validation step.

Trusted code execution

Lambda provides a code signing feature to ensure only trusted code is run in your Lambda function. When you enable code signing for a function, Lambda verifies (during every code deployment) that the code package is signed by a trusted source, the code has not been altered, and the signature has not expired or been revoked. For information about how to use code signing, refer to Configuring code signing 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. A function 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 KMS, and in-transit by using TLS 1.2+.

You can manage access to your functions and layers through AWS Lambda policies, or through resource-based permissions. For a full list of supported IAM features, refer to AWS services that work with IAM. The attribute-based access control (ABAC) in Lambda lets you control access to your function using tags attached to Lambda functions. Refer to attribute-based access control for Lambda for more information.

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

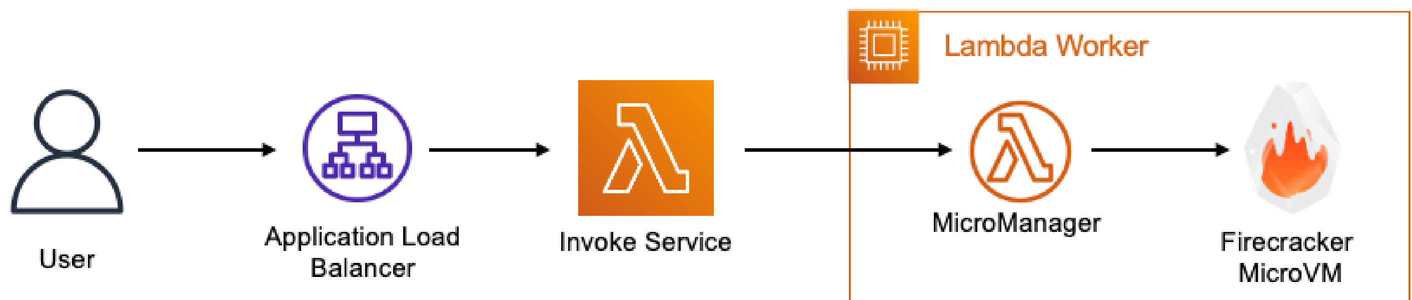
The Invoke API can be called in two modes: *event* mode and *request-response* mode.

- *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, refer to *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, refer to 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.



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 SQS queue. Queued events are always secured in-transit with TLS 1.2+, and are encrypted at rest using Server-Side-Encryption (SSE). 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 Amazon 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, refer to Using AWS Lambda with other services.

When an event fails all processing attempts, it is discarded by Lambda. The dead letter queue (DLQ) feature allows sending unprocessed events from asynchronous invocations to an Amazon SQS queue or an Amazon SNS topic defined by the customer. Enabling DLQ can help you to analyze unprocessed events or to define a workflow to reprocess them. For more information, refer to Dead-letter queues.

Lambda also supports function URLs, a built-in HTTPS endpoint for invoking functions. This provides developers with a simple way to configure HTTPS endpoints to a function without having to configure Amazon API Gateway and Application Load Balancer (ALB). Function URLs do not support AWS WAF; if you need WAF, use API Gateway instead. For more information, refer to Lambda function URLs.

Lambda executions

When Lambda runs 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 the Lambda control plane are protected in-transit by TLS. Customer content stored within Lambda's control plane is encrypted at rest using AWS KMS keys, which are designed to protect the content from unauthorized disclosure or tampering.

The data plane is Lambda's invoke API that cues 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, refer to 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 can 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, we recommend 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, refer to 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.

You 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 you have greater control over start-up performance of their serverless applications at any scale.

Other than through a provisioned concurrency configuration, you 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.

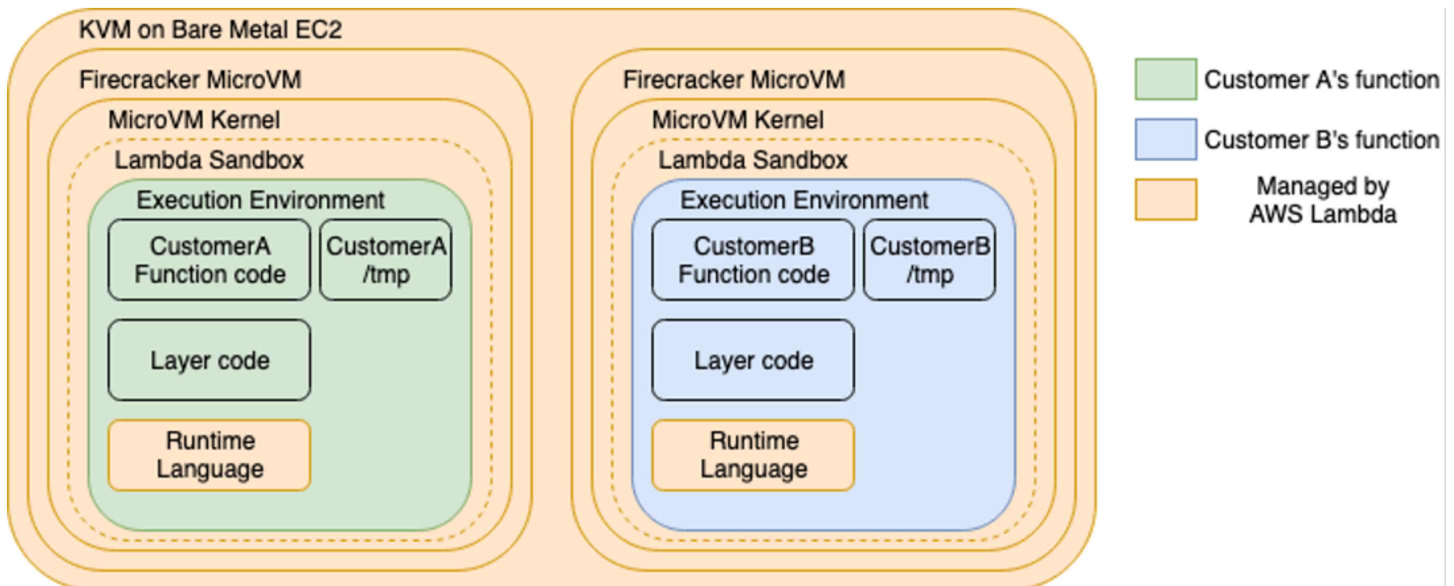
The Lambda service may also assume the execution role to perform certain control plane operations such as those related to creating and configuring elastic network interfaces (ENIs) for VPC functions, sending logs to Amazon CloudWatch Application Insights, sending traces to AWS X-Ray, or other non-invoke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

For more information on this subject, refer to the AWS Lambda execution role documentation.

Lambda MicroVMs and Workers

Lambda will create its execution environments on a fleet of Amazon EC2 instances called *AWS Lambda Workers*. Workers are bare metal Amazon EC2 AWS 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, refer to 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 Amazon Inspector to discover known potential security issues, as well as other custom security issue notification mechanisms and pre-disclosure lists, so that you don't need to manage the underlying security posture of their execution environment.



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, you cannot directly interact with a worker as it hosted in a network isolated Amazon VPC managed by Lambda in the Lambda 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-GCM and SHA-256 MAC. The convergent encryption method allows Lambda to securely deduplicate encrypted chunks. All keys required to decrypt customer content is protected using a customer managed AWS KMS key. If a customer-managed key is not provided, an AWS KMS

key owned by the customer and managed by Lambda is used. When a customer-managed KMS key is used, KMS key usage by the Lambda service is available to customers in AWS CloudTrail logs for tracking and auditing.

SnapStart is a performance optimization feature in Lambda to reduce a Java function's startup latency, commonly known as cold start time. With SnapStart, Lambda takes a Firecracker MicroVM snapshot of the initialized execution environment (memory and disk) when you publish a version and persists the encrypted snapshot. Upon concurrency scale-ups, Lambda clones this snapshotted sandbox and resumes the function execution from the pre-initialized state. SnapStart is currently supported with the Java11 runtime. SnapStart isn't supported with provisioned concurrency, arm64 architecture, extensions, Amazon Elastic File System (Amazon EFS), and ephemeral storage greater than 512 MB. If your application uses random values, you must evaluate your function code and verify that it is resilient to snapshot operations. For more information, refer to Handling uniqueness with Lambda SnapStart. If your code deals with network connections, you may need to validate and re-establish them as necessary. You can use a runtime hook to re-establish connections. Network connections established by an AWS SDK are mostly automatically resumed. For other connections, review the Best practices for working with Lambda SnapStart If your function deals with ephemeral data, such as temporary credentials or cached timestamps, during the initialization phase, you can use a runtime hook to refresh temporary data.

If you want to prevent certain data from being stored in a snapshot, use a `beforeCheckpoint` runtime hook to delete the data before Lambda creates the snapshot. Snapshots are encrypted using a customer-unique AWS KMS key in the customer account managed by Lambda, similarly to the encryption for container images above; however, chunks are not deduplicated in this case. You can also use a customer managed AWS KMS key for control over the key. If a function is not invoked for 14 consecutive days, the snapshot is deleted. All resources associated with the deleted snapshot are removed in compliance with GDPR retention policies.

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:

- **Control groups (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, refer to 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 encrypted at rest.

If you want to persist data to the file system outside of the execution environment, consider integrating Lambda with Amazon EFS. For more information, refer to Using Amazon EFS with Lambda.

If you don't want to persist data or state across invocations, we recommend that you do not use the execution context or execution environment to store data or state. If you want to actively prevent data or state leaking across invocations, we recommend you create distinct functions for each state. We do *not* recommend that you 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 these runtimes by continuously scanning for and deploying compatible updates and security patches, and by performing other runtime maintenance activities. This enables you to focus on just the maintenance and security of any code included in your function and layer. The Lambda team uses Amazon 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, refer to 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.

You 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, refer to 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 you 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 you want to continue to run deprecated or unsupported runtime versions, you can create your own custom AWS Lambda runtime. For details on when runtimes are deprecated, refer to our runtime deprecation 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

AWS 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, refer to Working with Lambda function metrics.

Amazon 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 the log files using AWS KMS and also use the 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. The X-Ray 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

This section discusses Lambda architecture and operations. For information about standard best practices for serverless applications, refer to the Serverless Applications 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.
- **Cost Optimization Pillar –** The continual process of refinement and improvement to ensure that business outcomes are achieved while minimizing cost as demand changes and technologies evolve.

The Serverless Applications 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, 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 you on whether (or which) compliance regimes are applicable to a particular use case.

Lambda Federal Information Processing Standard (FIPS) endpoint operates using FIPS 140-2 validated cryptographic modules. Lambda customers are responsible for encrypting and storing data they process in a way that meets their organizational requirements for data security.

For an up-to-date list of compliance information for Lambda, refer to 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 into the AWS Management 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. All services encrypt data in transit.
- Access from your Amazon Virtual Private Cloud using VPC endpoints (powered by AWS PrivateLink).
- Use Amazon CloudWatch Application Insights 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 described 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 AWS Lambda, and how it might improve your next workload solution.

Contributors

Contributors to this document include:

- Mayank Thakkar, Senior Manager (Global Accounts)
- Marc Brooker, Senior Principal Engineer (AWS Serverless)
- AWS Lambda Security Team

Further reading

For additional information, refer to:

- AWS Architecture Center
- The AWS Shared Responsibility Model, which explains how AWS thinks about security in general.
- The AWS risk and compliance program, which provides an overview of compliance in AWS.
- The Serverless Applications Lens, which covers the AWS Well-Architected Framework identifies key elements to ensure your workloads are architected according to best practices.
- Introduction to AWS Security, which provides a broad introduction to thinking about security in AWS.

Document revisions

To be notified about updates to this whitepaper, subscribe to the RSS feed.

Change	Description	Date
Document updated	New feature updates.	December 27, 2022
Document updated	Terminology update.	April 6, 2022
Initial publication	Significant updates.	February 15, 2021
Initial publication	Whitepaper published.	January 3, 2019

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.

© 2022 Amazon Web Services, Inc. or its affiliates. All rights reserved.

AWS Glossary

For the latest AWS terminology, see the AWS glossary in the *AWS Glossary Reference*.