Amazon Web Services Launches AWS Lambda

Easily run any code as an AWS-operated scalable, secure, and reliable cloud service

SEATTLE – (Nov XX, 2014) – Amazon Web Services LLC (AWS), an Amazon.com company (NASDAQ:AMZN), today announced the introduction of AWS Lambda, the simplest way to run code in the cloud. Previously, running code in the cloud meant creating a cloud service to host the application logic, and then operating the service, requiring developers to be experts in everything from automating failover to security to service reliability. Lambda eliminates the operational costs and learning curve for developers by turning any code into a secure, reliable and highly available cloud service with a web accessible end point within seconds. Lambda uses trusted AWS infrastructure to automatically match resources to incoming requests, ensuring the resulting service can instantaneously scale with no change in performance or behavior. This frees developers to focus on their application logic – there is no capacity planning or up-front resource type selection required to handle additional traffic. There is no learning curve to get started with Lambda – it supports familiar platforms like Java, Node.js, Python and Ruby, with rich support for each language's standard and third-party libraries. Lambda is priced at $XXX for each request handled by the developer's service and $YYY for each 250ms of execution time, making it cost effective at any amount of usage. To get started, visit http://aws.amazon.com/Lambda.

"Lambda has enabled us to deliver a world class cross device photo sharing app." says CTO of XXX. *"We had great success with our app on iOS, hitting 1M downloads in 3 weeks, which led to a huge demand for the Android version. As we considered ways to leverage the cloud to make our app cross platform, we chose Lambda as the cost-effective choice to host our critical image processing logic. The support for JavaScript and Git made it easy for us to get our backend up and running within a few days. Previously, we would have incurred a huge management and operational overhead to maintain our user promise of fast and reliable response time for our customers as they access their pictures. With Lambda and AWS, we know we have the scale, stability and performance to support our customers even in periods of unpredictable demand."*

Using Lambda is simple. Developers express their application logic as they choose – as a one-line Python script, a Java application using native JNI libraries, or even a binary executable compiled from C or C++. When ready, developers can upload their code as a ZIP file, point Lambda to their Git repository, or author code directly from any browser using the AWS Console. Lambda makes it easy to write code that securely integrates other AWS services, with built in support for AWS SDKs and automatic integration with AWS Identity management (IAM) and Cognito Identity Broker. Lambda turns the code into a secure, highly available service within seconds that can be called from any connected device or app and requires no code or configuration changes to handle additional traffic. Developers retain full visibility over the functional and operational performance of their code by monitoring their service using the AWS Lambda console and troubleshooting issues using Amazon CloudWatch Logging.

"Lambda has been an accelerator for my team to port and launch multiple services as we embrace the cloud." says Head of Corporate Development at XXX. *"We have been using Lambda for everything from automating instant account balance updates based on customer trade notifications, to running nightly backups and scrubbing of transaction data to S3. Handling a mix of internal and external users typically presents security challenges, but Lambda incorporates a lot of AWS best practices and controls that makes it just as easy for us to securely connect to our data stored in AWS as well as our internal services. In the past, managing our infrastructure has gotten in the way of quickly iterating on what our customers and internal users want. With Lambda, my developers don't have to be in the business of operating production services and can focus on quickly delivering differentiated business functionality within hours and days, not weeks. Lambda has also ensured my team hasn't given up anything in terms of tuning performance and debugging capabilities by going with a provider."*

Lambda charges for the total number of requests an application handles and the resources it consumes in units of 250ms and WWW GB, so developers pay only for what they use, without the complexity or commitment of up-front reservations. "Lambda is cost effective at one request or thousands per second, all with the same customer code," said Vice President of AWS Mobile. "Developers can wire up any application, script, or process to execute when needed, confident that there will never be a charge for idle time. Short-lived tasks, tasks where the scale changes rapidly, and periodic tasks are as cost effective as steady-state workloads."

Lambda is available immediately in all AWS regions. The first XXX requests and YYY MB-seconds per month are free to all users. For complete pricing details, visit aws.amazon.com/lambda.

External FAQ

General

1. What is Lambda?

Lambda is a secure, reliable and scalable service for running stateless applications in the AWS cloud. Lambda is ideal for developers who want to "just run some code" but have limited expertise or investment in cloud operations and do not want the management and operational overhead of owning a server pool. Lambda manages and scales the servers required to host and execute application code for developers, allowing them to focus on their application logic and stay productive by focusing on the problems they're trying to solve.

2. Who should use Lambda?

Lambda is an ideal solution for developers who don't have expertise to manage infrastructure, or do not require the investment to do so in order to "just run some code". Lambda targets ease of use and developer productivity by defining a standard operating system and language runtime versions; all developers provide is their code and any non-standard libraries it needs. Lambda abstracts away the operational aspects of running code in the cloud, so developers don't think about selecting instance types, cross-availability zone failover, or patching the operating system, runtime, or libraries, while still benefiting from the security, scalability, and availability of AWS. For developers who require direct access to the Amazon EC2 instances or who want to customize their environment, AWS Elastic Beanstalk offers an easy way to deploy and manage complete applications where developers retain control over the AWS resources.

3. What kind of applications can I use Lambda for?

Lambda can be used to host any application regardless of expected traffic, language and size that don't require a highly customized environment, long-lived state, or persistent database connection. Applications should store any persistent data in services like Amazon S3 or Amazon DynamoDB, and can be as simple as a single line of code or include multiple files and libraries (even natively implemented ones). Developers can create handlers that respond to other AWS service events, application backend services for mobile and tablet apps, run hosted cron jobs, and more. Authoring new applications is easy because it's written in familiar languages and uses existing libraries, including the built-in AWS SDKs.

Developing and deploying applications

4. How do I create and deploy an application?

Developing and deploying applications with Lambda is easy:

- Store the application code, and any required libraries, in Amazon S3 as a ZIP file.
- Use the AWS Console, command line, or Lambda invoke API to deploy the application.

5. How does my code get called when an application is invoked?

File naming conventions ("main.py") and optional code annotations make it easy to call a specific method in an application from a mobile client SDK or in response to an Amazon S3 or Amazon DynamoDB update event. This enables applications to be as simple and targeted as possible – an application can be as little as a one-line method in a single file. There are no complex frameworks to learn, deploy, or maintain. Python WSGI, Ruby Rack, and Node.js invocation patterns are also supported, so application authoring feels natural to developers familiar with web service implementations in those languages.

6. How do I deploy updates to an existing application to Lambda?

Save the new version in S3 or use the Simple Deployment Service to update it directly from a Git repository. The deployment process is the same when deploying code for the first time or updating an existing application: Just call Lambda's update application API with the URL of the code. Within a few seconds, new invocations of the application will be using the updated version.

7. How do I develop and test applications on my desktop?

Because Lambda uses stock versions of the language runtimes and standard libraries, developers can easily develop, test, and debug their code on a desktop or their own EC2 instances with the same environment that will be used when running it on Lambda.

8. How can I optimize my applications to run on Lambda?

Developers can lower costs and improve performance by minimizing startup and shutdown overhead, ensuring that each time an application is invoked it immediately begins doing useful work. Using the standard (default) version of libraries minimizes deployment and startup overhead.

Languages and libraries

9. What languages and libraries does Lambda support?

Lambda allows you to write applications in Java, JavaScript, Ruby and Python.

10. How can my applications access other AWS services?

Any service or data accessible from the Internet is available to applications hosted by Lambda. Outbound HTTP and HTTPS connections work normally; there are no special APIs to learn or use. Client SDKs for AWS services are preinstalled for convenience.

11. What if a library I want to use isn't available?

Developers can bundle third party libraries with their code, allowing them to use libraries (or versions of libraries) that aren't available by default. Native libraries are supported.

12. How does Lambda maintain the runtime environment?

Lambda handles operating system, language runtime, standard library, and preinstalled third-party library updates while ensuring continuous high availability. Developers do not need to take any explicit action to benefit from the automated maintenance of these software frameworks. Application code and libraries provided by developers will not be altered.

13. How does Lambda handle major updates to language runtimes?

Lambda offers the most popular version as the default runtime for each supported language. Other supported runtime versions can be selected in an application's configuration file.

14. Are there restrictions on the code I can run in Lambda?

Applications hosted by Lambda are stateless; persistent state should be stored in Amazon S3, Amazon DynamoDB, or another Internet-available storage service. Inbound network connections are managed by Lambda. For security reasons, some low-level system calls are restricted, but language features, and most libraries, function normally. Local file system access is intended as a temporary scratch space and is deleted between invocations. These restrictions enable Lambda to launch and scale applications on behalf of developers by ensuring that their code can run on Lambda infrastructure and that the service can launch as many copies of their application as needed to scale to the incoming request rate.

Invoking applications

15. How do I invoke my Lambda application?

Once an application has been deployed, it can be called programmatically through Lambda's invoke API, from the command line, or through the AWS Console. Developers can also call it from a mobile client, refer to it in an Amazon Simple Workflow Service crontab file, set it as the handler for an Amazon S3 or Amazon DynamoDB update, or name it as an Amazon Simple Notification Service publication target. Applications can also be invoked from the command line and through the AWS Console.

16. How do I invoke my Lambda application in response to an Amazon S3 PUT or COPY operation?

Use Amazon S3's SetUpdateHandler API to identify the Lambda application that represents the handler, the bucket and path to respond to, and whether to handle PUT, COPY, or both types of updates. After that, every successful PUT or COPY operation will trigger the application, enabling it to react to the S3 operation.

17. How do I invoke my Lambda application in response to an Amazon DynamoDB update?

Use Amazon DynamoDB's SetTableHandler API to identify the table to track and the Lambda application to invoke when the table's content changes. After that, every successful write to the table in question will trigger the corresponding application, enabling it to react to the DynamoDB update.

18. How can I invoke my Lambda application as a hosted cron job?

Developers have two options: In many cases, they can simply indicate the time of day to run when configuring applications using the Lambda console UI, command line tools, or programmatic API. For more complex scenarios, Amazon Simple Workflow Service supports the full power of crontab-specified jobs, allowing developers to identify activities within them to execute using Lambda.

19. How can I invoke my Lambda application as a scheduled batch job?

Developers can execute Lambda applications as batch jobs through the Lambda API as well as from within their application code. They can also maintain their own job queue using Amazon Simple Queue Service (SQS) and use Lambda's integration with SQS to monitor it for jobs to run.

Security

20. How does Lambda secure applications?

Applications execute in a sandbox that isolates them from one another, protecting the integrity of the code and data they contain.

21. How can I control the services and data accessible to an application?

Integration between Lambda and AWS Identity and Access Management (IAM) enables developers and administrators to explicitly control the data and services accessible to an application by assigning it a security role. This approach ensures that credentials can be constrained to the minimum necessary and that they possess limited lifespans.

22. How does Lambda secure my source code?

Lambda uses file system isolation techniques to ensure that each application can see only its own code. Application code in transit inside the service is always encrypted. Fixity tests are employed at multiple levels to prevent unanticipated changes to the code outside of developer-initiated deployments. Backup copies of code are stored in S3 using server-side encryption (SSE).

Capacity and Scale

23. How quickly can my application start scaling to requests?

Because it operates the underlying compute resources for all users, Lambda combines low latency execution with nearly instantaneous scalability – every time an application is invoked, Lambda quickly locates free capacity and securely runs the code. Infrequent or periodic jobs are cost effective, sharing capacity with other users and only charging for actual execution time. "Bursty" or unpredictable workloads, such as cloud-hosted mobile app backends that may experience sudden surges in popularity, scale seamlessly using AWS infrastructure. Developers aren't required to predict their scaling needs: Per-second memory billing enables Lambda to be both highly responsive *and* cost effective even for unpredictable or rapidly varying loads.

24. How does Lambda protect me from overcrowding effects?

When an application is invoked, the service automatically places it onto an EC2 instance with sufficient capacity to execute it. Lambda continuously monitors the performance of each application execution as well as the overall performance of its compute fleet and acquires additional capacity automatically to avoid overcrowding. Lambda's pricing model also ensures that developers only pay for the time that their code is running.

25. How quickly can Lambda scale up and down?

Because Lambda works by running an application every time it's triggered, it scales to match the rate of whatever is triggering it. Since applications are billed for CPU use, the associated charges (rounded up to the next second) end as soon as the application completes. Lambda imposes no warm-up or cool-down periods or charges.

26. What type of availability does Lambda offer?

Lambda is designed to provide 99.99% availability for both the service itself and for the applications running on it. There are no maintenance windows or scheduled downtimes.

Limits and Quotas

27. How long can an application run?

Applications representing batch and timed jobs are terminated after approximately four hours of continuous (wall clock) uptime, though maintenance or security needs may occasionally require them to be terminated earlier. Applications initiated by an HTTP request must complete within the request's lifetime, typically within 30 seconds, although they may invoke batch jobs that outlive the original request.

28. How much memory can an application use?

Each running application can allocate up to 1 GB of virtual memory.

Performance

29. What is the latency of invoking an application using the Lambda API?

Applications in steady use have typical latencies in the range of 20-50ms, determined by timing a simple "echo" application from a client hosted in Amazon EC2. Latency will be higher the first time an application is deployed and when an application has not been used recently.

30. How powerful is the CPU on which my code runs?

Applications running on Lambda execute on vCPUs with a minimum rating of 1 ECU.

31. How does Lambda support parallel processing?

Developers can run multiple applications and/or multiple copies of the same application simultaneously. They can also access Lambda APIs programmatically from within applications, using the AWS client SDK, which allows them to delegate and orchestrate work by running other applications.

Developers can use these techniques to expose parallelism inherent in their code. As an example, to create ten different representations of an image when the original is stored in S3, a single application could serially produce all ten representations. Alternatively, ten handlers could be registered, each producing just one of the required transformations. The latter approach will typically complete faster by enabling Lambda and Amazon S3 to parallelize the work.

Internal FAQ

1. When would we direct a customer to [not] use Lambda?

Mobile backends and any AWS service-embedded scripting uses (such as custom CloudWatch actions or custom video transcoder rules) will use Lambda "under the hood." Customers with these use cases will implicitly select this service.

AWS event handlers and batch/cron jobs where the job is readily expressed as an application are good targets for Lambda, which will offer the customer convenience and simplicity versus setting up their own instances, and where the service can complete jobs quickly by offering nearly unlimited burst capability to run many applications in parallel. At the same time, Lambda will be cost effective, compacting jobs onto the minimal set of EC2 instances, selecting instance types to minimize customer cost, and avoiding any charge for intra-hour idle time. Customers with these use cases may (and often should) target Lambda.

Customers with existing applications ("lift and shift"), those who want access to the underlying EC2 instances, who wish to write in languages other than those supported by the service, or who need "stateful" code cannot use Lambda and should target Beanstalk/EC2.

2. What are the Lambda tenets?

Our tenets, unless you know better ones, are:

- **Security without complexity** – Our service will protect customer data from unauthorized access and will be resilient to attack. We will support security features, perform audits, and achieve certifications that address developers' real and perceived security needs. Developers will benefit from up-to-date security patches at the operating system, runtime, and library level without taking explicit action. Each supported language will feel normal, despite running in a secured environment.

- **Simple and easy** – We will deliver a "NoOps" service that makes developers' lives easier by handling undifferentiated management and operational overhead for them. Defaults will be reasonable for the majority of users and options will be few and easy to understand. Users will have self-service access to deploy and manage their applications.

- **Scales up and down (to zero)** – Our service will scale customer applications without changes to their code or configuration. We will architect such that one application invocation per month and 1,000 per second are both well supported. We will automate fleet capacity analysis to avoid limiting customer capability.

- **Cost effective at any scale** – Our service will target fine-grained pay-for-use; developers will not pay for idle time. We will own the problem of application placement so that developers never experience waste through underutilized hosts. We will seek to minimize both costs and billing granularity.

- **AWS integration** – Our service will benefit from other AWS services by making them easy for application developers to access from within applications. We will make other AWS services better (and better together) by providing a common, hosted AWS application solution. Event triggers, rules engines, hosted cron jobs, and custom actions will be easy to host on Lambda.

- **Reliable** – Both our service and the applications running on it will provide predictable and reliable operational performance. We will provide developers with availability and latency targets and then hold ourselves to a higher bar internally. We will engineer our service to be resilient to failure and to minimize the impact any failure can have. We will monitor, and seek to optimize, aspects of application performance that we control through fleet composition, capacity, and job placement.

3. What operational metrics will we measure and optimize to improve customer experience?

We will seek to optimize three key dimensions of the customer experience for applications and applications running on Lambda: *latency*, *throughput*, and *availability*, and we will monitor a fourth, jitter, to protect customer experience.

Latency

We plan to offer a *publicly* visible measure of latency through a canary client running in EC2 that repeatedly invokes a Lambda-hosted 'echo' application. Monitoring and graphing the resulting latency from the client perspective offers a way to convey to developers the type of latency they will experience when using the service.

Internally we will also measure server-side latency (request arrival to response delay), the efficacy of our process retention and code caching techniques, and latency of process invocation to execution of customer code.

Throughput

We will measure per-host resource contention to identify applications experiencing system-induced delays. Paging rates, CPU utilization, and network bandwidth enable us to measure contention in key resources. Sustained high measures of any one indicates an over-subscribed compute host; sustained high measures fleet-wide trigger calls for additional EC2 capacity to increase the resource and/or a fleet mix shift to expose more of the needed resource. We also report wall clock execution time in logs and via Amazon CloudWatch.

Availability

Availability of the Lambda service itself (our control plane) will be monitored and reported similarly to other AWS services. Availability for applications (our "invoke" plane) will be measured and reported on a per-application basis through CloudWatch.

4. How can Lambda save AWS customers money?

Lambda offers fine-grained duration-based pricing. Like Amazon S3, it charges customers only for what they actually do with the service. Our pricing approach ensures that customers cannot overprovision or underutilize by design: customers utilize 100% of the computing power they're paying for when they run an application. Small and medium customers will benefit from the "scales to zero" aspect of the service, paying nothing at all for applications that are able to receive requests but not actually executing. Many "right tail" customers will fit comfortably into the perennial free tier and pay nothing at all for infrequently invoked applications, without giving up availability, burst capacity, or fault tolerance.

Large customers benefit from Lambda's placement engine, which effectively compacts their workloads: Each new request is placed with respect to minimizing the number of instances dedicated to that account (subject to maintaining latency, throughput, and availability goals). Spiky workloads, heterogeneous workloads, and short-lived jobs such as cron or batch applications all use capacity efficiently without any additional IT oversight on the customer's behalf, potentially saving them money through higher utilization as well as reduced IT staffing needs.

Lambda can also lower TCO by maintaining the operating system, language runtime, and libraries on the customer's behalf and, in collaboration with other AWS services, offering security, scaling, high availability, and job control for the applications it runs. Lower IT cost and complexity is a key requirement for mobile and tablet backend developers in particular, where the unpredictable and rapidly changing popularity of their apps makes accurate workload prediction difficult, but where instantaneous scale-up in response to a suddenly popular app is an effective requirement to deliver a great end-user experience.