AWS Lambda

- A serverless compute service.
- Lambda executes your code only when needed and scales automatically.
- Lambda functions are stateless – no affinity to the underlying infrastructure.
- You choose the amount of memory you want to allocate to your functions and AWS Lambda allocates proportional CPU power, network bandwidth, and disk I/O.
- AWS Lambda is SOC, HIPAA, PCI, ISO compliant.
- Natively supports the following languages:
    - Node.js
    - Java
    - C#
    - Go
    - Python
    - Ruby
    - PowerShell
- You can also provide your own custom runtime.

Introduction to AWS Lambda & Serverless Applications

Components of a Lambda Application

- **Function** – a script or program that runs in Lambda. Lambda passes invocation events to your function. The function processes an event and returns a response.
- **Execution environment** – a secure, isolated micro virtual machine where a Lambda function is executed.
- **Runtimes** – Lambda runtimes allow functions in different languages to run in the same base execution environment. The runtime sits in-between the Lambda service and your function code, relaying invocation events, context information, and responses between the two.
- **Environment variables** – key-value pairs that you can use to store configuration settings for your function. They can be used to pass dynamic parameters to your function at runtime, such as database connection strings, API keys, and other sensitive information.
- **Layers** – Lambda layers are a distribution mechanism for libraries, custom runtimes, and other function dependencies. Layers let you manage your in-development function code independently from the unchanging code and resources that it uses.

- **Event source** – an AWS service or a custom service that triggers your function and executes its logic.
- **Downstream resources** – an AWS service that your Lambda function calls once it is triggered.
- **Log streams** – While Lambda automatically monitors your function invocations and reports metrics to CloudWatch, you can annotate your function code with custom logging statements that allow you to analyze the execution flow and performance of your Lambda function.
- AWS Serverless Application Model

Lambda Functions

- You can upload your application code as a ZIP file or a container image hosted on Amazon Elastic Container Registry (Amazon ECR).
- To create a Lambda function, you first package your code and dependencies in a deployment package. Then, you upload the deployment package to create your Lambda function.
- After your Lambda function is in production, Lambda automatically monitors functions on your behalf, reporting metrics through Amazon CloudWatch.
- Configure **basic function settings,** including the description, memory usage, storage (512MB – 10GB), execution timeout (15 minutes max), and role that the function will use to execute your code.
- **Environment variables** are always encrypted at rest, and can be encrypted in transit as well.
- **Versions** – a snapshot of your function's state at a given time. When you publish a new version, a **:version-number** is appended to your function's ARN:
    - **arn:aws:lambda:us-east-2:123456789123:function:my-function:1**
- **Aliases** – serves as a pointer to a Lambda function version. Aliases create a human-readable version of the function's name, making it easier to remember and understand what the function does. An alias follows the following format:
    - **arn:aws:lambda:us-east-2:123456789123:function:my-function:MyAlias**
- A **layer** is a ZIP archive that contains libraries, a custom runtime, or other dependencies. Use layers to manage your function's dependencies independently and keep your deployment package small.
- You can configure a function to mount an Amazon EFS file system to a local directory. With Amazon EFS, your function code can access and modify shared resources securely and at high concurrency.

Invoking Lambda Functions

- Lambda supports **synchronous** and **asynchronous invocation** of a Lambda function.
- Synchronous invocation
    - when a function is invoked synchronously, AWS Lambda waits until the function is done processing, then returns the result.
    - examples of AWS services that invoke Lambda functions synchronously:
        - Amazon API Gateway Application Load Balancer
        - Amazon Cognito Amazon
        - Kinesis Data Firehose
        - Amazon CloudFront (Lambda@Edge)
- Asynchronous invocation
    - when a function is invoked asynchronously, AWS Lambda stores the event in an internal queue and handles the invocation
    - the Lambda function returns a *202 status code (Accepted)* immediately after being invoked, and the processing continues in the background. The 202 code just confirms that the event is queued; it does not indicate whether the function runs successfully or not.
    - typically used for long-latency processes that run in the background, such as batch operations, video encoding, and order processing.
    - can only accept a payload of up to 256 KB.
    - examples of AWS services that invoke Lambda functions asynchronously:
        - Amazon API Gateway (by specifying **Event** in the **X-Amz-Invocation-Type** request header of a non-proxy integration)
        - Amazon S3
        - Amazon CloudWatch Logs
        - Amazon EventBridge
        - AWS CodeCommit
        - AWS CloudFormation
        - AWS Config

Event Source Mapping

- *Event source mapping* is a Lambda resource that reads from a queue or stream and synchronously invokes a Lambda function.
- You can apply an event-filtering pattern to process events that are only relevant to your application. This allows you to save money by reducing the number of function invocations.
- Event source mapping invokes a function if one of the following conditions is met:
    - The batch size is reached
    - The maximum batching window is reached
    - The total payload is 6 MB

- Lambda provides event source mappings for the following services.
    - Amazon Kinesis
    - Amazon DynamoDB
    - Amazon Simple Queue Service
    - Amazon MQ
    - Amazon Managed Streaming for Apache Kafka (Amazon MSK)
    - Self-managed Apache Kafka

Deploying Codes with External Dependencies

- AWS Lambda includes a number of pre-built dependencies for specific runtimes. These dependencies can be used to run your code without having to include them in your deployment package.
- If you're using an external library/SDK/module in your Lambda code, do the following steps:
    1. Place all external dependencies locally in your application's folder.
    2. Create a ZIP deployment package of your Lambda function.
    3. Upload the deployment package to AWS Lambda. You can send the file directly to the AWS Lambda Console or store it first in Amazon S3 and deploy it from there.

Concurrency Management

- *Concurrency* is the number of instances that serve requests at a given time. When your function is invoked, Lambda allocates an instance of it to process the event. When the function finishes running, it can handle another request. If the function is invoked again while a request is still being processed, another instance is allocated, which increases the function's concurrency.
- To ensure that a function can always reach a certain level of concurrency, you can configure the function with **reserved concurrency**. When a function has reserved concurrency, no other function can use that concurrency. Reserved concurrency also limits the maximum concurrency for the function.
- To enable your function to scale without fluctuations in latency, use **provisioned concurrency**. By allocating provisioned concurrency before an increase in invocations, you can ensure that all requests are served by initialized instances with very low latency.

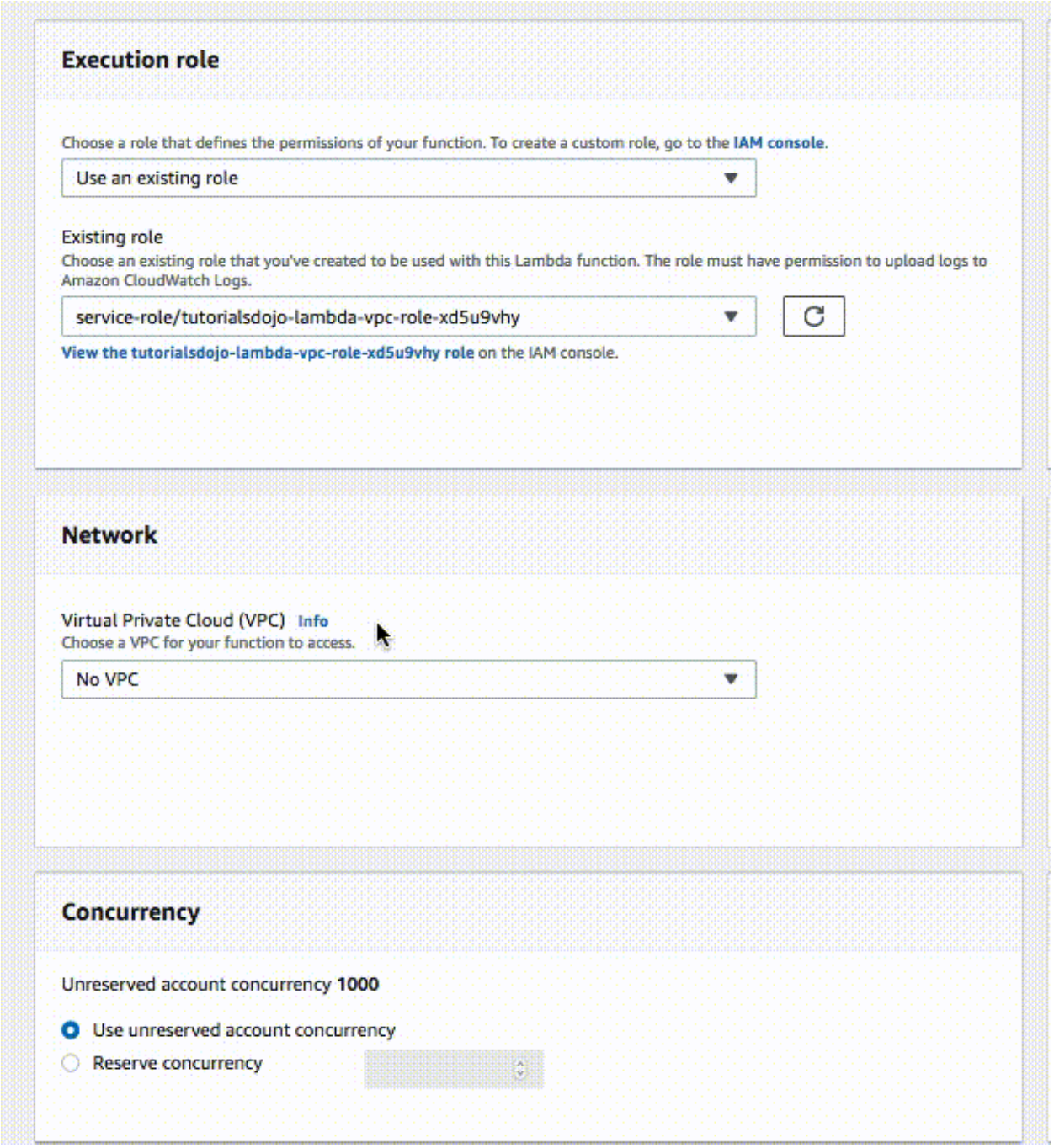
Lambda Function URL

- With the function URL feature of the AWS Lambda service, you can launch a secure HTTPS endpoint dedicated to your custom Lambda function.
- You don't need an intermediary service such as Amazon API Gateway to directly invoke your function, which was required in the past. Just send an HTTP request to the unique URL of your Lambda function to get started.
- Function URL endpoints are publicly accessible by default and have the following format:
    - **https://<url-id>.lambda-url.<region>.on.aws**
- A Lambda Function URL can be created and configured via the AWS Lambda console or through the Lambda API.
- Upon creating a function URL, AWS Lambda automatically generates a unique URL endpoint for you that you can immediately use.
- This URL endpoint is static and doesn't change once created.
- Lambda URLs are dual stack-enabled, which support both IPv4 and IPv6 protocols
- The URL can be invoked via a web browser, CURL, Postman, or any HTTP client.
- There are 2 authentication types for controlling access to a Lambda function URL:
    - **AWS_IAM** – uses IAM to authenticate and authorize users. Only IAM users or roles that have been granted permission to invoke the function through IAM policies will be able to do so.
    - **NONE** – allows anyone who has the function URL to execute the Lambda function whether they have an AWS account or not.
- You can access your function URL through the public Internet only and not via AWS PrivateLink (e.g., VPC Endpoints)
- Uses resource-based policies for security and access control. You can further secure your function URL by enabling cross-origin resource sharing (CORS) to whitelist origins permitted to invoke it.
- A function URL can be applied to any Lambda function alias or to the **$LATEST** unpublished function version but not to any other function version.

Configuring a Lambda Function to Access Resources in a VPC

In AWS Lambda, you can set up your function to establish a connection to your virtual private cloud (VPC). With this connection, your function can access the private resources of your VPC during execution like EC2, RDS and many others.

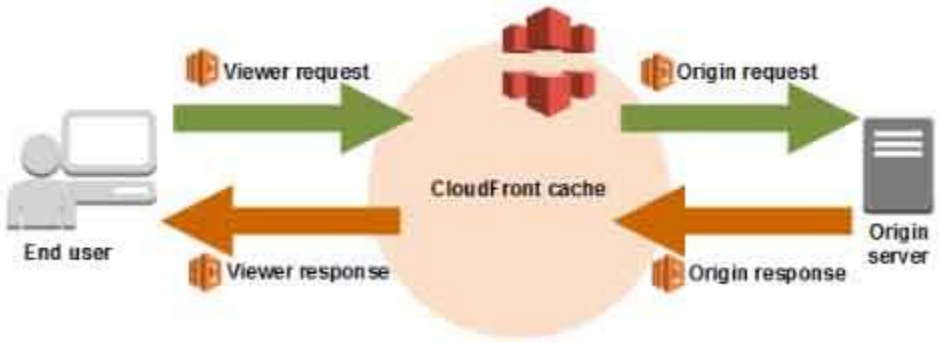
By default, AWS executes your Lambda function code securely within a VPC. Alternatively, you can enable your Lambda function to access resources inside your private VPC by providing additional VPC-specific configuration information such as VPC subnet IDs and security group IDs. It uses this information to set up elastic network interfaces which enable your Lambda function to connect securely to other resources within your VPC.



Lambda@Edge

- Lets you run Lambda functions to customize content that CloudFront delivers, executing the functions in AWS locations closer to the viewer. The functions run in response to CloudFront events, without provisioning or managing servers.
- You can use Lambda functions to change CloudFront requests and responses at the following points:

    - After CloudFront receives a request from a viewer (viewer request)
    - Before CloudFront forwards the request to the origin (origin request)
    - After CloudFront receives the response from the origin (origin response)
    - Before CloudFront forwards the response to the viewer (viewer response)



- You can automate your serverless application's release process using AWS CodePipeline and AWS CodeDeploy.
- Lambda will automatically track the behavior of your Lambda function invocations and provide feedback that you can monitor. In addition, it provides metrics that allows you to analyze the full function invocation spectrum, including event source integration and whether downstream resources perform as expected.

AWS Lambda SnapStart

- Lambda SnapStart speeds up your Java applications by reusing a single initialized snapshot to quickly resume multiple execution environments.
- You can use the Lambda SnapStart for Java feature to decrease the cold start time required without provisioning additional resources. This also removes the burden of implementing complex performance optimizations for your Java application

AWS Lambda Pricing

- You are charged based on the total number of requests for your functions and the duration, the time it takes for your code to execute.