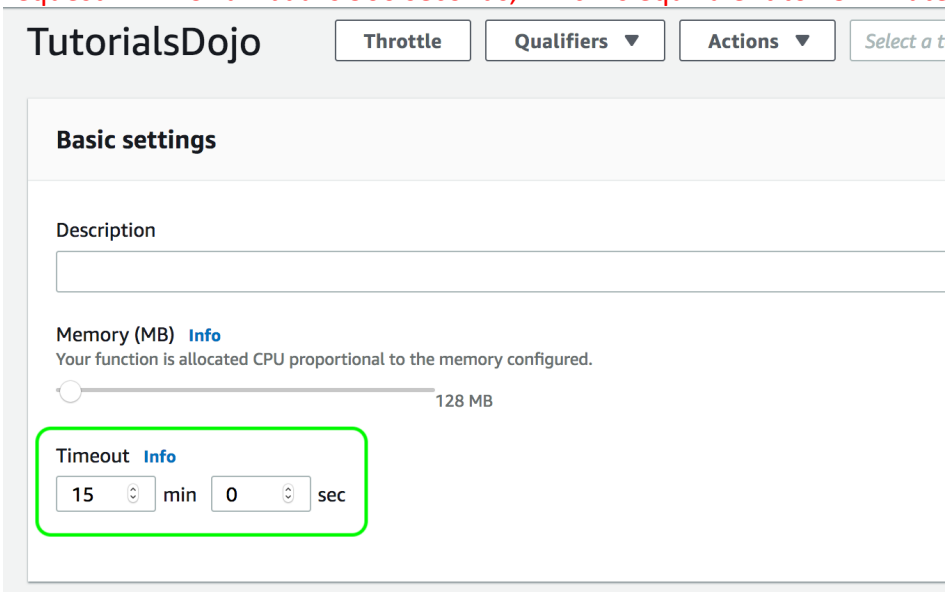
Lambda

- A Lambda function consists of code and any associated dependencies.
- In addition, a Lambda function also has configuration information associated with it.
- Initially, you specify the configuration information when you create a Lambda function.
- Lambda provides an API for you to update some of the configuration data.
- You pay for the AWS resources that are used to run your Lambda function.
- To prevent your Lambda function from running indefinitely, you specify a **timeout**.
- When the specified timeout is reached, AWS Lambda terminates execution of your Lambda function.
- It is recommended that you set this value based on your expected execution time.
- The default timeout is 3 seconds and the maximum execution duration per request in AWS Lambda is 900 seconds, which is equivalent to 15 minutes.



NOTE:

- Take note that you can invoke a Lambda function synchronously either by calling the Invoke operation or by using an AWS SDK in your preferred runtime.
- If you anticipate a long-running Lambda function, your client may time out before function execution completes.
- To avoid this, update the client timeout or your SDK configuration.

<table>
<tr>
<td>

Lambda

- AWS compute service that runs code without servers
- Runs code only when needed
- Scales automatically
  - Up to thousands of requests per second
- Billed by compute time

</td>
<td>

Languages Supported

- Node.js
- Java
- C#
- Go
- Python

</td>
</tr>
<tr>
<td>

Lambda Use Process

1. Customer builds the code
2. Customer launches th code as Lambda function
3. AWS selects server
4. Customer calls Lambd function as needed from applications

</td>
<td>

Reference:

https://docs.aws.amazon.com/lambda/latest/dg/limits.html

https://docs.aws.amazon.com/lambda/latest/dg/resource-model.html

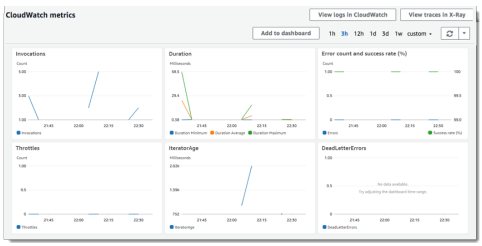
Check out this AWS Lambda Cheat Sheet:

https://tutorialsdojo.com/aws-cheat-sheet-aws-lambda/

</td>
</tr>
</table>

Lambda Metrics Monitoring:

AWS Lambda automatically monitors functions on your behalf, reporting metrics through Amazon CloudWatch.



Lambda monitoring graphs

- **Invocations** – The number of times that the function was invoked in each 5-minute period.
- **Duration** – The average, minimum, and maximum execution times.
- **Error count and success rate (%)** – The number of errors and the percentage of executions that completed without error.
- **Throttles** – The number of times that execution failed due to concurrency limits.

	• **IteratorAge** – For stream event sources, the age of the last item in the batch when Lambda received it and invoked the function. • **Async delivery failures** – The number of errors that occurred when Lambda attempted to write to a destination or dead-letter queue. • **Concurrent executions** – The number of function instances that are processing events.

You can monitor metrics for Lambda and view logs by using the Lambda console, the CloudWatch console, the AWS CLI, or the CloudWatch API.

To view metrics in the CloudWatch console

- Open the Amazon CloudWatch console Metrics page (AWS/Lambda namespace).
- Choose a dimension.
  - **By Function Name** (FunctionName) – View aggregate metrics for all versions and aliases of a function.
  - **By Resource** (Resource) – View metrics for a version or alias of a function.
  - **By Executed Version** (ExecutedVersion) – View metrics for a combination of alias and version. Use the ExecutedVersion dimension to compare error rates for two versions of a function that are both targets of a weighted alias.
  - **Across All Functions** (none) – View aggregate metrics for all functions in the current AWS Region.
- Choose metrics to add them to the graph

Function Metrics

- Invocation Metrics
- Performance Metrics
- Concurrency Metrics

https://docs.aws.amazon.com/lambda/latest/dg/monitoring-metrics.html

Using invocation metrics

- Invocation metrics are binary indicators of the outcome of an invocation.
- For example,
  - if the function returns an error, Lambda sends the Errors metric with a value of 1.
  - To get a count of the number of function errors that occurred each minute, view the Sum of the Errors metric with a period of one minute.

You should view the following metrics with the Sum statistic.

- **Invocation metrics**

- Invocations – The number of times your function code is executed, including successful executions and executions that result in a function error. Invocations aren't recorded if the invocation request is throttled or otherwise resulted in an invocation error. This equals the number of requests billed.
- Errors – The number of invocations that result in a function error. Function errors include exceptions thrown by your code and exceptions thrown by the Lambda runtime. The runtime returns errors for issues such as timeouts and configuration errors. To calculate the error rate, divide the value of Errors by the value of Invocations.
- DeadLetterErrors – For asynchronous invocation, the number of times Lambda attempts to send an event to a dead-letter queue but fails. Dead-letter errors can occur due to permissions errors, misconfigured resources, or size limits.
- DestinationDeliveryFailures – For asynchronous invocation, the number of times Lambda attempts to send an event to a destination but fails. Delivery errors can occur due to permissions errors, misconfigured resources, or size limits.
- Throttles – The number of invocation requests that are throttled. When all function instances are processing requests and no concurrency is available to scale up, Lambda rejects additional requests with TooManyRequestsException. Throttled requests and other invocation errors don't count as Invocations or Errors.
- ProvisionedConcurrencyInvocations – The number of times your function code is executed on provisioned concurrency.
- ProvisionedConcurrencySpilloverInvocations – The number of times your function code is executed on standard concurrency when all provisioned concurrency is in use.

Using performance metrics

- Performance metrics provide performance details about a single invocation. For example, the Duration metric indicates the amount of time in milliseconds that your function spends processing an event. To get a sense of how fast your function processes events, view these metrics with the Average or Max statistic.
- **Performance metrics**
  - Duration – The amount of time that your function code spends processing an event. For the first event processed by an instance of your function, this includes initialization time. The billed duration for an invocation is the value of Duration rounded up to the nearest 100 milliseconds.
  - IteratorAge – For event source mappings that read from streams, the age of the last record in the event. The age is the amount of time between when the stream receives the record and when the event source mapping sends the event to the function.

Duration also supports percentile statistics. Use percentiles to exclude outlier values that skew average and maximum statistics. For example, the P95 statistic shows the maximum duration of 95 percent of executions, excluding the slowest 5 percent.

Using concurrency metrics

Lambda reports concurrency metrics as an aggregate count of the number of instances processing events across a function, version, alias, or AWS Region. To see how close you are to hitting concurrency limits, view these metrics with the Max statistic.

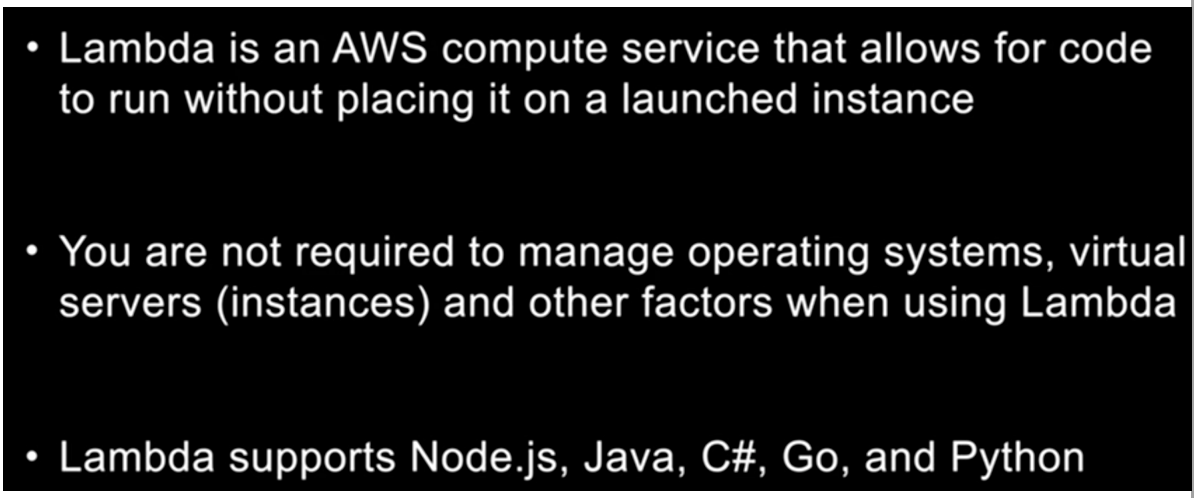
Concurrency metrics

- ConcurrentExecutions – The number of function instances that are processing events. If this number reaches your concurrent executions quota for the Region, or the reserved concurrency limit that you configured on the function, additional invocation requests are throttled.
- ProvisionedConcurrentExecutions – The number of function instances that are processing events on provisioned concurrency. For each invocation of an alias or version with provisioned concurrency, Lambda emits the current count.
- ProvisionedConcurrencyUtilization – For a version or alias, the value of ProvisionedConcurrentExecutions divided by the total amount of provisioned concurrency allocated. For example, .5 indicates that 50 percent of allocated provisioned concurrency is in use.
- UnreservedConcurrentExecutions – For an AWS Region, the number of events that are being processed by functions that don't have reserved concurrency.

References:

https://docs.aws.amazon.com/lambda/latest/dg/monitoring-functions-access-metrics.html

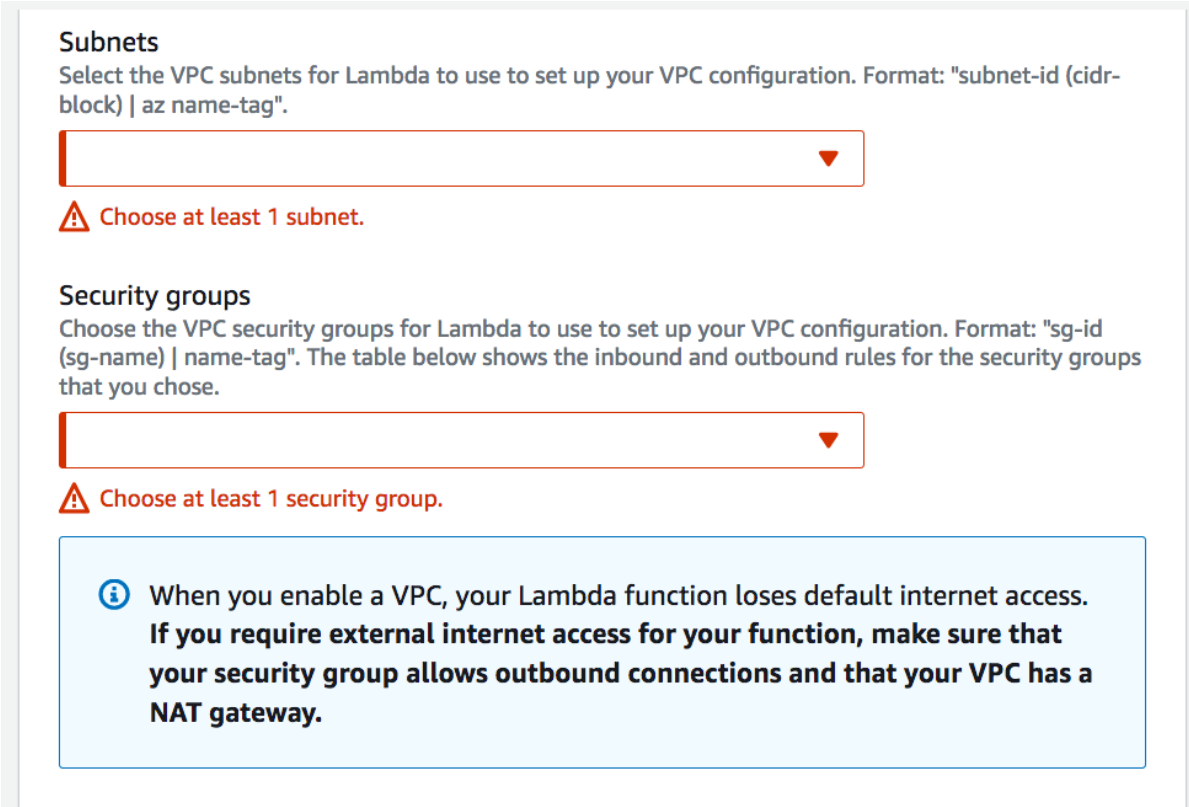
https://docs.aws.amazon.com/lambda/latest/dg/monitoring-functions-metrics.html



By default, Lambda runs your functions in a secure VPC with access to AWS services and the Internet. The VPC is owned by Lambda and does not connect to your account's default

VPC. When you connect a function to a VPC in your account, it does not have access to the Internet unless your VPC provides access.

AWS Lambda uses the VPC information you provide to set up ENIs that allow your Lambda function to access VPC resources. Each ENI is assigned a private IP address from the IP address range within the subnets you specify, but is not assigned any public IP addresses.



Therefore, if your Lambda function requires Internet access (for example, to access AWS services that don't have VPC endpoints), you can configure a NAT instance inside your VPC or you can use the Amazon VPC NAT gateway.

For more information, see NAT Gateways in the Amazon VPC User Guide. You cannot use an Internet gateway attached to your VPC, since that requires the ENI to have public IP addresses.

If your Lambda function needs Internet access, do not attach it to a public subnet or to a private subnet without Internet access. Instead, attach it only to private subnets with Internet access through a NAT instance or an Amazon VPC NAT gateway. You should also ensure that the associated security group of the Lambda function allows outbound connections. Hence, the correct answers in this scenario are ***ensuring that the associated security group of the Lambda function allows outbound connections*** *and* ***adding a NAT gateway to your VPC.***

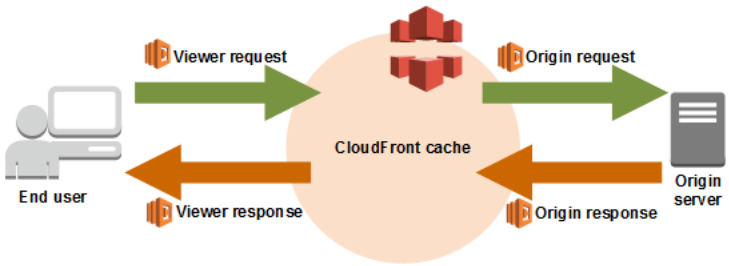
AWS Knowledge Center Videos: How do I give internet access to my Lambda function in a VPC?

https://www.youtube.com/watch?v=JcRKdEP94jM&feature=youtu.be

Lamda@Edge

- Lambda@Edge is a feature of Amazon CloudFront that lets you run code closer to users of your application, which improves performance and reduces latency.
- With Lambda@Edge, you don't have to provision or manage infrastructure in multiple locations around the world.
- You pay only for the compute time you consume - there is no charge when your code is not running.

- 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)



- With Lambda@Edge, you can enrich your web applications by making them globally distributed and improving their performance — all with zero server administration.
- Lambda@Edge runs your code in response to events generated by the Amazon CloudFront content delivery network (CDN).
- Just upload your code to AWS Lambda, which takes care of everything required to run and scale your code with high availability at an AWS location closest to your end user.

- By using Lambda@Edge and Kinesis together, you can process real-time streaming data so that you can track and analyze globally-distributed user activity on your website and mobile applications, including clickstream analysis

References:

https://aws.amazon.com/lambda/edge/

https://aws.amazon.com/blogs/networking-and-content-delivery/global-data-ingestion-with-amazon-cloudfront-and-lambdaedge/