

# Quarkus - Amazon SES Client

Amazon Simple Email Service (SES) is a flexible and highly-scalable email sending and receiving service. Using SES, you can send emails with any type of correspondence. You can find more information about SES at [the Amazon SES website](#).



The SES extension is based on [AWS Java SDK 2.x](#). It's a major rewrite of the 1.x code base that offers two programming models (Blocking & Async).

This technology is considered preview.



In *preview*, backward compatibility and presence in the ecosystem is not guaranteed. Specific improvements might require to change configuration or APIs and plans to become *stable* are under way. Feedback is welcome on our [mailing list](#) or as issues in our [GitHub issue tracker](#).

For a full list of possible extension statuses, check our [FAQ entry](#).

The Quarkus extension supports two programming models:

- Blocking access using URL Connection HTTP client (by default) or the Apache HTTP Client
- [Asynchronous programming](#) based on JDK's [CompletableFuture](#) objects and the Netty HTTP client.

In this guide, we see how you can get your REST services to use SES locally and on AWS.

## Prerequisites

To complete this guide, you need:

- JDK 1.8+ installed with [JAVA\\_HOME](#) configured appropriately
- an IDE
- Apache Maven 3.6.3
- An AWS Account to access the SES service
- Docker for your system to run SES locally for testing purposes

## Set up SES locally

The easiest way to start working with SES is to run a local instance as a container. However, local instance of SES is only mocks the SES APIs without the actual email sending capabilities. You can still use it for this guide to verify an API communication or integration test purposes.

```
docker run --rm --name local-ses -p 8012:4579 -e SERVICES=ses -e
START_WEB=0 -d localstack/localstack:0.11.1
```

This starts a SES instance that is accessible on port **8012**.

Create an AWS profile for your local instance using AWS CLI:

```
$ aws configure --profile localstack
AWS Access Key ID [None]: test-key
AWS Secret Access Key [None]: test-secret
Default region name [None]: us-east-1
Default output format [None]:
```

## Using SES on your AWS account

Amazon applies certain restrictions to new Amazon SES accounts, mainly to prevent fraud and abuse. All new accounts are in the Amazon SES **sandbox**. All the features of the Amazon SES are still available while in sandbox, but a following restrictions applies: - You can send mail to verified email addresses and domains or to the [Amazon SES mailbox simulator](#) - You can only send mail from verified email addresses and domains - You can send a maximum of 1 message per second.

Going production, you'd need to get your account of the sandbox following the [Amazon procedure](#).

## Set up AWS SES

We assume you are going to use AWS SES sandbox for the sake of this guide. But before sending any email, you must verify sender and recipient email addresses using AWS CLI. You can use your personal email or any temporary email service available if you wish.

```
aws ses verify-email-identity --email-address
<sender@email.address>
aws ses verify-email-identity --email-address
<recipient@email.address>
```

Now, you need to open a mailboxes of those email addresses in order to follow confirmation procedure. Once email is approved you can use it in your application.

If you are using local SES you still need to verify email addresses, otherwise your send email in order to let local SES accepting your request. However, no emails to be send as it only mocks the service APIs.

```
aws ses verify-email-identity --email-address  
<sender@email.address> --profile localstack --endpoint  
-url=http://localhost:8012  
aws ses verify-email-identity --email-address  
<recipient@email.address> --profile localstack --endpoint  
-url=http://localhost:8012
```

## Solution

The application built here allows sending text emails to the recipients that are verified on AWS SES.

We recommend that you follow the instructions in the next sections and create the application step by step. However, you can go right to the completed example.

Clone the Git repository: `git clone https://github.com/quarkusio/quarkus-quickstarts.git`, or download an [archive](#).

The solution is located in the `amazon-ses-quickstart` directory.

## Creating the Maven project

First, we need a new project. Create a new project with the following command:

```
mvn io.quarkus:quarkus-maven-plugin:1.7.0.CR1:create \  
-DprojectId=org.acme \  
-DprojectArtifactId=amazon-ses-quickstart \  
-DclassName="org.acme.ses.QuarkusSesSyncResource" \  
-Dpath="/sync" \  
-Dextensions="resteasy-jsonb,amazon-ses,resteasy-mutiny" \  
cd amazon-ses-quickstart
```

This command generates a Maven structure importing the RESTEasy/JAX-RS, Mutiny and Amazon SES Client extensions. After this, the `amazon-ses` extension has been added to your `pom.xml` as well as the Mutiny support for RESTEasy.

## Creating JSON REST service

Lets create a `org.acme.ses.QuarkusSesSyncResource` that will provide an API to send emails using the synchronous client.

```

package org.acme.ses;

import javax.inject.Inject;
import javax.ws.rs.Consumes;
import javax.ws.rs.POST;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;
import org.acme.ses.model.Email;
import software.amazon.awssdk.services.ses.SesClient;

@Path("/sync")
@Produces(MediaType.TEXT_PLAIN)
@Consumes(MediaType.APPLICATION_JSON)
public class QuarkusSesSyncResource {

    @Inject
    SesClient ses;

    @POST
    @Path("/email")
    public String encrypt(Email data) {
        return ses.sendEmail(req -> req
            .source(data.getFrom())
            .destination(d -> d.toAddresses(data.getTo()))
            .message(msg -> msg
                .subject(sub -> sub.data(data.getSubject()))
                .body(b -> b.text(txt ->
txt.data(data.getBody())))).messageId();
    }
}

```

## Configuring SES clients

Both SES clients (sync and async) are configurable via the `application.properties` file that can be provided in the `src/main/resources` directory. Additionally, you need to add to the classpath a proper implementation of the sync client. By default the extension uses the URL connection HTTP client, so you need to add a URL connection client dependency to the `pom.xml` file:

```

<dependency>
  <groupId>software.amazon.awssdk</groupId>
  <artifactId>url-connection-client</artifactId>
</dependency>

```

If you want to use Apache HTTP client instead, configure it as follows:

```
quarkus.ses.sync-client.type=apache
```

And add the following dependency to the application `pom.xml`:

```
<dependency>
  <groupId>software.amazon.awssdk</groupId>
  <artifactId>apache-client</artifactId>
</dependency>
```

If you're going to use a local SES instance, configure it as follows:

```
quarkus.ses.endpoint-override=http://localhost:8012

quarkus.ses.aws.region=us-east-1
quarkus.ses.aws.credentials.type=static
quarkus.ses.aws.credentials.static-provider.access-key-id=test-key
quarkus.ses.aws.credentials.static-provider.secret-access-key=test-secret
```

- `quarkus.ses.aws.region` - It's required by the client, but since you're using a local SES instance use `us-east-1` as it's a default region of localstack's SES.
- `quarkus.ses.aws.credentials.type` - Set `static` credentials provider with any values for `access-key-id` and `secret-access-key`
- `quarkus.ses.endpoint-override` - Override the SES client to use a local instance instead of an AWS service

If you want to work with an AWS account, you can simply remove or comment out all Amazon SES related properties. By default, the SES client extension will use the `default` credentials provider chain that looks for credentials in this order:

- Java System Properties - `aws.accessKeyId` and `aws.secretAccessKey`
- Environment Variables - `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY`
- Credential profiles file at the default location (`~/.aws/credentials`) shared by all AWS SDKs and the AWS CLI
- Credentials delivered through the Amazon ECS if the `AWS_CONTAINER_CREDENTIALS_RELATIVE_URI` environment variable is set and the security manager has permission to access the variable,
- Instance profile credentials delivered through the Amazon EC2 metadata service

And the region from your AWS CLI profile will be used.

# Next steps

## Packaging

Packaging your application is as simple as `./mvnw clean package`. It can be run with `java -jar target/amazon-ses-quickstart-1.0-SNAPSHOT-runner.jar`.

With GraalVM installed, you can also create a native executable binary: `./mvnw clean package -Dnative`. Depending on your system, that will take some time.

## Going asynchronous

Thanks to the AWS SDK v2.x used by the Quarkus extension, you can use the asynchronous programming model out of the box.

Create a `org.acme.ses.QuarkusSesAsyncResource` REST resource that will be similar to our `QuarkusSesSyncResource` but using an asynchronous programming model.

```

package org.acme.ses;

import io.smallrye.mutiny.Uni;
import javax.inject.Inject;
import javax.ws.rs.Consumes;
import javax.ws.rs.POST;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;
import org.acme.ses.model.Email;
import software.amazon.awssdk.services.ses.SesAsyncClient;
import software.amazon.awssdk.services.ses.model.SendEmailResponse;

@Path("/async")
@Produces(MediaType.TEXT_PLAIN)
@Consumes(MediaType.APPLICATION_JSON)
public class QuarkusSesAsyncResource {

    @Inject
    SesAsyncClient ses;

    @POST
    @Path("/email")
    public Uni<String> encrypt(Email data) {
        return Uni.createFrom()
            .completionStage(
                ses.sendEmail(req -> req
                    .source(data.getFrom())
                    .destination(d -> d.toAddresses(data.getTo()))
                    .message(msg -> msg
                        .subject(sub ->
sub.data(data.getSubject()))
                            .body(b -> b.text(txt ->
txt.data(data.getBody()))))))
            .onItem().apply(SendEmailResponse::messageId);
    }
}


```


We create **Uni** instances from the **CompletionStage** objects returned by the asynchronous SES client, and then transform the emitted item.

And we need to add the Netty HTTP client dependency to the **pom.xml**:

```
<dependency>
  <groupId>software.amazon.awssdk</groupId>
  <artifactId>netty-nio-client</artifactId>
</dependency>
```

## Configuration Reference

 Configuration property fixed at build time - All other configuration properties are overridable at runtime

Configuration property	Type	Default
 <code>quarkus.ses.interceptors</code>  List of execution interceptors that will have access to read and modify the request and response objects as they are processed by the AWS SDK. The list should consists of class names which implements <code>software.amazon.awssdk.core.interceptor.ExecutionInterceptor</code> interface.	list of class name	
 <code>quarkus.ses.sync-client.type</code>  Type of the sync HTTP client implementation	url, apache	url
AWS SDK client configurations	Type	Default
<code>quarkus.ses.endpoint-override</code>  The endpoint URI with which the SDK should communicate. If not specified, an appropriate endpoint to be used for the given service and region.	URI	
<code>quarkus.ses.api-call-timeout</code>  The amount of time to allow the client to complete the execution of an API call. This timeout covers the entire client execution except for marshalling. This includes request handler execution, all HTTP requests including retries, unmarshalling, etc. This value should always be positive, if present.	Duration ?	
<code>quarkus.ses.api-call-attempt-timeout</code>  The amount of time to wait for the HTTP request to complete before giving up and timing out. This value should always be positive, if present.	Duration ?	
AWS services configurations	Type	Default



<p><code>quarkus.ses.aws.region</code></p> <p>An Amazon Web Services region that hosts the given service.</p> <p>It overrides region provider chain with static value of region with which the service client should communicate.</p> <p>If not set, region is retrieved via the default providers chain in the following order:</p> <ul style="list-style-type: none"> <li>• <code>aws.region</code> system property</li> <li>• <code>region</code> property from the profile file</li> <li>• Instance profile file</li> </ul> <p>See <code>software.amazon.awssdk.regions.Region</code> for available regions.</p>	Region	
--	--------	--

`quarkus.ses.aws.credentials.type`

Configure the credentials provider that should be used to authenticate with AWS.




Available values:

- **default** - the provider will attempt to identify the credentials automatically using the following checks:
  - Java System Properties - `aws.accessKeyId` and `aws.secretKey`
  - Environment Variables - `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY`
  - Credential profiles file at the default location (`~/.aws/credentials`) shared by all AWS SDKs and the AWS CLI
  - Credentials delivered through the Amazon EC2 container service if `AWS_CONTAINER_CREDENTIALS_RELATIVE_URI` environment variable is set and security manager has permission to access the variable.
  - Instance profile credentials delivered through the Amazon EC2 metadata service
- **static** - the provider that uses the access key and secret access key specified in the `static-provider` section of the config.
- **system-property** - it loads credentials from the `aws.accessKeyId`, `aws.secretAccessKey` and `aws.sessionToken` system properties.
- **env-variable** - it loads credentials from the `AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY` and `AWS_SESSION_TOKEN` environment variables.
- **profile** - credentials are based on AWS configuration profiles. This loads credentials from a [profile file](#), allowing you to share multiple sets of AWS security credentials between different tools like the AWS SDK for Java and the AWS CLI.
- **container** - It loads credentials from a local metadata service. Containers currently supported by the AWS SDK are **Amazon Elastic Container Service (ECS)** and **AWS Greengrass**
- **instance-profile** - It loads credentials from the Amazon EC2 Instance Metadata Service.
- **process** - Credentials are loaded from an external process. This is used to support the `credential_process` setting in the profile credentials file. See [Sourcing Credentials From External Processes](#) for more information.
- **anonymous** - It always returns anonymous AWS credentials. Anonymous AWS credentials result in un-authenticated requests and will fail unless the resource or API's policy has been configured to specifically allow anonymous access.

default,  
static,  
system-  
prop-  
erty,  
env-  
variab-  
le,  
profil-  
e,  
contai-  
ner,  
instan-  
ce-  
profil-  
e,  
proces-  
s,  
anonym-  
ous

default

Default credentials provider configuration	Type	Default
<code>quarkus.ses.aws.credentials.default-provider.async-credential-update-enabled</code>  Whether this provider should fetch credentials asynchronously in the background. If this is <code>true</code> , threads are less likely to block, but additional resources are used to maintain the provider.	boolean	<code>false</code>
<code>quarkus.ses.aws.credentials.default-provider.reuse-last-provider-enabled</code>  Whether the provider should reuse the last successful credentials provider in the chain. Reusing the last successful credentials provider will typically return credentials faster than searching through the chain.	boolean	<code>true</code>
Static credentials provider configuration	Type	Default
<code>quarkus.ses.aws.credentials.static-provider.access-key-id</code>  AWS Access key id	string	
<code>quarkus.ses.aws.credentials.static-provider.secret-access-key</code>  AWS Secret access key	string	
AWS Profile credentials provider configuration	Type	Default
<code>quarkus.ses.aws.credentials.profile-provider.profile-name</code>  The name of the profile that should be used by this credentials provider. If not specified, the value in <code>AWS_PROFILE</code> environment variable or <code>aws.profile</code> system property is used and defaults to <code>default</code> name.	string	
Process credentials provider configuration	Type	Default
<code>quarkus.ses.aws.credentials.process-provider.async-credential-update-enabled</code>  Whether the provider should fetch credentials asynchronously in the background. If this is true, threads are less likely to block when credentials are loaded, but additional resources are used to maintain the provider.	boolean	<code>false</code>
<code>quarkus.ses.aws.credentials.process-provider.credential-refresh-threshold</code>  The amount of time between when the credentials expire and when the credentials should start to be refreshed. This allows the credentials to be refreshed <i>*before*</i> they are reported to expire.	Duration 	<code>15S</code>

<code>quarkus.ses.aws.credentials.process-provider.process-output-limit</code>  The maximum size of the output that can be returned by the external process before an exception is raised.	Memory Size 	1024
<code>quarkus.ses.aws.credentials.process-provider.command</code>  The command that should be executed to retrieve credentials.	string	
<b>Sync HTTP transport configurations</b>	<b>Type</b>	<b>Default</b>
<code>quarkus.ses.sync-client.connection-timeout</code>  The maximum amount of time to establish a connection before timing out.	Duration 	2S
<code>quarkus.ses.sync-client.socket-timeout</code>  The amount of time to wait for data to be transferred over an established, open connection before the connection is timed out.	Duration 	30S
<code>quarkus.ses.sync-client.tls-managers-provider.type</code>  TLS managers provider type.  Available providers: <ul style="list-style-type: none"> <li>• <b>none</b> - Use this provider if you don't want the client to present any certificates to the remote TLS host.</li> <li>• <b>system-property</b> - Provider checks the standard <code>javax.net.ssl.keyStore</code>, <code>javax.net.ssl.keyStorePassword</code>, and <code>javax.net.ssl.keyStoreType</code> properties defined by the <a href="#">JSSE</a>.</li> <li>• <b>file-store</b> - Provider that loads a the key store from a file.</li> </ul>	none, system-property, file-store	system-property
<code>quarkus.ses.sync-client.tls-managers-provider.file-store.path</code>  Path to the key store.	path	
<code>quarkus.ses.sync-client.tls-managers-provider.file-store.type</code>  Key store type. See the KeyStore section in the <a href="https://docs.oracle.com/javase/8/docs/technotes/guides/security/StandardNames.html#KeyStore[Java Cryptography Architecture Standard Algorithm Name Documentation]">https://docs.oracle.com/javase/8/docs/technotes/guides/security/StandardNames.html#KeyStore[Java Cryptography Architecture Standard Algorithm Name Documentation]</a> for information about standard keystore types.	string	

<code>quarkus.ses.sync-client.tls-managers-provider.file-store.password</code> Key store password	string	
<b>Apache HTTP client specific configurations</b>	<b>Type</b>	<b>Default</b>
<code>quarkus.ses.sync-client.apache.connection-acquisition-timeout</code> The amount of time to wait when acquiring a connection from the pool before giving up and timing out.	Duration ?	10S
<code>quarkus.ses.sync-client.apache.connection-max-idle-time</code> The maximum amount of time that a connection should be allowed to remain open while idle.	Duration ?	60S
<code>quarkus.ses.sync-client.apache.connection-time-to-live</code> The maximum amount of time that a connection should be allowed to remain open, regardless of usage frequency.	Duration ?	
<code>quarkus.ses.sync-client.apache.max-connections</code> The maximum number of connections allowed in the connection pool. Each built HTTP client has its own private connection pool.	int	50
<code>quarkus.ses.sync-client.apache.expect-continue-enabled</code> Whether the client should send an HTTP expect-continue handshake before each request.	boolean	true
<code>quarkus.ses.sync-client.apache.use-idle-connection-reaper</code> Whether the idle connections in the connection pool should be closed asynchronously. When enabled, connections left idling for longer than <code>quarkus..sync-client.connection-max-idle-time</code> will be closed. This will not close connections currently in use.	boolean	true
<code>quarkus.ses.sync-client.apache.proxy.enabled</code> Enable HTTP proxy	boolean	false

<code>quarkus.ses.sync-client.apache.proxy.endpoint</code>	URI	
The endpoint of the proxy server that the SDK should connect through. Currently, the endpoint is limited to a host and port. Any other URI components will result in an exception being raised.		
<code>quarkus.ses.sync-client.apache.proxy.username</code>	string	
The username to use when connecting through a proxy.		
<code>quarkus.ses.sync-client.apache.proxy.password</code>	string	
The password to use when connecting through a proxy.		
<code>quarkus.ses.sync-client.apache.proxy.ntlm-domain</code>	string	
For NTLM proxies - the Windows domain name to use when authenticating with the proxy.		
<code>quarkus.ses.sync-client.apache.proxy.ntlm-workstation</code>	string	
For NTLM proxies - the Windows workstation name to use when authenticating with the proxy.		
<code>quarkus.ses.sync-client.apache.proxy.preemptive-basic-authentication-enabled</code>	boolean	
Whether to attempt to authenticate preemptively against the proxy server using basic authentication.		
<code>quarkus.ses.sync-client.apache.proxy.non-proxy-hosts</code>	list of string	
The hosts that the client is allowed to access without going through the proxy.		
<b>Netty HTTP transport configurations</b>	<b>Type</b>	<b>Default</b>
<code>quarkus.ses.async-client.max-concurrency</code>	int	50
The maximum number of allowed concurrent requests. For HTTP/1.1 this is the same as max connections. For HTTP/2 the number of connections that will be used depends on the max streams allowed per connection.		
<code>quarkus.ses.async-client.max-pending-connection-acquires</code>	int	10000
The maximum number of pending acquires allowed. Once this exceeds, acquire tries will be failed.		

<code>quarkus.ses.async-client.read-timeout</code>	Duration ?	30S
The amount of time to wait for a read on a socket before an exception is thrown. Specify 0 to disable.		
<code>quarkus.ses.async-client.write-timeout</code>	Duration ?	30S
The amount of time to wait for a write on a socket before an exception is thrown. Specify 0 to disable.		
<code>quarkus.ses.async-client.connection-timeout</code>	Duration ?	10S
The amount of time to wait when initially establishing a connection before giving up and timing out.		
<code>quarkus.ses.async-client.connection-acquisition-timeout</code>	Duration ?	2S
The amount of time to wait when acquiring a connection from the pool before giving up and timing out.		
<code>quarkus.ses.async-client.connection-time-to-live</code>	Duration ?	
The maximum amount of time that a connection should be allowed to remain open, regardless of usage frequency.		
<code>quarkus.ses.async-client.connection-max-idle-time</code>	Duration ?	60S
The maximum amount of time that a connection should be allowed to remain open while idle. Currently has no effect if <code>quarkus..async-client.use-idle-connection-reaper</code> is false.		
<code>quarkus.ses.async-client.use-idle-connection-reaper</code>	boolean	true
Whether the idle connections in the connection pool should be closed. When enabled, connections left idling for longer than <code>quarkus..async-client.connection-max-idle-time</code> will be closed. This will not close connections currently in use.		
<code>quarkus.ses.async-client.protocol</code>	http1-1, http2	http1-1
The HTTP protocol to use.		
<code>quarkus.ses.async-client.ssl-provider</code>	jdk, openssl, openssl- refcnt	
The SSL Provider to be used in the Netty client. Default is <code>OPENSSL</code> if available, <code>JDK</code> otherwise.		

<code>quarkus.ses.async-client.http2.max-streams</code>  The maximum number of concurrent streams for an HTTP/2 connection. This setting is only respected when the HTTP/2 protocol is used.	long	4294967295
<code>quarkus.ses.async-client.http2.initial-window-size</code>  The initial window size for an HTTP/2 stream. This setting is only respected when the HTTP/2 protocol is used.	int	1048576
<code>quarkus.ses.async-client.http2.health-check-ping-period</code>  Sets the period that the Netty client will send <b>PING</b> frames to the remote endpoint to check the health of the connection. To disable this feature, set a duration of 0. This setting is only respected when the HTTP/2 protocol is used.	Duration ?	5
<code>quarkus.ses.async-client.proxy.enabled</code>  Enable HTTP proxy.	boolean	false
<code>quarkus.ses.async-client.proxy.endpoint</code>  The endpoint of the proxy server that the SDK should connect through. Currently, the endpoint is limited to a host and port. Any other URI components will result in an exception being raised.	URI	
<code>quarkus.ses.async-client.proxy.non-proxy-hosts</code>  The hosts that the client is allowed to access without going through the proxy.	list of string	
<code>quarkus.ses.async-client.tls-managers-provider.type</code>  TLS managers provider type.  Available providers: <ul style="list-style-type: none"> <li>• <b>none</b> - Use this provider if you don't want the client to present any certificates to the remote TLS host.</li> <li>• <b>system-property</b> - Provider checks the standard <code>javax.net.ssl.keyStore</code>, <code>javax.net.ssl.keyStorePassword</code>, and <code>javax.net.ssl.keyStoreType</code> properties defined by the <a href="#">JSSE</a>.</li> <li>• <b>file-store</b> - Provider that loads a the key store from a file.</li> </ul>	none, system-property, file-store	system-property
<code>quarkus.ses.async-client.tls-managers-provider.file-store.path</code>  Path to the key store.	path	



<code>quarkus.ses.async-client.tls-managers-provider.file-store.type</code>		
Key store type. See the KeyStore section in the <a href="https://docs.oracle.com/javase/8/docs/technotes/guides/security/StandardNames.html#KeyStore[Java Cryptography Architecture Standard Algorithm Name Documentation]">https://docs.oracle.com/javase/8/docs/technotes/guides/security/StandardNames.html#KeyStore[Java Cryptography Architecture Standard Algorithm Name Documentation]</a> for information about standard keystore types.	string	
<code>quarkus.ses.async-client.tls-managers-provider.file-store.password</code>		
Key store password	string	
<code>quarkus.ses.async-client.event-loop.override</code>		
Enable the custom configuration of the Netty event loop group.	boolean	<code>false</code>
<code>quarkus.ses.async-client.event-loop.number-of-threads</code>		
Number of threads to use for the event loop group. If not set, the default Netty thread count is used (which is double the number of available processors unless the <code>io.netty.eventLoopThreads</code> system property is set.	int	
<code>quarkus.ses.async-client.event-loop.thread-name-prefix</code>		
The thread name prefix for threads created by this thread factory used by event loop group. The prefix will be appended with a number unique to the thread factory and a number unique to the thread. If not specified it defaults to <code>aws-java-sdk-NettyEventLoop</code>	string	



#### About the Duration format

The format for durations uses the standard `java.time.Duration` format. You can learn more about it in the [Duration#parse\(\) javadoc](#).

You can also provide duration values starting with a number. In this case, if the value consists only of a number, the converter treats the value as seconds. Otherwise, `PT` is implicitly prepended to the value to obtain a standard `java.time.Duration` format.



#### About the MemorySize format

A size configuration option recognises string in this format (shown as a regular expression): `[0-9]+[KkMmGgTtPpEeZzYy]?`. If no suffix is given, assume bytes.