# DTCC Cloud Logger – Lambda Edition (aws-log-utils-lambda)

## Disclaimer

This logger is built for AD, and by AD. Please give us feedback as to how this may be improved. We’re very open to suggestions.

## What does the logger do?

The DTCC cloud logger is an enhanced log4j1 logger suitable for aws lambda functions. Though Lambda supports Log4j1 natively, its functionality is somewhat limited and the cloud logger aims to provide the following enhancements:

* **Very light configuration.** A pre-configured properties file is supplied with the library. It may be used as-is or modified to suit your needs, especially if you want to change the layout of your messages.
* **Better organization of log groups.** Groups are no longer organized using the standard *“/aws/lambda/”* tree, but rather by application. This way if your application is logically made up or several Lambda function all logs can be written to same group allowing for correlation across multiple functions.
* **Enhanced standard layout:** The standard layout that ships with the library pre-loads the MDC (mapped diagnostic context) with useful information that is automatically included with each event. Additionally, you can load information specific to your application and it will be automatically included too.
* **Splunk compatible layout:** If you plan on streaming logs to Splunk, you can simply create an instance of the Splunk logger that ships with the library and you’re ready to go. All log events will be formatted properly.
* **Ability to create incidents:** A special incident logger allows you to easily create incidents and publish them to DTCC monitoring systems. No configuration or setup is needed. Please note that the underlining incident infrastructure is still in development.
* **Resiliency (coming soon):** Future version of the logger will incorporate resiliency features and will provide multiple paths to logging, especially when it comes to creating incidents.
* **Fully compatible with Lambda’s native logger:** If you prefer to use Lambda’s native Appender, it comes preconfigured in the log4j properties file.
* **Easily enable aws request or wire tracing information:** These settings are present and turned off by default in the log4j properties file. Turn them on and watch your brain melt.

## Why Log4J1?

Lambda currently supports Log4j1; such is life. The ECD team also has a Log4j2 edition of this logger suitable for any other environment (anything EC2 or Linux based). When Lambda supports the latest framework, the two will be merged into a single super-duper logger.

## How does the logger work?

At its core the logger is collection of custom Appenders and Layouts with a static initializer to inject a CloudWatch client into the Appenders. You can either rely on the logger to create its own client or supply one yourself if you are running things locally. The ECD team has libraries that can help you make authentication easy, and those are described in this document.

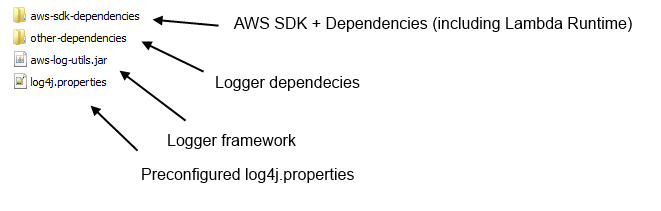
## Where can I find the logger?

The Logger will be available in Maven real soon. In the meantime, jar files are being distributed by the EDC team. Please contact Mark Hanegraaff for any help.

## How do I use install Lambda Logger?

Note that once available in Maven, you will simply include it as a dependency in your POM file

1. First, let’s see what’s inside the zip file



1. Copy all jars to your library folder. I separated the SDK jars, so you if you already use it in your project you may skip these. You will need to make sure the CloudWatch SDK jars are present.
2. Copy **log4j.properties** to your **/src/main/resources/** folder. This is a Lambda reqirement.

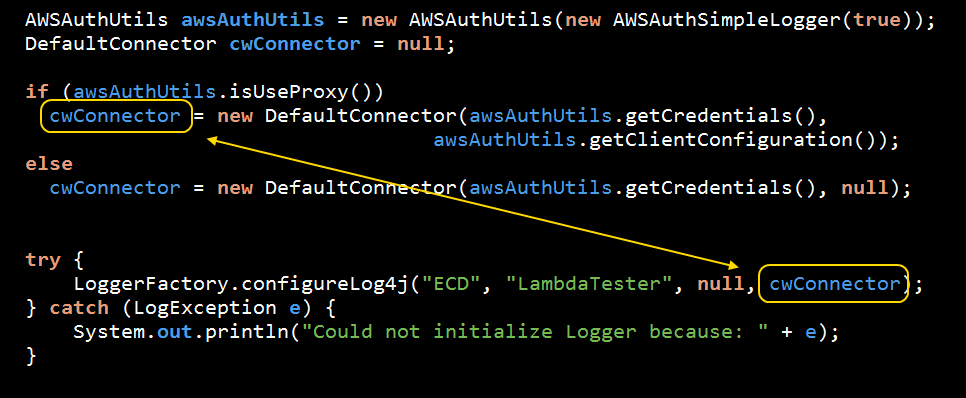
## How do I use the logger?

1. **Initialize the logger using the static initializer**. This only needs to be done once.



1a) If you are running Lambda inside a VPC you may need to access AWS resources using a proxy, in which case you can use the **CloudWatchClientUtils** utility class to supply an optional **ClientConfiguration** object with your proxy settings. Otherwise the logger will sort it out for you.

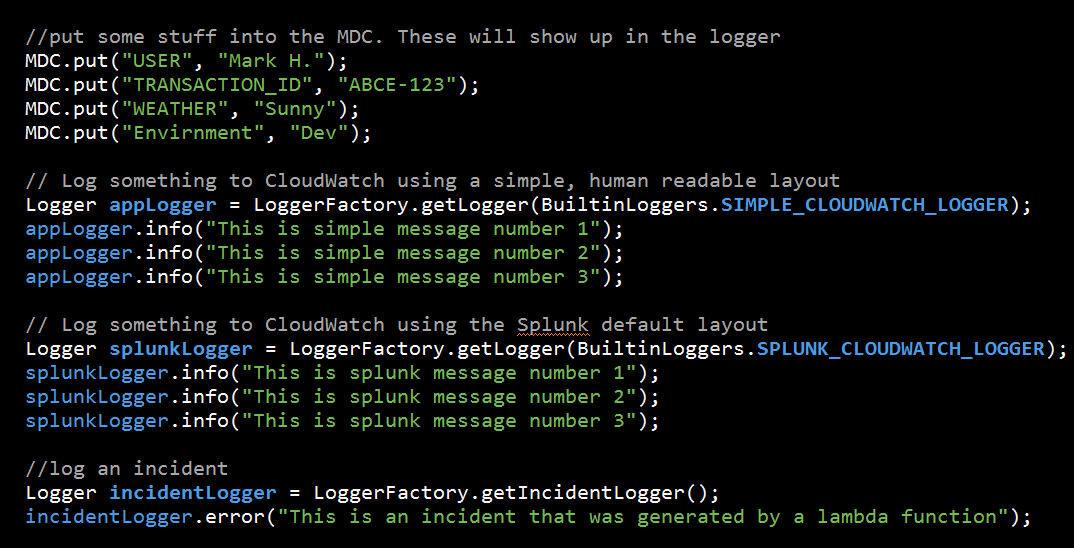
1b) If you are running locally then you will need to supply an appropriate client. I recommend using ECD’s temporary credentials library to do so (jar is already included). The code will look something like this:



Here is a link on how to use the temp credentials utilities:

[http://share.dtcc.com/projects/ecd/Pages/home.aspx?RootFolder=%2Fprojects%2FECD%2FDocuments%2FDevUtilities&FolderCTID=0x012000FA62C61CA52E0845BD2C0F41D803B08A&View={2F0BF55A-F453-4D8D-BC75-7309E49FA274}](http://share.dtcc.com/projects/ecd/Pages/home.aspx?RootFolder=%2Fprojects%2FECD%2FDocuments%2FDevUtilities&FolderCTID=0x012000FA62C61CA52E0845BD2C0F41D803B08A&View=%7b2F0BF55A-F453-4D8D-BC75-7309E49FA274%7d)

1. **Get the appropriate Logger and use it**. Here is an example



## General Tips:

1. The Logger object returned by the LoggerFactory is a just a plain Log4j1 logger. There are no restrictions on how to use it.
2. The framework makes good use of the MDC (mapped diagnostic context). I encourage you to use it, as the built-in layouts make use of it. The framework also pre-loads it with useful info.
3. The incident logger is also a plain Log4J1 logger. It does not matter how you use it. The underlining Appender will know to route the message.