Sharing Configurations

In this lesson, you'll study how to share configurations among Lambda functions.

WE'LL COVER THE FOLLOWING

Problem with a single configuration

Related functions often need to share configurations, such as names of AWS resources or access keys to external services. In this course, you kept all the functions in a single SAM template and configured them using environment variables. Using a single template makes it easy to ensure that everything is configured consistently. This is great for simple examples but only works if an entire application fits into a single SAM template. There are several reasons why a single template might not be the best option.

A single CloudFormation stack can only create 200 resources. With SAM, that number is actually even lower, because SAM works as a CloudFormation transformation, so each `AWS::Serverless::Function` block translates into several resources. An application requiring more than 200 resources will need more than one template.

First of all, large YAML files get difficult to maintain and are difficult to read. If you wouldn't create a single source code file with 200 variables, don't create a single CloudFormation template with 200 resources just because you can. In the same way that you'd split a large piece of code into smaller, more focused source code files, think about breaking up CloudFormation templates so that

they are easy to understand and manage.

Larger organisations often have teams working on individual parts of an application, each with their own SAM templates or even groups of templates. For existing applications, parts might already be using some other deployment tools, so sometimes it's necessary to share configurations outside CloudFormation or to import it from a different resource.

Options for sharing configurations

If a single template is not enough, there are three main options for sharing a configuration:

- Nested stacks
- Stack exports
- A parameter store

Nested stacks

One of the 200 available resources in a stack can actually be a separate CloudFormation template describing an embedded application called, more technically, a *nested stack*. To use a nested stack, create a resource with the type `AWS::CloudFormation::Stack` and then point to a local SAM or CloudFormation template file using the `TemplateURL` property. You can then set the parameters of a nested stack or read out its outputs and use them to configure other nested stacks.

In order to configure applications consisting of several CloudFormation stacks, I usually create a separate template file just for configuration. This corresponds to the main entry file in a program. This template (usually `main.yaml`) contains the global application parameters, and it is just responsible for configuring modules as nested stacks. You used something very similar in Chapter 10 when importing an application from the Serverless Application Repository. Nested stacks allow you to do the same thing, just importing from a local file.

The benefit of maintaining configuration with parent and nested stacks is that it is very simple. The downside is that updating configuration requires redeploying the stack. A constraint of this approach is that all the modules need to be deployed together so that a parent stack can configure them all at

once.

Stack exports

If you need to share configuration across modules that must be deployed separately, for example, if they are created by different teams but everything is still managed by CloudFormation, consider exporting outputs from a stack. Include an `Export` field with an output of a stack, and CloudFormation will publish the value of that output in a registry where other CloudFormation templates can access it. In related stacks, use the `!ImportValue` function to read a property from the public registry. Similar to nested stacks, updating exported and imported values also requires redeployment. For more information on this option, see the *Exporting Stack Output Values* documentation page.

A parameter store

If you want to manage parts of an application using CloudFormation and parts using other tools, then the best option for sharing configuration is to use a parameter store. There are several AWS products for managing parameters, including the AWS Secrets Manager and AWS Systems Manager Parameter Store.

AWS Secrets Manager and AWS Systems Manager Parameter Store are similar in many aspects. For example, you can read values from both in CloudFormation (using a feature called *dynamic references* or using the AWS SDK). This gives you the choice between applying configuration during deployment or reading the parameter values during Lambda function execution, which will increase the overhead and the price of individual functions but allow you to change the configuration without redeployment. As the name suggests, the Secrets Manager is intended for credentials such as passwords and access keys, and stores them securely. However, you can also require encrypted storage with the Systems Manager Parameter store.

There are two big differences between the tools. The Secrets Manager has some advanced features specific to storing access credentials, such as specifying a key rotation period and generating random values. It also costs significantly more. The Secrets Manager charges $0.40 for storing a single secret each month, and $0.05 for each 10,000 API calls. Using the Systems Manager Parameter store is effectively free for standard parameters

Manager Parameter store is effectively free for standard parameters.

That is all!
In the next lesson, you have a section of ‘Interesting Experiments’.