User Workflows and Request Routing

In this lesson, you will learn why you have to change the user workflow and you will get familiar with request routing.

WE'LL COVER THE FOLLOWING

This chapter explains how to manage session data in serverless applications and how to reduce operational costs by moving user workflows and application assets out of Lambda functions.

In Chapter 9, you moved resource orchestration from Lambda functions into the AWS platform. This significantly reduced operational costs, but you can do

even better. In fact, you can halve the costs for running your application by moving another coordination piece out of Lambda, this time related to user workflows. In the process, you can also make the application much more robust and user-friendly.

Up until this chapter, you used Lambda functions to generate HTML code for browsers. For example, once a user has uploaded a file, S3 will redirect them to the `/confirm` URL. This API resource is connected to the `ConfirmUploadFunction`, which will generate a download link for the results. The function will actually send a full HTML page, including the download link, back to the client so their browser can display it. This is simple and convenient for explaining the basic concepts, but it is far from ideal. There are three big problems with the current state of the application, mostly caused by the implicit relationship between uploads and downloads.

Problems with the current application state

No feedback from the conversion process

The first problem is that there is no feedback from the conversion process to the client. You generate the result link in the upload confirmation page. S3 triggers the Lambda function for file conversions around the same time as when it redirects users to the confirmation page, so it's anybody's guess what will happen first. This is why it is suggested to wait a few seconds before clicking the link in the previous chapter. The result link must not appear before the result is ready, or someone might click it while the conversion is still running and think that the application is broken. If you knew the duration of the conversion process, you could delay displaying the link on the client-side. Without a good feedback mechanism, it's very difficult to guess the duration.

No error-handling

The second problem is that there is no error handling. If something explodes in the background, the clients will never know about it. If an image takes a long time to convert, users won't know whether the process has stopped due to an error or due to a timeout on the server, or whether it's still running.

Sharing an implicit session state

The third problem is that you have two Lambda functions that share an

The third problem is that you have two Lambda functions that share an implicit session state, represented by the uploaded file key.

`ConfirmUploadFunction` generates the download link based on the file key from the redirect request, configured in `ShowFormFunction`. This is not really secure, because someone might modify the URL parameters and try to access a different file. You used GUIDs, so it's reasonably difficult to guess key identifiers. You need to make this implicit state significantly more secure.

All three problems require a much more explicit workflow and some sensible way of keeping the session state. The typical solution in a traditional web application would be to handle the user session workflow in the middle application layer. But with serverless architectures, keeping the user session workflows in the application layer can create all sorts of subtle problems.

Request routing

Website clusters usually work by sending all requests from a single user to the same machine so that web servers can keep session state in memory. With Lambda, application developers do not control request routing, so sticky sessions are not possible. Requests from the same user might reach two different Lambda instances with different memory session states.

Request routing is one of the least intuitive aspects of serverless applications for people used to web servers. There are many nice tools that make it easy to repackage classic server web applications into Lambda functions. For example, aws-serverless-express makes it possible to deploy existing Express applications as Lambda functions with minimal modifications. But beware that if such applications are based on server-side sessions, this is just looking for trouble. The application might work perfectly fine during testing because the load might not reach the threshold for Lambda to start more than one instance. However, once real user traffic starts hitting multiple containers at the same time, the application will mysteriously experience problems.

In the next lesson, you will move the session state out of Lambda functions.