Moving Session State out of Lambda Functions

In this lesson, you will get familiar with resumable sessions and how to minimise coordination between different Lambda executions.

WE'LL COVER THE FOLLOWING

- Resumable sessions
- Minimise coordination

Regardless of the application architecture, the session state can't reside in Lambda functions; it has to go somewhere else.

The usual solution for fault-tolerant session state in three-tier applications would be to put it into some kind of distributed data grid. DynamoDB would fit well in this case. Each Lambda instance could read out the session state at the beginning of a request, and save it towards the end. This would make each function slower and increase the operational cost of the application significantly. It would also make the application more error-prone since a function might experience problems between updating the session state and returning the result to the client.

There is an alternative that makes the application significantly cheaper. In Chapter 8, you moved the gate-keeper responsibility out of the application business code layer to the platform. As a result, client code can talk directly to AWS resources using temporary access grants. Service resources (such as S3) can directly prevent the client from performing something it is not authorised to do. Keeping the workflow on the server-side is not really improving security. This means it is safe to move user workflows all the way to the client devices, and with them, move user session data as well. Instead of trying to make the back end stateful, you can keep the user state on the front end.

If the client code is under your control, with serverless applications it's usually best to move user workflows and sessions to the client layer. This can significantly reduce operational costs and increase overall performance. You do not have to pay for Lambda functions waiting on resources, polling and coordinating work. Client devices will take that responsibility and that cost. Although the overall application data will be distributed across many different client devices, the session state of each individual user will be in a single place (the client device). This approach makes it easy to avoid all sorts of problems typically related to distributed data, and ensure consistency.

If someone else controls the client code, for example if you are building a Slack integration or a Facebook messenger chatbot, put session state in DynamoDB.

Resumable sessions

A big limitation of moving session state to client devices is that unexpected problems on the front end can cause users to lose session information. If the browser crashes half-way through a workflow, the client will not be able to resume the session. For your image thumbnail application, this is not an issue. The client can just re-request another upload policy and try again.

If you want to create resumable sessions or let users access the same workflow from multiple browser tabs or even devices concurrently, then you'll need to synchronise session state somehow. In typical three-tier architectures, the solution for that would be to keep the session in the application server or in a database. With serverless applications, there are several ways of synchronising client sessions without the application layer:

- Amazon Cognito has its own synchronisation mechanism for a small amount of user data such as preferences. It's called Cognito Sync.
- For more complex objects, you can give clients direct access to a DynamoDB table, where each user is restricted to reading and writing only their state.
- For situations where different users need to share session state (for example in collaborative editing), use AWS AppSync. AWS AppSync is a managed hierarchical database intended for direct use by client devices, and it can automatically synchronise state across multiple clients, resolve conflicts, and even deal with offline usage scenarios.

Of course, all three approaches work well with Lambda functions, and you can add triggers to act on changes to data from the server side.

Minimise coordination

With the state on the client, you need to minimise the chatter between the client devices and network services, and the amount of coordination between different Lambda executions. There are two ways to achieve this:

- For tightly coupled tasks, aggregate processing so that different requests are independent.
- For loosely coupled tasks, send full context information with each request.

The two Lambda functions you created in the previous chapter are quite tightly coupled. In order to securely generate the download link, you need to know the file key from the upload policy. With such a tight coupling, it's better to aggregate the processing.

The `ShowFormFunction` Lambda function should create both the upload policy and the download signature. That is the easiest way to ensure that users can only download conversion results for their own files. That function does not, however, need to produce HTML code. There is no need to coordinate the formatting of a web form between the client device and a Lambda function. User interface formatting can safely move to client devices, without any security risks. You can simplify the Lambda function by making it return the security grants in a simple format that a client device can understand, for example as a JSON object. Software running on the client side can then create the appropriate user interface. This would also allow you to create a nicer front end by using a popular front-end framework such as React or Angular, without complicating the Lambda code.

Generating both security signatures in `ShowFormFunction` means that you can completely remove the `ConfirmUploadFunction` function and reduce the operational costs by avoiding one Lambda execution and one API call for each upload. This will almost halve the operational costs for the application. You will not use redirects after uploading any more, because the fact that the upload finished is not really that important. The client code needs to know when the conversion finished, closing the whole feedback loop, in order to show the download link only after the actual result is available.

Controlling the user sessions from the client-side will allow you to create a much better end-user experience and reduce operational costs. The downside is that the client code will need to be more complex, and you will need some way of deploying and managing web assets and client-side code in addition to Lambda code.

In the next lesson, you’ll move on to writing some code to move static assets out of Lambda functions.