Direct Access to External Storage

In this lesson, you will learn how to remove the gatekeeper role from the Lambda functions to get away from traditional architecture.

This chapter shows how to start shifting the core responsibilities of typical servers to the AWS platform, and how to benefit from serverless architectures to make your application faster and cheaper. You'll also learn about the options to let client devices directly use AWS resources on your behalf.

In the previous chapters, you deployed a simple application that could be ready for millions of users if necessary but doesn't cost anything if nobody is using it. AWS operates and monitors the application, so most of the 'ops' part of DevOps is effectively included in the price. This is quite a nice outcome for such little effort, mostly achieved because you didn't have to write infrastructural code. With Lambda, API Gateway, and S3, all that infrastructural stuff comes out of the box.

Three-tier server setup

Our application still looks very similar to a typical three-tier server setup. In such an architecture, an application server usually contains the business logic, but also plays two more roles. It works as a gatekeeper and a workflow orchestration system. The application server approves or rejects client requests, talks to back-end storage, schedules asynchronous processing and passes back results to client devices. For highly available and scalable web applications, the code running on a client device usually talks to the application server through a load-balancing proxy. Your API Gateway resources work pretty much like the load-balancing proxy. Your form processing Lambda function might not control a network socket, but it's very much a gatekeeper to the S3 storage.

Although the application is already relatively cheap and fast, you can make it faster and cheaper by taking away the traditional server roles from Lambda functions. Out of the three responsibilities of the traditional application server, only the business logic should stay in Lambda functions. In this chapter, you'll remove the gatekeeper role from Lambda functions. In Chapter 9, you'll look at how to remove the orchestration role.

Removing the gatekeeper role from Lambda functions

The goal of extracting the gatekeeper role from Lambda functions is to remove an unnecessary intermediary between client devices and network resources, such as the S3 storage. The client devices can write directly to S3. This can be quite controversial for people used to three-tier architectures. It's easy to think about storage such as file repositories or databases as back-end resources that need protection from client devices. But with AWS, services such as DynamoDB or S3 are not really in the back end. They are available directly over the internet, to you, your users, and everyone else. S3 won't trust a request coming from a Lambda function any more than a request coming directly from a client device.

Having a Lambda function sitting between a user device and S3 is not really useful if it only performs authorisation. S3 can do that itself. There's no need to suffer through the additional latency of API proxying and Lambda

execution. For large files, this can make a huge difference, both in terms of speed and in terms of processing costs.

AWS provides several ways of temporarily granting clients the right to do very specific operations:

- You could let clients use the AWS SDK directly and set up IAM users for each client.
- You could set up templated IAM policies for groups of end-users authenticated with Amazon Cognito, a managed service for user credentials.
- You could use a Lambda function to create temporary grants for clients, so they can access your AWS resources in a limited way.

Using IAM directly

The first option, using IAM directly, is great for internal applications and with a small number of named users. Each user can get their own specific privileges and sign in to the client application with an IAM username and password, and the client application can directly use AWS SDK to access storage, databases, or queues. Client devices can even invoke Lambda functions directly to perform business logic that you do not want to expose to the client application code. Users can have multi-factor authentication devices for increased security.

The limiting factor of IAM is that you can only create about 1000 users for an AWS account, and those users can't register directly for your application. An administrator would have to set up individual users. Of course, you could build a Lambda function that can create IAM users and manage policies, but that function would need full access to your IAM resources, and that's a huge security risk.

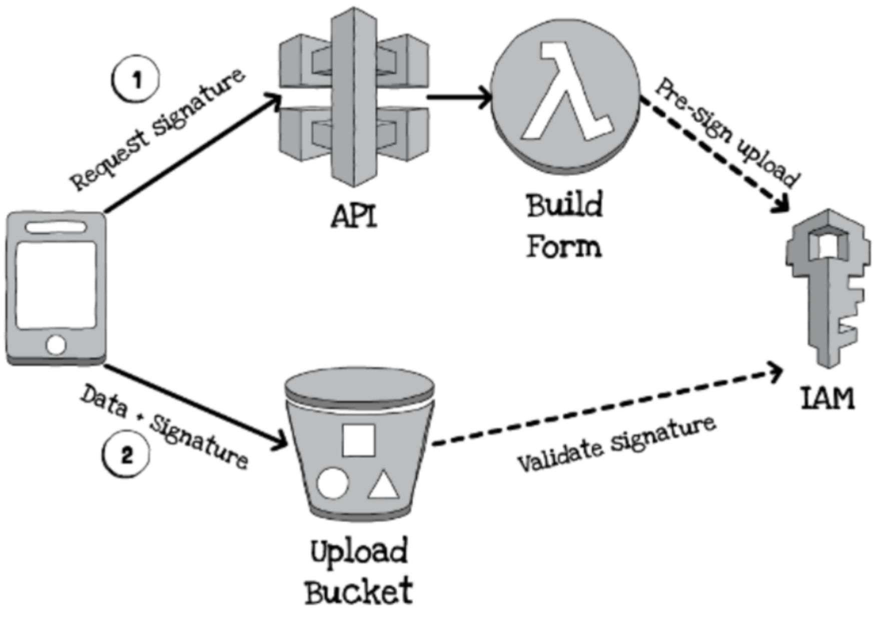
Using Cognito

The second option, using Cognito, is great for publicly facing applications. Cognito is a managed service for user data. It stores usernames and passwords securely, optionally allowing users to sign up themselves or even sign in through federated authentication systems (for example a company single sign-on or using Google accounts). You can configure Cognito to automatically allow end-users to have limited access to your AWS resources. For example, you could allow users to read files from an S3 bucket, optionally only those files with a prefix matching the account ID. Similarly, you could allow users to read or write only certain fields of a DynamoDB record matching their ID. IAM supports templates with Cognito identities, so you could achieve this with a single policy, without having to create and assign policies to individual users. Cognito can even manage user groups with separate security policies. For example, you could create groups for guests, authors, and moderators, with different access privileges.

Using a Lambda function to create temporary grants

The third option, using a Lambda function to create temporary grants, is great for anonymous access. This method allows you to safely pass on temporary access rights with the account credentials to client devices. It does not require the creation of any kind of user records or up-front registration. Temporary grants are also good for situations when templated policies are too restrictive, for example with Cognito users where access isn't easy to segment under a prefix matching the user identifier.

In this chapter, you'll change the form processing application to let users upload files directly to S3. You'll only use Lambda functions to authorise the requests. Since Lambda functions will no longer work as gatekeepers, the client requests will go through fewer intermediaries, execute faster, and the whole application will be cheaper to operate.



Lambda functions will authorise access to S3, so client code can upload and read files directly.

You’re ready to move on to the next lesson to see how you can implement direct access to external storage.