kschen@mit.edu

Using the AWS Serverless Application Model in the Indoor IoT Backend

What is serverless computing?

Serverless computing is a cloud computing model in which cloud providers allocate computing resources on-demand. In other words, the cloud providers take care of server needs so that their customers (typically developers) don't need to think about it. Thus, although the model is not "serverless" in that there are still servers present, the management of those servers is abstracted away from the customer.

What is the Serverless Application Model?

The Serverless Application Model (SAM) is an open-source framework created by AWS to build serverless applications with AWS resources (see Footnote 1). These serverless applications consist of resources such as Lambda functions, event sources, databases, and APIs that compose an application's infrastructure and allow it to perform tasks using the serverless computing model.

SAM allows developers to define serverless applications. This is primarily done through a SAM template specification. These specifications are YAML files that use the SAM template specification syntax to describe the resources used in your serverless application – functions, APIs, permissions, configurations, events, etc. The syntax is built on top of the AWS CloudFormation template specification syntax, and is compatible with CloudFormation definitions (see Footnote 2).

SAM also includes a command line interface (SAM CLI), which allows developers to validate templates, invoke Lambda functions locally, package and deploy applications locally and to the cloud, etc. on the command line.

Why should we use it?

Our current solution seems to be using the AWS browser console to configure and build individual resources. Although this approach is possible, the lack of versioning or any real deployment scheme, as well as the scatteredness of managing all the necessary AWS resources individually, will ultimately render it unsustainable, inefficient, and difficult to manage.

SAM allows us to define a serverless application as a single versioned and deployable entity, and operate on it as such. The SAM template provides a centralized source for infrastructure specifications for an application, defining the resources needed and relationships between the resources – this enforces compatibility and interoperability upfront. The SAM CLI allows us to easily build, package, and deploy the application as a whole without micromanaging across many different AWS resources. It also offers us the option to not only deploy to the cloud, but also locally, which is much more difficult to do by managing individual resources on the AWS console.

Finally, SAM provides the benefit of unifying our application into a single entity that can be interacted with programmatically, as well as through established software tools such as the command line and GitHub. For example, deployment across the whole stack can be done with a single terminal command, and both code and infrastructure can be modified and shared through a single GitHub repository.

How do we use it, and what do we need?
Currently, I've created a SAM template specification defining some of the basic infrastructure for our project. This is located in the "indoor-iot-backend" GitHub repo as *template.yaml* (see Footnote 3).

The template currently defines a DynamoDB database *IntermediateLocationsTable*, which takes in location updates from an updater; a DynamoDB database *LocalizedTagsTable*, which stores the localized tags' information which consumers can access; and a Lambda function *LocalizationTaskFunction*, which transforms *IntermediateLocationsTable* data to *LocalizedTagsTable* data.

To finish out creating the infrastructure for the project, we need a couple of things. One is to migrate the AppSync GraphQL endpoint from the console into our infrastructure. This can be done by defining the AppSync resources in the SAM template – as AWS CloudFormation includes syntax for AppSync resources, we can use it inside the template (see Footnote 4). After the AppSync endpoint is added, we can connect them to the DynamoDB databases in our template definition.

The other is to finalize the schemas of the information being stored inside *IntermediateLocationsTable* and *LocalizedTagsTable*, so that we can appropriately define the primary keys for both tables, then build the *LocalizationTaskFunction* logic to transform *IntermediateLocationsTable* data to *LocalizedTagsTable* data.

Once these two tasks are finished, the bulk of backend development will be complete.

Are there downsides to it?
One major downside to using SAM is that it ties our infrastructure to AWS. This makes it hard to migrate to other platforms, if required in the future. As a concept/model, SAM is not exactly fixed to AWS — it likely can be replicated with different technologies. That being said, it would be very very difficult to achieve the same level of cohesion that SAM provides, mainly because in terms of cloud compute resources, AWS is likely far ahead of competitors — it would probably be difficult to find another cloud platform that has so many different types of functionality, especially in a single system (for example, another platform that has all the equivalents of Lambdas, DynamoDB, etc.). Migrating would also likely lose many of the key benefits that come with SAM — namely 1) having the application be an atomic entity with regards to versioning and

deployment; 2) essentially one-click building and deployment; 3) ease of management/interoperability between components.

If migration were required, some main options would be:

1. Change the system architecture to adapt to a different model(s)/platform(s)
2. Approximate the current system by using similar/equivalent components on different platforms
3. Replicate the required technologies from scratch by implementing the components ourselves

The first two options are likely to involve many different platforms/technology, leading to infrastructure management headaches. The third option leaves us with the same benefits as SAM, but would likely take significant time, effort, and resources.

UPDATE: As of 1/28/2022, it seems like a viable way to further decouple the application infrastructure from AWS is to use the Serverless Framework (https://www.serverless.com/), which seems to allow users to choose which cloud provider to use. More research is needed to determine if and how this works.

kschen@mit.edu

Footnotes

1. For more information on AWS SAM, see https://docs.aws.amazon.com/serverless-application-model/latest/developerguide/what-is-sam.html.
2. For more information on AWS CloudFormation template syntax, see https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/template-reference.html.
3. To access the GitHub repo, see https://github.com/kevinsunchen/indoor-iot-backend.
4. For more information on the AWS AppSync resource type reference, see https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/AWS_AppSync.html.