

STACS SOLUTIONS ENGINEERING

Using AWS Lambda and Cognito to
implement a Scalable Authorisation and
Authentication Module



PREFACE

The Hashstacs Solutions Engineering team set out to implement an authentication and authorization module that would secure all application logins and API access in a secured, scalable and cost-effective manner. This module will also need to support SAML integration with Enterprise Identity Providers with the ability to support role-based access control to restrict what each user can see or access.

With the above requirements in mind, we decided to use AWS Cognito to handle the authentication while the authorization requires some custom work to implement which we will cover in the sections below.

CONTENTS

PREFACE.....	2
1. System Architecture.....	4
1.1. Authentication Microservice	4
1.2. Integration with Cognito.....	4
1.4 API Gateway and Lambda Authoriser	5
1.5 Roles and Permissions Database	6
1.6 Roles and Permissions Microservice.....	6
1.7 Roles and Permissions Microservice.....	7
1.8 Roles and Permissions Microservice.....	7
2. Technical Considerations	8

1. System Architecture

This module is a collection of microservices built in Java (Spring framework), integrated with AWS managed services. The architecture is shown in Fig 1 below.

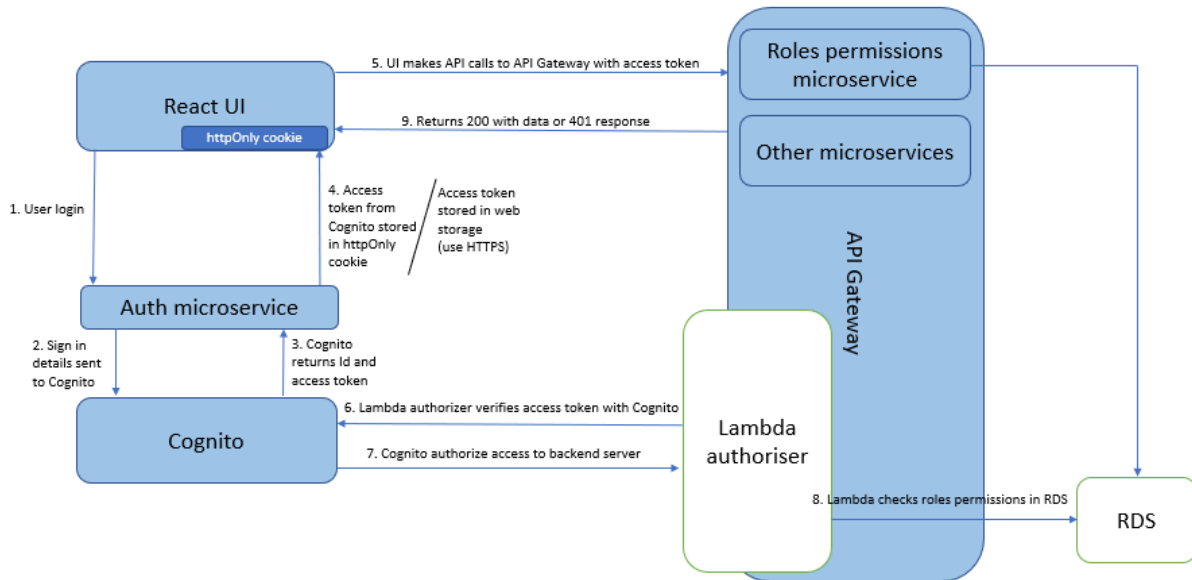


Figure 1: Architecture Diagram of Security module

1.1. Authentication Microservice

This microservice is deployed as an EC2 (Elastic Cloud Compute) instance that is managed via Elastic Beanstalk. It is a public-facing resource that uses the AWS Cognito SDK to handle the following actions:

- Account creation
- Account verification
- Sign in
- Reset password
- Refresh access token

1.2. Integration with Cognito

AWS Cognito is a service that controls user authentication, authorization and management for web and mobile applications. It also supports integration with third-party Security Assertion Markup Language (SAML), simplifying the onboarding process for users. Cognito enables us to offload the safe storage and management of user credentials to AWS.

The security module uses Cognito to authenticate users. Cognito issues an access token and a refresh token to the client when a user is successfully signed in. These two tokens are stored as a httpOnly cookie on the client browser, and every subsequent request from the client will carry the access token in the request header.

Cognito also natively handles the time-out of the refresh token which can be set from the Cognito console.

Which app clients will have access to this user pool?

The app clients that you add below will be given a unique ID and an optional secret key to access this user pool.

App client id

App client secret

Refresh token expiration (days)

Auth Flows Configuration

- Enable username password auth for admin APIs for authentication (ALLOW_ADMIN_USER_PASSWORD_AUTH) [Learn more.](#)
- Enable lambda trigger based custom authentication (ALLOW_CUSTOM_AUTH) [Learn more.](#)
- Enable username password based authentication (ALLOW_USER_PASSWORD_AUTH) [Learn more.](#)
- Enable SRP (secure remote password) protocol based authentication (ALLOW_USER_SRP_AUTH) [Learn more.](#)
- Enable refresh token based authentication (ALLOW_REFRESH_TOKEN_AUTH) [Learn more.](#)

Prevent User Existence Errors [Learn more.](#)

Legacy
 Enabled (Recommended)

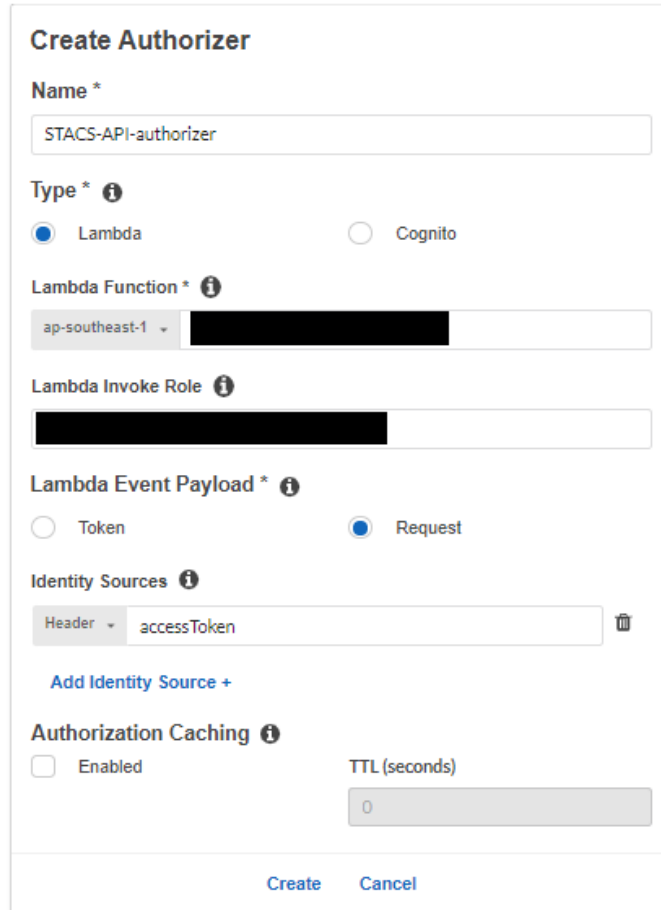
Figure 2: Setting up AWS Cognito app client.

1.4 API Gateway and Lambda Authoriser

The private APIs are deployed to AWS API Gateway, which is a fully managed scalable service that is able to handle concurrent API calls and manages traffic to and from our backend services. API Gateway also acts as a reverse proxy by routing the requests from client to the private APIs.

AWS Lambda is a serverless compute platform that is event-driven, which means that it is only executed when certain events take place. In the security module design, this Lambda function acts as a middleware to authenticate the access token with Cognito. Once the access token is verified, the Lambda function then sends a query to RDS to get a list of permissions for that particular user. If the requested resource is part of the list, the Lambda function will allow access to the resource specified in API Gateway.

Setup of the Lambda authorizer in API Gateway to enforce role-based access is easy to manage as well:



Create Authorizer

Name *
STACS-API-authorizer

Type * ⓘ
 Lambda Cognito

Lambda Function * ⓘ
 ap-southeast-1 [REDACTED]

Lambda Invoke Role ⓘ
 [REDACTED]

Lambda Event Payload * ⓘ
 Token Request

Identity Sources ⓘ
 Header accessToken [REDACTED] ⓘ

[Add Identity Source +](#)

Authorization Caching ⓘ
 Enabled TTL (seconds) [REDACTED]

[Create](#) [Cancel](#)

Figure 3: Setting up Lambda Authorizer in API Gateway

1.5 Roles and Permissions Database

The roles and permissions for each user is defined and stored persistently in an RDS schema.

The permissions defined in the database schema refers to the API resource that the user can access. The Lambda authorizer described in section 2.3 queries this database to get a list of permissions for the user that is sending the request

1.6 Roles and Permissions Microservice

This microservice handles actions related to user roles and permissions management. Similar to the authentication microservice described in section 2.1, it is also deployed as an EC2 instance and managed via Elastic Beanstalk. This microservice is a private resource, meaning that users will require an access token from Cognito to have access to it. The functions of this microservice are listed below:

- Fetch list of permissions for user
- Fetch list of roles for user
- Create role / permission
- Assign or remove role to user
- Assign or remove permission to role

1.7 Roles and Permissions Microservice

This microservice handles actions related to user roles and permissions management. Similar to the authentication microservice described in section 2.1, it is also deployed as an EC2 instance and managed via Elastic Beanstalk. This microservice is a private resource, meaning that users will require an access token from Cognito to have access to it. The functions of this microservice are listed below:

- Fetch list of permissions for user
- Fetch list of roles for user
- Create role / permission
- Assign or remove role to user
- Assign or remove permission to role

1.8 Roles and Permissions Microservice

The sequence diagram in Figure 4 shows an example of a user signing in and attempting to make a “createRole” request to the roles and permissions microservice.

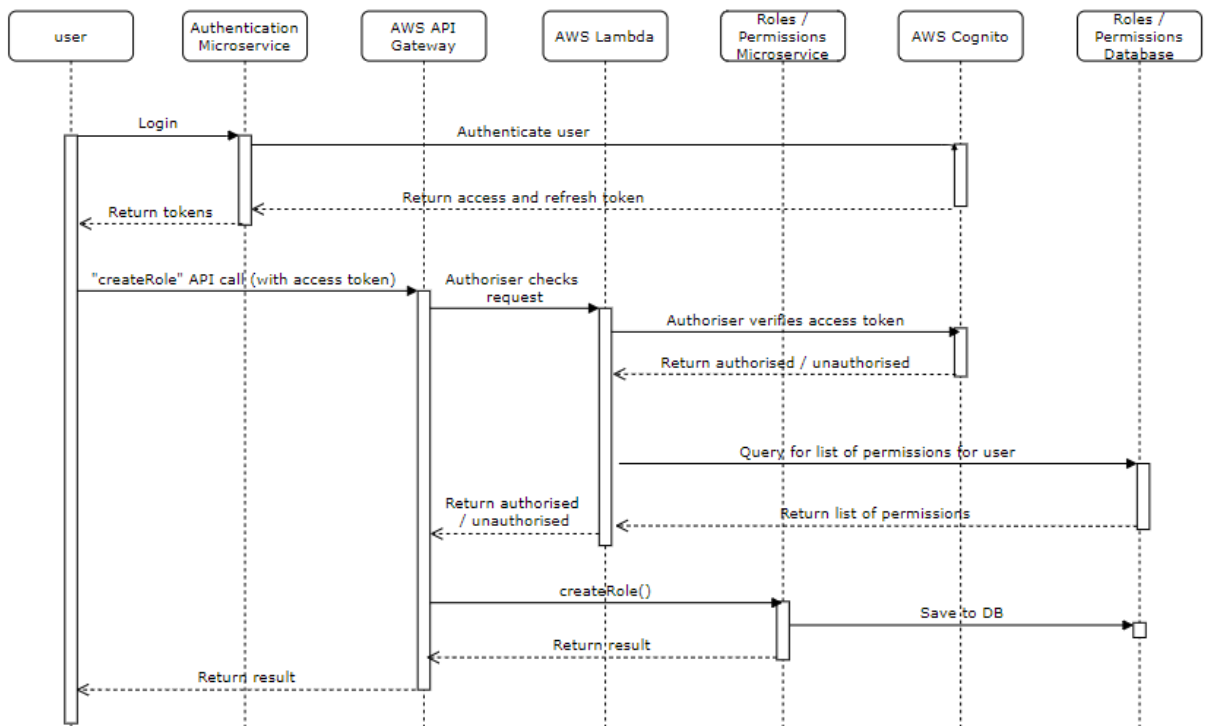


Figure 4: Sequence diagram of “createRole” request

2. Technical Considerations

The current design separates the authentication microservice from the roles and permissions microservice. The authentication microservice need to be accessible to the public due to the functions such as signing in and password reset, whereas the roles and permissions microservice needs to be secured as it interacts directly with the database. Since AWS API Gateway can define different permissions for different resources, these two microservices can potentially be simplified into a single codebase for easier maintenance.

AWS Cognito has a JavaScript library known as Amplify, which allows the client browser to interact with Cognito directly without having to go through the authentication microservice. This implementation eliminates the need for the authentication microservice as the Amplify library provides a set of functions to Cognito, simplifying the security architecture by introducing serverless authentication. However, it might require some of the authentication logic to be built on the front-end instead, which could potentially expose sensitive information in the browser if not managed well.