COMS E6998: **Microservices and Cloud Applications**

Lecture 6: OAuth2, FB, ~~Twitter~~, Cloud/AWS Security, ~~12 Factor Applications~~

Dr. Donald F. Ferguson
dff9@columbia.edu

Contents

Contents

- Questions, Comments, Discussion?
- (Web/Cloud) Security Concepts 101
  - Concepts
  - HTTPS
  - Certificates, DNS, …
- Authentication and Registration
  - SeekaTV demos: Email, Facebook, Twitter
  - Email registration/login
  - Social network registration/login
  - OAuth2 Overview; Facebook, Twitter
- Cloud Security Concepts and AWS Realization
  - Overview: Identity, Role, Resource, Policy
  - Application level versus AWS security management
  - Services and subsystems security
- Project 4: Simple Security
- 12 Factor Application

Questions? Comments? Discussion

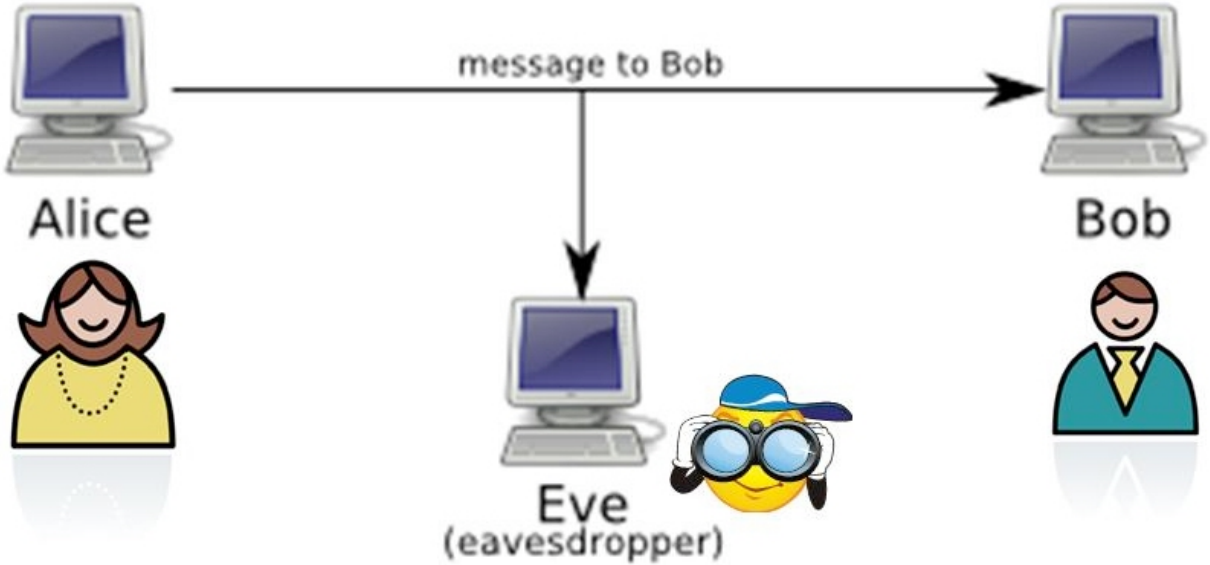
Web/Cloud Security 101

TLS/HTTPS
Certificates
DNS
CloudFront

(Some) Security Facets

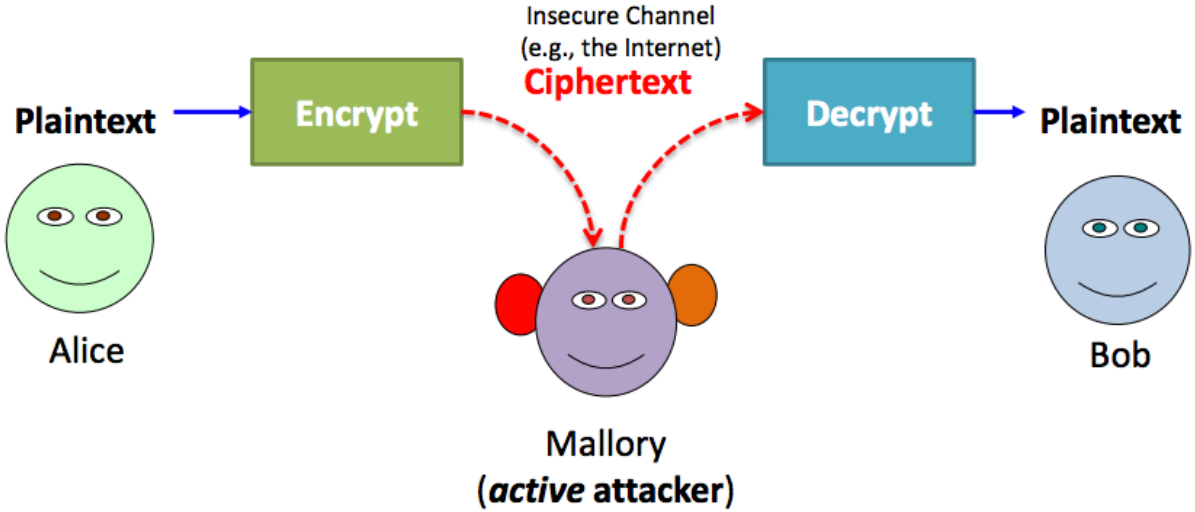
- Authentication: " (…) is the act of confirming the truth of an attribute of a single piece of data claimed true by an entity. In contrast with identification, which refers to the act of stating or otherwise indicating a claim purportedly attesting to a person or thing's identity, authentication is the process of actually confirming that identity. "
- Authorization:
  - "The process of authorization is distinct from that of authentication. Whereas authentication is the process of verifying that 'you are who you say you are', authorization is the process of verifying that 'you are permitted to do what you are trying to do'.
  - "**Authorization** is the function of specifying access rights/privileges to resources related to information security and computer security in general and to access control in particular [1]. More formally, "to authorize" is to define an access policy."
- (Communication) Security
  - "**Transport Layer Security** (**TLS**) and its predecessor, **Secure Sockets Layer** (**SSL**), both frequently referred to as "SSL", are cryptographic protocols that provide communications security over a computer network."
    - "The connection is *private* (or *secure*) because symmetric cryptography is used to encrypt the data transmitted."
    - The identity of the communicating parties can be *authenticated* using public-key cryptography.
    - The connection ensures *integrity* because each message transmitted includes a message integrity check using a message authentication code to prevent undetected loss or alteration of the data during transmission.
  - **HTTPS** (also called **HTTP over Transport Layer Security** [**TLS**],[1] **HTTP over SSL**,[2] and **HTTP Secure**[3][4]) is a communications protocol for secure communication over a computer network which is widely used on the Internet. HTTPS consists of communication over Hypertext Transfer Protocol (HTTP) within a connection encrypted by Transport Layer Security, or its predecessor, Secure Sockets Layer. The main motivation for HTTPS is authentication of the visited website and protection of the privacy and integrity of the exchanged data.

Alice – Bob – Eve



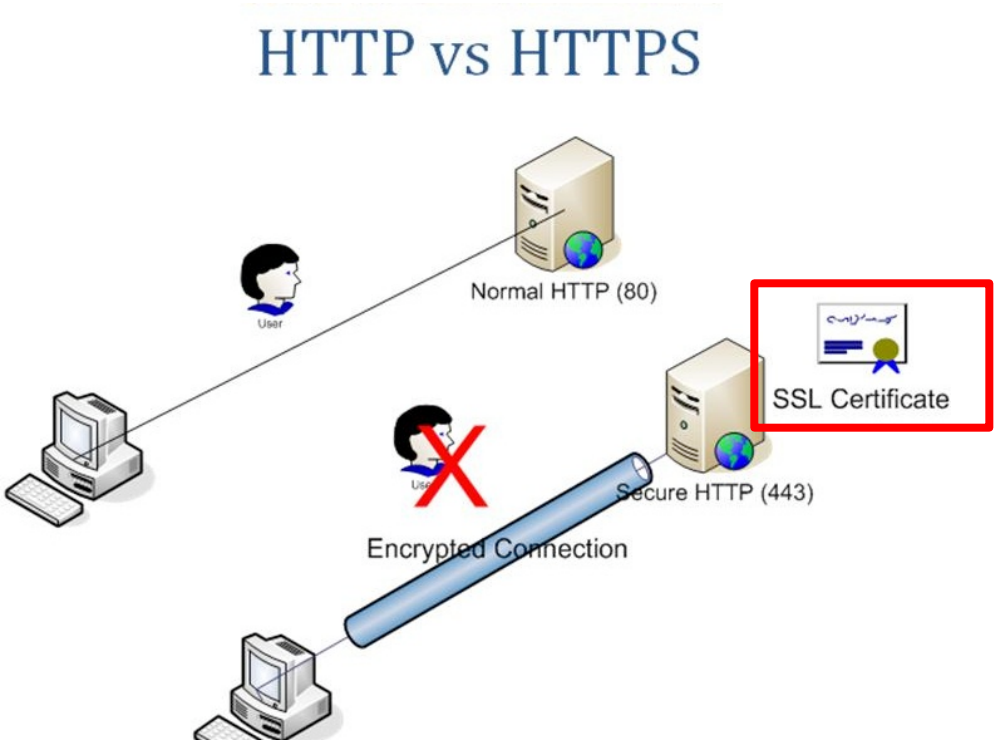
- Eve observes the unencrypted communication.
- Probably to do something malicious with the observed information, e.g.
  - Steal credit card numbers.
  - Steal login IDs and passwords.

Alice – Bob – Mallory



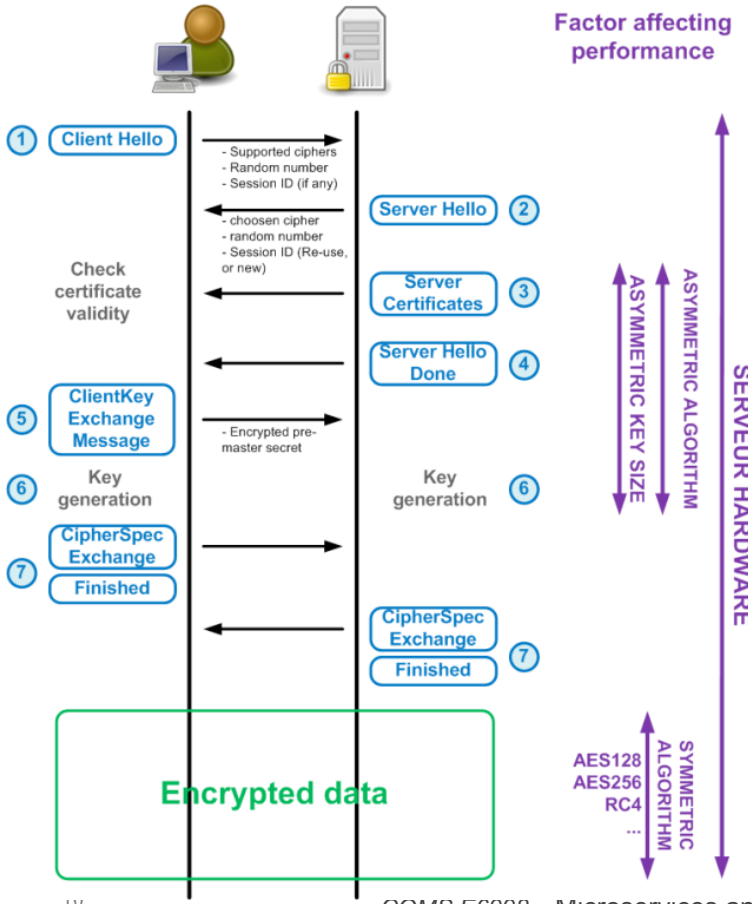
- Mallory intercepts the communication, including any security set up, e.g.
  - Keys
  - BASIC-AUTH
- Makes Bob think she is Alice and Alice think she is Bob to steal information or send evil messages (e.g. transfer money).

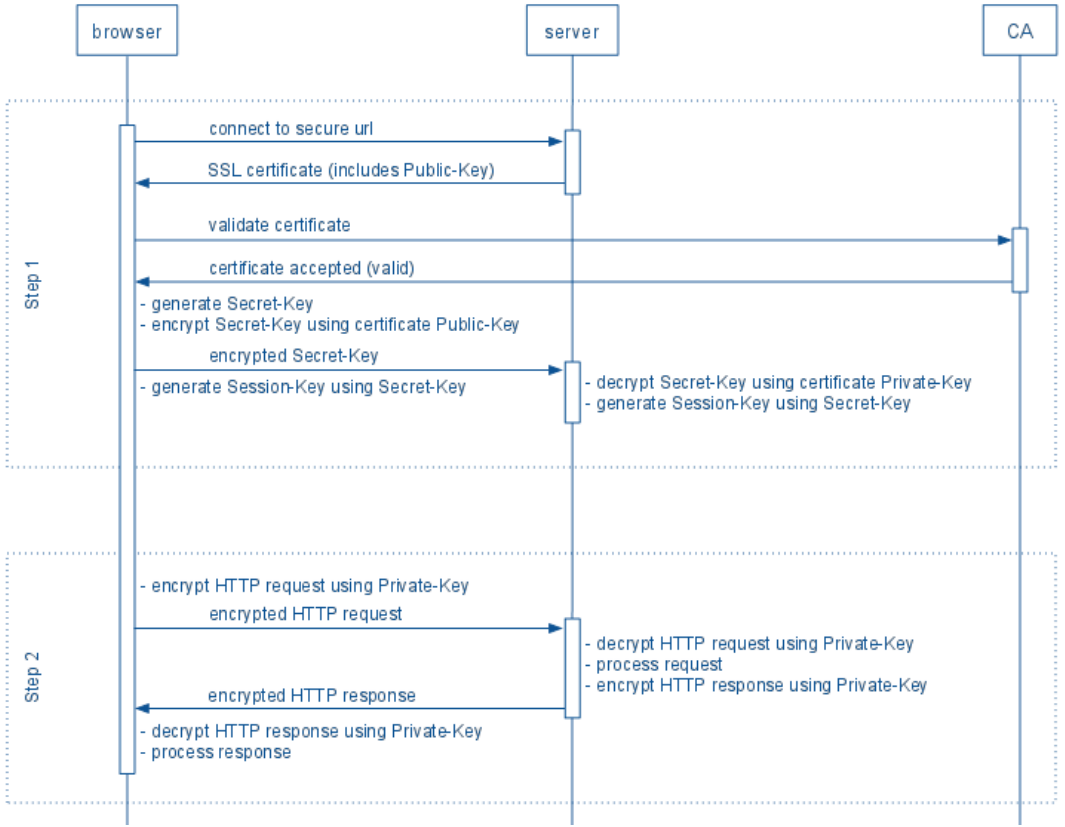
HTTPS



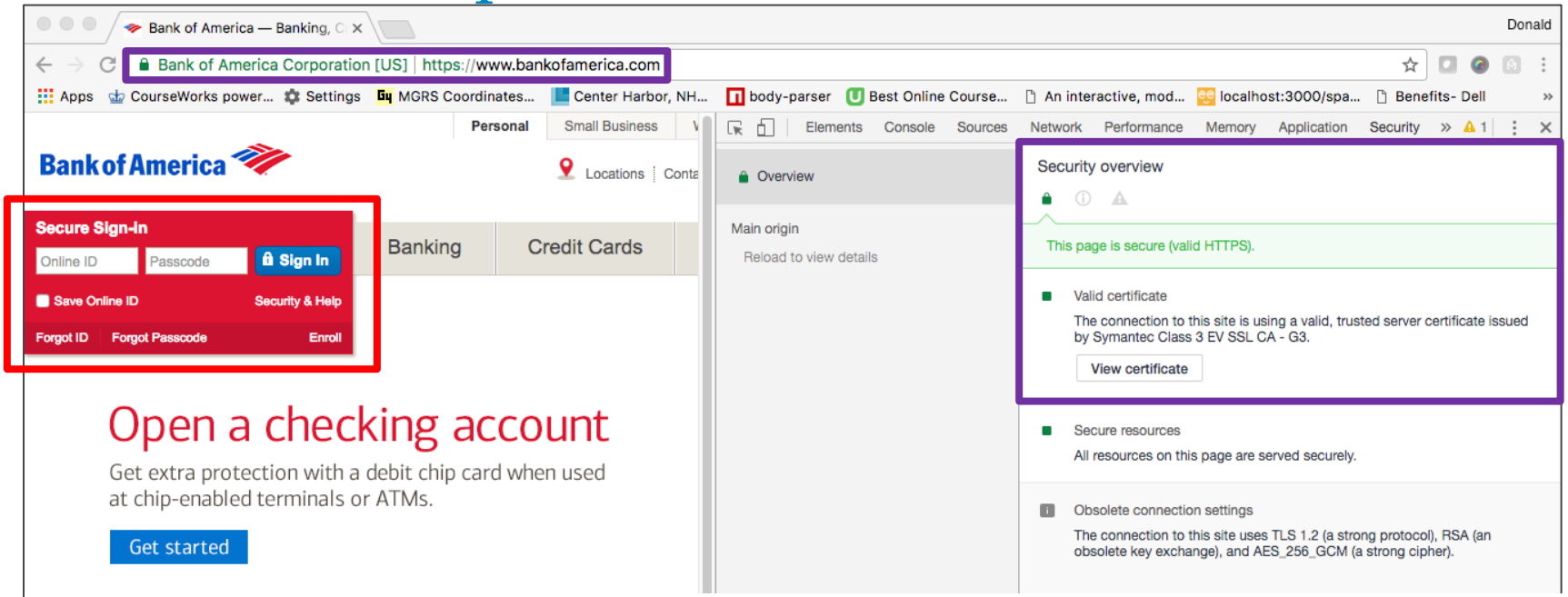
- Using
  - The SSL Certificate
  - PKI cryptography
- Bob can *prove* identity to Alice.
- Alice can send messages to Bob that only Bob can decrypt.
- This allows a
  - Key exchange
  - To set up a secure channel
- But, Alice has not proven identity to Bob.

Two Diagrams Explaining HTTPS/TLS



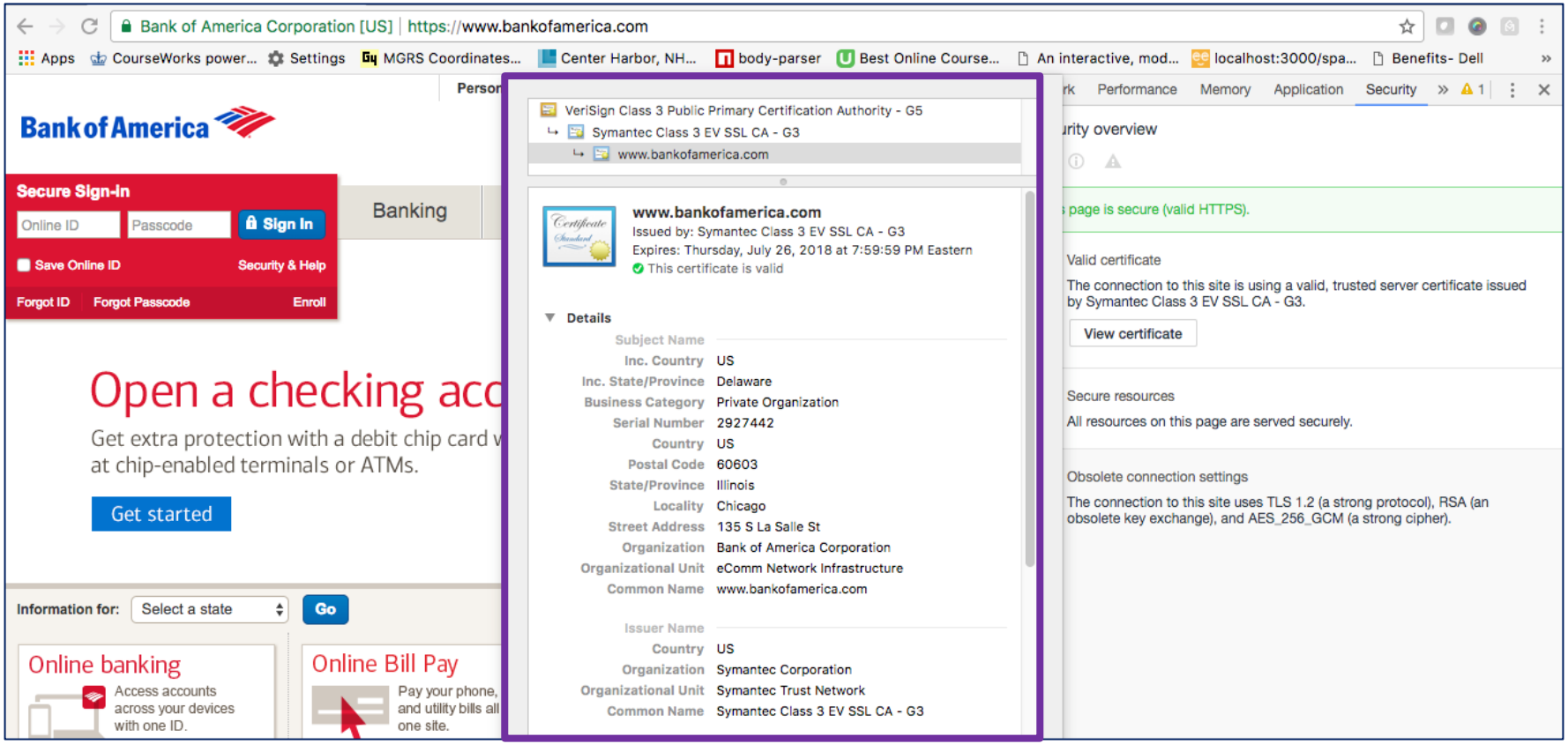


HTTPS Example

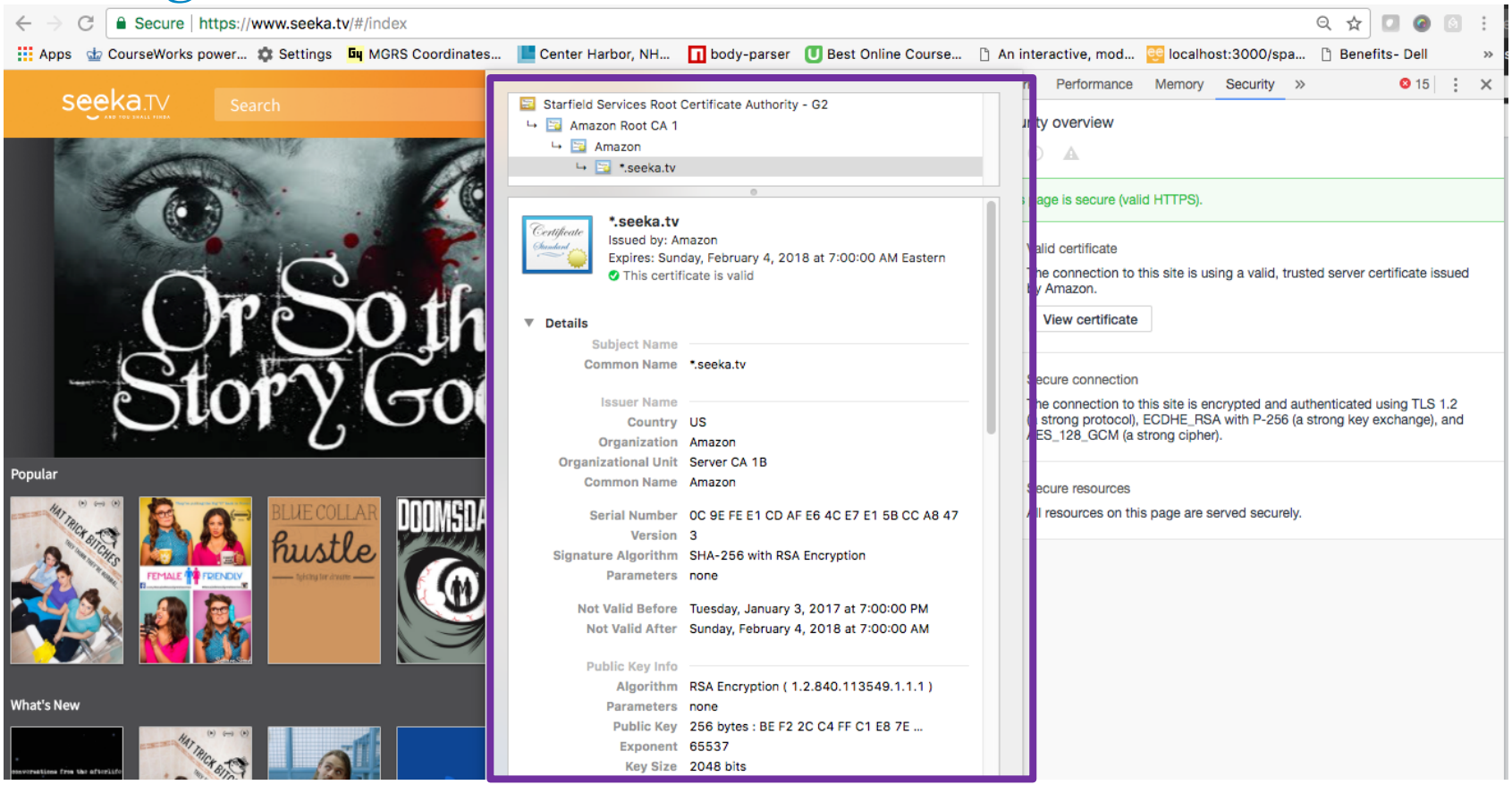


- I am connected to the owner of *bankofamerica.com* ➔
- I can send the site a shared secret, and be sure no one else can see it.
- This works for REST because REST is still HTTP(S).

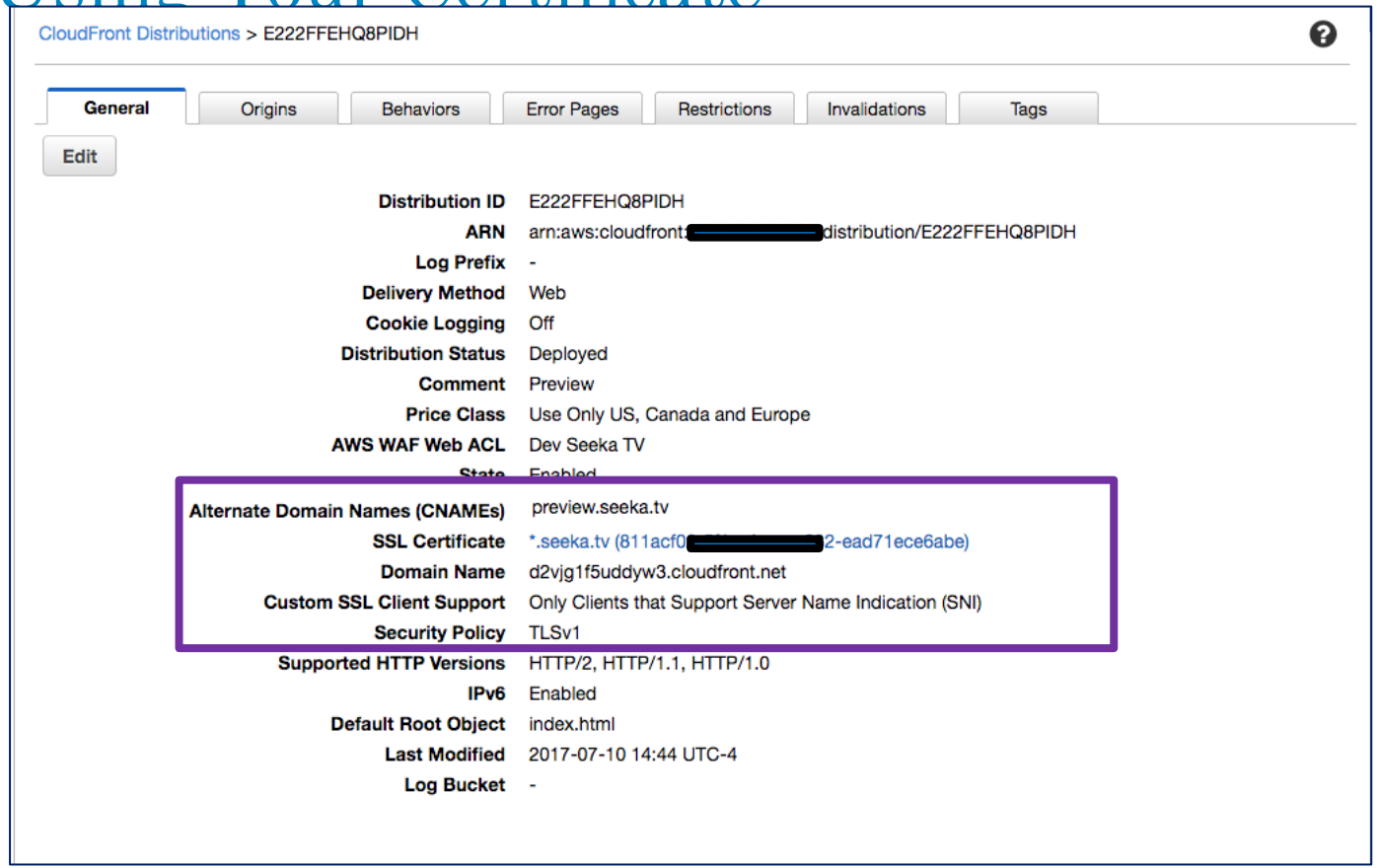
Certificate



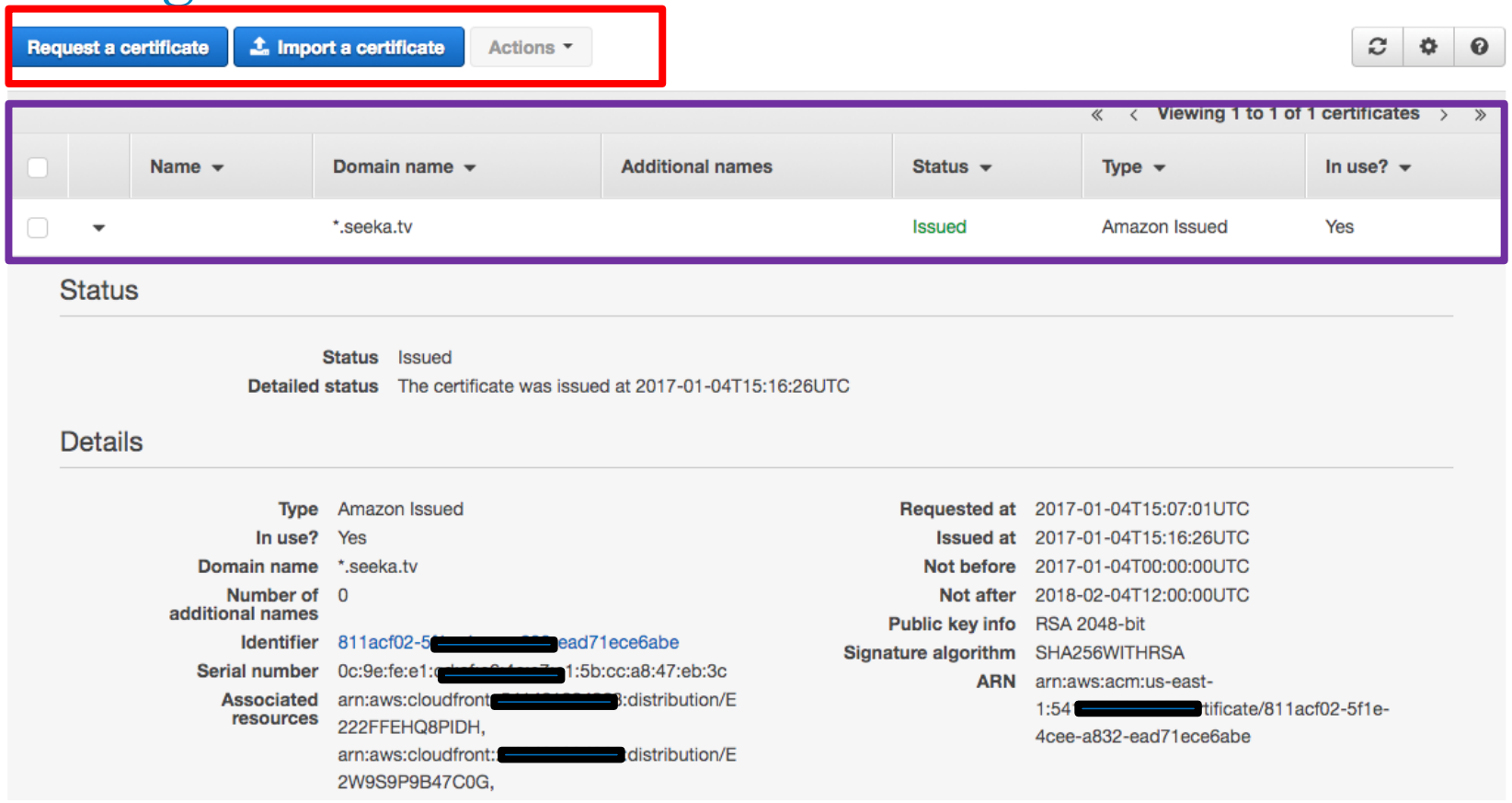
Using Your Certificate



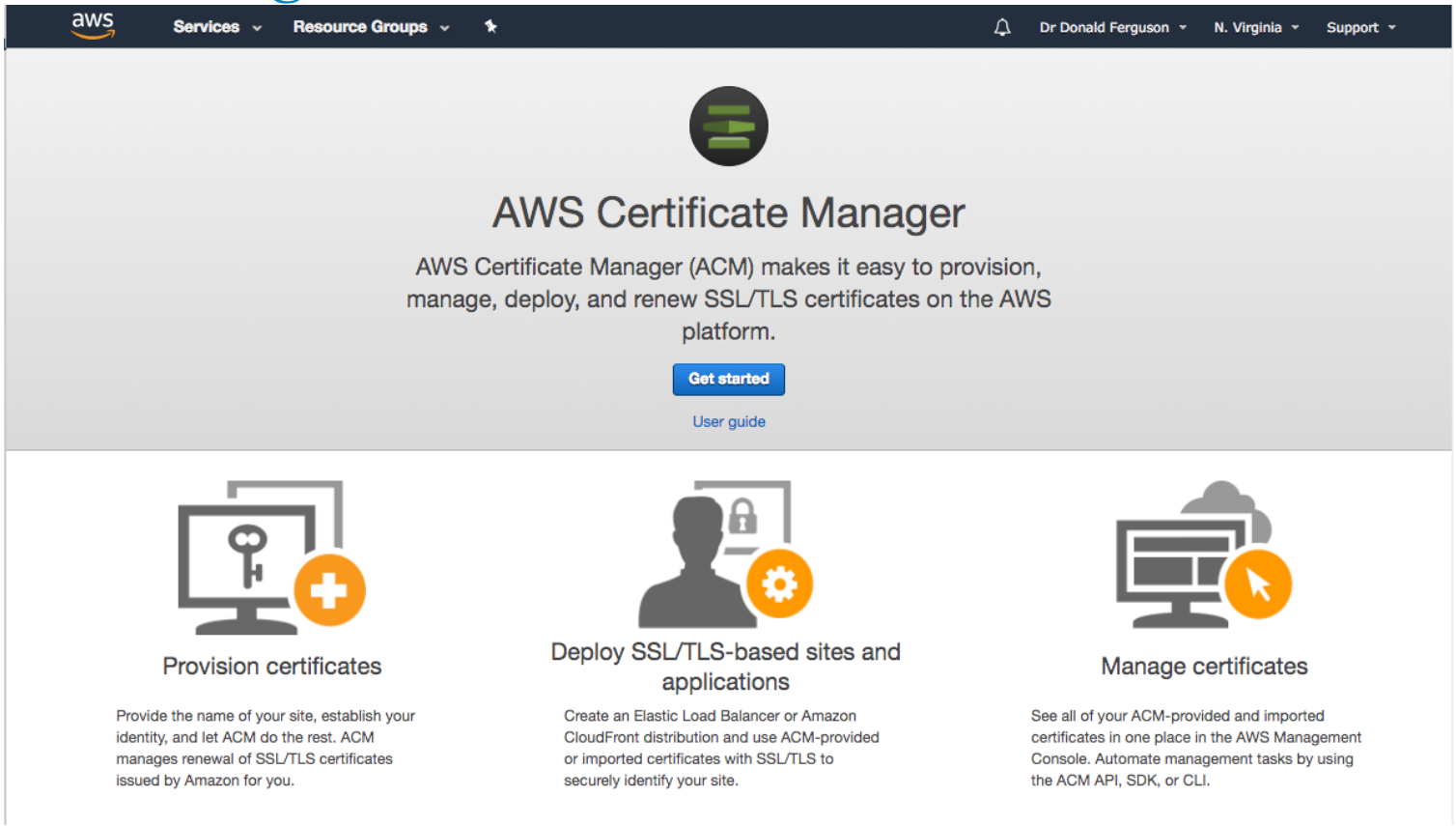
Using Your Certificate



Using Your Certificate



Obtaining a Certificate



You can use AWS Certificate Manager certificates with other AWS Services.

Choose **Import a certificate** to import an existing certificate instead of requesting a new one. Learn more. **Import a certificate**

Add domain names

Type the fully qualified domain name of the site you want to secure with an SSL/TLS certificate (for example, www.example.com). Use an asterisk (*) to request a wildcard certificate to protect several sites in the same domain. For example: *.example.com protects www.example.com, site.example.com and images.example.com.

Domain name*	Remove
donald-ferguson.net	

Add another name to this certificate

You can add additional names to this certificate. For example, if you're requesting a certificate for "www.example.com", you might want to add the name "example.com" so that customers can reach your site by either name. Learn more.

***At least one domain name is required**

Cancel **Review and request**

Review and request

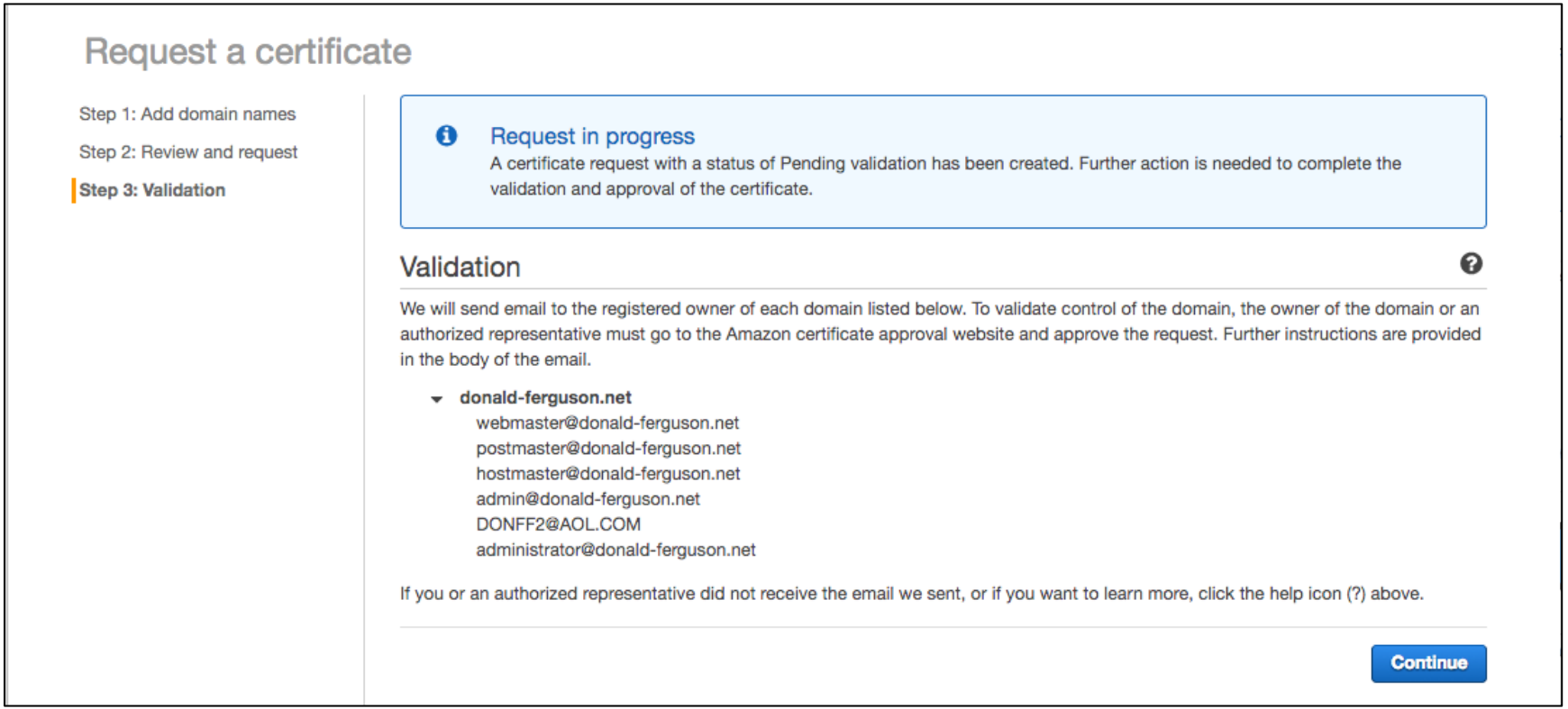
After you request the certificate, email will be sent to the registered owner of each domain name below. The domain o or an authorized representative can validate control of the domain and approve the certificate by following the instructions in the body of the email. After all of the domains are validated, the certifica will be issued.

Domain name

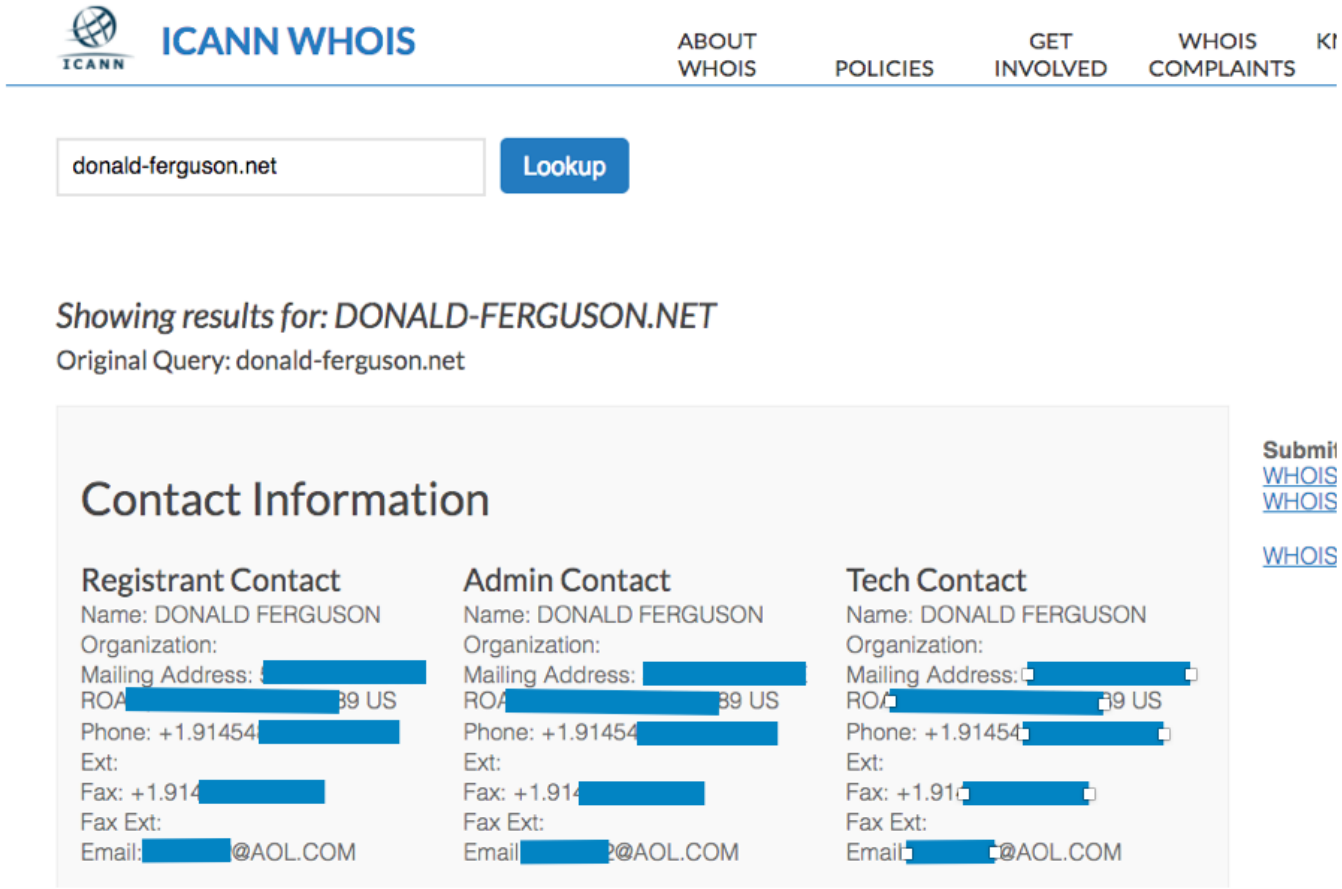
The name you want to secure with an SSL/TLS certificate.

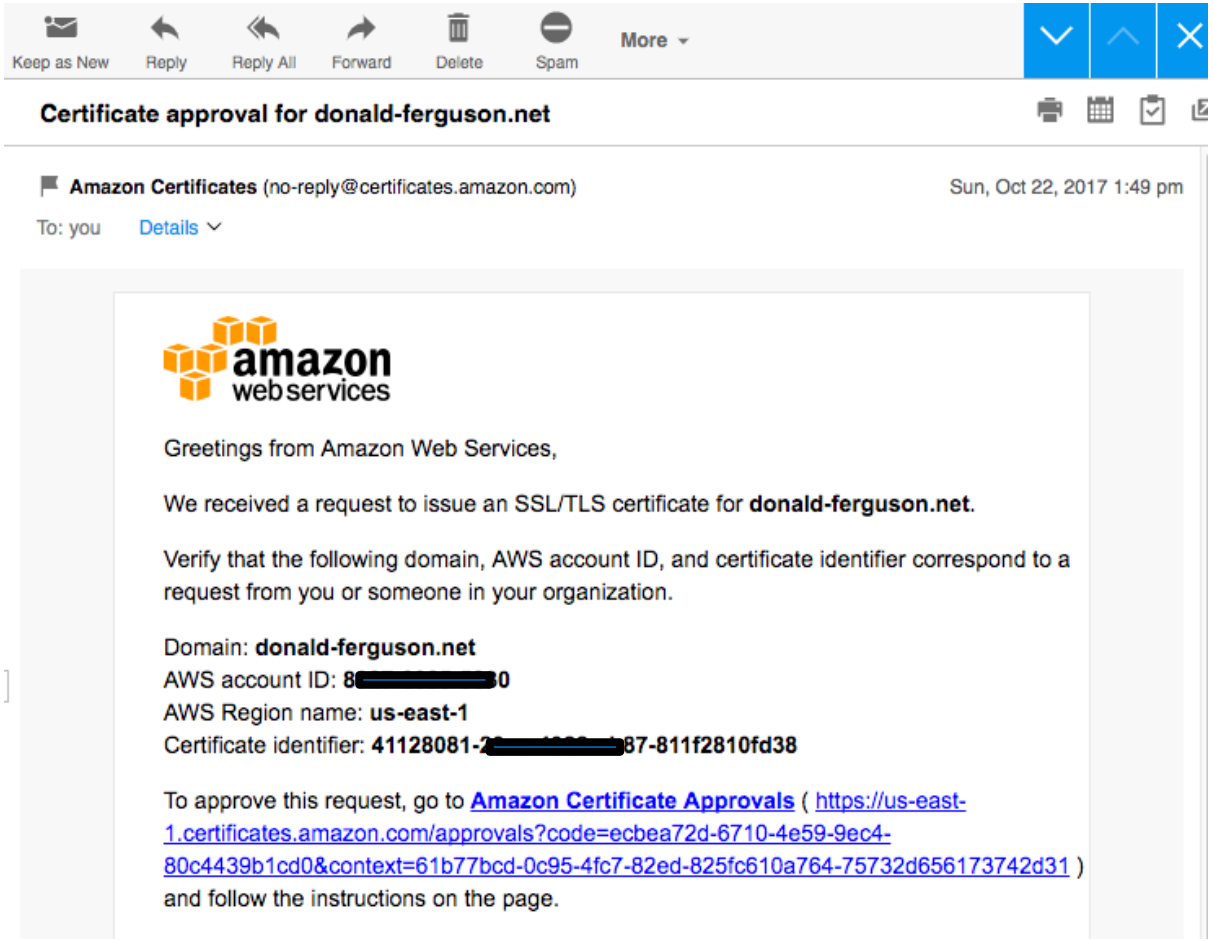
Domain name donald-ferguson.net

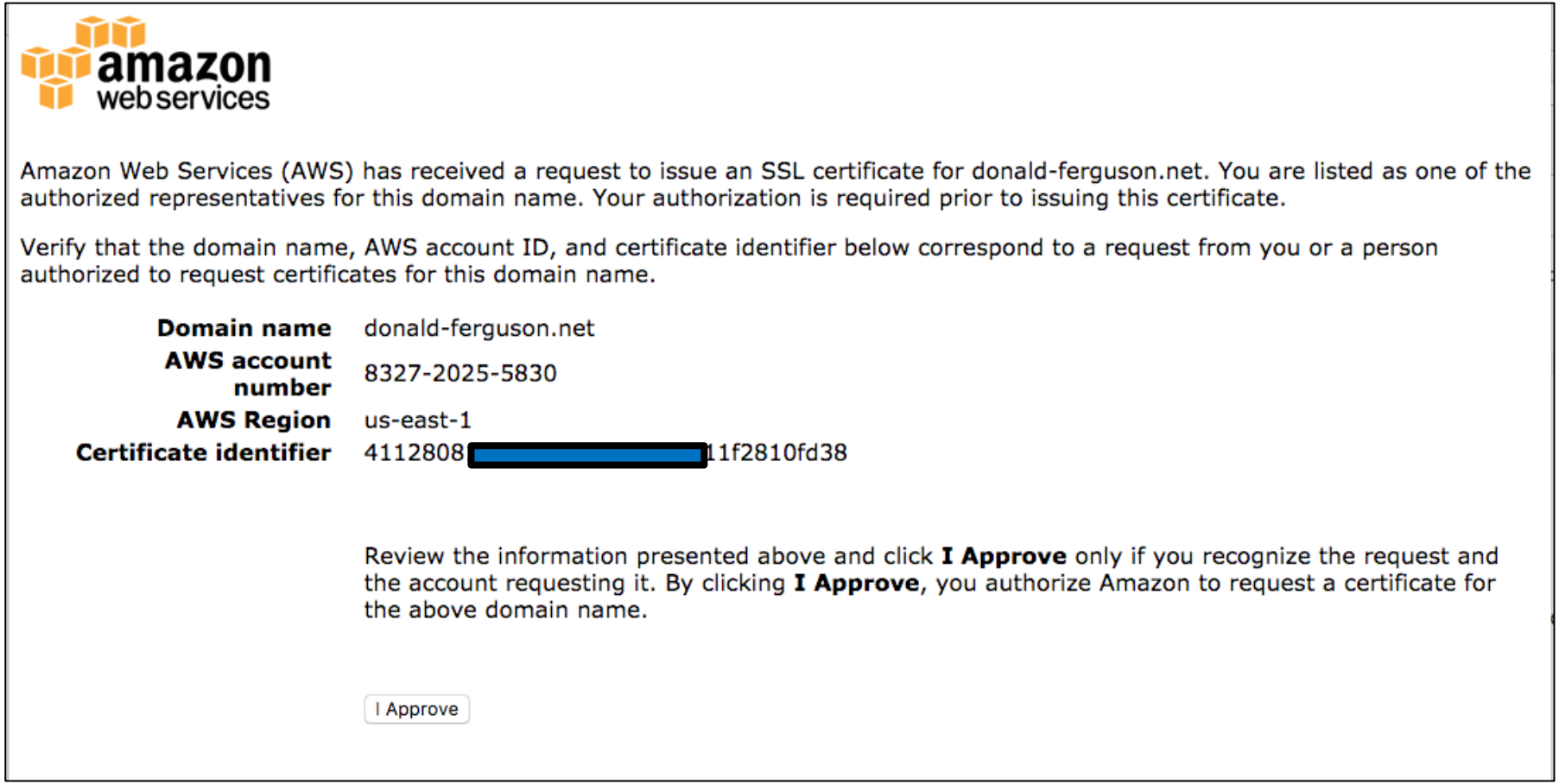
Cancel **Previous** **Confirm and request**



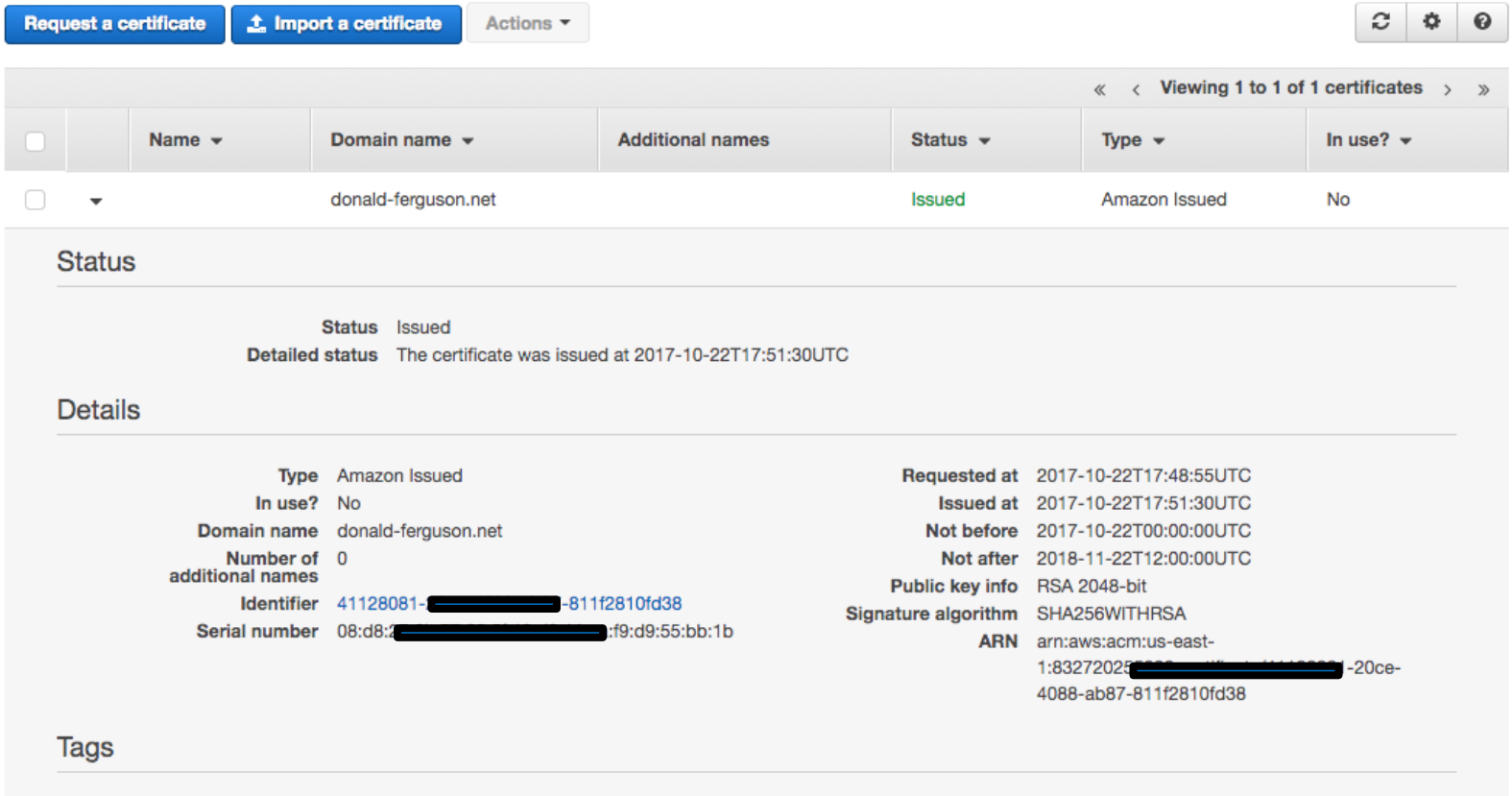
DNS Info



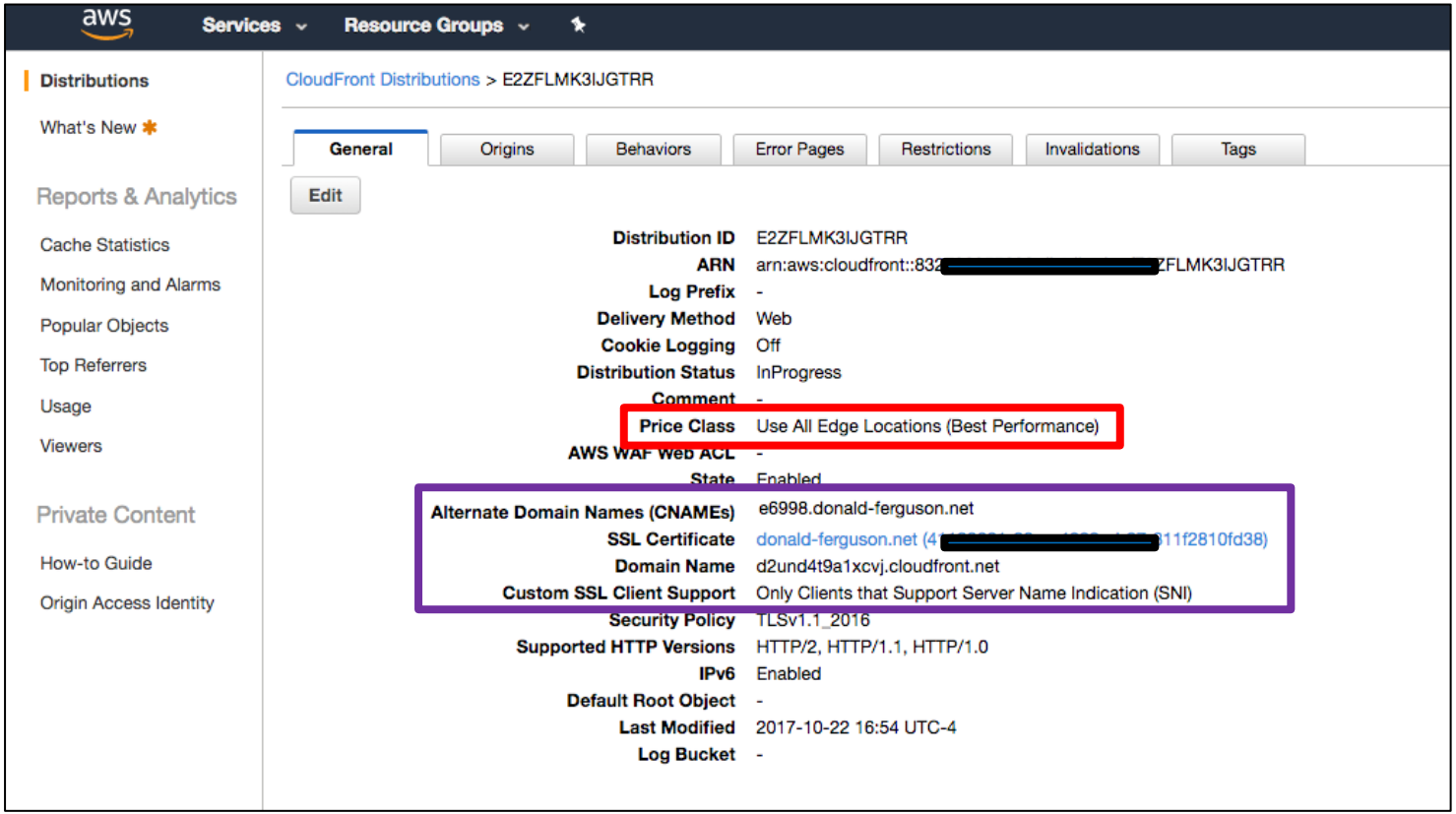




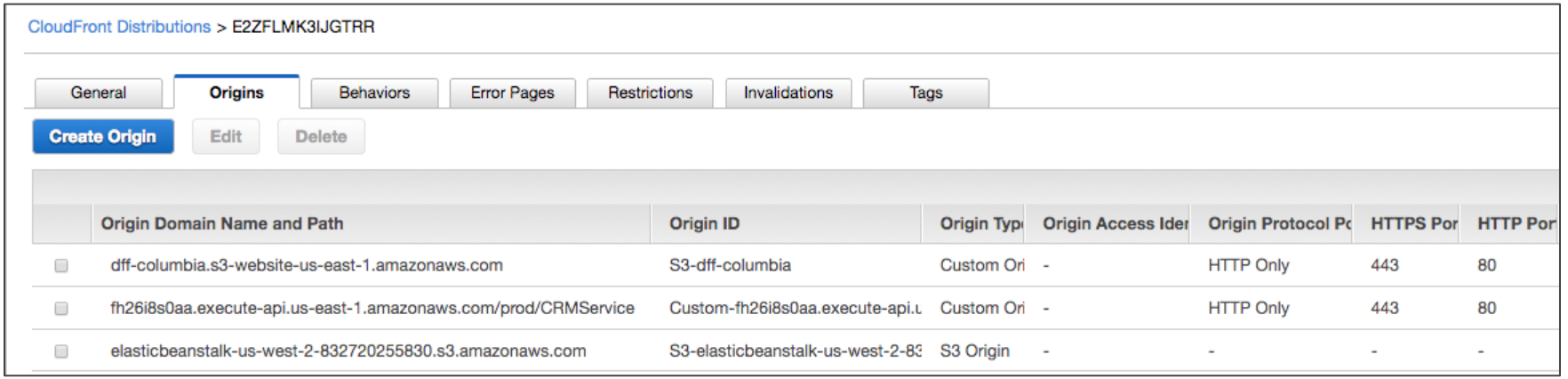
Success

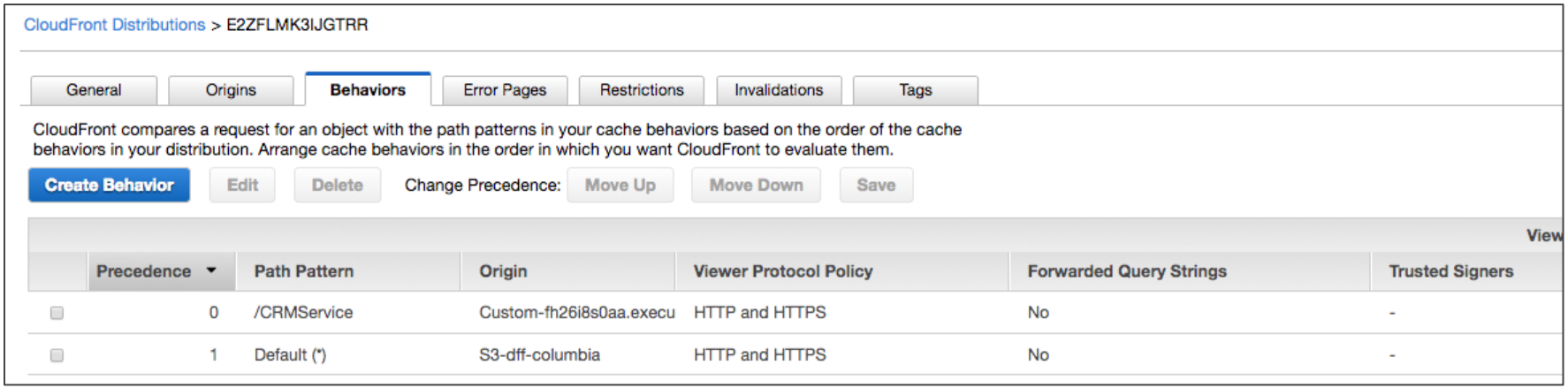


CloudFront Distribution

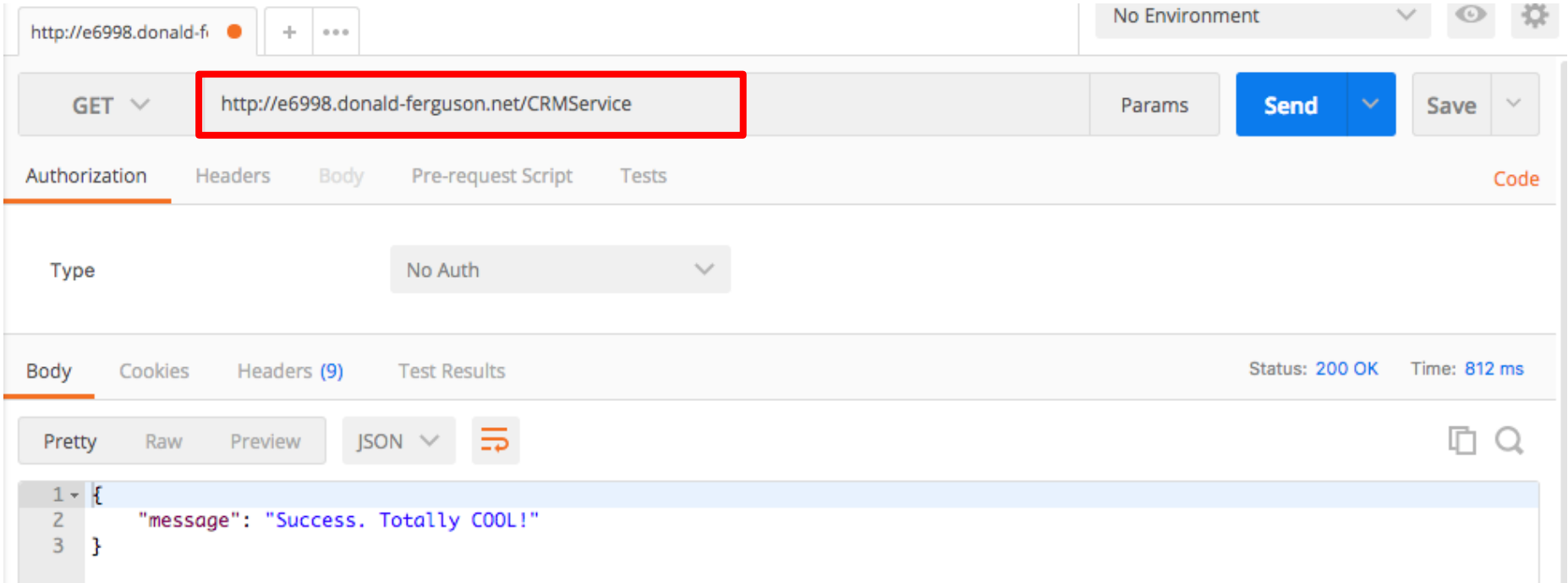


CloudFront Distribution

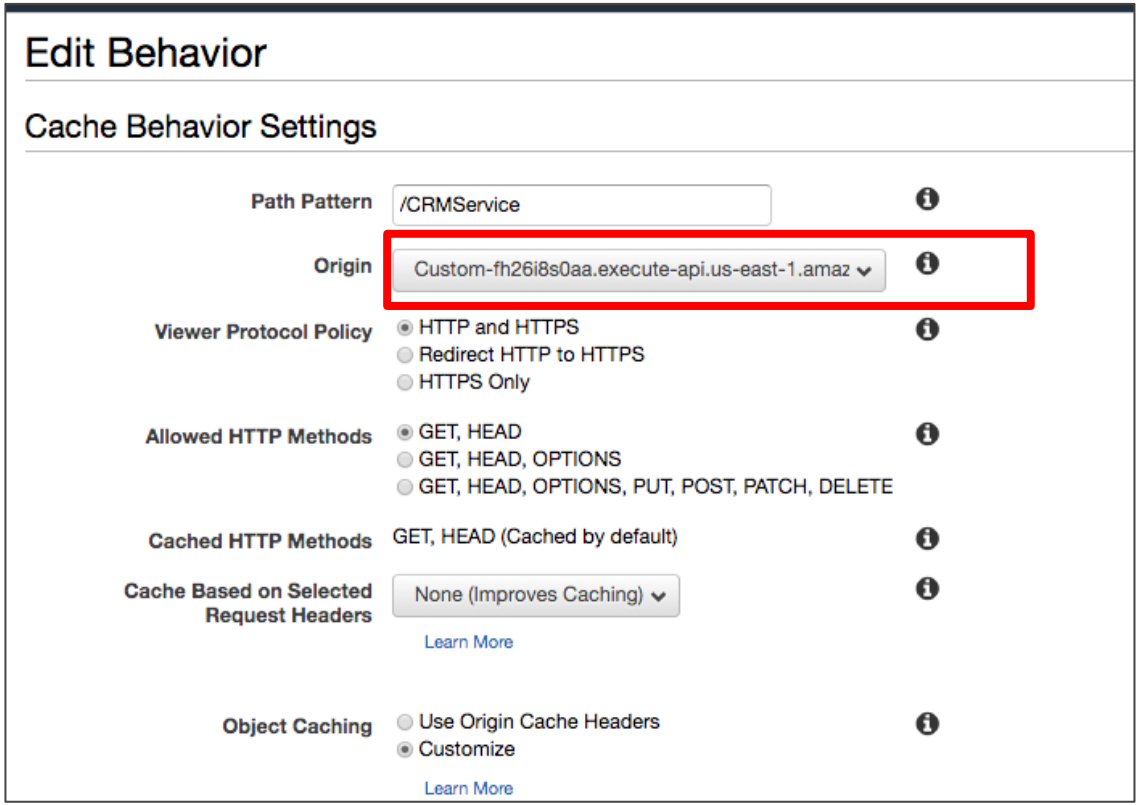




Behaviors

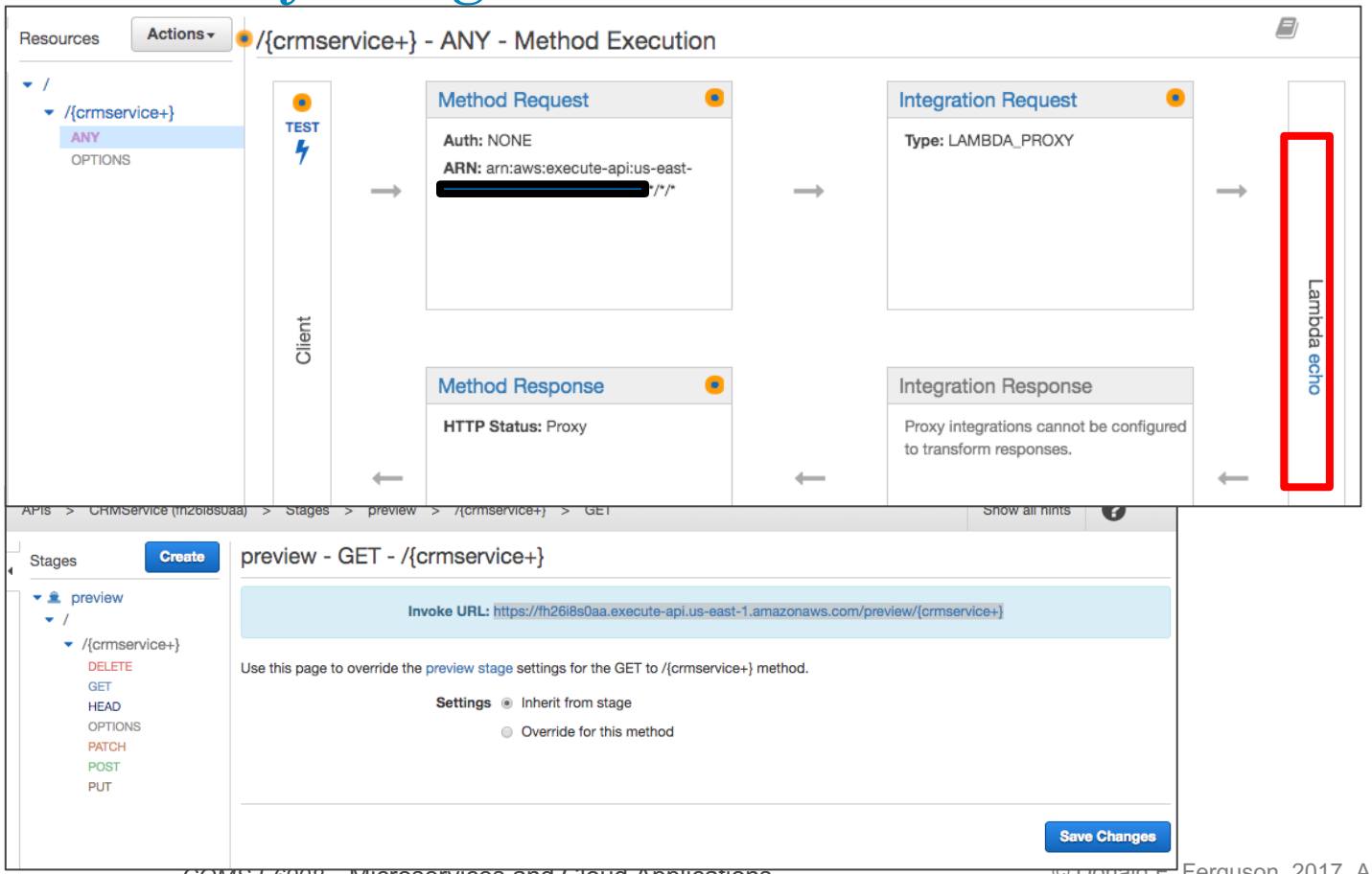


Behavior

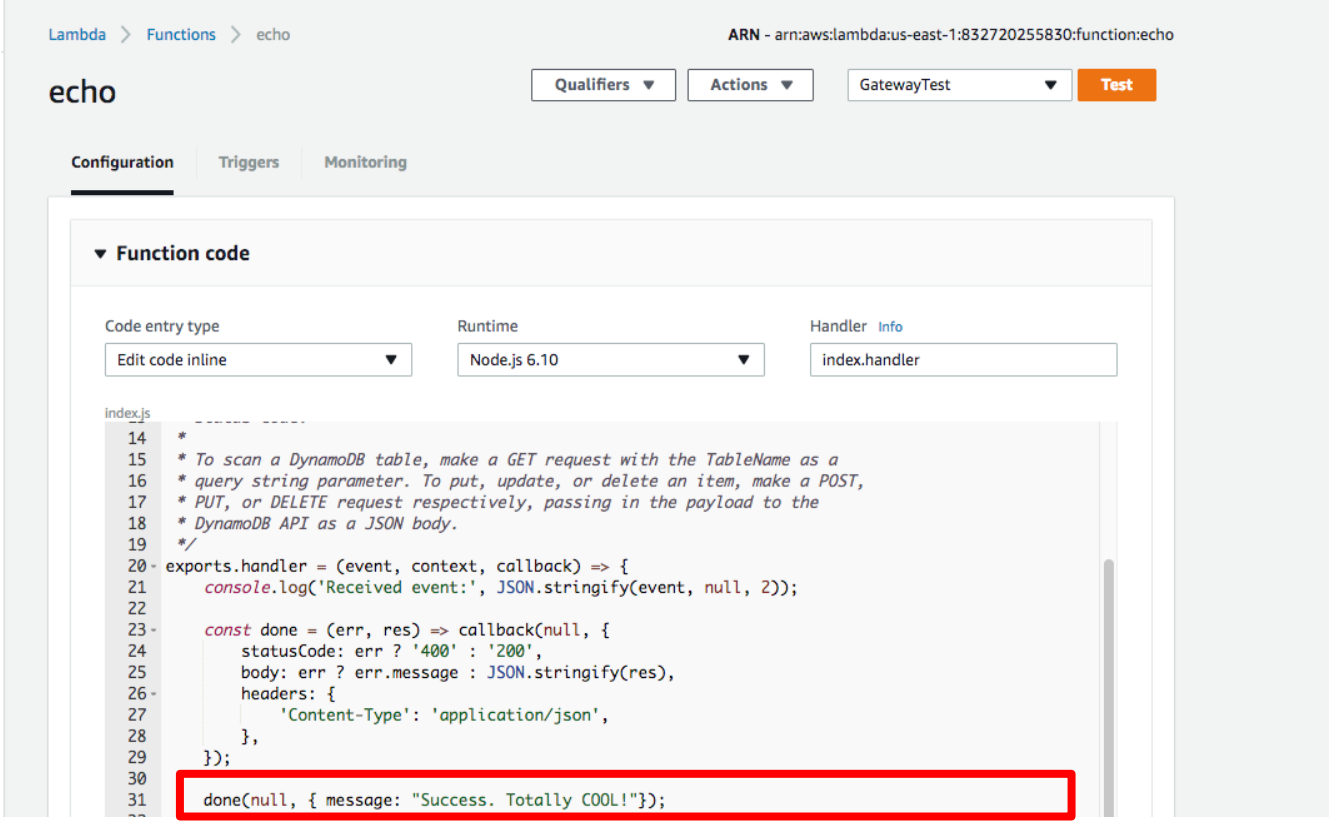


Invoke application
REST APIs via API Gateway

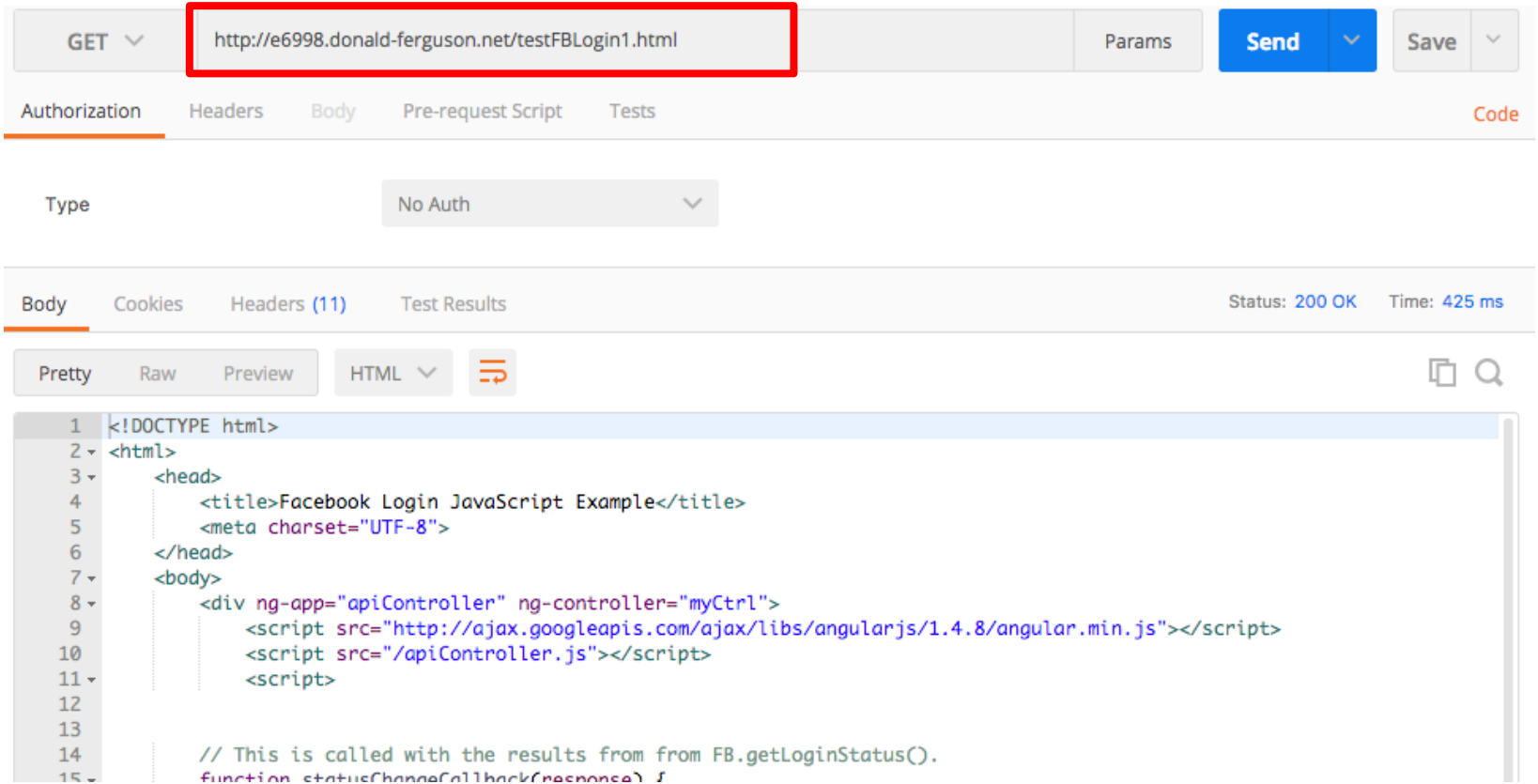
API Gateway Stage



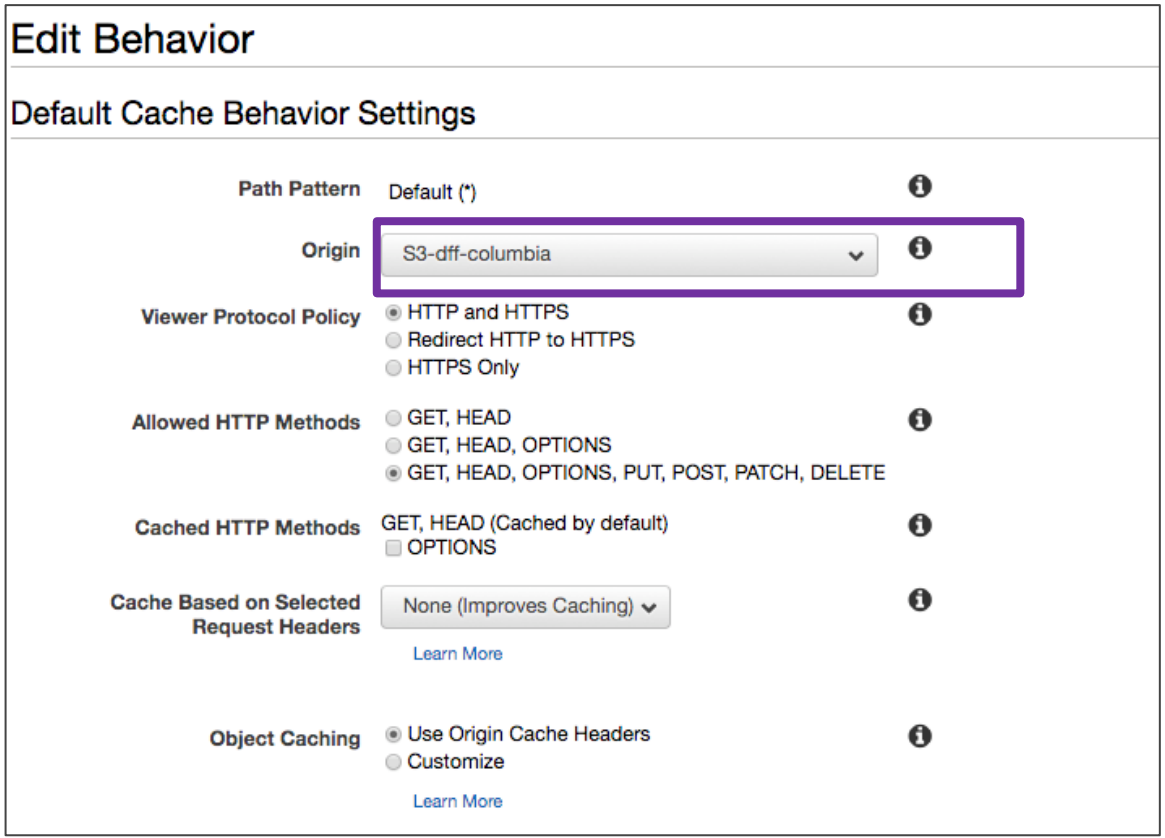
Implementation



Behaviors

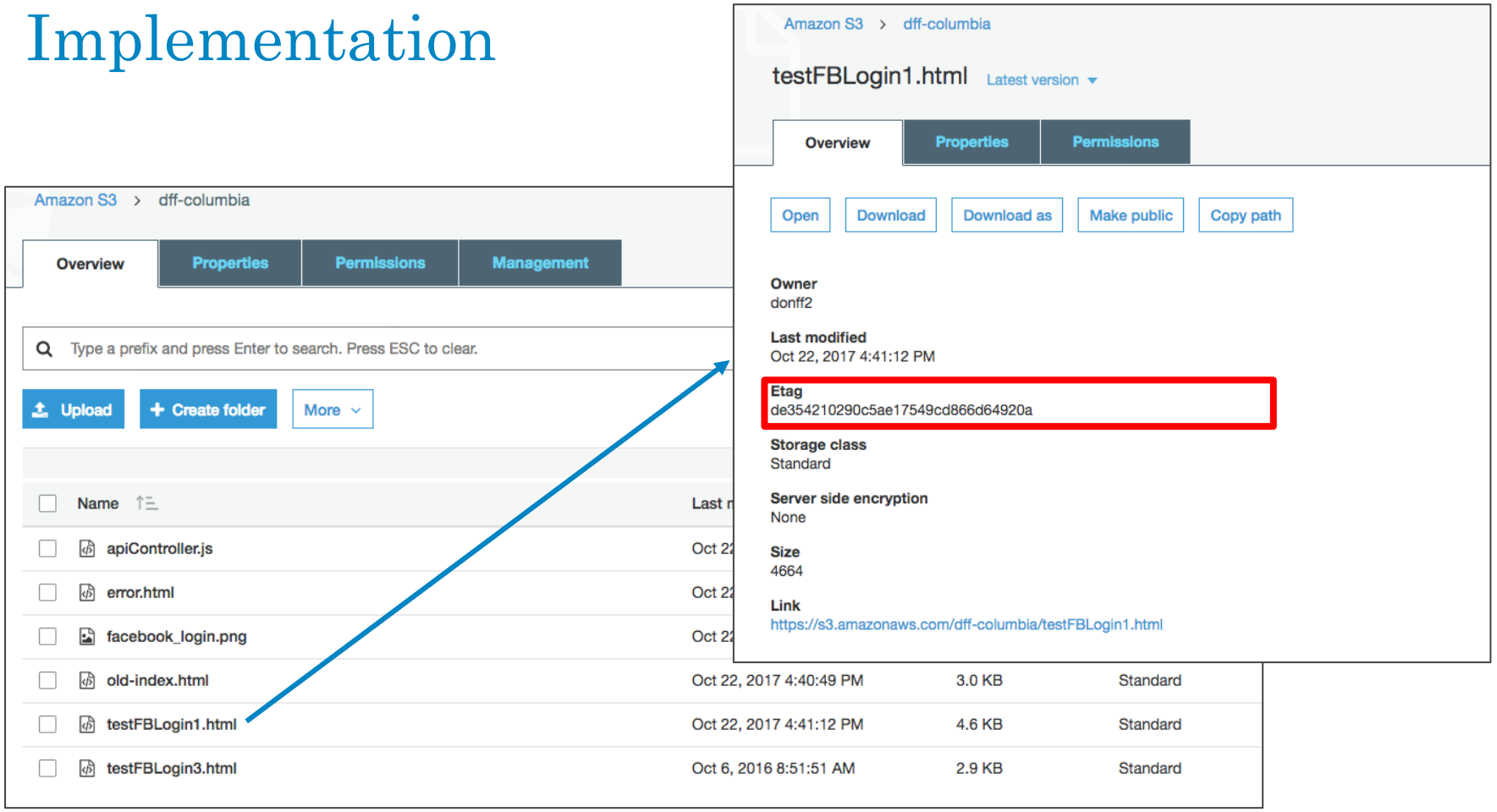


Behavior



Static content in S3

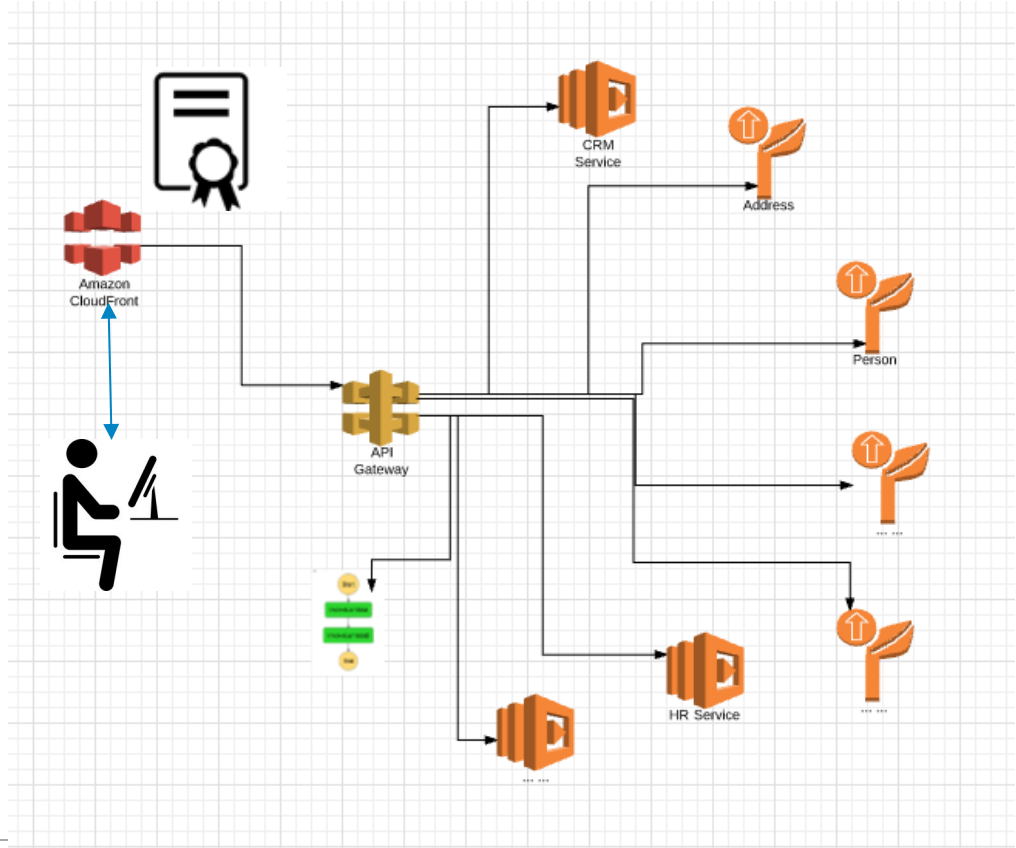
Implementation



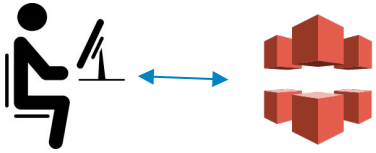
All of this to securely get to …

What Did We Set Up?

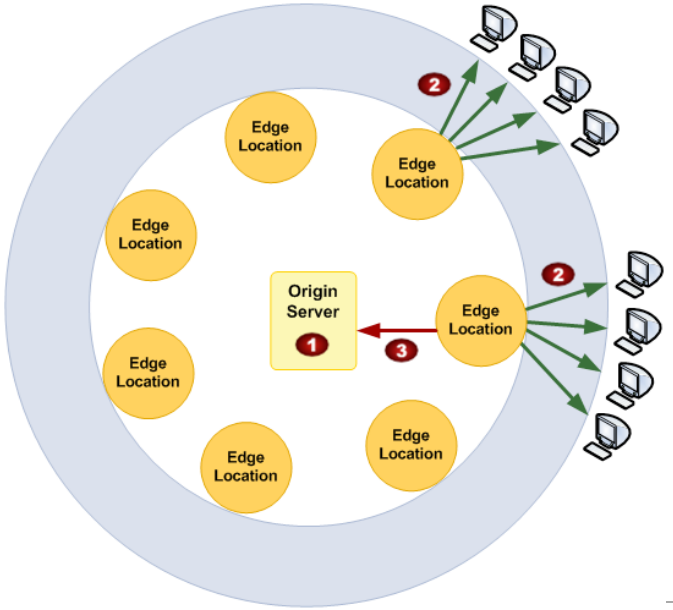
- We
  - Provide ownership of a domain.
  - Obtained a certificate.
  - Installed the certificate.
- Alice
  - Whether a person or REST client.
  - Can *authenticate* Bob.
  - And *authenticate* to Bob if,
    - They share a secret
    - Or trust someone else.

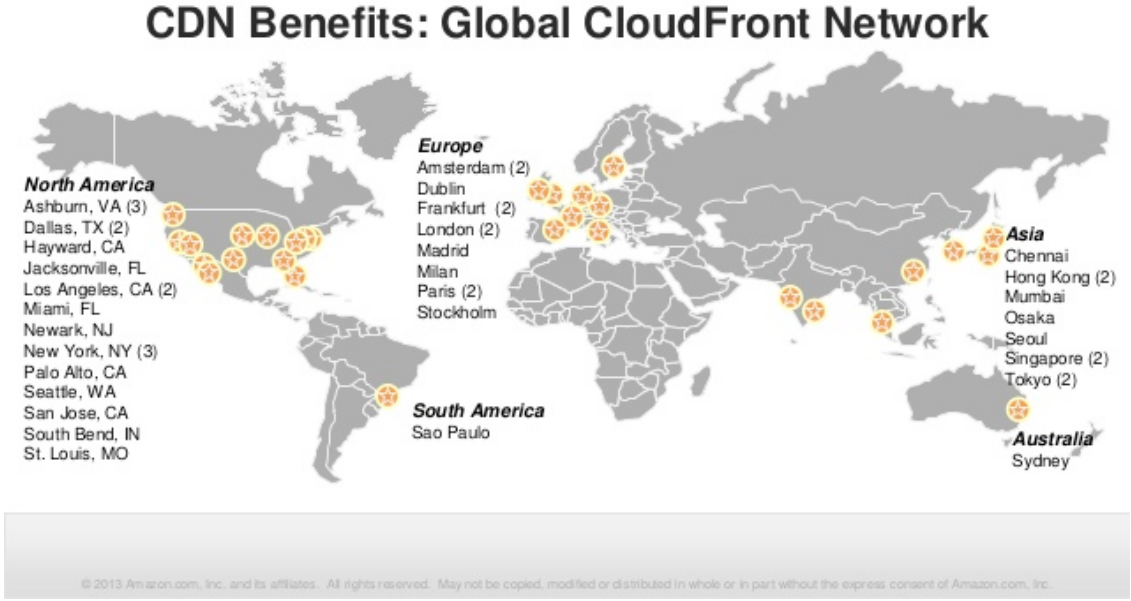


In Reality



The client – CloudFront connection is distributed and extremely local with file and REST API caching.





Once Again, …

- This was a very AWS specific explanation for simplicity.
- There are lots of technology choices for individual elements.
- Any realistic, non-trivial application solution requires
  - HTTPS
  - Obtaining and installing certificates.
  - Something implementing a single site image for multiple systems, and perhaps multiple data centers.
  - Caching and request routing.
  - API management and optimization.

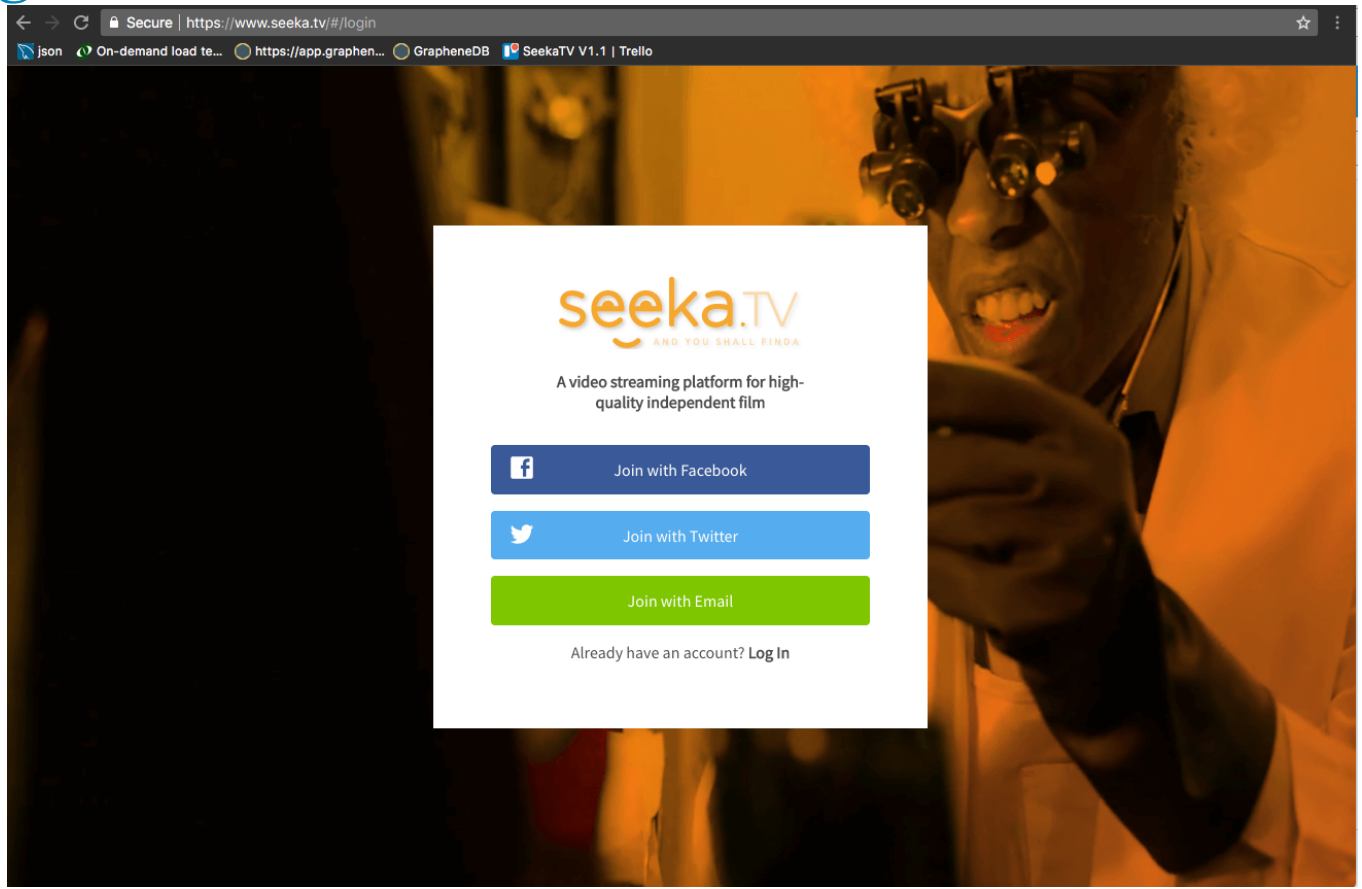
Seeka TV Demo

Email, FB, Twitter

OAuth2

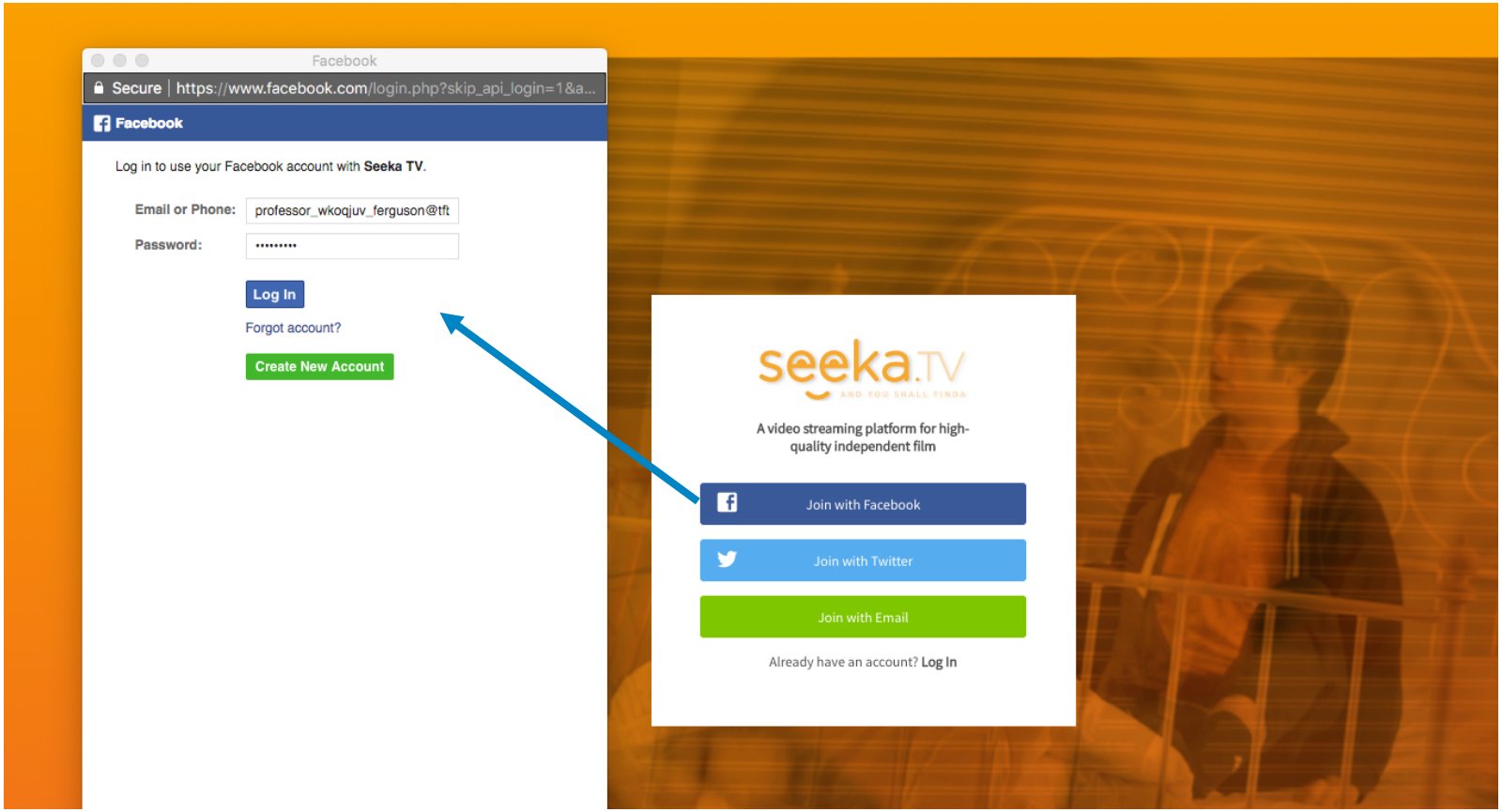
Demo

Registration

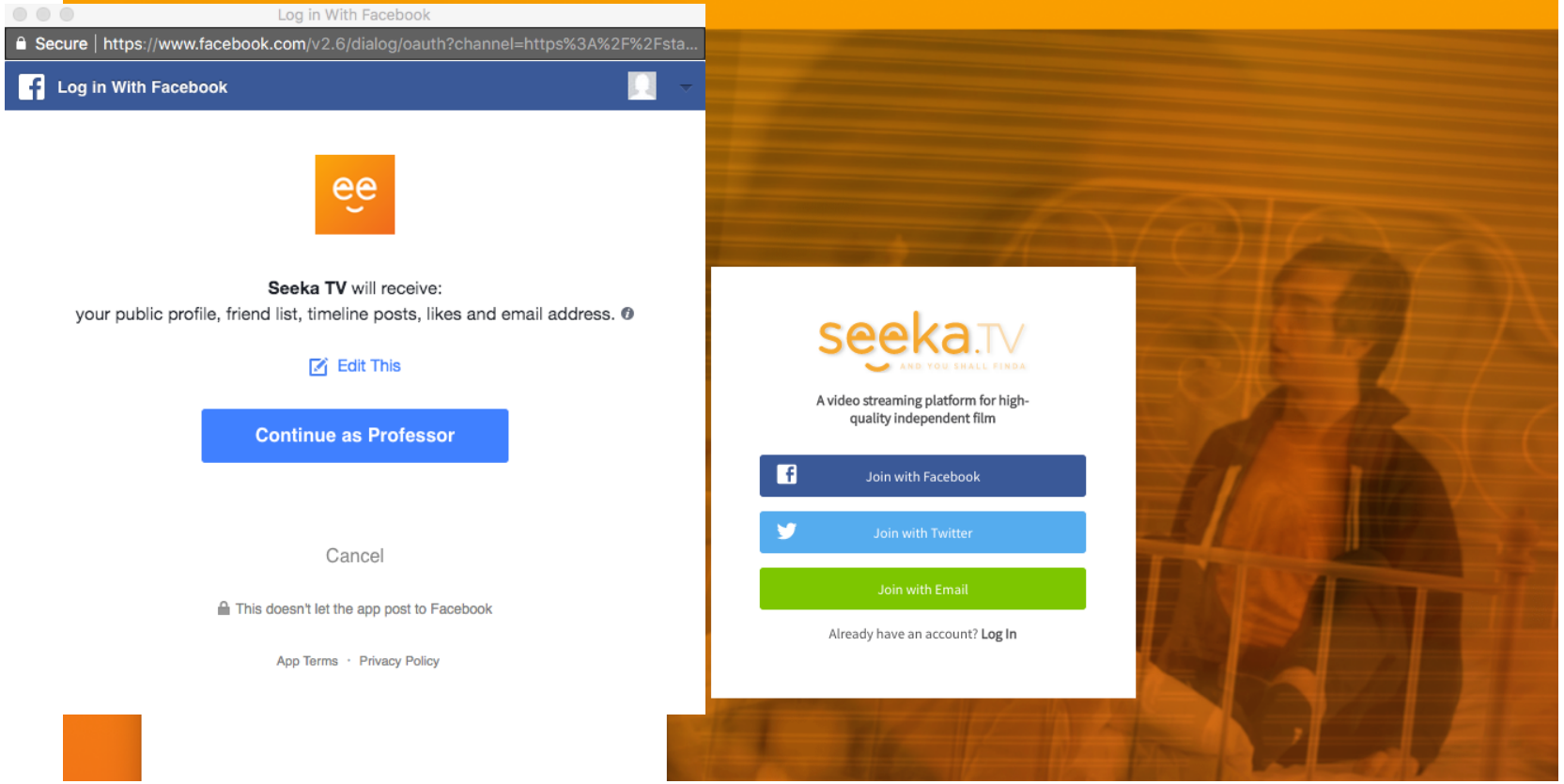


Registration

Registration



Registration



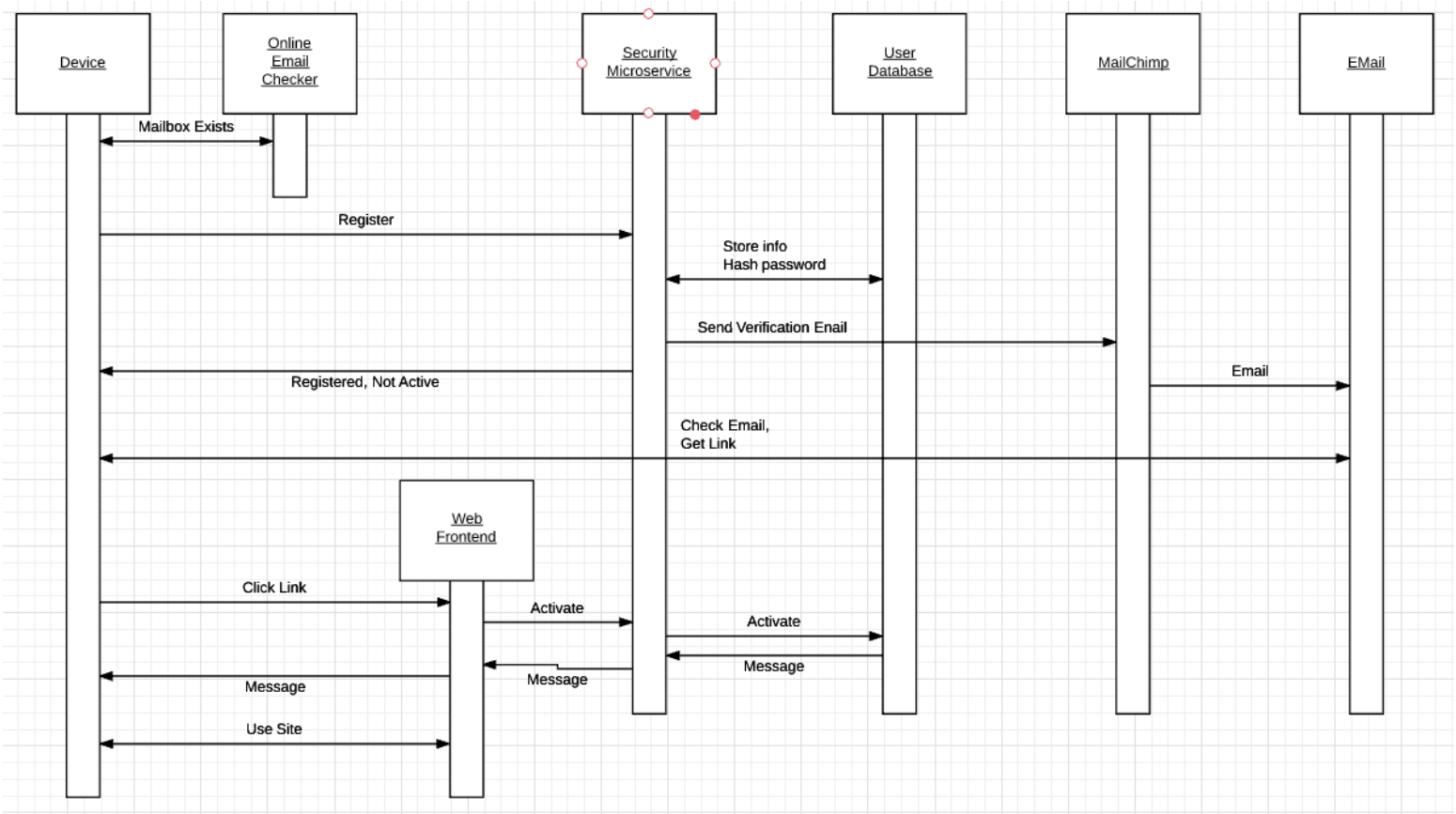
Pros and Cons of Social Media Logon

(https://econsultancy.com/blog/61911-the-pros-and-cons-of-a-facebook-login-on-ecommerce-sites)

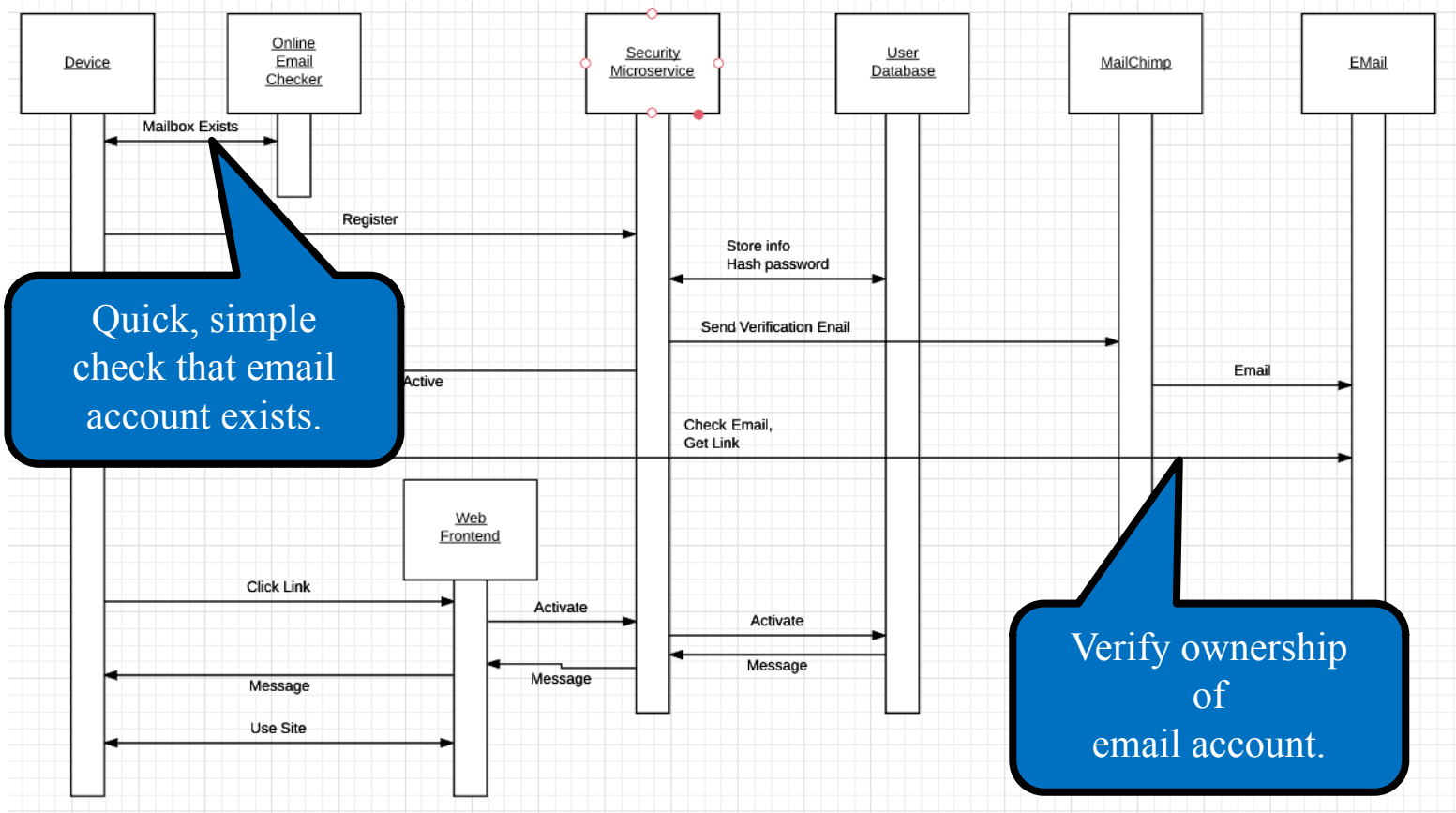
- Pros
  - One password for many services: Some people hate making up passwords
  - Familiarity: People trust Facebook and may not want to give you info.
  - Added convenience drives business: "Forced registration is a major cause of checkout abandonment, with a quarter (26%) of respondents in a recent Econsultancy survey stating that being forced to register would cause them to abandon a purchase."
  - Sharing: Simplifies customers sharing their use of your site → drives awareness.
  - Access to profile data: Do not require users to enter a new profile.
  - Your site does not have to have code for password reset, locked account, etc.
- Cons
  - Some users don't want everything to be connected.
  - Some people don't use Facebook.
  - Muddying your brand image.

Email

Email Registration

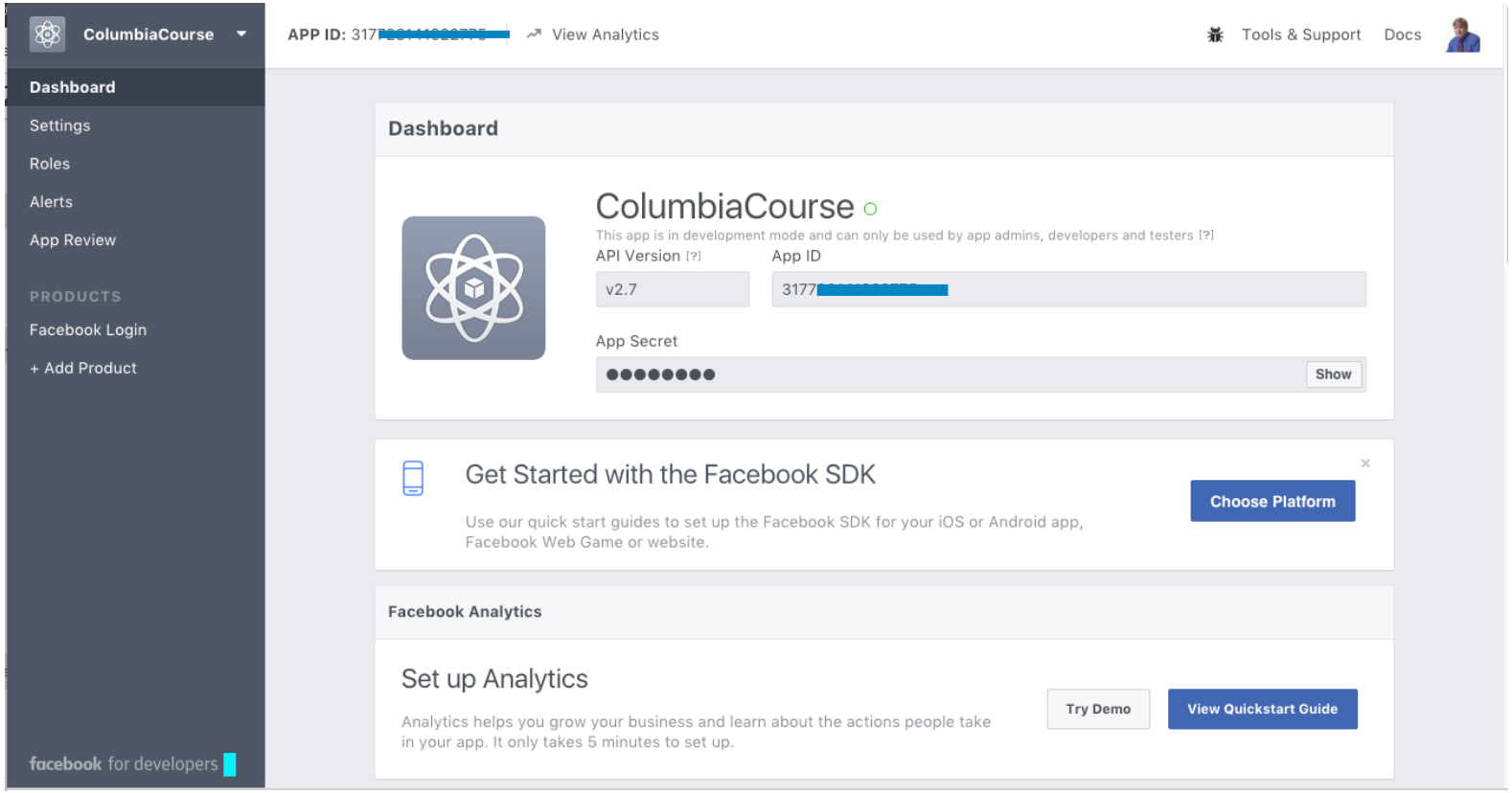


Email Registration

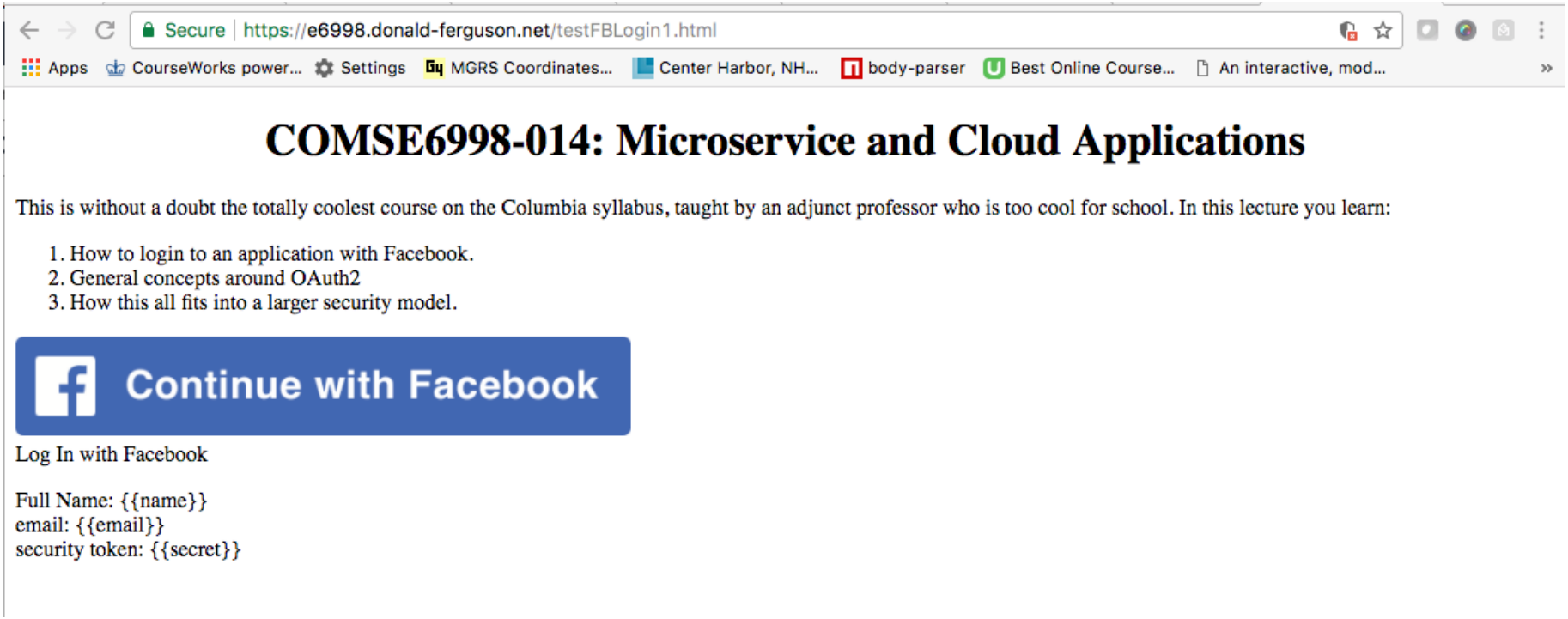


Social Media Registration (Facebook)

Define Application



Web Page



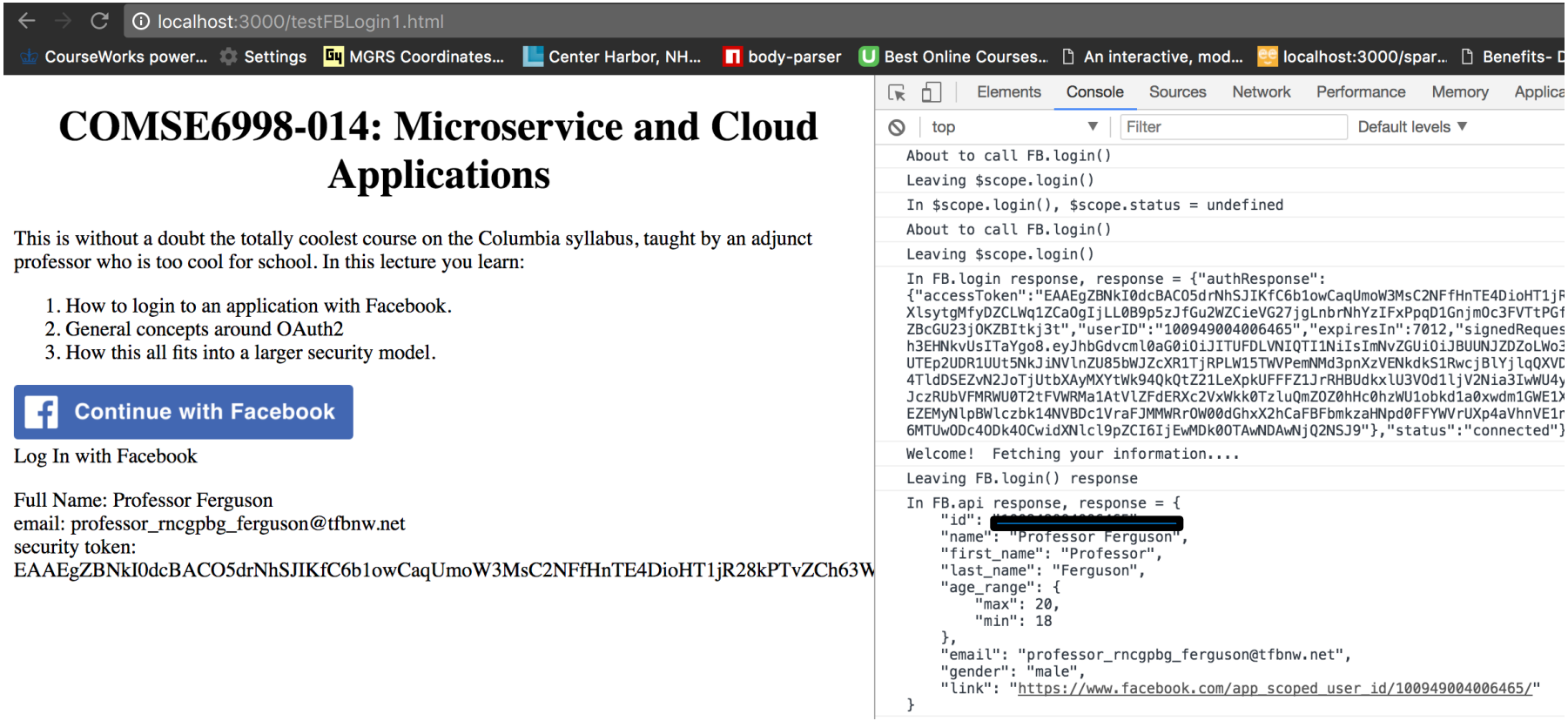
Code (I)

```
<div ng-app="apiController" ng-controller="myCtrl">
    <script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
    <script src="/apiController.js"></script>
    <script>


        // This function is called when someone finishes with the Login
        // Button.  See the onlogin handler attached to it in the sample
        // code below.
        function checkLoginState() {
            FB.getLoginStatus(function(response) {
                statusChangeCallback(response);
            });
        }

        window.fbAsyncInit = function() {
            FB.init({
                appId      : '3177[redacted]',
                cookie     : true,  // enable cookies to allow the server to access
                                    // the session
                xfbml      : true,  // parse social plugins on this page
                version    : 'v2.5' // use graph api version 2.5
            });

            // Now that we've initialized the JavaScript SDK, we call
            // FB.getLoginStatus().  This function gets the state of the
            // person visiting this page and can return one of three states to
            // the callback you provide.  They can be:
            //
            // 1. Logged into your app ('connected')
            // 2. Logged into Facebook, but not your app ('not_authorized')
            // 3. Not logged into Facebook and can't tell if they are logged into
            //    your app or not.
            //
            // These three cases are handled in the callback function.

            FB.getLoginStatus(function(response) {
                statusChangeCallback(response);
            });

        };

        // Load the SDK asynchronously
        (function(d, s, id) {
            var js, fjs = d.getElementsByTagName(s)[0];
            if (d.getElementById(id)) return;
            js = d.createElement(s); js.id = id;
            js.src = "//connect.facebook.net/en_US/sdk.js";
            fjs.parentNode.insertBefore(js, fjs);
        }(document, 'script', 'facebook-jssdk'));
```

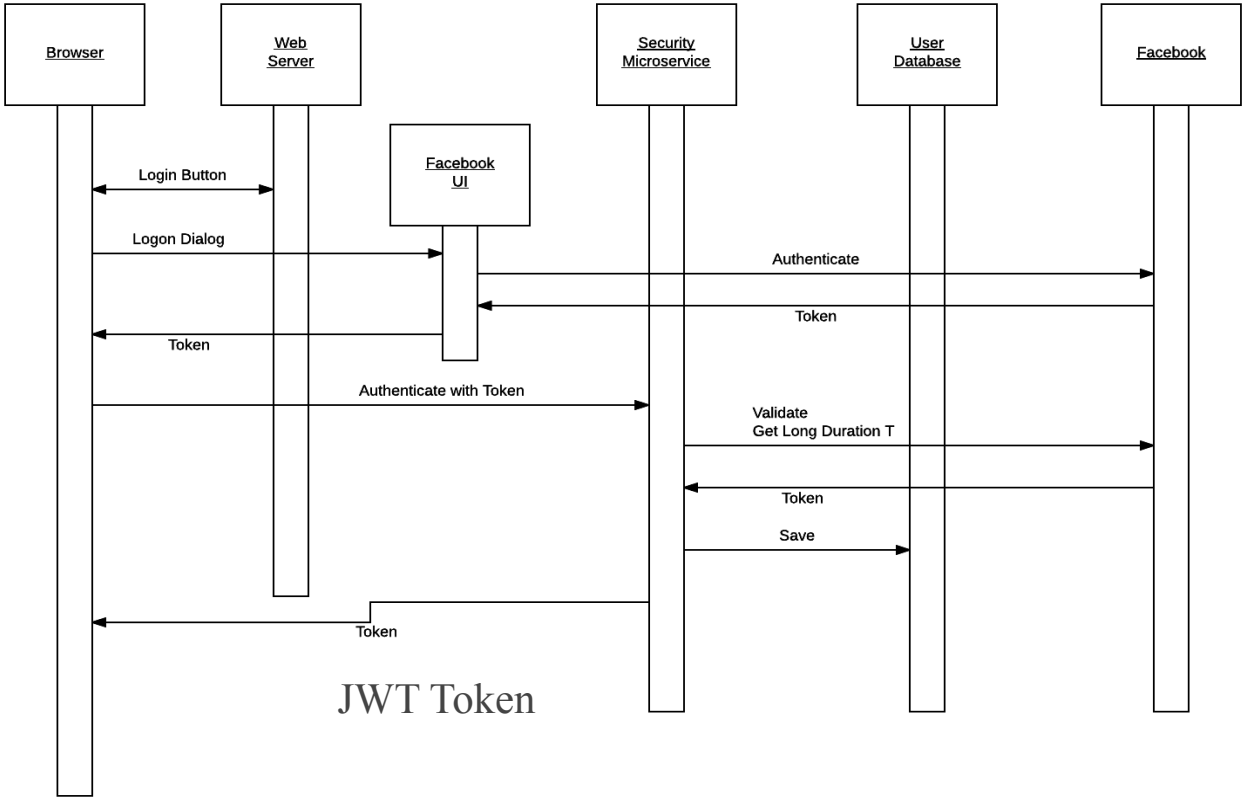
Code (II)

```
    <h1 style="text-align: center;">COMSE6998-014: Microservice and Cloud Applications</h1>
<p>
    This is without a doubt the totally coolest course on the Columbia syllabus, taught by an adjunct professor who is too cool for school. In this lecture you
    learn:
    <ol>
        <li>How to login to an application with Facebook.
        <li>General concepts around OAuth2
        <li>How this all fits into a larger security model.
    </ol>
<p>

<!--
<fb:login-button scope="public_profile,email" onlogin="checkLoginState();">
</fb:login-button>
-->
<div ng-click="login()">
    <img src="./facebook_login.png" style="width:40%;"><br>
    <span>Log In with Facebook</span>
</div>

<div>
    <p>
    Full Name: {{name}}<br>
    email: {{email}}<br>
    security token: {{secret}}<br>
    </p>

</div>
</div>
</body>
</html>
```

Facebook Login



Facebook Login



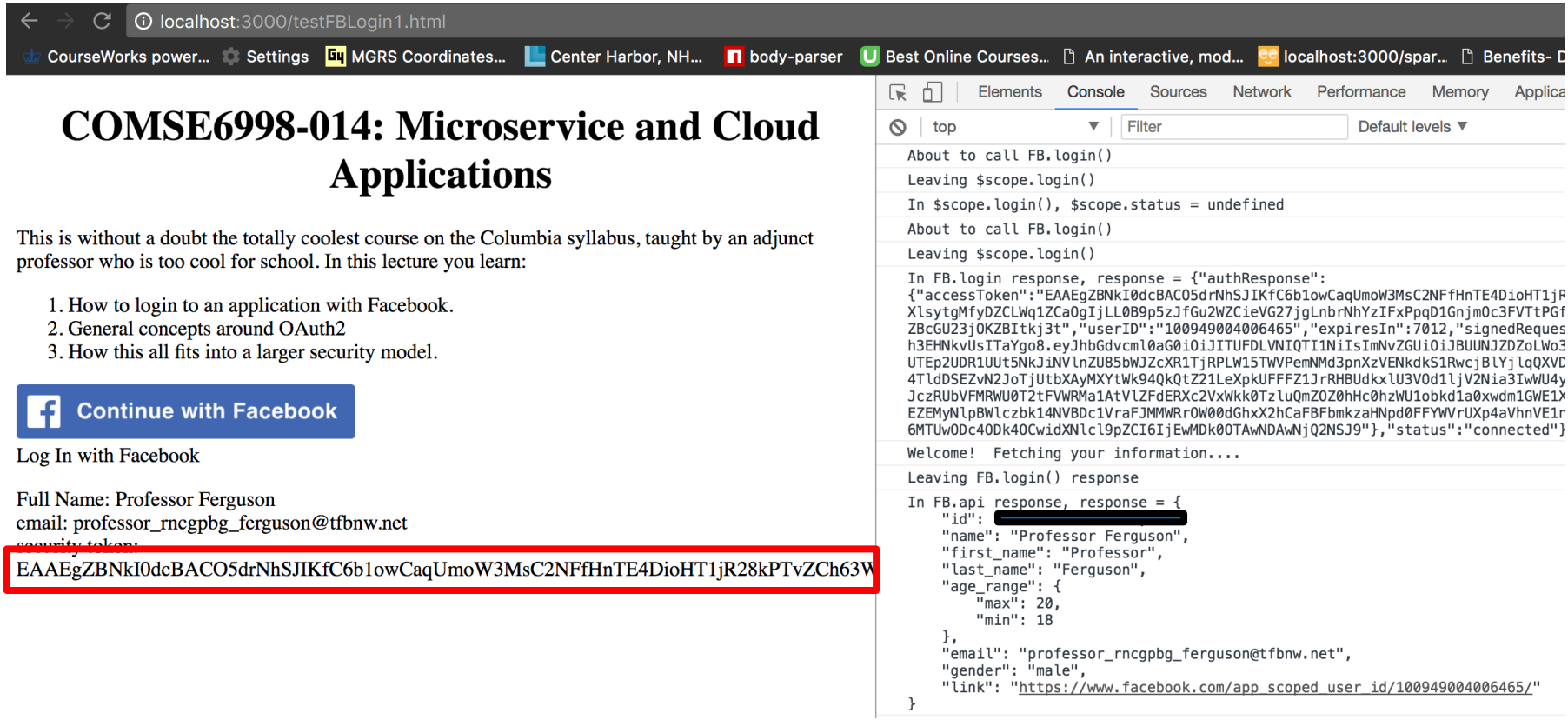
Validate Token

```
public class FBConnector {

    private static final Version FACEBOOK_APP_VERSION = Version.VERSION_2_6;

    private static final org.slf4j.Logger logger = LoggerFactory.getLogger(FBConnector.class);

    public static Gson prettyGson = new GsonBuilder().setPrettyPrinting().create();

    public static boolean validateUIDAndToken(String accessToken, String uid) {
        boolean authenticated = false;
        User theUser = null;

        try {
            theUser = FBConnector.fetchUserObjectAsJsonObject(accessToken, uid);

            if (theUser != null) {
                authenticated = true;
            }
        } catch (Exception e) {
            System.out.println("Error = " + e.toString());
            e.printStackTrace();
        }

        return authenticated;

    }
```

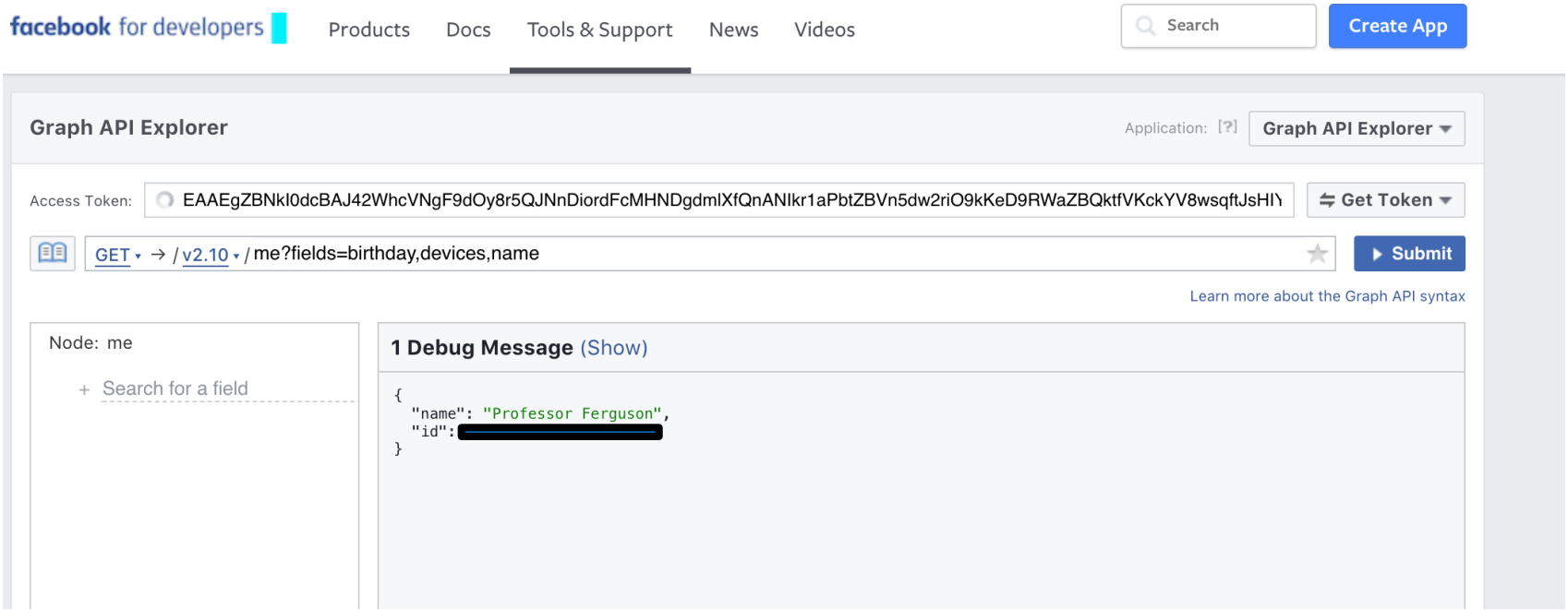
Validate Token

```
public static User fetchUserObjectAsJsonObject(String accessToken, String uid) {
    User theUser = null;

    try {
        logger.debug("In FB Connector.");

        DefaultFacebookClient facebookClient23 = new DefaultFacebookClient(accessToken, FACEBOOK_APP_VERSION);

        logger.debug("Got Facebook client");
        logger.debug("Calling FB for ID = " + uid);

        // Make the API call
        theUser = facebookClient23.fetchObject(uid, User.class);

        // logger.debug("Back from FB call, result = " +
        // prettyGson.toJson(theUser));
        logger.debug("Last name = " + theUser.getLastName());

        // Based on the FB App ID and access token, we sometimes get the
        // last_name and first_name,
        // and sometimes just the full name, e.g. "Hermione Grainger."
        if ((theUser.getLastName() == null) || (theUser.getFirstName() == null)) {
            theUser = updateNames(theUser);
        }
    } catch (Exception e) {
        System.out.println("Error = " + e.toString());
        e.printStackTrace();
        theUser =   null;
    }

    return theUser;

}
```

Facebook Token



Graph API Explorer Demo



OAuth

Twitter and Others

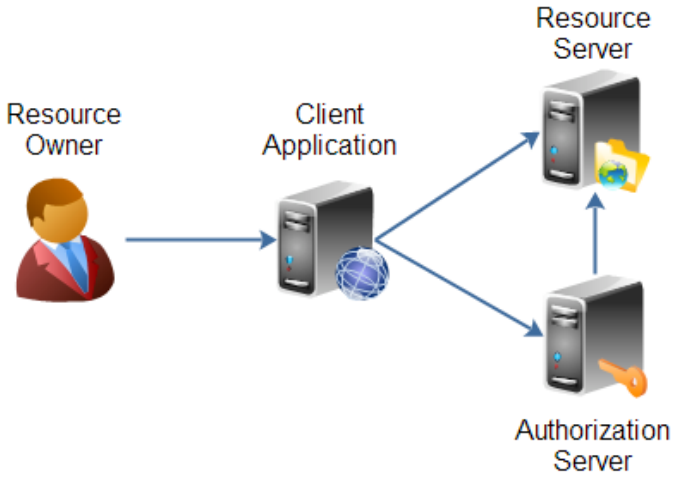
Overview (http://tutorials.jenkov.com/oauth2/index.html)

- Resource Owner
  - Controls *access* to the "data."
  - Facebook **user**, LinkedIn **user**, …
- Resource Server
  - The website that holds/manages info.
  - Facebook, LinkedIn, ...
  - And provides access API.
- Client Application
  - "The product you implemented."
  - Wants to read/update
    - "On your behalf"
    - The data the data that the Resource Server maintains, e.g. posts, status, tweets, ...
- Authorization Server
  - Grants/rejects authorization
  - Based on Resource Owner decisions.
  - Usually (logically) the same as Resource Server.

OAuth 2.0 defines the following roles of users and applications:

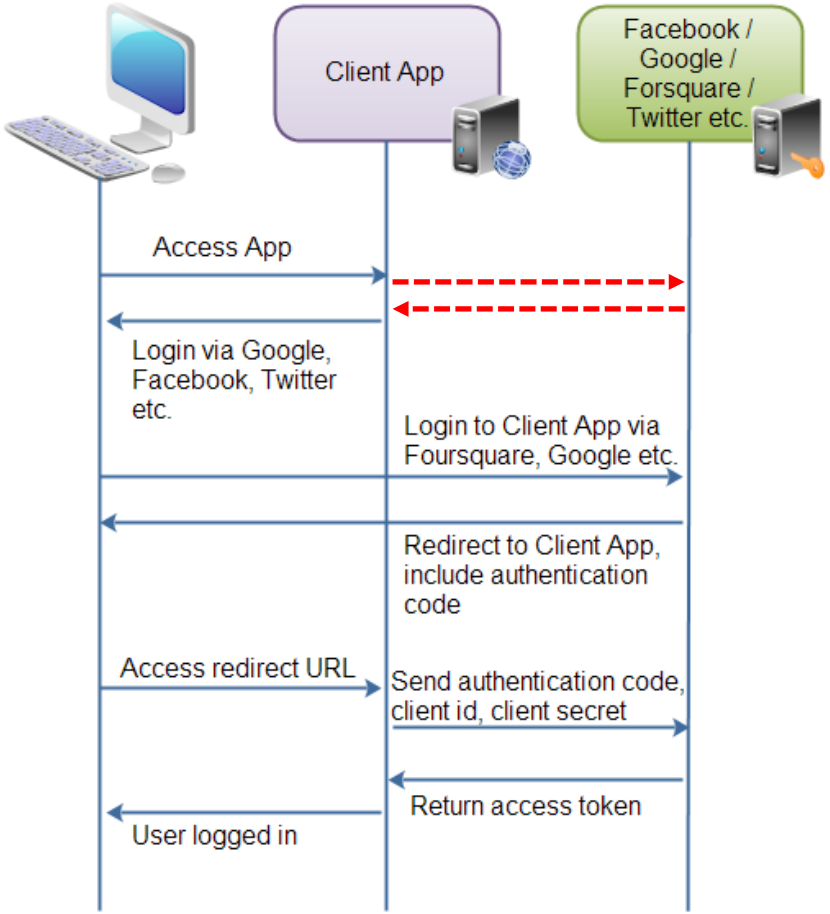
- Resource Owner
- Resource Server
- Client Application
- Authorization Server

These roles are illustrated in this diagram:



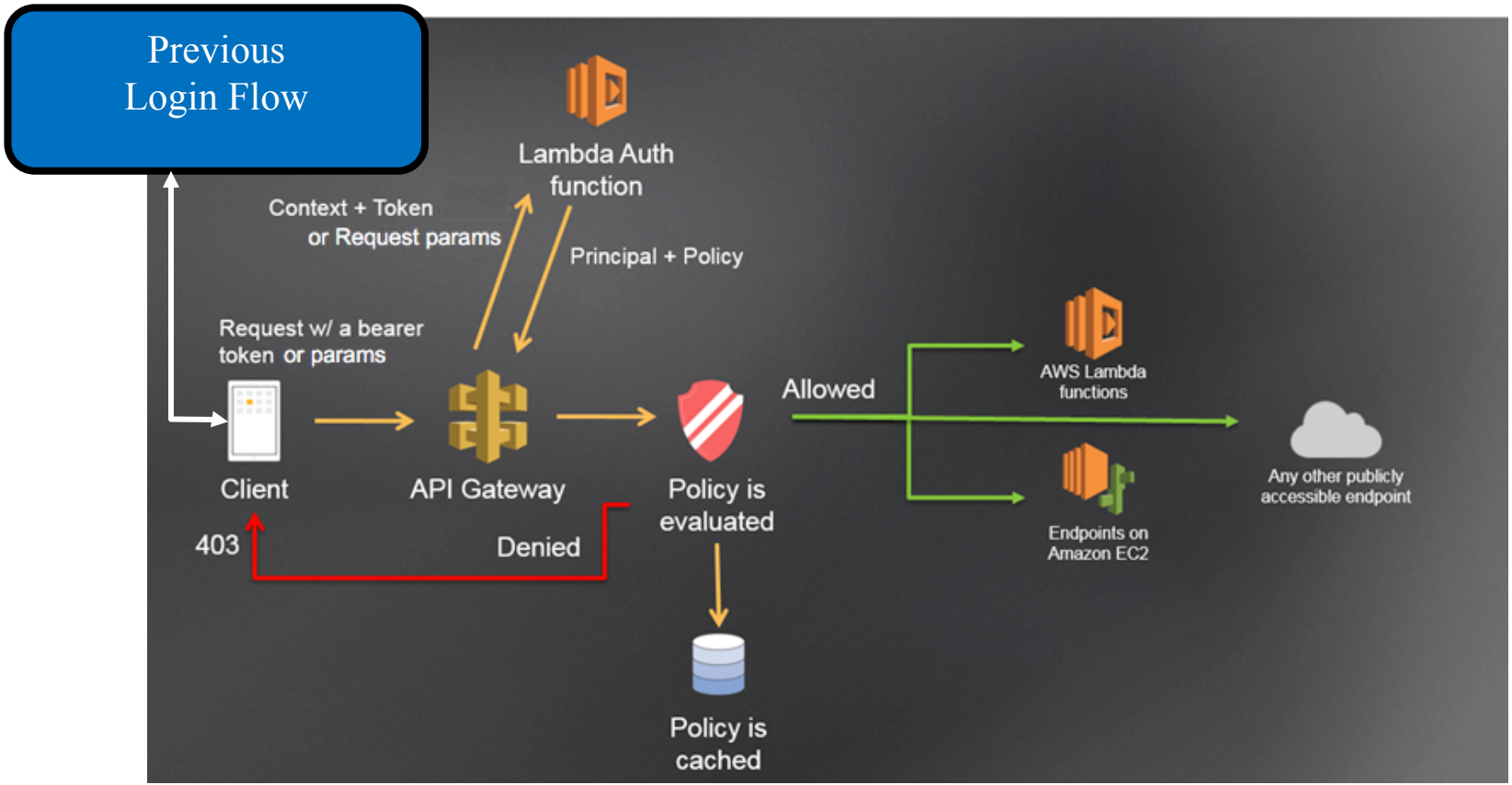
Roles/Flows

- User Clicks “Logon with XXX”
  - Redirect user to XXX
  - With Client App application ID.
  - And permissions app requests.
  - Code MAY have to call Resource Server to get a token to include in redirect URL.
- Browser redirected to XXX
  - Logon on to XXX prompt.
  - Followed by “do you grant permission to …”
  - Clicking button drives a redirect back to Client App. URL contains a temporary token.
- User/Browser
  - Redirected to Client App URL with token.
  - Client App calls XXX API
  - Obtains access token.
  - Returns to User.
- Client App can use access token on API calls.



Authorization

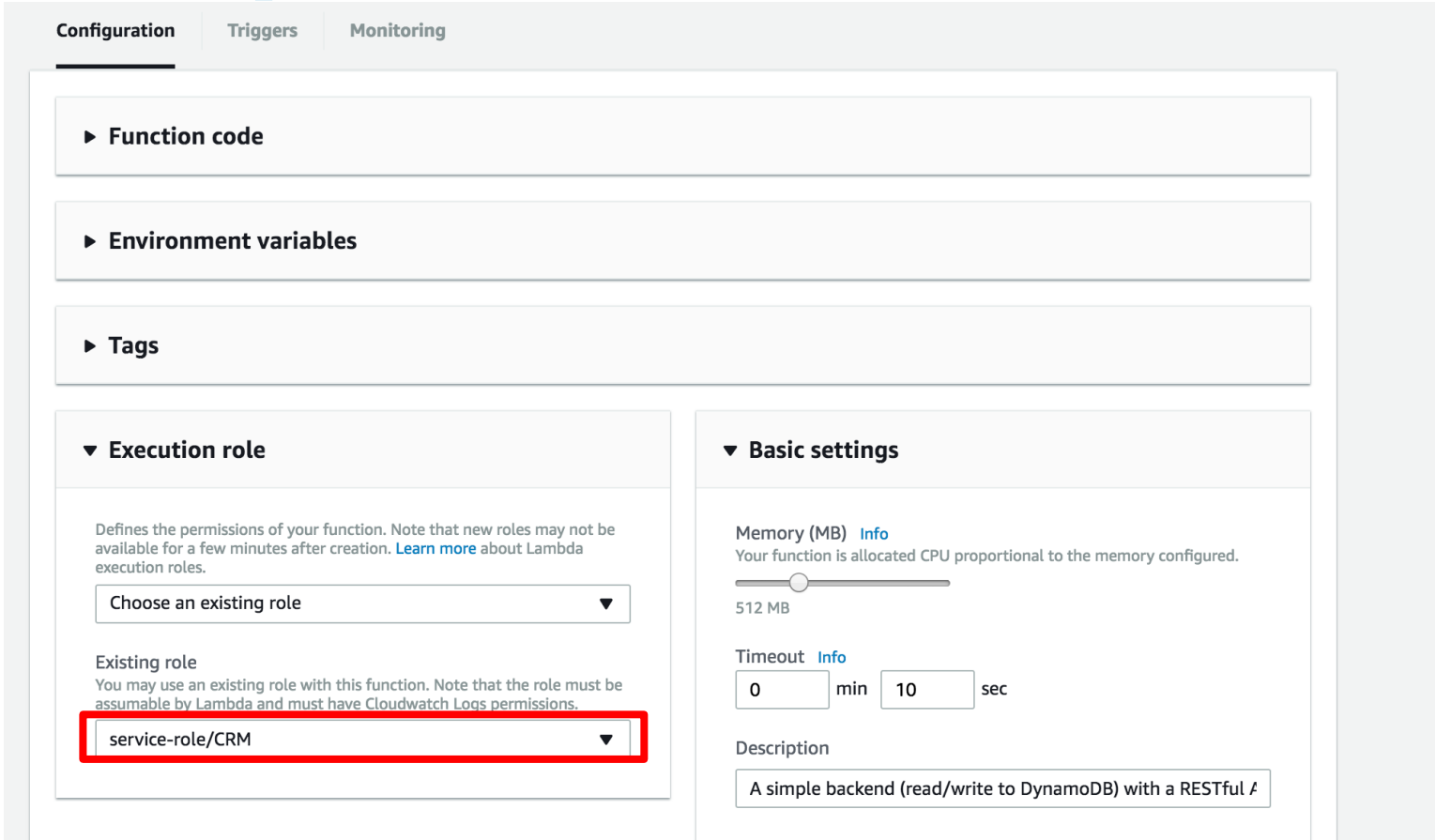
Custom Authorizer



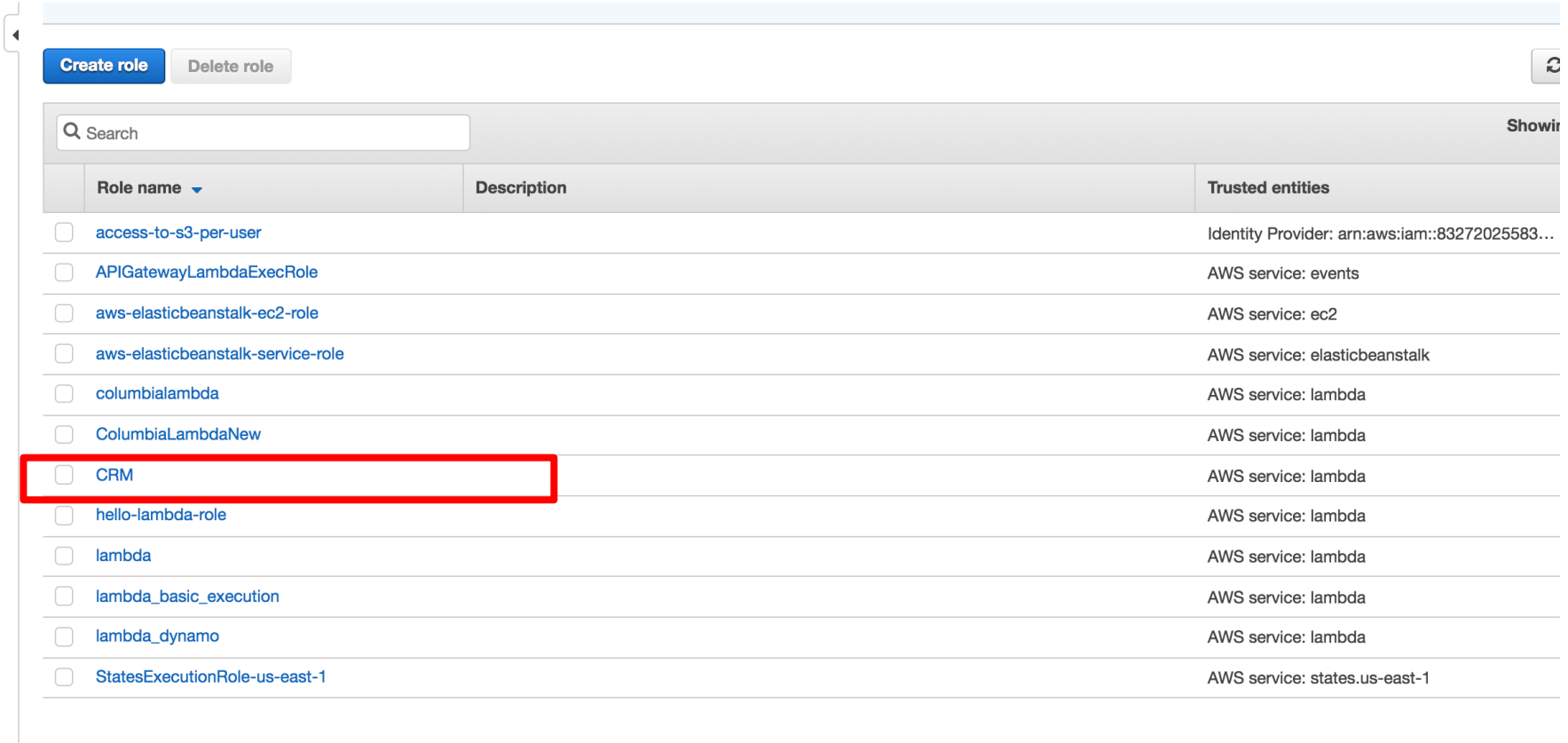
Authorization

- 1st check occurs at perimeter in API GW Authorization
  - Based on (bearer token) JWT token
  - Is this role authorized to invoke this URL?
  - Implementation
    - Lambda function that …
    - Checks an authorization table (role, URL)
    - Or the rights are encoded in the token.
- All down stream microservices
  - Receive the JWT token and perform their own authorization decisions.
  - Run with a set of AWS roles and permissions that determine if AWS allows access, which may include user role.

An Example – Lambda Function Role



Role/Policies



Role/Policy