**Why oauth2.0:**

## [What Is OAuth?](https://developer.okta.com/blog/2017/06/21/what-the-heck-is-oauth#what-is-oauth)

To begin at a high level, OAuth is not an API or a service: it’s an open standard for authorization and anyone can implement it.

More specifically, OAuth is a standard that apps can use to provide client applications with “secure delegated access”. OAuth works over HTTPS and authorizes devices, APIs, servers, and applications with access tokens rather than credentials.

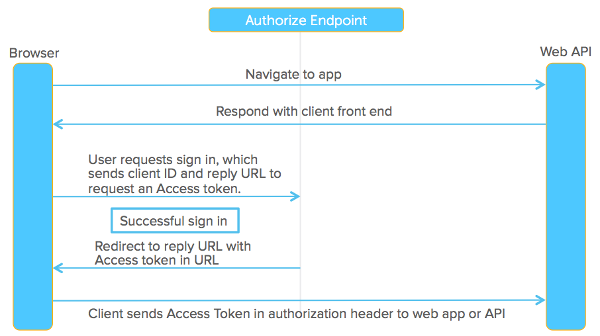
There are two versions of OAuth: [OAuth 1.0a](https://tools.ietf.org/html/rfc5849) and [OAuth 2.0](https://tools.ietf.org/html/rfc6749). These specifications are completely different from one another, and cannot be used together: there is no backwards compatibility between them.

Which one is more popular? Great question! Nowadays, OAuth 2.0 is the most widely used form of OAuth. So from now on, whenever I say “OAuth”, I’m talking about OAuth 2.0 – as it’s most likely what you’ll be using.

## [Why OAuth?](https://developer.okta.com/blog/2017/06/21/what-the-heck-is-oauth#why-oauth)

OAuth was created as a response to the direct authentication pattern. This pattern was made famous by HTTP Basic Authentication, where the user is prompted for a username and password. Basic Authentication is still used as a primitive form of API authentication for server-side applications: instead of sending a username and password to the server with each request, the user sends an API key ID and secret. Before OAuth, sites would prompt you to enter your username and password directly into a form and they would login to your data (e.g. your Gmail account) as you. This is often called [the password anti-pattern](https://arstechnica.com/information-technology/2010/01/oauth-and-oauth-wrap-defeating-the-password-anti-pattern/).

To create a better system for the web, federated identity was created for single sign-on (SSO). In this scenario, an end user talks to their identity provider, and the identity provider generates a cryptographically signed token which it hands off to the application to authenticate the user. The application trusts the identity provider. As long as that trust relationship works with the signed assertion, you’re good to go. The diagram below shows how this works.

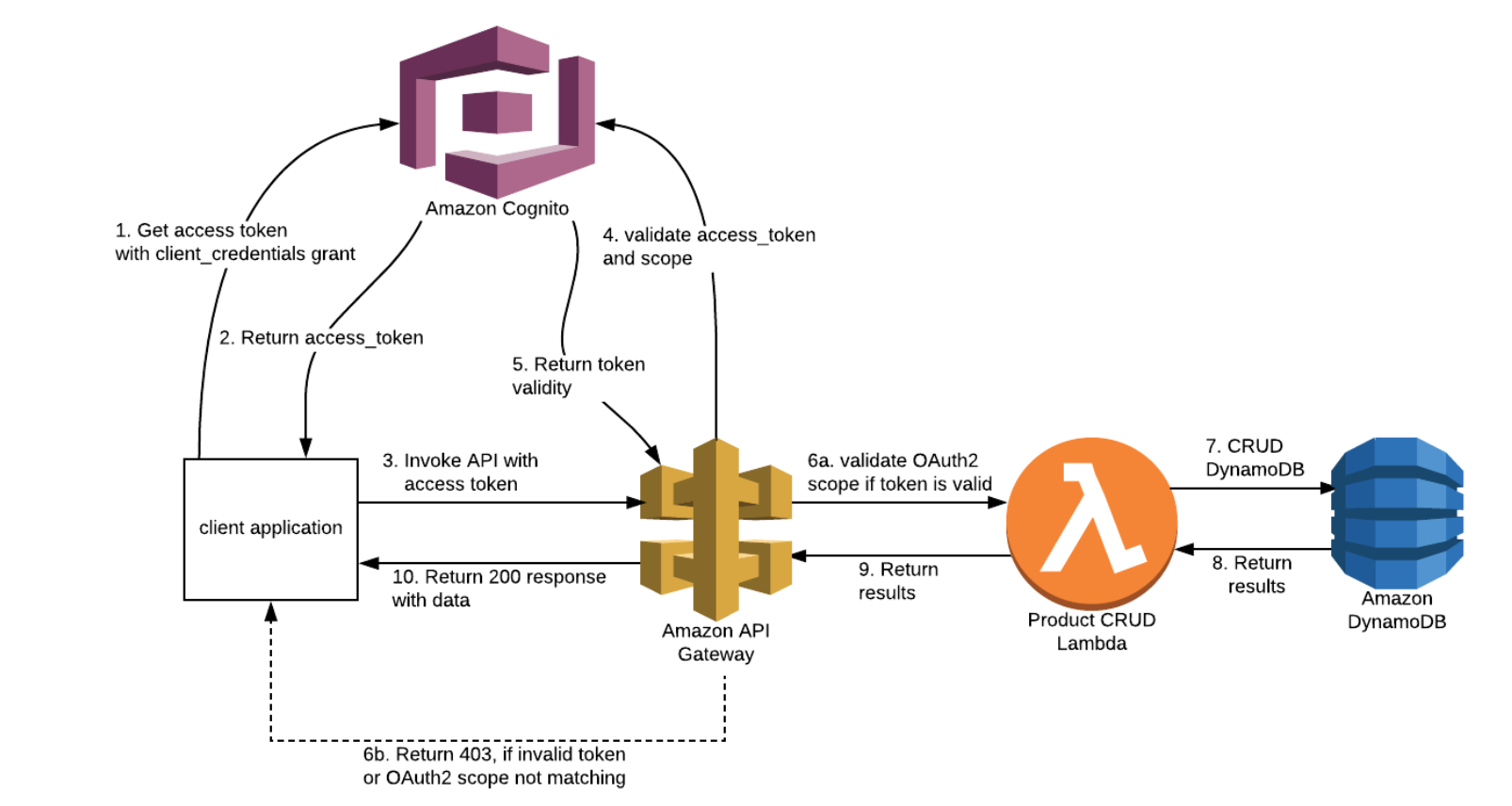


Federated identity was made famous by SAML 2.0, an OASIS Standard released on March 15, 2005. It’s a large spec but the main two components are its authentication request protocol (aka Web SSO) and the way it packages identity attributes and signs them, called SAML assertions. Okta does this with its SSO chiclets. We send a message, we sign the assertion, inside the assertion it says who the user is, and that it came from Okta. Slap a digital signature on it and you’re good to go.

**Use Case :**

Any organization building an API based architecture has to build a common security layer around these APIs, basically on the edge so that all the APIs are secured. There are multiple ways to build API security like writing some filters in the case of Java / J2EE application, installing some agents in front of APIs which can make policy decisions etc. One of the most widely used protocol for Authorization is OAuth2. AWS API Gateway provides built-in support to secure APIs using AWS Cognito OAuth2 scopes.

Below is the architecture diagram:



1. Invoke AWS Cognito /oauth2/token endpoint with grant\_type as client\_credentials. Refer <https://docs.aws.amazon.com/cognito/latest/developerguide/token-endpoint.html>
2. If the request is valid, AWS Cognito will return a JWT (JSON Web Token) formatted access\_token
3. Pass this token in Authorization header for all API calls
4. API Gateway makes a call to AWS Cognito to validate the access\_token.
5. AWS Cognito returns token validation response.
6. If token is valid, API Gateway will validate the OAuth2 scope in the JWT token and ALLOW or DENY API call. This is entirely handled by API Gateway once configuration is in place
7. Perform the actual API call whether it is a Lambda function or custom web service application.
8. Return the results from Lambda function.
9. Return results to API Gateway.
10. If there are no issues with the Lambda function, API Gateway will return a HTTP 200 with response data to the client application.

There are few prerequisites for setting up this integration:

1. AWS Account — business or free tier.
2. Knowledge on AWS API Gateway , AWS Cognito services
3. Knowledge on OAuth2 protocol

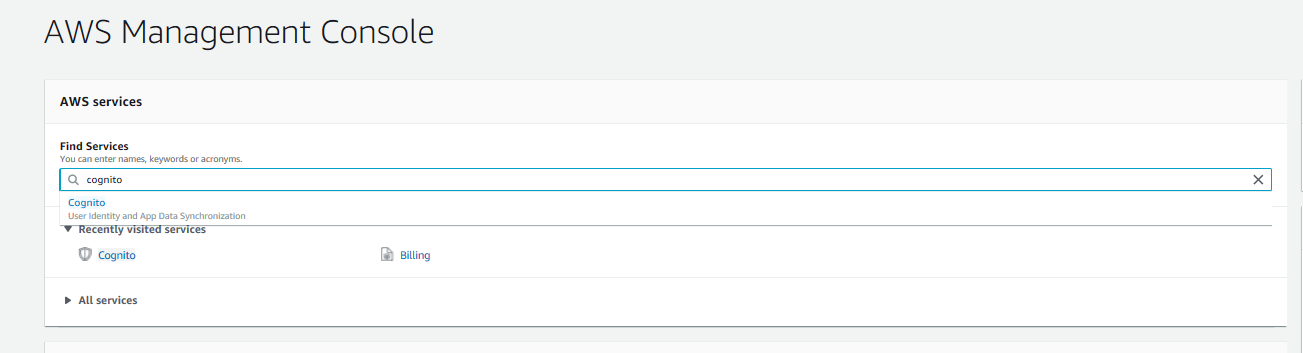
We have to perform the below steps for this integration:

1. Create a AWS Cognito user pool and configure OAuth agents
2. Deploy a sample micro webservice application using AWS API Gateway and Lambda/SQS
3. Configure Cognito Authorizer in API Gateway

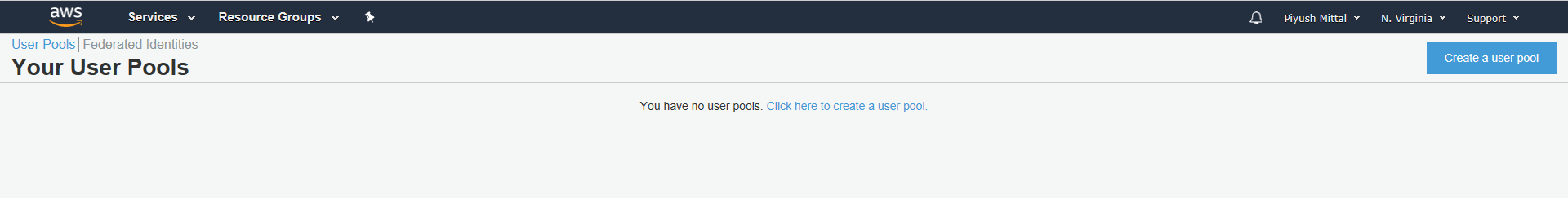
#### Step 1: Create AWS Cognito user pool and setup a OAuth application

* Login to AWS Management console and navigate to Cognito service
* Select “Manage your user pools” and click “Create a user pool”
* Enter a pool name and select “Review defaults”. Then select “Create pool”.

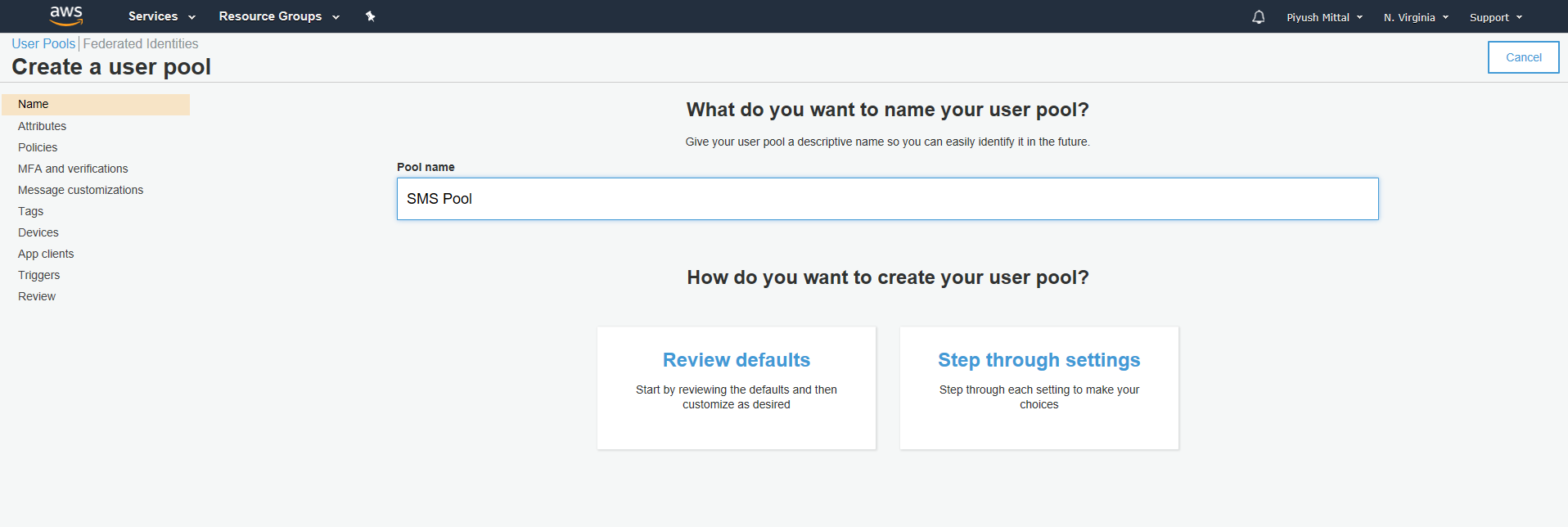
Goto AWS Management Console and search for cognito service.



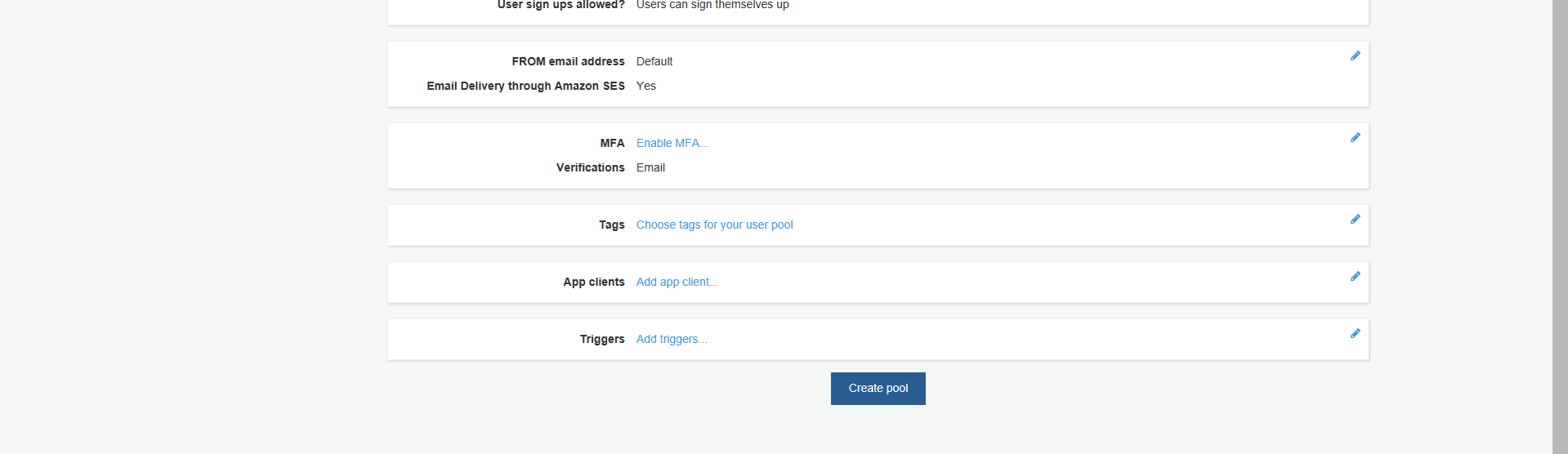
Click on “Create a user pool”



Give pool name and click on “Review defaults”

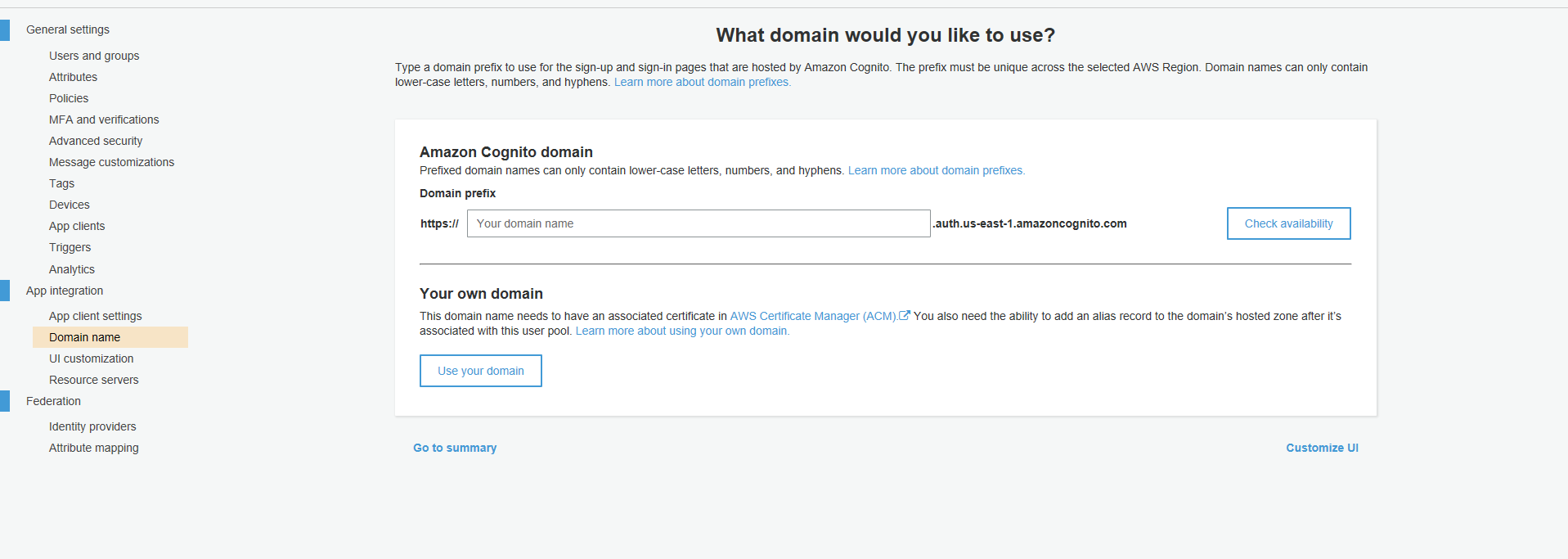


Then click on “Create pool”

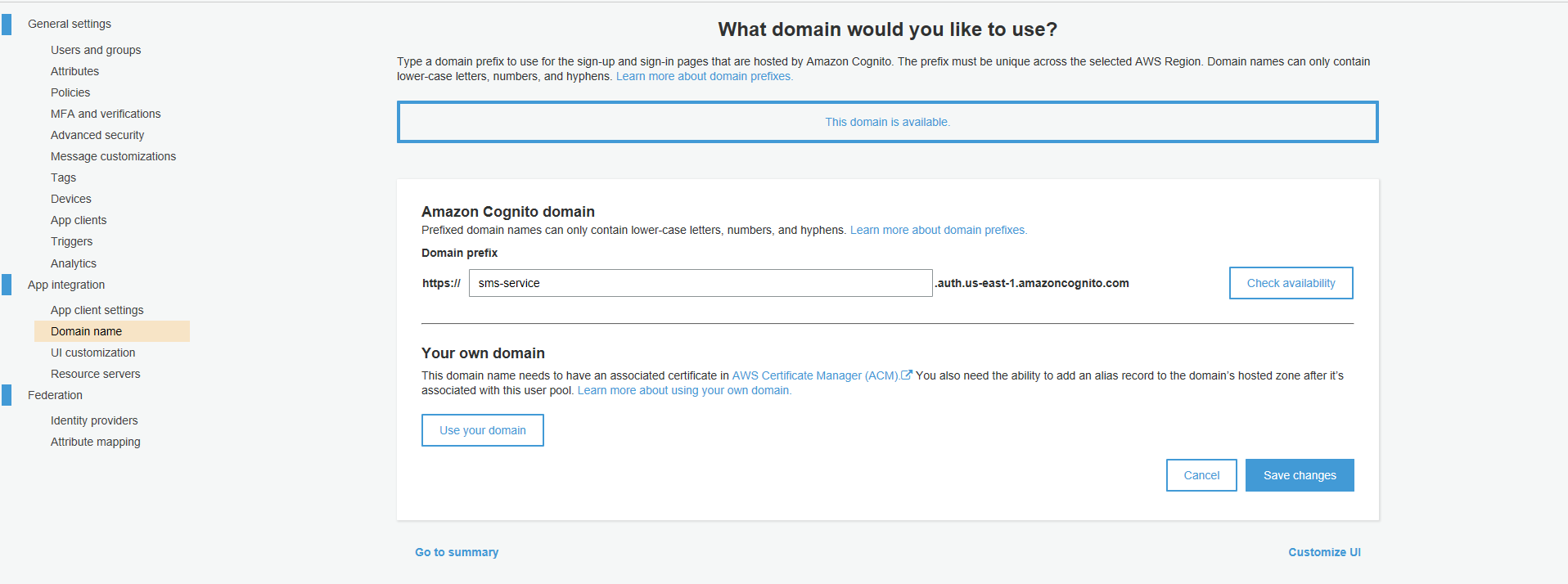


Now “user pool” has been created.

After creating “user pool” create a “Domain name”

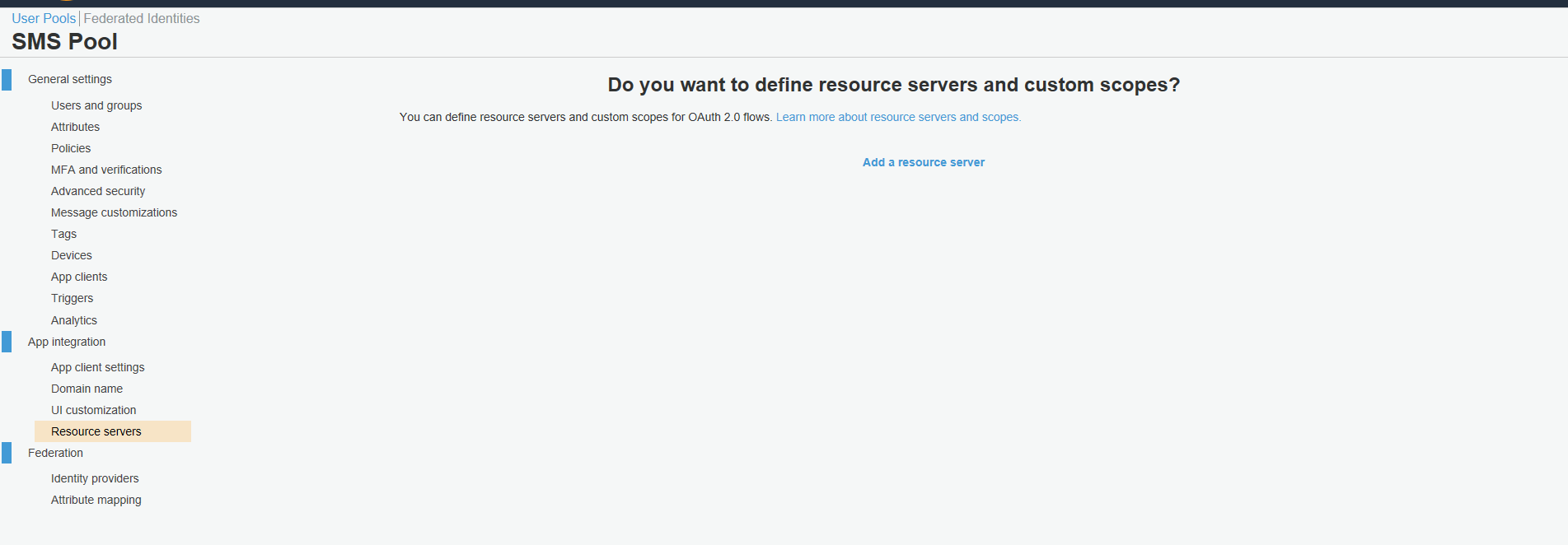


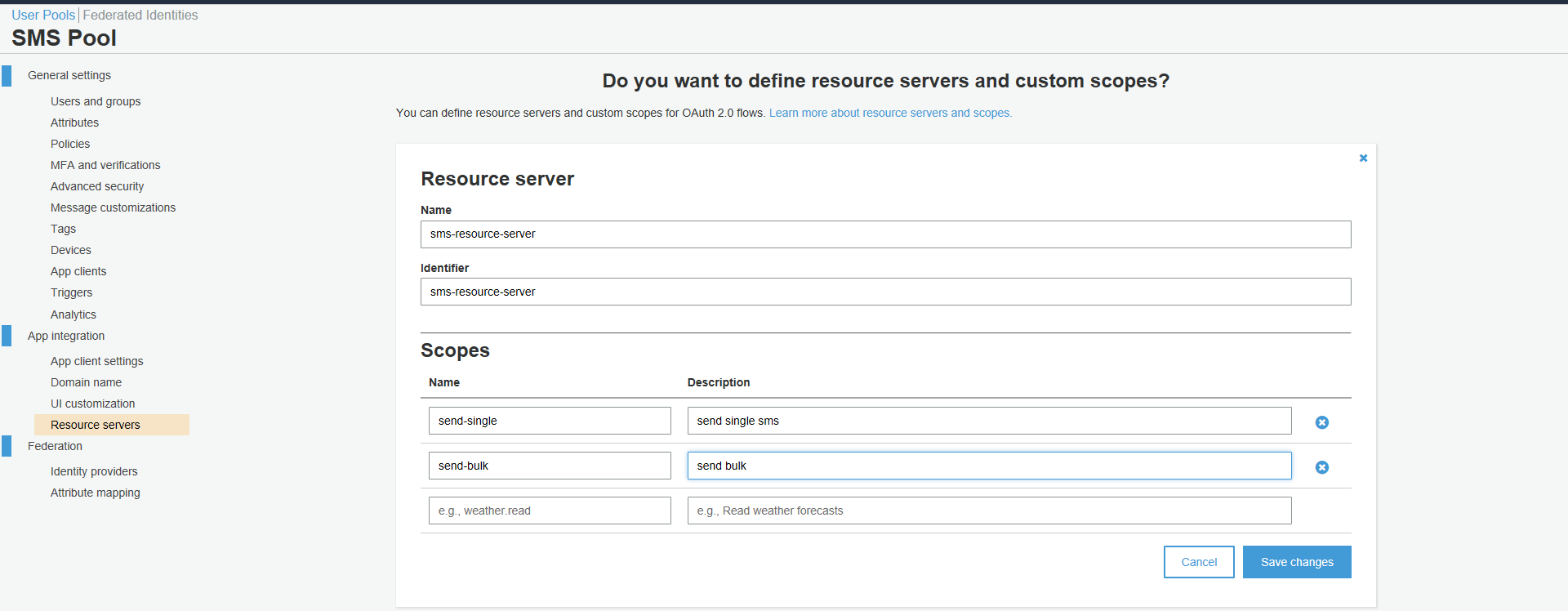
Before creating user pool first check availability of user pool. Once it is available click on “save changes”.



After saving changes create “Resource servers” and provide scopes

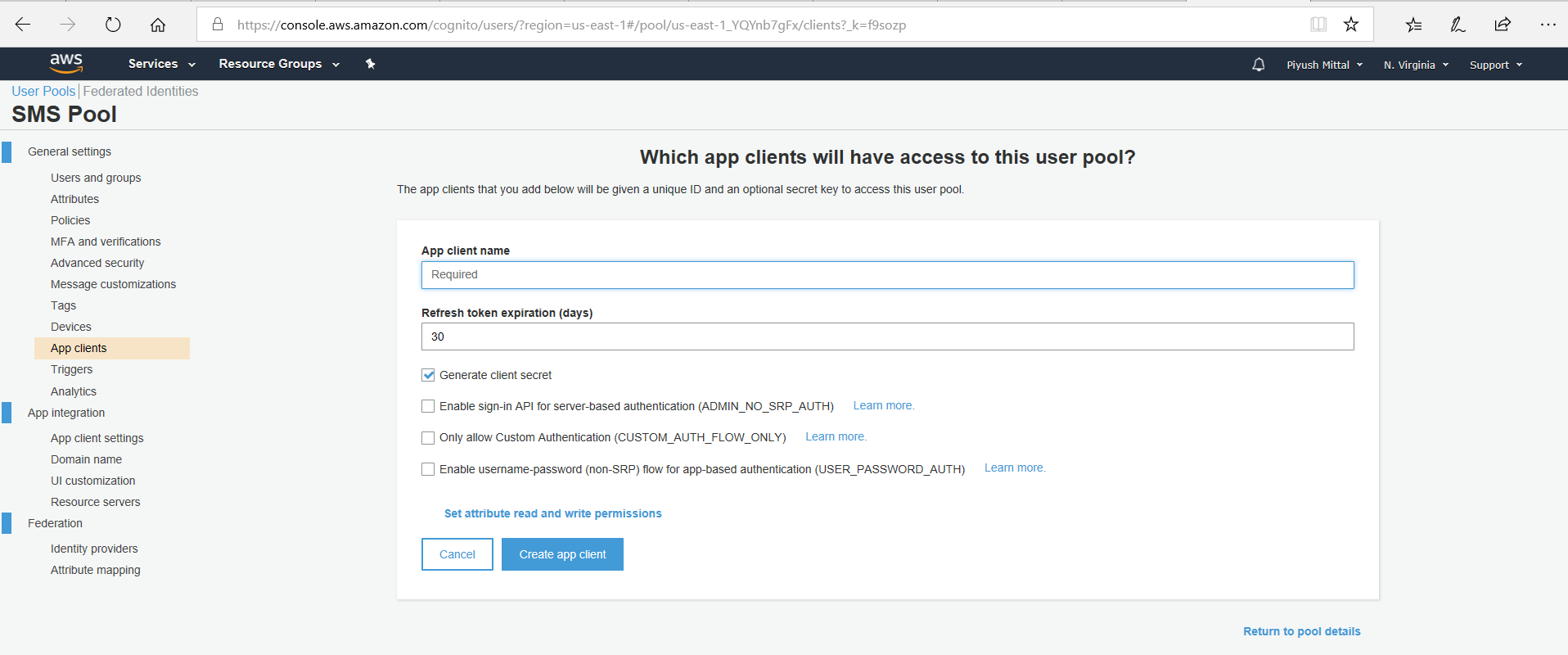
Click on Resource server





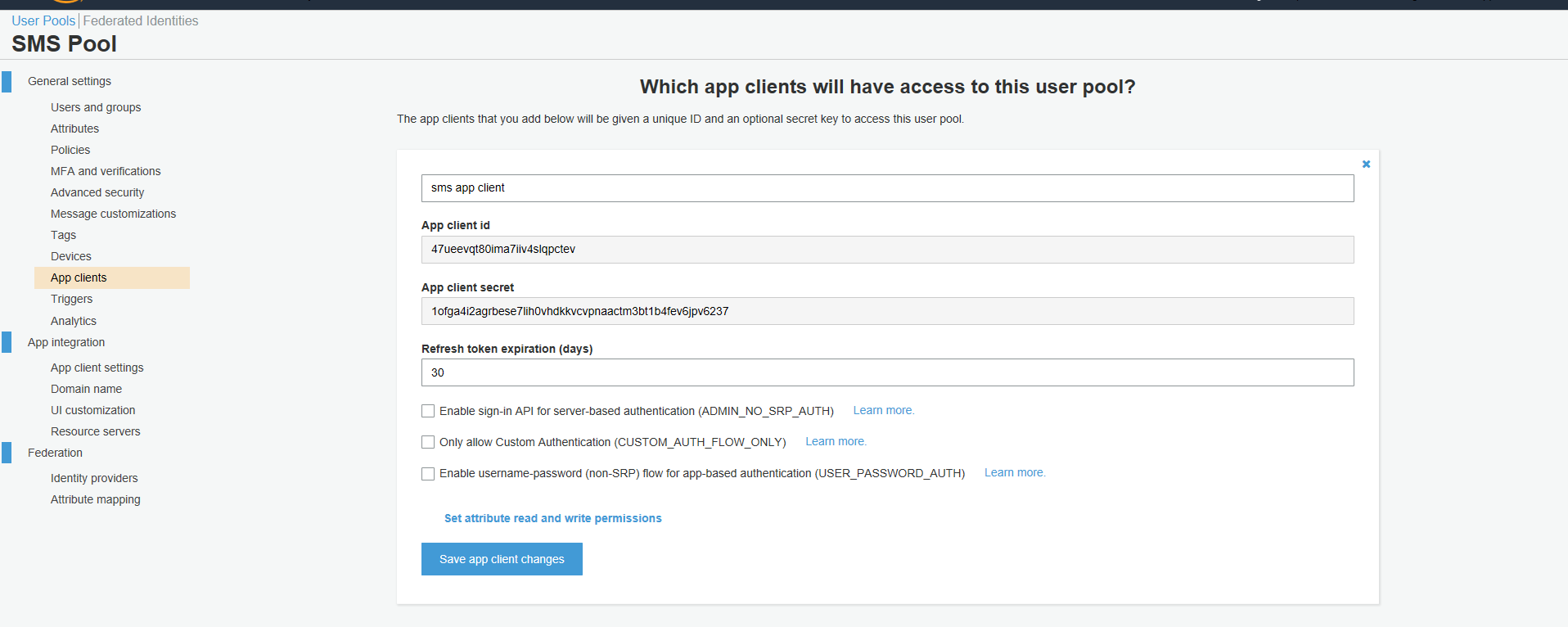
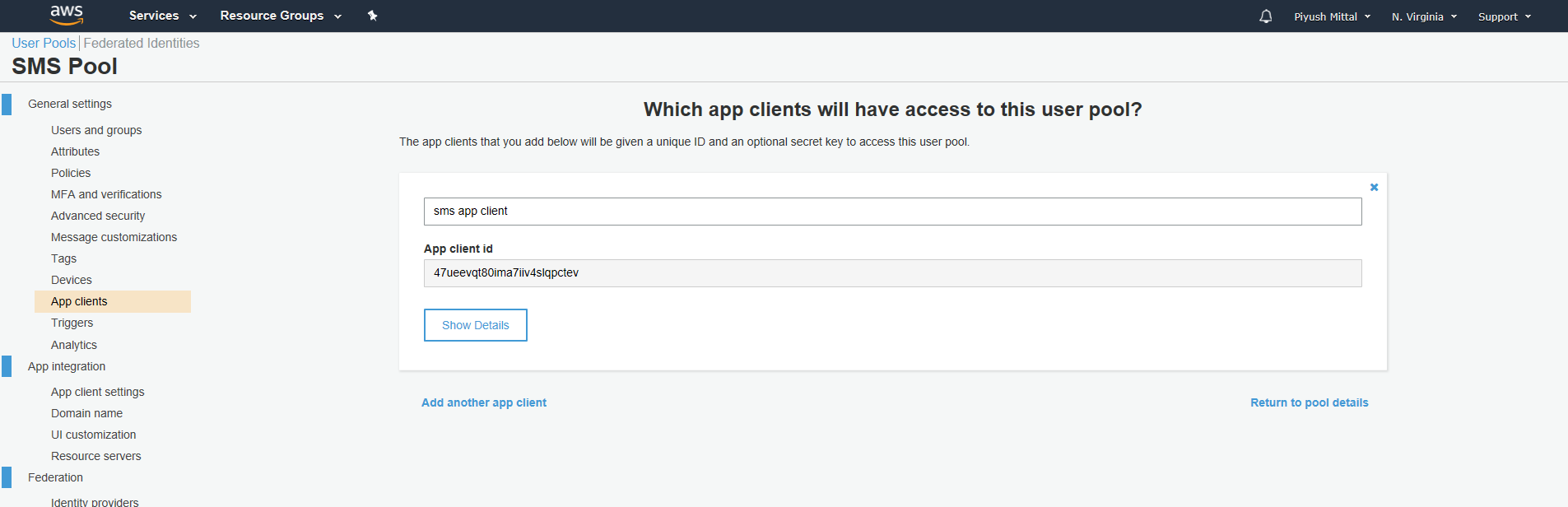
After creating resource server create an “App client”

Provide “App client name” and click on create app client.

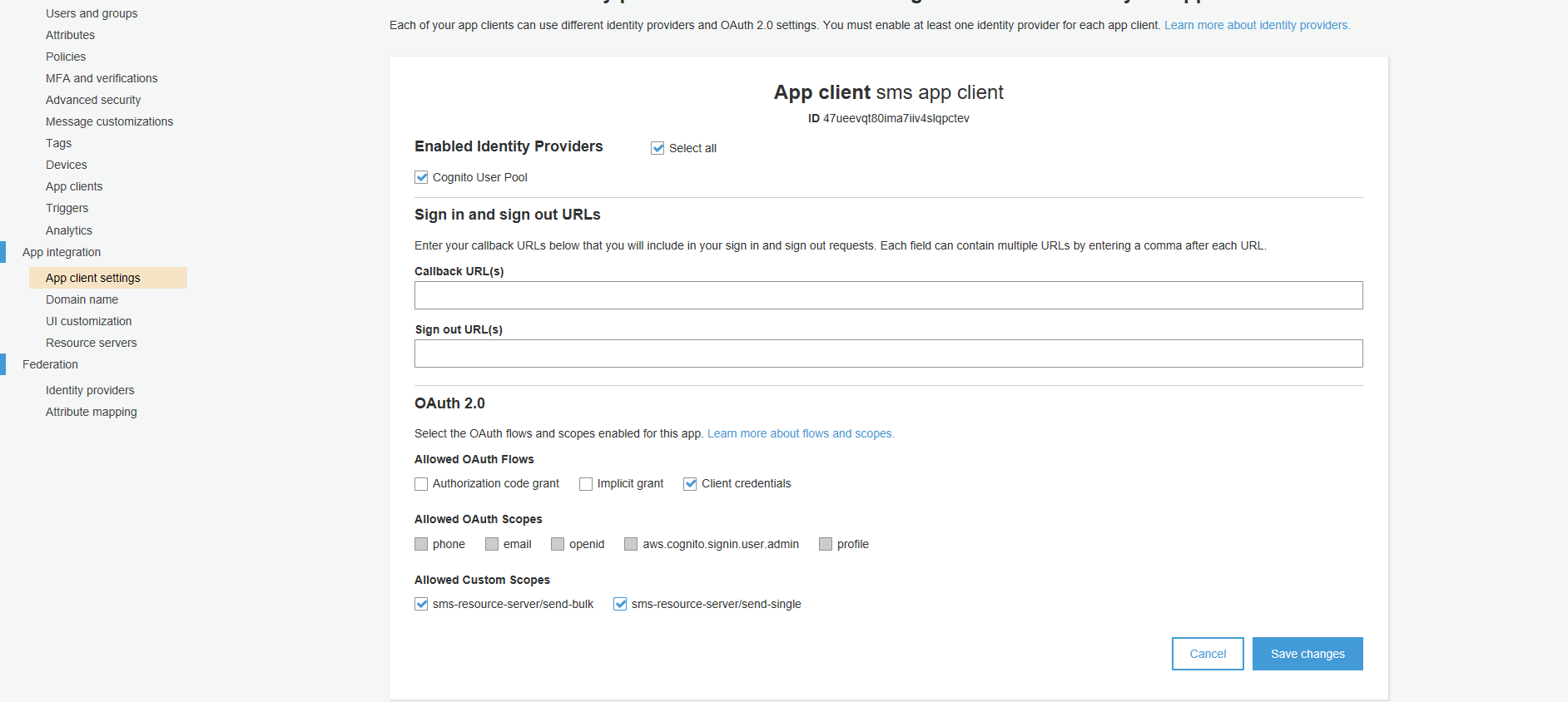
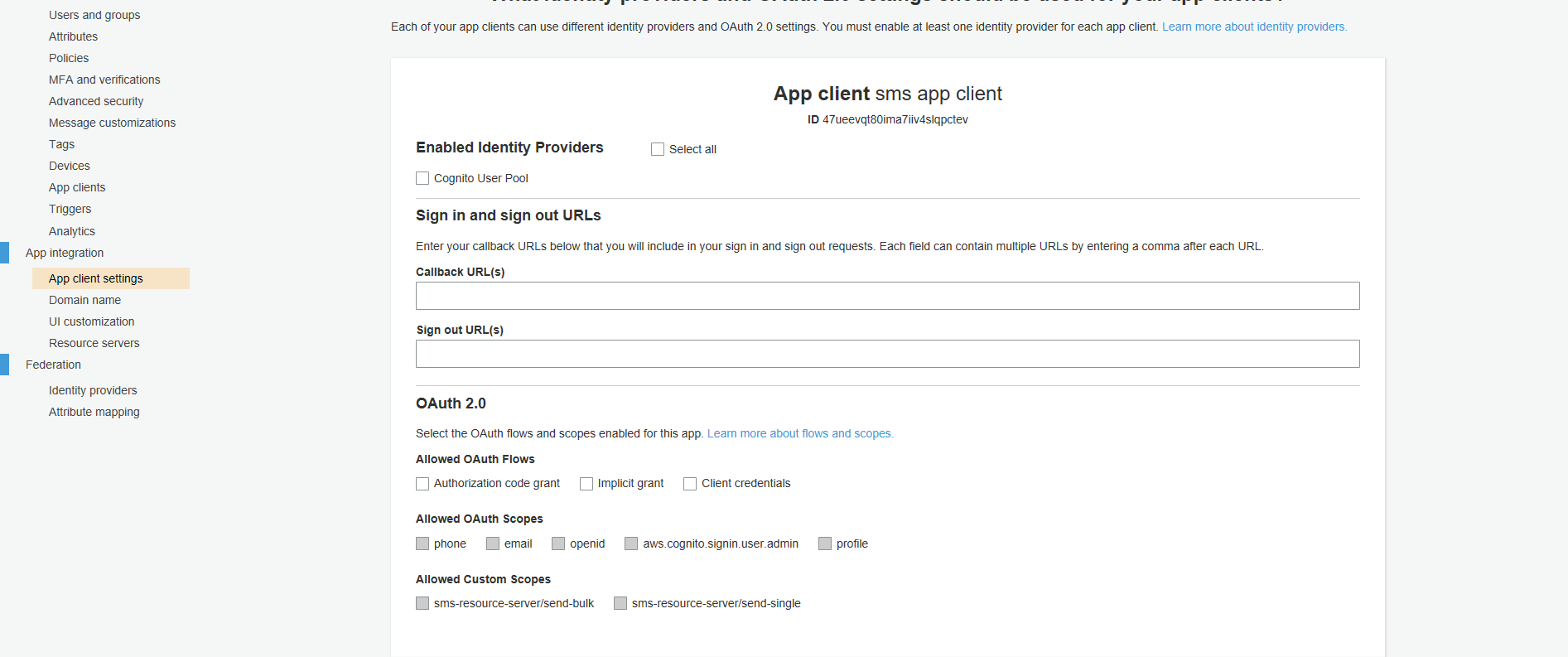


Once app client is created app client id and app client secret would be available.

Click on show details to get app client id and client secret.

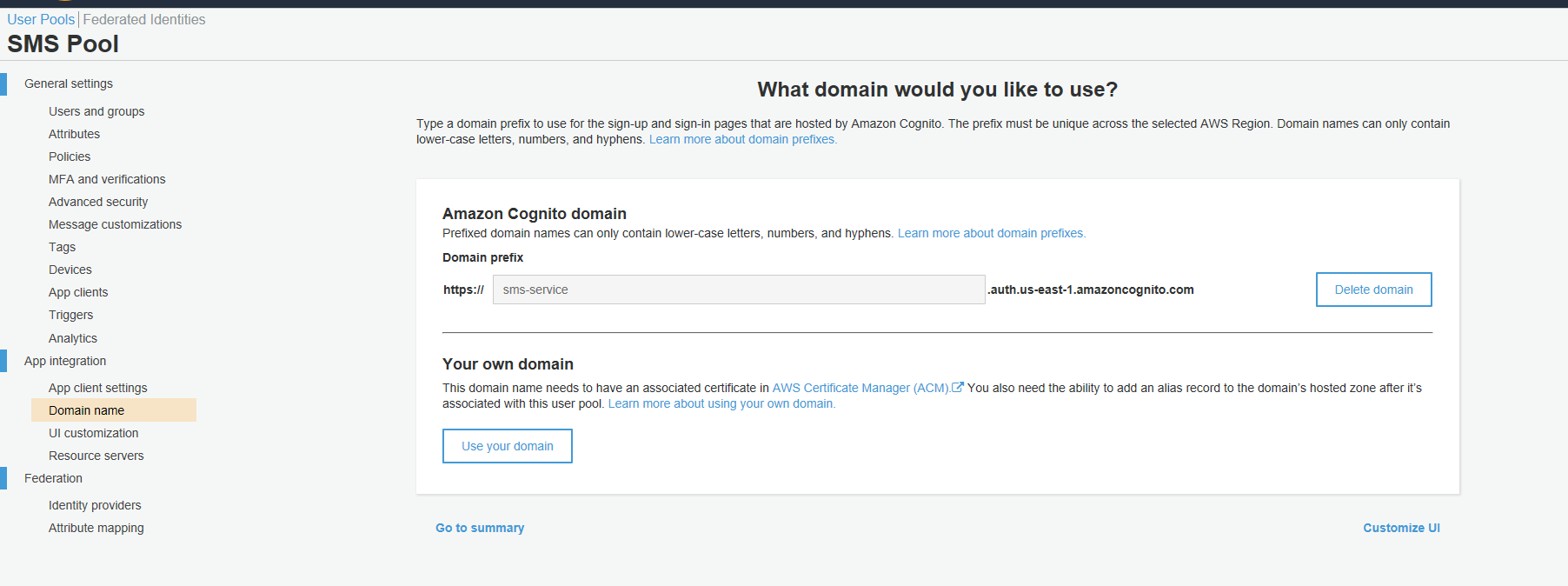


Now click on “App client settings” and check “Select all” and “Client credentials”. Then save changes.

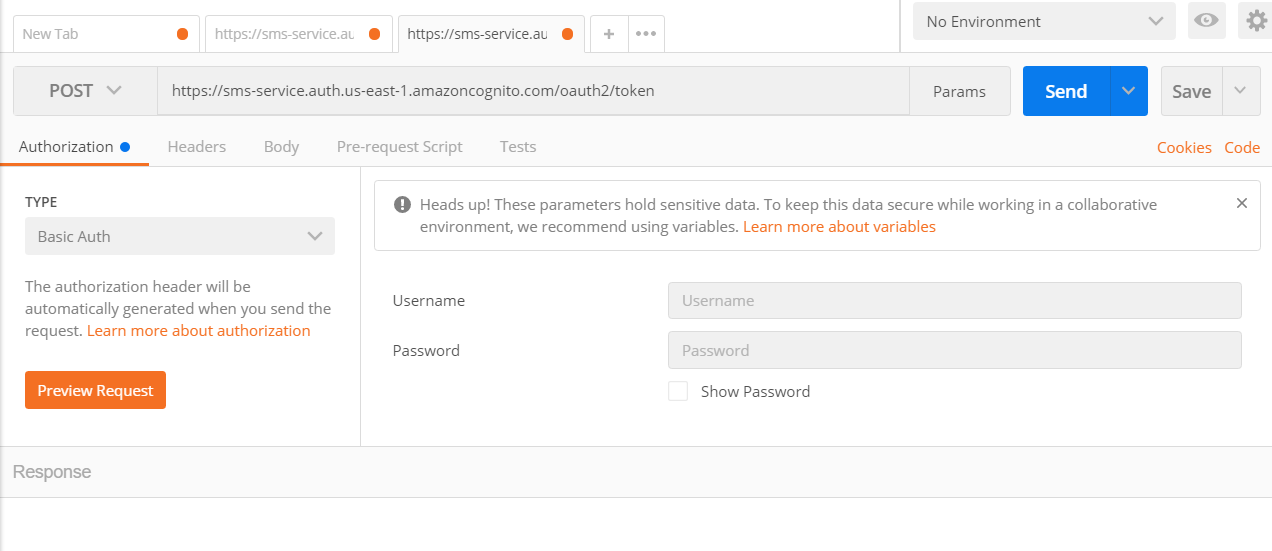


Now goto “Domain name” and copy “Amazon Cognito domain”.

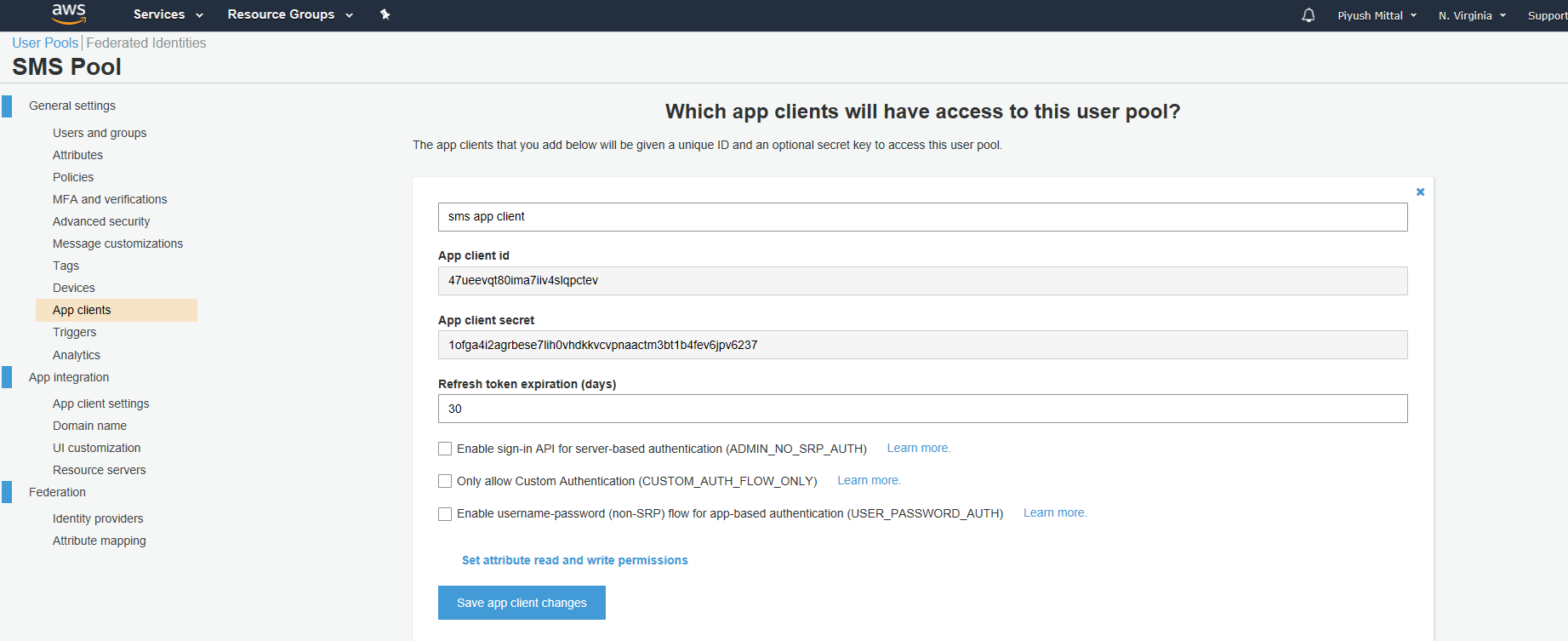
This api will be used to generate auth\_token using scopes.



Copy above domain. Open POSTMAN paste domain in post type request. Click on Authorization. tab

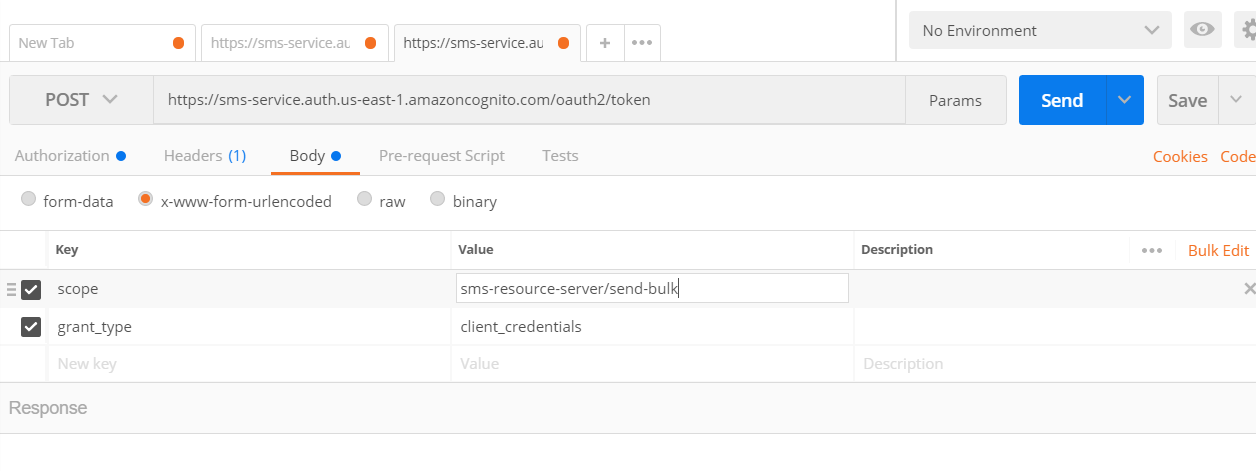


Copy App clinet id and App client secret and paste Appclient in Username and App client secret in password.



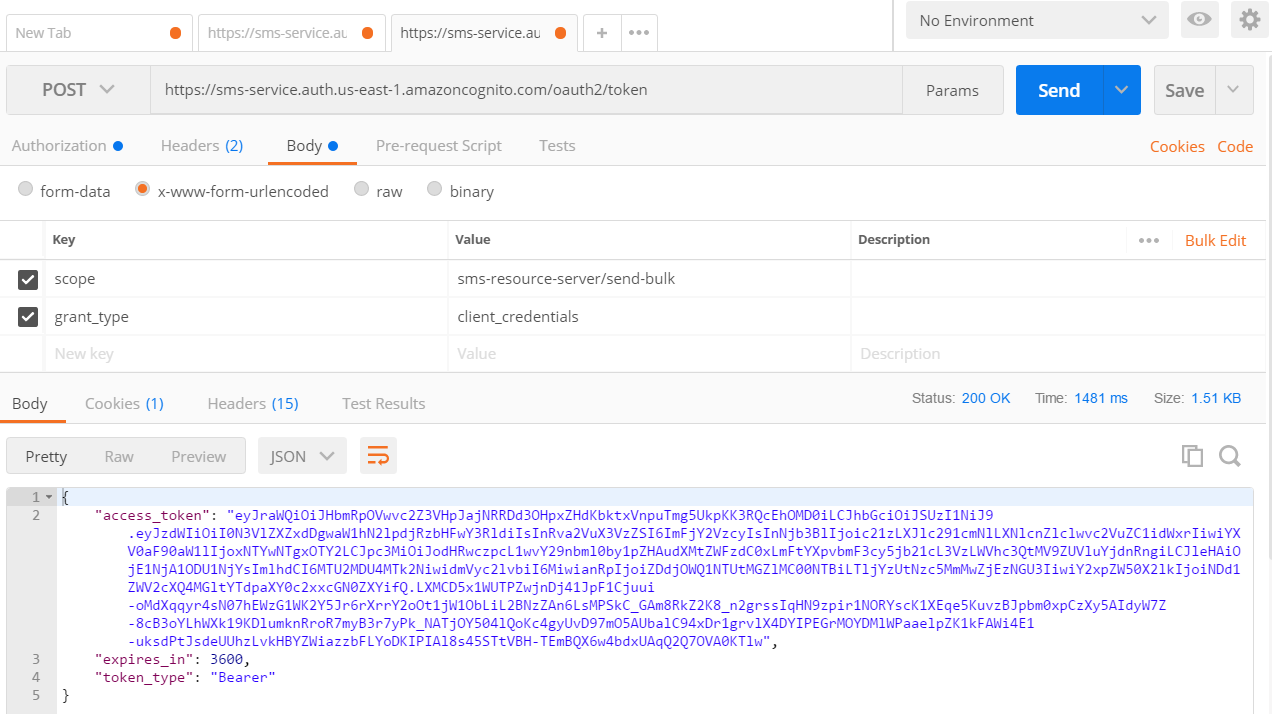
Now click on Body tab and put two keys “scope” and “grant\_type”.

“client\_credential” would be your grant\_type and provide already created scopes in scope.



Now post the request by clicking on “Send” button.

It will retuen “access\_token”



Cont…