review

lab title

Authentication and Synchronization of JavaScript Apps with AWS Cognito V1.00

Course title

BackSpace Academy
AWS Certified Associate

Table of Contents

Contents

About the Lab

These lab notes are to support the instructional videos with AWS in the BackSpace AWS Certified Associate preparation course.

Please refer to the AWS JavaScript SDK documentation at:

http://docs.aws.amazon.com/AWSJavaScriptSDK/latest/AWS/S3.html

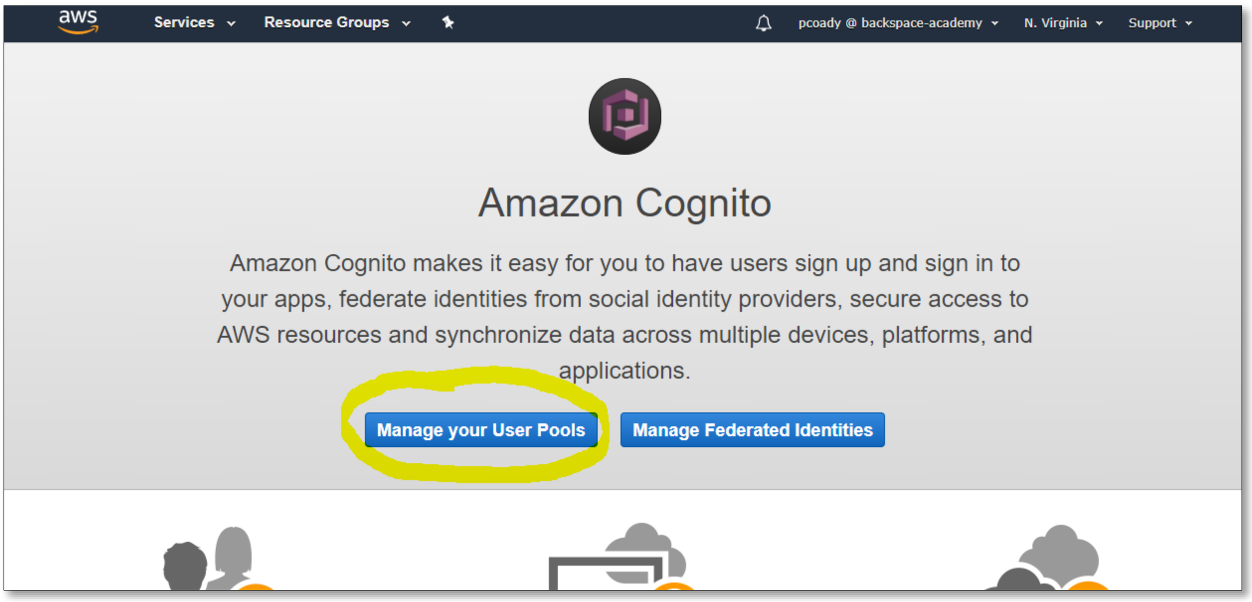
Please note that AWS services change on a weekly basis and it is extremely important you check the version number on this document to ensure you have the lastest version with any updates or corrections. The videos may not be as current as these lab notes so please follow these lab notes carefully.

Creating a Cognito User Pool

In this section we will use the Cognito service to create a user pool of authenticated users.

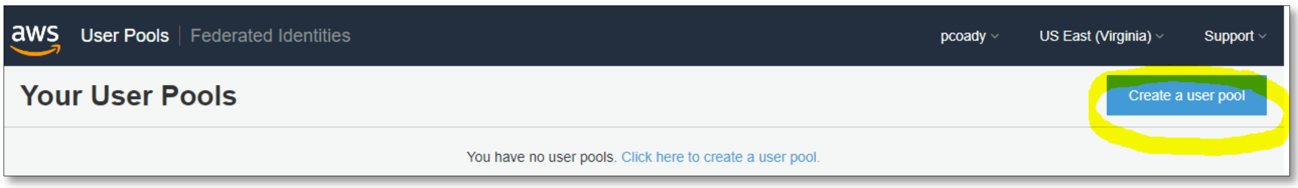
Select the Cognito Console.

Click "Manage your User Pools"



Amazon Cognito makes it easy for you to have users sign up and sign in to your apps, federate identities from social identity providers, secure access to AWS resources and synchronize data across multiple devices, platforms, and applications.

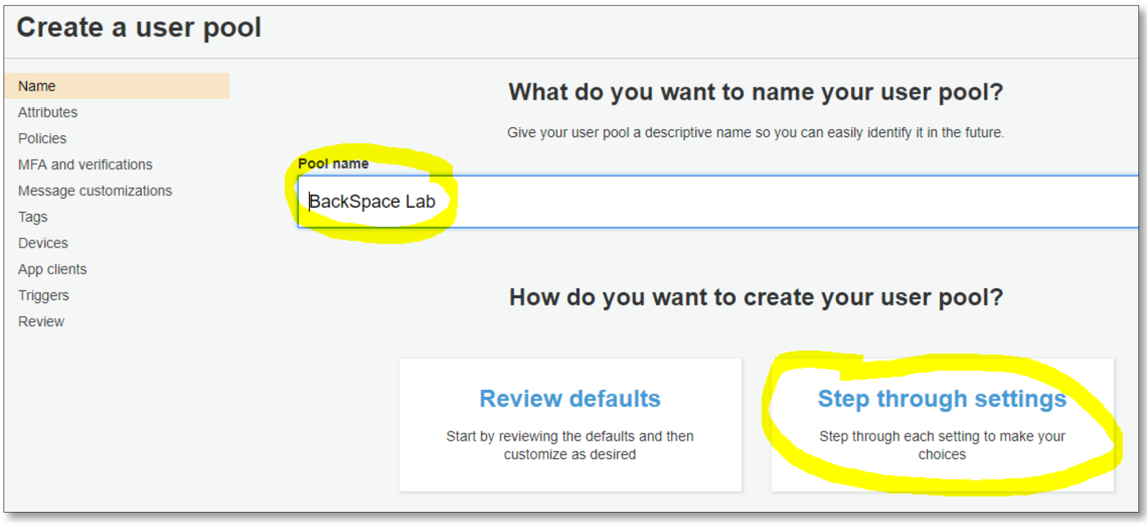
Click "Create a user pool"



You have no user pools. Click here to create a user pool.

Give your user pool a name

Click "Step through settings"



Create a user pool

What do you want to name your user pool?

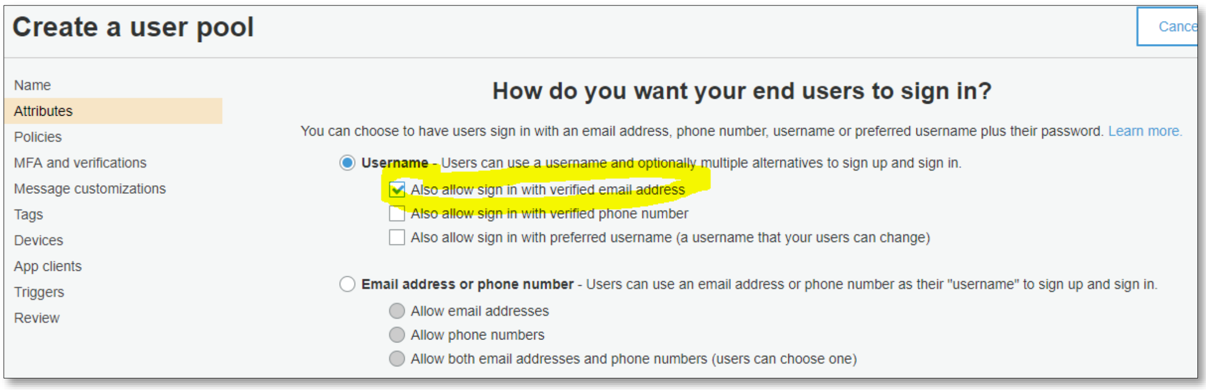
Give your user pool a descriptive name so you can easily identify it in the future.

Pool name: BackSpace Lab

How do you want to create your user pool?

Review defaults
Start by reviewing the defaults and then customize as desired
Step through settings
Step through each setting to make your choices

Select “Also allow sign in with verified email address”



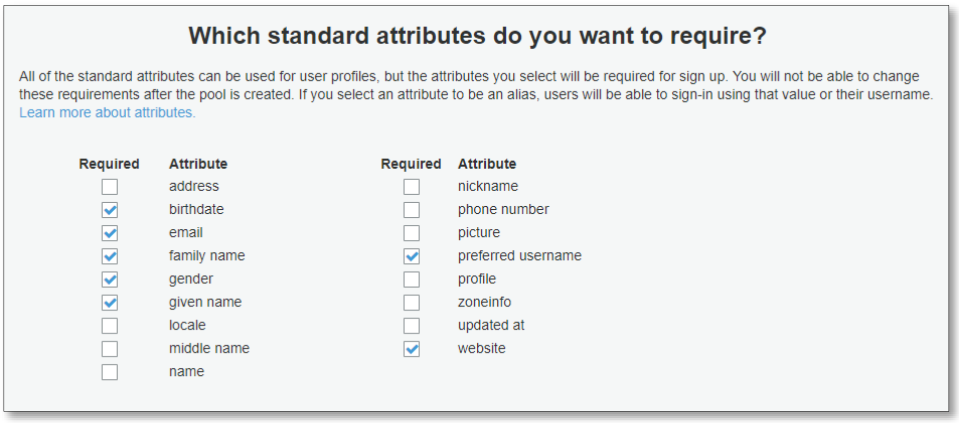
Create a user pool

How do you want your end users to sign in?

You can choose to have users sign in with an email address, phone number, username or preferred username plus their password. Learn more.

- [●] **Username** - Users can use a username and optionally multiple alternatives to sign up and sign in.
  - [✓] Also allow sign in with verified email address
  - [ ] Also allow sign in with verified phone number
  - [ ] Also allow sign in with preferred username (a username that your users can change)
- [ ] **Email address or phone number** - Users can use an email address or phone number as their "username" to sign up and sign in.
  - Allow email addresses
  - Allow phone numbers
  - Allow both email addresses and phone numbers (users can choose one)

Add some attributes you want to collect for the user



Which standard attributes do you want to require?

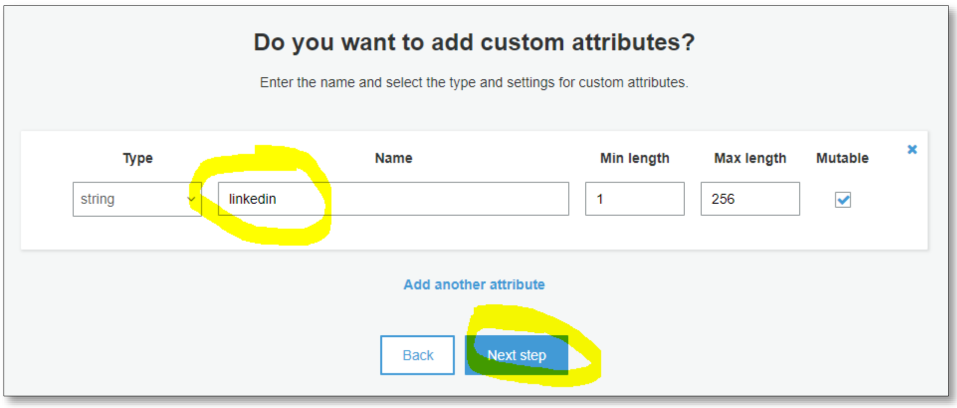
All of the standard attributes can be used for user profiles, but the attributes you select will be required for sign up. You will not be able to change these requirements after the pool is created. If you select an attribute to be an alias, users will be able to sign-in using that value or their username. Learn more about attributes.

Required	Attribute	Required	Attribute
☐	address	☐	nickname
✓	birthdate	☐	phone number
✓	email	☐	picture
✓	family name	✓	preferred username
✓	gender	☐	profile
✓	given name	☐	zoneinfo
☐	locale	☐	updated at
☐	middle name	✓	website
☐	name		

Click “Add custom attribute”

Add a custom attribute name “linkedin” you want to collect for the user

Click “Next step”

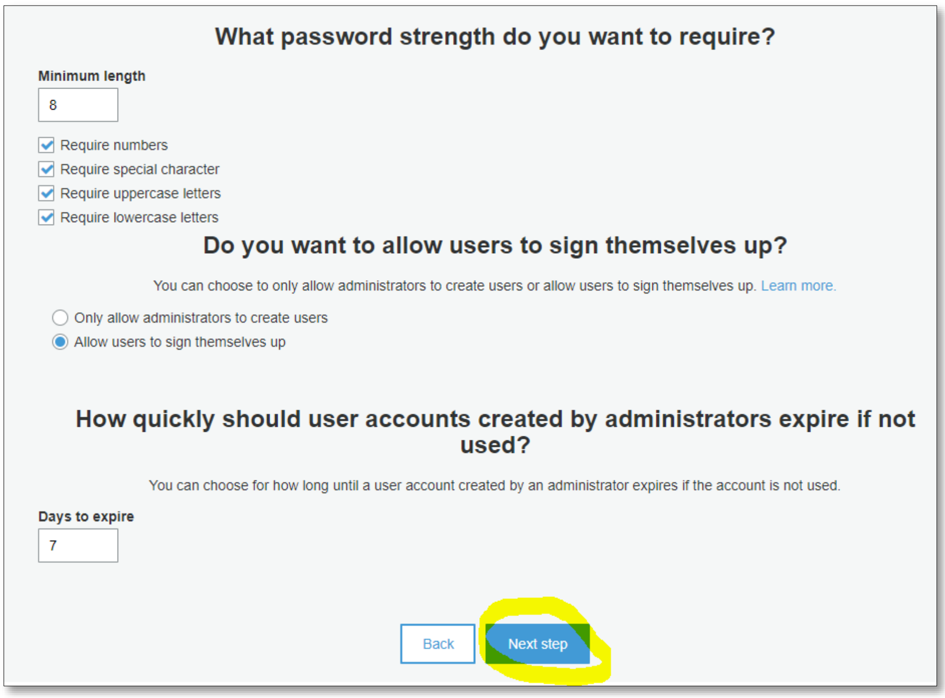


Do you want to add custom attributes?

Enter the name and select the type and settings for custom attributes.

Type	Name	Min length	Max length	Mutable
string	linkedin	1	256	✓

Leave the default settings for password strength, user sign up and account expiration.



What password strength do you want to require?

Minimum length: 8

- [✓] Require numbers
- [✓] Require special character
- [✓] Require uppercase letters
- [✓] Require lowercase letters

Do you want to allow users to sign themselves up?

You can choose to only allow administrators to create users or allow users to sign themselves up. Learn more.

- ○ Only allow administrators to create users
- ● Allow users to sign themselves up

How quickly should user accounts created by administrators expire if not used?

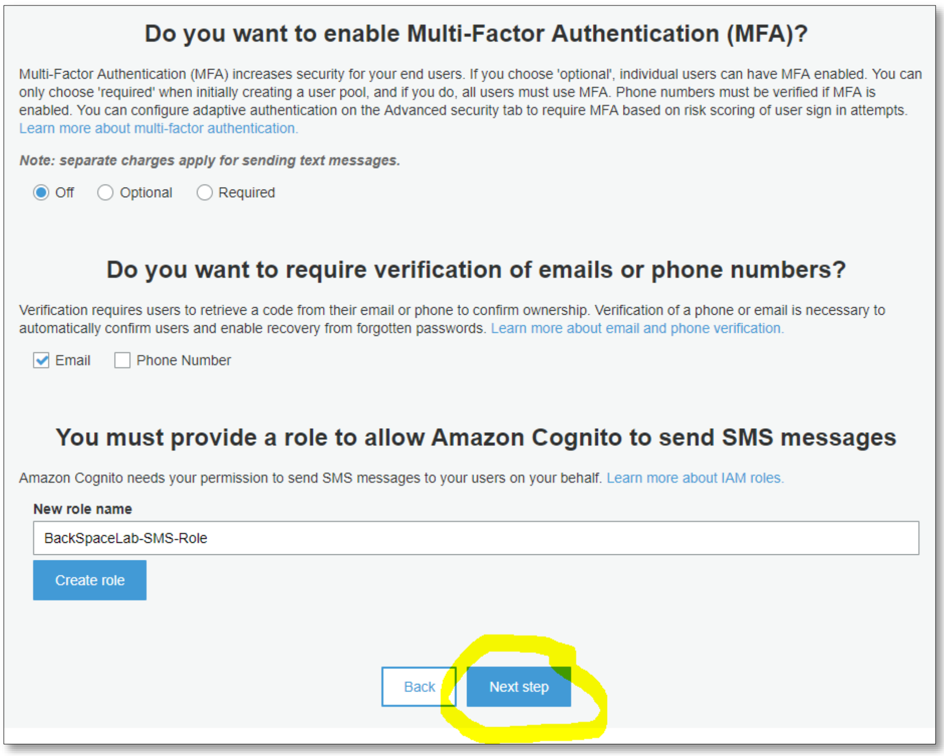
You can choose for how long until a user account created by an administrator expires if the account is not used.

Days to expire: 7

Leave the default settings for MFA and verification.

Do not create a role for sending SMS messages as we are not using MFA or phone number verification

Click “Next step”



Do you want to enable Multi-Factor Authentication (MFA)?

Multi-Factor Authentication (MFA) increases security for your end users. If you choose 'optional', individual users can have MFA enabled. You can only choose 'required' when initially creating a user pool, and if you do, all users must use MFA. Phone numbers must be verified if MFA is enabled. You can configure adaptive authentication on the Advanced security tab to require MFA based on risk scoring of user sign in attempts. Learn more about multi-factor authentication.

Note: separate charges apply for sending text messages.

[✓ Off] Optional Required

Do you want to require verification of emails or phone numbers?

Verification requires users to retrieve a code from their email or phone to confirm ownership. Verification of a phone or email is necessary to automatically confirm users and enable recovery from forgotten passwords. Learn more about email and phone verification.

[✓ Email] Phone Number

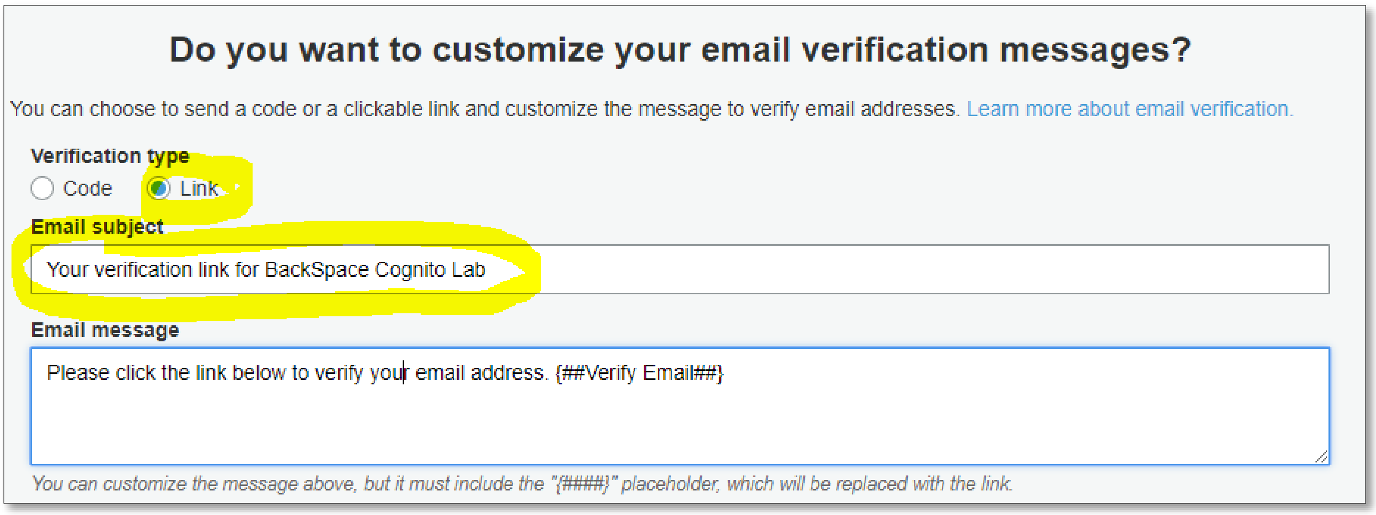
You must provide a role to allow Amazon Cognito to send SMS messages

Amazon Cognito needs your permission to send SMS messages to your users on your behalf. Learn more about IAM roles.

New role name: BackSpaceLab-SMS-Role

Change verification type to link

Give the email message a subject



Do you want to customize your email verification messages?

You can choose to send a code or a clickable link and customize the message to verify email addresses. Learn more about email verification.

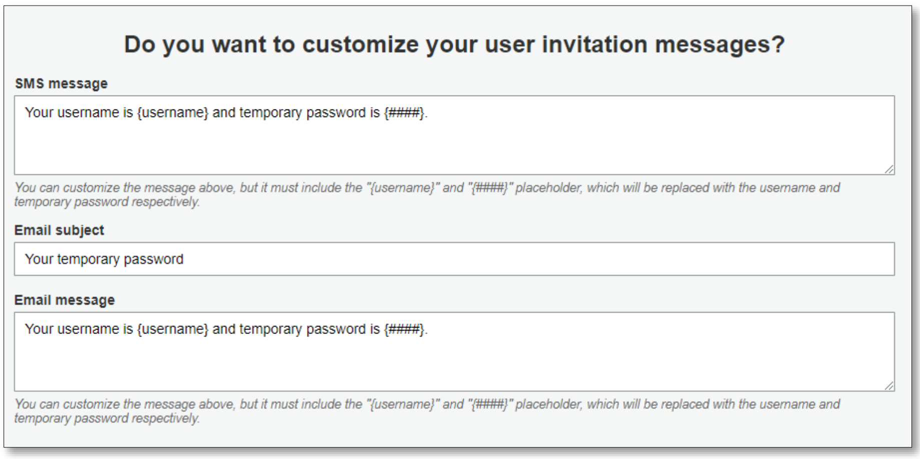
Verification type: Code [✓ Link]

Email subject: Your verification link for BackSpace Cognito Lab

Email message: Please click the link below to verify your email address. {##Verify Email##}

You can customize the message above, but it must include the "{####}" placeholder, which will be replaced with the link.

Leave the invitation message as is



Do you want to customize your user invitation messages?

SMS message

Your username is {username} and temporary password is {####}.

You can customize the message above, but it must include the "{username}" and "{####}" placeholder, which will be replaced with the username and temporary password respectively.

Email subject

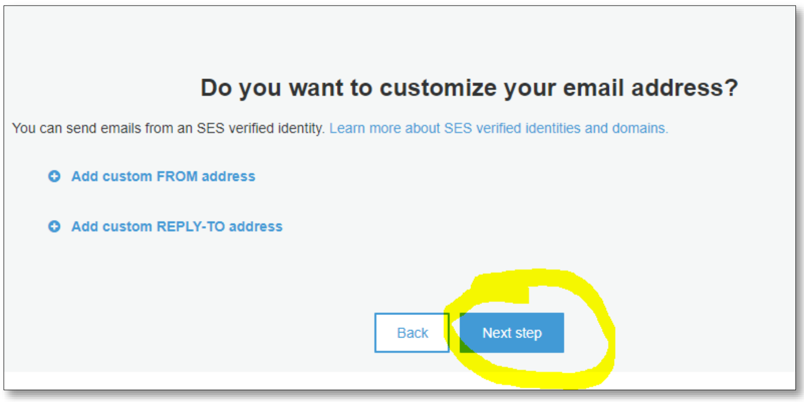
Your temporary password

Email message

Your username is {username} and temporary password is {####}.

You can customize the message above, but it must include the "{username}" and "{####}" placeholder, which will be replaced with the username and temporary password respectively.

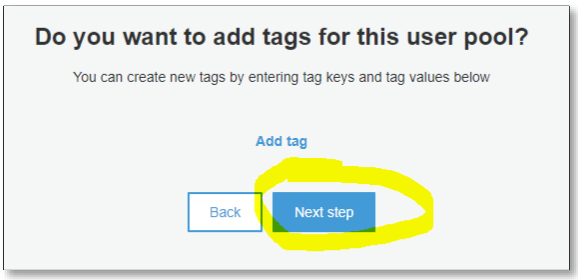
Click “Next step”



Do you want to customize your email address?

You can send emails from an SES verified identity. Learn more about SES verified identities and domains.

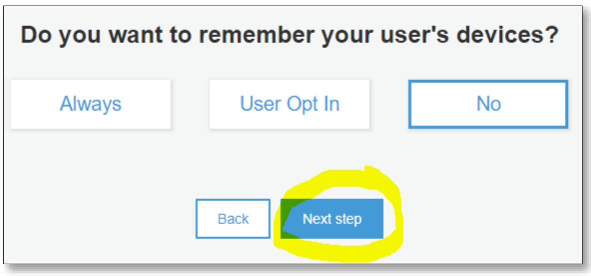
Click “Next step”



Do you want to add tags for this user pool?

You can create new tags by entering tag keys and tag values below

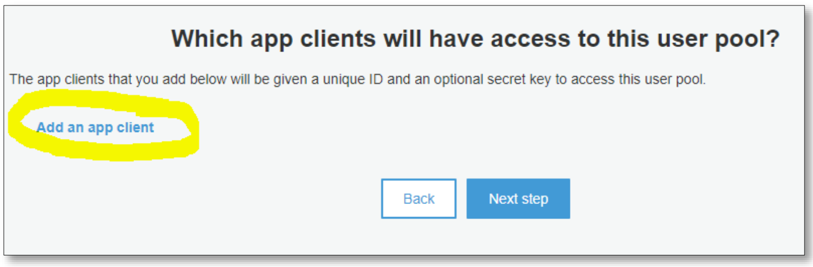
Click “Next step”



Do you want to remember your user's devices?

Always User Opt In [✓ No]

Click “Add an app client”



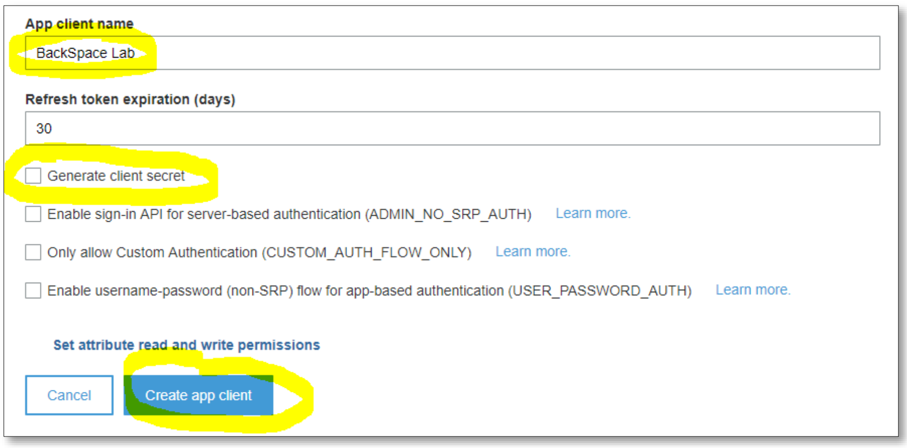
Which app clients will have access to this user pool?

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

Give your app a name

Uncheck “Generate client secret”

Click “Create app client”



App client name: BackSpace Lab

Refresh token expiration (days): 30

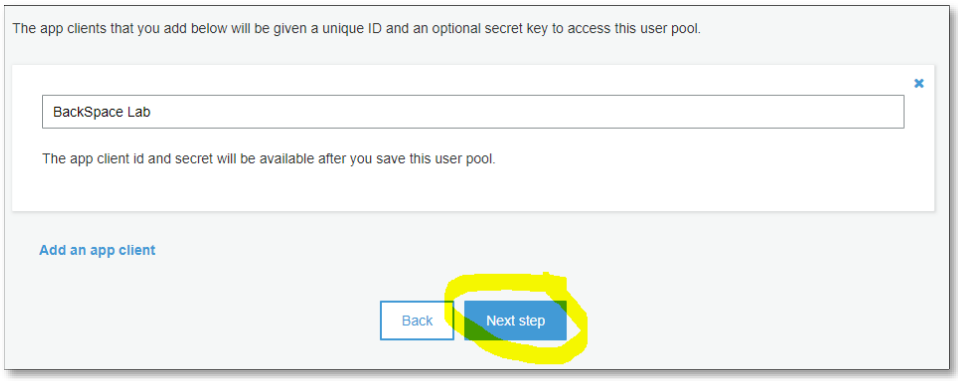
☐ Generate client secret

☐ Enable sign-in API for server-based authentication (ADMIN_NO_SRP_AUTH) Learn more.

☐ Only allow Custom Authentication (CUSTOM_AUTH_FLOW_ONLY) Learn more.

☐ Enable username-password (non-SRP) flow for app-based authentication (USER_PASSWORD_AUTH) Learn more.

Click “Next step”



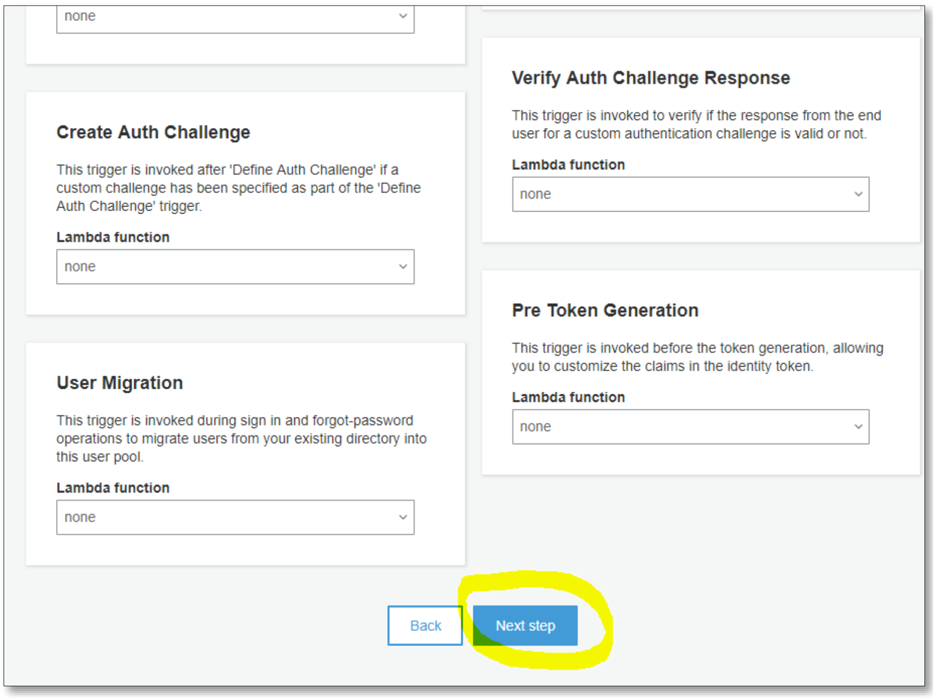
The app clients that you add below will be given a unique ID and an optional secret key to access this user pool.

BackSpace Lab

The app client id and secret will be available after you save this user pool.

Don’t create any workflow triggers

Click “Next step”



none

Create Auth Challenge

This trigger is invoked after 'Define Auth Challenge' if a custom challenge has been specified as part of the 'Define Auth Challenge' trigger.

Lambda function

none

User Migration

This trigger is invoked during sign in and forgot-password operations to migrate users from your existing directory into this user pool.

Lambda function

none

Verify Auth Challenge Response

This trigger is invoked to verify if the response from the end user for a custom authentication challenge is valid or not.

Lambda function

none

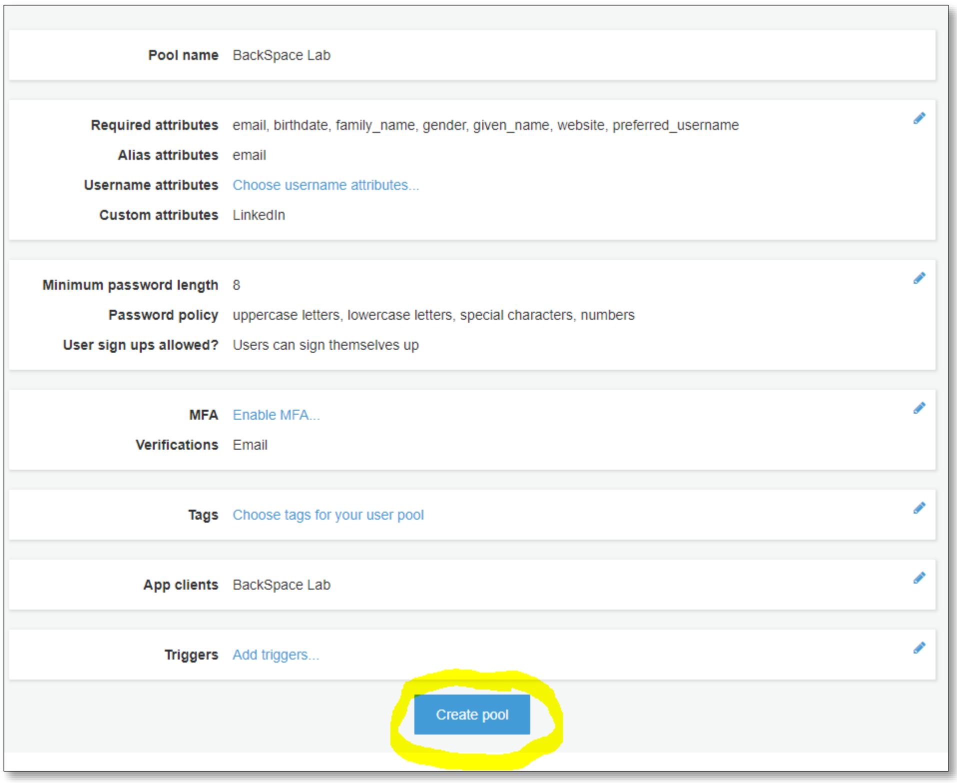
Pre Token Generation

This trigger is invoked before the token generation, allowing you to customize the claims in the identity token.

Lambda function

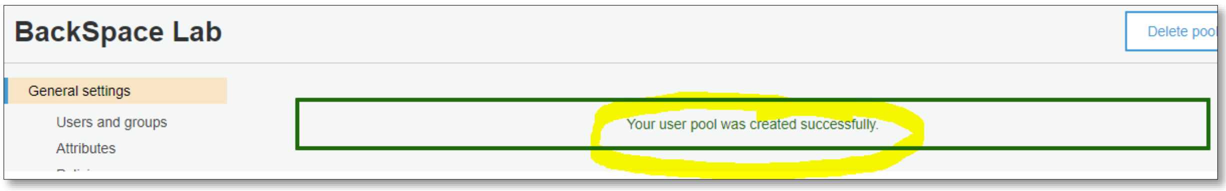
none

Review your settings and click “Create pool”



Pool name	BackSpace Lab
Required attributes	email, birthdate, family_name, gender, given_name, website, preferred_username
Alias attributes	email
Username attributes	Choose username attributes...
Custom attributes	LinkedIn
Minimum password length	8
Password policy	uppercase letters, lowercase letters, special characters, numbers
User sign ups allowed?	Users can sign themselves up
MFA	Enable MFA...
Verifications	Email
Tags	Choose tags for your user pool
App clients	BackSpace Lab
Triggers	Add triggers...

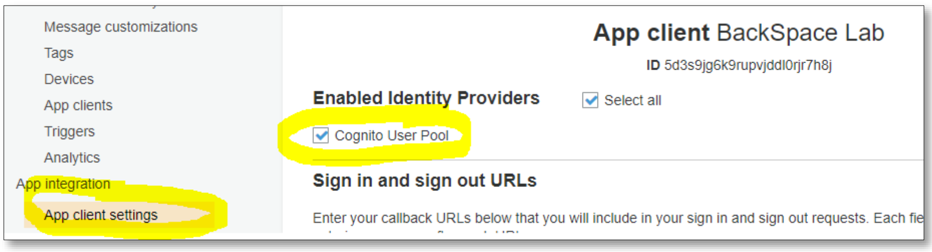
You should receive a success message



Your user pool was created successfully.

Click on "App client settings"

Check "Cognito User Pool"



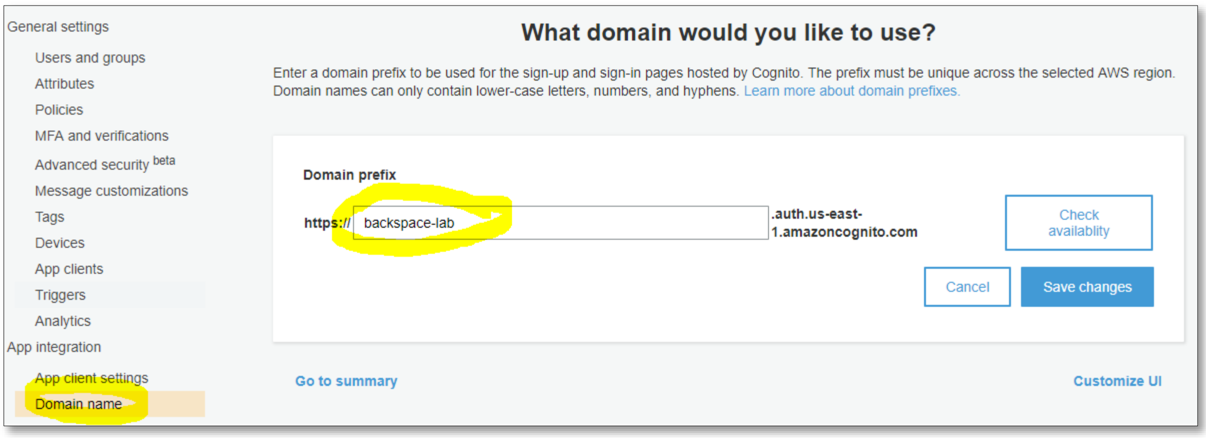
ID 5d3s9jg6k9rupvjddl0rjr7h8j

✓ Cognito User Pool

Enter your callback URLs below that you will include in your sign in and sign out requests. Each fie

Click on "Domain name"

Enter a unique domain prefix



Enter a domain prefix to be used for the sign-up and sign-in pages hosted by Cognito. The prefix must be unique across the selected AWS region. Domain names can only contain lower-case letters, numbers, and hyphens. Learn more about domain prefixes.

Domain prefix

https:// backspace-lab .auth.us-east-1.amazoncognito.com

Creating an AWS Cognito ID Pool

In this section we will use the IAM service to create a role for Federated users to access their own private folder in a bucket on S3. We will then create an AWS Cognito ID Pool to allow AWS Security Token Service (STS) temporary credentials to be issued to federated users.

Create a website with Amazon S3

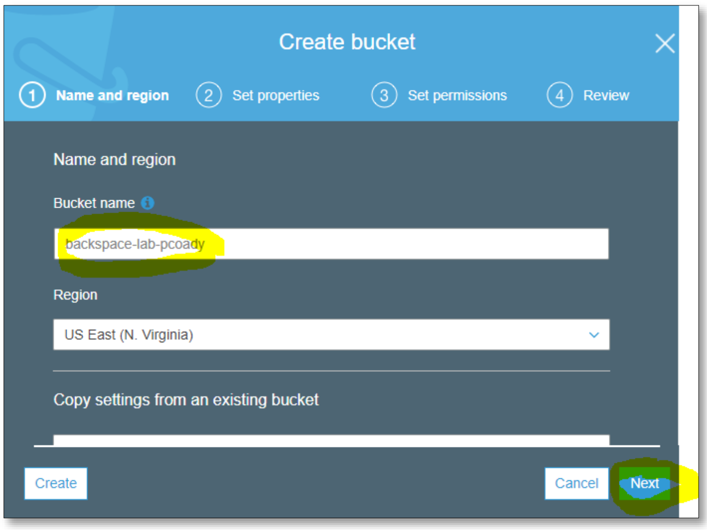
Before we create our Identity pool we need to create an S3 bucket to host our website. We will need to take note of the bucket name as we will use this in an IAM role later.

Clone or download and unzip the following Git repository:

https://github.com/backspace-academy/aws-cognito-lab

Open the S3 console

Create a bucket with a unique name



Name and region

Bucket name: backspace-lab-pcoady

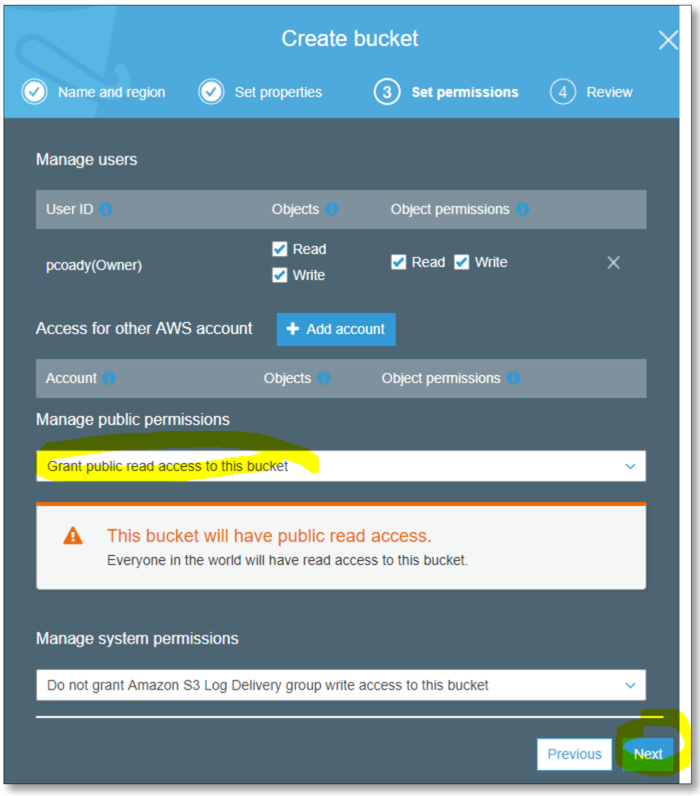
Region: US East (N. Virginia)

Copy settings from an existing bucket

Click “Next”

Select “Grant public read access to this bucket”

Click “Next”



Manage users

User ID	Objects	Object permissions
pcoady(Owner)	✓ Read ✓ Write	✓ Read ✓ Write

Access for other AWS account + Add account

Account	Objects	Object permissions

Manage public permissions

Grant public read access to this bucket

This bucket will have public read access.
Everyone in the world will have read access to this bucket.

Manage system permissions

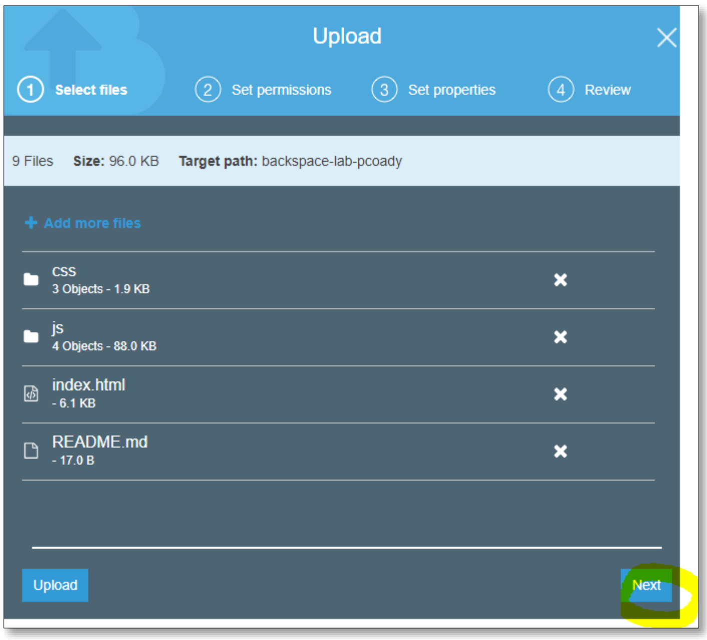
Do not grant Amazon S3 Log Delivery group write access to this bucket

Click “Create Bucket”

Select the created bucket

Click “Upload”

Drag and drop all the files and folders from the Git repository. Click “Next”

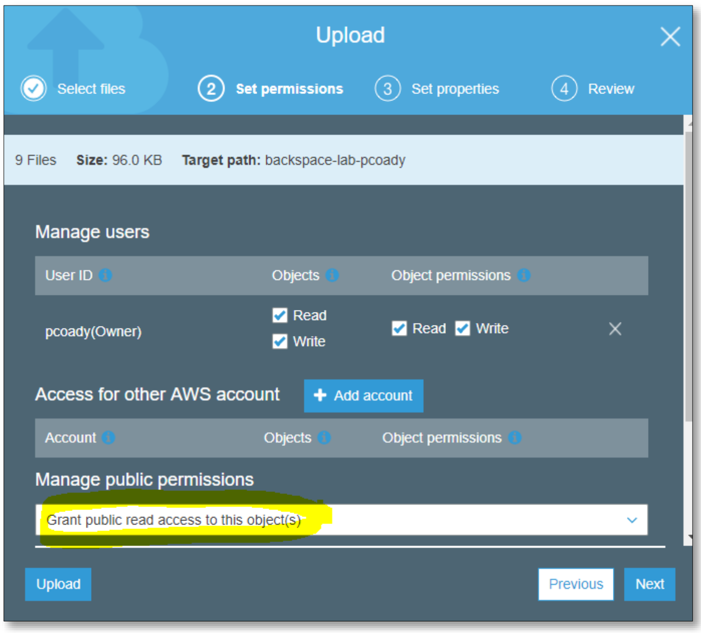


9 Files **Size:** 96.0 KB **Target path:** backspace-lab-pcoady

css	3 Objects - 1.9 KB
js	4 Objects - 88.0 KB
index.html	- 6.1 KB
README.md	- 17.0 B

Select “Grant public read access to this object(s)”

Click “Next”



9 Files **Size:** 96.0 KB **Target path:** backspace-lab-pcoady

Manage users

User ID	Objects	Object permissions
pcoady(Owner)	✓ Read ✓ Write	✓ Read ✓ Write

Access for other AWS account

Account	Objects	Object permissions

Manage public permissions

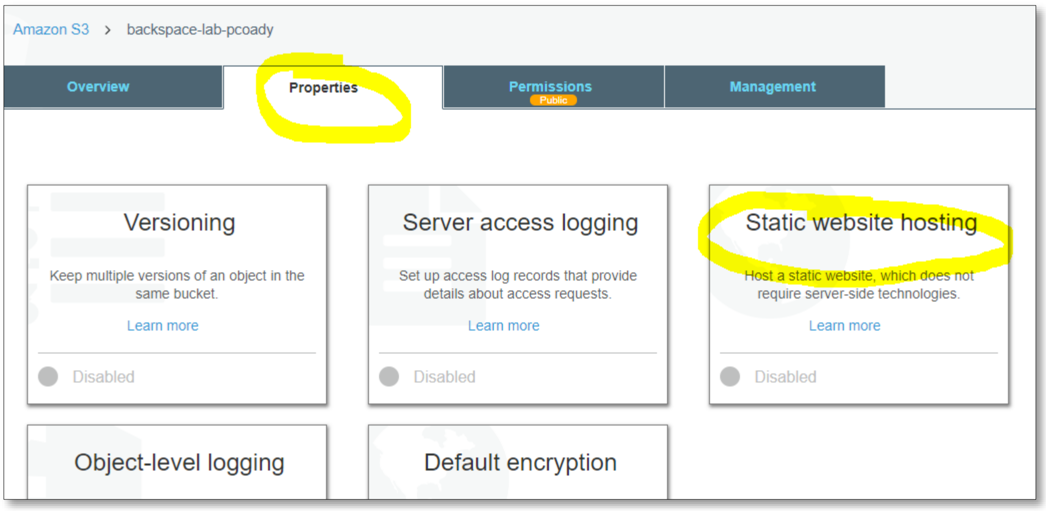
Grant public read access to this object(s)

Click “Next”

Click “Upload”

Select the “Properties” tab

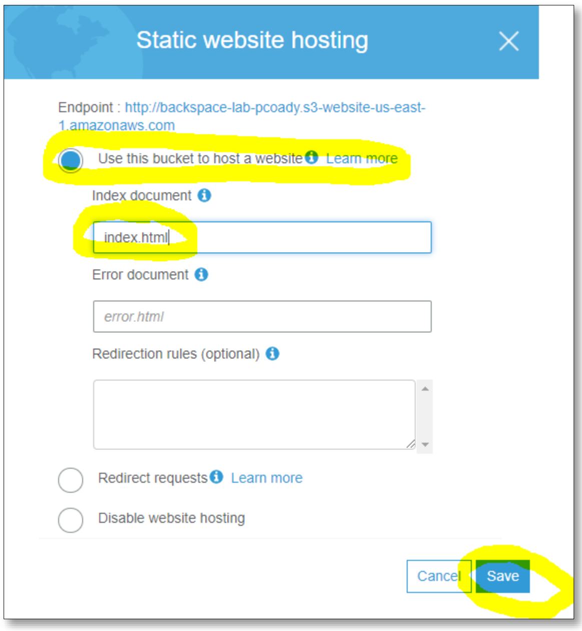
Select “Static website hosting”


Amazon S3 › backspace-lab-pcoady
Overview
Properties
Permissions
Public
Management
Versioning
Keep multiple versions of an object in the same bucket.
Learn more
Disabled
Server access logging
Set up access log records that provide details about access requests.
Learn more
Disabled
Static website hosting
Host a static website, which does not require server-side technologies.
Learn more
Disabled
Object-level logging
Default encryption

Select “Use this bucket to host a website”

Add “index.html” for index document

Click “Save


Static website hosting

Endpoint : http://backspace-lab-pcoady.s3-website-us-east-1.amazonaws.com

Use this bucket to host a website

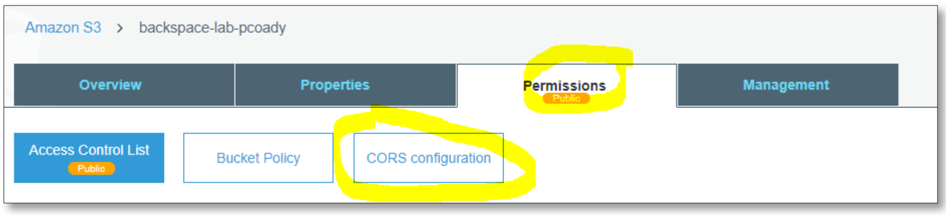
Index document: index.html

Error document: error.html

Redirection rules (optional)

Select the “Permissions” tab

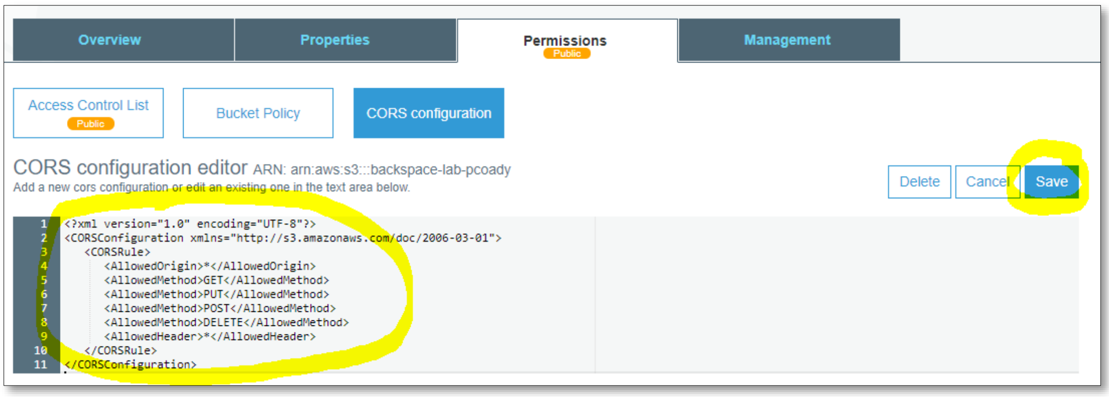
Select “CORS configuration”



Paste the following policy XML

```
<?xml version="1.0" encoding="UTF-8"?>
<CORSConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01">
   <CORSRule>
      <AllowedOrigin>*</AllowedOrigin>
      <AllowedMethod>GET</AllowedMethod>
      <AllowedMethod>PUT</AllowedMethod>
      <AllowedMethod>POST</AllowedMethod>
      <AllowedMethod>DELETE</AllowedMethod>
      <AllowedHeader>*</AllowedHeader>
   </CORSRule>
</CORSConfiguration>
```

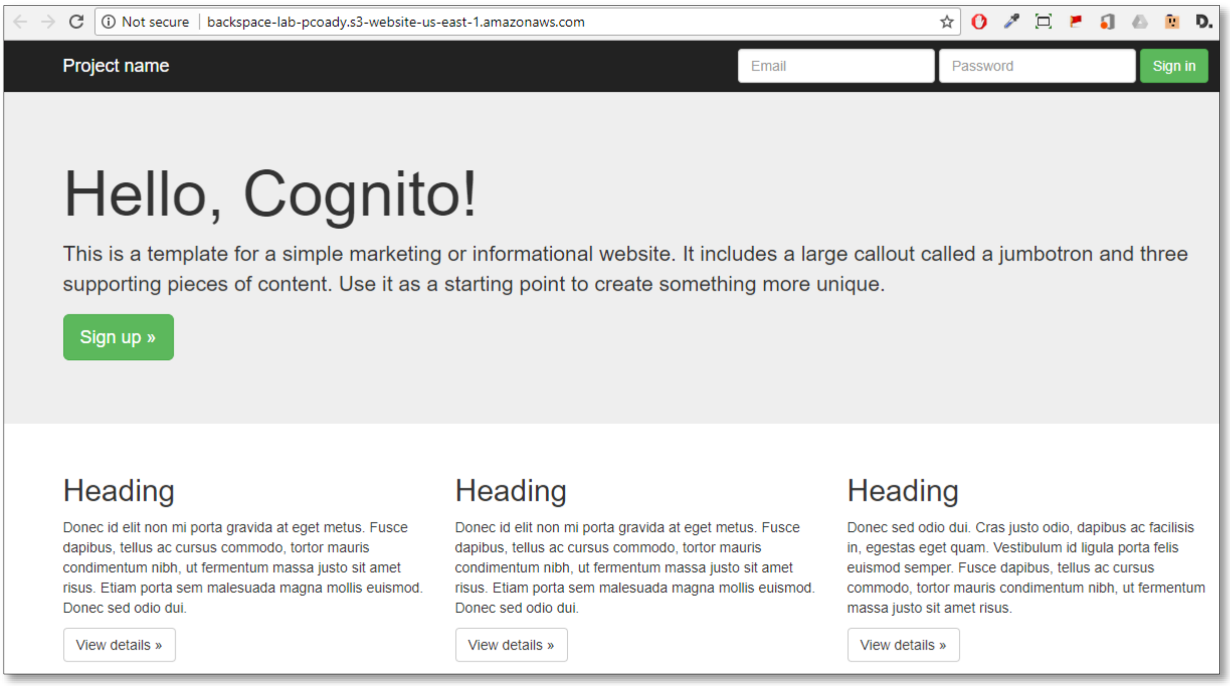
Click Save



CORS configuration editor ARN: arn:aws:s3:::backspace-lab-pcoady
Add a new cors configuration or edit an existing one in the text area below.

```
<?xml version="1.0" encoding="UTF-8"?>
<CORSConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01">
   <CORSRule>
      <AllowedOrigin>*</AllowedOrigin>
      <AllowedMethod>GET</AllowedMethod>
      <AllowedMethod>PUT</AllowedMethod>
      <AllowedMethod>POST</AllowedMethod>
      <AllowedMethod>DELETE</AllowedMethod>
      <AllowedHeader>*</AllowedHeader>
   </CORSRule>
</CORSConfiguration>
```

Navigate to the website endpoint with your browser to check the site is OK.


Not secure | backspace-lab-pcoady.s3-website-us-east-1.amazonaws.com
Project name
Email
Password
Sign in
Sign up »
View details »

Hello, Cognito!

This is a template for a simple marketing or informational website. It includes a large callout called a jumbotron and three supporting pieces of content. Use it as a starting point to create something more unique.

Heading

Donec id elit non mi porta gravida at eget metus. Fusce dapibus, tellus ac cursus commodo, tortor mauris condimentum nibh, ut fermentum massa justo sit amet risus. Etiam porta sem malesuada magna mollis euismod. Donec sed odio dui.

Heading

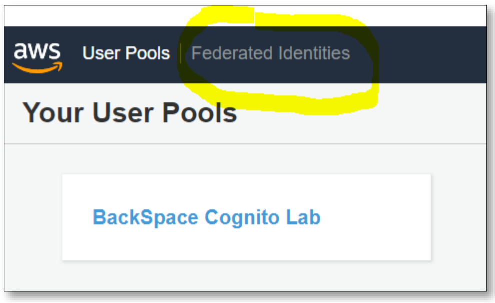
Donec id elit non mi porta gravida at eget metus. Fusce dapibus, tellus ac cursus commodo, tortor mauris condimentum nibh, ut fermentum massa justo sit amet risus. Etiam porta sem malesuada magna mollis euismod. Donec sed odio dui.

Heading

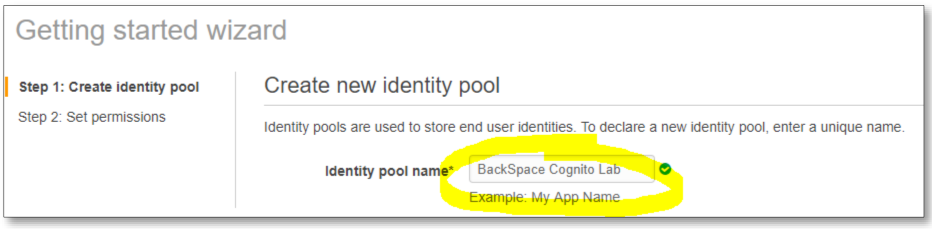
Donec sed odio dui. Cras justo odio, dapibus ac facilisis in, egestas eget quam. Vestibulum id ligula porta felis euismod semper. Fusce dapibus, tellus ac cursus commodo, tortor mauris condimentum nibh, ut fermentum massa justo sit amet risus.

Creating a Cognito ID Pool

Go back to the Cognito console and select “Federated Identities”



Give your identity pool a name



Create new identity pool

Identity pools are used to store end user identities. To declare a new identity pool, enter a unique name.

Identity pool name* BackSpace Cognito Lab

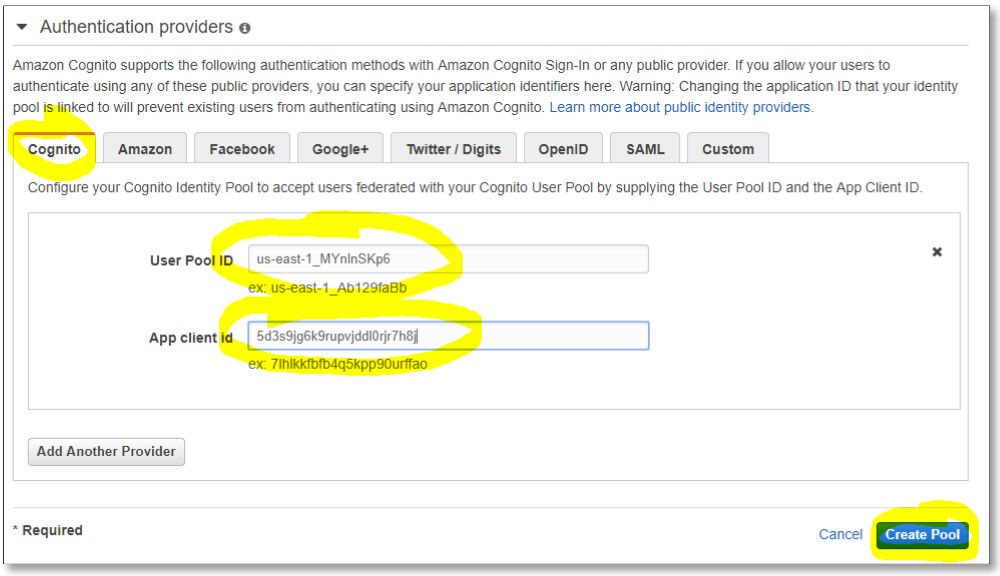
Example: My App Name

Expand “Authentication Providers”

Enter the User Pool ID (the same as used in your code previously)

Enter the App client ID (the same as used in your code previously)

Click “Create Pool”



Authentication providers

Amazon Cognito supports the following authentication methods with Amazon Cognito Sign-In or any public provider. If you allow your users to authenticate using any of these public providers, you can specify your application identifiers here. Warning: Changing the application ID that your identity pool is linked to will prevent existing users from authenticating using Amazon Cognito. Learn more about public identity providers.

Configure your Cognito Identity Pool to accept users federated with your Cognito User Pool by supplying the User Pool ID and the App Client ID.

User Pool ID: us-east-1_MYnInSKp6 (ex: us-east-1_Ab129faBb)

App client id: 5d3s9jg6k9rupvjddl0rjr7h8j (ex: 7lhlkkfbfb4q5kpp90urffao)

You will now be redirected to the IAM console

Expand “View Details”

Click “View Policy Document” for “Your authenticated identities would like access to Cognito.”

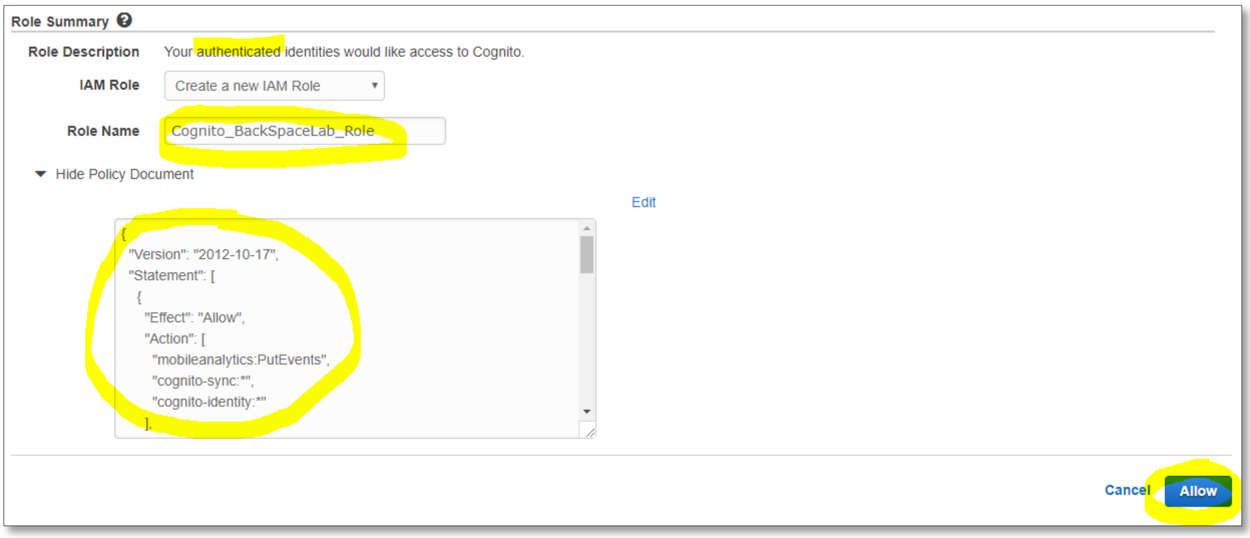
Click Edit

Give the policy a name

Change the policy to include access to Amazon S3:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "mobileanalytics:PutEvents",
        "cognito-sync:*",
        "cognito-identity:*"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3:::backspace-lab-pcoady"
      ],
      "Condition": {
        "StringLike": {
          "s3:prefix": [
            "cognito/backspace-academy/"
          ]
        }
      }
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:PutObject",
        "s3:DeleteObject"
      ],
      "Resource": [
        "arn:aws:s3:::backspace-lab-pcoady/cognito/backspace-academy/${cognito-
identity.amazonaws.com:sub}",
        "arn:aws:s3:::backspace-lab-pcoady/cognito/backspace-academy/${cognito-
identity.amazonaws.com:sub}/*"
      ]
    }
  ]
}
```

Click “Allow”



Role Summary

Role Description	Your authenticated identities would like access to Cognito.
IAM Role	Create a new IAM Role
Role Name	Cognito_BackSpaceLab_Role

Hide Policy Document

Edit

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "mobileanalytics:PutEvents",
    "cognito-sync:*",
    "cognito-identity:*"
   ],
```

You will now redirected to the Cognito console

Take note of your Identity pool ID, you will need it later

Authenticating Cognito Users for a Web Application

In this section we will use the Cognito SDK for Javascript to create authentication capability for a web application.

Create a Cognito connected app

Open your local copy of index.html in Atom IDE

At the bottom you will see both the AWS Cognito Javascript and the AWS Javascript SDKs have been included as modules. It must loaded before our application code (app.js). The buttons are hidden and a spinner image is shown to ensure the buttons are not clicked while the SDK is loading.

```
<!-- AWS SDKs
================================================== -->

<!-- AWS Cognito Javascript SDK -->
<!-- Latest AWS Cognito Javascript SDK can be downloaded from AWS Amplify github repository:--
>
<!-- https://raw.githubusercontent.com/aws/aws-amplify/master/packages/amazon-cognito-
identity-js/dist/amazon-cognito-identity.min.js -->
<script src="./js/amazon-cognito-identity.min.js"></script>

<!-- AWS Javascript SDK -->
<!-- Latest AWS Javascript SDK can be downloaded from AWS Javascript SDK github repository:-->
<!-- https://raw.githubusercontent.com/aws/aws-sdk-js/master/dist/aws-sdk.min.js -->
<script src="./js/aws-sdk-2.211.0.min.js"></script>

<!-- Where the magic happens! -->
<script type="module" src="./js/app.js"></script>
```

The starting code for this lab will be in js/app.js

A copy of the completed code is at js/app-final.js in case you cannot get it working yourself.

Open your local copy of js/app.js

After the click event listeners add the following code.

Change the items in red to suit your user pool, id pool and app.

```
    // Region must be defined
    AWS.config.region = 'us-east-1';

    // User pool
    var poolData = {
            UserPoolId : 'us-east-1_MYnlnSKp6', // Your user pool id here
            ClientId : '5d3s9jg6k9rupvjddl0rjr7h8j' // Your app client id here
    };

    // Your identity pool id here
    var identityPoolId = "us-east-1:eba34910-30e3-4b75-8540-8ee026e6c442"

    // Cognito Sync store name
    var cognitoDatasetName = "backspace-users";

    var cognitoUser, identityId, cognitosync;
```

Now we will create the sign up function. We will pass the user pool id, username, password and attributes to CognitoUserPool SignUp. If successful we get the Cognito user object returned. If we have set up our User pool for verification of email then an error will be returned with message 200 (OK). If this is the case the user will be created but not confirmed until the verification link has been clicked.

```
// Sign Up
function signUp(){
  console.log('Starting Sign up process');

  // Close the modal window
  $('#signUpModal').modal("hide");

  // Get sign up information from modal
  var userLogin = {
    username : $('#inputPreferredUsername').val(),
    password : $('#inputPassword').val()
  }

  var attributes = [
    {
      Name : 'given_name',
      Value : $('#inputGivenName').val()
    },
    {
        Name : 'family_name',
        Value : $('#inputFamilyName').val()
    },
    {
        Name : 'email',
        Value : $('#inputEmail').val()
    },
    {
        Name : 'preferred_username',
        Value : $('#inputPreferredUsername').val()
    },
    {
        Name : 'website',
        Value : $('#inputWebsite').val()
    },
    {
        Name : 'gender',
        Value : $('#inputGender').val()
    },
    {
        Name : 'birthdate',
        Value : $('#inputBirthdate').val()
    },
    {
        Name : 'custom:linkedin',
        Value : $('#inputLinkedin').val()
    }
  ];

  var params = {
    ClientId: poolData.ClientId,    /* required */
    Password: userLogin.password, /* required */
    Username: userLogin.username, /* required */
    UserAttributes: attributes
  };

  var cognitoidentityserviceprovider = new AWS.CognitoIdentityServiceProvider();
  cognitoidentityserviceprovider.signUp(params, function(err, data) {
    if (err) {
      console.log(err, err.stack); // an error occurred
      alert('Error: '+ JSON.stringify(err));
    }
    else {
      console.log(JSON.stringify(data));           // successful response
      if (data.UserConfirmed) {
        bootbox.alert('Please check your email for a verification link.');
      }
      else{
        bootbox.alert('Sign up successful.');
      }
    }
  });
}
```

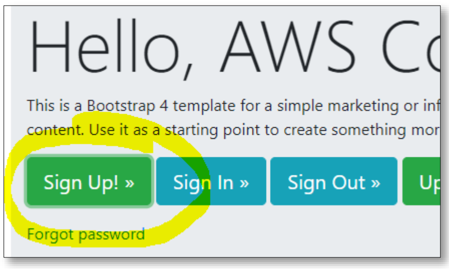
Upload the modified js/app.js to the S3 bucket

Make sure the object has public permissions

Clear your browser cache

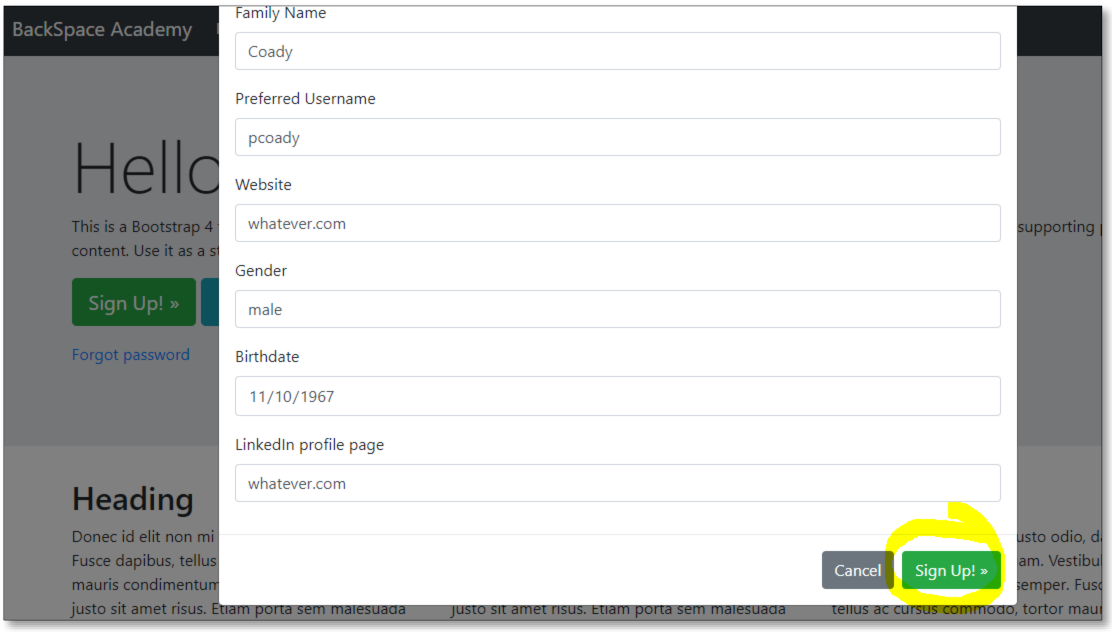
Go to the S3 website url

Click on “Sign Up”


Hello, AWS Co
This is a Bootstrap 4 template for a simple marketing or inf
content. Use it as a starting point to create something mor
Sign Up! »
Sign In »
Sign Out »
Up
Forgot password

Enter the profile details

Click “Sign Up”



Family Name: Coady

Preferred Username: pcoady

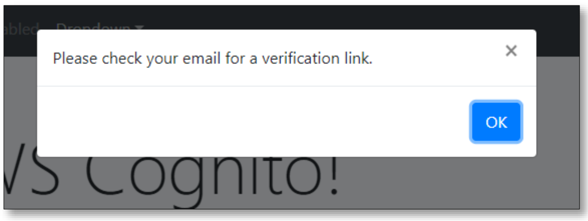
Website: whatever.com

Gender: male

Birthdate: 11/10/1967

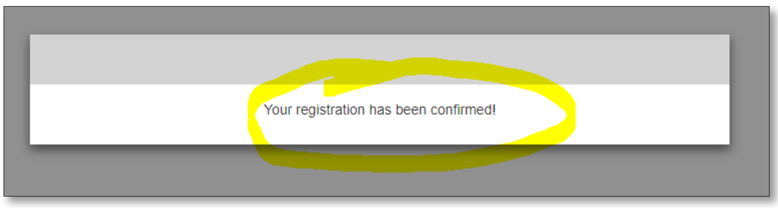
LinkedIn profile page: whatever.com

You will now receive a message to check your email



Please check your email for a verification link.

Go to your email and click on the link to confirm your email address



Your registration has been confirmed!

Now create the signIn function. If an error with message 200 (OK) is returned we need to check whether the sign in was successful by calling getCurrentUser.

```
// Sign In
function signIn(){
  var authenticationData = {
    Username : $('#inputUsername').val(), // Get username & password from modal
    Password : $('#inputPassword2').val()
  };
  $('#signInModal').modal("hide"); // Close the modal window
  var authenticationDetails = new AmazonCognitoIdentity.AuthenticationDetails(authenticationData
);
  var userPool = new AmazonCognitoIdentity.CognitoUserPool(poolData);
  var userData = {
    Username : authenticationData.Username,
    Pool : userPool
  };
  cognitoUser = new AmazonCognitoIdentity.CognitoUser(userData);
  cognitoUser.authenticateUser(authenticationDetails, {
    onSuccess: function (result) {
      createCredentials(result.getIdToken().getJwtToken());
      console.log("Signed in successfully");
    },
    onFailure: function(err) {
      if (err.message == '200'){  // 200 Success return
        cognitoUser = userPool.getCurrentUser();
        if (cognitoUser != null) {
          cognitoUser.getSession(function (err, result) { // Get ID token from session
            if (err) {
              alert(err);
            }
            if (result) {
              createCredentials(result.getIdToken().getJwtToken());
              console.log("Signed to CognitoID in successfully");
            }
          });
        }
        else {
          alert(JSON.stringify(err));
        }
      }
      else {
        alert(JSON.stringify(err));
      }
    },
  });
}
```

Now create a function get AWS Security Token Service (STS) temporary credentials with ID token from Cognito ID. We do this using CognitoIdentityCredentials in the Javascript SDK and passing pool id and federated login information. Change the Logins in red to suit your user pool id.

We then have to refresh the credentials before we can use them.

```
    function createCredentials(idToken) {
        AWS.config.credentials = new AWS.CognitoIdentityCredentials({
            IdentityPoolId: identityPoolId,
            Logins : {
                // Change the key below according to the specific region your user pool is in.
                'cognito-idp.us-east-1.amazonaws.com/us-east-1_MYnlnSKp6' : idToken
            }
        });
        //refreshes credentials using AWS.CognitoIdentity.getCredentialsForIdentity()
        AWS.config.credentials.refresh((error) => {
                if (error) {
                        console.error(error);
                } else {
                        // Instantiate aws sdk service objects now that the credentials have be
en updated.
                        // example: var s3 = new AWS.S3();
                        console.log('Successfully logged!');
                }
        });
    }
```

Upload the modified js/app.js to the S3 bucket

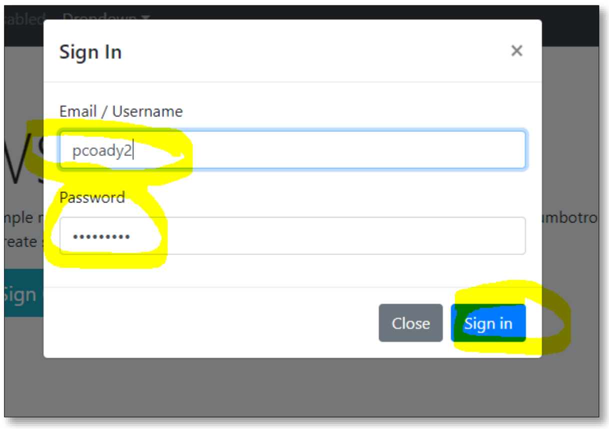
Make sure the object has public permissions

Clear your browser cache

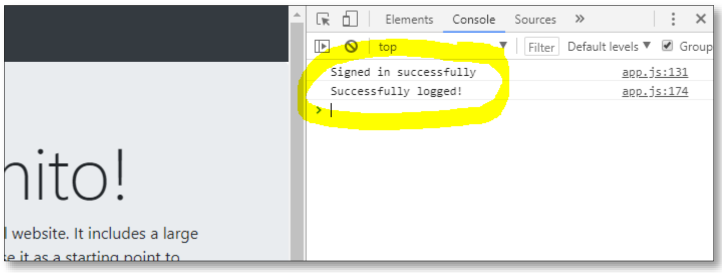
Go to the S3 website url

Click on “Sign In”

Enter your sign in details and click “Sign in”



Press F12 to see the console output



```
Signed in successfully          app.js:131
Successfully logged!            app.js:174
```

Now we will create the signOut function.

There are two options for signing out. A standard sign out and a global sign out that invalidates any tokens in Cognito.

```
function signOut() {
    if (cognitoUser != null) {
        bootbox.confirm({
            title: "Sign out",
        message: "Do you want to also invalidate all user data on this device?",
        buttons: {
            cancel: {
                label: '<i class="fa fa-times"></i> No'
            },
            confirm: {
                label: '<i class="fa fa-check"></i> Yes'
            }
            },
        callback: function (result) {
                if (result) {
                    cognitoUser.globalSignOut({
                        onSuccess: function (result) {
                            bootbox.alert("Successfully signed out and invalidated all app records.");
                        },
                        onFailure: function(err) {
                            alert(JSON.stringify(err));
                        }
                    });
                }
                else {
                    cognitoUser.signOut();
                    bootbox.alert("Signed out of app.");
                }
        }
        });
    }
    else {
        bootbox.alert("You are not signed in!");
    }
}
```

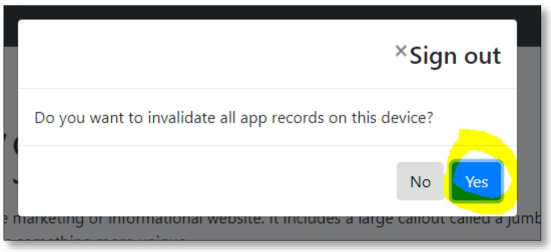
Upload the modified js/app.js to the S3 bucket

Make sure the object has public permissions

Clear your browser cache

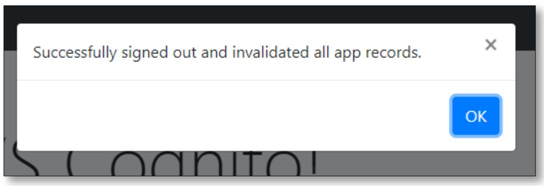
Go to the S3 website url

Click on “Sign Out”



Sign out

Do you want to invalidate all app records on this device?



Successfully signed out and invalidated all app records.

Now create the updateProfile function.

```
function updateProfile(){
    if (cognitoUser != null) {
        console.log('Starting update process');

        var attributes = [
            {
                Name : 'given_name',
                Value : $('#inputGivenName2').val()
            },
            {
                    Name : 'family_name',
                    Value : $('#inputFamilyName2').val()
            },
            {
                    Name : 'website',
                    Value : $('#inputWebsite2').val()
            },
            {
                    Name : 'gender',
                    Value : $('#inputGender2').val()
            },
            {
                    Name : 'birthdate',
                    Value : $('#inputBirthdate2').val()
            },
            {
                    Name : 'custom:linkedin',
                    Value : $('#inputLinkedin2').val()
            }
        ];

        console.log("Adding attributes");
        var attributeList = [];
        for (var a=0; a<attributes.length; ++a){
        var attributeTemp = new AmazonCognitoIdentity.CognitoUserAttribute(attributes[a]);
        attributeList.push(attributeTemp);
        }
        console.log("Updating profile");
        $('#updateModal').modal("hide"); // Close the modal window
        cognitoUser.updateAttributes(attributeList, function(err, result) {
        if (err) {
            alert(JSON.stringify(err.message));
            return;
        }
        console.log('call result: ' + JSON.stringify(result));
                bootbox.alert("Successfully updated!");
        });
        }
        else {
            bootbox.alert("You are not signed in!");
        }
    }
```

Upload the modified js/app.js to the S3 bucket

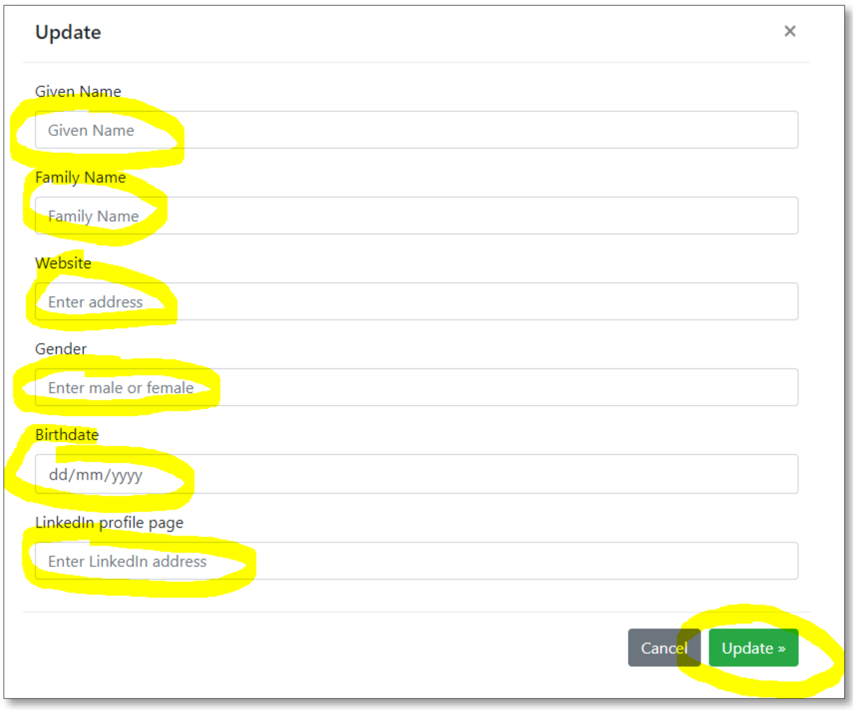
Make sure the object has public permissions

Clear your browser cache

Go to the S3 website url

Click on "Update Profile"

Enter new profile details



Now create the forgotPassword function

```
    function forgotPassword(){
        var verificationCode, newPassword, forgotUser;
        console.log('Forgot Password');
        bootbox.prompt("Enter username or email", function(result){
            console.log("User: " + result);
            forgotUser = result;
            var userPool = new AmazonCognitoIdentity.CognitoUserPool(poolData);
            var userData = {
                Username : forgotUser,
                Pool : userPool
          };
            console.log("Creating user " + JSON.stringify(userData));
          cognitoUser = new AmazonCognitoIdentity.CognitoUser(userData);
            cognitoUser.forgotPassword({
            onSuccess: function (data) {
                // successfully initiated reset password request
                  console.log('CodeDeliveryData from forgotPassword: ' + data);
            },
            onFailure: function(err) {
                console.log(JSON.stringify(err.message));
            },
            //Optional automatic callback
            inputVerificationCode: function(data) {
                console.log('Code sent to: ' + JSON.stringify(data));
                            bootbox.prompt('Please input verification code', function(result){
                                verificationCode = result;
                                bootbox.prompt('Enter new password ', function(result){
                                    newPassword = result;
                                    cognitoUser.confirmPassword(verificationCode, newPassword, {

                            onSuccess() {
                                console.log('Password confirmed!');
                                                    bootbox.alert('Password confirmed!');
                            },
                            onFailure(err) {
                                console.log(JSON.stringify(err.message));
                            }
                        });
                                });
                            });
            }
        });
        });
    }
```

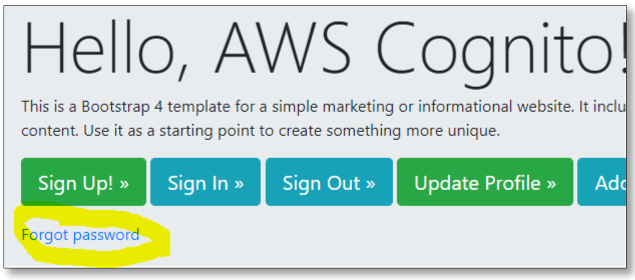
Upload the modified js/app.js to the S3 bucket

Make sure the object has public permissions

Clear your browser cache

Go to the S3 website url

Click on “Forgot password”

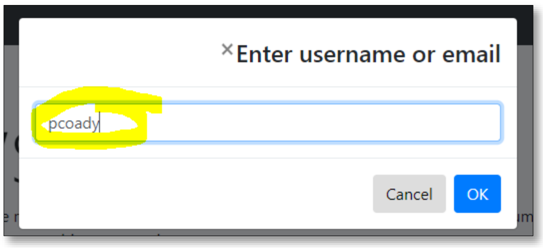


Hello, AWS Cognito!

This is a Bootstrap 4 template for a simple marketing or informational website. It inclu
content. Use it as a starting point to create something more unique.

Enter username or email

Click OK

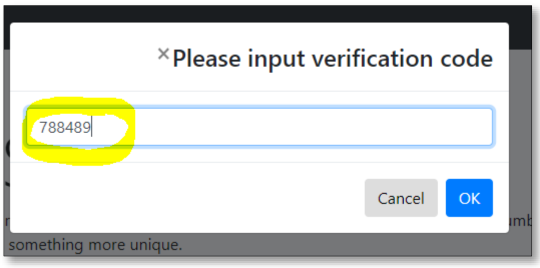


Enter username or email

pcoady

A verification code will be sent to your email.

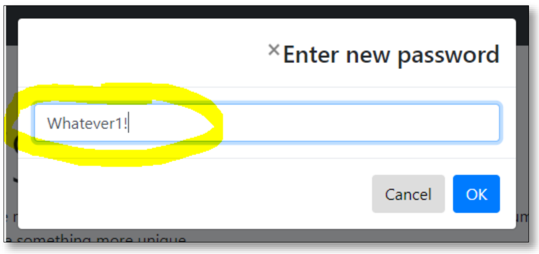
Enter the verificatioin code



Please input verification code

788489

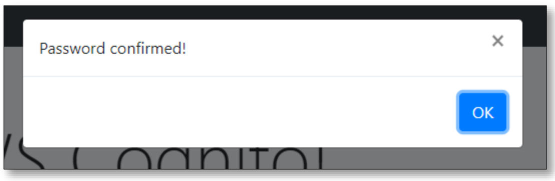
Enter a password



Enter new password

Whatever1!

Your password will be confirmed


Password confirmed!

Saving User Data across Devices with Cognito Sync

In this section we will use the Cognito SDK for Javascript to create an AWS CognitoSync key store for saving user information synchronized across devices.

Get CognitoSync Session Token

Now that we have our CognitoID credentials we can use these to access CognitoSync. First we need to use our new temporary credentials to create a CognitoSync session token.

We are going to create a new function to get our CognitoSync session token.

In order to get the token we must make a call to list records. If our dataset doesn't exist (as is the case now) it will be created automatically. We also get the sync count for the dataset which is needed later to add or change dataset records.

Now lets create the function:

```
    function getCognitoSynToken(){
      /* Other AWS SDKs will automatically use the Cognito Credentials provider */
      /* configured in the JavaScript SDK. */
        var cognitoSyncToken, cognitoSyncCount;
        identityId = AWS.config.credentials.identityId;
      cognitosync = new AWS.CognitoSync();
      cognitosync.listRecords({
        DatasetName: cognitoDatasetName, /* required */
        IdentityId: identityId,  /* required */
        IdentityPoolId: identityPoolId  /* required */
      }, function(err, data) {
        if (err) console.log("listRecords: " + err, err.stack); /* an error occurred */
          else {
            console.log("listRecords: " + JSON.stringify(data));
            cognitoSyncToken = data.SyncSessionToken;
            cognitoSyncCount = data.DatasetSyncCount;
            console.log("SyncSessionToken: " + cognitoSyncToken);           /* successful respon
se */
            console.log("DatasetSyncCount: " + cognitoSyncCount);
            addRecord(cognitoSyncToken, cognitoSyncCount);
          }
      });
    }
```

Now that we have our CognitoSync session token we can use this to add, modify or delete CognitoSync dataset records.

To demonstrate we are going to call addRecord to add a record. Now lets add a record called 'USER_ID' that stores the users Cognito ID. We need to not only pass the CognitoSync session token but also the syncount that we got from the call to listRecords.

```
function addRecord(cognitoSyncToken, cognitoSyncCount){
    var params = {
    DatasetName: cognitoDatasetName, /* required */
    IdentityId: identityId, /* required */
    IdentityPoolId: identityPoolId, /* required */
    SyncSessionToken: cognitoSyncToken, /* required */
    RecordPatches: [
      {
        Key: 'USER_ID', /* required */
        Op: 'replace', /* required */
        SyncCount: cognitoSyncCount, /* required */
        Value: identityId
      }
    ]
  };
  console.log("UserID: " + identityId);
  cognitosync.updateRecords(params, function(err, data) {
    if (err) {
            console.log("updateRecords: " + err, err.stack); /* an error occurred */
        }
    else {
            console.log("Value: " + JSON.stringify(data));           /* successful response */
        }
  });
}
```

Accessing AWS Resources with Cognito ID Credentials

In this section we will use the temporary credentials created by Security Token Service (STS) to access an Amazon S3 bucket.

The role we created for the Cognito ID pool allowed access to S3. Federated users can securely access a folder in the website bucket with the name of their Cognito ID.

First we need to get the identity ID (AWS.config.credentials.identityId) to create the prefix for the file path. Next we will use putObject to save an object with data to the user's personal folder.

```
    function createObject(){
        if (cognitoUser != null) {
            console.log("Creating S3 object");
            identityId = AWS.config.credentials.identityId;
            var prefix = 'cognito/backspace-academy/' + identityId;
            var key = prefix + '/' +  'test' + '.json';
            console.log('Key: ' + key)
            var data = {
                'test': 'It worked!'
            }
            var temp = JSON.stringify(data);
            var bucketName = 'backspace-lab-pcoady';
        var objParams = {
            Bucket: bucketName,
            Key: key,
            ContentType: 'json',
            Body: temp
        };
            // Save data to S3
            var s3 = new AWS.S3({
                params: {
                    Bucket: bucketName
                }
            });
            s3.putObject(objParams, function (err, data) {
            if (err) {
                console.log('Error saving to cloud: ' + err);
                alert('danger','Error.','Unable to save data to S3.');
            } else {
              alert('success','Finished','Data saved to S3.');
            }
        });

        }
        else {
            bootbox.alert('You are not signed in!');
        }
    }
```

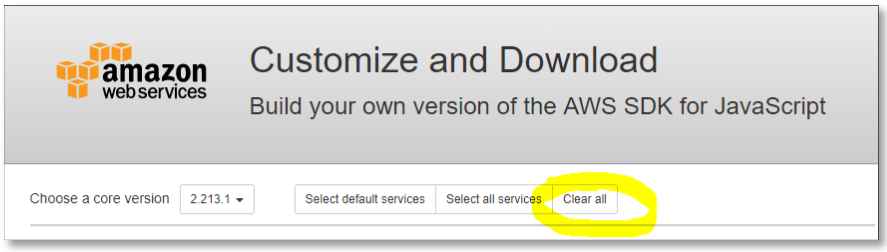
Building a Customised AWS SDK for Javascript version

In this section we will use the AWS SDK for Javascript builder service to create a customised version of the AWS SDK for Javascript.

Go to the AWS SDK for Javascript Builder at:

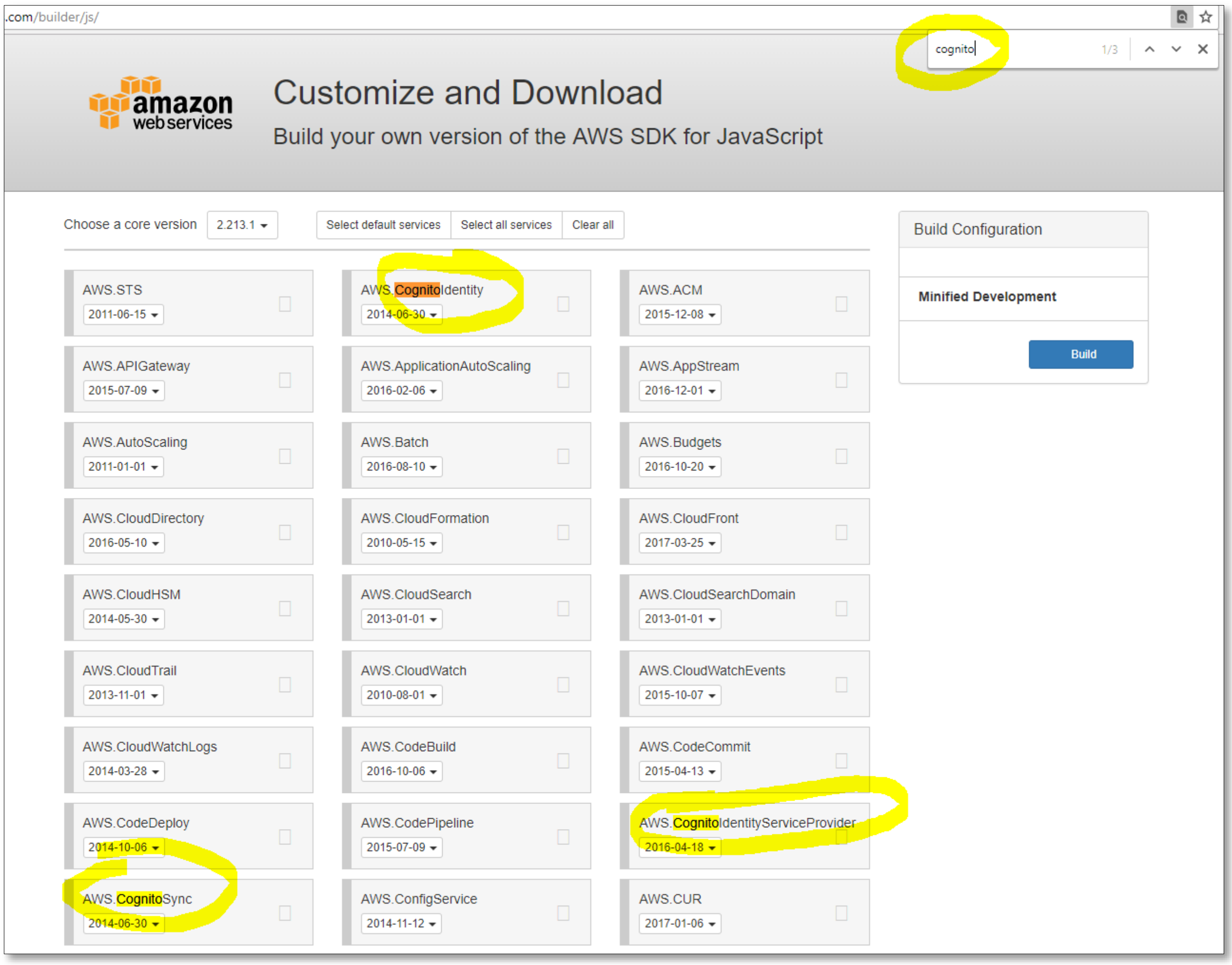
https://sdk.amazonaws.com/builder/js/

Click "Clear all"



Press Ctrl/Cmd F to find on the page

Find "Cognito"



.com/builder/js/

cognito 1/3

Customize and Download

Build your own version of the AWS SDK for JavaScript

Choose a core version 2.213.1

Select default services | Select all services | Clear all

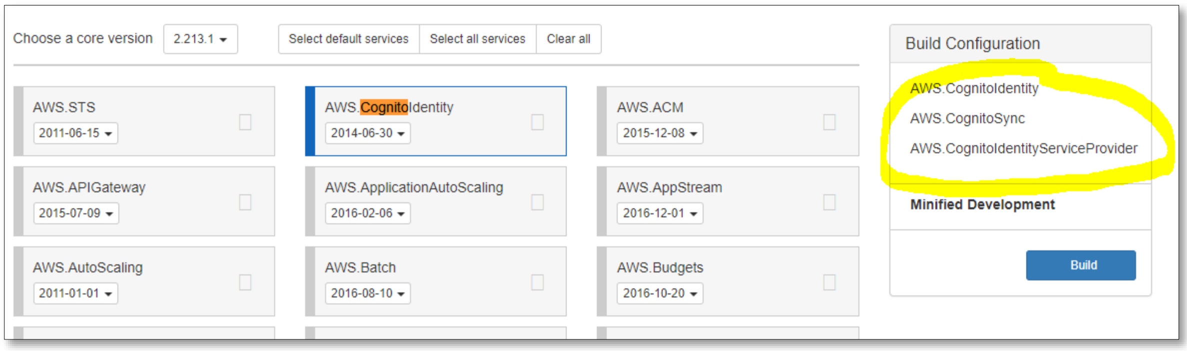
AWS.STS 2011-06-15	AWS.CognitoIdentity 2014-06-30	AWS.ACM 2015-12-08
AWS.APIGateway 2015-07-09	AWS.ApplicationAutoScaling 2016-02-06	AWS.AppStream 2016-12-01
AWS.AutoScaling 2011-01-01	AWS.Batch 2016-08-10	AWS.Budgets 2016-10-20
AWS.CloudDirectory 2016-05-10	AWS.CloudFormation 2010-05-15	AWS.CloudFront 2017-03-25
AWS.CloudHSM 2014-05-30	AWS.CloudSearch 2013-01-01	AWS.CloudSearchDomain 2013-01-01
AWS.CloudTrail 2013-11-01	AWS.CloudWatch 2010-08-01	AWS.CloudWatchEvents 2015-10-07
AWS.CloudWatchLogs 2014-03-28	AWS.CodeBuild 2016-10-06	AWS.CodeCommit 2015-04-13
AWS.CodeDeploy 2014-10-06	AWS.CodePipeline 2015-07-09	AWS.CognitoIdentityServiceProvider 2016-04-18
AWS.CognitoSync 2014-06-30	AWS.ConfigService 2014-11-12	AWS.CUR 2017-01-06

Build Configuration

Minified Development

Build

Click on the three Cognito services to add to the build configuration



Choose a core version 2.213.1

Select default services | Select all services | Clear all

AWS.STS 2011-06-15	AWS.CognitoIdentity 2014-06-30	AWS.ACM 2015-12-08
AWS.APIGateway 2015-07-09	AWS.ApplicationAutoScaling 2016-02-06	AWS.AppStream 2016-12-01
AWS.AutoScaling 2011-01-01	AWS.Batch 2016-08-10	AWS.Budgets 2016-10-20

Build Configuration

AWS.CognitoIdentity

AWS.CognitoSync

AWS.CognitoIdentityServiceProvider

Minified Development

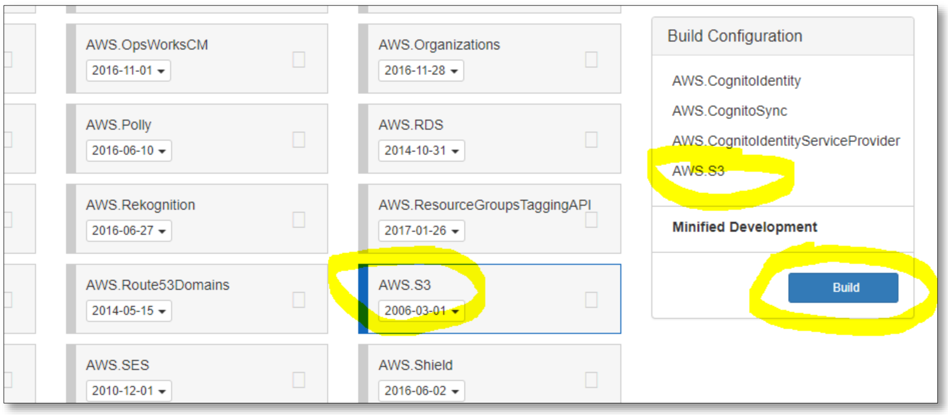
Build

Press Ctrl/Cmd F to find on the page

Find "S3"

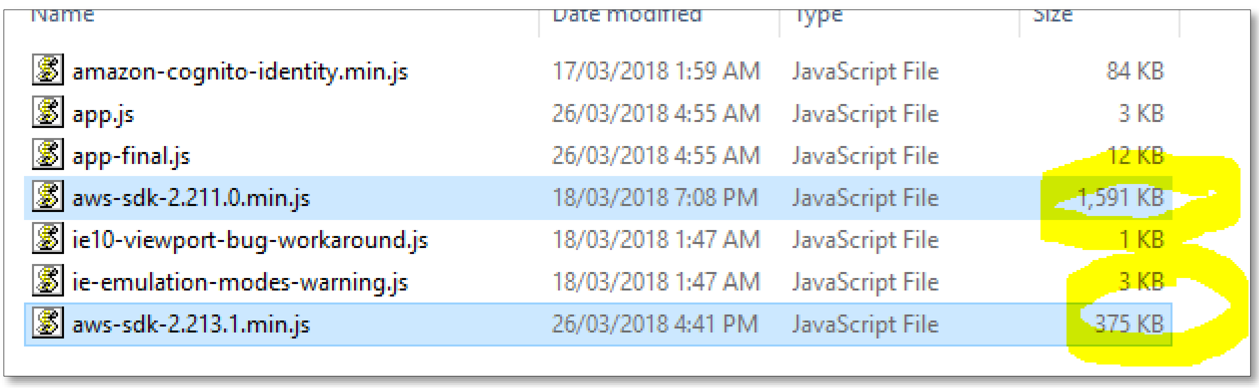
Add the S3 service to the build configuration

Click "Build"



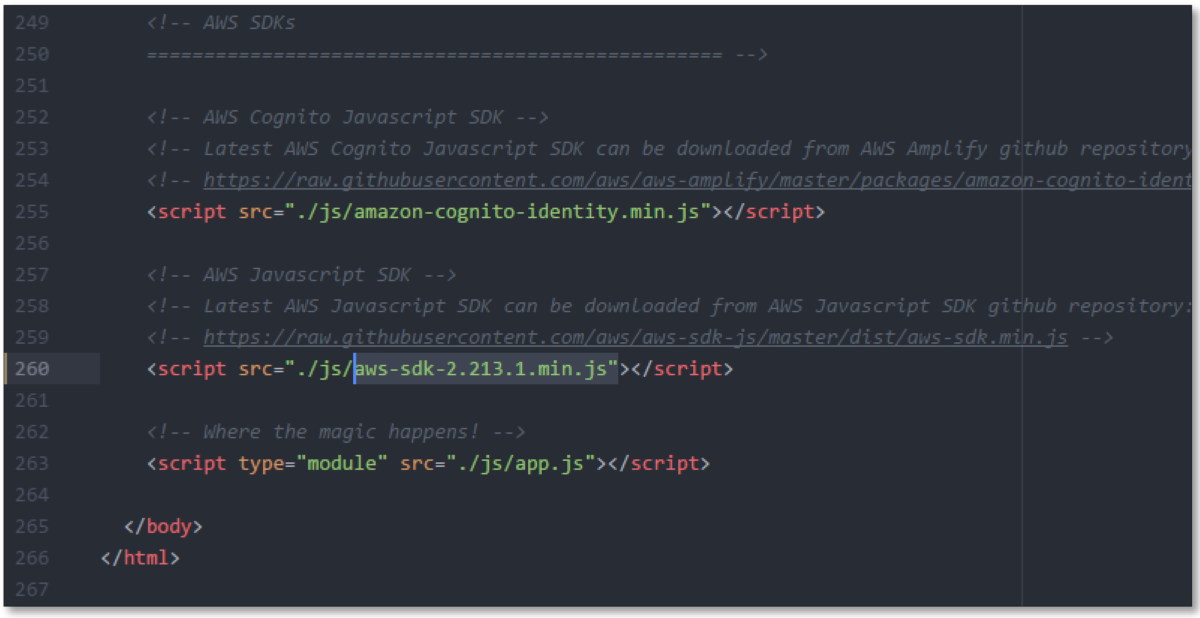
Save the file into the js folder of the application

The new SDK is significantly smaller than previously



Name	Date modified	Type	Size
amazon-cognito-identity.min.js	17/03/2018 1:59 AM	JavaScript File	84 KB
app.js	26/03/2018 4:55 AM	JavaScript File	3 KB
app-final.js	26/03/2018 4:55 AM	JavaScript File	12 KB
aws-sdk-2.211.0.min.js	18/03/2018 7:08 PM	JavaScript File	1,591 KB
ie10-viewport-bug-workaround.js	18/03/2018 1:47 AM	JavaScript File	1 KB
ie-emulation-modes-warning.js	18/03/2018 1:47 AM	JavaScript File	3 KB
aws-sdk-2.213.1.min.js	26/03/2018 4:41 PM	JavaScript File	375 KB

Open the index.html file with Atom IDE and update to your new AWS SDK version.



```
        <!-- AWS SDKs
        ================================================== -->

        <!-- AWS Cognito Javascript SDK -->
        <!-- Latest AWS Cognito Javascript SDK can be downloaded from AWS Amplify github repository
        <!-- https://raw.githubusercontent.com/aws/aws-amplify/master/packages/amazon-cognito-ident
        <script src="./js/amazon-cognito-identity.min.js"></script>

        <!-- AWS Javascript SDK -->
        <!-- Latest AWS Javascript SDK can be downloaded from AWS Javascript SDK github repository:
        <!-- https://raw.githubusercontent.com/aws/aws-sdk-js/master/dist/aws-sdk.min.js -->
        <script src="./js/aws-sdk-2.213.1.min.js"></script>

        <!-- Where the magic happens! -->
        <script type="module" src="./js/app.js"></script>

      </body>
    </html>
```

Alternative techniques

You can also use your existing build tools:

Wepack

Bundling Applications with Webpack

Browserify

Building the SDK as a Dependency with Browserify

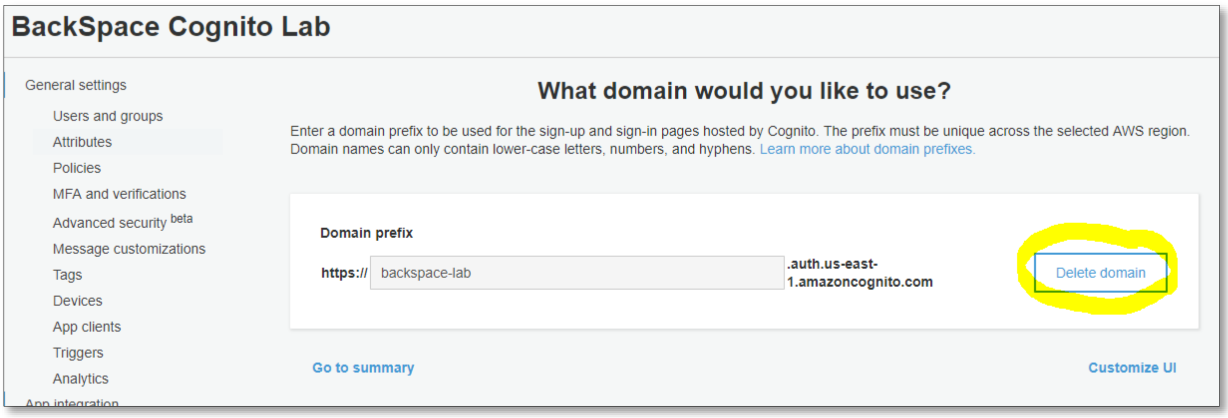
Clean Up

If you have finished with the lab you can delete the resources.

Delete the the website bucket in S3

Next delete the Cognito IID pool

Next delete the Cognito User pool



What domain would you like to use?

Enter a domain prefix to be used for the sign-up and sign-in pages hosted by Cognito. The prefix must be unique across the selected AWS region. Domain names can only contain lower-case letters, numbers, and hyphens. Learn more about domain prefixes.

Domain prefix

https:// backspace-lab .auth.us-east-1.amazoncognito.com

Completed App.js Code

```javascript
// Self-invoking anonymous function
(function($) {
    'use strict';

    // Click event listeners
    $('#btnSignUp').click(function() {
        signUp();
    });

    $('#btnSignIn').click(function() {
        signIn();
    });

    $('#btnSignOut').click(function() {
        signOut();
    });

    $('#btnUpdate').click(function() {
        updateProfile();
    });

    $('#forgotPassword').click(function() {
        forgotPassword();
    });

    $('#btnSync').click(function() {
        getCognitoSynToken();
    });

    $('#btnS3').click(function() {
        createObject();
    });

    // Region must be defined
    AWS.config.region = 'us-east-1';

    // User pool
    var poolData = {
        UserPoolId: 'us-east-1_MYnlnSKp6', // Your user pool id here
        ClientId: '5d3s9jg6k9rupvjddl0rjr7h8j' // Your app client id here
    };

    // Your identity pool id here
    var identityPoolId = "us-east-1:eba34910-30e3-4b75-8540-8ee026e6c442"

    // Cognito Sync store name
    var cognitoDatasetName = "backspace-users";

    var cognitoUser, identityId, cognitosync;

    // Sign Up
    function signUp() {
        console.log('Starting Sign up process');

        // Get sign up information from modal
        var userLogin = {
            username: $('#inputPreferredUsername').val(),
            password: $('#inputPassword').val()
        }

        var attributes = [{
                Name: 'given_name',
                Value: $('#inputGivenName').val()
            },
            {
                Name: 'family_name',
                Value: $('#inputFamilyName').val()
            },
            {
                Name: 'email',
                Value: $('#inputEmail').val()
            },
            {
                Name: 'preferred_username',
                Value: $('#inputPreferredUsername').val()
            },
            {
                Name: 'website',
                Value: $('#inputWebsite').val()
            },
            {
                Name: 'gender',
                Value: $('#inputGender').val()
            },
            {
                Name: 'birthdate',
                Value: $('#inputBirthdate').val()
            },
            {
                Name: 'custom:linkedin',
                Value: $('#inputLinkedin').val()
            }
        ];

        console.log("Adding attributes");
        var attributeList = [];
        for (var a = 0; a < attributes.length; ++a) {
            var attributeTemp = new AmazonCognitoIdentity.CognitoUserAttribute(attributes[a]);
            attributeList.push(attributeTemp);
        }

        console.log("Signing up");
        $('#signUpModal').modal("hide"); // Close the modal window
        var userPool = new AmazonCognitoIdentity.CognitoUserPool(poolData);
        userPool.signUp(userLogin.username, userLogin.password, attributeList, null, function(err, result) {
            if (err) {
                if (err.message == "200") // http 200 OK response, signup pending verfication
                    bootbox.alert('Please check your email for a verification link.');
                else
                    bootbox.alert(JSON.stringify(err.message)); // there is a problem
                return;
            }
            cognitoUser = result.user; // this response will not occur if signup pending verfication
            console.log('user name is ' + cognitoUser.getUsername());
            bootbox.alert('Please check your email for a verification link.');
        });
    }

    // Sign In
    function signIn() {
        var authenticationData = {
            Username: $('#inputUsername').val(), // Get username & password from modal
            Password: $('#inputPassword2').val()
        };
        $('#signInModal').modal("hide"); // Close the modal window
        var authenticationDetails = new AmazonCognitoIdentity.AuthenticationDetails(authenticationData);
        var userPool = new AmazonCognitoIdentity.CognitoUserPool(poolData);
        var userData = {
            Username: authenticationData.Username,
            Pool: userPool
        };
        cognitoUser = new AmazonCognitoIdentity.CognitoUser(userData);
        cognitoUser.authenticateUser(authenticationDetails, {
            onSuccess: function(result) {
                createCredentials(result.getIdToken().getJwtToken());
                console.log("Signed in successfully");
            },
            onFailure: function(err) {
                if (err.message == '200') { // 200 Success return
                    cognitoUser = userPool.getCurrentUser();
                    if (cognitoUser != null) {
                        cognitoUser.getSession(function(err, result) { // Get ID token from session
                            if (err) {
                                alert(err);
                            }
                            if (result) {
                                createCredentials(result.getIdToken().getJwtToken());
                                console.log("Signed in successfully");
                            }
                        });
                    } else {
                        alert(JSON.stringify(err));
                    }
                } else {
                    alert(JSON.stringify(err));
                }
            },
        });
    }

    function createCredentials(idToken) {
        AWS.config.credentials = new AWS.CognitoIdentityCredentials({
            IdentityPoolId: identityPoolId,
            Logins: {
                // Change the key below according to your user pool and region.
                'cognito-idp.us-east-1.amazonaws.com/us-east-1_MYnlnSKp6': idToken
            }
        });
        //refreshes credentials using AWS.CognitoIdentity.getCredentialsForIdentity()
        AWS.config.credentials.refresh((error) => {
            if (error) {
                console.error(error);
                bootbox.alert('Unable to sign in. Please try again.')
            } else {
                // Instantiate aws sdk service objects now that the credentials have been updated.
                // example: var s3 = new AWS.S3();
                console.log('Successfully logged!');
            }
        });
    }

    function signOut() {
        if (cognitoUser != null) {
            bootbox.confirm({
                title: "Sign out",
                message: "Do you want to also invalidate all user data on this device?",
                buttons: {
                    cancel: {
                        label: '<i class="fa fa-times"></i> No'
                    },
                    confirm: {
                        label: '<i class="fa fa-check"></i> Yes'
                    }
                },
                callback: function(result) {
                    if (result) {
                        cognitoUser.globalSignOut({
                            onSuccess: function(result) {
                                bootbox.alert("Successfully signed out and invalidated all app records.");
                            },
                            onFailure: function(err) {
                                alert(JSON.stringify(err));
                            }
                        });
                    } else {
                        cognitoUser.signOut();
                        bootbox.alert("Signed out of app.");
                    }
                }
            });
        } else {
            bootbox.alert("You are not signed in!");
        }
    }

    function updateProfile() {
        if (cognitoUser != null) {
            console.log('Starting update process');

            var attributes = [{
                    Name: 'given_name',
                    Value: $('#inputGivenName2').val()
                },
                {
                    Name: 'family_name',
                    Value: $('#inputFamilyName2').val()
                },
                {
                    Name: 'website',
                    Value: $('#inputWebsite2').val()
                },
                {
                    Name: 'gender',
                    Value: $('#inputGender2').val()
                },
                {
                    Name: 'birthdate',
                    Value: $('#inputBirthdate2').val()
                },
                {
                    Name: 'custom:linkedin',
                    Value: $('#inputLinkedin2').val()
                }
            ];

            console.log("Adding attributes");
            var attributeList = [];
            for (var a = 0; a < attributes.length; ++a) {
                var attributeTemp = new AmazonCognitoIdentity.CognitoUserAttribute(attributes[a]);
                attributeList.push(attributeTemp);
            }
            console.log("Updating profile");
            $('#updateModal').modal("hide"); // Close the modal window
            cognitoUser.updateAttributes(attributeList, function(err, result) {
                if (err) {
                    alert(JSON.stringify(err.message));
                    return;
                }
                console.log('call result: ' + JSON.stringify(result));
                bootbox.alert("Successfully updated!");
            });
        } else {
            bootbox.alert("You are not signed in!");
        }
    }

    function forgotPassword() {
        var verificationCode, newPassword, forgotUser;
        console.log('Forgot Password');
        bootbox.prompt("Enter username or email", function(result) {
            console.log("User: " + result);
            forgotUser = result;
            var userPool = new AmazonCognitoIdentity.CognitoUserPool(poolData);
            var userData = {
                Username: forgotUser,
                Pool: userPool
            };
            console.log("Creating user " + JSON.stringify(userData));
            cognitoUser = new AmazonCognitoIdentity.CognitoUser(userData);
            cognitoUser.forgotPassword({
                onSuccess: function(data) {
                    // successfully initiated reset password request
                    console.log('CodeDeliveryData from forgotPassword: ' + data);
                },
                onFailure: function(err) {
                    console.log(JSON.stringify(err.message));
                },
                //Optional automatic callback
                inputVerificationCode: function(data) {
                    console.log('Code sent to: ' + JSON.stringify(data));
                    bootbox.prompt('Please input verification code', function(result) {
                        verificationCode = result;
                        bootbox.prompt('Enter new password ', function(result) {
                            newPassword = result;
                            cognitoUser.confirmPassword(verificationCode, newPassword, {
                                onSuccess() {
                                    console.log('Password confirmed!');
                                    bootbox.alert('Password confirmed!');
                                },
                                onFailure(err) {
                                    console.log(JSON.stringify(err.message));
                                }
                            });
                        });
                    });
                }
            });
        });
    }

    function getCognitoSynToken() {
        /* Other AWS SDKs will automatically use the Cognito Credentials provider */
        /* configured in the JavaScript SDK. */
        var cognitoSyncToken, cognitoSyncCount;
        identityId = AWS.config.credentials.identityId;
        cognitosync = new AWS.CognitoSync();
        cognitosync.listRecords({
            DatasetName: cognitoDatasetName,
            /* required */
            IdentityId: identityId,
            /* required */
            IdentityPoolId: identityPoolId /* required */
        }, function(err, data) {
            if (err) console.log("listRecords: " + err, err.stack); /* an error occurred */
            else {
                console.log("listRecords: " + JSON.stringify(data));
                cognitoSyncToken = data.SyncSessionToken;
                cognitoSyncCount = data.DatasetSyncCount;
                console.log("SyncSessionToken: " + cognitoSyncToken); /* successful response */

                console.log("DatasetSyncCount: " + cognitoSyncCount);
                addRecord(cognitoSyncToken, cognitoSyncCount);
            }
        });
    }

    function addRecord(cognitoSyncToken, cognitoSyncCount) {
        var params = {
            DatasetName: cognitoDatasetName,
            /* required */
            IdentityId: identityId,
            /* required */
            IdentityPoolId: identityPoolId,
            /* required */
            SyncSessionToken: cognitoSyncToken,
            /* required */
            RecordPatches: [{
                Key: 'USER_ID',
                /* required */
                Op: 'replace',
                /* required */
                SyncCount: cognitoSyncCount,
                /* required */
                Value: identityId
            }]
        };
        console.log("UserID: " + identityId);
        cognitosync.updateRecords(params, function(err, data) {
            if (err) {
                console.log("updateRecords: " + err, err.stack); /* an error occurred */
            } else {
                console.log("Value: " + JSON.stringify(data)); /* successful response */
            }
        });
    }

    function createObject() {
        if (cognitoUser != null) {
            console.log("Creating S3 object");
            identityId = AWS.config.credentials.identityId;
            var prefix = 'cognito/backspace-academy/' + identityId;
            var key = prefix + '/' + 'test' + '.json';
            console.log('Key: ' + key)
            var data = {
                'test': 'It worked!'
            }
            var temp = JSON.stringify(data);
            var bucketName = 'backspace-lab-pcoady';
            var objParams = {
                Bucket: bucketName,
                Key: key,
                ContentType: 'json',
                Body: temp
            };
            // Save data to S3
            var s3 = new AWS.S3({
                params: {
                    Bucket: bucketName
                }
            });
            s3.putObject(objParams, function(err, data) {
                if (err) {
                    console.log('Error saving to cloud: ' + err);
                    alert('danger', 'Error.', 'Unable to save data to S3.');
                } else {
                    alert('success', 'Finished', 'Data saved to S3.');
                }
            });

        } else {
            bootbox.alert('You are not signed in!');
        }
    }

    // End  self-invoking anonymous function
})(jQuery);
```