IBM Single Sign On service for Bluemix – Build your first application using external authentication.

8/28/2015

IBM EcoDev

Contents

Introduction:

When developing an application supporting multiple users, one of the first decisions you will need to face is how to authenticate each users. It might seem like a simple approach to create a database table and populate it with credentials, but this quickly creates further challenges. For one, how will your application be able to integrate with existing applications and their authentication services? This lab will introduce the IBM Single Sign On (SSO) service for Bluemix which you can use to configure your web application to authenticate users from social media sites, enterprise directories via SAML federation, and also a ldap-based cloud identity source.

This lab will walk through the steps to add and configure the IBM Single Sign On service for Bluemix to the Node.js starter application. It will provide step-by-step instructions to create the application, add the service to the Bluemix workspace, configure a cloud identity source, bind and integrate the service into the application, and modify the application code to use external authentication.

The lab requires a workstation (Windows, Mac OS X or Linux are all fine) with the following items installed:

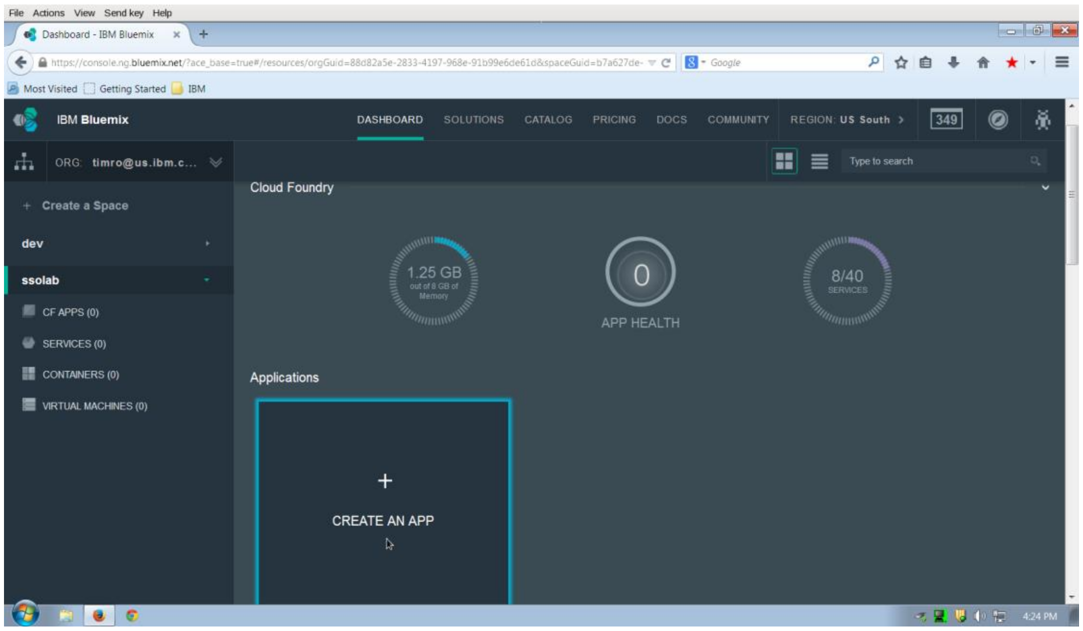
- bluemix.net account (sign up for a 30-day free trial)
- node.js version 0.12.x (download from: https://nodejs.org/download/)
- multiple web browsers (it's most convenient to use one browser for working with the Bluemix and Single Sign On control panel in one and a separate browser for testing your application, recent versions of Firefox, Chrome, Opera, Safari, Internet Explorer)
- Cloud Foundry CLI tool version 6.9 or later – there is a step in the application setup where a link is provided for installation if it is not already on your workstation.

The app.js source code shown in the screenshots is provided in an appendix to this lab if you choose not to type in the code for a particular step.

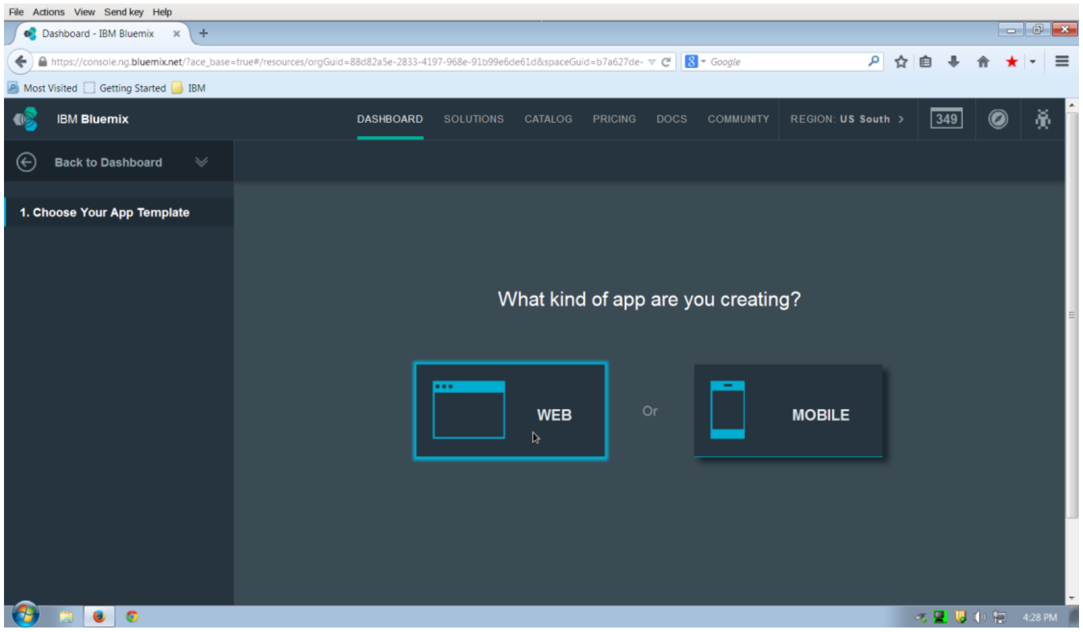
Now let’s get started.

Create Node.js starter application and add Single Sign On service to dashboard:

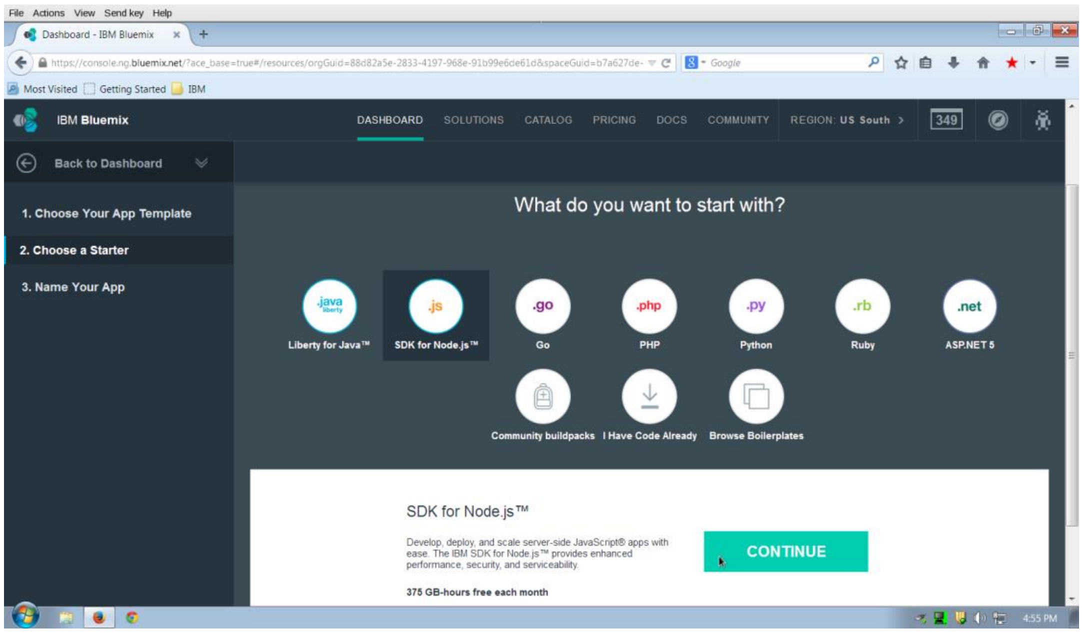
Sign in to Bluemix and go to the Dashboard. Under Cloud Foundry, click on the "CREATE AN APP" button to get started:



Then select Web app:


IBM Bluemix
Back to Dashboard
1. Choose Your App Template
What kind of app are you creating?
WEB
Or
MOBILE

Next, you will need to select the runtime for the application. For this lab select the SDK for Node.js, and confirm by clicking on CONTINUE:



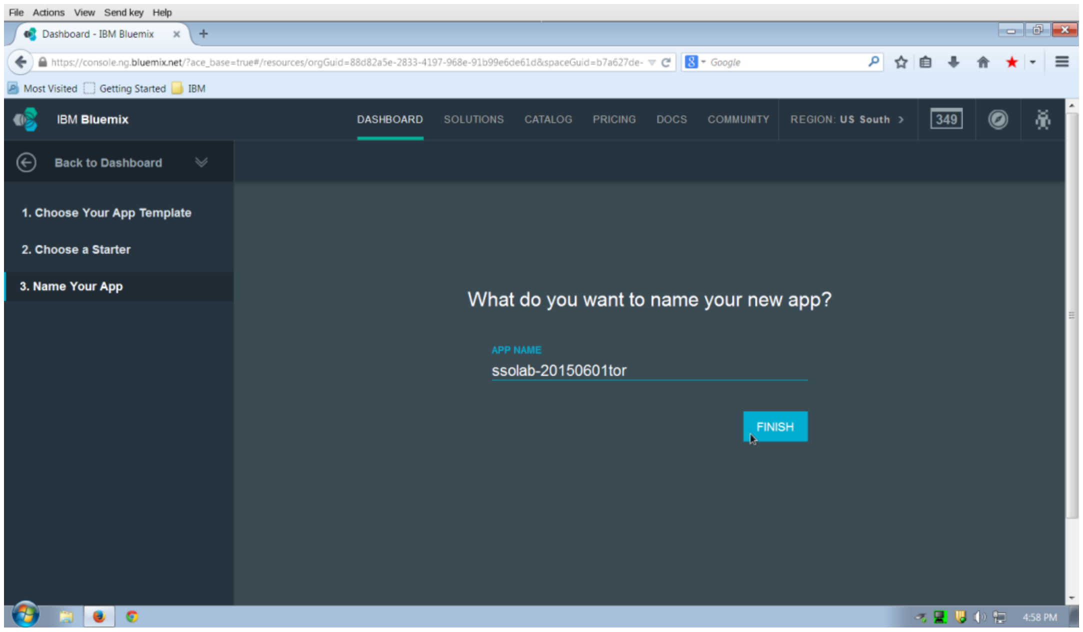
What do you want to start with?

SDK for Node.js™

Develop, deploy, and scale server-side JavaScript® apps with ease. The IBM SDK for Node.js™ provides enhanced performance, security, and serviceability.

375 GB-hours free each month

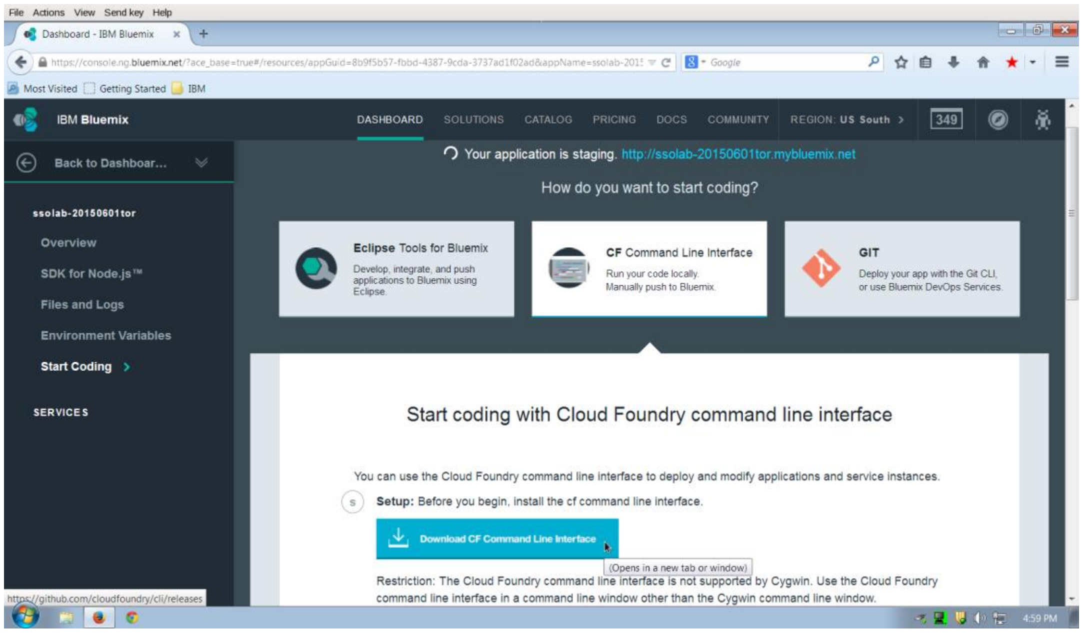
To complete adding the new application, provide a name. Each application in Bluemix public requires a unique name. One easy way to do this for a test app is to append the date and your initials to the name, and then select FINISH:



What do you want to name your new app?

APP NAME: ssolab-20150601tor

Bluemix will bring up the application's Start Coding panel with a message at the top about the application starting the staging process. If you do not already have the Cloud Foundry CLI tool installed on your workstation, click on the link shown to go to GitHub and download the current version for your workstation (it will be later than 6.9 but that is ok). After the download is completed, install the cf tool and then proceed to the next step.


File Actions View Send key Help
Dashboard - IBM Bluemix
IBM Bluemix
DASHBOARD SOLUTIONS CATALOG PRICING DOCS COMMUNITY REGION: US South
Back to Dashboar...
ssolab-20150601tor
Overview
SDK for Node.js™
Files and Logs
Environment Variables
Start Coding
SERVICES
Eclipse Tools for Bluemix
CF Command Line Interface
GIT
Download CF Command Line Interface
(Opens in a new tab or window)

Your application is staging. http://ssolab-20150601tor.mybluemix.net

How do you want to start coding?

Eclipse Tools for Bluemix: Develop, integrate, and push applications to Bluemix using Eclipse.

CF Command Line Interface: Run your code locally. Manually push to Bluemix.

GIT: Deploy your app with the Git CLI, or use Bluemix DevOps Services.

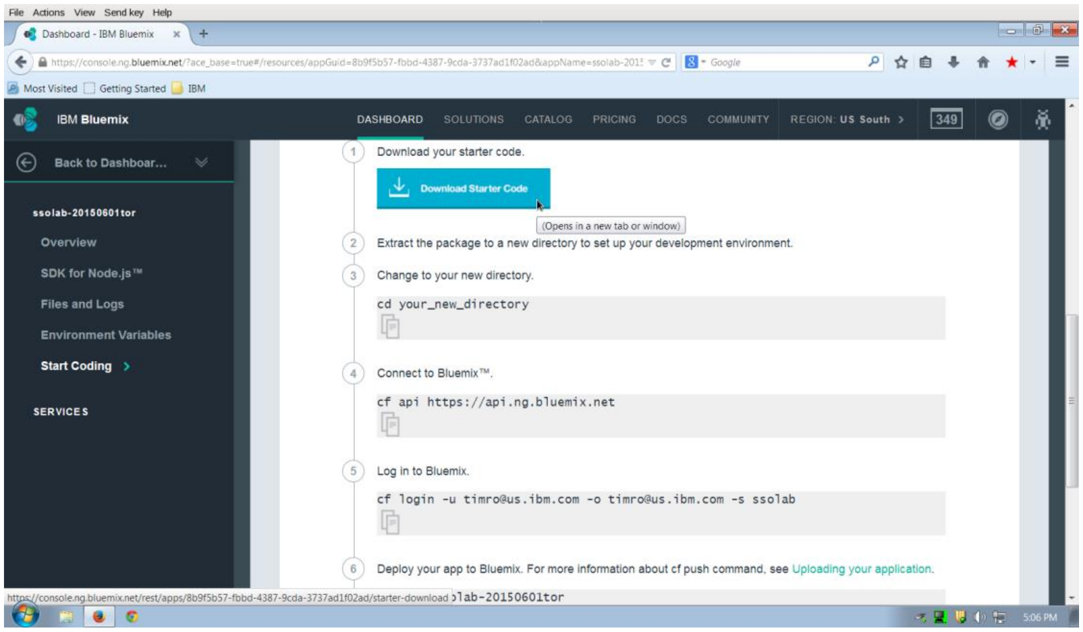
Start coding with Cloud Foundry command line interface

You can use the Cloud Foundry command line interface to deploy and modify applications and service instances.

Setup: Before you begin, install the cf command line interface.

Restriction: The Cloud Foundry command line interface is not supported by Cygwin. Use the Cloud Foundry command line interface in a command line window other than the Cygwin command line window.

After you have installed cf (or if you already had it installed), scroll down on the same panel to the series of numbered steps starting with the Download Starter Code button. Make a note of these steps (they will not match the picture and instead will be customized for your Bluemix id, current Bluemix space and application name). Complete steps 1, 2 and 3 as shown to get a command prompt with a copy of your code. You will use this command prompt to add the node **passport** module to the starter application later.



1. Download your starter code.
2. Extract the package to a new directory to set up your development environment.
3. Change to your new directory.

```
cd your_new_directory
```

4. Connect to Bluemix™.

```
cf api https://api.ng.bluemix.net
```

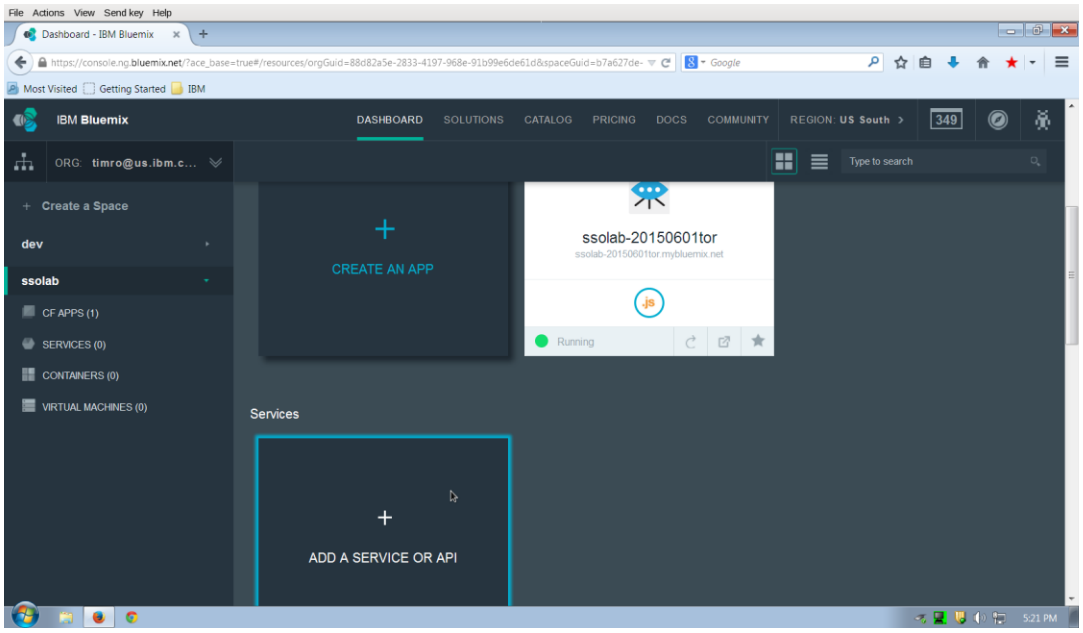
5. Log in to Bluemix.

```
cf login -u timro@us.ibm.com -o timro@us.ibm.com -s ssolab
```

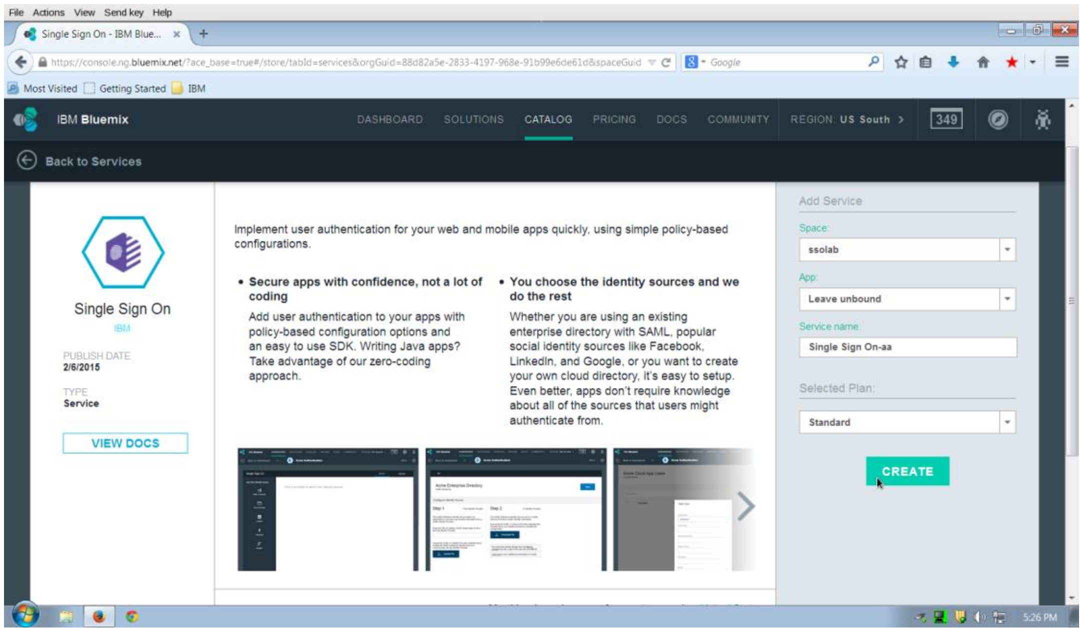
6. Deploy your app to Bluemix. For more information about cf push command, see Uploading your application.

Next, we will add the Single Sign On service to the Dashboard. Before proceeding, you will need to select an id that will become the prefix of the URL for the service when used when applications redirect users to the service for selecting an identity provider, and also for handling callbacks from identity providers. This id can be up to 32 characters, and must start with an alphanumeric character. Make a note of it here if you like: ____

Then, go back to the Bluemix Cloud Foundry Dashboard and select Add a Service:



Scroll down in the services list to the Security section and then click on the Single Sign On icon, this will bring up a configuration panel for adding the service. On the right hand side, select Leave unbound under the App: section. You can change the Service name: that will be shown in the dashboard or leave it at the default. Then click on the CREATE button:



PUBLISH DATE: 2/6/2015

TYPE: Service

Implement user authentication for your web and mobile apps quickly, using simple policy-based configurations.

- **Secure apps with confidence, not a lot of coding**
  Add user authentication to your apps with policy-based configuration options and an easy to use SDK. Writing Java apps? Take advantage of our zero-coding approach.
- **You choose the identity sources and we do the rest**
  Whether you are using an existing enterprise directory with SAML, popular social identity sources like Facebook, LinkedIn, and Google, or you want to create your own cloud directory, it's easy to setup. Even better, apps don't require knowledge about all of the sources that users might authenticate from.

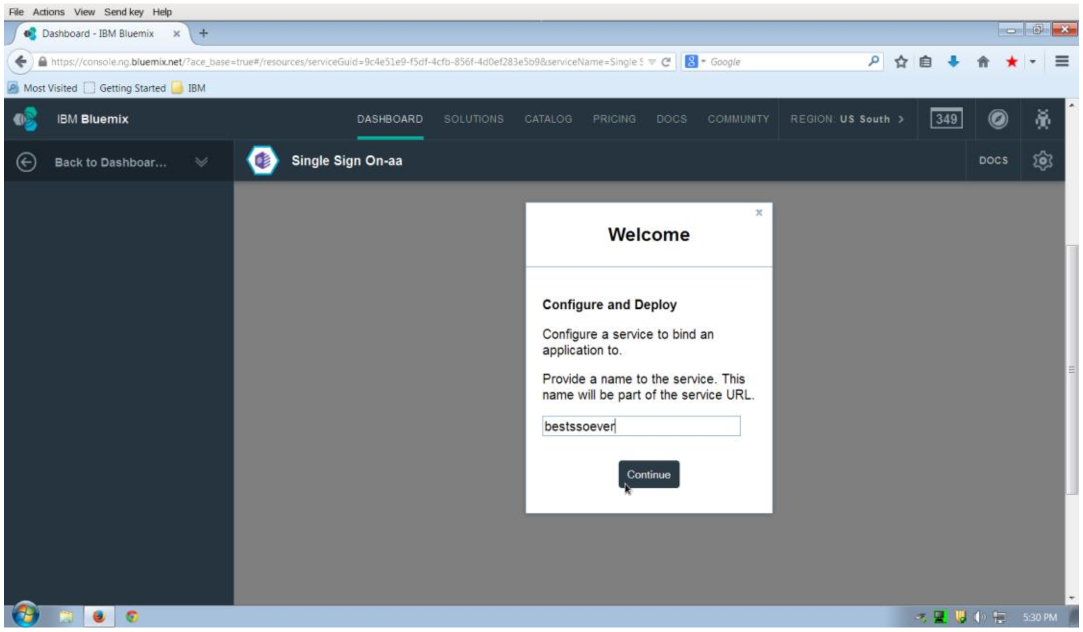
Space: ssolab

App: Leave unbound

Service name: Single Sign On-aa

Selected Plan: Standard

A panel will display to prompt for a “name” for the service. This is the id that you chose earlier, it is not the name that was shown in the previous panel as the Service name. Once provided, click on the Continue button:



Welcome

Configure and Deploy

Configure a service to bind an application to.

Provide a name to the service. This name will be part of the service URL.

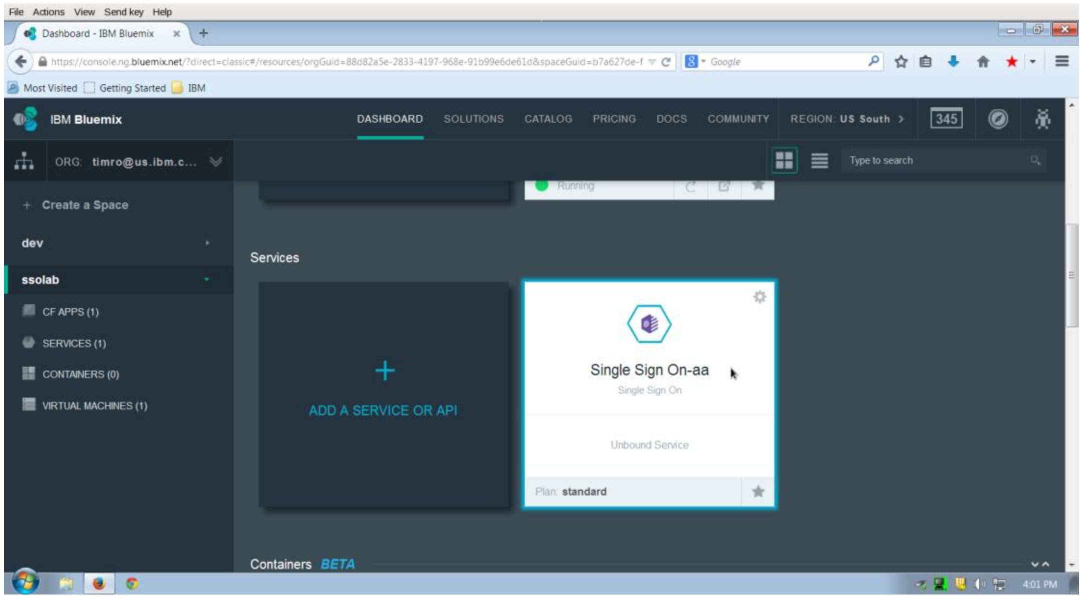
bestssoever

Continue

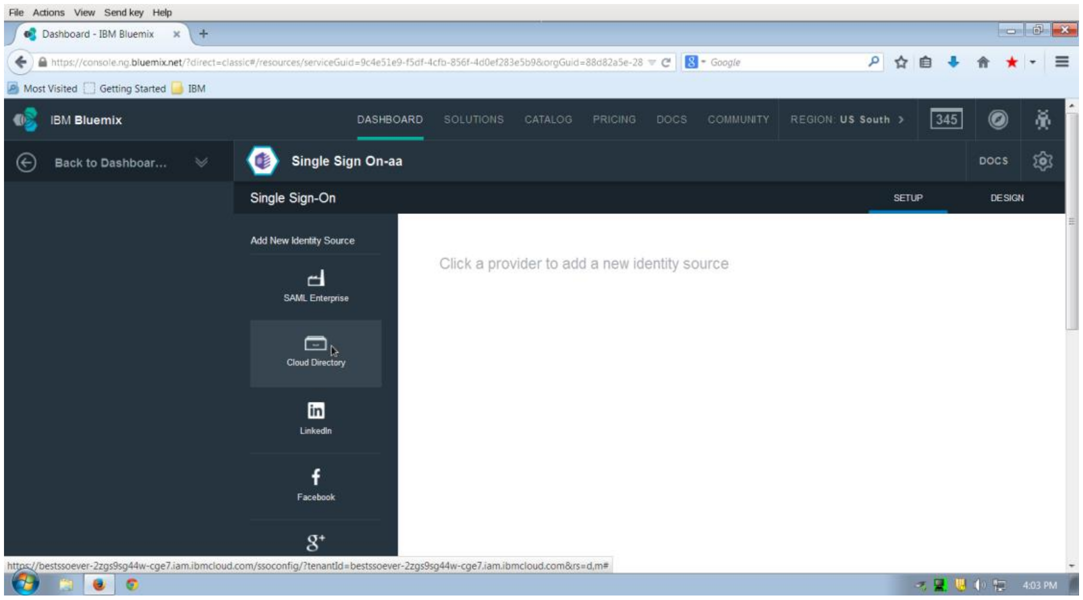
This will add the service to the dashboard and now we will be able to create a simple Cloud directory and add a testing user for external authentication in the next section of the lab.

Configure a Cloud directory identity source

Back in the IBM Bluemix Cloud Foundry dashboard, scroll down to find the SSO service that you just created. Click on the service to open up the configuration panel:

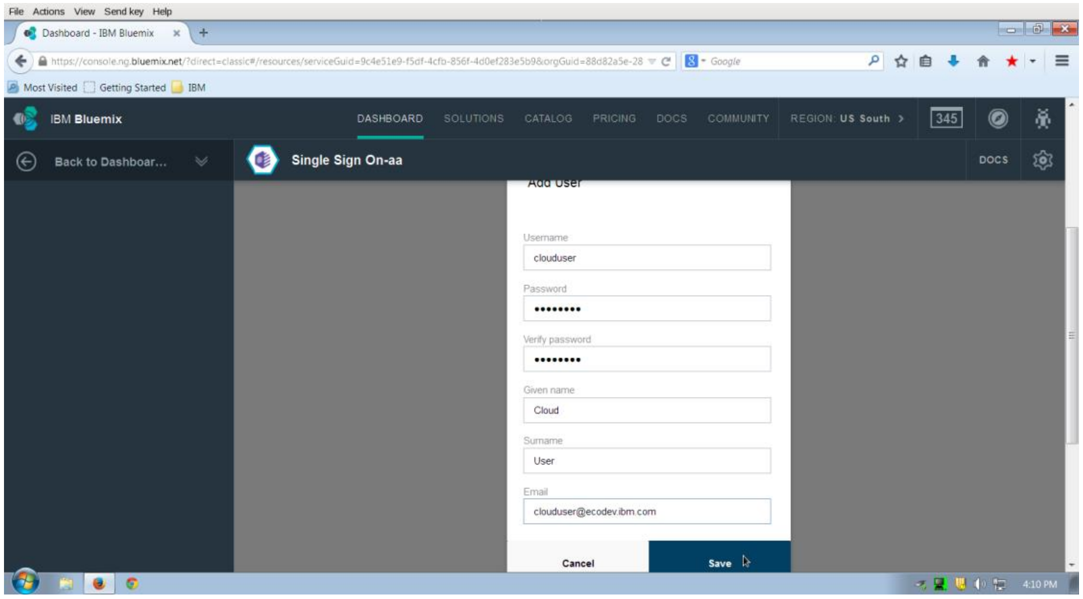


In the configuration panel, click on the Cloud Directory icon to add and configure an directory in the cloud for our lab exercises:



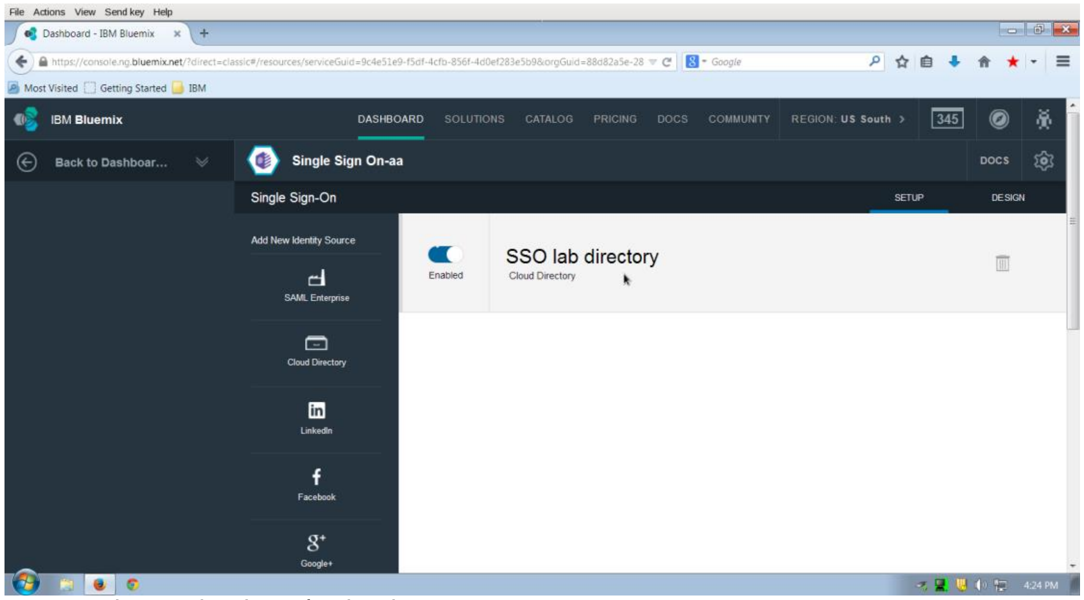
In the next screen, you may give the identity source a new name other than the default and see the current list of users. It's empty at first so click on the add icon:

And put in some information for a test user. This can be any user information that you prefer, however make a note of the userid and password set since that will be used when testing the external authentication service:



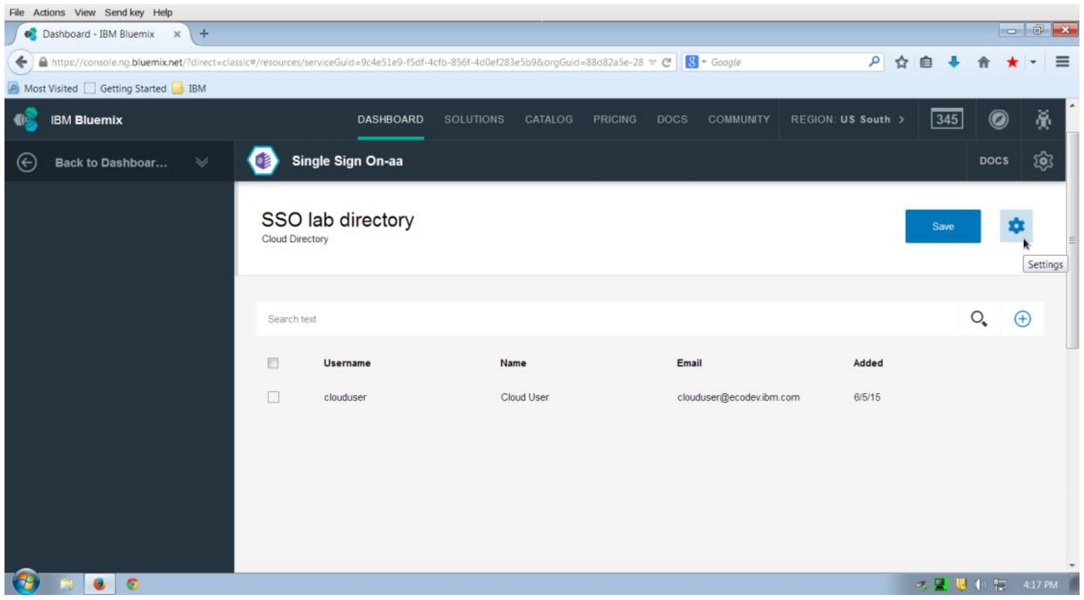
Username: clouduser
Password: ••••••••
Verify password: ••••••••
Given name: Cloud
Surname: User
Email: clouduser@ecodev.ibm.com

Save the user by clicking on the Save button, click on the Save button for the identity source configuration panel,



and then re-open the service by selecting it:

After re-opening the identity source configuration panel, a settings icon button will be shown in the upper right section of the panel. To manage the password policy applied to users in this identity source, you can click on the settings icon button for the identity source:



	Username	Name	Email	Added
☐	clouduser	Cloud User	clouduser@ecodev.ibm.com	6/5/15

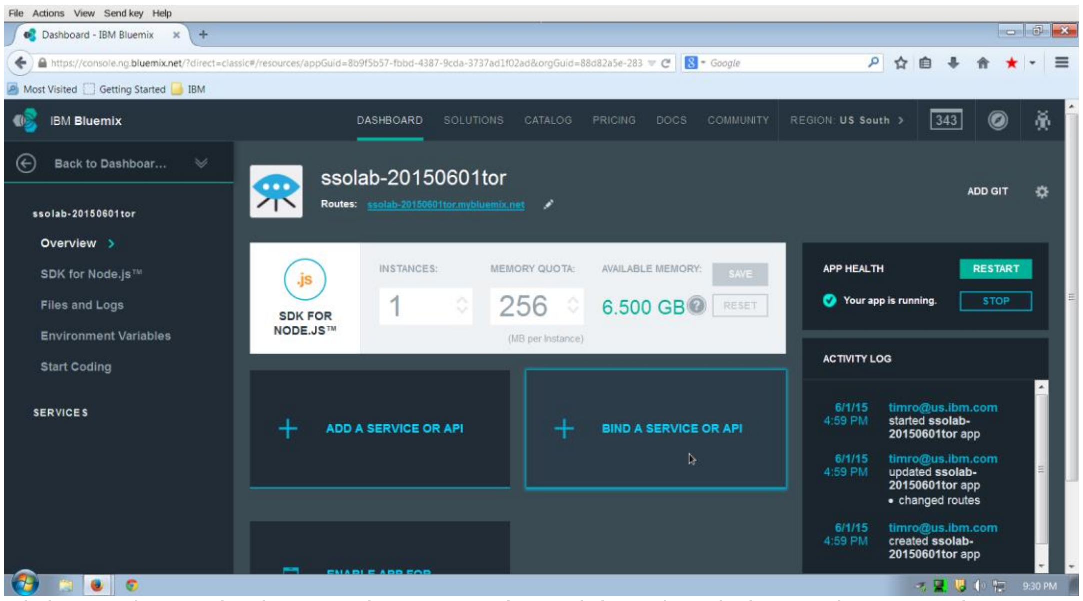
The settings control has two option panels. The Auto Consent option tells the SSO service if it should prompt the user after an authentication to allow the SSO service to retrieve personal (e.g. name and e-mail address) information from the identity source. Leave this setting to On for this lab to observe this prompt in action. The Password Policy option allows you to select various levels of password quality and lifetime. At the present time, these policies are fixed and you may select from one of the three. Choose Medium, and then click on Save.

Lastly, click on save in the identity source configuration panel to return to the main SSO configuration panel

The name shown for the identity source will be the name you selected or the default name of Cloud Directory. The toggle (shown as Enabled) next to the name allows the service administrator to enable or disable a particular identity source for use by applications. Leave the toggle at Enabled and click on the Back to Dashboard link in the upper left to return to the IBM Bluemix Cloud Foundry dashboard.

Bind the Single Sign On service to the application

To bind the Single Sign On service with the configured identity source, first click on the application icon in the



Your app is running.

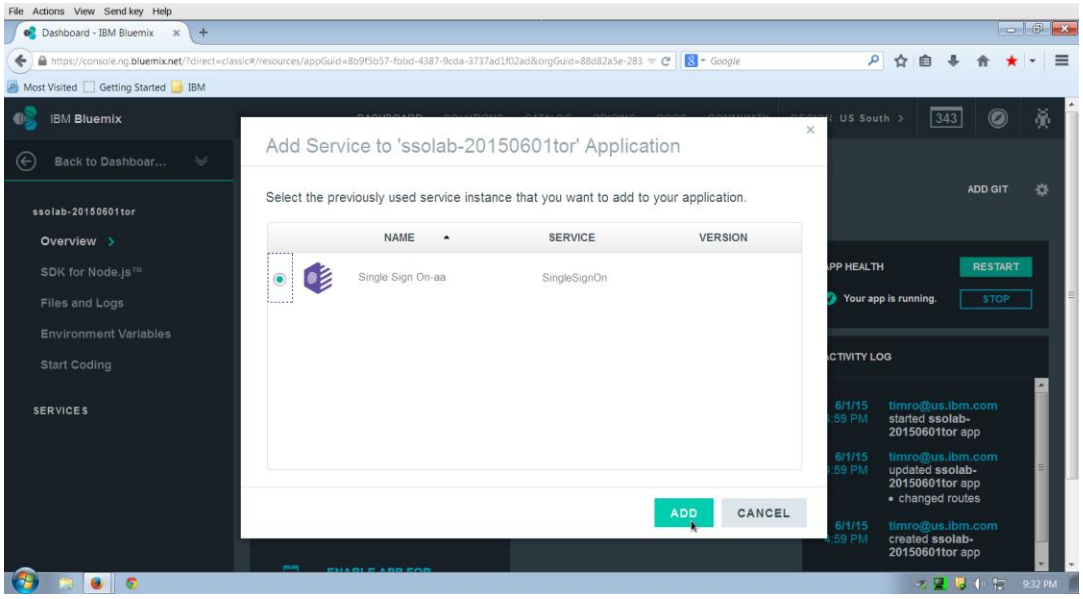
6/1/15 4:59 PM timro@us.ibm.com started ssolab-20150601tor app

6/1/15 4:59 PM timro@us.ibm.com updated ssolab-20150601tor app
• changed routes

6/1/15 4:59 PM timro@us.ibm.com created ssolab-20150601tor app

dashboard to bring up the application overview page. Then click on the Bind a Service or API button:

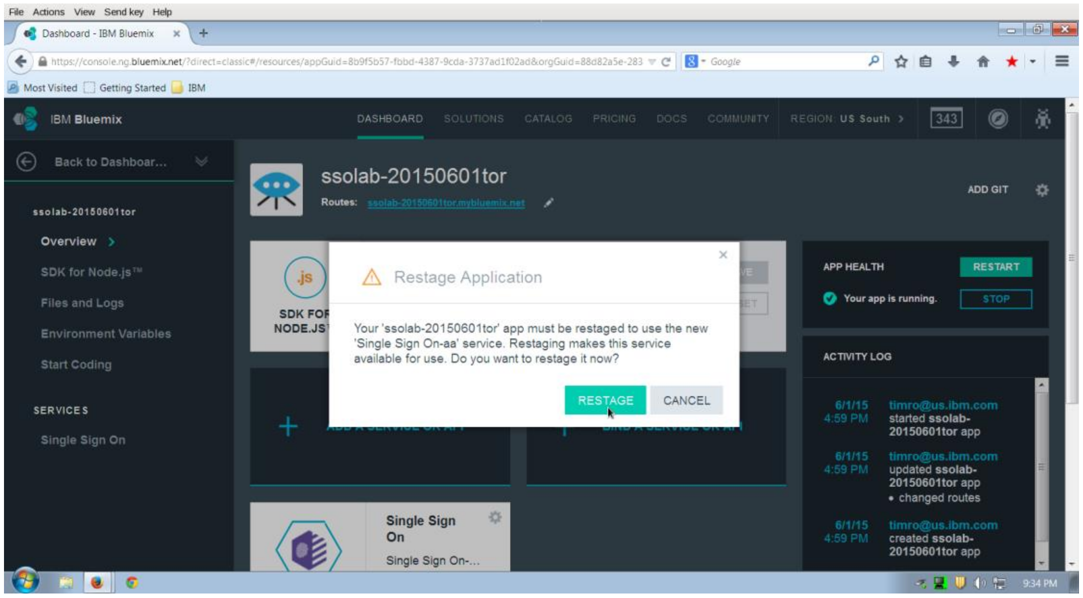
A panel will be displayed, select the radio button next to the Single Sign On service and then click on the ADD button:


Add Service to 'ssolab-20150601tor' Application

Select the previously used service instance that you want to add to your application.

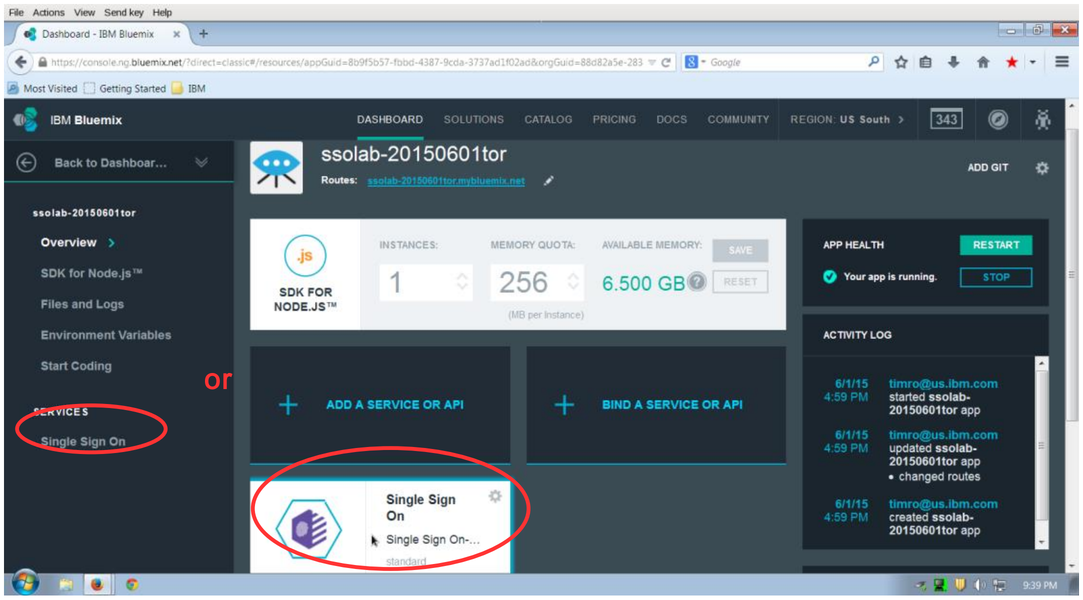
NAME	SERVICE	VERSION
Single Sign On-aa	SingleSignOn	

You will be prompted to restage the application. A restage is used to add the VCAP_SERVICES environment variables to the application so that it will be able to access the Single Sign On service bound to the application. Click on the RESTAGE button:

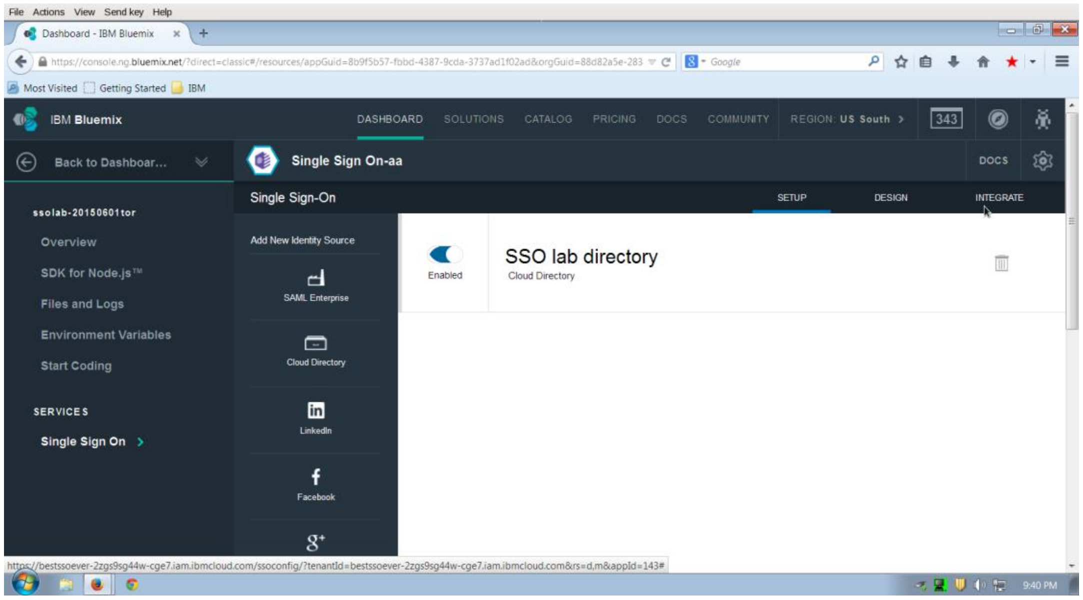

Restage Application

Your 'ssolab-20150601tor' app must be restaged to use the new 'Single Sign On-aa' service. Restaging makes this service available for use. Do you want to restage it now?

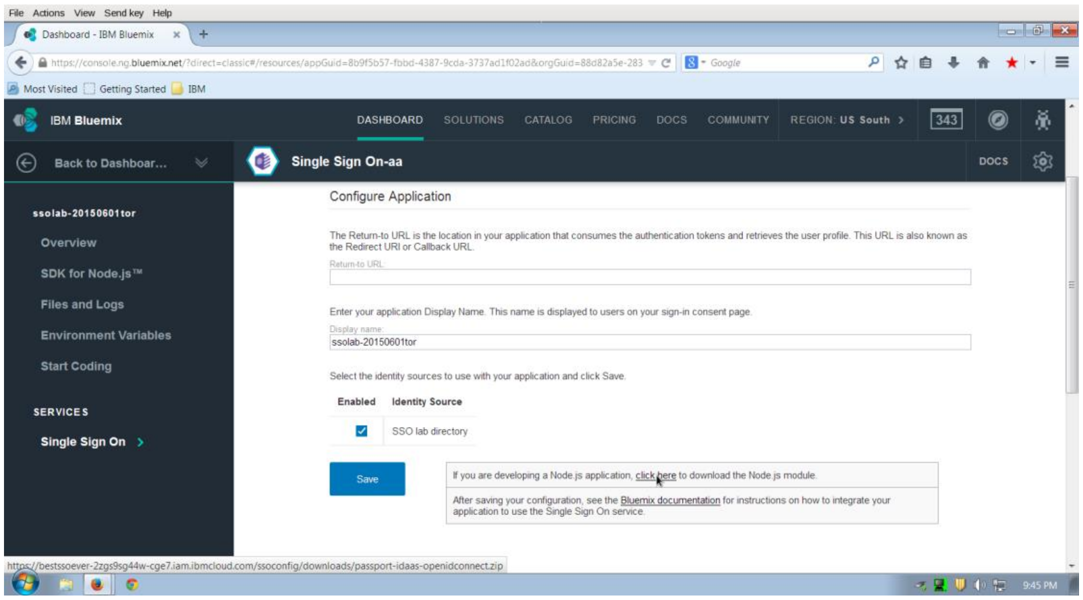
The restage and restart of the application will complete quickly, however, you do not need to wait before proceeding. On the left side of the application's overview panel, click on the Single Sign On label or click on the icon for the service. This will open the configuration panel for the service in the context of this application:



When accessing the configuration panel from within an application, an INTEGRATE tab will also be shown. For the Node.js application we are configuring, a module needs to be downloaded from the Single Sign On service. The link to this code is available from the INTEGRATE tab. Although we will return later to the INTEGRATE tab to finish configuring the application, click on the label to switch to this tab now:



You will need an additional Node.js module for your application that will provide support for OpenID Connect protocol for the passport service. To download this code, find the "click here" link in the light gray text box (see screenshot), then click on it to download the module.



Configure Application

The Return-to URL is the location in your application that consumes the authentication tokens and retrieves the user profile. This URL is also known as the Redirect URI or Callback URL.

Return-to URL:

Enter your application Display Name. This name is displayed to users on your sign-in consent page.

Display name: ssolab-20150601tor

Select the identity sources to use with your application and click Save.

Enabled	Identity Source
✓	SSO lab directory

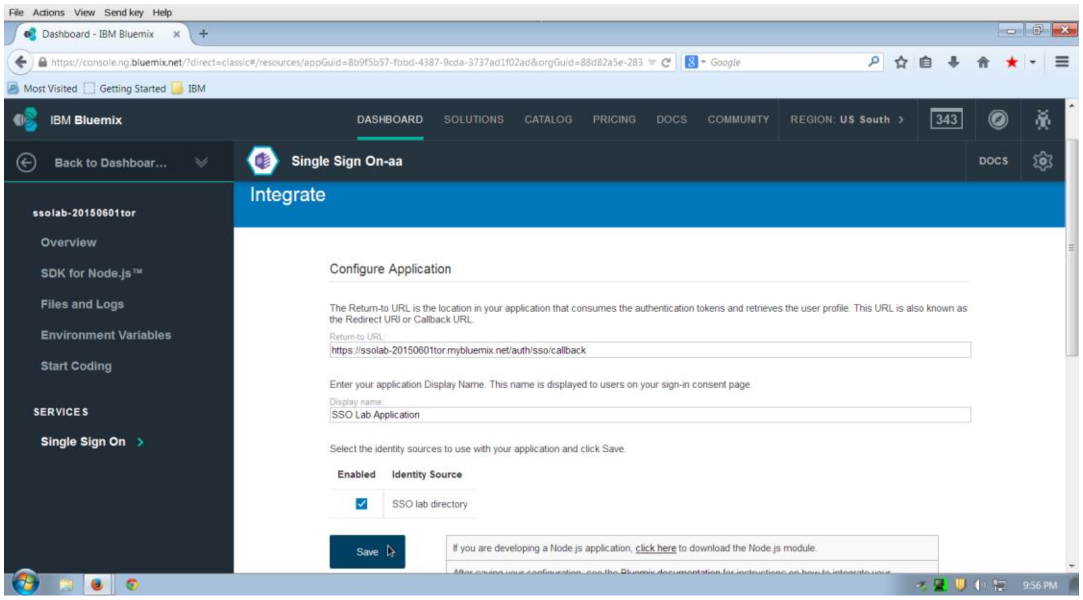
If you are developing a Node.js application, click here to download the Node.js module.

After saving your configuration, see the Bluemix documentation for instructions on how to integrate your application to use the Single Sign On service.

Next, you will enter in the Callback URL for the SSO service to invoke after the authentication is completed in the Return-to URL box. The host part of this Callback URL must match the host name you specified when you created the application. The rest of the host name should be the default bluemix domain. Finish the URL with the route "/auth/sso/callback", you will also update the code for this route in the Node.js starter application. In the examples shown in this lab, the correct url is:

https://ssolab-20150601tor.mybluemix.net/auth/sso/callback

You may change the Display name for the application from the default. This is the name used by the SSO service when prompting a user for consent to allow the retrieval of personal information from the identity source. Once the



Integrate

Configure Application

The Return-to URL is the location in your application that consumes the authentication tokens and retrieves the user profile. This URL is also known as the Redirect URI or Callback URL.

Return-to URL: https://ssolab-20150601tor.mybluemix.net/auth/sso/callback

Enter your application Display Name. This name is displayed to users on your sign-in consent page.

Display name: SSO Lab Application

Select the identity sources to use with your application and click Save.

Enabled	Identity Source
✓	SSO lab directory

If you are developing a Node.js application, click here to download the Node.js module.

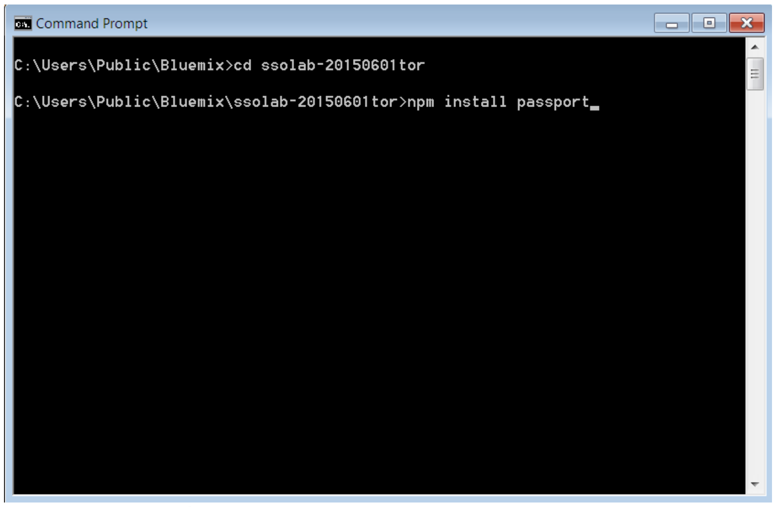
callback url has been entered and the Display name updated (if desired), click on the Save button:

After saving, click on Back to Dashboard in the upper left to return to the IBM Bluemix Cloud Foundry dashboard.

At this point, the SSO service is ready to be used with the application. Now it is time to update the application source code to use the service.

Update the Node.js starter application to use the Single Sign On service for external authentication:

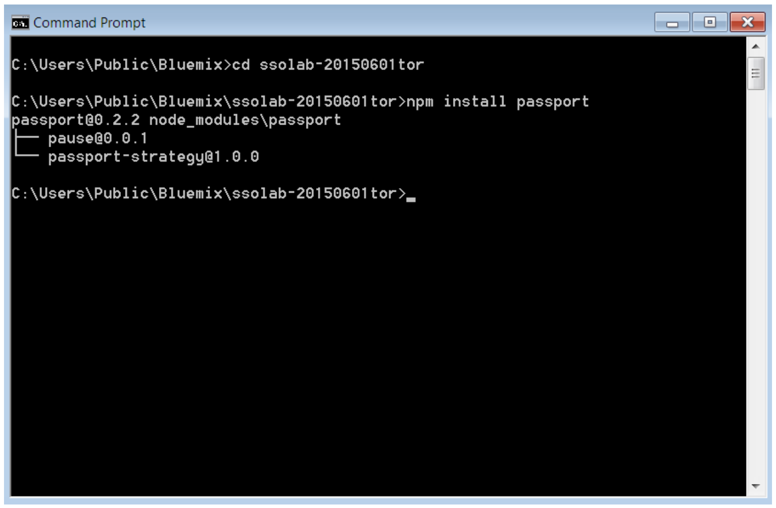
Click on your Node.js application from the dashboard view and then select the Start Coding panel to bring back the listing of steps that we saw in the first section. If you haven't done so already, open a terminal on your workstation and perform steps 1, 2 and 3. Within the top source code directory for your application on your workstation, use the



```
C:\Users\Public\Bluemix>cd ssolab-20150601tor

C:\Users\Public\Bluemix\ssolab-20150601tor>npm install passport_
```

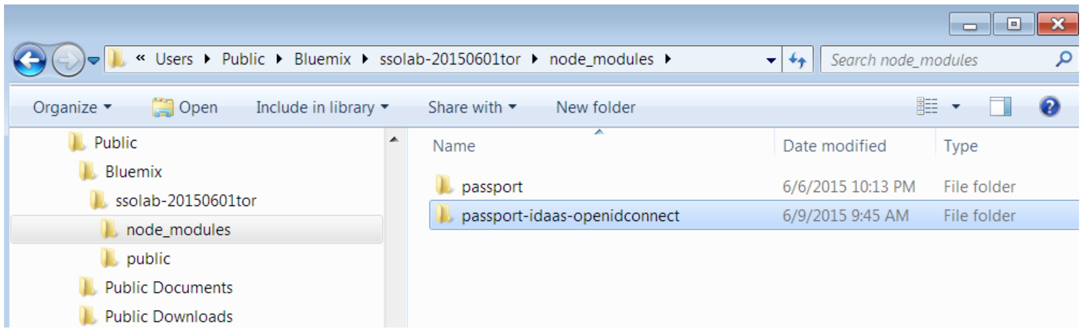
node npm tool to install the passport module:

This should complete without errors and place the passport module and any other dependencies inside the node_modules directory for the application:



```
C:\Users\Public\Bluemix>cd ssolab-20150601tor

C:\Users\Public\Bluemix\ssolab-20150601tor>npm install passport
passport@0.2.2 node_modules\passport
├── pause@0.0.1
└── passport-strategy@1.0.0

C:\Users\Public\Bluemix\ssolab-20150601tor>_
```

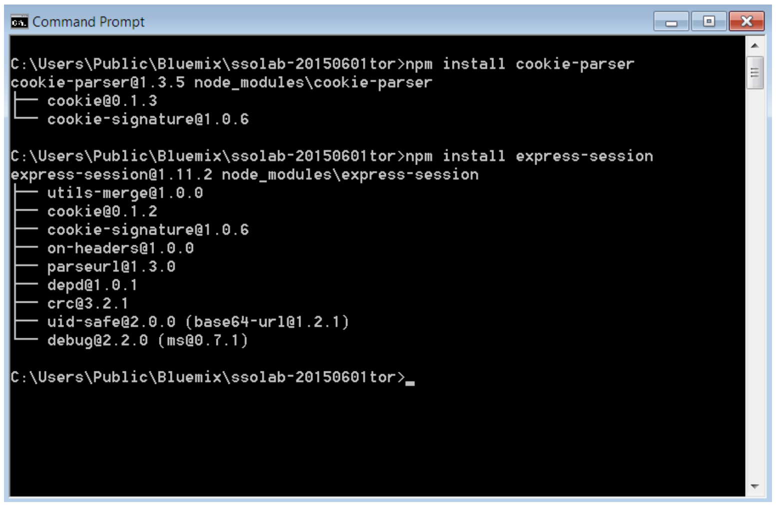
Next, locate from your default download location the node module from the integrate tab (the zip file name will be **passport-idaas-openidconnect.zip** if you need to do a local search for it). Extract the contents of the zip file into the node_modules folder of your application. Once complete, the node_modules folder contents should look like:



Name	Date modified	Type
passport	6/6/2015 10:13 PM	File folder
passport-idaas-openidconnect	6/9/2015 9:45 AM	File folder

From the same command window, run two more npm commands:

```
npm install cookie-parser
```



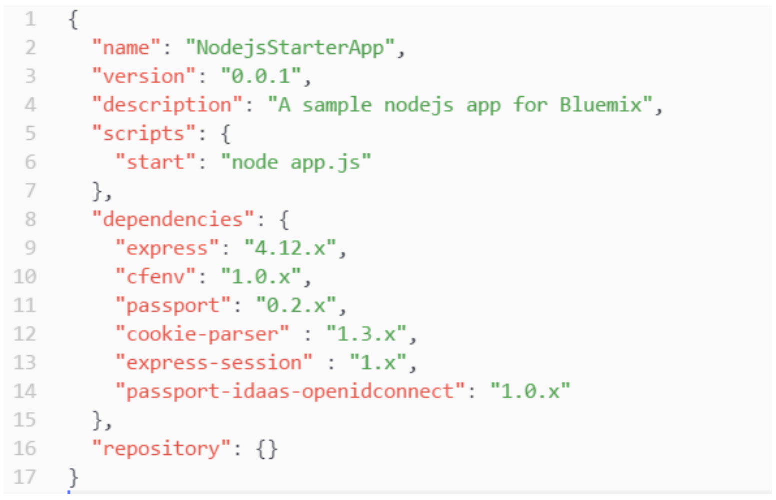
```
C:\Users\Public\Bluemix\ssolab-20150601tor>npm install cookie-parser
cookie-parser@1.3.5 node_modules\cookie-parser
├── cookie@0.1.3
└── cookie-signature@1.0.6

C:\Users\Public\Bluemix\ssolab-20150601tor>npm install express-session
express-session@1.11.2 node_modules\express-session
├── utils-merge@1.0.0
├── cookie@0.1.2
├── cookie-signature@1.0.6
├── on-headers@1.0.0
├── parseurl@1.3.0
├── depd@1.0.1
├── crc@3.2.1
├── uid-safe@2.0.0 (base64-url@1.2.1)
└── debug@2.2.0 (ms@0.7.1)

C:\Users\Public\Bluemix\ssolab-20150601tor>_
```

```
npm install express-session
```

To ensure that module dependencies are retained when you push the application back into bluemix, use your preferred text editor to open the **package.json** file from the application code top directory and add the following lines into the dependencies section:

```
"passport": "0.2.x",
"cookie-parser" : "1.3.x",
"express-session" : "1.x"
"passport-idaas-openidconnect": "1.0.x"
```

It should look something like this when complete:

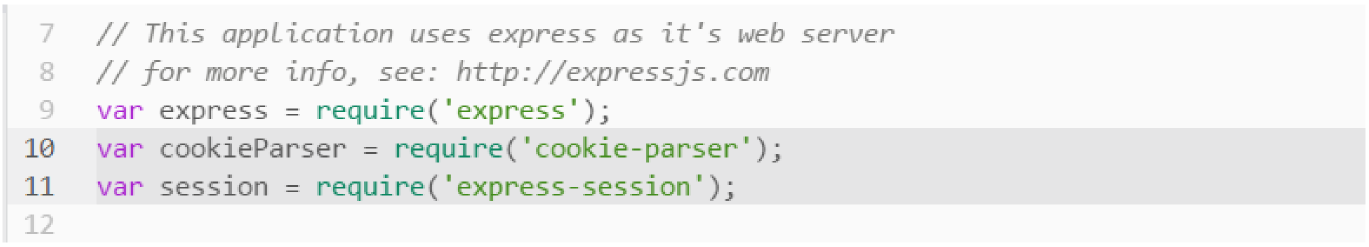


```json
{
  "name": "NodejsStarterApp",
  "version": "0.0.1",
  "description": "A sample nodejs app for Bluemix",
  "scripts": {
    "start": "node app.js"
  },
  "dependencies": {
    "express": "4.12.x",
    "cfenv": "1.0.x",
    "passport": "0.2.x",
    "cookie-parser" : "1.3.x",
    "express-session" : "1.x",
    "passport-idaas-openidconnect": "1.0.x"
  },
  "repository": {}
}
```

Lastly, if there is a **.cfignore** file in the application top-level directory, remove any line that specifies `node_modules` unless it contains a subdirectory (e.g. `node_modules/cookie-parser` is ok, but `node_modules` is not).

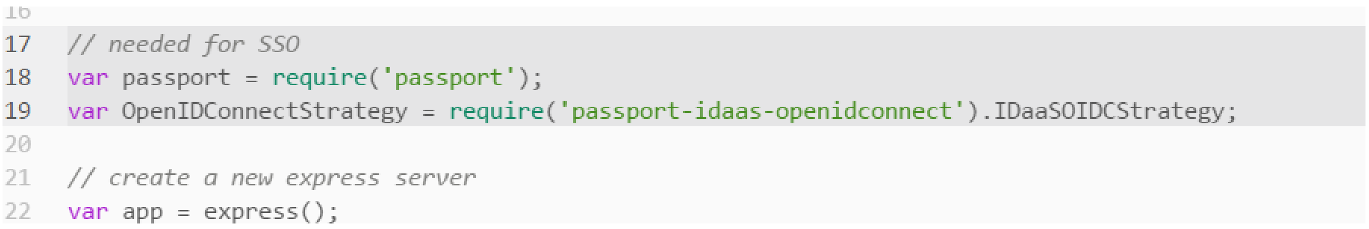
Next, we will add some code to the application to use the Single Sign On service to manage authentication of access to specified routes within the application. There is an appendix to the lab with the full source content if you would prefer to copy and paste in the updates.

Open the app.js file in the application top level directory and add the highlighted code just below the `var express = require('express');` statement:



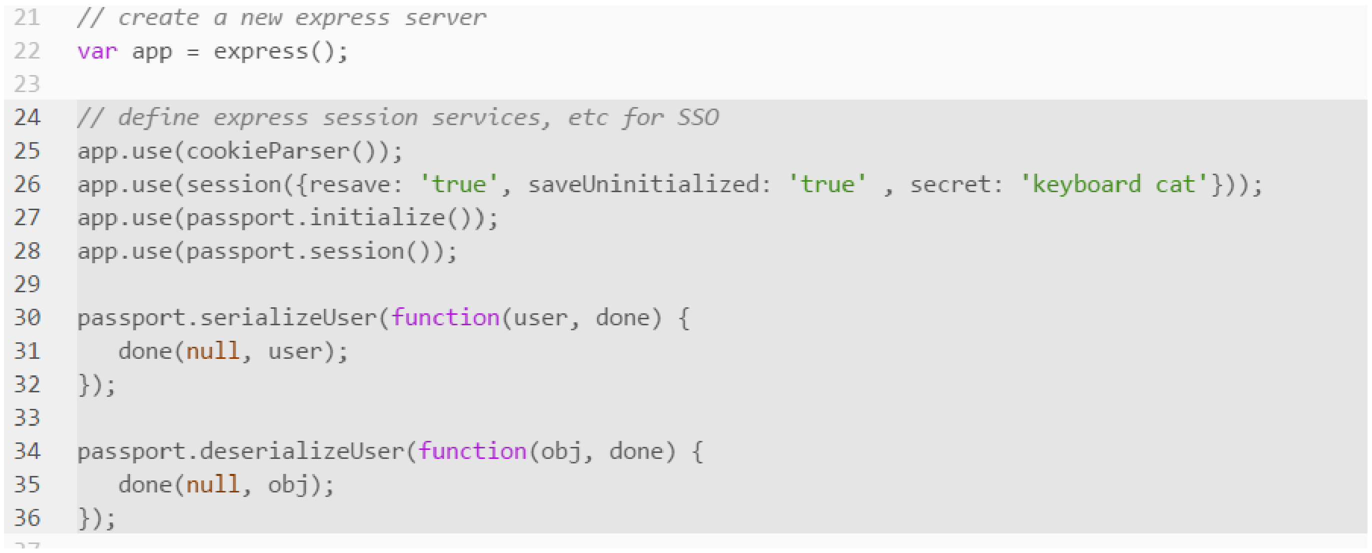
```javascript
// This application uses express as it's web server
// for more info, see: http://expressjs.com
var express = require('express');
var cookieParser = require('cookie-parser');
var session = require('express-session');

```

Before the `var app = express();` statement, add this highlighted code:



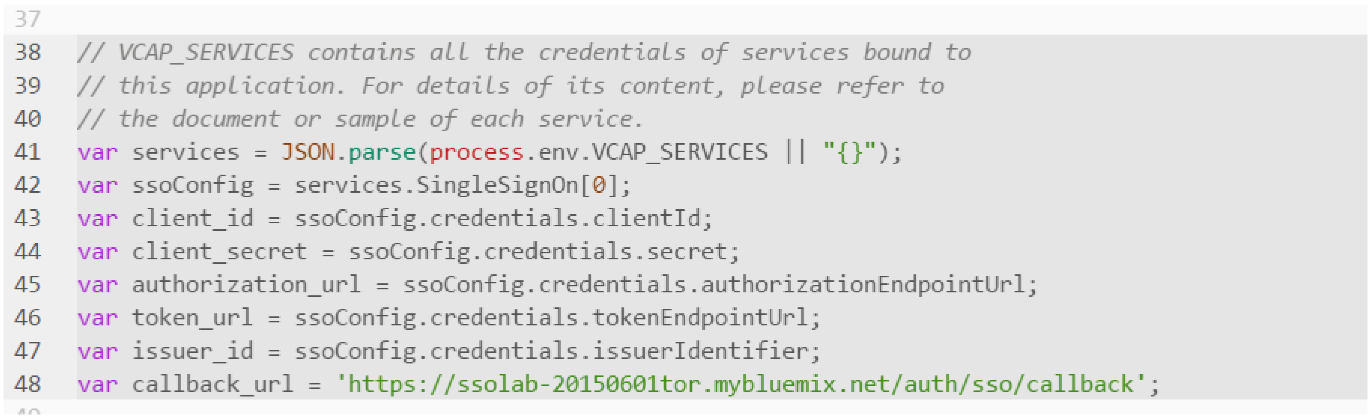
```javascript
// needed for SSO
var passport = require('passport');
var OpenIDConnectStrategy = require('passport-idaas-openidconnect').IDaaSOIDCStrategy;

// create a new express server
var app = express();
```

And after the `var app = express();` statement, add code to create session handling services in express and configure the passport module to use it with this highlighted code:



```
// create a new express server
var app = express();

// define express session services, etc for SSO
app.use(cookieParser());
app.use(session({resave: 'true', saveUninitialized: 'true' , secret: 'keyboard cat'}));
app.use(passport.initialize());
app.use(passport.session());

passport.serializeUser(function(user, done) {
   done(null, user);
});

passport.deserializeUser(function(obj, done) {
   done(null, obj);
});
```

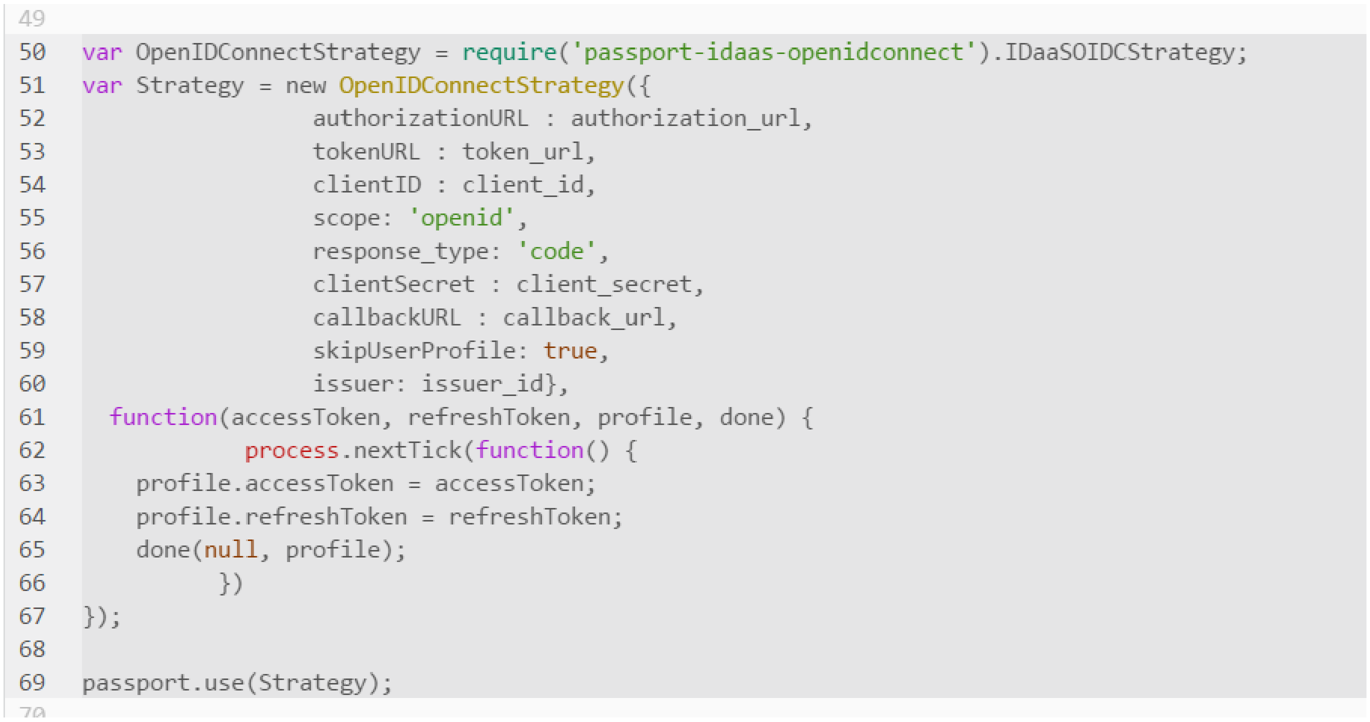
Following this code, we will continue by adding some logic to process the entries in the VCAP_SERVICES environment variable. When the SSO service is bound to the application, the service instance details including the client id, client secret, authorization url, token url, and issuer id used in the OpenID Connect stragegy for passport are set in VCAP_SERVICES. Along with these entries, you will need to add the string used for the callback url in the INTEGRATE tab previously:



```
// VCAP_SERVICES contains all the credentials of services bound to
// this application. For details of its content, please refer to
// the document or sample of each service.
var services = JSON.parse(process.env.VCAP_SERVICES || "{}");
var ssoConfig = services.SingleSignOn[0];
var client_id = ssoConfig.credentials.clientId;
var client_secret = ssoConfig.credentials.secret;
var authorization_url = ssoConfig.credentials.authorizationEndpointUrl;
var token_url = ssoConfig.credentials.tokenEndpointUrl;
var issuer_id = ssoConfig.credentials.issuerIdentifier;
var callback_url = 'https://ssolab-20150601tor.mybluemix.net/auth/sso/callback';
```

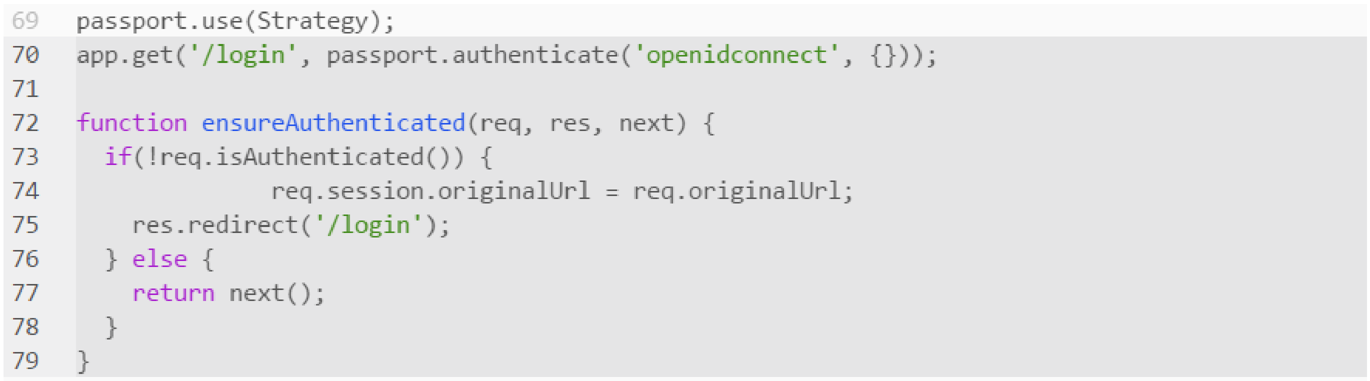
^^^^^^^^^^^^^^^^^^^^^ – replace with your application name to match your callback url.

These entries will then be used in a constructor for a new OpenID Connect strategy which will then be provided to passport. Code to be added is highlighted and immediately follows the previous code:



```

var OpenIDConnectStrategy = require('passport-idaas-openidconnect').IDaaSOIDCStrategy;
var Strategy = new OpenIDConnectStrategy({
                 authorizationURL : authorization_url,
                 tokenURL : token_url,
                 clientID : client_id,
                 scope: 'openid',
                 response_type: 'code',
                 clientSecret : client_secret,
                 callbackURL : callback_url,
                 skipUserProfile: true,
                 issuer: issuer_id},
  function(accessToken, refreshToken, profile, done) {
            process.nextTick(function() {
    profile.accessToken = accessToken;
    profile.refreshToken = refreshToken;
    done(null, profile);
          })
});

passport.use(Strategy);
```

For this sample application we will define a route in express that will be used to initiate authentication for a session. Then we will define a function that can be added to any express route to ensure that a session web session



```
passport.use(Strategy);
app.get('/login', passport.authenticate('openidconnect', {}));

function ensureAuthenticated(req, res, next) {
  if(!req.isAuthenticated()) {
              req.session.originalUrl = req.originalUrl;
    res.redirect('/login');
  } else {
    return next();
  }
}
```

accessing the route is authenticated:

Next, we need to define the authentication callback url. This authentication callback function implementation will read the original request url from the session and if the authentication has been successful, continue processing the original url request. If the authentication has been unsuccessful, the user will be routed to a simple failure route:

```
}

app.get('/auth/sso/callback',function(req,res,next) {
      var redirect_url = req.session.originalUrl;
            passport.authenticate('openidconnect',{
                 successRedirect: redirect_url,
                 failureRedirect: '/failure',
          })(req,res,next);
        });
```

Finally, we will add two routes to the application for user interaction. The first invokes the ensureAuthenticated() function to check to see if the session is authenticated and if not, this function will authenticate the user and return. When authenticated, the first route will return a simple message based upon the unique id provided to the Single Sign On service by the identity provider. The second route provides the message when an authentication fails:

```

app.get('/hello', ensureAuthenticated, function(req, res) {
        res.send('Hello, '+ req.user['id'] + '!'); });

app.get('/failure', function(req, res) {
        res.send('login failed'); });
```

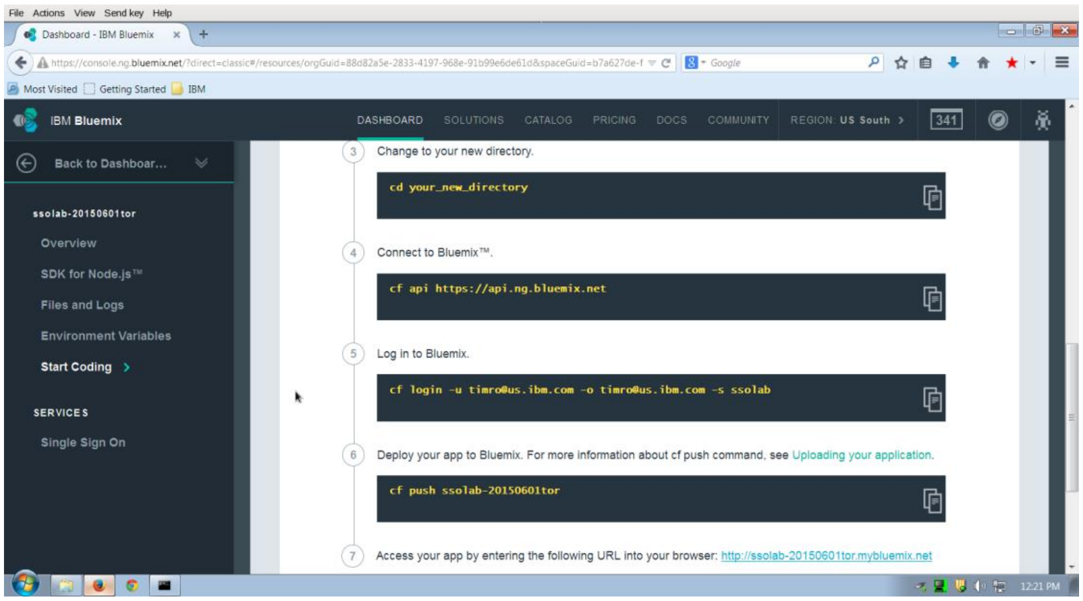
This completes the changes to the app.js code. Save from the editor and open up the **public/index.html** file where we will make some minor changes to the starter static html. Change the content in the <span class = "blue"> element to something more inspiring, and then add a simple form button as an additional table row:

```
          <h1>Hi there!</h1>
          <p>Thanks for creating a
            <span class = "blue">NodeJS Application using Single Sign On!</span>.
            Get started by reading our
            <a href = "https://www.ng.bluemix.net/docs/#starters/nodejs/index.html#nodejs">document
            or use the Start Coding guide under your app in your dashboard.
      <tr>
        <td>
          <form method="get" action="/hello">
            <input type="submit" value="Login">
          </form>
    </table>
```

Save this file and then return to the command prompt window for the next step.

Publish your updated application to Bluemix

Return to the Bluemix Cloud Foundry dashboard and click on your application icon. Then select the Start Coding



3 Change to your new directory.

```
cd your_new_directory
```

4 Connect to Bluemix™.

```
cf api https://api.ng.bluemix.net
```

5 Log in to Bluemix.

```
cf login -u timro@us.ibm.com -o timro@us.ibm.com -s ssolab
```

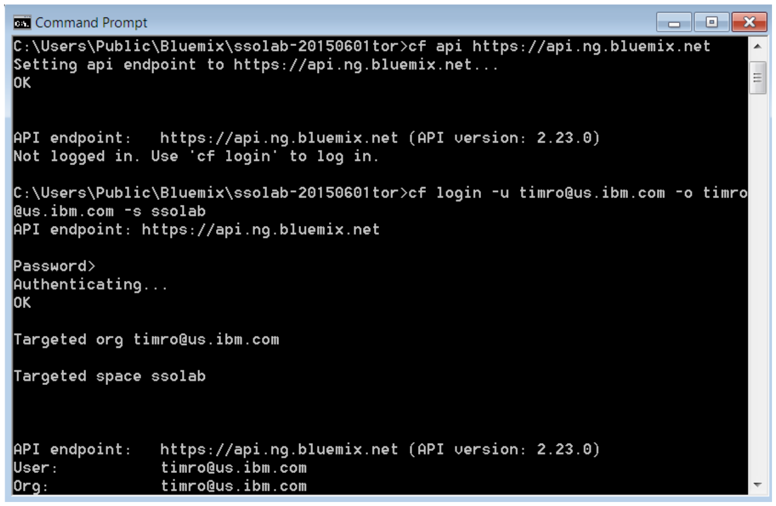
6 Deploy your app to Bluemix. For more information about cf push command, see Uploading your application.

```
cf push ssolab-20150601tor
```

7 Access your app by entering the following URL into your browser: http://ssolab-20150601tor.mybluemix.net

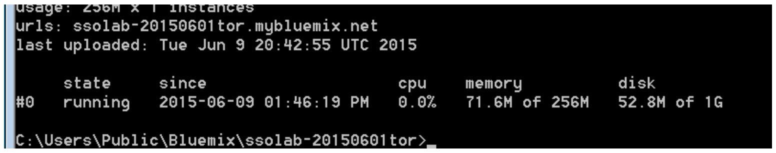
link from the left hand list. Scroll down in the display panel to see steps 4, 5, 6:

Return to or open a command prompt on your workstation, change to the directory with your updated application and perform steps 4 and 5 as shown (it will be different for your bluemix user, space and application name than the screenshot). You will see output like:



```
C:\Users\Public\Bluemix\ssolab-20150601tor>cf api https://api.ng.bluemix.net
Setting api endpoint to https://api.ng.bluemix.net...
OK


API endpoint:   https://api.ng.bluemix.net (API version: 2.23.0)
Not logged in. Use 'cf login' to log in.

C:\Users\Public\Bluemix\ssolab-20150601tor>cf login -u timro@us.ibm.com -o timro
@us.ibm.com -s ssolab
API endpoint: https://api.ng.bluemix.net

Password>
Authenticating...
OK

Targeted org timro@us.ibm.com

Targeted space ssolab



API endpoint:   https://api.ng.bluemix.net (API version: 2.23.0)
User:           timro@us.ibm.com
Org:            timro@us.ibm.com
```

Then push your application (step 6) using `cf push <appname>` . There will be a lot of output as the application is uploaded back to bluemix and then started. If all goes well, you will see something like this at the end:

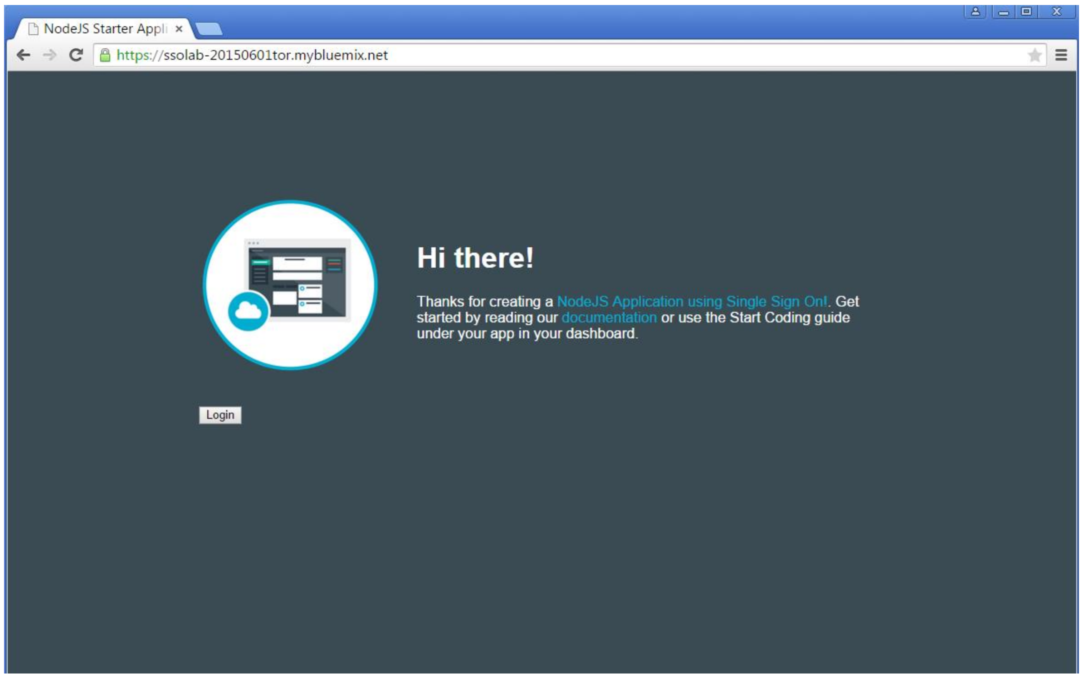


```
usage: 256M x 1 instances
urls: ssolab-20150601tor.mybluemix.net
last uploaded: Tue Jun 9 20:42:55 UTC 2015

     state     since                    cpu    memory         disk
#0   running   2015-06-09 01:46:19 PM   0.0%   71.6M of 256M   52.8M of 1G

C:\Users\Public\Bluemix\ssolab-20150601tor>_
```

Testing the application

When testing with the Single Sign On service with the Cloud Directory, it can be easier to use a different browser security and cookie context (for example in a new "private" window in Firefox) to simulate the experience of a new user to your application. Or you can also just use an entirely different browser. Here we'll use a Chrome browser in the screenshots.

Begin by opening your application's url: https://<appname>.mybluemix.net (be sure to use https!):



NodeJS Starter Appli

https://ssolab-20150601tor.mybluemix.net

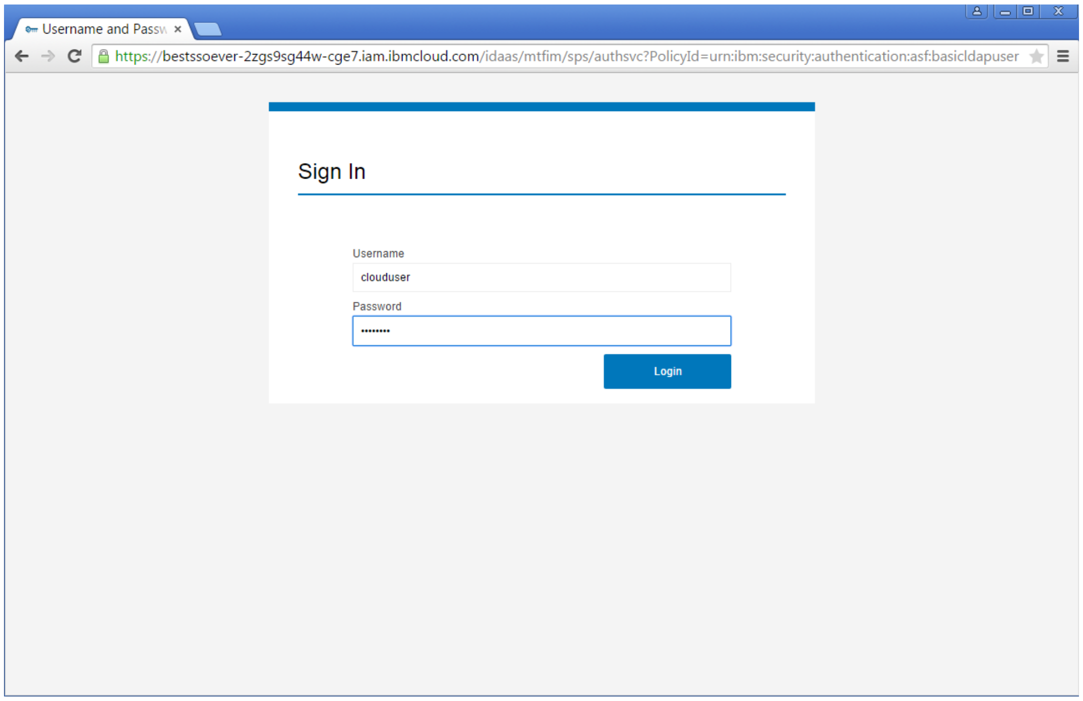
Hi there!

Thanks for creating a NodeJS Application using Single Sign On!. Get started by reading our documentation or use the Start Coding guide under your app in your dashboard.

Login

Note that your customized message is appearing in the blue text and that the login button you added is present. Click on the login button to bring up the Single Sign On identity provider chooser list:

Since the Single Sign On service is configured with only one identity source, only one option is shown. Click on Cloud Directory and sign in as the user you added to the directory previously:


Username and Passw ×

https://bestssoever-2zgs9sg44w-cge7.iam.ibmcloud.com/idaas/mtfim/sps/authsvc?PolicyId=urn:ibm:security:authentication:asf:basicldapuser

Sign In

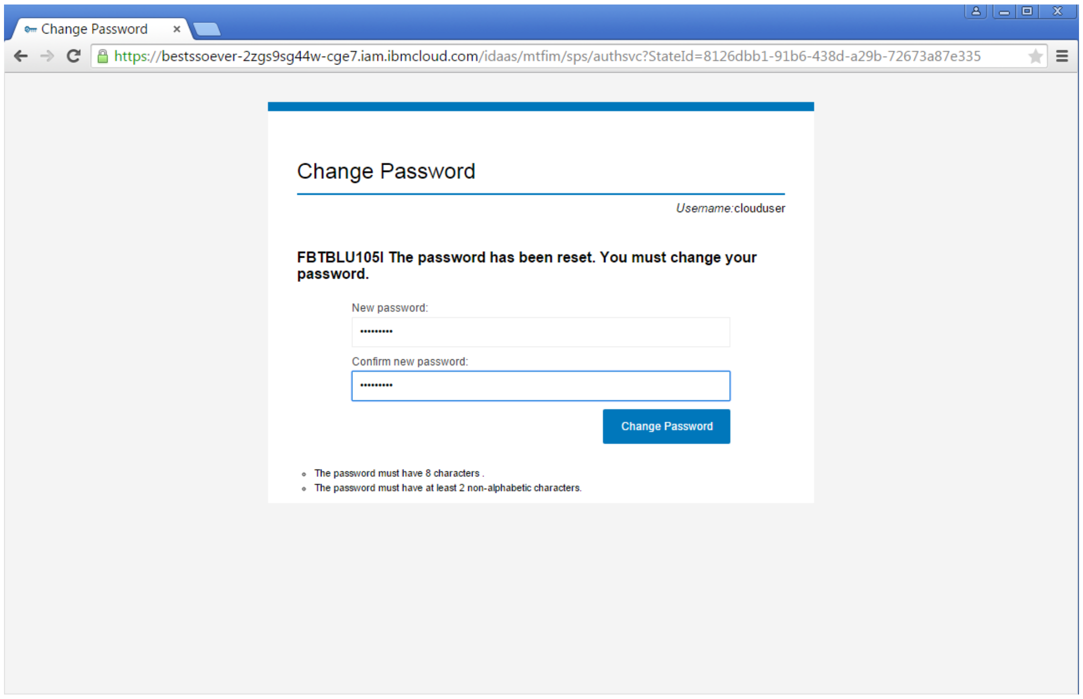
Username

clouduser

Password

••••••••

After you click on the Login button – a prompt will be displayed (this happens by policy when using the Cloud Directory) stating that the user password has been reset and must be changed. Note that requirements for the password are displayed in the form. Enter in an updated password:


Change Password ×

https://bestssoever-2zgs9sg44w-cge7.iam.ibmcloud.com/idaas/mtfim/sps/authsvc?StateId=8126dbb1-91b6-438d-a29b-72673a87e335

Change Password

*Username:*clouduser

FBTBLU105I The password has been reset. You must change your password.

New password:

•••••••••

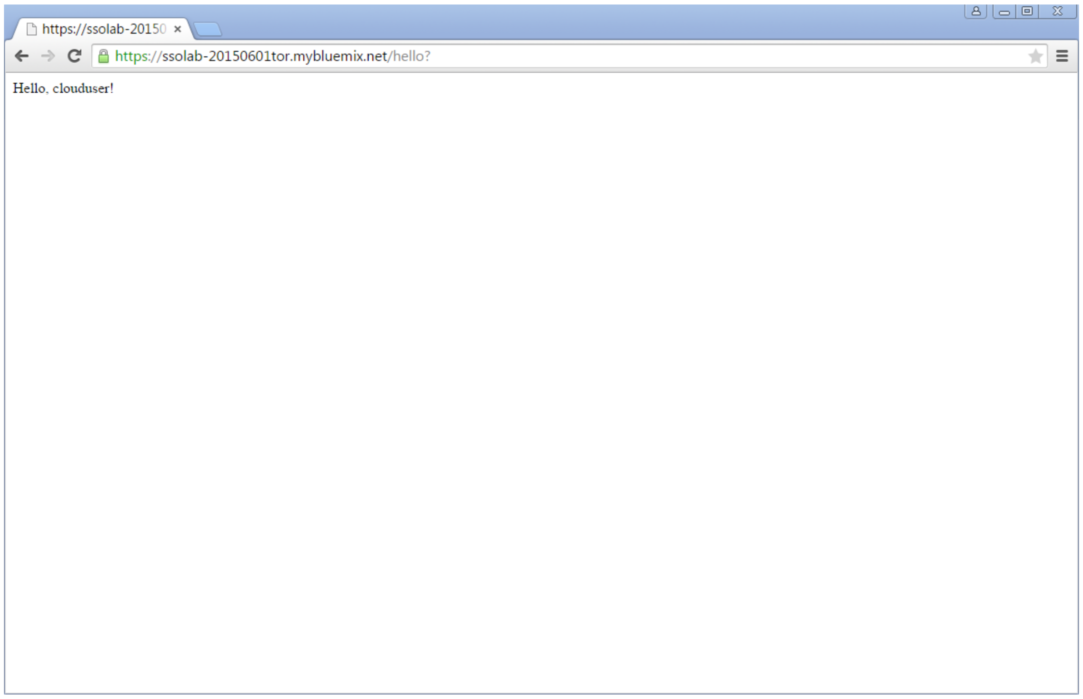
Confirm new password:

•••••••••

- The password must have 8 characters .
- The password must have at least 2 non-alphabetic characters.

Next, there will be a panel asking for permission (from the application name you set in the INTEGRATE panel) to retrieve your personal information from the external authentication identity provider. You may select either Allow or Allow and Remember. If you select Allow, then each time you authenticate with this identity provider through the Single Sign On service, the prompt will be displayed.

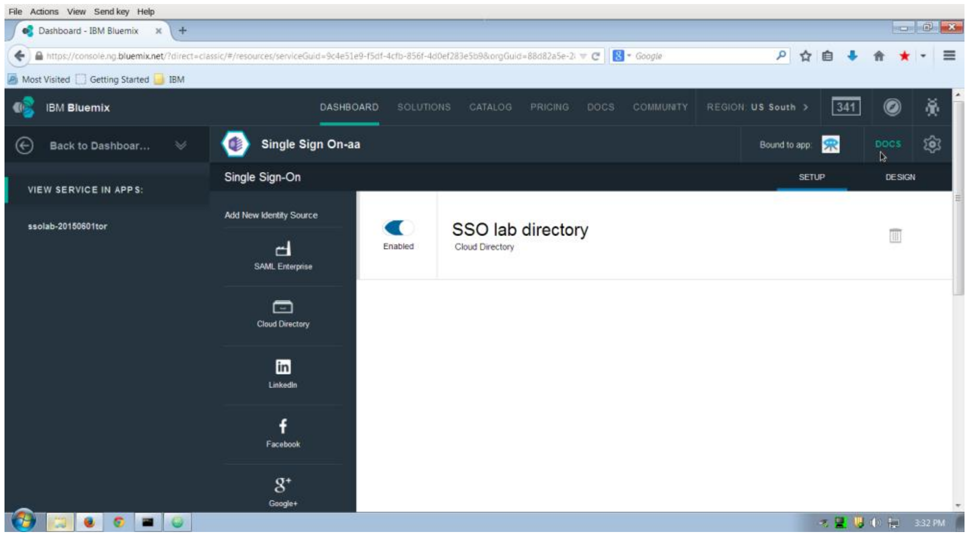
Then the application displays the successful authentication message:



Hello, clouduser!

Summary and next steps

Now you have an application that can use the Single Sign On service with a simple identity provider. A very good next step would be to continue on with the Bluemix SSO documentation to add a Social Media identity provider. There are step-by-step instructions for this as well as configuration of SAML Enterprise identity sources. To access the documentation, from the configuration panel for the Single Sign On service, click on the DOCS button:



Appendix – Code Sample for app.js

Here is a copy of the app.js file that includes the additions from all of the lab sections. If you would prefer to copy and paste code to move through the lab faster and avoid possible typos this should be help. Be sure to update the callback_url variable to match your Bluemix application hostname!

==

```
/*jshint node:true*/



//------------------------------------------------------------------------------

// node.js starter application for Bluemix

//------------------------------------------------------------------------------



// This application uses express as it's web server

// for more info, see: http://expressjs.com

var express = require('express');

var cookieParser = require('cookie-parser');

var session = require('express-session');



// cfenv provides access to your Cloud Foundry environment

// for more info, see: https://www.npmjs.com/package/cfenv

var cfenv = require('cfenv');



// needed for SSO

var passport = require('passport');

var OpenIDConnectStrategy = require('passport-idaas-openidconnect').IDaaSOIDCStrategy;
```

```
// create a new express server

var app = express();



// define express session services, etc for SSO

app.use(cookieParser());

app.use(session({resave:'true', saveUninitialized:'true' , secret:'keyboard cat'}));

app.use(passport.initialize());

app.use(passport.session());



passport.serializeUser(function(user, done) {

   done(null, user);

});



passport.deserializeUser(function(obj, done) {

   done(null, obj);

});



// VCAP_SERVICES contains all the credentials of services bound to

// this application. For details of its content, please refer to

// the document or sample of each service.

var services = JSON.parse(process.env.VCAP_SERVICES || "{}");

var ssoConfig = services.SingleSignOn[0];

var client_id = ssoConfig.credentials.clientId;
var client_secret = ssoConfig.credentials.secret;

var authorization_url = ssoConfig.credentials.authorizationEndpointUrl;

var token_url = ssoConfig.credentials.tokenEndpointUrl;

var issuer_id = ssoConfig.credentials.issuerIdentifier;

var callback_url = 'https://ssolab-20150601tor.mybluemix.net/auth/sso/callback';


var OpenIDConnectStrategy = require('passport-idaas-openidconnect').IDaaSOIDCStrategy;

var Strategy = new OpenIDConnectStrategy({

                authorizationURL : authorization_url,

                tokenURL : token_url,

                clientID : client_id,

                scope: 'openid',

                response_type: 'code',

                clientSecret : client_secret,

                callbackURL : callback_url,

                skipUserProfile: true,

                issuer: issuer_id},

        function(accessToken, refreshToken, profile, done) {

                        process.nextTick(function() {

                profile.accessToken = accessToken;

                profile.refreshToken = refreshToken;

                done(null, profile);

                })
```

```
});


passport.use(Strategy);

app.get('/login', passport.authenticate('openidconnect', {}));


function ensureAuthenticated(req, res, next) {

       if(!req.isAuthenticated()) {

                     req.session.originalUrl = req.originalUrl;

              res.redirect('/login');

       } else {

              return next();

       }

}


app.get('/auth/sso/callback',function(req,res,next) {

           var redirect_url = req.session.originalUrl;

            passport.authenticate('openidconnect',{

                 successRedirect: redirect_url,

                 failureRedirect:'/failure',

          })(req,res,next);

        });


app.get('/hello', ensureAuthenticated, function(req, res) {
        res.send('Hello, '+ req.user['id'] + '!'); });


app.get('/failure', function(req, res) {

        res.send('login failed'); });


// serve the files out of ./public as our main files

app.use(express.static(__dirname + '/public'));


// get the app environment from Cloud Foundry

var appEnv = cfenv.getAppEnv();


// start server on the specified port and binding host

app.listen(appEnv.port, appEnv.bind, function() {


       // print a message when the server starts listening

  console.log("server starting on " + appEnv.url);

});
```