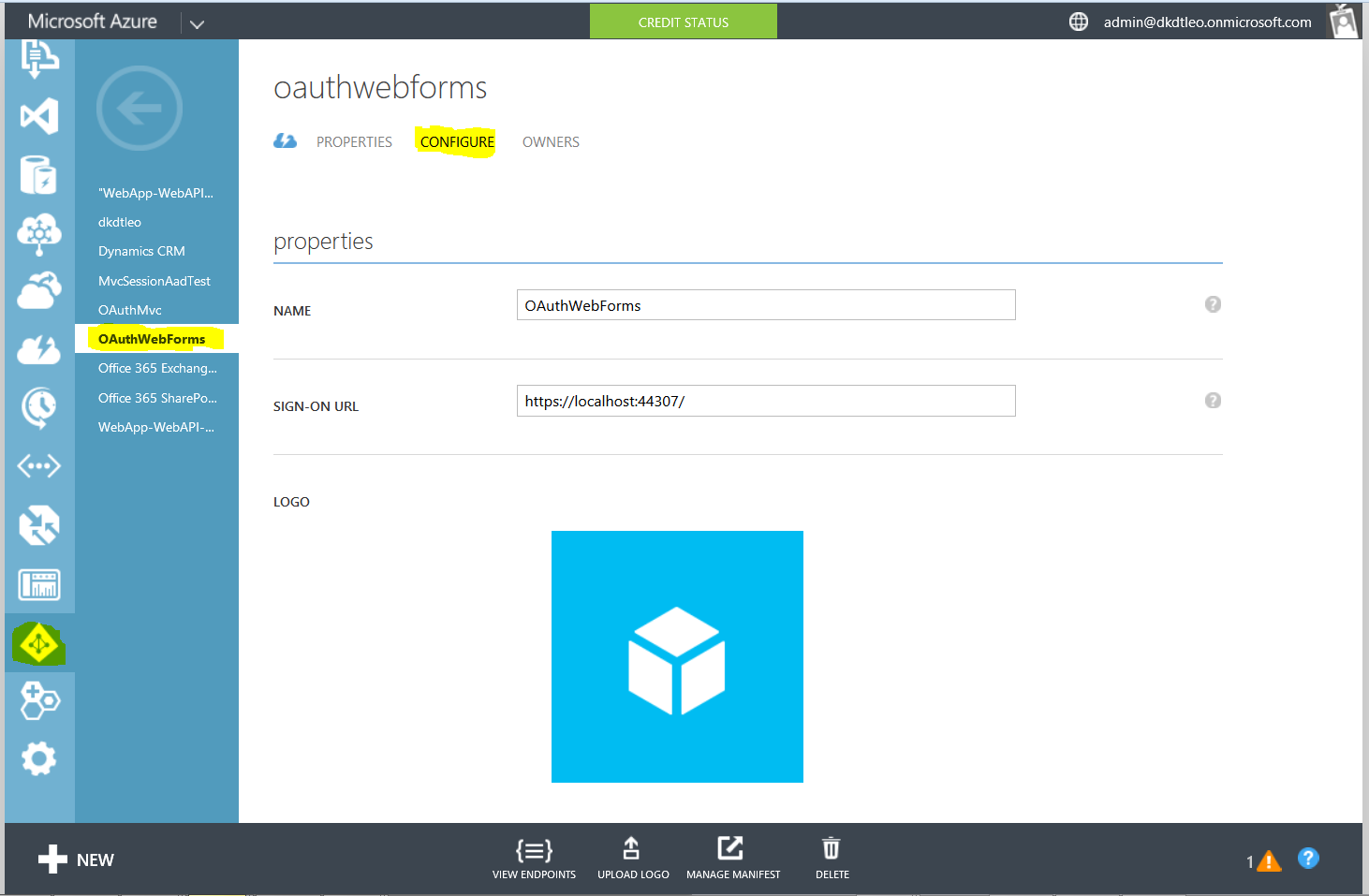
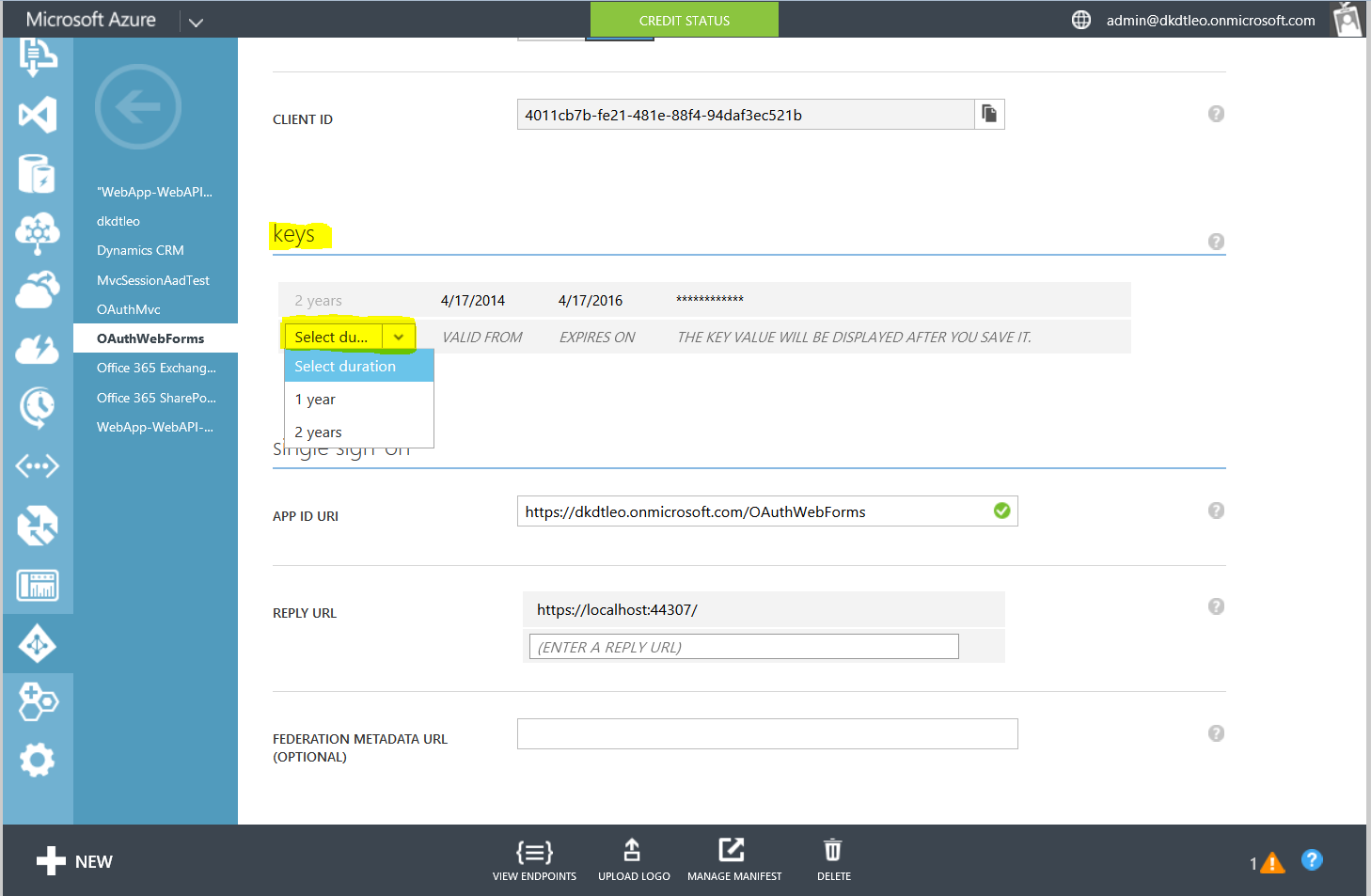
1. Open the starter solution.
2. Create a new MVC or Web Forms project and configure it to use Azure Active Directory. See the following video to get started:

<http://azure.microsoft.com/en-us/documentation/videos/azure-identity-application-to-authenticate/>

1. Navigate to your application’s Azure Active Directory configuration settings in the Azure Portal:



1. Create a key in the keys section of the configuration settings:



1. Make sure you save the key value. Once you leave the configuration settings page, the key can no longer be seen.
2. Add a project reference to the Devkeydet.OAuthHelper project.
3. In the web app project, add the following XML snippet to the <appSettings/> section of the web.config:

<add key="ida:ClientId" value="[CLIENT ID from the Azure portal here]" />

<add key="ida:AppKey" value="="[KEY from the Azure portal here]=" />

<add key="ida:AADInstance" value="https://login.windows.net/{0}" />

<add key="ida:ResourceId" value="https://[YOURORG].crm.dynamics.com" />

<add key="ida:OAuthRedirect" value="[url for your page/controller to handle OAuth redirect" />

1. Next, you will need to create a controller (if MVC) or a page (if Web Forms) to point Azure Active Directory to in order to process the returned access code and request a token that allows you to make requests to CRM web services using the logged in users identity. The **ida:OAuthRedirect** value should be in the form of “/routetoyourcontroller” or “/urlofyourpage”
2. Create the page or controller. You can call it what you want, as long as you update the config file appropriately, but I prefer to call it OAuthRedirectController or OAuthRedirect.aspx.
3. If you are using MVC, replace the code for the Index() action method with the following:

public ActionResult Index(string code, string error, string error\_description, string resource, string state)

        {

            var returnUrl = OAuthHelper.ProcessAccessTokenAndGetReturnUrl(code, error, error\_description, state);

            return Redirect(returnUrl);

        }

1. Fix using statements
2. If you are using Web Forms, replace the contents of the Page\_Load event handler with the following code:

string code = Request.QueryString["code"];

            string error = Request.QueryString["error"];

            string error\_description = Request.QueryString["error\_description"];

            string state = Request.QueryString["state"];

            var returnUrl = OAuthHelper.ProcessAccessTokenAndGetReturnUrl(code, error, error\_description, state);

            Response.Redirect(returnUrl);

1. Now go to the controller or page where you want to actually make web service calls that use the token.
2. For MVC, use the following code:

var accessToken = OAuthHelper.GetAccessTokenFromCacheOrRefreshToken();

            // If we don't have the access or refresh token cached, then we need to redirect to AAD to get a token THIS APP can use

            if (accessToken == null)

            {

                return Redirect(OAuthHelper.GetAuthorizationUrl());

            }

// Use token to make web service calls

1. For Web Forms, use the following code:

var accessToken = OAuthHelper.GetAccessTokenFromCacheOrRefreshToken();

            // If we don't have the access or refresh token cached, then we need to redirect to AAD to get a token THIS APP can use

            if (accessToken == null)

            {

                Response.Redirect(OAuthHelper.GetAuthorizationUrl());

            }

// Use token to make web service calls

1. The sample app has two examples of making OData calls. The first is just raw http requests. The second uses a LINQ to OData by way of a code generated OData context. To create one of these contexts, see: <http://blogs.msdn.com/b/devkeydet/archive/2012/06/10/using-the-crm-2011-odata-service-from-a-metro-style-app.aspx>. Start with the text **Ok, now it’s time to “Add Service Reference” to the OData service**. You can ignore the rest of the blog post.