Working with Salesforce Outbound Messages in .NET

What are Outbound Messages?

Outbound Messages in Salesforce are another way for your application to receive notifications of changes to your Salesforce data. Imagine you have a scenario, where you would like your external application to be notified whenever a *Lead* record in Salesforce is created (or edited) and the *City* field on the Lead is set to Chicago. It is possible in Salesforce, using an Outbound Message which is triggered from a Workflow Rule like the one I just described. I'll discuss in detail what both of those Salesforce features are in this post.

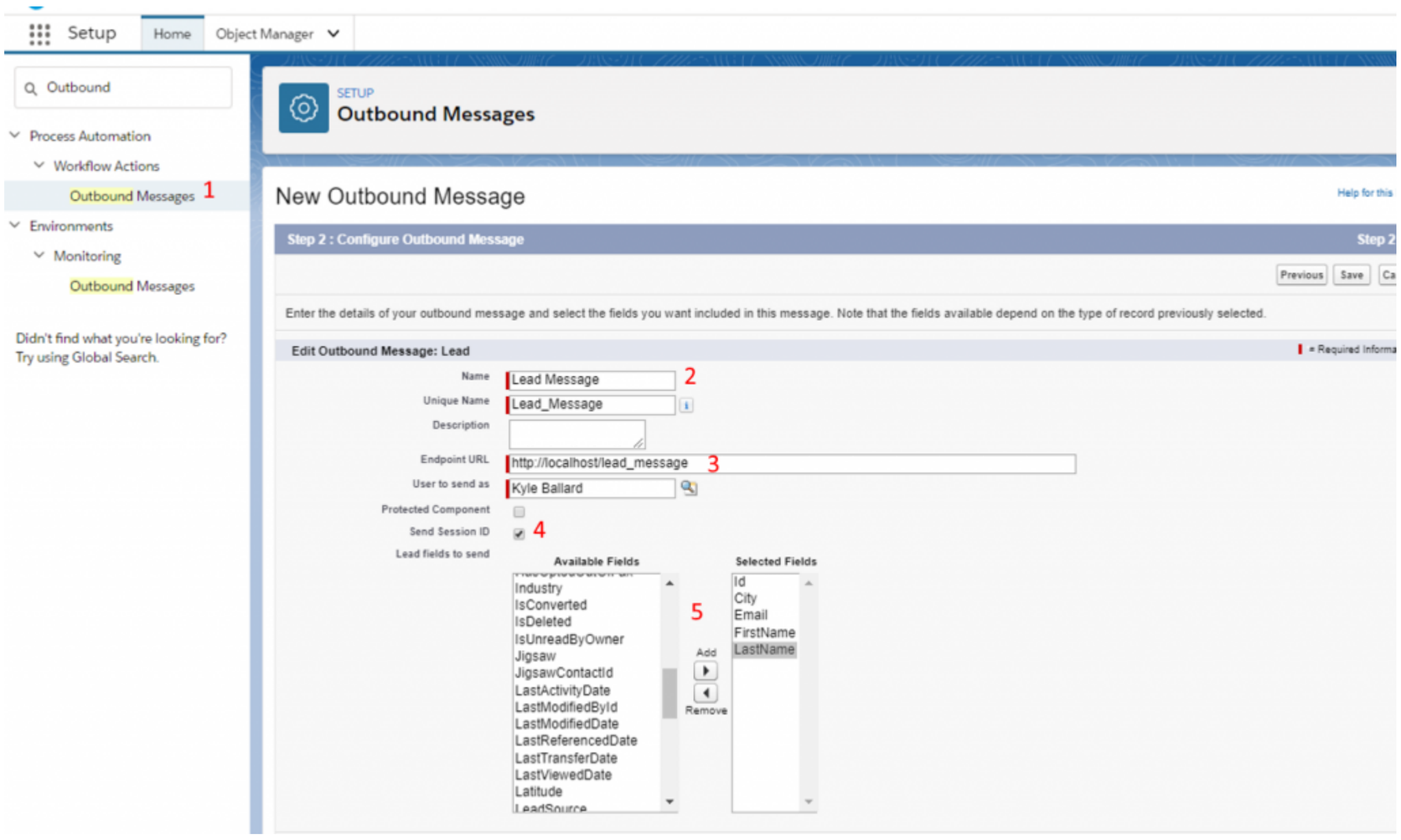
We have already discussed the Streaming API here in the past, but there are a few key differences to observe with Outbound Messages:

- There are no API limitations around the number of Outbound Messages which can be sent. The Streaming API has a set number of events which are allocated to you.
- Outbound Messages require you to expose a public URL which receives the notifications, and there are some security considerations to be aware of. The Streaming API does not require a public facing URL and can run behind a firewall.
- Workflow Rules allow a finer set of criteria based on field filters, or a formula field, which can give you more control over which notifications your application receives.
- There is no 'Replay' behavior in Outbound Messages and there is no guarantee that notifications will come in the order they occurred. You will receive an Id for each notification, but there is no way to replay notifications. Similarly, if a record is changed before your message is sent, it may only contain the latest data from the most recent change.

Important note: You should be cognizant of creating circular/infinite loops with Outbound Messages. If you create a Workflow Rule that says to create a message every time an account is changed, and that updates the opportunity, which fires a rule that updates the account, you may find yourself creating a difficult to troubleshoot scenario. To circumvent this problem, create a *Profile* in Salesforce which corresponds to a user which can receive outbound notifications, but has the "Send Outbound Messages" field set to *off* on the profile. This way the user can receive messages but updating changes will not trigger additional cascading notifications.

Defining an Outbound Message

Creating our Outbound Message in Salesforce will be a two step process. The first step will be to define the actual message. Think of this as defining the data model for the process. Later, when we discuss the Workflow Rule, this will be the controller aspect that determines when our messages are sent. Head over to your Setup menu inside your sandbox/developer org and use the Quick Find behavior to locate "Process Automation > Workflow Actions > Outbound Messages" (1). From the list of current outbound messages, select "New Outbound Message". On the first screen, indicate we are creating a message for the *Lead* record type.

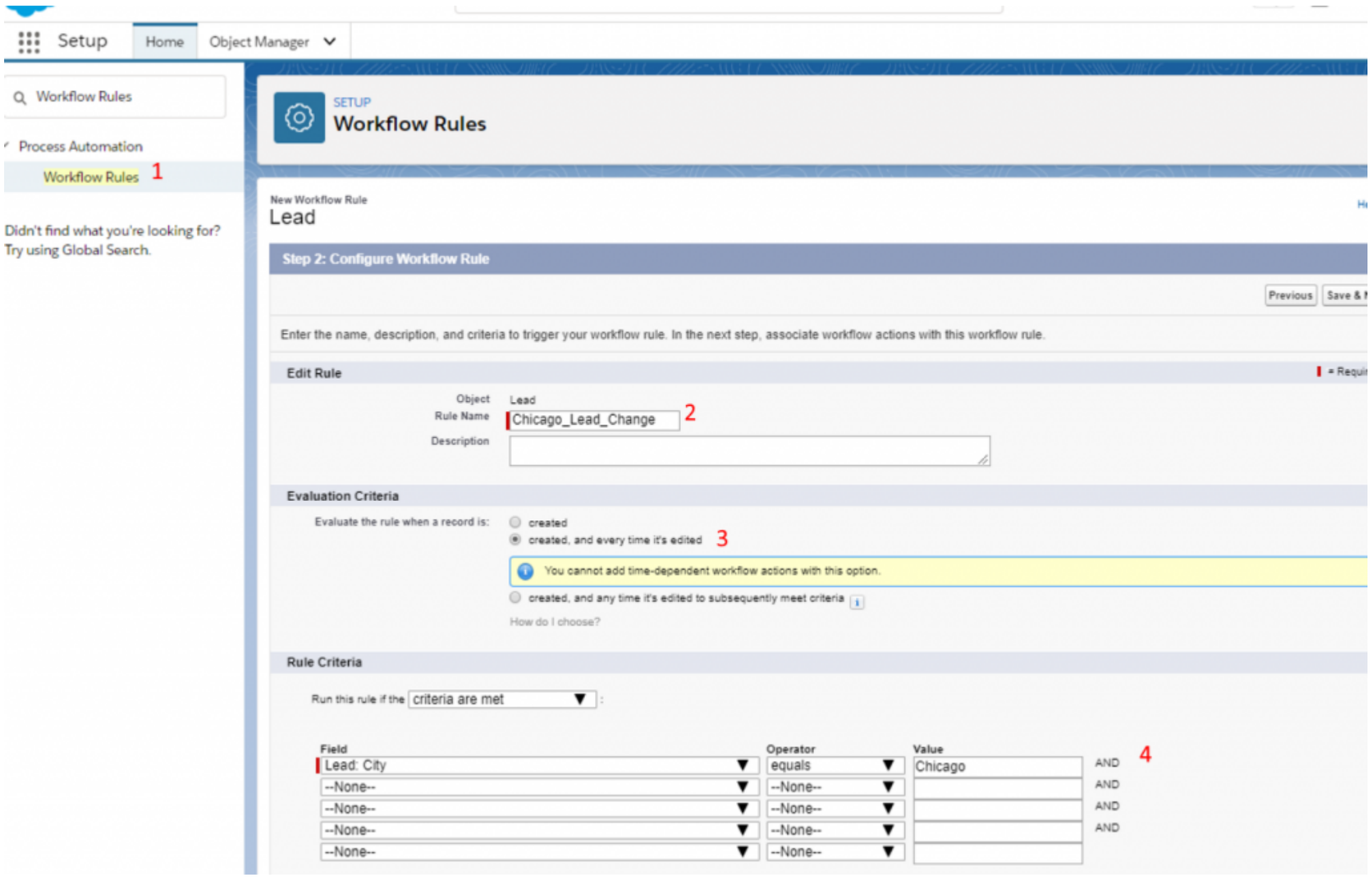


On the next screen, specify a name (2) for our message that is representative of what is being delivered in each notification. Also, provide a generic Endpoint URL (3). We are going to change this later, so just set this to a valid URL for now. Also, enable the Session ID (4) so we get a parameter which indicates who triggered the message being sent. Also, specify the fields you'd like to receive in your payload (5) when the notification is delivered. As a suggestion, try to limit this to just the fields you need. This will reduce the transmission time over the wire and also reduce the time to deserialize the messages as well.

Once the message has been saved/created, the final step we need to perform is to capture the Endpoint WSDL value. Click the link that says "Click for WSDL" and save the contents that results on your workstation with the name "leads.wsdl".

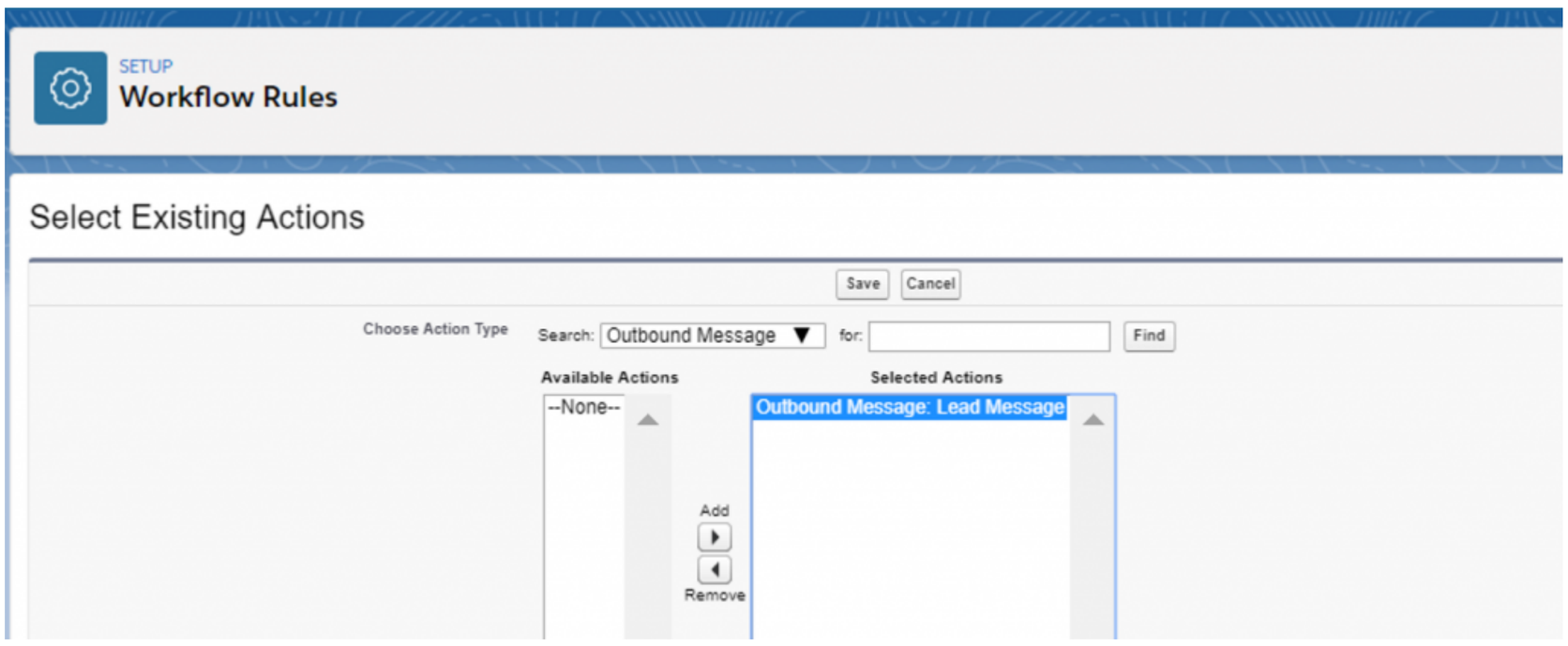
Creating the Workflow Rule

The next step in our two step process, is to create a Workflow Rule which will trigger our Outbound Message to actually be delivered. In setup, enter "Workflow Rule" in the Quick Find and select "Process Automation > Workflow Rules". On the screen that lists your current workflow rules, select the "New Rule" button and designate that we are creating a rule that works on the *Lead* object.



On the next screen, provide an informative name for your rule. In your evaluation criteria you have three options. When the records are created and meet criteria, when they are created or edited and meet criteria, and the final option to evaluate the record only when the record changes from it's original state to meet the criteria (i.e. the city was Phoenix, but it is now Chicago and criteria is city = Chicago). We'll select the second option (3) which runs every time our criteria is met, even if that criteria didn't change.

For the rule criteria, we're given a set of contextual (based on the object type we're working with) parameters we can use to determine if our rule should be executed. We could switch this to a formula also, but that is outside the scope of our tutorial. For now, let's stick to Lead: City is equal to Chicago (4). On the next screen, select the "Add Workflow Action > Select Existing Action" option just below the block for Immediate Workflow Actions. From here, indicate the action type is "Outbound Message" and pick the message we created in the Outbound Message creation step above. Click "Save" and we are done with the Workflow Rule.



Create the Listener Service

We've reached the development portion of the tutorial. Because .NET Core does not support the server/hosting side of SOAP services, we're going to use a .NET 4.6.1 project to complete our integration. The first step is to ensure your wsdl.exe command is in the path of your workstation. If the command "*wsdl.exe -?*" does not produce a valid response, you will need to track this down.

Once we have everything ready, let's run the command below. The leads.wsdl should have been captured from the first step where we setup the outbound message: *wsdl.exe /serverInterface "c:\path\to\your\leads.wsdl" /out:NotificationServiceInterface.cs*

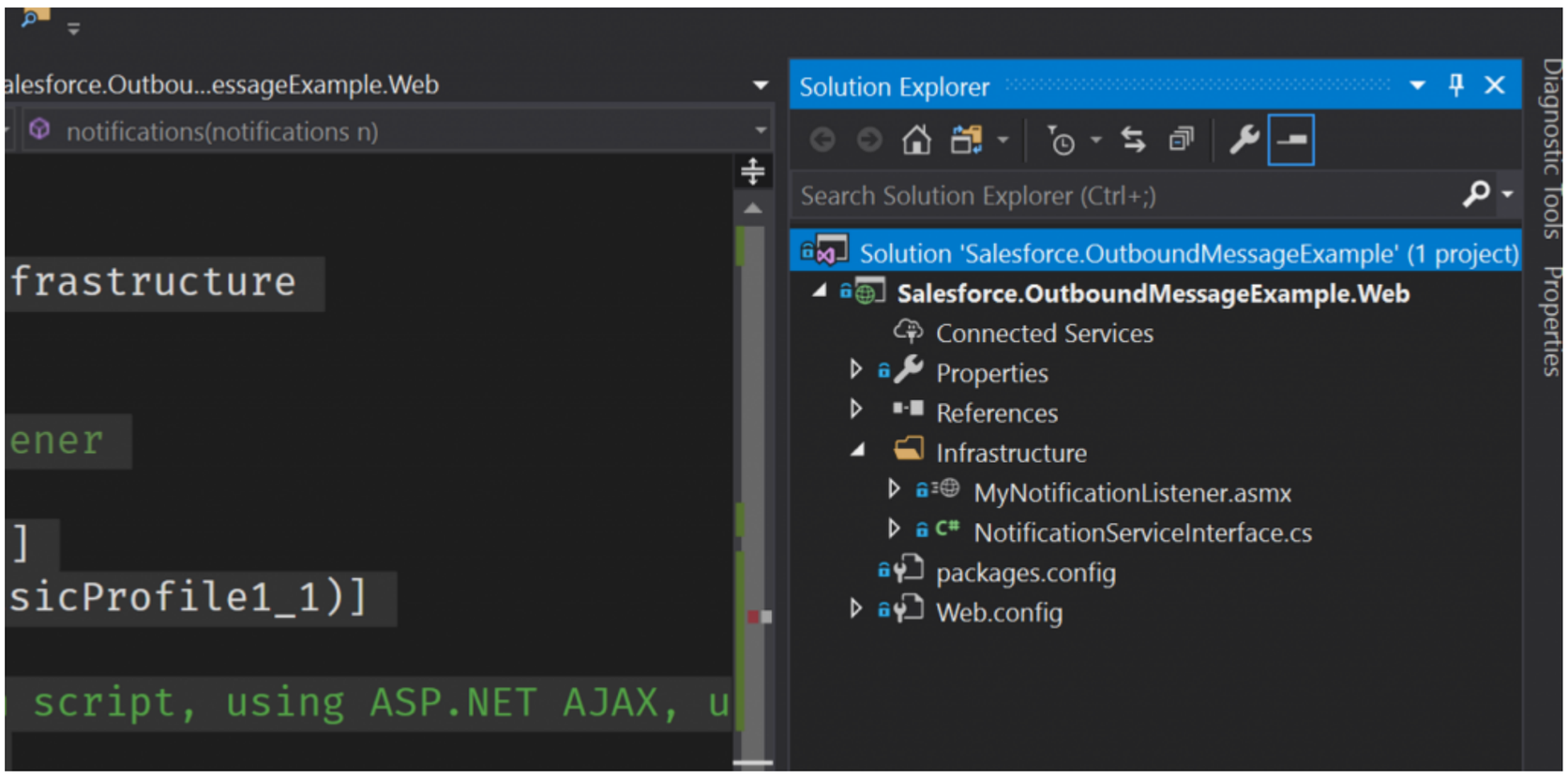
This command will generate the server/hosting side contracts (/serverInterface) for the WSDL service since Salesforce is sending the message to us. Make a note of where your NotificationServiceInterface.cs file is located because we'll be adding this to our project in a moment. The next step is to create a new "ASP.NET Web Application (.NET Framework)" to indicate we want the full version of .NET. You can create this with the "Empty" template type since we aren't doing anything else here other than just prototyping something.

Once this is done, add a new folder called Infrastructure, and add your previously generated NotificationServiceInterface.cs file to it. You may need/want to update the namespace for that file to ensure it matches your solution. The last step, will be to "Add New Item" to the Infrastructure folder we created, and specify the type as "Web Service (ASMX)" and name our file *MyNotificationListener.asmx*. The contents of that file should be as follows:

```
using System.Web.Services;

namespace Salesforce.OutboundMessageExample.Web.Infrastructure
{
    /// <summary>
    /// Summary description for MyNotificationListener
    /// </summary>
    [WebService(Namespace = "http://tempuri.org/")]
    [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
    [System.ComponentModel.ToolboxItem(false)]
    // To allow this Web Service to be called from script, using ASP.NET AJAX, uncomment the
following line.
    // [System.Web.Script.Services.ScriptService]
```

```
    public class MyNotificationListener : INotificationBinding
    {
        public notificationsResponse notifications(notifications n)
        {
            notificationsResponse r = new notificationsResponse();
            r.Ack = true;
            return r;
        }
    }
}
```



Your solution should look like this when completed.

Important Security Considerations

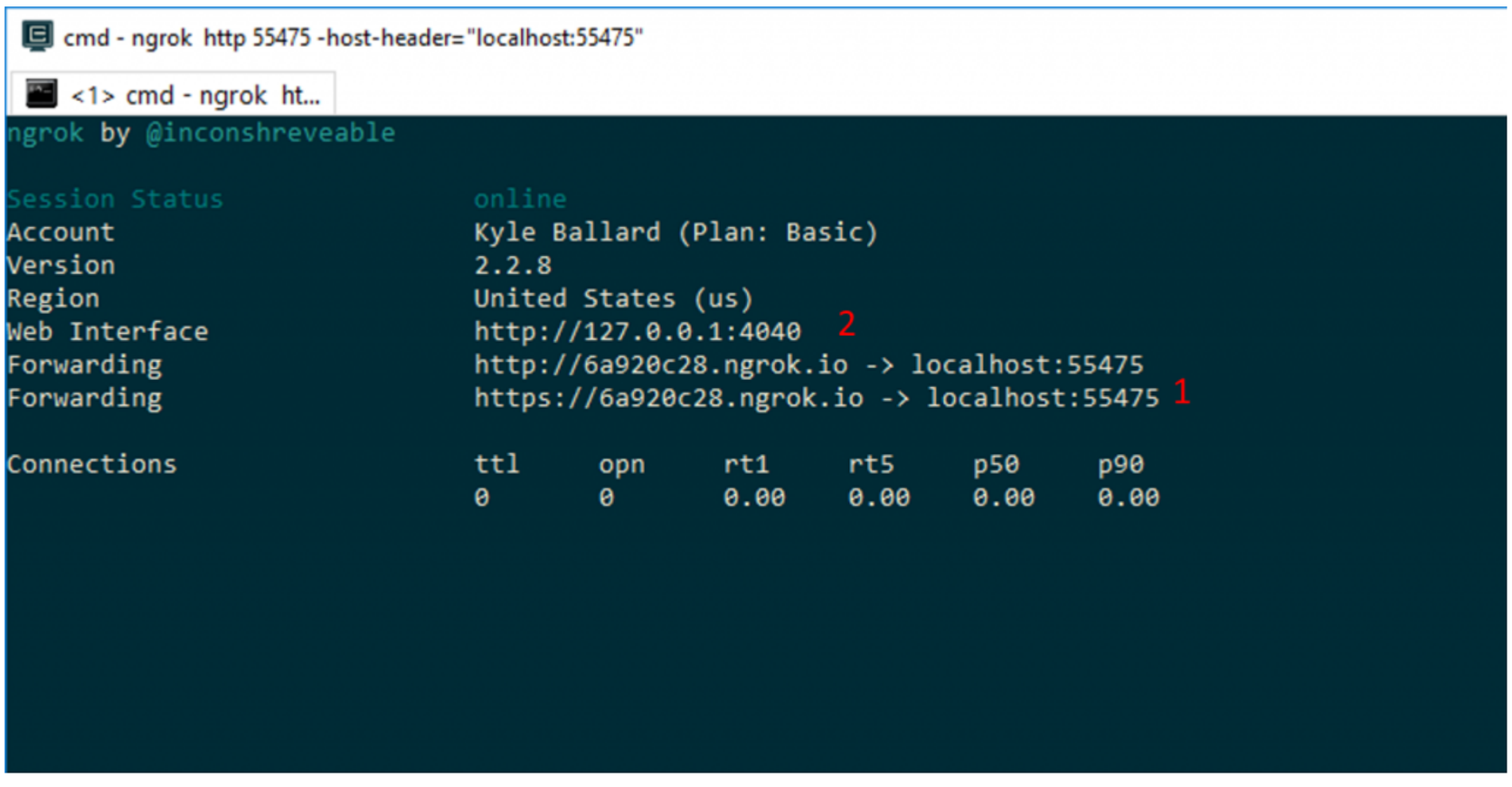
There are some important security changes you should apply when running this application in production, and this is a good point to discuss some of them:

- Ensure that your service is white-listed to only allow Salesforce IP addresses.
- Use TLS/SSL endpoints for your services to ensure there are no parties which could intercept your sensitive Salesforce data.
- An OrganizationId parameter is included in each message. Validate this on each incoming message to a locally stored parameter for your organization.
- If your application allows (or requires) it, download the Salesforce Client Certificate to validate the message is indeed coming from Salesforce.

Testing Locally with Ngrok

Because Salesforce needs to send a message to your application, and it cannot read localhost as a publicly accessible endpoint, we need to provide Salesforce with an address it can use to post messages to us. This is where Ngrok comes in. Ngrok allows you to create a public URL which will tunnel to your local instance of your application. In the first step, when creating the Outbound Message, we just provided a dummy URL. We're going to update that now to a valid one.

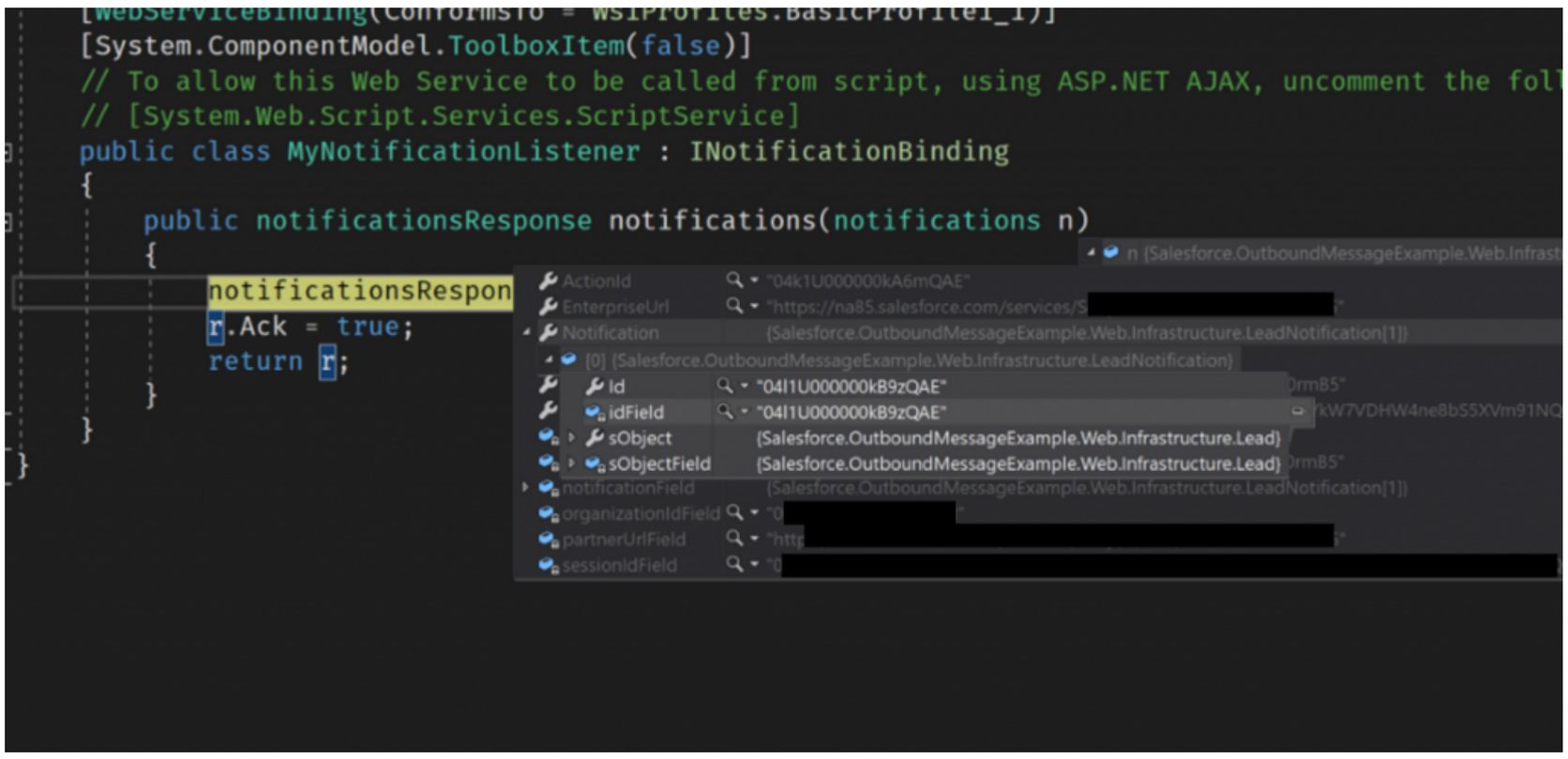
Start the debug for your application, and make a mental note of the port. On my machine the address is:*http://localhost:**55475**/Infrastructure/MyNotificationListener.asmx*. Armed with this information, and a downloaded copy of ngrok which is in my system %path% variable, I will run the command: *ngrok http 55475 -host-header="localhost:55475"* . You can replace 55475 with the port for your project.



You can then see we're given an http/https URL to use (1) and also a web proxy we can use to monitor requests/responses (2). If I were to request *https://6a920c28.ngrok.io/Infrastructure/MyNotificationListener.asmx* the result should look the same as if I had requested the page on localhost:55475. Armed with this information, head back over to the Outbound Message you created in the first step, and update the *Endpoint URL* parameter with this new public URL you have just created.

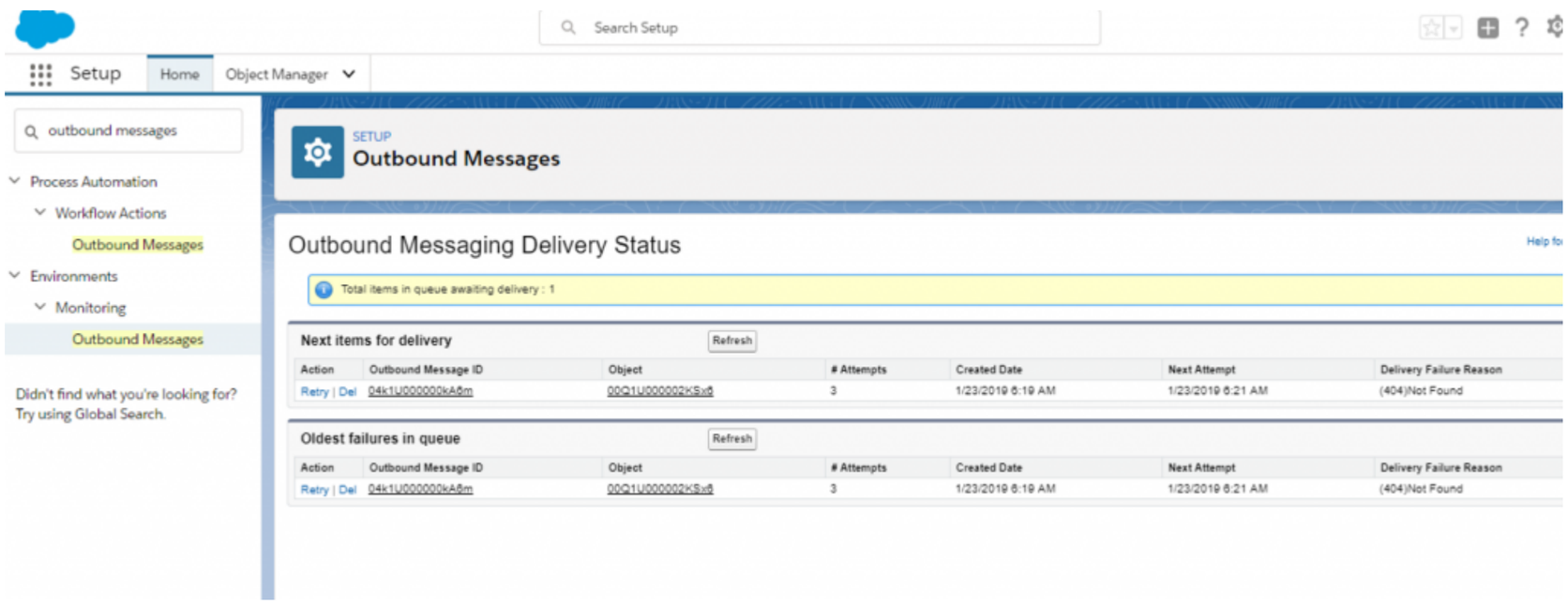
Firing the Workflow Rule

Ok, we're pretty much all set at this point. The last item we need to do is head to the Workflow Rule we created in the second step and click "Activate" on the rule. If we don't *activate* the Workflow Rule, we won't get any messages delivered to our endpoint. Ok! Are you ready? Now comes the exciting part. Go head over to the "Leads" section of Salesforce and edit one of the leads which already exist there (or create a new one). Set any values you like. The only important thing to consider is to ensure that the "City" field is set to "Chicago" per our Workflow Rule. One you save the record, your breakpoint should be hit and you can debug the incoming object to see all of the information you are being provided about the record. Boom!



Viewing Outbound Message Status

Having any trouble? Head over to to the Setup location "Environments > Monitoring > Outbound Messages". Here you will see a snapshot of messages which are queued for delivery as well as any recent failures. A helpful description will be provided as well. In my case, you can see a 404 not found was provided because I didn't have ngrok running.



Summary

Outbound Messaging is a great way to receive notifications of changes in your Salesforce platform based around specific criteria. You can be as specific or generic as you'd like. Salesforce will deliver up to 100 notifications at a time to your application. It will also try to re-transmit those messages for up to 24 hours, but will gradually increase the gap duration between retries for each failure up to 2 hours between attempts. I hope you enjoy this feature and are able to utilize it to keep your applications in sync with Salesforce.