Lab Guide

CCW3970

Building and debugging outbound REST API Integrations

Bryan Barnard

Silas Smith

Azfar Kazmi

Lab Instance Details

Lab instance: http://clabs.link/*ccw3970*

Default Login / Password:

admin / Knowledge17

itil / Knowledge17

employee / Knowledge17

This

Page

Intentionally

Left

Blank

Lab Goal

Lab Setup

Before we get started building and debugging outbound REST API integrations we need to get our lab instance setup. In this lab you will be modifying an existing scoped application. Start out by importing the **CCW3970** application from Source Control. Follow the directions below to fork this application to your GitHub account and begin working.

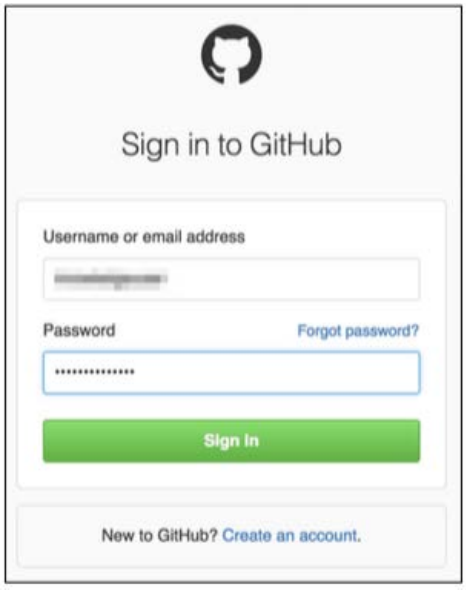
Prerequisites

In order to complete this lab, you must:

1. Create a GitHub account if you do not already have one.
2. Install Postman from https://getpostman.com if you do not already have it installed.

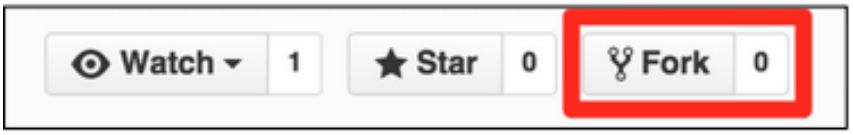
Fork the Lab GitHub Repository

1. Log in to your GitHub account at:
   https://github.com/login



2. Navigate to:
   https://github.com/CreatorCon17/CCW3970-Build-Debug-Outbound-REST-App

3. Click **Fork.** To fork the repository.



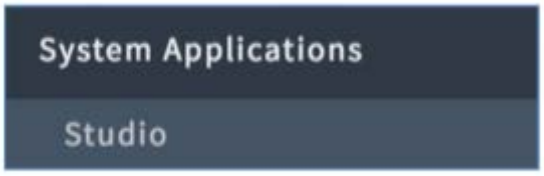
4. Note in the upper left that the repository has been copied to your account. You now have a copy of the lab material for reference after the conference!

5. Locate the HTTPS field and click the clipboard to the right. This action copies the URL in the clipboard.

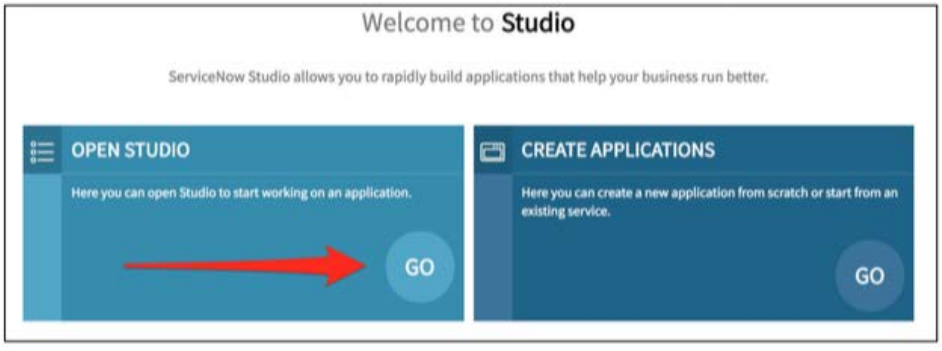
IMPORTANT: Be sure to copy the **HTTPS** repo URL in GitHub.

Import the CCW3970 Application from Source Control

6. Log in to your lab instance with the **admin** credentials provided on the cover sheet of this document.
7. Navigate to System **Applications > Studio.**

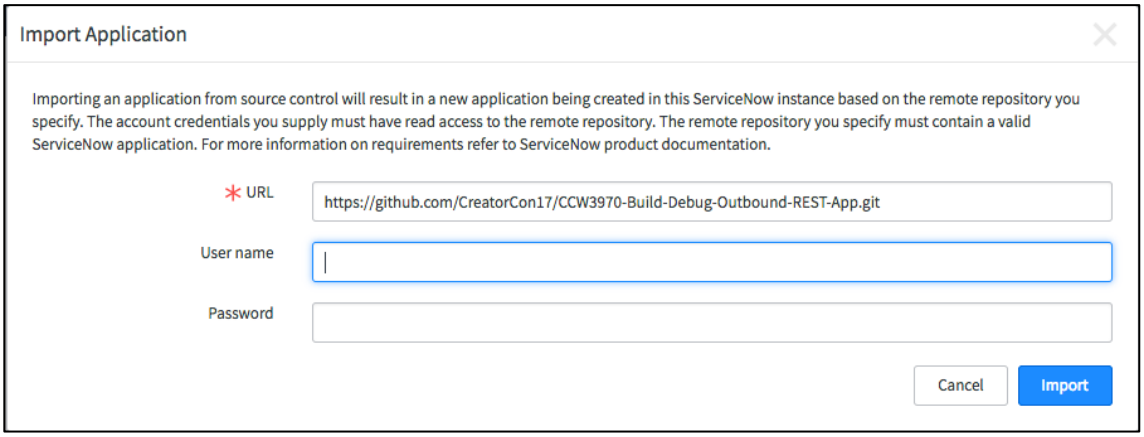


8. Click **Go** in the **Open Studio** section on the left.



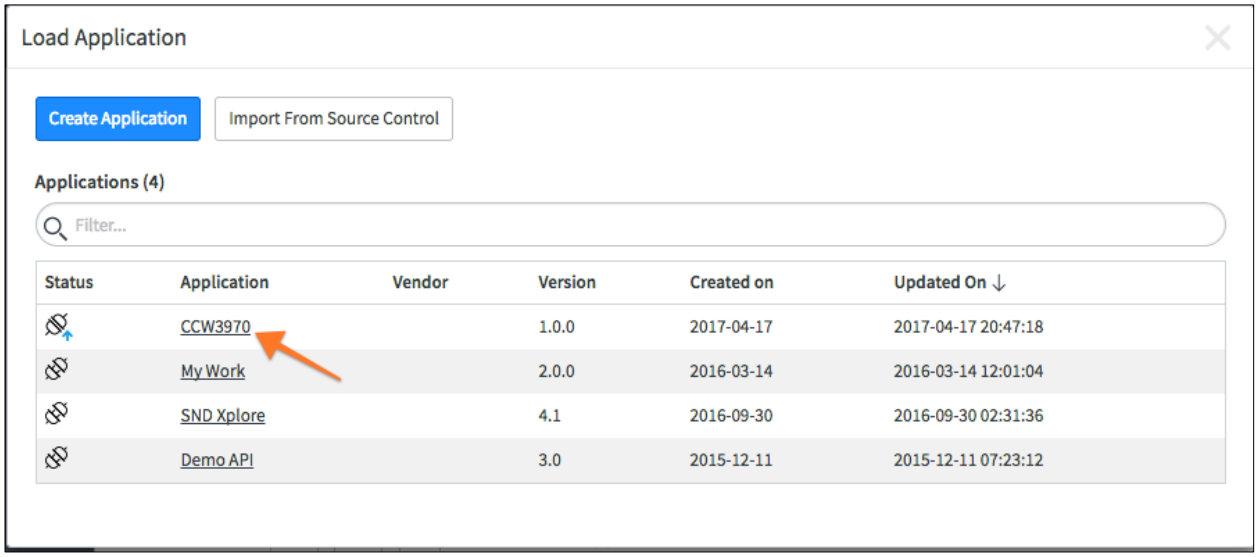
9. Click **Import from Source Control.**

10. In the Import Application window, paste the URL copied in step 5 and provide your GitHub credentials. Click **Import.**



11. When the import completes, click **Select Application.**

12. Click on the **CCW3970** application you just imported to load this application into Studio.



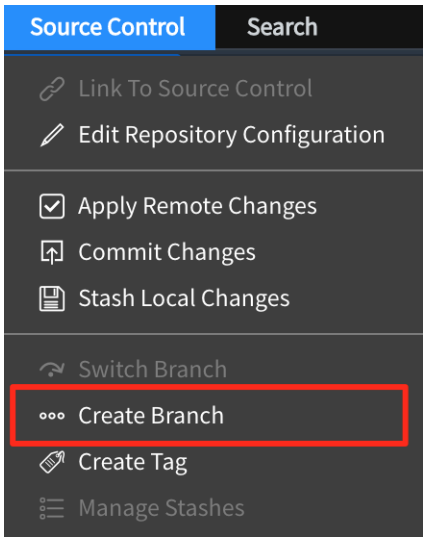
You've now successfully imported your forked version of the application for use in this workshop.

Get ready for Lab 2 – Create a new branch from Lab2-start tag in Studio

1. **Yes**, you read that correctly, we won't be using ServiceNow again until Lab2, but we want to get you ready ahead of time.

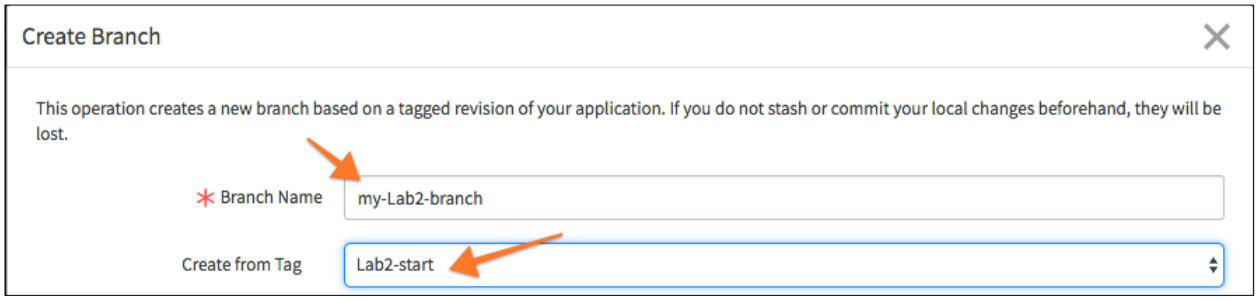
   NOTE: This is worth mentioning, not a typo, you are importing and opening this application in Studio but we will not be using ServiceNow again until you start **Lab 2**.

In Studio, navigate to Source Control > Create Branch.

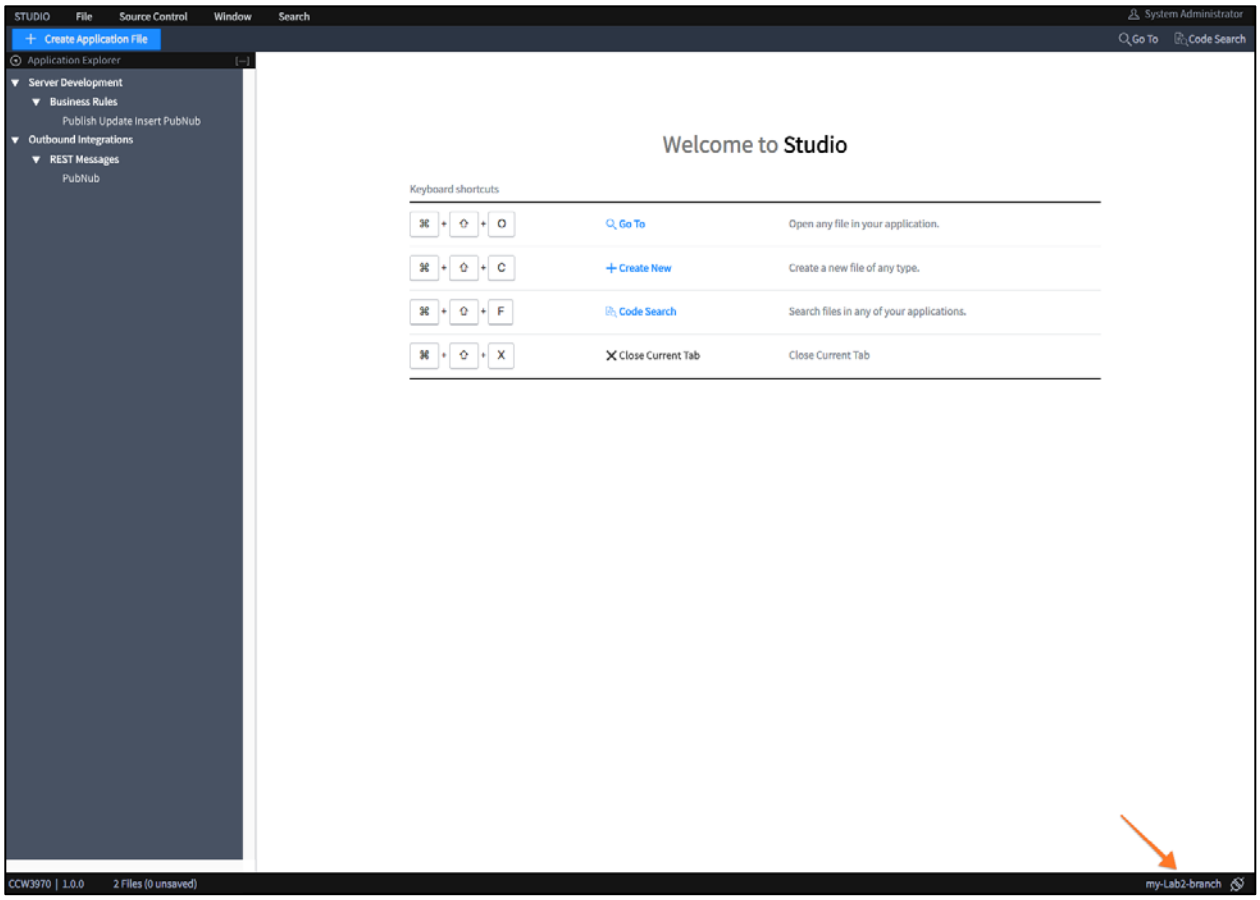


2. In the pop-up window, enter a branch name, then select **Lab2-start** from the Create from Tag menu, and click Create Branch.

   Branch: **my-Lab2-branch**
   Create from Tag: **Lab2-start**



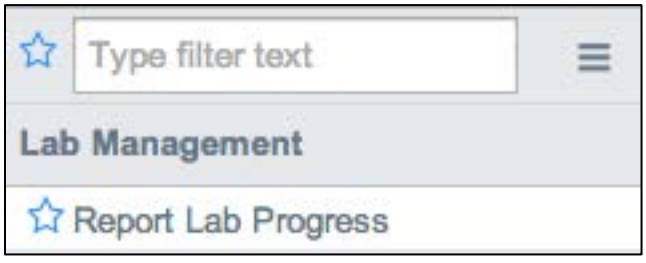
3. When the switch is complete, click **Close Dialog** in the Create Branch pop-up.
4. Verify Studio is on branch **my-Lab2-branch** (bottom right hand corner).



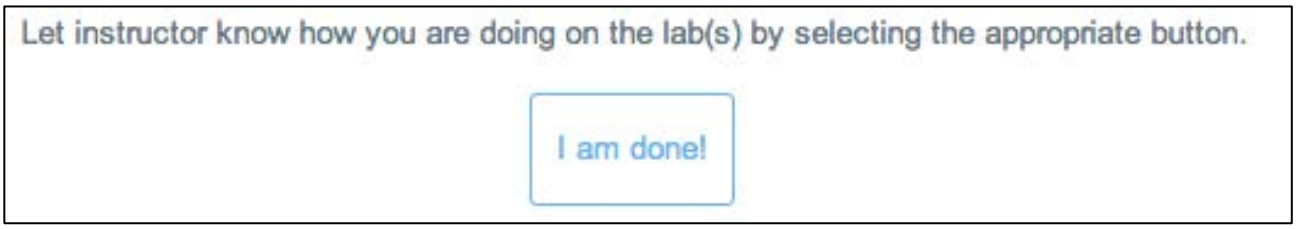
5. Lab setup is complete. You are now ready to start **Lab 1.**

Progress Report

1. Navigate to **Lab Management> Report Lab Progress**.



2. Click **I am done!**



Lab Goal

Lab 1
Publish Message with Postman "Hello

The purpose of this workshop is to familiarize yourself with ServiceNow Outbound Messaging capabilities available to you for building integrations with 3rd party REST APIs as well as how you can debug your integrations.

In this first **lab** you'll familiarize yourself with the **PubNub** 3rd party REST API we'll be working with for the rest of this workshop and use **Postman** (an API testing tool) to build requests you can execute and review from your localhost.

Prerequisites

- Knowledge of REST APIs
- Knowledge of HTTP clients
- Postman API testing tool Installed. If you still need to install the Postman go to: https://www.getpostman.com/

Check out PubNub

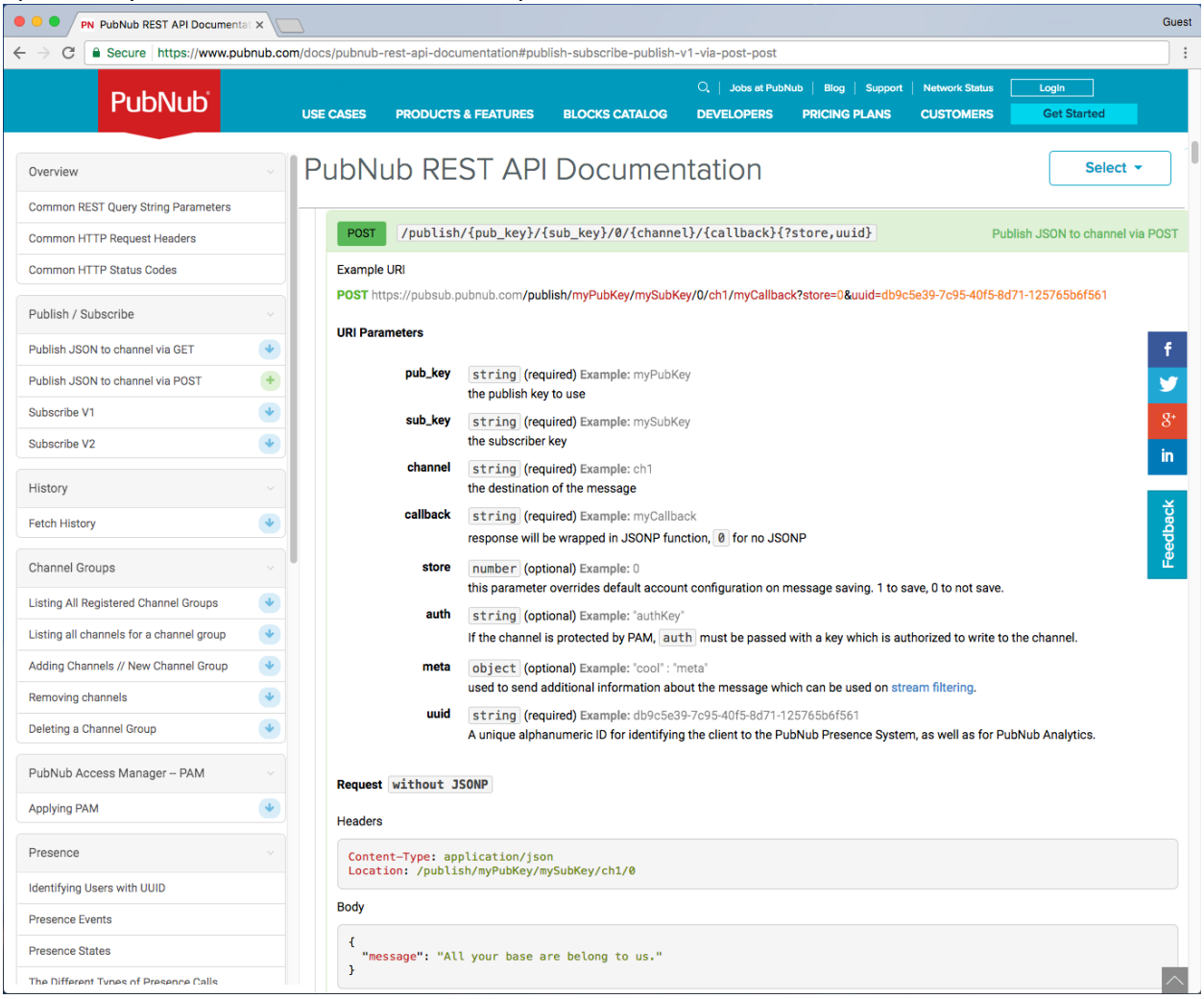
When building an integration between cloud based service providers it's a good idea to start out by mocking up your requests with a tool that you can run on your localhost such as cURL, Postman or Paw. Each of these tools allow you to build and execute HTTP requests from your localhost (laptop, desktop, etc.…). This enables you to build and execute your requests in a very agile way and also provides you with a working example to reference when you start building your integration in ServiceNow.

In this lab we'll be working with PubNub, PubNub is a 3rd paryt streaming data service. You'll start getting familiar with the API by using Postman to make a request to the Publish Message operation of the PubNub REST API. This operation allows you to publish messages via HTTP.

PubNub offers a rich set of functionality but for the purposes of this workshop we'll only be using their Publish Message capabilities. If you are interested in finding out more about PubNub and PubSub services you should check out their website and API docs.

Start out by briefly review the API for PubNub at:
https://www.pubnub.com/docs/pubnub-rest-api-documentation - publish-subscribe-publish-v1-via-post-post

Specifically look at the Publish via POST operation.



You'll be using the Postman to make a request to the PubNub REST API and publish a message to a channel. The PubNub REST API provides an endpoint that accepts a POST request to publish a message onto a channel that other clients can subscribe to. Per the documentation this method requires the following parameters **pub_key**, **sub_key**, and **channel** be specified as URL path parameters and a **uuid** be provided as a query parameter.

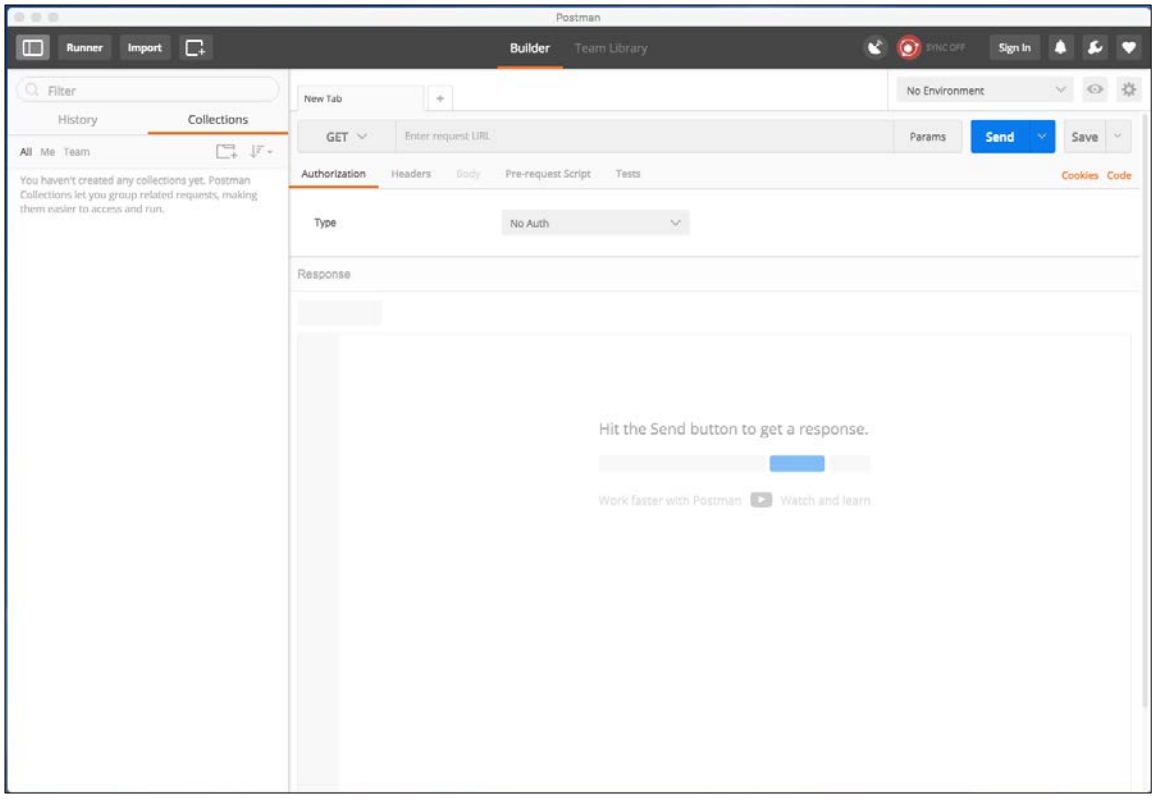
Example URI

POST https://pubsub.pubnub.com/publish/{pub_key}/{sub_key}/0/{channel}/0?store=1&uuid={client}

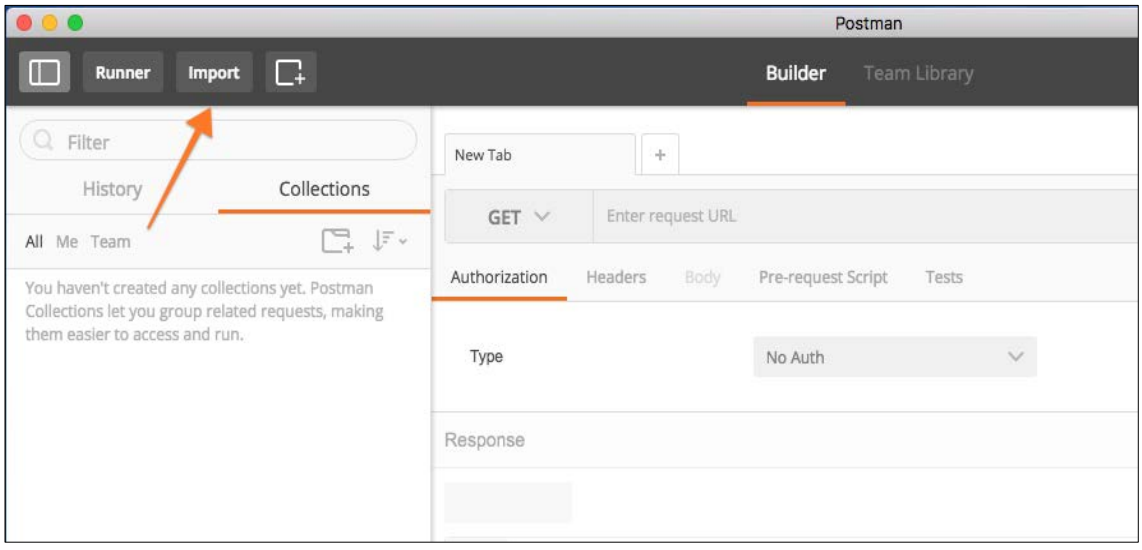
Build and Send a Request to Publish with Postman

Let's begin by building a request to publish a message to PubNub in Postman. We've built and made a prebuilt Postman collection to help you get started.

1. Open the **Postman** application on your laptop.

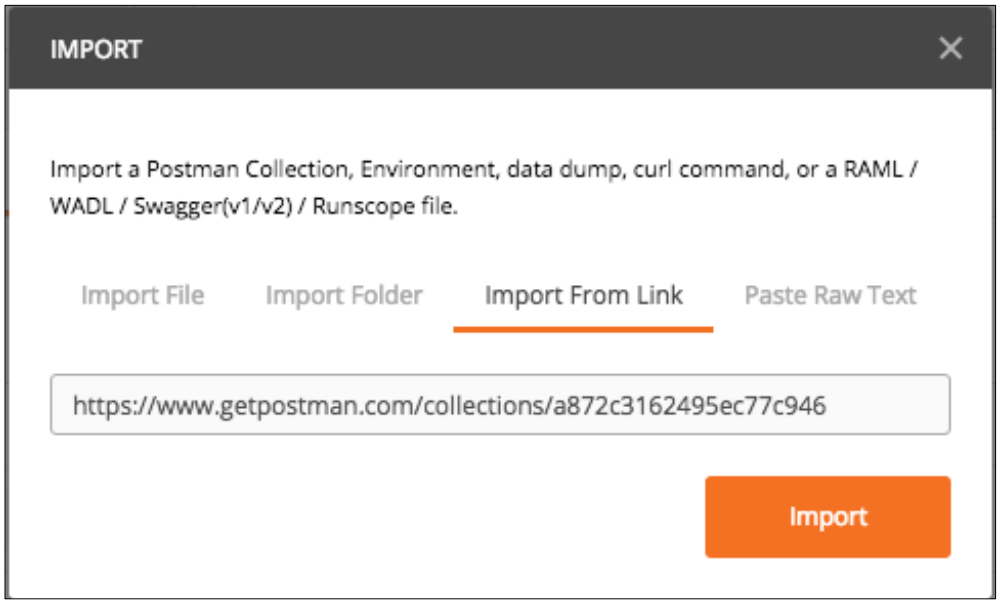


2. Import the Postman collection we will be using for this workshop. In Postman, click **Import.**

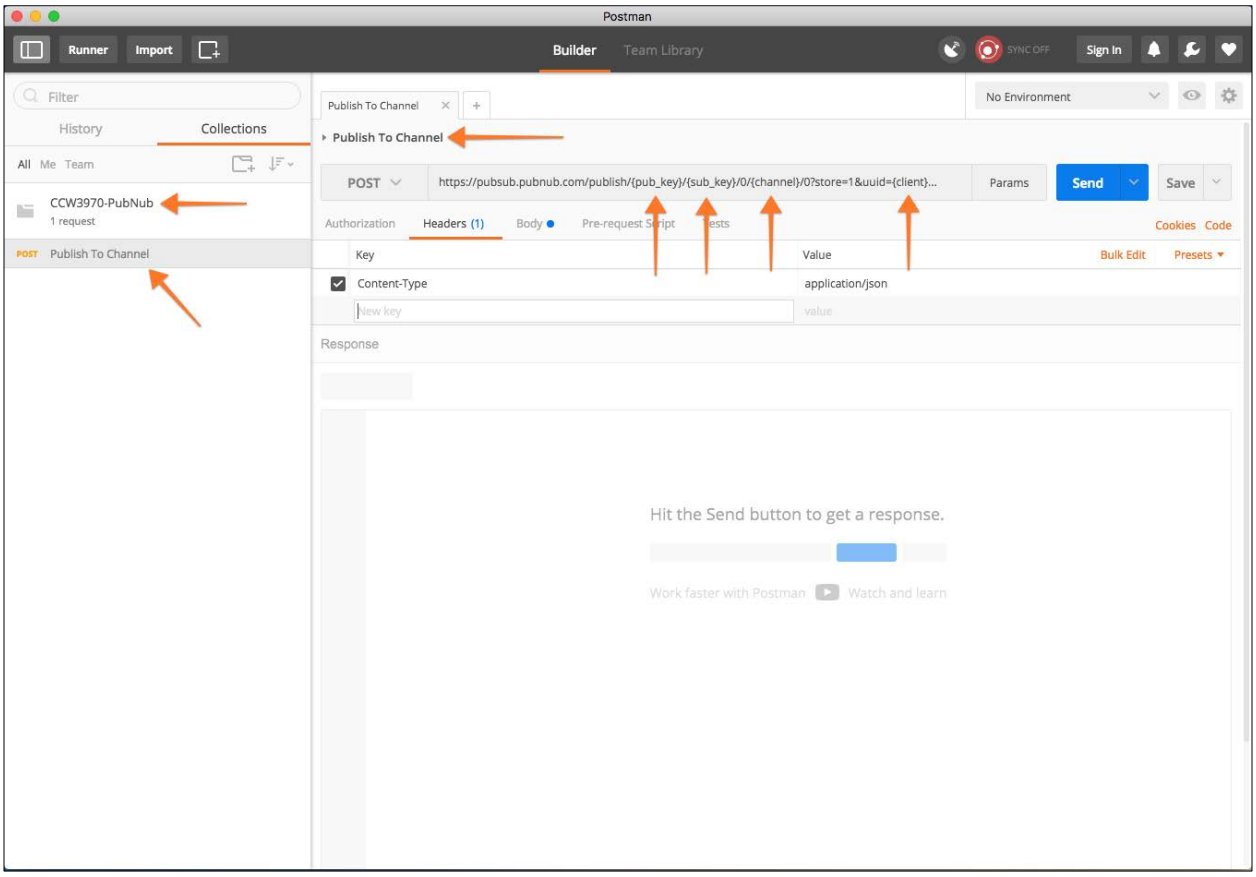


3. **Postman Collection Link:**
   https://www.getpostman.com/collections/a872c3162495ec77c946

Paste the link to our Postman collection in the **Import From Link** input box.



4. Verify you have the **CCW3970-PubNub** collection loaded by searching for it in the navigator on the left hand side.



PubNub Keys:

pub_key: pub-c-11b9ede6-f9ee-4da8-a829-944a45f29eb8
sub_key: sub-c-dafe9b8c-1ae1-11e7-bc52-02ee2ddab7fe

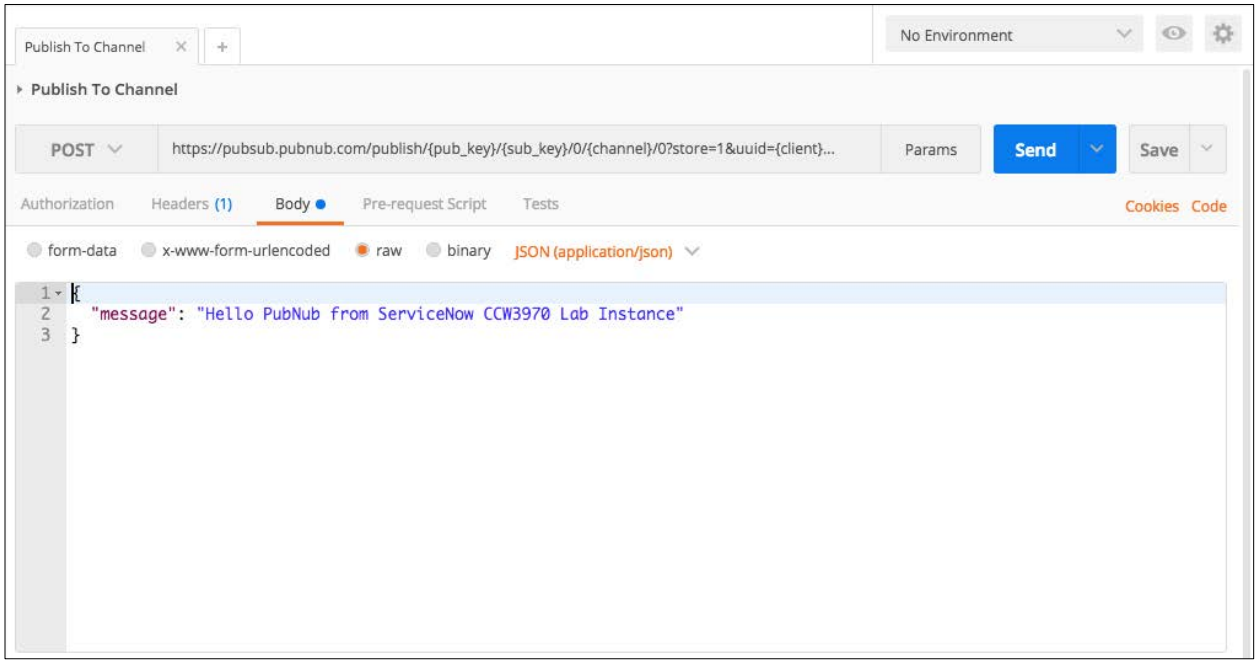
5. In the **CCW3970-PubNub** collection select the **Publish To Channel** operation.
    a. Replace the {pub_key} parameter with the pub_key provided in this lab doc.
    b. Replace the {sub_key} parameter with the sub_key provided in this lab doc.
    c. Replace the {client} parameter with your lab instance name (e.g., if your lab instance is lab1.service-now.com, replace the {client} param as 'lab1'.
    d. Replace the {channel} parameter with "CCW3970-{instance-name}" where {instance-name} is the name of your lab instance (e.g., CCW3970-lab1).

    Example:

    ```
    https://pubsub.pubnub.com/publish/pub-c-11b9ede6-f9ee-4da8-a829-944a45f29eb8/sub-c-dafe9b8c-1ae1-11e7-bc52-02ee2ddab7fe/0/CCW3970_lab1/0?store=1&lab1
    ```

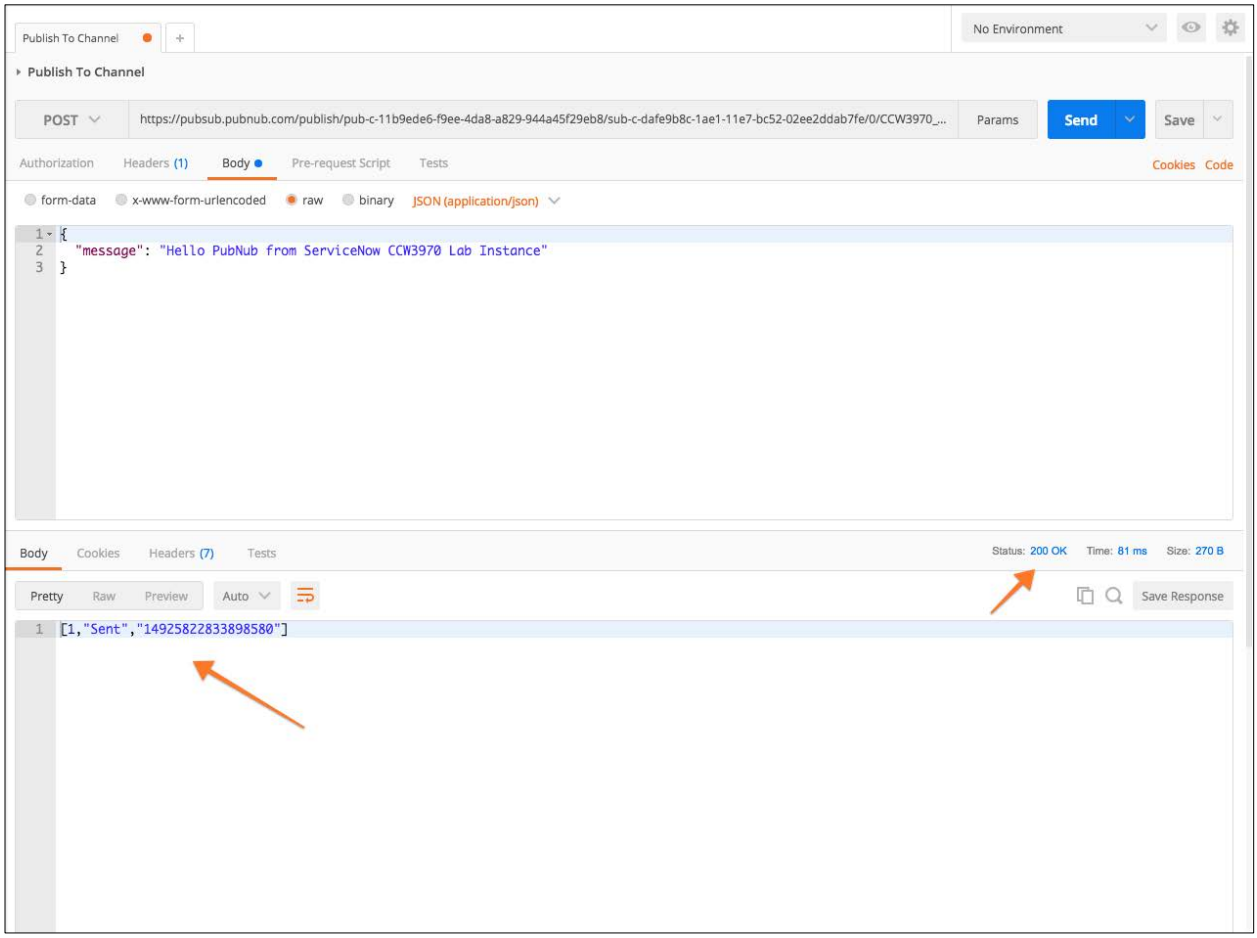
    e. Verify the headers specify 'Content-Type: application/json'.
    f. Verify the body includes the following as JSON.

```
{"message": "Hello PubNub from ServiceNow CCW3970 Lab Instance"}
```



g. Click **Send** to send the HTTP Request.

6. Verify the request was successful by looking for the **200 OK** status code and that the response payload contains “sent” as shown.



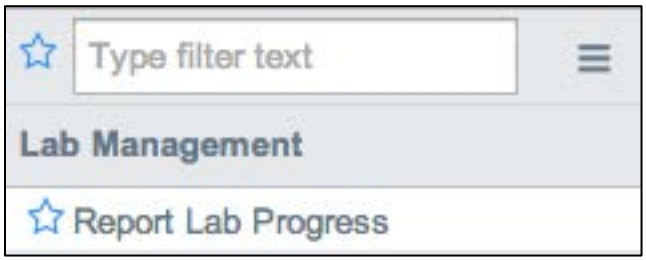
If you see **200 OK** you’ve now successfully published a message to PubNub using Postman from your local host. If you had errors check your URI and parameters or ask a Lab Guru for assistance.

This is an important step in building an integration because using a tool like Postman allows you to quickly familiarize yourself with a 3rd party API so that when you build your integration in ServiceNow you know that you’ve had a working request, understand how to format your request to send successfully and can refer back to this when building and testing your request in ServiceNow. In addition, it is becoming common for REST API providers to provide either cURL or Postman samples for consuming their APIs which can speed this process along.

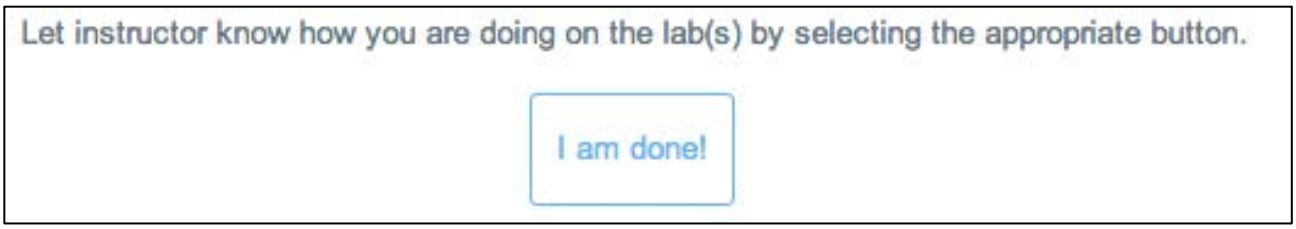
In the next lab you’ll use ServiceNow to issue HTTP requests to PubNub to publish messages from ServiceNow similar to how you used Postman in this **lab**.

Progress Report

3. Navigate to **Lab Management> Report Lab Progress**.



4. Click **I am done!**



Lab Goal

Lab 2
Publish Message with ServiceNow

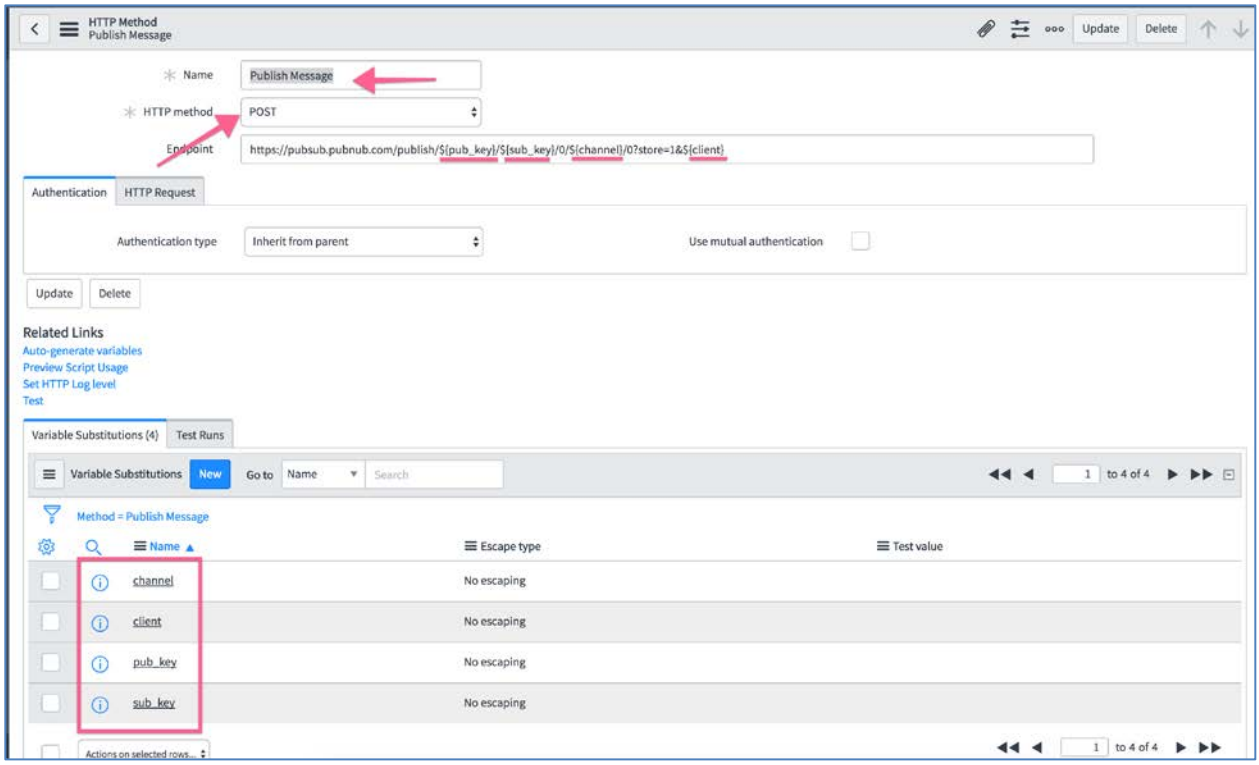
In the first **lab** you used Postman to publish messages to PubNub using their REST API. In this **lab** you'll use the ServiceNow RESTMessage capabilities to publish messages to PubNub. You'll start by configuring a RESTMessage record and testing your configuration using scripts background. Next you'll use business rules to trigger messages publishing to PubNub when Incident records are mutated. In addition, you'll use the outbound http request logs in ServiceNow to debug your requests.

Create Lab 2 starting branch

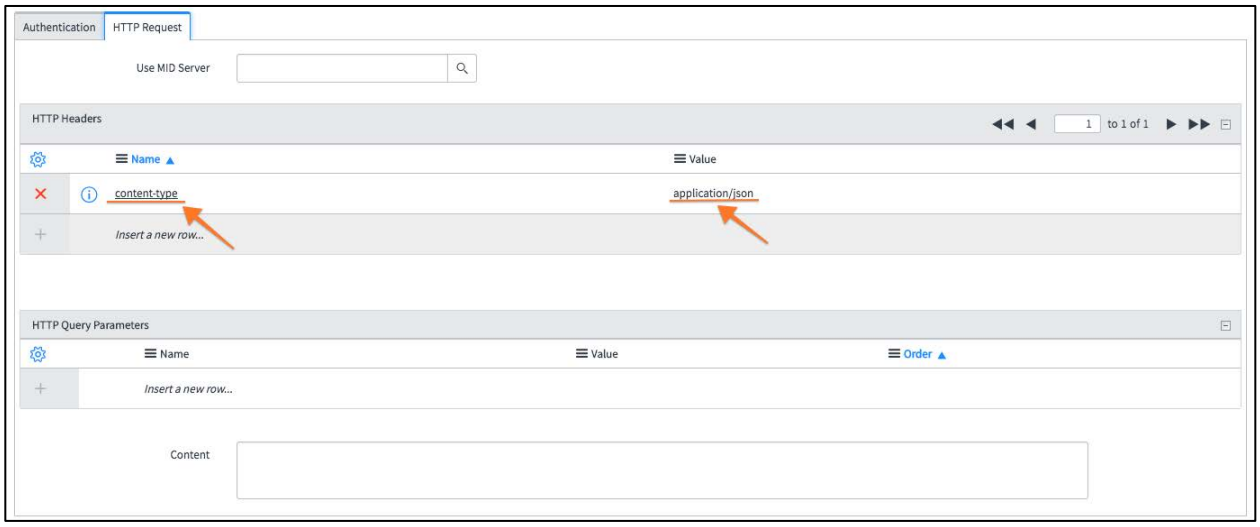
1. If you completed the lab setup, proceed to the next step. If you haven't yet completed lab setup, follow the steps in lab setup to create the **my-Lab2-branch** from the **Lab2-start** git tag.

Configure and Test with RESTMessage

1. In your ServiceNow lab instance navigate to **System Web Services -> REST Message**
2. Navigate to PubNub-> **Publish Message Record**. This message has been partially configured to send messages to the same PubNub REST API operation we sent a request to in Postman. Note the variables we've specified in the **Endpoint** field and the **Variable Substitutions** that exist in the related list at the bottom. This will allow us to easily specify these variables as parameters when using this RESTMessage HTTP Method from script.
3. Verify method is **POST**.

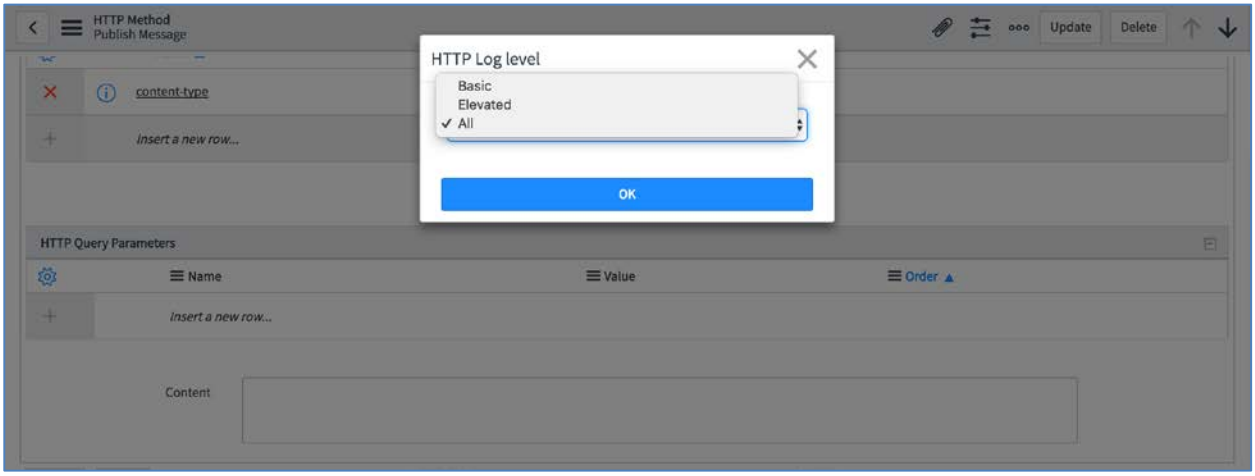


4. Populate Header "**Content-Type**" : "**application/json**"



5. Click **Preview Script Usage** in the list of Related Links, and you'll see auto generated sample script that can be used to execute this request from anywhere in ServiceNow where you can use Server Side script (e.g., Business Rules, Workflows, Script Actions).

6. Now set the **HTTP Log level** for this record to **All**. This allows you to control what level of detail is logged when outbound messages are sent from ServiceNow.
7. **Note**: For more information about what is included in each log level see [Outbound HTTP Logging] in the ServiceNow docs. No additional info about logging levels is necessary for this lab.

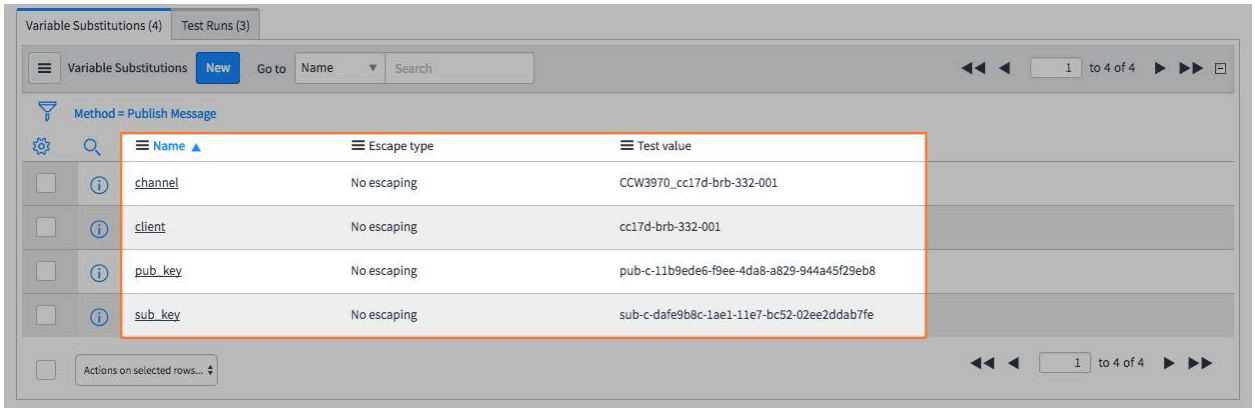


PubNub Keys:

pub_key: pub-c-11b9ede6-f9ee-4da8-a829-944a45f29eb8
sub_key: sub-c-dafe9b8c-1ae1-11e7-bc52-02ee2ddab7fe

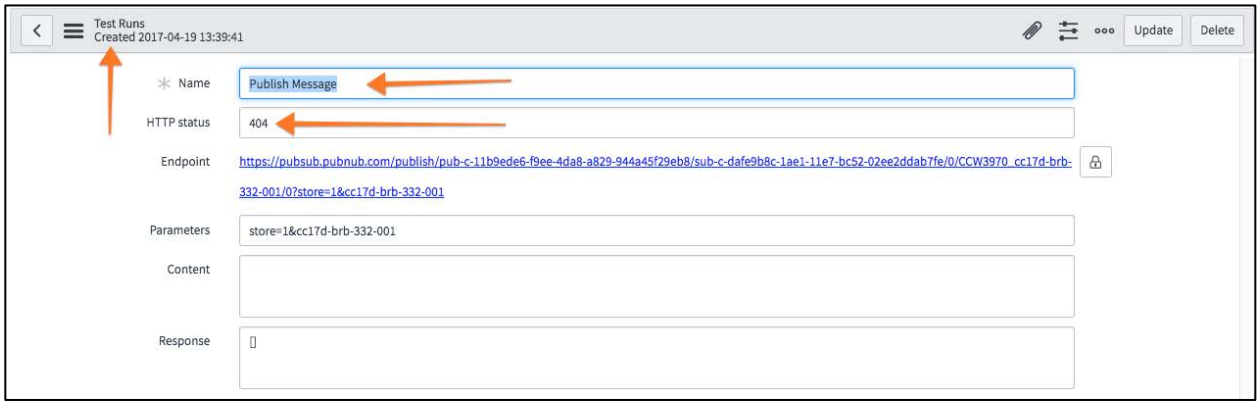
8. Populate Test Variables in **Variable Substitution** for:
    a. **pub_key**, specify pub_key provided in this lab guide
    b. **sub_key**, specify sub_key provided in this lab guide
    c. **client**, specify your lab instance name

d. **channel**, specify “CCW3970_{you lab instance name} (e.g, if your instance name iscc17d-brb-332-001, for the channel you would specify “CCW3970_cc17d-brb-332-001”.



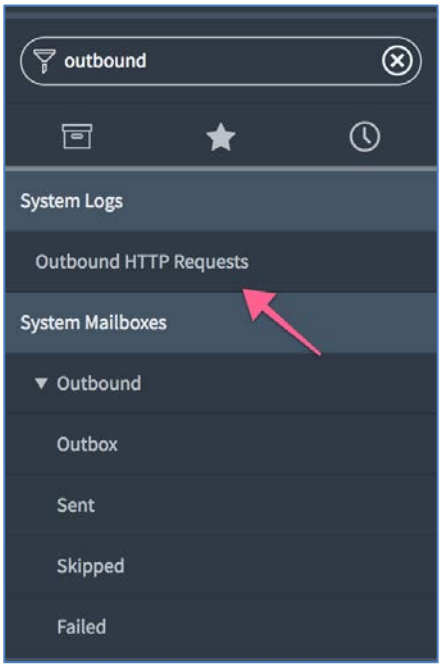
9. Click **Test**

10. Verify the request fails with HTTP status **404,** (this is expected).

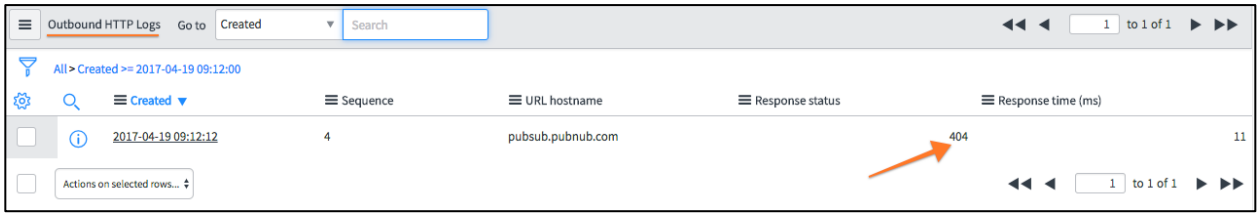


11. Now let’s figure out why. Go to the system logs to get a better idea of the request we sent and the response we received from PubNub. This will allow us to compare the request sent from ServiceNow with the successful request we sent from Postman and determine what we need to change.

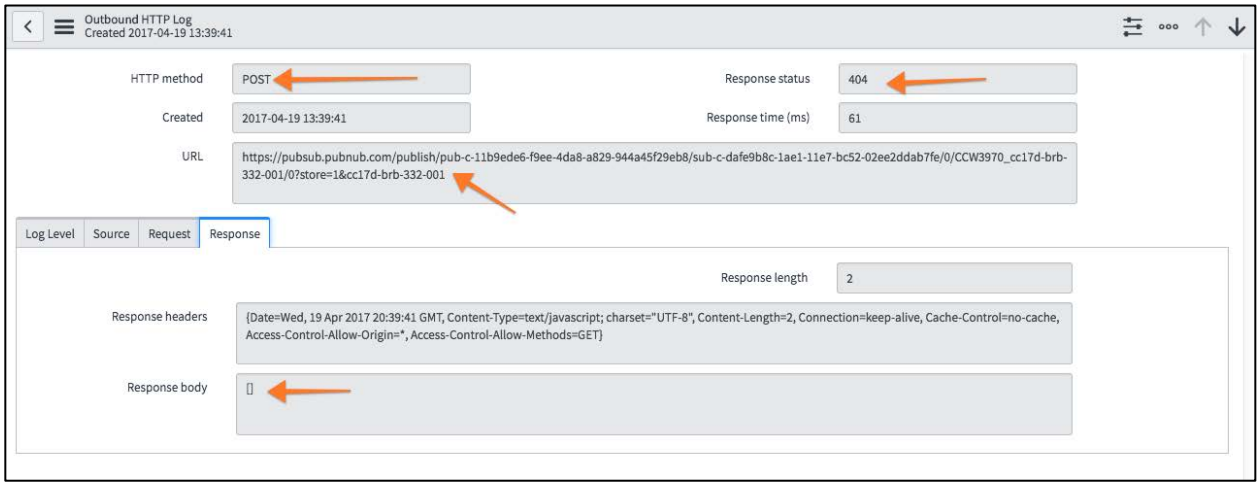
12. Navigate to **System Logs -> Outbound HTTP Requests.**



13. Review the list of recent outbound http requests.



14. Find the last sent message and view the log contents including:
    - Method
    - URL
    - Response Status
    - Response Time
    - Headers
    - Body



15. Compare the request sent from ServiceNow that **failed** with the successful request sent from Postman. What differs? Are there any messages in the response that indicate what the problem was? (Hint: look at the request body you sent from Postman and the one you sent from ServiceNow).

16. Go back to the **Publish Message** record in Studio and specify **content** of

```
{"message": "Hello PubNub from ServiceNow CCW3970 Lab Instance"}
```

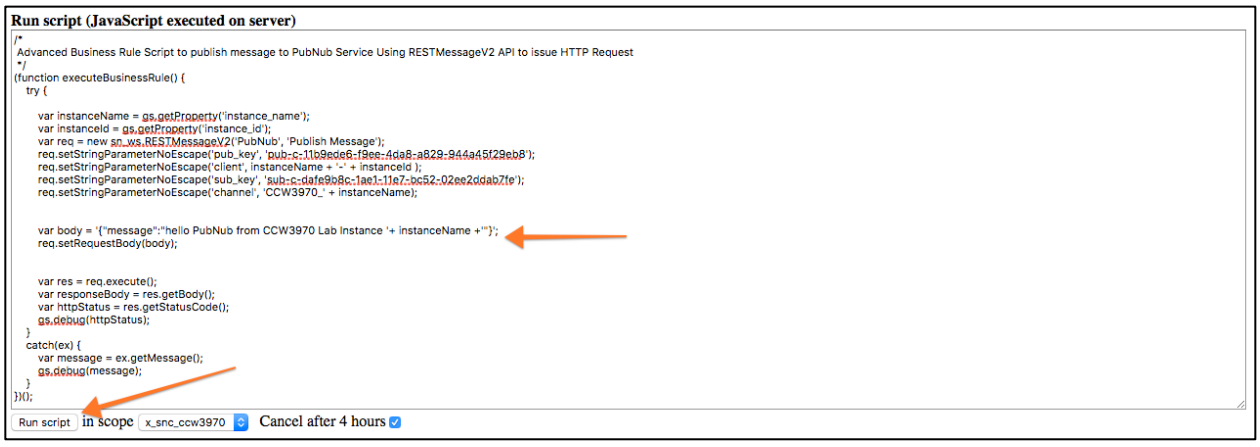
17. Save the record and run another test and verify your HTTP status is now **200.**

Use RESTMessage from Script

1. Now let's use the usage script to make a request and include a request body. First grab the usage script at by clicking **Preview Script Usage.**

2. Copy the usage script from the RESTMessage Record and navigate to **System Definition -> Scripts - Background.**

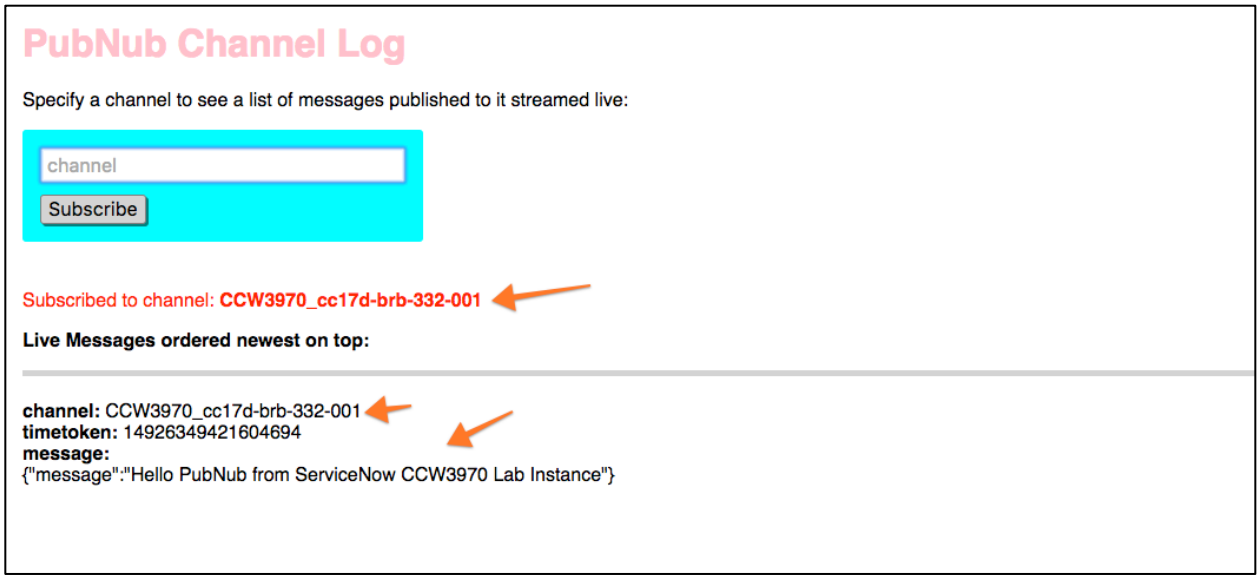
3. Paste into **Scripts Background**, update values (as shown below) to include a request body. this script can be copied from: https://raw.githubusercontent.com/CreatorCon17/CCW3970-Build-Debug-Outbound-REST-Snippets/master/ccw3970_scripts_background_restmessage_hello_pubnub.js



4. Send the request by clicking **Run Script**.
5. You should see a debug message indicating the response status code is **200** indicating a successful request. Let's look at the **Outbound HTTP Log** to see a bit more detail about the request and response.
6. Navigate to **System Logs -> Outbound HTTP Requests**.

7. Open the most recently sent message and review the sent request details. This allows you to see all the details of the sent request from ServiceNow to PubNub and the corresponding response. Having access to this level of detail is invaluable when trying to debug or verify communication between cloud based systems.
8. Let's also verify the message was received on PubNub. In a new browser tab navigate to https://ccw3970-demo.glitch.me/

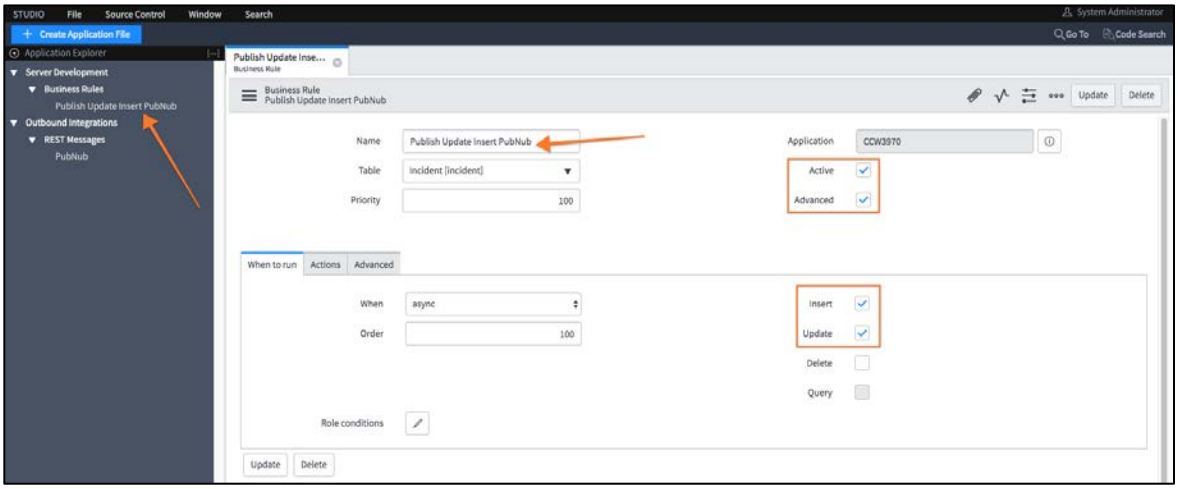
   Enter the channel name you specified when sending the request (e.g., CCW3970_cc17d-brb-332-001) and click **Subscribe**. This is a lightweight web application that can subscribe to the PubNub channels and will automatically update when messages are published to the channel it's subscribed to.



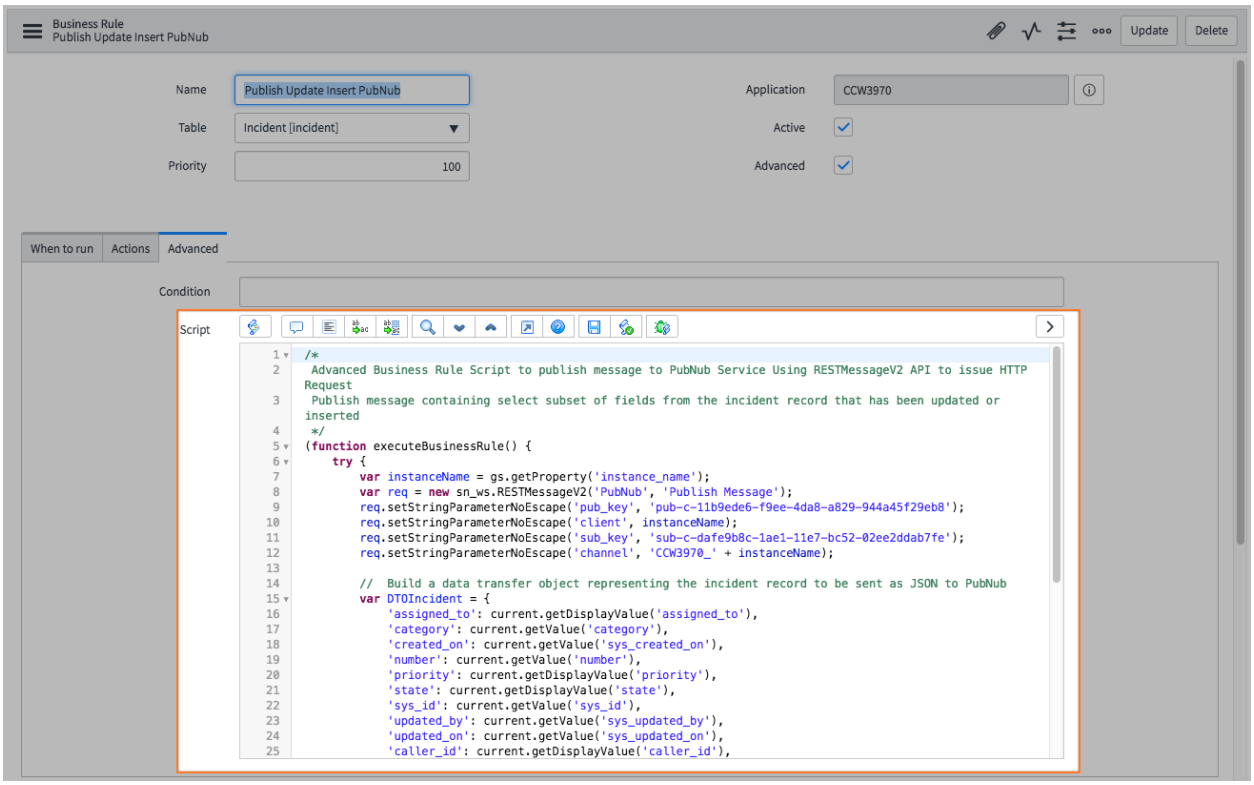
9. Issue another request using Scripts Background and then and you should see your message show up in the PubNub Channel Log at https://ccw3970-demo.glitch.me/ without needing to update (page will auto update when new messages are published).
10. Now that you've seen how you can publish a message (send a HTTP request) from a script let's put this to use and configure a business rule to publish messages to PubNub when an Incident is either inserted or updated in your lab instance.

Configure Business rule to Publish Messages to PubNub

1. In **Studio** open the **Publish Update Insert PubNub** business rule which is part of the **CCW3970** application.



2. Verify the business rule is configured to run on **insert** and **update**, **advanced** is checked and when is set to run **aysnc**. You will be making changes to this business rule so that when it is triggered, on insert or update, of an Incident record a message will be published to PubNub.
3. In the business rule set the advanced script by copying and pasting values from. https://raw.githubusercontent.com/CreatorCon17/CCW3970-Build-Debug-Outbound-REST-Snippets/master/ccw3970_advanced_business_rule_publish_updates_to_pubnub.js



4. **Save** the record and review this **Script**. Note we are still using the RESTMessage record but now we are populating the body with values from the inserted or updated Incident.

```
/*
 Advanced Business Rule Script to publish message to PubNub Service Using RESTMessageV2 API to issue HTTP Request
 Publish message containing select subset of fields from the incident record that has been updated or inserted
 */
(function executeBusinessRule() {
    try {
        var instanceName = gs.getProperty('instance_name');
        var req = new sn_ws.RESTMessageV2('PubNub', 'Publish Message');
        req.setStringParameterNoEscape('pub_key', 'pub-c-11b9ede6-f9ee-4da8-a829-944a45f29eb8');
        req.setStringParameterNoEscape('client', instanceName);
        req.setStringParameterNoEscape('sub_key', 'sub-c-dafe9b8c-1ae1-11e7-bc52-02ee2ddab7fe');
        req.setStringParameterNoEscape('channel', 'CCW3970_' + instanceName);

        //  Build a data transfer object representing the incident record to be sent as JSON to PubNub
        var DTOIncident = {
            'assigned_to': current.getDisplayValue('assigned_to'),
            'category': current.getValue('category'),
            'created_on': current.getValue('sys_created_on'),
            'number': current.getValue('number'),
            'priority': current.getDisplayValue('priority'),
            'state': current.getDisplayValue('state'),
            'sys_id': current.getValue('sys_id'),
            'updated_by': current.getValue('sys_updated_by'),
            'updated_on': current.getValue('sys_updated_on'),
            'caller_id': current.getDisplayValue('caller_id'),
            'active': current.getValue('active')
        };

        //  Convert DTO to JSON string
        var body = JSON.stringify(DTOIncident);
        req.setRequestBody(body);


        //  Execute request
        var res = req.execute();
        var responseBody = res.getBody();
        var httpStatus = res.getStatusCode();
        gs.debug(httpStatus);
    }
    catch (ex) {
        var message = ex.getMessage();
        gs.debug(message);
    }
})();
```

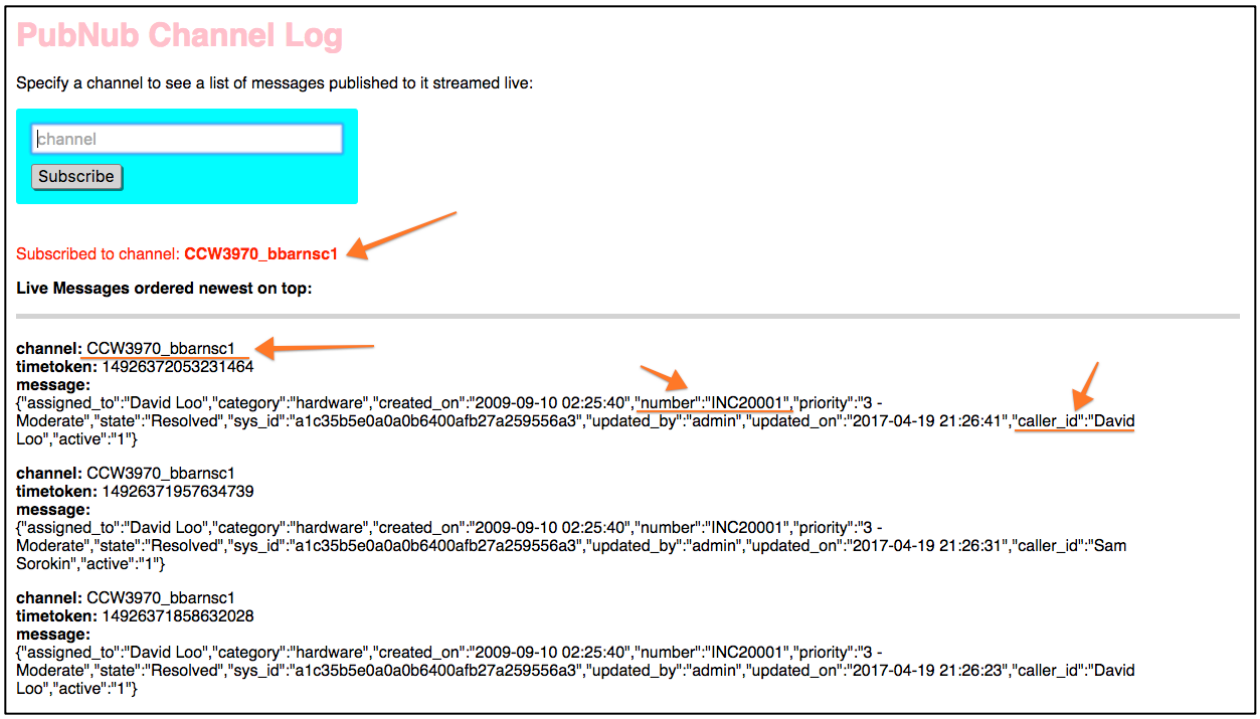
Build request

Build data transfer object from Incident

Convert to JSON string

Publish message

5. The **business rule** is now configured to publish the JSON (data transfer object) representation of the Incident to PubNub whenever an Incident record is created or updated.
6. **Try it out**, update an Incident Record. Change the **Caller** to **David Loo**. Save the incident record and be sure to note the Incident Number.
7. Go to the Outbound HTTP Logs and verify that a request with a payload including this Incident number was sent to PubNub and that the response status was **200**.
8. Verify on PubNub that the message was received. If you still have your other browser tab open to **https://ccw3970-demo.glitch.me/** then you should see a new message has been added to the top of the log. If you closed your browser tab then you'll need to reopen it and subscribe to the appropriate channel. **Note**: The channel name should be "CCW3970_{your lab instance_name}. You can always go back to your advanced business rule script and find it as well. Channel names are case sensitive.
9. In my example shown below. I updated **INC20001,** setting the caller to **David Loo**. My instance name was bbarnsc1 and the corresponding channel name that I subscribed to was **CCW3970_bbarnsc1**. You should see something similar.



If you see your messages great! You've successfully completed this lab and you've now configured your ServiceNow instance to publish messages to PubNub using Business Rules, and the RESTMessageV2 API when Incidents are created or updated in your lab instance.

If You don't see these messages in the outbound HTTP log or in the PubNub Channel Log then review your script for variances or ask a lab guru for help.

In the next lab we'll see how we can do this using ServiceNow Workflow and Orchestration.

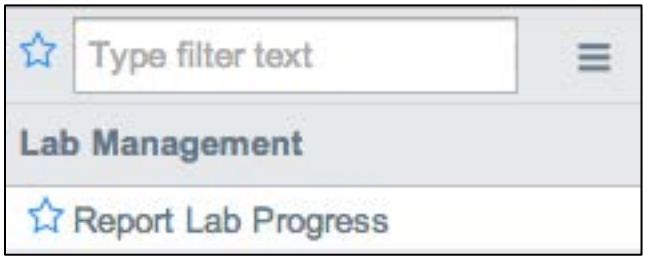
Catch Up

1. If you had problems with this lab and want to fast forward to the end of lab 2 to review the completed updates you can follow the same process, you followed in **Lab Setup** to create a branch from the **Lab2-end tag**. This will update your application to a state that you would be if you successfully completed Lab2.
2. In Studio, navigate to **Source Control** > **Create Branch**.
3. In the pop-up window, enter a branch name, then select **Lab2-end** from the **Create from Tag** menu, and click **Create Branch**.

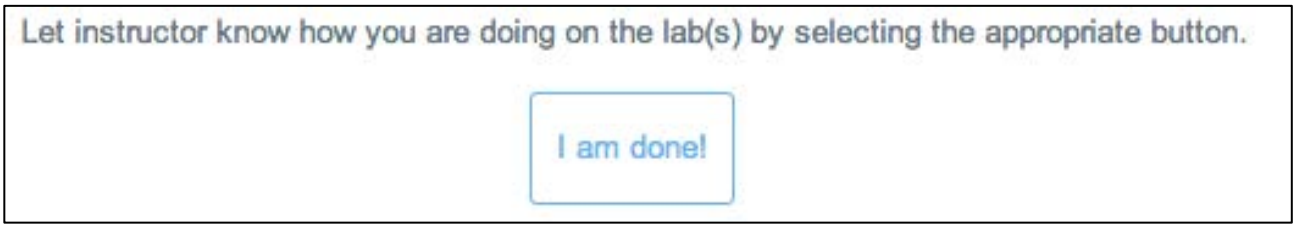
   Branch: **my-Lab2-end-branch**
   Create from Tag: **Lab2-end**

Progress Report

1. Navigate to **Lab Management> Report Lab Progress**.



2. Click **I am done!**



Lab Goal

Now we're going to implement the same use case as the previous lab, but using Orchestration and Workflow. Orchestration is a powerful tool for building low-code and no-code integrations. Workflow enables the simple automation of processes and tying together actions in ServiceNow.

We'll start with a simple "Hello, world" PubNub REST Activity, then expand it to be triggered from an Incident being created in ServiceNow using Workflow.

Lab 3
Use Orchestration

Create Lab 3 starting branch

1. If you completed the lab setup, proceed to the next step.

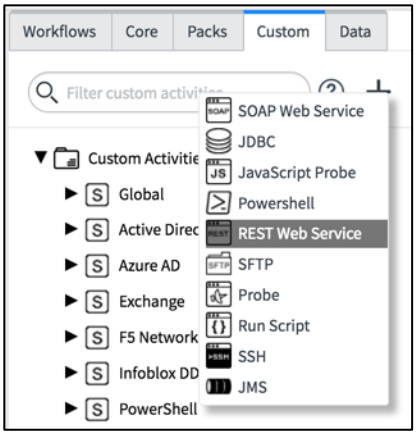
   If you haven't yet completed lab setup, follow the steps in lab setup to create the

   my-Lab3-branch from the **Lab3-start** git tag.

Create a REST Activity for PubNub

1. Open the **Orchestration > Workflow Editor**.

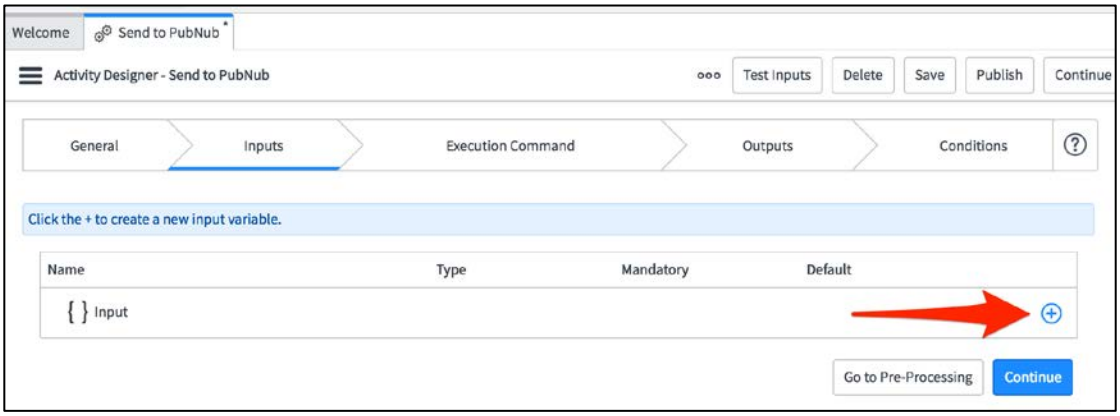
2. From the **Custom** tab, click the '**+**' icon to create a new **REST Web Service** activity.



   This will open the Activity Designer using the REST Web Service template.

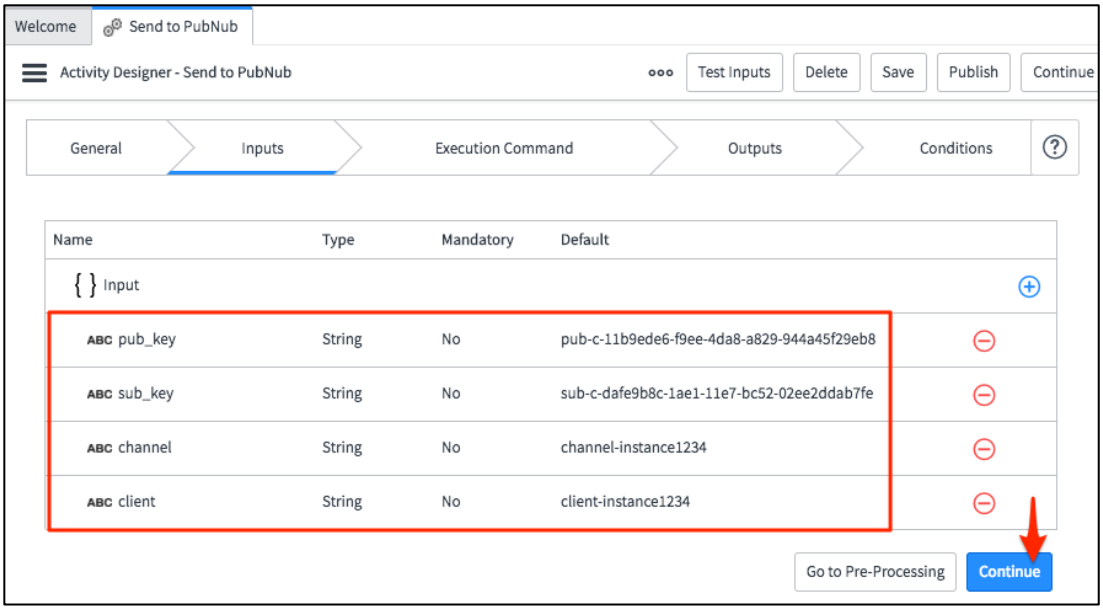
   Give the activity a name such as "**Send to PubNub Hello World**" and click **Continue**.

3. On the **Inputs** tab, click '**+**' to define an input variable.



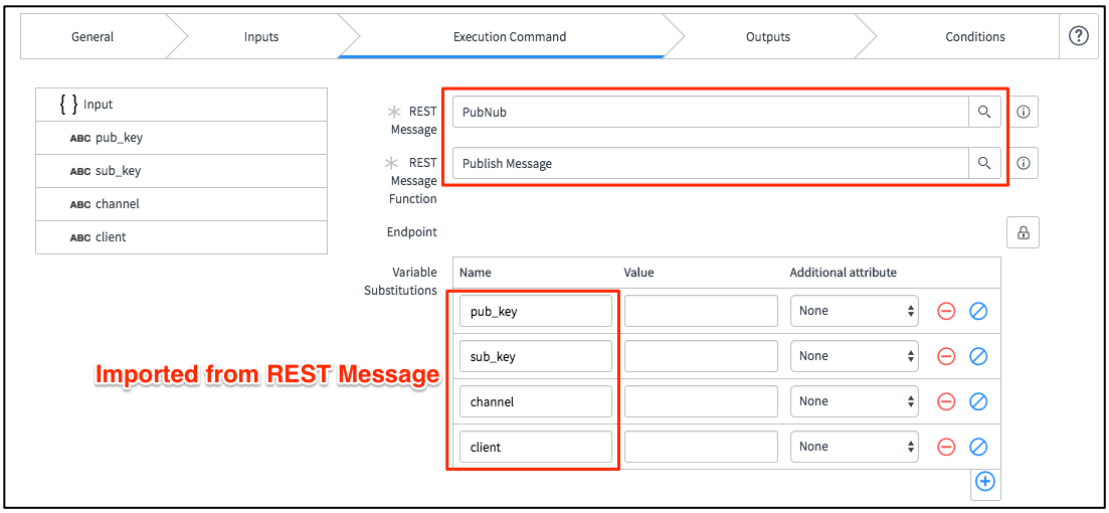
Define the following input variables:

pub_key (default value: **pub-c-11b9ede6-f9ee-4da8-a829-944a45f29eb8**)
sub_key (default value: **sub-c-dafe9b8c-1ae1-11e7-bc52-02ee2ddab7fe**)
channel (default value: **CCW3970_{*lab instance name*}**)
client (default value: **{*lab instance name*}**)

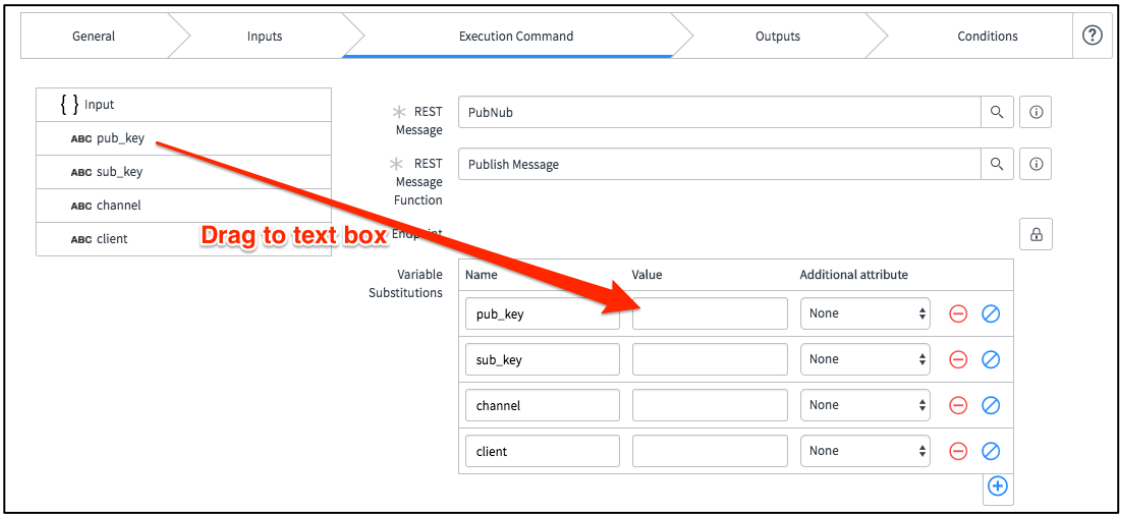


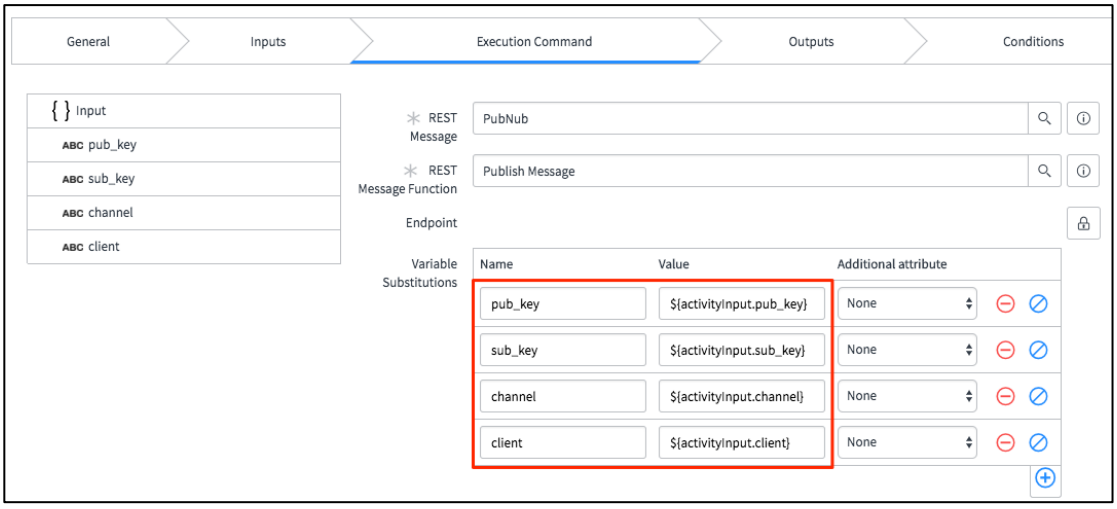
4. On the **Execution Command** tab, select your REST Message and function. Reference the REST Message and HTTP Method created in the previous lab.

NOTE: if the REST Message defined Variable Substitutions, then they will be automatically imported into the REST Activity. But if not, you can click the '**+**' button to just define them here.



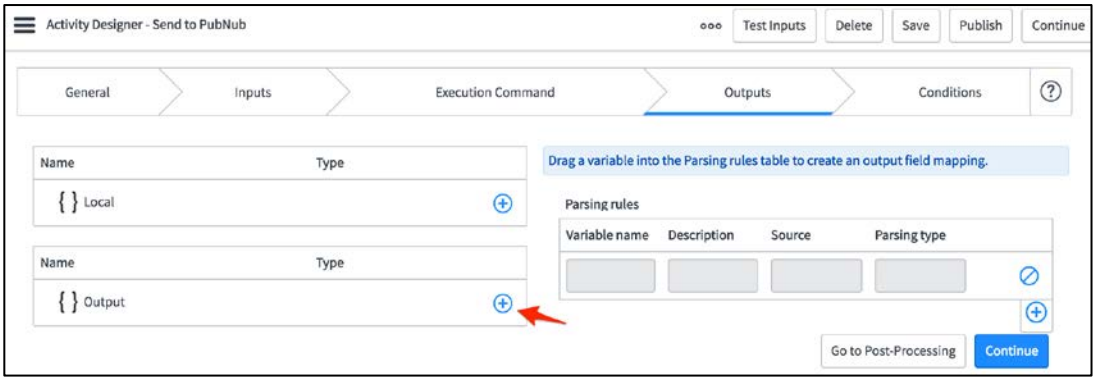
5. Drag & drop the **message** and **channel** input variables to the corresponding Variable substitution field.



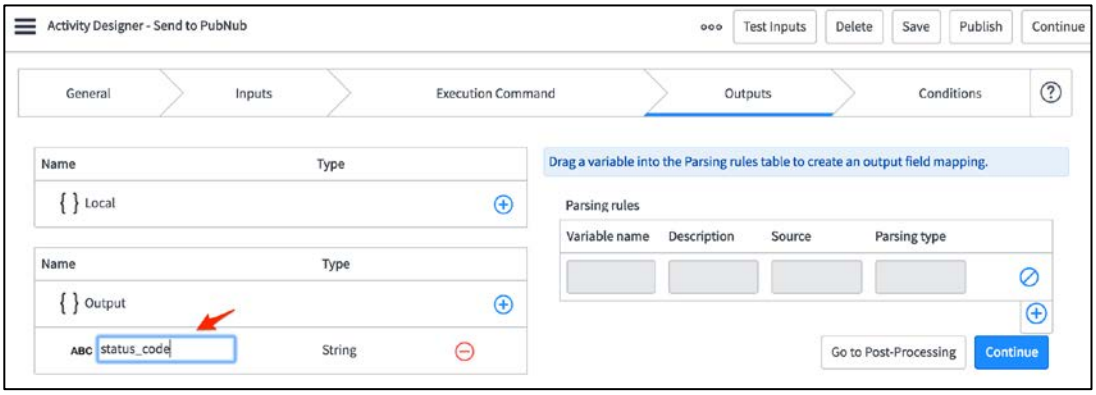


Then click **Continue**.

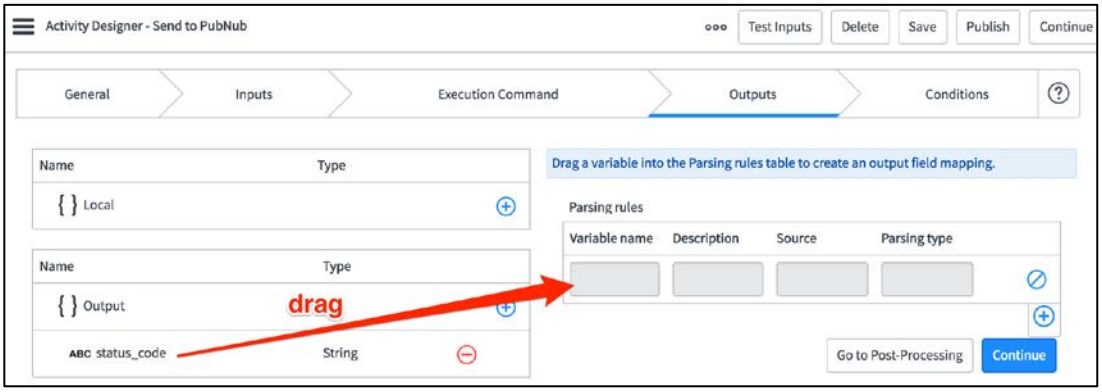
6. On the **Outputs** tab, add the HTTP status code as an output variable



Name the variable '**status_code**'

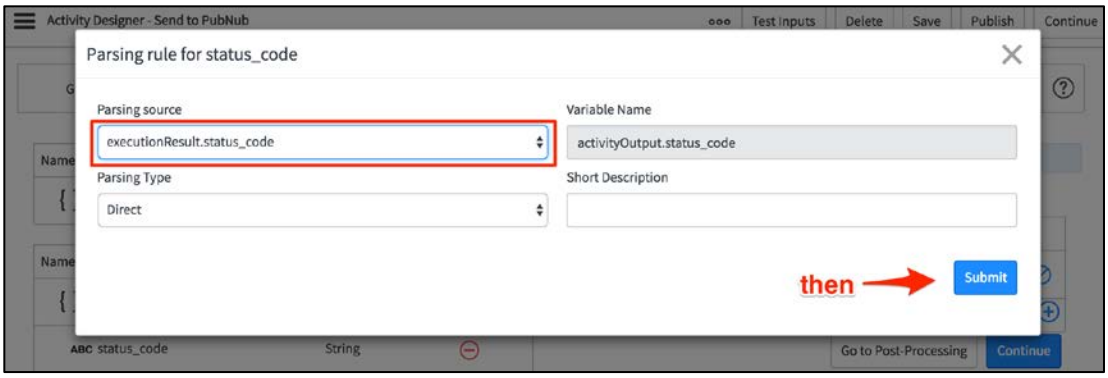


7. Drag the **status_code** variable to the parsing rules field

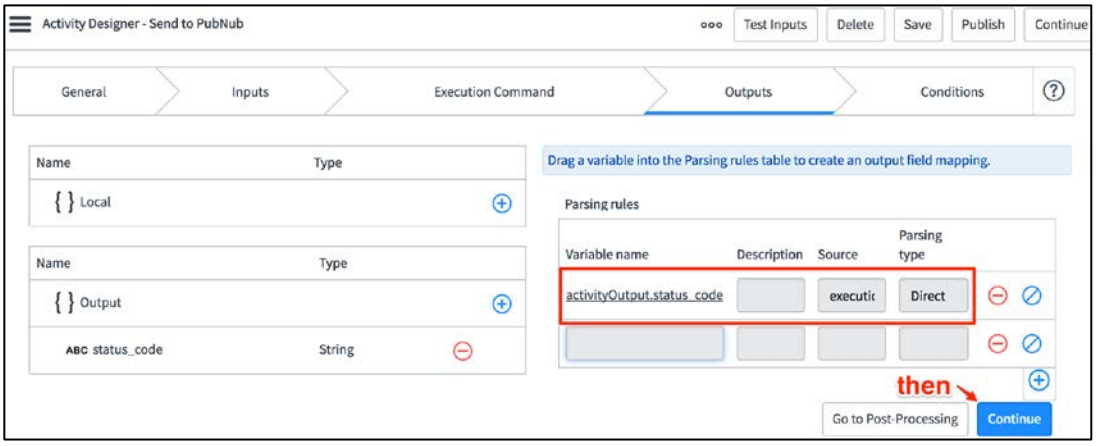


8. In the **Parsing rule** popup:

Parsing source: executionResult.status_code
Parsing Type: Direct



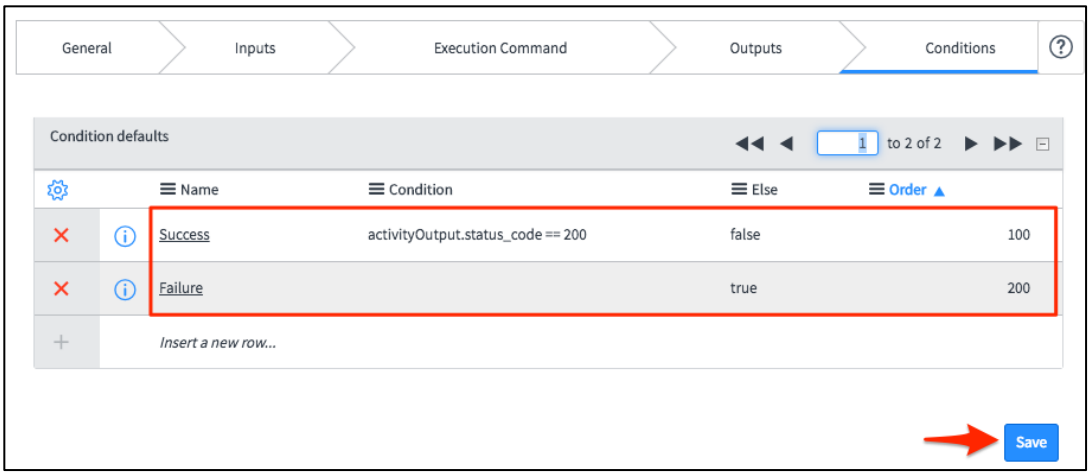
9. Now click **Continue** on the Outputs tab



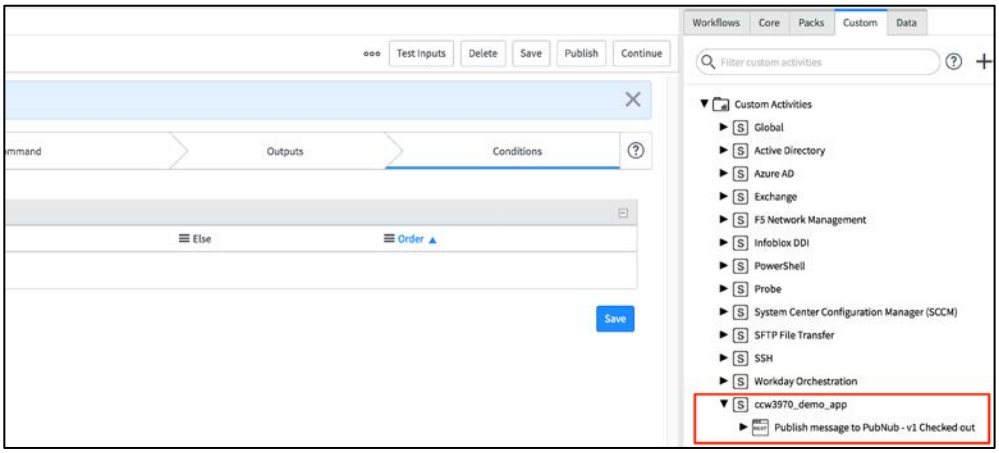
10. On the **Conditions** tab, add the following two conditions, then click **Save**.

    Name: Success
    Condition: activityOutput.status_code == 200
    Else: false
    Order: 100

    Name: Failure
    Condition: (empty)
    Else: true
    Order: 200



11. Note that your new REST Activity is defined as a Custom activity in the right-hand pane.



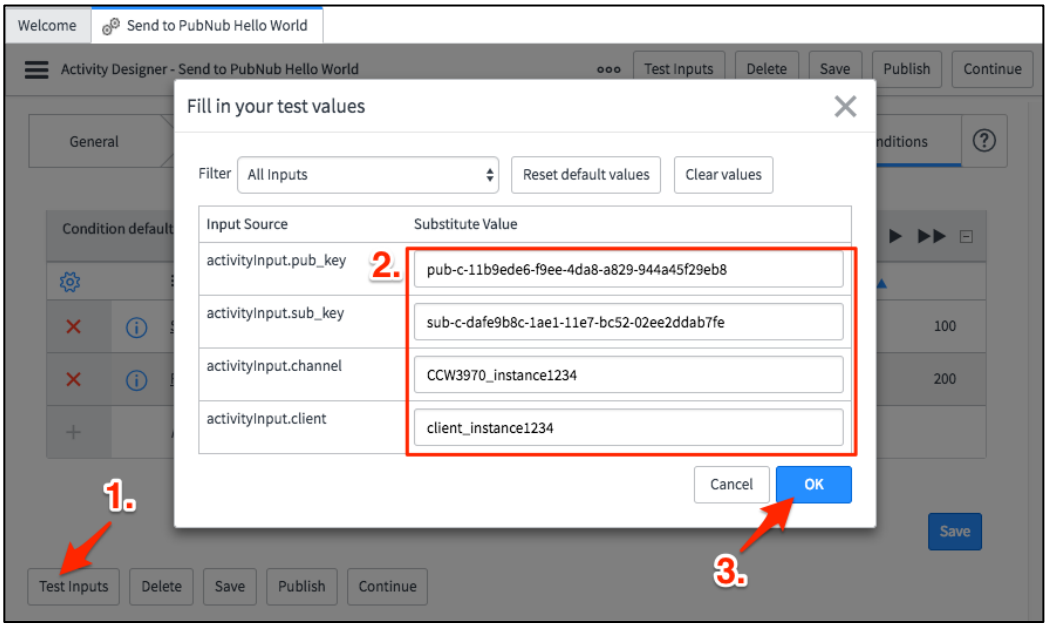
Test the REST Activity

1. To view the message that was received by PubNub. In your browser navigate to **https://ccw3970-demo.glitch.me/** and enter values for the channel name you plan to use and click Subscribe. Now send the test.

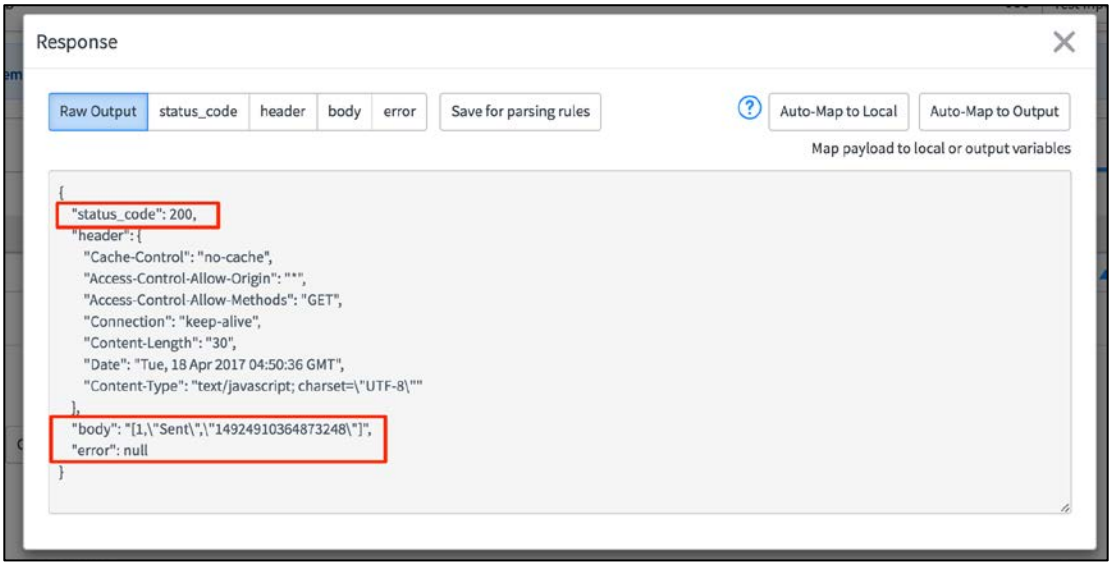
   This should be the default value, **CCW3970_<*lab instance name*>,** but can be specified ad-hoc.

2. Click **Test Inputs**. On the popup, enter values for the input fields. These should already be populated from default values from the Activity, but if not, enter as below.

   activityInput.pub_key: pub-c-11b9ede6-f9ee-4da8-a829-944a45f29eb8
   activityInput.sub_key: sub-c-dafe9b8c-1ae1-11e7-bc52-02ee2ddab7fe
   activityInput.channel: CCW3970_instance1234
   activityInput.client: client_instance1234



3. On the Response page, verify the status code is **200** and that the message was **sent**.



4. Navigate to **System Logs > Outbound HTTP Requests** again to view the most recent HTTP request. Verify the request is listed.

5. Verify the message was received for the channel at **https://ccw3970-demo.glitch.me/**.

   NOTE: You need to subscribe to the channel you are publishing to **BEFORE** the message is sent in order to see it in the PubNub Channel Log app.

6. You can repeat the test steps above as many times as you'd like.

7. When the REST Activity is behaving as expected, **Publish** the Activity.

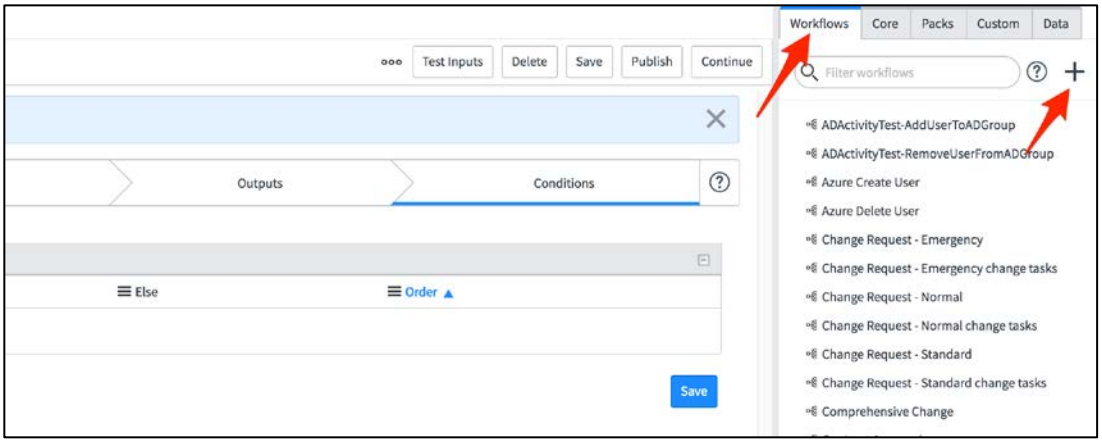
Mid-way Catch Up

4. This is the midpoint of lab 4. If you had problems with this lab up to this point and want to fast forward to catch up to review the completed updates you can follow the same process, you followed in **Lab Setup** to create a branch from the **Lab3-mid tag**. This will update your application to a state that you would be if you successfully completed Lab 3 up to the current midpoint.
5. In Studio, navigate to **Source Control** > **Create Branch**.
6. In the pop-up window, enter a branch name, then select **Lab3-mid** from the **Create from Tag**
   menu, and click **Create Branch**.

   Branch: **my-Lab3-mid-branch**
   Create from Tag: **Lab3-mid**

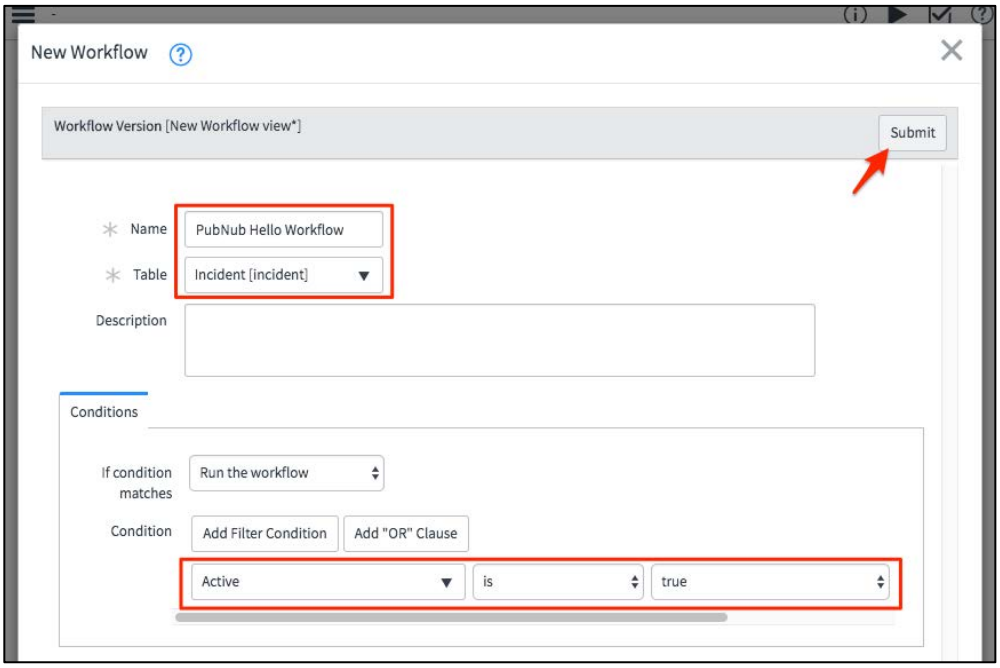
Use the Hello World REST Activity in a Workflow

1. Click the **Workflow** tab, then click the '**+**' icon to create a new Workflow.

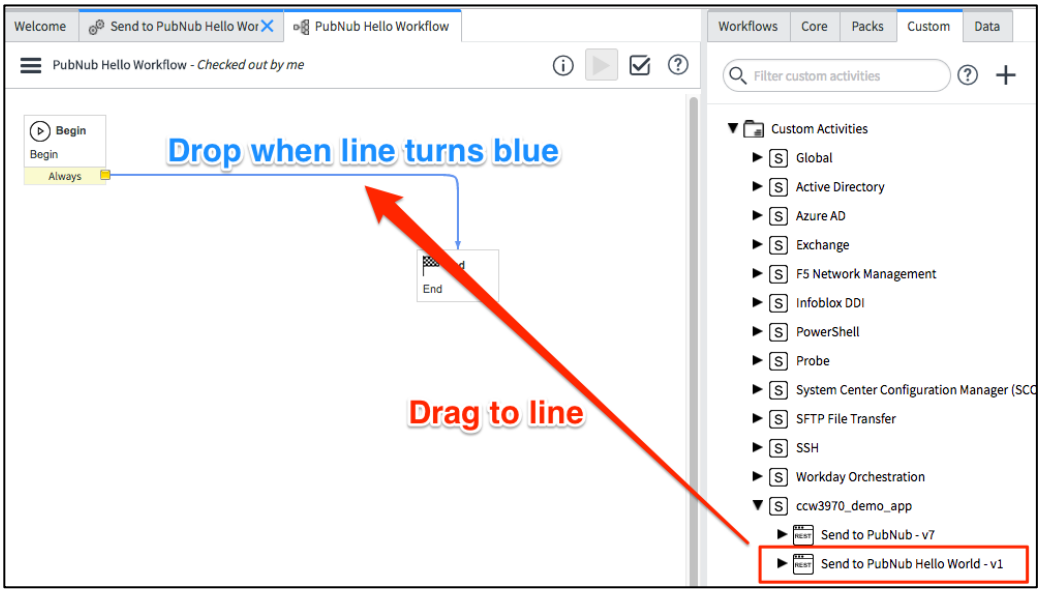


2. On the New Workflow page, give the Workflow a Name and configure as follows, then click **Submit**.

Table: Incident
Condition: Active is true



3. From the **Custom** tab, Drag and drop the PubNub REST Activity to the Workflow. Drop it on the Workflow when the line turns blue.

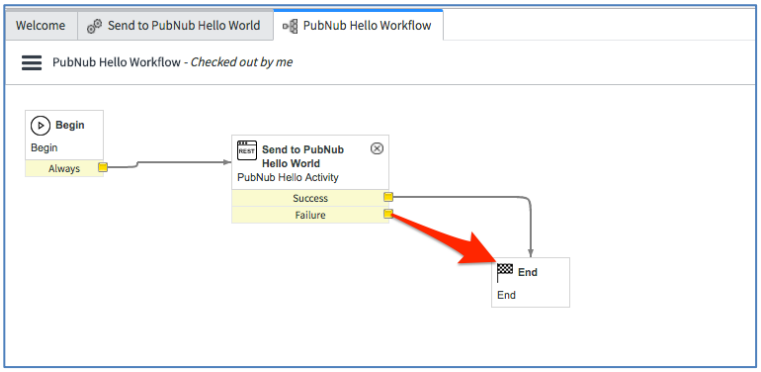


4. Give the Workflow Activity a name, and enter the Activity inputs as before.

Name: PubNub Hello Activity
Pub Key: pub-c-11b9ede6-f9ee-4da8-a829-944a45f29eb8

Sub Key: sub-c-dafe9b8c-1ae1-11e7-bc52-02ee2ddab7fe
Channel: CCW3970_instance1234
Client: instance1234

5. Drag a line to connect from the Failure condition to the End state.



NOTE:
In a real application, handling failure is critical. You can imagine how receiving an HTTP 400 error response might result in the failure being logged somewhere, and the call NOT being retried since that is an unrecoverable condition.

What should be the behavior if an HTTP 500 error is received? Hint: retry. Limit the number of retries, and consider an exponential back-off (wait) period between retries. You can implement this retry logic in the workflow logic.

Test the Workflow

1. Get ready to view the message was received by PubNub. In your browser navigate to **https://ccw3970-demo.glitch.me/** and enter values for the channel name you plan to use to send the test.

   This should be the default value, **CCW3970_<*lab instance name*>,** but can be specified ad-hoc.

2. Create a new **Incident** record.

3. Navigate to **System Logs > Outbound HTTP Requests** again to view the most recent HTTP request. Verify the request is listed.

4. Verify the message was received for the channel at **https://ccw3970-demo.glitch.me/**.

   NOTE: You need to subscribe to be subscribed to the channel BEFORE the message is sent in

order to receive it in the glitch app.

5. You can repeat the test steps above as many times as you'd like.

Update the REST Activity to pull data from the Incident record

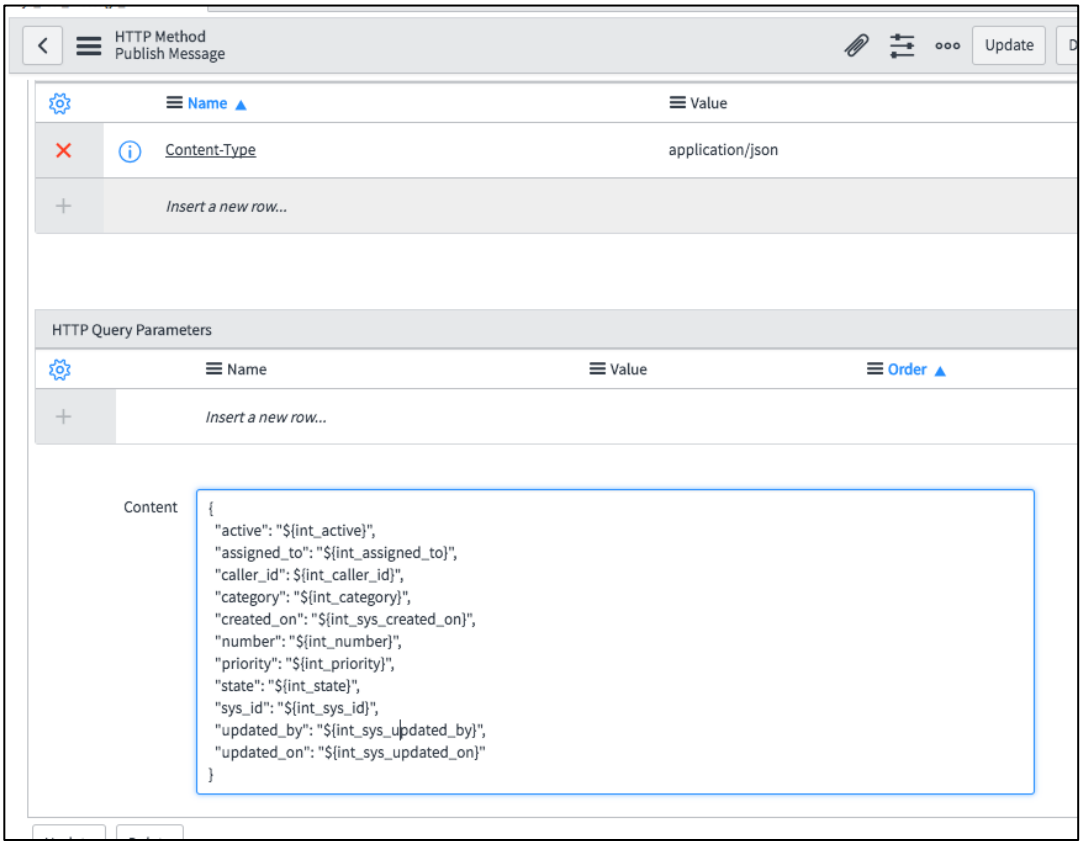
1. To save time, we've pre-built a REST Activity. Now we need to modify it to include an additional field **caller_id** in the payload message.

   In **Studio**, Open the PubNub **REST Message**. We're going to make further use of REST Message template variables e.g. ${sys_created_on}.

   Open the **HTTP Method** "Publish Message", and select the **HTTP Request** tab. Note the **Content** field still contains the message payload from before.

   ```
   {"message": "Hello PubNub from ServiceNow CCW3970 Lab Instance"}
   ```

2. Replace the **Content** field with this payload. Note that it contains template variables.

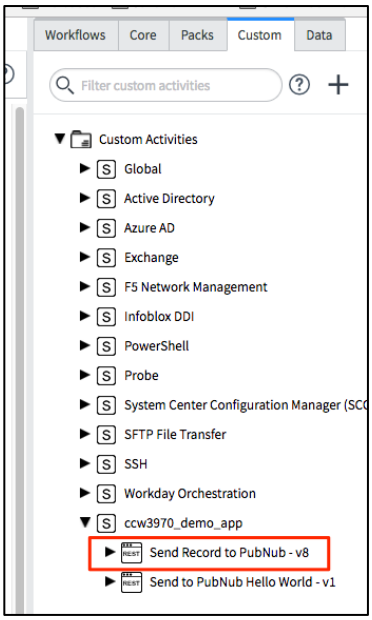


   Note the payload can be copy/pasted from the GitHub snippets repo (file **ccw3970_restmessage_content_body.json**) or directly from:

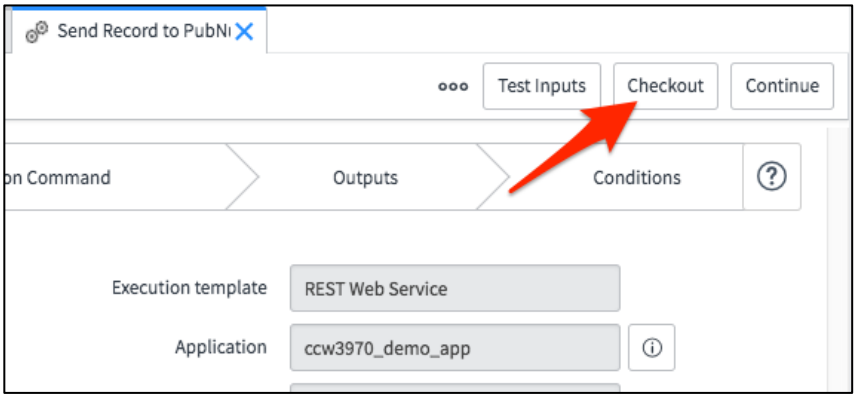
https://raw.githubusercontent.com/CreatorCon17/CCW3970-Build-Debug-Outbound-REST-Snippets/master/ccw3970_restmessage_content_body.json

3. From REST Message, click **Test** to test the new payload is sent correctly to PubNub. Verify channel settings are correct in the "Test Value" field.

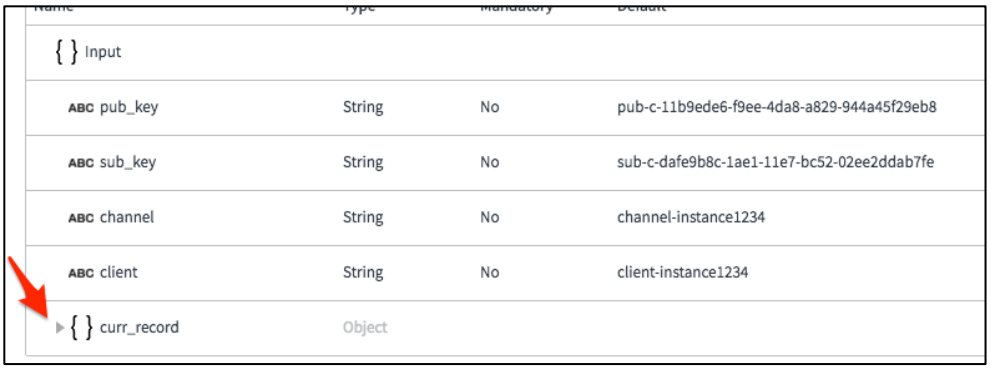
4. Go back to the **Orchestration** > **Workflow Editor**. Open the **Send Record to PubNub** Activity.



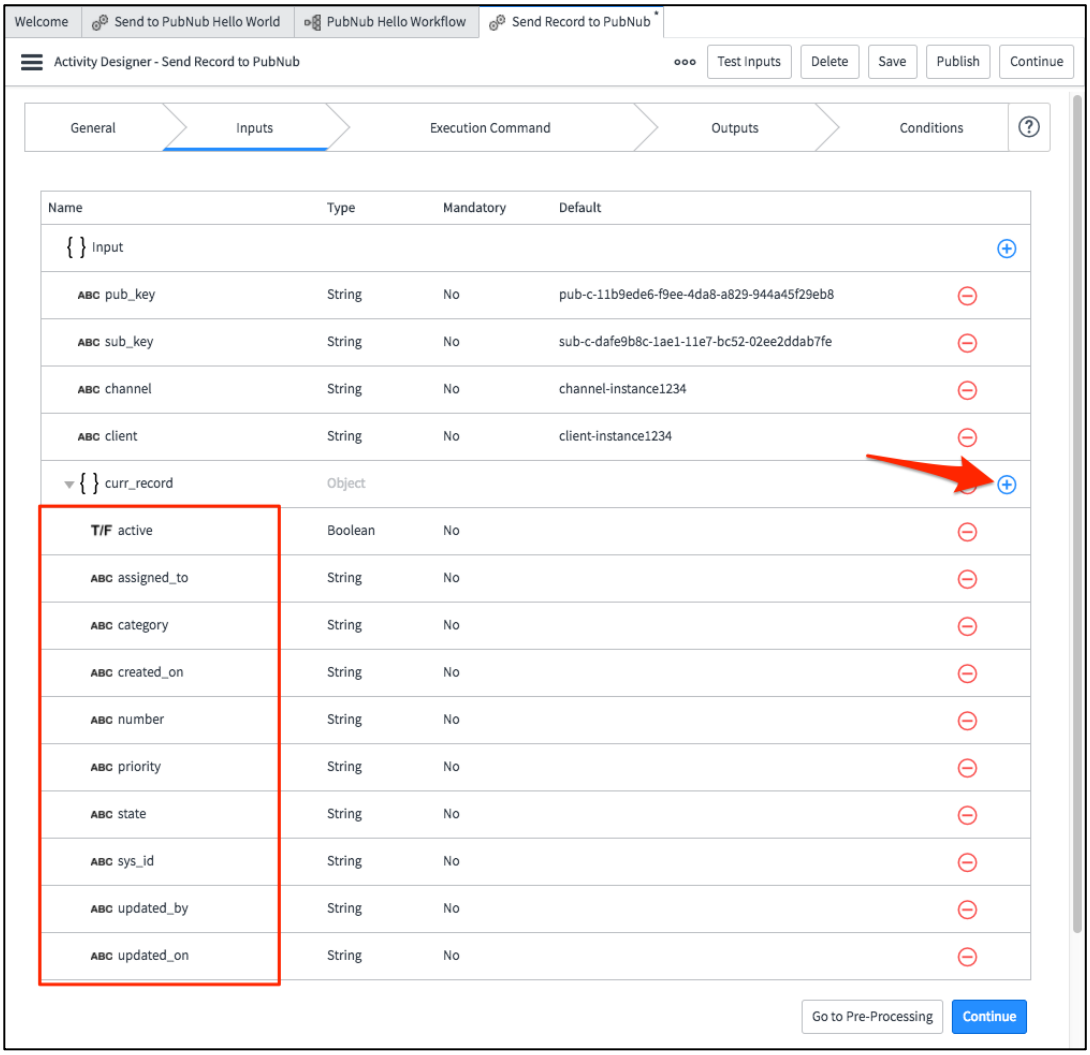
5. Click **Checkout** to edit the Activity.



6. On the **Inputs** tab, we need to add a new input field to the *curr_record* Object. Expand the *curr_record* Object.



7. Note that **caller_id** is not currently defined. Click the '**+**' icon to add a new field to the Object definition.



8. Give the field the name “**caller_id**” and press **Enter** key, then click **Continue**.

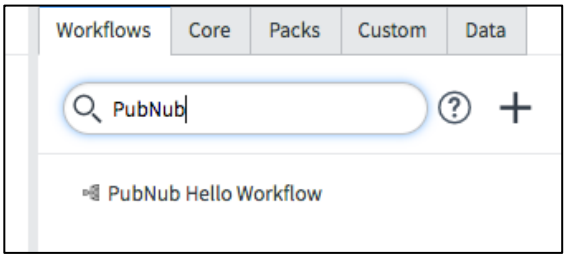
9. On the **Execution Command** tab, click ‘**+**’ to add a new **Variable substitution**. The variable name is “int_caller_id” defined on the REST message, so it needs to match here.

   Then drag the new **caller_id** input to the variable value. Here is the finished state.

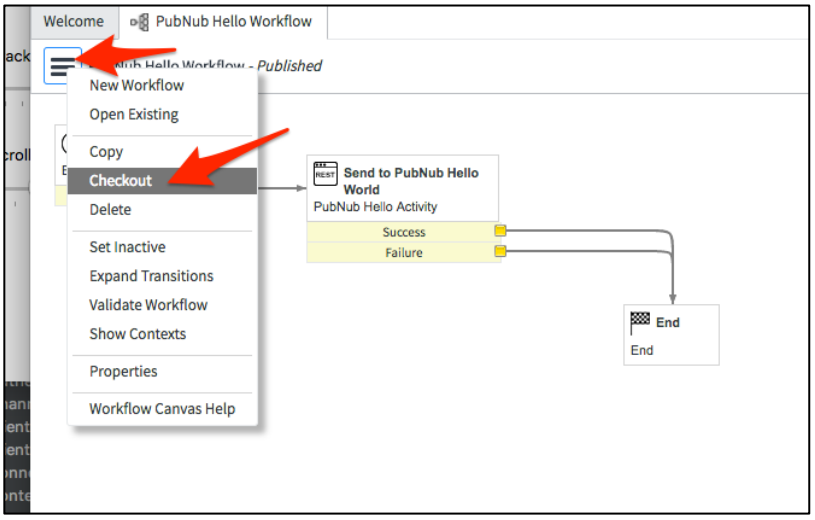
10. **Save** and **Publish** the REST Activity.

Use the new REST Activity in a Workflow

1. Open the PubNub Workflow previously created. You can find it in the **Workflow navigation tab** by typing “PubNub”.

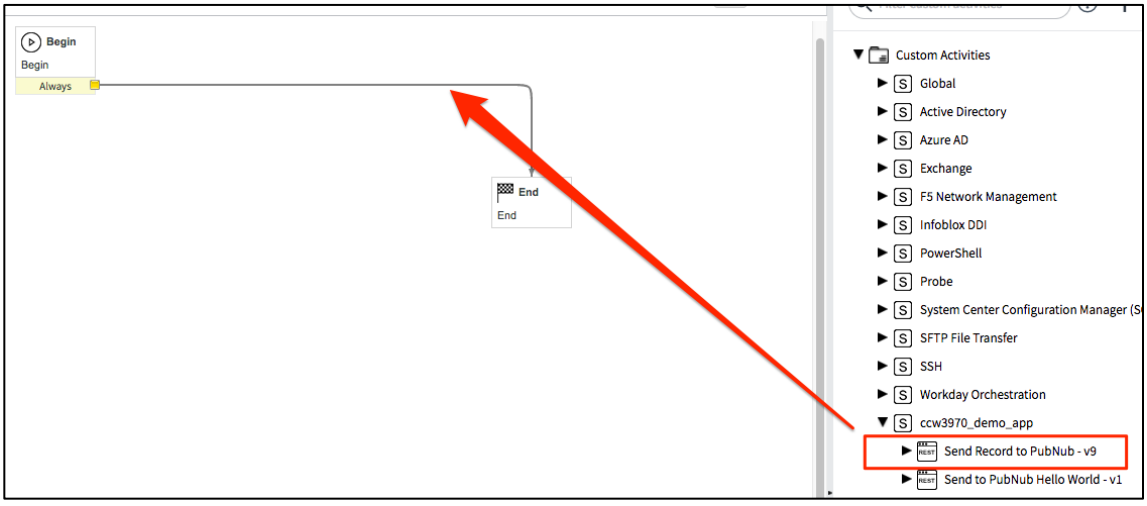


2. If necessary, **Checkout** the Workflow. (It may already be checked out from before)

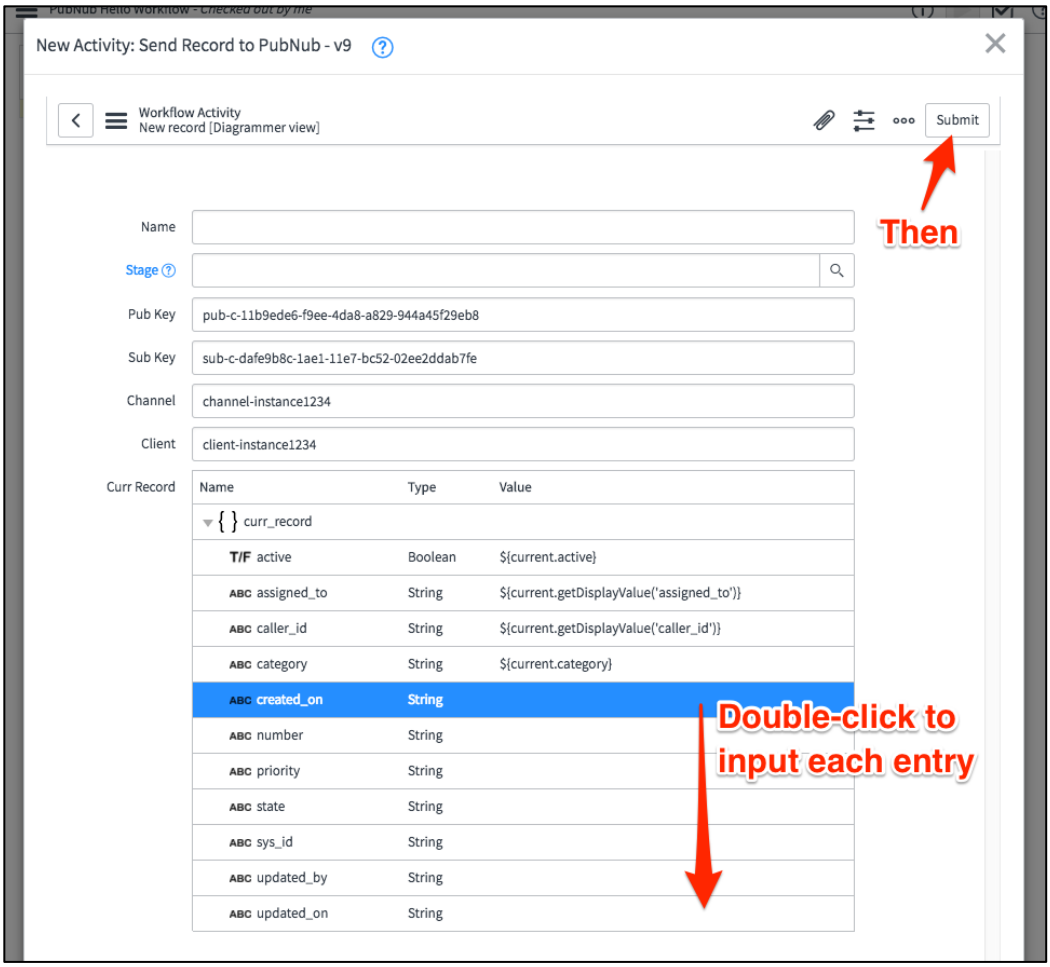


3. Hover and click the **X** icon to **delete** the Hello World Activity from the Workflow.

4. From the **Custom** tab, Drag the **Send Record to PubNub** Activity to the Workflow.



5. Now you need to reference the values from the **current** record (the Incident that was created), to bind them to the **Input variables** from the **REST Activity**.



Note the needed values can be copy/pasted one-by-one from the GitHub snippets repo (file **ccw3970_rest_activity_current_inputs.txt**) repo or directly from: https://raw.githubusercontent.com/CreatorCon17/CCW3970-Build-Debug-Outbound-REST-Snippets/master/ccw3970_rest_activity_current_inputs.txt
Then click **Submit**.

6. Test the Workflow again by creating a **new Incident** record. Be sure to fill out the various fields such as **Caller**, **Assigned To**, etc in order to see the values for those fields be extracted and sent to PubNub.

   Verify the message are received on your PubNub channel and that they're fully populated with data from the Incident record.

   Remember, you can view the **HTTP Outbound Request Log** to see what was sent from ServiceNow.

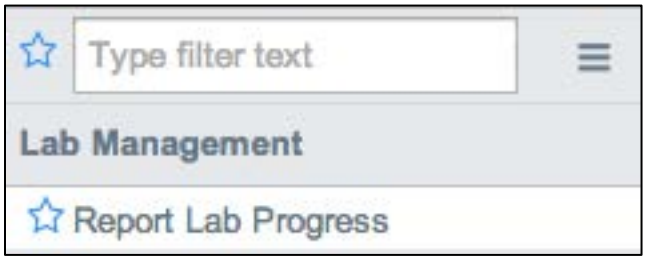
Catch Up

1. If you had problems with this lab and want to fast forward to the end of lab 3 to review the completed updates you can follow the same process, you followed in **Lab Setup** to create a branch from the **Lab3-end tag**. This will update your application to a state that you would be if you successfully completed Lab3.
2. In Studio, navigate to **Source Control > Create Branch**.
3. In the pop-up window, enter a branch name, then select **Lab3-end** from the **Create from Tag** menu, and click **Create Branch**.

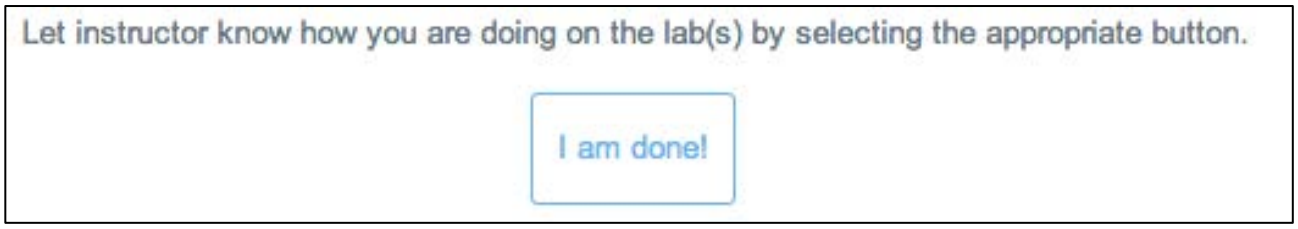
   Branch: **my-Lab3-end-branch**
   Create from Tag: **Lab3-end**

Progress Report

3. Navigate to **Lab Management> Report Lab Progress**.



4. Click **I am done!**



7.

--Intentionally Left Blank—

REFERENCE PAGES