

# 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

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.

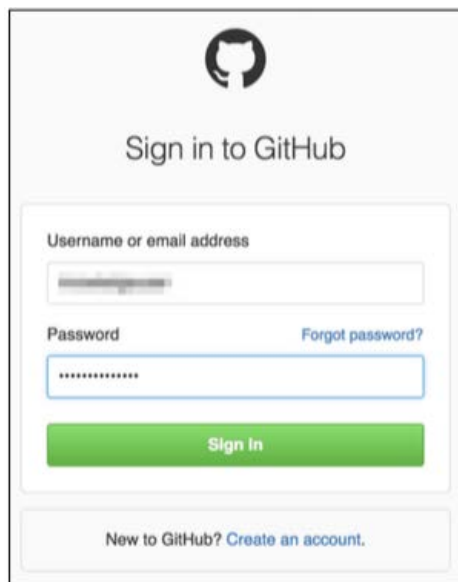
## Prerequisites

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

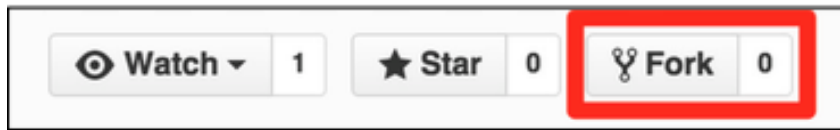
1. Log in to your GitHub account at:  
<https://github.com/login>

A screenshot of the GitHub login page. At the top is the GitHub logo (Octocat) and the text "Sign in to GitHub". Below this is a form with two input fields: "Username or email address" and "Password". The password field is masked with dots. To the right of the password field is a link that says "Forgot password?". Below the input fields is a green "Sign In" button. At the bottom of the form is a link that says "New to GitHub? Create an account."

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

## Lab Setup

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



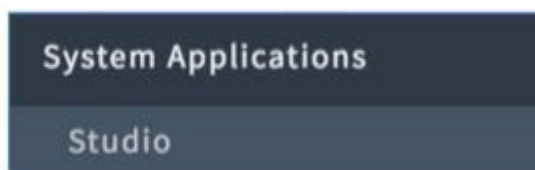
4. Note in the upper left that the repository has been copied to your account. You now have a copy of the lab material for reference after the conference!
5. Locate the HTTPS field and click the clipboard to the right. This action copies the URL in the clipboard.

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

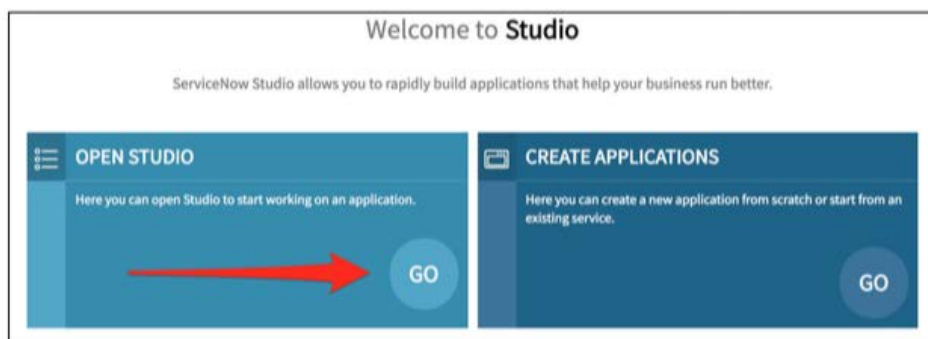


## Import the CCW3970 Application from Source Control

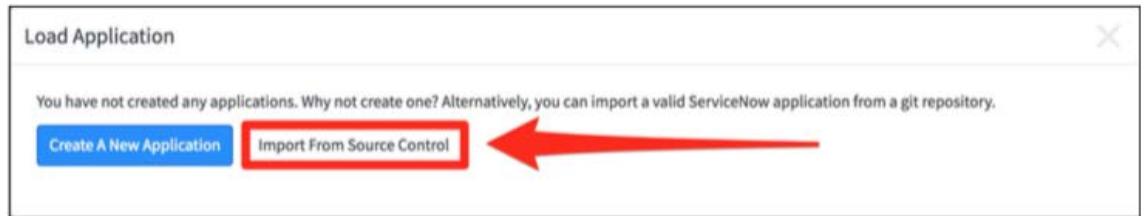
6. Log in to your lab instance with the **admin** credentials provided on the cover sheet of this document.
7. Navigate to System **Applications > Studio**.



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



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



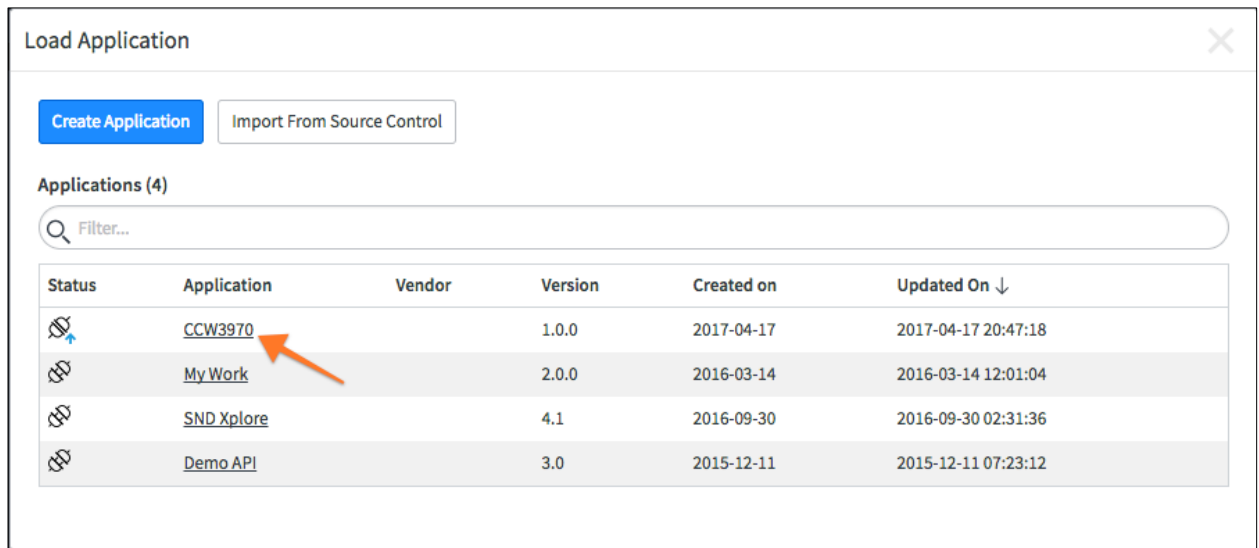
10. In the Import Application window, paste the URL copied in step 5 and provide your GitHub credentials. Click **Import**.

The screenshot shows an 'Import Application' dialog box. It contains the text: 'Importing an application from source control will result in a new application being created in this ServiceNow instance based on the remote repository you specify. The account credentials you supply must have read access to the remote repository. The remote repository you specify must contain a valid ServiceNow application. For more information on requirements refer to ServiceNow product documentation.' Below this text are three input fields: 'URL' (with a red asterisk icon), 'User name', and 'Password'. The 'URL' field is filled with 'https://github.com/CreatorCon17/CCW3970-Build-Debug-Outbound-REST-App.git'. The 'User name' and 'Password' fields are empty. At the bottom right, there are two buttons: 'Cancel' and 'Import'. The 'Import' button is highlighted.

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

The screenshot shows the 'Import Application' dialog box after a successful import. It displays a green checkmark and the word 'Success'. Below this is a blue progress bar. The text 'Successfully applied commit f46f7019e8090de045f9511f87eef8328927a1e6 from source control' is shown. At the bottom right, there is a 'Select Application' button, which is highlighted.

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



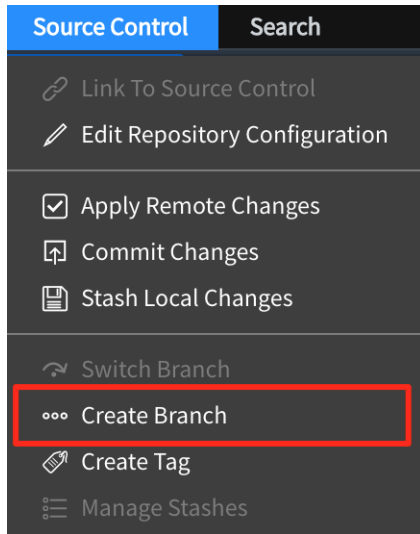
You've now successfully imported your forked version of the application for use in this workshop.

## 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**.

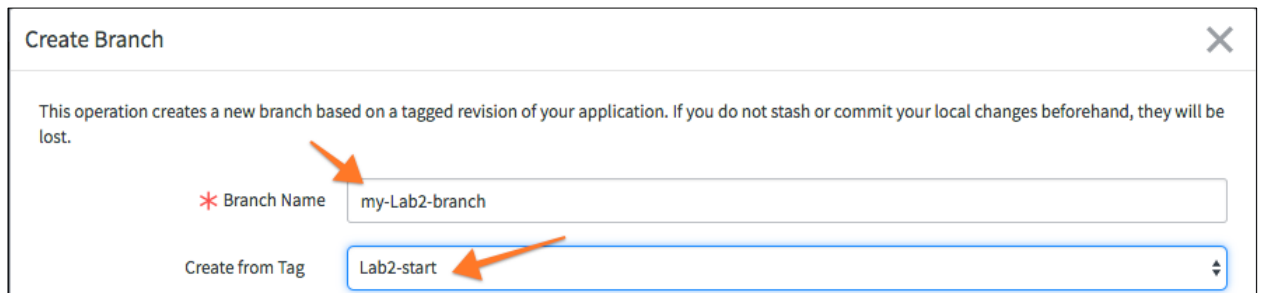
In Studio, navigate to Source Control > Create Branch.



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

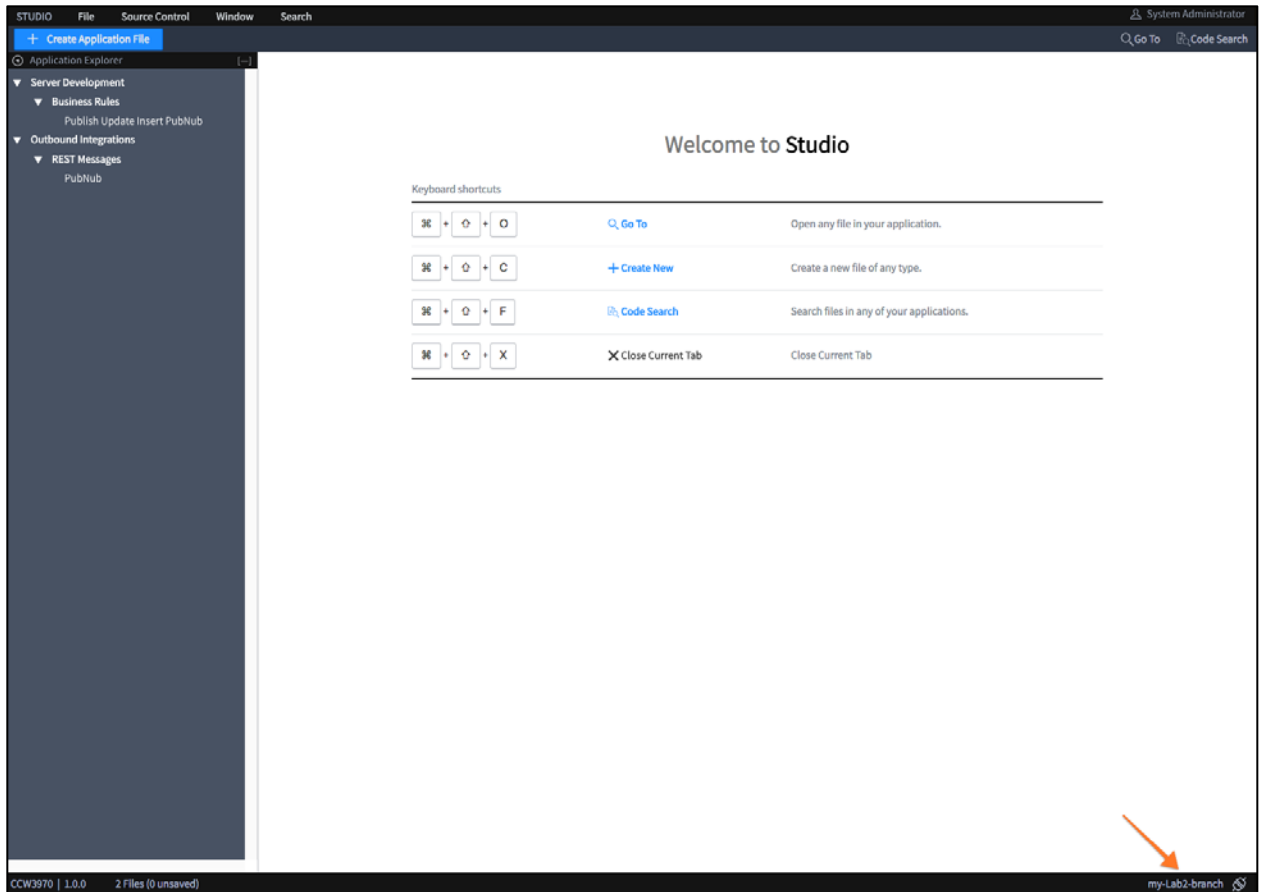
Branch: **my-Lab2-branch**

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).

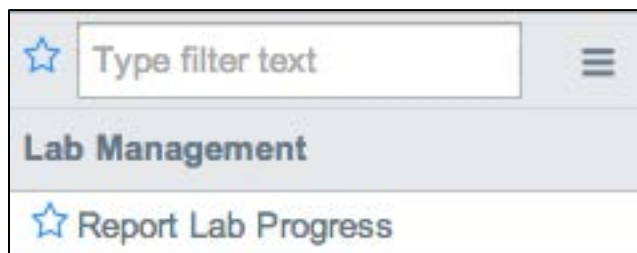




5. Lab setup is complete. You are now ready to start **Lab 1**.

## Progress Report

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



2. Click **I am done!**

Let instructor know how you are doing on the lab(s) by selecting the appropriate button.

I am done!

# Lab Goal

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

In this first **lab** you'll familiarize yourself with the **PubNub** 3<sup>rd</sup> 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.

Lab 1  
Publish  
Message  
with  
Postman  
"Hello  
World"

## 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 3<sup>rd</sup> party 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](https://www.pubnub.com/docs/pubnub-rest-api-documentation-pubnub-v1-via-post-post)

Specifically look at the Publish via POST operation.

The screenshot shows the PubNub REST API Documentation page. The left sidebar contains a navigation menu with sections like Overview, Publish / Subscribe, History, Channel Groups, PubNub Access Manager - PAM, and Presence. The main content area is titled 'PubNub REST API Documentation' and features a 'Select' dropdown. The selected endpoint is 'POST /publish/{pub\_key}/{sub\_key}/0/{channel}/{callback}{?store,uuid}' with the description 'Publish JSON to channel via POST'. Below this, an 'Example URI' is provided: 'POST https://pubsub.pubnub.com/publish/myPubKey/mySubKey/0/ch1/myCallback?store=0&uuid=db9c5e39-7c95-40f5-8d71-125765b6f561'. A 'URI Parameters' section lists: pub\_key (string, required), sub\_key (string, required), channel (string, required), callback (string, required), store (number, optional), auth (string, optional), meta (object, optional), and uuid (string, required). Below the parameters, the 'Request' section shows 'without JSONP', and the 'Headers' section lists 'Content-Type: application/json' and 'Location: /publish/myPubKey/mySubKey/ch1/0'. The 'Body' section shows a JSON object: {'message': 'All your base are belong to us.'}.

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.

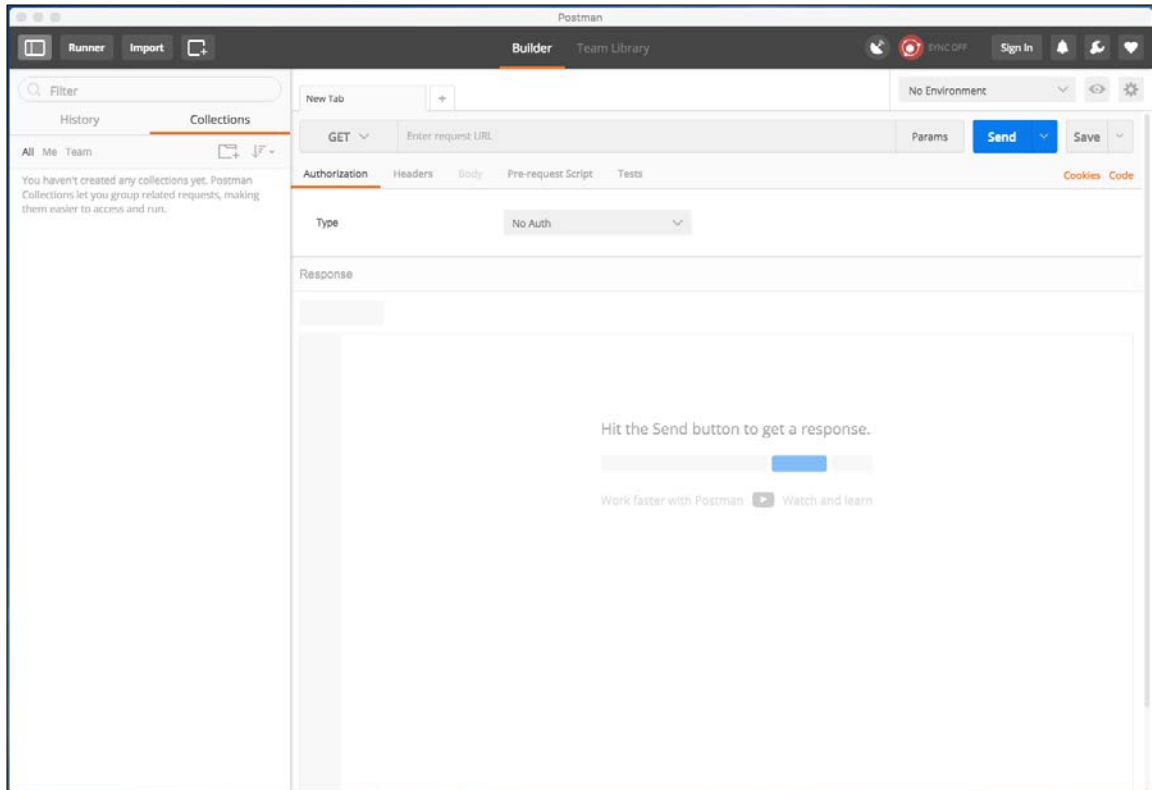
### Example URI

**POST** [https://pubsub.pubnub.com/publish/{pub\\_key}/{sub\\_key}/0/{channel}/0?store=1&uuid={client}](https://pubsub.pubnub.com/publish/{pub_key}/{sub_key}/0/{channel}/0?store=1&uuid={client})

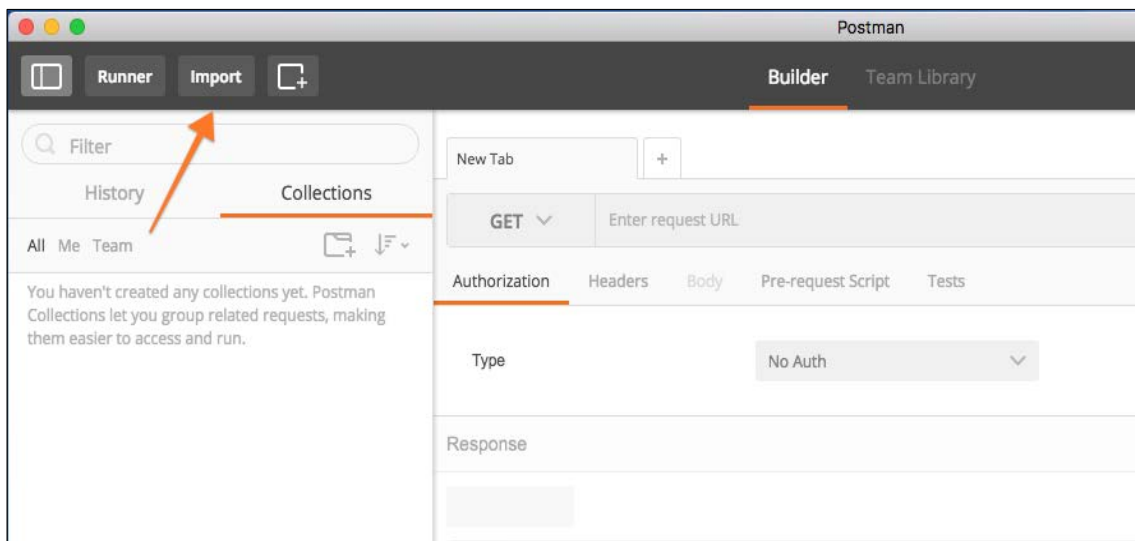
## Build and Send a Request to Publish with Postman

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.

IMPORT

×

Import a Postman Collection, Environment, data dump, curl command, or a RAML / WADL / Swagger(v1/v2) / Runscope file.

Import File

Import Folder

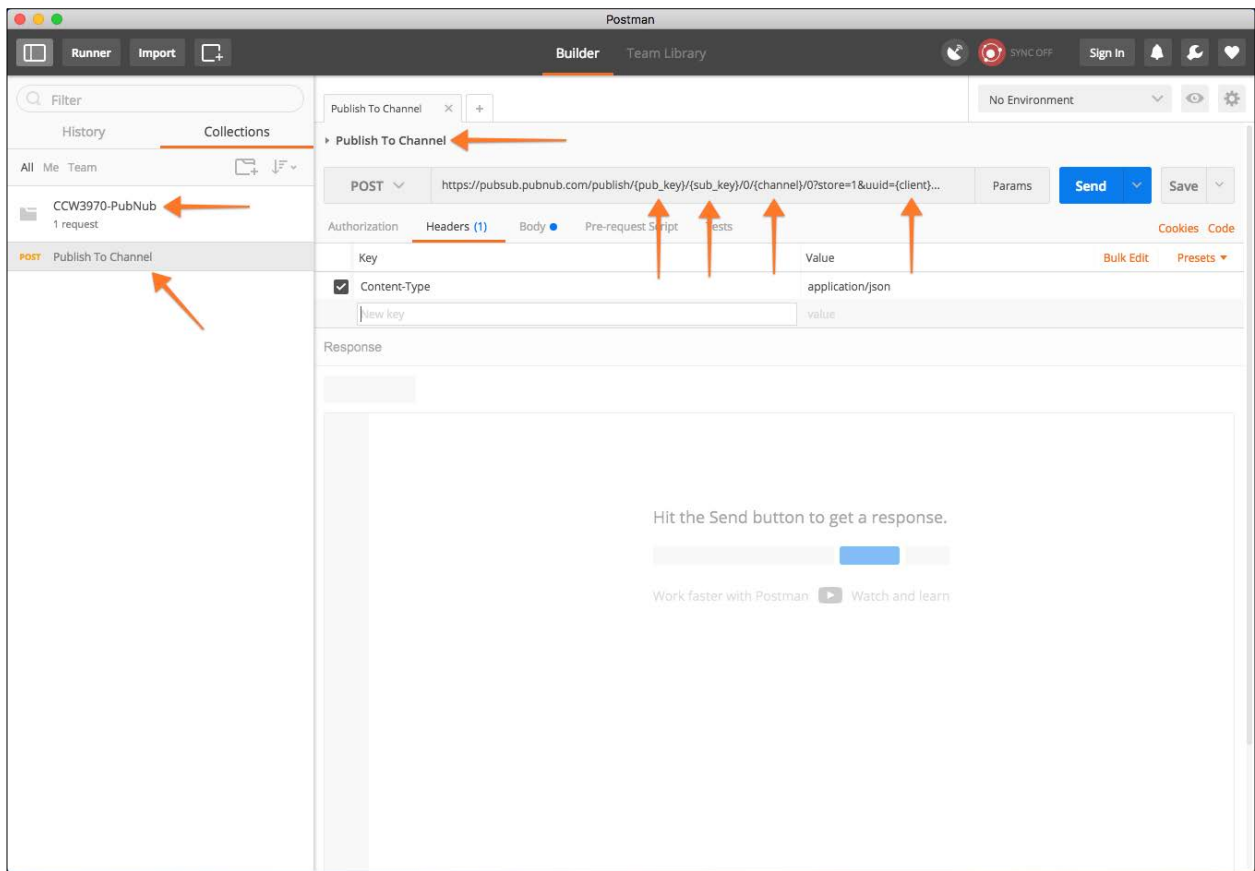
Import From Link

Paste Raw Text

https://www.getpostman.com/collections/a872c3162495ec77c946

Import

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

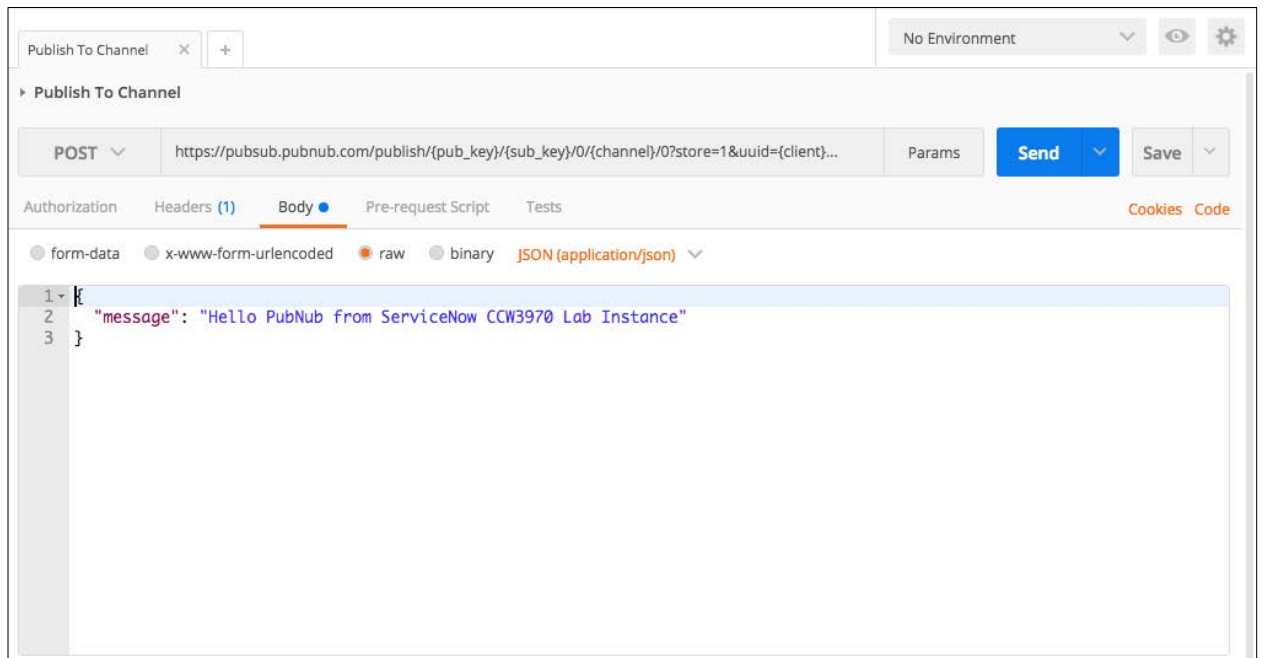
5. In the **CCW3970-PubNub** collection select the **Publish To Channel** operation.
- Replace the {pub\_key} parameter with the pub\_key provided in this lab doc.
  - Replace the {sub\_key} parameter with the sub\_key provided in this lab doc.
  - Replace the {client} parameter with your lab instance name (e.g., if your lab instance is lab1.service-now.com, replace the {client} param as 'lab1'.
  - 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`

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

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

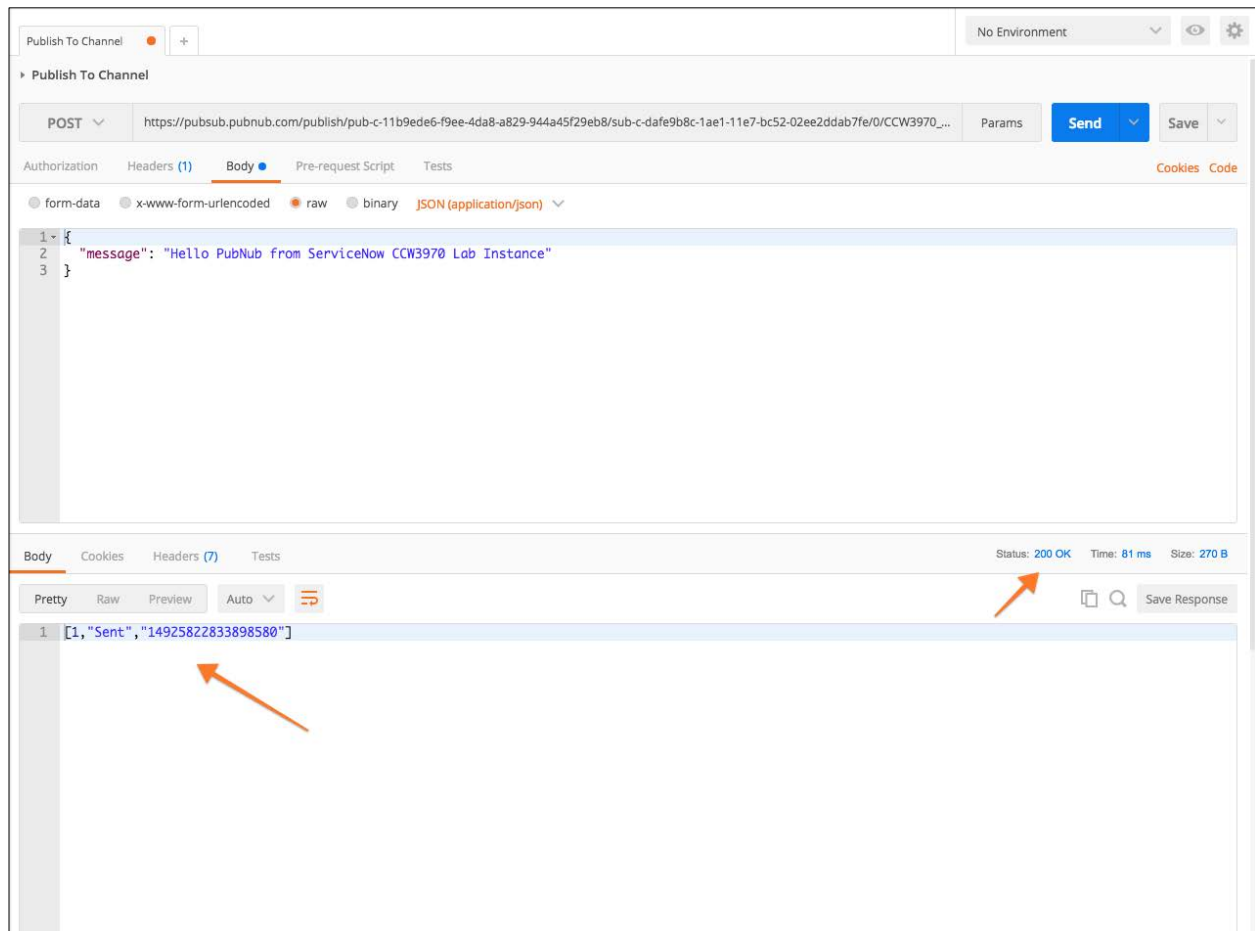


g. Click **Send** to send the HTTP Request.





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



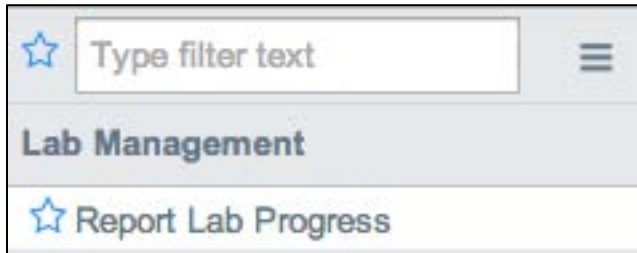
If you see **200 OK** you’ve now successfully published a message to PubNub using Postman from your local host. If you had errors check your URI and parameters or ask a Lab Guru for assistance.

This is an important step in building an integration because using a tool like Postman allows you to quickly familiarize yourself with a 3<sup>rd</sup> 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.

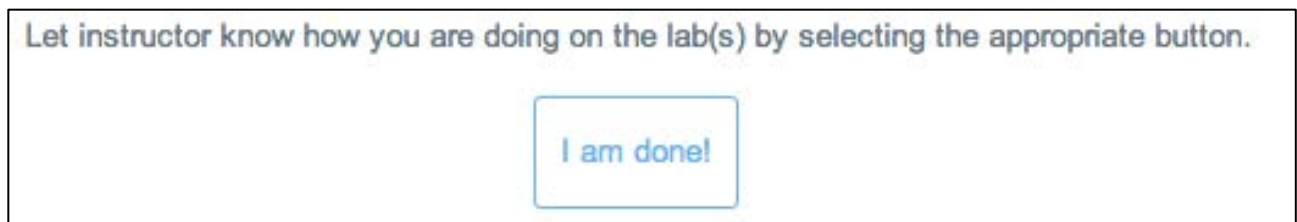
In the next lab you’ll use ServiceNow to issue HTTP requests to PubNub to publish messages from ServiceNow similar to how you used Postman in this **lab**.

## Progress Report

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



4. Click **I am done!**



# Lab Goal

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.

## Lab 2 Publish Message with ServiceNow

### 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**.

HTTP Method Publish Message

Name: Publish Message

HTTP method: POST

Endpoint: https://pubsub.pubnub.com/publish/\${pub\_key}/\${sub\_key}/0/\${channel}/0?store=1&\${client}

Authentication: HTTP Request

Authentication type: Inherit from parent

Use mutual authentication: ☐

Update Delete

Related Links

- [Auto-generate variables](#)
- [Preview Script Usage](#)
- [Set HTTP Log level](#)
- [Test](#)

Variable Substitutions (4)

Method = Publish Message	Name	Escape type	Test value
<input type="checkbox"/>	channel	No escaping	
<input type="checkbox"/>	client	No escaping	
<input type="checkbox"/>	pub_key	No escaping	
<input type="checkbox"/>	sub_key	No escaping	

#### 4. Populate Header “Content-Type” : “application/json”

Authentication HTTP Request

Use MID Server:

HTTP Headers

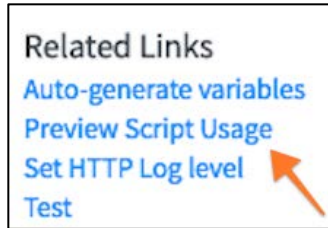
Name	Value
content-type	application/json
Insert a new row...	

HTTP Query Parameters

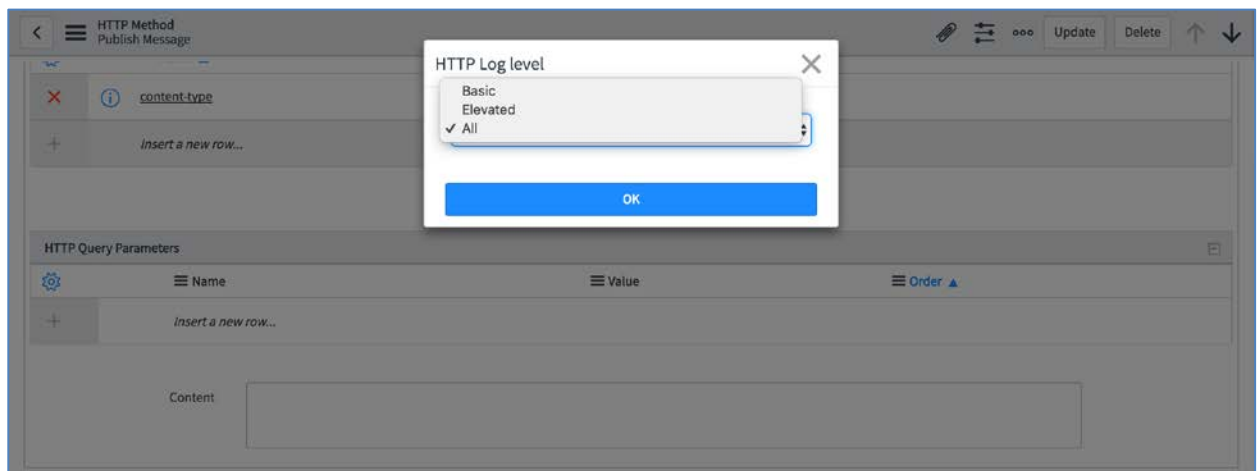
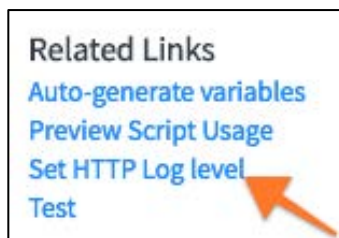
Name	Value	Order
Insert a new row...		

Content:

- 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.



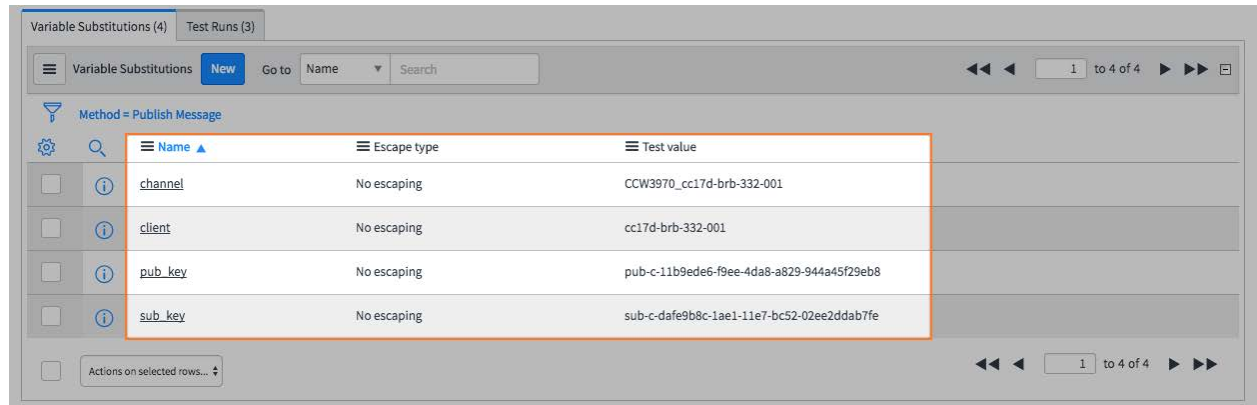
### PubNub Keys:

**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

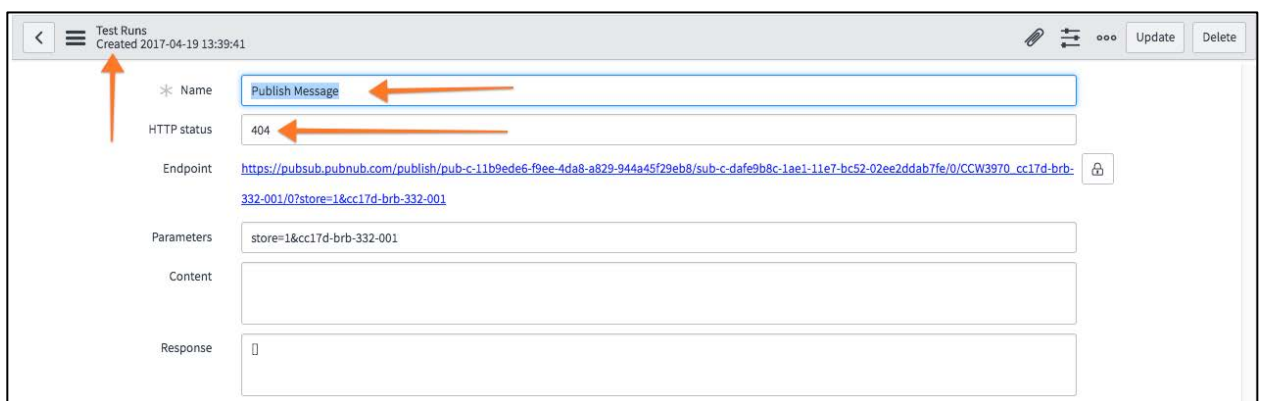
- d. **channel**, specify “CCW3970\_{you lab instance name}” (e.g, if your instance name iscc17d-brb-332-001, for the channel you would specify “CCW3970\_cc17d-brb-332-001”.



9. Click **Test**

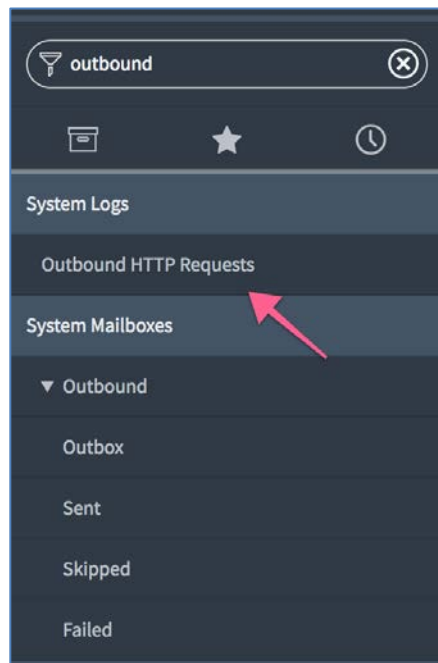


10. Verify the request fails with HTTP status **404**, (this is expected).



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**.

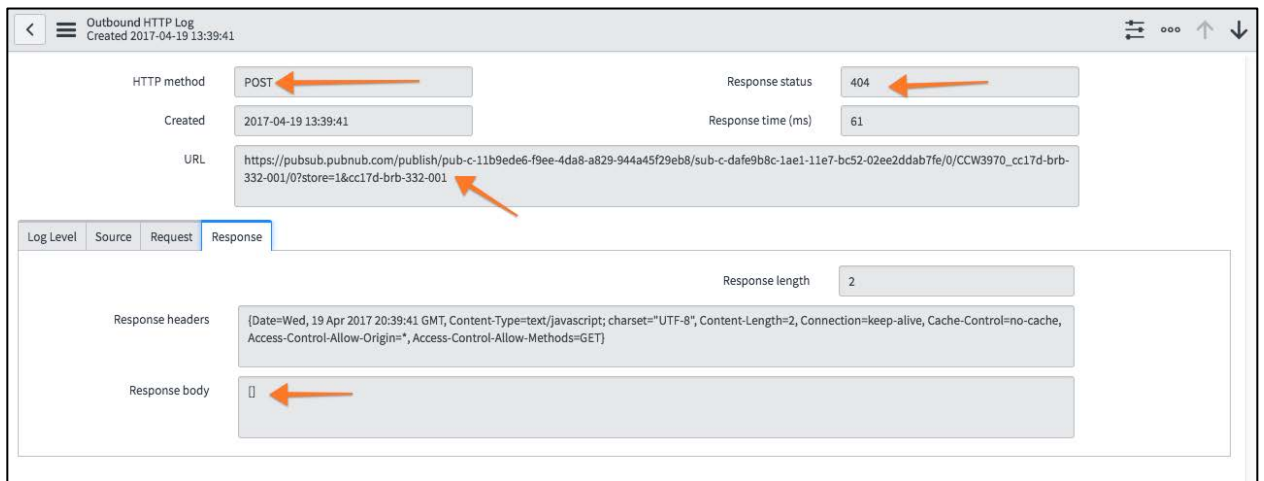


13. Review the list of recent outbound http requests.

A screenshot of the 'Outbound HTTP Logs' table in ServiceNow. The table has a header with columns: 'Created', 'Sequence', 'URL hostname', 'Response status', and 'Response time (ms)'. The first row of data shows a log entry with the following values: '2017-04-19 09:12:12', '4', 'pubsub.pubnub.com', '404', and '11'. An orange arrow points to the '404' value in the 'Response status' column. The table also includes a search bar, a filter icon, and a 'Go to' dropdown menu.

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).



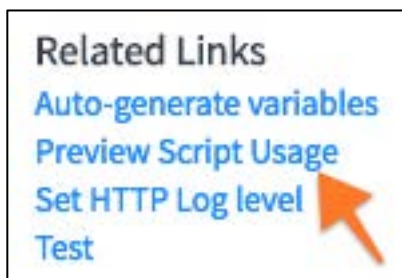
16. Go back to the **Publish Message** record in Studio and specify **content** of

```
{"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](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.

### PubNub Channel Log

Specify a channel to see a list of messages published to it streamed live:

Subscribed to channel: **CCW3970\_cc17d-brb-332-001**

Live Messages ordered newest on top:

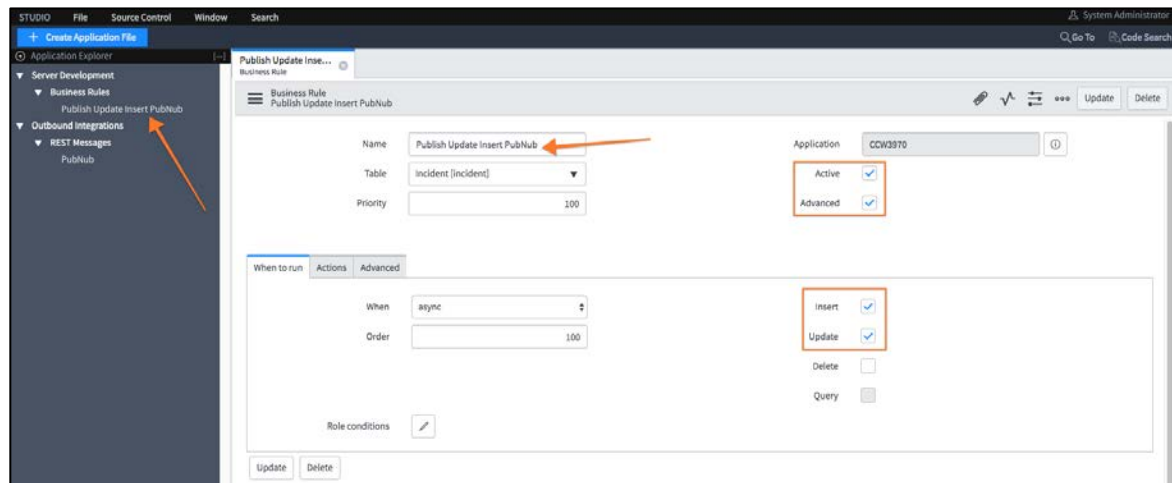
---

**channel:** CCW3970\_cc17d-brb-332-001  
**timetoken:** 14926349421604694  
**message:**  
{"message":"Hello PubNub from ServiceNow CCW3970 Lab Instance"}

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 **ajsync**. 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](https://raw.githubusercontent.com/CreatorCon17/CCW3970-Build-Debug-Outbound-REST-Snippets/master/ccw3970_advanced_business_rule_publish_updates_to_pubnub.js)

Business Rule  
Publish Update Insert PubNub

Name: Publish Update Insert PubNub

Table: Incident [Incident]

Priority: 100

Application: CCW3970

Active: ☒

Advanced: ☒

When to run: Actions Advanced

Condition:

Script

```

1  /*
2  Advanced Business Rule Script to publish message to PubNub Service Using RESTMessageV2 API to issue HTTP
3  Request
4  Publish message containing select subset of fields from the incident record that has been updated or
5  inserted
6  */
7  (function executeBusinessRule() {
8      try {
9          var instanceName = gs.getProperty('instance_name');
10         var req = new sn_ws.RESTMessageV2('PubNub', 'Publish Message');
11         req.setStringParameterNoEscape('pub_key', 'pub-c-11b9ede6-f9ee-4da8-a829-944a45f29eb8');
12         req.setStringParameterNoEscape('client', instanceName);
13         req.setStringParameterNoEscape('sub_key', 'sub-c-dafe9b8c-1ae1-11e7-bc52-02ee2ddab7fe');
14         req.setStringParameterNoEscape('channel', 'CCW3970_' + instanceName);
15
16         // Build a data transfer object representing the incident record to be sent as JSON to PubNub
17         var DT0Incident = {
18             'assigned_to': current.getDisplayValue('assigned_to'),
19             'category': current.getValue('category'),
20             'created_on': current.getValue('sys_created_on'),
21             'number': current.getValue('number'),
22             'priority': current.getDisplayValue('priority'),
23             'state': current.getDisplayValue('state'),
24             'sys_id': current.getValue('sys_id'),
25             'updated_by': current.getValue('sys_updated_by'),
26             'updated_on': current.getValue('sys_updated_on'),
27             'caller_id': current.getDisplayValue('caller_id'),

```

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.

```

1  /*
2  Advanced Business Rule Script to publish message to PubNub Service Using RESTMessageV2 API to issue HTTP Request
3  Publish message containing select subset of fields from the incident record that has been updated or inserted
4  */
5  (function executeBusinessRule() {
6  try {
7      var instanceName = gs.getProperty('instance_name');
8      var req = new sn_ws.RESTMessageV2('PubNub', 'Publish Message');
9      req.setStringParameterNoEscape('pub_key', 'pub-c-11b9ede6-f9ee-4da8-a829-944a45f29eb8');
10     req.setStringParameterNoEscape('client', instanceName);
11     req.setStringParameterNoEscape('sub_key', 'sub-c-dafe9b8c-1ae1-11e7-bc52-02ee2ddab7fe');
12     req.setStringParameterNoEscape('channel', 'CCW3970_' + instanceName);
13
14     // Build a data transfer object representing the incident record to be sent as JSON to PubNub
15     var DT0Incident = {
16         'assigned_to': current.getDisplayValue('assigned_to'),
17         'category': current.getValue('category'),
18         'created_on': current.getValue('sys_created_on'),
19         'number': current.getValue('number'),
20         'priority': current.getDisplayValue('priority'),
21         'state': current.getDisplayValue('state'),
22         'sys_id': current.getValue('sys_id'),
23         'updated_by': current.getValue('sys_updated_by'),
24         'updated_on': current.getValue('sys_updated_on'),
25         'caller_id': current.getDisplayValue('caller_id'),
26         'active': current.getValue('active')
27     };
28
29     // Convert DT0 to JSON string
30     var body = JSON.stringify(DT0Incident);
31     req.setRequestBody(body);
32
33     // Execute request
34     var res = req.execute();
35     var responseBody = res.getBody();
36     var httpStatus = res.getStatusCode();
37     gs.debug(httpStatus);
38
39 } catch (ex) {
40     var message = ex.getMessage();
41     gs.debug(message);
42 }
43 }
44 })();

```

**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.

## PubNub Channel Log

Specify a channel to see a list of messages published to it streamed live:

Subscribed to channel: **CCW3970\_bbarnsc1**

**Live Messages ordered newest on top:**

---

```

channel: CCW3970_bbarnsc1
timetoken: 14926372053231464
message:
{"assigned_to":"David Loo","category":"hardware","created_on":"2009-09-10 02:25:40","number":"INC20001","priority":"3 - Moderate","state":"Resolved","sys_id":"a1c35b5e0a0b6400afb27a259556a3","updated_by":"admin","updated_on":"2017-04-19 21:26:41","caller_id":"David Loo","active":"1"}

channel: CCW3970_bbarnsc1
timetoken: 14926371957634739
message:
{"assigned_to":"David Loo","category":"hardware","created_on":"2009-09-10 02:25:40","number":"INC20001","priority":"3 - Moderate","state":"Resolved","sys_id":"a1c35b5e0a0b6400afb27a259556a3","updated_by":"admin","updated_on":"2017-04-19 21:26:31","caller_id":"Sam Sorokin","active":"1"}

channel: CCW3970_bbarnsc1
timetoken: 14926371858632028
message:
{"assigned_to":"David Loo","category":"hardware","created_on":"2009-09-10 02:25:40","number":"INC20001","priority":"3 - Moderate","state":"Resolved","sys_id":"a1c35b5e0a0b6400afb27a259556a3","updated_by":"admin","updated_on":"2017-04-19 21:26:23","caller_id":"David Loo","active":"1"}

```

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.

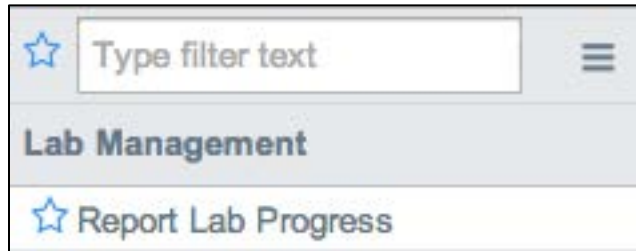
## Catch Up

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!**

Let instructor know how you are doing on the lab(s) by selecting the appropriate button.

[I am done!](#)



# REFERENCE PAGES