Solace Essentials – Activity Guide

This activity guide accompanies the Solace Essentials online course available on Solace Academy.

August 20, 2021

http://www.solace.com

Table of Contents

solace.

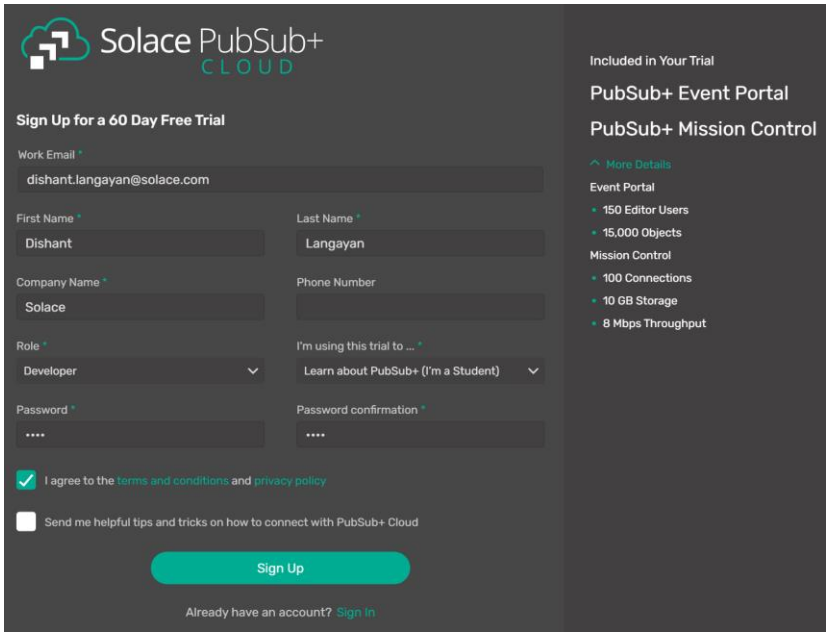
Exercise 1: PubSub+ Cloud Account

If you have previously tried PubSub+ Cloud using a **Trial** Account or part of an existing PubSub+ Cloud **Enterprise** Org account, then you can skip this exercise and simply confirm your access by signing into your account.

IMPORTANT NOTE: You can perform all hands-on activity using either the Trial or Enterprise accounts.

To create a PubSub+ Cloud account:

1. Open a browser and navigate to: https://console.solace.cloud/login/new-account
2. Enter the details requested on the form and click **Sign Up**.



3. To Sign into your account, navigate to: https://console.solace.cloud/login

Rideshare Use Case

Consider you are a solution architect, and your company has put you on a very important assignment to digitally transform their taxi service that is being outdated by modern rideshare companies like Uber and Lyft. Your company gave you a week to show them a quick design of the applications that would be involved in passengers requesting rides from their mobile apps and drivers can accepting rides, and the interactions & flow of events between those applications, so your development team can start coding and create a prototype.

You are a smart architect and know there is a tool out there called the Event Portal, through which you can mock up your designs and choreograph the event flow between applications in just a few hours, giving you rest of the week to binge watch your favourite show.

You get to work and use the below steps to complete the design:

1. You document the events, which applications will need to produce or consume
2. You add the schema to those events, so developers know what payload to add when producing or consuming events
3. You map the events to the applications and create a visual graph of the event flows

4. But to impress your company you take it a step further and generate the plumbing code for the applications that produce and consume your designed events.

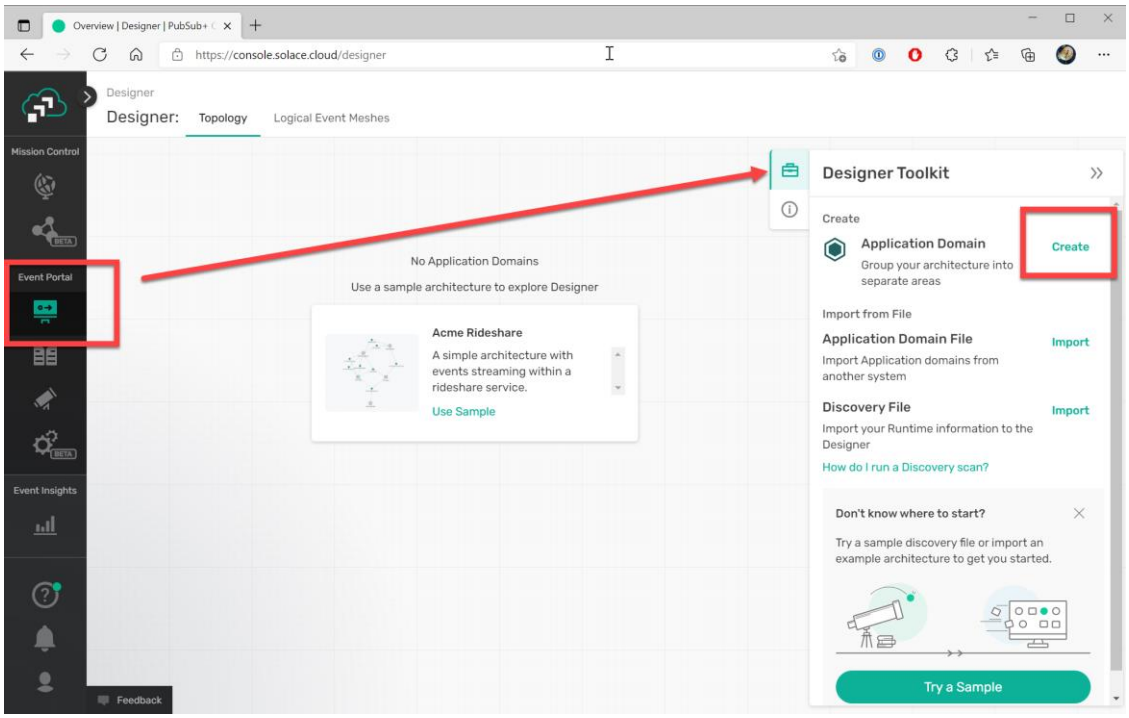
Exercise 2: Application Domain

When working on a new event-driven architecture, it is important to organize your applications, events, and payload schemas by setting up an application domain. This enables you to keep your work separate from other lines-of-businesses in your company that maybe be on the same Enterprise org account, but still allows you to share events across other application domains if the need arise.

Create the application domain

From the left menu options on the PubSub+ Cloud console, select the Event Designer.

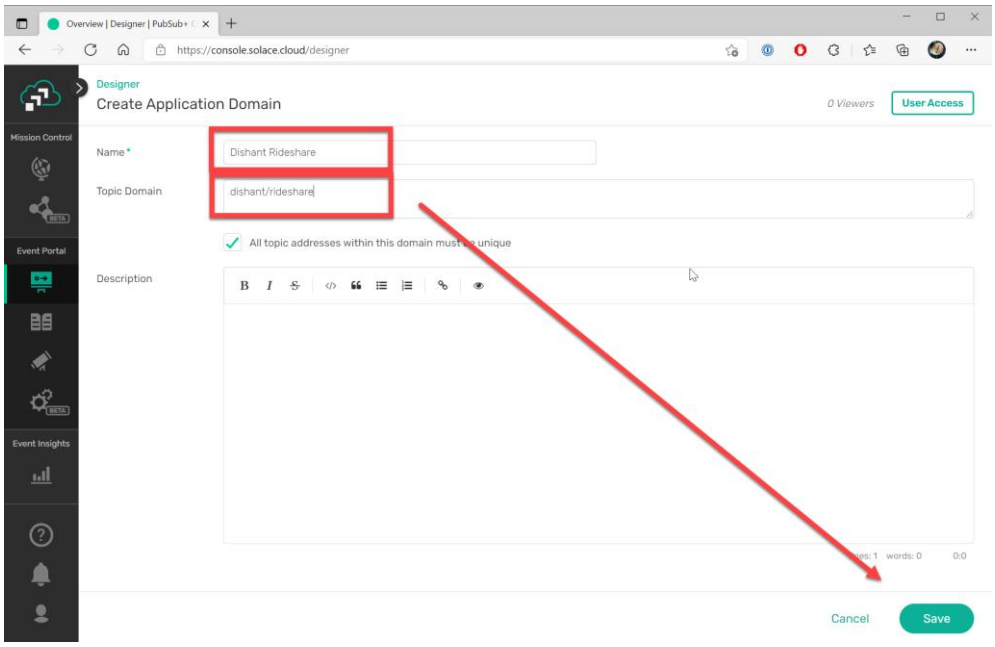
1. Select the **Create** button



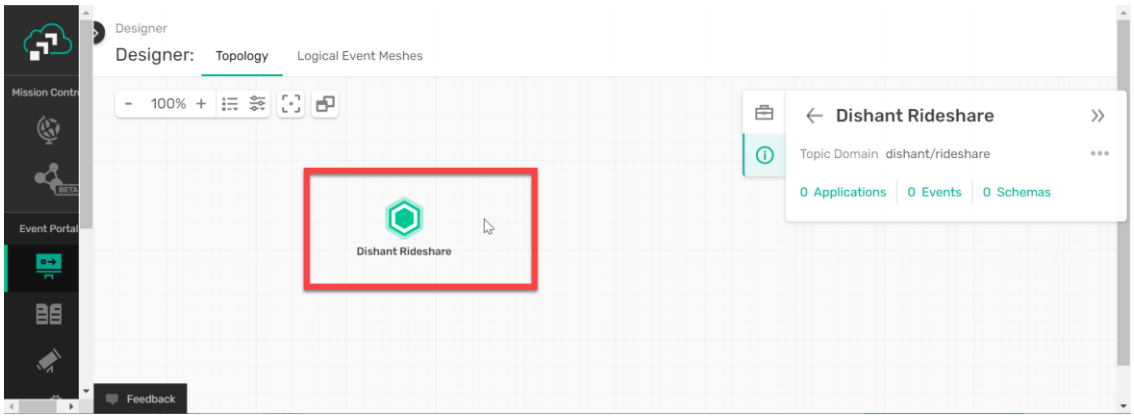
2. Enter the application domain name as: **<YOUR_NAME> Rideshare**

   and the root topic as: **<YOUR_NAME>/rideshare**

solace.



3. Select the **Save** button.
4. Double click on your newly created application domain to open it.



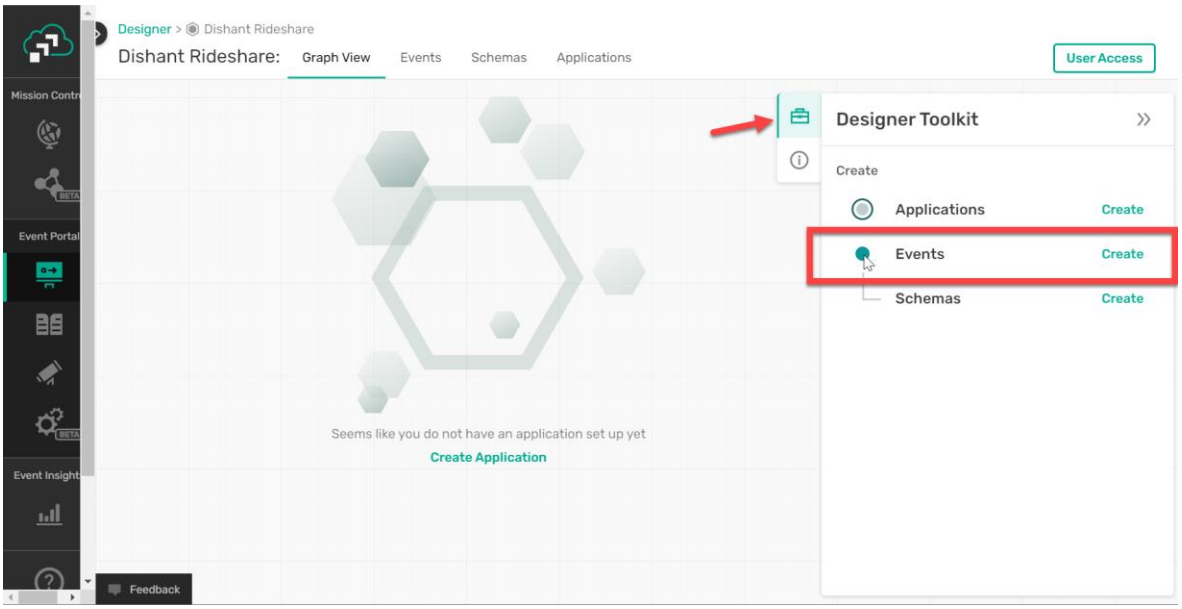
You have now created your application domain and can start adding events, schemas, and applications to your design.

Exercise 3: Events, Topics, and Schemas

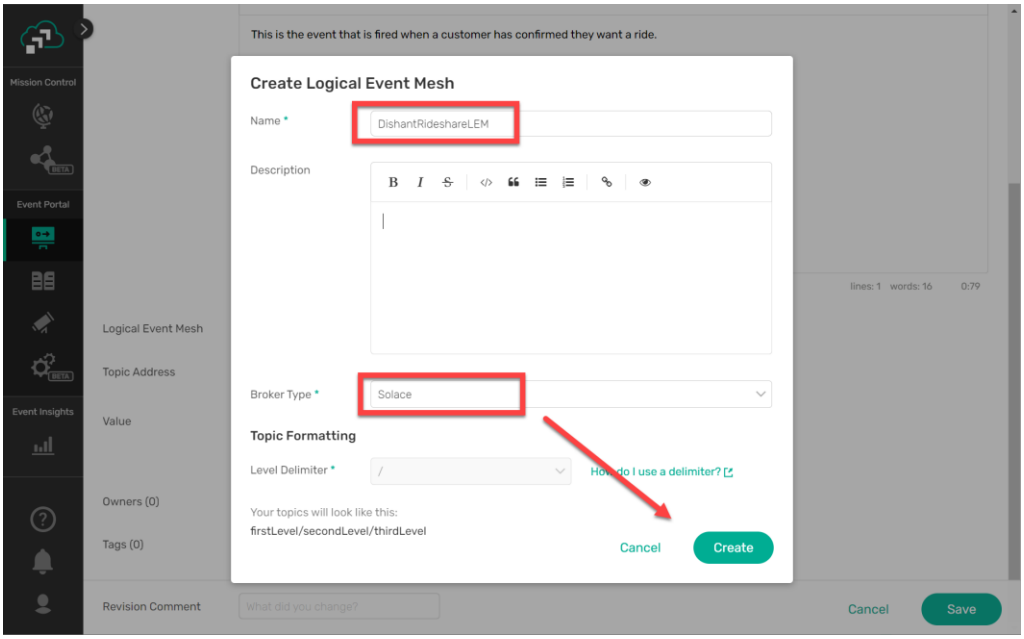
In this exercise you will document an event, by creating an event object, giving it a name, topics on which it will be published, and associate a payload schema with the event. There are multiple ways to create events and schemas. This exercise will walk you through creating them independently, so they can then be associated with an application. Another approached show later is to create the event and schema, when creating the application.

Creating an Event

1. Select the **Event** button from the Designer Toolkit within your application domain



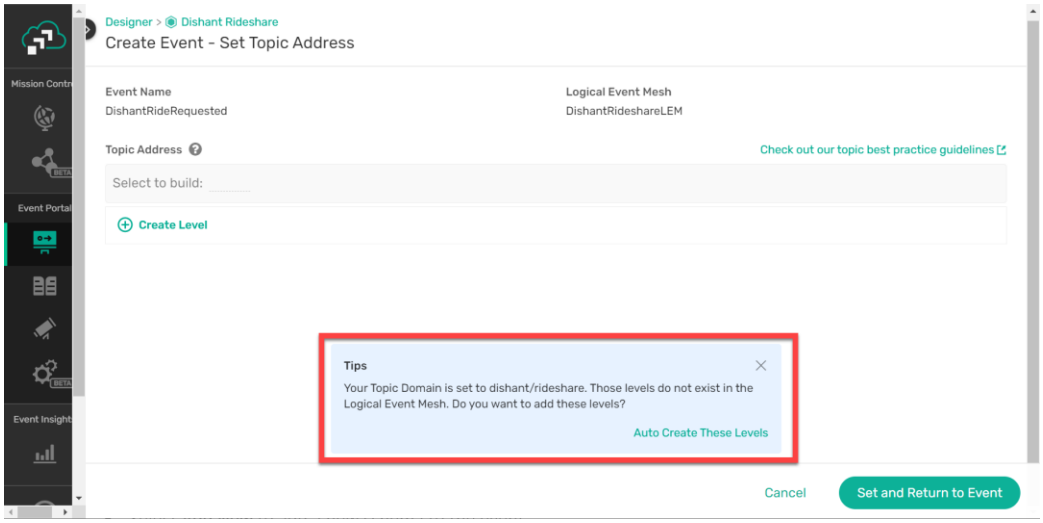
2. Enter the following:
    - **Name:** <YOUR_NAME>RideRequested
    - **Description:** This is the event that is fired when a customer has confirmed they want a ride.
3. Create a new Logical Event Mesh (LEM): **<YOUR_NAME>RideshareLEM** using the **Add** link. **NOTE**: you only have to do this once and for all future events in our domain we will select the newly create LEM.



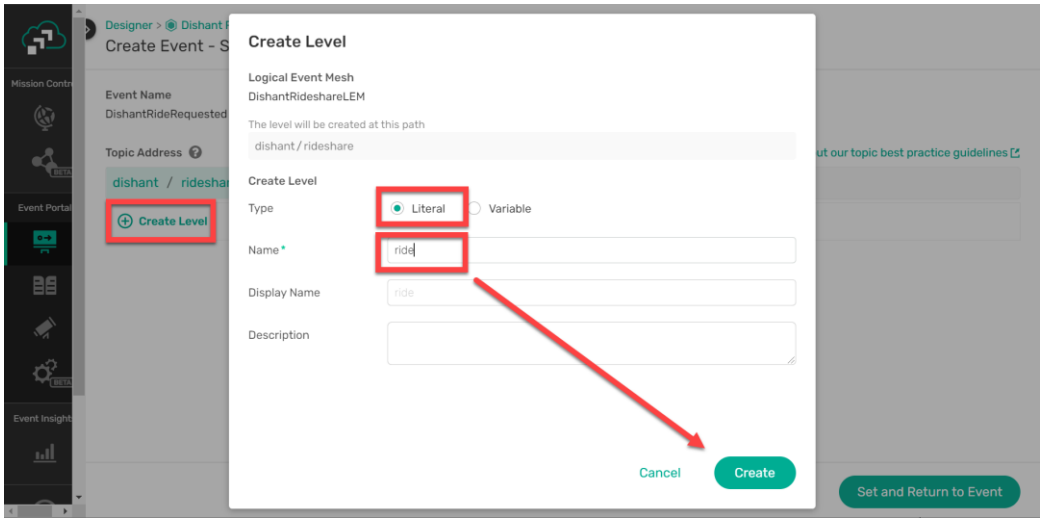
4. Select **Set Topic Address**, to set the topic on which this event will be published on.
    - The topic to set is:

```
<YOUR_NAME>/rideshare/ride/requested/2.0.1/{start_latitude}/{start_longitude}/json/{customer_id}/{request_id}
```

    - **NOTE:** The **<YOUR_NAME>/rideshare** is your Topic Domain set when creating the Application Domain. You can use the **Auto Create These Levels** link on the Tip screen to add the Topic Domain.
    - Remember: Levels in a topic in Solace are separated by a "/" character. In the above example, **ride** is a level and **requested** is another level. Each of these levels much be created to set the Topic Address.

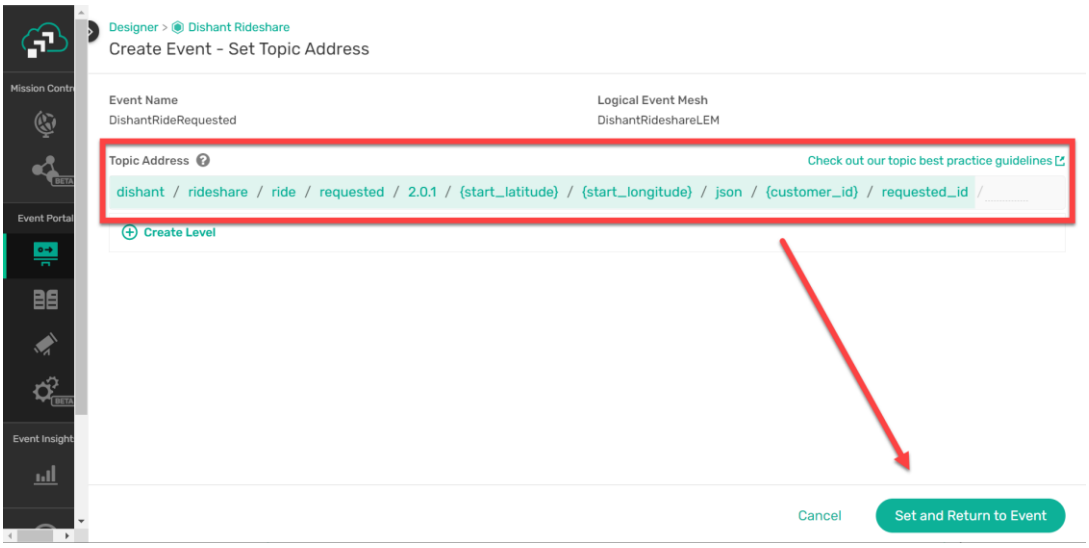


    - Next, select the Create level button to construct the remain part of our topic mentioned above:

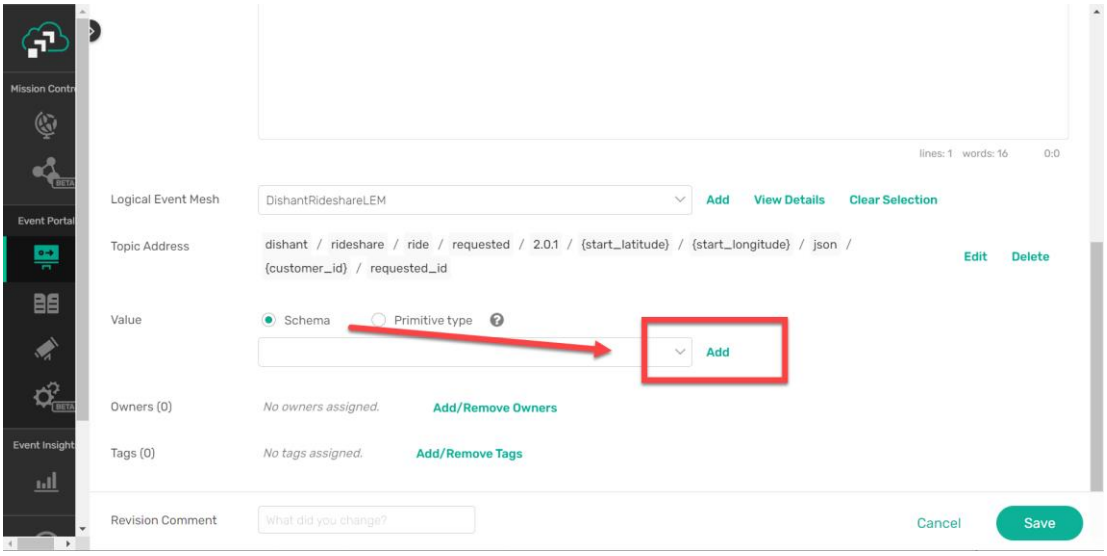


    - NOTE: A literal value is a fixed part of our topic that does not change for the event. Variables are values that can dynamically change for the same event. In our case the Longitude & Latitude are value that change dynamically for this event and will be set as variable.
    - NOTE: in our topic mentioned above anything within the "{}" brackets is a Variable. When entering the string you can exclude the "{}" which will be set when the type Variable is selected.

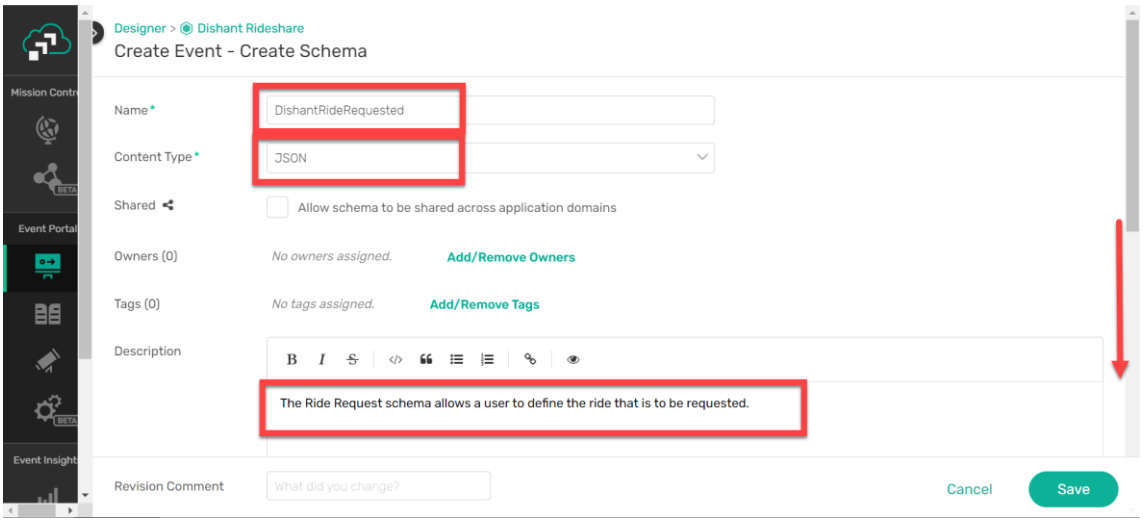
5. Once all the Topic levels have been created click the **Set and Return to Event** button.



6. Next add Schema for the event using the **Add** link for the Value field.



7. Enter the following:
    - **Name: <YOUR_NAME>**RideRequested
    - **Description:** The Ride Request schema allows a user to define the ride that is to be requested.
    - **Content Type:** JSON
    - **Optional:** you can optionally set yourself or another team member as the owner of the Schema as well as set Tags to allow for easier searching and filtering of the schema.
    - **Content:** copy and paste the contents of the **RideRequested.json** file or use the import button to import the schema from the file.
        i. **NOTE:** schema files are available along with the course online on Solace Academy.



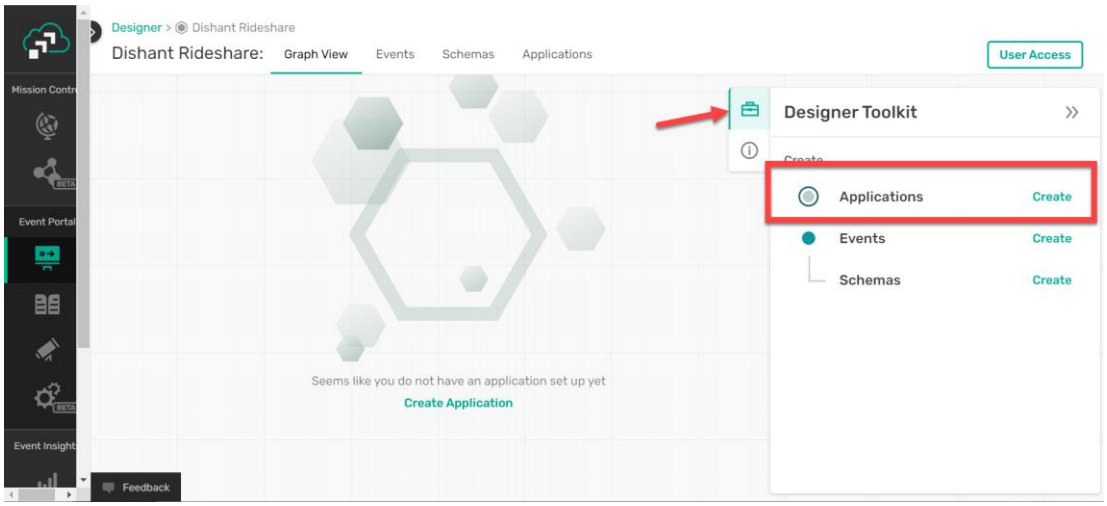
8. Click **Save** and **Save** again.

Exercise 4: Passenger App

Here we will create the passenger app in the designer, which will produce a ride requested event. You can create events either independently from the application or while create the application. In this exercise you will create the **RideAccepted** event while creating the application and associate the **RideRequested** event created in the previous exercise.

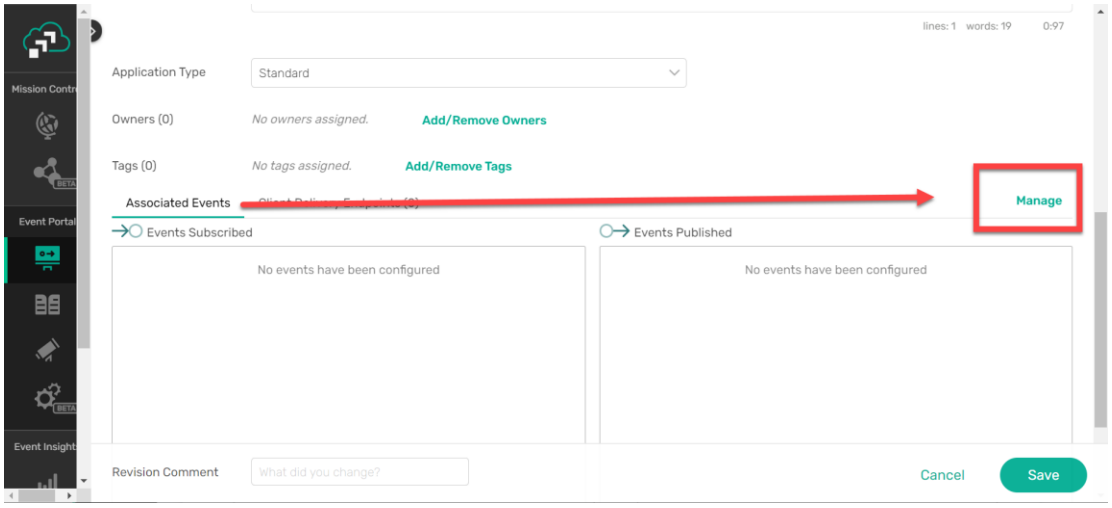
Create an application

1. Select the **Application** button

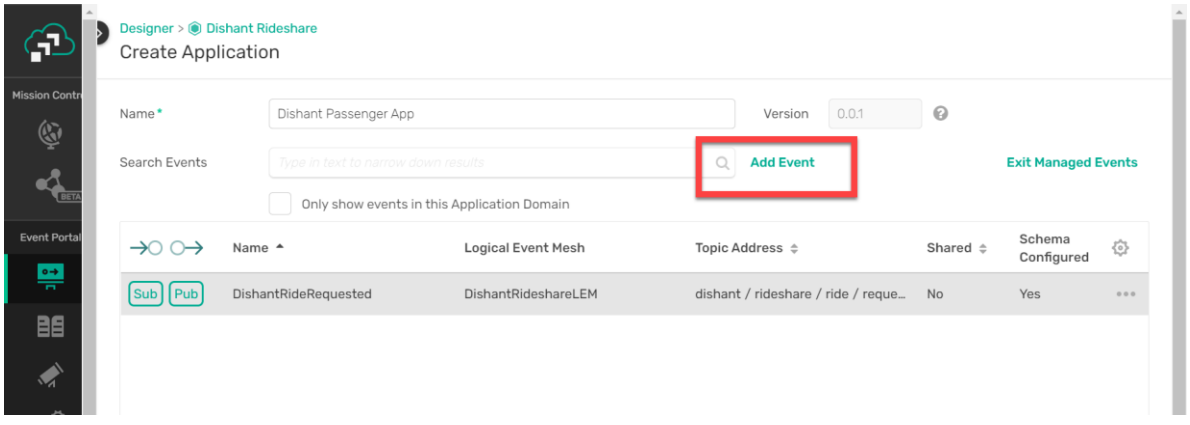


2. Enter the following:
    - **Name:** <YOUR_NAME> Passenger App
    - **Description:** This is the app that a user of acme rideshare uses in order to obtain rides from willing drivers.

    And select **Manage Events** under the **Associated Events** sections.

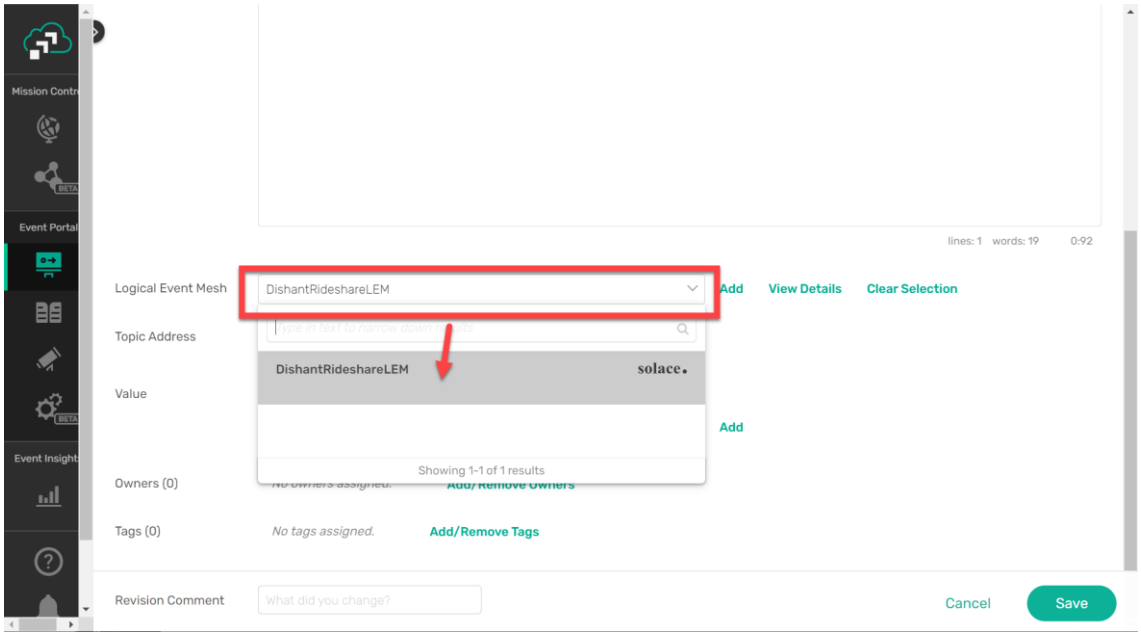


3. Select **Add Event**



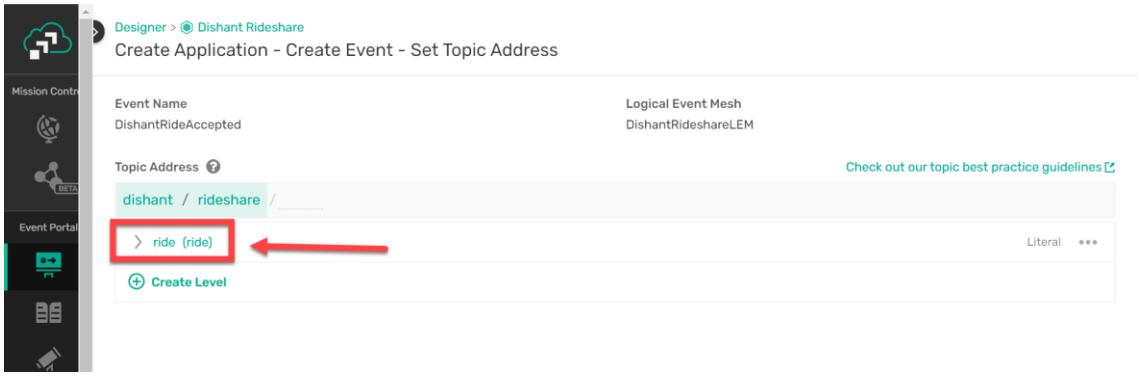
solace.

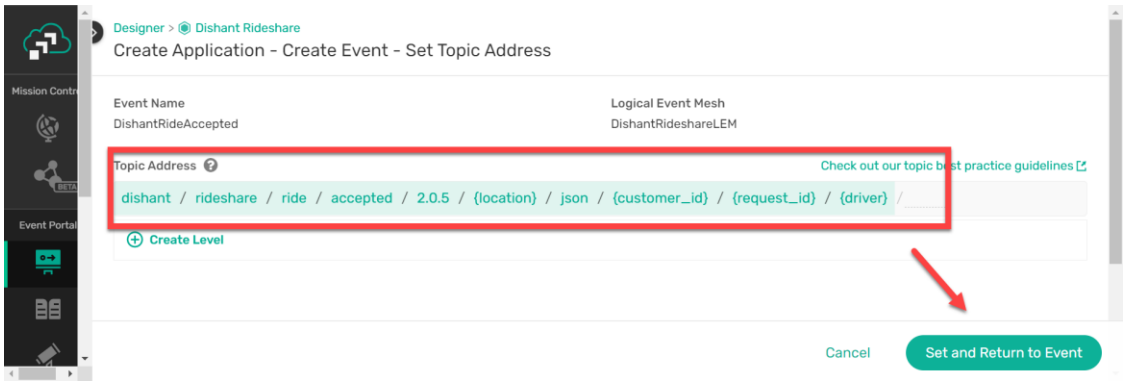
4. Enter the following:
    - **Name: <YOUR_NAME>**RideAccepted
    - **Description:** This is the event which is fired when a ride has been accepted and committed to by a driver.
    - **Logical Event Mesh:** select the Logical Event Mesh created in the previous exercise.



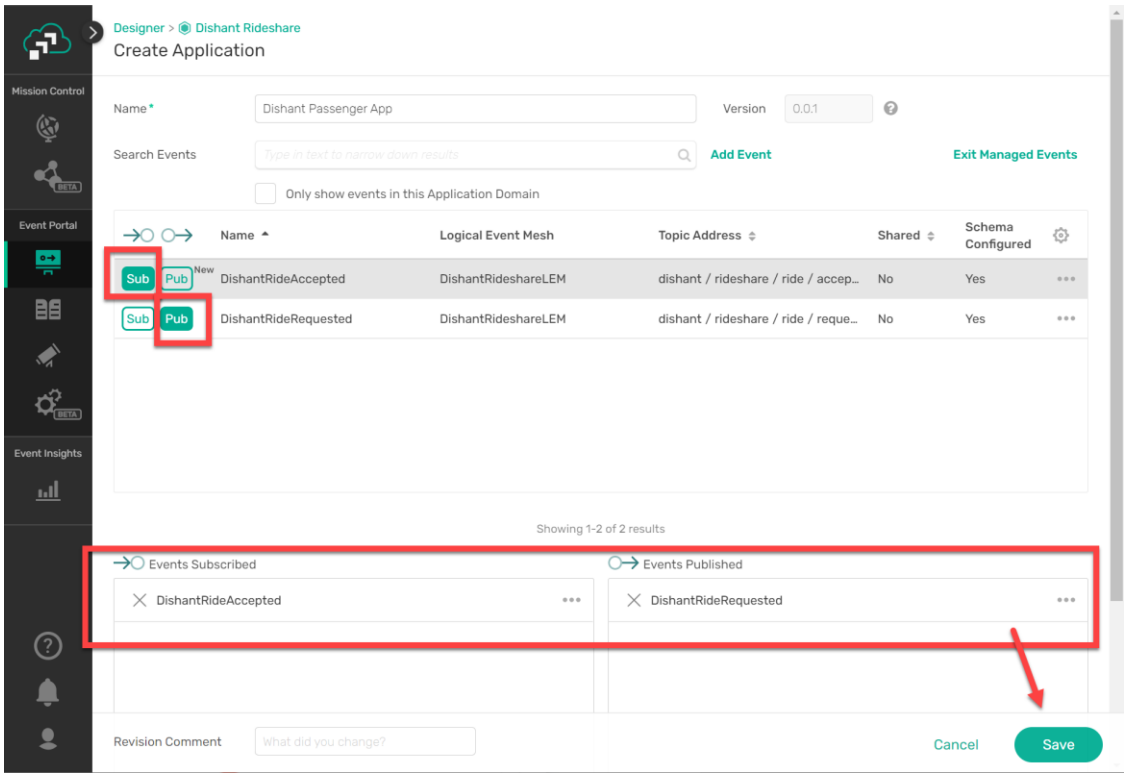
    - **Topic:** use the Set Topic Address to set the below topic. NOTE: you can select any levels we created before by selecting it (as shown below), instead of creating the level again.

```
<YOUR_NAME>/rideshare/ride/accepted/2.0.5/{location}/json/{customer_id}/{request_id}/{driver}
```

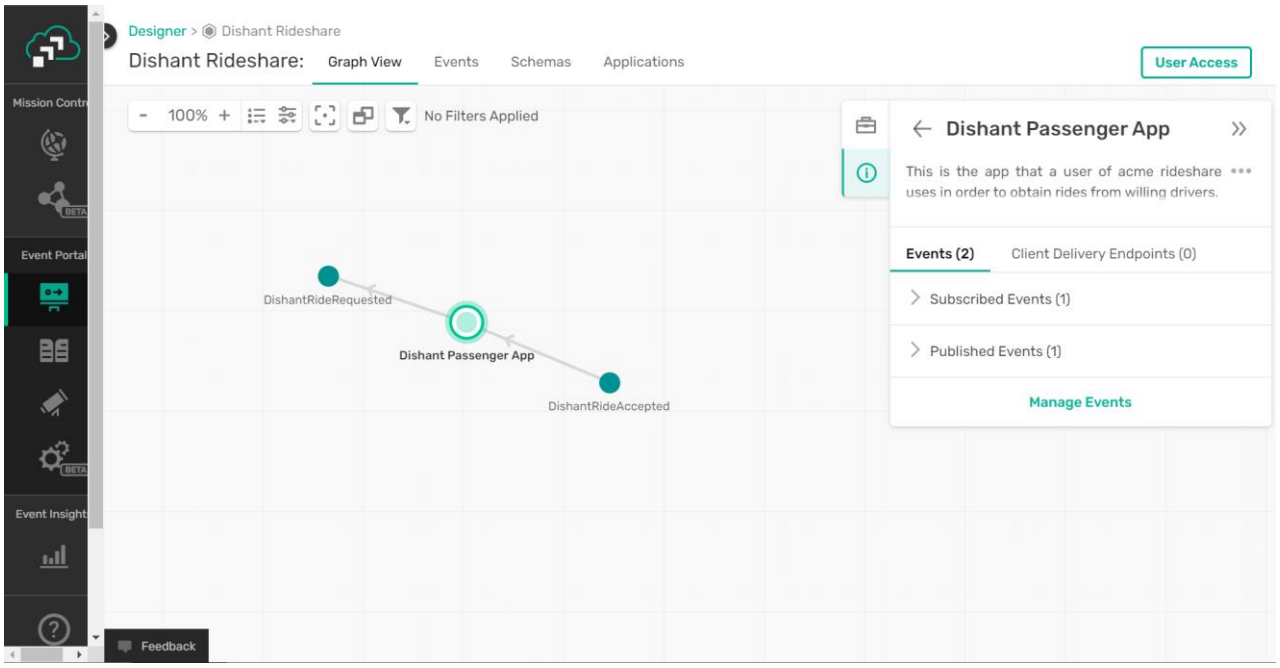




5. Select **Add New** to add a new schema to the event and enter the following:
    - **Name: <YOUR_NAME>**DriverRideAccept
    - **Description:** Data format that represents the information required for a driver to accept a ride.
    - **Content Type:** JSON
    - **Content:** copy and paste the contents of the **DriverRideAccept.json** file or use the import button to import the schema from the file.
6. Select **Save** and **Save**.
7. Select the **Sub** button for the **RideAccepted** and **Pub** button for **RideRequested** events. Verify the events appear in the Events Published and Subscribed list, and then select **Save**.



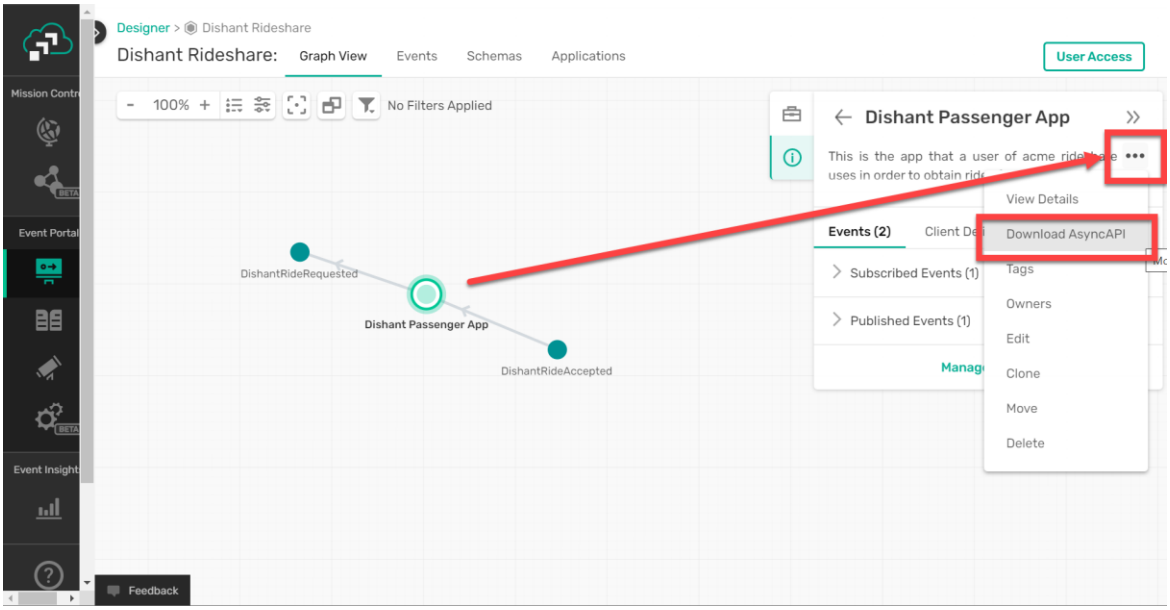
You have now created an application at design time and associated the events to that application.



Exercise 5: Generating AsyncAPI Schemas

In this exercise you will generate the AsyncAPI schema for the Passenger App created in the previous exercise.

1. Select the Passenger App icon from your Application Domain workspace and select the 3 dots from the Application overview panel.



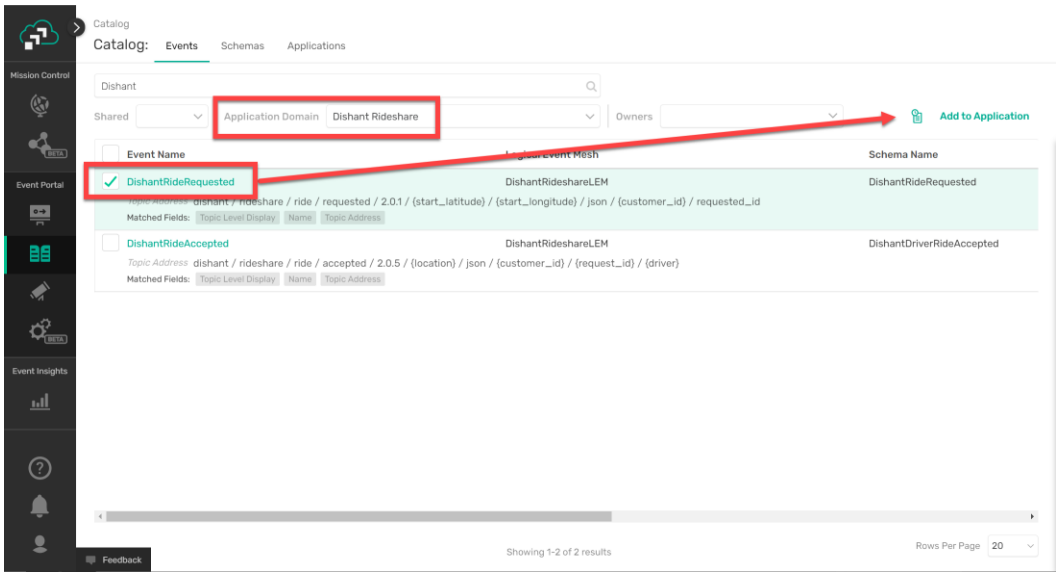
2. Select **Download AsyncAPI** from the menu options
3. Select your either JSON or YAML file format and click **Download** button. This will download a AsyncAPI schema file for your application to your local computer.

solace.

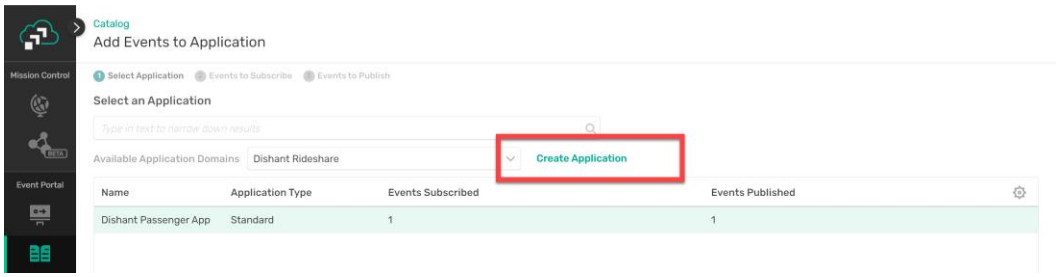
Exercise 6: Creating a new Application from the Event Catalog

In this exercise you will search for the **RideRequested** event you created previously and add it to a new **DriverManagementApp** application.

1. Select the Event Catalog from the PubSub+ Cloud Console menu
2. Search for all events beginning with your name or search by your Application Domain, select the **<YOUR_NAME>RideRequested** event from the search result, and then select **Add to Application**.

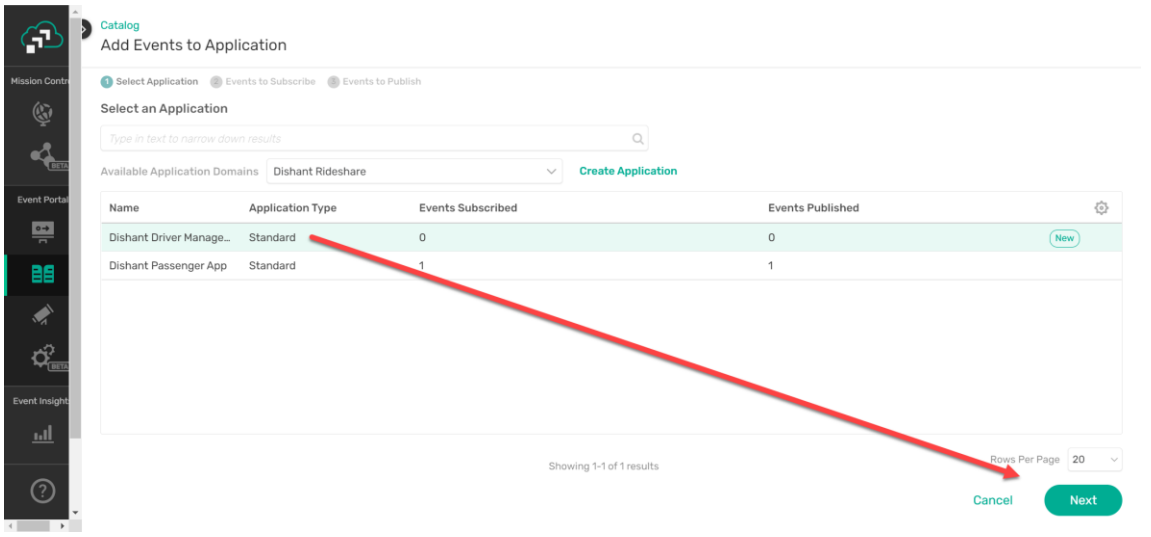


3. Select **Create Application**

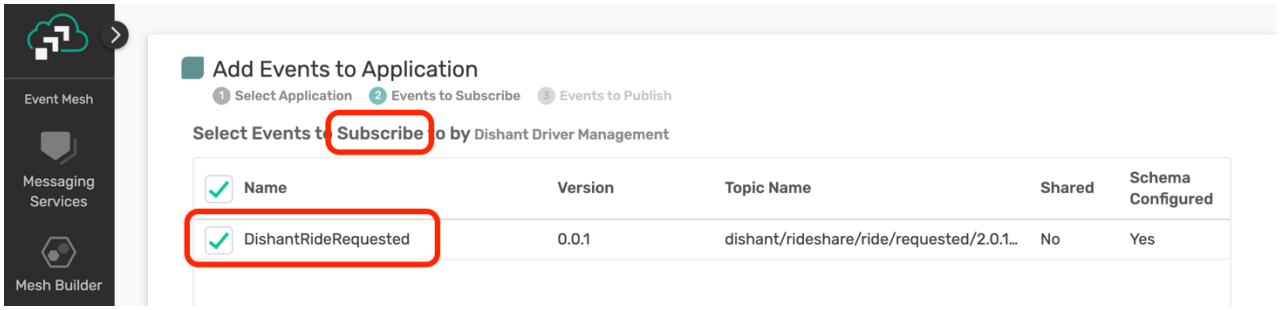


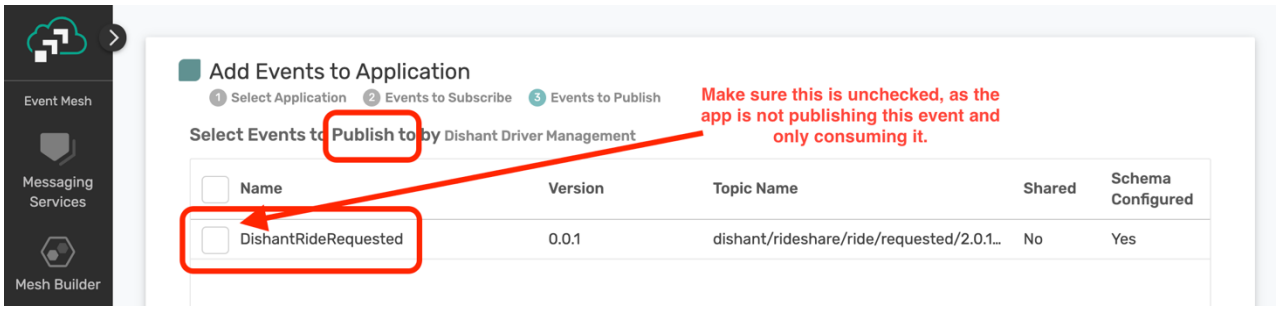
4. Enter the following for the application:
   a. **Name:** <YOUR_NAME> Driver Management
   b. **Description:** This application controls which rides need to be fulfilled, which are in route, and completed
   c. And click the **Save** button.
5. Next, ensure the new application is selected and click **Next**.

solace.



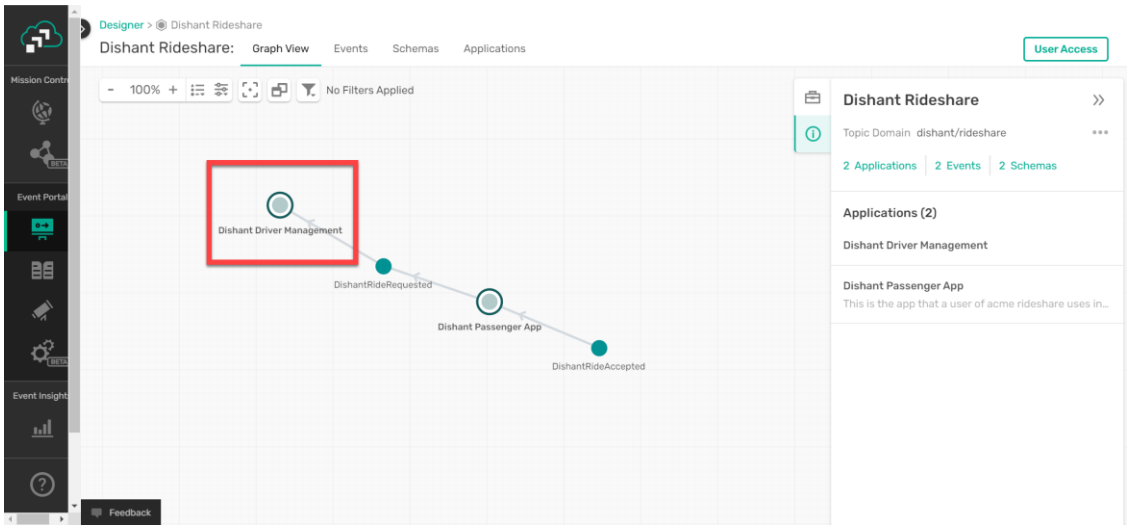
6. Select the event this new application is going to subscribe to and click **Next**. Note, on the window that follows after, you want to ensure the event is not selected as this application will not be publishing that event and only consumes it.





7. Click the **Save** button.

8. Once the application has been created, select the **Event Designer** from the menu, and open your **Application Domain** to view the visual representation of the Driver Management application and its linkage to the RideRequested event.



NOTE: For the remaining exercises, details are provided on the events, schema, and applications you need to create, without the step-by-step instructions.

Exercise 7: List of Events and Schemas to Create or Update

In this exercise you will create all events produced and consumed in the rideshare system, or update an existing event created before.

Event: DriverRideAccepted

Name	<YOUR_NAME>DriverRideAccepted
Description	This is the application that a driver uses to interact with the Acme Rideshare ecosystem.
Topic	<YOUR_NAME>/rideshare/driver/accepted/2.0.5/{location}/json/{customer_id}/{request_id}/{driver}
Shared	No
Schema Name	<YOUR_NAME>DriverRideAccept (NOTE: This schema was already created in previous exercises, so you should reuse that schema)
Schema Description	NA
Schema Format	NA
Schema Payload File	NA

Event: DriverStatusUpdated

Name	<YOUR_NAME>DriverStatusUpdated
Description	When a driver status changes, this event is generated to notify up status changes.
Topic	<YOUR_NAME>/rideshare/driver/status/updated/1.0.2/{driver_id}/{activation_status}
Shared	No
Schema Name	<YOUR_NAME>DriverStatus
Schema Description	Schema represents the data format that the driver apps should be sending periodically that represent the "state" of the driver.
Schema Format	JSON
Schema Payload File	DriverStatus.json

Event: TripUpdated

Name	<YOUR_NAME>TripUpdated
Description	This is the event which is fired periodically when a driver is on an active trip.
Topic	<YOUR_NAME>/rideshare/trip/updated/7.0.2/json/{driver_id}/{trip_id}/{status}

Shared	Yes
Schema Name	<YOUR_NAME>Trip
Schema Description	Data Schema represents the information of a trip status
Schema Format	JSON
Schema Payload File	Trip.json

Event: DriverFundsDeposited

Name	<YOUR_NAME>DriverFundsDeposited
Description	This event is triggered after billing has successfully deposited money into a drivers account
Topic	<YOUR_NAME>/rideshare/driver/funds/deposited/1.0.1/{driver_id}/{trip_id}/{payment_id}
Shared	No
Schema Name	<YOUR_NAME>DriverPayment
Schema Description	This Schema represents the data needed to inform drivers about being paid for a given ride and the payment breakdown
Schema Format	JSON
Schema Payload File	DriverPayment.json

Event: CustomerReceiptGenerated

Name	<YOUR_NAME>CustomerReceiptGenerated
Description	This is the event which contains a customers receipt information once payment has gone though
Topic	<YOUR_NAME>/rideshare/billing/receipt/created/1.0.1/json/{request_id}
Shared	No
Schema Name	<YOUR_NAME>RideReceipt
Schema Description	This Schema conforms to the format of a customer receipt and should only be used when providing receipts to end customers.
Schema Format	JSON
Schema Payload File	RideReceipt.json

Event: DriverRideRequested

Name	<YOUR_NAME>DriverRideRequested

Description	This is the event triggered to active drivers that are within a geospatial proximity to the passenger
Topic	<YOUR_NAME>/rideshare/driver/requested/3.8.1/{start_latitude}/{start_longitude}/json/{customer_id}/{request_id}
Shared	No
Schema Name	<YOUR_NAME>RideRequest (NOTE: This schema was already created in previous exercises, so you should reuse that schema)
Schema Description	NA
Schema Format	NA
Schema Payload File	NA

Exercise 8: List of Applications to Create or Update

NOTE: Update any existing application that were created in previous exercises instead of creating a new one, for ex. the Passenger App.

Application: Passenger App

Name	<YOUR_NAME> Passenger App
Description	This is the app that a user of acme rideshare uses in order to obtain rides from willing drivers.
Events Published	<YOUR_NAME>RideRequested
Events Subscribed	<YOUR_NAME>CustomerReceiptGenerated <YOUR_NAME>RideAccepted

Application: Driver App

Name	<YOUR_NAME> Driver App
Description	This is the application that a driver uses to interact with the Acme Rideshare ecosystem.
Events Published	<YOUR_NAME>DriverRideAccepted <YOUR_NAME>DriverStatusUpdated <YOUR_NAME>TripUpdated
Events Subscribed	<YOUR_NAME>DriverFundsDeposited <YOUR_NAME>DriverRideRequested

Application: Billing App

Name	<YOUR_NAME> Billing App
Description	This app processes customer payments and also pays the drivers.
Events Published	<YOUR_NAME>CustomerReceiptGenerated <YOUR_NAME>DriverFundsDeposited
Events Subscribed	<YOUR_NAME>TripUpdated

Application: Driver Management

NOTE: We already created this app in a previous exercise. Here we will update the app to produce & consume other events.

Name	<YOUR_NAME> Driver Management
Description	This application controls which rides need to be fulfilled, which are in route, and completed
Events Published	<YOUR_NAME>DriverRideRequested <YOUR_NAME>RideAccepted
Events Subscribed	<YOUR_NAME>DriverRideAccepted <YOUR_NAME>DriverStatusUpdated <YOUR_NAME>RideRequested <YOUR_NAME>TripUpdated

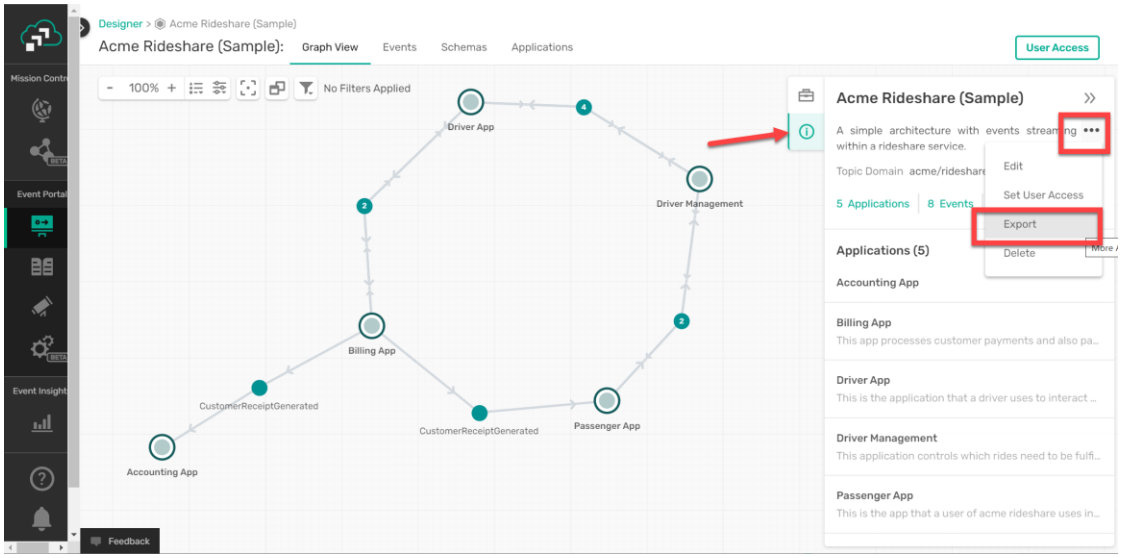
Application: Expense Integration

This is left as an optional bonus exercises for the users to create. The Expense Integration app consumes the event containing the customer receipt.

Exporting Domain

In this exercise you will export your domain to a JSON file. This allows you to backup your domain and/or recreate the domain in another org account.

1. Go to your domain and select the 3 dots from the overview panel.



2. Select **Export** from the menu options, and then select **Continue** when prompted.

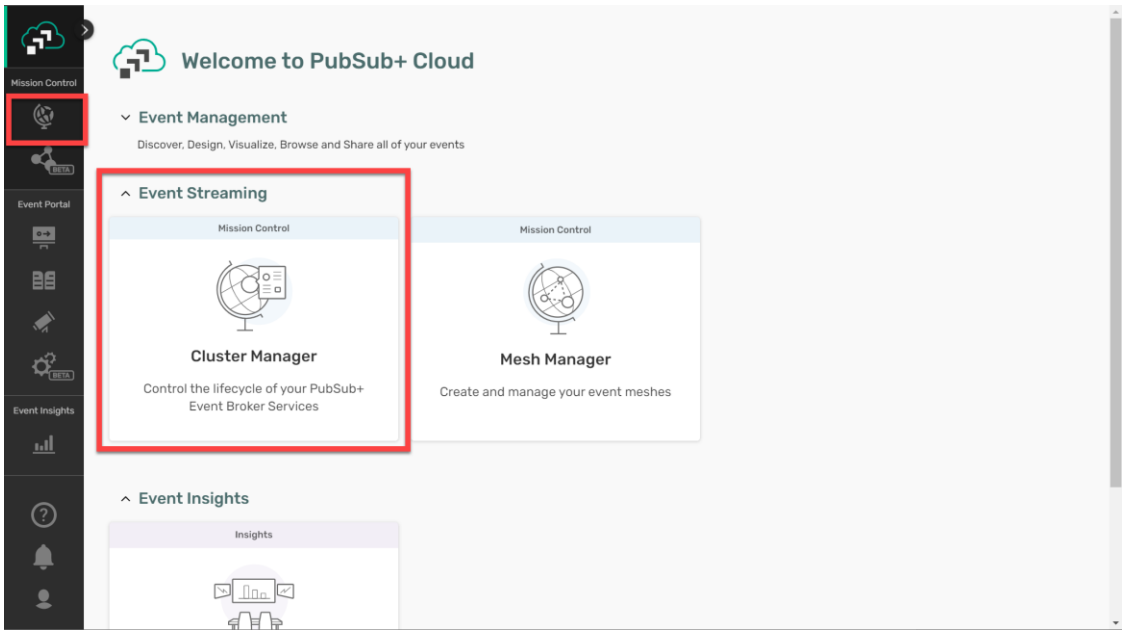
You have now completed all activities part of the PubSub+ Event Portal module.

Exercise 9: Deploying your PubSub+ Cloud Event Streaming Service

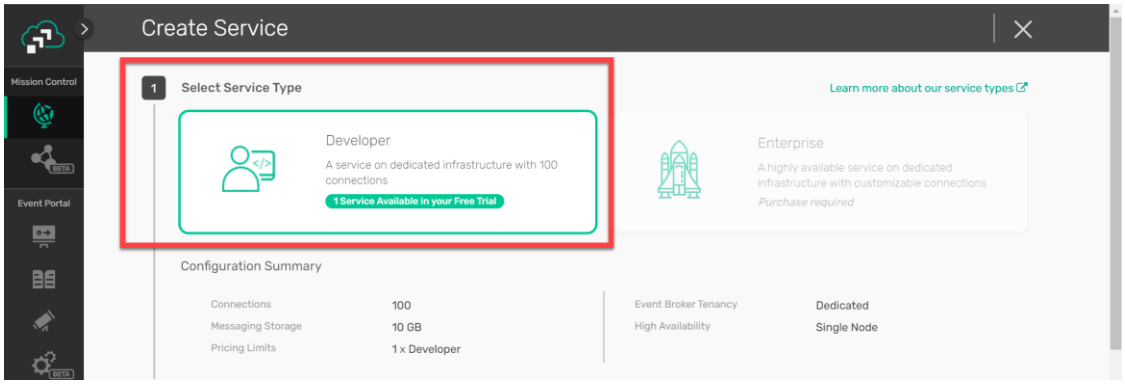
NOTE: For all Trial account you can deploy 1 free Developer event streaming service. If you have previously created a service, you will need to delete that services, or you can continue to use the same services for the remaining exercises.

You will deploy a Developer event streaming service in this exercise:

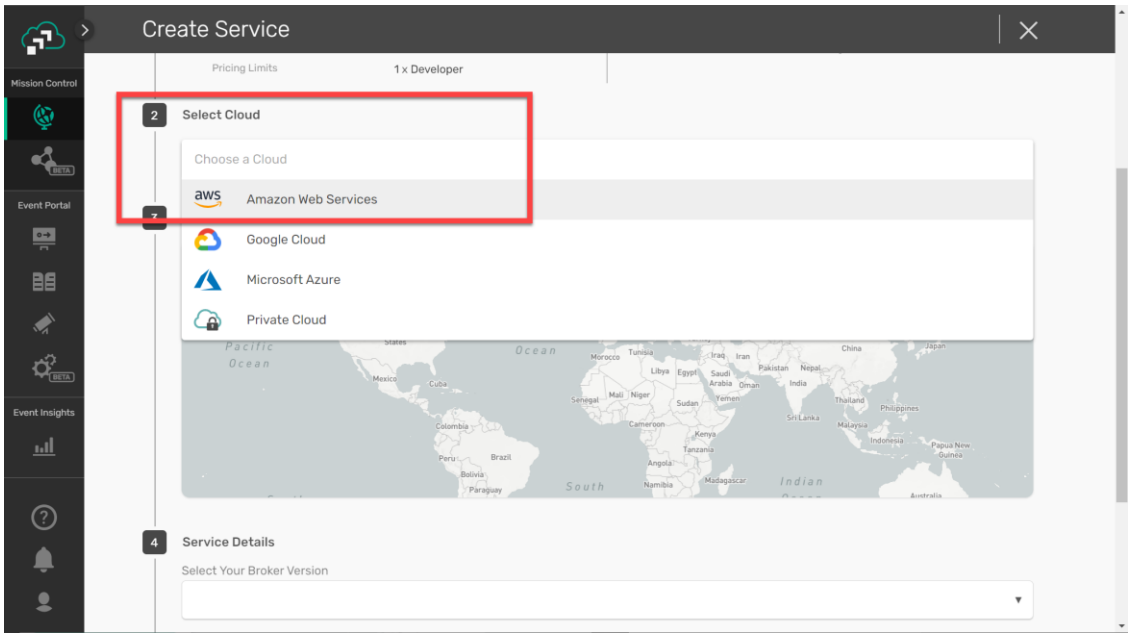
1. From the home page of PubSub+ Cloud Console select **Cluster Manager**:



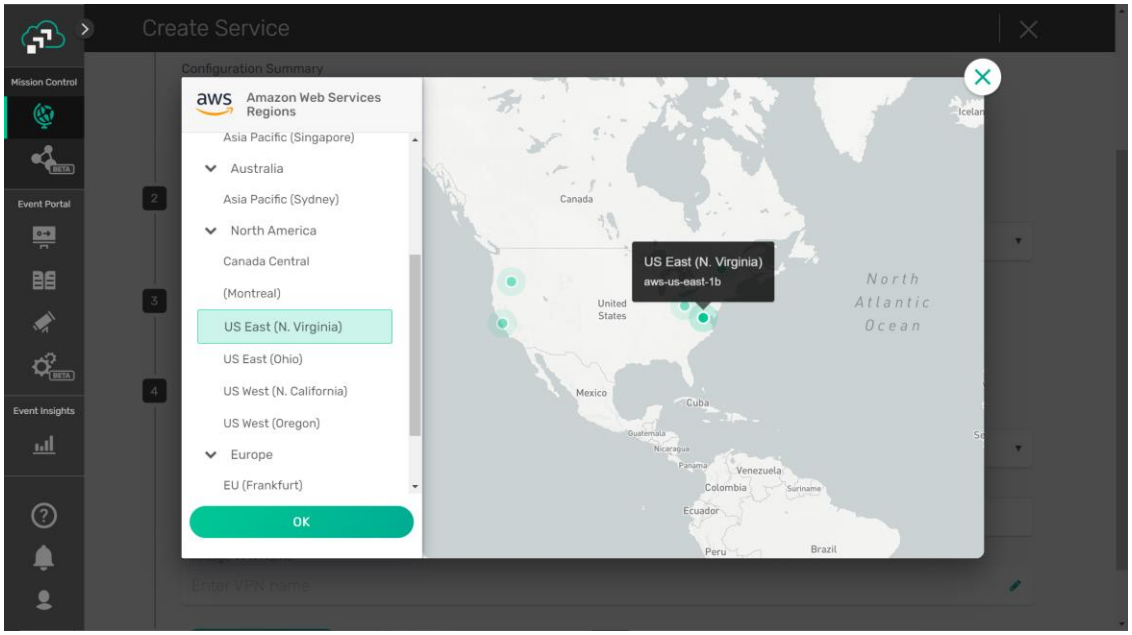
2. From the Services pages select the "+" button or **Create Service** button.
3. Select the **Developer** service type:



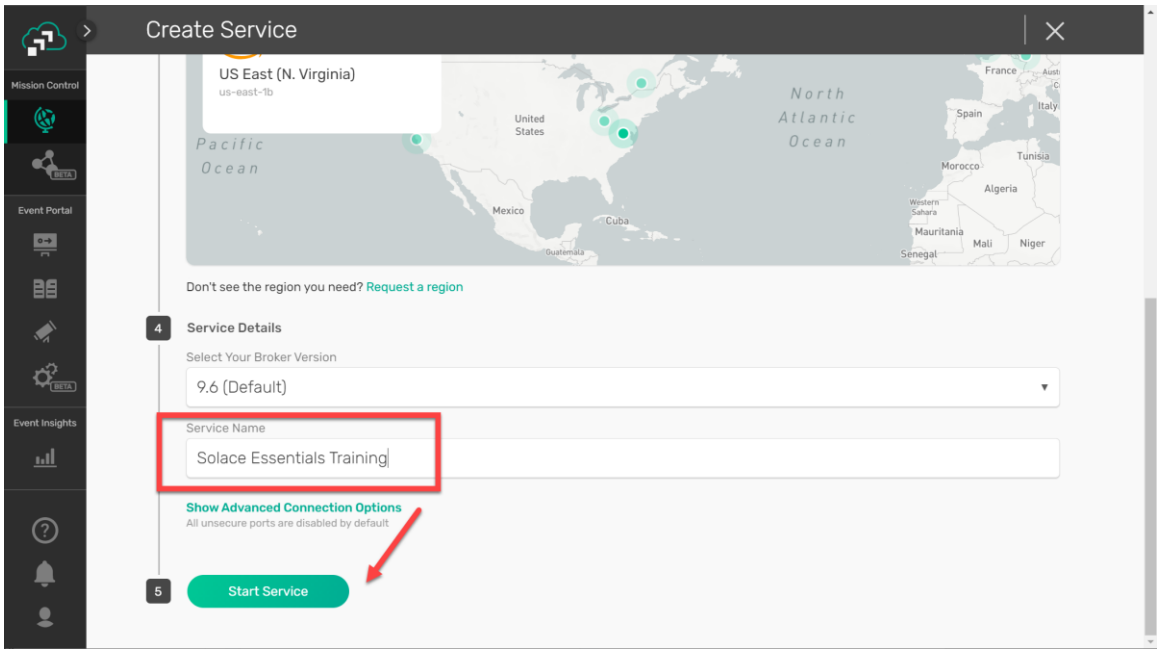
4. Select a cloud from the dropdown box. Choose between either Amazon Web Services/Google Cloud Platform/Microsoft Azure.



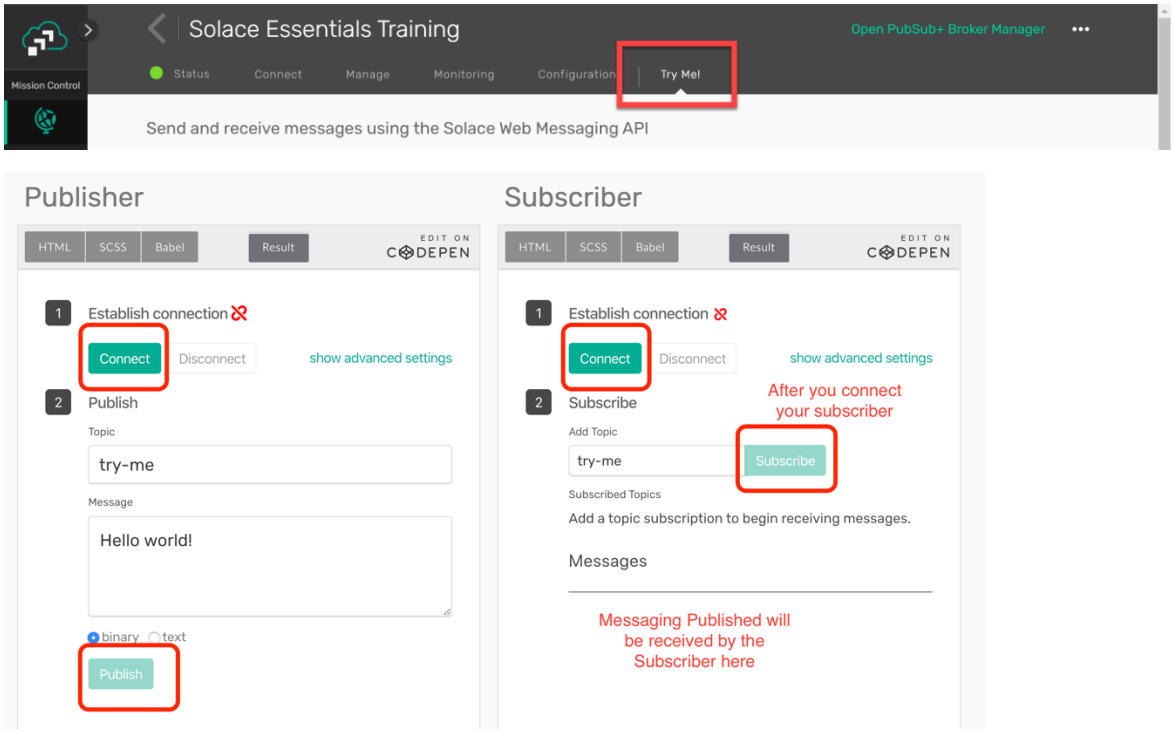
5. Select any available region in the cloud selected.



6. Give your service a name: **Solace Essentials Training** and select **Start Service**.



7. Your service will be created. This may take 5-10 mins to deploy.
8. Once the service is created you can click on the **Try Me** tab to test your service.



You have now successfully deployed a PubSub+ messaging service in a public cloud. We will be using this service for some of the other activities in this guide.

Exercise 10: Deploying PubSub+ on Docker for Desktop

You will deploy a PubSub+ Software Event Broker Standard Edition on Docker for Desktop in this exercise. Installation of Docker is out of the scope of this course. You can also follow the instructions provided on the Solace webpage (https://solace.com/products/event-broker/software/getting-started/) for similar instructions.

Note: you deploy the free standard edition on another environment like VirtualBox if Docker is not available or you are unable to install Docker on your desktop.

1. From a command-line terminal copy and paste the following Docker command:

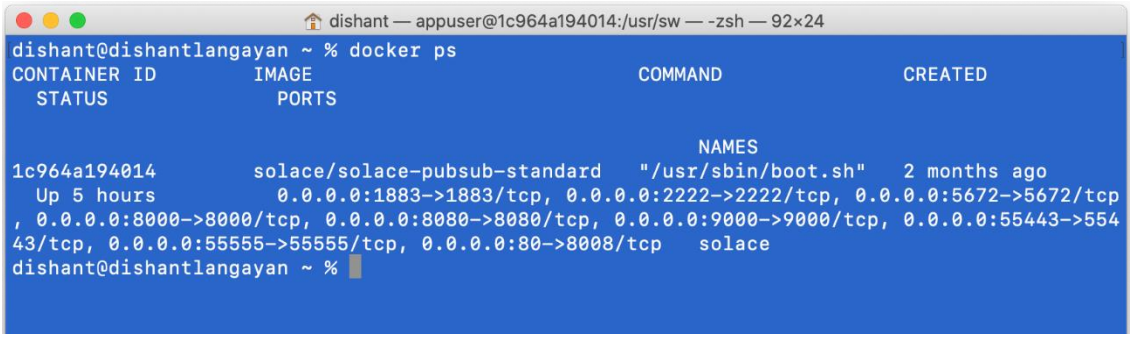
```
docker run -d -p 8080:8080 -p 55555:55555 -p:80:8008 -p:1883:1883 -p:8000:8000 -p:5672:5672 -p:9000:9000 -p:2222:2222 --shm-size=2g --env username_admin_globalaccesslevel=admin --env username_admin_password=admin --name=solace solace/solace-pubsub-standard
```

   This command starts a message broker container named solace, using the latest PubSub+ Standard image pulled from Docker Hub, creates an admin user with global access permissions, and publishes the following message broker container ports to the same ports on your host machine:

   - port 8080 — Use this port when configuring the message broker container with Solace PubSub+ Manager.
   - port 55555 — Your applications can use Solace APIs to connect to the message broker on this port.
   - port 80 — The JavaScript sample applications below use this port to pass Web Messaging traffic through the message broker.
   - ports 1883 & 8000 — Ports for MQTT connectivity, over TCP and over WebSockets respectively
   - port 5672 — AMQP 1.0 applications using Apache QPID APIs would connect here
   - port 9000 — Use REST to send messaging and event data with Solace's RESTful API port
   - port 2222 — Use SSH to connect to the Solace Command Line Interface (CLI) for advanced configuration

2. Verify the PubSub+ instance up and running:

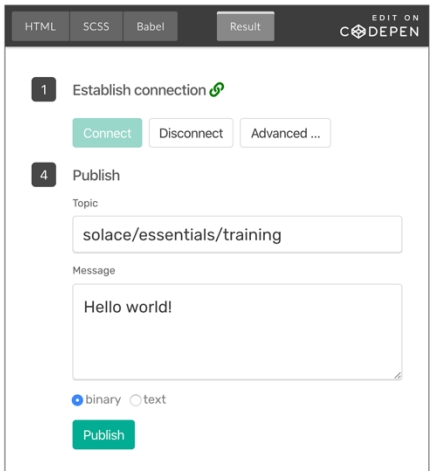
```
docker ps
```



solace.

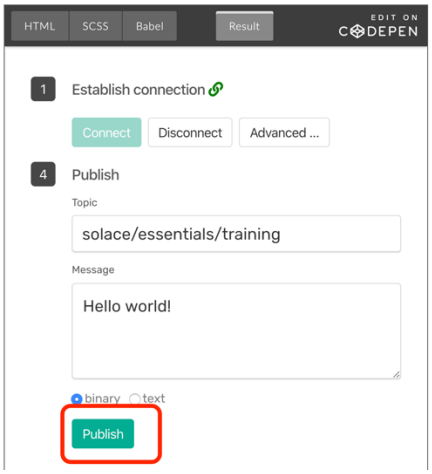
3. Open the following webpage to run the Publisher and Subscriber samples: https://solace.com/products/event-broker/software/getting-started/
4. Connect the Publisher and the topic: **solace/essentials/training**

Publisher

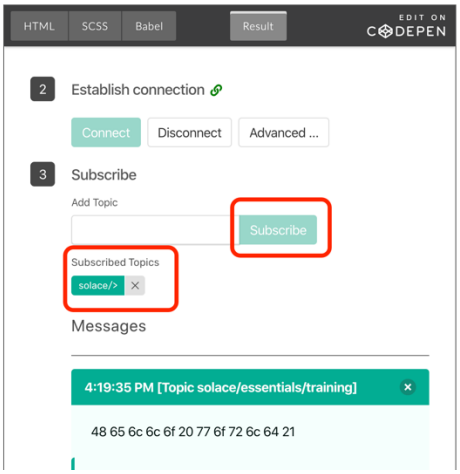


5. Next Connect the Subscriber, and subscribe to the topic: **solace/>**
6. Once the subscription has been added, select the Publish button and verify you can receive messages on the subscriber.

Publisher



Subscriber



7. Experiment with different topics and topic subscription. For more information on Topics refer to the Topics and Wildcard lesson part of the Introduction module in the online course.

You have now successfully deployed a PubSub+ Software Event Broker on Docker for Desktop and completed all activities part of the Event Broker module. We will be using this service for some of the other activities in this guide.

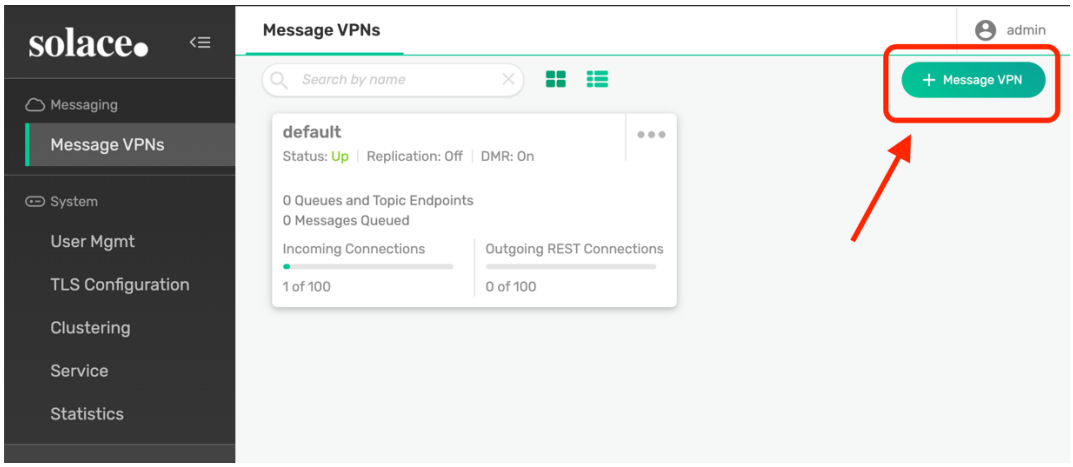
Exercise 11: Creating a Message-VPN using PubSub+ Manager

In the exercise you will create a new Message-VPN on the PubSub+ Software event broker deployed on Docker. We will use PubSub+ Manager to create this VPN.

1. Ensure PubSub+ is running locally on your machine.
2. Open a browser and enter the following URL:

   ```
   http://127.0.0.1:8080/
   ```

3. Log in using the username **admin** and password **admin**.
4. Next click on the Create Message-VPN button

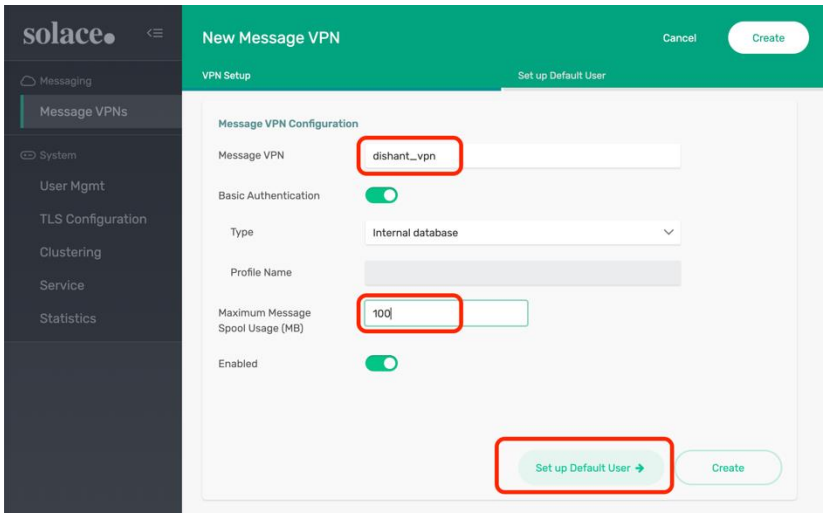


5. Enter the following settings:

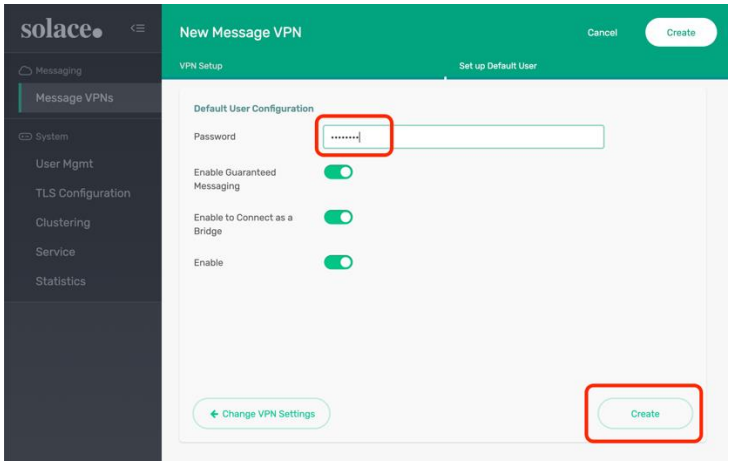
   Message VPN Name: <YOUR_FIRSTNAME>_vpn

   Maximum Message Spool Usage (MB): 100

   And click **Set up Default User** button.



6. For the password enter: **password** and select **Create** button.

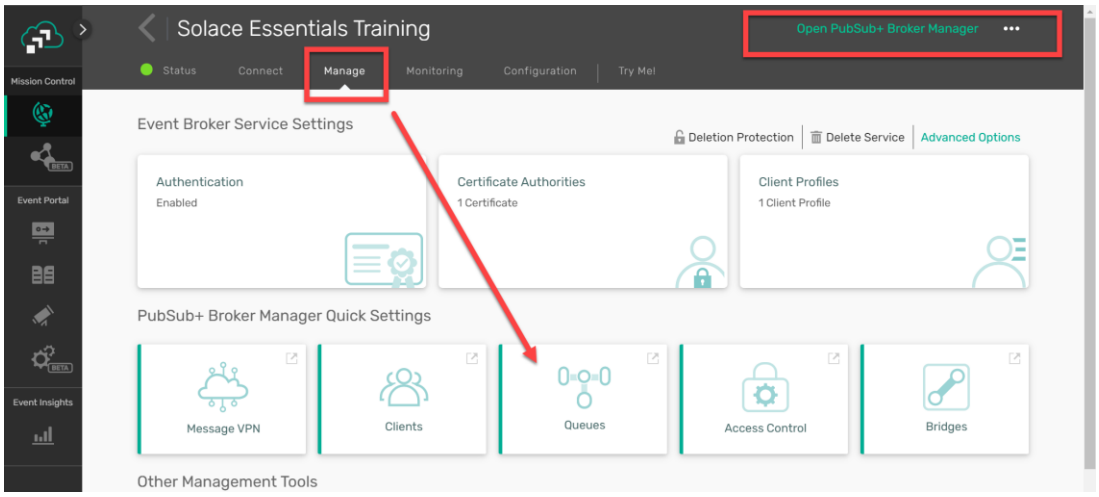


You have now created your Message-VPN.

Exercise 12: Creating a Queue using PubSub+ Manager

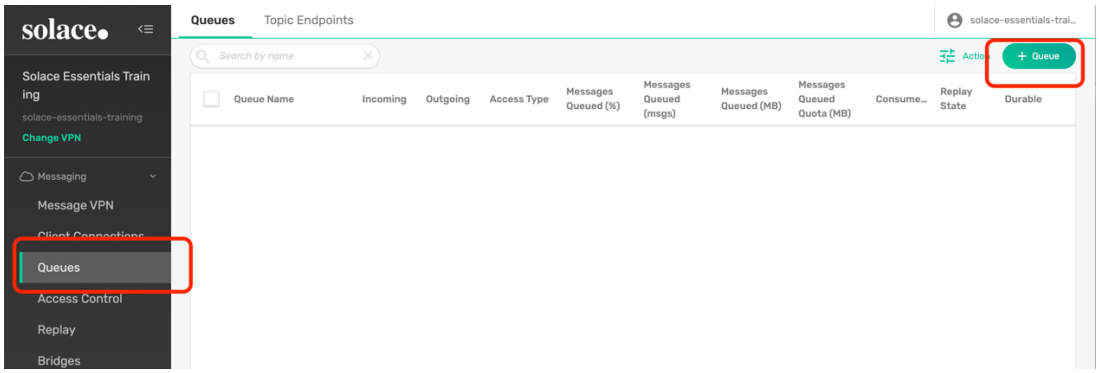
In the exercise we are going to create a Queue on the PubSub+ Cloud messaging service using PubSub+ Manager.
NOTE: the process to create queues using PubSub+ Manager is same for both the Software broker and PubSub+ Cloud. In this exercise we are just looking at another way to access PubSub+ Manager for our cloud service we deployed before.

1. Logon to your **PubSub+ Cloud Console** and select **Cluster Manager** to view your services created in earlier exercise: https://console.solace.cloud/services
2. Select your event streaming service
3. From the **Manage** tab select **Queues** (you can optionally select Manage Service button at the top right corner, and then when PubSub+ Manager opens select Queues from the menu options).

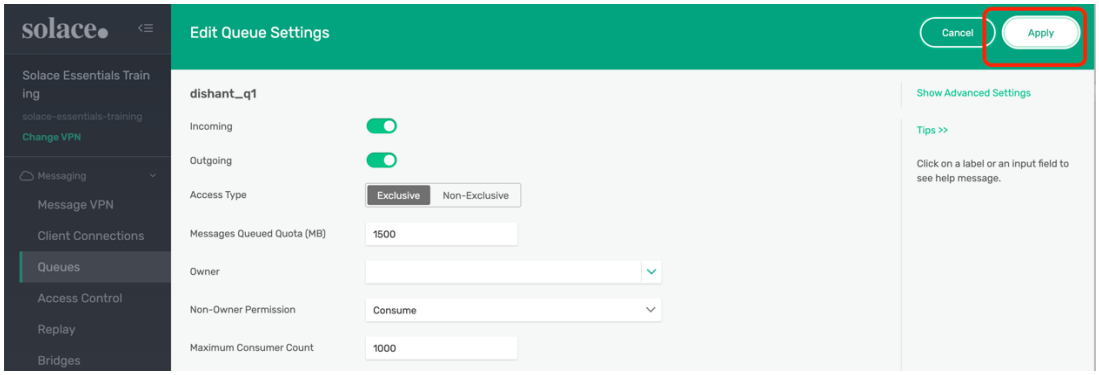


4. Click on the create Queue icon, and give you queue the following name:

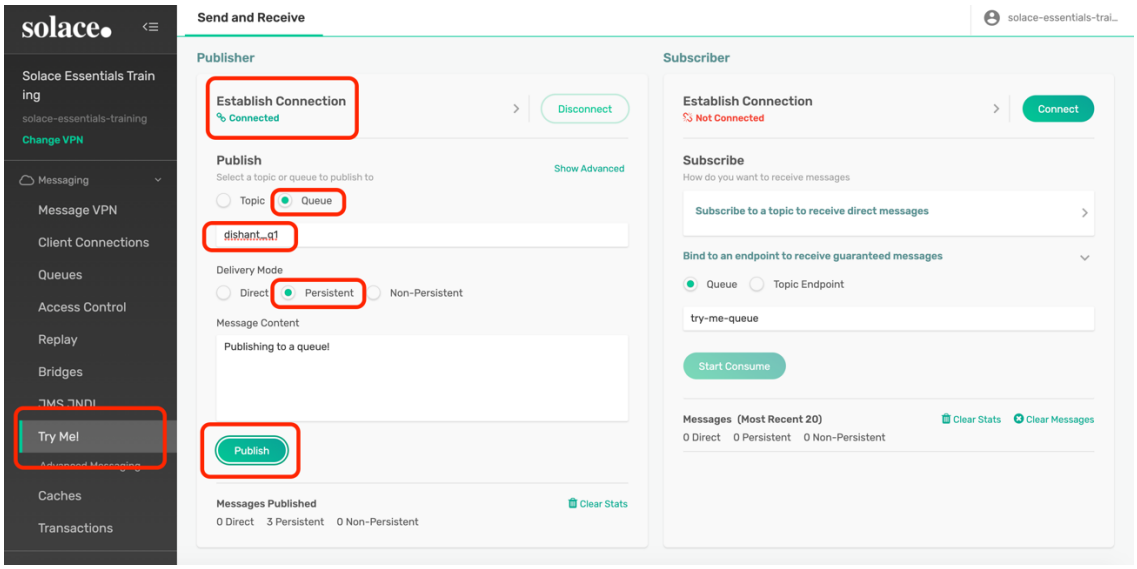
   Queue Name: <YOUR_FIRSTNAME>_q1



5. Use all the default settings and select **Apply**.



6. Your queue should now be created.
7. To test your queue, lets publish & consume some messages using the queue.
8. Select the **Try Me** from the left menu.
9. **Connect** the sample Publisher, and select **Queue**, enter your queue name <YOUR_FIRSTNAME>_q1, select the delivery mode as **Persistent**, and **Publish** a few messages.



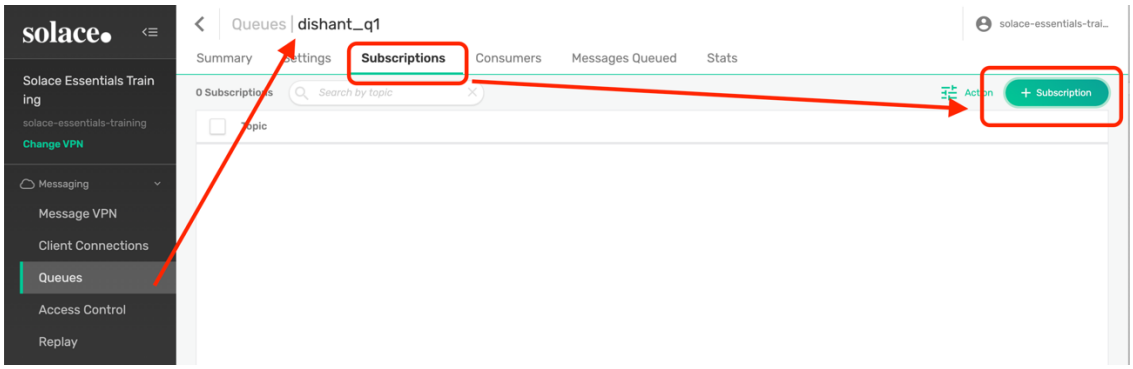
10. NOTE: we have not yet connected the subscriber so our messages should be queued up and we can verify this from the Queues page.
11. **Connect** your sample **Subscriber**, select **Bind to an endpoint to receive guaranteed messages**, select **Queue**, enter your queue name, and select **Start Consume**.
12. Verify you have received the messages. You can also go back to the actual queue to verify the messages have been consumed and acknowledged.

You have now created a queue and tested it by sending and receiving messages using the Try Me feature.

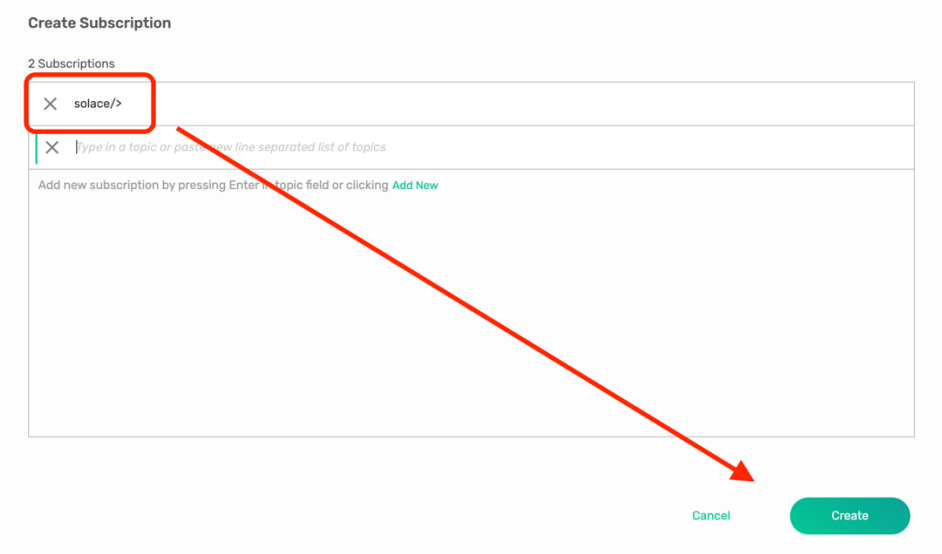
Exercise 13: Mapping Topics to Queue

In the exercise we are going to map a topic to the queue we created in the previous exercise.

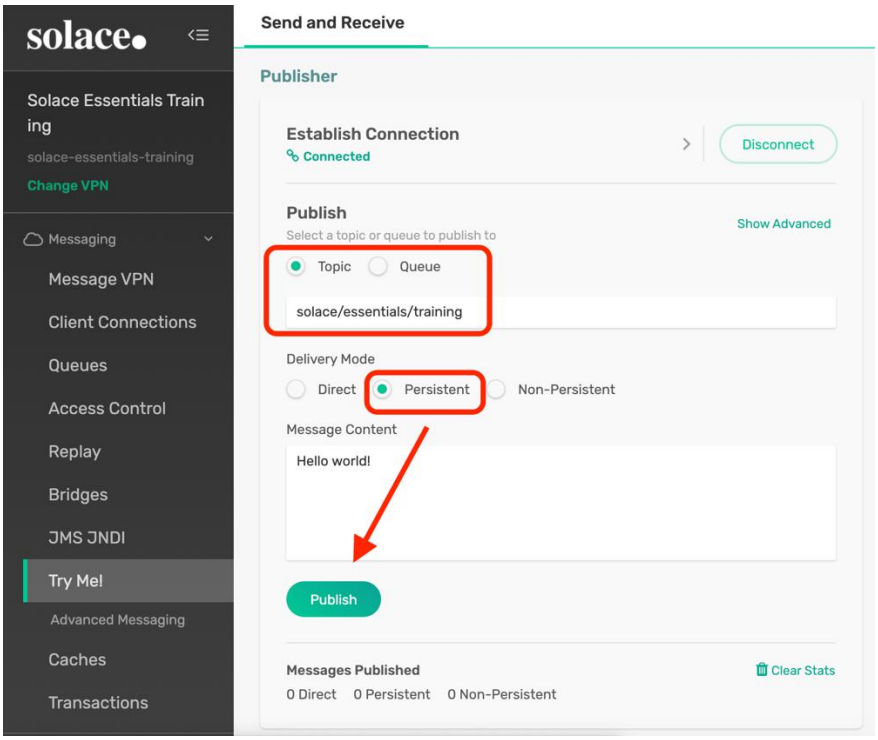
1. From the previous exercise, access the PubSub+ Manager for your cloud service, and select the queue you had created (<YOUR_FIRSTNAME>_q1)
2. Click on the queue to view its details
3. Then select **Subscriptions** and click on the **Create Subscription** button.



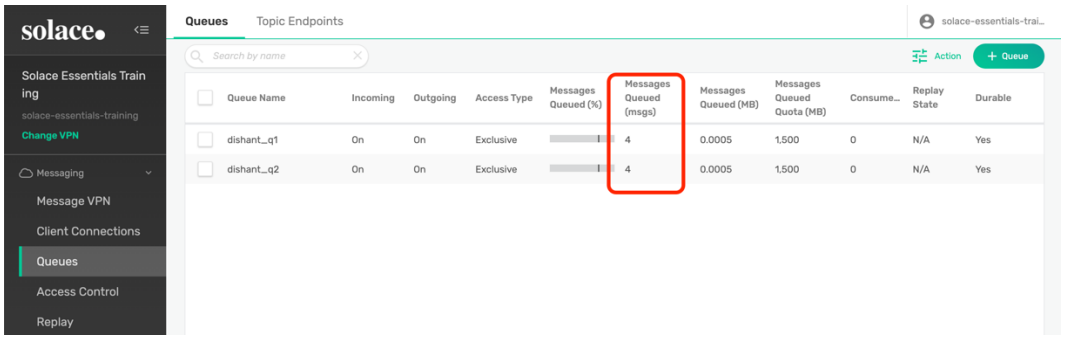
4. Add the following topic subscription: **solace/>** and select **Create**.



5. You have now mapped a topic subscription to a queue.
6. Create another queue <YOUR_FIRSTNAME>_q2 and repeat the above steps to add the same topic subscription to this second queue.
7. Now test Publish-Subscribe exchange using the Try Me tab. Note this time publish message to the topic **solace/essentials/training** instead of publishing to the queue directly.



8. Verify both queues receive the message.



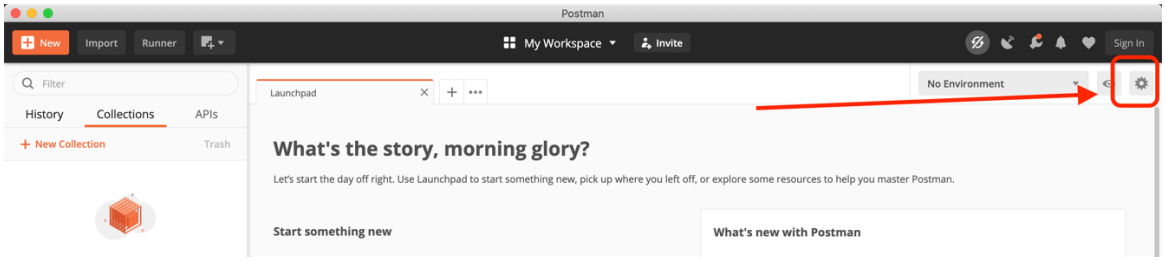
You have now completed mapping topics to queues activity and demonstrated how to perform publish-subscribe using guaranteed messaging and queues.

Exercise 14: Managing a Queue using SEMP

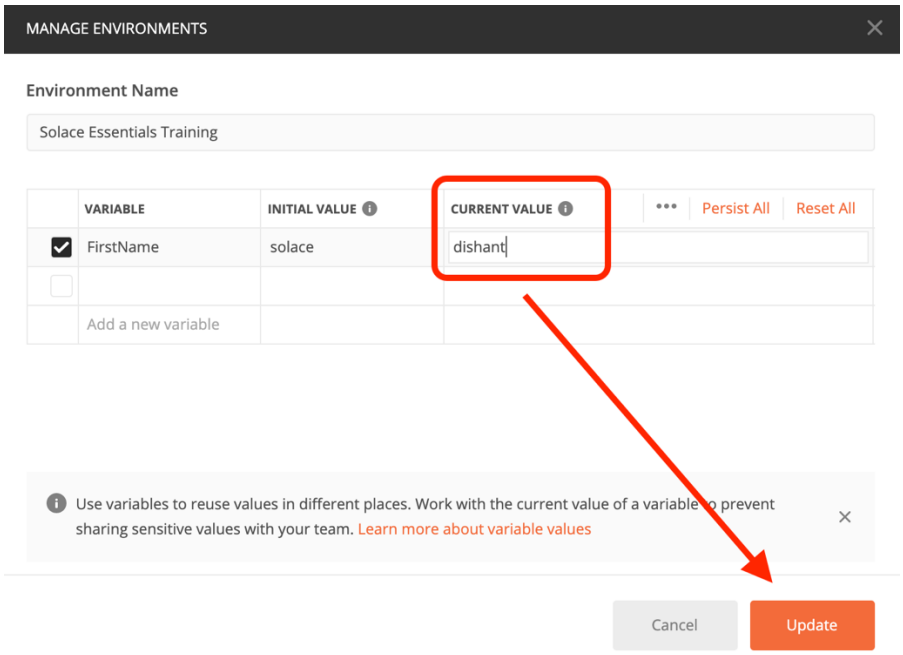
In the exercise we are going to manage Queues on the Software event broker deployed locally on Docker using SEMP.

For this activity we will use Postman utility, but you can also use other similar tools or command-line based tools like cURL.

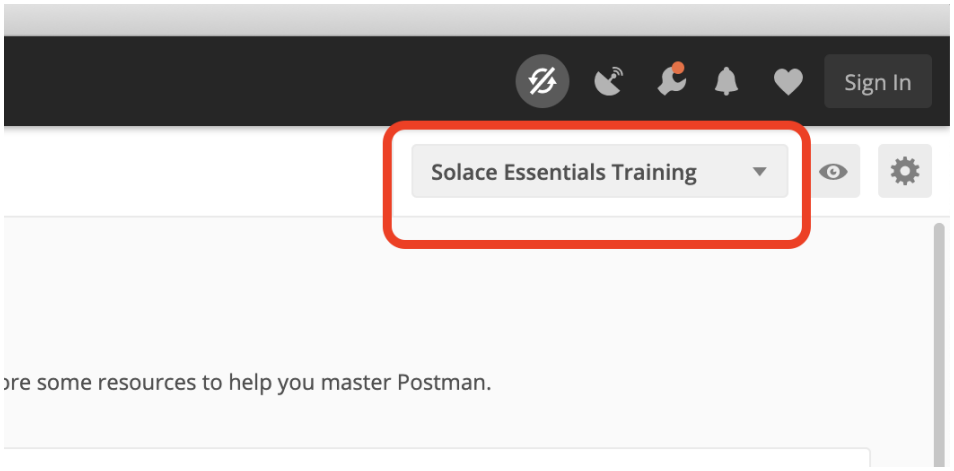
1. Open Postman.
2. In the Activity Guide.zip downloaded locate the SEMP Postman Collection folder and import the **Solace Essentials Training.postman_environment.json** in Postman environment



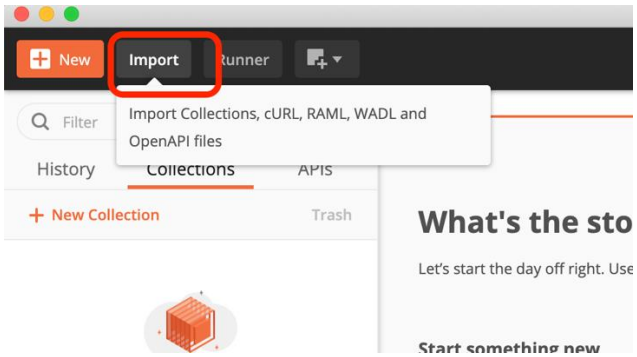
3. Edit the imported environment and update the current value for **FirstName** to your first name.



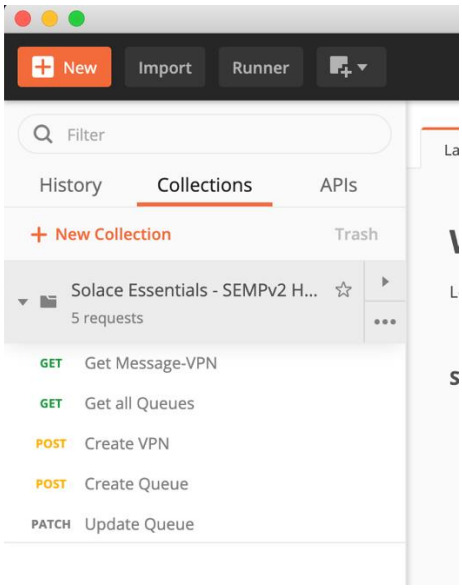
4. Make sure to select the recently imported environment



5. Next import the SEMP collection **Solace Essentials - SEMPv2 Hands-On.postman_collection.json**.



6. Verify the SEMP collection has been imported



7. Run **Get all Queues** SEMP request.
    a. **NOTE #1** the scripts assume you are using the local PubSub+ Software broker deployed on Docker. If you broker is running on another machine with a different IP, and the URL should be adjusted accordingly.
    b. **NOTE #2** the scripts also assume your Message-VPN is named as <YOUR_FIRSTNAME>_vpn. If you named you VPN differently, then this will need to be adjusted in the URL as well.
8. Verify the command run successfully with a **200 OK HTTP** response.
9. Similarly, run the **Create Queue** and **Update Queue** SEMP requests.

You have now completed the SEMP activity using Postman. Additional SEMP requests and documentation can be found here: https://docs.solace.com/SEMP/Using-SEMP.htm

Exercise 15: CLI Basics

In the exercise we are going to access the Solace CLI and explore some basic commands. CLI can be useful for managing PubSub+ Software or Appliance Event Brokers specially for those users that prefer command-line based tools.

In this activity we will run the CLI on the Software broker deployed on Docker. For accessing CLI for other deployments refer to the documentation here: https://docs.solace.com/Solace-CLI/Using-Solace-CLI.htm

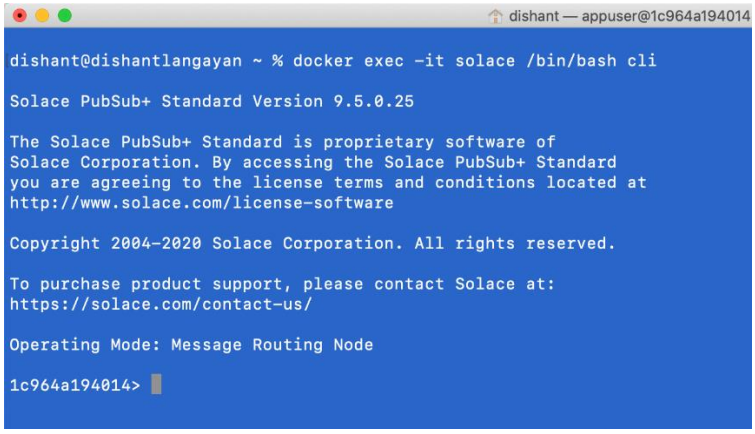
1. Open command-line terminal window and verify your PubSub+ docker container is running

```
docker ps
```

2. Next enter the following Docker command to open a CLI session to the broker:

```
docker exec -it solace /bin/bash cli
```

3. Verify the Broker banner is displayed with the broker version.



4. Enter the following command to view a list of all Message-VPNs on your broker.

```
show message-vpn *
```

5. Enter the following command to edit a queue configuration, say updating the Max Msg Spool Usage (assuming you had created a VPN from the previous exercises with the format <YOUR_FIRSTNAME>_vpn).

```
enable
config
message-spool message-vpn <YOUR_FIRSTNAME>_vpn
queue <YOUR_FIRSTNAME>_q1 (Or the name of the queue you have on this broker; to
create a new queue using CLI using the create queue <name> command)
max-spool-usage 300
```

6. Run the following command to verify your changes:

```
show queue <YOUR_FIRSTNAME>_q1 detail
```

7. To go back to the root-level enter **home** or **end**.

You have now completed the CLI activity. You can explore more commands and operations from the Solace documentation or using the help command on the CLI.

Exercise 16: Viewing Syslog Logs Files

In the exercise we are going to view the Syslog events and other log files on the PubSub+ Software Event Broker deployed on Docker.

1. Enter the following Docker command to open access the host docker container itself:

```
docker exec -it solace /bin/bash
```

2. Change to the following directory:

```
cd /usr/sw/jail/logs/
```

3. Verify using **ls -l** you can see the following log files:

```
[appuser@1c964a194014 logs]$
[appuser@1c964a194014 logs]$ ls -l
total 708
-rw------- 1 appuser root  67066 Jul  3 15:40 command.log
-rw------- 1 appuser root 468902 Jul  3 14:11 debug.log
-rw------- 1 appuser root 110961 Jul  3 15:34 event.log
-rw------- 1 appuser root   3564 Jul  2 18:45 kernel.log
-rw------- 1 appuser root  51392 Jul  3 12:15 system.log
[appuser@1c964a194014 logs]$
```

4. Use **vi** to view the event.log and system.log file. Perform basic search operations to find events you are interested in. For ex. locate the **CLIENT_CLIENT_CONNECT** & **CLIENT_CLIENT_DISCONNECT** events for the Try Me Publisher and Subscribe we connected in the previous exercises. You can also use Linux greg and other tools to search for text within these log files.

```
tjs/chrome-80.0.3987-OSX-10.15.5/0426590641) authScheme(Basic) authorizationGroup() clientProfile(default) ACLProfile(default) SSLDowngradedToPlainText(No)
NegotiatedTo() SslRevocation(Not Checked), Capabilities(unbind-ack, bind-response-endpoint-error-id)
2020-06-30T20:25:37.127+00:00 <local3.info> 1c964a194014 event: CLIENT: CLIENT_CLIENT_DISCONNECT: default solclientjs/chrome-80.0.3987-OSX-10.15.5/04265906
001 Client (7) solclientjs/chrome-80.0.3987-OSX-10.15.5/0426590641/0001 username default WebSessionId (0x7a64950d226483b5) reason(Inactive Connection) fina
atistics - dp(116, 115, 0, 1, 116, 116, 1910, 2678, 0, 56, 1910, 2734, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0) conn(0, 0, 172.17.0.1:40466, CLOSD, 0, 0, 0) zip(0, 0,
0, 0.00, 0.00, 0, 0, 0, 0, 0, 0, 0, 0) web(116, 116, 4462, 6468, 0, 0, 0), SslVersion(), SslCipher()
2020-06-30T20:25:38.887+00:00 <local3.info> 1c964a194014 event: CLIENT: CLIENT_CLIENT_DISCONNECT: default solclientjs/chrome-80.0.3987-OSX-10.15.5/35552157
001 Client (6) solclientjs/chrome-80.0.3987-OSX-10.15.5/3555215741/0001 username default WebSessionId (0x380cd8d9cc59a0b0) reason(Inactive Connection) fina
atistics - dp(152, 152, 1, 0, 153, 152, 2323, 3418, 56, 0, 2379, 3418, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0) conn(0, 0, 172.17.0.1:40462, CLOSD, 0, 0, 0) zip(0, 0,
0, 0.00, 0.00, 0, 0, 0, 0, 0, 0, 0, 0) web(153, 153, 5767, 8326, 0, 0, 0), SslVersion(), SslCipher()
2020-06-30T20:36:56.174+00:00 <local3.notice> 1c964a194014 event: SYSTEM: SYSTEM_AUTHENTICATION_SESSION_OPENED: - - SEMP session 172.17.0.1 internal authen
tion opened for user admin (admin)
2020-06-30T20:37:06.040+00:00 <local3.warning> 1c964a194014 event: VPN: VPN_VPN_STATE_CHANGE: bridgedemo - Message VPN (1) bridgedemo State Changed to: dow
2020-06-30T20:37:06.153+00:00 <local3.info> 1c964a194014 event: VPN: VPN_BRIDGING_LINK_DOWN: bridgedemo - Message VPN (1) bridgedemo Bridge bridgedemo_defa
from v:1c964a194014 VPN default down: shutdown
2020-06-30T20:37:06.174+00:00 <local3.info> 1c964a194014 event: CLIENT: CLIENT_CLIENT_DISCONNECT: bridgedemo #bridge/local/bridgedemo_default/1c964a194014/
0 Client (0) #bridge/local/bridgedemo_default/1c964a194014/259/0 username #client-username WebSessionId (N/A) reason(Logout) final statistics - dp(13321, 1
, 0, 0, 13321, 13323, 640005, 640089, 0, 0, 640005, 640089, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0) conn(0, 0, 127.0.0.1:55555, ESTAB, 0, 0, 0) zip(0, 0, 0, 0, 0.00,
0, 0, 0, 0, 0, 0, 0, 0, 0) web(0, 0, 0, 0, 0, 0, 0), SslVersion(), SslCipher()
2020-06-30T20:37:06.190+00:00 <local3.notice> 1c964a194014 event: CLIENT: CLIENT_CLIENT_UNBIND: bridgedemo #bridge/local/bridgedemo_default/1c964a194014/25
Client (0) #bridge/local/bridgedemo_default/1c964a194014/259/0 username #client-username Unbind to Flow Id (1), ForwardingMode(StoreAndForward), final stat
cs - flow(255, 0, 0, 0, 0, 0, 0, 0), isActive(No), Reason(Client disconnected)
2020-06-30T20:37:06.224+00:00 <local3.info> 1c964a194014 event: VPN: VPN_BRIDGING_LINK_DOWN: default - Message VPN (0) default Bridge bridgedemo_default fr
:1c964a194014 VPN bridgedemo down: connection down
@
@
/CLIENT_CLIENT_DISCONNECT                                                                                                    255,73
```

5. To access these events from the CLI, either type cli on the command line or open a CLI session as done in the previous exercise.

6. Enter the following command to view for example the event.log file and search for the CLIENT_CLIENT_DISCONNECT events:

```
show log event lines 1000 find CLIENT_CLIENT_DISCONNECT
```

```
1c964a194014>
1c964a194014> show log event
find   lines
1c964a194014> show log event lines 1000 find CLIENT_CLIENT_DISCONNECT
2020-04-21T21:35:45.151+00:00 <local3.info> 1c964a194014 event: CLIENT: CLIENT_CLIENT_DISCONNECT: default #client Client (0) #client username #client-username
WebSessionId (N/A) reason(Login Respond Error) final statistics - dp(1, 0, 0, 0, 1, 0, 186, 0, 0, 0, 186, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0) conn(0, 0, 127.0.0.1
:59530, ESTAB, 0, 0, 0) zip(0, 0, 0, 0, 0.00, 0.00, 0, 0, 0, 0, 0, 0, 0, 0) web(0, 0, 0, 0, 0, 0, 0), SslVersion(), SslCipher()
2020-04-29T12:55:34.305+00:00 <local3.info> 1c964a194014 event: CLIENT: CLIENT_CLIENT_DISCONNECT: default #client Client (0) #client username #client-username
WebSessionId (N/A) reason(Login Respond Error) final statistics - dp(1, 0, 0, 0, 1, 0, 186, 0, 0, 0, 186, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0) conn(0, 0, 127.0.0.1
:48768, ESTAB, 0, 0, 0) zip(0, 0, 0, 0, 0.00, 0.00, 0, 0, 0, 0, 0, 0, 0, 0) web(0, 0, 0, 0, 0, 0, 0), SslVersion(), SslCipher()
```

You have completed the Syslog log files activity.

Exercise 17: Gathering Diagnostics for Docker Containers

In the exercise we are going to gather diagnostics file using the utility delivered with the Docker container event broker. These diagnostics file can be useful for Solace Support Team to troubleshoot issues with your event brokers when raise a support ticket.

NOTE: There are two different ways to generate diagnostics files for docker-based containers. The first approach is to extract the utility from the docker container and run from a Linux-based host machine with Python 2.7 or above installed. The 2nd approach is when you are using Docker for Windows or Docker for Mac and generate a sidecar container to run the utility from.

We will use the 2nd approach in this exercise and assuming you are running your PubSub+ event broker on Docker for Mac. If you are using Docker for Windows, and running it another docker runtime environment, then refer to the documentation for complete instructions.

1. Open a command terminal to access the docker container
2. To use this utility in Docker for Mac, create a new sidecar container with privileged settings based on the software event broker Docker image with a different ENTRYPOINT command (gather-diagnostics-host)..

   NOTE: update the highlight text below to output the file to your specified directory, and PubSub+ version.

   ```
   docker run --rm -t --privileged --volumes-from solace -v
   /Users/username/storage/diags:/tmp -v /var/run/docker.sock:/var/run/docker.sock
   solace-pubsub-standard:8.x.x.x /usr/share/solace/gather-diagnostics-host --
   sidecar
   ```

3. This file copied to the folder you specified above can then be provided to Solace support team when raising a ticket by uploading it to https://filedrop.solace.com

You have now completed the Gathering Diagnostics activity.

Additional Learning Resources

Practice will help you learn and understand PubSub+ concepts better, so here are some additional learning resources to follow-up on.

1. Event-Driven Microservices
2. Hybrid-Cloud
3. PubSub+ for Developers
4. Solace Community
5. Developer CodeLabs
6. Solace YouTube Channel

GitHub Resources

Solace Products

Repositories containing open source Solace Products. Here you'll find quick-start code to help you configure the PubSub+ Event Broker with Kubernetes,VMware Tanzu, OpenShift, Spring Boot, AWS, Azure, and more.

Learn more at github.com/SolaceProduct

Solace Labs

Includes reference demos, examples, integration guides, and more, created by the Solace Community.

Learn more at github.com/SolaceLab

Solace Samples

Solace developer samples that show you how to get started using PubSub+ Event Broker with many of the Solace Messaging APIs and protocols.

Learn more at github.com/SolaceSamples