

Lab 6: Serverless Monitoring

Hands-on Lab

Released:

Terms of Use

© 2017 Microsoft Corporation. All rights reserved.

Information in this document, including URL and other Internet Web site references, is subject to change without notice. Unless otherwise noted, the companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place, or event is intended or should be inferred. Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

For more information, see **Microsoft Copyright Permissions** at [**http://www.microsoft.com/permission**](http://www.microsoft.com/permission)

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

The Microsoft company name and Microsoft products mentioned herein may be either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

**This document reflects current views and assumptions as of the date of development and is subject to change.  Actual and future results and trends may differ materially from any forward-looking statements.  Microsoft assumes no responsibility for errors or omissions in the materials.**

**THIS DOCUMENT IS FOR INFORMATIONAL AND TRAINING PURPOSES ONLY AND IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT.**

Table of Contents

[Lab Overview 1](#_Toc510083468)

[Exercise 1: Functions and Application Insights 2](#_Toc510083469)

[Exercise 2: Logic App with trackedProperties and OMS 12](#_Toc510083470)

# Lab Overview

###### Abstract

In this lab, we will be reusing some of the previous lab components with Azure Functions and extending them to apply Application Insights and quickly demonstrate the basic functionality of monitoring. This lab will follow some of the components of Lab 3.

###### Learning Objectives

After completing the exercises in this lab, you will be able to:

* Understand how to enable Application Insights for an Azure Functions application and how to do basic monitoring.
* Understand how to utilize Operations Management Suite for inclusion in a serverless solution.

**Estimated time to complete this lab: *30* minutes**

# Exercise : Functions and Application Insights

#### Scenario

In this exercise, you will be extending an Azure Function App to include Application Insights. In Lab 3 we used the Event Grid to send events to an Azure Function and a Logic App. The Events were created from a Windows Console App. In this lab we will create an additional Azure Function with Application Insights enabled and then use the Console App to send a large number of events through to the Function App. We will also code the Function App to simulate longer run times and failures to see these events in App Insights telemetry.

After completing this exercise, you will understand:

* Application monitoring of Azure Functions with Application Insights.
* Live Stream feature of Application Insights.

#### Prerequisites:

To complete this lab, you should have a basic understanding of Azure Portal. You should also have completed Lab 3.

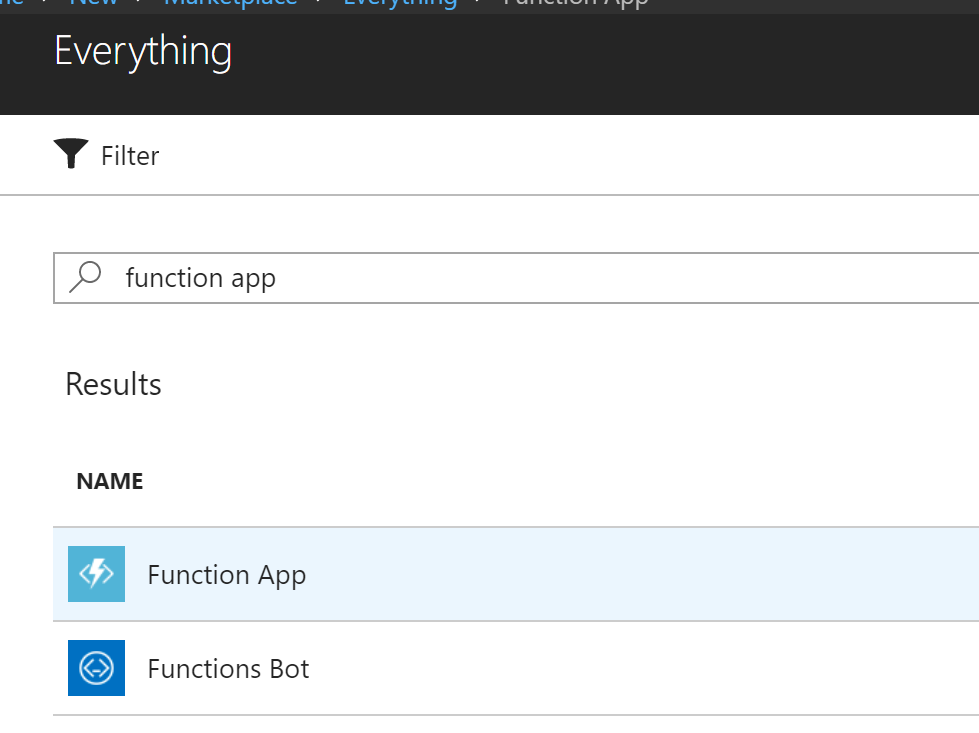
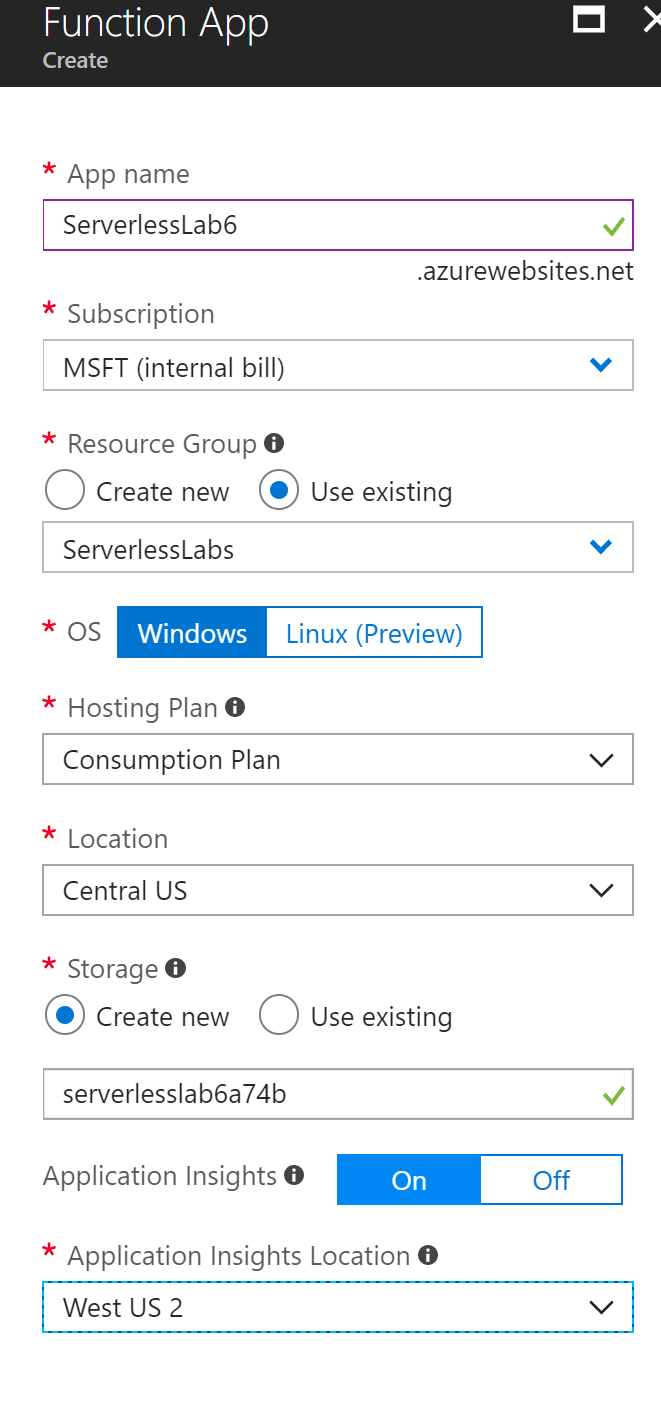
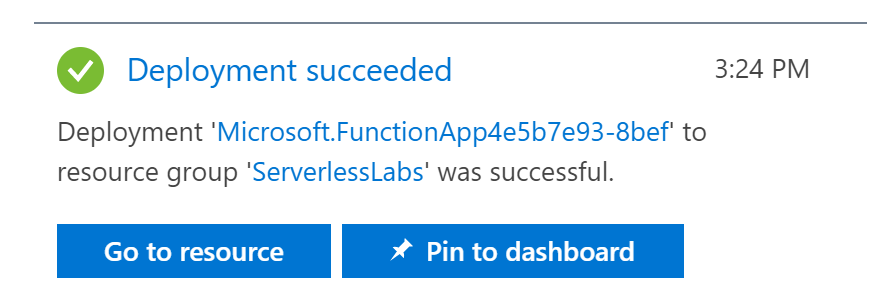
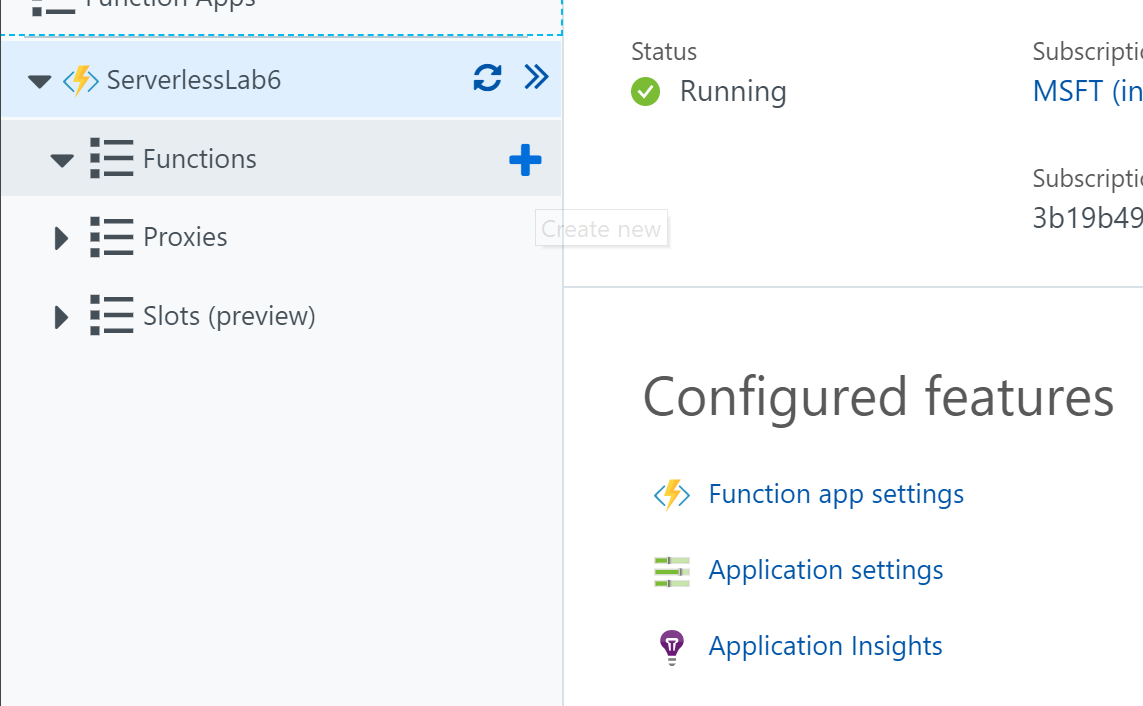
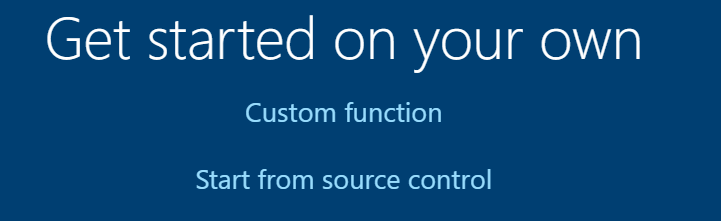
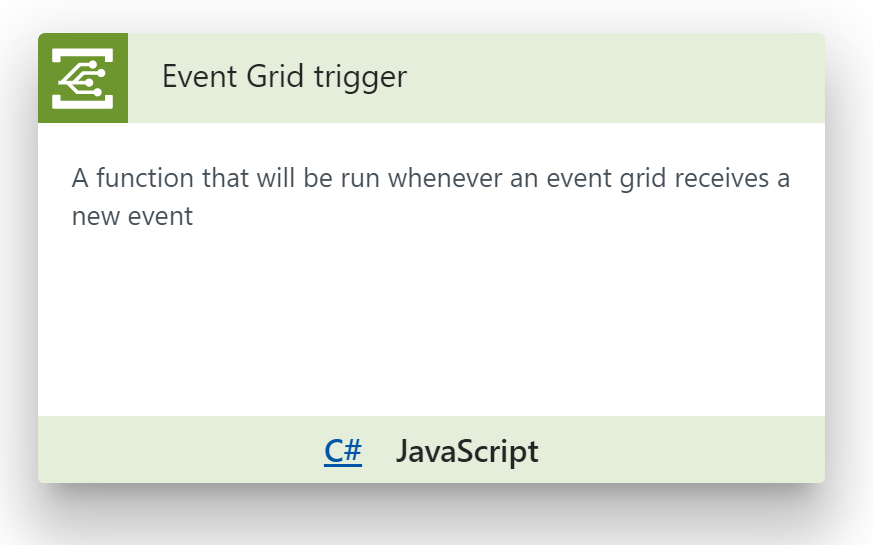
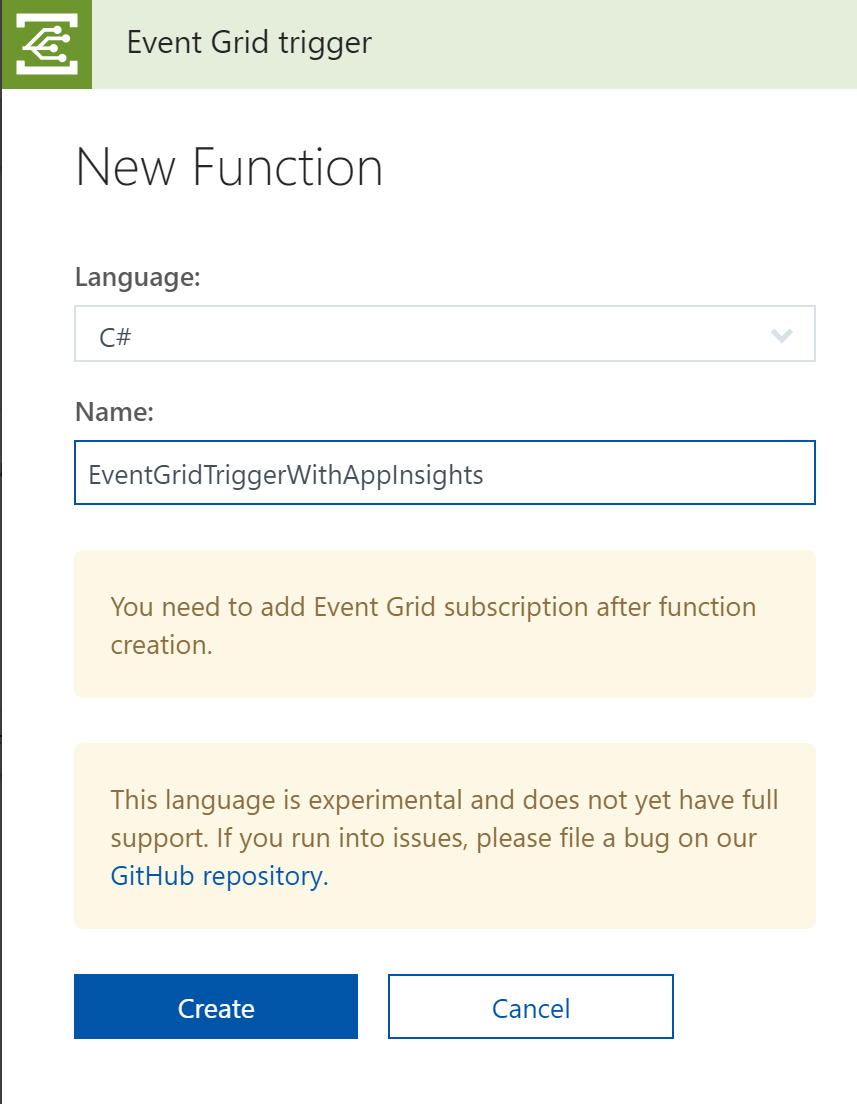
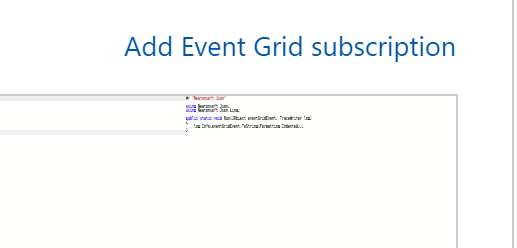
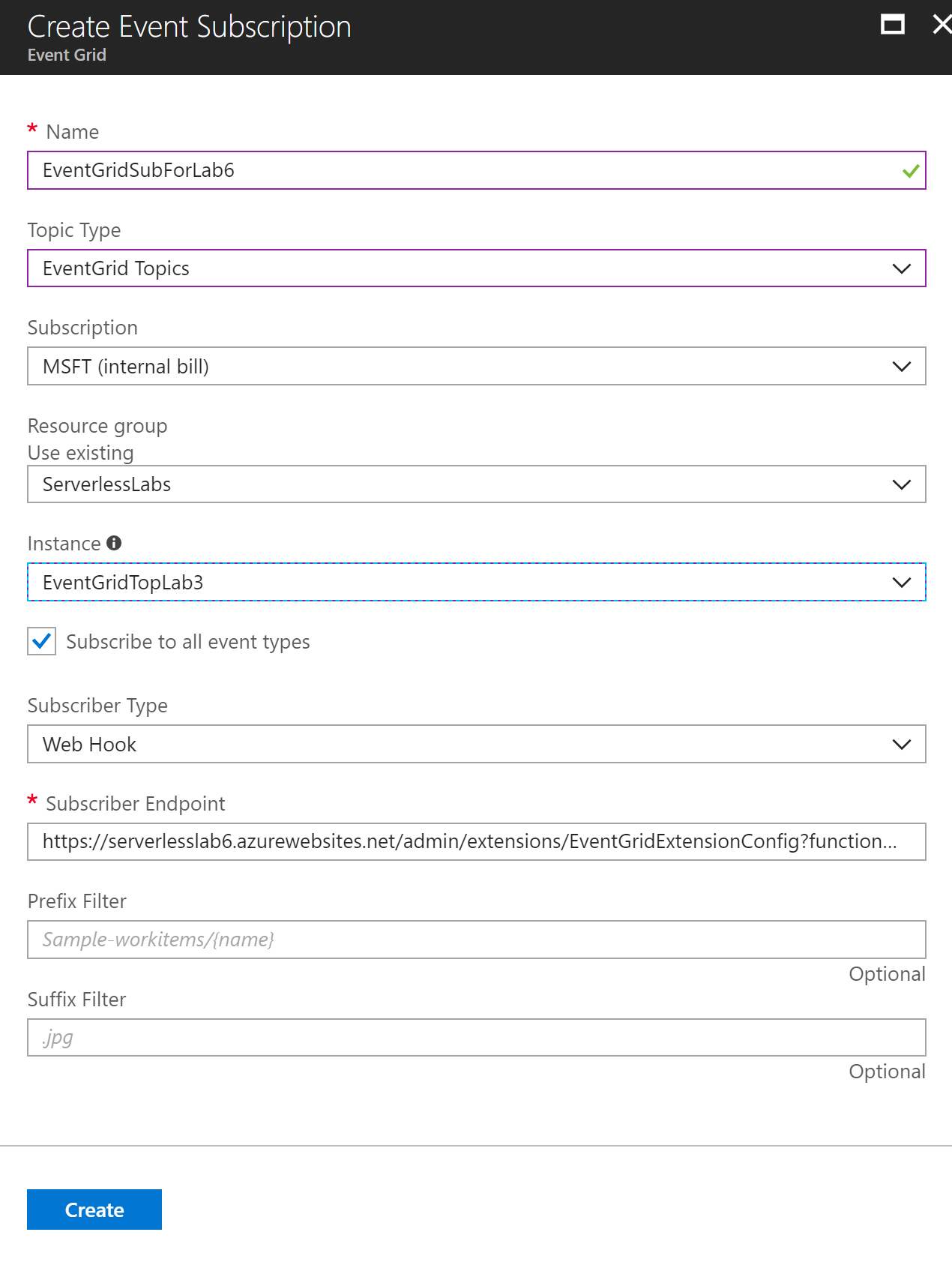
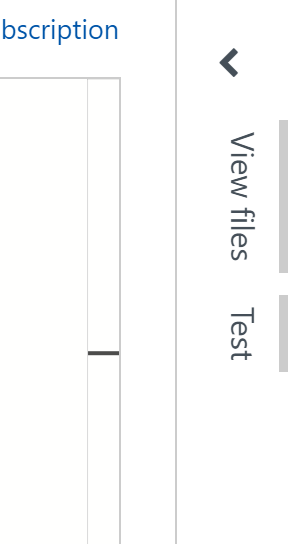
#### Install and configure:

You will need:

* A desktop computer (host) running Windows 10.
* An active Azure subscription (MSDN credits will suffice)
* An active Internet connection

#### Create Function App with Application Insights

This Lab will make use of some of the components used in Lab 3, specifically the Event Grid and later the Logic App. Since you are adding a new Function to the Event Grid and sending it more event, the old Function and Logic App will also be trigger. You can either disconnect those, enable a filter on them or just let them run as you push additional events to the Event Grid.

1. Navigate to the Azure Management Portal (http://portal.azure.com) and search for: Function App.  
   
2. After locating Function App in the search results, click **Create** to how the create Function App blade.
3. Next, fill out the blade with the Function App information, be sure to **select Application Insights On** and then click **Create.**   
   
4. Once the Function App is created, Navigate to it.  
   
5. You should notice under the **Configured features** heading the Application Insights, indicating that Application Insights has been installed. Now click on the **Plus sign** next to the Functions heading to create a new function.  
   
6. You will want to create a new Event Grid function, click on **Custom function**, and then select **Event Grid trigger**.  
     
   
7. Fill in the New Function form with the **Name** for the function and click **Create.**  
   
8. Click **Add Event Grid subscription** in the upper right corner to configure the Function to connect to the Event Grid that you previously had created in Lab 3.  
   
9. Create the subscription to the Event Grid Topic from Lab 3. Set the **Topic Type** and **Resource Group** to be able to select the correct **Instance**. Use a unique **Name** that represents this Lab. The **Subscriber Type** will be Web Hook and the **Subscriber Endpoint** will automatically populate. Once completed click **Create**.  
   
10. Now let’s re-use the same JSON test document from Lab 3 and do some test runs on the function and then test the Event Grid submission app. Click the **Test** label on the right had side of the screen.  
    
11. Replace the **Request body** with the following text.

{"data": {

"orderid": "abc1",

"itemid": 123,

"itemdesc": "Fluffy Bunny",

"quantity": 1,

"itemprice": 9.99,

"extendedprice": 29.97,

"customerid": "61072b3d-50b7-41ee-beec-ed191162177b"

},

"id": "f65c40f0-3db9-4c90-acd0-fcf33c7bc540",

"subject": "serverless/training/lab3/order",

"eventType": "orderSubmit",

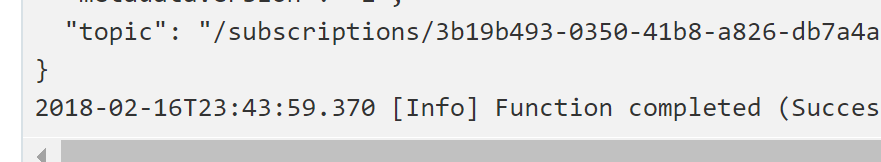
"eventTime": "2018-02-09T05:18:22.0210628Z",

"dataVersion": "",

"metadataVersion": "1",

"topic": "/subscriptions/3b19b493-0350-41b8-a826-db7a4abb9f10/resourceGroups/ServerlessLabs/providers/Microsoft.EventGrid/topics/EventGridTopLab3"

}

1. Then click the **Save and Run** or **Run** button to validate that everything is currently working. You should see the following output in the Logs section of the screen.  
   
2. Now add the following 3 code lines to the bottom of the Function, which will let the function throw some exceptions and also delay the executions based on the input data.

dynamic orderEvent = eventGridEvent;

if ( (((int) orderEvent.data.quantity) % 4) == 0 )

throw new Exception("random error");

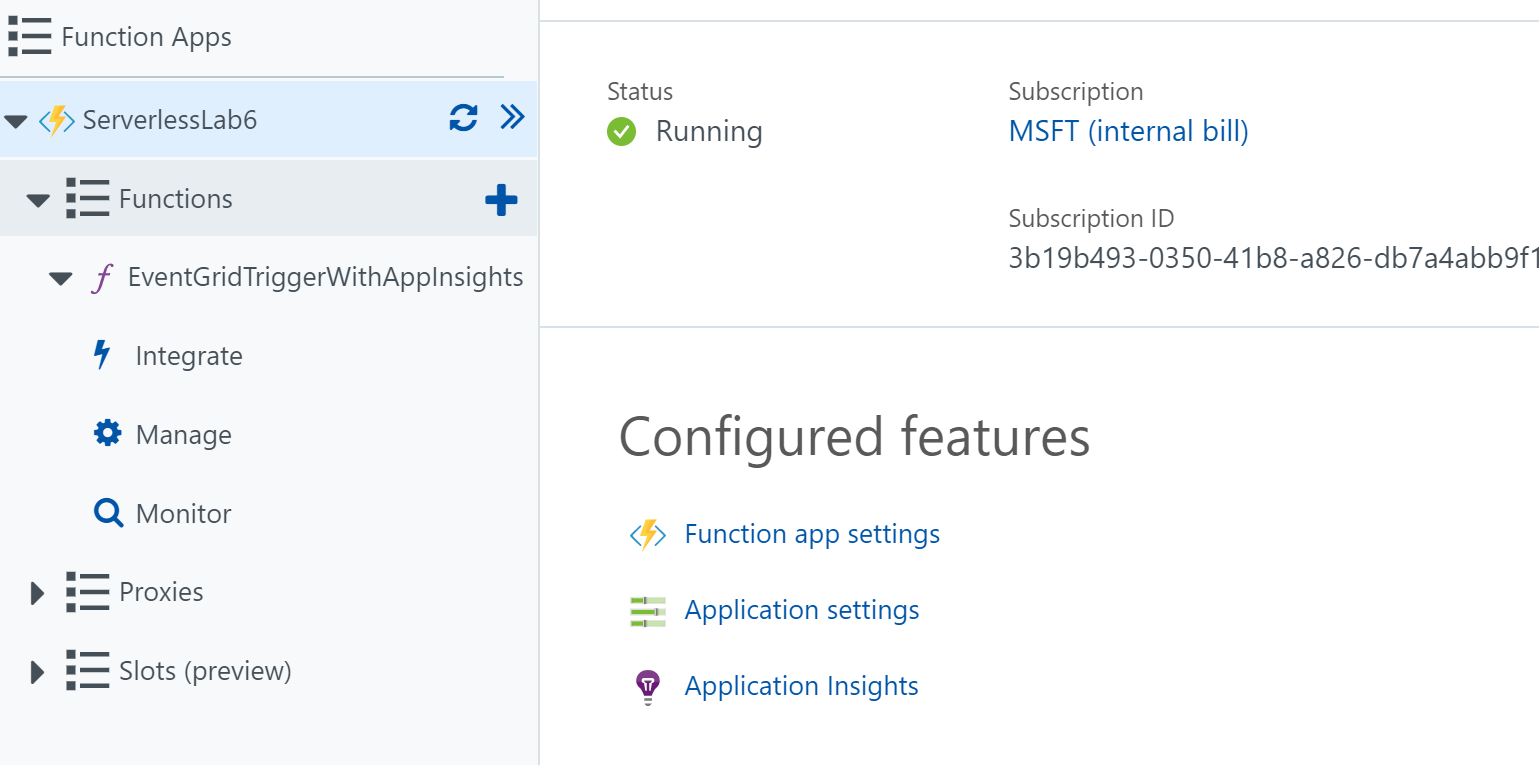
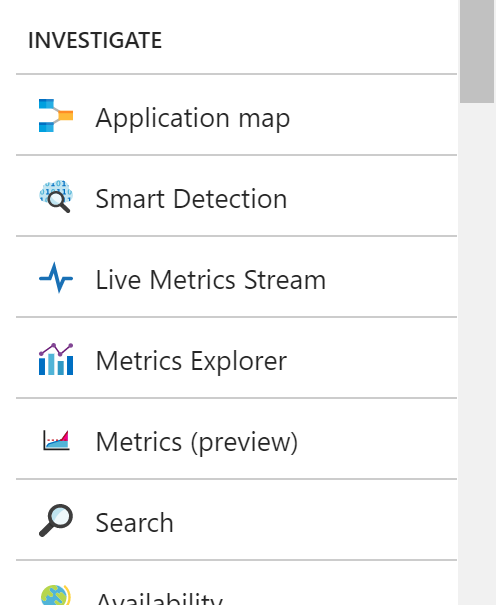
System.Threading.Thread.Sleep((int) orderEvent.data.quantity \* 50);

1. Test the Function again to make sure it is still compiling and executes successfully. Change the input quantity to 4 and the Function should throw an exception.
2. Click **Run** one last time so that the Function will be running and we can go start to feed some inputs to the Function.

#### Executing the Event Submissions

1. In Lab 3 you would have downloaded the Console App submit events to your Event Grid. If you still have that downloaded just reuse all of that. If not follow these directions to download it again.
2. You’ll want to download the source code and utility that can POST event to the Event Grid Topic. You can find that at <https://github.com/criter/PostToEventGridTopic/> or download the [tree as a zip file](https://github.com/criter/PostToEventGridTopic/archive/master.zip). Either way you can open it up with Visual Studio on your machine after you have downloaded it from github. If you do download the zip file make sure you Unblock the file by right clicking it in Explorer, selecting Properties and checking the Unblock checkbox at the bottom.
3. You can see via the README.txt file that there is a precompiled version of the program in the binn folder. Open a command prompt and navigate to the binn folder.
4. You’ll want to execute the **SubmitToEventGridTopic.exe**. As noted in the README.txt file this takes 3 parameters.

* Event Grid Endpoint URL – this can be found by going back into the properties of the Event Grid Topic that was created previously, click the **Overview** property and look for the **Topic EndPoint**, copy this value.
* SAS Key – this can be found by clicking the **Access keys** property on the same Event Grid Topic page and then copying either **Key 1** or Key 2.
* Repeat – this value is the number of times you want the program to repeat submitting the message.

1. Just to test things execute the SubmitToEventGridTopic.exe with a **Repeat** value of **2** and watch the Logs window in the Functions app portal for the executions. You can see them execute because the orderid and the quantity will change based on the repeat value. If everything is working successfully, we can move on and start sending many more messages through with much larger repeat values. ***Be sure to leave this command window open you will be re-using it later in a later exercise.***
2. Next in the Portal navigate to the Application Insights for your Function App. You can get to this by navigating to the Function App and then under **Configured features** click on **Application Insights.** This will take you to the Application Insights portal.   
   
3. Next, click on the **Live Metrics Stream** on the left.  
   
4. With the Live Metrics Stream showing, execute the console app with a **Repeat** value of **400** or some other large value (if you didn’t disconnect the Logic App, be aware this will generate an number of emails.)
5. You should quickly seem some graph data in the Live Metrics Stream. There will be many incoming requests and the response times should generally be going up due to the logic we put into the Function with the sleep timer. There will also be a number of exceptions and failures due to our simulated code. CPU will be consumed as well due to the exception handling as well. Clearly this is a simplistic case but enough to demonstrate how you will be able to get some basic telemetry out of a Function app. In the upper right corner it will keep track of how many servers are currently online for your Function app. In this scenario there is only one.  
   

# Exercise 2: Logic App with trackedProperties and OMS

#### Scenario

In this exercise, you will be adding additional information to your Logic App so that it can show up in Log Analytics. This additional information uses what is called trackedProperties and can then be utilized in Log Analytics and OMS to build dashboard and support specialized queries for activities.

After completing this exercise, you will understand:

* How to use the Code View in Logic Apps
* How to add trackedProperties to you Logic App
* How to reference other properties in the workflow definition language
* How to build a query in Log Analytics

#### Prerequisites:

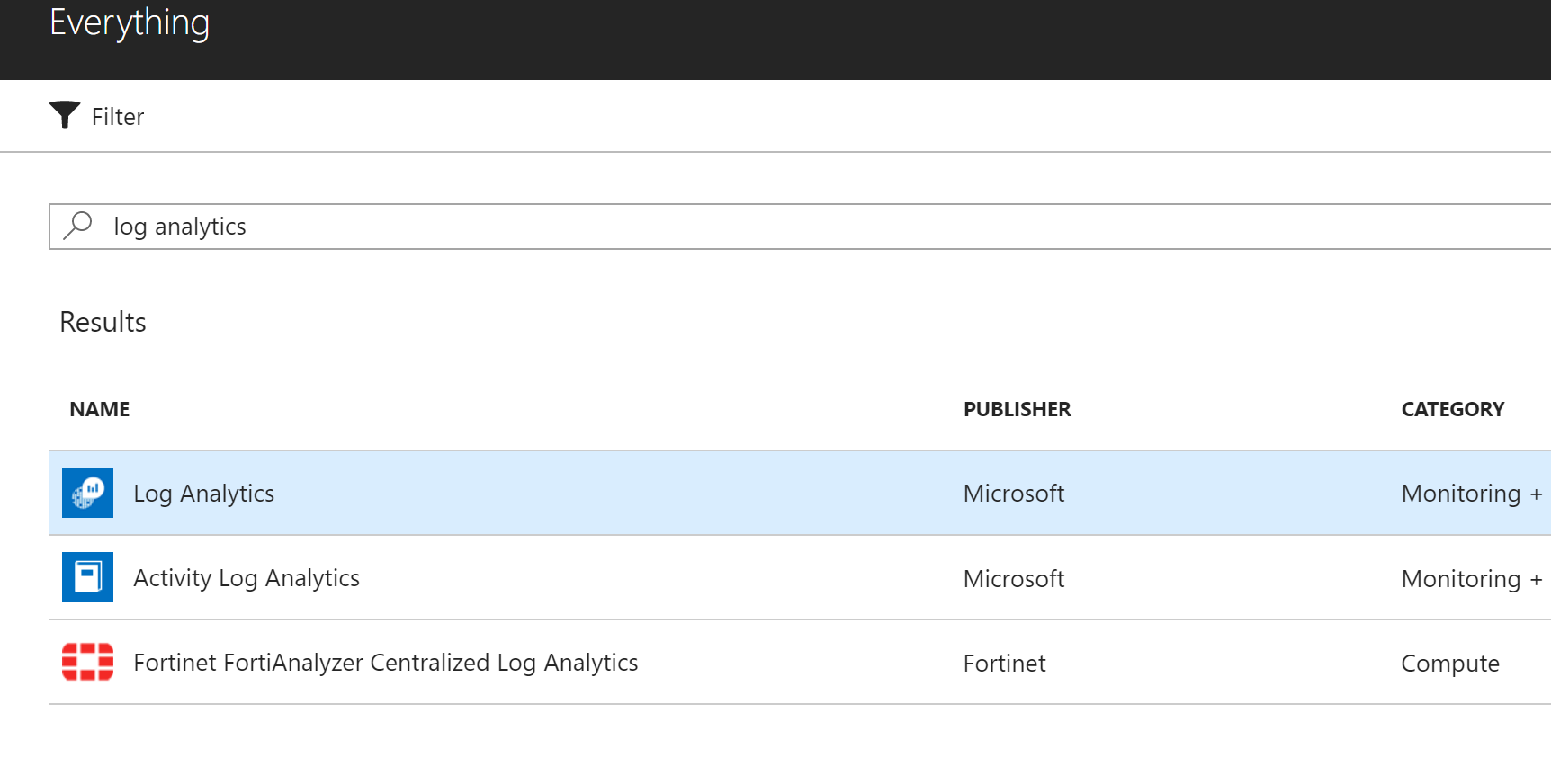
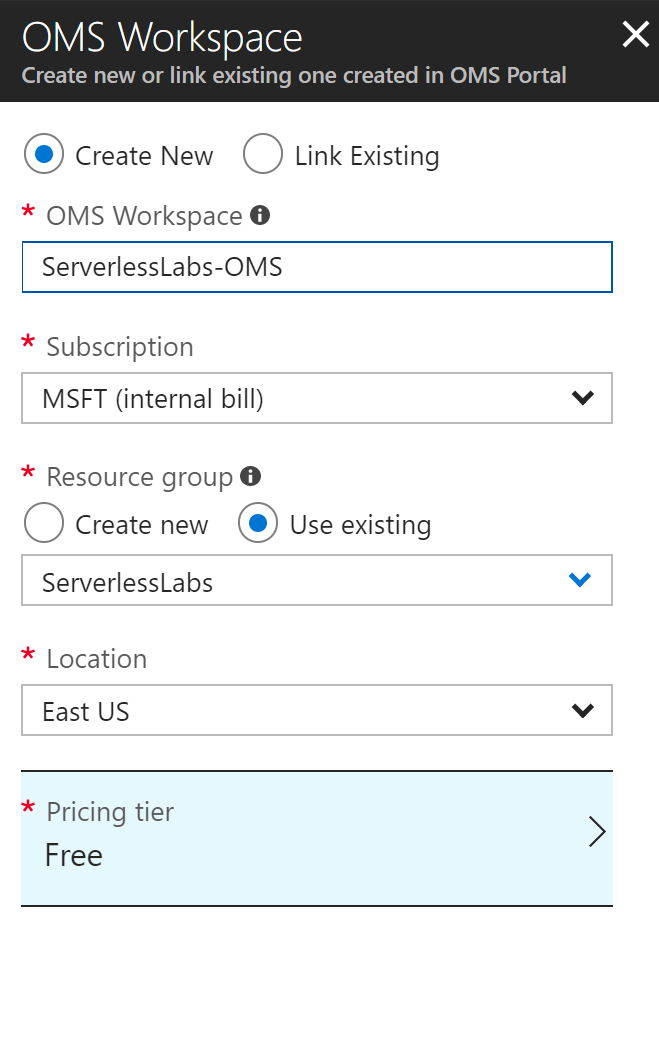
To complete this lab, you should have a basic understanding of Azure Portal.

#### Install and configure:

You will need:

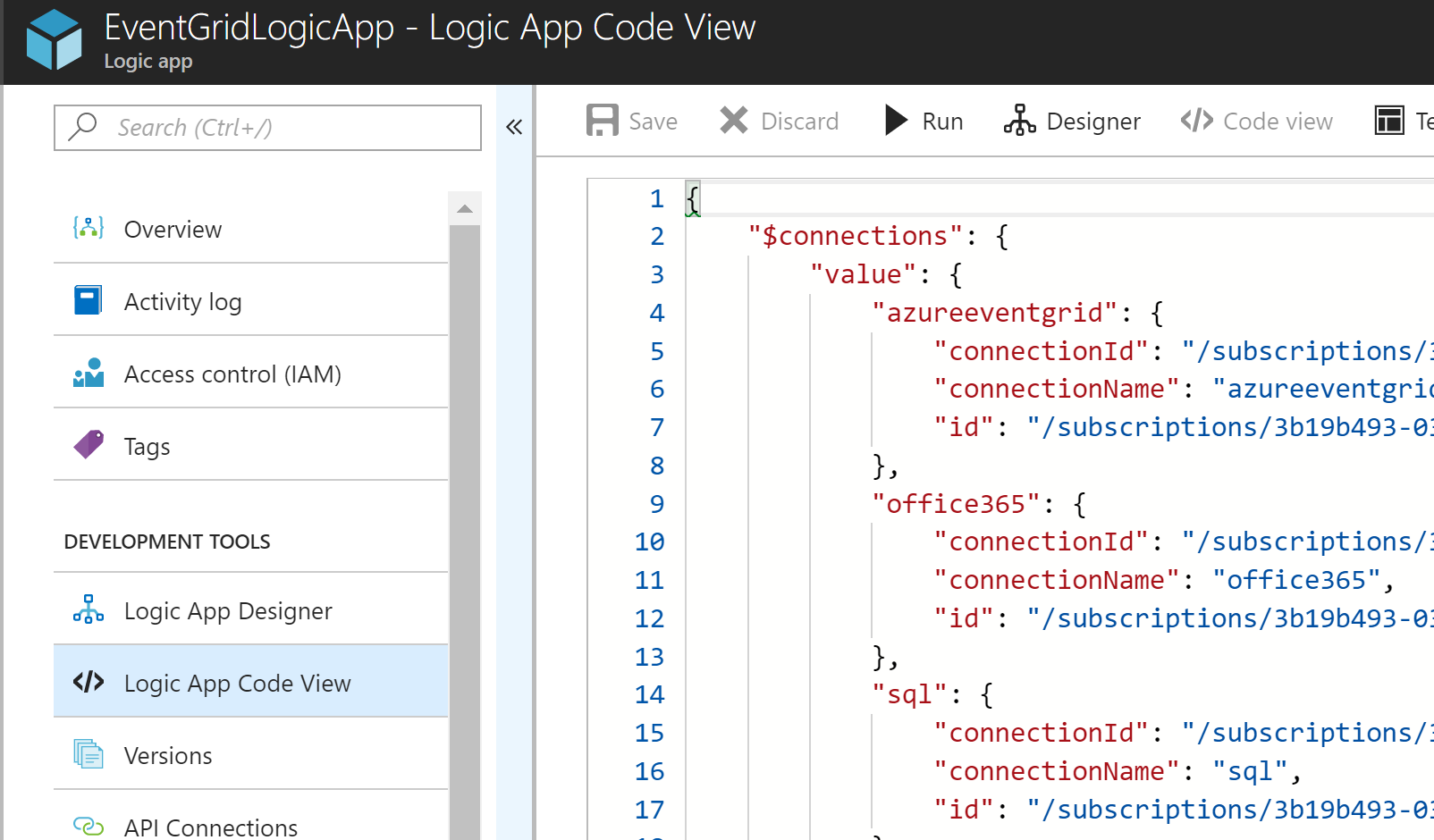
* A desktop computer (host) running Windows 10
* An active Azure subscription (MSDN credits will suffice)
* An active Internet connection

#### Creating Operations Management Suite for Log Analytics

1. Create a Log Analytics and OMS Workspace instance. In the upper left hand corner click **Create a resource** and then search for **Log Analytics** and select the line and click **Create**.  
   
2. Next fill out the OMS Workspace blade. Create a unique **OMS Workspace** name, select your **Subscription** and use an existing **Resource Group** for ease of management. Select the appropriate **Location** for your area and use the Free **Pricing Tier**. Then click **Create**.  
   
3. We will let the OMS workspace finish deploying while we make changes to the Logic App.

#### Add trackedProperties to Logic App

*You should still have you Logic App that you used from Lab 3. We will extend that and add the trackedProperties to that one. If you don’t have Lab 3 still available, you can download the JSON for the Logic App* [*here*](https://gist.github.com/criter/033e16094ddda3f15bb5590ead50b995)*. Note that you will need to update the connections. You can use the Code view editor to paste in the JSON.*

1. Navigate back to the Logic App from Lab 3. Open the editor and go to **Code view**.  
   
2. Now you need to scroll down in the code editor to the **Insert\_row** action and after the **runAfter** and before the **type** properties, you need to insert the **trackedProperties** property, as shown.  
     
   the text for the trackedProperties is

"trackedProperties": {

"customerid": "@{body('Insert\_row')?['customerid']}",

"extendedprice": "@float(body('Insert\_row')?['extendedprice'])",

"itemid": "@int(body('Insert\_row')?['itemid'])",

"orderid": "@{body('Insert\_row')?['orderid']}"

},

1. Next you’ll want to insert the trackedProperties with the Parse JSON action. So, scroll down to the **Parse\_JSON** action and paste in the follow text just after the runAfter property in the Parse\_JSON action.

"trackedProperties": {

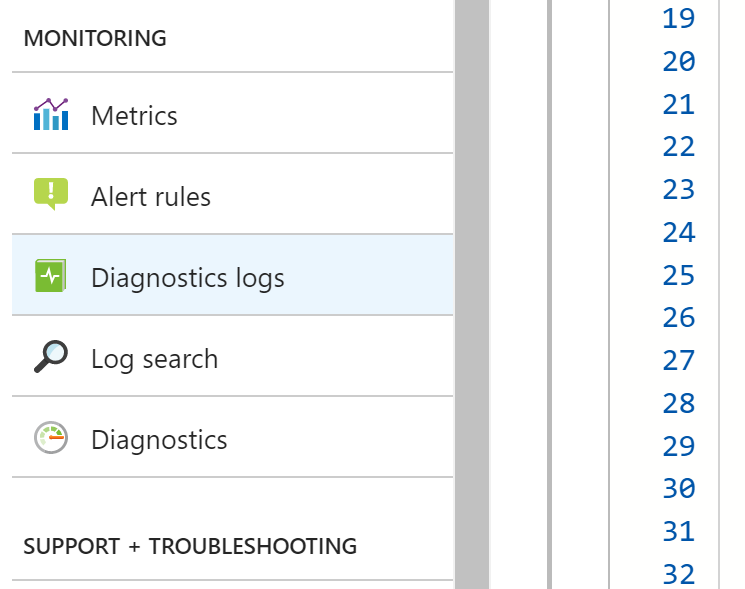
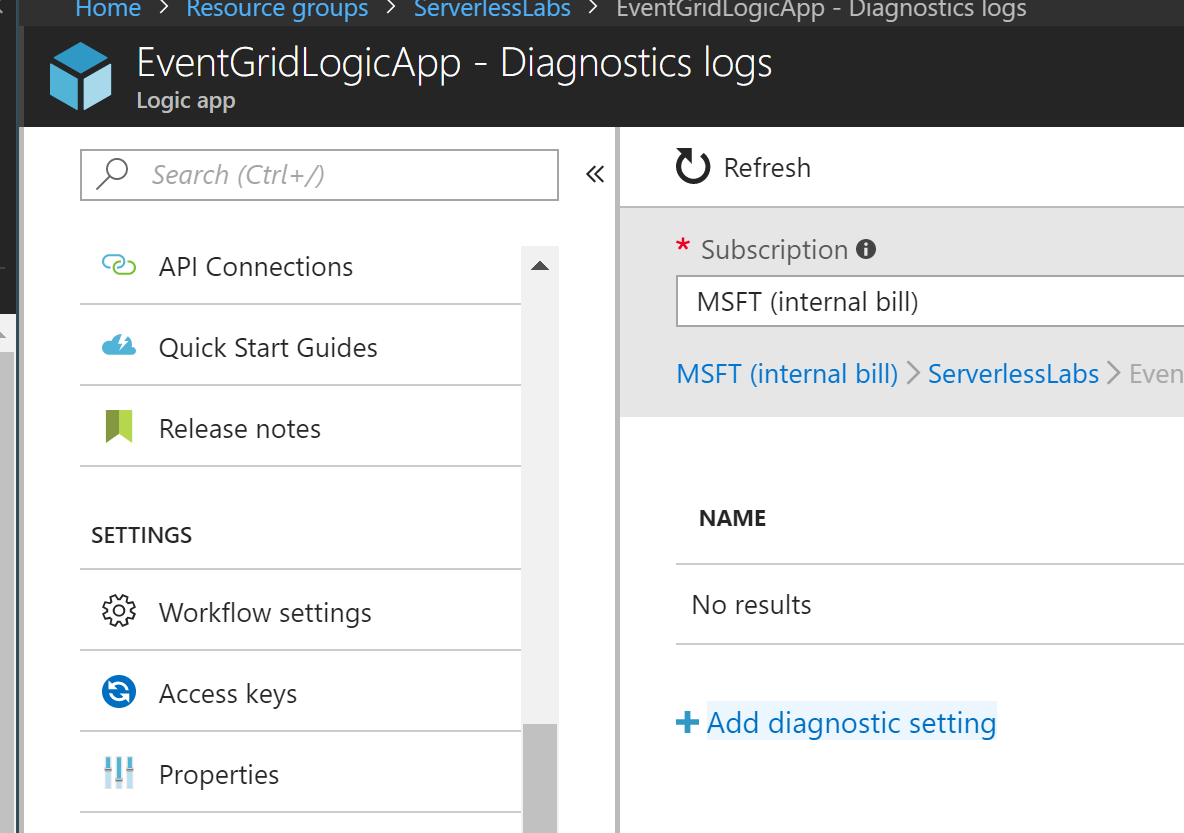
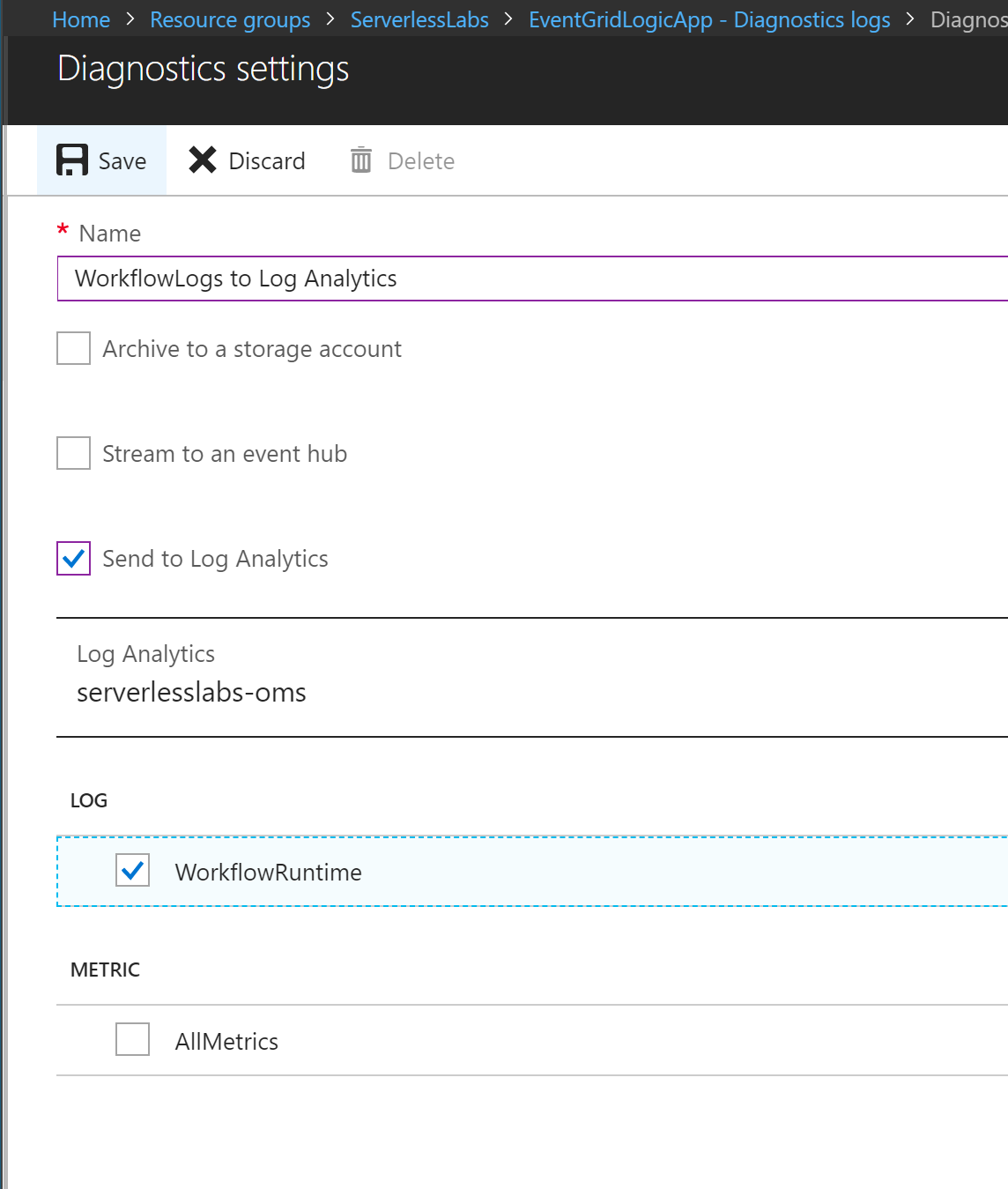
"customerid": "@{body('Parse\_JSON')?['data']?['customerid']}",

"extendedprice": "@float(body('Parse\_JSON')?['data']?['extendedprice'])",

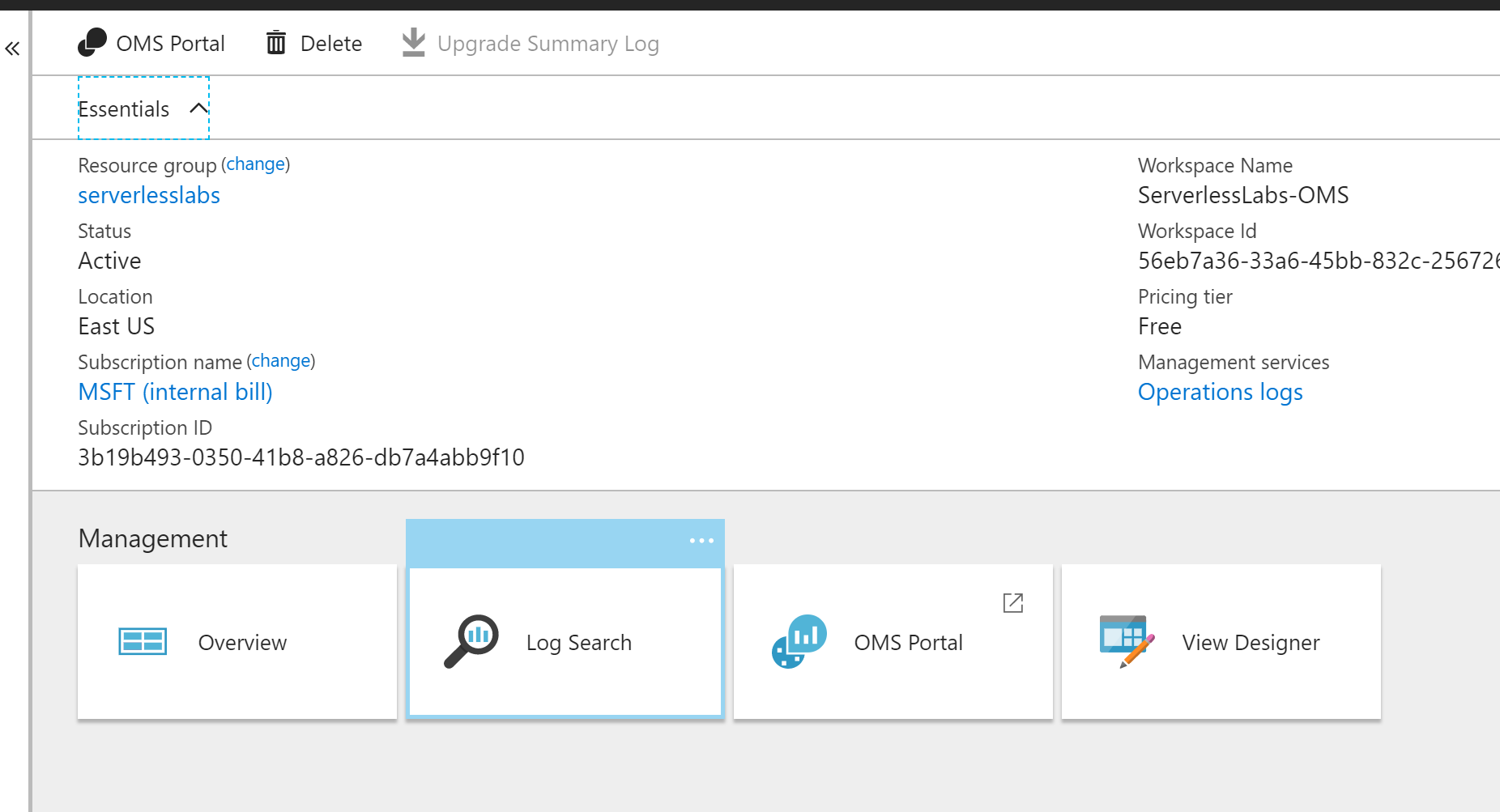
"itemid": "@int(body('Parse\_JSON')?['data']?['itemid'])",

"orderid": "@{body('Parse\_JSON')?['data']?['orderid']}"

},

1. Then **Save** the code which should validate successful. If there are errors double check what the validate error is and correct the code.
2. Next, you need to connect the Logic App to the OMS Workspace. On the left-hand side, under the **Monitoring** section, click the **Diagnostics logs** link.  
   
3. Click the **Add diagnostic setting** link to create a link to send the logs to the OMS Workspace.  
   
4. Enter a **Name** and check the **Send to Log Analytics** and configure the **OMS Workspace** previously created. Check the **WorkflowRuntime** which will send all of the Workflow Runtime steps including the Tracked Properties data to Log Analytics. Then click the **Save** button at the top.  
   
5. To view this connection in the list, after you close the create window you need to click the **Refresh** button at the list of the Diagnostics logs list.
6. With the connection now in place, we need to send some transactions through, so the executions will be made, and the logs will be sent to the OMS Workspace. It usually takes about 5 minutes from the Logic App executions until they can be queried in OMS.
7. You should still have the command line open from Exercise 1. Re-run the **SubmitToEventGridTopic.exe**, using a repeat value of 10 so that you have a handful of records to look at.

#### Review records in Log Analytics

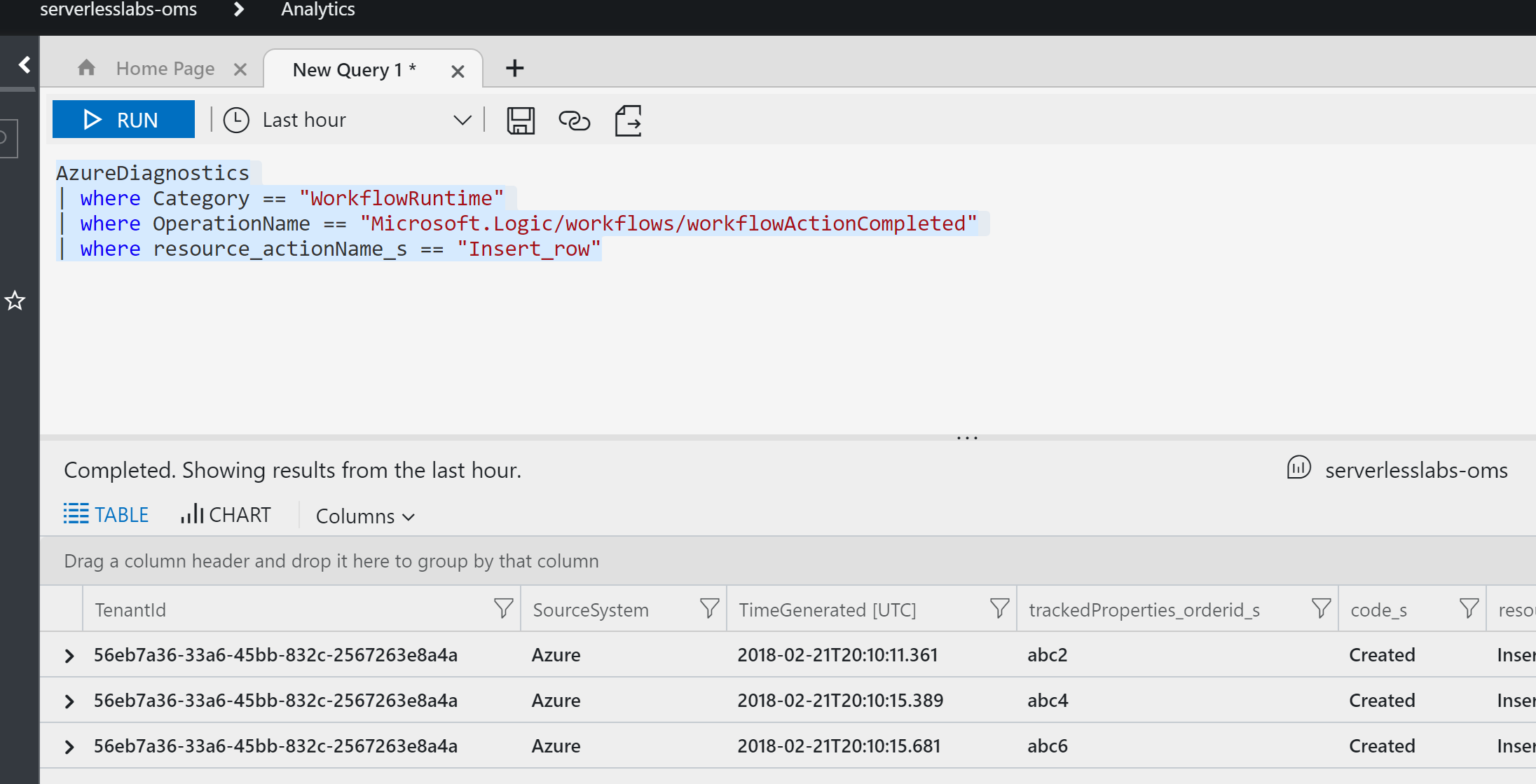
1. Navigate to Log Analytics created in the first section of this Exercise. You can do that by going to the Resource Group you created it in, and then selecting the **Log Analytics** type.  
   
2. Then click **Log Search**.  
   

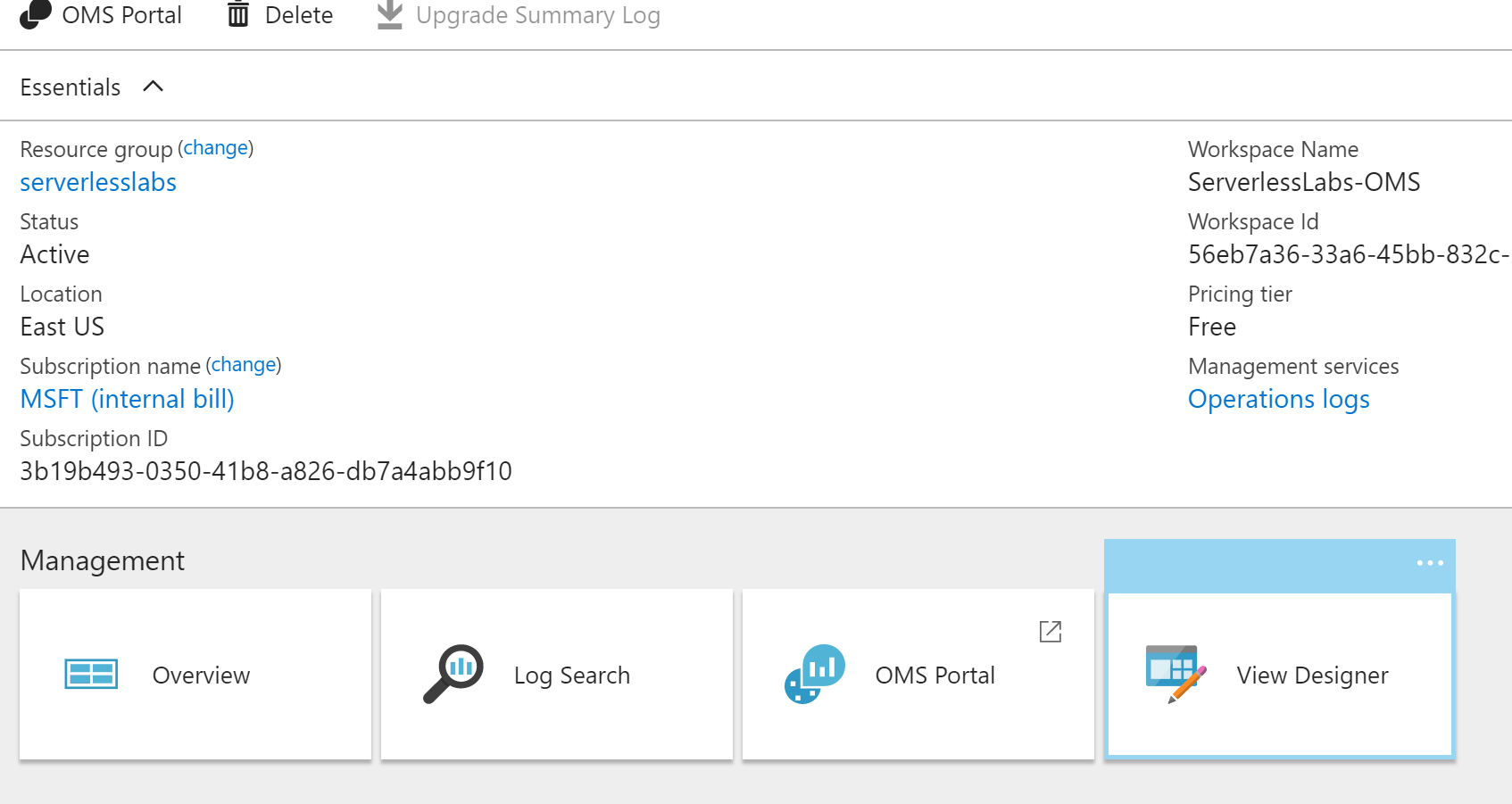
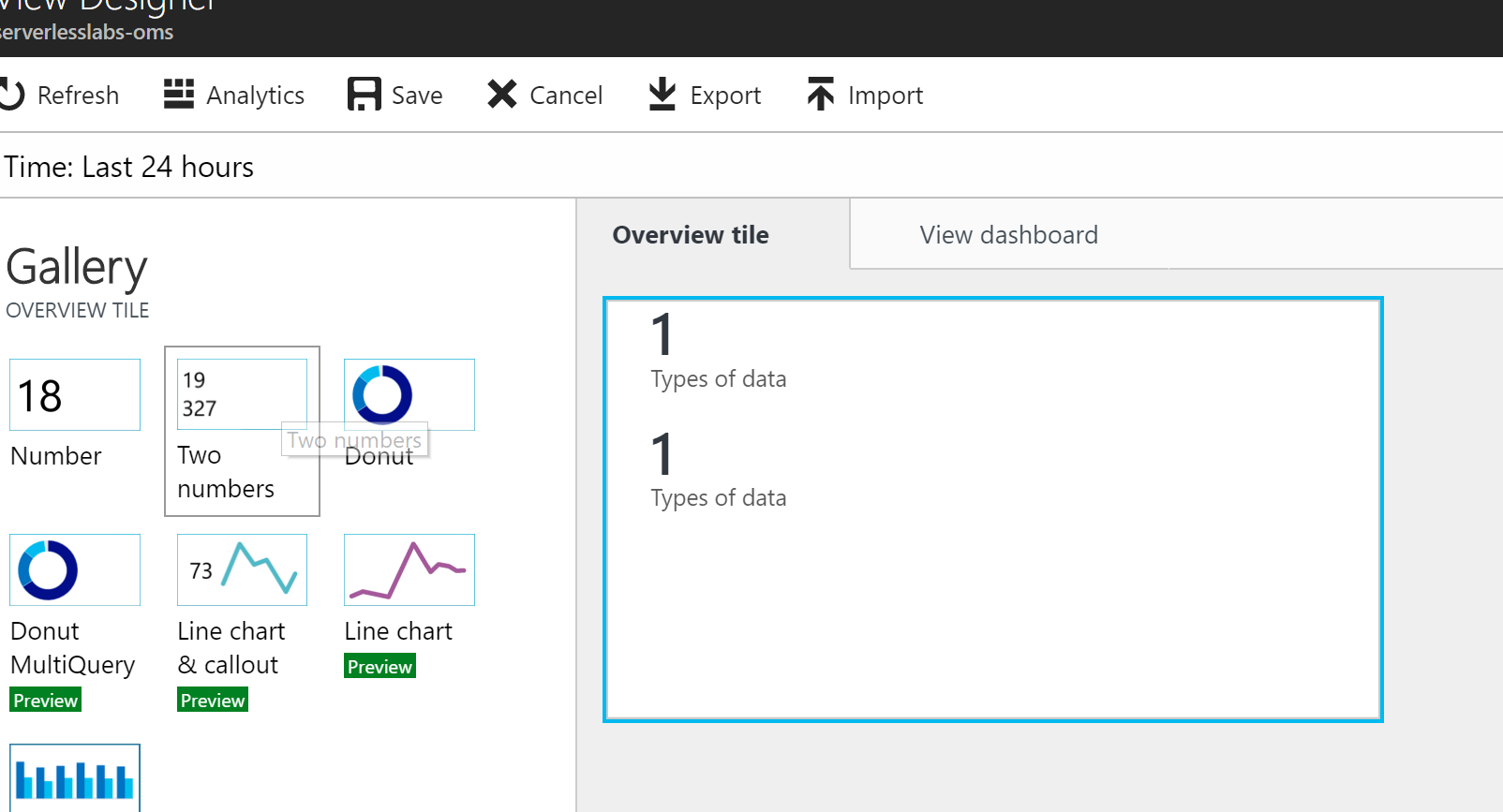
Next click, **Advanced Analytics**, the Advanced Analytics portal has a better query editor. Then click on the **+** to open a new query window and put in the following query.:  
AzureDiagnostics

| where Category == "WorkflowRuntime"

| where OperationName == "Microsoft.Logic/workflows/workflowActionCompleted"

| where resource\_actionName\_s == "Insert\_row"

and then click the **Run** button.  


1. At this point you should be able to see the trackedProperties in the table output.
2. Next you need to create a Dashboard. You should still have the previous Log Search tab open before you opened the Advanced Analytics. Leave the Advanced Analytics open and click back to the open Log Search window. Scroll the window to the left and the click on the **View Designer** button.  
   
3. Select the **Two numbers** tile option.  
   
4. In the designer we can file out the properties to provide the queries for the total number of entry rows for Insert\_row action which will be the number of Sales.

AzureDiagnostics

| where Category == "WorkflowRuntime"

| where OperationName == "Microsoft.Logic/workflows/workflowActionCompleted"

| where resource\_actionName\_s == "Insert\_row" | summarize count()

Next will be the sum of extended price which will give the Total Value of the Sales.

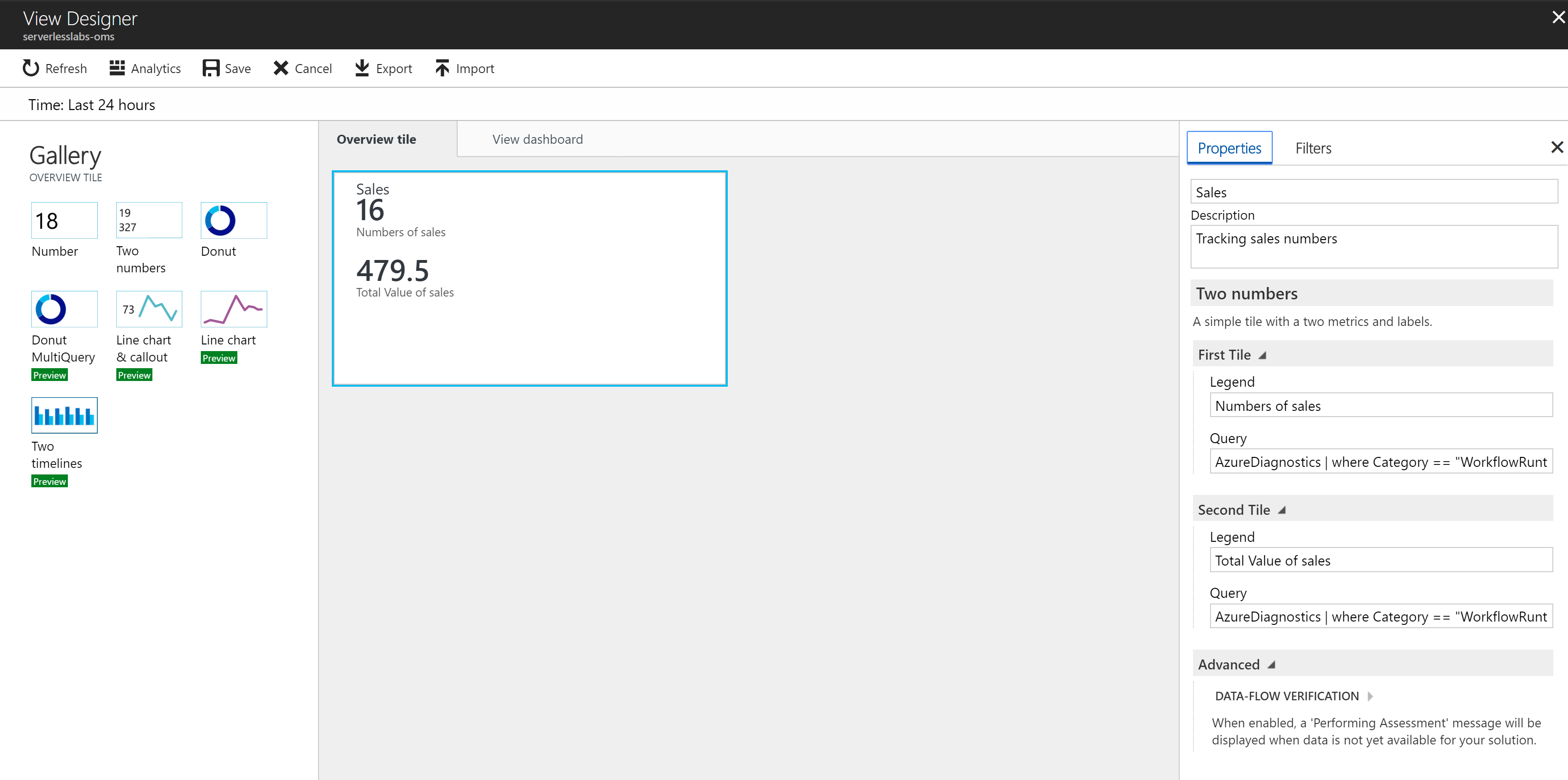
AzureDiagnostics

| where Category == "WorkflowRuntime"

| where OperationName == "Microsoft.Logic/workflows/workflowActionCompleted"

| where resource\_actionName\_s == "Insert\_row" | summarize sum(trackedProperties\_extendedprice\_d)

Each of these queries should be used for the two numbers, the first one for the **First Tile** and the second one for the **Second Tile**.



1. Then click the **Save** button.
2. At this point, you will have created the Dashboard tile using Log Analytics, reading tracked properties from the Logic App.