Building a Bigdata streaming architecture with AWS services and webhook

https://aws.amazon.com/blogs/architecture/field-notes-building-a-shared-account-structure-using-aws-organizations/
reference

https://w.amazon.com/bin/view/AWS/Teams/SA/MediaSeries/AWS_Architecture_Blog/

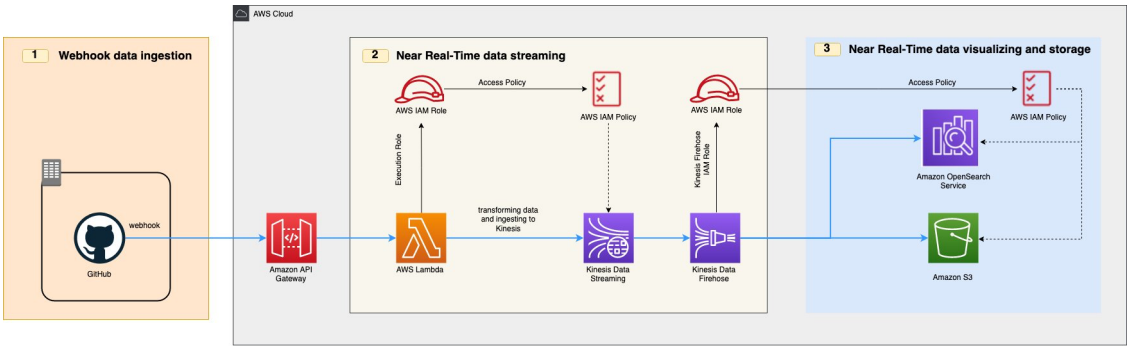
For customers who has 3rd party data present in various services like bitbucket, Rally, JIRA, ServiceNow, GitHub etc. It is very challenging to model a common cloud data streaming architecture. Most of the customer end-up creating a pull based model and face REST API rate limit errors. AWS Kinesis Data Streaming with API Gateway make it possible to build the right architecture to stream the near-real-time events from any data-source using their webhook feature. And Amazon OpenSearch service integration provides near-real-time search feature to their ingested data.

Partners and customers who want to stream their 3rd party data needs to build a new API and link it through the 3rd party service webhook trigger. This is so that all of the events generated on those 3rd party service will be posted to the AWS REST API.

While linking or transferring their 3rd party data to AWS cloud, the customer must check the security settings. It should not compromise any customer security controls and continue to provide full control on data transfer.

Architecture Diagram

The following architecture represents the big data streaming setup which is streaming GitHub webhook events. This architecture can scale for any 3rd party data source which can emit events via webhook events.



GitHub will push each event through webhook to Amazon API Gateway URL. Amazon API Gateway will trigger a lambda function. AWS Lambda function will post the event data to Amazon Kinesis Data Stream which then stream the events using Amazon Kinesis Firehose to Amazon S3 bucket for source of truth data. At the same time Amazon Kinesis Firehose will ingest the same event to the AMazon OpenSearch Service.

Security considerations

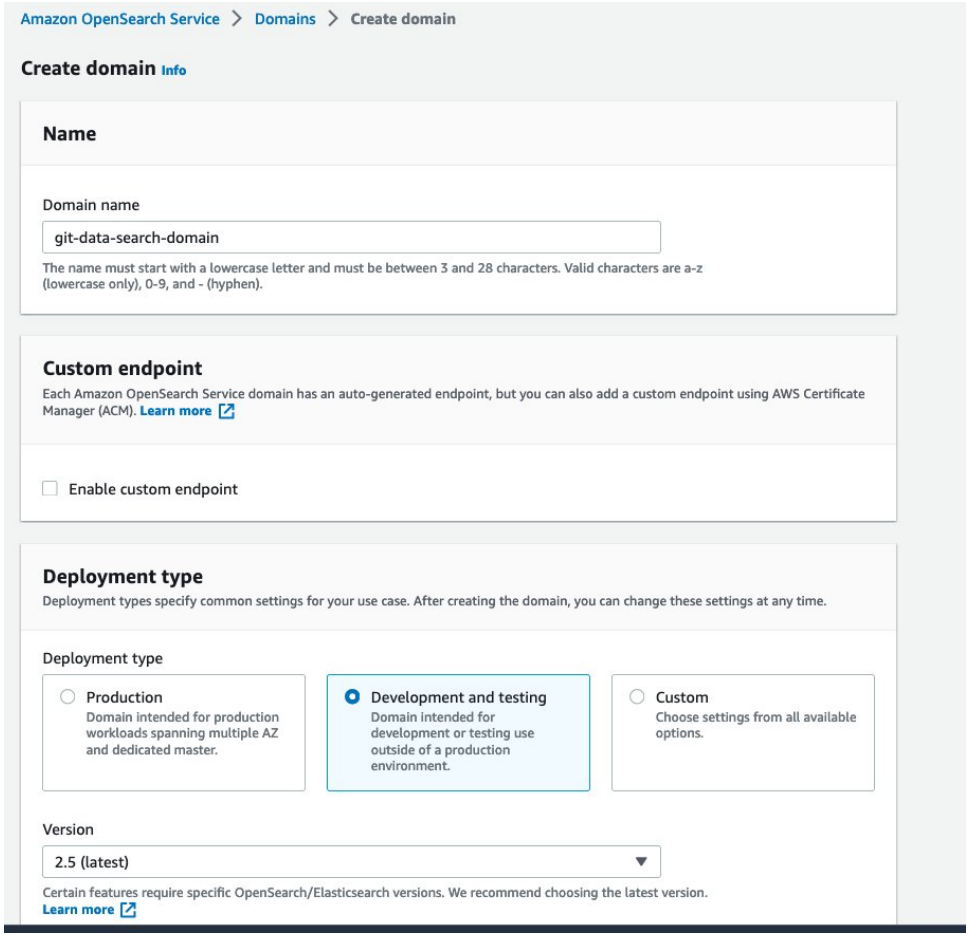
The following table highlights both Amazon Kinesis Firehose and AWS Lambda responsibilities and access controls provided by the architecture in the previous section.

For security reason we create 2 IAM roles one for AWS Lambda execution role and another for Amazon Kinesis Firehose role. Each role will have appropriate permissions like AWS Lambda execution role will have permission to send the data to Kinesis Data Stream and Amazon Kinesis Firehose IAM role will have permission mapping in Amazon OpenSearch as a backend role to have access to create index. Also you can create a secret for webhook data encryption and in AWS API gateway create an Authorizer AWS Lambda function to validate the secret key before ingesting git event.

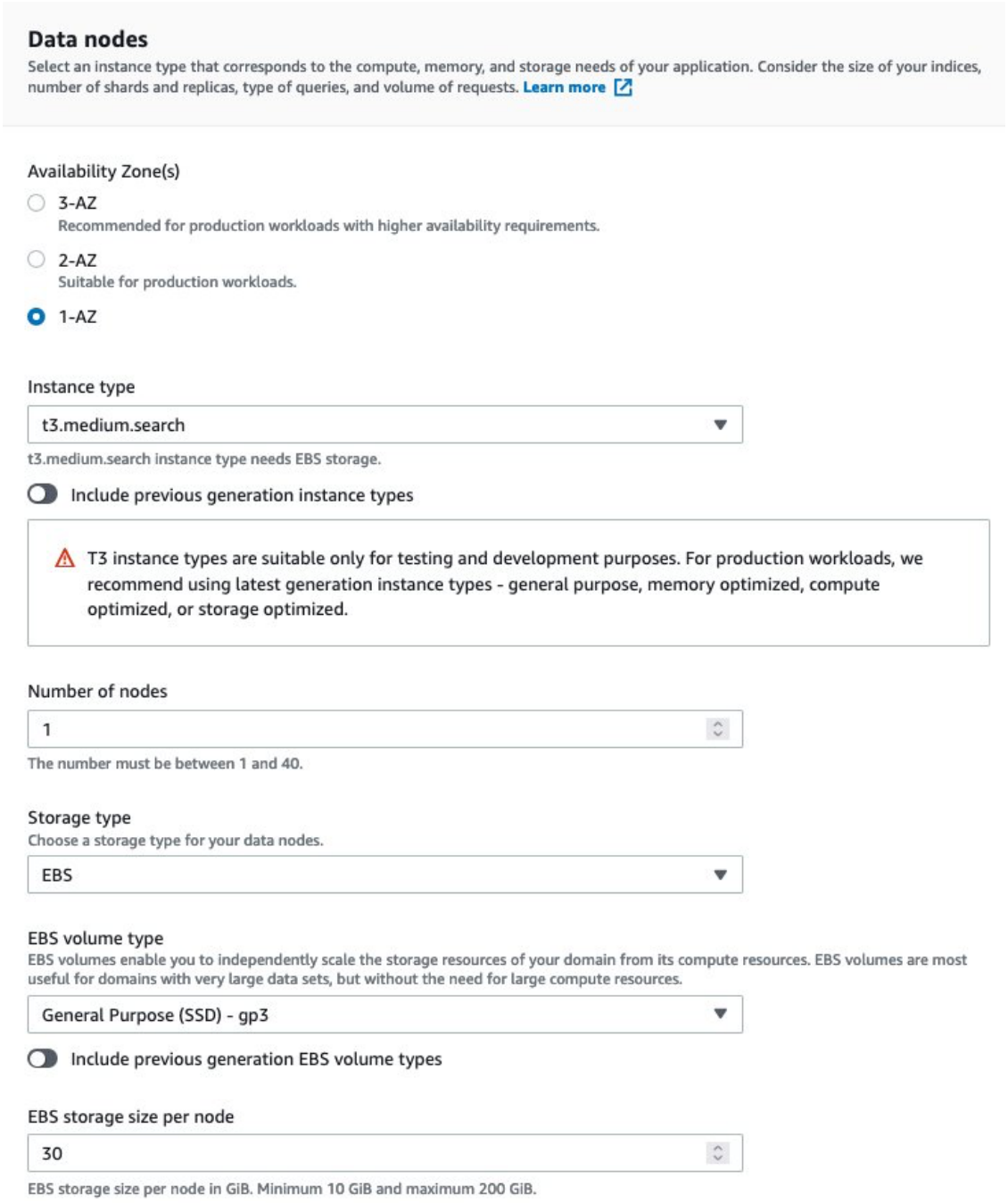
git data streaming with kinesis(2).xml

Steps to create the solution in AWS

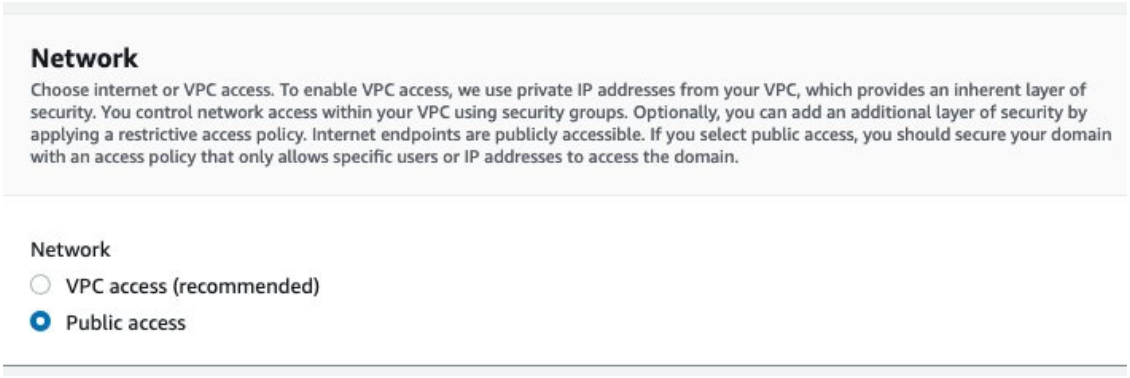
1. Create OpenSearch cluster for this demo. Give the domain name **git-data-search-domain** and deployment type as Development and testing with latest version.



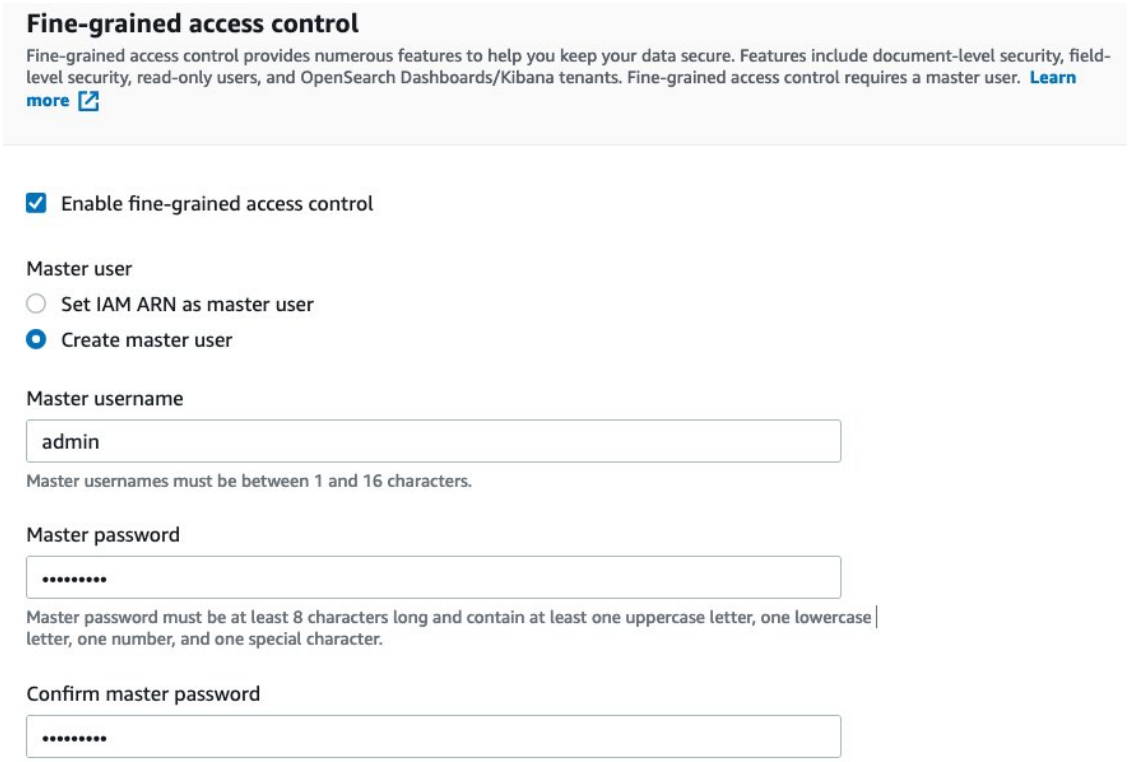
For Data nodes select 1-AZ (in production you may want to go for 3-AZ), select instance type as **t3.medium.search,** number of nodes is 1.



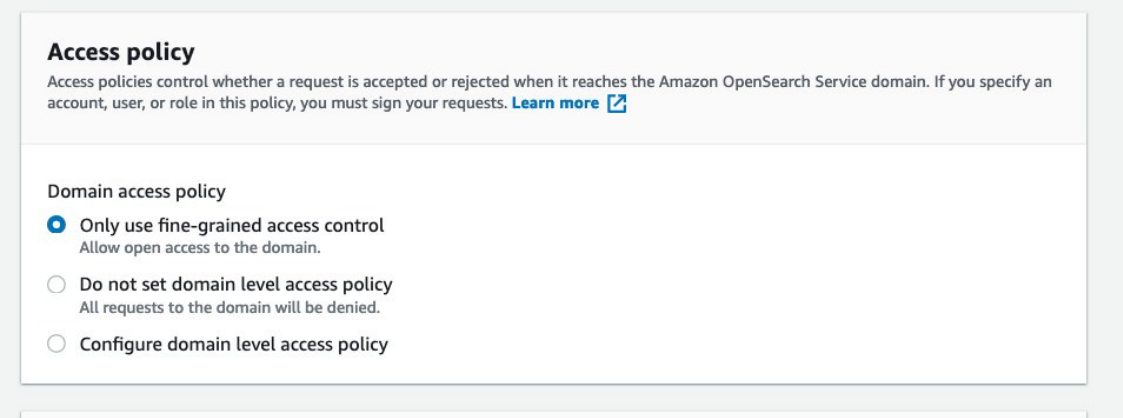
In Network select Public access, in production you may want to select VPC access recommended option.



In **Fine-grained access control** select Create master user and enter user name and password as desired.

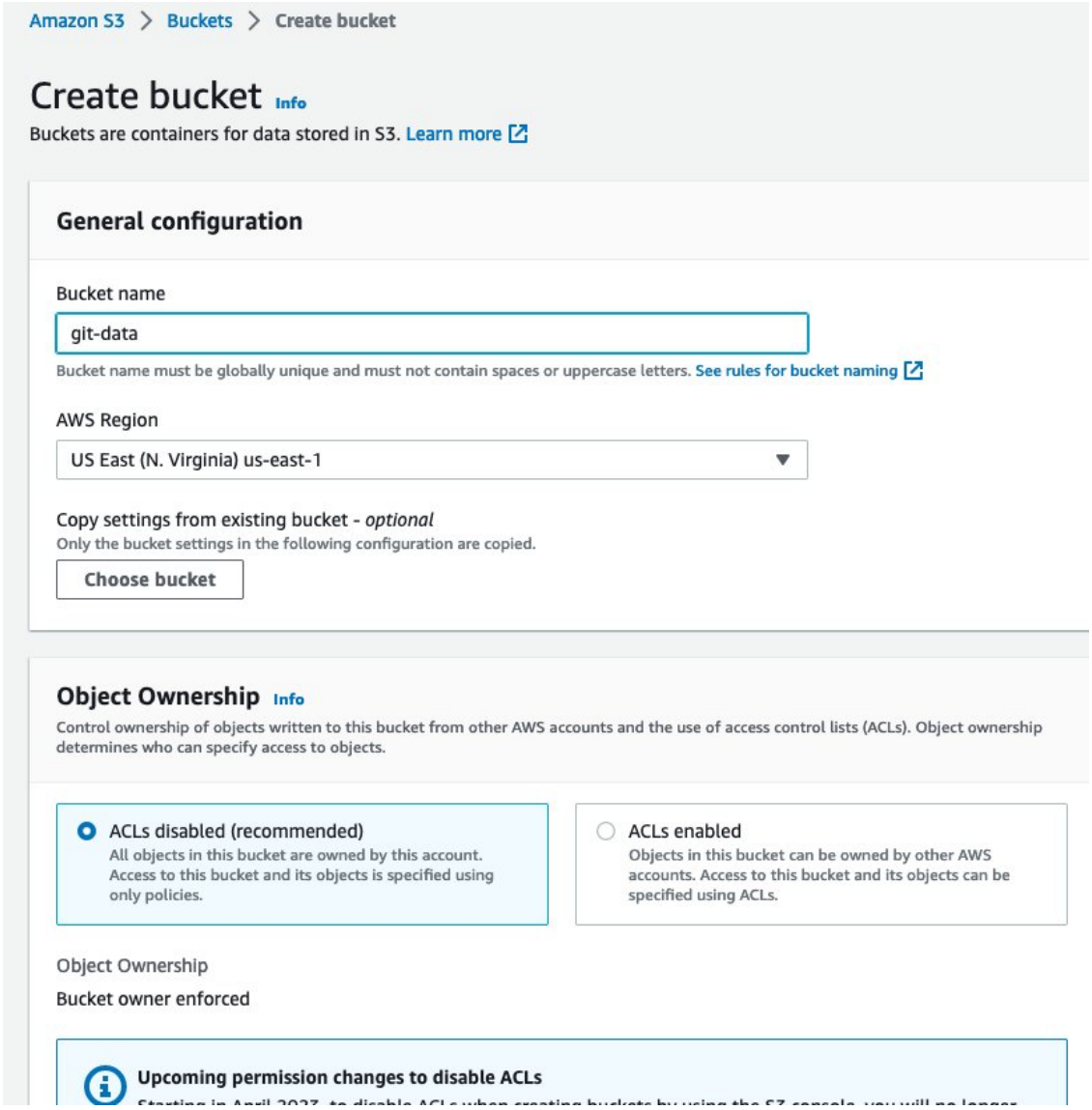


In Access policy use Only use fine-grained access control



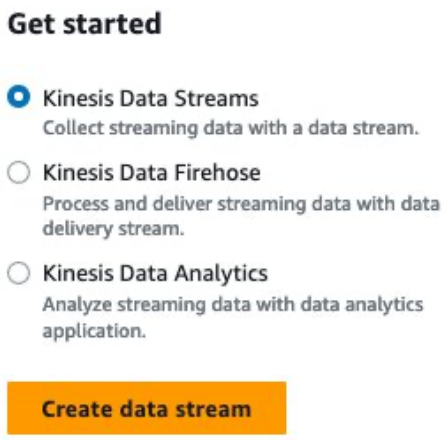
Rest settings keep as default and select create button.

2. Create S3 bucket where you want to save the GitHub logs. I named the bucket as git-data

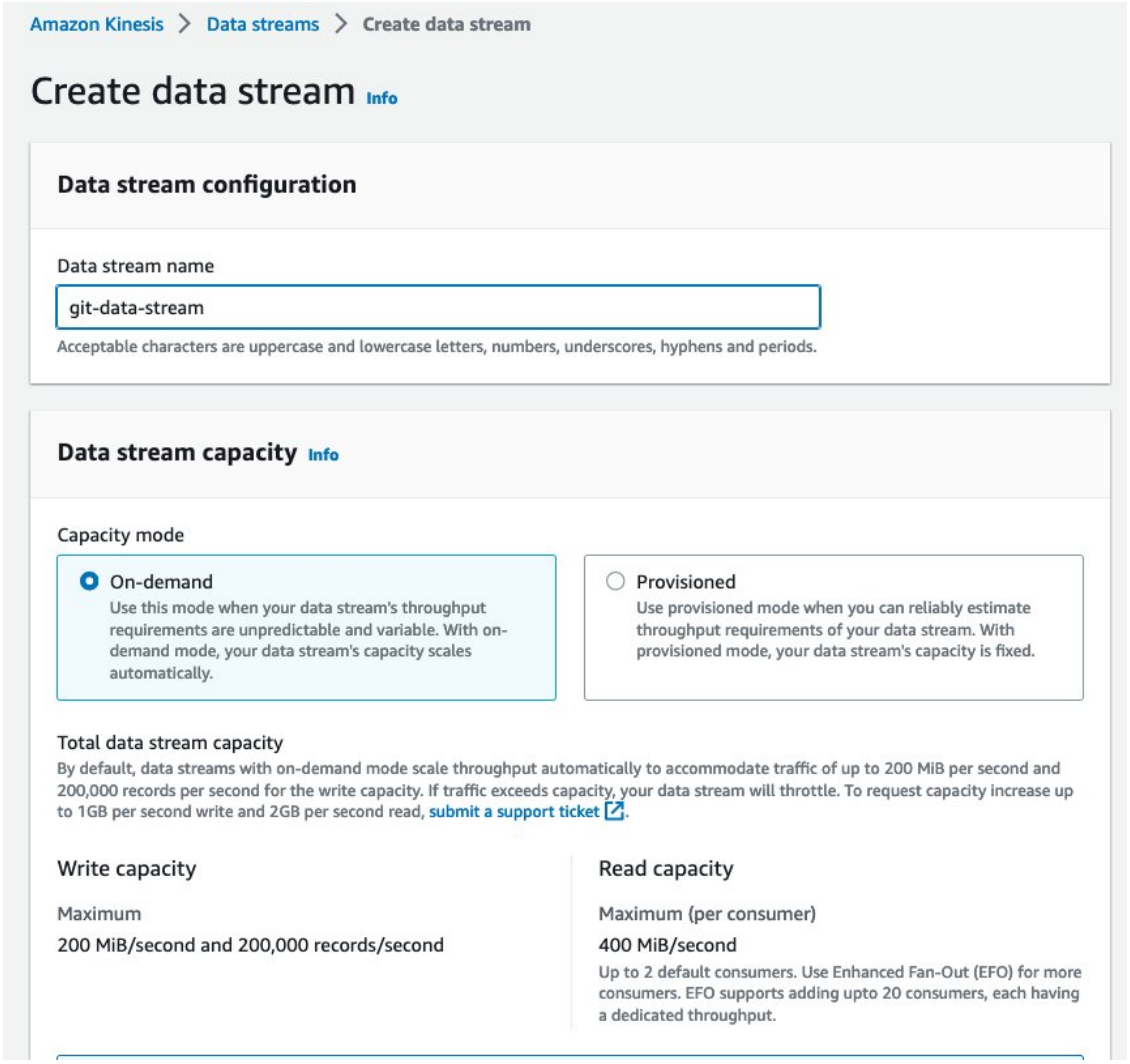


3. Create **Amazon Kinesis Data Stream** for streaming git events

Select Kinesis Data Streams and select Create data stream button



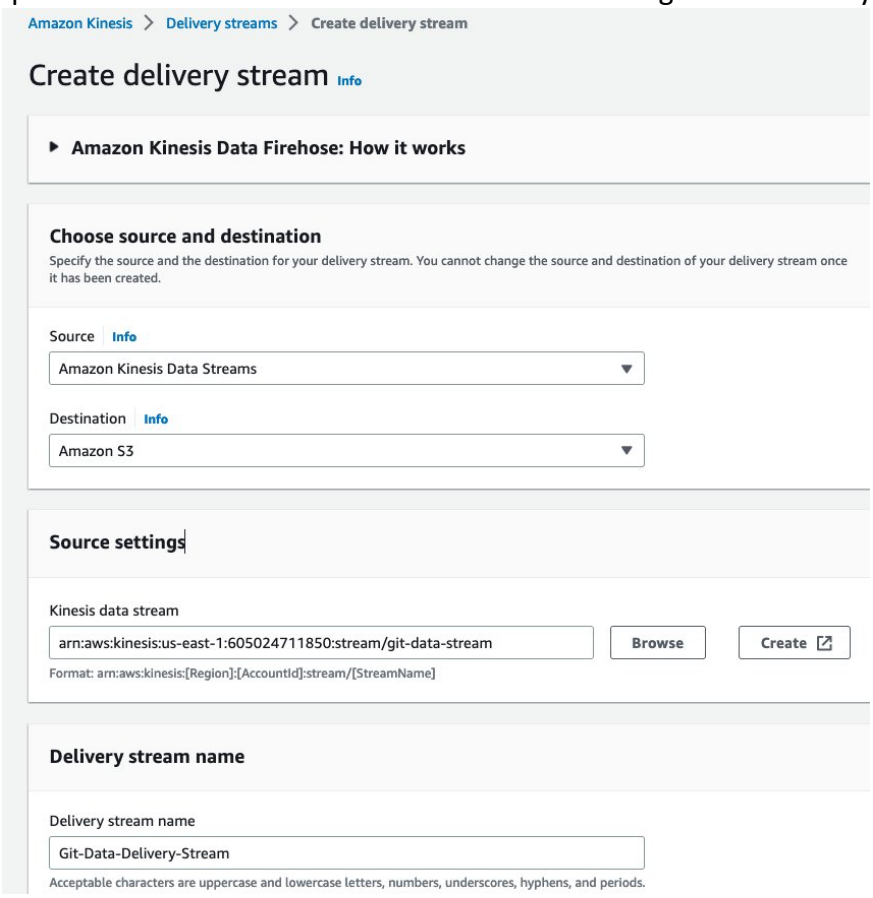
Select on-demand and select create button.



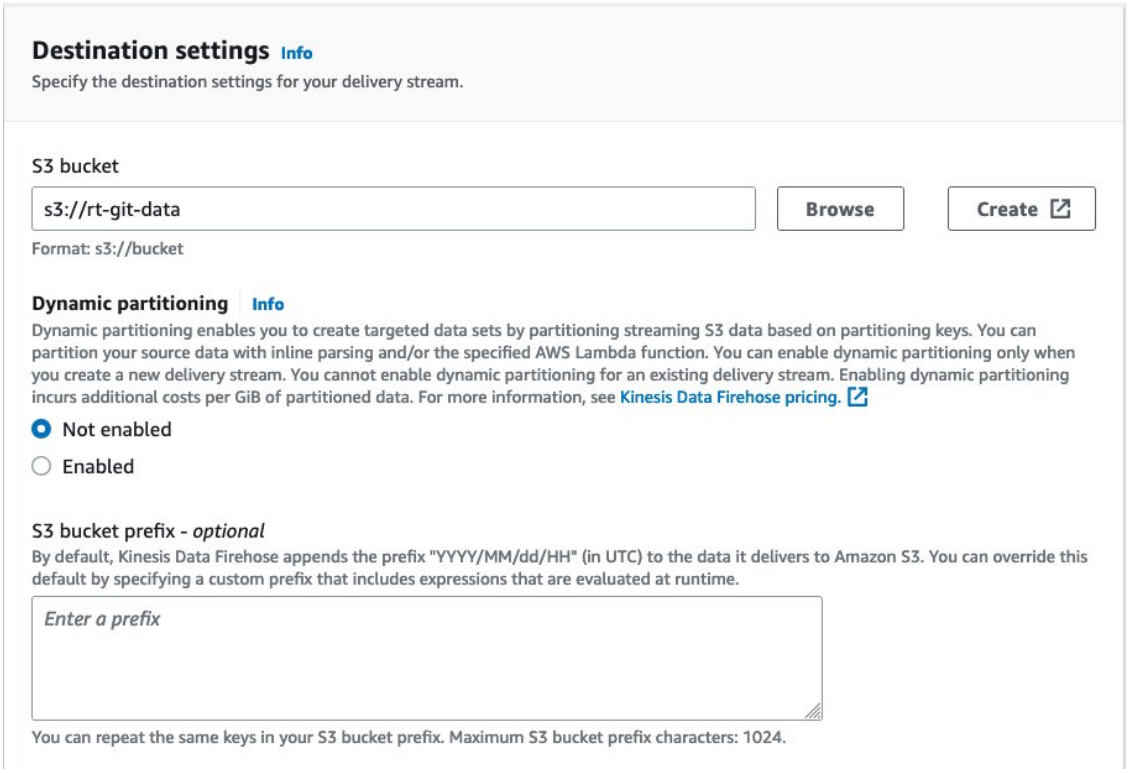
3. Create **Delivery Streams**

Select Delivery strems in Amazon Kinesis. Choose Source as Amazon Kinesis Data Streams and Destination as Amazon S3. In **Source settings** select the Kinesis data stream (git-data-stream) that we created in previous step. In **Delivery stream name** give the name as **git-data-delivery-stream.**

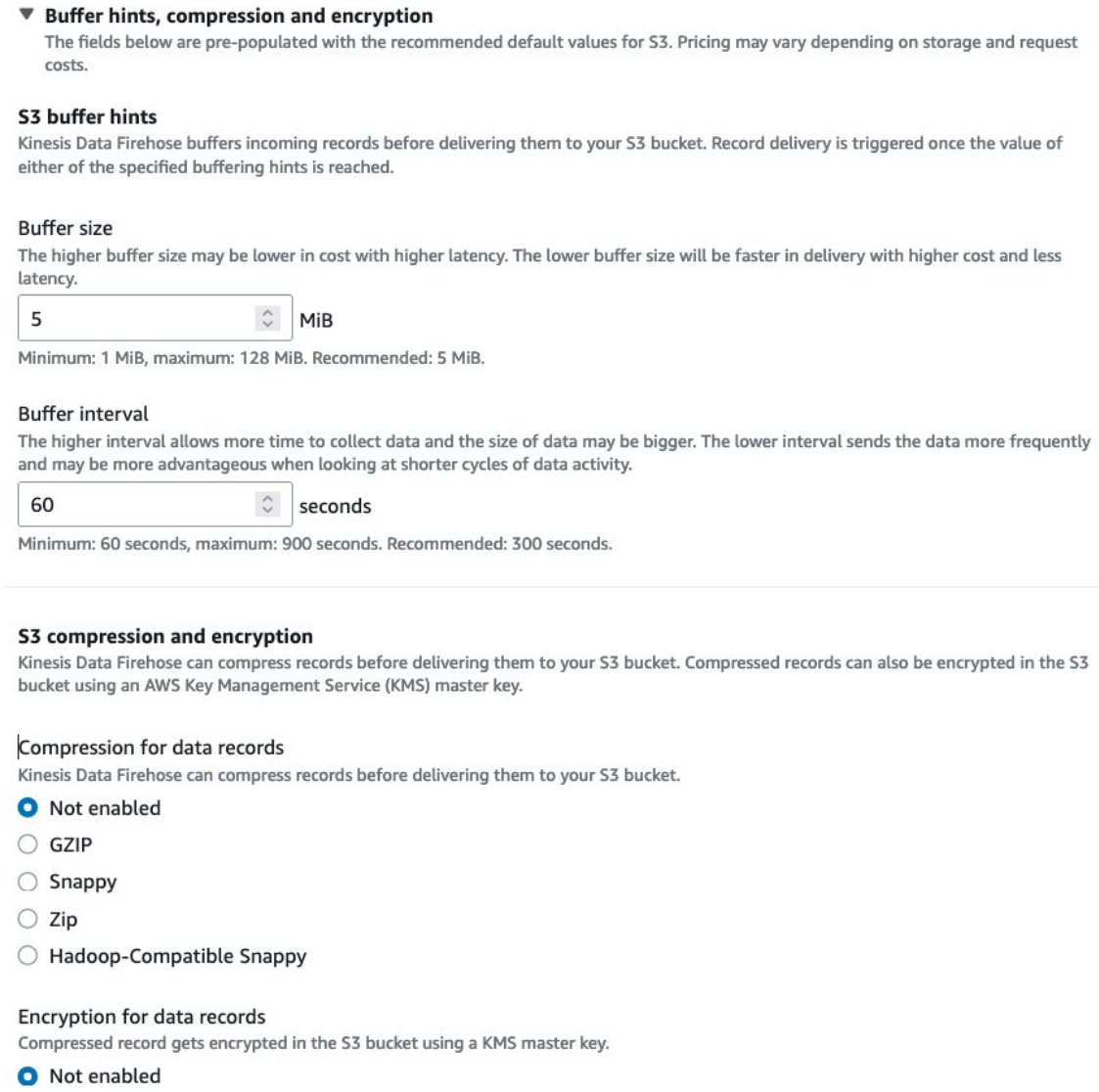
Create Kinesis Delivery Streaming, using Kinesis Firehose to stream the git logs to S3 bucket and keep the Buffer interval to 60 seconds. Select create IAM role AWS will create role and assign permissions required to add new files to S3 bucket. Name this as git-data-delivery-stream.



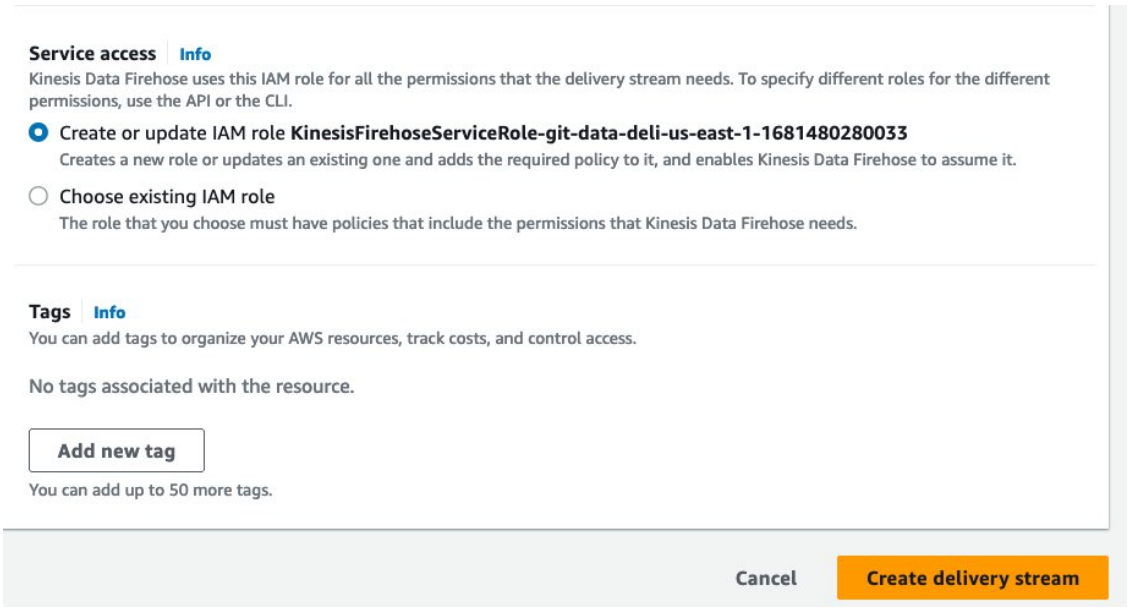
In **Destination settings** select the S3 bucket (git-data) that we created in previous step.



Expand **Buffer hints, compression and encryption** section and change the **Buffer interval** to 60 seconds. In production based on your data size you can change this time to maximum 900 seconds.

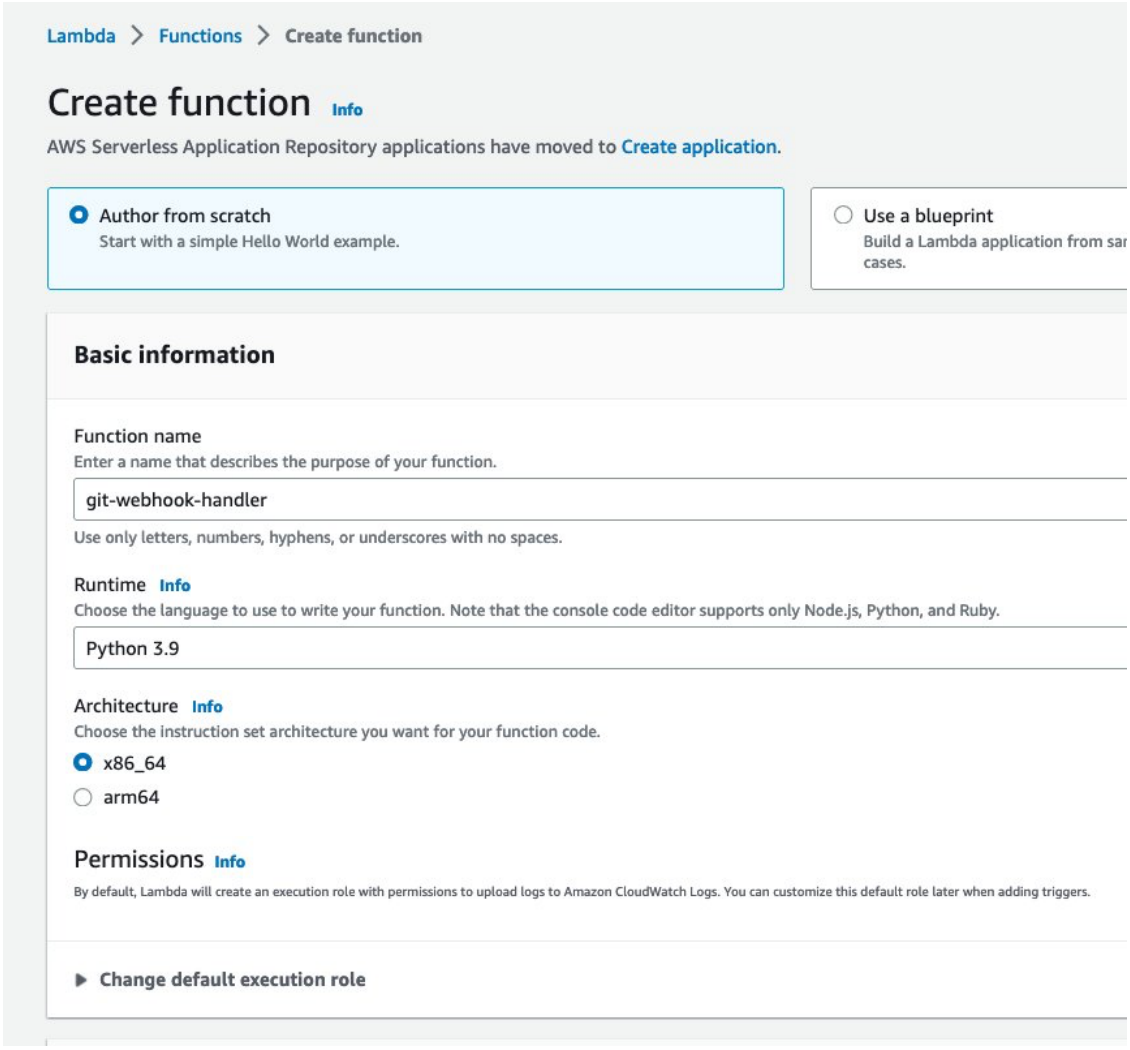


In **Advance settings** service access select Create or update IAM role. This role will have desired permissions to create git events files in S3 bucket. Finally select Create delivery stream button.

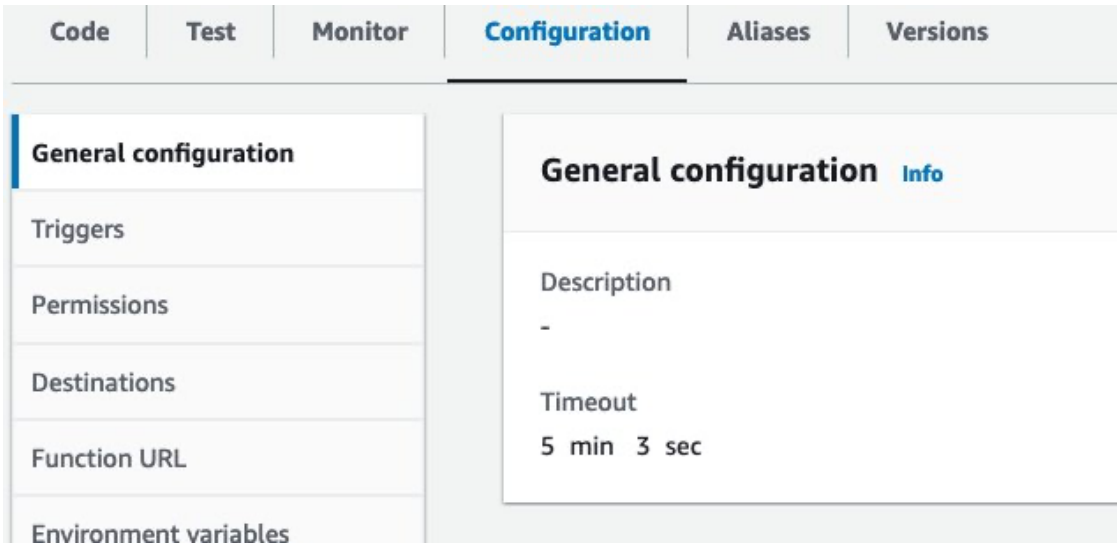


4. Now we will create **AWS Lambda** to receive the GitHub data and post to Amazon Kinesis Data Stream.

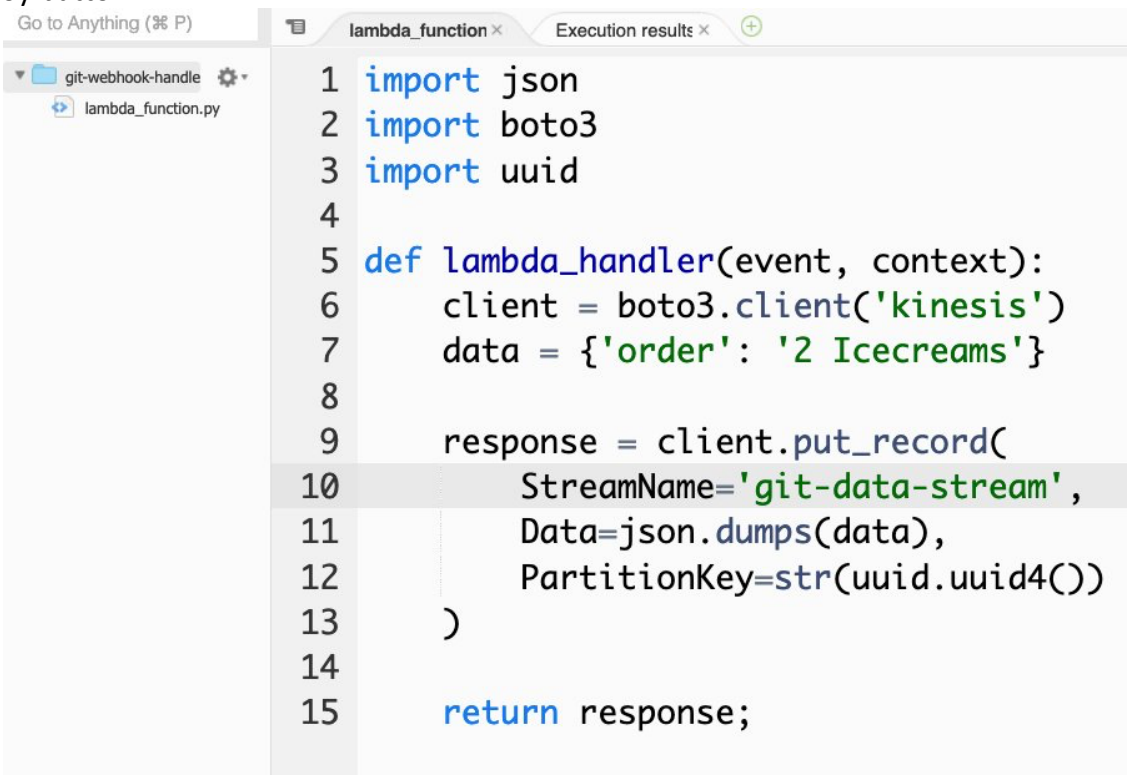
Create a function from AWS Lambda page. Give name as **git-webhook-handler** select **Runtime** as Python 3.9 and select Create function button.



Select Configuration tab, select Edit button, change the **Timeout** to 5 mins or anything more than default 3 seconds and select save button.



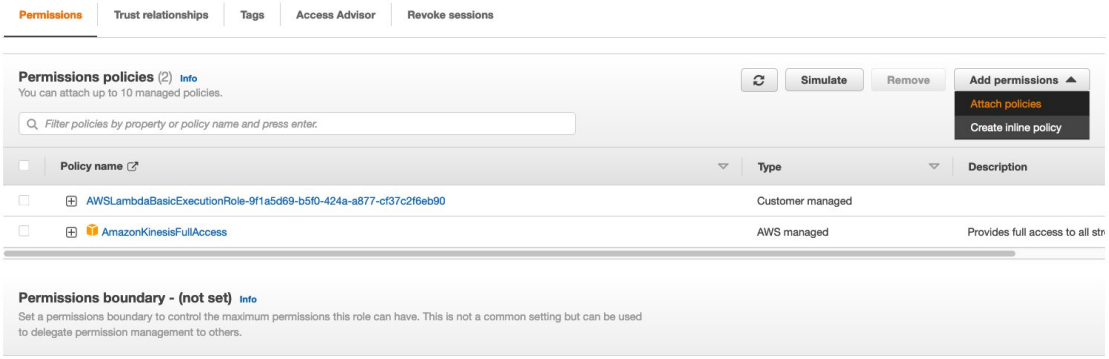
Get the code from https://gist.github.com/rupeshtiwari/1b25f0f57ccf85c25b9a33e689cb58f3 and paste it in the AWS lambda function make sure to change the **StreamName** if you have given different name. This code will stream the git event to Kinesis Data Stream. Next select Deploy button.



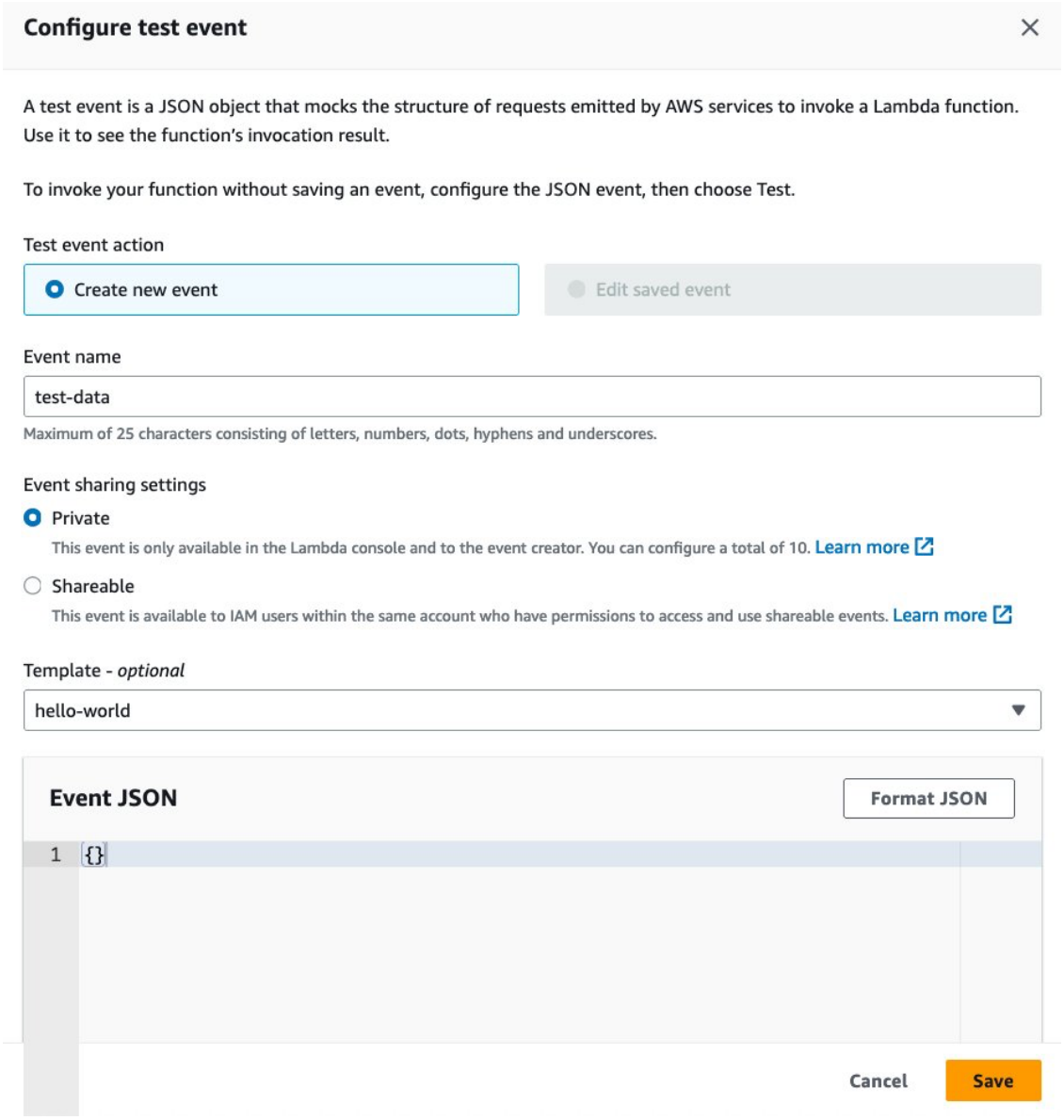
5. Add permission to AWS Lambda function's execution role to put records on Amazon Kinesis Data Streaming.

Select **Configuration** tab, select **Permissions** and select the role link. Select **Add permissions** options and select **Attach policies** option**.** Search for AmazonKinesisFullAccess policy in production you may want to just select READ and Write access to Amazon Kinesis for this demo I will give full access. Then select **Add permissions** button.

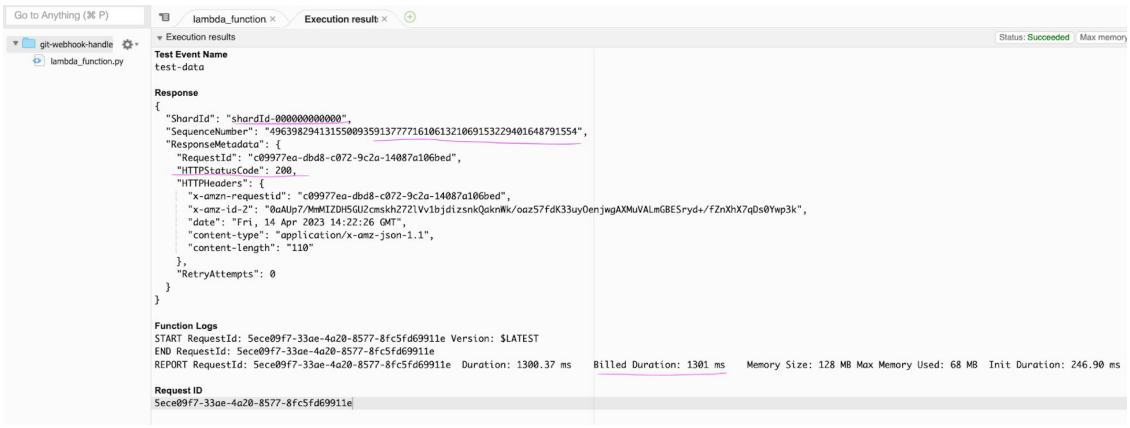
Go to AWS Lambda function configuration, permissions section and select the role. Select Attach policy and add AmazonKinesisFullAccess policy to the role and save it.



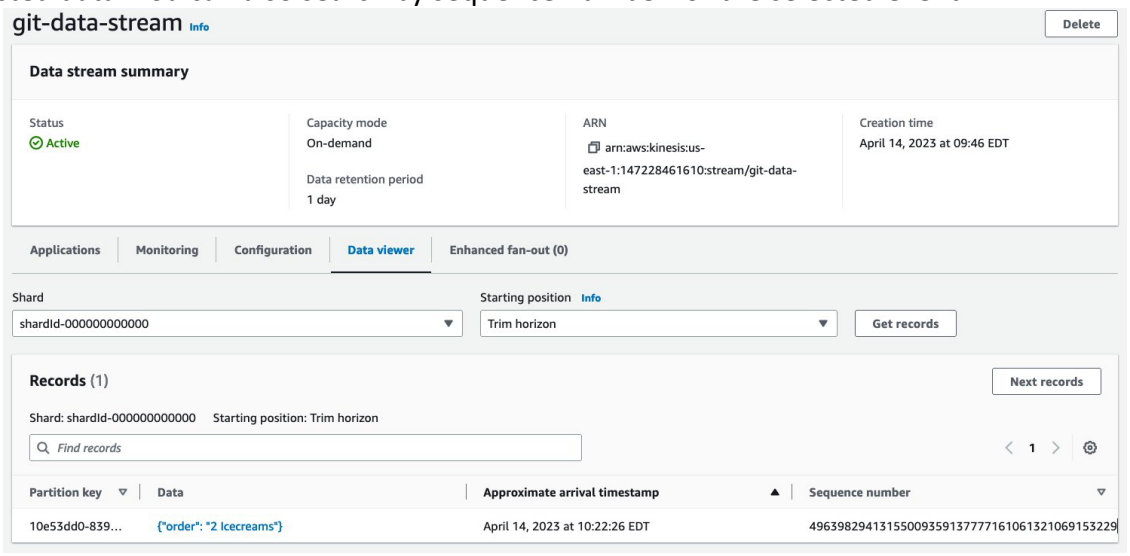
6. Now we will test the streaming. Navigate to AWS Lambda function, select test button and give the name as **test-data** and select **save button.**

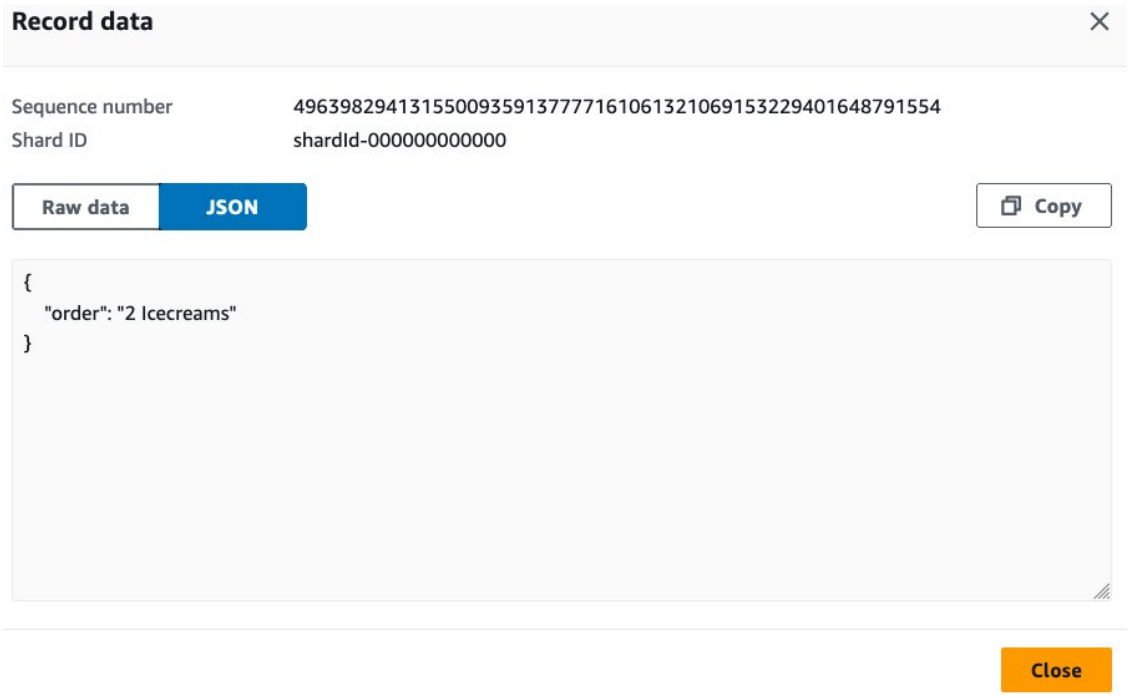


Next select the **test** button and In the **Execution result** tab notice the **ShardId**, **SequenceNumber**, **HTTPStatus** code as 200. Which means we successfully stream the data to Kinesis data stream.

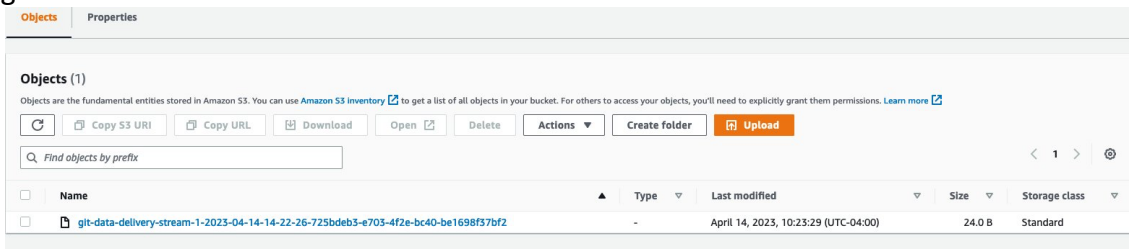


To verify, open **Kinesis Data stream**, select **Data viewer** tab, select the **Shared** Id displayed in function response, select **Trim horizon** and select **Get records.** You will be able to view the ingested data. You can also search by sequence number for the selected event.

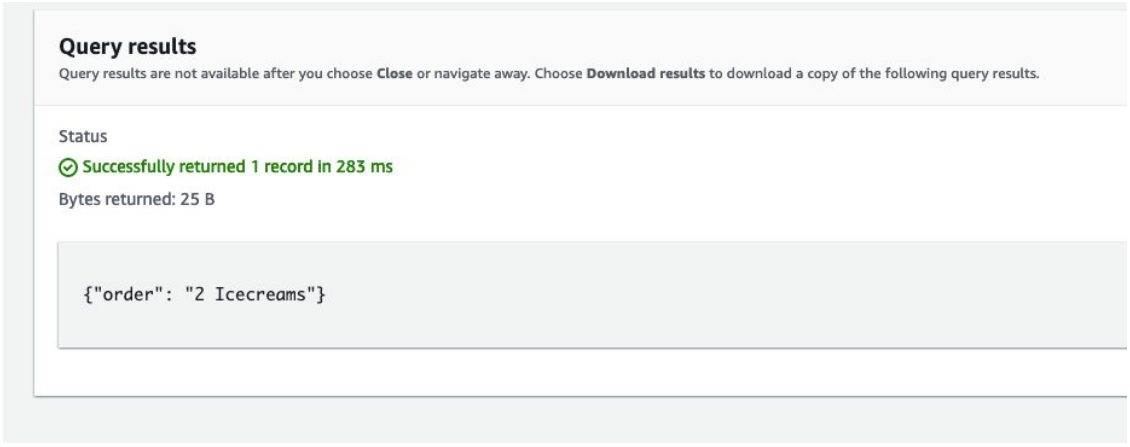




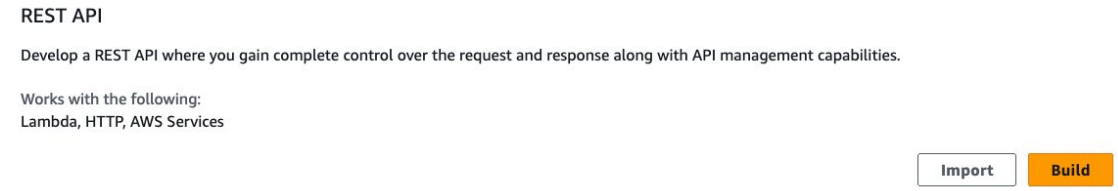
Navigate to S3 bucket and confirm that the record is also created in the bucket.



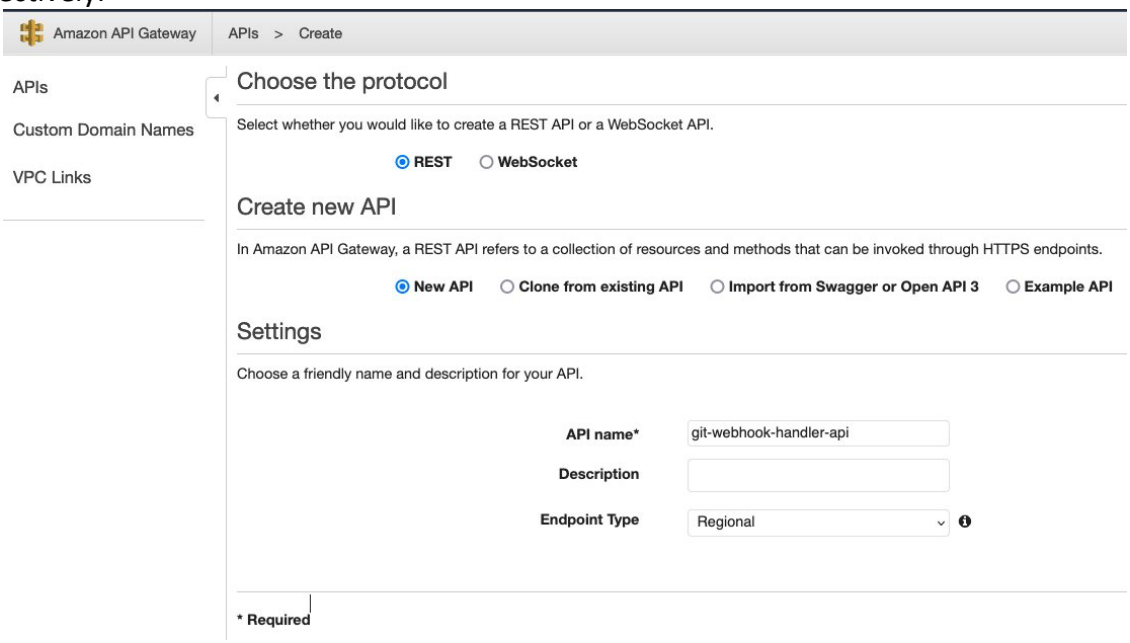
Select Query with S3 select and **Run SQL query** to visualize the data.



7. Next we will create a **Amazon API Gateway.** Select **REST API** and build button.

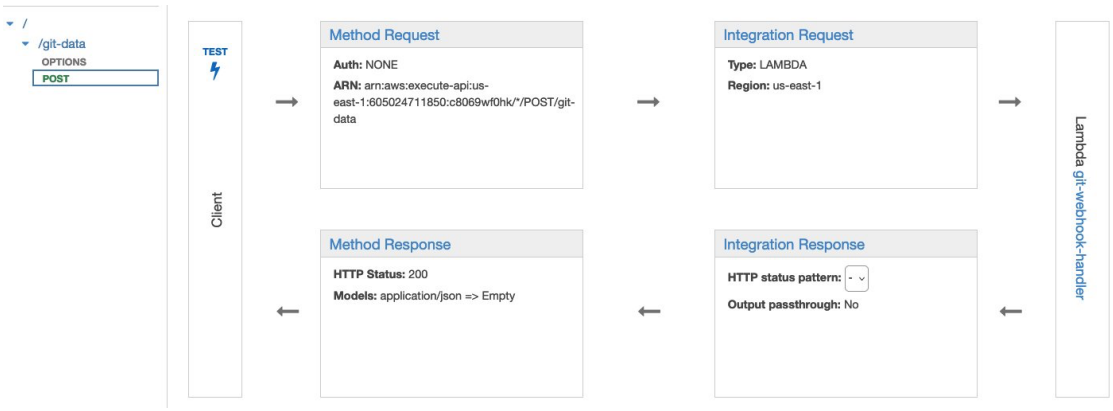


Select New API, give API name as **git-webhook-handler-api** and select **Create API** button. For instructions on how to create and deploy an API by using the API Gateway console, see [Creating a REST API in Amazon API Gateway] and [Deploying a REST API in Amazon API Gateway], respectively.

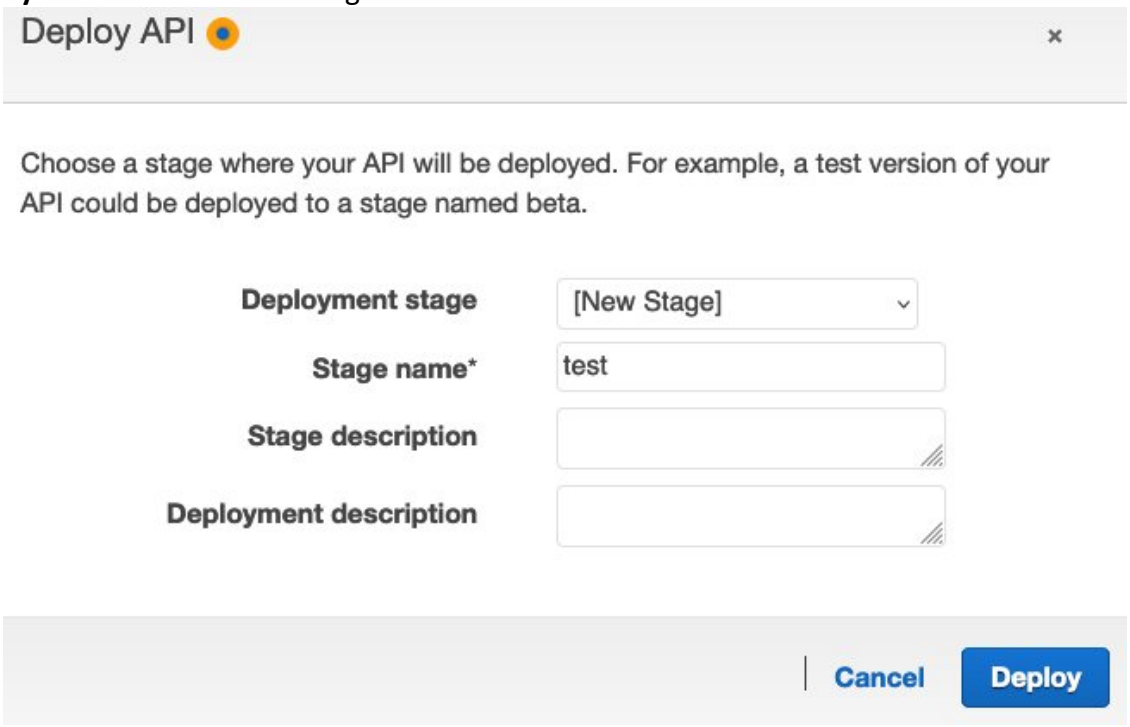


Select **Actions** as **Create a resource**, enter Resource Name as **git-data** and select **Create Resource**

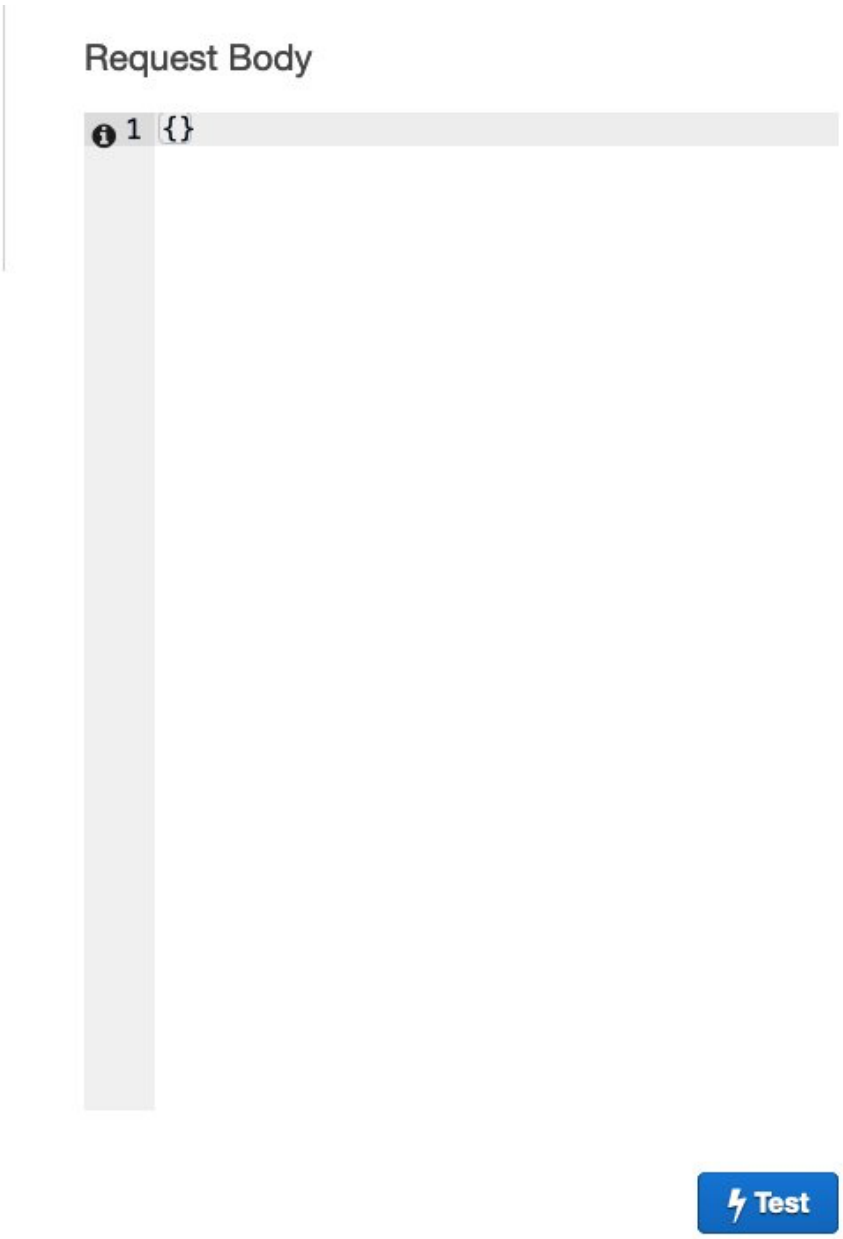
Select **Actions** as **Create Method,** select **POST** and select the tick button. Enter **Lambda Function** name as **git-webhook-handler** and select **Save** button.



Deploy the REST API to **test** stage.



Next select the Test button give empty JSON for body and select **Test**



Validate the data is ingested to Kinesis Successfully.

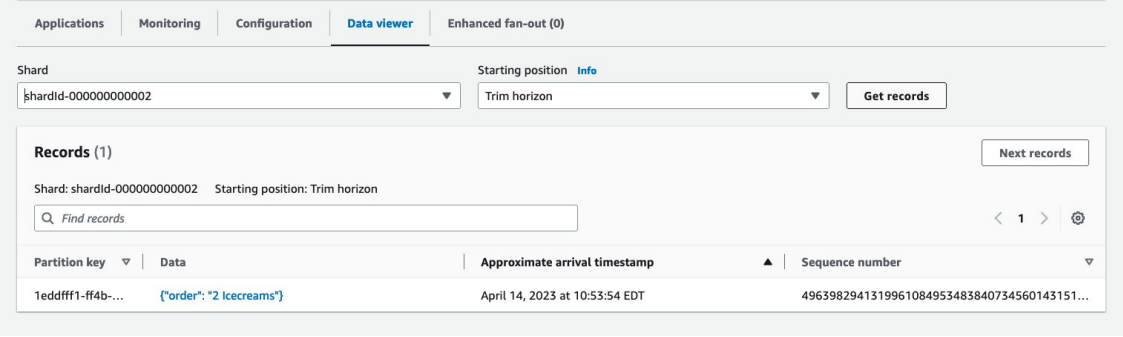
Request: /git-data

Status: 200

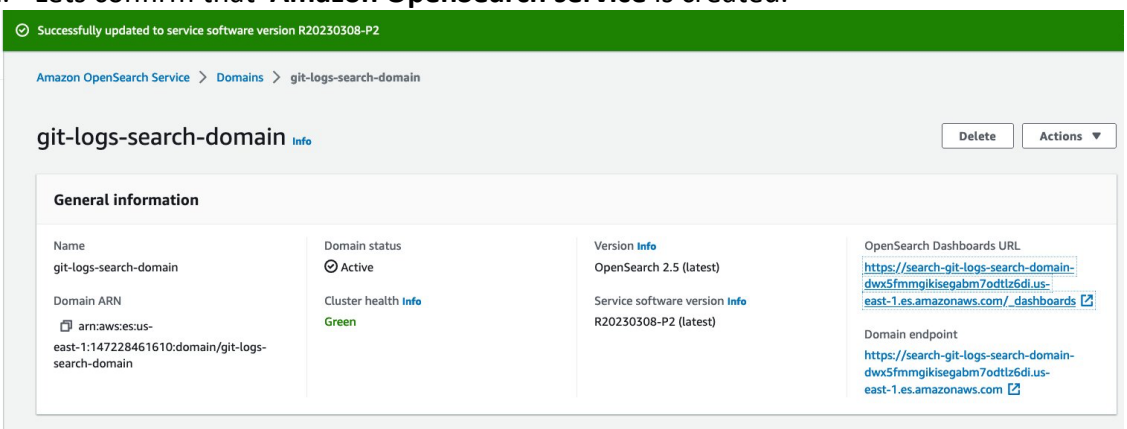
Latency: 1793 ms

Response Body

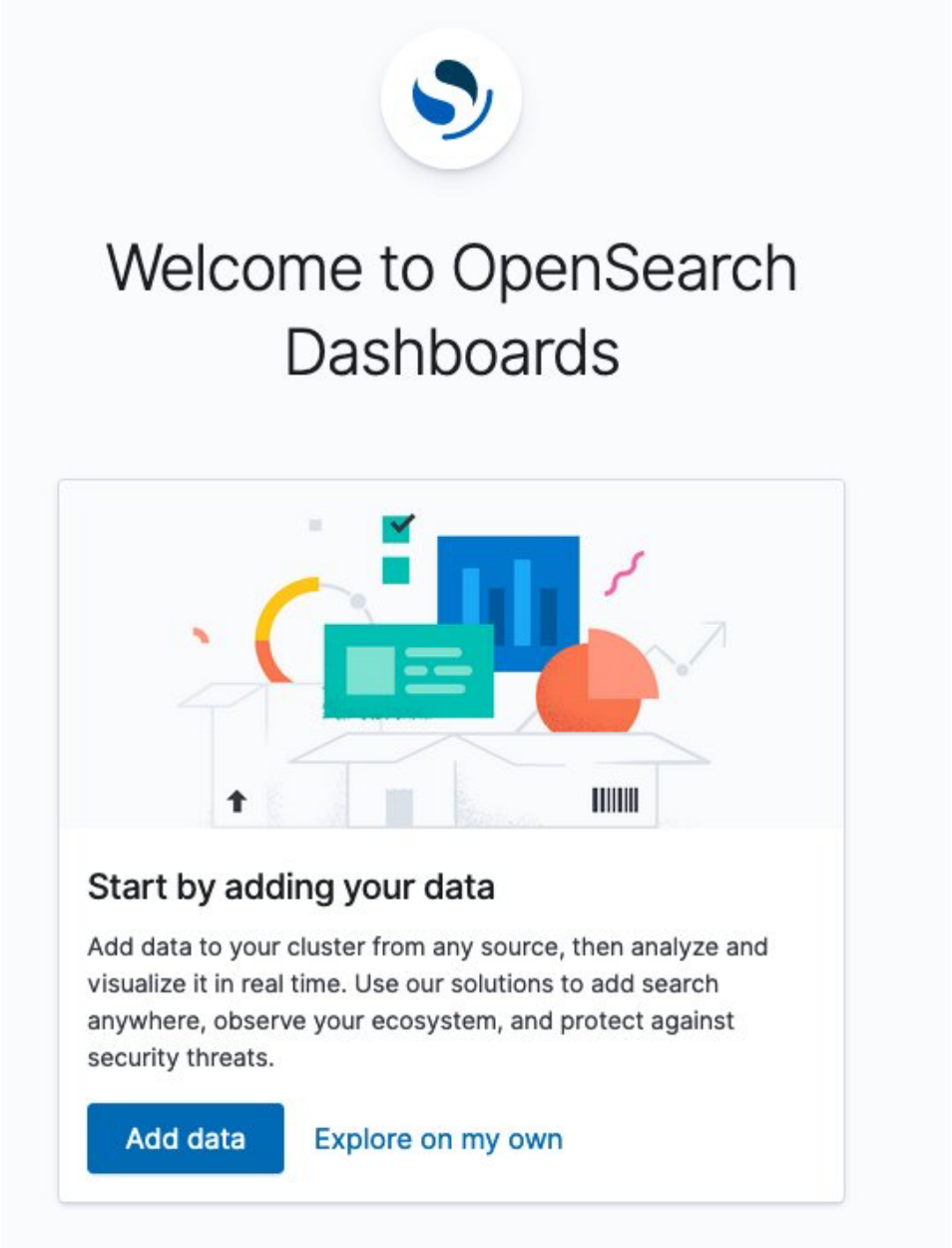

```
{"ShardId": "shardId-000000000002", "SequenceNumber": "49639829
413199610849534838407345601431518270427488059426", "ResponseMet
adata": {"RequestId": "c27a58e9-1685-4649-9ec9-33ab91ab2d17", "
HTTPStatusCode": 200, "HTTPHeaders": {"x-amzn-requestid": "c27a
58e9-1685-4649-9ec9-33ab91ab2d17", "x-amz-id-2": "fj5g9GZvAgwPY
WNh5ySC00E0/v8xilmMvGEC/V/juxTVhARQuS0PdmU6YJDVDTj7Sa/P+mXwl7wj
aVQNoV/RLhToJmaPD8B9", "date": "Fri, 14 Apr 2023 14:53:54 GMT",
"content-type": "application/x-amz-json-1.1", "content-length":
"110"}, "RetryAttempts": 0}}
```

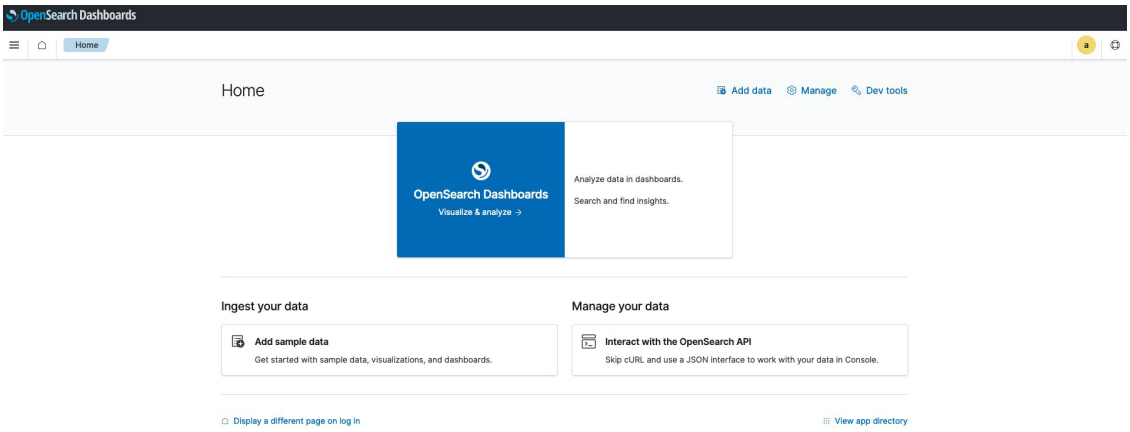
1. Lets confirm that **Amazon OpenSearch service** is created.



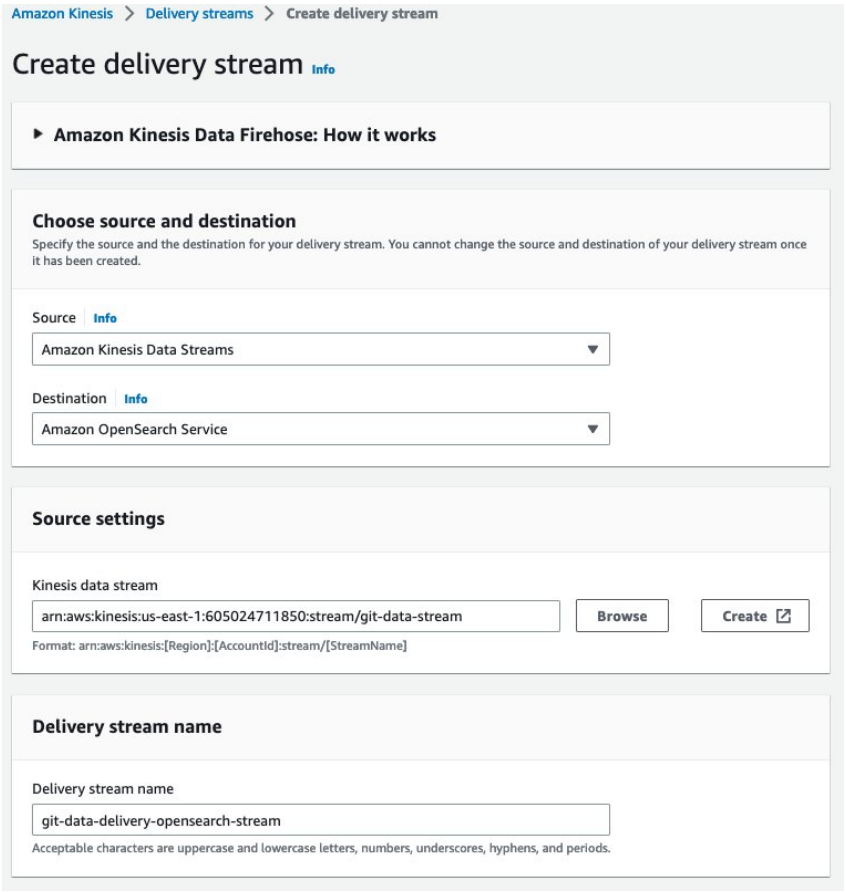
Open the Dashboard for OpenSearch by selecting the URL. Enter the user id and password that you selected in previous step and you should see the welcome page.



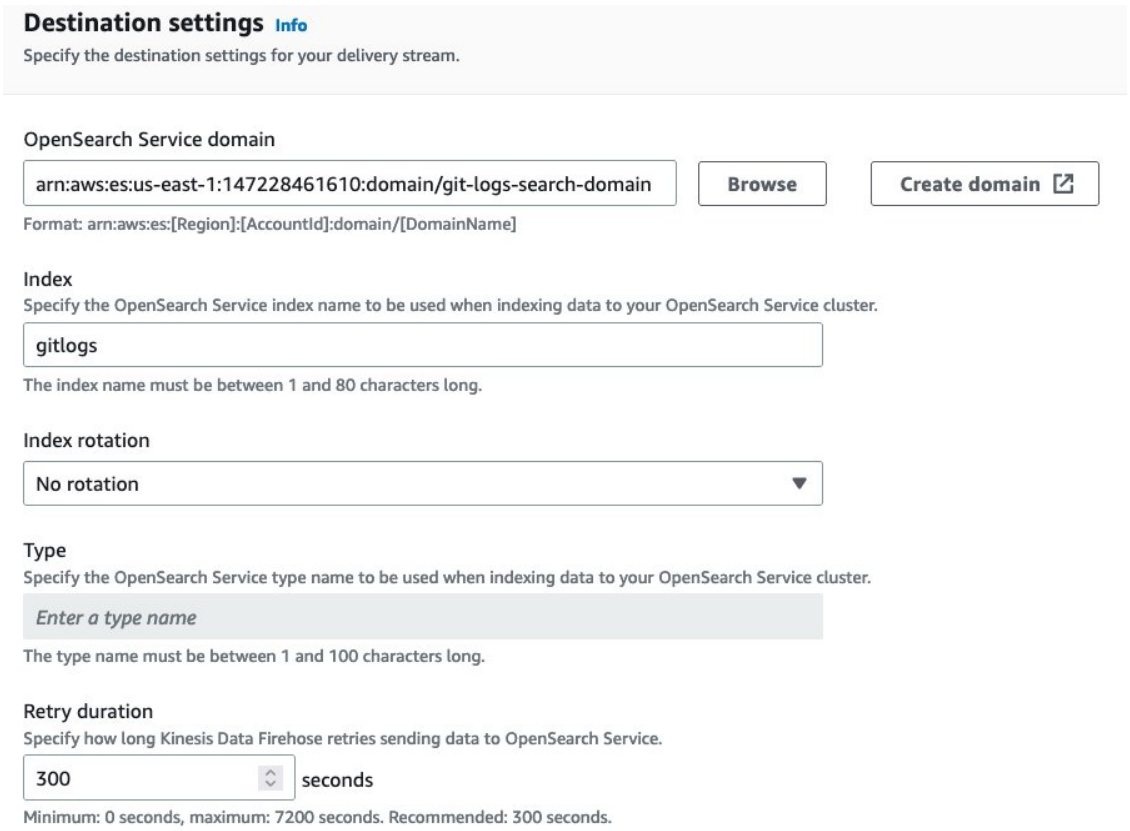
Select Explore on my own link and then Global and select Confirm button. You see the home page.



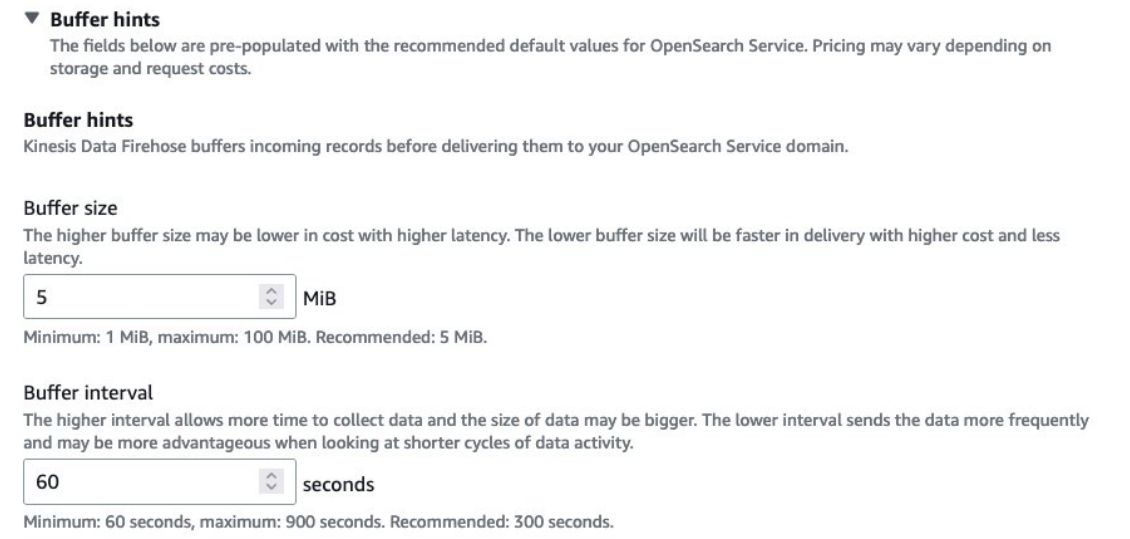
8. Create another **Kinesis Data Delivery Stream** to stream the Kineis git data to **Amazon OpenSearch** service. Select the source as **Amazon Kinesis Data Stream** and destination as **Amazon OpenSearch Service. In Source settings select the Kinesis data stream that we created previously git-data-stream. Enter the name as git-data-delivery-opensearch-stream.**



In Destination settings select the OpenSearch that we created earlier. Enter **gitlogs** as **Index**



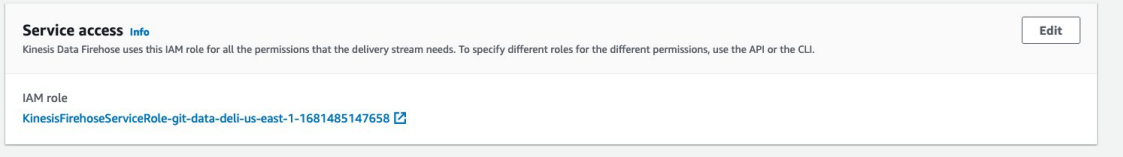
Expand **Buffer hints** and enter 60 as **Buffer Interval.**



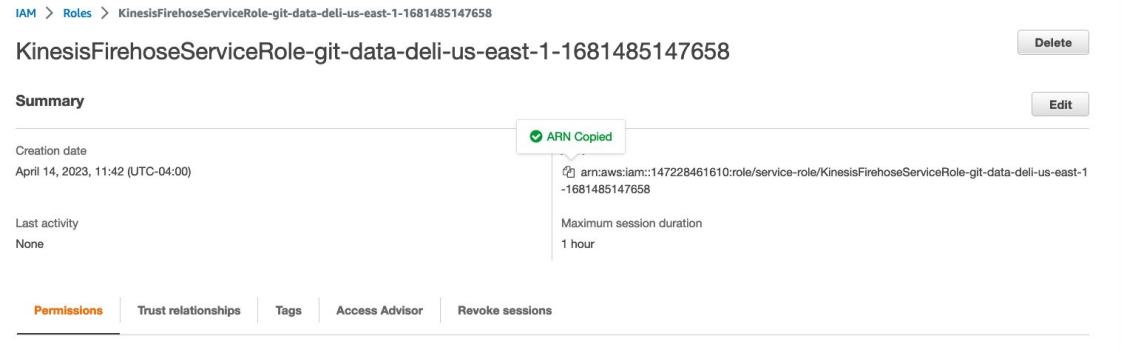
For **Backup settings** selected **Failed data only** and create a S3 bucket to save the failed data so that you will not loose them.

1. Now we will give Kinesis Firehose IAM role permission to create index on Amazon OpenSearch by mapping IAM Role with Amazon OpenSearch backend role.

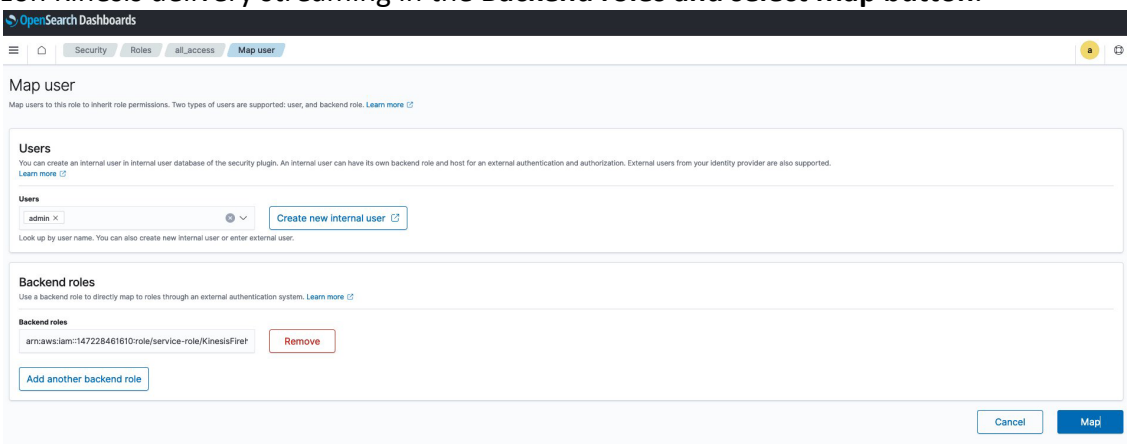
Therefore, open Amazon Kinesis delivery stream that we created in previous step (**git-data-delivery-opensearch-stream**). Select configuration tab, in Service access section select IAM role.

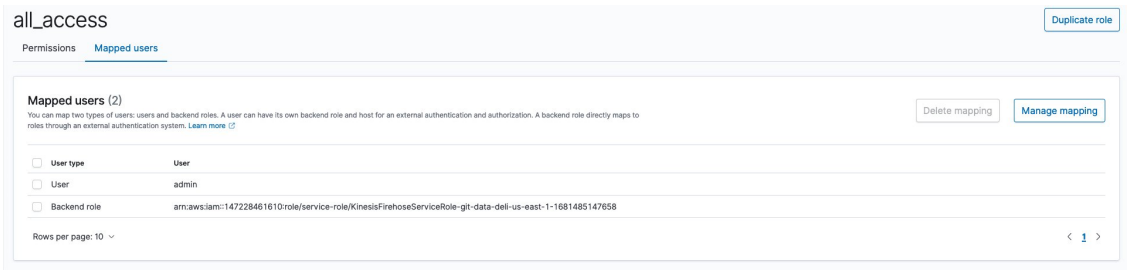


Copy the ARN value of this IAM role.



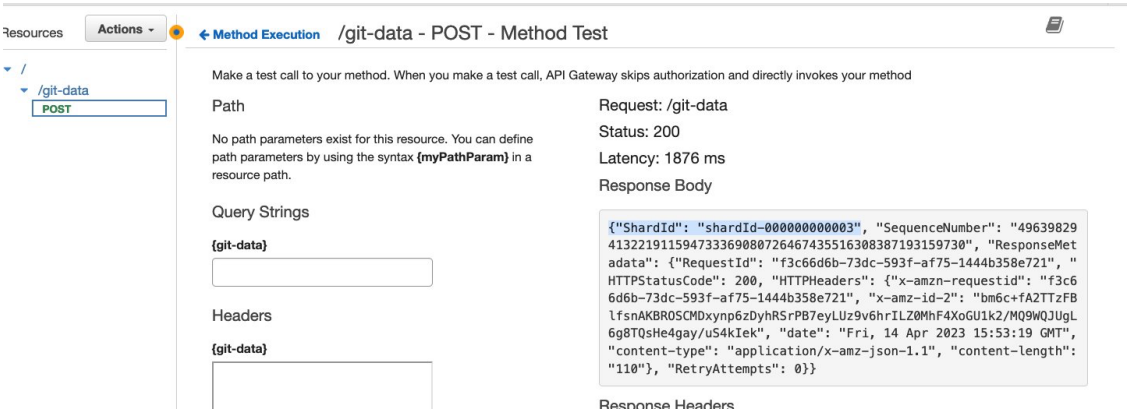
open Amazon OpenSearch Dashboard and go to security page, navigate to Roles pane, select all_access, go to Mapped users tab, select Manage Mappings and enter IAM role ARN of Amazon Kinesis delivery streaming in the **Backend roles and select Map button**.



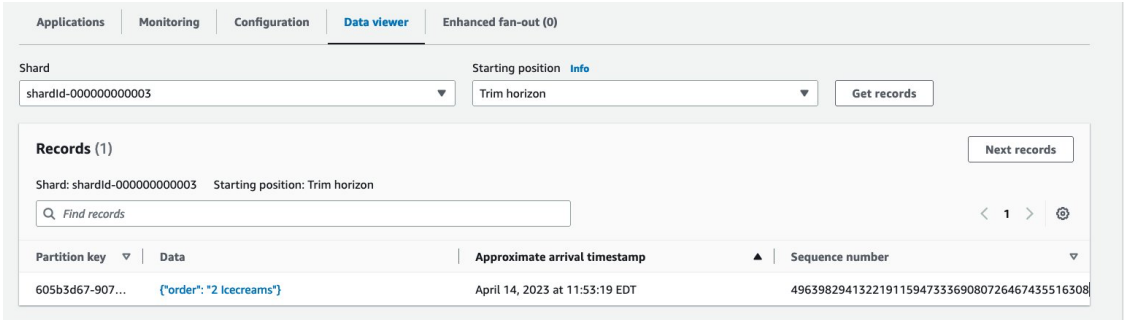


10. Test data delivery make sure data is delivered to OpenSearch service.

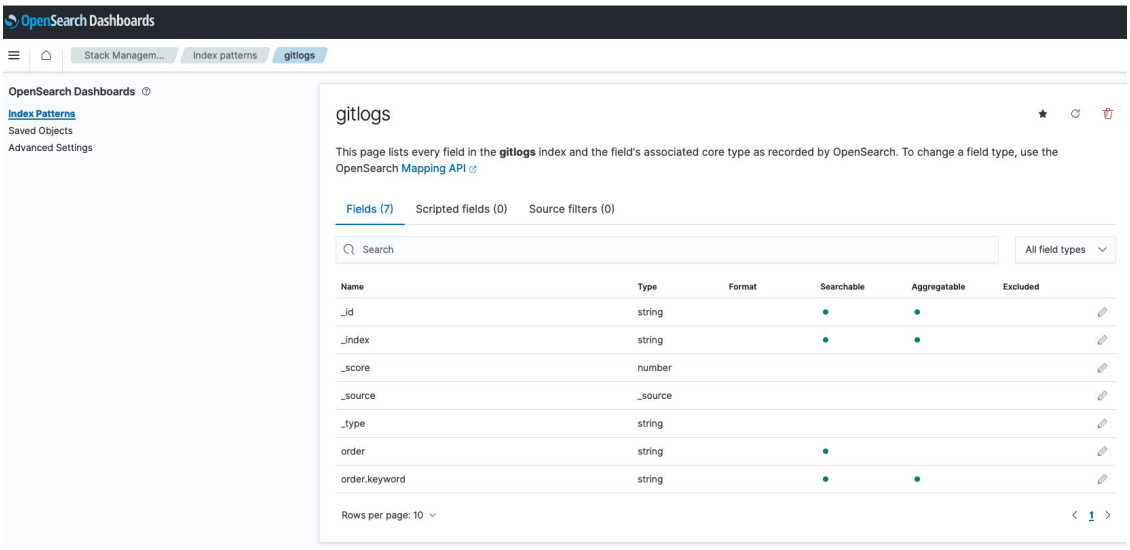
Go to API Gateway and click the test button again to send the data to Amazon Kinesis.



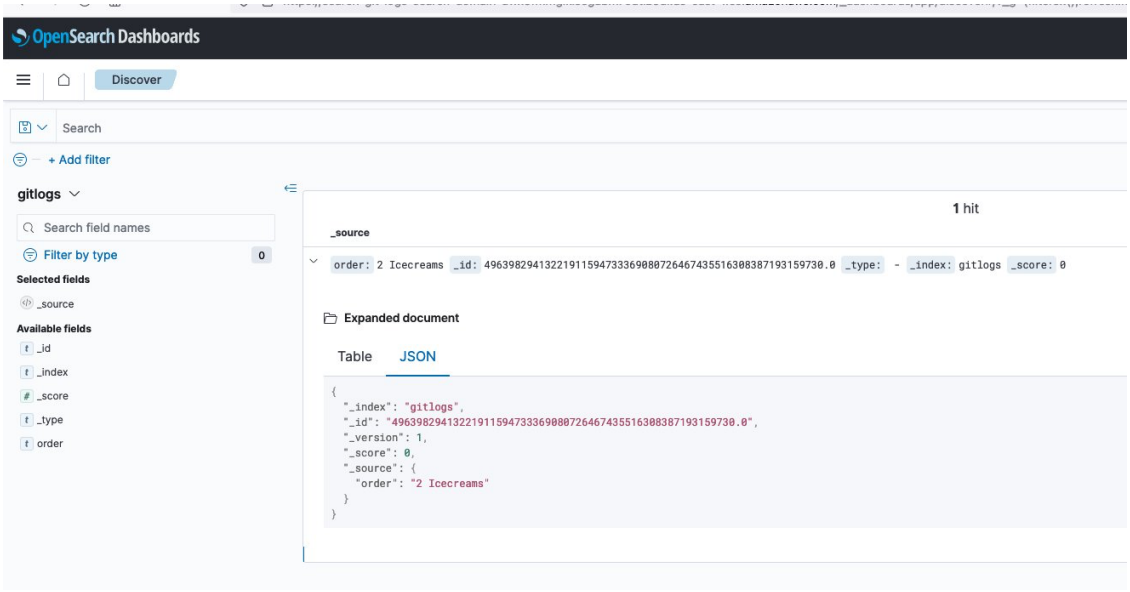
Notice data ingested to ShardId - 3 this time.



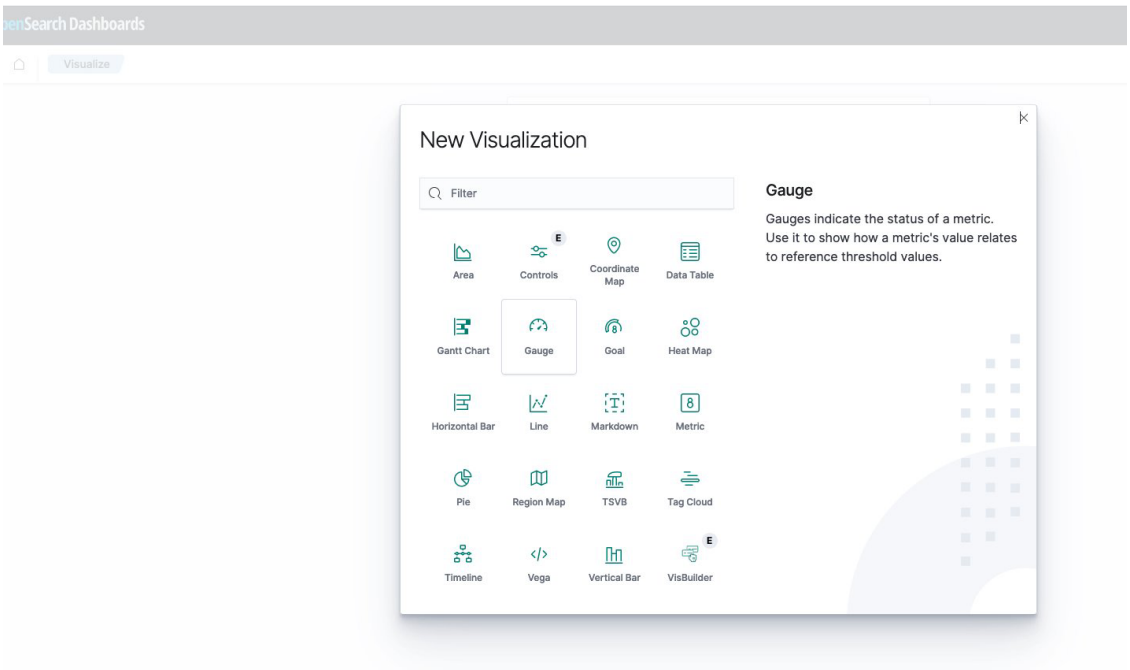
Lets confirm the data got streamed to AmazoOpenSearch after 1 minute. Go to Amazon OpenSearch, select Stack Management, Index Patterns and select Create Indesx pattern button. Enter **gitlogs** as your index name, select **Next step** and **Create index pattern.**

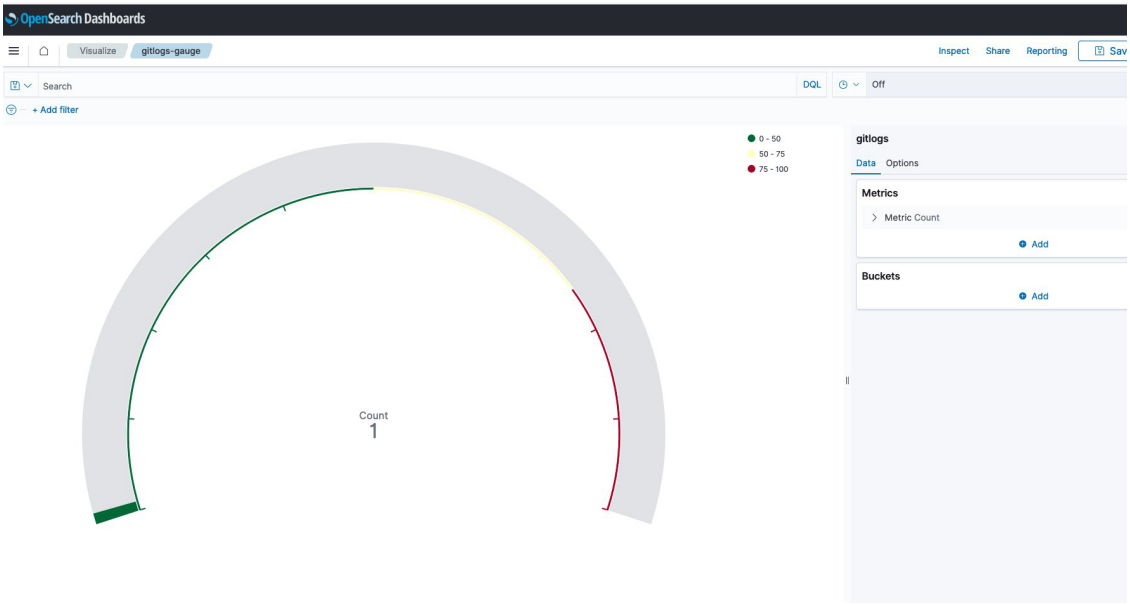


Next click on **Discover** page and you will notice the record entry is present in the dashboard.

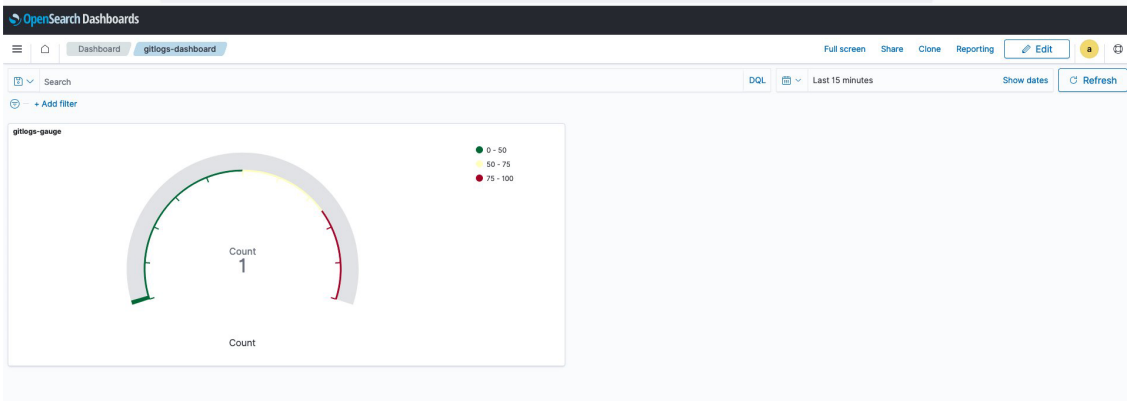


Next select Visualize, Gauge and select gitlogs and Save it, give name as **gitlogs-gauge.**

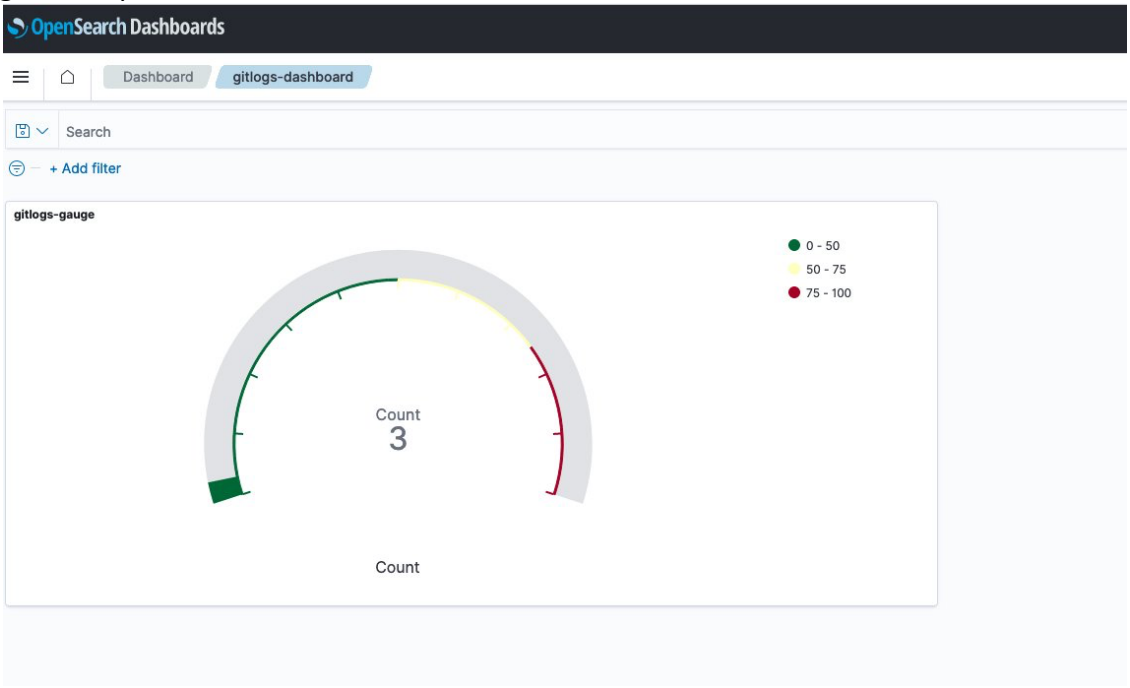




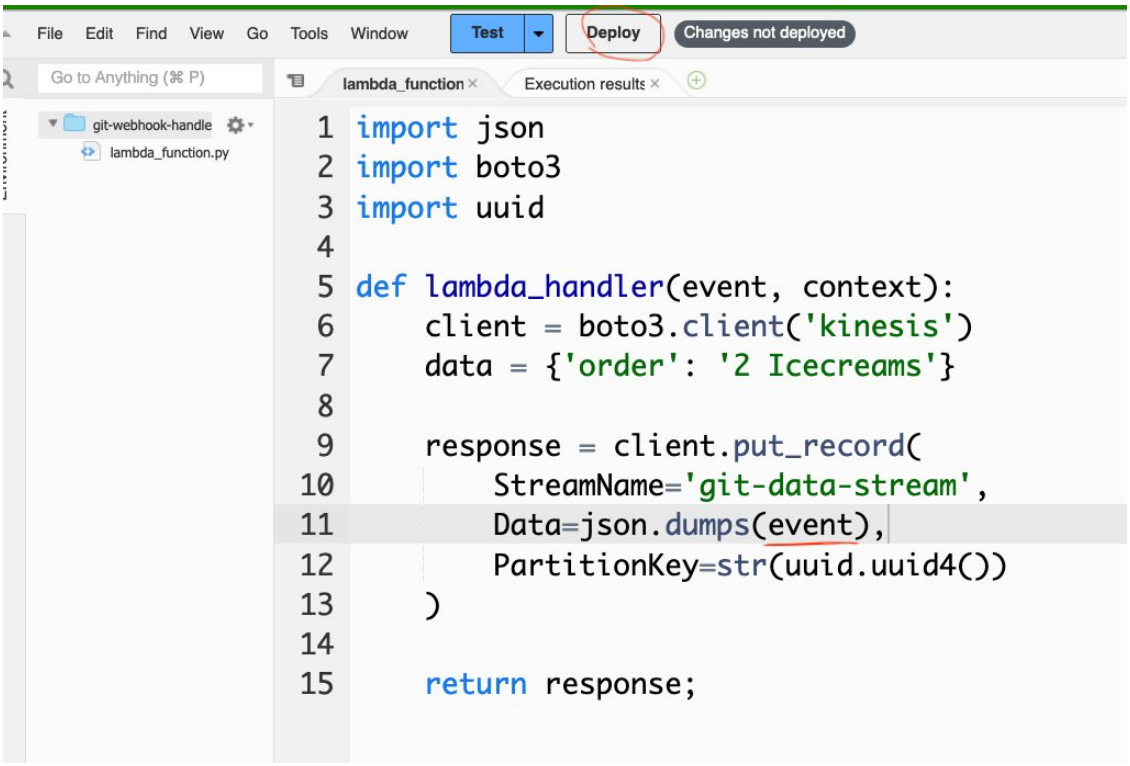
Next select Dashboard, Create new dashboard, Select Add an existing, select gitlogs-gauge, Save it, give name as gitlogs-dashboard.



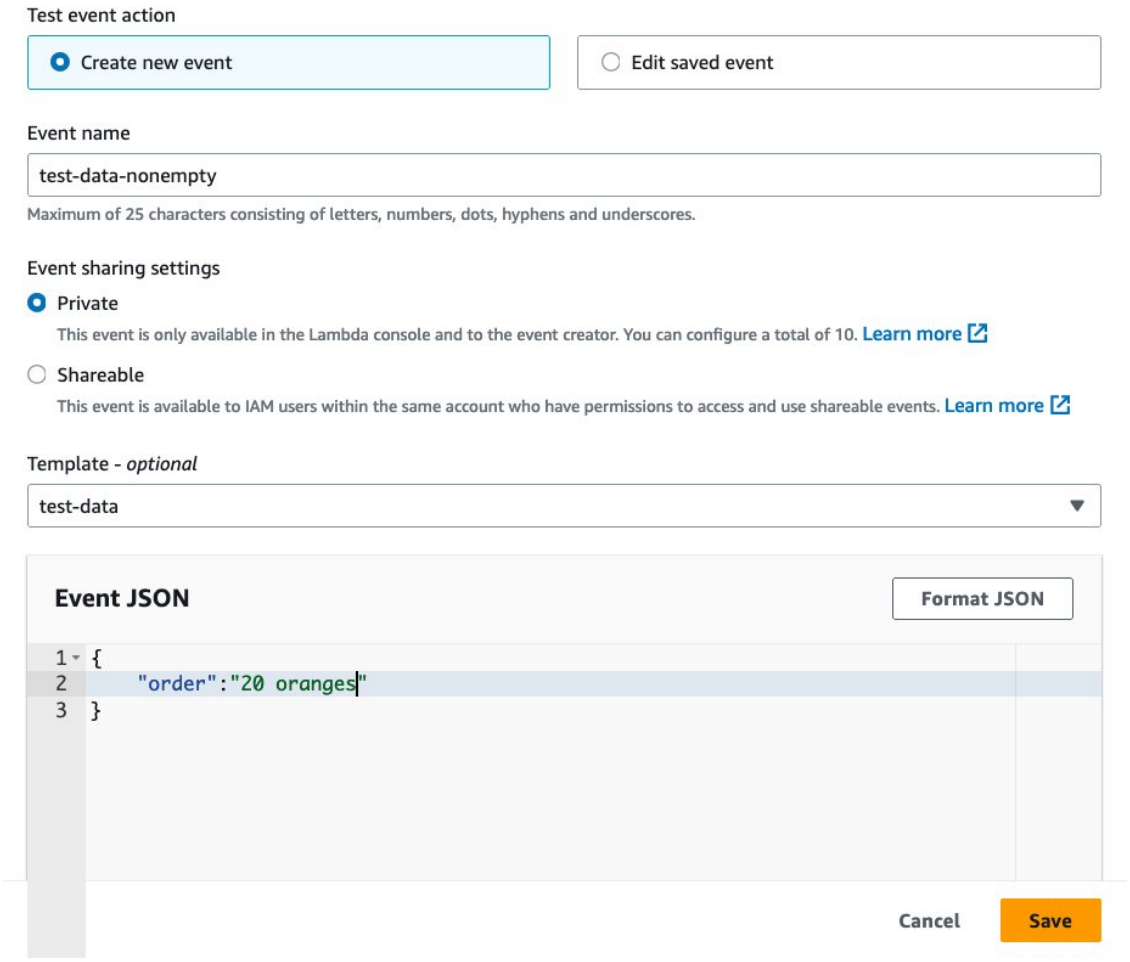
Now go ahead and test REST API 2 more times to confirm the dashboard count is changed to 3. Navigate to OpenSearch and refresh the dashboard.



11. Next update the AWS Lambda to pass the event object rather hardcoded data. This way we can stream the GitHub event coming via API Gateway. Go to AWS lambda change the code and deploy it.

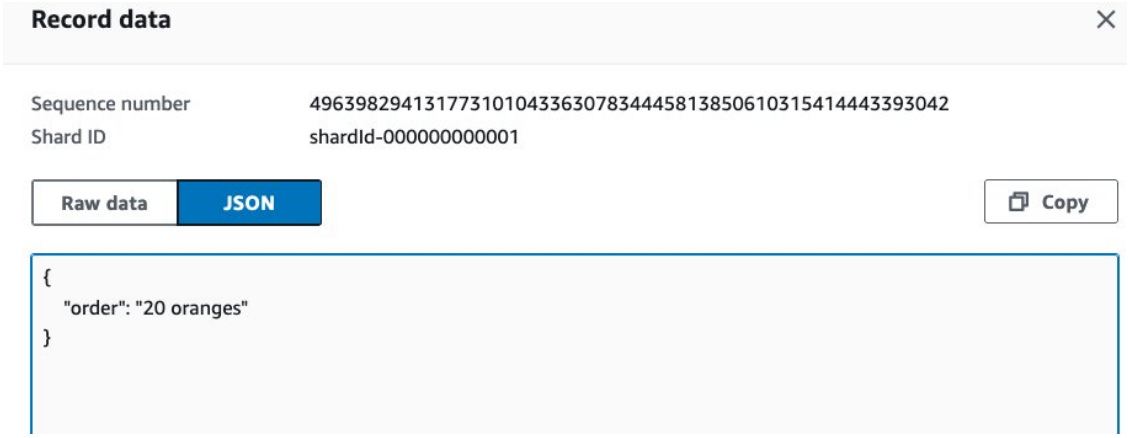


Create new test data

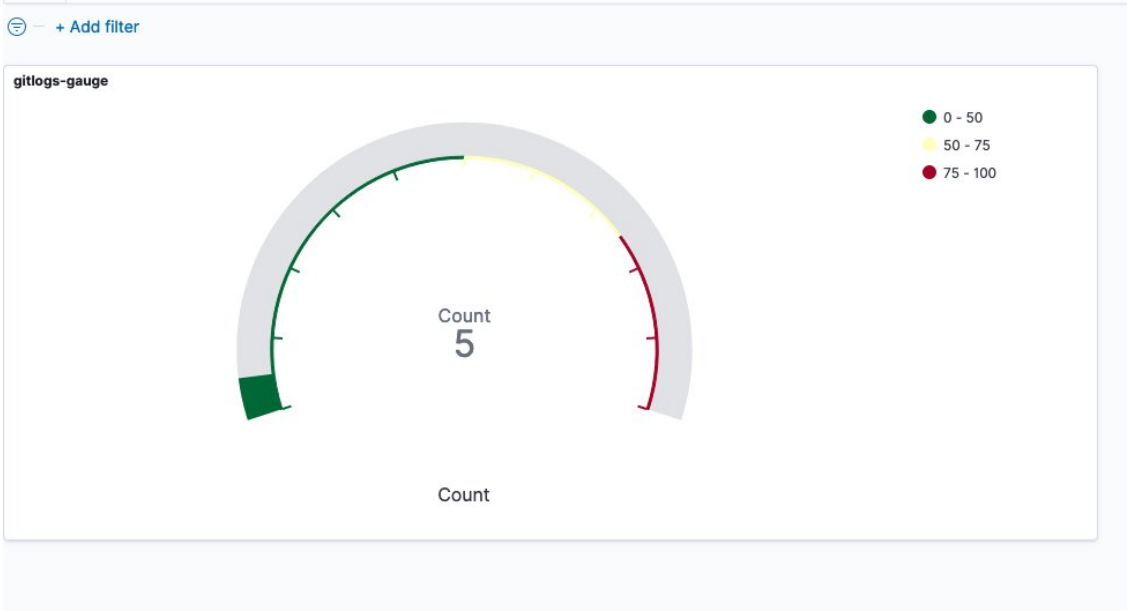


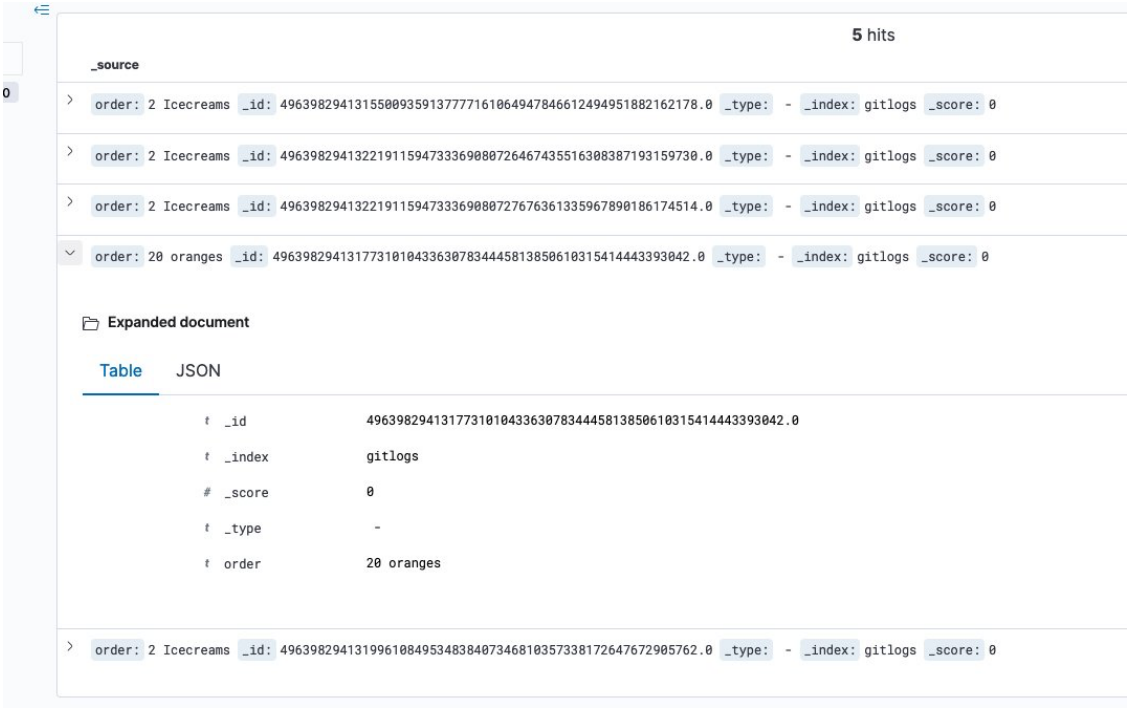
And test this data notice it got ingested successfully to shard 1.

Data ingested in Kinesis as well.



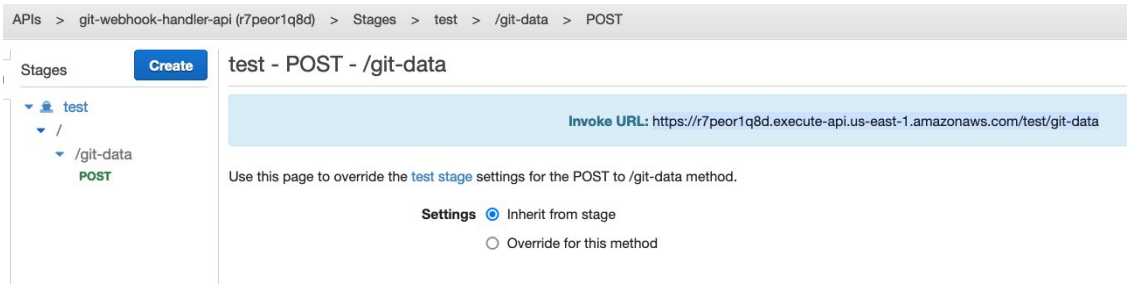
OpenSearch count is updated as well



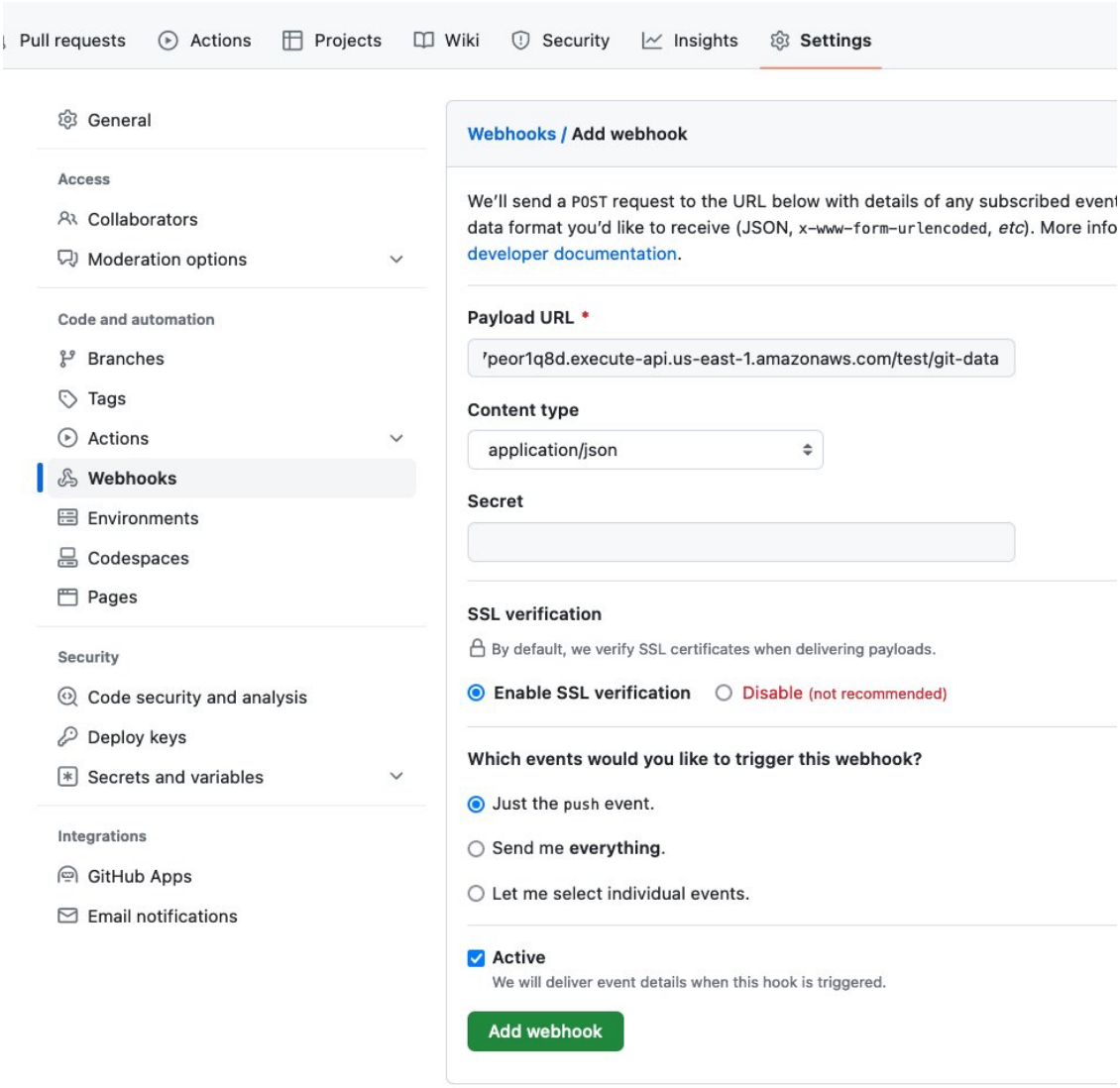


12. Next lets update the GitHub Repository Webhook URL to AWS REST API and stream real GitHub events.

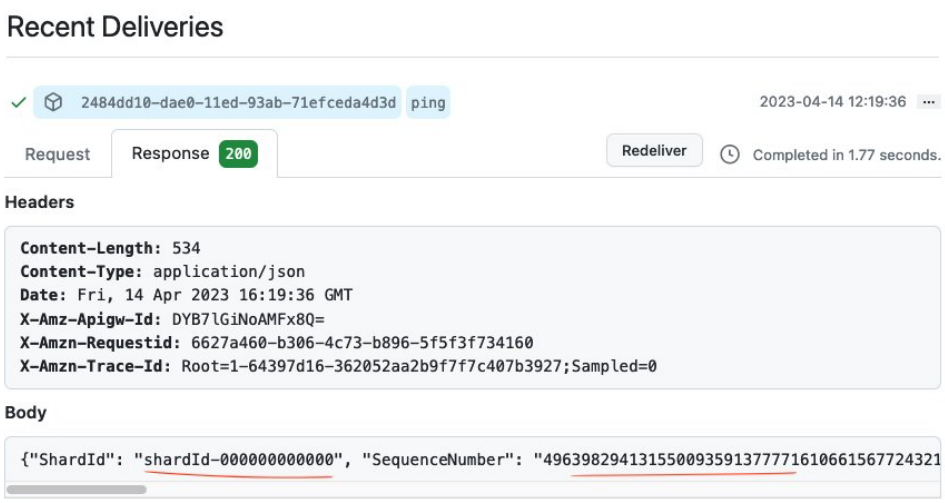
Copy the POST URL from REST API



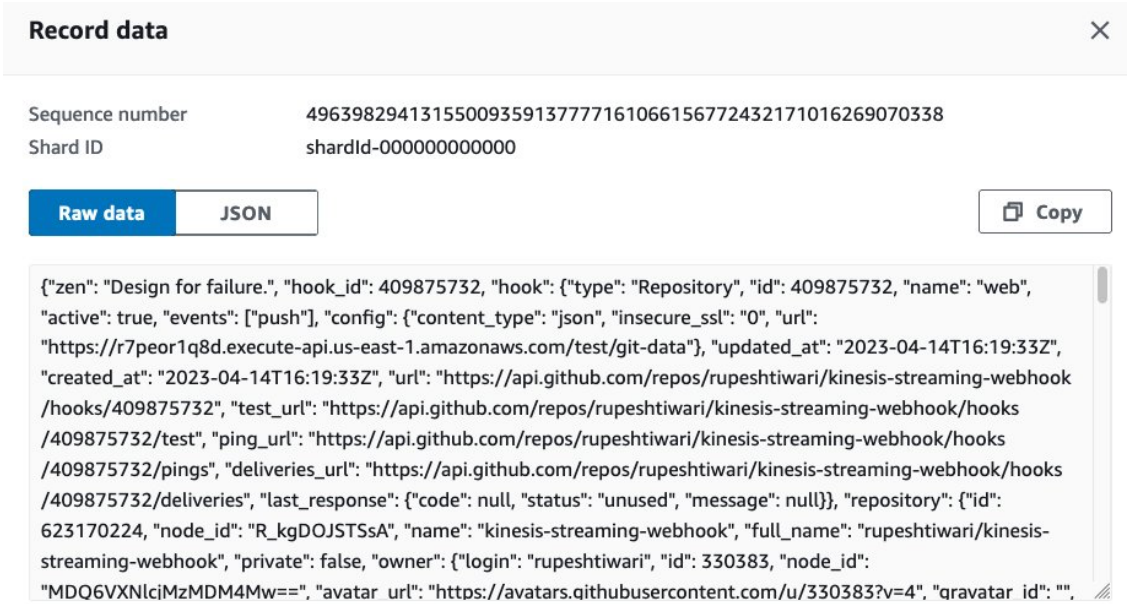
Go to GitHub open existing repository or create a new one then go to the **Settings,** select **Webhooks ,** select **Add Webhook,** enter AWS REST API URL, change content type to **application/json** and select Add webhook button.



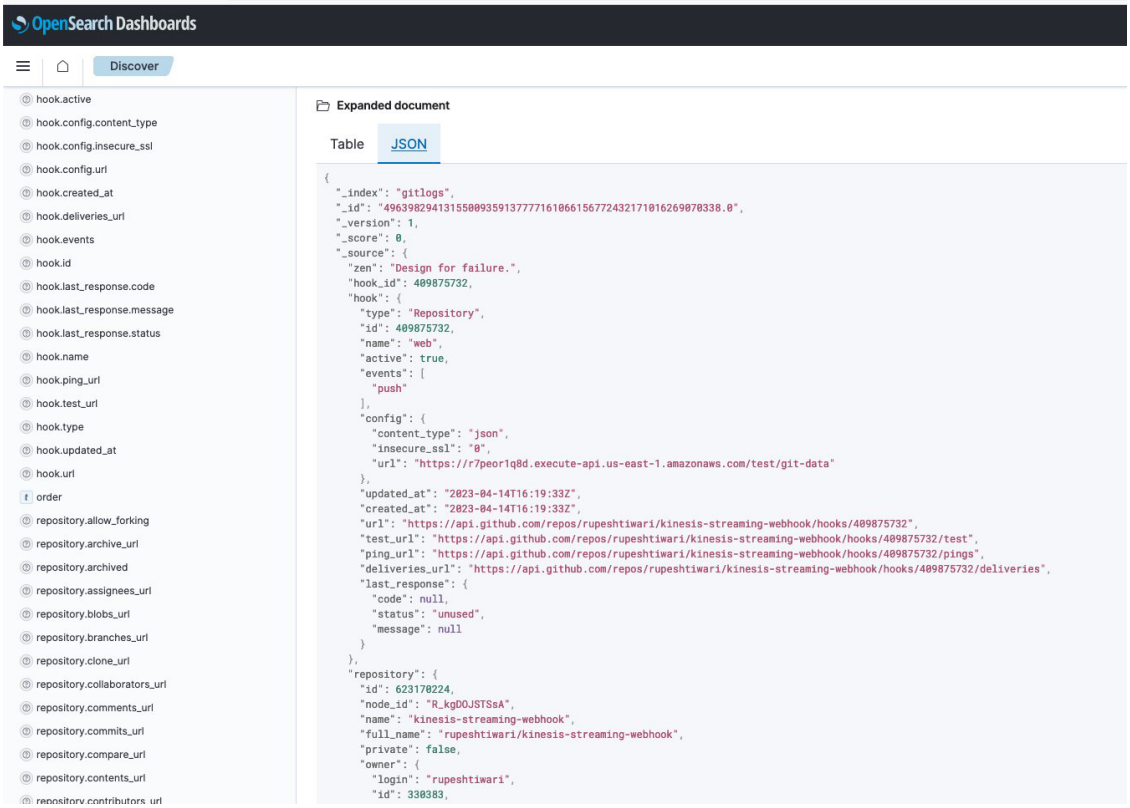
Next select the Star button in the GitHub repository. Open Webhook

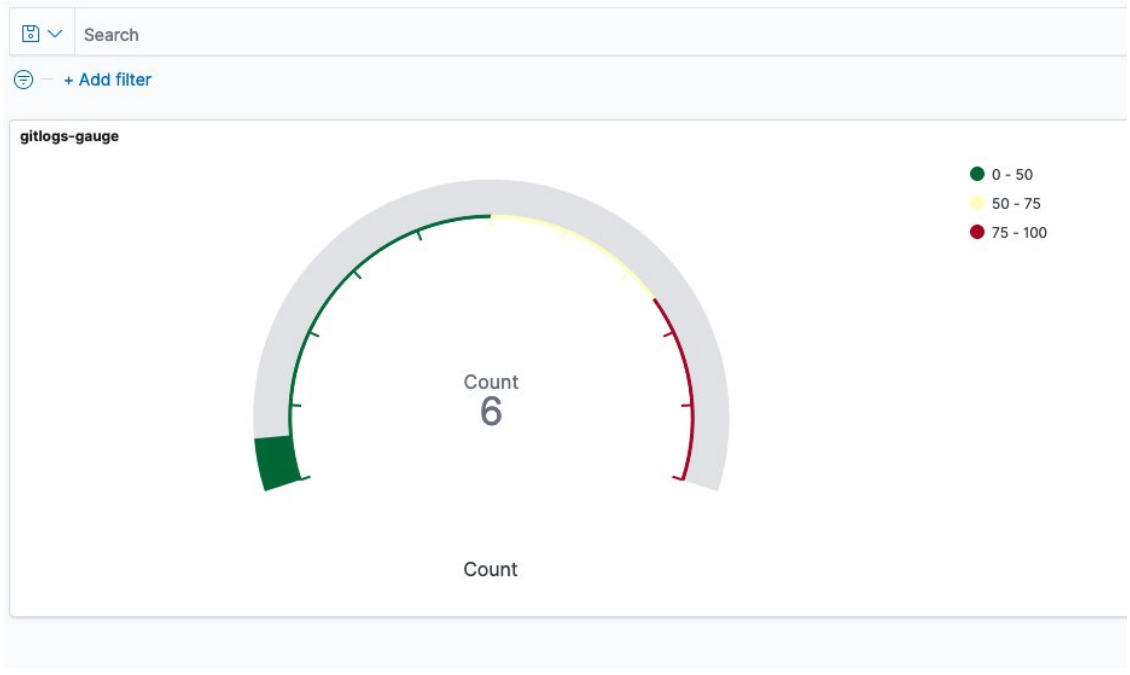


Open Kinesis and confirm git log is arrived.

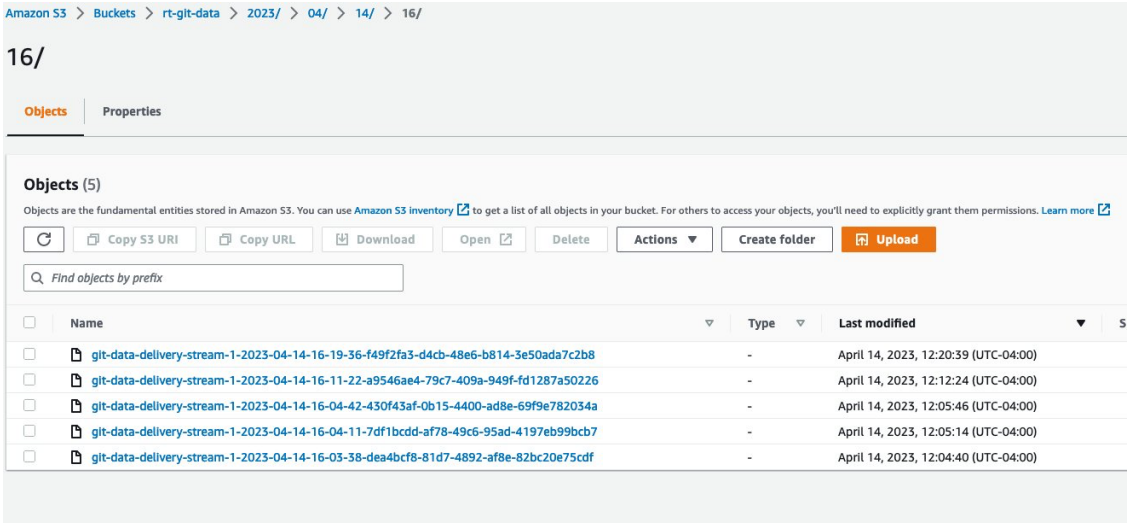


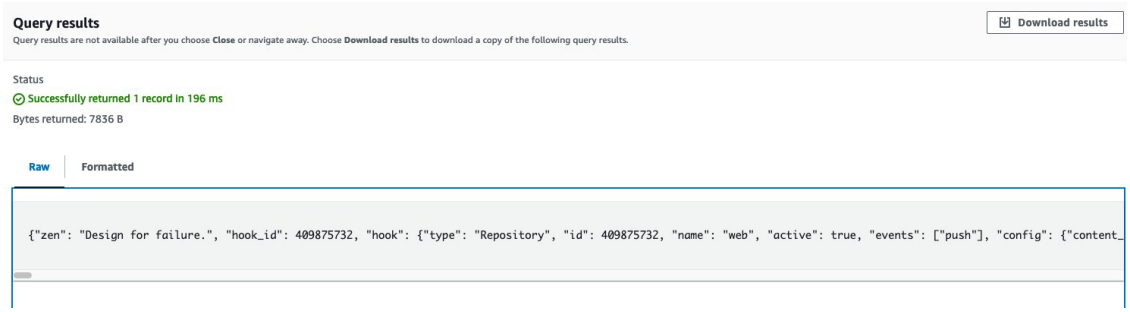
Open Amazon OpenSearch service and confirm git log is arrived.





Open S3 bucket and verify the git log is arrived


Amazon S3 > Buckets > rt-git-data > 2023/ > 04/ > 14/ > 16/
16/
Objects
Properties
Objects (5)
Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more
Copy S3 URI
Copy URL
Download
Open
Delete
Actions
Create folder
Upload
Find objects by prefix
Name
Type
Last modified
git-data-delivery-stream-1-2023-04-14-16-19-36-f49f2fa3-d4cb-48e6-b814-3e50ada7c2b8 - April 14, 2023, 12:20:39 (UTC-04:00)
git-data-delivery-stream-1-2023-04-14-16-11-22-a9546ae4-79c7-409a-949f-fd1287a50226 - April 14, 2023, 12:12:24 (UTC-04:00)
git-data-delivery-stream-1-2023-04-14-16-04-42-430f43af-0b15-4400-ad8e-69f9e782034a - April 14, 2023, 12:05:46 (UTC-04:00)
git-data-delivery-stream-1-2023-04-14-16-04-11-7df1bcdd-af78-49c6-95ad-4197eb99bcb7 - April 14, 2023, 12:05:14 (UTC-04:00)
git-data-delivery-stream-1-2023-04-14-16-03-38-dea4bcf8-81d7-4892-af8e-82bc20e75cdf - April 14, 2023, 12:04:40 (UTC-04:00)


Query results
Query results are not available after you choose Close or navigate away. Choose Download results to download a copy of the following query results.
Download results
Status
Successfully returned 1 record in 196 ms
Bytes returned: 7836 B
Raw
Formatted
{"zen": "Design for failure.", "hook_id": 409875732, "hook": {"type": "Repository", "id": 409875732, "name": "web", "active": true, "events": ["push"], "config": {"content_