accelerated-data-lake

Nov 2023

https://github.com/aws-samples/accelerated-data-lake

CloudFormation

- `wildrydes-dev-datalake-structure`
- `wildrydes-dev-datalake-elasticsearch`
- wildrydes-dev-datalake-elasticsearch-lambdas
- wildrydes-dev-datalake-staging-engine

AWS Accelerated Data Lake (3x3x3)

A packaged Data Lake solution, that builds a highly functional Data Lake,
Elasticsearch

License

This library is licensed under the Apache 2.0 License.

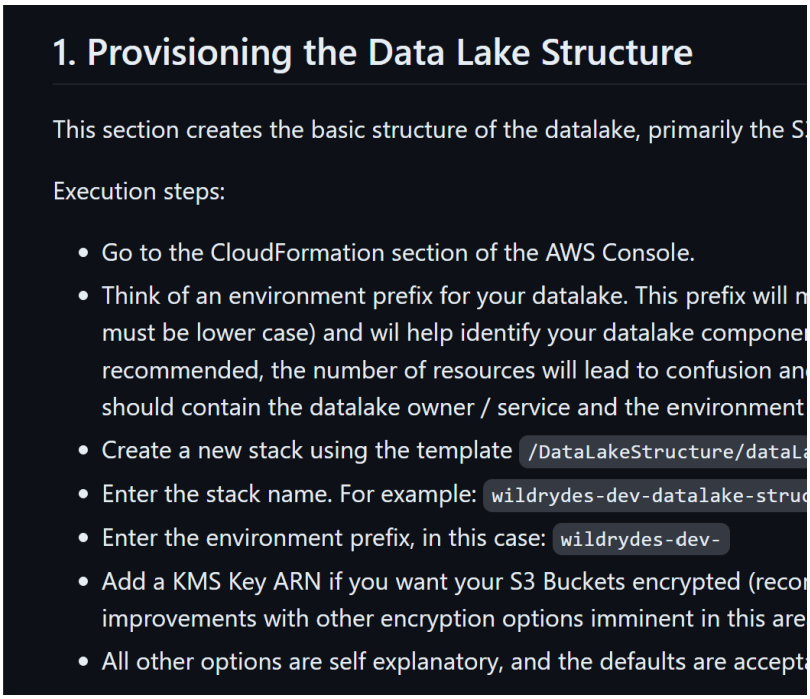
3x3x3 DataLake installation instructi

These are the steps required to provision the 3x3x3 Packaged Datalake So

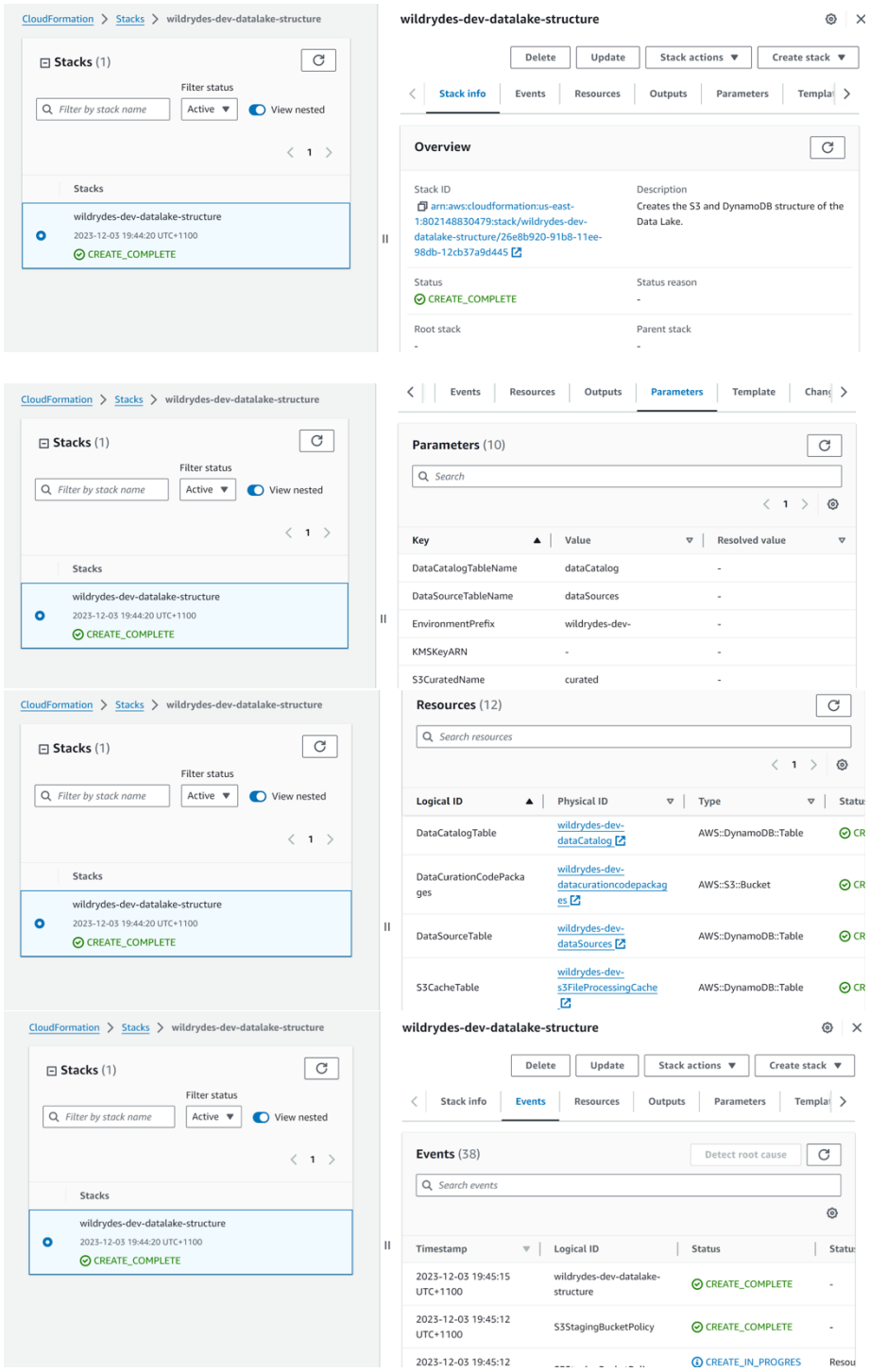
- Provision the Data Lake Structure (5 minutes)
- Provision the Visualisation
  - Provision Elasticsearch (15 minutes)
  - Provision Visualisation Lambdas (5 minutes)
- Provision the Staging Engine and add a trigger (10 minutes)
- Configure a sample data source and add data (5 minutes)
  - Configure the sample data source
  - Ingress a sample file for the new data source
  - Ingress a sample file that has an incorrect schema
- Initialise and use Elasticsearch / Kibana (5 minutes)

Once these steps are complete, the provided sample data file (or any file
can be dropped in the DataLake's raw bucket and will be immediately sta

STEPS:

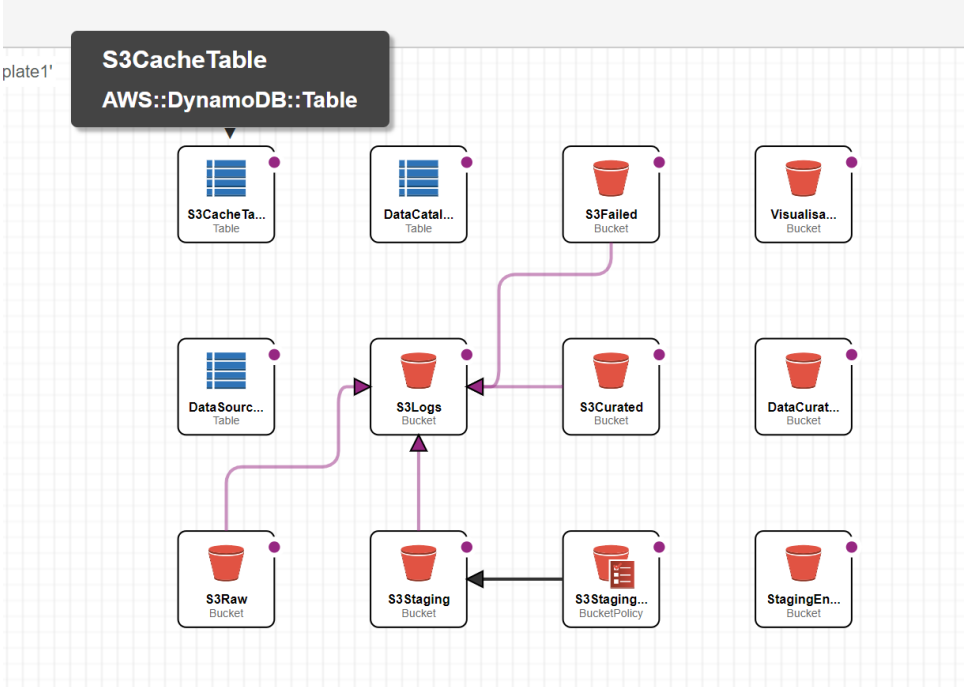


wildrydes-dev-datalake-structure (CloudFormation)



Resources

- DataCatalogTable
- DataCurationCodePackages
- DataSourceTable
- S3CacheTable
- S3Curated
- S3Failed
- S3Logs
- S3Raw
- S3Staging
- S3StagingBucketPolicy
- StagingEngineCodePackages
- VisualisationCodePackages



2. Provisioning the Visualisations

This step is optional, but highly recommended. If the customer does not want elasticsearch, a temporary cluster will allow debugging while the datalake is set up, and will illustrate its value.

If you want elasticsearch visualisation, both of the following steps are required (in order).

2.1 Provision Elasticsearch

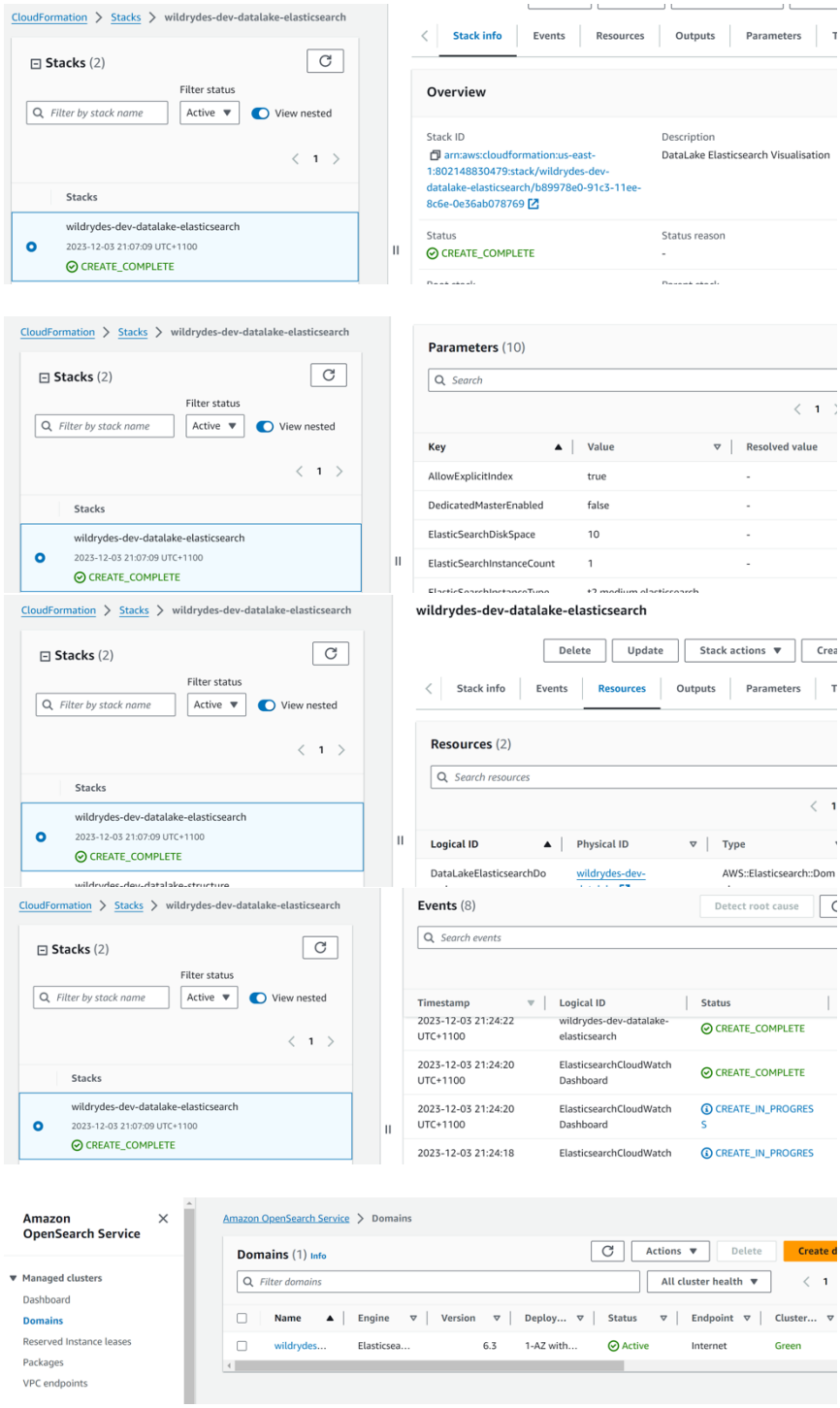
This step creates the elasticsearch cluster the datalake will use. For dev databases, a single T2 instance is acceptable. For production instances, standard elasticsearch best practise apply.

NOTE: Elasticsearch is a very easy service to over-provision. To assist in determining the correct resources, the cloudformation template includes a CloudWatch dashboard to display all relevant cluster metrics.

Execution steps:

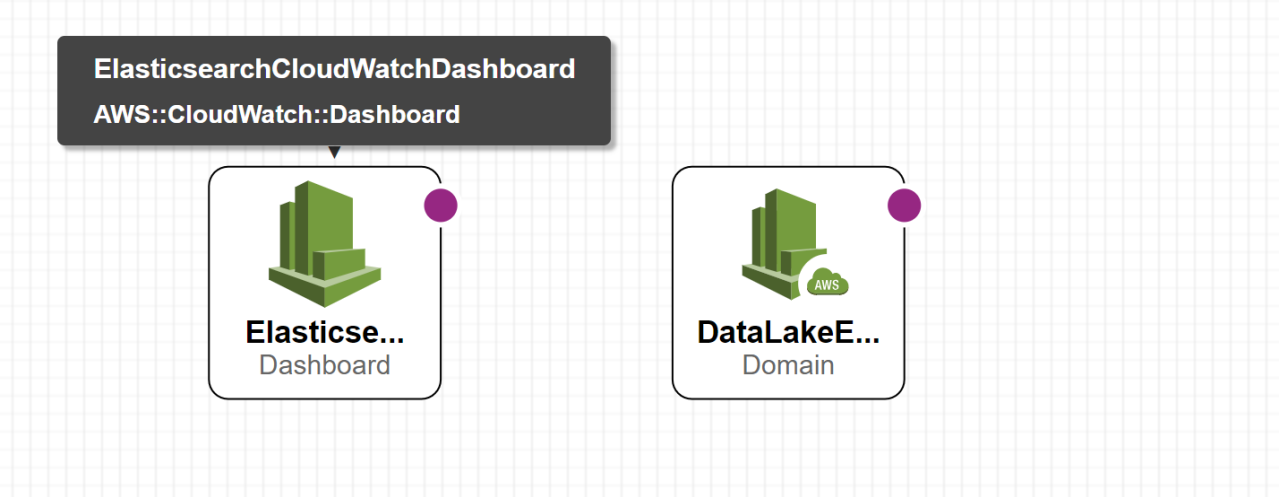
- Go to the CloudFormation section of the AWS Console.
- Create a new stack using the template `/Visualisation/elasticsearch/elasticsearch.yaml`
- Enter the stack name. For example: `wildrydes-dev-datalake-elasticsearch`
- Enter the environment prefix, in this case: `wildrydes-dev-`
- Enter the your ip addresses (comma separated), in this case: `<your_ip/32>,<another_cidr>`
- Change the other parameters as per requirements / best practise. The default values will provision a single instance `t2.medium` node - this is adequate for low tps dev and testing.

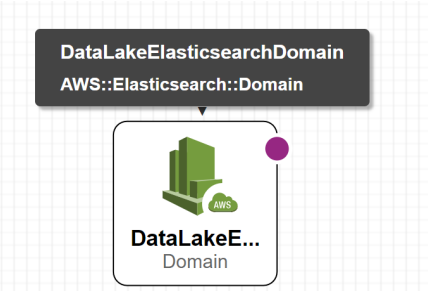
wildrydes-dev-datalake-elasticsearch (CloudFormation)



Resources

- DataLakeElasticsearchDomain
- ElasticsearchCloudWatchDashboard





2.2 Provision the Visualisation Lambdas

This step creates a lambda which is triggered by changes to the data catalog DynamoDB table. The lambda takes the changes and sends them to the elasticsearch cluster created above.

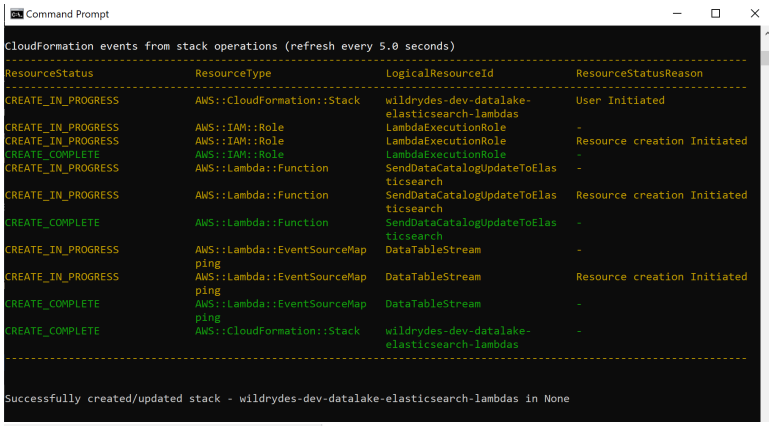
Execution steps:

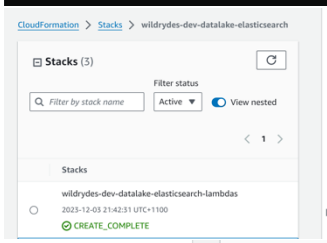
- Create a data lake IAM user, with CLI access.
- Configure the AWS CLI with the user's access key and secret access key.
- Install AWS SAM.
- Open a terminal / command line and move to the Visualisation/lambdas/ folder
- Package and deploy the lambda functions. There are following two ways to deploy it:
  - Execute the `./deploy.sh <environment_prefix>` script OR
  - Execute the AWS SAM package and deploy commands detailed in: deploy.txt

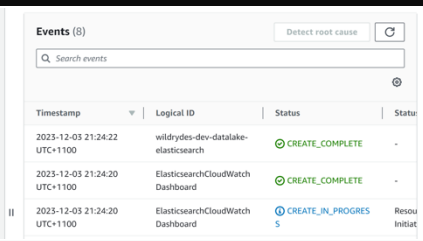
For this example, the commands should be:

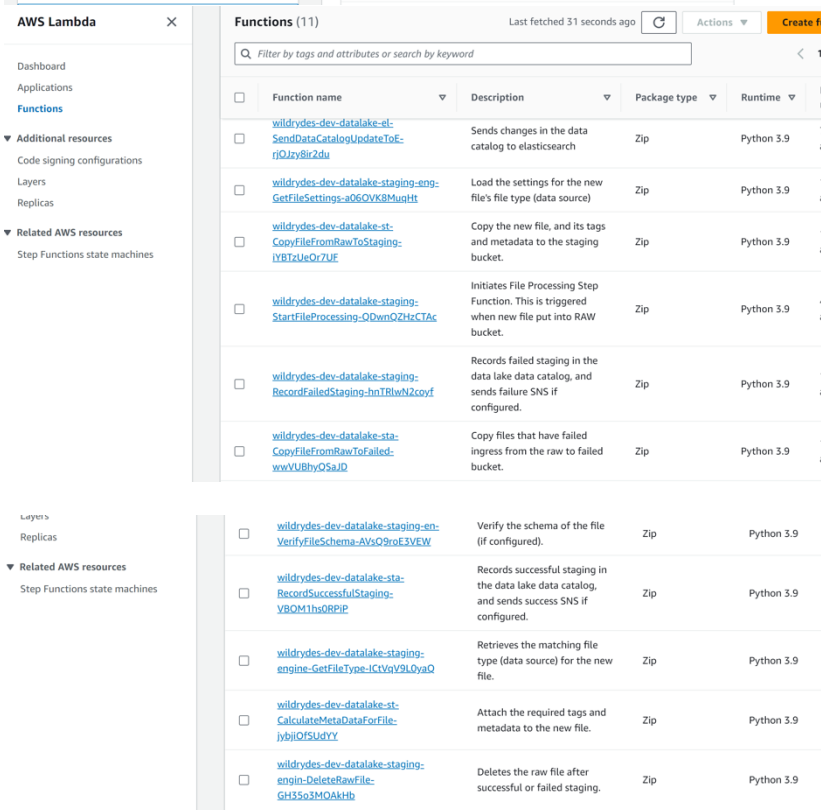
```
sam package --template-file ./lambdaDeploy.yaml --output-template-file lambdaDeployCFN.yaml --s3-bucket r

sam deploy --template-file lambdaDeployCFN.yaml --stack-name wildrydes-dev-datalake-elasticsearch-lambdas --c
```

wildrydes-dev-datalake-elasticsearch-lambdas
(CloudFormation)



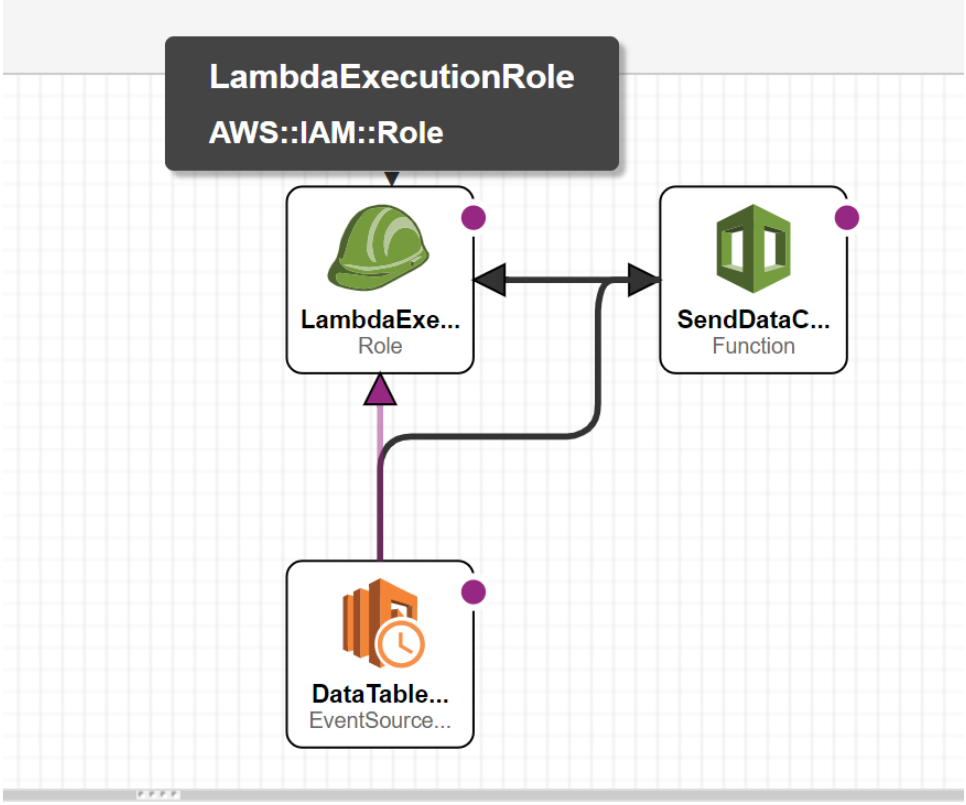


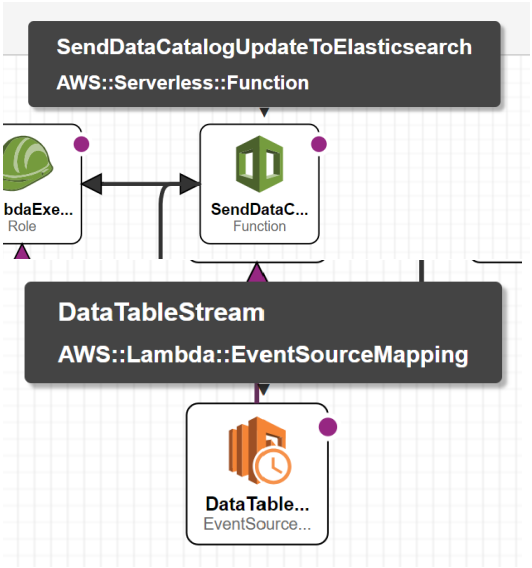




Resources

- DataTableStream
- LambdaExecutionRole
- SendDataCatalogUpdateToElasticsearch


LambdaExecutionRole
AWS::IAM::Role
LambdaExe...
Role
SendDataC...
Function
DataTable...
EventSource...


SendDataCatalogUpdateToElasticsearch
AWS::Serverless::Function
bdaExe...
Role
SendDataC...
Function
DataTableStream
AWS::Lambda::EventSourceMapping
DataTable...
EventSource...

3. Provision the Staging Engine and add a trigger

This is the workhorse of 3x3x3 - it creates lambdas and a step function, that takes new files dropped into the raw bucket, verifies their source and schema, applies tags and metadata, then copies the file to the staging bucket.

On both success and failure, 3x3x3 updates the DataCatalog table in DynamoDB. All changes to this table are sent to elasticsearch, allowing users to see the full history of all imports and see what input files were used in each DataLake query.

Execution steps: (ignore these steps if already done in the visualisation step)

- Create a data lake IAM user, with CLI access.
- Configure the AWS CLI with the user's access key and secret access key.
- Install AWS SAM. (mandatory steps)
- Open a terminal / command line and move to the StagingEngine/ folder
- Package and deploy the lambda functions. There are following two ways to deploy it: * Execute the `./deploy.sh <environment_prefix>` script OR * Execute the AWS SAM package and deploy commands detailed in: `deploy.txt`

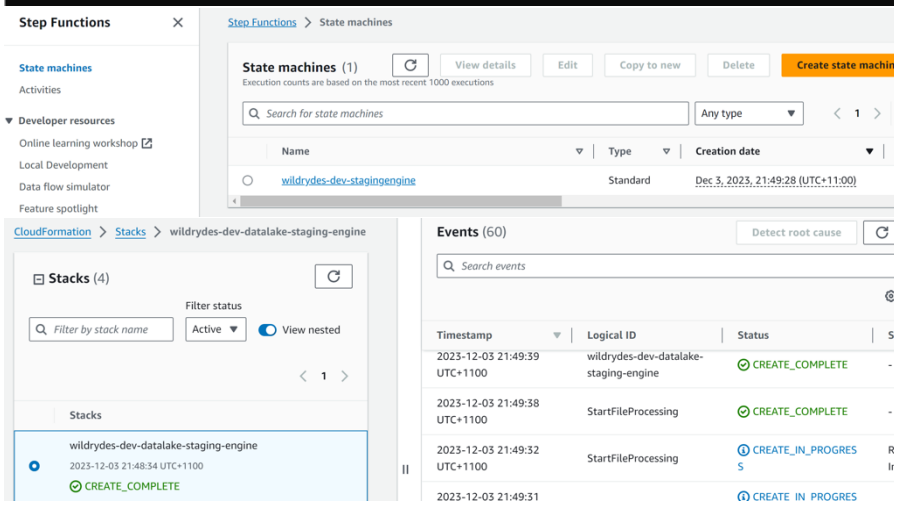
For this example, the commands should be:

```
sam package --template-file ./stagingEngine.yaml --output-template-file stagingEngineDeploy.yaml --s3-bu

sam deploy --template-file stagingEngineDeploy.yaml --stack-name wildrydes-dev-datalake-staging-engine --capa
```

wildrydes-dev-datalake- staging-engine (CloudFormation)

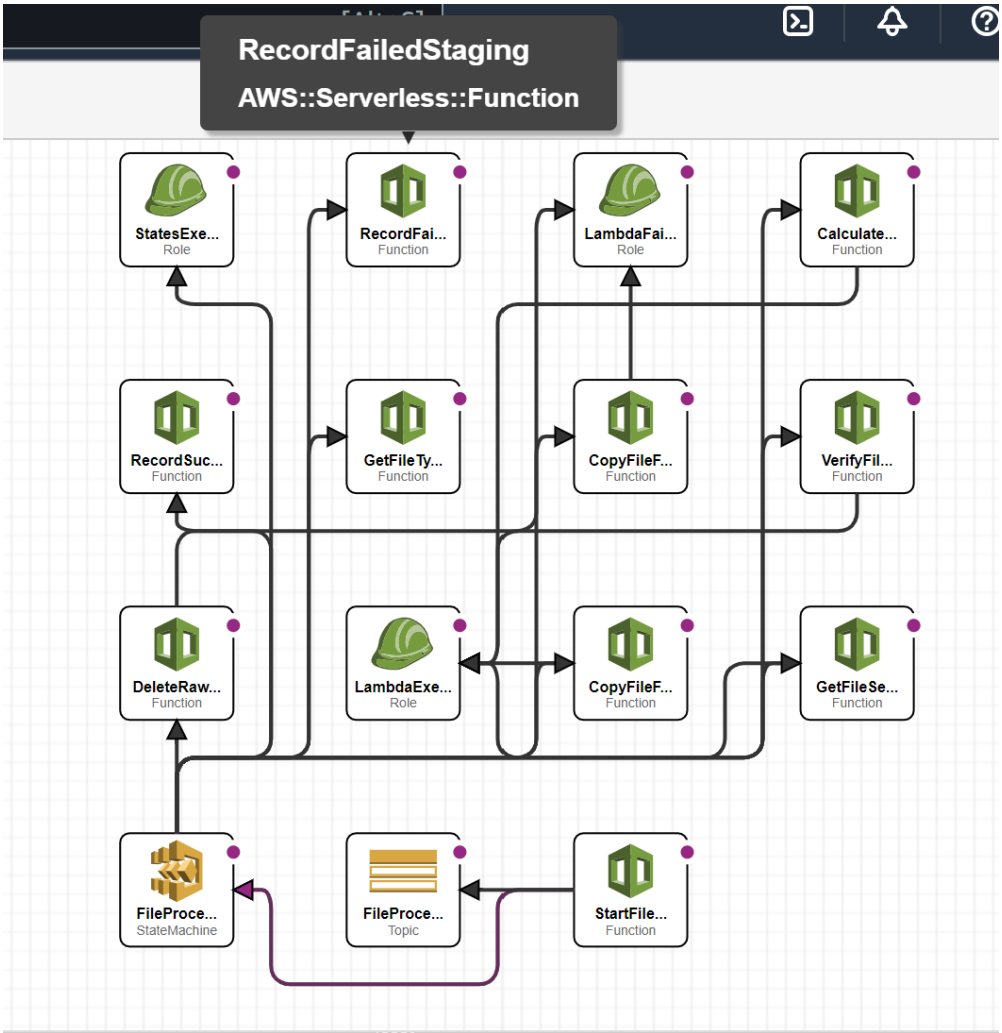
```
C:\Users\rohit\Downloads\accelerated-data-lake-master\StagingEngine>sam package --template-file ./stagingEngine.ya
output-template-file stagingEngineDeploy.yaml --s3-bucket wildrydes-dev-stagingenginecodepackages
        Uploading to c2c746de4475cbbb009e7955f50d82d4  2430 / 2430  (100.00%)
        Uploading to e22a256a6546a346b3299d765436d246  1963 / 1963  (100.00%)
        Uploading to e69b14b90713f2217e913f9b0c01500d  1175 / 1175  (100.00%)
        Uploading to d8f6db6adcb98efe82cea901fe1fef8c  105238 / 105238  (100.00%)
        Uploading to 740c13b275fd437021967e5fe84a1b57  1121 / 1121  (100.00%)
        Uploading to 8230e5c153012360ebcada722932c03a  1931 / 1931  (100.00%)
        Uploading to 2984cd61284eefe16ade2bada54e8ea4  657 / 657  (100.00%)
        Uploading to 93818b71fa01ef39dec70ebc5f654197  1367 / 1367  (100.00%)
        Uploading to 0bd59e94d249580630bed97ccfd9cdfe  758 / 758  (100.00%)
        Uploading to b9b4a82873d0f81831e8bbd02d92f663  1536 / 1536  (100.00%)

Successfully packaged artifacts and wrote output template to file stagingEngineDeploy.yaml.
Execute the following command to deploy the packaged template
sam deploy --template-file C:\Users\rohit\Downloads\accelerated-data-lake-master\StagingEngine\stagingEngineDeploy
 --stack-name <YOUR STACK NAME>
```

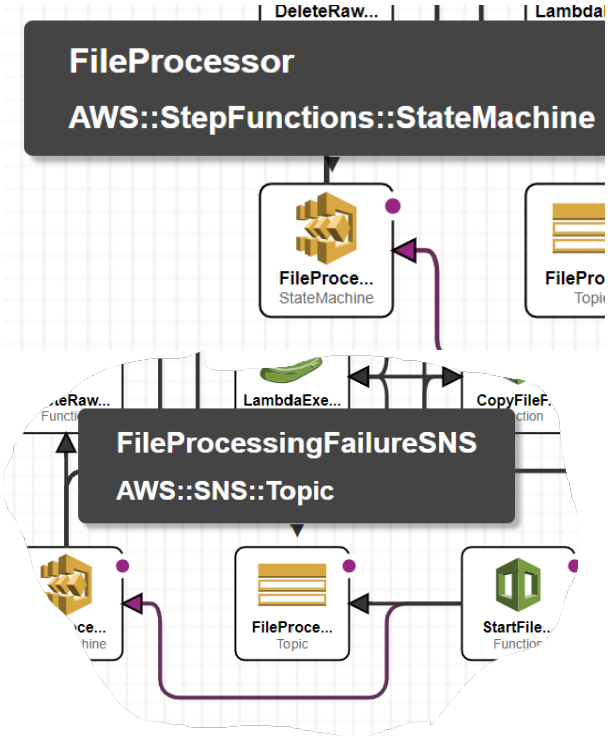
```
Command Prompt
CREATE_IN_PROGRESS            AWS::Lambda::Function          RecordFailedStaging            -
CREATE_IN_PROGRESS            AWS::Lambda::Function          RecordFailedStaging            Resource creation
                                                                                            Initiated
CREATE_COMPLETE               AWS::Lambda::Function          RecordSuccessfulStaging        -
CREATE_COMPLETE               AWS::Lambda::Function          GetFileType                    -
CREATE_COMPLETE               AWS::Lambda::Function          CopyFileFromRawToStaging       -
CREATE_COMPLETE               AWS::Lambda::Function          CalculateMetaDataForFile       -
CREATE_COMPLETE               AWS::Lambda::Function          VerifyFileSchema               -
CREATE_COMPLETE               AWS::Lambda::Function          DeleteRawFile                  -
CREATE_COMPLETE               AWS::Lambda::Function          GetFileSettings                -
CREATE_COMPLETE               AWS::Lambda::Function          CopyFileFromRawToFailed        -
CREATE_COMPLETE               AWS::Lambda::Function          RecordFailedStaging            -
CREATE_IN_PROGRESS            AWS::StepFunctions::StateM     FileProcessor                  -
                              achine
CREATE_IN_PROGRESS            AWS::StepFunctions::StateM     FileProcessor                  Resource creation
                              achine                                                        Initiated
CREATE_COMPLETE               AWS::StepFunctions::StateM     FileProcessor                  -
                              achine
CREATE_IN_PROGRESS            AWS::Lambda::Function          StartFileProcessing            -
CREATE_IN_PROGRESS            AWS::Lambda::Function          StartFileProcessing            Resource creation
                                                                                            Initiated
CREATE_COMPLETE               AWS::Lambda::Function          StartFileProcessing            -
CREATE_COMPLETE               AWS::CloudFormation::Stack     wildrydes-dev-datalake-        -
                                                             staging-engine
-------------------------------------------------------------------------------------------------------------

Successfully created/updated stack - wildrydes-dev-datalake-staging-engine in None
```



Resources

- CalculateMetaDataForFile
- CopyFileFromRawToFailed
- CopyFileFromRawToStaging
- DeleteRawFile
- FileProcessingFailureSNS
- FileProcessor
- GetFileSettings
- GetFileType
- GetFileTypeRole
- LambdaExecutionRole
- LambdaFailedFileProcessor Role
- RecordFailedStaging
- RecordFailedStagingRole
- RecordSuccessfulStaging
- RecordSuccessfulStagingRole
- StartFileProcessing
- StartFileProcessingRole
- StatesExecutionRole
- VerifyFileSchema


RecordFailedStaging
AWS::Serverless::Function
StatesExe...
Role
RecordFai...
Function
LambdaFai...
Role
Calculate...
Function
RecordSuc...
Function
GetFileTy...
Function
CopyFileF...
Function
VerifyFil...
Function
DeleteRaw...
Function
LambdaExe...
Role
CopyFileF...
Function
GetFileSe...
Function
FileProce...
StateMachine
FileProce...
Topic
StartFile...
Function


DeleteRaw...
Lambdal
FileProcessor
AWS::StepFunctions::StateMachine
FileProce...
StateMachine
FilePro
Topi
LambdaExe...
CopyFileF.
FileProcessingFailureSNS
AWS::SNS::Topic
FileProce...
Topic
StartFile..

3.1 Add the Staging trigger

Cloudwatch cannot create and trigger lambdas from changes to existing S3 buckets - this forces the Staging Engine startFileProcessing lambda to have its S3 PUT trigger attached manually.

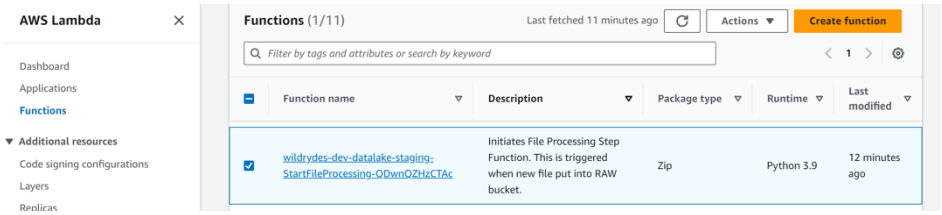
Execution steps:

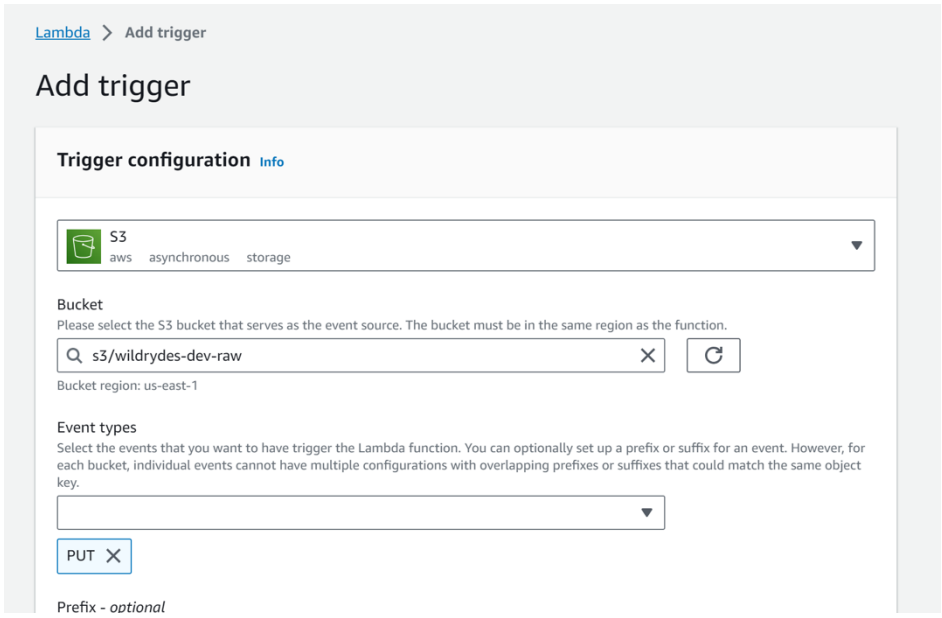
- Go into the AWS Console, Lambda screen.
- Find the lambda named: `<ENVIRONMENT_PREFIX>datalake-staging-StartFileProcessing-<RANDOM CHARS ADDED BY SAM>`
- Manually add an S3 trigger, generated from PUT events on the RAW bucket you created (in this example, this would be `wildrydes-dev-raw`)

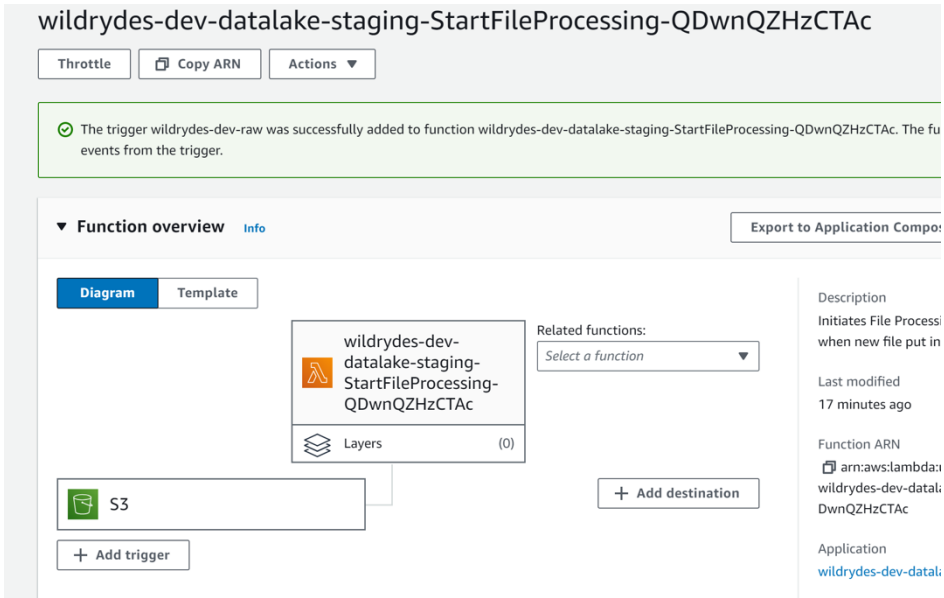
NOTE: Do not use "Object Created (All)" as a trigger - 3x3x3 copies new files when it adds their metadata, so a trigger on All will cause the staging process to begin again after the copy.

Congratulations! 3x3x3 is now fully provisioned! Now let's configure a datasource and add some data.

wildrydes-dev-datalake-structure (CloudFormation)







Paramaters

- DataCatalogTableNam e
- DataSourceTableName

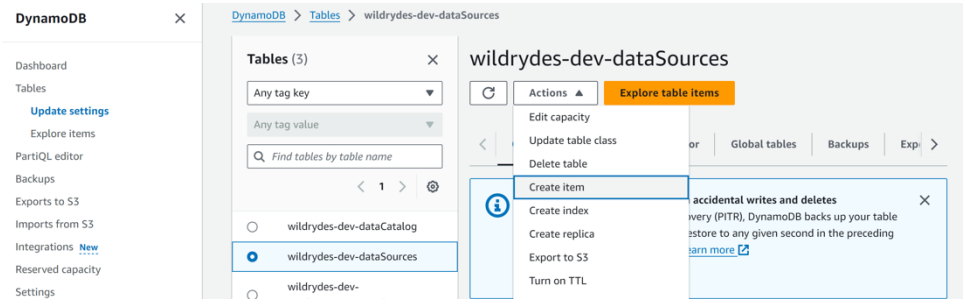
4. Configure a sample data source and add data

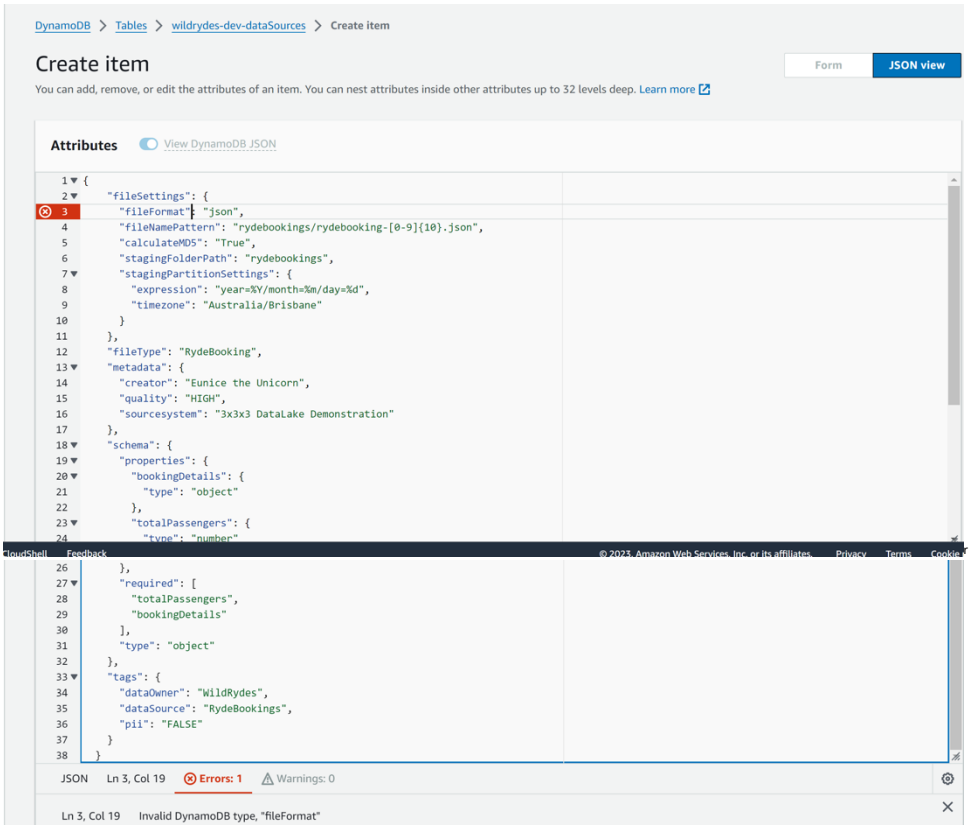
4.1 Configure the sample data source

Execution steps:

- Open the file `DataSources/RydeBookings/ddbDataSourceConfig.json`
- Copy the file's contents to the clipboard.
- Go into the AWS Console, DynamoDB screen.
- Open the DataSource table, which for the environment prefix used in this demonstration will be: `wildrydes-dev-dataSources`
- Go to the Items tab, click Create Item, switch to 'Text' view and paste in the contents of the `ddbDataSourceConfig.json` file.
- Save the item.

You now have a fully configured DataSource. The individual config attributes will be explained in the next version of this documentation.





GOT ERROR

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

4.2 Ingress a sample file for the new data source

Execution steps:

- Go into the AWS Console, S3 screen, open the raw bucket (`wildrydes-dev-raw` in this example)
- Create a folder "rydebookings". This is because the data source is configured to expect its data to be ingressed into a folder with this name (just use the bucket settings for the new folder).
- Using the console, upload the file `DataSources/RydeBookings/rydebooking-1234567890.json` into this folder.
- Confirm the file has appeared in the staging folder, with a path similar to: `wildrydes-dev-staging/rydebookings/2018/10/26/rydebooking-1234567890.json`

If the file is not in the staging folder, one of the earlier steps has been executed incorrectly.

4.3 Optional. Ingress a sample file that has an incorrect schema

Execution steps:

- Go into the AWS Console, S3 screen, open the raw bucket (`wildrydes-dev-raw` in this example)
- Create a folder "rydebookings" if it does not already exist.
- Using the console, upload the file `DataSources/RydeBookings/rydebooking-2000000000.json` into this folder.
- Confirm the file has appeared in the failed folder, with a path similar to: `wildrydes-dev-failed/rydebookings/rydebooking-2000000000.json`

If the file is not in the failed folder, one of the earlier steps has been executed incorrectly.

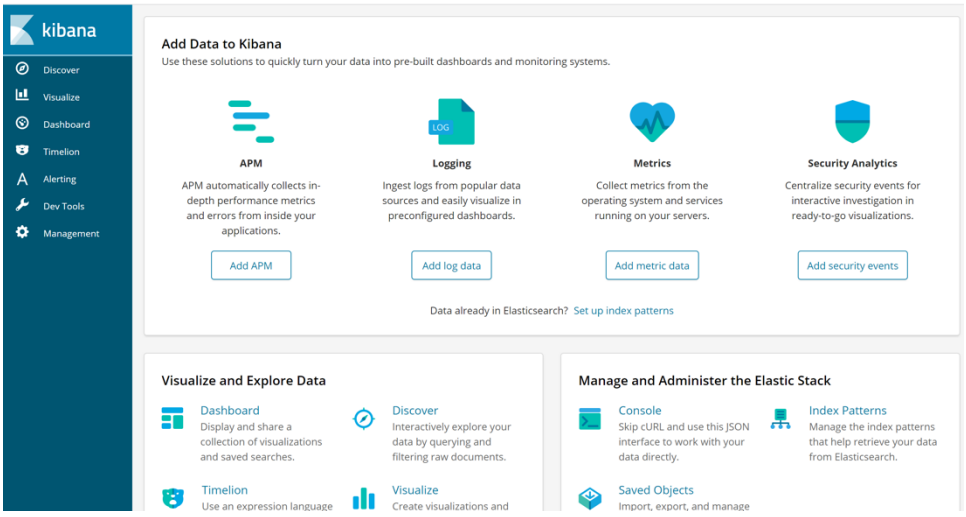
5. Initialise and use Elasticsearch / Kibana

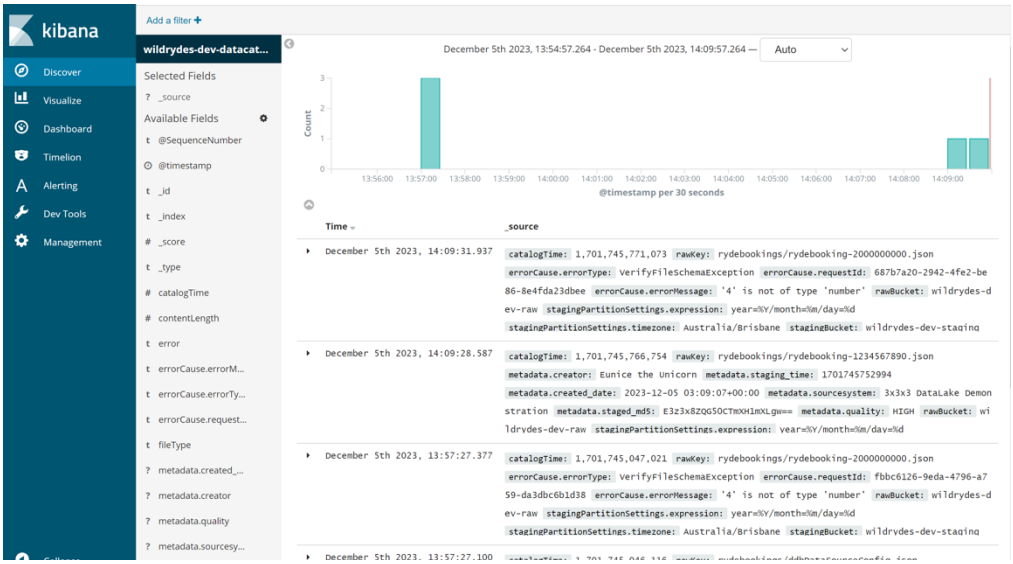
The above steps will have resulted in the rydebookings files being entered into the DynamoDB datacatalog table (`wildrydes2-dev-dataCatalog`). The visualisation steps subscribed to these table's stream and all updates are now sent to elasticsearch.

The data will already be in elasticsearch, we just need to create a suitable index.

Execution steps:

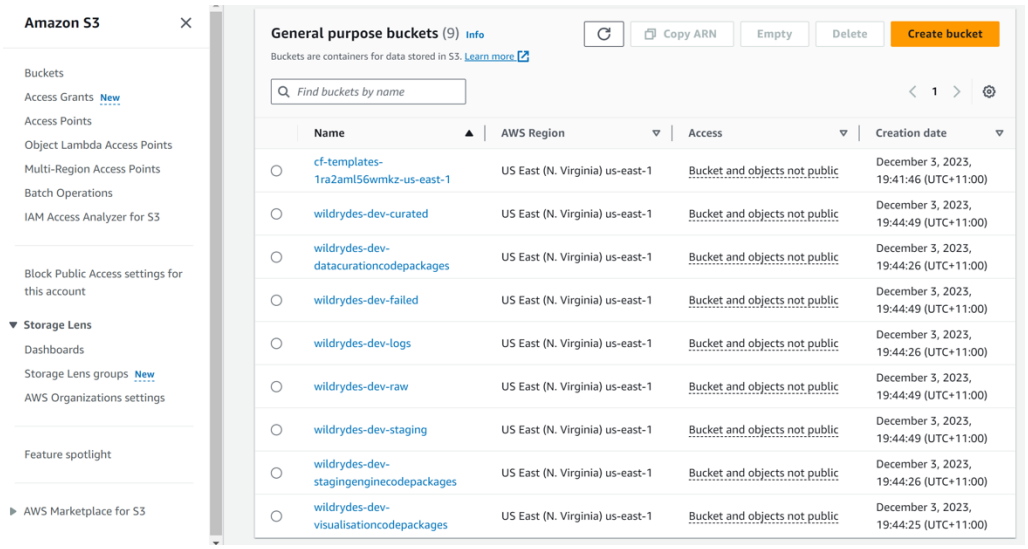
- Go to the kibana url (found in the AWS Console, under elasticsearch)
- You will see there is no data - this is because the index needs to be created (the data is present, so we will let kibana auto-create it)
- Click on the management tab, on the left.
- Click "Index Patterns"
- Paste in: `wildrydes-dev-datacatalog` (so `<ENVIRONMENT_PREFIX>datacatalog`). You will see this name in the available index patterns at the base of the screen.
- Click "Next step"
- Select `@Timestamp` in the "Time Filter field name" field - this is very important, otherwise you will not get the excellent kibana timeline.
- Click "Create Index Pattern" and the index will be created. Click on the Discover tab to see your data catalog and details of your failed and successful ingress.



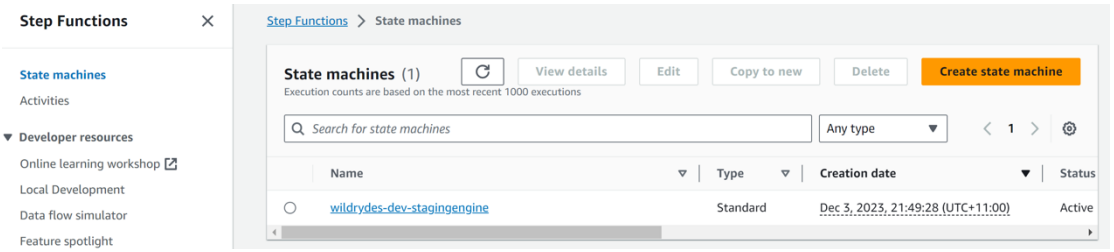


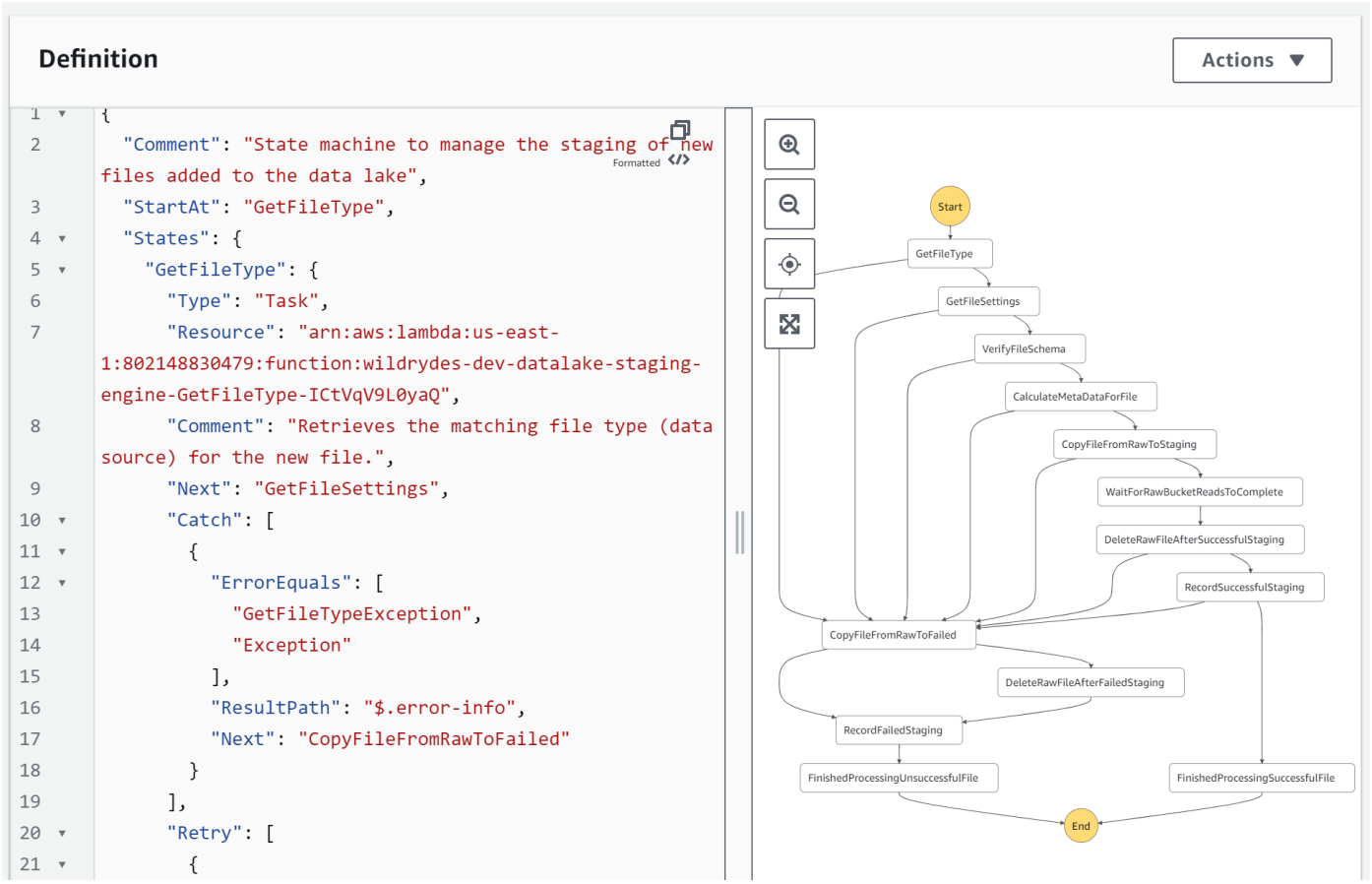
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

S3 Bucket



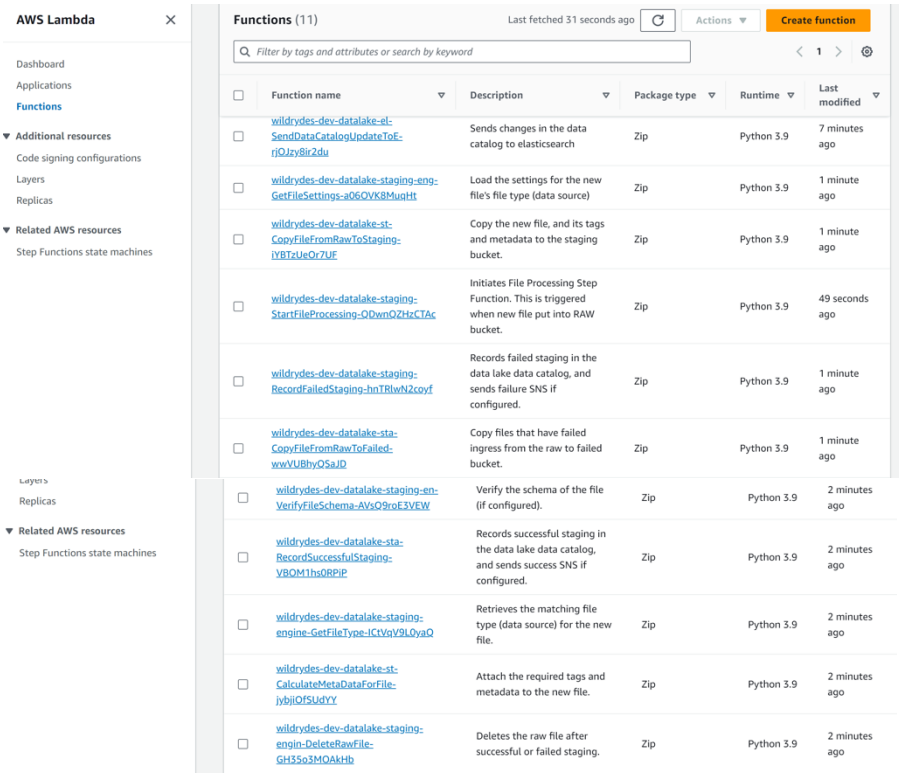
STEP FUNCTION





All these boxes are **States** of the file(data) that are obtained by passing through every single **Lambda Function**.

LAMBDAS



GetFileType

1. # GetFileSettings

2. # VerifyFileSchema

3. # CalculateMetaDataForFile

4. # CopyFileFromRawToStaging

5. # WaitForRawBucketReadsToComplete

6. **DeleteRawFileAfterSuccessfulStatging**

7. **RecordSuccessfulStaging**

8. **CopyFileFromRawToFailed**

9. **DeleteRawFileAfterFailedStaging**

10. **RecordFailedStaging**