

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, with Elasticsearch

License

This library is licensed under the Apache 2.0 License.

3x3x3 DataLake installation instructions

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

- 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 you choose) can be dropped in the DataLake's raw bucket and will be immediately staged.

STEPS:

1. Provisioning the Data Lake Structure

This section creates the basic structure of the datalake, primarily the S3 buckets.

Execution steps:

- Go to the CloudFormation section of the AWS Console.
- Think of an environment prefix for your datalake. This prefix will name all resources (it must be lower case) and will help identify your datalake components. It is highly recommended, the number of resources will lead to confusion and it is better to have a prefix that should contain the datalake owner / service and the environment.
- Create a new stack using the template [/DataLakeStructure/datalake-structure](#).
- Enter the stack name. For example: `wildrydes-dev-datalake-structure`
- Enter the environment prefix, in this case: `wildrydes-dev-`
- Add a KMS Key ARN if you want your S3 Buckets encrypted (recommendations and improvements with other encryption options imminent in this area).
- All other options are self explanatory, and the defaults are acceptable.

wildrydes-dev-datalake-structure (CloudFormation)

Resources

- DataCatalogTable
- DataCurationCodePackage
- DataSourceTable
- S3CacheTable
- S3Curated
- S3Failed
- S3Logs
- S3Raw
- S3Staging
- S3StagingBucketPolicy
- StagingEngineCodePackag
- es
- VisualisationCodePackage
- s

The screenshot illustrates the AWS CloudFormation interface during the creation of a new stack. On the left, the 'Stacks' list shows a single stack named 'wildrydes-dev-datalake-structure' with a status of 'CREATE_COMPLETE'. On the right, the 'Overview' tab displays the stack's ID, description (which creates S3 and DynamoDB structures for the Data Lake), and current status. The 'Parameters' tab lists several parameters with their resolved values, such as 'DataCatalogTableName' set to 'dataCatalog'. Below these, the 'Resources' tab shows four resources: 'DataCatalogTable', 'DataCurationCodePackages', 'DataSourceTable', and 'S3CacheTable', each with its logical ID and physical ID.

CloudFormation > Stacks > wildrydes-dev-datalake-structure

Stacks (1)

Stacks

wildrydes-dev-datalake-structure
2023-12-03 19:44:20 UTC+1100
CREATE_COMPLETE

wildrydes-dev-datalake-structure

Overview

Stack ID: arn:aws:cloudformation:us-east-1:80214830479:stack/wildrydes-dev-datalake-structure/26e8b920-91b8-11ee-98db-12cb37a9d445

Description: Creates the S3 and DynamoDB structure of the Data Lake.

Status: CREATE_COMPLETE

Root stack: -

Parameters (10)

Key Value Resolved value

DataCatalogTableName dataCatalog -

DataSourceTableName dataSources -

EnvironmentPrefix wildrydes-dev- -

KMSKeyARN - -

S3CuratedName curated -

Resources (12)

Logical ID Physical ID Type Status

DataCatalogTable wildrydes-dev-dataCatalog AWS::DynamoDB::Table CR

DataCurationCodePackages wildrydes-dev-datacurationcodepackages AWS::S3::Bucket CR

DataSourceTable wildrydes-dev-dataSources AWS::DynamoDB::Table CR

S3CacheTable wildrydes-dev-s3fileprocessingCache AWS::DynamoDB::Table CR

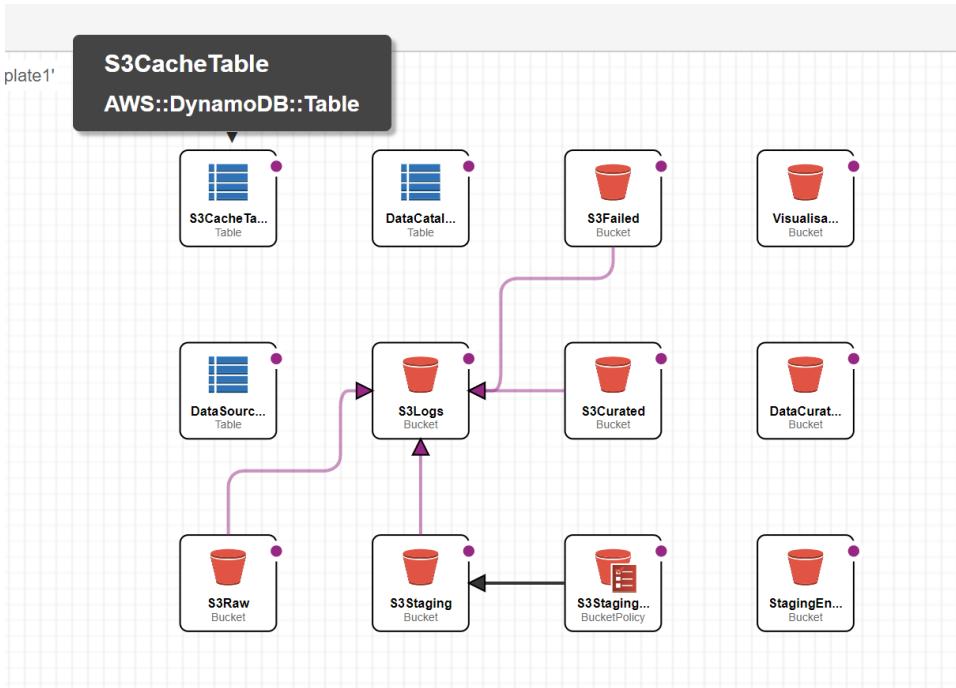
wildrydes-dev-datalake-structure

Events (38)

Detect root cause

Search events

Timestamp	Logical ID	Status	...
2023-12-03 19:45:15 UTC+1100	wildrydes-dev-datalake-structure	CREATE_COMPLETE	-
2023-12-03 19:45:12 UTC+1100	S3StagingBucketPolicy	CREATE_COMPLETE	-
2023-12-03 19:45:12	...	CREATE_IN_PROGRESS	Resource



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

The screenshot displays two main sections of the AWS CloudFormation interface and one section of the Amazon OpenSearch Service interface.

CloudFormation (Top Left): Shows the 'Stacks' list with one stack named 'wildrydes-dev-datalake-elasticsearch'. The stack status is 'CREATE_COMPLETE' and it was created on '2023-12-03 21:07:09 UTC+1100'. A 'Stack info' tab is open, showing the Stack ID: 'arn:aws:cloudformation:us-east-1:802148830479:stack/wildrydes-dev-datalake-elasticsearch/b89978e0-91c3-11ee-8c6e-0e36ab078769' and Status: 'CREATE_COMPLETE'.

CloudFormation (Middle Left): Shows the same 'Stacks' list and stack details, but with a different view of the 'Parameters' tab, which lists 10 parameters including 'AllowExplicitIndex', 'DedicatedMasterEnabled', 'ElasticSearchDiskSpace', and 'ElasticSearchInstanceCount'.

CloudFormation (Bottom Left): Shows the 'Stacks' list and stack details, but with a different view of the 'Events' tab, which lists 8 events related to the stack's creation, including 'CREATE_COMPLETE' and 'CREATE_IN_PROGRESS' events for the ElasticsearchCloudWatch Dashboard.

Amazon OpenSearch Service (Bottom Right): Shows the 'Domains' list with one domain named 'wildrydes...'. The domain status is 'Active' and its engine is 'Elasticsea...'. The cluster health is 'Green'.



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

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

Resources

```
Command Prompt
CloudFormation events from stack operations (refresh every 5.0 seconds)

ResourceStatus ResourceType LogicalResourceId ResourceStatusReason
CREATE_IN_PROGRESS AWS::CloudFormation::Stack wildrydes-dev-datalake-elasticsearch-lambdas User Initiated
CREATE_IN_PROGRESS AWS::IAM::Role LambdaExecutionRole -
CREATE_IN_PROGRESS AWS::IAM::Role LambdaExecutionRole Resource creation Initiated
CREATE_COMPLETE AWS::Lambda::Function SendDataCatalogUpdateToElasticsearch -
CREATE_IN_PROGRESS AWS::Lambda::Function ElasticsearchSendDataCatalogUpdateToElasticsearch -
CREATE_COMPLETE AWS::Lambda::Function SendDataCatalogUpdateToElasticsearch -
CREATE_IN_PROGRESS AWS::Lambda::EventSourceMap DataTableStream DataTableStream Resource creation Initiated
CREATE_IN_PROGRESS AWS::Lambda::EventSourceMap DataTableStream -
CREATE_COMPLETE AWS::Lambda::EventSourceMap DataTableStream -
CREATE_COMPLETE AWS::CloudFormation::Stack wildrydes-dev-datalake-elasticsearch-lambdas -
Successfully created/updated stack - wildrydes-dev-datalake-elasticsearch-lambdas in None
```

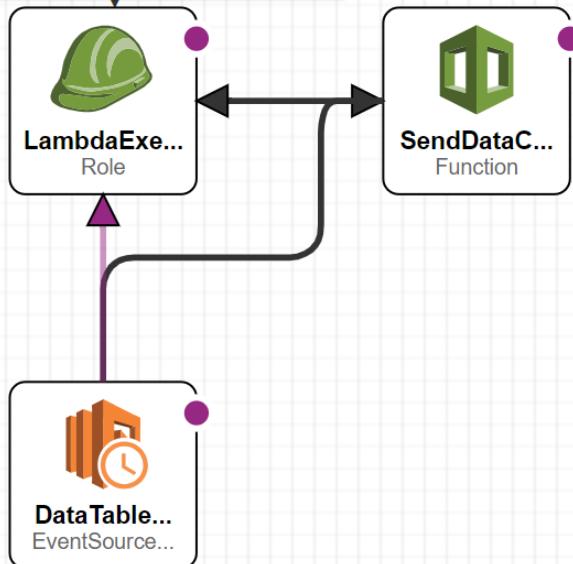
Timestamp	Logical ID	Status
2023-12-03 21:24:22 UTC+1100	wildrydes-dev-datalake-elasticsearch	CREATE_COMPLETE
2023-12-03 21:24:20 UTC+1100	ElasticsearchCloudWatch Dashboard	CREATE_COMPLETE
2023-12-03 21:24:20 UTC+1100	ElasticsearchCloudWatch Dashboard	CREATE_IN_PROGRESS

Function name	Description	Package type	Runtime
wildrydes-dev-datalake-elasticsearch-SendDataCatalogUpdateToElasticsearch-riOjzy8ir2du	Sends changes in the data catalog to elasticsearch	Zip	Python 3.9
wildrydes-dev-datalake-staging-engine-GetFileSettings-a0GOVK8MugHt	Load the settings for the new file's file type (data source)	Zip	Python 3.9
wildrydes-dev-datalake-staging-CopyFileFromRawToStaging-IVBTzUjeQr7UF	Copy the new file, and its tags and metadata to the staging bucket.	Zip	Python 3.9
wildrydes-dev-datalake-staging-StartFileProcessing-QDwnQ7HzCTAc	Initiates File Processing Step Function. This is triggered when new file put into RAW bucket.	Zip	Python 3.9
wildrydes-dev-datalake-staging-RecordFailedStaging-hn1RlwNz2coyf	Records failed staging in the data lake data catalog, and sends failure SNS if configured.	Zip	Python 3.9
wildrydes-dev-datalake-staging-CopyFileFromRawToFailed-wwVU8hyOSajQ	Copy files that have failed ingress from the raw to failed bucket.	Zip	Python 3.9
wildrydes-dev-datalake-staging-VerifyFileSchema-AVxQ9roE3IVEW	Verify the schema of the file (if configured).	Zip	Python 3.9
wildrydes-dev-datalake-staging-RecordSuccessfulStaging-VBOM1hs0RPiP	Records successful staging in the data lake data catalog, and sends success SNS if configured.	Zip	Python 3.9
wildrydes-dev-datalake-staging-engine-GetFileType-ICVnV9LyQaQ	Retrieves the matching file type (data source) for the new file.	Zip	Python 3.9
wildrydes-dev-datalake-staging-CalculateMetaDataForFile-jybjiOFSUdYY	Attach the required tags and metadata to the new file.	Zip	Python 3.9
wildrydes-dev-datalake-staging-engine-DeleteRawFile-GH35o3MOAkHb	Deletes the raw file after successful or failed staging.	Zip	Python 3.9

- DataTableStream
- LambdaExecutionRole
- SendDataCatalogUpdateToElasticsearch

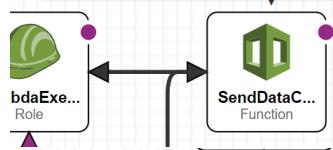
LambdaExecutionRole

AWS::IAM::Role



SendDataCatalogUpdateToElasticsearch

AWS::Serverless::Function



DataTableStream

AWS::Lambda::EventSourceMapping



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-bucket wilddrydes-dev-datalake-staging-engine  
sam deploy --template-file stagingEngineDeploy.yaml --stack-name wilddrydes-dev-datalake-staging-engine --capabilities CAPABILITY_IAM
```

[wilddrydes-dev-datalake- staging-engine \(CloudFormation\)](#)

Resources

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

```
C:\Users\rohit\Downloads\accelerated-data-lake-master\StagingEngine>sam package --template-file ./stagingEngine.yaml
output-template-file stagingEngineDeploy.yaml --s3-bucket wildrydes-dev-stagingenginecodepackages
Uploading to c2c746de4475cbba09e7955ff0d82d4 2430 / 2430 (100.00%)
Uploading to e22a256a6546a346b3299d765436d246 1963 / 1963 (100.00%)
Uploading to e69b14b90f13f2217e913fb90c01500d 1175 / 1175 (100.00%)
Uploading to d8f6db6adcbb98ebe82ce901fe1feef8c 105238 / 105238 (100.00%)
Uploading to 740c13b275fd437021967e5fe84a1b57 1121 / 1121 (100.00%)
Uploading to 8230e5c1530123600bcada722932c93a 1931 / 1931 (100.00%)
Uploading to 2984cd61284eefe16ade2bada54e8ea4 657 / 657 (100.00%)
Uploading to 93818b71fa0lef39dec70ebc5f654197 1367 / 1367 (100.00%)
Uploading to 0bd59e94d249580630bed97ccfd9dcfe 758 / 758 (100.00%)
Uploading to b9b4a82873d0f81831e8bbd02d92f663 1536 / 1536 (100.00%)

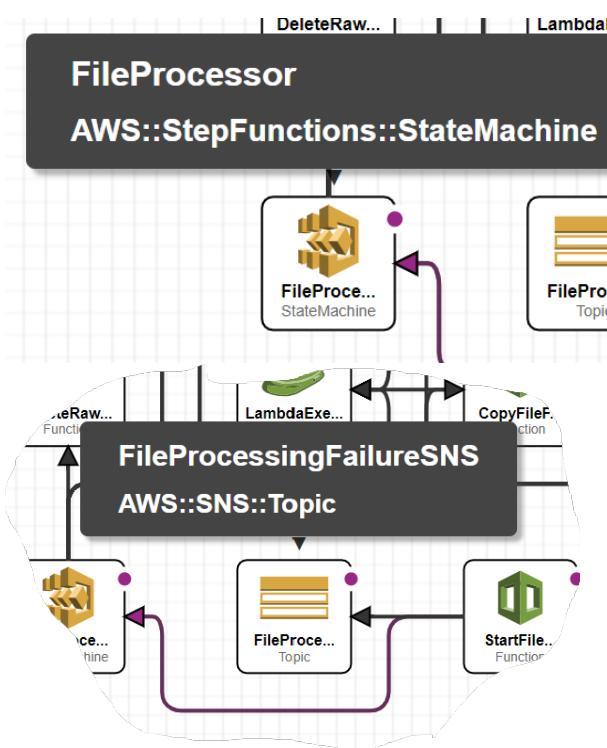
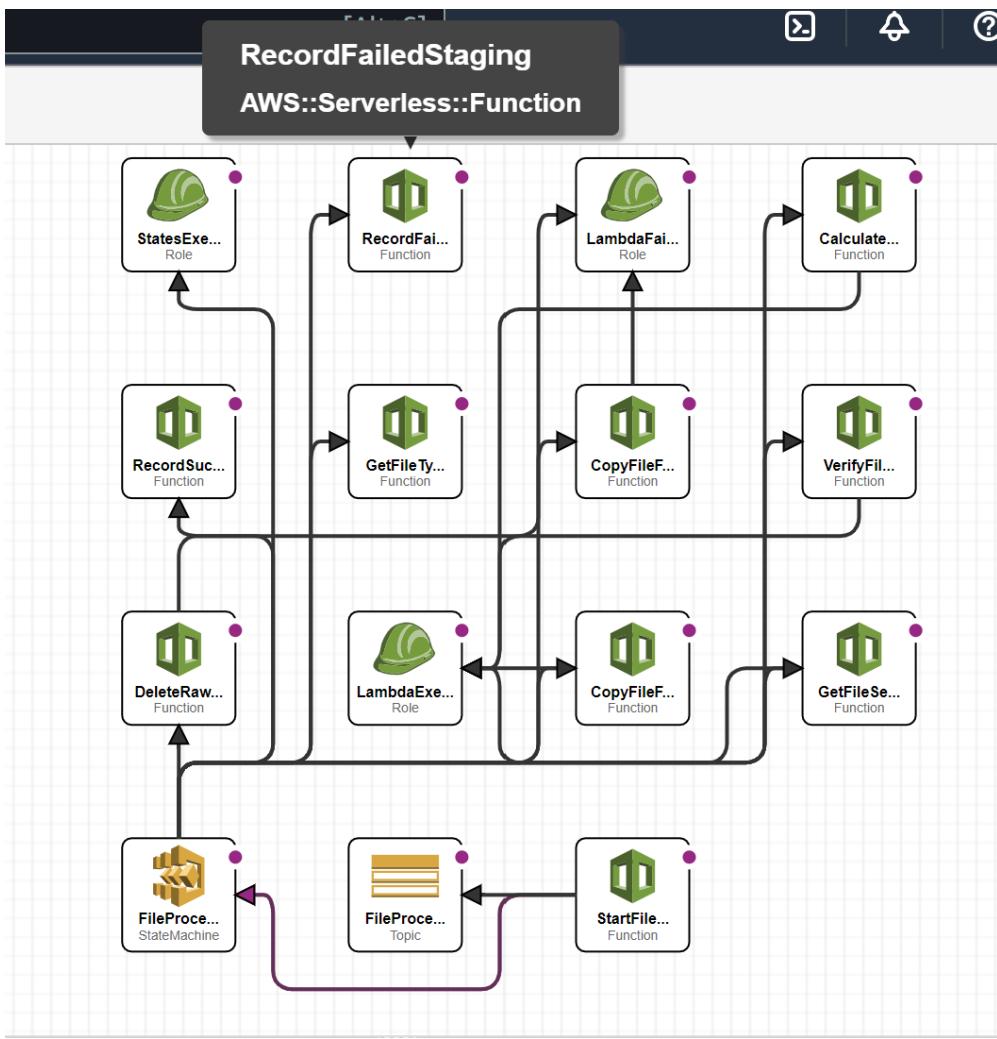
Successfully packaged artifacts and wrote output template to file stagingEngineDeploy.yaml.
Execute the following command to deploy the packaged template
sam deploy --template-file C:\Users\rohit\Downloads\accelerated-data-lake-master\StagingEngine\stagingEngineDeploy.yaml --stack-name <YOUR STACK NAME>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

```
Command Prompt
CREATE_IN_PROGRESS AWS::Lambda::Function RecordFailedStaging - Resource creation
CREATE_IN_PROGRESS AWS::Lambda::Function RecordFailedStaging - 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::StateMachine AWS::StepFunctions::StateMachine FileProcessor - -
CREATE_IN_PROGRESS AWS::StepFunctions::StateMachine AWS::StepFunctions::StateMachine FileProcessor - Resource creation
CREATE_COMPLETE AWS::StepFunctions::StateMachine AWS::StepFunctions::StateMachine FileProcessor - Initiated
CREATE_IN_PROGRESS AWS::Lambda::Function StartFileProcessing - -
CREATE_IN_PROGRESS AWS::Lambda::Function StartFileProcessing - Resource creation
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

Name	Type	Creation date
wildrydes-dev-stagingengine	Standard	Dec 3, 2023, 21:49:28 (UTC+11:00)

Timestamp	Logical ID	Status
2023-12-03 21:49:39 UTC+1100	wildrydes-dev-datalake-staging-engine	CREATE_COMPLETE
2023-12-03 21:49:38 UTC+1100	StartFileProcessing	CREATE_COMPLETE
2023-12-03 21:49:32 UTC+1100	StartFileProcessing	CREATE_IN_PROGRESS
2023-12-03 21:49:31 UTC+1100		CREATE_IN_PROGRESS



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.

AWS Lambda

Functions (1/11)

Function name	Description	Package type	Runtime	Last modified
wildrydes-dev-datalake-staging-StartFileProcessing-QDwnQZHzCTAc	Initiates File Processing Step Function. This is triggered when new file put into RAW bucket.	Zip	Python 3.9	12 minutes ago

Lambda > Add trigger

Add trigger

Trigger configuration [Info](#)

S3 aws asynchronous storage

Bucket
Please select the S3 bucket that serves as the event source. The bucket must be in the same region as the function.

s3/wildrydes-dev-raw [X](#) [C](#)

Bucket region: us-east-1

Event types
Select the events that you want to have trigger the Lambda function. You can optionally set up a prefix or suffix for an event. However, for each bucket, individual events cannot have multiple configurations with overlapping prefixes or suffixes that could match the same object key.

PUT [X](#)

Prefix - optional

wildrydes-dev-datalake-staging-StartFileProcessing-QDwnQZHzCTAc

[Throttle](#) [Copy ARN](#) [Actions ▾](#)

The trigger wildrydes-dev-raw was successfully added to function wildrydes-dev-datalake-staging-StartFileProcessing-QDwnQZHzCTAc. The function will receive events from the trigger.

Function overview [Info](#) [Export to Application Composer](#)

[Diagram](#) [Template](#)

wildrydes-dev-datalake-staging-StartFileProcessing-QDwnQZHzCTAc

Related functions: [Select a function](#)

Description: Initiates File Processing Step Function. This is triggered when new file put into RAW bucket.

Last modified: 17 minutes ago

Function ARN: arn:aws:lambda:us-east-1:123456789012:function:wildrydes-dev-datalake-staging-StartFileProcessing-QDwnQZHzCTAc

Application: wildrydes-dev-datalake

[+ Add destination](#)

[S3](#) [+ Add trigger](#)

Paramaters

- DataCatalogTableName
- DataSourceTableName

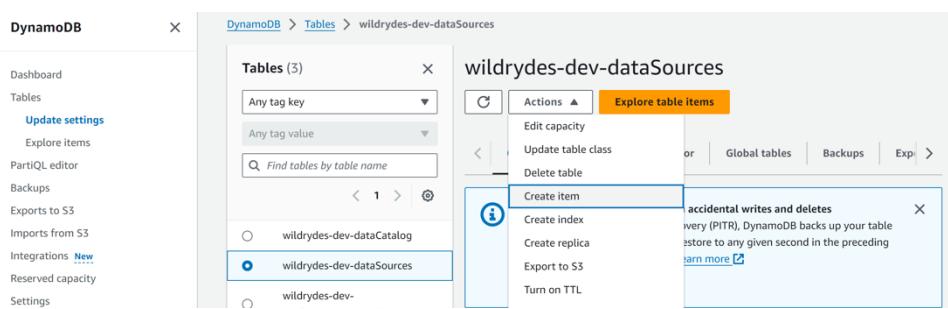
4. Configure a sample data source and add data

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.



DynamoDB > Tables > [wildrydes-dev-dataSources](#) > Create item

Create item

You can add, remove, or edit the attributes of an item. You can nest attributes up to 32 levels deep. [Learn more](#)

Attributes [View DynamoDB JSON](#)

```

1▼ {
2▼   "fileSettings": {
3    "fileFormat": "json",
4    "fileNamePattern": "rydebookings/rydebooking-[0-9]{10}.json",
5    "calculateMD5": "True",
6    "stagingFolderPath": "rydebookings",
7▼   "stagingPartitionSettings": {
8     "expression": "year=%Y/month=%m/day=%d",
9     "timezone": "Australia/Brisbane"
10   }
11 },
12   "fileType": "RydeBooking",
13   "metadata": {
14     "creator": "Unice the Unicorn",
15     "quality": "HIGH",
16     "sourcesystem": "3x3x3 DataLake Demonstration"
17   },
18   "schema": {
19     "properties": {
20       "bookingDetails": {
21         "type": "object"
22       },
23       "totalPassengers": {
24         "type": "number"
25       },
26       "required": [
27         "totalPassengers",
28         "bookingDetails"
29       ],
30       "type": "object"
31     },
32   },
33   "tags": {
34     "dataOwner": "WildRydes",
35     "dataSource": "RydeBookings",
36     "pii": "FALSE"
37   }
38 }
```

CloudShell Feedback © 2023, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookies

JSON Ln 3, Col 19 [Errors: 1](#) [Warnings: 0](#)

Ln 3, Col 19 Invalid DynamoDB type, "fileFormat"

GOT ERROR

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

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-200000000.json` into this folder.
- Confirm the file has appeared in the failed folder, with a path similar to: `wildrydes-dev-failed/rydebookings/rydebooking-200000000.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.

kibana

Add Data to Kibana

Use these solutions to quickly turn your data into pre-built dashboards and monitoring systems.

- APM**: APM automatically collects in-depth performance metrics and errors from inside your applications. [Add APM](#)
- Logging**: Ingest logs from popular data sources and easily visualize in preconfigured dashboards. [Add log data](#)
- Metrics**: Collect metrics from the operating system and services running on your servers. [Add metric data](#)
- Security Analytics**: Centralize security events for interactive investigation in ready-to-go visualizations. [Add security events](#)

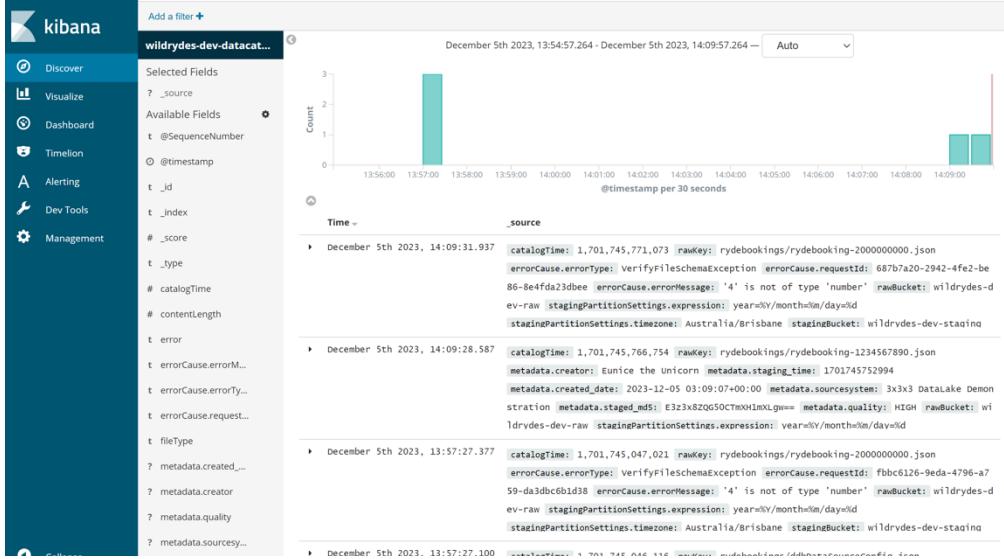
Data already in Elasticsearch? [Set up index patterns](#)

Visualize and Explore Data

- Dashboard**: Display and share a collection of visualizations and saved searches.
- Dev Tools**: Use an expression language.
- Management**

Manage and Administer the Elastic Stack

- Console**: Skip cURL and use this JSON interface to work with your data directly.
- Index Patterns**: Manage the index patterns that help retrieve your data from Elasticsearch.
- Saved Objects**: Import, export, and manage



XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

S3 Bucket

The screenshot shows the Amazon S3 console. On the left, there's a navigation sidebar with sections like Buckets, Access Grants, Access Points, Object Lambda Access Points, Multi-Region Access Points, Batch Operations, IAM Access Analyzer for S3, Block Public Access settings, Storage Lens, Feature spotlight, and AWS Marketplace for S3. The main area is titled "General purpose buckets (9) info". It shows a table of buckets with columns: Name, AWS Region, Access, and Creation date. The buckets listed are:

Name	AWS Region	Access	Creation date
cf-templates-1ra2aml56wmkz-us-east-1	US East (N. Virginia) us-east-1	Bucket and objects not public	December 3, 2023, 19:41:46 (UTC+11:00)
wildrydes-dev-curated	US East (N. Virginia) us-east-1	Bucket and objects not public	December 3, 2023, 19:44:49 (UTC+11:00)
wildrydes-dev-datacurationcodepackages	US East (N. Virginia) us-east-1	Bucket and objects not public	December 3, 2023, 19:44:26 (UTC+11:00)
wildrydes-dev-failed	US East (N. Virginia) us-east-1	Bucket and objects not public	December 3, 2023, 19:44:49 (UTC+11:00)
wildrydes-dev-logs	US East (N. Virginia) us-east-1	Bucket and objects not public	December 3, 2023, 19:44:26 (UTC+11:00)
wildrydes-dev-raw	US East (N. Virginia) us-east-1	Bucket and objects not public	December 3, 2023, 19:44:49 (UTC+11:00)
wildrydes-dev-staging	US East (N. Virginia) us-east-1	Bucket and objects not public	December 3, 2023, 19:44:49 (UTC+11:00)
wildrydes-dev-stagingengineeringcodepackages	US East (N. Virginia) us-east-1	Bucket and objects not public	December 3, 2023, 19:44:26 (UTC+11:00)
wildrydes-dev-visualisationcodepackages	US East (N. Virginia) us-east-1	Bucket and objects not public	December 3, 2023, 19:44:25 (UTC+11:00)

STEP FUNCTION

Step Functions

Step Functions > State machines

State machines (1)

Execution counts are based on the most recent 1000 executions

Create state machine

Search for state machines

Name **Type** **Creation date** **Status**

wildrydes-dev-stagingengine Standard Dec 3, 2023, 21:49:28 (UTC+11:00) Active

Definition

Actions ▾

```
1 {  
2     "Comment": "State machine to manage the staging of new  
3         files added to the data lake",  
4         Formatted <>  
5             "StartAt": "GetFileType",  
6             "States": {  
7                 "GetFileType": {  
8                     "Type": "Task",  
9                     "Resource": "arn:aws:lambda:us-east-  
10                    1:802148830479:function:wildrydes-dev-datalake-staging-  
11                    engine-GetFileType-ICtVqV9L0yaQ",  
12                     "Comment": "Retrieves the matching file type (data  
13                     source) for the new file.",  
14                     "Next": "GetFileSettings",  
15                     "Catch": [  
16                         {  
17                             "ErrorEquals": [  
18                                 "GetFileTypeException",  
19                                 "Exception"  
20                             ],  
21                             "ResultPath": "$.error-info",  
22                             "Next": "CopyFileFromRawToFailed"  
23                         }  
24                     ],  
25                     "Retry": [  
26                         {  
27                         }  
28                     ]  
29                 }  
30             }  
31         }  
32     }  
33 }
```

The state machine diagram illustrates the workflow for managing new files added to the data lake. It begins with a Start state, followed by a GetFileType state. This is followed by a GetFileSettings state, which then leads to VerifyFileSchema. The VerifyFileSchema state branches into two paths: one leading to CalculateMetaDataForFile and another leading to CopyFileFromRawToStaging. The CalculateMetaDataForFile path leads to WaitForRawBucketReadsToComplete, DeleteRawFileAfterSuccessfulStaging, and RecordSuccessfulStaging. The CopyFileFromRawToStaging path leads to WaitForRawBucketReadsToComplete, DeleteRawFileAfterSuccessfulStaging, and RecordSuccessfulStaging. Both paths converge at a CopyFileFromRawToFailed state, which then leads to RecordFailedStaging, FinishedProcessingUnsuccessfulFile, and Finally to an End state.

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

LAMBDAS

Functions (11)						
	Function name	Description	Package type	Runtime	Last modified	
<input type="checkbox"/>	wildrydes-dev-datalake-el-SendDataCatalogUpdateToE-riQJzyBir2du	Sends changes in the data catalog to elasticsearch	Zip	Python 3.9	7 minutes ago	
<input type="checkbox"/>	wildrydes-dev-datalake-staging-eng-GetFileSettings-a06GOVK8MuqHT	Load the settings for the new file's file type (data source)	Zip	Python 3.9	1 minute ago	
<input type="checkbox"/>	wildrydes-dev-datalake-st-CopyFileFromRawToStaging-iYBTzLieOr7Uf	Copy the new file, and its tags and metadata to the staging bucket.	Zip	Python 3.9	1 minute ago	
<input type="checkbox"/>	wildrydes-dev-datalake-staging-StartFileProcessing-ODwnOZhzCTAc	Initiates File Processing Step Function. This is triggered when new file put into RAW bucket.	Zip	Python 3.9	49 seconds ago	
<input type="checkbox"/>	wildrydes-dev-datalake-staging-RecordFailedStaging-hn7RlwNzcoyf	Records failed staging in the data lake data catalog, and sends failure SNS if configured.	Zip	Python 3.9	1 minute ago	
<input type="checkbox"/>	wildrydes-dev-datalake-sta-CopyFileFromRawToFailed-wwVUByOsajD	Copy files that have failed ingress from the raw to failed bucket.	Zip	Python 3.9	1 minute ago	
<input type="checkbox"/>	wildrydes-dev-datalake-staging-en-VerifyFileSchema-AVsOprofE3VEW	Verify the schema of the file (if configured).	Zip	Python 3.9	2 minutes ago	
<input type="checkbox"/>	wildrydes-dev-datalake-sta-RecordSuccessfulStaging-VBOM1hs0RPiP	Records successful staging in the data lake data catalog, and sends success SNS if configured.	Zip	Python 3.9	2 minutes ago	
<input type="checkbox"/>	wildrydes-dev-datalake-staging-engine-GetFileType-ICtvnV9lOyaQ	Retrieves the matching file type (data source) for the new file.	Zip	Python 3.9	2 minutes ago	
<input type="checkbox"/>	wildrydes-dev-datalake-st-CalculateMetaDataForFile-jyBjliOfSludy	Attach the required tags and metadata to the new file.	Zip	Python 3.9	2 minutes ago	
<input type="checkbox"/>	wildrydes-dev-datalake-staging-engin-DeleteRawFile-Gh35o3MOAkHb	Deletes the raw file after successful or failed staging.	Zip	Python 3.9	2 minutes ago	

GetFileType

GetFileSettings

VerifyFileSchema

CalculateMetaDataForFile

CopyFileFromRawToStaging

WaitForRawBucketReadsToComplete

6. DeleteRawFileAfterSuccessfulStatging

7. RecordSuccessfulStaging

8. CopyFileFromRawToFailed

9. DeleteRawFileAfterFailedStaging

10. RecordFailedStaging