

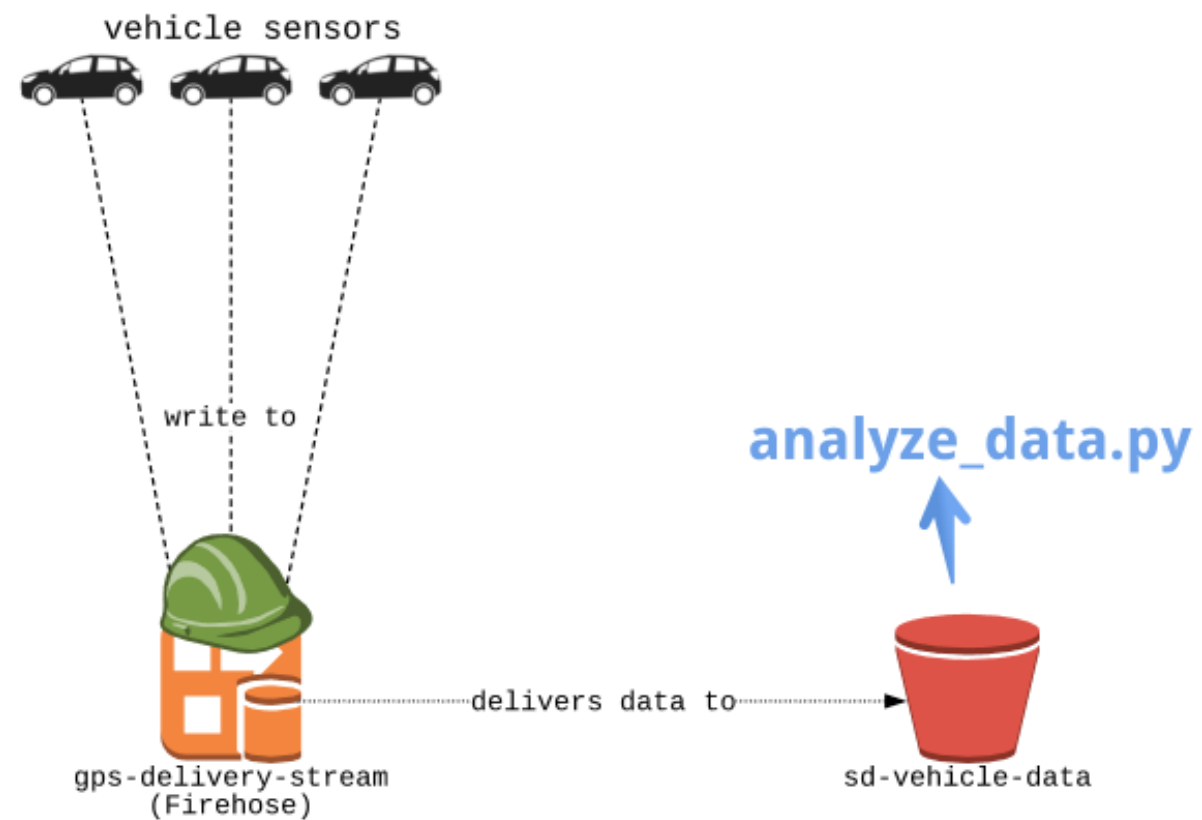
# Going serverless

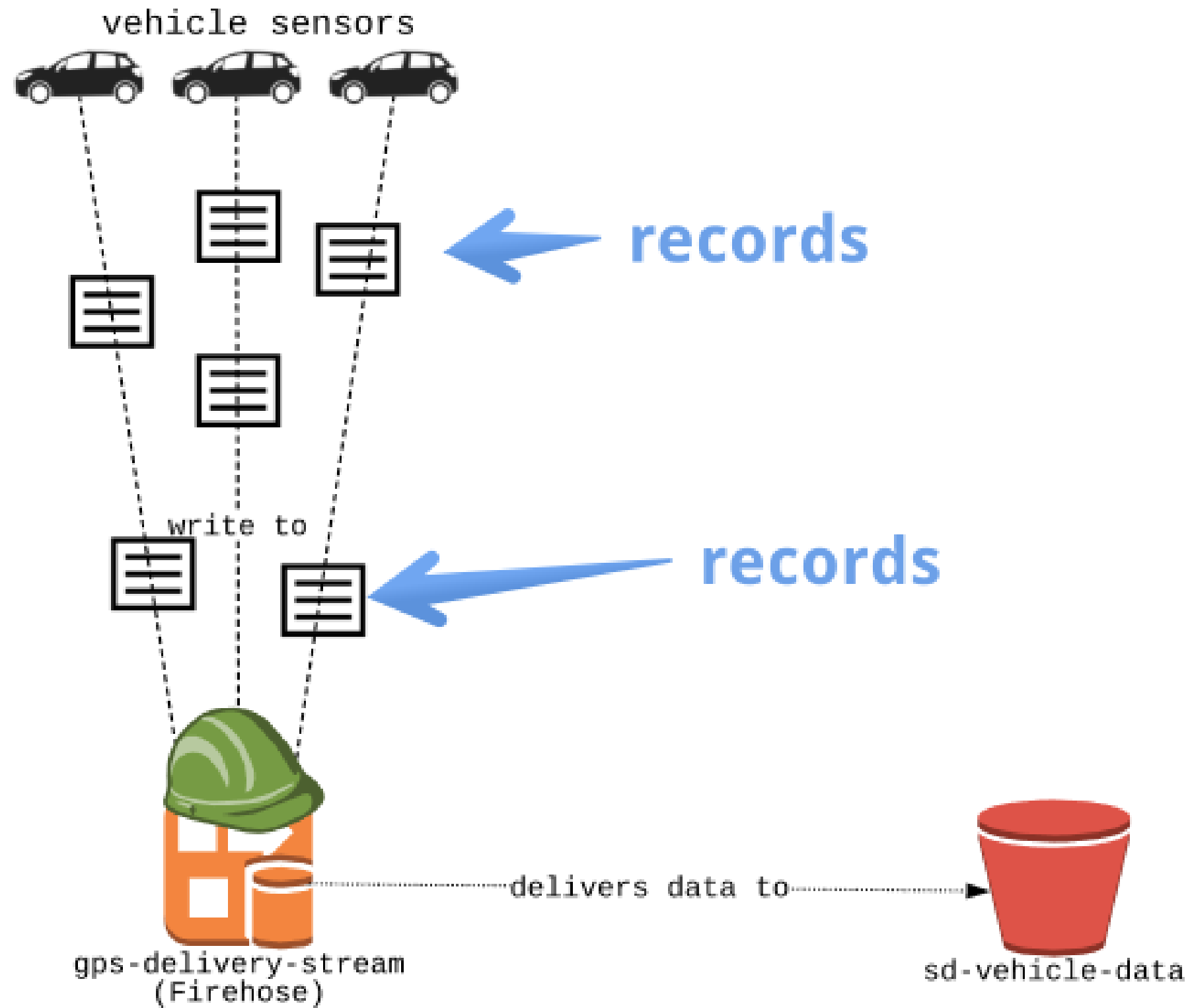
STREAMING DATA WITH AWS KINESIS AND LAMBDA

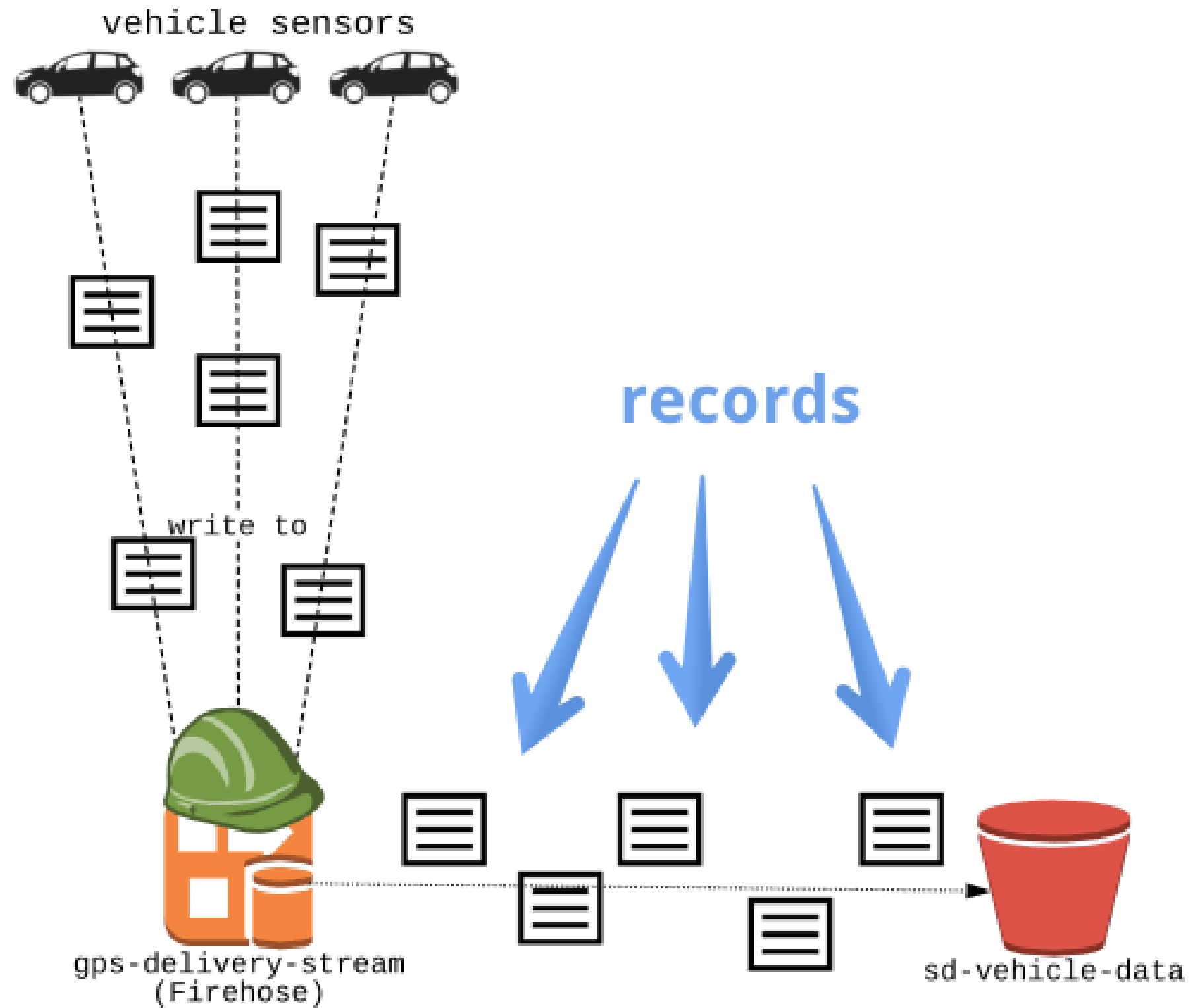


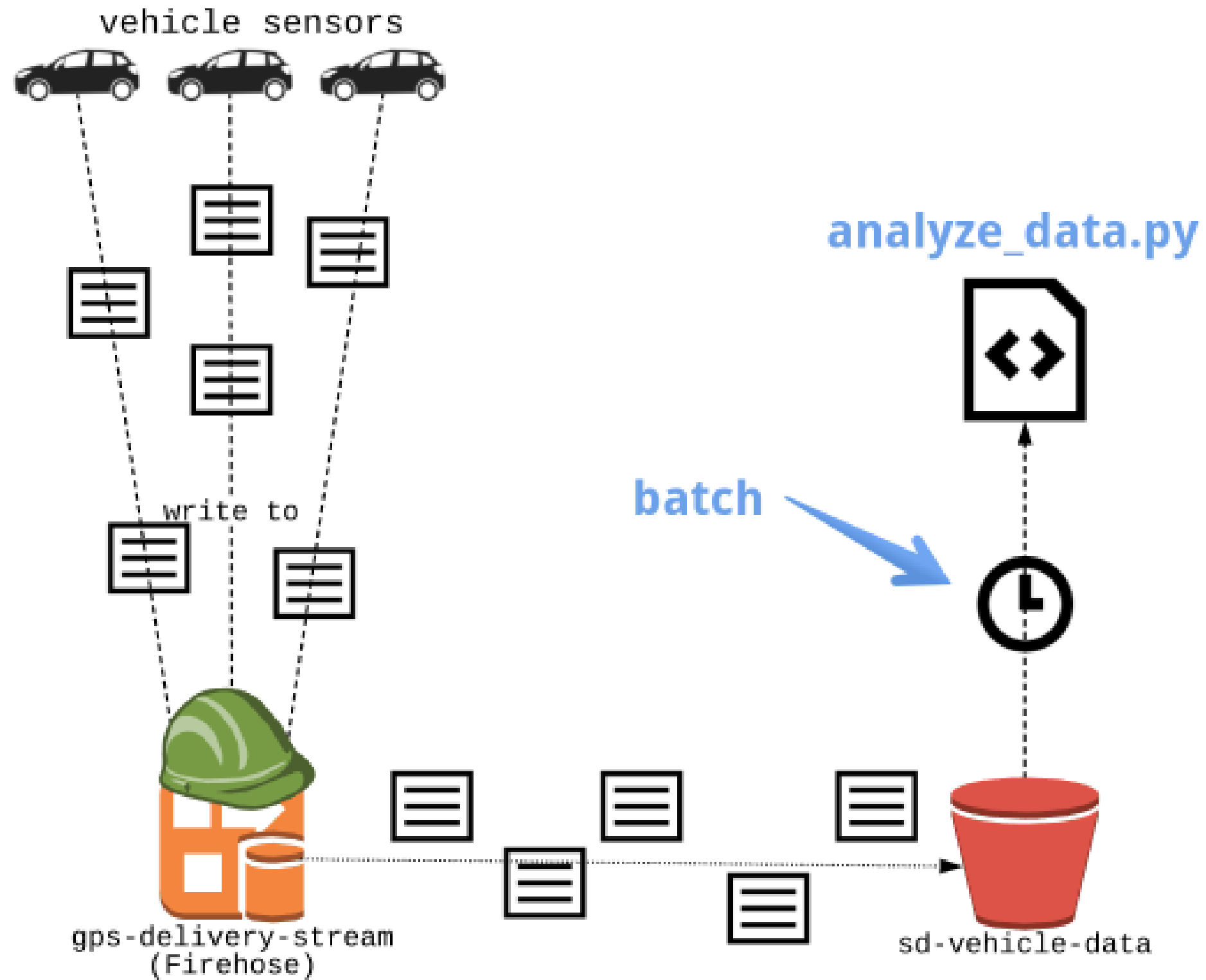
**Maksim Pecherskiy**  
Data Engineer

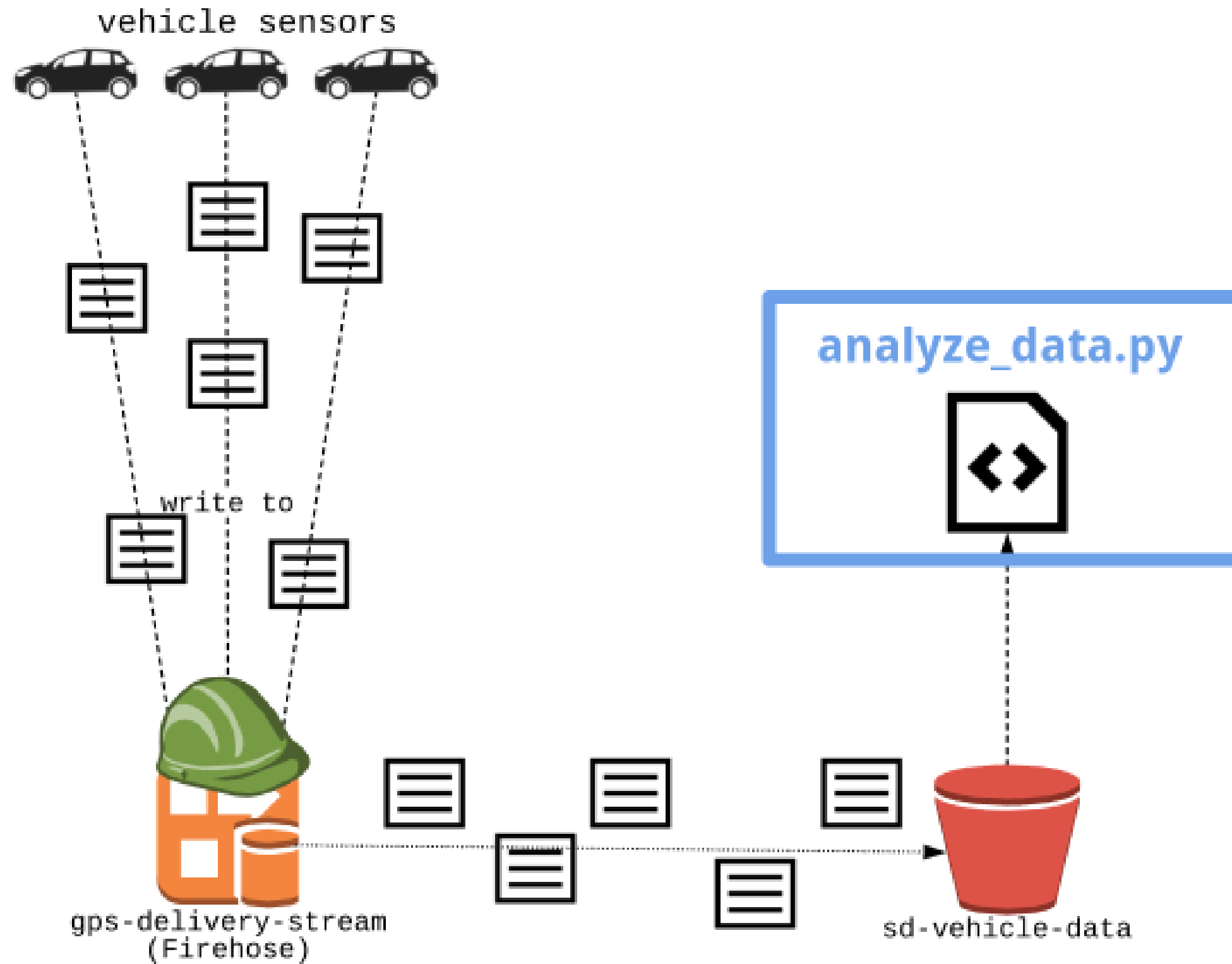
# Last chapter

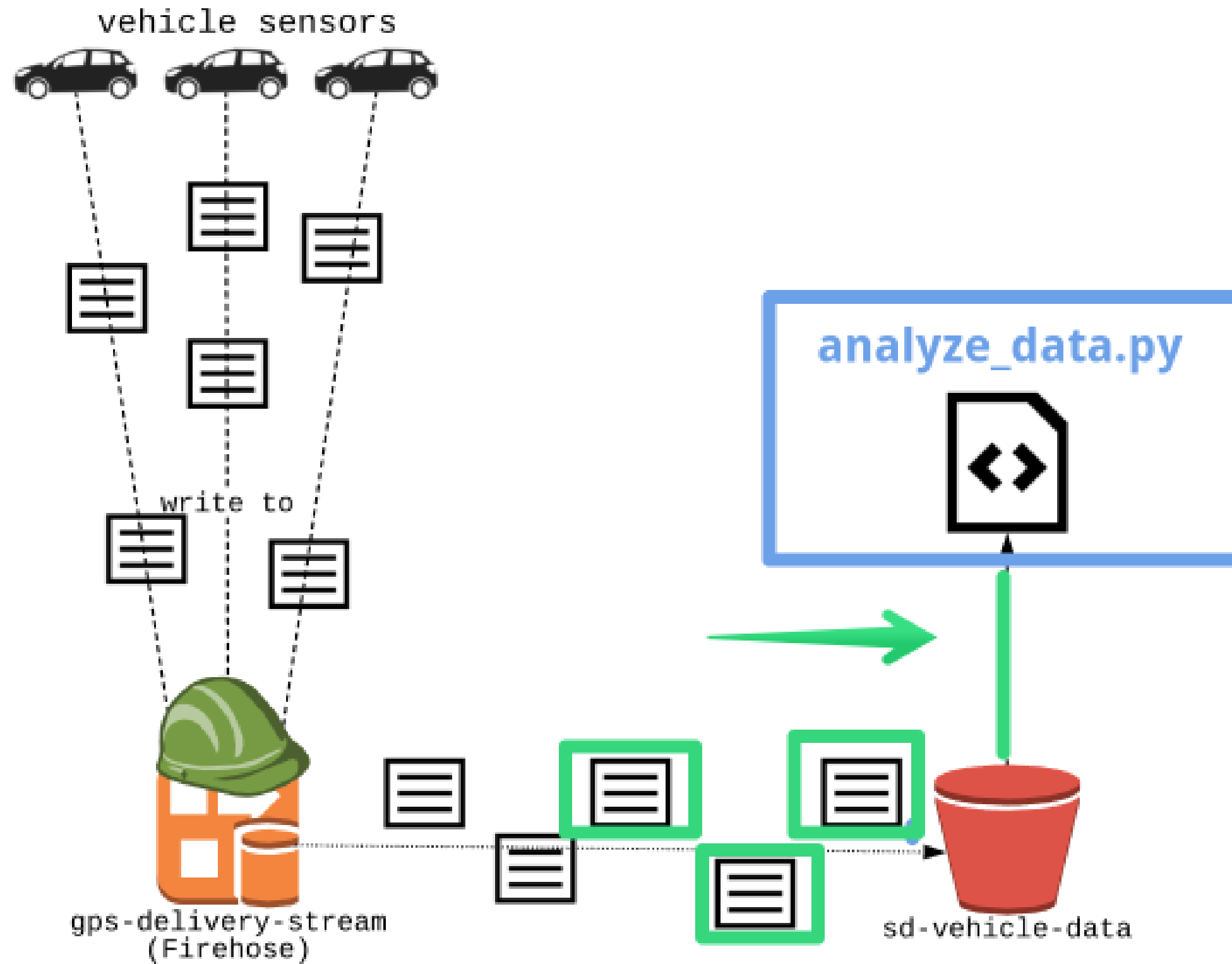












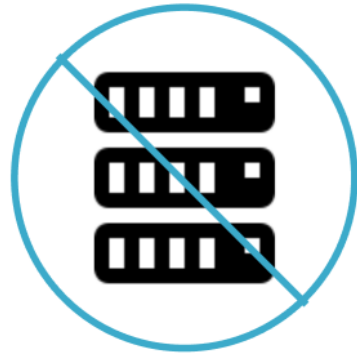
# Lambda



**Execute code.  
In the cloud.  
Triggered by an event.**



# Serverless



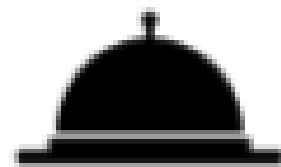
- Servers managed by cloud provider
- Pay per execution
- Automatic scaling
- Lower memory and exec time limits
- Great for quick, targeted functions
- Responding to an event with a callback

# Servers (Traditional)



- Servers managed by user
- Pay per hour of running the machine
- Manual scaling
- Higher memory and exec time limits
- Great for long running code executions
- Training a machine learning model

# 1. Trigger

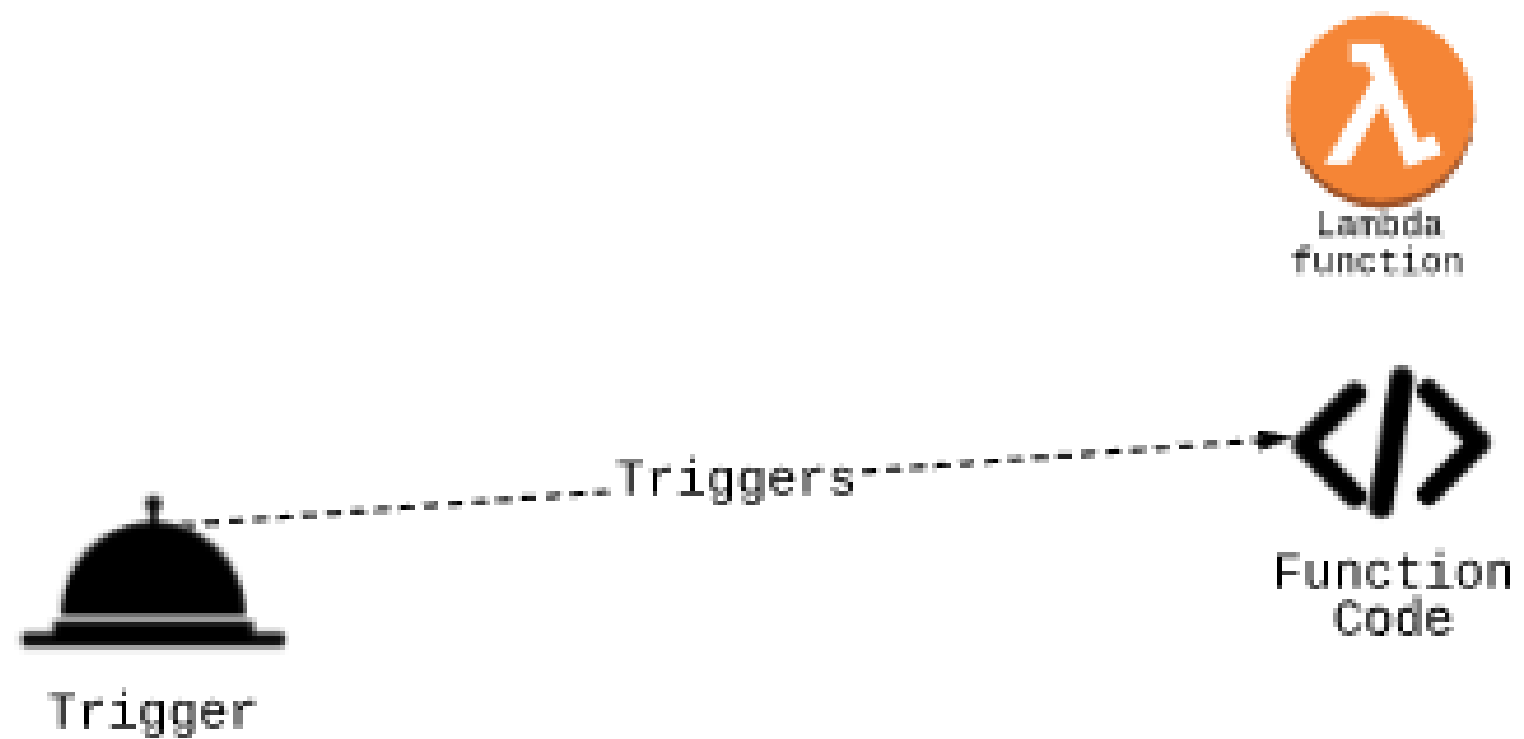


Trigger

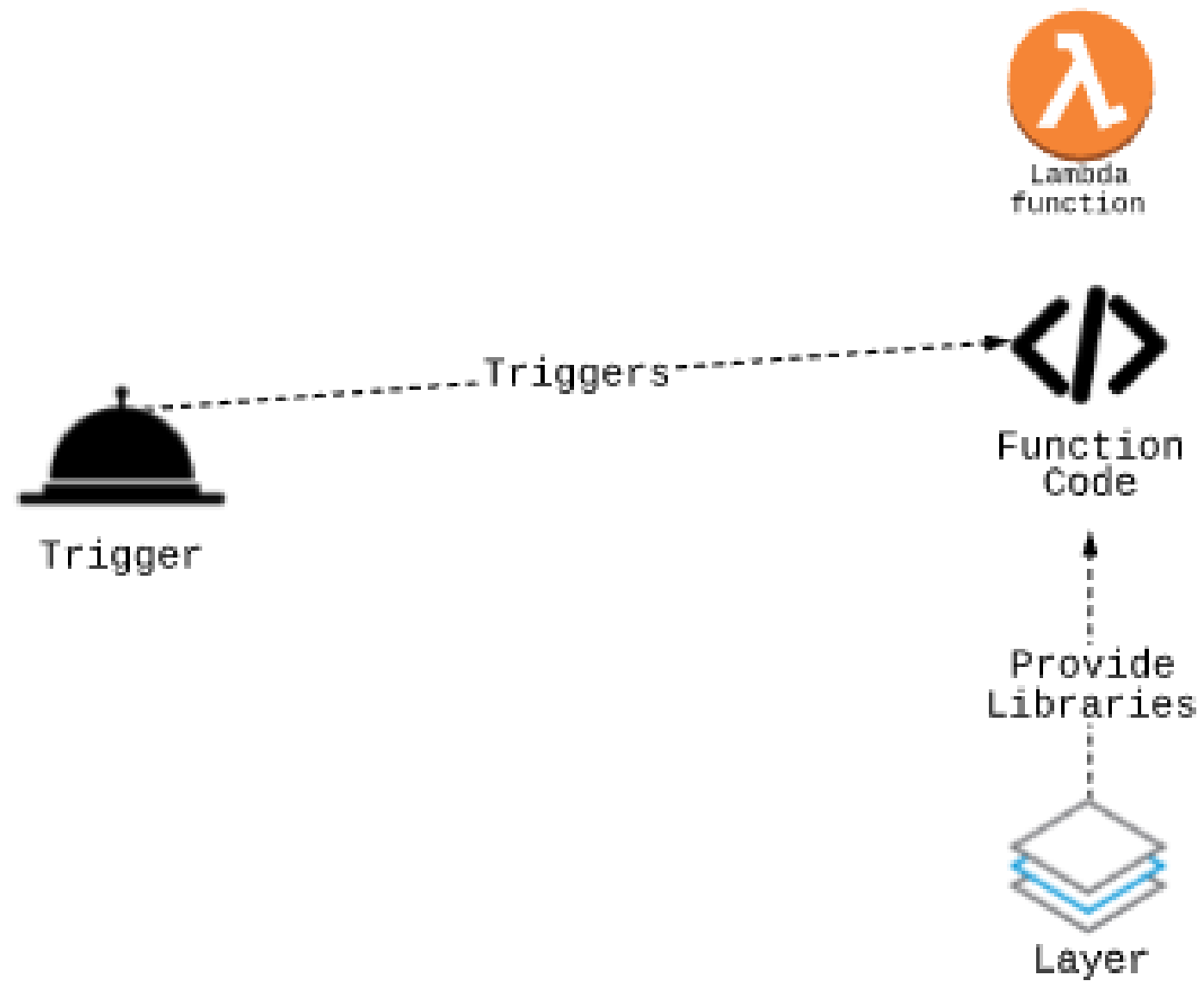


Lambda  
function

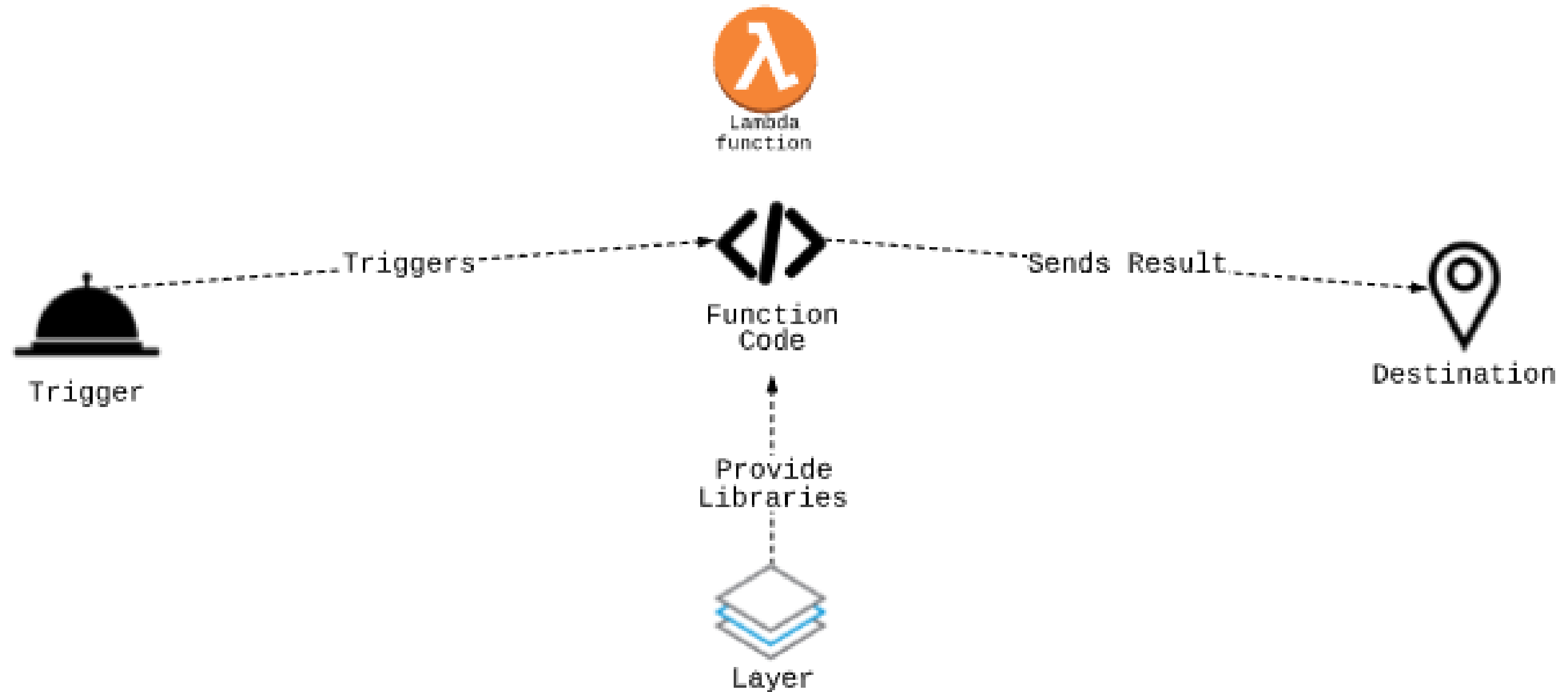
## 2. Handler



# 3. Layer

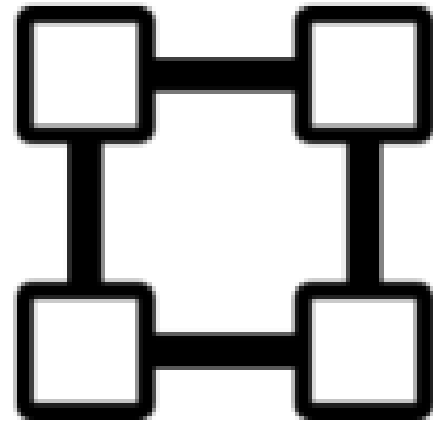


## 4. Destination



# Powerful combination

Data Transformation



Alerting



API



Alexa



# Sample handler

```
import json

def lambda_handler(event, context):
    return {
        'statusCode': 200,
        'body': json.dumps('Hello from Lambda!')
    }
```

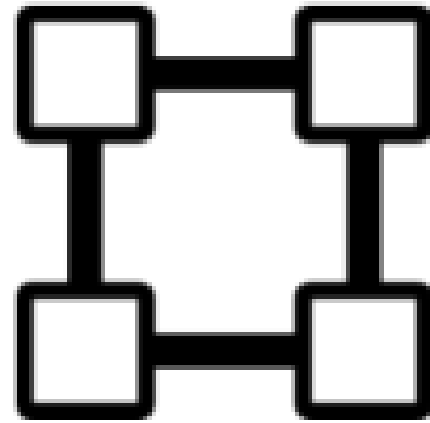
# Review





# Review

Data Transformation



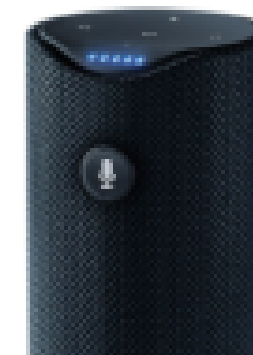
Alerting



API

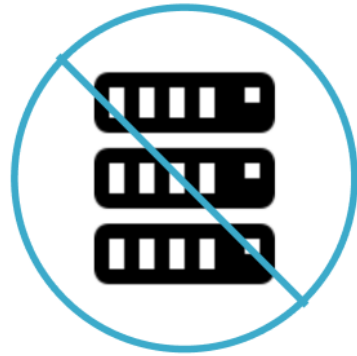


Alexa



# Review

## Serverless



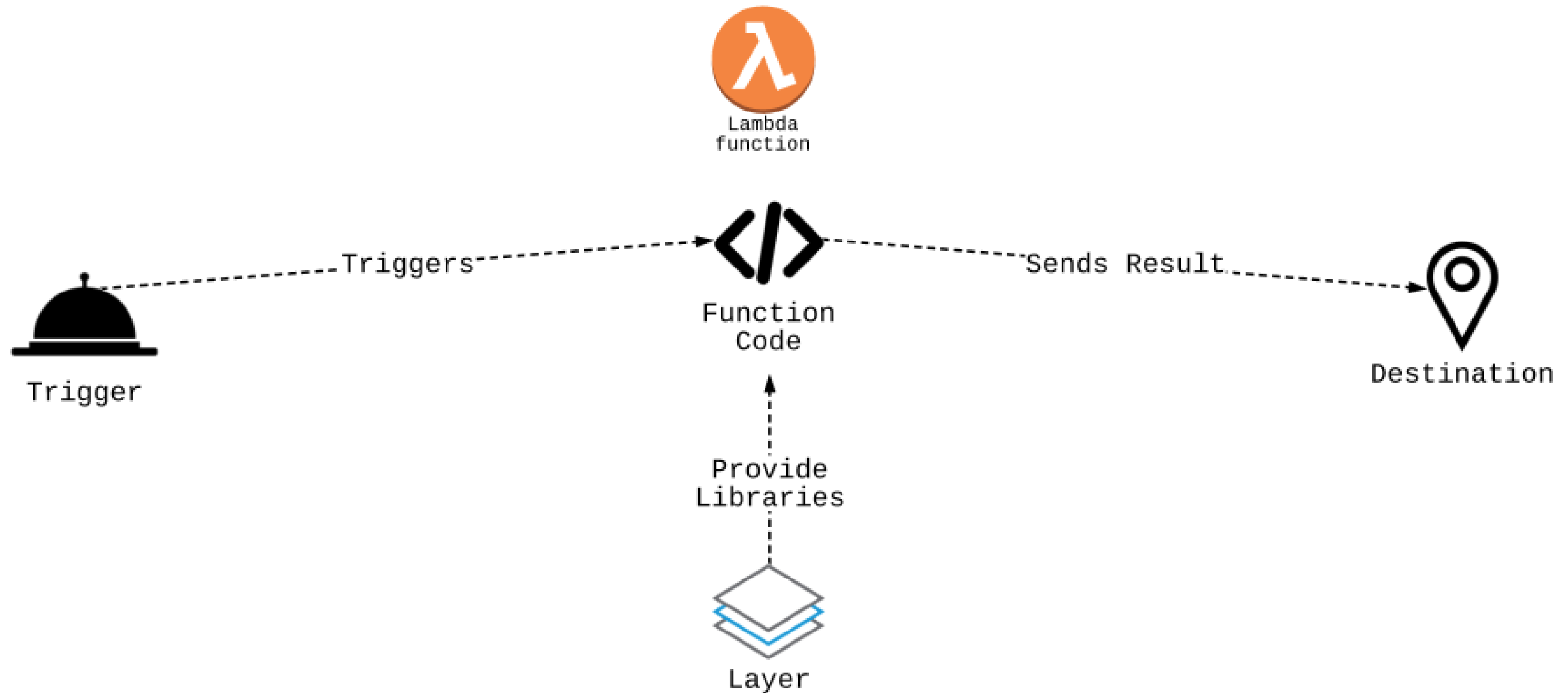
- Responding to an event

## Traditional



- Training an ML model

# Review

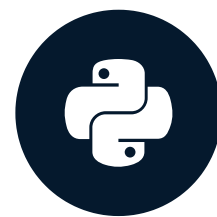


# Let's practice!

STREAMING DATA WITH AWS KINESIS AND LAMBDA

# Creating and running Lambda functions

STREAMING DATA WITH AWS KINESIS AND LAMBDA



**Maksim Pecherskiy**  
Data Engineer

# Let's practice!

STREAMING DATA WITH AWS KINESIS AND LAMBDA

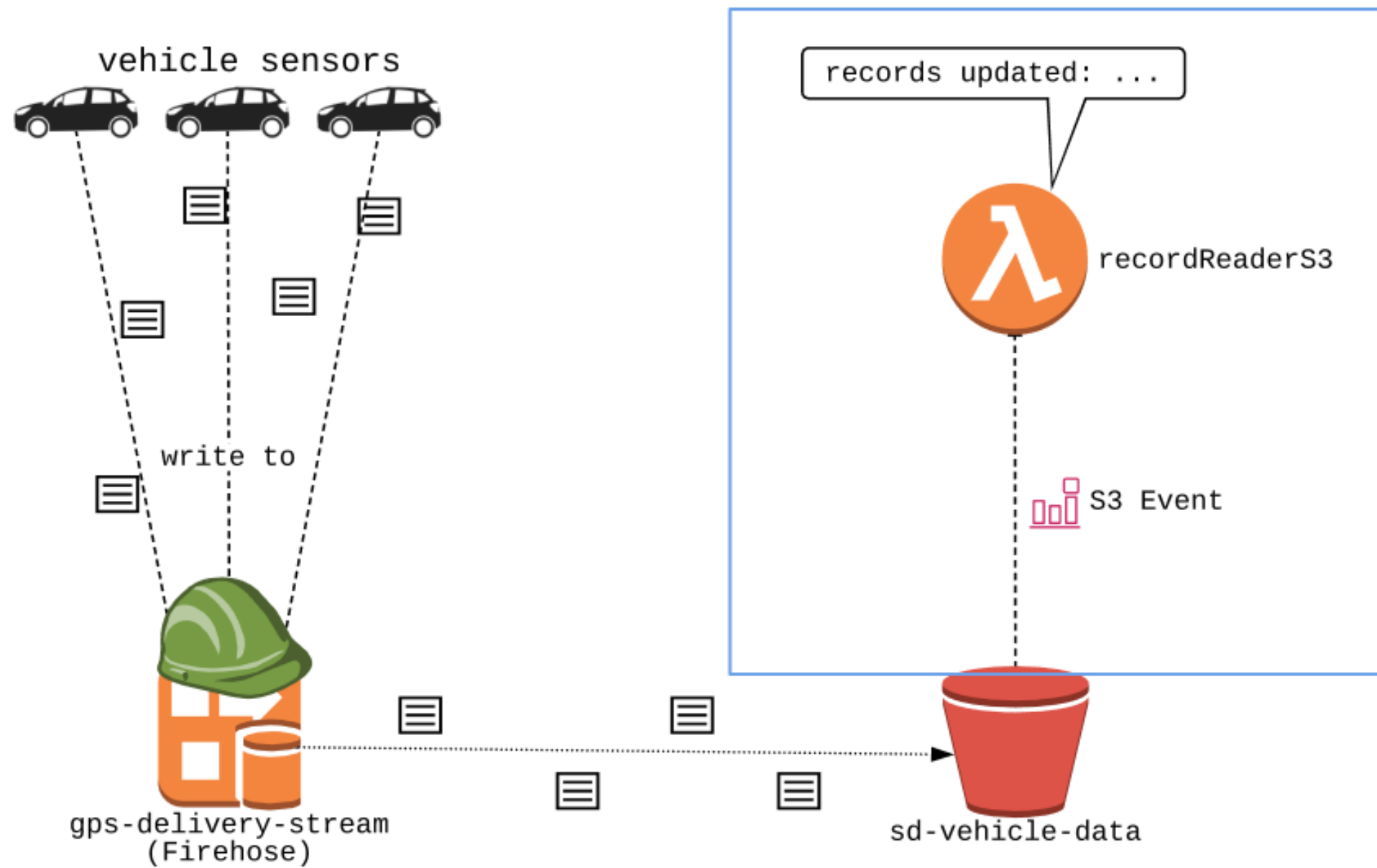
# Your first live lambda!

STREAMING DATA WITH AWS KINESIS AND LAMBDA



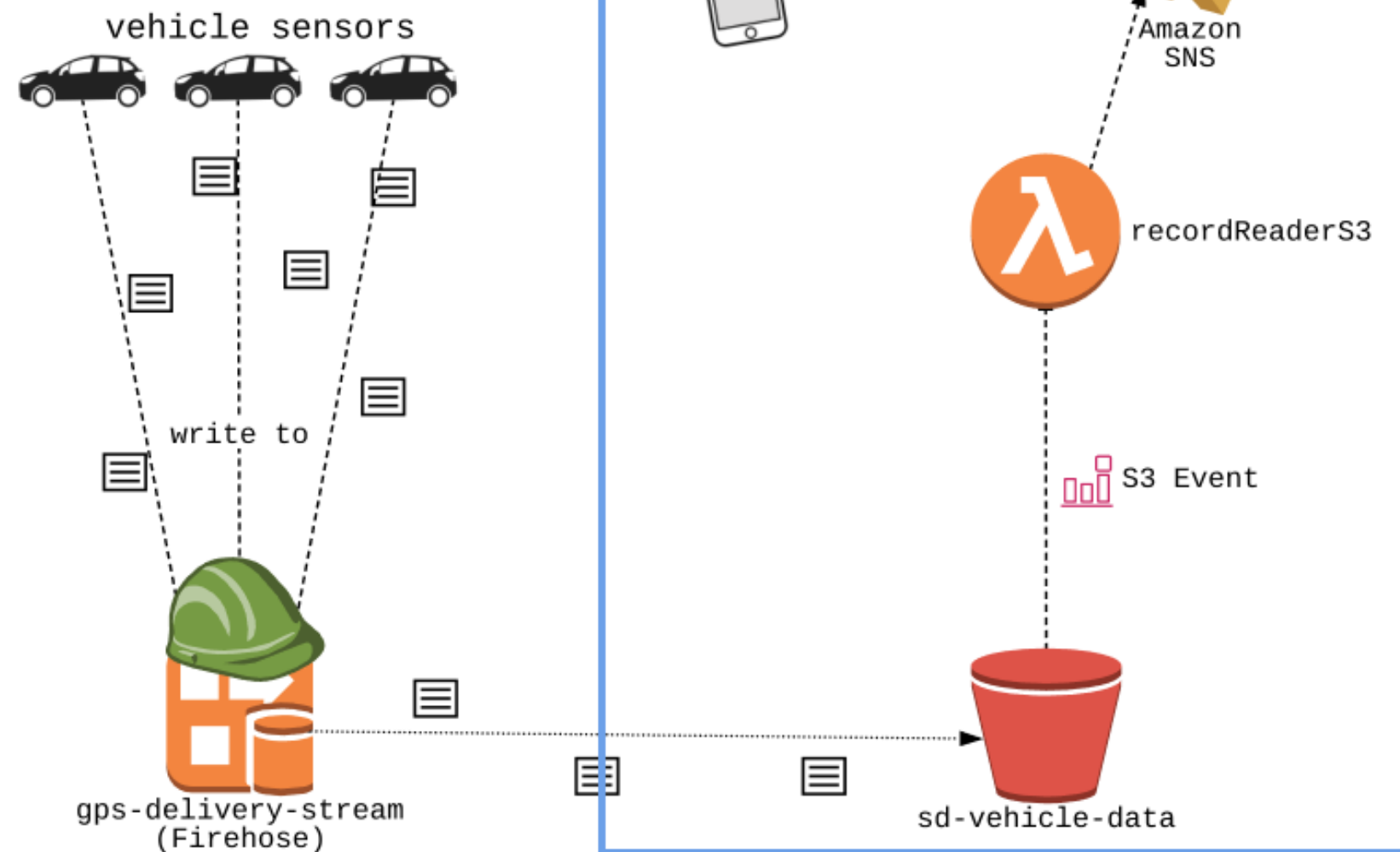
**Maksim Pecherskiy**  
Data Engineer

# Last lesson





# This lesson



# Common log format

```
bc75da5f-1bf6-444c-80ad-49c180e1b8de 23:16:06.000 3FTEX1G5XAK844393 -76.6990172 2.481207 40  
ff8e7131-408d-463b-8d07-d016419b0656 20:26:44.000 2LAXX1C8XAK844292 114.39239199999999 36.097577 90  
f29a5b3d-d0fa-43c0-9e1a-e2a5cdb8be7a 8:10:47.000 3FTEX1G5XAK844393 108.58068100000001 34.79925 37
```

# Reading S3 files

```
import pandas as pd
```

results in:

Response:

```
{  
  "errorMessage": "Unable to import module 'lambda_function': No module named 'panda"  
  "errorType": "Runtime.ImportModuleError"  
}
```

# Updating the handler

```
#lambda_function.py
import json, boto3, pandas as pd
# Initialize clients
...
SPEED_ALERT_THRESHOLD = 45
ALERT_PHONE_NUMBER = "+1234567890"
# Helper function to get dataframe from written records
def get_new_data(event):
    pd.read_csv()...
    ...
    return data
# Lambda function handler
def record_created_handler(event, context):
    data = get_new_data(event)
    ...
    sns.publish()
    ...
```

# get\_new\_data()

```
def get_new_data(event):  
    # Create a list to store new object keys.  
    written_objects = []  
  
    # Iterate over each S3 event record.  
    for record in event['Records']:  
  
        # Get the variables to check for  
        event_name = record['eventName']  
        bucket_name = record['s3']['bucket']['name']  
        obj_key = record['s3']['object']['key']
```

# get\_new\_data()

```
def get_new_data(event):  
    ...  
    # Verify that event is created from sd-vehicle-data bucket.  
    if event_name == 'ObjectCreated:Put' and bucket_name == 'sd-vehicle-data':  
        obj = s3.get_object(Bucket=bucket_name, Key = obj_key)  
        df = pd.read_csv(obj['Body'], delimiter = " ",  
                        names=["record_id", "timestamp", "vin", "lon", "lat", "speed"])  
        written_objects.append(df)  
    # Concatenate new records into a single dataframe.  
    return pd.concat(written_objects)
```

# record\_created\_handler()

```
SPEED_ALERT_THRESHOLD = 45
ALERT_PHONE_NUMBER = "+1234567890"
...
def record_created_handler(event, context):
    # Call the helper method
    data = get_new_data(event)
    ## Get the top speeds
    top_speeds = data.groupby(['vin'])['speed'].max().reset_index()
    ## Get top speeds that exceed the limit of 45
    too_fast = top_speeds.loc[top_speeds.speed > SPEED_ALERT_THRESHOLD, :]
```

# record\_created\_handler()

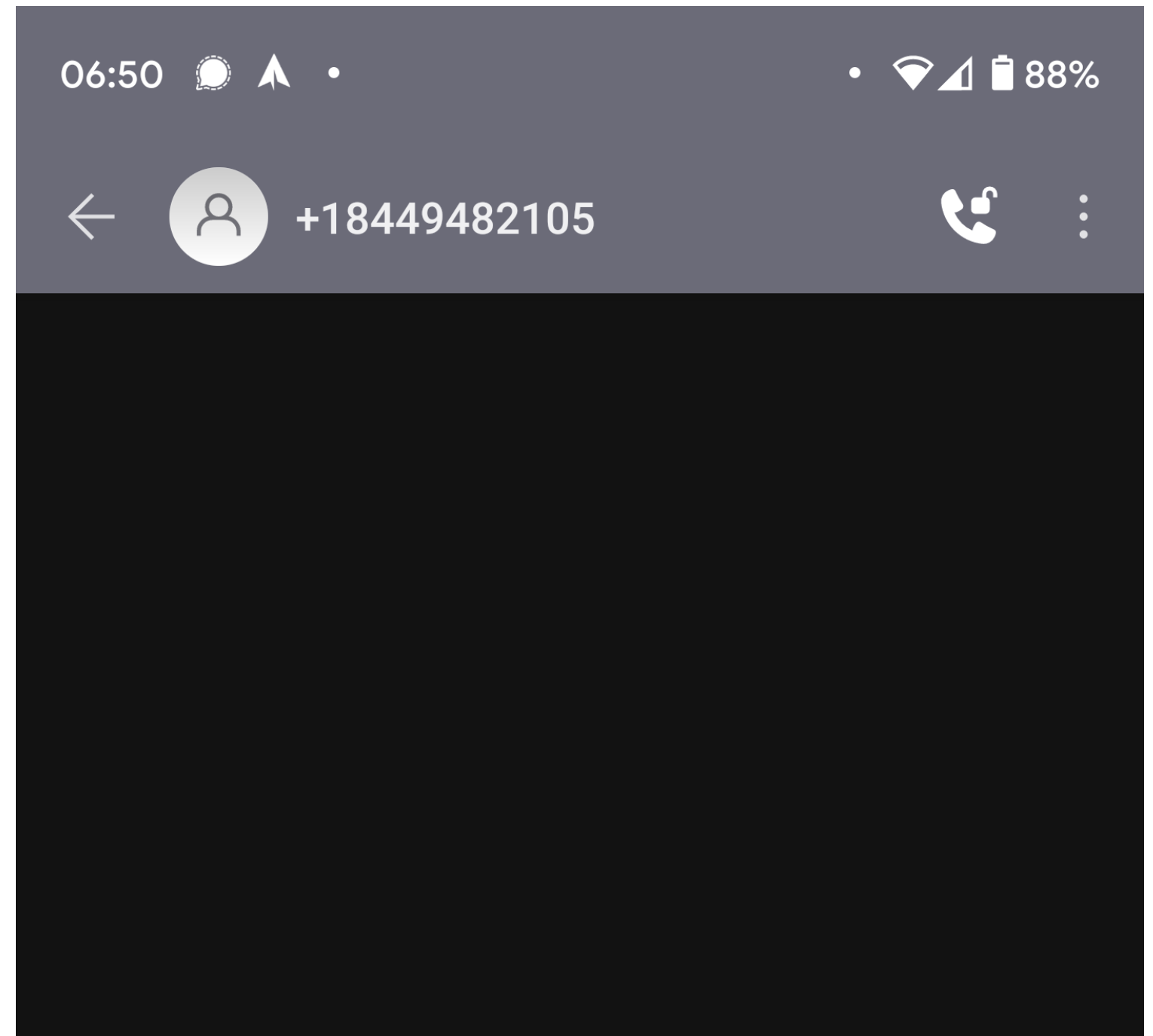
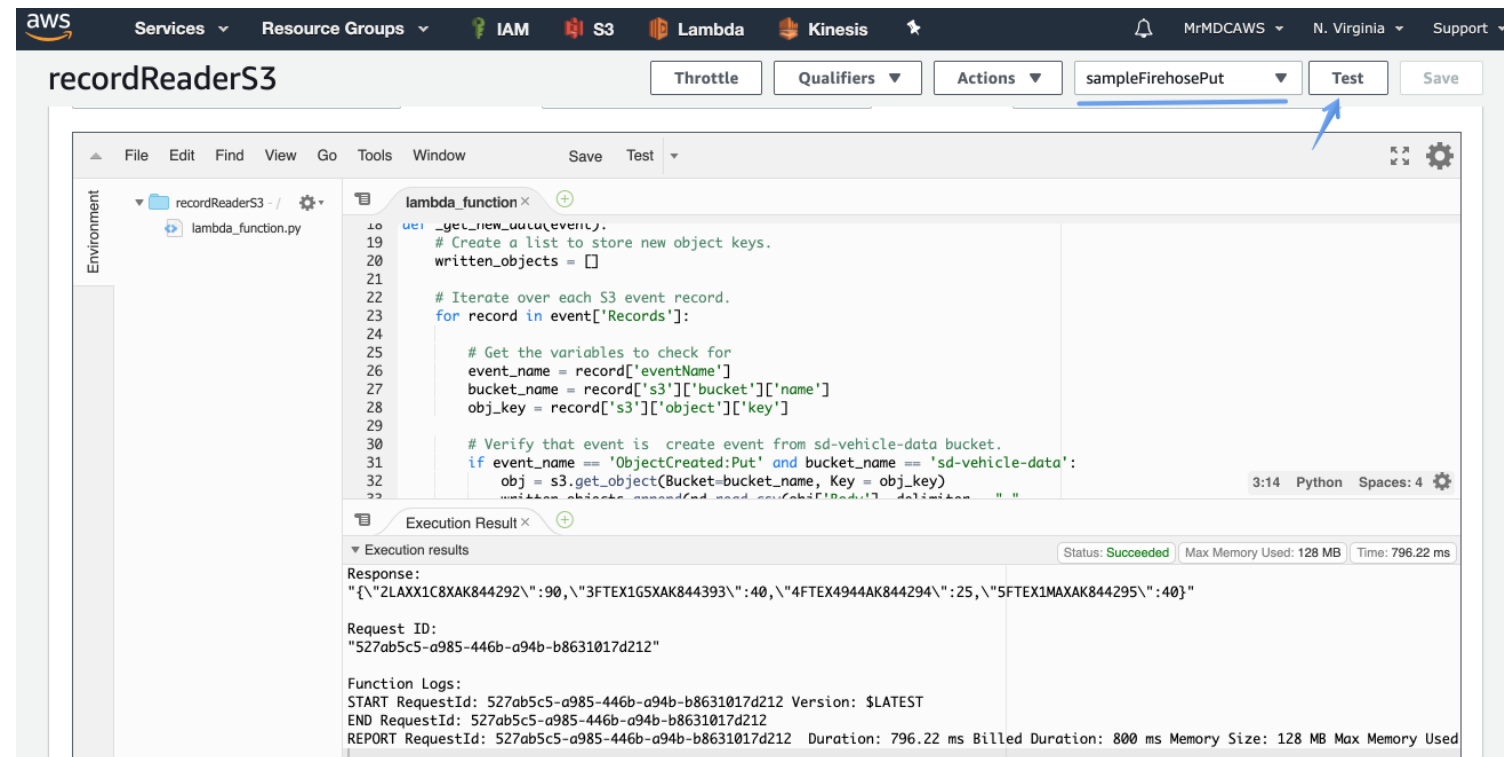
```
SPEED_ALERT_THRESHOLD = 45
ALERT_PHONE_NUMBER = "+1234567890"

...

def record_created_handler(event, context):
    ...
    ## Send SMS
    sns.publish(PhoneNumber=ALERT_PHONE_NUMBER,
                Message="Speeding Alert \n" + too_fast.to_string())
    ## This doesn't go anywhere yet, but we need to return something.
    totals = data.groupby(['vin'])['speed'].max().reset_index()
    return totals.to_csv(sep=" ", index=False)
```



# Test the Lambda function



# Adding environment variables

```
import os  
os.environ.get("ENV_VARIABLE_NAME", "DEFAULT_VALUE")
```

```
import os  
SPEED_ALERT_THRESHOLD = os.environ.get("SPEED_ALERT_THRESHOLD", 45)  
ALERT_PHONE_NUMBER = os.environ.get("ALERT_PHONE_NUMBER", None)  
...  
  
def record_created_handler(event, context):  
    ...
```

# Adding environment variables

63

`return totals.to_csv(sep=" ", index=False)`

56:6 Python Spaces: 4 ⚙

Environment variables (0) Edit

Key	Value
No environment variables	
No environment variables associated with this function.	
<span>Manage environment variables</span>	

# Adding environment variables

## Environment variables

You can define environment variables as key-value pairs that are accessible from your function code. These are useful to store configuration settings without the need to change function code. [Learn more](#)

Key	Value	
ALERT_PHONE_NUMBER	+17736777755	Remove
SPEED_ALERT_THRESHOLD	45	Remove
<button>Add environment variable</button>		

► Encryption configuration

CancelSave



# Adding a trigger

Lambda > Functions > recordReaderS3 ARN - arn:aws:lambda:us-east-1:458913182630:function:recordReaderS3


## recordReaderS3


Throttle Qualifiers ▼ Actions ▼ sampleFirehosePut ▼ Test Save

Configuration Permissions Monitoring

▼ Designer

+ Add trigger

 recordReaderS3

 Layers (1)

+ Add destination

# Adding a trigger

## Trigger configuration



S3

aws

### Bucket

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

sd-vehicle-data

### Event type

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.

All object create events

Prefix - optional

# Adding a trigger

Lambda > Functions > recordReaderS3 ARN - arn:aws:lambda:us-east-1:458913182630:function:recordReaderS3

recordReaderS3 Throttle Qualifiers ▼ Actions ▼ sampleFirehosePut ▼ Test Save

✓ The trigger sd-vehicle-data was successfully added to function recordReaderS3. The function is now receiving events from the trigger. ✕

**Configuration** | Permissions | Monitoring

▼ Designer

recordReaderS3  
Layers (1)

S3 ✕

+ Add trigger

+ Add destination

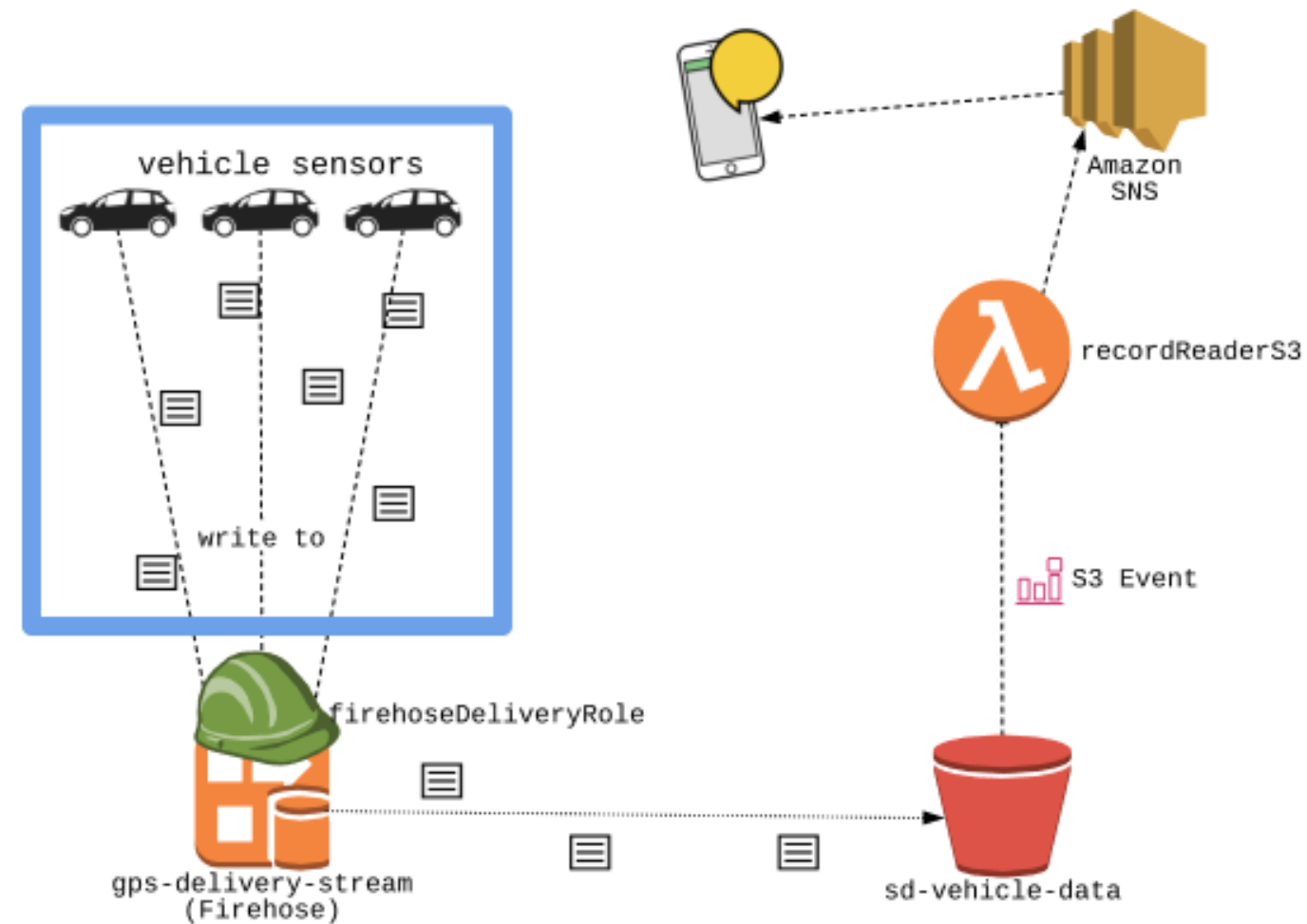
**S3**

**sd-vehicle-data**  
arn:aws:s3::sd-vehicle-data  
Bucket: s3/sd-vehicle-data Event type: ObjectCreated Notification name: 3408089e-9983-4929-82ef-9a0539b28e29

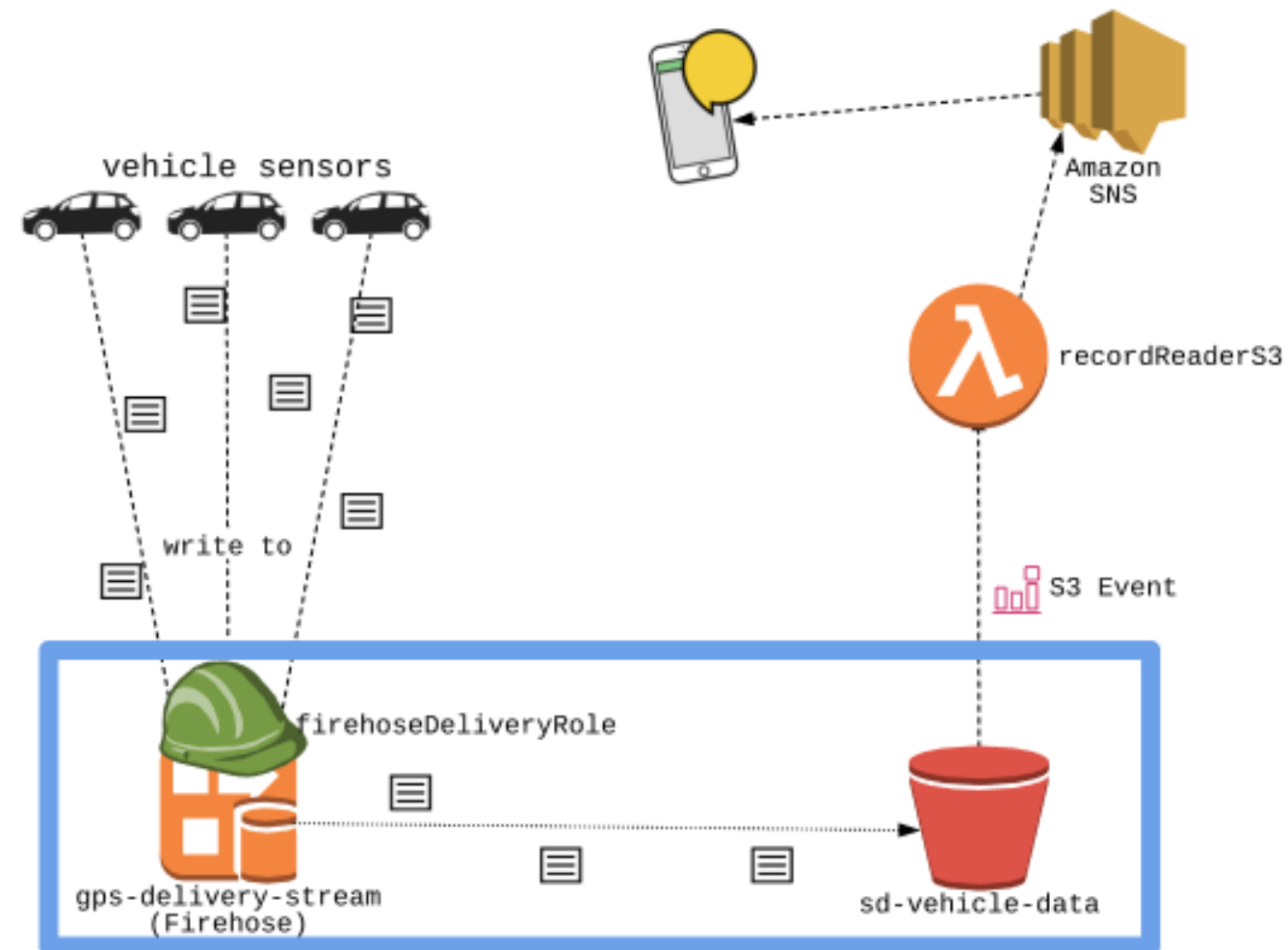
Enabled Delete



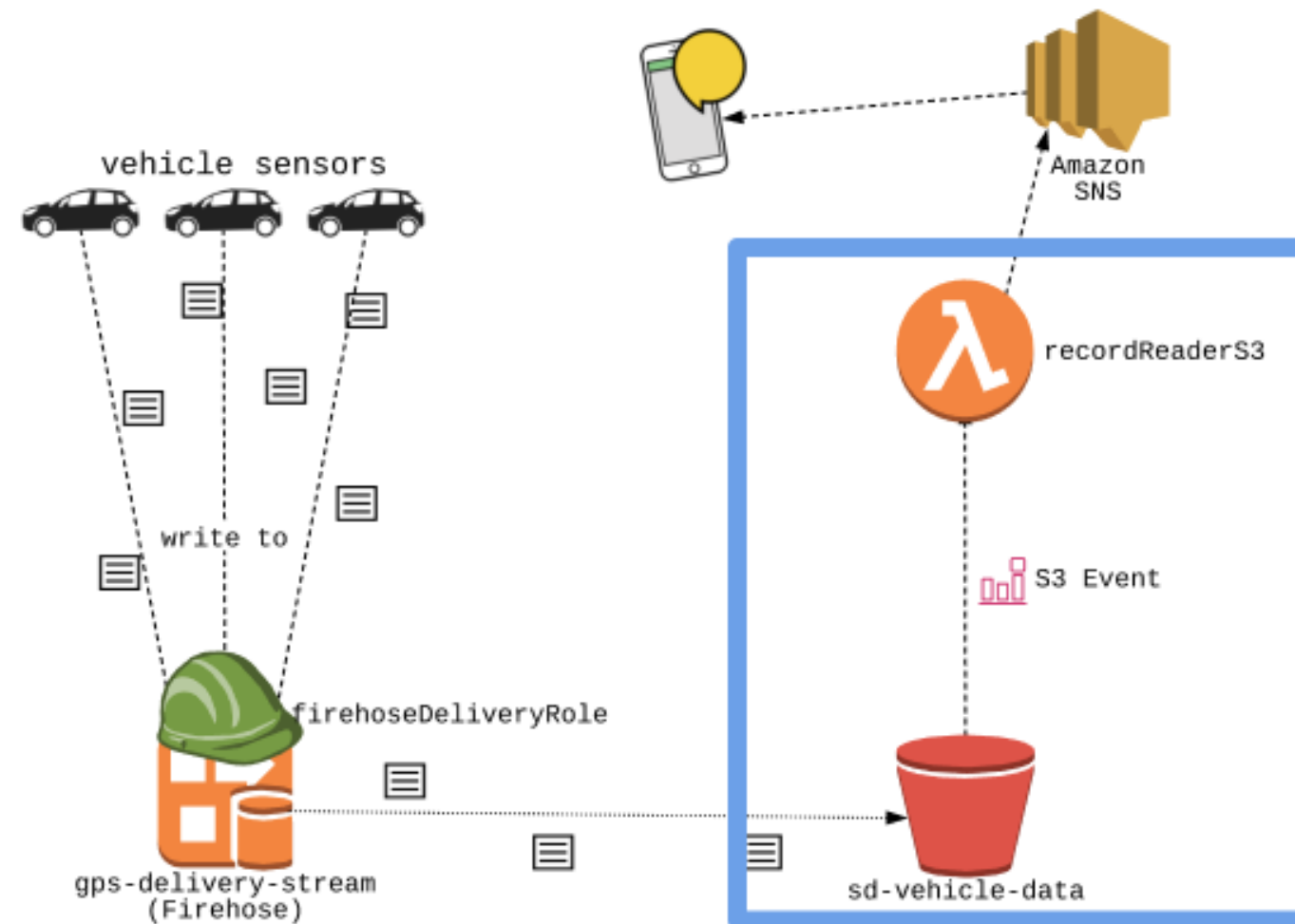
# Review



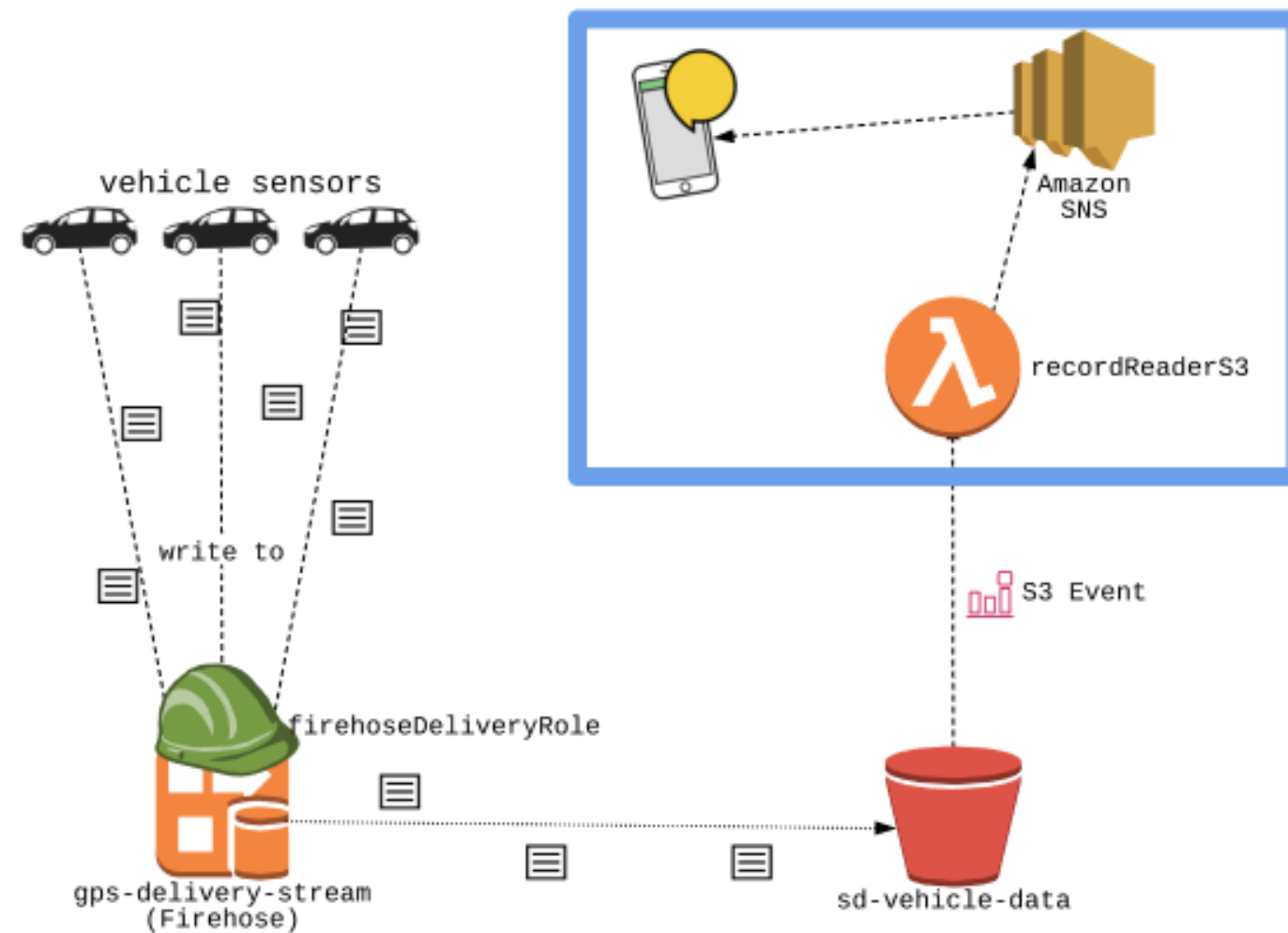
# Review



# Review



# Review



# Let's practice!

STREAMING DATA WITH AWS KINESIS AND LAMBDA

# Adding a lambda layer

STREAMING DATA WITH AWS KINESIS AND LAMBDA



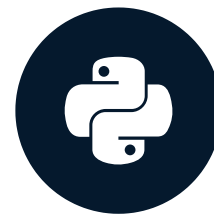
**Maksim Pecherskiy**  
Data Engineer

# Let's practice!

STREAMING DATA WITH AWS KINESIS AND LAMBDA

# Serverless data workflow

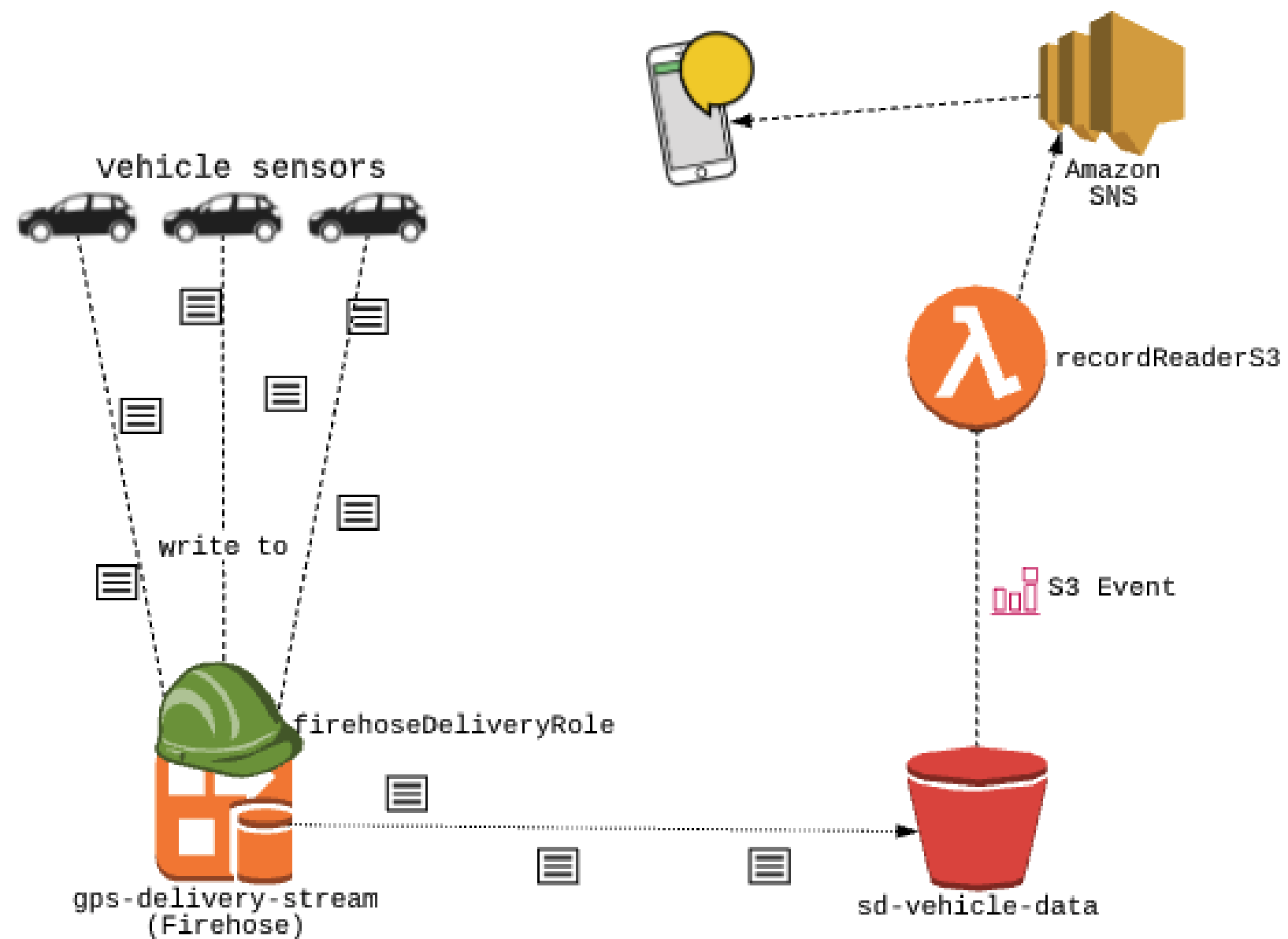
STREAMING DATA WITH AWS KINESIS AND LAMBDA



**Maksim Pecherskiy**  
Data Engineer



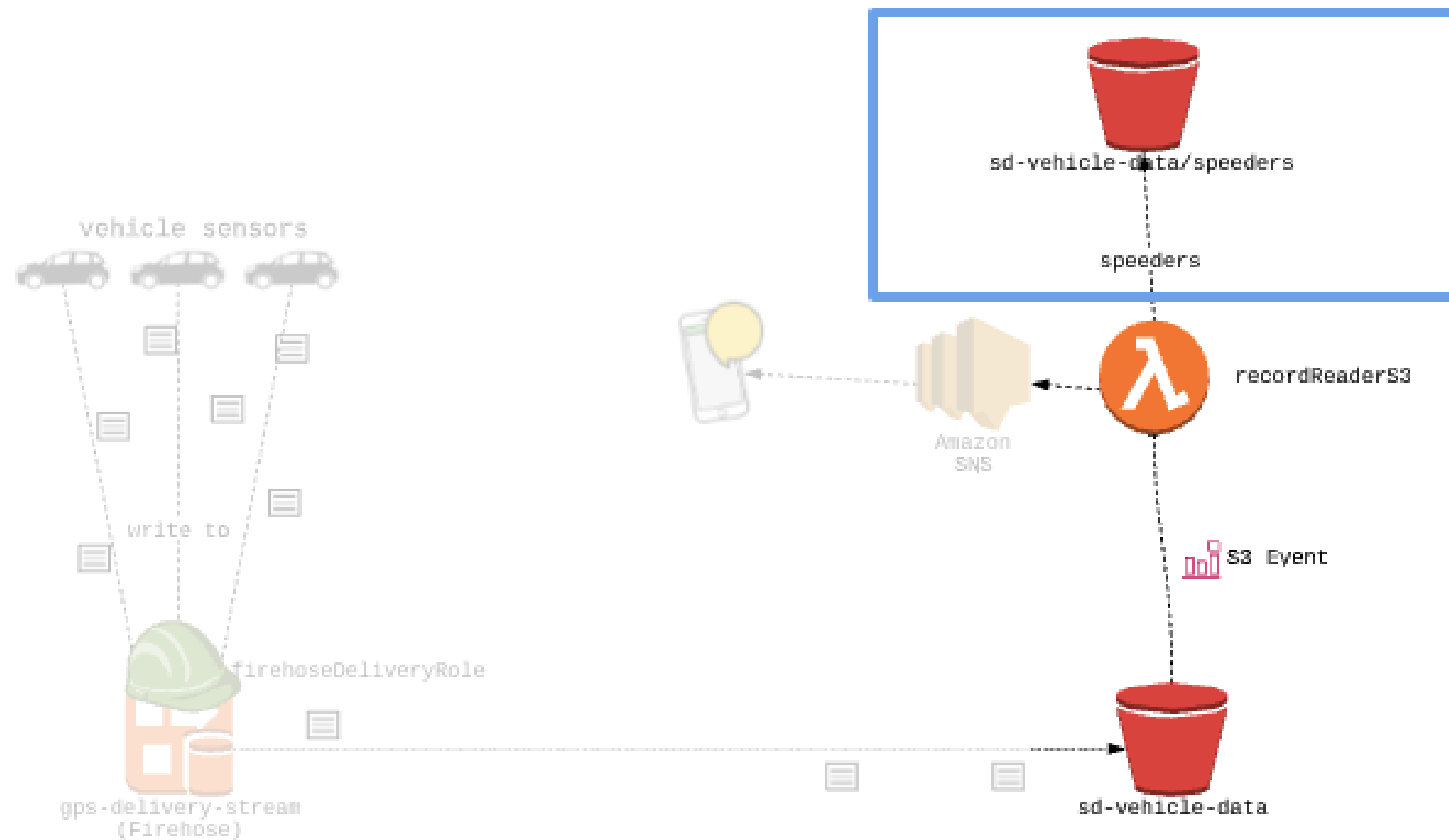
# A look back



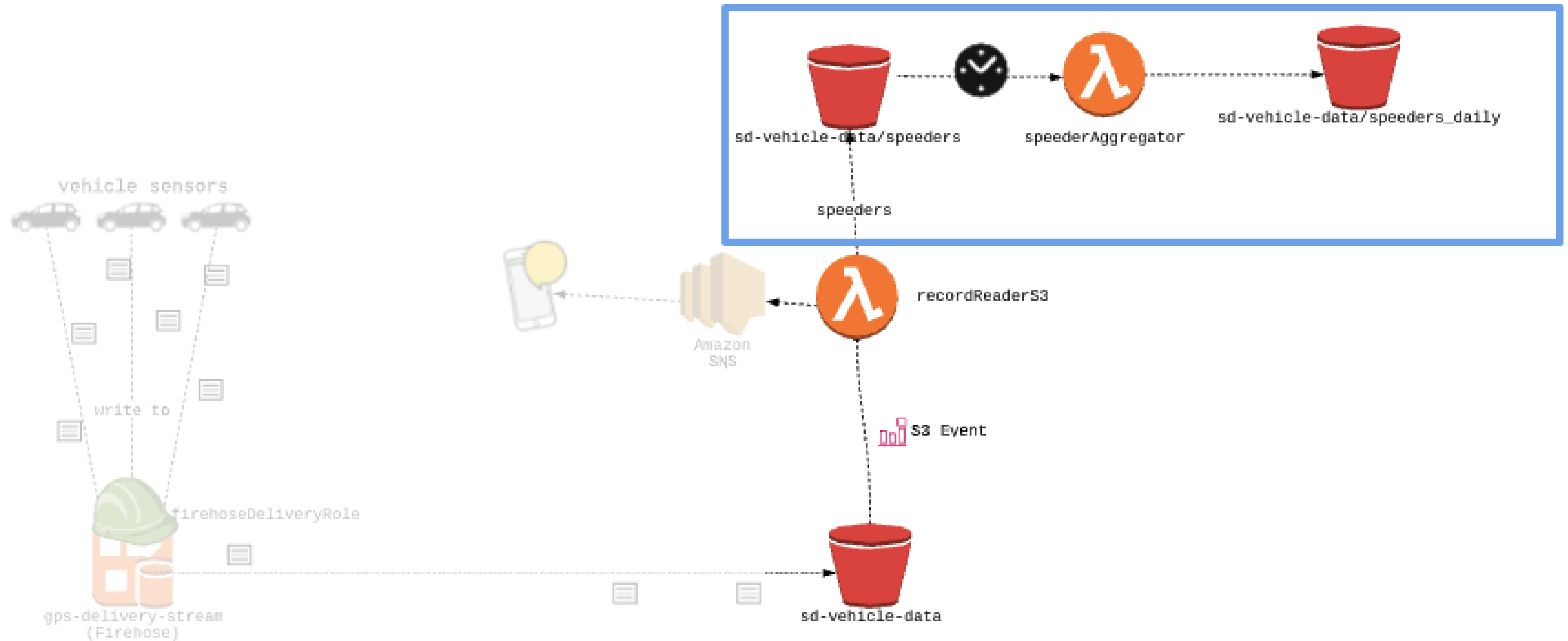
# Current partitioning

Name		^	\$
▼	2020		
▼	04	← Month	
▼	13	← day	hour
▼	15	←	
	gps-delivery-stream-1-2020-...d4-4fef-b177-874b0152dc57		
▼	22		
▼	11		
	gps-delivery-stream-1-2020-...3f-4cd5-a18e-1590520c0198		

# Filtering out speeders



# Aggregating by day




# Open recordReaderS3


Lambda > Functions > recordReaderS3 ARN - [arn:aws:lambda:us-east-1:458913182630:function:recordReaderS3](#)


recordReaderS3 Throttle Qualifiers ▼ Actions ▼ sampleFirehosePut ▼ Test Save

**Configuration** | Permissions | Monitoring

▼ Designer

 recordReaderS3  
[i](#) Unsaved changes

 Layers (1)

 S3 ×

+ Add trigger

+ Add destination

Function code [Info](#) Actions ▼

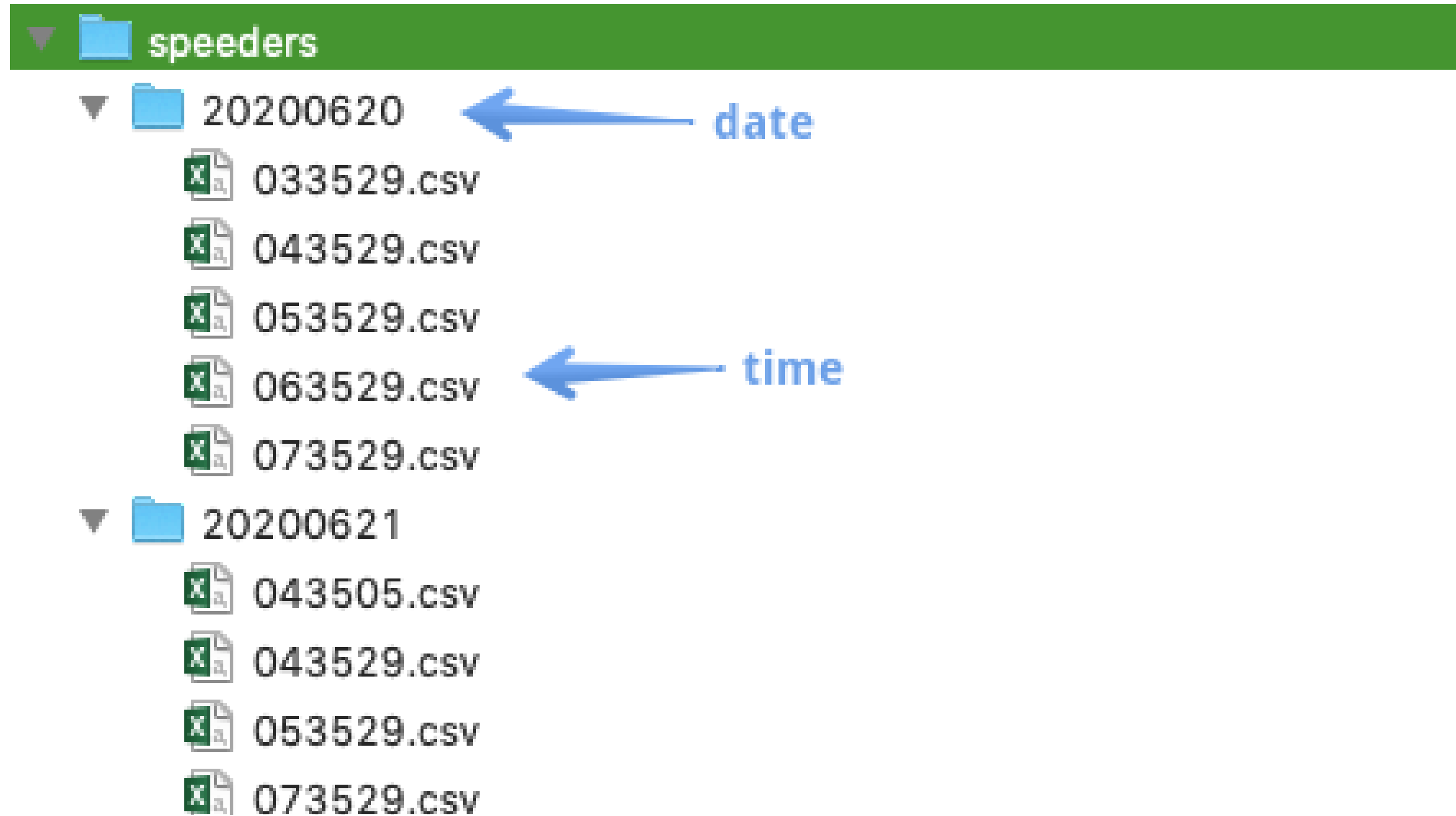
# Editing recordReaderS3

```
...  
import pytz  
from datetime import datetime  
tz = pytz.timezone('America/Los_Angeles')
```

# Editing recordReaderS3

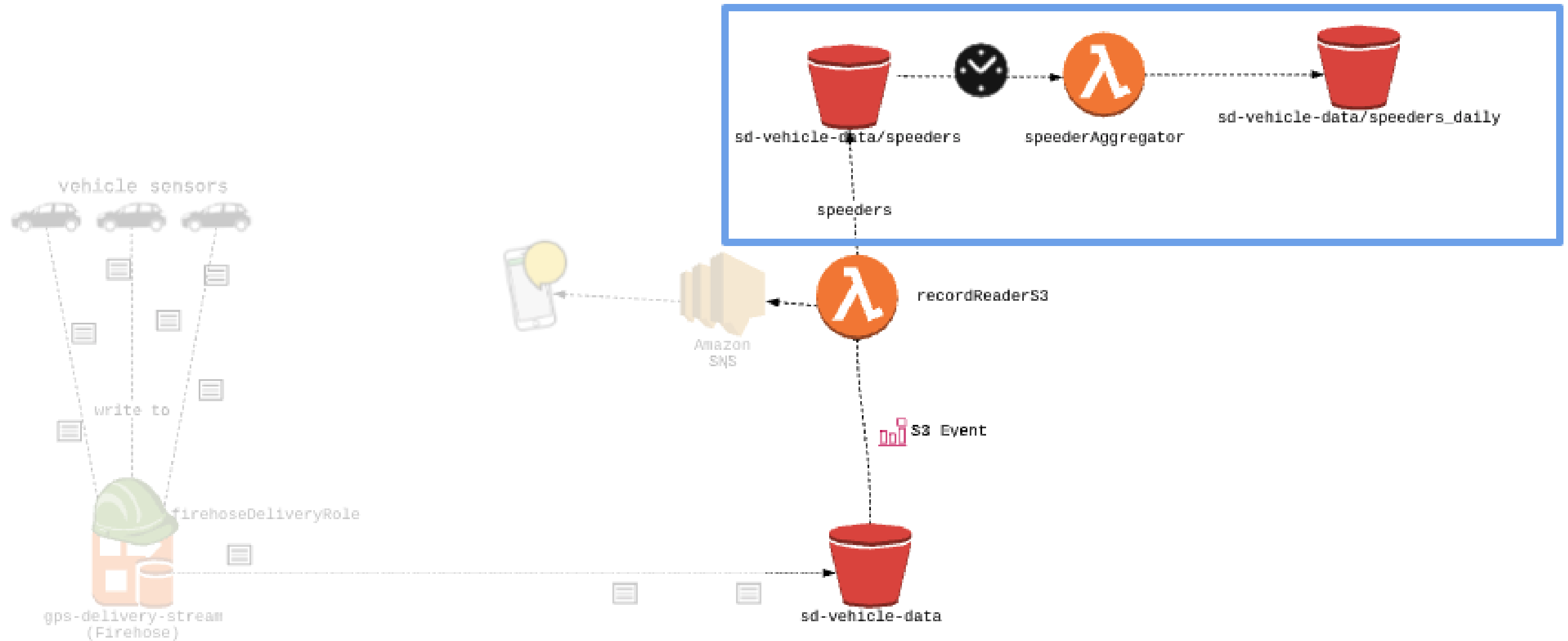
```
def record_created_handler(event, context):  
    ...  
    ## Get top speeds that exceed the limit of 45  
    too_fast = top_speeds.loc[top_speeds.speed > SPEED_ALERT_THRESHOLD :]  
    ## Generate object key  
    fdate = datetime.now(tz).strftime("%Y%m%d/%H%M%S")  
    obj_key = f"speeders/{fdate}.csv" # filename in speeders folder  
  
    ## Write the object to S3  
    s3.put_object(Bucket='sd-vehicle-data',  
                  Key=obj_key,  
                  Body=too_fast.to_csv(sep=" ", index=False)  
    )
```

# Writing speeders by date and time





# speederAggregator



# Create speederAggregator

## Basic information

### Function name

Enter a name that describes the purpose of your function.

speederAggregator

Use only letters, numbers, hyphens, or underscores with no spaces.

### Runtime [Info](#)

Choose the language to use to write your function.

Python 3.8

## Permissions [Info](#)

Lambda will create an execution role with permission to upload logs to Amazon CloudWatch Logs. You can configure and modify permissions further when you add triggers.

### ▼ Choose or create an execution role

#### Execution role

Choose a role that defines the permissions of your function. To create a custom role, go to the [IAM console](#).

- ☒ Create a new role with basic Lambda permissions
- ☐ Use an existing role
- ☐ Create a new role from AWS policy templates

# Add AWS data wrangler layer

[Lambda](#) > [Layers](#) > Add layer to function

## Add layer to function

### Layer selection

Choose from layers that are compatible with your function's runtime, or specify the Amazon Resource Name (ARN) of a layer version.

☒ Select from list of runtime compatible layers  
☐ Provide a layer version ARN

### Compatible layers

Name

awsDataWrangler ▼

Version

1 ▼

Cancel Add

# speederAggregator Resources

[Lambda](#) > [Functions](#) > [speederAggregator](#) > Edit basic settings

## Edit basic settings

### Basic settings

Description - *optional*

Runtime

Python 3.8 ▼

Handler [Info](#)

lambda\_function.speeder\_aggregator

Memory (MB) [Info](#)

Your function is allocated CPU proportional to the memory configured.

512 MB


Timeout [Info](#)

0 min  15 sec

# Create the timed trigger

## Add trigger

### Trigger configuration

 **EventBridge (CloudWatch Events)**  
aws events management-tools

▼

#### Rule

Pick an existing rule, or create a new one.

Create a new rule ▼

Select or create a new rule

**Rule name\***  
Enter a name to uniquely identify your rule.

dailyMidnight

#### Rule description

Provide an optional description for your rule.

Run every night right before midnight PST

**Rule type**  
Trigger your target based on an event pattern, or based on an automated schedule.

☐ Event pattern

☒ Schedule expression

**Schedule expression\***  
Self-trigger your target on an automated schedule using Cron or rate expressions. Cron expressions are in UTC.

cron(50 6 \* \* ? \*)

e.g. rate(1 day), cron(0 17 ? \* MON-FRI \*)

Lambda will add the necessary permissions for Amazon EventBridge (CloudWatch Events) to invoke your Lambda

```
cron(Minutes Hours Day-of-month Month Day-of-week Year)
```

```
cron(50 6 * * ? *)
```

- On Minute 50
- Of the 6th hour (UTC)
- Of Every Day of the Month
- Of Every Month
- Of Every Day of the Week
- Of Every Year

# speederAggregator callback

```
import boto3, pytz, pandas as pd

s3 = boto3.client("s3" ...)

...

def speeder_aggregator(event, context):
    tz = pytz.timezone('America/Los_Angeles')
    filter_date = datetime.now(tz).strftime("%Y%m%d")
```

# speederAggregator callback

```
def speeder_aggregator(event, context):  
    ...  
    objects = s3.list_objects_v2(  
        Bucket='sd-vehicle-data', Prefix=f'speeders/{filter_date}')  
    day_data = []  
  
    for obj in objects['Contents']:  
        print(obj['Key'])  
        day_record = s3.get_object(Bucket='sd-vehicle-data', Key = obj['Key'])  
        day_data.append(  
            pd.read_csv(day_record['Body'], delimiter = " "))  
  
    # Concatenate new records into a single dataframe.  
    data = pd.concat(day_data)  
    data.columns = ["record_id", "timestamp", "vin", "lon", "lat", "speed"]
```



# awswrangler package

```
import awswrangler as wr
session = boto3.Session(aws_access_key_id = AWS_KEY,
                        aws_secret_access_key = AWS_SECRET, region_name="us-east-1")
df = wr.s3.read_csv(f"s3://sd-vehicle-data/speeders/{filter_date}")
df.head()
```

# Old way vs awswrangler

```
def speeder_aggregator(event, context):  
    ...  
    objects = s3.list_objects_v2(  
        Bucket='sd-vehicle-data', Prefix=f'speeders/{filter_date}')  
    day_data = []  
  
    for obj in objects['Contents']:  
        print(obj['Key'])  
        day_record = s3.get_object(Bucket='sd-vehicle-data', Key = obj['Key'])  
        day_data.append(  
            pd.read_csv(day_record['Body'], delimiter = " "))  
  
    # Concatenate new records into a single dataframe.  
    data = pd.concat(day_data)  
    data.columns = ["record_id", "timestamp", "vin", "lon", "lat", "speed"]
```

# Old way vs awswrangler

```
import awswrangler as wr


session = boto3.Session(aws_access_key_id = AWS_KEY,
                        aws_secret_access_key = AWS_SECRET, region_name="us-east-1")

def speeder_aggregator(event, context):
    df = wr.s3.read_csv(f"s3://sd-vehicle-data/speeders/{filter_date}")
    df.head()
```

# Writing aggregate speeders file

```
wr.s3.to_csv(df,  
             f"s3://sd-vehicle-data/speeders_daily/{filter_date}.csv",  
             boto3_session=session,  
             sep=" ", index=False)
```


# Save and test


Lambda > Functions > speederAggregator ARN -  arn:aws:lambda:us-east-1:458913182630:function:speederAggregator


speederAggregator Throttle Qualifiers ▼ Actions ▼ testEvent ▼ Test Save

**Configuration** | Permissions | Monitoring

▼ Designer

 speederAggregator

 Layers (1)

 EventBridge (CloudWatch Events) ×

+ Add trigger

+ Add destination


# Enable the trigger


Lambda > Functions > speederAggregator ARN - [arn:aws:lambda:us-east-1:458913182630:function:speederAggregator](#)


speederAggregator Throttle Qualifiers ▼ Actions ▼ testEvent ▼ Test Save

**Configuration** | Permissions | Monitoring

▼ Designer

 speederAggregator

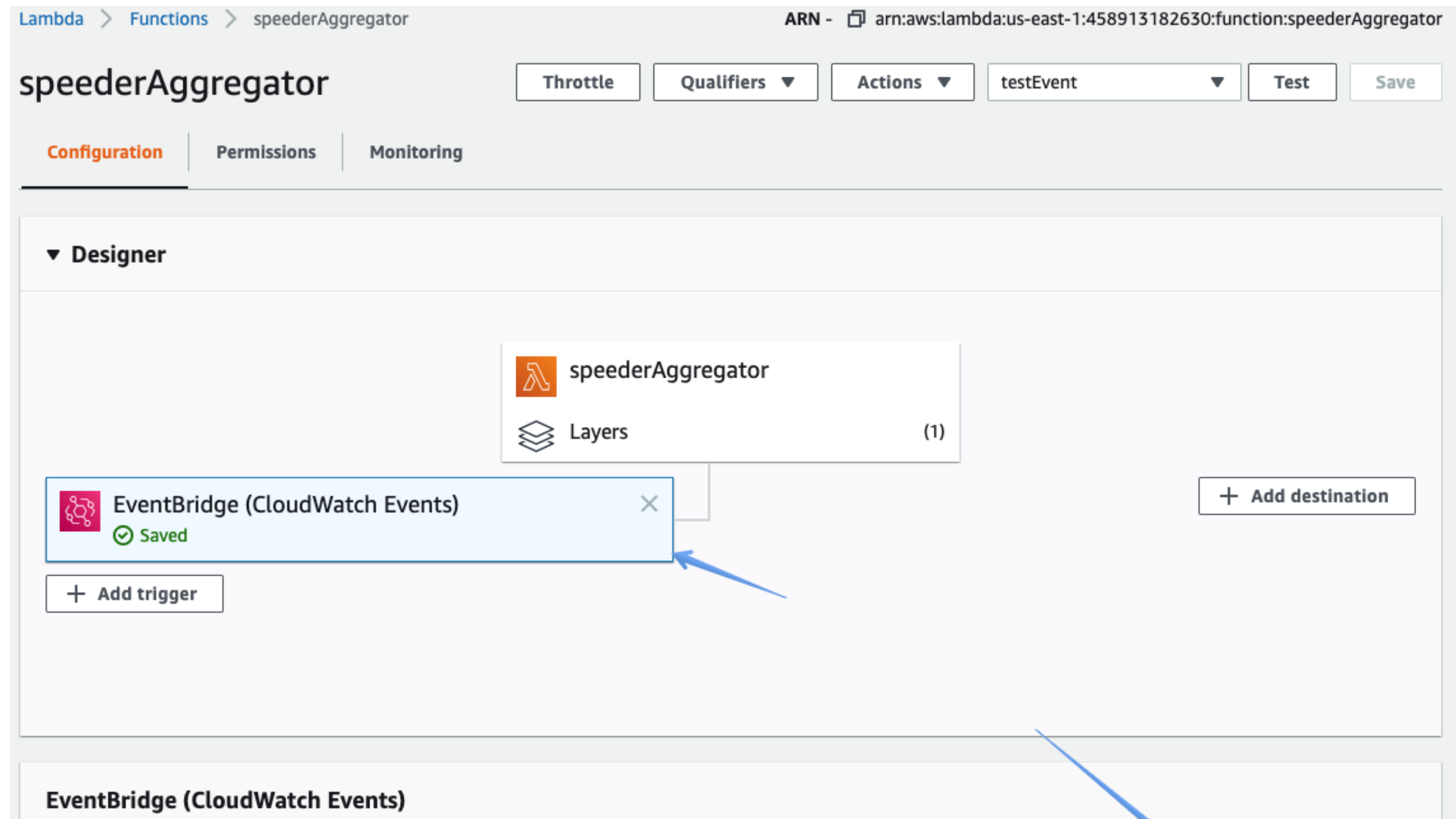
 Layers (1)

 EventBridge (CloudWatch Events)  
✔ Saved

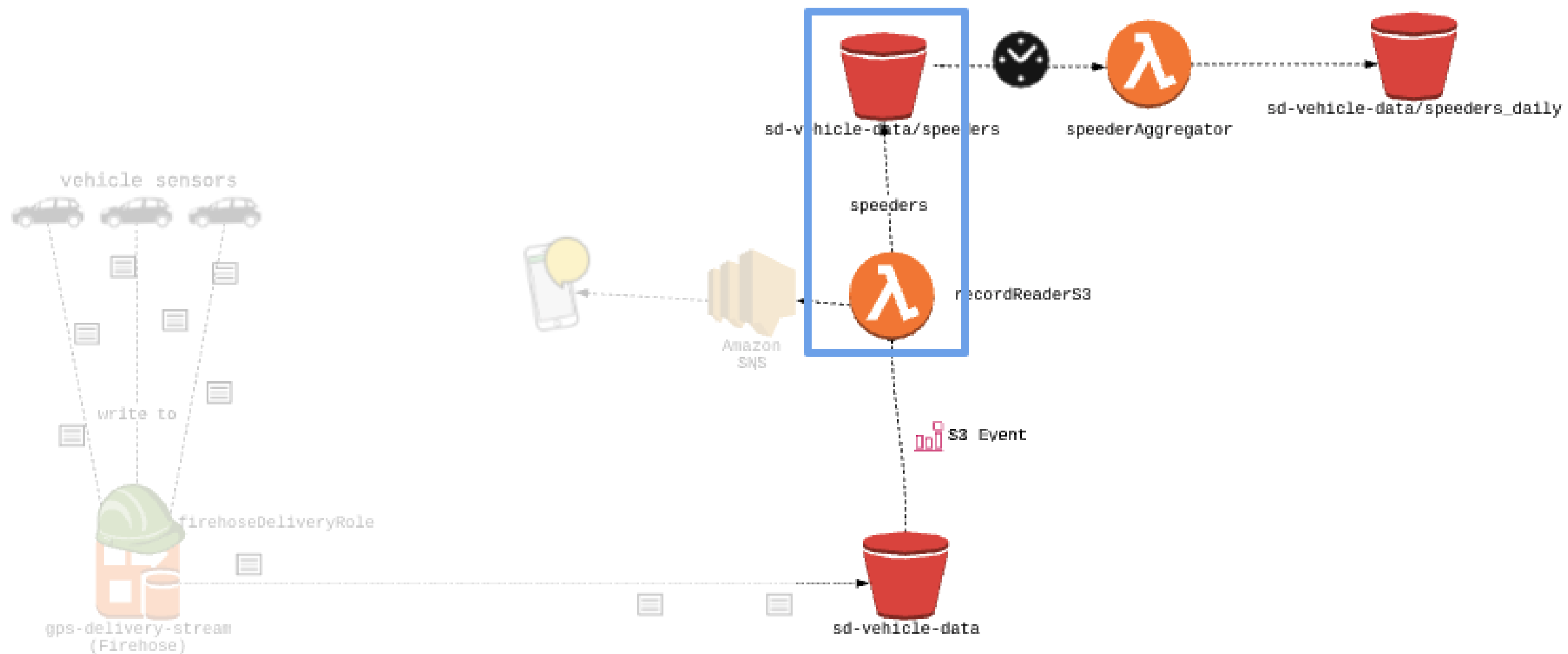
+ Add trigger

+ Add destination

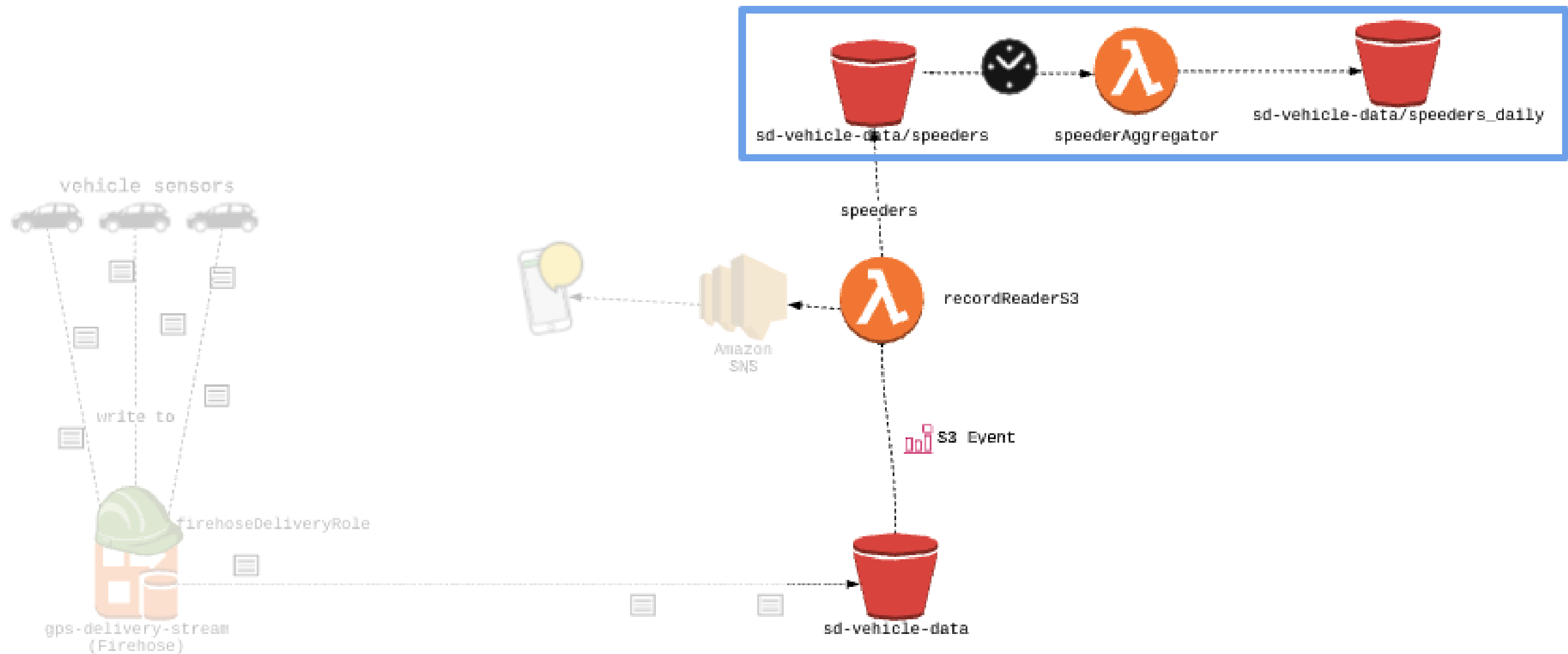
EventBridge (CloudWatch Events)



# Review



# Review





# Review

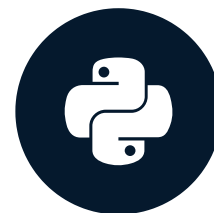
- Cron for scheduling
- `wr.s3.read_csv()`
- `wr.s3.write_csv()`

# Let's practice!

STREAMING DATA WITH AWS KINESIS AND LAMBDA

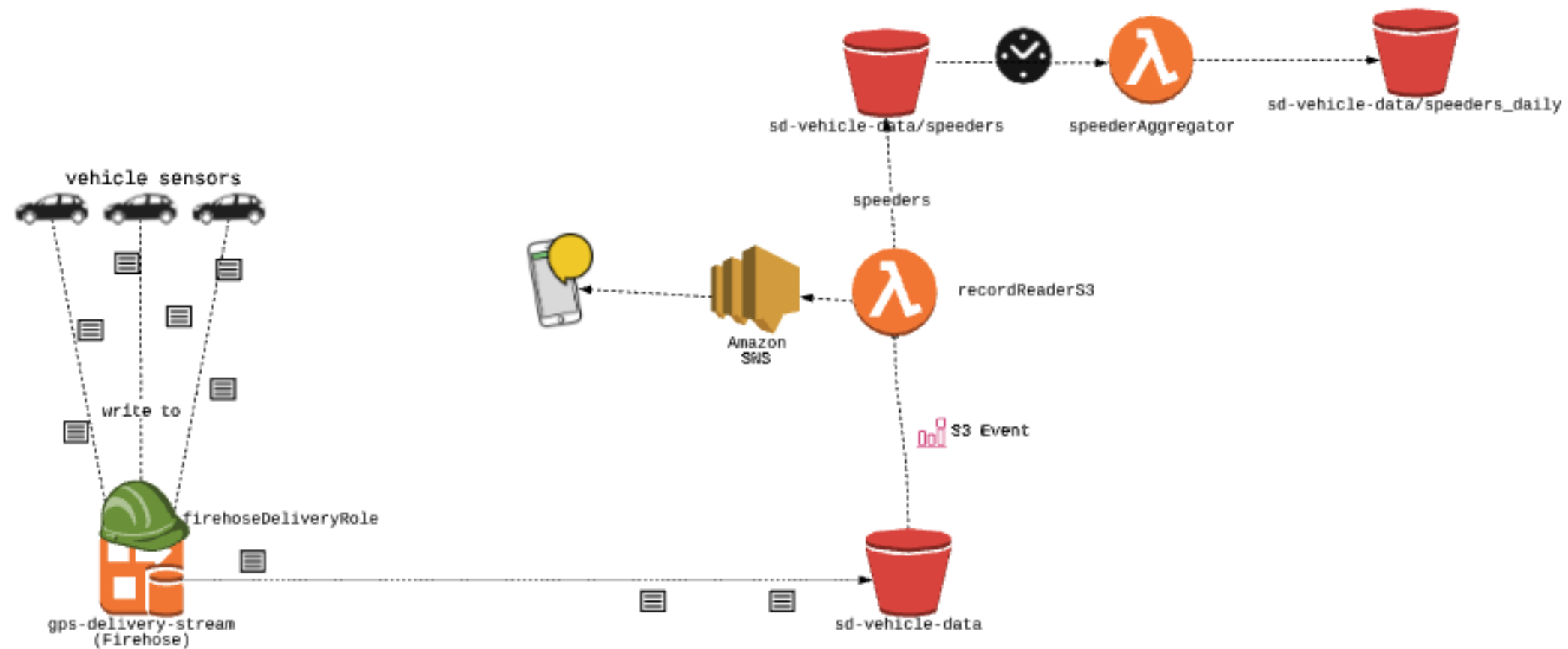
# Serverless APIs

STREAMING DATA WITH AWS KINESIS AND LAMBDA



**Maksim Pecherskiy**  
Data Engineer

# Last lesson

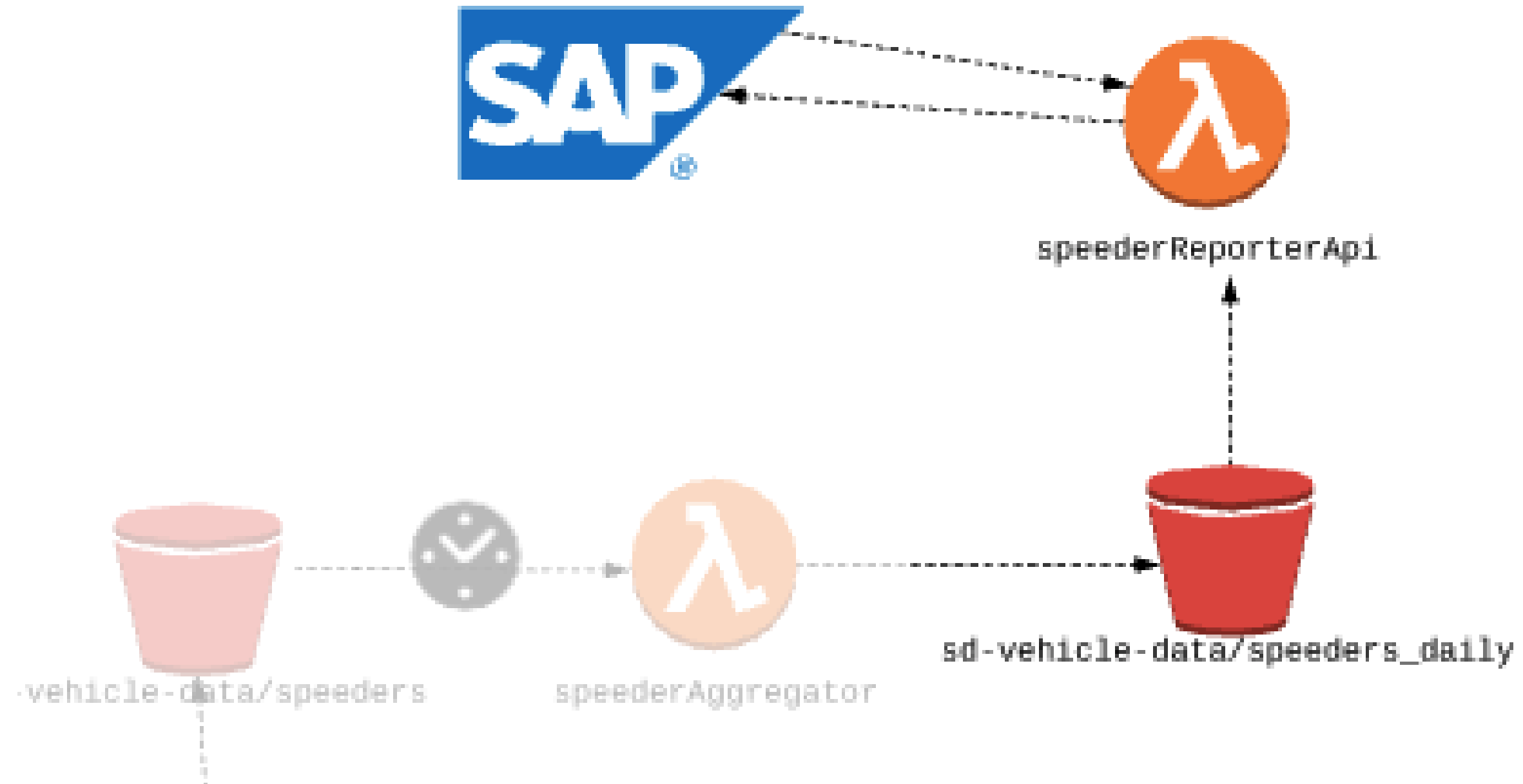


# Giving access to data

---

▶	2020	--	
▶	speeders	--	
▼	speeders_daily	--	
	20200620.csv	29.3 KB	Today at 10:32:37
	20200621.csv	29.3 KB	Today at 10:32:34

# Simple API



# Simple API

## Basic information

### Function name

Enter a name that describes the purpose of your function.

speederReporterApi

Use only letters, numbers, hyphens, or underscores with no spaces.

### Runtime [Info](#)

Choose the language to use to write your function.

Python 3.8

## Permissions [Info](#)

Lambda will create an execution role with permission to upload logs to Amazon CloudWatch Logs. You can configure and modify permissions further when you add triggers.

### ▼ Choose or create an execution role


#### Execution role


Choose a role that defines the permissions of your function. To create a custom role, go to the [IAM console](#).

- ☒ Create a new role with basic Lambda permissions
- ☐ Use an existing role
- ☐ Create a new role from AWS policy templates

# Simple API

▼ Designer

 speederReporterApi

 Layers (1)

+ Add trigger

+ Add destination

Layers [Info](#)

Add a layer


Edit

Merge order	Name	Layer version	Version ARN
1	awsDataWrangler	1	arn:aws:lambda:us-east-1:458913182630:layer:awsDataWrangler:1



# Add a trigger

**Trigger configuration**

 **API Gateway**  
api application-services aws serverless

Add an API to your Lambda function to create an HTTP endpoint that invokes your function. API Gateway supports two types of RESTful APIs: HTTP APIs and REST APIs. [Learn more](#)

**API**  
Create a new API or attach an existing one.  

Create an API

**API type**

☒ **HTTP API**  
Create an HTTP API.

☐ **REST API**  
Create a REST API.

**Security**  
Configure the security mechanism for your API endpoint.  

Open

▼ **Additional settings**


**API name**  
Choose a name for your API. API names don't need to be unique.  

getSpeedersByDate

**Deployment stage**  
The name of your API's deployment stage.  


default


☐ **Grant admin resource actions (CORS)**


 datacamp

STREAMING DATA WITH AWS KINESIS AND LAMBDA

# Test your new API

 speederReporterApi

 Layers (1)

 API Gateway

+ Add trigger

+ Add destination

### API Gateway

getSpeedersByDate

arn:aws:execute-api:us-east-1:458913182630:5npe7gpq87/\*/\*/speederReporterApi

▶ API endpoint: <https://5npe7gpq87.execute-api.us-east-1.amazonaws.com/default/speederReporterApi>

API type:

HTTP Authorization: NONE

Enabled Delete

# Our first API

The screenshot displays the AWS Lambda console interface for configuring a new API endpoint. At the top, a box for 'speederReporterApi' is shown with a 'Layers' section indicating '(1)' layer. Below this, an 'API Gateway' box is visible with an 'Add trigger' button. To the right, an 'Add destination' button is present. The main section is titled 'Function code' and includes an 'Info' link and an 'Actions' dropdown. Below the title bar is a menu with 'File', 'Edit', 'Find', 'View', 'Go', 'Tools', 'Window', 'Save', and 'Test'. The 'Environment' sidebar on the left shows the project structure with 'speederReporterApi' and 'lambda\_function.py'. The main editor area shows the 'lambda\_function' code with the following Python code:

```
1 import json
2
3 def lambda_handler(event, context):
4     # TODO implement
5     return {
6         'statusCode': 200,
7         'body': json.dumps('Hello from Lambda!')
8     }
9
```

# API parameters

## Sample request

```
https://.../speederReporterApi?date=20200620
```

## Sample event

```
{  
  "queryStringParameters": {  
    "date": "20200620"  
  },  
  "isBase64Encoded": false  
}
```

# Lambda handler

```
import json, boto3
import awswrangler as wr
import pandas as pd
session = boto3.Session(aws_access_key_id = AWS_KEY,
                        aws_secret_access_key = AWS_SECRET,
                        region_name="us-east-1")
```

# Lambda handler

```
def lambda_handler(event, context):  
    filter_date = event['queryStringParameters']['date']  
    df = wr.s3.read_csv(  
        f"s3://sd-vehicle-data/speeders_daily/{filter_date}.csv",  
        boto3_session=session,  
        delimiter=" ")
```

# Respond with data

```
def lambda_handler(event, context):  
    ...  
    return {  
        'statusCode': 200,  
        'headers': {  
            "content-type" : "application/json"  
        },  
        'body': df.to_json()  
    }
```

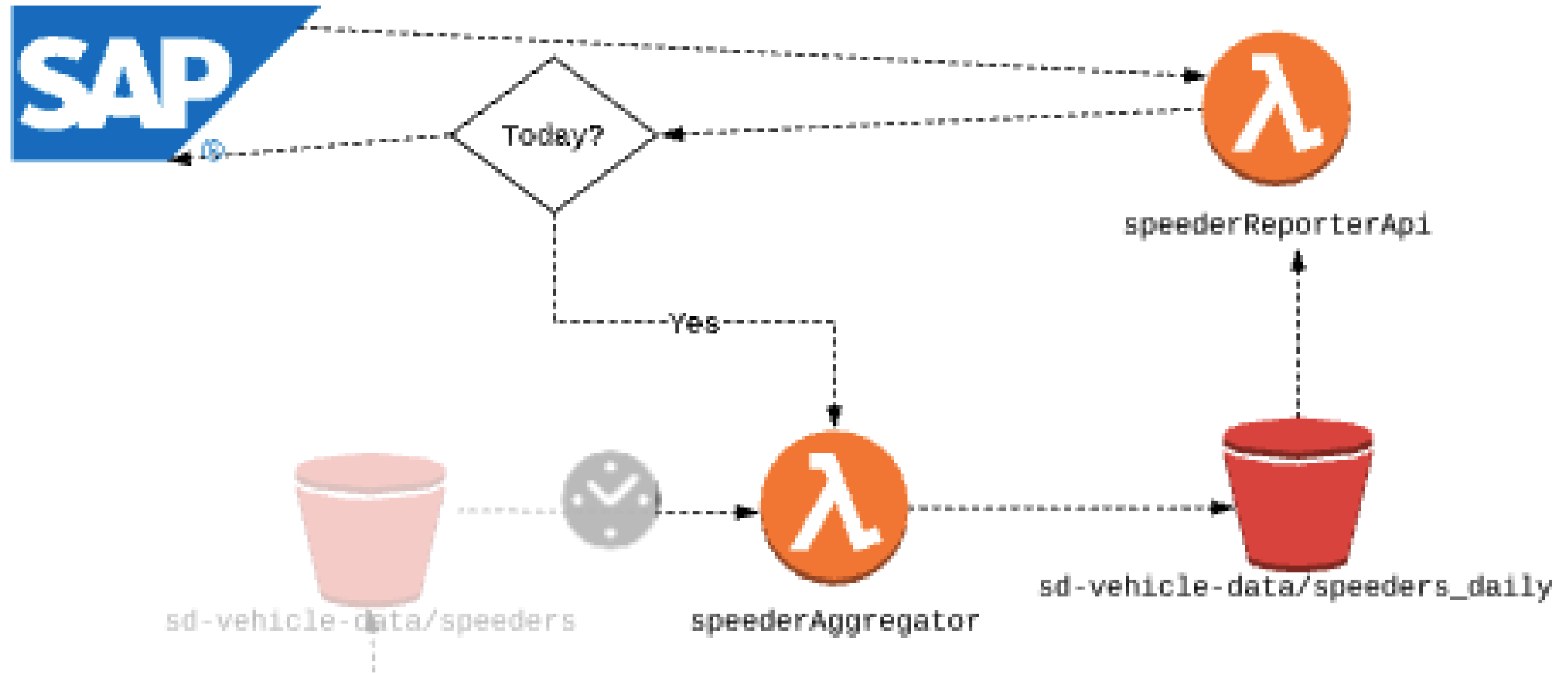
# Live response

<https://.../default/speederReporterApi?date=20200621>

```
[
  {
    "record_id": "939ed1d1-1740-420c-8906-445278573c7f",
    "timestamp": "4:25:06.000",
    "vin": "4FTEX4944AK844294",
    "lon": 106.9447146,
    "lat": -6.3385652,
    "speed": 25
  },
  {
    "record_id": "f29a5b3d-d0fa-43c0-9e1a-e2a5cdb8be7a",
    ...
  },
  ...
]
```



# Trigger another Lambda




# Trigger another Lambda

```
def trigger_recalc():  
    lambda_client = boto3.client("s3",  
                                  aws_access_key_id = AWS_KEY,  
                                  aws_secret_access_key = AWS_SECRET,  
                                  region_name = 'us-east-1')  
  
def lambda_handler(event, context):  
    ...
```

# speederAggregator ARN

[Lambda](#) > [Functions](#) > speederAggregator

ARN -  arn:aws:lambda:us-east-1:458913182630:function:speederAggregator

## speederAggregator

Throttle

Qualifiers ▼

Actions ▼

testEvent ▼

Test

Save

Configuration

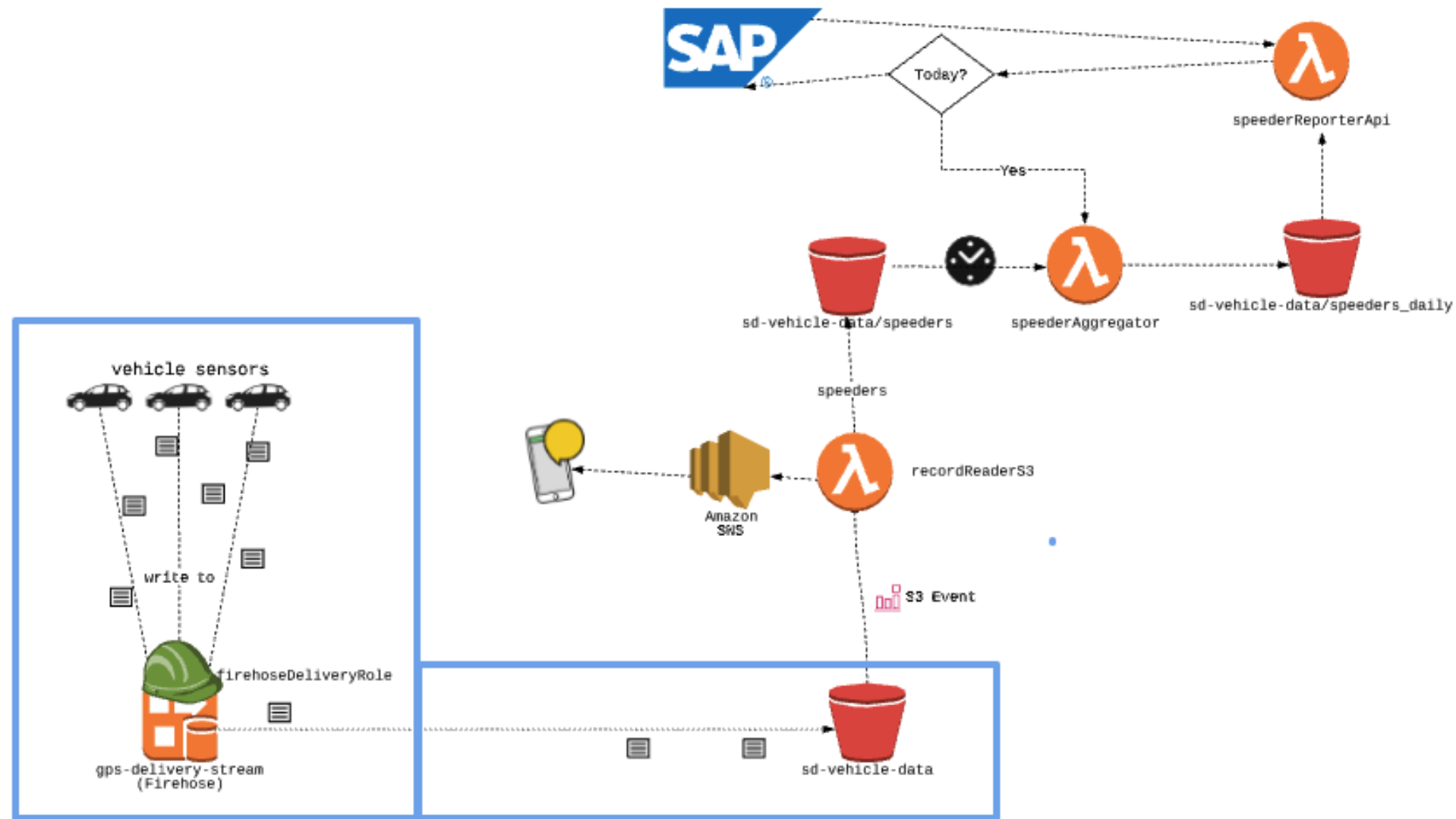
Permissions

Monitoring

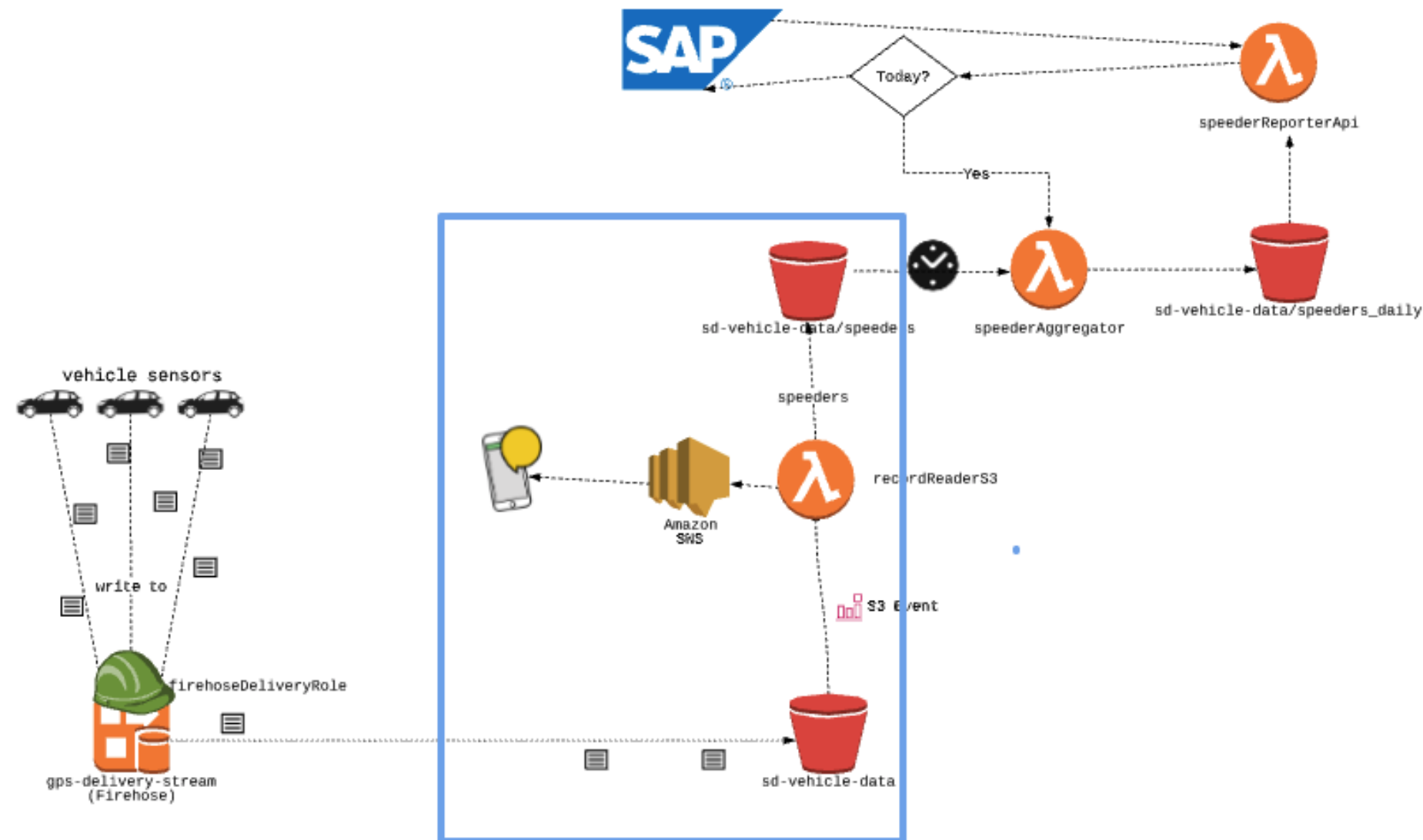
# Invoke

```
def trigger_recalc():  
    ...  
    lambda_client.invoke(  
        FunctionName='arn:aws:lambda:us-east-1:458913182630:function:speederAggregator',  
        InvocationType='Event', # (or RequestResponse)  
    )
```

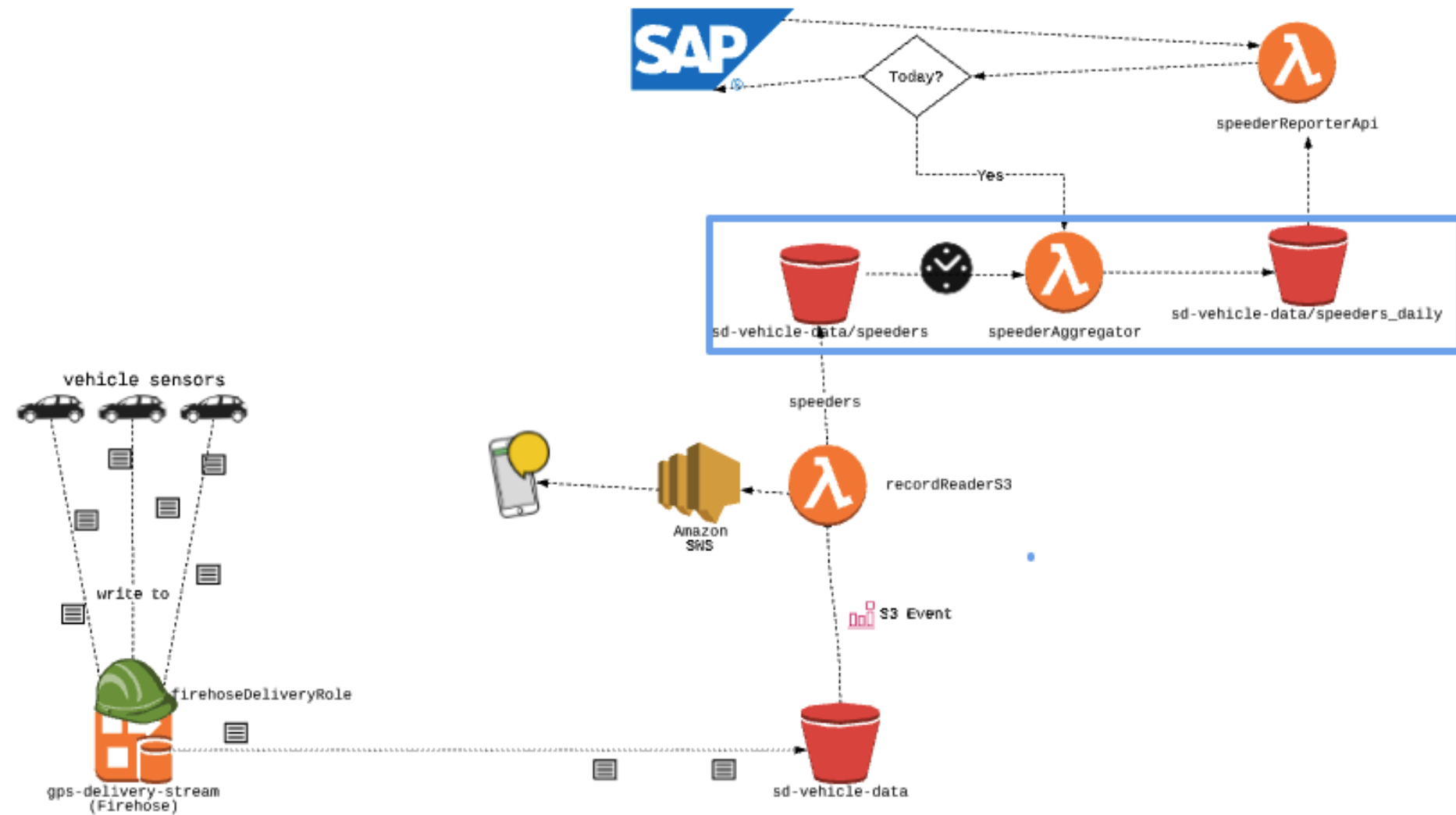
# Review



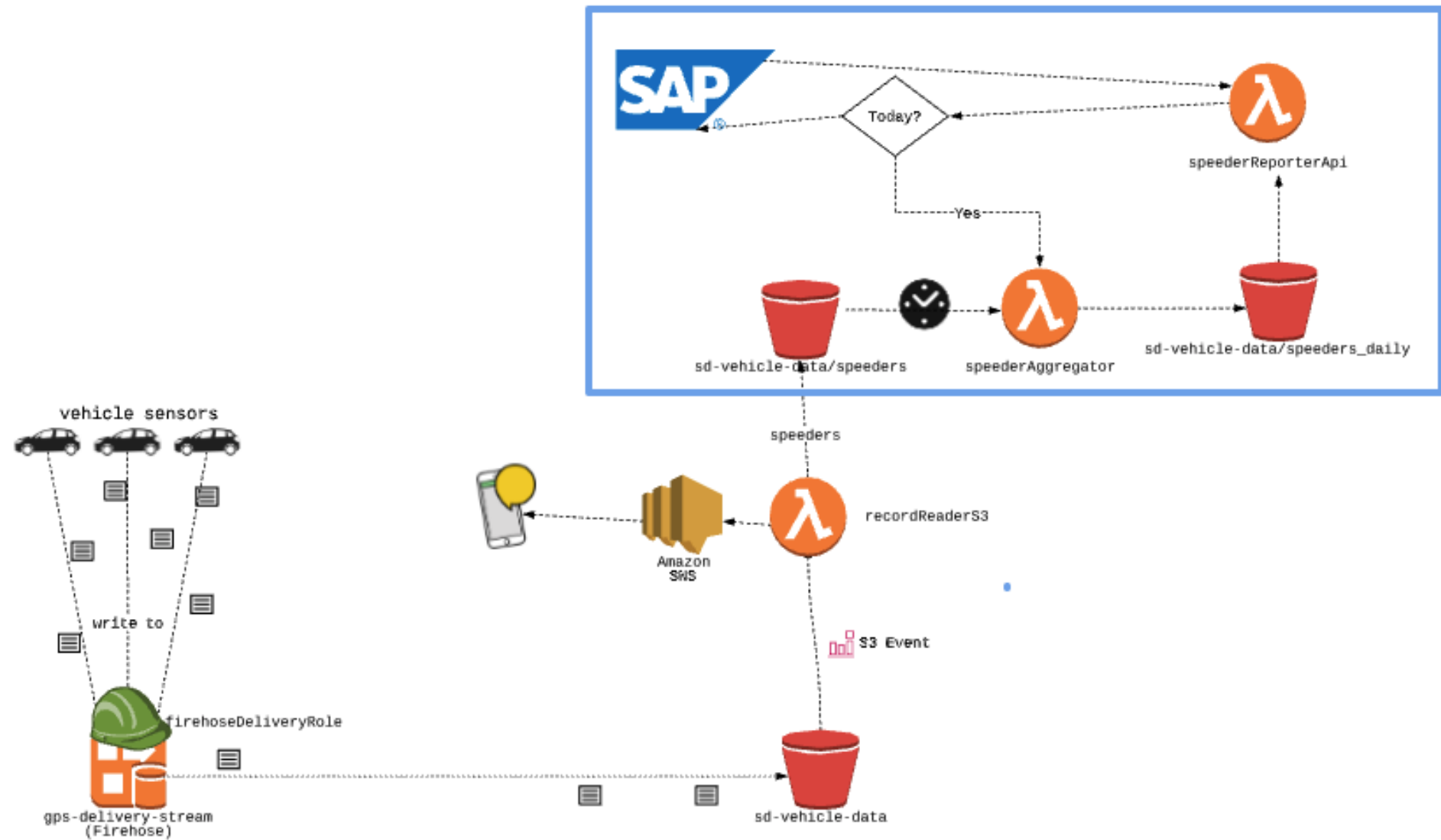
# Review



# Review



# Review





# Let's practice!

STREAMING DATA WITH AWS KINESIS AND LAMBDA