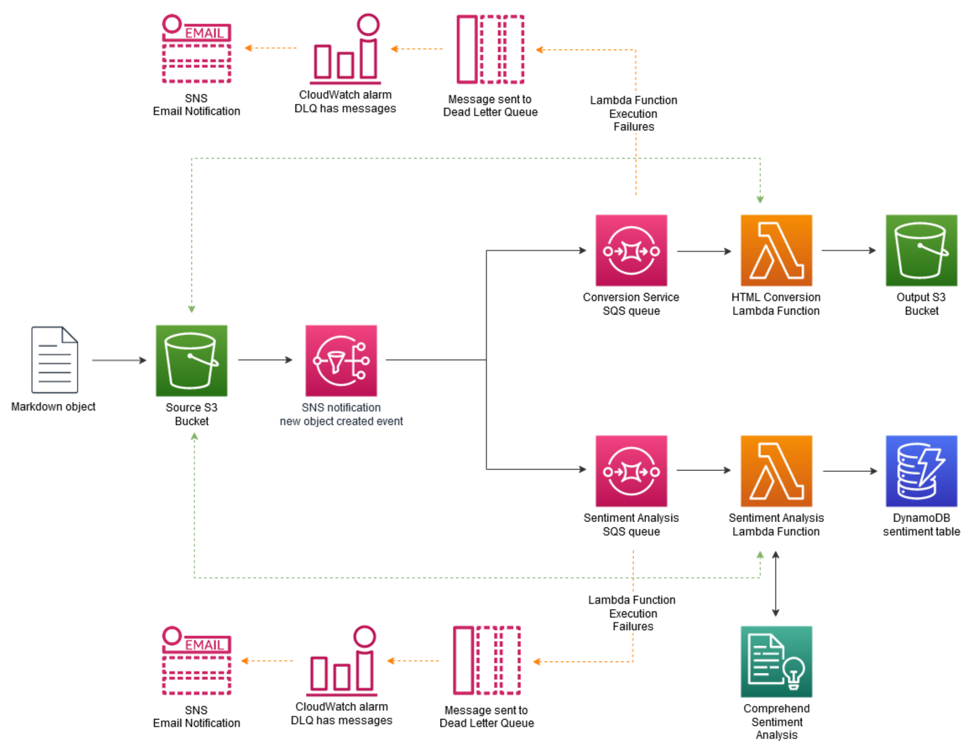
SERVERLESS IOT DATA PROCESSING

Phase 5submission :

The Real-time File Processing reference architecture is a general-purpose, event-driven, parallel data processing architecture that uses AWS Lambda. This architecture is ideal for workloads that need more than one data derivative of an object.

In this example application, we deliver notes from an interview in Markdown format to S3. S3 Events are used to trigger multiple processing flows - one to convert and persist Markdown files to HTML and another to detect and persist sentiment.

Architectural Diagram



Application Components

Event Trigger

In this architecture, individual files are processed as they arrive. To achive this, we utilize AWS S3 Events and Amazon Simple Notification Service. When an object is created in S3, an event is emitted to a SNS topic. We deliver our event to 2 seperate SQS Queues, representing 2 different workflows. Refer to What is Amazon Simple Notification Service? for more information about eligible targets.

Conversion Workflow

Our function will take Markdown files stored in our **InputBucket**, convert them to HTML, and store them in our **OutputBucket**. The **ConversionQueue** SQS queue captures the S3 Event JSON payload, allowing for more control of our **ConversionFunction** and better error handling. Refer to Using AWS Lambda with Amazon SQS for more details.

If our **ConversionFunction** cannot remove the messages from the **ConversionQueue**, they are sent to **ConversionDlq**, a dead-letter queue (DLQ), for inspection. A CloudWatch Alarm is configured to send notification to an email address when there are any messages in the **ConversionDlq**.

Sentiment Analysis Workflow

Our function will take Markdown files stored in our **InputBucket**, detect the overall sentiment for each file, and store the result in our **SentimentTable**.

We are using Amazon Comprehend to detect overall interview sentiment. Amazon Comprehend is a machine learning powered service that makes it easy to find insights and relationships in text. We use the Sentiment Analysis API to understand whether interview responses are positive or negative.

The Sentiment workflow uses the same SQS-to-Lambda Function pattern as the Coversion workflow.

If our **SentimentFunction** cannot remove the messages from the **SentimentQueue**, they are sent to **SentimentDlq**, a dead-letter queue (DLQ), for inspection. A CloudWatch Alarm is configured to send notification to an email address when there are any messages in the **SentimentDlq**.

Building and Deploying the Application with the AWS Serverless Application Model (AWS SAM)

This application is deployed using the AWS Serverless Application Model (AWS SAM). AWS SAM is an open-source framework that enables you to build serverless applications on AWS. It provides you with a template specification to define your serverless application, and a command line interface (CLI) tool.

Pre-requisites

- AWS CLI version 2
- AWS SAM CLI (0.41.0 or higher)
- Docker

Clone the Repository

Clone with SSH

```
git clone git@github.com:aws-samples/lambda-refarch-fileprocessing.git
```

Clone with HTTPS

```
git clone https://github.com/aws-samples/lambda-refarch-fileprocessing.git
```

Build

The AWS SAM CLI comes with abstractions for a number of Lambda runtimes to build your dependencies, and copies the source code into staging folders so that everything is ready to be packaged and deployed. The *sam build* command builds any dependencies that your application has, and copies your application source code to folders under *.aws-sam/build* to be zipped and uploaded to Lambda.

```
sam build --use-container
```

Note

Be sure to use v0.41.0 of the AWS SAM CLI or newer. Failure to use the proper version of the AWS SAM CLI will result in a `InvalidDocumentException` exception. The `EventInvokeConfig` property is not recognized in earlier versions of the AWS SAM CLI. To confirm your version of AWS SAM, run the command `sam --version`.

Deploy

For the first deployment, please run the following command and save the generated configuration file *samconfig.toml*. Please use **lambda-file-refarch** for the stack name.

```
sam deploy --guided
```

You will be prompted to enter data for *ConversionLogLevel* and *SentimentLogLevel*. The default value for each is *INFO* but you can also enter *DEBUG*. You will also be prompted for *AlarmRecipientEmailAddress*.

Subsequent deployments can use the simplified `sam deploy`. The command will use the generated configuration file *samconfig.toml*.

You will receive an email asking you to confirm subscription to the `lambda-file-refarch-AlarmTopic` SNS topic that will receive alerts should either the `ConversionDlq` SQS queue or `SentimentDlq` SQS queue receive messages.

Testing the Example

After you have created the stack using the CloudFormation template, you can manually test the system by uploading a Markdown file to the InputBucket that was created in the stack.

Alternatively you test it by utilising the pipeline tests.sh script, however the test script removes the resources it creates, so if you wish to explore the solution and see the output files and DynamoDB tables manually uploading is the better option.

Manually testing

You can use the any of the sample-xx.md files in the repository /**tests** directory as example files. After the files have been uploaded, you can see the resulting HTML file in the output bucket of your stack. You can also view the CloudWatch logs for each of the functions in order to see the details of their execution.

You can use the following commands to copy a sample file from the provided S3 bucket into the input bucket of your stack.

```
INPUT_BUCKET=$(aws cloudformation describe-stack-resource --stack-name lambda-file-
refarch --logical-resource-id InputBucket --query
"StackResourceDetail.PhysicalResourceId" --output text)
aws s3 cp ./tests/sample-01.md s3://${INPUT_BUCKET}/sample-01.md
aws s3 cp ./tests/sample-02.md s3://${INPUT_BUCKET}/sample-02.md
```

Once the input files has been uploaded to the input bucket, a series of events are put into motion.

1. The input Markdown files are converted and stored in a separate S3 bucket.

```
OUTPUT_BUCKET=$(aws cloudformation describe-stack-resource --stack-name lambda-file-refarch --logical-resource-id ConversionTargetBucket --query "StackResourceDetail.PhysicalResourceId" --output text)
aws s3 ls s3://${OUTPUT_BUCKET}
```

2. The input Markdown files are analyzed and their sentiment published to a DynamoDB table.

```
DYNAMO_TABLE=$(aws cloudformation describe-stack-resource --stack-name lambda-file-refarch --logical-resource-id SentimentTable --query "StackResourceDetail.PhysicalResourceId" --output text)
aws dynamodb scan --table-name ${DYNAMO_TABLE} --query "Items[*]"
```

You can also view the CloudWatch logs generated by the Lambda functions.

Using the test script

The pipeline end to end test script can be manually executed, you will need to ensure you have adequate permissions to perform the test script actions.

- Describing stack resources
- Uploading and deleting files from the S3 input bucket
- Deleting files from the S3 output bucket
- Reading and deleting entries from the DynamoDB table

```
bash ./tests.sh lambda-file-refarch
```

While the script is executing you will see all the stages output to the command line. The samples are uploaded to the **InputBucket**, the script will then wait for files to appear in the **OutputBucket** before checking they have all been processed and the matching html file exists in the **OutputBucket**. It will also check that the sentiment for each of the files has been recorded in the **SentimentTable**. Once complete the script will remove all the files created and the entries from the **SentimentTable**.

Extra credit testing

Try uploading (or adding to ./tests if you are using the script) an oversized (>100MB) or invalid file type to the input bucket. You can check in X-ray to explore how you can trace these kind of errors within the solution.

- Linux command

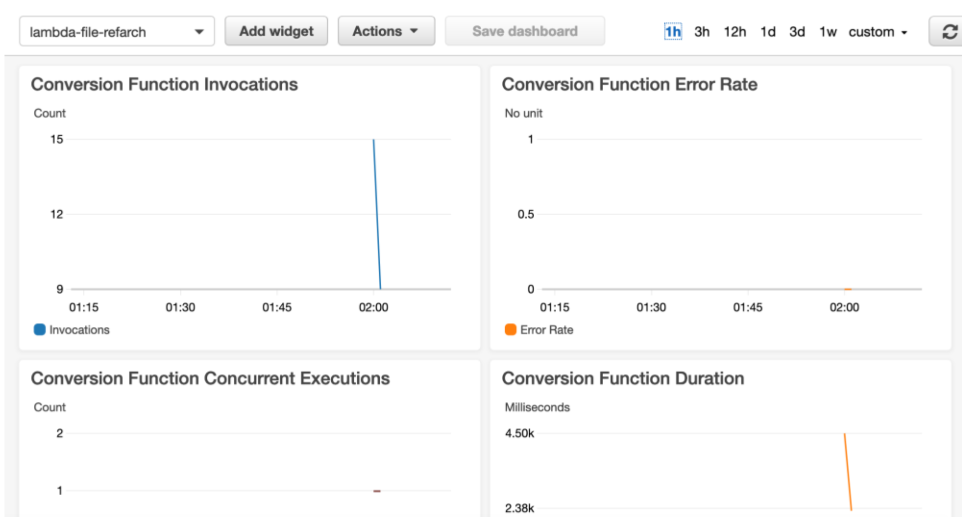
```
fallocate -l 110M ./tests/sample-oversize.md
```

- Mac OS X command

```
mkfile 110m ./tests/sample-oversize.md
```

Viewing the CloudWatch dashboard

A dashboard is created as a part of the stack creation process. Metrics are published for the conversion and sentiment analysis processes. In addition, the alarms and alarm states are published.



Cleaning Up the Example Resources

To remove all resources created by this example, run the following command:

```
bash cleanup.sh
```

What Is Happening in the Script?

Objects are cleared out from the `InputBucket` and `ConversionTargetBucket`.

```
for bucket in InputBucket ConversionTargetBucket; do
  echo "Clearing out ${bucket}..."
  BUCKET=$(aws cloudformation describe-stack-resource --stack-name lambda-file-
refarch --logical-resource-id ${bucket} --query
"StackResourceDetail.PhysicalResourceId" --output text)
  aws s3 rm s3://${BUCKET} --recursive
  echo
done
```

The CloudFormation stack is deleted.

```
aws cloudformation delete-stack \
--stack-name lambda-file-refarch
```

The CloudWatch Logs Groups associated with the Lambda functions are deleted.

```
for log_group in $(aws logs describe-log-groups --log-group-name-prefix
'/aws/lambda/lambda-file-refarch-' --query "logGroups[*].logGroupName" --output
text); do
  echo "Removing log group ${log_group}..."
  aws logs delete-log-group --log-group-name ${log_group}
  echo
done
```

SAM Template Resources

Resources

The provided template creates the following resources:

- **InputBucket** - A S3 bucket that holds the raw Markdown files. Uploading a file to this bucket will trigger processing functions.
- **NotificationTopic** - A SNS topic that receives S3 events from the **InputBucket**.
- **NotificationTopicPolicy** - A SNS topic policy that allows the **InputBucket** to publish events to the **NotificationTopic**.

- **NotificationQueuePolicy** - A SQS queue policy that allows the **NotificationTopic** to publish events to the **ConversionQueue** and **SentimentQueue**.
- **ApplyS3NotificationLambdaFunction** - A Lambda function that adds a S3 bucket notification when objects are created in the **InputBucket**. The function is called by **ApplyInputBucketTrigger**.
- **ApplyInputBucketTrigger** - A CloudFormation Custom Resource that invokes the **ApplyS3NotificationLambdaFunction** when a CloudFormation stack is created.
- **ConversionSubscription** - A SNS subscription that allows the **ConversionQueue** to receive messages from **NotificationTopic**.
- **ConversionQueue** - A SQS queue that is used to store events for conversion from Markdown to HTML.
- **ConversionDlq** - A SQS queue that is used to capture messages that cannot be processed by the **ConversionFunction**. The *RedrivePolicy* on the **ConversionQueue** is used to manage how traffic makes it to this queue.
- **ConversionFunction** - A Lambda function that takes the input file, converts it to HTML, and stores the resulting file to **ConversionTargetBucket**.
- **ConversionTargetBucket** - A S3 bucket that stores the converted HTML.
- **SentimentSubscription** - A SNS subscription that allows the **SentimentQueue** to receive messages from **NotificationTopic**.
- **SentimentQueue** - A SQS queue that is used to store events for sentiment analysis processing.
- **SentimentDlq** - A SQS queue that is used to capture messages that cannot be processed by the **SentimentFunction**. The *RedrivePolicy* on the **SentimentQueue** is used to manage how traffic makes it to this queue.
- **SentimentFunction** - A Lambda function that takes the input file, performs sentiment analysis, and stores the output to the **SentimentTable**.
- **SentimentTable** - A DynamoDB table that stores the input file along with the sentiment.
- **AlarmTopic** - A SNS topic that has an email as a subscriber. This topic is used to receive alarms from the **ConversionDlqAlarm**, **SentimentDlqAlarm**, **ConversionQueueAlarm**, **SentimentQueueAlarm**, **ConversionFunctionErrorRateAlarm**, **SentimentFunctionErrorRateAlarm**, **ConversionFunctionThrottleRateAlarm**, and **SentimentFunctionThrottleRateAlarm**.

- **ConversionDlqAlarm** - A CloudWatch Alarm that detects when there there are any messages sent to the **ConvesionDlq** within a 1 minute period and sends a notification to the **AlarmTopic**.
- **SentimentDlqAlarm** - A CloudWatch Alarm that detects when there there are any messages sent to the **SentimentDlq** within a 1 minute period and sends a notification to the **AlarmTopic**.
- **ConversionQueueAlarm** - A CloudWatch Alarm that detects when there are 20 or more messages in the **ConversionQueue** within a 1 minute period and sends a notification to the **AlarmTopic**.
- **SentimentQueueAlarm** - A CloudWatch Alarm that detects when there are 20 or more messages in the **SentimentQueue** within a 1 minute period and sends a notification to the **AlarmTopic**.
- **ConversionFunctionErrorRateAlarm** - A CloudWatch Alarm that detects when there is an error rate of 5% over a 5 minute period for the **ConversionFunction** and sends a notification to the **AlarmTopic**.
- **SentimentFunctionErrorRateAlarm** - A CloudWatch Alarm that detects when there is an error rate of 5% over a 5 minute period for the **SentimentFunction** and sends a notification to the **AlarmTopic**.
- **ConversionFunctionThrottleRateAlarm** - A CloudWatch Alarm that detects when ther is a throttle rate of 1% over a 5 minute period for the **ConversionFunction** and sends a notification to the **AlarmTopic**.
- **SentimentFunctionThrottleRateAlarm** - A CloudWatch Alarm that detects when ther is a throttle rate of 1% over a 5 minute period for the **SentimentFunction** and sends a notification to the **AlarmTopic**.
- **ApplicationDashboard** - A CloudWatch Dashboard that displays Conversion Function Invocations, Conversion Function Error Rate, Conversion Function Throttle Rate, Conversion DLQ Length, Sentiment Function Invocations, Sentiment Function Error Rate, Sentiment Function Throttle Rate, and Sentiment DLQ Length.

License

Python

```
import json

def process_data(event):
  """This function processes the data received from the IoT
device."""

  # Get the sensor reading from the event data.
  sensor_reading = event["data"]["sensor_reading"]

  # Filter the data based on certain criteria.
  if sensor_reading > 100:
    # Trigger an automated routine.
    send_alert_email()

def send_alert_email():
  """This function sends an alert email."""

  # Create an email message.
  email_message = """
  Subject: Sensor alert!

  The sensor reading has exceeded the threshold.
  """

  # Send the email message.
  send_email(email_message)

# Register the function to be triggered when an event is
received from the IoT device.
IBMCloudFunctions.register_function(process_data,
"iot_device_event")
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import datetime
```

```
%matplotlib inline
import scipy.integrate as integrate
from scipy.optimize import curve_fit
pd.options.display.max_rows = 12
```

```
data = pd.read_csv("../input/data.txt", sep = ' ',header = None, names = ['date',
'time','epoch','moteid','temp','humidity','light','voltage'])
data.head()
```

Out[2]:

	DATE	TIME	EPOCH	MOTEID	TEMP	HUMIDITY	LIGHT	VOLTAGE
0	2004-03-31	03:38:15.757551	2	1.0	122.1530	-3.91901	11.04	2.03397
1	2004-02-28	00:59:16.02785	3	1.0	19.9884	37.09330	45.08	2.69964
2	2004-02-28	01:03:16.33393	11	1.0	19.3024	38.46290	45.08	2.68742
3	2004-02-28	01:06:16.013453	17	1.0	19.1652	38.80390	45.08	2.68742
4	2004-02-28	01:06:46.778088	18	1.0	19.1750	38.83790	45.08	2.69964

```
data.shape
```

Out[3]:

```
(2313682, 8)
```

In [4]:

```
data.describe()
```

data.describe()

Out[4]:

	EPOCH	MOTEID	TEMP	HUMIDITY	LIGHT	VOLTAGE
count	2.313682e+06	2.313156e+06	2.312781e+06	2.312780e+06	2.219804e+06	2.313156e+06

mean	3.303993e+04	2.854412e+01	3.920700e+01	3.390814e+01	4.072110e+02	2.492552e+00
std	1.836852e+04	5.062408e+01	3.741923e+01	1.732152e+01	5.394276e+02	1.795743e-01
min	0.000000e+00	1.000000e+00	-3.840000e+01	-8.983130e+03	0.000000e+00	9.100830e-03
25%	1.757200e+04	1.700000e+01	2.040980e+01	3.187760e+01	3.956000e+01	2.385220e+00
50%	3.332700e+04	2.900000e+01	2.243840e+01	3.928030e+01	1.582400e+02	2.527320e+00
75%	4.778900e+04	4.100000e+01	2.702480e+01	4.358550e+01	5.372800e+02	2.627960e+00
max	6.553500e+04	6.540700e+04	3.855680e+02	1.375120e+02	1.847360e+03	1.856000e+01

```
data.isnull().sum()
```

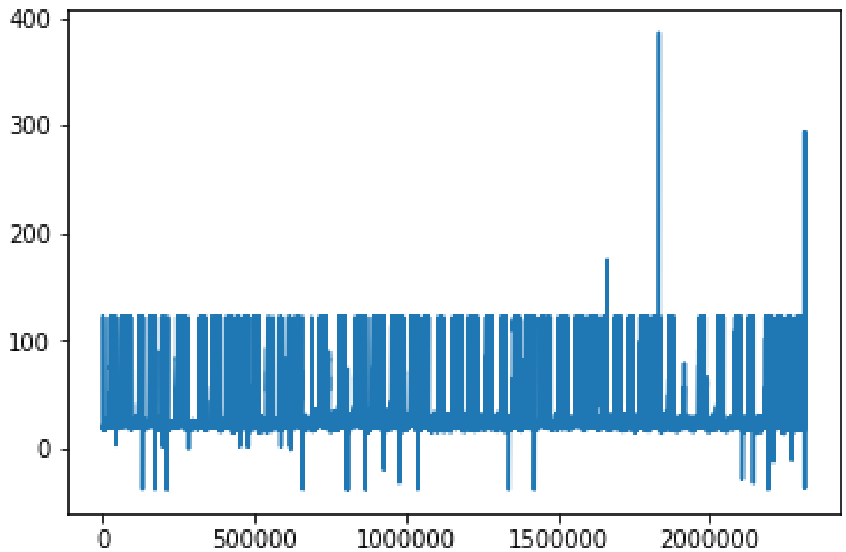
Out[5]:

```
date            0
time            0
epoch           0
moteid        526
temp          901
humidity      902
light       93878
voltage       526
dtype: int64
```

In [6]:

```
data['temp'].plot()
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f83485fb1d0>




```
data.fillna(0, inplace=True)
data['epoch'].replace(regex=True, inplace=True, to_replace=r'[^0-9.\-]',
value=r'')
data['epoch'] = data['epoch'].astype(int)
```

In [8]:

```
data.isnull().sum()
```

date 0
time 0
epoch 0
moteid 0

```
temp        0
humidity    0
light       0
voltage     0
dtype: int64
```

In [9]:

```python
data.groupby('moteid').mean()
```

epoch	temp	humidity	light	voltage	
moteid					
0.0	22031.159696	0.000000	0.000000	0.000000	0.000000
1.0	31994.793830	35.882437	34.319280	156.575828	2.519643
2.0	36491.176404	40.201817	34.298739	212.159717	2.458436
3.0	33013.771411	33.715289	34.787174	146.049674	2.517961
4.0	32688.045167	45.464410	29.991159	155.491512	2.468170
5.0	5836.428571	0.000000	0.000000	0.000000	2.516088
...	...	...	...	...	...
56.0	54586.336657	20.987656	39.261422	104.099945	2.887558
57.0	1499.000000	0.000000	17.887833	0.000000	2.482020
58.0	53830.011773	21.014753	38.541575	364.111968	2.899639
6485.0	35112.000000	262.656000	0.000000	0.000000	0.393324
33117.0	30493.000000	-36.204800	0.000000	0.000000	0.144859
65407.0	18182.000000	294.251000	0.000000	0.000000	0.013056

```
data['Timestamp'] = data[['date', 'time']].apply(lambda x: ' '.join(x.astype(str)), axis=1)
new_data = data
```

In [11]:

```
data.drop(['date','time'],axis=1,inplace =True)
data.set_index(pd.to_datetime(data.Timestamp), inplace=True)
```

In [12]:

```
data[['moteid','temp','humidity','light','voltage']] =
data[['moteid','temp','humidity','light','voltage']].apply(pd.to_numeric)
```

```
data['moteid'].value_counts()
```

Out[13]:

```
31.0       65694
29.0       64391
23.0       62440
26.0       61521
22.0       60165
21.0       58525
           ...
0.0          526
5.0           35
57.0           3
65407.0        1
6485.0         1
33117.0        1
Name: moteid, Length: 62, dtype: int64
```

```
moteid_grp = data.groupby(['moteid'])
```

In [15]:

```
corr_id = moteid_grp.corr(method='pearson')
corr_id.fillna(0, inplace=True)
corr_id
```

epoch	humidity	light	temp	voltage		
moteid						
0.0	epoch	1.0	0.000000	0.000000	0.000000	0.000000
	humidity	0.0	0.000000	0.000000	0.000000	0.000000
	light	0.0	0.000000	0.000000	0.000000	0.000000
	temp	0.0	0.000000	0.000000	0.000000	0.000000
	voltage	0.0	0.000000	0.000000	0.000000	0.000000
1.0	epoch	1.0	-0.195395	0.014605	0.315767	-0.726957
...	...	...	...	...	...	...
33117.0	voltage	0.0	0.000000	0.000000	0.000000	0.000000
65407.0	epoch	0.0	0.000000	0.000000	0.000000	0.000000
	humidity	0.0	0.000000	0.000000	0.000000	0.000000
	light	0.0	0.000000	0.000000	0.000000	0.000000
	temp	0.0	0.000000	0.000000	0.000000	0.000000
	voltage	0.0	0.000000	0.000000	0.000000	0.000000

310 rows × 5 columns

epoch	moteid	temp	humidity	light	voltage	Timestamp	
Timestamp							
2004-03-31 03:38:15.757551	2	1.0	122.1530	-3.91901	11.04	2.03397	2004-03-31 03:38:15.757551
2004-02-28 00:59:16.027850	3	1.0	19.9884	37.09330	45.08	2.69964	2004-02-28 00:59:16.02785
2004-02-28 01:03:16.333930	11	1.0	19.3024	38.46290	45.08	2.68742	2004-02-28 01:03:16.33393

2004-02-28 01:06:16.013453	17	1.0	19.1652	38.80390	45.08	2.68742	2004-02-28 01:06:16.013453
2004-02-28 01:06:46.778088	18	1.0	19.1750	38.83790	45.08	2.69964	2004-02-28 01:06:46.778088

Temperature_data

de22	Node23	Node24	Node25	Node26	Node27	Node28	Node29					
Timestamp	Timestamp	Timestamp	Timestamp									
2004	3	1	0	18.892649	17.888361	18.682962	17.909755	17.859044	17.673663	18.159542	16.981087	17.684646
			1	17.828778	17.513253	18.301935	17.535843	17.494400	17.259217	17.705163	16.576858	17.429180
			2	17.478076	17.269043	18.038685	17.276184	17.222923	16.988568	17.443851	16.263400	17.193869
			3	17.265336	17.036493	17.833192	17.068284	17.011023	16.778655	17.242236	16.098262	17.046103
			4	16.976067	16.728080	17.526390	16.730619	16.708245	16.424560	16.829928	15.745938	16.794936
			5	16.678958	16.295632	17.130163	16.236195	16.223800	15.896133	16.312522	15.345038	16.430650
		...	...	...	...	...	...	...	...	...	...	...
		21	18	21.580820	20.955944	24.125349	32.503942	21.184272	21.099364	22.016777	20.406817	21.141881
			19	21.213658	20.334054	24.385489	35.667856	20.598625	20.551900	21.721577	19.656331	20.547413
			20	20.893717	19.951263	24.994137	40.132790	20.216815	20.256095	21.630513	19.215485	20.144796
			21	20.408241	19.569963	25.837693	45.156295	19.830846	19.937289	21.666446	18.845821	19.723914
			22	20.171971	19.167160	27.153425	51.666562	19.390084	19.724252	21.884614	18.476750	19.241492
			23	19.997640	18.620570	28.617012	59.078113	18.886997	19.474558	22.185723	17.960217	18.716850

```
463 rows × 9 columns
```

```
In [19]:
```

```
Humidity_data
```

Out[19]:

				Node14	Node22	Node23	Node24	Node25	Node26	Node27	Node28	Node29
Timestamp	Timestamp	Timestamp	Timestamp									
2004	3	1	0	42.549285	44.530996	43.168231	44.878085	44.633242	47.021085	45.492716	48.549510	46.209795
			1	45.355191	45.693293	44.364838	46.024384	45.775104	48.405524	46.959683	49.996614	46.909245
			2	46.163976	46.084623	44.710712	46.487651	46.130820	48.831494	47.188011	50.559377	47.271673
			3	46.613439	46.469025	45.070823	46.854633	46.512194	49.220666	47.632574	50.843643	47.565288
			4	47.179816	47.056620	45.602291	47.544280	47.070312	49.981491	48.445950	51.645920	48.041444
			5	47.897721	48.318876	46.838235	49.095143	48.524784	51.741095	50.494748	52.944230	49.184805
		...	...	...	...	...	...	...	...	...	...	...
		21	18	48.074626	47.581688	46.317189	48.043332	45.830167	48.557737	46.804405	48.878659	46.799120
			19	48.199534	48.236579	46.810787	48.842767	46.393657	49.506822	47.598095	50.102531	47.288588
			20	48.285621	48.535250	47.076702	49.253592	46.754700	49.895563	48.139502	50.566962	47.463996
			21	49.033441	48.811621	47.416511	49.718356	47.142598	50.570302	48.718429	50.572297	47.832622
			22	49.483676	49.228436	47.773140	50.317610	47.654925	51.163117	49.398291	50.813785	48.433595
			23	49.850957	50.196400	48.584533	51.448840	48.664630	52.343430	50.653467	51.896545	49.429506

Covariance models in time for the temperature

Temperature time lags from 0 to 9

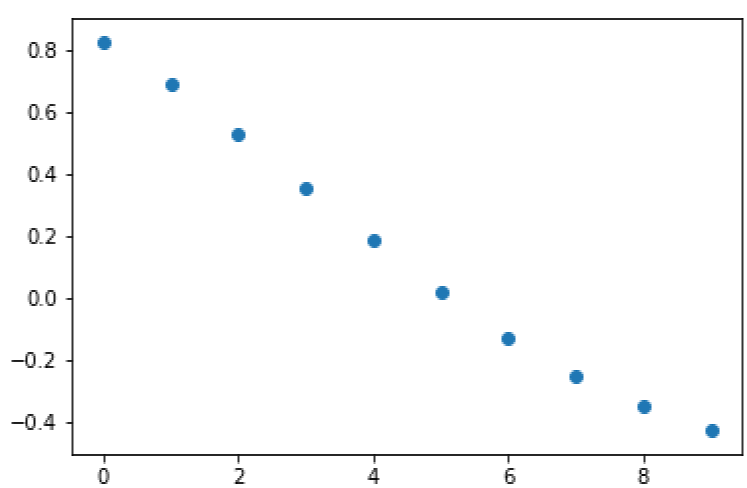
In [20]:

```
node_distance = pd.read_csv('../input/intel_nodes_distances.csv')

lists_hy = []
```

```
lists_hx = []

for z in range(10):
    # Calculation of the covariance for lag i hour for temp
    humy = [None] *45
    humx = [None] *45
    k=0
    i=0
    for first_column in Temperature_data:
        df1 = Temperature_data[first_column][:-(1+z)]
        std_test1=np.std(df1)
        j = 0
        for second_column in Temperature_data:
            if j <= i:
                df2 = Temperature_data[second_column][(1+z):]
                std_test2=np.std(df2)
                humy[k] = (np.cov(df1,df2)/(std_test1*std_test2)).item((0, 1))
                humx[k] = node_distance.iloc[i,j]
                j = j + 1
                k = k + 1
        i = i + 1
    lists_hy.append(humy)
    lists_hx.append(humx)
```

Out[21]:

<matplotlib.collections.PathCollection at 0x7f8327bc9550>



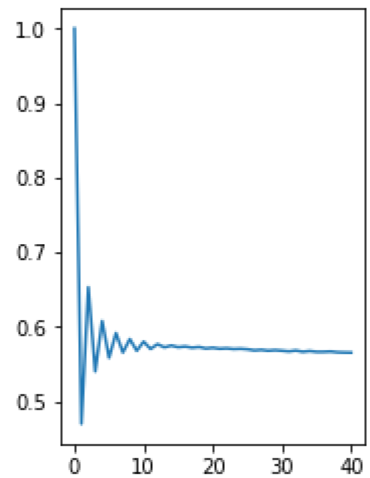
In [22]:

```
from statsmodels.tsa.stattools import acf,pacf
lag_acf = acf(data['temp'])
```

#Plot pACF: a
plt.subplot(121)
plt.plot(lag_acf)

Out[22]:

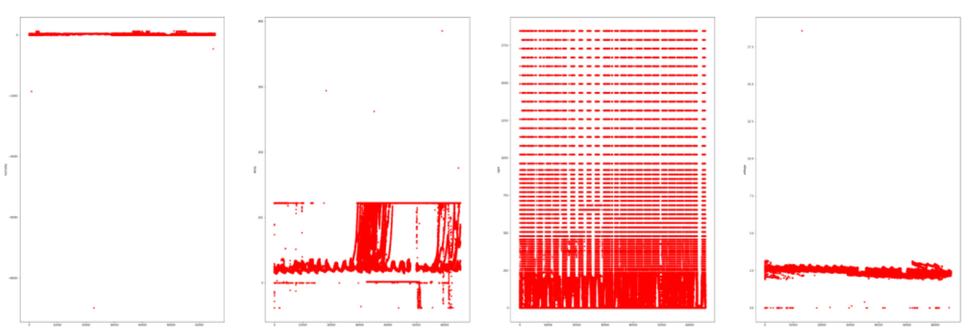
```
[<matplotlib.lines.Line2D at 0x7f8326a94470>]
```



Variation in humidity, temp, light, voltage with epoch

In [23]:

```
fig, axes = plt.subplots(nrows=1, ncols=4, figsize=(60,20))

for xcol, ax in zip(['humidity', 'temp', 'light','voltage'], axes):
    data.plot(kind='scatter', x='epoch', y=xcol, ax=ax, alpha=1, color='r')
```

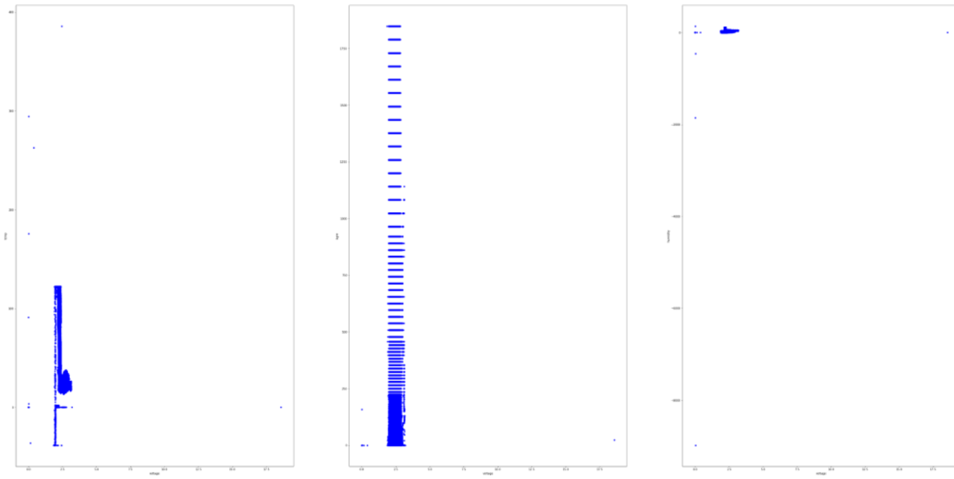


Variation with Voltage

In [24]:

```
fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(60,30))
for xcol, ax in zip(['temp', 'light','humidity'], axes):
```

```
    data.plot(kind='scatter', x='voltage', y=xcol, ax=ax, color='b')
```



Variation with light

In [25]:

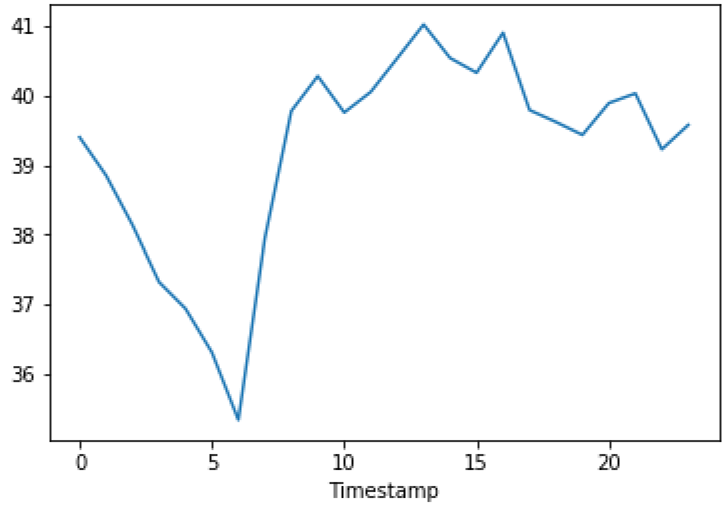
```
fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(60,30))
for xcol, ax in zip(['temp', 'humidity','voltage'], axes):
    data.plot(kind='scatter', x='light', y=xcol, ax=ax, alpha=1, color='g')
```

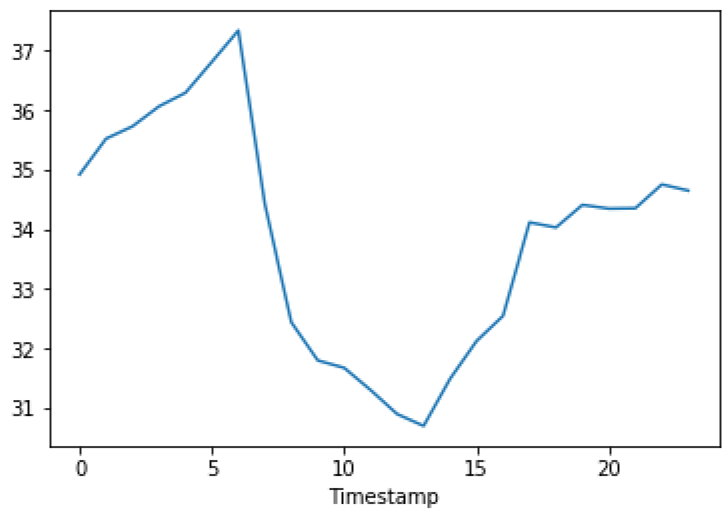
Variation with Humidity

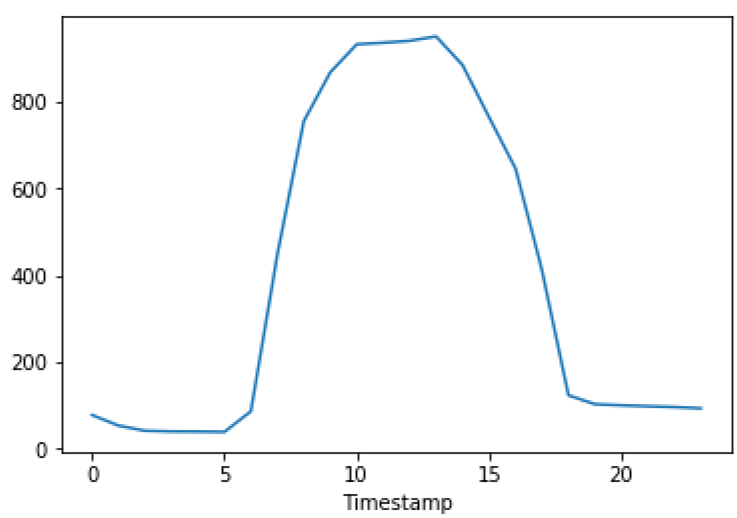
In [26]:

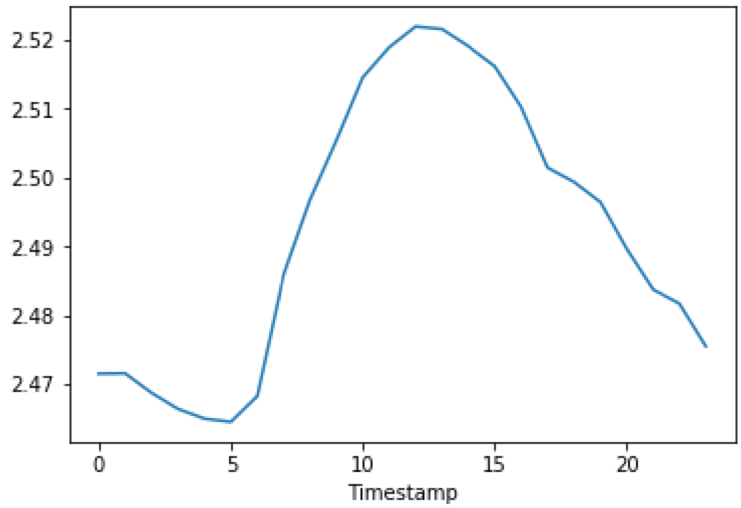
```
fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(60,30))
for xcol, ax in zip(['temp', 'light','voltage'], axes):
    data.plot(kind='scatter', x='humidity', y=xcol, ax=ax, alpha=1,
```

`color='y'` Variation in Tempeature, Humidity, light and Voltage over time



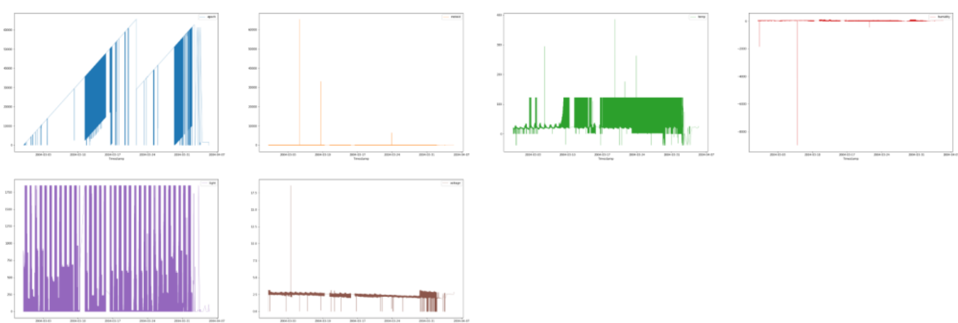






In [29]:

```
new_data.plot(subplots=True,linewidth=0.5,
                layout=(2, 4),figsize=(60, 20),
                sharex=False,
                sharey=False)

plt.show()
```



Pearson Correlation for the multivariate time series

In [30]:

```
new_data.corr(method='pearson')
```

Out[30]:

	EPOCH	MOTEID	TEMP	HUMIDITY	LIGHT	VOLTAGE
epoch	1.000000	0.006596	0.340746	-0.221278	0.041511	-0.654189

moteid	0.006596	1.000000	-0.014062	0.023541	0.029538	0.001789
temp	0.340746	-0.014062	1.000000	-0.707251	0.021948	-0.737583
humidity	-0.221278	0.023541	-0.707251	1.000000	-0.095084	0.507445
light	0.041511	0.029538	0.021948	-0.095084	1.000000	0.055835
voltage	-0.654189	0.001789	-0.737583	0.507445	0.055835	1.000000

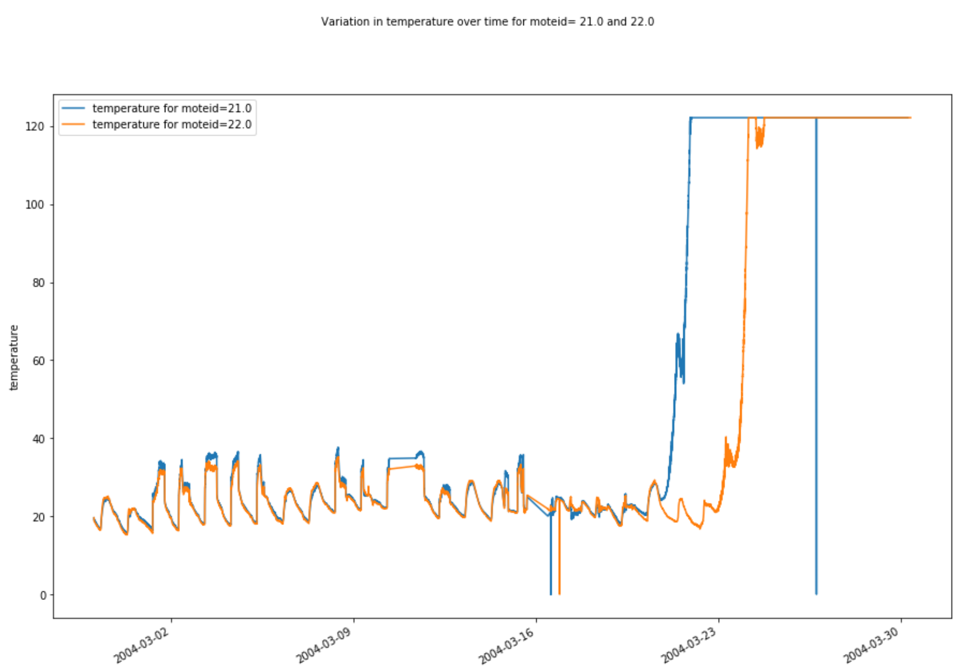
Variation in temperature readings over time for moteid's: 21 and 22

In [31]:

```
from matplotlib import pyplot as plt
d_m21 = data.loc[data['moteid'] == 21.0]
d_m22 = data.loc[data['moteid'] == 22.0]
d_m10 = data.loc[data['moteid'] == 10.0]
fig2 = plt.figure(figsize = (15,10))
d_m21['temp'].plot(label='temperature for moteid=21.0')
d_m22['temp'].plot(label='temperature for moteid=22.0')
fig2.suptitle('Variation in temperature over time for moteid= 21.0 and 22.0',
fontsize=10)
plt.xlabel('timestamp', fontsize=10)
plt.ylabel('temperature', fontsize=10)
plt.legend()
```

Out[31]:

<matplotlib.legend.Legend at 0x7f8327b78ba8>



Anomaly Detection using moving average method

For moteid:10 and window size: 20, we calculate the mean and standard deviation of the data.If the next entry in the dataframe lies between mean(+-)sd*2, it is considered normal else it is considered an anamoly.

Anomaly can be seen by blue *

In [32]:

```
from itertools import count
import matplotlib.pyplot as plt
from numpy import linspace, loadtxt, ones, convolve
import numpy as np
import pandas as pd
import collections
from random import randint
from matplotlib import style
%matplotlib inline
def mov_average(data, window_size):

    window = np.ones(int(window_size))/float(window_size)
    return np.convolve(data, window, 'same')
def find_anomalies(y, window_size, sigma=1.0):
    avg = mov_average(y, window_size).tolist()
    residual = y - avg
    std = np.std(residual)
    return {'standard_deviation': round(std, 3),
            'anomalies_dict': collections.OrderedDict([(index, y_i) for index,
y_i, avg_i in zip(count(), y, avg)
              if (y_i > avg_i + (sigma*std)) | (y_i < avg_i - (sigma*std))])}
def plot_results(x, y, window_size, sigma_value=1,
                 text_xlabel="X Axis", text_ylabel="Y Axis",
applying_rolling_std=False):

    plt.figure(figsize=(15, 8))
    plt.plot(x, y, "k.")
    y_av = mov_average(y, window_size)
    plt.plot(x, y_av, color='green')
    plt.xlim(0, 40000)
    plt.xlabel(text_xlabel)
    plt.ylabel(text_ylabel)
    events = {}
    events = find_anomalies(y, window_size=window_size, sigma=sigma_value)
```

```
    x_anom = np.fromiter(events['anomalies_dict'].keys(), dtype=int,
count=len(events['anomalies_dict']))
    y_anom = np.fromiter(events['anomalies_dict'].values(),
dtype=float,count=len(events['anomalies_dict']))
    plt.plot(x_anom, y_anom, "b*")
    print(x_anom)
    plt.grid(True)
    plt.show()
x = d_m10['epoch']
Y = d_m10['temp']
plot_results(x, y=Y, window_size=50, text_xlabel="Date",
sigma_value=3,text_ylabel="temperature")
```

[23743 23751 23761 23999 24206 24302 24303 24324 24350 25415 26094 26101
 26325 26336 26371 26422 26437 26549 26551 26581 26588 26622 26624 26636
 26713 26723 29505 29506 29507 36012 36017 40733 40748 40775 40779 40787
 40840 40841 40848 40982 41115 41201]