ChordFileSystem Report

System components:

1 Uploadserver

Run on port 5058

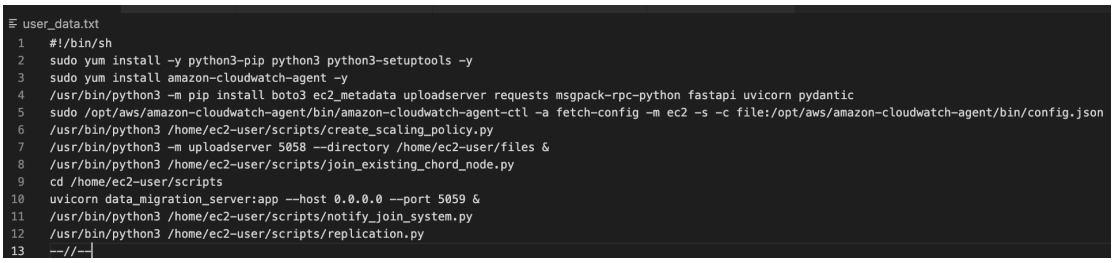
2 Chord

Run on port 5057

3 Data migration server

Run on port 5059

4 User data (Something happen after my instance start)



4.1 First install python and amazon-cloudwatch -agent

4.2 Use pip install some python package needed following steps

4.3 Start the CloudWatch agent and create the metric by my config file (/opt/aws/amazon-cloudwatch-agent/bin/config.json)

這個 metric 之後可用 instance_id、所屬 ASG 來找到、且會監聽 /

目錄下的 disk 用量百分比

4.4 接著會執行 create_scaling_policy.py

```python
# Create CloudWatchWrapper
cloudwatchClient = boto3.client('cloudwatch', region_name='us-east-1')
cloudwatchResource = boto3.resource('cloudwatch', region_name='us-east-1')
cloudWatchWrapper = CloudWatchWrapper(cloudwatchResource)

# Create Autoscaling Client
autoscalingClient = boto3.client('autoscaling', region_name='us-east-1')
autoscalingWrapper = AutoScalingWrapper(autoscalingClient)

policy = autoscalingWrapper.put_scaling_policy(
    "MyChordFileSystemASG", "disk_used_persent")
print(policy)


exist_instanceIDs = autoscalingWrapper.getAutoScalingGroupInstanceIDs(
    "MyChordFileSystemASG")
print("instanceIDs: ", exist_instanceIDs)
```

首先找到MyChordSystemASG 這個 ASG 並用 put_scaling_policy

建立 policy。（之後會用到該 policy 的 ARN 來 attach alarm ）

接著，每個在 ASG 中的 instamce 都有一個前一步驟建立的 metric 來監聽，所以我們需要拿到 ASG 中所有的 instance_id 來取得這些 metric。

```
metric_querys = []
for i in range(len(exist_instanceIDs)):
    metric_querys.append(
        {
            "Id": f'm_{i}',
            "MetricStat": {
                "Metric": {
                    "Namespace": "CWAgent",
                    "MetricName": "disk_used_percent",
                    "Dimensions": [
                        {
                            'Name': 'AutoScalingGroupName',
                            'Value': 'MyChordFileSystemASG'
                        },
                        {
                            "Name": "InstanceId",
                            "Value": exist_instanceIDs[i]
                        },
                        {
                            'Name': 'device',
                            'Value': 'xvda1'
                        },
                        {
                            'Name': 'fstype',
                            'Value': 'xfs'
                        },
                        {
                            'Name': 'path',
                            'Value': '/'
                        },
                    ]
                },
                "Period": 60,
                "Stat": "Average",
                "Unit": "Percent"
            },
            "ReturnData": False
        }
    )
expression_querys = [{
    'Id': 'expr_1',
    'Expression': 'AVG(METRICS())',
    'ReturnData': True
}]


metrics = metric_querys + expression_querys
print("metrics: ", metrics)
```

接著我們用剛剛得到的 instance_id 找到對應的 metric 放入 metric_query 中，也定義我們的 expression_querys 來算所有 metric 的回傳值平均。並把他們合併變成最終的 metrics 參數（put_metric_alarm 會用到）。

```
tryCnt = 0
while (True):
    list_metrics_result = list(cloudWatchWrapper.list_metrics(
        "CWAgent", "disk_used_percent", ec2_metadata.instance_id))
    if len(list_metrics_result) == 0:
        tryCnt += 1
        print("tryCnt: ", tryCnt)
        time.sleep(20)
    else:
        print("list_metrics_result len: ", len(list_metrics_result))
        disk_used_percent_metric = list_metrics_result[0]
        print("disk_used_percent_metric: ", disk_used_percent_metric)
        alarm = cloudwatchClient.put_metric_alarm(AlarmName='MyChordFileSystemASGDiskUsedAlarm',
                                                  AlarmActions=[
                                                      policy['PolicyARN']],
                                                  Metrics=metrics,
                                                  EvaluationPeriods=2,
                                                  Threshold=30,
                                                  ComparisonOperator='GreaterThanThreshold')
        print("alarm: ", alarm)
        break

```

因為前一步驟建立 cloudwatch metric 會需要一些時間，我這裡跑 while 迴圈一直去用 instance_id 來 list metrics 看看該 instance 是否成功建立的 metric。

直到我成功建立 metric 後，我會用上個步驟的 metrics 參數和一開始拿到的 policy ARN 來建立 alarm。

4.5 接著執行/usr/bin/python3 -m uploadserver 5058 --directory /home/ec2-user/files &

把 upload server 跑起來在 5058 port 且之後上傳的檔案都會放到 /home/ec2-user/files 這個目錄

4.6 接著執行 join_existing_chord_node.py

```
autoscalingClient = boto3.client('autoscaling', region_name='us-east-1')
autoScalingWrapper = AutoScalingWrapper(autoscalingClient)
exist_instanceIDs = autoScalingWrapper.getAutoScalingGroupInstanceIDs(
    "MyChordFileSystemASG")
print("instanceIDs: ", exist_instanceIDs)

# Start Chord node
os.system("/home/ec2-user/scripts/chord {} 5057 &".format(ec2_metadata.public_ipv4))
print("Chord Node start listen in {}:5057".format(ec2_metadata.public_ipv4))
time.sleep(5)
# Join existing Chord system
if len(exist_instanceIDs) == 1:
    create(ec2_metadata.public_ipv4)
else:
    if getPublicIP(exist_instanceIDs[0]) == ec2_metadata.public_ipv4:
        join(ec2_metadata.public_ipv4, getPublicIP(exist_instanceIDs[1]))
    else:
        join(ec2_metadata.public_ipv4, getPublicIP(exist_instanceIDs[0]))
```

首先取得所有在 MyChordFileSystemASG 裡的 instanceID，並把助教提供的 chord 執行檔跑起來在 5057 並使用自己的 public_ip

因為 chord 跑起來要時間所以等 5 秒

接著若自己是 ASG 中第一個 instance，則用自己的 public_ip create chord system

若已經有其他 instance 在 ASG 中，則用他的 public_ip 將自己的 public_ip join 進 chord system。

4.7 接著將 data migration server 跑起來

```python
@app.post("/notify_join_system", status_code=HTTPStatus.OK)
async def upload_file_to_predecessor(predecessor: Node):
    source_path = '/home/ec2-user/files/'

    files_to_upload = []

    for f in listdir(source_path):
        file_owner_ip = get_file_owner_ip(ec2_metadata.public_ipv4, f)
        print("file_owner_ip: ", file_owner_ip)
        if file_owner_ip == predecessor.ip:
            files_to_upload.append(f)

    for f in files_to_upload:
        files = {
            'files': open(source_path + f, 'rb'),
        }
        print("Uploading file to http://{}".format(predecessor.ip))
        requests.post(
            'http://{}:5058/upload'.format(predecessor.ip), files=files)

    return {"files": files_to_upload}
```

我用 fastapi 簡單建一個 server，裡面有一隻 api 叫做 /notify_join_system，當有新 instance join ASG 時他會對他的 successor 打這個 API 並將自己的 Node 資訊(ip 以及 chord id)傳給 successor。

這裡收到訊息後用會掃描所有存放在該 instance 中的檔案，把新 instance 應該負責的檔案 upload 給新的 instance。

```python
def get_file_owner_ip(ip, file_name):
    h = hash(file_name)
    client = new_client(ip, 5057)
    file_onwer = client.call("find_successor", h)
    print("get file {} onwer.".format(file_name))
    return file_onwer[0].decode()
```

4.8 接著執行 notify_join_system

```python
time.sleep(20)

mynode = get_info(ec2_metadata.public_ipv4)
successor = get_successor(ec2_metadata.public_ipv4, 0)
successor_ip = successor[0].decode()
print("successor: ", successor)
node = {
    'ip': mynode[0].decode(),
    'port': mynode[1],
    'id': mynode[2]
}
print('http://{}:5059/notify_join_system"'.format(successor_ip))
print("node: ", node)
response = requests.post(
    'http://{}:5059/notify_join_system'.format(successor[0].decode()), json=node)

print(response.content)

```

這就是上一步驟說當新 instance 加入 ASG 時會執行的程式，call successor 的/nptify_join_system api，和他要應該存在自己這裡的 file。因為新 instance join chord system 需要時間 stabalize 所以等 20 秒再要資料。

4.9 接著執行 replication.py

```python
files_set = set()
old_files_set_size = 0
neighbor = Neighbor(predecessor_ip='', first_successor_ip='',
                    second_successor_ip='')
while True:
    time.sleep(8)
    print("start replication")
    neighbor_same = check_neighbor_same()
    replication(neighbor_same)
```

首先會用 files_set 來記錄已經被 check 過的 file，用 old_files_set_size 來紀錄已經被 check 過的 file 的數量。

另外還需要紀錄 Neighbor 的 ip 資訊(predecessor、first_successor、second_successor)

每隔 8 秒會 check 一次 neighbor 是否有變動，並且檢查是否要 replication。

```python
def check_neighbor_same():
    global neighbor
    print("check_neighbor_same")
    new_predecessor_ip = get_predecessor_ip(ec2_metadata.public_ipv4)
    new_first_successor_ip = get_successor_ip(ec2_metadata.public_ipv4, 0)
    new_second_successor_ip = get_successor_ip(ec2_metadata.public_ipv4, 1)
    alive = ((neighbor.predecessor_ip == new_predecessor_ip) & (neighbor.first_successor_ip ==
             new_first_successor_ip) & (neighbor.second_successor_ip == new_second_successor_ip))
    neighbor.predecessor_ip = new_predecessor_ip
    neighbor.first_successor_ip = new_first_successor_ip
    neighbor.second_successor_ip = new_second_successor_ip
    return alive
```

Check_neighbor_same 會去用 rpc call 來檢查自己的 neighbor 是否有變，並更新 neighbor 資訊。

```python
def replication(neighbor_same: bool):
    global old_files_set_size
    global files_set
    source_path = '/home/ec2-user/files/'
    files = [f for f in listdir(source_path)]
    print("old_files_set_size: ", old_files_set_size)
    print("new_files_set_size: ", len(files))
    need_check_files = []
    if (len(files) == old_files_set_size) & neighbor_same:
        return
    elif neighbor_same:
        need_check_files = [f for f in files if f not in files_set]
    else:
        need_check_files = files

    for f in need_check_files:
        files_set.add(f)
        file_owner_ip = get_file_owner_ip(ec2_metadata.public_ipv4, f)
        print("file_owner_ip: ", file_owner_ip)
        if file_owner_ip == ec2_metadata.public_ipv4:
            files = {
                'files': open(source_path + f, 'rb'),
            }

            response = requests.get(
                "http://{}:5058/{}".format(neighbor.first_successor_ip, f))
            if (response.status_code != 200):
                print("Uploading file to http://{}".format(neighbor.first_successor_ip))
                requests.post(
                    'http://{}:5058/upload'.format(neighbor.first_successor_ip), files=files)
            response = requests.get(
                "http://{}:5058/{}".format(neighbor.second_successor_ip, f))
            if (response.status_code != 200):
                print("Uploading file to http://{}".format(neighbor.second_successor_ip))
                requests.post(
                    'http://{}:5058/upload'.format(neighbor.second_successor_ip), files=files)
    old_files_set_size = len(files_set)
```

Replication 中首先判斷 instance 中是否有新的 file，若沒有新 file 且 neighbor 資訊沒變，就什麼都不用做。

若 neighbor 沒變，但有新 file 則只需檢查新的 files

若 neighbor 資訊變了則全部 file 都需要檢查

接著將需要檢查的 files 都加進 files-set 中，變看看這個 file 是不是自己要負責 replication，如果是，則檢查後兩個 successor 有沒有這個 file 資料，如果沒有救 upload 給他們。

最後更新 old_files-set_size 資訊

System functionalities:

1 Data Migration：

詳細步驟請看 System components 的 4.7, 4.8。

2 File Chunks:

這裡主要在 client 這裡處理。分 upload.py 和 download.py 討論。

2.1 upload.py

```python
chunk_size = 4000  # 4KB

# for i in range(1, 20):
with open(filepath, 'rb') as f:
    chunk_num = 1
    while True:
        # chunk_stream = io.BytesIO()
        chunk_data = f.read(chunk_size)
        if not chunk_data:
            break
        chunk_file_name = '{}_chunk_{}'.format(filename, chunk_num)
        h = hash(chunk_file_name)
        print("Hash of {} is {}".format(chunk_file_name, h))
        node = client.call("find_successor", h)
        node_ip = node[0].decode()
        chunk_stream = io.BytesIO(chunk_data)
        chunk_stream.name = chunk_file_name
        files = {
            'files': io.BufferedReader(chunk_stream),
        }
        print(files)
        print("Uploading file to http://{}".format(node_ip))
        response = requests.post(
            'http://{}:5058/upload'.format(node_ip), files=files)
        chunk_num += 1
```

首先定義 chunk_size 為 4KB。

接著每次會讀 chunk_size 的檔案並將檔名命名為 filename_chunk_i

(filename 為原始檔名，i 為 index)

把他 hash 後用 find_successor 找到要 upload 的 ip 做 upload。

直到讀不到資料後結束迴圈。

2.2 download.py

```python
chunk_num = 1
with open(output_file_name, 'wb') as f:
    while True:
        chunk_file_name = output_file_name + '_chunk_' + str(chunk_num)
        h = hash(chunk_file_name)
        print("Hash of {} is {}".format(chunk_file_name, h))
        node = client.call("find_successor", h)
        node_ip = node[0].decode()
        print("Downloading file from http://{}".format(node_ip))
        response = requests.get("http://{}:5058/{}".format(node_ip, chunk_file_name))
        print(response)
        if(response.status_code != 200):
            break
        f.write(response.content)
        chunk_num += 1
```

Download 時則按照 chunk id 一個一個下載直到下載時找不到檔案

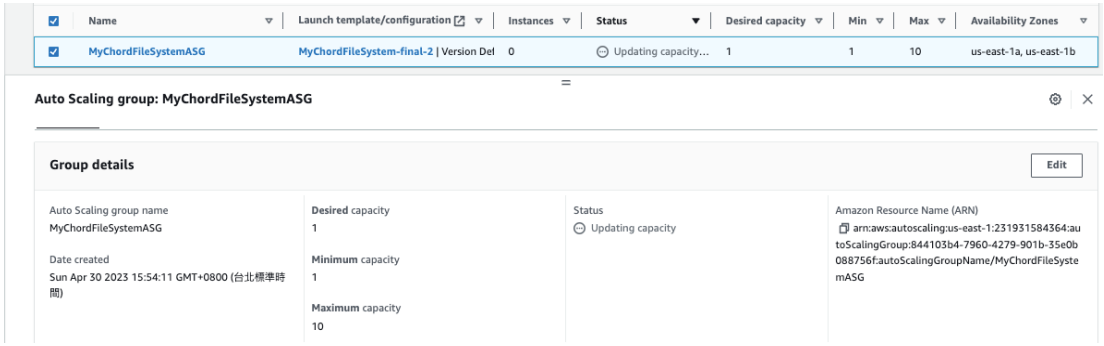
每次載完會把下載到的結果寫入下來。

3 Replication

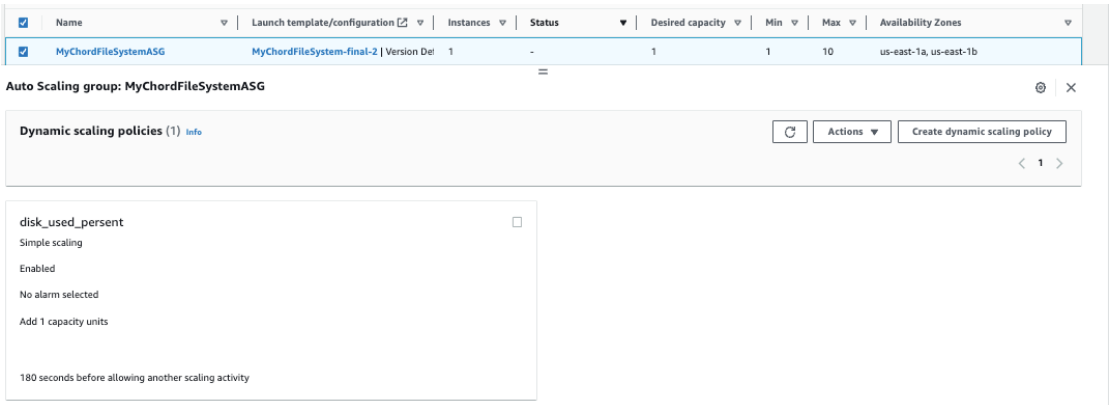
詳細步驟請看 System components 的 4.9。

Experiment:

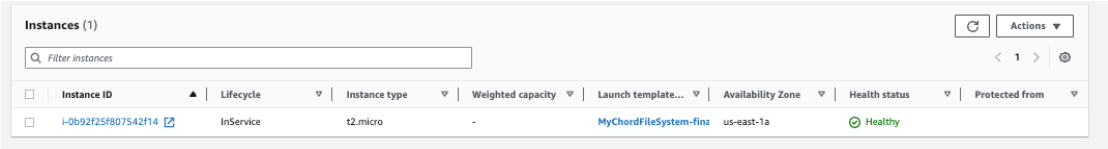
首先先建立一個 ASG 把 Desired capacity 設為 1。

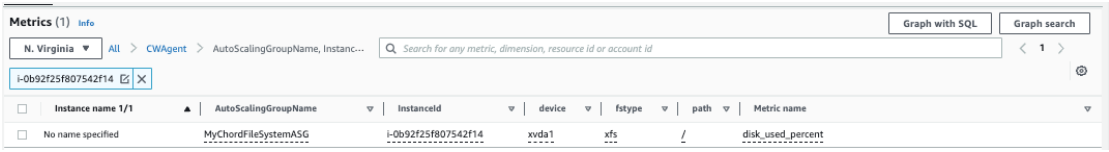


過陣子後發現 scaling policies 有新的 policy 出現

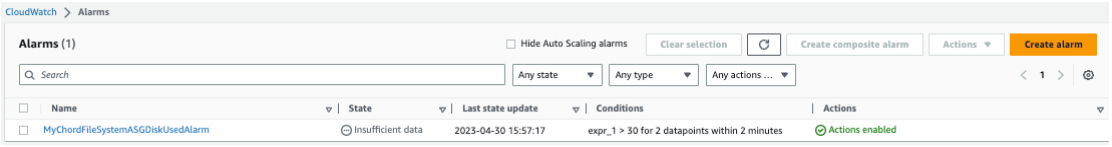


此時去 cloudwatch 找對應 instance 的 metric

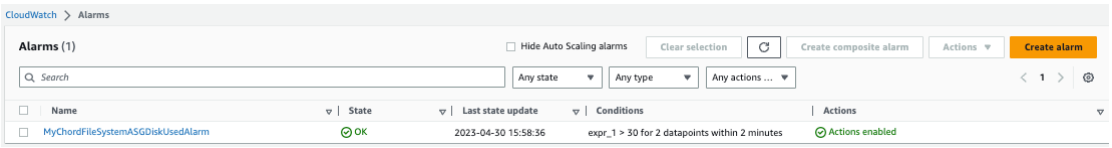




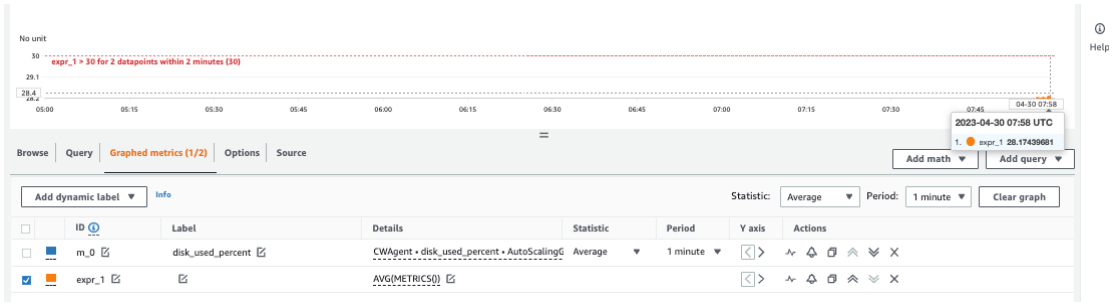
發現 alarm 也出現了



過陣子狀態變 ok



可以看到 metric 中現在有一個 metric 和一個 expression



連線進 ec2 看看發現 files 目錄下沒東西

```
[ec2-user@ip-172-31-14-115 files]$ ls
[ec2-user@ip-172-31-14-115 files]$
```

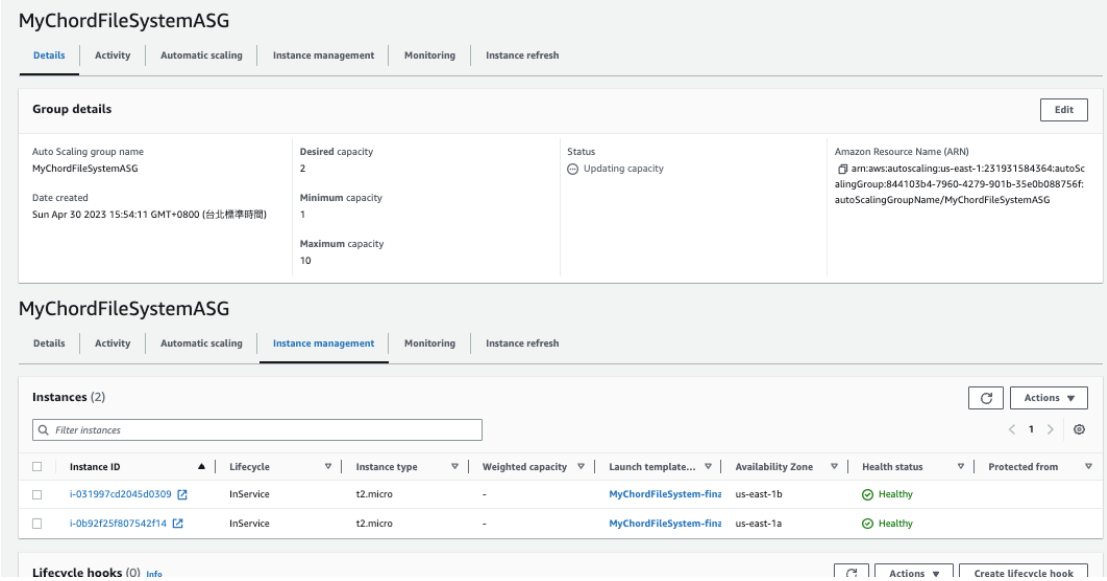
從 client 用 upload.py 上傳幾個檔案試試

```
> cd uploadserver
> python3 upload.py b.txt 3.215.184.16
filename:  b.txt
Hash of b.txt_chunk_1 is 3239240481
{'files': <_io.BufferedReader name='b.txt_chunk_1'>}
Uploading file to http://3.215.184.16
> python3 upload.py c.txt 3.215.184.16
filename:  c.txt
Hash of c.txt_chunk_1 is 415054527
{'files': <_io.BufferedReader name='c.txt_chunk_1'>}
Uploading file to http://3.215.184.16
> vim d.txt
> python3 upload.py d.txt 3.215.184.16
filename:  d.txt
Hash of d.txt_chunk_1 is 4055362934
{'files': <_io.BufferedReader name='d.txt_chunk_1'>}
Uploading file to http://3.215.184.16
```

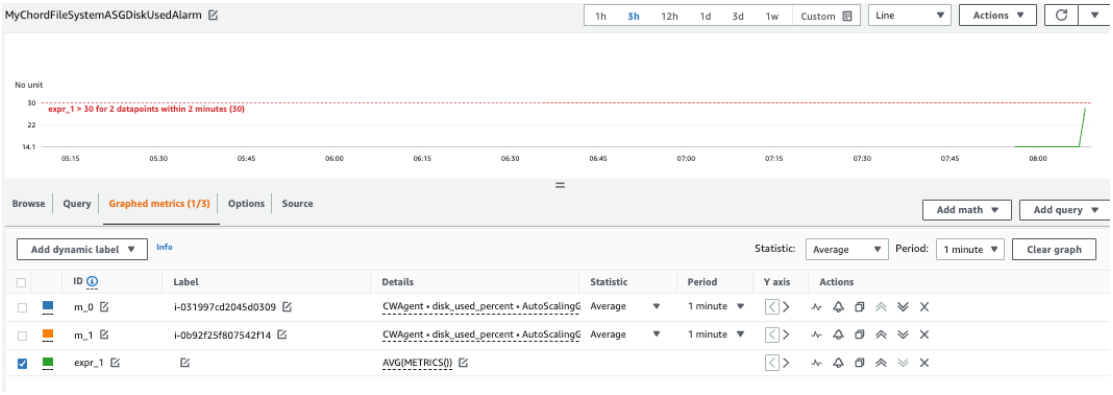
結果如下：

```
[ec2-user@ip-172-31-14-115 files]$ ls
[ec2-user@ip-172-31-14-115 files]$ ls
b.txt_chunk_1  c.txt_chunk_1  d.txt_chunk_1
[ec2-user@ip-172-31-14-115 files]$
```

此時把 ASG Desired capacity 設為 2



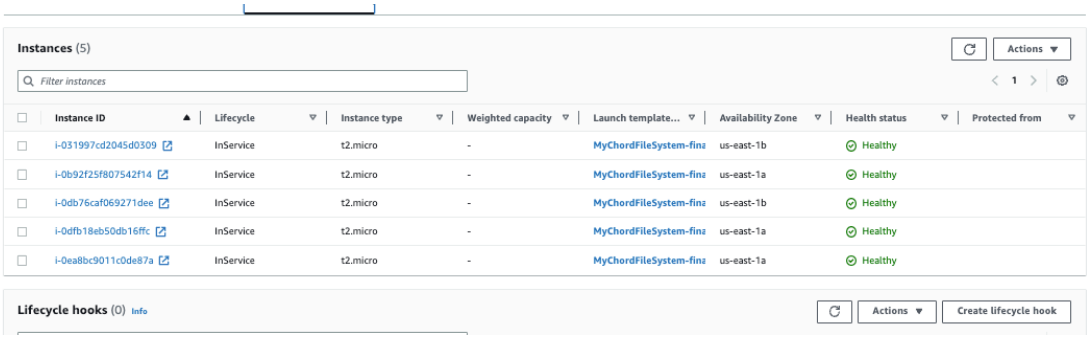
去 cloudwatch 看 alarm 的詳細資訊，發現有兩個 metric 和一個 expression



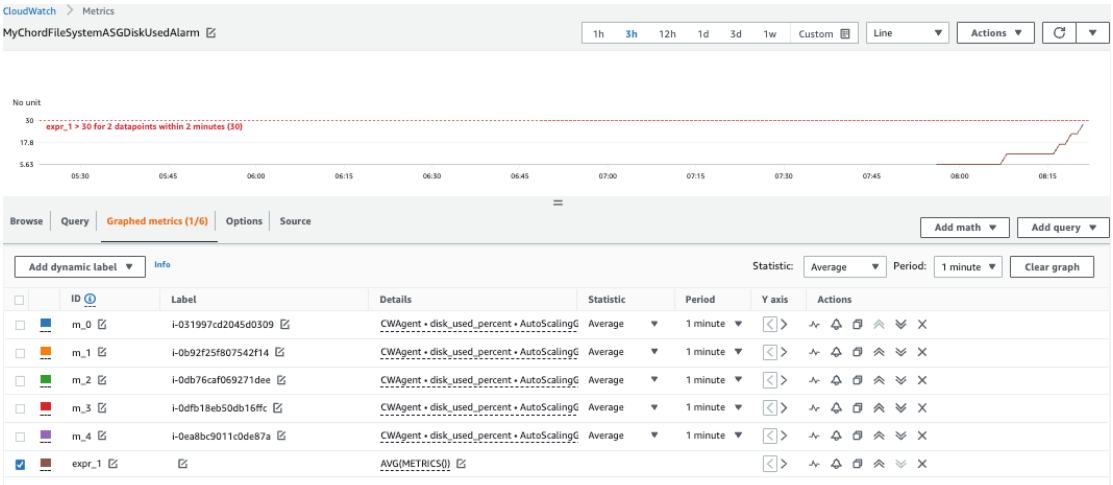
連進第二個 instance 看發現 flie 有被 migration 和 replica 進來

```
[ec2-user@ip-172-31-82-215 files]$ ls
b.txt_chunk_1  c.txt_chunk_1  d.txt_chunk_1
[ec2-user@ip-172-31-82-215 files]$
```

接著逐步把 instance 擴充到 5 個



去 cloudwatch 看 alarm 結果



把大檔案 a.txt 從 client 端上傳發現有被切成 7 個 chunk 上傳到不同地方

```
> python3 upload.py a.txt 3.215.184.16
filename:  a.txt
Hash of a.txt_chunk_1 is 2756827572
{'files': <_io.BufferedReader name='a.txt_chunk_1'>}
Uploading file to http://54.152.226.234
Hash of a.txt_chunk_2 is 1370087605
{'files': <_io.BufferedReader name='a.txt_chunk_2'>}
Uploading file to http://3.215.184.16
Hash of a.txt_chunk_3 is 2707073571
{'files': <_io.BufferedReader name='a.txt_chunk_3'>}
Uploading file to http://54.152.226.234
Hash of a.txt_chunk_4 is 3608324371
{'files': <_io.BufferedReader name='a.txt_chunk_4'>}
Uploading file to http://3.91.178.60
Hash of a.txt_chunk_5 is 844703452
{'files': <_io.BufferedReader name='a.txt_chunk_5'>}
Uploading file to http://3.91.178.60
Hash of a.txt_chunk_6 is 1307373017
{'files': <_io.BufferedReader name='a.txt_chunk_6'>}
Uploading file to http://3.215.184.16
Hash of a.txt_chunk_7 is 2238907549
{'files': <_io.BufferedReader name='a.txt_chunk_7'>}
Uploading file to http://3.85.99.175

~/Desktop/專案/分散式系統/chord-part-2/uploadserver ma
```

Ssh 進每個 instance 看

```
[ec2-user@ip-172-31-14-115 files]$ ls
[ec2-user@ip-172-31-14-115 files]$ ls
b.txt_chunk_1  c.txt_chunk_1  d.txt_chunk_1
[ec2-user@ip-172-31-14-115 files]$ ls
b.txt_chunk_1  c.txt_chunk_1  d.txt_chunk_1
[ec2-user@ip-172-31-14-115 files]$ ls
a.txt_chunk_1  a.txt_chunk_3  a.txt_chunk_5  b.txt_chunk_1  d.txt_chunk_1
a.txt_chunk_2  a.txt_chunk_4  a.txt_chunk_6  c.txt_chunk_1
[ec2-user@ip-172-31-14-115 files]$
```

```
[ec2-user@ip-172-31-82-215 files]$ ls
b.txt_chunk_1  c.txt_chunk_1  d.txt_chunk_1
[ec2-user@ip-172-31-82-215 files]$ ls
a.txt_chunk_2  a.txt_chunk_6  a.txt_chunk_7  b.txt_chunk_1  c.txt_chunk_1  d.txt_chunk_1
[ec2-user@ip-172-31-82-215 files]$
```

```
[ec2-user@ip-172-31-1-79 files]$ ls
a.txt_chunk_1  a.txt_chunk_3  a.txt_chunk_7  b.txt_chunk_1
[ec2-user@ip-172-31-1-79 files]$
```

```
[ec2-user@ip-172-31-11-94 ~]$ cd files/
[ec2-user@ip-172-31-11-94 files]$ ls
a.txt_chunk_2  a.txt_chunk_5  b.txt_chunk_1  d.txt_chunk_1
a.txt_chunk_4  a.txt_chunk_6  c.txt_chunk_1
[ec2-user@ip-172-31-11-94 files]$
```

```
[ec2-user@ip-172-31-82-172 ~]$ cd files/
[ec2-user@ip-172-31-82-172 files]$ ls
a.txt_chunk_1  a.txt_chunk_4  a.txt_chunk_7  c.txt_chunk_1
a.txt_chunk_3  a.txt_chunk_5  b.txt_chunk_1  d.txt_chunk_1
[ec2-user@ip-172-31-82-172 files]$
```

發現所有 file 都有三個 replica

References:

https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/autoscaling.html

https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/cloudwatch.html

https://pypi.org/project/ec2-metadata/

https://github.com/awsdocs/aws-doc-sdk-examples/blob/main/python/example_code/cloudwatch/cloudwatch_basics.py

https://docs.aws.amazon.com/zh_tw/code-library/latest/ug/python_3_auto-scaling_code_examples.html

https://docs.aws.amazon.com/zh_tw/code-library/latest/ug/python_3_cloudwatch_code_examples.html

https://docs.aws.amazon.com/zh_tw/autoscaling/application/userguide/create-step-scaling-policy-cli.html