Ein Fargate-Anwendungsfall

In unserem Anwendungsfall konzentrieren wir uns auf das aktivierende Ereignis und erstellen eine AWS-Ereignisregel, die auf den von uns verwendeten AWS ECS-Cluster verweist:

```
{
  "detail": {
    "clusterArn": ["arn:aws:ecs:eu-central-1:1234567890:cluster/clustername"],
    "lastStatus": ["ACTIVATING"]
  },
  "detail-type": ["ECS Task State Change"],
  "source": ["aws.ecs"]
}
```

Das Ziel für diese Ereignisregel ist eine AWS Lambda-Funktion mit den entsprechenden IAM-Berechtigungen:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ec2:CreateTrafficMirrorSession",
                "ec2:DescribeTrafficMirrorSessions",
                "ec2:CreateTags"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "logs:CreateLogGroup",
            «Resource»: «arn:aws:logs:eu-central-1:1234567890:*»
        },
        {
            "Effect": "Allow",
            "Action": "ecs:StopTask",
            "Resource": "arn:aws:ecs:eu-central-1:1234567890:task/*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogStream",
                "logs:PutLogEvents"
            ],
            "Resource": [
                "arn:aws:logs:eu-central-1:1234567890:log-group:/aws/lambda/
loggroupname:*"
            ]
        }
    ]
}
```

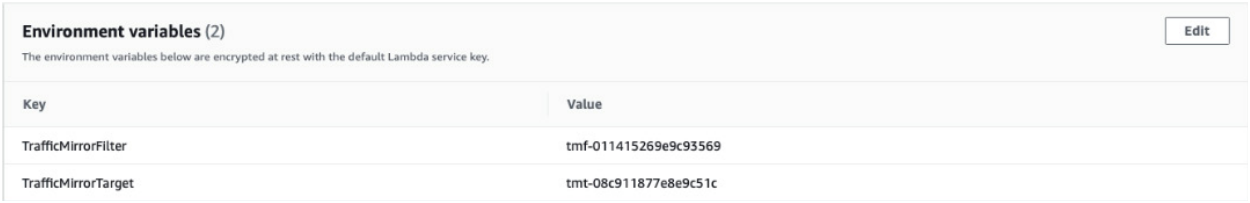
Konfigurieren einer vertrauenswürdigen Entität

Damit Lambda mit der IAM-Rolle ausgeführt werden kann, benötigen wir eine vertrauenswürdige Entität:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "Service": "lambda.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

Die AWS Lambda-Funktion muss auch den zuvor erstellten Traffic Mirror-Filter und das Traffic Mirror-Ziel kennen.

Wir haben unser Lambda so konfiguriert, dass diese Werte als Umgebungsvariablen verfügbar sind:



Der Lambda-Quellcode

Mit diesem Setup können wir den Lambda-Quellcode selbst vorbereiten. Es ruft die erforderlichen Daten ab – wie die Elastic Network-Schnittstellen des neu erstellten Containers – und verwendet sie, um eine Traffic Mirror-Sitzung mit dieser Schnittstelle durchzuführen.

Der Quellcode dafür lautet:

```
import json
import boto3
import logging
import os
from botocore.exceptions import ClientError


log = logging.getLogger()
log.setLevel(logging.INFO)

def lambda_handler(event, context):
    TrafficMirrorFilter = os.environ.get('TrafficMirrorFilter')
    TrafficMirrorTarget = os.environ.get('TrafficMirrorTarget')
    client = boto3.client('ec2')
    client_ecs = boto3.client('ecs')
    #print(json.dumps(event))

    if event[«source»] != «aws.ecs» or event[«detail-type»] != «ECS Task State
Change»:
       log.info(f'Function only supports input from events with a source type of: aws.
ecs')
       raise ValueError(«Function only supports input from events with a source type
of: aws.ecs»)
    log.info(f'laststatus: {event[«detail»][«lastStatus»]}')
    TaskId = event[«resources»][0].split('/')[2]
    Cluster = event[«resources»][0].split('/')[1]
    log.info(f'Container Task Id: {TaskId} on Cluster: {Cluster}')
    for attachment in event[«detail»][«attachments»]:
      #log.info(f'{json.dumps(event)}')
      if attachment.get(«status») == «ATTACHED» and attachment.get(«type») == «eni»:
        for property in attachment.get(«details»):
          if property[«name»] == «networkInterfaceId» and event[«detail»]
[«lastStatus»] == «ACTIVATING»:
            print(property[«value»])
            #check if tms allready exists
            existing_tms = client.describe_traffic_mirror_sessions(
                Filters=[
                    {
                        'Name': 'network-interface-id',
                        'Values': [
                            property[«value»],
                        ]
                    },
```

Der Lambda-Quellcode

```
                ],
                MaxResults=5
            ).get('TrafficMirrorSessions', [])
            if existing_tms:
              log.info(f'TrafficMirrorSession for interface {property[«value»]} already
exists')
            else:
              log.info(f'Creating TrafficMirrorSession for interface
{property[«value»]}')
              try:
                response = client.create_traffic_mirror_session(
                    NetworkInterfaceId=property[«value»],
                    TrafficMirrorTargetId=TrafficMirrorTarget,
                    TrafficMirrorFilterId=TrafficMirrorFilter,
                    SessionNumber=1,
                    Description=»Traffic mirror Source created by lambda»,
                    TagSpecifications=[
                        {
                            'ResourceType': 'traffic-mirror-session',
                            'Tags': [
                                {
                                    'Key': 'Name',
                                    'Value': «Session for interface « +
property[«value»]
                                },
                            ]
                        },
                    ],
                    DryRun=False
                )
              except ClientError as e:
                if e.response['Error']['Code'] == 'InvalidNetworkInterfaceID.
NotFound':
                  log.info(f'Eni not present yet {property[«value»]}')
                elif e.response['Error']['Code'] ==
'NetworkInterfaceNotSupportedException':
                  response = client_ecs.stop_task(
                      cluster=Cluster,
                      task=TaskId,
                      reason='killed by lambda due to not spawned on nitro instance
for trafficmirror reasons'
                  )
                  log.info(f'Killing Task: {TaskId} on Cluster {Cluster} since the
task was not started on a nitro instance (Traffic mirror not available)')
                else:
                                  log.info(f'Error for  {property[«value»]} is Code:
            {e.response[«Error»][«Code»]} Message: {e.response[«Error»][«Message»]}')
```

Von nun an wird dieses Setup ausgelöst, wenn ein neuer Amazon Elastic Container Service (ECS) gestartet wird. Es erstellt automatisch eine Traffic Mirror-Sitzung und hängt sie an das zuvor erstellte Traffic Mirror-Ziel an.

Auf diese Weise empfängt die Lösung jeden eingehenden Datenverkehr von allen Containern, die innerhalb des Fargate-Service ausgeführt werden.