Fachhochschule Nordwestschweiz
Hochschule für Architektur, Bau und Geomatik

PCLS - Case Study

Lambda-trigger-demo

Ilija Kljajic, Marcel Soltermann, Sebastian Grau und Ismail Cadaroski

Brugg, 09.01.2023

Inhaltsverzeichnis

1 Use Case Beschreibung

Als Ausgangslage für den Use Case wird ein KMU angenommen, welches eine statische Webseite hostet und den Besuchenden der Webseite ermöglicht via Formular eine Lambda Funktion zu triggern. Hierfür sollte die Lösung tiefe Kosten generieren, eine hohe Skalierbarkeit bieten und Hochverfügbar sein.

2 Gewählte Architektur

Um die Anforderungen des Use Case zu erfüllen, wird eine hochverfügbare Infrastruktur benötigt. Der Aufbau jener Umgebung wird in verschiedene Bereiche segmentiert und nach jener Aufteilung nachgebaut, um falls nötig Services oder Dienste austauschen zu können.

Dies wäre wichtig, da neben den Kosten auch neue Erkenntnisse aufkommen könnten und daraufhin die Möglichkeit auf einen Services Wechsel besteht. In der folgenden Beschreibung wird die schlussendlich eingesetzte Lösung beschrieben. Auf die zuvor geplanten Services und deren Zusammenarbeit wird im Kapitel «Alternativen» näher eingegangen.

Unsere Architektur setzt sich folgendermassen zusammen:

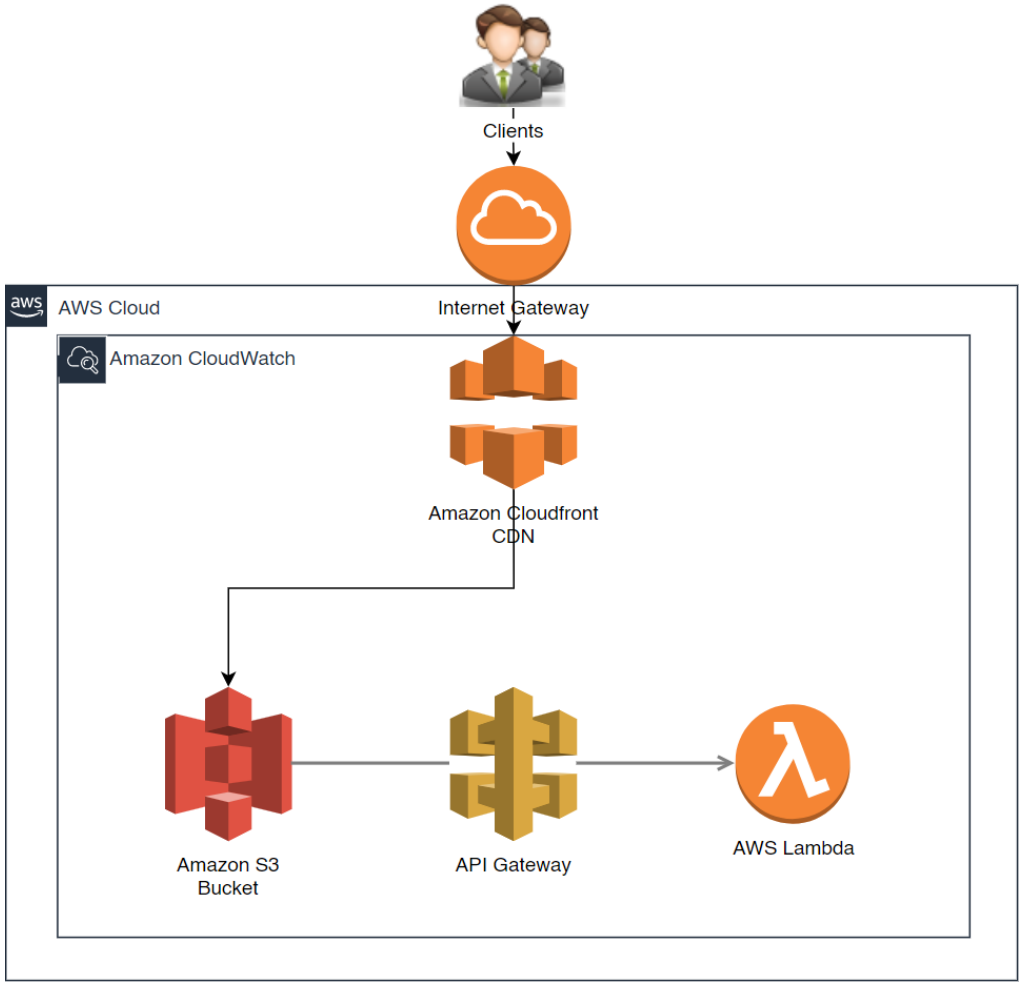

Abbildung 1 - Architektur Zusammenarbeit

2.1 Storage – S3 Bucket

Auf dem S3 Bucket werden die statischen Daten abgelegt, bevor sie mit der Hilfe des CDN zu den Benutzern verschickt werden. S3 wurde wegen seiner hohen Skalierbarkeit und tiefen Latenz ausgewählt. Des Weiteren ist es Simpel Daten vom S3 Bucket herunterzuladen, was die allfällige Analyse der gespeicherten Daten vereinfacht.

2.2 Lambda

Da Lambda ein Serverless Computing Service ist und dadurch keine Bereitstellung oder Verwaltung eines Servers benötigt, wurde er für die Implementation eingesetzt. Ausserdem entstehen die Kosten pro Request und wenn die Berechnungen der Lambda Funktion nur selten genutzt wird, ist dies kostengünstiger als dauerhaft Rechenleistung bereit zu stellen.

2.3 Load balancing – Application Load Balancer

Damit ein Ausgleich der Webserver – Belastung bewerkstelligt werden kann, wird der Applikation Load Balancer (Layer 7) eingesetzt. Mit diesem Load Balancer werden die allfällige HTTPS-Anfragen auf alle aktiven Webservern verteilt. Er wurde in der aktuellen Iteration nicht mehr aktiv benötigt, wurde jedoch belassen um aufzuzeigen dass der Aufwand für die Konfiguration schon stattfand und dadurch die Kompatibilität getestet wurde.

2.4 API-Gateway

Mit dem Amazon API Gateway wurde eine Web-API mit einem http Endpunkt für die Lambda Funktion erstellt. Die Kombination vom API-Gateway ist Serverlos und dadurch kosten-effektiv und sehr gut skalierbar. Der Webclient kann nun die API-Aufrufen und er API-Gateway kann die Anforderungen an Lambda umleiten.

2.5 CDN – CloudFront

Mit dem CDN (Content Delivery Network) werden Daten für die Benutzer des Umfrageportals mit reduzierter Latenz und gleichzeitiger Entlastung der S3 Instanz zwischengespeichert und zugeschickt.

2.6 Monitoring - CloudWatch

Mithilfe von Cloudwatch werden die beim Betrieb benötigten Ressourcen überwacht. Dabei wird jeweils wichtige Metriken der einzelnen Services überwacht.

2.7 Alternativen

Eine Alternative wäre gewesen eine EC2-Instanz für das Webhosting zu verwenden. Auf der EC2-Instanz hätte ein Webserver installiert werden, Storage angebunden und allenfalls eine Datenbank bereitgestellt werden können. Wobei es sicher besser wäre die DB als separaten Service auf einer anderen EC2-Instanz oder Serverless zu beziehen. Durch dieses alternative Setup wäre es möglich gewesen eine dynamische Webseite zu hosten. Dadurch hätte man auf S3, Lambda und das API-Gateway verzichten können. EC2-Instanzen sind aber relativ teuer und da es sich dabei um Infrastructure-as-a-Service handelt, müsste man sich auch um die ganze VM inklusive Betriebssystem und installierter Software kümmern. Der Wartungsaufwand wäre also deutlich grösser. Für unser einfaches Case Study Setup reicht eine

statische Webseite auf S3 und eine Lambda Funktion, welche die Berechnungen übernimmt. Das ist einfacher und kostengünstiger, da die Lambda Funktion nur dann Kosten verursacht, wenn sie ausgeführt wird und nicht immer laufen muss, wie es bei der EC2-Instanz der Fall wäre.

Eine Alternative zum API-Gateway wäre gewesen, direkt die Lambda Function-URL zu verwenden. Die Function-URL ist ein relativ neues Feature, welches zu AWS-Lambda hinzugefügt wurde und erlaubt Lambda Funktionen direkt anzusprechen. Das API-Gateway, welches bei uns zum Einsatz kommt, gehört eher zu den klassischen Wegen wie man eine Lambda Funktion triggert. Dabei handelt es sich zwar um einen zusätzlichen Service, den man bereitstellen muss, dafür ist der API-Gateway etwas flexibler und mächtiger, als die Lambda Function-URL. Bestimmte Features wie beispielsweise custom DNS-Support fehlen bei der Lambda Function-URL. Durch das grössere Feature Set beim API-Gateway und die höhere Flexibilität kann das Setup später einfacher erweitert und ausgebaut werden.

Statt das Monitoring mit CloudWatch aufzubauen hätte beispielsweise auch Amazon Managed Service for Prometheus und Amazon Managed Grafana eingesetzt werden können. Beides sind etablierte Werkzeuge für Monitoring und hätten das Risiko für ein Vendor lock-in verringert, da beide Services auch ausserhalb von AWS zur Verfügung stehen. Allerdings wäre das Aufsetzen umständlicher gewesen. CloudWatch ist verhältnismässig einfach, bestens in AWS integriert und reicht für unsere Case Study als Monitoring Lösung aus. Es bietet ausserdem den Vorteil, dass ein Service sowohl das Sammeln als auch das Visualisieren der Metriken übernimmt.

2.8 Entscheidungsweg

Der Entscheidungsweg hin zum aktuellen Setup war ein wenig holprig, da während der Implementierung die Architektur angepasst wurde. Bei der ersten Iteration war angedacht eine Applikation als Docker Container auf Elastic Container Service zu deployen. Dabei kam es aber zu einigen Problemen, insbesondere mit den benötigten Volumes und der Verbindung zur Datenbank. Also haben wir uns nach einiger Zeit und diversen fehlgeschlagenen Versuchen für ein alternatives Setup entschieden und die Architektur geändert. Neu sollte eine statische Webseite auf S3 eine Lambda Funktion triggern. Angedacht war, dass die User CloudFront aufrufen, der Request an einen Loadbalancer weitergeleitet wird, der wiederum auf die statische Webseite zeigt. Allerdings hat sich während der Implementation auch diese Architektur als nicht korrekt herausgestellt. Als erste Massnahme wurden die Logs des Loadbalancers in den bestehenden S3 Bucket geschrieben. Nach einer Analysephase wurde die Architektur ein drittes Mal verändert. Der Loadbalancer wurde mit dem API-Gateway ersetzt. Cloudfront leitet die Anfragen nun direkt an die statische Webseite, welche auf dem S3 Bucket gehostet wird, weiter. Ein Formular auf der Webseite sendet einen POST Request an das API-Gateway, welches die Anfrage entgegennimmt und die Lambda Funktion triggert. Überwacht wird das ganze Setup, wie von Anfang an geplant, durch CloudWatch. Der Loadbalancer, welcher Logs in den S3 Bucket schreibt haben wir im Deployment drin belassen. Es wäre denkbar später das Setup zu erweitern und Beispielsweise EC2-Instanzen hinter den Loadbalancer zu hängen. Aktuell wird der Loadbalancer aber nicht verwendet.

3 Vorgehensweise bei der Umsetzung

3.1 Zusammenarbeit und Tools

Für die Zusammenarbeit wurde ein privates Repository von GitHub eingesetzt. Damit konnten alle Projektteilnehmer an den verschiedenen Services ungestört arbeiten. Es wurde dabei für jeden neuen Service ein separater Branch erstellt welcher dann bei erfolgreicher Ausführung mit dem Rest der Konfiguration zusammengeschlossen und getestet wurde.

Da ein Teil der Projektteilnehmer schon Erfahrung mit Terraform hatte, wurde es für die Umsetzung der Konfiguration eingesetzt.

3.2 Starten der Konfiguration

Folgendes muss für die Konfiguration der Informationen in der variable.tf Datei bearbeitet werden:

```
variable "aws_access_key" {
  default = "<your_aws_access_key_id>"
}
variable "aws_secret_key" {
  default = "<your_aws_secret_access_key>"
}
variable "aws_session_token" {
  default = "<your_aws_session_token>"
}
variable "account_id" {
  default = "<your_aws_account _id>"
}
variable "region" {
  default = "<your_aws_default_region>"
}

variable "emailaddress" {
  description = "Email address for the SNS notifications"
  default = "<your_email_address>"
}
```

Alle benötigten Informationen können im AWS Learner Lab unter AWS-Details aufgefunden werden.

Darauffolgend kann die Ausführung beginnen:

1. Initialisierung des Arbeitsverzeichnisses mit: Terraform init
2. Verifizieren des Skripts mit: Terrafrom plan
3. Ausrollen der Konfigurierten Infrastruktur mit: Terraform apply

Sobald das Setup einmal ausgeführt wurde, muss in der Datei index.html die URL vom API-Gateway durch die aktuelle URL des API-Gateway ersetzt werden. Anschliessend kann man die Datei auf dem S3 Bucket (z.B. via GUI Upload Button) ersetzen.

3.3 Informationsbeschaffung und Zusammenführung der Services

Die benötigten Informationen wurden neben der Dokumentation des PCLS-Modules aus verschiedenen Tutorials bezogen. Die Zusammenführung von zwei bis drei Services war dabei schnell zu finden, jedoch war die Suche darüber hinaus ein wenig aufwendiger als erwartet. Diese Informationen wurden dann im Team in verschiedene Sitzung kommuniziert und zusammengesetzt.

Darauffolgenden wurden die Services isoliert voneinander erstellt und getestet, um dann miteinander verbunden zu werden. Dabei wurde auch bemerkt, dass die erste geplante Iteration nicht wie gewünscht miteinander verbunden werden konnte und deswegen eine weitere Variante erstellt wurde, welche in den Kapiteln 2.7 und 2.8 beschrieben wurden.

4 Implementation

4.1 S3-Bucket

Zu Beginn wurde evaluiert welche Einstellungen konfiguriert werden müssen, um den anderen Services eine Einbindung zu ermöglichen. Daraus entstand eine Liste von nötigen Konfigurationen:

- Grundkonfiguration
  - Name
  - Tags
  - Access-Point (Für mögliche Erweiterungen)
- HTML-Files für Webseite
  - index.html
  - error.html
- Webseiten Access
  - Zugriff zu index.html erlauben
  - Cross-Origin Anfragen erlauben (für Lambda)

Die beschriebenen Konfigurationen wurden mittels einer Terraform-Datei zusammengestellt und durch die depends_on Funktion eingereiht, um Abhängigkeiten zu adressieren.

Die Grundkonfiguration umfasst den Namen des S3-Buckets sowie die Tags und eine Erstellung des Access-Points, für allfällige Erweiterungen. Die Terraformzeilen sind grundlegend einfach gehalten, lediglich bei der Policy für die HTML-Files sowie für die Erlaubnis der Cross-Origin Anfrage wurde spezifischer konfiguriert. Für die HTML-File-Policy wurde entsprechend eine Bucket-Policy erstellt welche alle Files im Bocket als Public deklariert, und somit GET-Requests zulässt:

```
resource "aws_s3_bucket_policy" "prod_website" {
  bucket =  "${var.s3Bucket}"
  policy = <<POLICY
   {
      "Version": "2012-10-17",
      "Statement": [
       {
          "Sid": "PublicReadGetObject",
          "Effect": "Allow",
          "Principal": "*",
          "Action": [
             "s3:GetObject"
          ],
```

```
            "Resource": [
              "arn:aws:s3:::${var.s3Bucket}/*"
            ]
            },
            {
            "Effect": "Allow",
            "Principal": {
              "AWS": "arn:aws:iam::127311923021:root"
            },
            "Action": "s3:PutObject",
            "Resource": "arn:aws:s3:::${var.s3Bucket}/AWSLogs/${var.account_id}/*"
          }
        ]
    }
    POLICY
  depends_on = [aws_s3_bucket.webpage_bucket_casestudy_fhnw]
}
```

Um bei POST-Requests an die Lambdafunktion kein CORS-Error zu generieren, ist es notwendig entsprechende Anfragen zuzulassen. Wird dies nicht konfiguriert, erhält die Lambdafunktion ein leeres JSON, was zu Parsingerrors führt. Entsprechend wurde eine CORS-Konfiguration erstellt welche POST und PUT Requests zulässt:

```
resource "aws_s3_bucket_cors_configuration" "s3_config" {
    bucket = "${var.bucket_name}"

    cors_rule {
        allowed_headers = ["*"]
        allowed_methods = ["PUT", "POST"]
        allowed_origins = ["*"]
        expose_headers  = ["ETag"]
        max_age_seconds = 3000
    }

    cors_rule {
        allowed_methods = ["GET"]
        allowed_origins = ["*"]
    }
}
```

4.2 Lambda

Die Lambda Funktion übernimmt die eigentliche Berechnung des Anwendungsfalles. Sie stellt die Computing Power zur Verfügung. Getriggert wird die Lambda Funktion durch ein API-Gateway welches durch ein Formular auf der statischen Webseite aufgerufen wird.

Als Programmiersprache für die Funktion wurde Python gewählt. Die Funktion nimmt einen Event entgegen und liefert als Antwort den HTTP-Statuscode 200, sowie den String «Resultat einfügen» zurück. Ursprünglich war geplant eine Liste mit zwei Zahlen als Input anzunehmen und das Resultat einer Ganzzahldivision als Output zurückzuliefern. Auf Grund von Problemen mit dem Erstellen eines Lambda Events respektive parsen des Bodies wurde dieser Teil aber auskommentiert. Offenbar ist es nicht trivial von extern (z.B. via curl) eine Lambda konforme Request zu generieren und den Body in der Funktion korrekt zu übersetzen. Entstanden ist deshalb die nachfolgende Lambda Funktion:

```python
import json
def handler(event, context):
    #numerator = event.get('numerator')
    #denominator = event.get('denominator')
    #result = numerator/denominator
    result = "{\"Resultat\": \"einfuegen\"}"
    response = {
        "statusCode": 200,
        "statusDescription": "200 OK",
        "isBase64Encoded": False,
        "headers": {
            "Content-Type": "text/plain",
        },
        "body": json.dumps(result)
        }

    return response
```

Deployed wird die Lambda Funktion mittels Terraform. Zusätzlich zur Funktion selbst wird auch noch eine Function URL deployed. Diese wird theoretisch benötigt, damit das Formular auf der Webseite die Lambda Funktion direkt triggern kann. In unserem Setup verwenden wir aber API-Gateway um die Lambda Funktion zu triggern. Wichtig ist dabei auch die CORS Policy zu konfigurieren und dort die URL der Webseite einzutragen um einen simple-Request (statt Preflight-cors-request) zu ermöglichen:

```
resource "aws_lambda_function" "lambda_aws_cli" {
  filename         = data.archive_file.zip.output_path
  source_code_hash = data.archive_file.zip.output_base64sha256
  function_name            = "${var.lambdaname}"
  role                  = "arn:aws:iam::${var.account_id}:role/LabRole"
  handler          = "func.handler"
  runtime          = "python3.8"


  depends_on = [
    aws_cloudwatch_log_group.aws_cli_log_group
  ]

  tags = var.tags
}

data "archive_file" "zip" {
  type        = "zip"
  source_dir = "../Lambda/"
  output_path = "../Lambda/packedlambda.zip"
}

resource "aws_lambda_function_url" "casestudy_lambda_url" {
  function_name      = aws_lambda_function.lambda_aws_cli.function_name
```

```
  authorization_type = "NONE"
  depends_on = [aws_lambda_function.lambda_aws_cli]

  cors {
    allow_credentials = true
    allow_origins     = ["http://${var.s3Bucket}.s3-website-us-east-1.amazonaws.com"]
    allow_methods     = ["*"]
    allow_headers     = ["date", "keep-alive"]
    expose_headers    = ["keep-alive", "date", "Access-Control-Allow-Origin", "Access-Control-Allow-Head-
ers"]
    max_age           = 86400
  }
}
```

4.3 Application Load Balancer

Das Learner Lab hat die Begrenzung, dass keine Zertifikate bearbeitet oder gelöscht werden können, jedoch ist es möglich ein Self-Signed-Certificate zu erstellen. Dafür wurden folgende Ressourcen via Terraformdatei erzeugt, welche einmalig pro AWS-Account erstellt werden können:

- TLS Private Key
- TLS Self Signed Certificate
- ACM Certificate (Für Cloudfront falls nicht das Default-Certificate verwendet werden soll)
- IAM Server Certificate

Sind diese erstellt, ist es möglich den Load Balancer zu erstellen (Für HTTP wäre kein Zertifikat nötig):

```
resource "aws_elb" "loadbalancer_casestudy_fhnw" {
  name               = "${var.laodbalancer}"
  availability_zones = ["us-east-1a", "us-east-1b", "us-east-1c"]

  listener {
    instance_port     = 8000
    instance_protocol = "http"
    lb_port           = 80
    lb_protocol       = "http"
  }

  listener {
    instance_port      = 8000
    instance_protocol  = "http"
    lb_port            = 443
    lb_protocol        = "https"
    ssl_certificate_id = "arn:aws:iam::918617678239:server-certificate/fhnw_cert"
  }

  health_check {
    healthy_threshold   = 2
    unhealthy_threshold = 2
    timeout             = 3
    target              = "HTTP:8000/"
    interval            = 30
  }
  tags = {
    Name = "loadbalancer-http-8080"
```

```
  }
  depends_on = [
    module.s3,
  ]
}
```

4.4 API-Gateway

Um einer möglichen Überlastung der Lambdafunktion entgegenzusteuern wurde entschieden, ein API-Gateway als Lambda-Trigger zu verwenden. Dafür wurde zuerst ein HTTP-API-Gateway erstellt, welches mittels Lambda URI ein Trigger generierte. Für diese Case Study wurde als Integration ein HTTP-Typ erstellt, da lediglich die Interaktion getestet werden soll, und nicht ein Backend besteht welches Lambda benötigt. Mittels AWS_Lambda_Permission wurde ein Zugriffserlaubnis konfiguriert, damit der Trigger entsprechend die Lambda auslösen kann.

```
resource "aws_lambda_permission" "apigw" {
  statement_id  = "AllowAPIGatewayInvoke"
  action = "lambda:InvokeFunction"
  function_name = "${aws_lambda_function.lambda_aws_cli.function_name}"
 source_arn = "${aws_api_gateway_rest_api.lambda_casestudy_api_gateway.execution_arn}/*/*"
}
```

4.5 CloudFront

Bei der Implementation des CDN ist zu beachten, dass der Domänen Name und die ID des S3 Bucket benötigt wurde, da wie die Abbildung 1 schon aufzeigt, jener hinter Cloudfront gestellt wird. Damit kann CloudFront der Instanz allfällige Lasten in der Form von Abfragen abnehmen:

```
resource "aws_cloudfront_distribution" "webpage_cf_cdn_casestudy_fhnw" {
  origin {
    domain_name = aws_s3_bucket.webpage_bucket_casestudy_fhnw.bucket_domain_name
    origin_id   = "http://${var.s3Bucket}.s3-website-us-east-1.amazonaws.com"
```

Die Port- und Protokollkonfigurationen wurden auf http und https mit den benötigten SSL-Zertifikaten belassen, da nur über jene der Zugriff möglich sein sollte:

```
    custom_origin_config {
      http_port              = 80
      https_port             = 443
      origin_protocol_policy = "http-only"
      origin_ssl_protocols   = ["TLSv1.2"]
    }
  }
```

Folgend wurden die HTTP-Methoden festgelegt, welche durch CloudFront prozessiert und weitergeleitet werden sollen. Die weitergeleiteten Werte und Cookies wurden als Grundkonfiguration belassen:

```
default_cache_behavior {
  allowed_methods        = ["GET", "HEAD", "OPTIONS", "PUT", "POST", "PATCH", "DELETE"]
  cached_methods         = ["GET", "HEAD", "OPTIONS"]
  target_origin_id       = "http://${var.s3Bucket}.s3-website-us-east-1.amazonaws.com"
  viewer_protocol_policy = "redirect-to-https"
  forwarded_values {
```

```
    headers      = []
    query_string = true
    cookies {
      forward = "all"
    }
  }
}
```

Als letztes wurde die Schweiz als weiteres Land in der Whitelist hinzugefügt und unter depends_on die S3 Bucket Policy definiert:

```
restrictions {
    geo_restriction {
      restriction_type = "whitelist"
      locations        = ["IN", "US", "CA", "CH"]
    }
} tags = var.tags
  viewer_certificate {
    cloudfront_default_certificate = true
  }
  depends_on = [
        aws_s3_bucket_policy.prod_website
  ]
}
```

4.6 CloudWatch

CloudWatch wurde ebenfalls mit Terraform konfiguriert werden. Dafür wurde der Simple Notification Service benötigt, mit diesem ist es möglich Benachrichtigungen an die Administratoren zu versenden. Damit kann sichergestellt werden, dass bei Problemen eine Benachrichtigung versendet wird.

Der Simple Notification Service beinhaltet zwei Ressourcen, die via Terraform angelegt werden müssen. Für die Benachrichtigungen sind jeweils ein Topic und eine Subscription zu defineiren. Mehrere Topics können beispielsweise hilfreich sein, wenn für S3-Alarme jemand anders verantwortlich ist als für Cloudfront-Alarme. Die Subscription beinhaltet dann das zu verwendende Protokoll und die Emailadresse/n der zu alarmierenden Personen.

```
resource "aws_sns_topic" "case_study_sns" {
  name = var.sns_name
}

resource "aws_sns_topic_subscription" "sns_subscription" {
  topic_arn = aws_sns_topic.case_study_sns.arn
  protocol  = "email"
  endpoint  = "aws@punraz.ch"
}
```

Es wurde sich bei der Case Study auf acht Alarme entschieden und diese überwachen die Services Cloudfront, API-Gateway, S3 und Lambda. Ein Problem, welches aufgetaucht ist, ist das CloudWatch die Datensammlung erst nach 48h als genügend erachtet und deswegen stets «unzureichende Daten» als Meldung bringt und da die Labs jeweils nur 4h laufen, können die Daten nicht gezeigt werden.

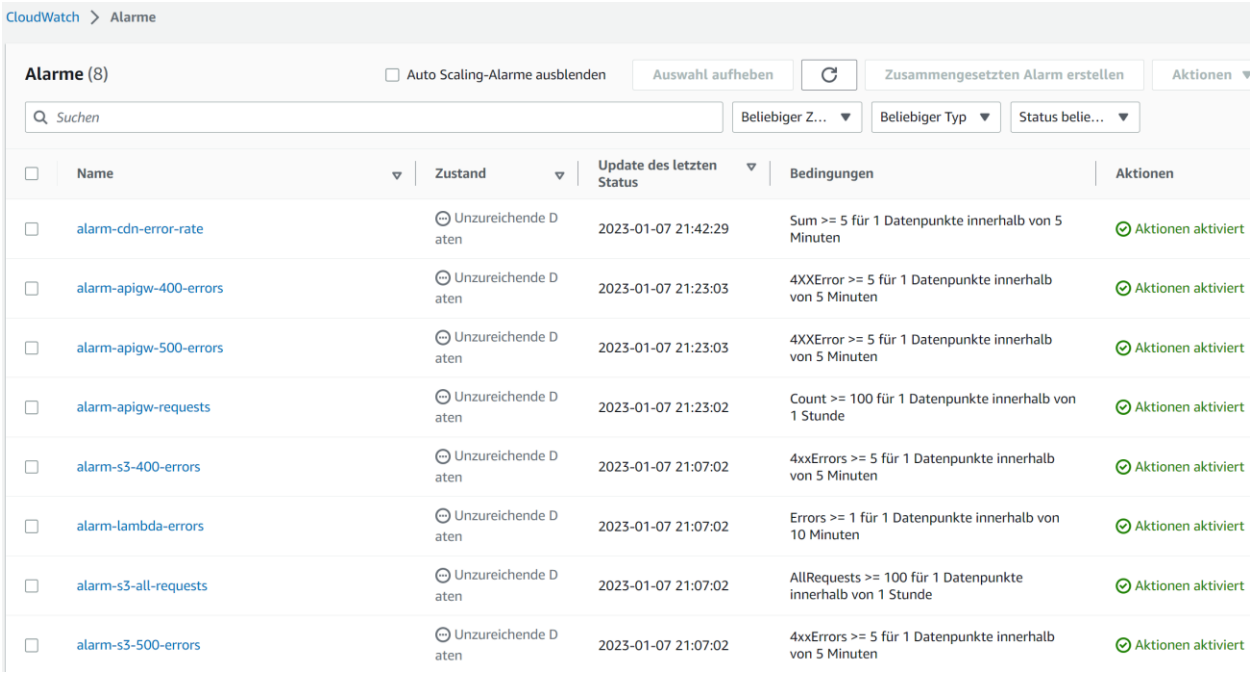

Abbildung 2 - Alarme

4.6.1 Lambda

Bei Lambda werden alle Fehler überwacht, die bei der Ausführung des Lambda-Ausdruckes passieren, da dies eine aussagekräftigste Metrik ist. Es wird alarmiert, wenn in 10min eine oder mehr Fehler passieren, da Fehler im Lambda-Ausdruck wichtig zu wissen sind. Als Aktion ist das definierte Topic hinterlegt. Um Platz zu sparen, wird nur die Alarmdefinition des Lambdaservices als Code eingefügt, die anderen können im Repository betrachtet werden. Denn nennenswertes, was sich ändert wird in Textform erklärt. Dies ist jeweils der Name der Metrik, der Zeitraum, die «statistic» (Durchschnitt, Anzahl, Summe der Ereignisse) und der Wert, bei dem alarmiert wird, wenn er über- oder unterschritten wird.

```
resource "aws_cloudwatch_metric_alarm" "cwa_lambda_errors" {
  alarm_name = "alarm-lambda-errors"
  comparison_operator = "GreaterThanOrEqualToThreshold"
  evaluation_periods = 1
  metric_name = "Errors"
  namespace = "AWS/Lambda"
  period = 600 # time in seconds, 10 min -> 600s
  statistic = "Sum"
  threshold = 1
  alarm_description = "Monitor errors in Lambda function"
  insufficient_data_actions = []
  dimensions = { functionName =  var.lambdaname }
  alarm_actions = [aws_sns_topic.case_study_sns.arn]
}
```

4.6.2 S3

Für S3 wurden drei Metriken ausgewählt. Es wird die Anzahl Requests alarmiert, wenn innerhalb einer Stunde mehr als 100 Requests passieren, da die Case Study kein Produktivsystem einer Firma ist, sollten niemals so viele Requests auftauchen. Ausserdem werden sowohl http 400er und http 500er Fehler alarmiert, wenn mehr als fünf Fehler innerhalb von fünf Minuten passieren.

4.6.3 API-Gateway

Beim API-Gateway wurden ebenfalls die Anzahl Requests und die 400er und 500er Fehler gewählt, dies aus denselben Gründen wie beim S3 Service und es kann so besser beurteilt werden, wo die Fehler liegen, wenn Fehler auftauchen.

4.6.4 CloudFront

In Cloudfront wird die Fehlerrate überwacht, sprich die Gesamtanzahl aller fehlerhaften Requests, die im CDN ankommen.

5 Erkenntnisse

Im Verlauf der Case Study sind Probleme und Konfigurationsfehler aufgekommen, welche Änderungen im Terraformskript oder zu Serviceaustausch geführt haben. Die Probleme und dazugehörigen Lösungen werden folgend aufgeführt und diskutiert.

5.1 ECS Docker Deployment

Zu Beginn der Case Study war eine ECS-Instanz vorgesehen, welche mittels einem Docker-Container ein Umfragetool zur Verfügung stellt. Das Container-Deployment benötigte einen Mountpoint, welcher nicht richtig konfiguriert werden konnte. Dies führte zu mehrfachen Error-Deployments der ECS-Instanzen und somit zu Abbruch des Terraform Build Cycles. Die Dokumentationen zu diesem Problem waren spärlich und zum Teil auch nicht vorhanden, was uns zu der Entscheidung brachte, die Servicestruktur anzupassen.

5.2 Abhängigkeiten im Terraformskript

Durch das Arbeiten via GitHub und Branches, wurden einzelne Ressourcen separat erstellt und anschliessend zusammengeführt. Dies führte dazu, dass einzeln getestet die Ressourcen erstellt wurden und funktionierten, jedoch bei paralleler Erstellung Dependance-Probleme aufkamen. Eine Ablaufplanung wurde somit erforderlich, die einzelnen Ressourcen wurden auf Abhängigkeiten untersucht und entsprechend eingereiht.

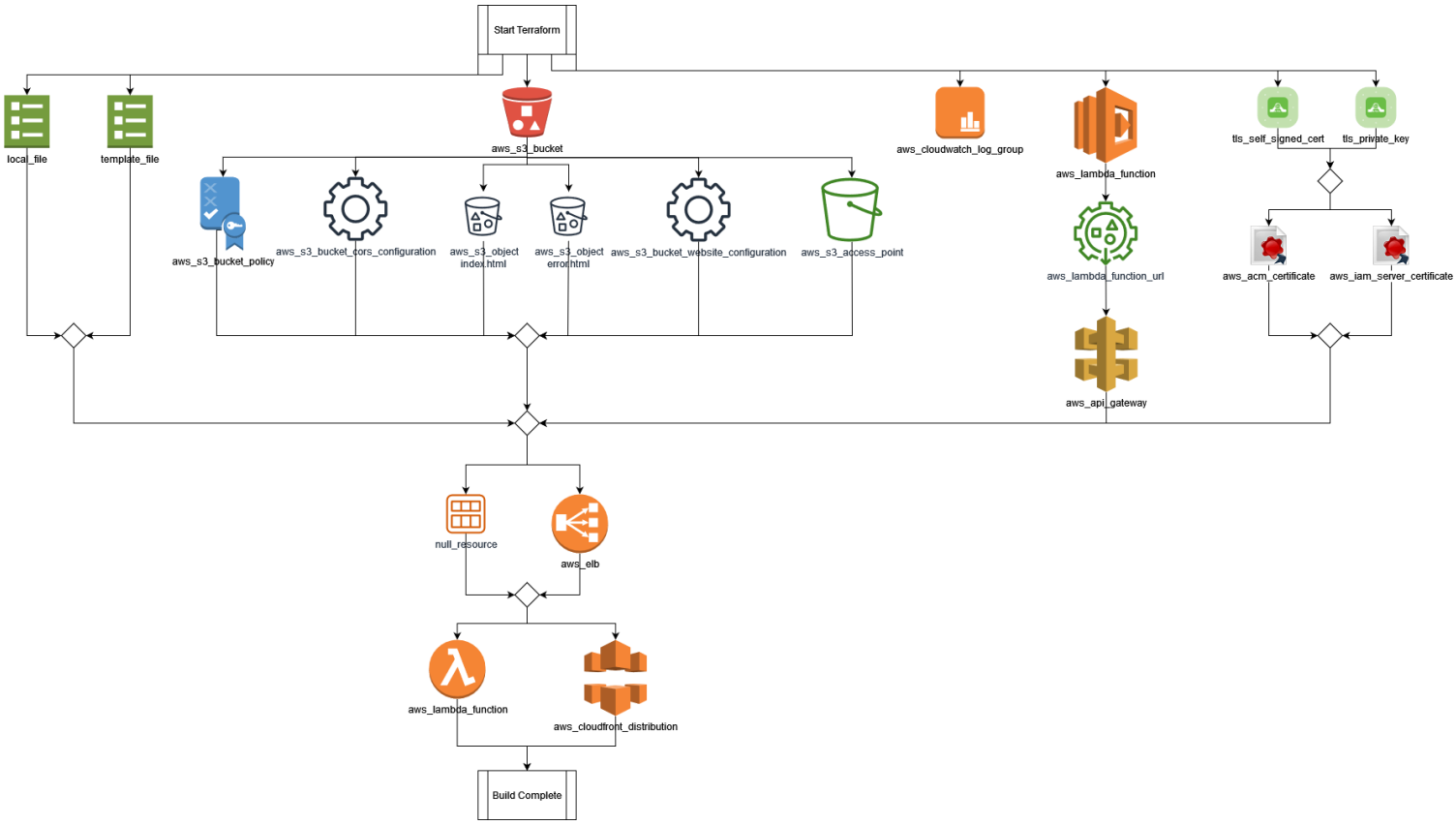

Abbildung 3: Ablaufdiagram des Terraformskripts

5.3 Zertifikate

Um den Loadbalancer ebenfalls für HTTPS-Anfragen zu erstellen, sind Zertifikate von Nöten. Das Lerner-Lab ist für Zertifikate nicht ausgelegt und verbietet das Editieren, Löschen und Managen von Zertifikaten, jedoch ist es möglich via Terraform ein Self-Signed-Certificate zu erstellen. Diese Lösung erwies sich aber als zweckmässig, das Zertifikat wird mit einer Gültigkeitsdauer von zwölf Stunden erstellt und muss dann aus dem Terraformskript entfernt werden. Wird es nicht entfernt, wirft AWS in der Build-Phase ein AccessDenied-Error und bricht das Skript ab. Mittels Pfades ist es jedoch möglich das zuvor erstellte Zertifikat dem Loadbalancer zu übergeben und somit ein HTTPS-Listener zu erzeugen.

5.4 Loadbalancer

Zuerst wurde vom Projektteam angenommen, der Zugriff auf die statische S3-Bucket-Webseite könne mittels dem LoadBalancer abgefertigt werden. Wie sich aber herausstellte wird der LoadBalancer an eine ECS-Instanz angehängt und nicht an eine beliebige Ressource. Entsprechend wurde der S3-Bucket umkonfiguriert, um die Logs des LoadBalancers aufzunehmen.

5.5 CloudFront Zertifikat

Damit CloudFront funktioniert sind ebenfalls Zertifikate nötig, diese wurden in der Phase für die LoadBalancer-Zertifikate erstellt. Jedoch werden die Self-Signed-Certificates nicht akzeptiert, entweder wegen den LernerLab-Policies oder von AWS selbst. Es musste entsprechend auf ein CloudFront-Default-Certificate ausgewichen werden, welches aber nur funktioniert, wenn man das «alias»-Value auskommentiert.

5.6 Lambda Events

Beim triggern der Lambda Funktion hatten wir einige Probleme. Zum einen mit cross-origin requests und zum anderen mit dem Parsen des Events Body vom Request. Bezüglich cross-origin musste bei der Lambda Function URL als «allow_origins» die URL der statischen Webseite eingetragen, sowie die Header «Access-Control-Allow-Origin» und «Access-Control-Allow-Headers» exposed werden. Das Parsen vom Event Body haben wir schlussendlich weggelassen, da die Funktion immer Fehler warf, wenn sie von extern (via curl oder via Formular auf der Webseite) getriggert wurde. Dabei hat es auch keine Rolle gespielt, ob ein API-Gateway dazwischen geschaltet wurde. Der externe Request wurde nicht als AWS Lambda Event erkannt, respektive der Payload, der in der «event» Variable drinstecken sollte, konnte nicht korrekt geparsed werden. Dieses Verhalten war aber nur bei externen Requests zu beobachten. Bei internen Requests, also via AWS GUI Test, funktionierte die Lambda Funktion und lieferte (inklusive korrektem Parsen), das gewünschte Resultat.

6 Fazit

Diese Case Study zeigte, wie schon in Kapitel fünf beschrieben, einige Überraschungen und Hindernisse auf, die mehrere Iterationen der Planung und Implementation anforderten. Im Nachhinein hätten ein Grossteil der Fehler, die im späteren Verlauf ziemlich viel Zeit kosteten, schon in der Planungsphase der Architektur mit mehr Erfahrung umgangen werden können.

Ein bestehendes Problem, welches bei der Erstellung des CloudFront Dienstes erkannt wurde, ist die Zuweisung der Berechtigungen des AWS Leaner Lab Accounts. Dadurch mussten Wege gefunden werden, die Löschungen und Erstellungen von Zertifikaten umgingen.

Es wäre interessant gewesen, den gleichen Aufbau in einer Azure Umgebung zu Implementieren und dabei herauszufinden, wie der gleiche Aufbau dort aussehen würde. Damit könnte man besser aufzuzeigen wo Azure und AWS bei einer aktiven Implementation die grösste Differenz im Public Cloud Bereich hätten.

Grundsätzlich war es eine sehr interessante Case Study, bei der dem Team einfach noch Erfahrung zur Auswahl und Implementation der Architektur fehlte.

7 Literatur

Hands-on-cloud, “Managing Amazon CloudFront using Terraform”, [Online] verfügbar: https://hands-on.cloud/managing-amazon-cloudfront-using-terraform/ (Abfragedatum: 29.12.2022).

„Viewing CloudFront and edge function metrics - Amazon CloudFront“. Amazon. https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/viewing-cloudfront-metrics.html (Zugriff am 7. Januar 2023).

B. Delb. „Create a CloudWatch alert with Terraform (AWS)“. Medium. https://medium.com/open-devops-academy/create-a-cloudwatch-alert-with-terraform-aws-81a6cac77f80 (Zugriff am 7. Januar 2023).

8 Abbildungen

9 Script

```
# Credentials for Terrafom and AWS-CLI
provider "aws" {
  access_key = "${var.aws_access_key}"
  secret_key = "${var.aws_secret_key}"
  token = "${var.aws_session_token}"
  region = "${var.region}"
}


#========[ S3 BUCKET ]==============
resource "aws_s3_bucket" "webpage_bucket_casestudy_fhnw" {
  bucket = "${var.s3Bucket}"
  acl = "${var.acl_value}"

  tags = {
    "use": "static webpage",
    "loadbalanced": "yes"
  }
}

resource "aws_s3_bucket_cors_configuration" "webpage_config" {
  bucket = "${var.s3Bucket}"

  cors_rule {
    allowed_headers = ["*"]
    allowed_methods = ["PUT", "POST"]
    allowed_origins = ["*"]
    expose_headers  = ["ETag"]
    max_age_seconds = 3000
  }

  cors_rule {
```

```
    allowed_methods = ["GET"]
    allowed_origins = ["*"]
  }
}

resource "aws_s3_object" "file_upload_index" {
  bucket      =  "${var.s3Bucket}"
  key         =  "index.html"
  acl         = "public-read"
  source      =  "${path.module}/webpage/index.html"
  content_type = "text/html"

  depends_on = [aws_s3_bucket.webpage_bucket_casestudy_fhnw]
}


resource "aws_s3_object" "file_upload_error" {
  bucket      =  "${var.s3Bucket}"
  key         =  "error.html"
  acl         = "public-read"
  source      =  "${path.module}/webpage/error.html"
  content_type = "text/html"

  depends_on = [aws_s3_bucket.webpage_bucket_casestudy_fhnw]
}



resource "aws_s3_bucket_website_configuration" "case_study_webpage_fhnw" {
  bucket =  "${var.s3Bucket}"

  index_document {
    suffix = "index.html"
  }

  error_document {
    key = "error.html"
  }
  depends_on = [
    aws_s3_object.file_upload_index,
    aws_s3_object.file_upload_error
  ]
}

resource "aws_s3_bucket_policy" "prod_website" {
  bucket =  "${var.s3Bucket}"
  policy = <<POLICY
    {
        "Version": "2012-10-17",
        "Statement": [
          {
              "Sid": "PublicReadGetObject",
              "Effect": "Allow",
              "Principal": "*",
              "Action": [
                 "s3:GetObject"
              ],
              "Resource": [
                 "arn:aws:s3:::${var.s3Bucket}/*"
              ]
```

```
            },
            {
            "Effect": "Allow",
            "Principal": {
              "AWS": "arn:aws:iam::127311923021:root"
            },
            "Action": "s3:PutObject",
            "Resource": "arn:aws:s3:::${var.s3Bucket}/AWSLogs/${var.ac-
count_id}/*"
          }
        ]
    }
    POLICY
  depends_on = [aws_s3_bucket.webpage_bucket_casestudy_fhnw]
}

resource "aws_s3_access_point" "s3_accesspoint_fhnw_pcls" {
  bucket = "${var.s3Bucket}"
  name   = "lambdavaluespage"

  depends_on = [aws_s3_bucket.webpage_bucket_casestudy_fhnw]
}
#========[ S3 BUCKET ]==============


#========[ Certificates ]==============
#=============================================================
# following is the needed cert creation for HTTPS

resource "tls_private_key" "fhnw_private_casestudy" {
  algorithm = "RSA"
  rsa_bits = "4096"
}

# The error for a missing value can be ignored (new Terraform rule, field is
READONLY)
/*
resource "tls_self_signed_cert" "cert_fhnw_casestudy" {

  private_key_pem = tls_private_key.fhnw_private_casestudy.private_key_pem

  subject {
    common_name  = "${var.s3Bucket}.s3.amazonaws.com"
    organization = "FHNW PCLS CaseStudy"
  }

  validity_period_hours = 12

  allowed_uses = [
    "key_encipherment",
    "digital_signature",
    "server_auth",
  ]

  depends_on = [tls_private_key.fhnw_private_casestudy]
}


resource "aws_acm_certificate" "cert" {
  private_key      = tls_private_key.fhnw_private_casestudy.private_key_pem
```

```
  certificate_body = tls_self_signed_cert.cert_fhnw_casestudy.cert_pem

  depends_on = [tls_self_signed_cert.cert_fhnw_casestudy]
}

resource "aws_iam_server_certificate" "fhnw_cert" {
  name             = "fhnw_cert"
  private_key      = tls_private_key.fhnw_private_casestudy.private_key_pem
  certificate_body = tls_self_signed_cert.cert_fhnw_casestudy.cert_pem

  depends_on = [tls_self_signed_cert.cert_fhnw_casestudy]

}
*/
# ====================================================================
#========[ Certificates ]==============


#========[ LoadBalancer ]==============
resource "aws_elb" "loadbalancer_casestudy_fhnw" {
  name               = "${var.laodbalancer}"
  availability_zones = ["us-east-1a", "us-east-1b", "us-east-1c"]


  listener {
    instance_port     = 8000
    instance_protocol = "http"
    lb_port           = 80
    lb_protocol       = "http"
  }

  # comment in for HTTPS

  listener {
    instance_port      = 8000
    instance_protocol  = "http"
    lb_port            = 443
    lb_protocol        = "https"
    ssl_certificate_id = "arn:aws:iam::${var.account_id}:server-certifi-
cate/fhnw_cert" #change to your AWS id
  }

  health_check {
    healthy_threshold   = 2
    unhealthy_threshold = 2
    timeout             = 3
    target              = "HTTP:8000/"
    interval            = 30
  }


  access_logs {
    bucket = "${var.s3Bucket}"
  }
  cross_zone_load_balancing   = true
  idle_timeout                = 400
  connection_draining         = true
  connection_draining_timeout = 400

  tags = {
```

```
    Name = "loadbalancer-http-8080"
  }

  depends_on = [
    aws_s3_bucket_policy.prod_website,
    # comment the fhnw_cert if already run once
    #aws_iam_server_certificate.fhnw_cert
  ]
}
#========[ LoadBalancer ]==============

#========[ Lambda ]==============
# deploy lambda function
resource "aws_lambda_function" "lambda_aws_cli" {
  filename         = data.archive_file.zip.output_path
  source_code_hash = data.archive_file.zip.output_base64sha256
  function_name             = "${var.lambdaname}"
  role                      = "arn:aws:iam::${var.account_id}:role/LabRole"
  handler          = "func.handler"
  runtime          = "python3.8"


  depends_on = [
    aws_cloudwatch_log_group.aws_cli_log_group
  ]

  tags = var.tags
}

data "archive_file" "zip" {
  type        = "zip"
  source_dir = "../Lambda/"
  output_path = "../Lambda/packedlambda.zip"
}

resource "aws_lambda_function_url" "casestudy_lambda_url" {
  function_name      = aws_lambda_function.lambda_aws_cli.function_name
  authorization_type = "NONE"
  depends_on = [aws_lambda_function.lambda_aws_cli]

  cors {
    allow_credentials = true
    allow_origins     = ["http://${var.s3Bucket}.s3-website-us-east-1.amazo-
naws.com"]
    allow_methods     = ["*"]
    allow_headers     = ["date", "keep-alive", "content-type", "Access-Con-
trol-Allow-Origin", "Access-Control-Allow-Headers"]
    expose_headers    = ["keep-alive", "date", "Access-Control-Allow-Origin",
"Access-Control-Allow-Headers"]
    max_age           = 86400
  }
}
#========[ Lambda ]==============



#========[ API-GateWay ]==============
resource "aws_apigatewayv2_api" "lambda_casestudy_api_gateway" {
  name          = "LambdaAPI"
  protocol_type = "HTTP"
```

```
  cors_configuration {
    allow_origins = ["*"]
    allow_methods = ["*"]
    allow_headers = ["date", "keep-alive", "content-type", "Access-Control-
Allow-Origin", "Access-Control-Allow-Headers"]
    expose_headers    = ["keep-alive", "date", "Access-Control-Allow-Origin",
"Access-Control-Allow-Headers"]
    max_age = 86400
  }
  depends_on = [aws_lambda_function_url.casestudy_lambda_url]
}

resource "aws_apigatewayv2_stage" "lambda_casestudy_api_gateway" {
  api_id = aws_apigatewayv2_api.lambda_casestudy_api_gateway.id

  name        = "test"
  auto_deploy = true
}

resource "aws_apigatewayv2_integration" "lambda_trigger" {
  api_id = aws_apigatewayv2_api.lambda_casestudy_api_gateway.id

  integration_uri    = aws_lambda_function.lambda_aws_cli.invoke_arn
  integration_type   = "AWS_PROXY"
  integration_method = "POST"

}

resource "aws_apigatewayv2_route" "lambda_trigger" {
  api_id = aws_apigatewayv2_api.lambda_casestudy_api_gateway.id

  route_key = "POST /trigger"
  target    = "integrations/${aws_apigatewayv2_integration.lambda_trig-
ger.id}"
}

resource "aws_lambda_permission" "api_gw" {
  statement_id  = "AllowExecutionFromAPIGateway"
  action        = "lambda:InvokeFunction"
  function_name = "${aws_lambda_function.lambda_aws_cli.function_name}"
  principal     = "apigateway.amazonaws.com"

  source_arn = "${aws_apigatewayv2_api.lambda_casestudy_api_gateway.execu-
tion_arn}/*/*"
}

#========[ API-GateWay ]==============


#========[ CloudWatch ]==============
resource "aws_cloudwatch_log_group" "aws_cli_log_group" {
  name              = "/aws/lambda/casestudylambda"
  retention_in_days = 14

  tags = var.tags

}

resource "aws_iam_policy" "lambda_logging" {
  name        = "lambda_logging"
```

```
  path        = "/"
  description = "IAM policy for logging from a lambda"

  policy = <<EOF
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents"
      ],
      "Resource": "arn:aws:logs:*:*:*",
      "Effect": "Allow"
    }
  ]
}
EOF

}

resource "aws_sns_topic" "case_study_sns" {
  name = var.sns_name
}

resource "aws_sns_topic_subscription" "sns_subscription" {
  topic_arn = aws_sns_topic.case_study_sns.arn
  protocol  = "email"
  endpoint  = var.emailaddress
}

resource "aws_cloudwatch_metric_alarm" "cwa_lambda_errors" {
  alarm_name = "alarm-lambda-errors"
  comparison_operator = "GreaterThanOrEqualToThreshold"
  evaluation_periods = 1
  metric_name = "Errors"
  namespace = "AWS/Lambda"
  period = 600 # time in seconds, 10 min -> 600s
  statistic = "Sum"
  threshold = 1
  alarm_description = "Monitor errors in Lambda function"
  insufficient_data_actions = []
  dimensions = { functionName =  var.lambdaname }
  alarm_actions = [aws_sns_topic.case_study_sns.arn]
}

resource "aws_cloudwatch_metric_alarm" "cwa_s3_allrequests" {
  alarm_name = "alarm-s3-all-requests"
  comparison_operator = "GreaterThanOrEqualToThreshold"
  evaluation_periods = 1
  metric_name = "AllRequests"
  namespace = "AWS/S3"
  period = 3600 # time in seconds, 1h --> 3600
  statistic = "Sum"
  threshold = 100 # not a lot of traffic is expected, so we decided for 100
requests in 1h
  alarm_description = "Monitor all Requests (Get, Put, etc.) in S3 bucket"
  insufficient_data_actions = []
  alarm_actions = [aws_sns_topic.case_study_sns.arn]
```

```
}

resource "aws_cloudwatch_metric_alarm" "cwa_s3_400errors" {
  alarm_name = "alarm-s3-400-errors"
  comparison_operator = "GreaterThanOrEqualToThreshold"
  evaluation_periods = 1
  metric_name = "4xxErrors"
  namespace = "AWS/S3"
  period = 300 # time in seconds, 5min --> 300
  statistic = "Sum"
  threshold = 5
  alarm_description = "Monitor all HTTP 400 errors to the S3 Bucket"
  insufficient_data_actions = []
  alarm_actions = [aws_sns_topic.case_study_sns.arn]
}

resource "aws_cloudwatch_metric_alarm" "cwa_s3_500errors" {
  alarm_name = "alarm-s3-500-errors"
  comparison_operator = "GreaterThanOrEqualToThreshold"
  evaluation_periods = 1
  metric_name = "4xxErrors"
  namespace = "AWS/S3"
  period = 300 # time in seconds, 5min --> 300
  statistic = "Sum"
  threshold = 5
  alarm_description = "Monitor all HTTP 500 errors to the S3 Bucket"
  insufficient_data_actions = []
  alarm_actions = [aws_sns_topic.case_study_sns.arn]
}

resource "aws_cloudwatch_metric_alarm" "cwa_apigw_400errors" {
  alarm_name = "alarm-apigw-400-errors"
  comparison_operator = "GreaterThanOrEqualToThreshold"
  evaluation_periods = 1
  metric_name = "4XXError"
  namespace = "AWS/ApiGateway"
  period = 300 # time in seconds, 5min --> 300
  statistic = "SampleCount"
  threshold = 5
  alarm_description = "Monitor all HTTP 400 errors to the API Gateway"
  insufficient_data_actions = []
  alarm_actions = [aws_sns_topic.case_study_sns.arn]
}

resource "aws_cloudwatch_metric_alarm" "cwa_apigw_500errors" {
  alarm_name = "alarm-apigw-500-errors"
  comparison_operator = "GreaterThanOrEqualToThreshold"
  evaluation_periods = 1
  metric_name = "4XXError"
  namespace = "AWS/ApiGateway"
  period = 300 # time in seconds, 5min --> 300
  statistic = "SampleCount"
  threshold = 5
  alarm_description = "Monitor all HTTP 500 errors to the API Gateway"
  insufficient_data_actions = []
  alarm_actions = [aws_sns_topic.case_study_sns.arn]
}

# Request count, can later also be interesting for billing estimation
resource "aws_cloudwatch_metric_alarm" "cwa_apigw_requests" {
```

```
  alarm_name = "alarm-apigw-requests"
  comparison_operator = "GreaterThanOrEqualToThreshold"
  evaluation_periods = 1
  metric_name = "Count"
  namespace = "AWS/ApiGateway"
  period = 3600 # time in seconds, 1h --> 3600
  statistic = "SampleCount"
  threshold = 100
  alarm_description = "Monitor all Requests to the API Gateway"
  insufficient_data_actions = []
  alarm_actions = [aws_sns_topic.case_study_sns.arn]
}

resource "aws_cloudwatch_metric_alarm" "cwa_cdn_error_rate" {
  alarm_name = "alarm-cdn-error-rate"
  comparison_operator = "GreaterThanOrEqualToThreshold"
  evaluation_periods = 1
  metric_name = "Sum"
  namespace = "AWS/ApiGateway"
  period = 300 # time in seconds, 5min --> 300
  statistic = "SampleCount"
  threshold = 5
  alarm_description = "Monitor all Requests to the API Gateway"
  insufficient_data_actions = []
  alarm_actions = [aws_sns_topic.case_study_sns.arn]
}

#========[ CloudWatch ]==============


#========[ CloudFront ]==============
resource "aws_cloudfront_distribution" "webpage_cf_cdn_casestudy_fhnw" {
  origin {
    //domain_name = aws_s3_bucket.webpage_bucket_casestudy_fhnw.website_end-
point
    domain_name = aws_s3_bucket.webpage_bucket_casestudy_fhnw.bucket_do-
main_name
    origin_id   = "http://${var.s3Bucket}.s3-website-us-east-1.amazonaws.com"

    custom_origin_config {
      http_port              = 80
      https_port             = 443
      origin_protocol_policy = "http-only"
      origin_ssl_protocols   = ["TLSv1.2"]
    }
  }

  enabled             = true
  is_ipv6_enabled     = true
  default_root_object = "index.html"

  custom_error_response {
    error_caching_min_ttl = 0
    error_code            = 404
    response_code         = 200
    response_page_path    = "/error.html"
  }


  default_cache_behavior {
```

```
    allowed_methods        = ["GET", "HEAD", "OPTIONS", "PUT", "POST",
"PATCH", "DELETE"]
    cached_methods         = ["GET", "HEAD", "OPTIONS"]
    target_origin_id       = "http://${var.s3Bucket}.s3-website-us-east-
1.amazonaws.com"
    viewer_protocol_policy = "redirect-to-https"

    forwarded_values {
      headers      = []
      query_string = true

      cookies {
        forward = "all"
      }
    }

  }

  restrictions {
    geo_restriction {
      restriction_type = "whitelist"
      locations        = ["IN", "US", "CA", "CH"]
    }
  }

  tags = var.tags



  viewer_certificate {
    cloudfront_default_certificate = true
  }


  depends_on = [
    aws_s3_bucket_policy.prod_website
  ]


}
#========[ CloudFront ]==============
```