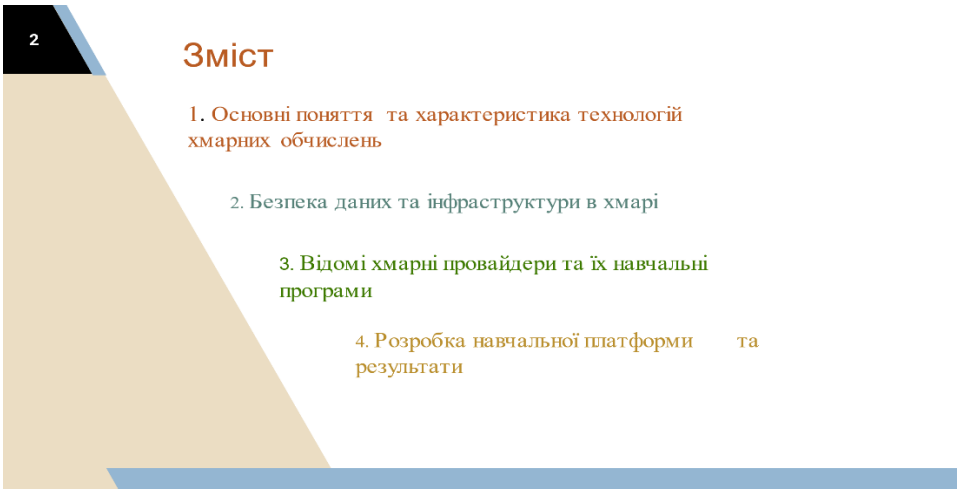
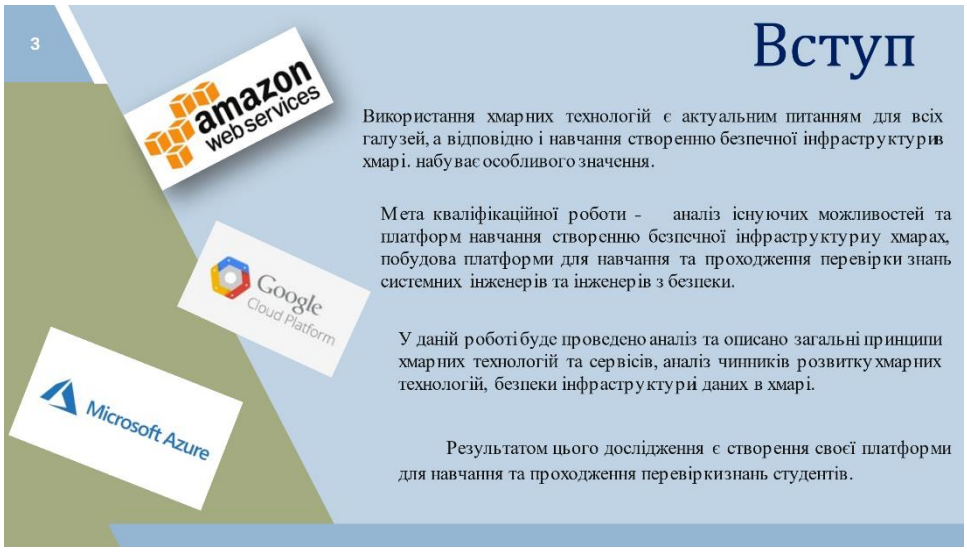
# ДОДАТОК А
# СЛАЙДИ ПРЕЗЕНТАЦІЇ

4

# 1. ОСНОВНІ ПОНЯТТЯ ТА ХАРАКТЕРИСТИКА ТЕХНОЛОГІЙ ХМАРНИХ ОБЧИСЛЕНЬ

Let’s start with the first set of slides


*Хмара - це середовище для зберігання та обробки даних, яке об'єднує в собі апаратні засоби, програмне забезпечення, мережу і підтримку користувачів.*

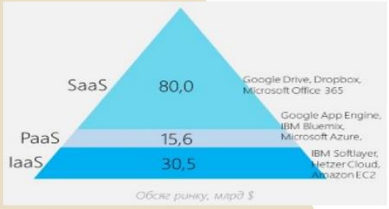



*Рисунок 1.2 – Основні види хмарних сервісів*

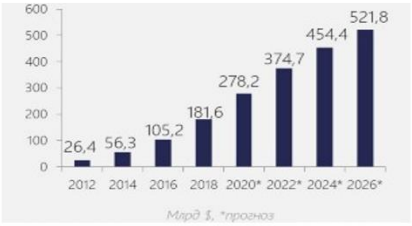
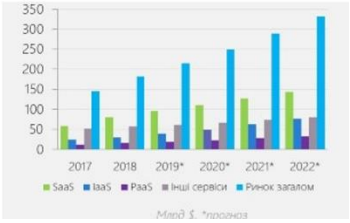



*Рисунок 1.1 - Обсяг світового хмарного ринку*

*Рисунок 1.3 – Найпопулярніші хмарні сервіси*



7

## Основні переваги хмарних технологій для бізнесу:

- відмовостійкість;
- підвищена безпека та надійність;
- висока швидкість обробки даних;
- зниження витрат на апаратне і програмне забезпечення, на обслуговування і електроенергію;
- споживані ресурси можуть миттєво масштабуватися на вимогу замовника, в залежності від зміни навантажень;
- непомітна для користувачів міграція на нові версії і нові платформи, замовники завжди працюють з найостаннішими версіями додатків;
- потрібні сервіси доступні практично миттєво не потрібно ніяких додаткових робіт по розгортанню і конфігурації використовуваних інформаційних систем;

## МАЙБУТНЄ ХМАРНИХ ТЕХНОЛОГІЙ

*Основні тенденції розвитку хмарних технологій в майбутньому:*

- *Гібридна хмара та мультихмарна інфраструктура*
- *IoT (Інтернет речей)*
- *Штучний інтелект*
- *Безсерверні обчислення*
- *Резервне копіювання та аварійне відновлення*
- *Контейнери та Kubernetes*
- *DevSecOps*

9

# 2. Безпека даних та інфраструктури в хмарі

10

## Безпека хмарної інфраструктури

**Провайдер повинен:**

- безпека віртуального середовища клієнта це один з базових елементів, на яких тримається бізнес хмарного провайдера;
- відповідальність провайдера перед компанією-клієнтом регулюється угодою про рівень послуг (SLA);
- хмарний провайдер вкладає значні ресурси в розвиток систем захисту і збереження даних. Сюди можна включити як програмні продукти і апаратні засоби, так і оперативний контроль IT-фахівців над функціонуванням хмари;
- спрямованість дій, обумовлених світовими стандартами безпеки CIS Benchmark, HIPAA, PCI DSS, ISO 27001, NIST CSF v1.1, FedRAMP та інші.

**Відповідальність провайдера :**

- ✓ забезпечення безперебійності роботи обладнання;
- ✓ забезпечення безпеки фізичної інфраструктури;
- ✓ управління хост-системами.

- Постачальник послуг відповідає за безпеку основних компонентів хмари: мережі, накопичувачі, сервери і віртуалізація.
- У кожного провайдера є величезний штат співробітників з великим досвідом, які контролюють всі процеси і підтримують користувачів.

11

## Безпека хмарної інфраструктури

**Відповідальність клієнта :**

- управління та обслуговування систем контролю доступу;
- управління політиками і правами доступу користувачів, включаючи парольний захист;
- стандартне управління оновленнями ОС і додатків;
- ведення та аналіз реєстраційних журналів;
- моніторинг активності користувачів.

Згідно з результатами дослідження «CSI Computer Crime and Security Survey», : з DoS-атаками зіткнулися тільки 17% респондентів, а з шкідливими діями персоналу - 25%.

Опитуванняна RSA Conference, показало, що сьогодні 63% фахівців в області безпеки вважають співробітників найбільш небезпечною загрозою для компаній.

12

# 3. ВІДОМІ ХМАРНІ ПРОВАЙДЕРИ ТА ЇХ НАВЧАЛЬНІ ПРОГРАМИ

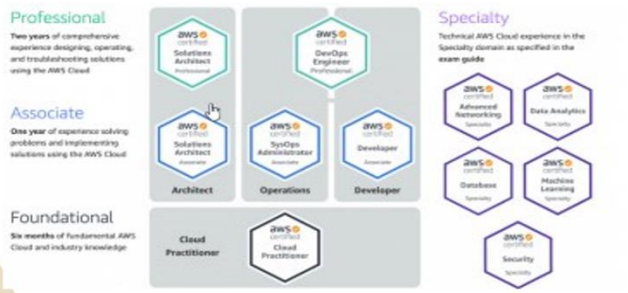



Рисунок 2.1 - *Сертифікації AWS*

- Amazon Web Services
- Microsoft Azure
- Google cloud platform
- Сертификация IBM Cloud
- Cloud Security Alliance


Рисунок 2.3 - *Gartner Magic Quadrant*

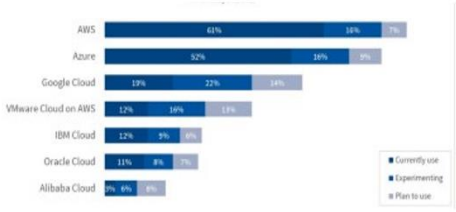



Рисунок 2.2 – *Частка хмарного ринку, яку займають провайдери*

16

# 4. Результати та висновки дослідження

17

## ОСНОВНІ НАПРЯМКИ РОЗРОБКИ ПРОЕКТУ

AUTOMATION DEPLOYMENT

– автоматизація розгортання курсу та інших процесів візуалізація процесу проходження курсу

MONITORING

– створення механізмів контролю проходження (старт проходження, та кінець, аналіз прогресу проходження користувача);

MULTI-USER CAPABILITY

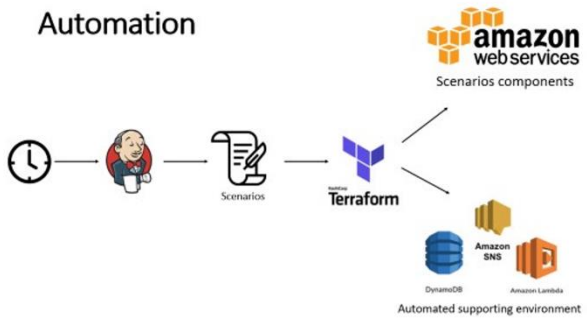
– мультикористувацький режим (можливість одночасного проходження сценаріїв користувачами, оптимізація ресурсів)

NOTIFICATION

– розробка механізму нотифікації (старт курсу, підписка на нотифікації, старт раундів, проходження сценаріїв, завершення курсу)

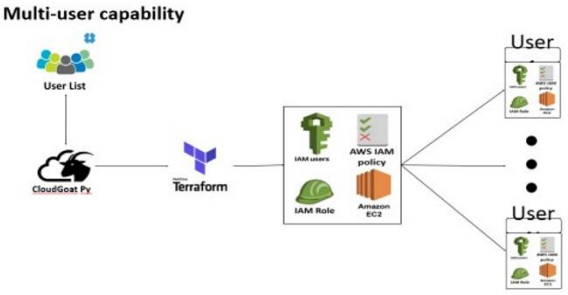
18

## АВТОМАТИЗАЦІЯ



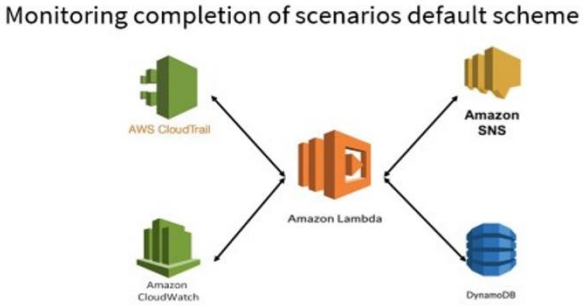

19

## МУЛЬТИКОРИСТУВАЦЬКИЙ РЕЖИМ



20

## МОНІТОРИНГ ТА КОТИФІКАЦІЇ




21

## Результати дослідження

| Учасники | Досвід | Кількість пройдених рівнів | Час проходження | Використані підказки | Не пройдені рівні |
|---|---|---|---|---|---|
| Учасник 1 | Senior | 5 | 6 год. 37 хв. | - | - |
| Учасник 2 | Senior | 5 | 7 год. 7 хв. | - | - |
| Учасник 3 | Senior | 4 | 7 год. 11хв. | - | 5 |
| Учасник 4 | Middle | 5 | 6 год. 45 хв. | 1 | - |
| Учасник 5 | Middle | 5 | 6 год. 28 хв. | 2 | - |
| Учасник 6 | Middle | 5 | 7 год. 30 хв. | 3 | - |
| Учасник 7 | Middle | 4 | 5 год. 42 хв. | 2 | 5 |
| Учасник 8 | Middle | 3 | 3 год. 7хв. | 2 | 3, 5 |

*Рисунок 4.1 – Результати проходження рівнів платформи*

| | | | | | |
|---|---|---|---|---|---|
| Учасник 9 | Junior | 4 | 6 год. 39 хв. | 4 | 5 |
| Учасник10 | Junior | 4 | 6 год. 14 хв. | 5 | 5 |
| Учасник 11 | Junior | 3 | 4 год. 12 хв. | 1 | 3, 5 |
| Учасник 12 | Junior | 3 | 2 год. 15 хв. | 3 | 4, 5 |
| Учасник 13 | Junior | 3 | 3 год. 46 хв. | 2 | 4, 5 |
| Учасник 14 | Student | 3 | 4 год. 10 хв. | 7 | 4, 5 |
| Учасник 15 | Student | 2 | 2 год. 16 хв. | 4 | 3, 4, 5 |

*Рисунок 4.1а Результати проходження рівнів платформи*

22

# Дякую за увагу!

# ДОДАТОК Б
# КОД ПРОЕКТУ

```
import json
from datetime import datetime
import boto3, time, os
from boto3.dynamodb.conditions import Key, Attr


def lambda_handler(event, context):
    dynamodb = boto3.resource('dynamodb')
    table = dynamodb.Table('users')
    sns = boto3.client('sns')
    scenario_id = os.environ['ScenarioName']

    # Takes away the date from the script completion date and sends messages to tho
se who didn't make it 24 hours before the script completion.
    for item in table.scan()['Items']:
        if item['scenarios'][scenario_id]['passed'] == False:
            if (datetime.strptime(item['scenarios'][scenario_id]['destroy_date'], "
%Y-%m-%dT%H:%M:%SZ") - datetime.utcnow()).days <= 1 :
                print("Alert sent to" item['email'])
                sns.publish(
                    TopicArn = item['topic_arn'],
                    Message = 'You have 24 hours to complete the scenario, after th
at time it will be destroyed.'
                )
```

```
import boto3, os, re, json, time
from boto3.dynamodb.conditions import Key, Attr
from datetime import datetime

send_regex = re.compile(r'^(secret|access|username|target).*$')
dynamodb = boto3.resource('dynamodb')
table = dynamodb.Table('users')
lambda_client = boto3.client('lambda')

def lambda_handler(event, context):
    response = table.scan()
    items = response['Items']

    scenario_id = os.environ['ScenarioName']
    with table.batch_writer() as batch:
        for item in items:

            if "scenarios" in item:
                time.sleep(5)
                print("working on", item['email'])
```

```
                for key in list(item["scenarios"][scenario_id]):
                    print('Found something like secret/access key')
                    try :
                        placeholders = {}
                        for key, value in item["scenarios"][scenario_id].items():
                            if send_regex.search(key):
                                placeholders[key] = value
                    except KeyError :
                        print("Keys not Exists")
                    print('Sending email')
                    lambda_client.invoke(
                        FunctionName='SendingNotifications',
                        InvocationType='Event',
                        Payload=json.dumps({
                            'template_name': os.environ['ScenarioName'] + '_data',
                            'email': item['email'],
                             'placeholders': placeholders
                        })
                    )
                    break
```

```
# Database creation
 resource "aws_dynamodb_table" "users" {
   hash_key         = "email"
   name             = var.dbName
   stream_enabled   = true
   stream_view_type = "NEW_AND_OLD_IMAGES"
   billing_mode = "PAY_PER_REQUEST"

   attribute {
     name = "email"
     type = "S"
   }
 }

# Creation items for dynamodb table
  resource "aws_dynamodb_table_item" "users" {
  count = length(local.emails)
  table_name = aws_dynamodb_table.users.name
  hash_key   = aws_dynamodb_table.users.hash_key
  item = <<ITEM
  {
    "email"        : { "S": "${local.emails[count.index]}" },
    "category"     : { "S": "${local.categories[count.index]}" },
    "notifications": { "M": {"test" :{ "S":"sometext"} } },
    "scenarios":
    {
        "M":
        {
            "${var.scenarios[1]}":
```

```
            {
                "M":
                {
                    "passed"      : { "BOOL": false },
                    "cheat"       : { "BOOL": false },
                    "deploy_date" : { "S"   : "${timestamp()}" },
                    "destroy_date": { "S"   : "${timeadd(timestamp(), "4h")}"},
                    "events" : { "L": [] }
                }
            },
            "${var.scenarios[2]}":
            {
                "M":
                {
                    "passed"      : { "BOOL": false },
                    "cheat"       : { "BOOL": false },
                    "deploy_date" : { "S"   : "1580900058" },
                    "destroy_date": { "S"   : "1580908058" },
                    "events"      : { "L"   : [] }
                }
            },
            "${var.scenarios[3]}":
            {
                "M":
                {
                    "passed"         : { "BOOL": false },
                    "cheat"          : { "BOOL": false },
                    "deploy_date"    : { "S"   : "1580900058" },
                    "destroy_date"   : { "S"   : "1580908058" },
                    "instance_events": { "L"   : [] },
                    "user_events"    : { "L"   : [] }
                }
            },
            "${var.scenarios[4]}":
            {
                "M":
                {
                    "passed"      : { "BOOL": false },
                    "cheat"       : { "BOOL": false },
                    "deploy_date" : { "S"   : "1580900058" },
                    "destroy_date": { "S"   : "1580908058" },
                    "events"      : { "L"   : [] }
                }
            },
            "${var.scenarios[5]}":
            {
                "M":
                {
                    "passed"      : { "BOOL": false },
                    "cheat"       : { "BOOL": false },
                    "deploy_date" : { "S"   : "1580900058" },
```

```
                    "destroy_date": { "S"   : "1580908058" }
                }
            },
        }
    }
  }
ITEM
}
```

```
# Create lambda_function
resource "aws_lambda_function" "NotifyScenarioEnding" {
  filename         = "../lambda/NotifyScenarioEnding.zip"
  function_name    = "NotifyScenarioEnding"
  role             = aws_iam_role.SendInitData.arn
  handler          = "NotifyScenarioEnding.lambda_handler"
  source_code_hash = filebase64sha256("../lambda/NotifyScenarioEnding.zip")
  runtime          = "python3.8"
  timeout          = 900
  environment {
    variables = {
      ScenarioName = "scenario1"
    }
  }
}
#CloudWatch event rule and target for Lambda
resource "aws_cloudwatch_event_rule" "NotifyScenarioEnding" {
  name                = "NotifyScenarioEnding"
  schedule_expression = "rate(24 hours)"
}
resource "aws_cloudwatch_event_target" "NotifyScenarioEnding" {
  rule      = aws_cloudwatch_event_rule.NotifyScenarioEnding.name
  target_id = "NotifyScenarioEnding"
  arn       = aws_lambda_function.NotifyScenarioEnding.arn
}
# Create lambda_function
resource "aws_lambda_function" "SendingNotifications" {
  filename         = "../assets/SendingNotifications.zip"
  function_name    = "SendingNotifications"
  role             = aws_iam_role.SendingNotifications.arn
  handler          = "SendingNotifications.lambda_handler"
  source_code_hash = filebase64sha256("../assets/SendingNotifications.zip")
  runtime          = "python3.8"
  memory_size      = 256
  timeout          = 900
}

#
Lambda permission
resource "aws_iam_role" "SendingNotifications" {
  name = "SendingNotifications"
```

```
  assume_role_policy = <<EOF
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": "sts:AssumeRole",
      "Principal": {
        "Service": "lambda.amazonaws.com"
      },
      "Effect": "Allow",
      "Sid": ""
    }
  ]
}
EOF
}
# Create Inline IAM policy for Cloud Watch
resource "aws_iam_policy" "SendingNotifications_AWSCloudWatchlogs" {
  name   = "SendingNotifications_AWSCloudWatchlogs"
  policy = <<EOF
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogStream",
                "logs:PutLogEvents",
                "logs:CreateLogGroup"
            ],
            "Resource": "arn:aws:logs:*:*:*"
        }
    ]
}
EOF
}
# Block policy_attachment
resource "aws_iam_role_policy_attachment" "SendingNotifications" {
  role       = aws_iam_role.SendingNotifications.name
  policy_arn = aws_iam_policy.SendingNotifications_AWSCloudWatchlogs.arn
}

resource "aws_iam_role_policy_attachment" "SendingNotifications2" {
  role       = aws_iam_role.SendingNotifications.name
  policy_arn = "arn:aws:iam::aws:policy/AmazonDynamoDBFullAccess"
}

resource "aws_iam_role_policy_attachment" "SendingNotifications3" {
  role       = aws_iam_role.SendingNotifications.name
```

```
  policy_arn = "arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole"
}
```

```
 # Read list of users from file
 data "local_file" "users" {
   filename = "${path.module}/../../users.txt"
 }
  # Read list of users from file
 data "local_file" "categories" {
   filename = "${path.module}/../../categories.txt"
 }

 # Import usernames from emails
 locals {
     read = split( "\n", data.local_file.users.content)
     read_categories = split( "\n", data.local_file.categories.content)
     emails = [
         for email in local.read:
            email
            if email != ""
     ]
     categories = [
       for category in local.read_categories:
            category
            if category != ""
     ]
 }
```

```
import re
import boto3
import json

# define steps event names
STEPS = ['ListAttachedUserPolicies', 'ListPolicyVersions', 'GetPolicyVersion']

# get all SetDefaultPolicyVersion event from cloudtrail
FINAL_EVENT = 'SetDefaultPolicyVersion'

def lambda_handler(event, context):
    # Clollection for all users
    users=[]

    # Collection for all event names
    eventList = []

    # Get users for scenario
    dynamodb = boto3.resource('dynamodb')
    table = dynamodb.Table('users')
    # Connect to cloudtrail by boto3
```

```
trail = boto3.client('cloudtrail')

# Get table items
items = table.scan()['Items']

# Read emails and get usernames
for item in items:
    if item['Scenarios'][0]['passed'] == False :
        match = re.search('.*@', item['email'])[0]
        user={}
        user['name'] = match[:-1]
        user['email'] = item['email']
        user['events'] = item['Scenarios'][0]['CTevents']
        user['arn'] = item['topic_arn']
        users.append(user)

# Collect and process all events without error by user
for user in users :
    events = trail.lookup_events(
    LookupAttributes=[
        {
            'AttributeKey': 'Username',
            'AttributeValue': user['name']
        },
    ])

    # Get all events
    for event in events['Events']:
        if event['detail']['requestParameters'] != None :
            eventList.append(event['EventName'])

    # Get only unique events for db and trail
    if user['events'] == {} : user['events']=[]
    user['events'] = list(set((eventList + user['events'])))

    # Check completion
    if user['events'].count(FINAL_EVENT) !=0:
        print("works")
        # Check is user a cheater
        try:

            # Check steps existing at log
            for step in STEPS:
                if eventList.index(step) >= 0:
                    user['cheat'] = False

        except ValueError:
            user['cheat'] = True

        # Send congratulation email
        send_message(user['arn'], user['cheat'])
```

```
            # Update user at db CTevents
            update = table.update_item(
            Key={
                    'email': user['email']
                },
                UpdateExpression="set Scenarios[0].#passed = :r, Scenarios[0].#chea
t = :c, Scenarios[0].#ev = :l",
                ExpressionAttributeNames = {
                "#passed" : "passed",
                "#cheat" : "cheat",
                "#ev" : "CTevents"
                },
                ExpressionAttributeValues={
                    ':r': True,
                    ':c': user['cheat'],
                    ':l': user['events']
                },
                ReturnValues="UPDATED_NEW"
            )
            print(update)
        else :
            # Update user at db CTevents
            update = table.update_item(
                Key={
                    'email': user['email']
                },
                UpdateExpression="set Scenarios[0].#ev = :l",
                ExpressionAttributeNames = {
                    "#ev" : "CTevents"
                },
                ExpressionAttributeValues = {
                    ':l' : user['events']
                },
                ReturnValues="UPDATED_NEW"
            )
            print(update)
# Send congrats message
def send_message(topicArn, cheat):
    sns = boto3.client('sns')
    if not cheat :
        cheater = False
        sns.publish(
            TopicArn = topicArn,
            Message = 'Congratulations, you are successfully comlepted first scenar
io of EPAM AWS security challenge'
        )
    else :
        cheater = True
        sns.publish(
            TopicArn = topicArn,
```

```
            Message = 'Congratulations, you are successfully comlepted first scenar
io of EPAM AWS security challenge, but you are cheater!!!'
        )
```

```
import re, boto3, json, os

# define steps event names
STEPS = ['ListAttachedUserPolicies', 'ListPolicyVersions', 'GetPolicyVersion']

# get all SetDefaultPolicyVersion event from cloudtrail
FINAL_EVENT = 'SetDefaultPolicyVersion'

def lambda_handler(event, context):
    # Clollection for all users
    users=[]

    # Get scenario name from environment
    scenario_id = os.environ['ScenarioName']

    # Get users for scenario
    dynamodb = boto3.resource('dynamodb')
    table = dynamodb.Table('users')

    # Connect to cloudtrail by boto3
    trail = boto3.client('cloudtrail')

    # Get table items
    items = table.scan()['Items']

    # Read emails and get usernames
    for item in items:
        if item['scenarios'][scenario_id]['passed'] == False :
            user={}
            user['username'] = item['scenarios'][scenario_id]['username']
            user['email'] = item['email']
            user['events'] = item['scenarios'][scenario_id]['user_events']
            user['topic_arn'] = item['topic_arn']
            users.append(user)

    # Collect and process all events without error by user
    for user in users :
        # Collection for all event names
        eventList = []

        events = trail.lookup_events(
        LookupAttributes=[
            {
```

```
                'AttributeKey': 'Username',
                'AttributeValue': user['username']
            },
        ])['Events']

        # Get all events
        for event in events :
            try :
                json.loads(event['CloudTrailEvent'])['errorCode']
            except KeyError:
                eventList.append(event['EventName'])

        # Get only unique events for db and trail
        if user['events'] == {} : user['events']=[]
        user['events'] = list(set((eventList + user['events'])))

        # Check completion
        if FINAL_EVENT in user['events'] :
            print("works")
            # Check is user a cheater
            try:

                # Check steps existing at log
                for step in STEPS:
                    if eventList.index(step) >= 0:
                        user['cheat'] = False

            except ValueError:
                user['cheat'] = True

            # Send congratulation email
            send_message(user['topic_arn'], user['cheat'])

            # Update user at db user_events
            update = table.update_item(
            Key={
                    'email': user['email']
                },
                UpdateExpression="set scenarios.scenario1.#passed = :r, scenarios.s
cenario1.#cheat = :c, scenarios.scenario1.#ev = :l",
                ExpressionAttributeNames = {
                "#passed" : "passed",
                "#cheat" : "cheat",
                "#ev" : "user_events"
                },
                ExpressionAttributeValues={
                    ':r': True,
                    ':c': user['cheat'],
                    ':l': user['events']
                },
                ReturnValues="UPDATED_NEW"
```

```
            )
            print(update)
        else :
            # Update user at db user_events
            update = table.update_item(
                Key={
                    'email': user['email']
                },
                UpdateExpression="set scenarios.scenario1.#ev = :l",
                ExpressionAttributeNames = {
                    "#ev" : "user_events"
                },
                ExpressionAttributeValues = {
                    ':l' : user['events']
                },
                ReturnValues="UPDATED_NEW"
            )
            print(update)

# Send congrats message
def send_message(topicArn, cheat):
    sns = boto3.client('sns')
    if not cheat :
        cheater = False
        sns.publish(
            TopicArn = topicArn,
            Message = 'Congratulations, you are successfully comlepted first scenar
io of EPAM AWS security challenge'
        )
    else :
        cheater = True
        sns.publish(
            TopicArn = topicArn,
            Message = 'Congratulations, you are successfully comlepted first scenar
io of EPAM AWS security challenge, but you are cheater!!!'
        )
```

```
import re, boto3, json, os

# Define steps event names
STEPS = ['ListFunctions20150331']

# Get users for scenario
dynamodb = boto3.resource('dynamodb')
table = dynamodb.Table('users')

# Initialize user dictionary
user = {}

# Get scenario name from environment
scenario_id = os.environ['ScenarioName']
```

```
# Main handler
def lambda_handler(event, context):
    item = None
    for key in event.keys():
        try:
            # Get user from DB
            item = table.scan(
                FilterExpression = "scenarios.#scenario.username = :i",
                ExpressionAttributeNames = {
                    "#scenario" :scenario_id
                },
                ExpressionAttributeValues = {
                    ":i" : event[key]
                }
            )['Items'][0]
            user['name'] = event[key]
            break
        except IndexError:
            print("Exception user " + event[key] + "not exists")
            continue
    if item == None:
        return 1

    # Read user data to map
    if item['scenarios'][scenario_id]['passed'] == False :
        print("user exists")
        user['events'] = item['scenarios'][scenario_id]['events']
        user['arn'] = item['topic_arn']
        user['cheat'] = False
        user['email'] = item['email']

        # Get trail events and created instance id
        user['events'] = get_trail_events_by_user(user['name'])

        # Check cheating of user
        if len(subfinder(user['events'], STEPS)) == 0 :
            user['cheat'] = True

        # update user at db
        updated = table.update_item(
            Key={
                'email': user['email']
            },
            UpdateExpression="set scenarios.#scenario.passed = :r, scenarios.#scena
rio.#cheat = :c, scenarios.#scenario.#uev = :u",
            ExpressionAttributeNames = {
                "#scenario" : scenario_id,
                "#cheat"    : "cheat",
                "#uev"      : "events"
            },
```

```
            ExpressionAttributeValues={
                ':r': True,
                ':c': user['cheat'],
                ':u': user['events']
            },
            ReturnValues="UPDATED_NEW"
        )
        print(updated)

        # Send congratulation email
        send_message(user['arn'], user['cheat'])
        return

def get_trail_events_by_user(username):

    # Connect to cloudtrail by boto3
    trail = boto3.client('cloudtrail')

    # Collect and process all events by username
    events = trail.lookup_events(
    LookupAttributes=[
        {
            'AttributeKey': 'Username',
            'AttributeValue': username
        }
    ])

    # Reset list for all event names
    eventList = []

    # Get all user events without error
    for event in events['Events']:
        try :
            json.loads(event['CloudTrailEvent'])['errorCode']
        except KeyError:
            eventList.append(event['EventName'])

    return list(set(eventList))


def send_message(topicArn, cheat):
    sns = boto3.client('sns')
    if not cheat :
        cheater = False
        sns.publish(
            TopicArn = topicArn,
            Message = 'Congratulations, you are successfully comlepted scenario #4
of EPAM AWS security challenge'
        )
        print("msg sent not cheat")
    else :
```

```
        cheater = True
        sns.publish(
            TopicArn = topicArn,
            Message = 'Congratulations, you are successfully comlepted scenario #4
of EPAM AWS security challenge, but you are cheater!!!'
        )
        print("msg sent cheat")

def subfinder(mylist, pattern):
    pattern = set(pattern)
    return [x for x in mylist if x in pattern]
```

```
import json, boto3, os
#from boto3.dynamodb.conditions import Key, Attr
from datetime import datetime

def lambda_handler(data, context):
    dynamodb = boto3.resource('dynamodb')
    table = dynamodb.Table('users')

    # Get mail current user
    email = os.environ['UserEmail']

    # Get scenario id from environment
    scenario_id = os.environ['ScenarioName']

    # Get logs for user email
    logs = log_getting(scenario_id, email)

    if len(data.values()) > 1 :
        return{
            'Message':'To many keys, I`m unbreakable!!!',
            'Tip':'If you wanna complete scenario try to uset the correct payload',
            'TheCorrectPayload':'{"Key":"your_secretkey_from_rds"}'
        }
    elif len(data.values()) == 0 :
        return{
            'Message':'Try to add payload',
            'Tip':'If you wanna complete scenario, try to use the correct payload',
            'TheCorrectPayload':'{"Key":"your_secretkey_from_rds"}'
        }

    for key in data.values():
        try:
            # Get user information from DB
            user = table.scan(
                FilterExpression = 'scenarios.#scenario.completion_key = :i',
                ExpressionAttributeNames = {
                    "#scenario": scenario_id
                },
```

```
                ExpressionAttributeValues = {
                    ":i" : key
                },
                ProjectionExpression = ' \
                    email, \
                    scenarios.#scenario.passed \
                '
            )['Items'][0]

            # Verifying that the key belongs to the user
            if email == user['email'] :
                if not user['scenarios'][scenario_id]['passed'] :
                    #  Change of state passing scenario to true
                    total_time = countTotalTime(logs['check_user_trail1']['events']
,logs['check_user_trail2']['events'])
                    updated = table.update_item(
                        Key={
                            'email': user['email']
                        },
                        UpdateExpression='SET scenarios.#scenario.#passed = :l, sce
narios.#scenario.#total_time = :t',
                        ExpressionAttributeNames = {
                            "#scenario" : scenario_id,
                            "#passed"   : "passed",
                            "#total_time" : "total_time"
                        },
                        ExpressionAttributeValues={
                            ':l': True,
                            ':t': total_time
                        },
                        ReturnValues="UPDATED_NEW"
                    )

                    # Send congrats message
                    print(updated)
                    sendEmail(email)
                    return ("Congratulations, the key: " + key + " is yours\n")
                else :
                    return ("Congratulations, the key: " + key + " is yours, but yo
u are already pass it")
            else :
                return ("The key: " + key + " is not yours\n")

        except IndexError:
            return("Key " + key + " not exists\n")
        except Exception as e:
            return {
                'Error':str(e),
                'Message': 'And you tried to broke me by this, It is so unfair of y
ou!!',
                'Remember': 'I`m unbrekable!!!',
```

```
                'Tip':'If you wanna complete scenario try to uset the correct paylo
ad',
                'TheCorrectPayload':'{"Key":"your_secretkey_from_rds"}'
            }

def sendEmail(email):
    lambda_client = boto3.client('lambda')
    lambda_client.invoke(
        FunctionName='SendingNotifications',
        InvocationType='Event',
        Payload=json.dumps({
            'template_name': 'congratulations',
            'email': email,
            'placeholders': {
                'ScenarioName': os.environ['ScenarioName']
            }
        })
    )

def log_getting(scenario_id, email):

    # Connect to dynamodb table
    dynamodb = boto3.resource('dynamodb')
    table = dynamodb.Table('users')

    # Get item to update
    item = table.get_item(
        Key={ "email":email },
        ProjectionExpression = ' \
            scenarios.#scenario.username_user1, \
            scenarios.#scenario.username_user2, \
            scenarios.#scenario.user1_events, \
            scenarios.#scenario.user2_events, \
            scenarios.#scenario.passed \
        ',
        ExpressionAttributeNames = {
            "#scenario" : scenario_id
        }
    )['Item']

    # Init user collection
    user={}

    # Read user data to map
    if item['scenarios'][scenario_id]['passed'] == False :
        user['username1'] = item['scenarios'][scenario_id]['username_user1']
        user['username2'] = item['scenarios'][scenario_id]['username_user2']
        user['user1_events'] = item['scenarios'][scenario_id]['user1_events']
        user['user2_events'] = item['scenarios'][scenario_id]['user2_events']
        user['cheat'] = False
        print(user)
```

| Позначення | Найменування | Дод. відомості |
|---|---|---|
| | Текстові документи | |
| 1. | Пояснювальна записка | 73 с. |
| | Графічні документи | |
| 2. | Слайди презентації | 22 |

| Змін. | Арк. | Номер докум. | Підпис | Дата | | | |
|---|---|---|---|---|---|---|---|
| Розроб. | | Абіх | | | Дослідження шляхів підвищення ефективності навчання цифровій безпеці в AWS | Літ. | Аркуш 1 / Аркушів 1 |
| Перевір. | | Костромицький | | | | У | |
| | | | | | | ХНУРЕ кафедра ІМІ | |
| Н. контр. | | Костромицький | | | | | |
| Затв. | | Безрук | | | Відомість кваліфікаційної роботи | | |