מעבדה במחשוב ענן – חלק ב'

בחלק זה של הניסוי נתמקד בשרותים של אמאזון שלא קשורים לניהול שרתים או תשתיות אחרות. בסוג כזה של שרותים, אמאזון מחויבת לתת שרות זמין לכל עומס שיכול להיווצר, והתשלום הוא לפי שימוש.

מטרות

במפגש הזה אנחנו ניישם תהליך פשוט של עבודה באינטרנט הדברים. נדמה מכשירים מסויימים שמדווחים מידע לענן, הענן יעבד את הנתונים ויקבל החלטות. אם יש צורך (ונדאג שיהיה), הענן יעביר פקודות למכשירים אחרים.

הניסוי יעבוד במודל פשטני, על מנת שיהיה קל יחסית להבנה. מי שיתעמק בעתיד בשירות הזה, עומדים לפניו כלים יותר מתוחכמים אשר מספקים שרותים ברמה יותר גבוהה.

בתרגיל הזה המחשב שלכם יתפקד בתור רכיב (Thing) אחד או יותר, וישלח מידע על הרכיבים לענן. כל רכיב כזה ייוצג ע"י תכנית מחשב בשפת python אשר תקבל פרמטרים שמאפיינים את הרכיב.

למשל, אם אנחנו רוצים לדמות מצב בו אנחנו מחברים לשרות 2 מכונות כביסה ושואב אבק, אנחנו נריץ פעמיים סקריפט שמדמה מכונת כביסה, ועוד פעם אחת סקריפט שמדמה שואב אבק.

על מנת שלא תתחילו מאפס, הכנו עבורכם קוד בסיס אשר מבצע את המשימות. הקוד מחולק ל-2 תכניות: אחת מדווחת סטטוס לענן, והשנייה מקבלת פקודות מהענן לביצוע.

תוך כדי הניסוי אתם תערכו את קבצי הקוד כדי למלא אחר המשימות הנדרשות.

לכל אורך הניסוי, אתם תשתמשו באותם מפתחות הצפנה אשר ייצרתם במסגרת ההכנה.

תרגיל 0: הכנת העמדה

מטרת התרגיל

את המפגש הזה נבצע על מחשבי המעבדה.

בתרגיל זה נוריד את הקבצים והסקריפטים הנדרשים להמשך הניסוי. בשלב זה אנחנו נייצר סביבת IoT, בה מחשב המעבדה יתפקד בתור מכשיר שמתחבר ב-IoT. בתום התהליך, על המחשב תותקן תכנית בשפת Python אשר מתחברת אל הענן, שולחת ומקבלת לו הודעות בפרוטוקול MQTT.

בדרך יתבצעו הפעולות הבאות:

- Register: המכשיר החדש יירשם בשירות IoT
- Download: תתבצע הורדה של קובץ zip המכיל בתוכו מפתחות הצפנה, סרטיפיקט וסקריפטים
- התקנת ספריית AWS IoT Device SDK במחשב שלכם.

הכנה

על מנת להקל על תהליך ההגדרה, שרות ה-IoT כולל אשף (wizard) המוליך אותנו צעד צעד ועוזר לנו להגדיר את המכשיר. אנחנו נשתמש באשף הזה.

1. היכנסו לשרות IoT Core בחשבון שלכם
2. בצד השמאלי של המסך, בחרו Onboard. תקבלו מסך עם 3 אפשרויות.
3. בחרו באופציה השמאלית – Configure a Device (לחצו על Get Started ובמסך הבא שוב על Get Started)
4. תחת choose a platform בחרו Linux. תחת choose SDK בחרו Python.
5. Step 1/3: תנו שם כלשהו למכשיר ולחצו על Next Step
6. Step 2/3: לחצו על הכפתור תחת Download connection kit for. אחרי הלחיצה הדפדפן מוריד קובץ zip למחשב שלכם. התום ההורדה לחצו על Next Step.
7. Step 3/3: הוא עצמו מורכב מ-3 צעדים נעבור שלב שלב:

צעד 1

Step 1: Unzip the connection kit on the device

```
unzip connect_device_package.zip
```

הגדירו תיקיה חדשה, נניח AWS, ופיתחו את קובץ ה-zip בתוכה.

התיקיה מכילה:

- קבצי מפתחות שהוגדרו ספציפית עבור המכשיר החדש
- קובץ סרטיפיקט לצרכי אבטחה
- סקריפט start.sh בלינוקס.

פיתחו את הסקריפט לעריכה ונראה מה הוא מכיל:

- הבלוק השני מוודא שיש סרטיפיקט שמור באותה תיקיה. אם לא שיניתם כלום, למעשה הקוד לא ירוץ כי התנאי לא מתקיים
- הבלוק השלישי מוודא שספריית AWS IoT Device SDK קיימת תחת אותה תיקייה. בריצה ראשונה הספריה אכן לא שם, והקוד ירוץ ויוריד את הספריה בעזרת תכנת git. מהריצה השניה והלאה הקוד הזה לא יעשה כלום.
- הבלוק הרביעי מריץ תכנית דוגמא מתוך ה-SDK. זהו קטע הקוד היחיד שיפעל בכל הרצה של הסקריפט.

שורת ההרצה נראית כך:

```
python aws-iot-device-sdk-python/samples/basicPubSub/basicPubSub.py -e
a2sf29psgyb94e.iot.us-east-1.amazonaws.com -r root-CA.crt -c qqq.cert.pem -k
qqq.private.key
```

שימו לב לפרמטרים:

בכחול – תכנית הדוגמא אותה מריצים.
בירוק – כתובת Endpoint של מכשירי IoT השייכים לחשבון שלכם
בצהוב – הסרטיפיקט
בסגול – מפתחות ההצפנה.

במהלך הניסוי אנחנו נכתוב תכניות אחרות, כאשר נשתמש בתכנית הדוגמא כקוד בסיס. לצורך כך נכתוב סקריפטים דומים, המכילים רק את שורת הפקודה הזו (כי הרי הבלוקים האחרים כבר לא משחקים תפקיד). ונשנה רק את המחרוזת הכחולה שתכיל את שם התכנית שלכם.

צעד 2

הפקודה האמצעית נותנת הרשאות ריצה לסקריפט start. וודאו שההרשאות האלו נשארות בתרגילים בהם תתבקשו לשכפל את הסקריפט הזה.

Step 2: Add execution permissions

```
chmod +x start.sh
```

מתוך טרמינל, הריצו את הפקודה

Step 3: Run the start script. Messages from your thing will appear below

```
./start.sh
```

הריצה הראשונה כנראה תסתיים בהודעת שגיאה. הריצו פעם נוספת. אם יש בעיות – קראו למדריך.

אם התכנית רצה כהלכה, אתם תקבלו הדפסות מהסוג הזה:

```
Received a new message:
New Message 94
from topic:
sdk/test/Python
```

ננסה כעת להבין את החלק החשוב בתכנית.
חלקו התחתון של הקובץ נראה כך:

```
#Connect and subscribe to AWS IoT
myAWSIoTMQTTClient.connect()
myAWSIoTMQTTClient.subscribe("sdk/test/Python", 1, customCallback)
time.sleep(2)

#Publish to the same topic in a loop forever
loopCount = 0
while True:
   myAWSIoTMQTTClient.publish("sdk/test/Python","New Message "+ str(loopCount), 1)
   loopCount += 1
   time.sleep(1)
```

3 השורות הראשונות מבצעות יצירת קשר עם שרת ה-AWS (לפי פרמטרים שהוגדרו קודם) ופעולת subscribe לטופיק sdk/test/Python.

לאחר מכן יש לולאה אינסופית, בה מבצעים פעולת publish לאותו טופיק כל שניה.

מה שאמור לקרות הוא שהתכנית מגיעה ללולאה, ומבצעת את פעולת ה-publish. ההודעה מגיעה ל-AWS שמשמש גם כ-broker. הוא מפיץ את ההודעה למי שעשה subscribe שזו אותה תכנית. כשההודעה מגיעה לתכנית, נקרא ה-callback.

פונקציית ה-callback מוגדרת יחסית בתחילת הקוד, ונראית כך:

```
#Custom MQTT message callback
def customCallback(client, userdata, message):
        print("Received a new message: ")
        print(message.payload)
        print("from topic: ")
        print(message.topic)
        print("--------------\n\n")
```

הפונקציה מדפיסה חיווי שההודעה הגיעה, ואז את ה-payload וה-topic שלה.

תרגיל 1: מבוא אינטרנט הדברים IOT

מטרת התרגיל

בתרגיל הזה תורידו למחשב שלכם קוד מוכן קיים, ותתאימו אותו כך שיוכל לרוץ על המחשב שלכם ולדמות התקני IoT. לפני כן, נציג כלי עזר חשוב מאד בפיתוח ובדיקות יישום IoT.

IOT TEST

מסך זה מאפשר שליחה של הודעות MQTT וגם ביצוע subscribe לטופיקים. זהו חלק חשוב בפיתוח יישומי IoT משום שהוא מאפשר לצפות בזרימה של הודעות מהמקור ליעד. החשיבות שלו גדולה ביחוד כאשר מצפים להתנהגות מסויימת של המערכת אשר בפועל לא קורית, ובעזרת הכלי הזה מקבלים תמונה יותר טובה לגבי היכן נופלת ההודעה.

מומלץ ביותר לאורך כל המפגש להשאיר לשונית אחת פתוחה עם דף הטסט.

על מנת לתרגל קצת את הכלי, נבצע כמה פעולות שליחה וקבלה

- היכנסו למסך Test תחת שירות IoT
- בצעו subscribe לטופיקים הבאים

1. nssl/a/b/c
2. nssl/a/+/c
3. nssl/+/+/c
4. nssl/#

בצעו publish לטופיקים ברשימה מתחת (התוכן כרגע לא רלוונטי, לכן אין צורך לערוך אותו):

a. nssl/a/b/c
b. nssl/b/b/c
c. nssl/a/c/c

d. nssl/cloud/lab
e. my/personal/topic

שאלה מס' 1.2 העתיקו את הטבלה הבאה ומלאו אותה

Published topic	Accepted by subscribers
a	1,2,3,4
b	
c	
d	
e	

שימוש בסקריפטים ותכניות פייתון

כעת נעתיק את תוכניות הבסיס למחשב שלכם ונשמיש אותם. התכניות האלו נגזרו מתוך תכנית ה-basicPubSub שראיתם בשלב ההכנה. גם מבנה הפרמטרים של התכניות האלו הוא אותו מבנה. לכן לכל תכנית בסיס כזו אנחנו ניצור קובץ start מתאים, שמבוסס על קובץ ה- start שלכם.

מטרת השלב הזה היא להצליח להריץ את תכניות הבסיס בסביבה שלכם, ולהבין את מה שכתוב בתוכם.

התכניות הבסיסיות לשליחת נתונים וקבלת פקודות נמצא להורדה בלאב אדמין.

הגדרות אבטחה

הגדרות האבטחה של תכנית basicPubSub מקשיחות מאד את התנאים של שליחת הודעות MQTT. לצורך הניסוי, אנחנו נשחרר את ההגבלות האלו, על מנת לאפשר לנו גמישות יותר גדולה בהגדרות שנבצע. כמו תמיד, מומלץ למשתמשים יותר מיומנים לטרוח יותר ולקבל סביבה יותר מאובטחת. אנחנו בונים על כך שאת כל הגדרות הניסוי אנחנו מוחקים תוך זמן קצת, ולכן ניתן לאפשר הקלות מסויימות באבטחה.

1. בשירות IoT Core, תחת Secure->Policy, ליחצו על המלבן של ה-policy שהשתמשתם בו בהכנה.
2. ליחצו על Edit Policy document
3. מיחקו את כל התוכן, והכניסו במקומו את הטקסט הזה:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iot:*",
      "Resource": "*"
    }
  ]
}
```

4. ליחצו על שמירה

תכנית בסיס ל-PUBLISH

התכנית הזו מתמקדת רק בשליחת הודעת MQTT מהמחשב שלכם לענן. היא כוללת בניית אובייקט בג'ייסון ושליחה שלו.

1. הורידו את תכניות הפייתון, ושמרו אותם על מחשב שלכם, בתיקיה בה ממוקם קובץ ה-start אותו הרצתם במסגרת ההכנה.

הקבצים מקבלים את אותם פרמטרים שמקבל הקובץ basicPubSub. לכן נעתיק את הקובץ start לקובץ חדש, ונשנה אותו על מנת להריץ את התוכנית שלנו.

2. העתיקו את start שלכם לקובץ בשם start_pub. שמרו על אותה סיומת.
3. ערכו את הקובץ:
   a. מחקו את כל השורות שבודקות סרטיפיקט והתקנה של ה-SDK (כיסינו את זה בהכנה). זה אמור להשאיר אתכם עם קובץ ממש קצר.
   b. בשורת ההרצה שנו את שם קובץ הפייתון ל-basicPub.py. לא לשכוח למחוק את ה-path כי הקובץ נמצא בתיקייה שלנו ולא תחת samples.
4. שמרו את הקובץ. משתמשי לינוקס, וודאו שיש לו הרשאות הרצה (+x).
5. הריצו את קובץ ה-start_pub.
6. על מנת לוודא שהסקריפט מבצע את פעולתו, היכנסו לחלון ה- Test בו השתמשנו בסעיף הקודם, ובצעו subscribe לטופיק שמופיע בתכנית.

ניתוח התכנית

בחלקו התחתון של הקובץ אנחנו בונים אובייקט ג'ייסון ושולחים אותו. הנה הקוד:

```
data = {}
data['id'] = str(1)
data['timestamp'] = str(time.time())
data['info'] = {}
data['info']['temp'] = str(loopCount)
data['info']['humidity'] = str(80)
json_data = json.dumps(data)
```

האובייקט בנוי מ-3 שדות. הערך של 2 השדות הראשונים הוא מספר (מזהה החיישן וזמן הדגימה), ושל השלישי הוא אובייקט עם נתוני הדגימה. לאובייקט הפנימי 2 שדות משלו, טמפ' ולחות. כפי שניתן לראות, הבניה היא די פשוטה ומשתמשת ב-dictionary. הפקודה האחרונה ממירה את מבנה ה-dictionary למחרוזת בפורמט ג'ייסון.

במקרה הזה הקוד בונה את האובייקט הבא:

```
{id:1,timestamp:234,info:{temp:34,humidity:80}
```

תכנית בסיס ל-SUBSCRIBE

התכנית הזו מתמקדת רק בפעולת ה-subscribe. התכנית מגדירה לאיזה טופיק רוצים לעשות subscribe וכן callback שייקרא כאשר מגיעה הודעה מכיוון ה-message broker.

1. העתיקו את start_pub שלכם לקובץ בשם start_sub. שמרו על אותה סיומת.
2. ערכו את הקובץ:
   a. בשורת ההרצה שנו את שם קובץ הפייתון ל-basicSub.py
3. שמרו את הקובץ. משתמשי לינוקס, וודאו שיש לו הרשאות הרצה (+x).

ניתוח התכנית

בחלקו התחתון של התכנית ישנה לולאה שמדפיסה את הערך power_mod.

בפקודה מעל ללולאה, אנחנו מבצעים subscribe לטופיק מסויים. כאשר מקבלים הודעה עבור טופיק זה, תיקרא הפונקציה customCallback.

הפונקציה customCallback מוגדרת בחלק העליון של התכנית.

בתחילה, הפונקציה מדפיסה את הטופיק וה-payload של ההודעה.

לאחר מכן, היא ממירה את אובייקט הג'ייסון ל-dictionary, ומחלצת ממנו את הערך של השדה power_on.

לכן, כדי שהקוד יבצע את המוטל עליו, יש צורך להכניס ב-payload של ההודעה אובייקט ג'ייסון שבו יש את השדה power_on.

4. הריצו את קובץ ה-start_sub.
5. על מנת לוודא שהסקריפט מבצע את פעולתו, היכנסו שוב למסך ה-Test, ובצעו publish לטופיק שמופיע בתכנית, עם אובייקט ג'ייסון מתאים.

<u>שאלה מס' 1.3</u> מהו אובייקט הג'ייסון שלכם?

<u>שאלה מס' 1.4</u> שימרו הדפסה של התכנית (לא יותר מ-20 שורות) עם 2 ערכי power mod שונים.

תרגיל 2: חיישנים

עבור התרגיל הזה והבאים אחריו, אתם צריכים לקבל סיפור מקרה מהמדריך. אם לא קיבלתם אחד עד כה, בקשו זאת ממנו.

בתרגיל הזה אנחנו נממש מערכת של חיישנים, עבור סיפור המקרה. בתרגיל אנחנו נבנה מודל עבור החיישנים, ונעלה דיווח לענן.

שאלה מס' 2.1 תכננו אובייקט ג'ייסון עבור החיישנים, והציגו דוגמא אחת של האובייקט הזה. האובייקט צריך לכלול גם זמן ומזהה מכשיר, לטובת תיעוד במסדי נתונים.

שאלה מס' 2.2 תכננו מבנה של טופיק, אשר יתאים גם למקרה כללי יותר של מספר בקרים.

לפני שממשיכים, הראו את מה שבניתם למדריך לאישור.

עתה נממש את המודל בקובץ פייתון

1. העתיקו את הקובץ basicPub.py לקובץ בשם sensorPub.py ופיתחו אותו לעריכה
2. שנו את שם הטופיק לזה שלכם
3. שנו את ה-payload כך שיפרסם את האובייקט ג'ייסון שתכננתם עבור החיישנים.
4. חזרו ל-console ובדף ה-Test הוסיפו subscribe לטופיק של החיישנים.
5. הריצו את התכנית שלכם, וודאו שאתם רואים הודעות מגיעות ל-Test. אפשר להשתמש ב-start_pub אחרי שינוי קל, או להריץ את הפקודה עצמה עם שם הקובץ שלכם.
6. אחרי שראיתם הודעות, עצרו את הסקריפט.

תרגיל 3: התקנים

בתרגיל הזה אנחנו נממש התקן. ההתקן מקבל הודעה להתחיל או להפסיק עבודה. כאשר מתקבלת פקודה כזו אנחנו נדפיס הודעה.

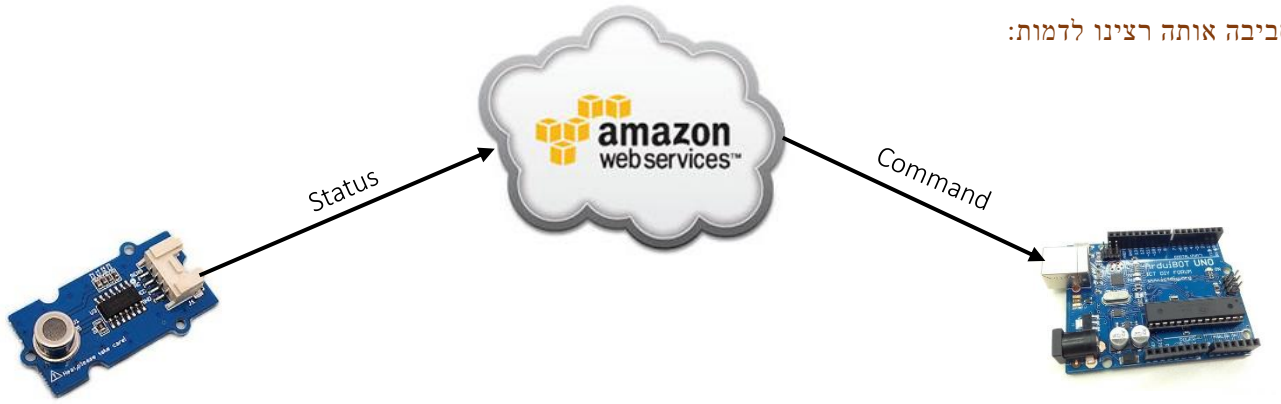
שאלה מס' 3.1 תכננו טופיק שמתאים להתקנים

נממש בפייתון

1. העתיקו את הקובץ basicSub.py לקובץ deviceSub.py
2. שנו את שם הטופיק אליו מבצעים subscribe לזה שלכם
3. עברו לקובץ ה-callback. פענחו את האובייקט, וכיתבו הדפסה המעידה האם המכשיר דלוק או כבוי.
4. הריצו את התכנית
5. ב-console, בדף ה-Test, שילחו הודעות MQTT לכיבוי והדלקה. תוודאו שסקריפט ההתקן מגיב כצפוי. שימו לב שצריך לעטוף במרכאות כפולות את כל משתני ה-JSON.
6. אחרי שראיתם הדפסה בטרמינל, אתם יכולים להפסיק את הרצת התכנית.

בשלב הזה למעשה כבר מימשנו את הסקריפטים אשר מממשים חיישנים והתקנים. הנה ציור שממחיש את הסביבה אותה רצינו לדמות, לעומת הסביבה בפועל שייצרנו:

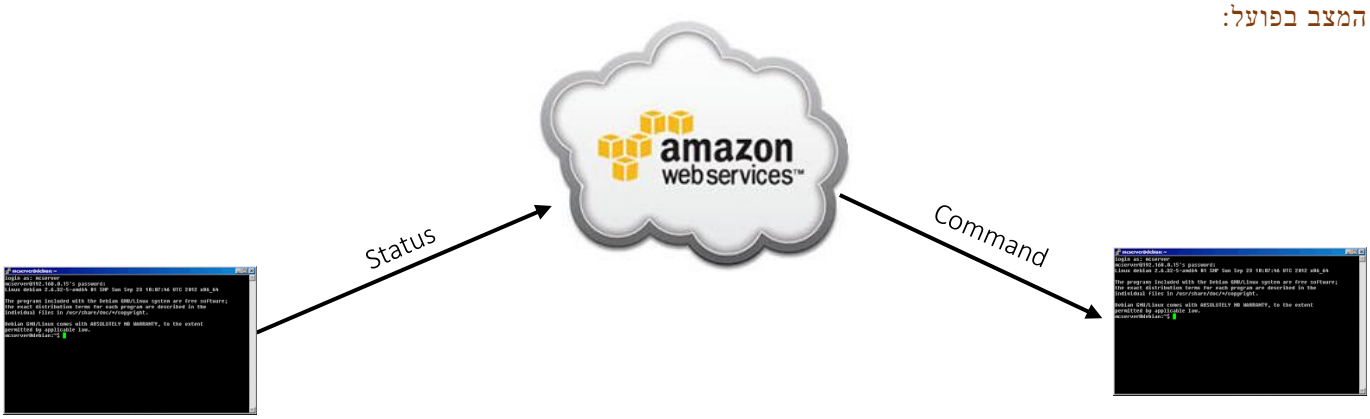
הסביבה אותה רצינו לדמות:



חיישן

התקן

המצב בפועל:



sensorPub.py

deviceSub.py

כמו כן וידאנו בעזרת ה-Test ש-2 התכניות מ-2 הצדדים רצות ומתקשרות עם השרות AWS IoT. אולם כרגע אין קשר בין 2 התכניות האלו, ובכלל הענן מקבל הודעות MQTT ולא עושה איתם שום דבר. לכך נדאג בתרגיל הבא.

תרגיל 4: פעולת REPUBLISH

פעולת Republish יודעת לקרוא הודעת MQTT, ובתגובה לשלוח הודעת MQTT. במקרה שלנו ה-AWS יקבל הודעה מהחיישן, וישלח אותה לכיוון ההתקן. ההודעה הנשלחת יכולה להיות אותה הודעה או הודעה שונה. כמו כן, ניתן להפעיל לוגיקה כדי להחליט אם לשלוח הודעה בתגובה או לא. לצורך כך נשתמש ב-rule engine אשר הוסבר עליו בחוברת ההכנה. למי שלא הבין או לא זוכר את הפרטים, כדאי לפתוח את הפרק שמסביר על כך בחוברת ההכנה.

בתרגיל הזה נפעיל קצת לוגיקה על ה-rule ובתנאים מסויימים נבצע פעולת Republish. פעולה זו שולחת הודעות MQTT אחרות, שבעזרתן התקנים יבצעו פעולות.

- ב-Console היכנסו לשירות IoT ובחרו Act
- הגדירו Rule חדש ע"י לחיצה על כפתור create a rule קיראו לו בשם Republish0.

באזור Rule Query Statement יש שדה לעריכת טקסט, בו אמורים להכניס את ה-Rule כפי שהוסבר בחוברת ההכנה. אפשר שוב להסתכל על ההסבר, או ללחוץ על AWS IoT SQL Reference כדי לקרוא את ההסברים שלהם.

מכיוון שכרגע אנחנו מעוניינים להעביר את ה-Attrubutes כפי שהם, נסמן אותם עם *. אנחנו גם עדיין לא משתמשים בתנאי. לכן הביטוי שלנו פשוט והוא

```
SELECT * FROM <topic Filter>
```

- ב-Topic Filter רישמו את הטופיק שאתם שולחים בתכנית ה-Pub לא לשכוח להקיף בגרשיים את הטופיק, אחרת הוא לא קורא אותו כמו שצריך.
- תחת Set one or more actions לחצו על Add Action
- בחלון select an action סמנו Republish messages to an AWS IoT topic ולחצו על Configure action
- ב-Topic רשמו את הטופיק אותו הגדרתם בשאלה מס' 3.1
- בשדה create a role צרו Role חדש עם השם IoTRole. נשתמש בו גם הלאה. תפקידו הוא לאפשר שליחת הודעות MQTT עם הטופיק הזה.
- בחרו את ה-Role הזה מתוך הרשימה, ולחצו על Update Role. כל פעם נשלח טופיק אחר נצטרך לעדכן את הRole שיתמוך גם בטופיק החדש, ע"י לחיצה על Update role.
- לחצו על Add action.
- במסך ה-Create a Rule לחצו על הכפתור Create Rule בתחתית המסך

בדיקה

לפני שנעבור הלאה, נוודא שהשלב הזה עבר כמו שצריך. לשם כך נחזור למסך ה-test. הפעם נבצע גם publish (לטופיק שה-Rule מאזין לו) וגם subcribe (לטופיק שה-Rule מייצר).

- עברו למסך Test
- באיזור ה-subscribe רישמו את הטופיק שה-rule מייצר
- באיזור ה-publish רישמו את הטופיק שה-rule מאזין לו.

אם הכל הולך כשורה, אתם אמורים לשלוח הודעה ולקבל את אותה הודעה אבל עם טופיק שונה.

ATTRIBUTE

עד עכשיו לא נגענו ב-Attribute וכתוצאה מכך תמיד שלחנו כפרמטר את אותו אובייקט ג'ייסון ששלחנו ב-payload.

עכשיו נשנה את ה-payload ע"י הכנסת ערך ב-Attribute .

- במסך Rules (לחיצה על Act בתפריט), בחרו את ה-Republish0
- לחצו על Edit לשדה Rule query statement
- במקום * הכניסו את הערך הבא, ולחצו על update

```
 0 AS power_on
```

נסו שוב במסך Test לעשות Publish. האם ה-payload השתנה?

שימוש ב-CONDITION ב-RULE

שדה ה-Condition מאפשר לנו לסנן הודעות המגיעות ל-Rule. למשל, אם אני רוצה להתריע רק כאשר הטמפ' גבוהה בחדר, אני אשתמש בשדה הזה כדי לגרום רק להודעות הרלוונטיות לעבור.

כדי להשתמש ב-condition, תוכן ההודעה צריך להיות בפורמט ג'ייסון. אפשר לבצע התניה על ערכים עמוקים יותר בהיררכיה, וכן התניה מרובה.

לדוגמא

נניח את האובייקט הבא

```
{
   floor: 3,
   room: 373,
   info: {
       temp: 30
       humidity: 80
   }
   timestamp: 23944035
}
```

אנחנו יכולים להגדיר תנאי כזה:

```
floor=3 AND info.temp>25
```

הביטוי כולו יקבל את המבנה הבא:

```
SELECT 0 AS power_on FROM 'nssl/myTopic' WHERE floor=3 AND info.temp>25
```

היכנסו שוב ל-Rule של Rebublish0, ולחצו על Edit עבור Rule query statement.

ערכו את שדה Rule query statement בהתאם לסיפור המקרה, והוסיפו את התנאי בסוף הביטוי. ניתן להשתמש ב-Test כדי לראות אם התנאי שהכנסתם באמת מפלטר הודעות. לשם כך הכניסו הודעה שמקיימת את התנאי ואחת אשר אינה מקיימת.

חיבור לתכנית DEVICE SUB

בתרגיל 3: הגדרנו תכנית אשר מדמה התקן ומאזינה להודעות. עכשיו נריץ את התכנית ונראה אם היא מקבלת הודעות.

- הריצו את התכנית deviceSub.py
- הריצו את התכנית sensorPub.py

וודאו שהודעות מגיעות מקצה לקצה.

קיראו למדריך והראו לו את העבודה שעשיתם.

תרגיל 5: קישור הודעות לפונקציה למדה

בתרגיל הזה נפעיל פונקציית למדה שתעזור לנו לשלוח הודעה למשתמש כלשהו במקרה שצריך לשלוח התרעה.

הגדרת הרשאות ב-AWS

לפני שנתפנה לתרגיל הבא, דרוש הסבר קצר לגבי הרשאות.

ל-AWS יש מערכת הרשאות מורכבת, המאפשרת לתת (או לחסום) הרשאות, מכל פעולה וע"י כל סוג של משתמש. ההגדרה של ההרשאות מורכבת מ-3 אלמנטים:

- ARN: מזהה משאב. זוהי מחרוזת ארוכה שמזהה את המשאב אליו מתייחסים, וכוללת בין היתר את סוג השירות, חשבון המשתמש והמשאב הספציפי בתוך השירות.
- IAM Policy: כאן מגדירים את ההרשאות עצמן. ההגדרה כוללת את השירות וסוג הפעולה לה נותנים הרשאה, וכן מזהה ARN של מי שמורשה לבצע את הפעולה.
- IAM Role: זוהי ישות שנותנים לה הרשאות לביצוע פעולות. אם רוצים לתת הרשאות לכמה משתמשים, נניח במקרה שלנו, לאפשר שליחת אימייל, מגדירים Role שמאפשר זאת, ומדביקים את ה-Role הזה לכל המשתמשים הרלוונטים. Role יכול להכיל כמה Policies.

כדי לשלוח מייל מפונקציית למדה אנחנו צריכים להגדיר policy מתאים, אחרת נקבל שגיאת הרשאות. במסגרת הניסוי לא נהיה קפדנים, ונפתח הרשאות כמה שיותר חזקות כדי לחסוך בפרטים.

1. בשירות IAM, בחרו Roles בתפריט השירות, ולחצו על Create New Role.
2. שלב 1: בחרו Lambda
3. שלב 2: בחרו Administrator Access. זה מאפשר לעשות כל פעולה שהיא, מבלי שנזדקק להגדרות מפורטות
4. שלב 1: תנו לו שם, קיראו לו למשל LambdaRole.
5. סיימו את התהליך.

הגדרת פונקציית LAMBDA

המטרה שלנו עוד מעט תהיה להפעיל פונקציית למדה כתגובה להודעת MQTT שמגיעה ממכשיר. אנו נגדיר בתחילה פונקציית למדה במנותק מ-IoT ובשלב הבא נחבר בין השניים.

6. צרו פונקציית למדה חדשה, בעזרת הבלו פרינט hello-world-python.
7. מסך Basic Information: תנו שם לפונקציה, ובחרו את ה-role שזה עתה הגדרתם. לסיום לחצו על Create function.

8. מסך Configuation חלק עליון: כרגע נדלג על השלב של הגדרת הטריגר, אשר יהיה ה-Rule אותו נגדיר בסעיף הבא.
9. מסך Configuation איזור Function code:
   בחלק זה אפשר לערוך את קוד הפייתון
   a. וודאו שאתם עובדים בשפת פייתון גירסה 2.7
   b. בקוד הקצר שנותר למטה, הוסיפו שורה שמדפיסה את ה- event.,

10. לחצו על שמירה (כפתור למעלה מימין)

כדי לבדוק שאין לנו שגיאות, נריץ תחילה את הקוד ב-Test, בדומה למה שנעשה בהכנה.

11. לחצו על Test וצרו event עם JSON דיפולטיבי של 3 Keys ו-Values. בדקו שקיבלתם הדפסה באיזור ה- Log Output של האובייקט event.

בשלב הבא נקלוט את ההודעות לעיבוד ע"י הענן.

הגדרת RULE

בסעיף הזה נשתמש שוב ב Rule Engine.

ב-Console היכנסו לשירות IoT ובחרו Act

1. הגדירו Rule חדש ע"י לחיצה על כפתור create
2. תנו לו שם, נניח SensorToLambda.
3. ב-Attribute סמנו * על מנת שהכל יתפוס
4. ב-Topic Filter כתבו את שם הטופיק שממנו אתם מבצעים publish.
5. השאירו את ה- condition ריק.
6. לחצו על add Action כדי לעבור למסך הבא.
7. בחלון Select an Action בחרו פונקציית Lambda.
8. בחלון Configure Action בחרו את הפונקציה שהגדרנו בשלב הקודם, ולחצו על Add action.
9. לחצו על הכפתור Create rule. אם הוא לא לחיץ, כנראה שלא השלמתם את ההגדרות כראוי.

בואו נוודא שה-Rule עובד כמו שצריך. לשם כך נפרסם הודעת MQTT עם הטופיק של החיישן. אם פונקציית הלמדה תרוץ, אנחנו אמורים לראות הודעה על כך בלוג של שרות CloudWatch.

10. חיזרו למסך Test בחלון IoT.
11. בחלק של publish הגדירו את הטופיק אליו כתבתם את ה-rule, ולחצו submit.
12. עברו לשרות CloudWatch. חפשו לוג המתאים לפונקציית הלמדה, וודאו שאכן יש הדפסה בעקבות ה-test publish.

אם צלחתם את הצעד הקודם, הצעד הבא יהיה לקשר את הסקריפט sesnorPub לפונקציית הלמדה.

13. הריצו שוב את הסקריפט sensorPub. עקבו אחרי הלוג וודאו שאתם ממשיכים לקבל הודעות, הפעם מכיוון "החיישנים", עם אובייקטים שכוללים את נתוני החיישנים. לסיום עצרו את הסקריפט.
14. שנו את הקוד: חלצו את הערכים השונים מתוך האובייקטים, והדפיסו אותם ללוג. השתמשו במשתנים מקומיים להשמה של כל ערך.
15. הריצו שוב וודאו שהלוג מדפיס את האובייקט שבנה ה-sensorPub.

16. עצרו את הסקריפט

שאלה מס' 5.1 הדביקו את הלוג לדו"ח, עם החלק שמדפיס את המידע של החיישנים.

שאלה מס' 5.2 הדביקו את פונ' הלמדה שלכם לדוח

נעשה עתה כמה צעדים לקראת התרגיל הבא (SES), בו נשלח אימייל מתוך פונקציית הלמדה
עתה נגדיר אירוע חריג ב-sensorLambda, ע"י הכנסת ערך לחיישן מחוץ לטווח הקיים שלו. כך נוודא שהאירוע החריג קורה פעם אחת.

17. הכניסו תנאי בלולאה בתחתית הקוד של sensorPub, שמשימה ערך מחוץ לטווח החיישן כאשר loopcount הוא 4. באופן זה האירוע החד פעמי יקרה כ-15 שניות לתוך הרצת התכנית.
18. הוסיפו תנאי ל-SensortoLambda Rule שמריץ את פונקציית הלמדה רק כאשר הערך החריג קיים. תזכורת, יש לשנות את ה-condition בתוך ה-Rule query statement.
19. הריצו שוב את הסקריפט למשך 30 שניות.
20. היכנסו ללוג, וודאו שרק האירוע החריג הודפס.

שאלה מס' 5.3 הדביקו את הלוג לדו"ח עם ההדפסה הרלוונטית.

תרגיל 6: תרגיל SES

מבוא

שירות SES הוא שירות בו ניתן לשלוח אימיילים בתפוצה רחבה, למשל אימיילים שיווקיים. כבר ראינו שליחת אימיילים בשירות SNS אולם שירות זה שונה בכמה היבטים:

- SNS מיועד בעיקר לשליחת התרעות בעקבות אירוע (Notifications) בעוד SES הוא שירות אימייל כללי
- SES יכול לשלוח אימייל למספר מאסיבי של נמנעים
- SES תומך בפורמטים רבים של אימייל, כולל למשל attachments.

מצד שני אין הכוונה כאן ליצור מנוע לספאם. לכן יש כמה מנגנוני בקרה שמגבילים את התהליך, ועל הדרך גם מסרבלים אותו במקצת. אנחנו נעבור צעד צעד כדי לעבור את מנגנוני הבקרה האלו.

בתרגיל הזה נשתמש ב-2 כתובות אימייל: אחת לשליחה ואחת לקבלה. לאורך התרגיל, לשם הפשטות, נניח שאתם שולחים מכתובת campus.technion.ac.il לכתובת gmail.com. אין זה אומר כמובן שאתם צריכים להשתמש בדומיינים האלו. אתם יכולים להשתמש, גם בשליחה וגם בקבלה בכל כתובת אימייל שהיא, ואפילו באותה כתובת אימייל לשתי המטרות. אפשר גם לשלוח מייל מסטודנט אחד לסטודנט שני.

מטרת התרגיל

בתרגיל הזה נשלב כמה שרותים בו זמנית. ננצל את השירות SES לשליחת אימיילים, נשלב אותו עם מערכת ה-IoT עליה עבדנו עד כה, ונכתוב פונקציית למדה קצת יותר מתוחכמת שמגיבה להודעה חריגה מחיישן ושולחת אימייל בעקבות זאת.

כדאי לציין שבדרך כלל נעשה שימוש ב-SDK לא מתוך למדה, אלא משרתים רגילים. הפיתוח נעשה בעזרת סביבות פיתוח סטנדרטיות (למשל Eclipse) ובמגוון שפות תכנות בהם Aws SDK תומך, למשל Java, PHP, NET. ועוד. פרוייקטים שמשלבים AWS SDK עשויים גם להיות בגודל של עשרות ומאות אלפי שורות קוד.

מצד שני, למדה נותן בדרך כלל פיתרון פשוט למטרה נקודתית. בניסוי אנחנו משתמשים בלמדה על מנת להתנסות ב-SDK ובו בזמן להימנע מהצורך להרים סביבת פיתוח.

שלבי התרגיל

- בעזרת AWS Console, נתן הרשאות לשליחת מייל מ-campus ל-gmail, ונשלח מיילים לנסיון.
- נשתמש בפונקצית הלמדה מהתרגיל הקודם כדי לקרוא אובייקט
- נסדיר את נושא ההרשאות לשליחת אימייל מפונקצית למדה, ונרחיב את פונקצית הלמדה שתשלח אימייל
- נאחד את כל הקוד למטרה הסופית שלנו: כאשר מקבלים ערך מיוחד באחד האובייקטים, נשלח מייל עם הערך הזה.

הגדרת כתובות אימייל

1. היכנסו לשירות SES תחת Application Services. בתפריט השירות מצד שמאל, לחצו על Email Addresses
2. לחצו על הכפתור הכחול – Verify a New Email Address
3. הוסיפו את כתובת ה-campus שלכם, ובאותו אופן גם את כתובת ה-gmail שלכם
4. היכנסו לתיבות הדואר ואשרו את הרישום, ע"י לחיצה על הלינק הארוך בגוף המייל.

באופן שוטף, אין צורך לעבור את תהליך האישור גם עבור המייל המקבל. אולם מי שחדש בשירות נמצא במוד שנקרא sandbox שבו יש מגבלות גם על כתובות היעד וגם על כמות הנמנעים. אנו נישאר במוד הזה במהלך כל הניסוי.

5. לחצו על Send Test Email ושלחו מייל לעצמכם.

עריכת פונקצית הלמדה

המשימה שלכם בחלק זה הינה שליחת אימייל במקרים מסויימים.

הנה קוד דוגמא לשליחת אימייל ב-SES בפייתון. את הקוד הזה תעתיקו ותשנו בהתאם לפרמטרים שלכם.

```
import boto3
…
    ses_client = boto3.client('ses')
    src =  'from_somebody@example.com'
    msg = {
        'Subject': {
            'Data': 'from lambda'
        },
        'Body': {
```

```
            'Text': {
                'Data': 'That is the email body'
            }
        }
    }
    dest = {
        'ToAddresses': ['to_somebody@example.com']
    }
    response = ses_client.send_email(Source=src, Destination=dest,
Message=msg)
```

1. העתיקו את הקוד, ושנו את הערכים לאלו שמתאימים לכתובות האימייל שלכם.
2. שנו את גוף המייל כך שיכיל את הערך עליו צריך להתריע.
3. הריצו שוב את הסקריפט של sensorPub, עד לשלב שבו נשלח הערך החריג.
4. וודאו שקיבלתם אימייל עם התרעה.

להגשה

אחרי שהצלחתם לשלוח אימייל כתוצאה מהאירוע החריג:

שאלה מס' 6.1 הדביקו את הקוד שלכם

שאלה מס' 6.2 הדביקו את האימייל שקיבלתם.

תרגיל 7: קיפול הניסוי

לפני שנקפל, נביט שוב בחיובים, כפי שעשינו בחלק הראשון

BILLING

לחצו על שם המשתמש שלכם למעלה מימין, ובחרו Billing and Cost Management.

1. לחצו על Bills. כמה שילמתם על כל שירות?
2. לחצו על Cost Explorer. לחצו על Launch.
3. בחרו את הדוח Monthly cost by service. עברו ל-Day (מעל הגרף משמאל).

שאלה מס' 7.1 על אילו שירותים קיבלתם חיוב?

הורדת שירותים

גם שרותים שניתנים חינם, חשוב להוריד כדי לא לשכוח אותם, אלא אם כן אתם יודעים בוודאות שתשתמשו בהם בקרוב. תזכרו שכל מה שעשיתם ניתן לשחזור בקלות, ולכן לא צריך כל כך לחשוש מלמחוק משאבים.

להוריד:

- EC2
- ELB
- Private Image (Derigister)
- VPC
- Rules
- IoT
- Lambda
- SES

מה בהמשך

עכשיו משסיימתם את מטלות הניסוי, אבל עדיין יש לכם חשבון AWS פתוח, עומדות בפניכם כמה אפשרויות

- להמשיך לחקור על הענן, במסגרת עצמאית או במסגרת פרוייקט. תוכלו ליהנות מהאופציות החינמיות ומהקרדיט של החשבון שלכם, אבל כדאי לזכור:
  1. לא להשאיר אחרי כל התנסות שרותים פתוחים, על מנת לא לבזבז כסף
  2. לשים לב לאילו מהשרותים הקרדיט תופס ולאילו לא. הרשימה המלאה נמצאת כאן:

     My account->Credits->Applicable Products

  3. לבדוק מדי פעם את החיובים שלכם, גם את מזמן לא השתמשתם
- להשאיר את החשבון "רדום", למקרה שתרצו לחזור אליו בעתיד. חשבון יכול להישאר רדום שנים ולא יקרה לו כלום וגם לא תחויבו על כלום.
- לסגור את החשבון. אופציה זו שמורה לחשדנים ולאילו שכל עניין הענן ממש לא נראה להם. ניתן לעשות זאת בתחתית הדף של My Account.