ניסוי ענן – הכנה למפגש שני

יש לבצע רק אחרי המפגש הראשון

מבוא ורקע

בחלק השני של הניסוי יהיו תרגילים המערבים קוד בפייתון. הקוד הוא פשוט, בצורה של סקריפט, עם כמה עשרות שורות. אמנם כל מי שיש לו מעט נסיון יתמודד ללא קושי עם הקוד, בכל זאת נבצע מיני הכנה לפייתון.

מבוא מזורז (מאד) לפייתון

דקדוק ואינדנטציה

פייתון עובד ללא סוגריים מסולסלים לסימון בלוקים (נניח של קוד של פונקציה או של לולאה). במקום זה יש שימוש באינדנטציה, כלומר קוד שנכנס טאב אחד ימינה. המשמעות של זה היא שקוד ללא אינדנטציה נכונה לא ירוץ טוב.

דוגמא להגדרת פונקציה:

```
def hello_world():
   print "hello world"
```

טיפוסים

אין הגדרה של טיפוסים, וגם לא צריך להצהיר על משתנים. הם מוגדרים בפעם הראשונה שמשתמשים בהם, והטיפוס נקבע לפי ערך ההשמה.

לדוגמא:

```
>>>i=2
>>>j=3
>>>i+j
5
```

דוגמא למחרוזות:

```
>>> i='two'
>>> j='three'
>>> i+j
'twothree'
```

נוסף למשתנים הפשוטים כמו string ו-int, אפשר להגדיר מערכים

```
>>> myList=[1,2,3,4,5,6]
>>> myList[2]
3
>>> myList[3:]
[4, 5, 6]
```

הטיפוס המעניין לניסוי הוא Dictionary. הוא דומה ל-map בשפות אחרות, כלומר ההכנסה והשליפה מבוצעות לפי key ו-value. כאן יש צורך בהגדרת המשתנה מראש.

למשל:

```
>>> age={}
>>> age['moti']=16
>>> age['sara']=22
>>> age['avi']=4
>>> age
{'moti': 16, 'sara': 22, 'avi': 4}
```

גישה ישירה לאיבר במילון תהיה כך:

```
>>> age['sara']
22
```

אין הכרח שכל ה-values ב-dictionary יהיו מאותו טיפוס. יכול למשל להיות מצב ש-value מסויים הוא int, וזה שלידו הוא array.

איטרציות על מערכים ומילונים

איטרציות על מערכים מאפשרים לנו לבצע פעולות בלולאה על כל איבר במערך.

במערך זה נראה כך:

```
>>> arr = ['a', 'b', 'c', 'd', 'e']
>>> for letter in arr:
...      print letter + ' ',
...
a  b  c  d  e
```

הערה: פסיק בסוף פקודת print גורמת לכך שלא נרד שורה אחרי ההדפסה.

במילון אפשר לעשות איטרציה על המפתחות או על ה-values או על שניהם.

איטרציה על מפתחות:

```
>>> for k in d:
...     print k,
...
y x z
```

איטרציה על שניהם:

```
>>> d = {'x': 1, 'y': 2, 'z': 3}
>>> for r, m in d.iteritems():
...     print "{0}->{1}  ".format(r, m),
...
y->2   x->1   z->3
```

הדפסת מחרוזת שמכילה משתנים

המקבילה של הפקודה הבאה ב-c:

```
printf("There are %d students in %s class \n", num_students,class_name );
```

יכולה להיכתב בפיתון כך:

```
print "There are {0} students in {1} class \n".format(num_students,class_name)
```

אם אתם רואים דוגמאות הדפסה שמזכירות שפת c עם סימני אחוזים, זהו פורמט ישן שעדיף לא להשתמש בו.

טיפ: בחלק הזה תשתמשו ב-3 שרותים שונים: SNS, Lambda, CloudWatch. אתם תזפזפו לא מעט בין שלשתם. מומלץ להחזיק כל שירות בלשונית נפרדת, לצורך מעבר מהיר משירות לשירות.

קצת על JSON ופייתון

JSON הינו פורמט לייצוג מבני נתונים, מבוסס טקסט. למשל אובייקט person עם שדות first_name ו-last_name ייראה כך:

```
obj={“first_name”:”Roy”, “last_name”:”Mitrany”}
```

בפייתון, ניתן להמיר את המבנה הזה ל- dictionary, ואז הגישה אליו היא כזו:

```
name=  obj[“last_name”]
print name
>>> Mitrany
```

בהרבה מקרים, הערך של שדה הוא אובייקט בעצמו, ואז מקבלים קינון של אובייקטים. בנוסף, הערך יכול להיות מערך של מחרוזות או אובייקטים.

למשל

(דוגמא מהרשת)

```
data = {
"country abbreviation": "US",
"places": [
    {
        "place name": "Belmont",
        "longitude": "-71.4594",
        "post code": "02178",
        "latitude": "42.4464"
    },
    {
        "place name": "Belmont",
        "longitude": "-71.2044",
        "post code": "02478",
        "latitude": "42.4128"
    }
],
"country": "United States",
"place name": "Belmont",
"state": "Massachusetts",
"state abbreviation": "MA"
}
```

האובייקט הראשי הוא בעל 6 שדות. השדה השני, places הוא מערך של אובייקטים. כדי להדפיס את השדה המסומן בצהוב, צריך לכתוב:

```
print data['places'][1]['post code']
>>> 02478
```

זאת כי:

- data[places] מחזיר מערך של אובייקטים

- data['places'][1] מחזיר את האיבר השני במערך
- data['places'][1]['post code'] מחזיר את הערך עבור השדה post code עבור אובייקט זה.
-

למדה (Lambda)

שירות למדה מאפשר לנו להריץ קוד, כתגובה על אירוע מסויים שקורה בענן, כמו למשל שינוי במסד נתונים או אחסון, או קבלת הודעה. סוג האירוע נקרא טריגר.

בניגוד לשרותים שראינו עד כה, אין כאן הקצאת משאב כמו אחסון או כח חישוב. אנחנו לא יודעים איפה הקוד רץ, ומובטח לנו שלא תהיינה בעיות עומס, זמינות, תקשורת וכך הלאה. זהו מודל של Platform as a Service, בשונה מהמודל שראינו עד כה שנקרא Infrastructure as a Service.

מודל התמחור כאן הוא לפי זמן ריצה של קוד, וכמות הזיכרון שהוא תופס בזמן ריצה. כלומר, לא הרצת – לא שילמת. התשלום ליחידה אחת של זמן ריצה הוא כרגיל שברירי דולר (באופן גס על קוד פשוט וקצר 7000 הרצות עולות דולר), אבל אתרים עסוקים ירוצו לא מעט פעמים ובסוף זה יצטבר.

במקרה של הניסוי, כמו המשאבים האחרים, יש מסלול Free Tier אליו זכאים הסטודנטים, שמשמעותו שקוד שתופס 128MB זיכרון ורץ פחות מ-100 מילי שניה ניתן להריץ מעל 3 מליון פעמים.

ניתן להריץ קוד למדה בכמה שפות, כולל פייתון, Nodes.js, ג'אווה. אמנם ג'אווה היא שפה יותר מוכרת אצלינו בפקולטה, אבל העבודה איתה היא יותר מסורבלת, מכיוון שצריך לכתוב את הקוד במקום חיצוני ולייבא אותו, ואילו בפייתון יש עורך מובנה בתוך שירות הלמדה. לכן נעבוד עם פייתון.

הקוד עצמו הוא קוד עצמאי וחסר מצב. אין טעם לשמור מבני נתונים משמעותיים, כי הם ימותו בסוף הריצה. אם כן רוצים לשמור משהו, אפשר בעזרת פקודות אמאזון לכתוב ל-s3 או לשרות אחסון אחר, בהנחה שיש לקוד את ההרשאות המתאימות לעשות כך.

הקוד מכיל פונקציה שנקראת כאשר הקוד נכנס לעבודה:

```
def lambda_handler(event, context)
```

הפרמטר event הוא אובייקט ג'ייסון שמכיל את ההודעה שהגיעה עם הרבה מידע עליה

הפרמטר context מכיל מידע על סביבת הריצה, כמו למשל איך נקרא הלוג אליו כותבים, מה זמן הריצה של הפונקציה ועוד.

אנחנו נשתמש בניסוי רק בפרמטר event.

אינטרנט של הדברים

האינטרנט של הדברים, שמעתה נקרא לו IoT הוא מושג שיווקי שמתאר איך מכשירים שונים יכולים להתחבר לאינטרנט, לשלוח מידע מהמכשיר ולקבל פקודות ממקומות מרוחקים.

דוגמאות (שחוקות למדי)

הבית החכם: חיבור מכשיר חשמל, כמו דוד, מקרר, שואב אבק לאינטרנט. דוד יכבה ויודלק מרחוק בעזרת אפליקציה, שואב אבק יפעל מרחוק ויידווח מתי סיים לנקות. מקרר יכול בתאוריה להיות הרבה יותר מתוחכם, ואולי בעתיד ידווח מה התכולה שלו ויעזור להרכיב רשימת קניות.

יש עוד מערכות בית חכם, למשל אזעקה, מנעולים, תאורה, חלונות ועוד.

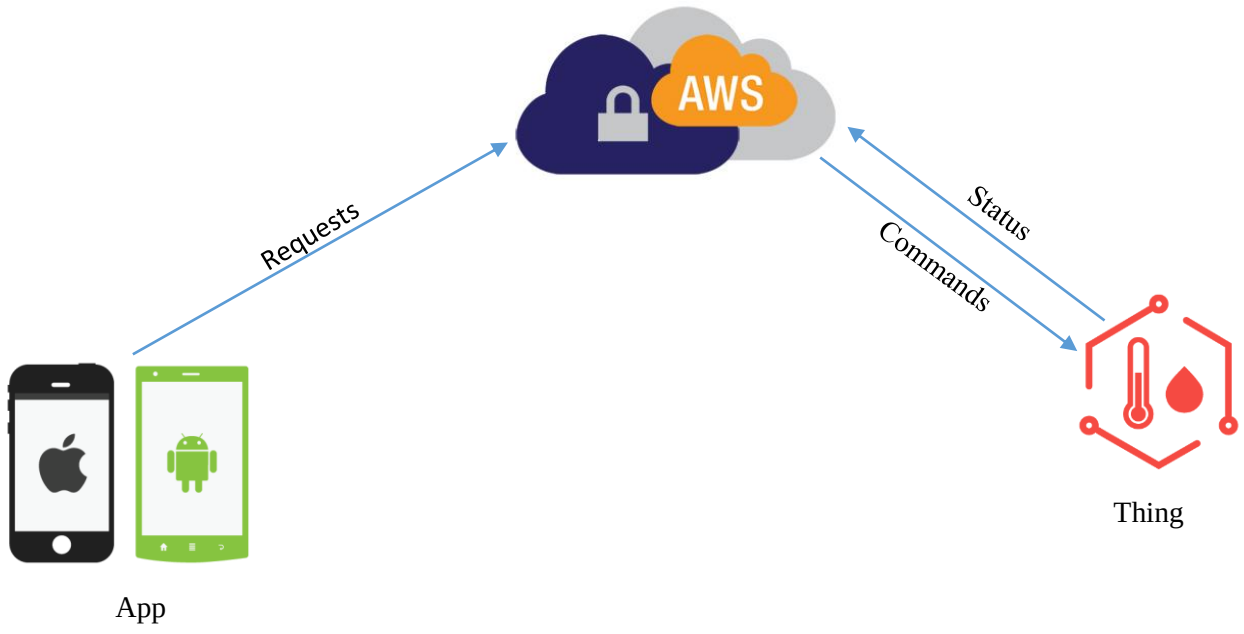
עיר חכמה: פיזור מצלמות וחיישנים ברחבי העיר שמדווחים על המצב, כולל: פקקי תנועה, אירועים בטחוניים, שריפות, פחים מלאים ועוד.

בהרבה מקרים המכשיר עצמו לא כולל מודול של רשת. במקרה כזה מצמידים בקר (לא כזה שאוכלים אלא controller) אשר מחובר לאינטרנט, ושולט על המכשיר עצמו באמצעים שונים.

אחד האתגרים של IoT הינו אבטחה. לא היינו רוצים למשל שהאקר יוכל לשגע אותנו ויתחיל לפתוח ולסגור תריסים אצלנו בסלון בלי שנשלוט על כך. כעיקרון הבעיה הזו כבר פתורה, אולם אותו מכשיר כבר נדרש לרמת תחכום גבוהה יותר שדורשת הצפנה והחלפת מפתחות, מה שמחייב מעבדים יותר חזקים.

מודל כללי של IoT

ב-IoT שמתבצע דרך ענן יש שלושה שחקנים מרכזיים:

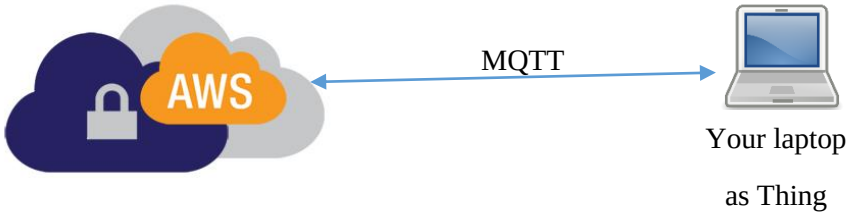


- בקצה אחד המשתמש, שעובד עם אפליקציה או דפדפן,
- בקצה השני המכשיר, שמקבל פקודות ושולח סטטוסים
- ביניהם הענן שאצלו מרוכז כל המידע, והוא משמש כשרת הן למשתמש והן למכשיר.

אפשר להסתכל על הענן כמעין מתווך בין המשתמש למכשיר, אשר דואג לדבר עם כל קצה בשפה שלו, וגם למלא דרישות נלוות כמו אבטחה, קישוריות וכו'.

המודל בו נשתמש בניסוי

בניסוי עצמו נשתמש במודל ספציפי ופשוט יותר. יהיו רק שני שחקנים: הענן והמכשיר. את תפקיד המכשיר ימלא הלפטופ שלכם. הפרוטוקול בו יתקשרו השניים יהיה פרוטוקול MQTT (עליו נרחיב עוד רגע), ולרוב נתרגל שליחת סטטוסים בלבד מהמכשיר לענן.



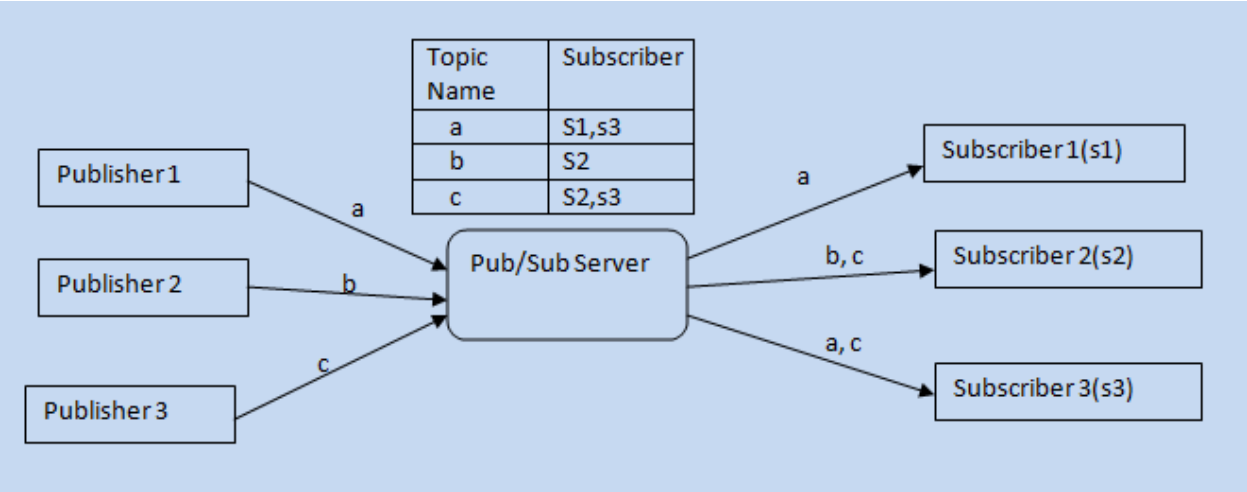
פרוטוקול MQTT

זהו הפרוטוקול הנפוץ ביותר כיום לתקשורת עם מכשירים ב-IoT. כמו שכבר פגשנו ב-SNS, פרוטוקול MQTT עובד גם הוא במודל publish-subscribe.

pub/sub model

תזכורת, למי ששכח, לגבי המודל. זהו מודל העברת הודעות בין שני קליינטים, דרך שרת שנקרא message broker. ניתוב ההודעות נעשה בעזרת נושא (טופיק). מי שרוצה לקבל הודעות בנושא מסויים, נרשם אצל הברוקר. השולח לא צריך להירשם לנושא, אלא מספיק שישלח הודעה עם הנושא, וכל מי שנרשם לנושא זה יקבל את ההודעה. השולח לא צריך לדעת מי קיבל את ההודעה.

הברוקר מחזיק טבלא עם ה-subscribers לכל טופיק, ומנתב הודעות בהתאם לטבלא.



כל מכשיר יכול להיות גם publisher וגם subscriber. ענן ה-AWS משמש כ-Broker, ובמקרים מסויימים (אותם אנחנו נתרגל) שרותים של AWS יהיו publishers או subscribers.

מבנה topic

לטופיק יש מבנה היררכי, כאשר קו נטוי "/" מפריד בין הרמות השונות בהיררכיה (כמו במערכת קבצים). בנוסף יש שימוש בתווים חופשיים (wildcards) אשר יכולים להחליף כל תו אחר.

תו + יכול להחליף שכבה בודדת בכל חלק של ה-topic.

תו # יכול להחליף כל מספר של היררכיות. אפשר להשתמש בו רק בקצה המחרוזת.

Payload

אין כאן כלל מסויים שמוכתב ע"י הפרוטוקול, אולם במקרים רבים תוכן ההודעה יהיה מורכב מאובייקט ג'ייסון.

אנחנו נצלול ישר לדוגמאות, ומכאן נבין.

דוגמא 1

נניח שיש במעבדה חיישנים שונים בכל חדר: חיישן טמפרטורה, לחות, רמת רעש.

יש במעבדה 3 חדרים: 373, 375, 381.

אפשר להגדיר את החיישנים בהיררכיה הבאה: קודם כל שם המעבדה, אחר כך שמדובר בחיישנים (אולי יש לי עוד מכשירים שלא מוזכרים כאן), מספר החדר וסוג החיישן. תוכן ההודעה יהיה רק מספר שמהווה את ערך המדידה.

כל חיישן נקנפג כך שיפרסם את הטופיק המתאים לו. לחיישן אין מידע לאן ההודעה תגיע, הוא דואג רק לכך שיפרסם עם הטופיק הנכון כך שמי שרוצה לקבל את ההודעה, יעשה subscribe לטופיק הנכון וה-broker כבר יעביר לו אותה.

לכן יוגדרו ה- topics האלו:

```
nssl/sensor/371/tempreture
nssl/sensor/373/tempreture
nssl/sensor/375/tempreture
nssl/sensor/371/humidity
nssl/sensor/373/humidity
nssl/sensor/375/humidity
nssl/sensor/371/noise
nssl/sensor/373/noise
nssl/sensor/375/noise
```

- אם יתבצע subscribe ל-**nssl/sensor/373/noise** נקבל הודעות רק מסנסור בודד.
- אפליקציה שמעוניינת בנתוני לחות לכל החדרים במעבדה תעשה subscribe ל- **nssl/sensor/+/humidity**
- אפליקציה שמעוניינת בכל הנתונים נתונים עבור חדר 373 תעשה subscribe ל- **nssl/sensor/373/+**
- שרות לוג שרוצה לתעד כל דבר יעשה subscribe ל- **nssl/#** ויקבל נתונים על כל המעבדה (לא רק חיישנים).

דוגמא 2:

נניח אותו חדר עם אותם חיישנים, אבל במקרה הזה יש בקר שמעביר את כל הנתונים בהודעה אחת על מנת להוריד עומס ועלות שמייצרת כמות גדולה של הודעות.

לכן הטופיק הוא יותר פשוט, והתוכן יותר מסובך ובמקום מספר בודד מכיל אובייקט ג'ייסון מהסוג הבא:

Topic: nssl/sensor/381

Payload: {temp: 25, humidity: 40, noise: 20}

באחריות מקבל ההודעה להכיר את מבנה האובייקט ולחלץ ממנו את הנתונים שמעניינים אותו.

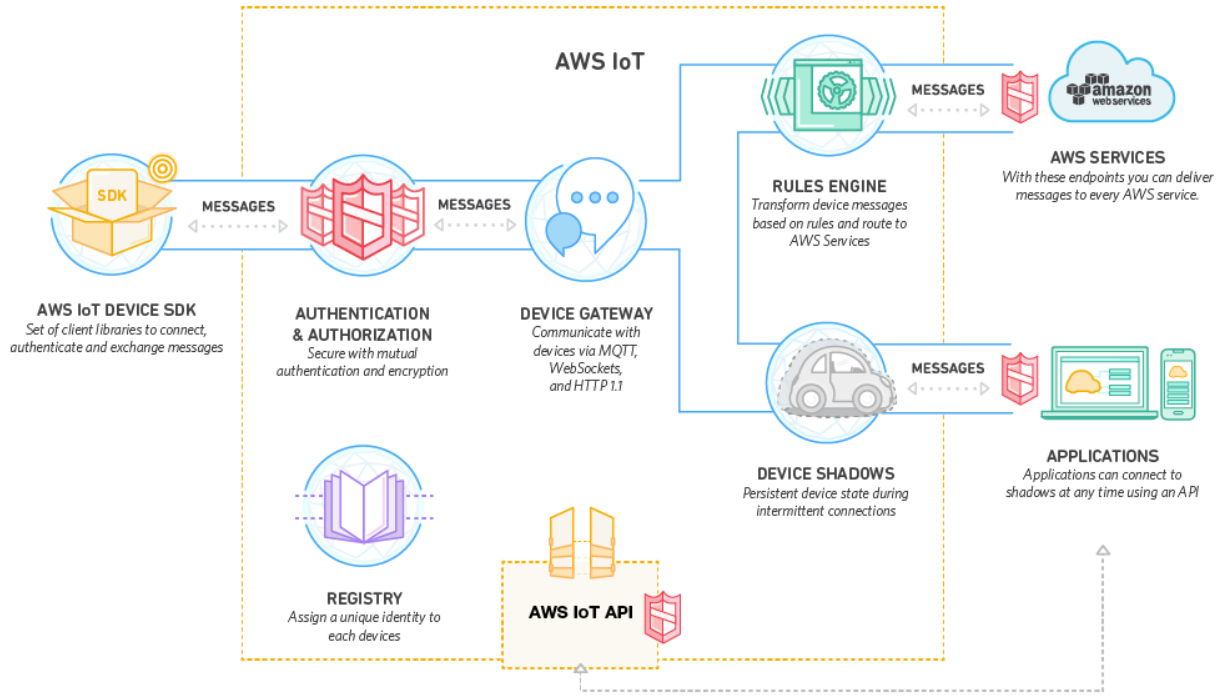
שרות AWS IoT

השרות במתכונתו הנוכחית הושק בנובמבר 2016 כך שהוא חדש יחסית. מטרתו העיקרית היא לתקשר עם מכשירים מצד אחד ועם אפליקציות ענן מהצד השני. יש ביכולתו לתמוך במיליארדי מכשירים וטריליוני הודעות בו זמנית.

שרות IoT מתממשק בקלות לשירותים אחרים של AWS, כמו Lambda, CloudWatch, Amazon S3 ואחרים.

בדומה לשירות למדה, לא משלמים כלל על הגדרות, והתשלום הוא על השימוש בלבד. נכון לכתיבת שורות אלו, המחיר נע בין 5 ל-8 דולר למיליון הודעות.

הנה כמה מהמרכיבים העיקריים שלו:



AWS IoT Device SDK

ספריה שחלקה המרכזי הוא Client שיודע להתחבר לשרת של אמאזון בפרוטוקול מאובטח, למשל בעזרת פרוטוקול MQTT. המרכיב הזה רץ מחוץ לענן. אנחנו בניסוי נשתמש בספריה הממומשת בפייתון.

Authentication and Authorization

אחראי על נושאי אבטחה. מוודא שרק מי שמורשה שולט על המכשיר, ורק מכשירים מורשים משתמשים בשירות. בנוסף, מצפין את ההודעות מהמכשיר ואליו.

אנחנו נשתמש בשירות הזה, אחרת לא נצליח להתחבר, אבל לא נעמיק כלל בפרטים.

Registry

מוסיף מכשירים חדשים לשירות

Device Shadows

המנגנון הזה בשימוש נפוץ אצל מפתחי IoT באמאזון, אבל אנחנו לא נכסה את זה בניסוי. לכן לא חובה לקרוא אותו וניתן לדלג לסעיף הבא של Rules Engine.

המנגנון מחזיק אובייקט בענן שמייצג את המכשיר. כאשר מתבצע שינוי מכיוון האפליקציה או מכיוון המכשיר, הוא עובר דרך ה-shadow אשר שולח הודעות על השינוי דרך מנגנון ה-pub/sub.

דוגמא:

נניח יש אובייקט שמייצג מתג. המצב של המתג מייוצג ע"י שדה power שיכול לקבל את הערכים on/off.

במצב התחלתי, ה-shadow ייראה כך:

```
Desired:{power:off}
Reported:{power:off}
```

ברגע שמדליקים את המתג דרך האפליקציה, ירוץ קוד שישלח אובייקט shadow שמשנה את מצב ה-desired.
עכשיו המצב יהיה כזה:

```
Desired:{power:on}
Reported:{power:off}
```

שרות ה-IoT מזהה את הפער בין ה-desired ל-reported וישלח בהתאם הודעת MQTT עם ההבדלים בין שני המצבים, בטופיק שמתאים למכשיר. המכשיר יקבל את ההודעה, ויפעל לפי הצורך. בתום הפעולה, המכשיר ישלח הודעה הכוללת את התוכן הבא:

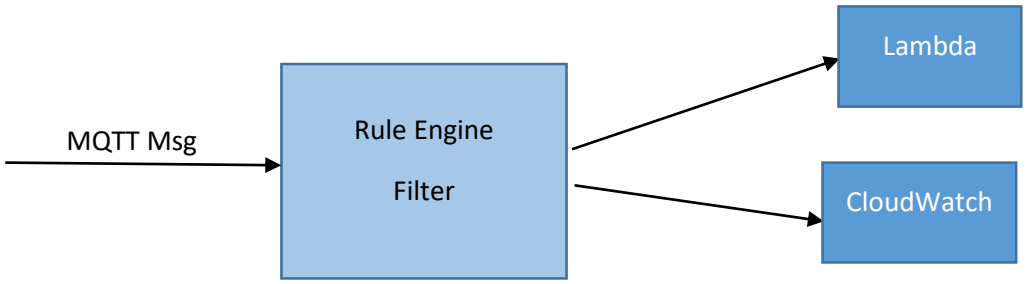
```
Desired:{power:on}
Reported:{power:on}
```

Rules Engine

מנגנון שיודע בעצמו להיות subscriber של הודעות, לקלוט אותם ולבצע פעולה כלשהי, כמו למשל להעביר לשרות אחר.

אנחנו נעבוד בניסוי עם מנגנון ה-rules engine באופן יחסית יותר מעמיק, ולכן נפרט עליו כאן קצת יותר.

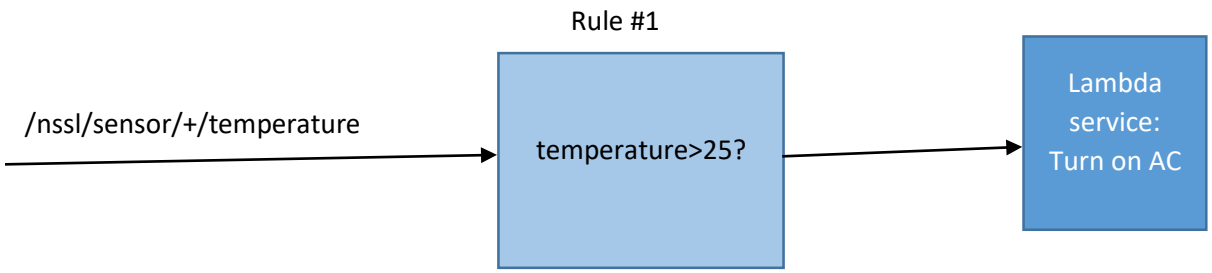
הבסיס של החלק הזה הוא להאזין להודעת MQTT בתור Client, לנתח את ההודעה, ואם היא עומדת בתנאים מסויימים, להפעיל שרות אחר. אפשר להעביר לשרות הבא כפרמטר את הודעת ה-MQTT כפי שהיא, או הודעה שונה או חלקית.

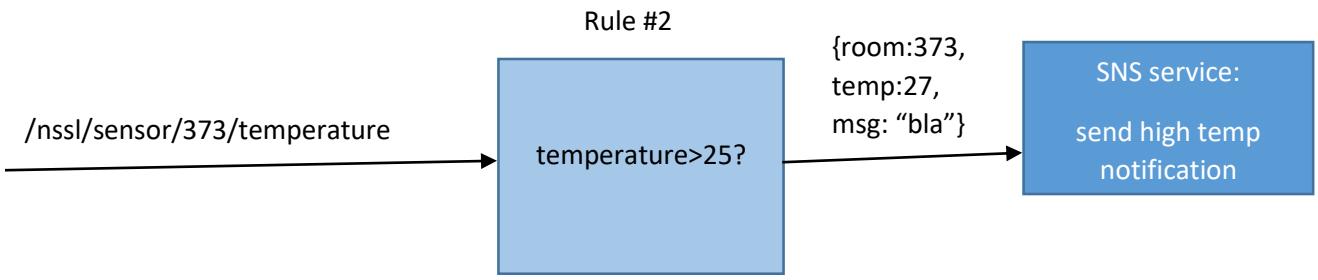


אפשר להגדיר כמה Rule Engines על אותה הודעה וכולם יפעלו.

נמשיך בדוגמא של החיישנים במעבדה אותם הכרנו בפרק ההסבר על MQTT.

נניח שכשהטמפ עולה בכל אחד מהחדרים, ארצה להדליק מזגן באותו חדר. אבל ספציפית אם היא עולה בחדר 373, שם יש לי מכשירים רגישים לחום, ארצה גם לשלוח הודעת SMS לאחראי מעבדה. לכן יהיו לי 2 Rule Engines:





שימו לב שב- Rule השני גם שינינו את תוכן ההודעה.

פורמט השאילתא

הפורמט של השאילתא ב-Rule הוא SQL. זהו פורמט הלקוח מעולם ה-Database.

המדריך לשימוש בשאילתת SELECT נמצא כאן: http://docs.aws.amazon.com/iot/latest/developerguide/iot-sql-select.html

אנחנו נתעכב על הפרטים הרלוונטים לניסוי.

מבנה השאילתא הוא כזה:

```
SELECT <Attribute> FROM <topic filter> WHERE <condition>
```

Attribute: מציין אילו שדות לקחת מתוך ה-payload של הודעת ה-MQTT. התו * מסמן להעביר את כל האובייקט כפי שהוא.

Topic filter: מציין לאיזה טופיק ה-Rule מאזין

Condition: נותן אופציה לפלטר רק ערכים מעניינים מתוך ה-payload.

בהמשך לדוגמא של ה-Rules שהצגנו קודם.

נניח שמגיעה ההודעה הבאה:

```
topic: nssl/sensor/373/temperature
payload: {value:26}
```

כדי שנפעיל את פונקציית הלמדה, שאילתת ה-rule תהיה זו:

```
SELECT * FROM "nssl/sensor/373/temperature" WHERE value > 25
```

במקרה הזה פונקציית הלמדה תקבל כפרמטר את ה-payload של הודעת ה-MQTT.

כדי להפעיל הודעת SNS השאילתה תהיה זו:

```
SELECT “Room 373: High temperature Alert!!” AS msg_text, value AS temperature FROM
‘nssl/sensor/373/temperature’ WHERE value > 25
```

במקרה הזה אנחנו מוספים ערך חדש של תוכן הודעה, ומחליפים את שם השדה value במילה temperature. לכן שרות ה-SNS יקבל את האובייקט הזה בתור פרמטר:

```
{
   msg_text:“Room 373: High temperature Alert!!”,
   temperature:26
}
```

תרגילי הכנה

Python

הדרך הקלה ביותר להריץ את תרגילי הפייתון שאתם צריכים להגיש היא לפתוח טרמינל של מכונת לינוקס, ולכתוב python. כך תיכנסו ל-python console ותוכלו להריץ כל פקודה. אם התרגיל מכיל כמה שורות ודורש אינדנטציה, אפשר לערוך ב-Notepad או כל עורך טקסט אחר על המחשב שלכם, ואז להדביק את כל השורות יחד לתוך הקונסול. אתם צריכים להגיש print screen של חלון הטרמינל. השתמשו ב-snipping tool לצורך כך.

תרגיל 1:

הדפיסו למסך מחרוזת עם השמות הפרטיים והמשפחה שלכם

תרגיל 2:

הדפיסו את השמות שלכם, הפעם משולבים בתוך המשפט הזה:

My fist name is ____ and my last name is ____ !!!

תרגיל 3:

בנו dictionary שמייצג גליון ציונים (הוא לא צריך להיות אמיתי). כתבו לולאה שמדפיסה את ממוצע הציונים, לא משוקלל.

תרגיל 4:

בעזרת אותו dictionary, מצאו את הציון הגבוה ביותר ואת הציון הנמוך ביותר. הדפיסו את המשפטים הבאים:

My lowest grade this semester was <low_grade>

My highest grade this semester was <high_grade>

תרגיל 5:

השתמשו בדוגמא למעלה של אובייקט JSON.

הוסיפו עוד אובייקט של place למערך places

הדפיסו את ה-longitude של 3 האובייקטים

שירות IAM

IAM הוא השירות דרכו מגדירים משתמשים והרשאות עבורם. מטרתו היא גם לשרת קבוצות גדולות של משתמשים על אותו חשבון. למשל, למעבדת NSSL יש חשבון ב-AWS. כל סטודנט שמבצע פרוייקט במעבדה ומשתמש בחשבון זה, מקבל שם משתמש אחר, ולכל אחד הרשאות אחרות בהתאם לתפקידו.

במקרה שלכם, מכיוון שהחשבון אישי, לא נחוץ יותר מדי בשלב זה. בכל זאת נגדיר משתמש אחד עם הרשאות אדמין, כדי שתהיה לו גישה לכל מקום בלי להסתבך.

1. מהחלון הראשי של AWS בדפדפן, בחרו Services->Security&Identity->IAM.
2. היכנסו ל-Groups ולחצו על Create Group

3. תנו שם קבוצה - Administrators
4. בשלב הבא, תנו לקבוצה Administrator Access (הבחירה הראשונה), וסיימו את התהליך (אין עוד שלבים)
5. היכנסו ל-Users ולחצו על Create User.
6. שלב 1: תנו לו שם adminuser.
7. שלב 1: בחלק של access type תאפשרו גישה להכל, ותגדירו סיסמא.
8. שלב 2: אחרי שהמשתמש נוצר ומופיע ברשימה, הוסיפו אותו לקבוצה.
9. שלב 4: אתם אמורים לקבל טבלא של שורה אחת עם פרטי המשתמש שנוצרו. בעמודה הימנית יש לינק לשליחת הוראות לוגין באימייל. לחצו על הלינק כדי לקבל את ההוראות, ושמרו אותן אצלכם.

למדה

על מנת להריץ קוד למדה, אנחנו זקוקים למחשב עליו מותקן aws cli, וכן משתמש עם הרשאות אדמין. את התרגיל עצמו תבצעו תחת המשתמש adminuser.

מטרת התרגיל

בתרגיל נקבל הודעת למדה ונדפיס לתוך קובץ לוג את תוכן ההודעה.

על מנת לבצע זאת אנחנו נגדיר פונקצית למדה שהטריגר שלה הוא הודעת SNS. קבצי הלוג נשמרים בשירות CloudWatch.

כניסה ל-AWS עם משתמש adminuser

סעיפים 1-4 מיועדים למי שלא שמר את ההוראות לביצוע לוגין עם adminuser.

1. מיצאו את מספר החשבון שלכם. הוא נמצא ב-My Account. העתיקו אותו.
2. בצעו logout מ-aws
3. היכנסו ללינק הזה: https://<account_number>.signin.aws.amazon.com/console. הלינק הזה מאפשר לכם להיכנס לחשבון שלכם אבל לא עם משתמש ראשי אלא משתמש אחר.
4. הכניסו שם משתמש adminuser וסיסמא
5. היכנסו לשרות למדה (תחת compute)
6. לחצו על Create Function.
7. בחרו Blueprints
8. בחלק בתחתון של המסך מופיעים קבצים לבחירה. על מנת לסנן אפשרויות, כיתבו את המילה Hello בשדה הפילטר.
9. בחרו את ה- Hello world python בגירסה 2.7, וליחצו על כפתור Configure.
10. מסך Basic Information: הכניסו שם לפונקציה
11. מסך Basic Information: מתחת לשם הפונקציה אתם מתבקשים להכניס Role. Role הוא משהו שמגדירים בשרות IAM (היכן שהגדרנו משתמשים), שתפקידו להגדיר הרשאות גישה. בחרו create role from template ותנו שם ל-Role. מעבר לזה לא נדרשות מכם הגדרות נוספות.
12. מסך Basic Information: לחצו על כפתור Create Function בתחתית המסך.
13. מסך הפונקציה: בחלק העליון של המסך מופיע תרשים שמורכב מ-3 חלקים: במרכז תראו את הפונקציה שזה עתה יצרתם, מימין מקושר פלט הפונקציה שהוא CloudWatch Logs, ומשמאל ישמ מקום ל-trigger אחד או יותר, אשר עוד רגע נבחר. שימו לה שלחיצה על כל אחד מהמלבנים משנה את התוכן בחלק

התחתון של המסך. בחרו SNS בתור הטריגר מצד שמאל. אחרי שבחרתם אותו, הפוקוס עובר אוטומטית אליו. בחלק התחתון של המסך, בחרו את ה- SNS topic שלכם. וודאו ש enable trigger לחוץ, ולחצו על כפתור Add.

14. מסך הפונקציה: לסיום תהליך בחירת ה-trigger, לחצו על כפתור Save בראש המסך.
15. מסך הפונקציה: כעת נחזור לפונקציה עצמה. כדי לחזור לראות לראות את קוד הפייתון, לחצו בתרשים למעלה על שם הפונקציה. יש בקוד כבר די הרבה שורות print. יש עוד שורת print בהערה. תוציאו את השורה מהערה כדי שגם תודפס. כך נראה איך נראית ההודעה בשלמותה כשהיא מגיעה לפונקציה. הכניסו להערה את 4 השורות שמתחת. אחרי שסיימתם, לחצו שוב על כפתור Save למעלה.

בשלב ראשון, נבדוק את הקוד שלכם פנימית. לצורך כך יש את כפתור ה-Test שמריץ את הקוד. כדי שהפלט יהיה רלוונטי, נדאג שהטסט מקבל פלט במסנה של הודעת SNS.

16. לחצו על כפתור Configure Test Event בראש המסך.
17. נפתח חלון Configure Test Event. בחרו SNS Event Template, תנו שם ל-event ולחצו על כפתור Create.
18. הריצו טסט, ובדקו ב-log Output שההרצה התבצעה כמו שצריך.

תמונת מצב: הגדרנו topic, והגדרנו פונקצית למדה שמאזינה לטופיק הזה ומריצה קוד פייתון כאשר הודעה נשלחת עם הטופיק הזה. קוד הפייתון מדפיס את ההודעה.

19. בשירות SNS, בצעו publish message עבור הטופיק. תנו מחרוזת משמעותית ולא ארוכה בנושא ובתוכן. מומלץ למספר את ההודעות (למשל tst1 בנושא) כי קרוב לוודאי שתחזרו על הפעולה הזו כמה פעמים, וכדאי שתראו באיזו מההודעות מדובר.
20. כאמור, הודעות ה-Log נמצאות בשירות CloudWatch. היכנסו לשירות זה ומצאו את ההודעות תחת Logs. וודאו שקיבלתם והצלחתם להדפיס את ההודעה.
21. בואו ננסה להדפיס שדה אחד מההודעה: הוסיפו את השורה הזו:

```
print("----------Event source = " + event['Records'][0]['EventSource'])
```

22. באותו אופן הדפיסו את ה-subject ואת ה-message. שימו לב שיש עוד רמת קינון כדי להגיע לערכים האלו.

להגשה:

- הקוד של פונקציית הלמדה
- צילום של הלוג
- הפלט של הלוג

IoT

1. קראו על פרוטוקול MQTT (למשל כאן או כאן) והסבירו מדוע הוא שימושי באפליקציות IoT. הדגימו באמצעות דוגמה פשוטה.
2. הסבירו מהי פונקציית Callback ומדוע היא חשובה ביישומי IoT.