DynamoDB

DynamoDB, או בקיצור DDB, הוא אחד השירותים המפורסמים והוותיקים של AWS, ששוחרר לאוויר העולם ב־2012. כמו הרבה שירותים ראשוניים של AWS, המקורות שלו הגיעו מדרישות פנימיות של Amazon. ימי המכירות המפורסמים של Amazon כגון Cyber Monday יצרו עומס עצום על הטכנולוגיות שבהן השתמשה החברה באותה תקופה, ולכן הוחלט ליצור פתרון כדוגמת DDB. במקור DDB היה מסד נתונים פנימי של Amazon שנקרא Dynamo. ב־2007 פורסם מאמר[27] המתאר את הפתרון הפנימי, שייהפך לימים ל־DDB. על בסיס המאמר המפורסם הזה נכתבו מסדי נתונים, כגון MongoDB ו־Cassandra, שלמעשה היו פתרון Open Source למבנה שהוצג בו.

בפרק זה נסקור בזריזות את היתרונות וההבדלים בינו לבין מסדי נתונים רלציוניים (relational databases), נעבור על מונחים בסיסיים ונראה כיצד הם באים לידי ביטוי ב־DDB, ולסיום נמשיך לעבוד על הפרויקט וסוף־סוף נוסיף לו state.

הפרק הנוכחי מניח שאתם מכירים ויודעים מהו מסד נתונים רלציוני.[28]

נתחיל עם הגדרה בסיסית - DDB הוא Serverless Database הפועל במודל NoSQL. נבין בהמשך מהי המשמעות של NoSQL.

למה לי פוליטיקה עכשיו

מדוע DDB הוא בחירה מעולה לעולם ה־cloud? זו שאלה לגיטימית מאוד. האם יש לו יתרונות ברורים על פני הטכנולוגיה המוכרת כמעט לכולם - מסד נתונים רלציוני?

1. Scale - נושא זה הוא אחת המטרות המרכזיות של בניית DDB. בניגוד למסדי

---

27 "Dynamo: Amazon's Highly Available Key-value Store". https://www.allthingsdistributed.com/files/amazon-dynamo-sosp2007.pdf.

28 https://he.wikipedia.org/w/index.php?oldid=34715307.

נתונים רלציוניים, שחווים דגרדציה ככל שכמות המידע גדלה, DDB נבנה בצורה כזאת שהוא סקיילבילי בצורה אינסופית. ניתן להוסיף עוד ועוד מכונות כדי לתמוך בכמות הנתונים הגדלה. למעשה, ה־Scale בא לידי ביטוי הן בגודל הטבלאות, שאין עליו מגבלה, והן בביצועים של הפעולות, כגון קריאה או כתיבה. DDB מבטיח ביצועים קבועים של מילי־שניות של שאילתות, קריאות וכתיבות, ללא קשר לכמות הנתונים בטבלה.

2. אבטחה ו־IAC – עולם הענן המודרני מכווין את המפתחים למודל עבודה מסוים. עיקרון אחד בו הוא היכולת להגדיר קונפיגורציה כקוד, ועיקרון נוסף הוא היכולת להגדיר אבטחה בצורה פשוטה. DDB הוא אורח של כבוד בשירות ה־IAC של AWS CloudFormation, כלומר ניתן להגדיר בצורה פשוטה מאוד את הטבלאות, את ההרשאות עליהן ואת הפעולות השונות המותרות עד רמת השורה הבודדת. אין צורך ללמוד שפה חדשה או לנהל מודל הרשאות חדש, זו אותה שפה שלמדנו בפרקים הקודמים. נתרגל זאת בהמשך הפרק באמצעות AWS SAM.
3. DDB - Serverless הוא פתרון Serverless מלא, קרי אין צורך לנהל מכונות, והעלויות הן על פי שימוש. הדבר מוריד באופן משמעותי את התקורה הניהולית.

מידול נתונים ב־DDB

מידול נכון ב־DDB יכול בקלות למלא ספר שלם. כאן רק נטעם מהנושא. אני ממליץ בחום על ספרו של אלכס דה-ברי (DeBrie), *The DynamoDB Book*.[29] בחלק זה ננסה למדל את המידע שנשמור בפרויקט שאנו בונים. תחילה נציג כיצד היינו ממדלים אותו במסד נתונים רלציוני ולאחר מכן נמדל אותו ב־DDB – להזכירכם, הוא מסוג NoSQL.

כשמסתכלים על מסד נתונים רלציוני, ישנה סכמה מאוד ברורה של הנתונים שנשמרים, ופעמים רבות משתמשים בכוחו של ה־SQL JOIN על מנת לענות על שאלות מורכבות. Join היא פעולה חישובית יקרה, ואחד ההבדלים המרכזיים בין DDB למסדי נתונים

29 The DynamoDB Book -- The most comprehensive book on data modeling with Amazon DynamoDB.

רלציוניים הוא העובדה שאין Join. נשאלת השאלה, מהו המקור ל־Join? מסדי נתונים רלציוניים פותחו בתקופה שבה עלות אחסון הייתה יקרה מאוד, ולכן הן שפת הפיתוח, SQL, והן המנוע שמפרשן אותה עברו אופטימיזציה לחיסכון באחסון. הרעיון הכללי שעומד מאחורי מסד נתונים רלציוני הוא הנורמליזציה, נרמול הנתונים, שמאפשר לשמור כל נתון פעם אחת בלבד. הסתכלות על הנתונים בכללותם מתבצעת באמצעות Join, כלומר ה־CPU של מנוע מסד הנתונים מבצע את היתוך המידע מתוך הטבלאות השונות. כיום אחסון הוא זול מאוד, ולכן אין צורך לחסוך במקום אחסון ולבצע נורמליזציה של הנתונים. על כן Join מאבד ממשמעותו. אי־קיומו של Join הוא אחת הסיבות למהירות האדירה של DDB בהחזרת תשובות.

רגע לפני שנעבור על המידול ב־DDB, נעבור על מונחים בסיסיים שנשתמש בהם לאחר מכן כדי למדל את המידע ולבצע עליו חיפושים:

- Attribute (מאפיין) – לכל פריט מידע שנכנס יכולים להיות כמה מאפיינים. לדוגמה, לסטודנט יש שם, תאריך לידה וכדומה. לא כל התכונות הן בנות חיפוש, כלומר יכול להיות שניתן לחפש לפי תאריך לידה בלבד ולא באמצעות שם.
- Item (פריט) – כמו שורה בטבלה, לכל פריט יכולים להיות כמה מאפיינים. ב־DDB אין סכמה שעל פיה עובדים, כלומר לכל פריט יכולים להיות מאפיינים שונים. חשוב לציין שאף על פי ש־DDB לא אוכף את הסכמה, מומלץ לעשות זאת בקוד.
- Table (טבלה) – כל טבלה מכילה כמה פריטים.
- Partition key (מפתח ראשי) – כאן באה לידי ביטוי אחת התכונות המרכזיות של DDB. בניגוד למסד נתונים רלציוני, שניתן לאנדקס בו כל מאפיין ולבצע באמצעותו שאילתה, ב־DDB ניתן להגדיר אך ורק מאפיין אחד כמאפיין מאונדקס. המאפיין המאונדקס נקרא "מפתח ראשי" ואפשר להגדיר רק אחד כזה. הערך של מפתח ראשי יכול להופיע רק פעם אחת בטבלה. השאילתה היחידה שניתן לבצע על מפתח ראשי היא שוויון, ולכן תמיד יוחזר ערך אחד לכל היותר.
- Sort key (מפתח מיון) – לפעמים מעוניינים לבצע שאילתה שמחזירה טווח ולאו דווקא ערך ספציפי. ניתן להגדיר מאפיין שכמפתח מיון יחזיר את כל הפריטים

שנמצאים בטווח מסוים. כמו המפתח הראשי, יכול להיות מפתח מיון אחד. לדוגמה, החזר את כל הפריטים בטווח תאריכים מסוים. ניתן לבצע שאילתות של >, <, =, startswith, between. לכל פריט בטבלה יכול להיות רק שילוב אחד של מפתח ראשי ומפתח מיון. מפתח שמוגדר על פי מפתח ראשי ומפתח מיון נקרא composite key.

- Query operation (שאילתה) – קריאת API המאפשרת לנו לקבל פריטים ספציפיים בטבלה לפי השוואה למפתח הראשי ולמפתח המיון.
- Scan operation (פעולת סריקה) – קריאת API המאפשרת לקבל את כל הפריטים בטבלה, ללא שום שימוש באחד המפתחות. בהתאם לגודל הטבלה, פעולה זו יכולה להיות איטית.

בואו נבין כיצד המונחים הבסיסיים האלה באים לידי ביטוי בכך שנבנה טבלת DDB שתייצג קורסים באוניברסיטה ואת כל הנרשמים אליהם. שימו לב, הטבלה היא ביחיד ולא ברבים, ואתעכב על נקודה זו בהמשך.

אחת הנקודות החשובות באפיון טבלת DDB היא שממדלים טבלה על בסיס השאילתות שרוצים להפעיל עליה. בניגוד למסד נתונים רלציוני, שבו טבלה מאפשרת להפעיל שאילתה על כל מאפיין, ב־DDB חייבים להגדיר מראש מה הם המאפיינים שעליהם מעוניינים להפעיל את השאילתות ואפשר לבחור לכל היותר שני מאפיינים – אחד מהם הוא המפתח הראשי והאחר הוא מפתח המיון.

כעת עלינו לחשוב אילו שאילתות נרצה לבצע בטבלה, כך שנוכל לקבוע את המפתח הראשי שלנו.

מהו השם של קורס שמספרו XYZ?

במקרה כזה, נוכל להגדיר את המאפיין מספר קורס כמפתח ראשי. מספר קורס הוא מאפיין ייחודי, ולכן אנו מניחים שכל מספר יופיע רק פעם אחת בטבלה. אם כך, שאילתה כגון course_id = 123 תחזיר פריט אחד בלבד.

course_id (primary key)	course_name
123	DDB 101

מצא את כל הסטודנטים שרשומים למספר קורס XYZ.
במקרה כזה אנו מחזירים טווח, ולכן עלינו להשתמש במפתח מיון על מאפיין המייל של הסטודנט. מפתח מיון אינו יכול להתקיים לבדו, אלא צריך גם מפתח ראשי.
שימו לב לנקודה חשובה נוספת - במסד נתונים רלציוני היינו שמים את פריט המידע הזה בטבלה אחרת. לעומת זאת, DDB מכווין לתבנית עיצוב (Design Pattern) שנקראת Single Table Design, ובה ממדלים את הישויות לכדי טבלה אחת. למה?

- ביצועים - אין צורך לגשת לכמה טבלאות על מנת לשלוף נתונים - locality.
- ניהול - אין צורך לנהל מבחינת ביצועים, קונפיגורציה, אבטחה ומספר טבלאות.

course_id (primary key)	data_type	course_ name	student_ name
123	COURSE	DDB 101	
123	STUDENT_guy@mail.com		Guy
456	COURSE	AWS 101	
456	STUDENT_guy@mail.com		Guy

במקרה כזה, נוכל להגדיר את המאפיין מספר קורס כמפתח ראשי וניצור עמודה נוספת שתאפשר לנו לבנות טבלה המכילה שני סוגי נתונים. נשמור בטבלה זו הן את פרטי הקורס והן את הסטודנטים הרשומים אליו. את המאפיין הנוסף נגדיר כמפתח מיון, וכעת נוכל לבצע שאילתות מהסוג הבא:

course_id = 123 and data_type starts_with STUDENT

עבור השאילתה הזאת נקבל את הפריט הבא:

123	STUDENT_guy@mail.com		Guy

אם נרצה לקבל את פרטי הקורס, נצטרך לעדכן את השאילתה כך:

course_id = 123 and data_type starts_with COURSE

ונקבל:

123	COURSE	DDB 101	

כמה נקודות חשובות שצפות כאן:

- הסכמה אינה מוגדרת, לא לכל הפריטים יש את כל המאפיינים.
- אנו משכפלים מידע - פרטי הסטודנט חוזרים על עצמם. זהו תהליך של דה־נורמליזציה, בניגוד מוחלט למה שקורה במסדי נתונים רלציוניים. אנו נמנעים מ־Join.
- שילוב של מפתח ראשי עם מפתח מיון הוא ייחודי, הערך

123	COURSE

יכול להופיע פעם אחת לכל היותר. לכן אנחנו מוסיפים לכל STUDENT את המייל שלו כדי ליצור ייחודיות.

- המידול שיצרנו עונה על השאלות הספציפיות שהגדרנו. איננו יכולים לענות על שאלות אחרות, כגון לאילו קורסים גיא רשום.

לאחר שלמדנו את הבסיס למידול ב־DDB, בואו ניצור טבלה לדוגמה דרך הקונסולה ונתשאל אותה.

יצירת טבלה ותשאולה

בחלק הנוכחי ניצור את טבלת הקורסים שתיארנו לעיל. נגדיר את עמודת המפתח הראשי ואת עמודת מפתח המיון.

1. בשורת החיפוש של הקונסולה רשמו DynamoDB ובחרו באופציה הראשונה שמופיעה.
2. בתפריט בצד שמאל בחרו Tables ולחצו על כפתור Create Table.

3. תנו לטבלה את השם Courses. תחת Partition Key כתבו PK ותחת Sort Key רשמו SK. שימו לב לסוג הנתונים שיישמרו במפתחות הללו - ודאו שהבחירה היא String.

Table details Info

DynamoDB is a schemaless database that requires only a table name and a primary key when you

Table name

This will be used to identify your table.

Courses

Between 3 and 255 characters, containing only letters, numbers, underscores (_), hyphens (-), and periods (.).

Partition key

The partition key is part of the table's primary key. It is a hash value that is used to retrieve items f hosts for scalability and availability.

PK | String

1 to 255 characters and case sensitive.

Sort key - *optional*

You can use a sort key as the second part of a table's primary key. The sort key allows you to sort o same partition key.

SK | String

1 to 255 characters and case sensitive.

4. לחצו על Create table.
5. לאחר שהבחירה הסתיימה, בחרו את הטבלה על ידי לחיצה על שמה.
6. כעת לא נתמקד בקונפיגורציות השונות (נעבור על חלקן בפרקים הבאים) אלא ביצירת פריטים חדשים ובתשאולים.
7. לאחר היצירה לחצו על שם הטבלה.
8. לחצו על Explore table items ושימו לב שהחלק התחתון של העמוד מאפשר ליצור פריטים חדשים. לחצו על Create Item.

9. העורך מציג לנו את שני המאפיינים שהגדרנו כמפתחות. שימו לב שבניגוד למסד נתונים רלציוני, ניתן להוסיף מאפיינים נוספים, אך לא לבצע שאילתות יעילות על המאפיינים הללו, כי הם אינם חלק מהמפתחות. המידע הזה יחזור אלינו כחלק מהתשובה לשאילתה. כמו כן המאפיינים שאינם חלק מהגדרת המפתחות אינם מחויבים להיות בכל הפריטים. נוסיף גם שם לחלק מהפריטים שלנו.

10. הוסיפו: את הערכים הבאים:

PK	SK	course_name	student_name
123	COURSE	DDB 101	
123	STUDENT_guy@mail.com		Guy
456	COURSE	AWS 101	
456	STUDENT_guy@mail.com		Guy

שימו לב, ניתן להוסיף מאפיינים נוספים על ידי לחיצה על Add new attribute:

לחצו בסיום של כל הוספה על Create Item.

11. בחלקו העליון של העמוד ניתן לבצע שאילתות:

Scan or query items

Scan | Query

Select a table or index: Table - Courses

Select attribute projection: All attributes

Filters

Run | Reset

12. יש שני סוגי שאילתות מרכזיות ב־DDB. האחד הוא שאילתת הבא הכול (Scan). זו שאילתה שמביאה את כל הפריטים בטבלה, ללא הצבת שום תנאי, ומזכירה בצורה כזאת או אחרת את select * from X בעולם מסדי הנתונים הרלציוניים. לחצו על Scan ומייד לאחר מכן על Run. שימו לב שאתם מקבלים ארבעה פריטים.
13. הסוג השני הוא שאילתה המחזירה ערכים ספציפיים התואמים תנאי מסוים (Query). כעת נבצע שאילתה שתביא לנו את פרטי קורס 123. בחרו Query. תחת PK רשמו 123 ותחת SK רשמו COURSE. הריצו את השאילתה. אתם אמורים לקבל ערך אחד.

Scan or query items

Scan | Query (1)

Select a table or index: Table - Courses

Select attribute projection: All attributes

PK (Partition key): 123 (2)

SK (Sort key): Equal to | COURSE (3) | Sort descending

14. כעת נחזיר את כל הסטודנטים שנמצאים בקורס 123. שנו את סוג השאילתה תחת ה־SK ל־Begins With ורשמו שם STUDENT. הריצו. אתם אמורים לקבל פריט אחד.

▼ Scan or query items

○ Scan | ◉ Query (1)

Select a table or index: Table - Courses ▼

Select attribute projection: All attributes ▼

PK (Partition key): 123 (2)

SK (Sort key): Equal to ▼ | COURSE (3) | ☐ Sort descending

15. ננסה למצוא את כל המשתמשים שנקראים גיא. המידול לא מאפשר לנו לבצע שאילתה כזאת, אנו גם לא יכולים להניח שהמייל של המשתמש מעיד על שמו הפרטי. ניתן לפתור את הבעיה בכמה דרכים:
    א. למדל מחדש את הטבלה כך שנוכל לענות על השאילתה.
    ב. לבצע פעולת Scan ולפלטר את המאפיין של השם.
16. בחרו Scan ותחת Filter הוסיפו ערך חדש עבור המאפיין student_name. אתם אמורים לקבל שני פריטים.

○ Scan | ◉ Query (1)

Select a table or index: Table - Courses ▼

Select attribute projection: All attributes ▼

PK (Partition key): 123 (2)

SK (Sort key): Begins with ▼ | STUDENT (3) | ☐ Sort descending

▶ Filters

Run | Reset

17. נראה שפעולת הפִלטוּר באמצעות Scan פותרת את כל הבעיות. ניתן להפעיל כמעט כל תנאי על המאפיינים, אבל ישנו חיסרון משמעותי והוא העובדה שעדיין מתבצעת פעולת סריקה מלאה, קרי פעולה שעוברת על כל הנתונים והיא איטית מטבעה. רק לאחר הבאת כל הנתונים מתבצע מאחורי הקלעים תהליך הפלטור. אם הנחת העבודה שלנו היא שמספר הערכים בטבלה נמוך, אין מניעה מלהשתמש בפעולת Scan.

אנו מוכנים לשלב האחרון, רגע לפני שנמשיך בפרויקט, והוא מידול המידע עבור הפרויקט.

הגיע הזמן לקודד

רגע לפני שנכתוב את הקוד שייגש לטבלאות שניצור, עלינו למדל את הנתונים, ומעל לכול להבין מהו סוג השאלות שנרצה לשאול, כי זו הדרך היחידה למדל נכון את מסד הנתונים.
להזכירכם, נבנה מערכת לניהול רשימות תפוצה. המערכת תציג את הדברים הבאים:

- ממשק ניהול המאפשר ליצור ולמחוק רשימות תפוצה.
- ממשק המאפשר למשתמשים:
  - להירשם לרשימת תפוצה.
  - לבטל רישום לרשימת תפוצה.

השאלות שנרצה לענות עליהן:

1. לאילו רשימת תפוצה משתמש Y רשום?
2. אילו משתמשים רשומים ברשימת תפוצה Y?

יש לנו שני סוגי פריטים – רשימת תפוצה ומשתמשים. הקשר ביניהם הוא many to many, כלומר רשימת תפוצה מכילה המון משתמשים ומשתמשים רשומים להמון רשימות תפוצה. ישנם פתרונות רבים למידול של many to many ב־DDB. אנו נבחר פתרון שהוא יחסית פשוט אך אינו מיטבי (נסביר בהמשך מדוע).

בטבלה זו נשמור שני סוגי נתונים:

- משתמשים - עבור כל משתמש והקבוצה שאליה הוא רשום קיים פריט נפרד.
- רשימות תפוצה - יהיה פריט שיכיל פרטים כלליים על הקבוצה ופריט נפרד לכל משתמש שרשום לקבוצה.

<table>
<tr><th>PK</th><th>SK</th><th>Attributes</th></tr>
<tr><td>USER#guy@mail.com</td><td>AWS</td><td>None</td></tr>
<tr><td>GROUP#AWS</td><td>#METADATA</td><td><table><tr><th>Name</th><th>Description</th></tr><tr><td>AWS</td><td>Lorem Ipsum</td></tr></table></td></tr>
<tr><td>GROUP#AWS</td><td>USER#guy@mail.com</td><td>None</td></tr>
</table>

ישנם כמה מאפיינים למידול הנ"ל:

- הוא משתמש בטבלה אחת - one table design. זוהי טבלה בודדת המכילה את שני סוגי הפריטים.
- על מנת לבדל בין שני הסוגים השונים נשתמש בתחיליות מבדלות.
- כל פריט מתאפיין במאפיינים שונים - כפי שאמרנו בתחילת הדרך, אין סכמה מוגדרת לטבלה, אך יש סכמה ברורה, שתיאכף על ידי קוד לכל סוג פריט.
- המידול משכפל נתונים - הן המייל של המשתמשים והן שם הקבוצה חוזרים מספר רב של פעמים. משמעות הדבר היא שבכל פעם שמייל של משתמש או שם הקבוצה ישתנו, המערכת תצטרך לעבור על כמה פריטים ולשנות את ערכם. בלא ספק, זה לא יעיל במיוחד, ומדובר בחיסרון של הפתרון המוצע כאן, אך יש לזכור שפעולה של שינוי מייל של משתמש או של שם קבוצה אינה נפוצה, ולכן חוסר היעילות פחות נורא.

שאילתות

בואו נראה כיצד שאילתות שונות מחזירות לנו את המידע שאנו מחפשים.

1. מחקו את הטבלה הקודמת שיצרתם. עשו זאת באמצעות בחירתה ומתפריט ה-Actions בחרו Delete table. אשרו את המחיקה באמצעות כתיבת הערך delete.

Delete table

You are about to delete a table.

- Subscriber

☑ Delete all CloudWatch alarms for this table.

☐ Create a backup of this table before deleting it.
If you do not select this check box, you will not be able to restore data being deleted.

To confirm the deletion of this table, type *delete* in the box.

delete

Cancel | Delete table

2. צרו טבלה חדשה בשם GroupsSubscribers. שם המפתח הראשי יהיה PK ושם מפתח המיון יהיה SK, שניהם מסוג String. לחצו על יצירת טבלה.
3. כדי לחסוך עבודה ביצירת התוכן של הטבלה שמרו את הקובץ GroupsSubscribers.json שנמצא בספרייה dynamodb-chapter בגיט של הספר לסביבת Cloud 9.
4. הריצו דרך הטרמינל את הפקודה הבאה:

```
aws dynamodb batch-write-item --request-items file://GroupsSubscribers.
json
```

5. הפקודה טוענת נתונים מוכנים מראש בטבלה שיצרתם. גשו אל הטבלה דרך הקונסולה ובחרו Explore Table Items.
6. ודאו ש־Scan נבחר ולחצו על Run. אמורים להופיע שמונה פריטים.

רשימת כל הרשומים לקבוצה מסוימת

שאילתה ראשונה מחזירה את רשימת כל המשתמשים הרשומים לקבוצה ספציפית, לדוגמה קבוצת AWS. הערך של PK הוא GROUP#AWS, הערך של SK הוא USER# והתנאי ל־SK הוא Begins with.

Scan
Query
Select a table or index
Table - GroupsSubscribers
Select attribute projection
All attributes
PK (Partition key)
USER#guy@mail.com
SK (Sort key)
Begins with
Enter sort key value
Sort descending

השאילתה כאן גם היא מחזירה טווח. היא אמורה להחזיר שני פריטים.

רשימת כל הקבוצות שהמשתמש רשום אליהן

שאילתת טווחים נוספת אמורה לאפשר למשתמש לקבל את רשימת כל הקבוצות שהוא חבר בהן. לדוגמה, המשתמש guy@mail.com. הערך של PK הוא USER#guy@mail.com, ואילו הערך של SK הוא ריק.

Scan
Query
Select a table or index
Table - GroupsSubscribers
Select attribute projection
All attributes
PK (Partition key)
USER#guy@mail.com
SK (Sort key)
Begins with
Enter sort key value
Sort descending

השאילתה כאן אמורה להחזיר פריט אחד.

רשימת כל הקבוצות

הטבלה לא נבנתה להחזיר את רשימת כל הקבוצות בשאילתה בודדת, וזאת למען הפשטות במידול. לשם כך ניתן להשתמש ב־Scan. נבדוק שהערך של SK שווה ל־METADATA#:

Scan | Query

Select a table or index: Table - GroupsSubscribers

Select attribute projection: All attributes

Filters

Attribute name	Type	Condition	Value	
PK	String	Begins w...	GROUP#	Remove
SK	String	Equal to	METADATA#	Remove

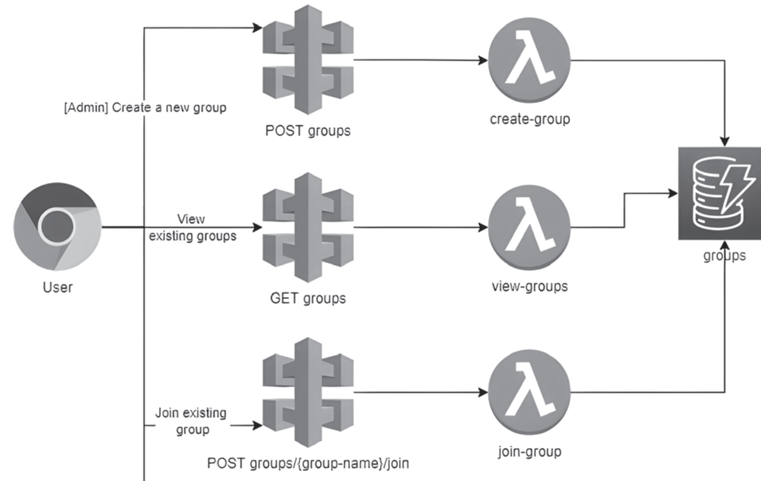
השאילתה כאן אמורה להחזיר שני פריטים.

פרטי קבוצה

השאילתה האחרונה מחזירה פריט בודד ואינה שאילתת טווחים. היא מחזירה את שם הקבוצה ואת תיאורה. נסו, כחלק משאלות ההרחבה, ליצור שאילתה כזאת.

כעת, כשיש בידנו את השאילתות ומידול הנתונים המתאים לכך, אנו מוכנים לכתוב את הקוד עבור הלומדות שלנו.
מחקו את הטבלה שיצרתם.

מתחילים לבנות



הפתרון מורכב מטבלה אחת שתספק מענה לשלוש פעולות:

- צפייה בקבוצות קיימות.
- יצירת קבוצה חדשה.
- הצטרפות לקבוצה חדשה.

המידול של הטבלה יתבסס על החלק הקודם בפרק.

יצירת טבלה

הוסיפו את החלק הבא תחת ההגדרה של ViewGroupsFunction, שנמצאת בקובץ template.yaml:

```
1.   GroupsSubscribersTable:
2.        Type: AWS::DynamoDB::Table
3.        Properties:
4.          AttributeDefinitions:
5.             - AttributeName: PK
6.               AttributeType: S
7.             - AttributeName: SK
8.               AttributeType: S
9.          KeySchema:
10.            - AttributeName: PK
```

```
11.             KeyType: HASH
12.           - AttributeName: SK
13.             KeyType: RANGE
14.         BillingMode: PROVISIONED
15.         ProvisionedThroughput:
16.           ReadCapacityUnits: 5
17.           WriteCapacityUnits: 5
```

- AttributeDefinitions (שורות 8-4) - מגדיר את המאפיינים שנאנדקס באמצעות מפתח ראשי או מפתח מיון. במקרה שלנו אנו מגדירים את SK ו־PK, שמכילים שניהם ערכים מסוג מחרוזת.
- KeySchema (שורות 13-9) - מגדיר את הסכמה של המפתחות. המאפיין PK מוגדר כמפתח ראשי, ואילו SK מוגדר כמפתח מיון.
- נתעלם כרגע משאר ההגדרות.

הריצו באמצעות sam deploy && sam build, וטבלה חדשה אמורה להיווצר. אתם יכולים לבדוק את שמה בקונסולה.

על מנת להקל עלינו את העבודה נטען את הטבלה במידע מוכן מראש. השתמשו באותו קובץ שהשתמשנו בו בחלק הקודם על מנת לטעון מידע. הדבר היחיד שצריך לשנות בו הוא שם הטבלה. פתחו אותו ובשורה השנייה שנו את השם מ־GroupsSubscribers לשם הטבלה שלכם והריצו:

```
aws dynamodb batch-write-item --request-items file://GroupsSubscribers.json
```

צפייה בקבוצות קיימות

כעת נשנה את הפונקציה שאיתה אנו מקבלים את רשימת הקבוצות הקיימות. תחת view_groups שנו את התוכן של app.py לקוד הבא:

```
1.  import boto3
2.  from boto3.dynamodb.conditions import Attr
3.  import json
```

```
4.     import os
5.
6.     dynamodb = boto3.resource("dynamodb")
7.     table = dynamodb.Table("subscribers-GroupsSubscribersTable-
8.     XZMH8QWGK79J")
9.
10.    def lambda_handler(event, context):
11.        response =
12.    table.scan(FilterExpression=Attr('SK').eq('METADATA#'))
13.        return response
```

מדוע אנו מאתחלים את הטבלה מחוץ ל־handler?

בתחילת הקובץ (שורות 6-7) אנו מבצעים אתחול של הטבלה שניגש אליה. אתחול המשתנים שניגשים למשאבים השונים מחוץ לפונקציית ה־handler הוא סוג של זיכרון מטמון (cache). זִכרו - Lambda מאותחלת בשתי צורות:

- cold
- warm

אתחול cold קורה רק פעם אחת, ובמהלכו שורות הקוד מחוץ ל־handler נקראות, ולכן אתחול משתנים מחוץ ל־handler מאפשר לנו להימנע מאתחול חוזר ונשנה בריצות הבאות של ה־Lambda.

שימו לב שצריך לשנות את שם הטבלה לפי השם שמופיע בקונסולה (שורה 7). הקוד כשלעצמו פשוט, אובייקט טבלה מאותחל עם שם הטבלה המתאימה, שבעזרתו אנו מבצעים סריקה פשוטה ומפלטרים את הפריטים שמכילים במאפיין SK את הערך METADATA.

הפונקציה זקוקה להרשאות קריאה על מנת לגשת לטבלה. בקובץ template.yaml, תחת ההגדרה של ViewGroupsFunction, הוסיפו הרשאות קריאה:

```
Policies:
    - DynamoDBReadPolicy:
        TableName: !Ref GroupsSubscribersTable
```

שמות ההרשאות נלקחות מ:
https://docs.aws.amazon.com/serverless-application-model/latest/developerguide/serverless-policy-templates.html.
אם תלחצו על הקישור ששמו DynamoDBReadPolicy המופיע בטבלה, ייפתח עמוד המכיל את הגדרת המקור של ההרשאה. שימו לב שלהגדרת המקור ישנו קטע קוד הנראה כך:

```
"tableName": {
 "Ref": "TableName"
}
```

הערך שמופיע לאחר ה־Ref הוא שם המשתנה שההרשאה הזאת צריכה כדי לעבוד. במקרה שלנו המשתנה מכיל את שם הטבלה שעליה אנו מעוניינים לקבל הרשאות קריאה.

הריצו באמצעות sam deploy && sam build ובצעו ריצת בדיקה דרך הקונסולה. אמורה לחזור תשובה דומה לזאת:

```
1.    {
2.        "Items": [
3.          {
4.            "SK": "METADATA#",
5.            "description": "Everything about AWS",
6.            "PK": "GROUP#aws",
7.            "name": "AWS"
8.          },
9.          {
10.           "SK": "METADATA#",
11.           "description": "Daily news about technology",
12.           "PK": "GROUP#tech-news",
13.           "name": "Tech news"
14.         }
15.       ],
```

```
16.         "Count": 2,
17.         "ScannedCount": 8,
18.         "ResponseMetadata": {
            ...
      }
```

התשובה מורכבת משני חלקים מרכזיים:

1. הקבוצות עצמן יחד עם המאפיינים שלהן (שורות 2-15) - זה המידע שנרצה, לאחר שינויים קלים, להחזיר למשתמשים.
2. מידע כללי על הבקשה (שורות 16-18) - במקרה של כישלון, יהיה כאן מידע נוסף שיעזור לנו להבין את מקור הבעיה.

בואו נשנה את הקוד, כך שיחזיר רק את המידע שמעניין אותנו:

- שם הקבוצה.
- מזהה חד־חד־ערכי לקבוצה.
- תיאור הקבוצה.

שנו את ה־lambda_handler לקוד הבא:

```
1.  def lambda_handler(event, context):
2.      response =
3.  table.scan(FilterExpression=Attr('SK').eq('METADATA#'))
4.      items = response["Items"]
5.      groups = []
6.      for item in items:
7.          group = {
8.              "id": item["PK"],
9.              "name": item["name"],
10.             "description": item["description"]
11.         }
12.         groups.append(group)
13.
```

```
14.     return {
15.         "statusCode": 200,
16.         "body": json.dumps(groups)
17.     }
```

- שורה 2 – מבצעים שאילתה על מנת לקבל את רשימת כל הקבוצות.
- שורות 12-6 – מתבצעת לולאה על כל הערכים שנמצאים תחת המאפיין Items. עבור כל קבוצה אנו מייצרים מבנה json המכיל id ו־name description.
- שורה 14 – מחזירים תשובה בפורמט ש־API Gateway מצפה לקבל.

הריצו sam deploy && sam build. הפעם בדקו את זה כחלק מקריאת API. אתם אמורים לקבל את הדומיין של ה־Api Gateway בסיום הריצה של AWS SAM.

```
Outputs
-----------------------------------------------------------------------------------------
Key                 ViewGroupsApi
Description         API Gateway endpoint URL for Prod stage for ViewGroupsFunction function
Value               https://          .execute-api.us-east-1.amazonaws.com/Prod/groups/
-----------------------------------------------------------------------------------------
```

רגע לפני שנכתוב את שאר ה-Lambda, נקודה אחת מציקה לי מאוד, והיא העובדה שאנו מקודדים את שם הטבלה בקוד. הבעיה בצורת כתיבה כזאת היא שבכל פעם שמתבצעת התקנה של האפליקציה בסביבה אחרת, צריך לגשת ולשנות את הקוד. במקרה שלנו, כשיש Lambda אחת וכל מה שנדרש הוא לשנות שורה אחת, הדבר פחות נורא, אך חשבו מה קורה כשיש מאות Lambda ומאות טבלאות. זה מתכון לאסון.

הפתרון הנכון הוא בעצם לקבל את השם מ־AWS SAM ולהזריק אותו לתוך הקוד. נעשה זאת באמצעות משתני סביבה.

בקובץ temple.yaml הוסיפו תחת Properties בהגדרה של ה־Lambda את התוכן הבא:

```
Environment:
    Variables:
      GROUPS_SUBSCRIBERS_TABLE_NAME: !Ref GroupsSubscribersTable
```

אנו מגדירים כאן משתנה סביבה בשם GROUPS_SUBSCRIBERS_TABLE_NAME שערכו הוא שם הטבלה. GroupsSubscribersTable הוא השם הלוגי שמופיע ב־template.yaml. כשמשתמשים בפונקציה Ref של AWS SAM על השם הלוגי של הטבלה, היא מחזירה את השם האמיתי שלה בסביבה.

בקובץ app.py שנו את שורת אתחול הטבלה לשורה הבאה:

```
table = dynamodb.Table(os.environ["GROUPS_SUBSCRIBERS_TABLE_NAME"])
```

אנו טוענים כאן את ערך משתנה הסביבה שיצרנו בסעיף הקודם והוא ישמש אותנו כשם הטבלה.

כרגיל, הריצו sam build && sam deploy ובדקו את השינוי.

דרך נוספת לבדוק את הימצאותם של משתני סביבה היא לגשת לקונסולה. תחת הלשונית של Configuration תמצאו לשונית נוספת של Environment variables:

Code | Test | Monitor | Configuration | Aliases | Versions

General configuration
Triggers
Permissions
Destinations
Function URL
Environment variables

Environment variables (1)
The environment variables below are encrypted at rest with the default Lambda service key.

Key	Value
GROUPS_SUBSCRIBERS_TABLE_NAME	sam-app-GroupsSubscribersTable-1W0AV8I664A05

שם תראו את משתני הסביבה שהוגדרו עבור ה־Lambda.

יצירת קבוצה חדשה

הוסיפו בקובץ template.yaml פונקציה חדשה עם התוכן הבא:

```
1.        CreateGroupFunction:
2.            Type: AWS::Serverless::Function
3.            Properties:
4.               CodeUri: create_group/
5.               Handler: app.lambda_handler
6.               Runtime: python3.7
7.               Environment:
8.                  Variables:
9.                      GROUPS_SUBSCRIBERS_TABLE_NAME: !Ref
10.     GroupsSubscribersTable
11.              Policies:
12.                 - DynamoDBWritePolicy:
13.                     TableName: !Ref GroupsSubscribersTable
14.              Events:
15.                 MailingList:
16.                    Type: Api
17.                    Properties:
18                       Path: /groups
19.                       Method: post
```

שימו לב שמיקום הקוד CodeUri (שורה 4) השתנה, שסוג ההרשאות השתנה לכתיבה בלבד DynamoDBWritePolicy (שורה 12) ושהפונקציה מחוברת ל־API Gateway באמצעות פועל post (שורה 19).

לאחר שהוספנו את ההגדרה, יש להוסיף את הקוד. במקרה שלנו, הדבר הפשוט ביותר יהיה לשכפל את הספרייה view_groups, לשנות את שמה ל־create_group (שעליה CodeUri מצביע) ולהחליף את התוכן של app.py בתוכן הבא:

```
1.  import boto3
2.  import json
3.  import os
4.  import re
5.
6.  dynamodb = boto3.resource("dynamodb")
7.  table = dynamodb.Table(os.environ["GROUPS_SUBSCRIBERS_TABLE_NAME"])
8.
9.  def lambda_handler(event, context):
10.     group_details = json.loads(event["body"])
11.     # Only alphabetic characters and space are allowed
12.     if not re.match(r'[a-zA-Z\s]+$', group_details["name"]):
13.         body = {"description": "Only alphabetic characters and space
14.  are allowed"}
15.         return {"statusCode": 500, "body": json.dumps(body)}
16.     group_name = group_details["name"]
17.     group_id = re.sub(r'\s+', '-', group_name)
18.     group_description = group_details["description"]
19.
20.     item = {"PK": f"GROUP#{group_id}", "SK": "METADATA#", "description":
21.   group_description, "name": group_name}
22.     table.put_item(Item=item)
23.
24.     return {"statusCode": 200, "body": json.dumps(item)}
25.
```

כל בקשה מכילה שני מאפיינים - שם הקבוצה ותיאורה - ולכן אנו מצפים שבקשה תהיה בעלת המבנה הבא:

```
{
    "name":"Welcome group",
    "description":"Hello group"
}
```

- שורות 7-6 - אתחול הטבלה שמתבצע מחוץ ל־handler.
- שורה 10 - תוכן הבקשה מועבר כמאפיין ששמו body כחלק ממשתנה ה־event.
- שורות 14-12 - לאחר טעינת הבקשה לתוך group_details אנו מבצעים בדיקה ששם הקבוצה מכיל רק תווים תקינים. תו תקין הוא מספר, רווח או אות לטינית. הבדיקה מתבצעת באמצעות regex expression. אם הבדיקה נכשלת, מחזירים למשתמש קוד 500 עם תיאור הבעיה. שימו לב שהמבנה תואם את התשובה ש־API Gateway מצפה לה.
- שורה 17 - מחליפים כל רווח בתו -, וזאת בעיקר על מנת לגשת לקבוצה באמצעות כתובת URL בהמשך הדרך. מבחינת DDB, לא אמורה להיות בעיה עם רווחים.
- שורות 21-20 - לסיום מבצעים את הכתיבה לפי המבנה שהגדרנו בפסקאות הקודמות ומחזירים קוד 200. כדי לפשט את כתיבת המחרוזת של SK, נשתמש במאפיין ייחודי של Python, המאפשר לכתוב מחרוזת שבתוכה ניתן להכניס משתנים. לשם כך, לפני תחילת המחרוזת נוסיף את התו f ולציון משתנה נקיף אותו ב־{}.

כרגיל הריצו sam deploy && sam build. לסיום בדקו את היצירה באמצעות שימוש בפקודת curl. הריצו בטרמינל את הפקודה הבאה:

```
curl --request POST https://<id>.execute-api.us-east-1.amazonaws.com/
Prod/groups/ -d '{"name":"Welcome group","description": "Hello group"}'
```

אתם אמורים לקבל קוד 200. מייד לאחר מכן בצעו קריאה על מנת לקבל את רשימת כל הקבוצות וראו שהקבוצה הצטרפה לרשימת הקבוצות הקיימות.

הצטרפות לקבוצה חדשה

את ה־Lambda האחרונה אשאיר לכם לכתוב. ניתן לראות את הפתרון המלא בגיט של הספר.[30] ישנן כמה נקודות שחשוב להכיר לפני הכתיבה.

ה־URL שמריץ את ה־Lambda אמור להיות עם Path Parameters, כלומר דומה ל: /Prod/groups/Welcome-group

30 https://github.com/aws-hebrew-book/dynamodb-chapter.

מגדירים את ה־Path Parameters באמצעות הוספת טקסט מוקף בסוגריים מסולסלים שמגדירים את ה־ID של הקבוצה שאנו מעוניינים להצטרף אליה. הגדרה כזאת מתבצעת בצורה הזאת בקובץ ה־template.yaml (שורה 5):

```
1.    Events:
2.          MailingList:
3.            Type: Api
4.            Properties:
5.              Path: /groups/{group-id}
6.              Method: post
```

לאחר מכן ניתן לגשת לפרמטר הזה בקוד באמצעות הקריאה הבאה:

```
event["pathParameters"]["group-id"]
```

עלויות[31]

לכל שירות ב־AWS יש עלות הפעלה. לחלק מהשירותים יש מסלול חינמי (Free Tier) לכמות מסוימת של עבודה. נבצע סקירה של העלויות של הפעלת השירות ושל מה שנכלל כחלק מהמסלול החינמי. כל העלויות המוזכרות כאן הן עבור אזור us-east-1 נכון לפברואר 2024

ישנם שני מודלים של שימוש ב־DDB:

- On Demand - מתבצע תשלום אך ורק בזמן פעולה על הטבלאות, כגון קריאה, כתיבה, שאילתה וכדומה. זהו המודל הקלאסי של Serverless.
- Provisioned Capacity - אם אפשר להתחייב מראש למספר הפעולות שיתבצעו בכל רגע נתון, לדוגמה כתיבה של חמישה פריטים לכל היותר בכל שנייה. אומנם נשלם על הפעולות הללו מראש, גם אם לא ביצענו אותן, אך התשלום יהיה זול משמעותית מ־On Demand.

פעולות קריאה יהיו תמיד זולות יותר.

31 https://aws.amazon.com/dynamodb/pricing/.

On demand

- כתיבה - מיליון פעולות כתיבה עולות כ־1.25 דולר.
- קריאה - מיליון פעולות קריאה עולות כ־0.25 דולר.
- שמירת נתונים - עבור כל ג'יגהבייט של נתונים העלות היא 0.25 דולר. 25 ג'יגהבייט ראשונים הם חינמיים.

Provisioned Capacity

- כתיבה - התחייבות לפעולת כתיבה אחת לשנייה. עלות לשעה - 0.00065 דולר.
- קריאה - התחייבות לפעולת קריאה אחת לשנייה. עלות לשעה - 0.00013 דולר.

בניגוד למודל ה־On Demand, התשלום במקרה זה הוא לשעת עבודה. ניתן לקרוא את המספרים הללו בצורה הבאה: אם אני מתחייב ל־1,000 פעולות כתיבה, על כל שעת התחייבות אשלם 0.00065 * 1,000 = 0.65 דולר, או בחישוב יומי 15.6 דולר.

- שמירת נתונים - כמו במסלול הראשון, עבור כל ג'יגהבייט של נתונים העלות היא 0.25 דולר. 25 ג'יגהבייט הראשונים הם חינמיים.

בניגוד ל־On Demand, יש מסלול חינמי ל־Provisioned Capacity של עד 25 פעולות בשנייה בחינם.

ניקוי הסביבה בסיום העבודה

בסיום הפרק מומלץ לנקות את סביבת ה־AWS מהשירותים השונים שיצרתם במהלכו. ניתן לבצע זאת באמצעות הרצה של **sam delete**.

תרגילים להרחבה

1. כחלק מהעבודה על טבלת GroupsSubscribers, החזירו את פרטי המשתמש שכתובת המייל שלו היא gili@m.io.

2. ב־Lambda אשר מוסיפה קבוצה חדשה, בצעו בדיקה אם הקבוצה כבר קיימת. אם היא קיימת, החזירו תשובה.
3. הוסיפו Lambda המצרפת מייל לקבוצה קיימת. בדקו שה־Lambda עובדת באמצעות שימוש ב־Curl ובקונסולה.

ניתן למצוא את כל דוגמאות הקוד המופיעות בפרק זה בגיטהאב של הספר בכתובת https://github.com/having-fun-serverless/learning-serverless-in-hebrew/tree/main/dynamodb-chapter