Тема: SNS симулатор

Предмет: Приложно-програмни интерфейси за работа с облачни архитектури с Амазон Уеб Услуги (AWS)

Изготвил: Бориславa Анатолиева Багалийска, фн: 81665, имейл: bbagaliyska@gmail.com

Лектор: доц. Милен Петров, година: 2019/2020

Съдържание

1 Условие

Serverless симулация на услугата SNS на AWS.

SNS е услуга за изпращане на известия към различни видове получатели. Във варианта, в който е реализиран в проекта, възможни получатели на съобщения могат да бъдат HTTP URL-и, AWS Lambda функции, AWS SQS опашки за съобщения и имейл клиенти.

2 Въведение

Тъй като в основата на AWS услугите е те да са скалируеми и с висока наличност, избрах да реализирам всички функционалности от условието в AWS пространството, възползвайки се от стабилността и свойствата на различните услуги, които то предлага. Избрах имплементацията да е serverless, защото нямах опит с подобни приложения досега и мисля, че предлага по-голяма гъвкавост за подобрение и разширение на решението, както и по-ефикасно използване на хардуерните ресурси.
Популярна опция за вдигане на serverless приложение, използвайки AWS е използване на комбинация от AWS Lambda и DynamoDB, което е и начина, по който е организирана и инфраструктурата на проекта. AWS услугите, използвани в решението са:

- Lambda
- DynamoDB
- SQS
- SES
- IAM

3 Теория

Ще опиша различните лимитации на двете основни използвани услуги и как е добре да се адресират.

Използвайки лимитите по подразбиране, **AWS Lambda** поддържа 1000 конкурентни изпълниния, 75GB данни, от които за едно асинхронно извикване са позволени 256 KB, а за синхронно - 6 MB. Текущото решение не се опитва да оптимизира тези стойности, но със сравнително малко промени би могло да

го прави. Идеята на всяка функция е да има сравнително лека и бърза задача и при нужда да се обръща към други Lambda функции, за да реализират различни подзадачи. Подобно нещо се случва при основната Lambda, приемаща заявка за изпращане на съобщение, която делегира към Lambda функции, обработващи само един вид съобщения. Това, което не е взето предвид е големината на входните данни - хубаво е ако надвишават горепосочените лимити, да се разбиват на по-малки блокове и да се дават за паралелна обработка от няколко инстанции на Lambda функции.
DynamoDB предлага по 40,000 единици заявки за четене и 40,000 единици заявки за писане на таблица. Приложението използва една таблица, така че това са и лимитите на приложението. Единицата в случаи на консистентно четене представлява едно четене в секунда на данни до 4KB; съответно едно писане в секунда на данни до 1KB. За писане това е предостатъчно, тъй като начина на използване на приложението най-вероятно няма да води до толкова чести записи - записът представлява добавяне на тема или абонат към тема. Към момента приложението е направено така, че да се инсталира локално в инсфраструктурата на всеки ползващ го потребител, така че за един потребител при нормална употреба тези стойности са недостижими. Вътрешно при изпращане на съобщение се четат всички абонати към съответната тема от базата и ако те са много на брой е възможно да трябва да се чете на части от няколко инстанции на ламбди, но това не е реализирано. Адресирах възможността за паралелни заявки, като структурирах данните по определен начин в таблицата в различни атрибути от тип StringSet, защото по този начин може добавянето и махането на абонат да стане в една заявка вместо с две (четене и после промяна) и така процеса е атомарен, стига вътрешно AWS да са реализирали атомарно заявката.

4 Използвани технологии

Първоначално бях замислила да се използва Java 11, но реших да експериментирам с Python 3.8, с който не бях работила досега, но изглеждаше като удобен избор за целта.

- **Python** - version 3.8.
- **boto3** - version 1.13.9.

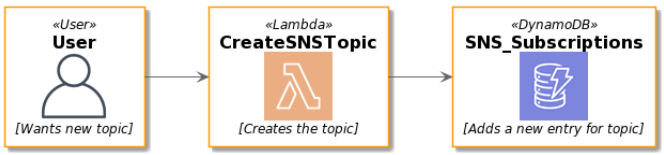
5 Инсталация и настройки

Стъпка 1. Вдигане на нужната инсфраструктура, зададена във файла configurations.md

Стъпка 2. Сваляне на клиентската библиотека на Python 3

6 Кратко ръководство за потребителя

Графики, които описват движението на контрола при заявка от потребителя:
Създаване на тема



Фигура 1: Създаване на тема

Изтриване на тема

Фигура 2: Изтриване на тема

Добавяне на абонамент

Фигура 3: Добавяне на абонамент

Премахване на абонамент
Изпращане на съобщение

Таблица 1: SNS_Subscriptions DynamoDB table - Emails

Topic	email_subscribers
news	{"email1gmail.com" "email2gmail.com"}
software-updates	{'email3gmail.com'}

Таблица 2: SNS_Subscriptions DynamoDB table - SQS

Topic	sqs_subscribers
news	{"https://sqs.region-name.amazonaws.com/account-id/queue-id"}
software-updates	{}

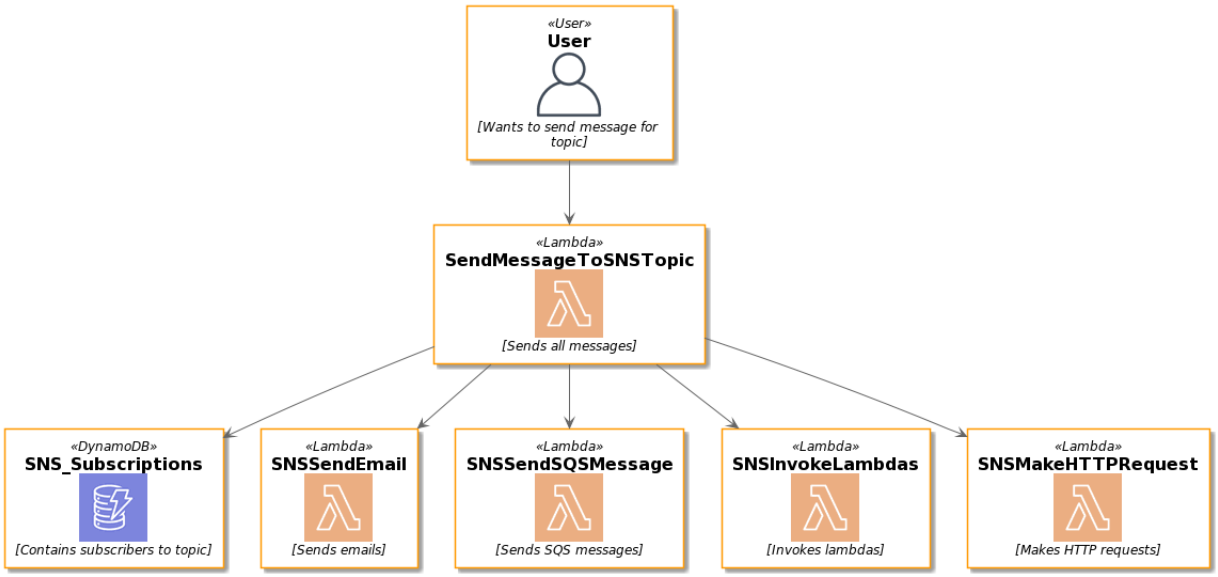
Таблица 3: SNS_Subscriptions DynamoDB table - Lambda

Topic	lambda_subscribers
news	{}
software-updates	{"arn:aws:lambda:region-name:account-id:function:lambda-id"}

Таблица 4: SNS_Subscriptions DynamoDB table - HTTP

Topic	http_subscribers
news	{}
software-updates	{"https://www.google.com/"}

Фигура 4: Премахване на абонамент



Фигура 5: Изпращане на съобщение

7 Примерни данни за SNS_Subscription таблицата

8 Описание на програмния код

8.1 Функции за работа с DynamoDB

- CreateSNSTopic
- DeleteSNSTopic
- SubscribeToSNSTopic
- UnsubscribeFromSNSTopic
- SendMessageToSNSTopic

8.2 Функции за работа с други AWS Lambdas

- SNSInvokeLambdas
- SendMessageToSNSTopic

8.3 Функции за работа с AWS SQS

- SNSSendSQSMessage

8.4 Функции за работа с AWS SES

- SNSSendEmail

Базата данни е реализирана като има ключ (partition key) - темата. За всяка тема (topic) се пазят по един атрибут за всеки тип абонамент - имейл, HTTP, SQS, Lambda. Тези атрибути представляват StringSets от идентификатори на абонираните ресурси. За тип имейл - множество от имейли, към които да се пращат електронни съобщения - в случаи на потребители в SandBox тези имейли трябва да бъдат верифицирани в SES За тип HTTP - множество от URL-и, към които да се пращат POST заявка За тип SQS - множество от URL идентификатори на SQS опашки, за които съответния потребител трябва да има достъп да изпраща съобщение За тип Lambda - множество от ARN идентификатори (може и имена) на Lambda функции, които съотвения потребител трябва да има право да изпълнява В тази база се записват новите теми и абонати и от нея се четат, когато трябва да се изпрати съобщение.

9 Приноси на студента, ограничения и възможности за бъдещо развитие

9.1 Приноси на студента

Реализирана е основната функционалност на SNS, използвайки редица AWS услуги, severless approach и избор на програмен език и инструменти, които позволяват изчистена и ясна имплементация, отворена към разширение на функционалностите и реализиране на основни нефункционални изисквания, на които се базира AWS: high availability, scalability, durability, security.

9.2 Ограничения

Реализацията към момента се ограничава от това, че инфраструктурата се инсталира в локалното AWS пространство на всеки използващ го потребител, вместо да се обръща към endpoints. Това решение е взето главно заради това, че услугите не са безплатни и студентският акаунт е по-скоро тестови цели и лимитиран. Лесно може да се мине и на централно решение без да се модифицира текущия код. Други ограничения са свързани с ограниченията в капацитета на ламбдите, които могат да се адресират с вкарване на паралелизъм, четене от базата на по-малки порции и пускане на повече от една ламбда за една задача, но не успях да реализирам тези неща от времеви ограничения.

9.3 Възможности за бъдещо развитие

- Поддържане на други типове абонаменти - като пращане на SMS или push notifications
- Пазене на мета информация за всеки тип тема или потребител, които да могат да се конфигурират при създаване
- Вкарване на многонишково програмиране и on-demand викане на нужния брой ламбди за обработване на по-големи заявки
- Интегриране с Dead Letter Queue и/или CloudWatch за автоматизирано проследяване на грешките при изпращането на съобщения
- Създаване на CloudFormation script за бърза инсталация на нужната инфраструктура
- Създаване на Command Line Interface tool за работа с услугата

10 Какво научих

Добих полезен опит с езика Python, който беше нов за мен. След доста прекарано време в четене на AWS документация и експерименти с услугите, вече имам основна обща представа за това как се работи със средствата на AWS, за Какво бих могла да ги ползвам и колко лесно и интуитивно е да се интегрират в едно приложение. Осъзнавам, че предимствата, които предлагат от страна на капацитет, сигурност и динамична скалируемост са много трудни (ако не и невъзможни) за постигане от един обикновен екип, работещ по някакъв проект и нуждаещ се от тези свойства като основа, върху която да пишат и специфичната бизнес логика. Затова ако приложението, върху което работя, се нуждае от

такива големи ресурси и възможности, не бих се поколебала да се възползвам от AWS.

11 Списък с фигури и таблици

Списък на таблиците

Списък на фигурите

12 Използвани източници

[1] Python 3 AWS SDK Documentation [2] Official AWS Documentation