

Cloud Platform for Automated User Provisioning

Brett Biscoll

Johns Hopkins University

Cloud Computing in the Public Sector

Professor Janis Butkevics

December 15, 2023

Problem

While the process of managing colleague access to company software may begin with as little as a simple policy at the startup stage, the consequences for failing to adjust once the company scales up are significant. For a freelancer working on their own, it is easily handled by the fact that one person can access and do everything; for small firms of less than ten people, processes may still be simple enough that everyone who has access does so already. Any further than this, however, the system quickly becomes complicated. Many small businesses operate on what is known as *discretionary access control* - that is, a single admin has control over who accesses the program, and who can grant and revoke privileges on an *ad hoc* basis. However, there are many issues with this system: organizing who should be able to access which software for which role at a firm of even only 100 people gets extraordinarily complex, especially factoring in when colleagues begin joining and leaving the firm.

There are several further issues with the discretionary access control model, the most fundamental of which is the question of who should be the admin in control of access to the software systems. Having a single admin for a large company can increase the risk if that account were to be hacked, while having several admins leads to higher odds of a break as more accounts and individuals must be monitored. Additional issues crop up when one factors in the removal of users from programs, also known as deprovisioning: in particular, whether users are being properly removed from access once they have left the firm. If they are not, a security risk exists for these non-firm users - with a worst-case scenario of a disgruntled employee causing havoc, or one in which the employee has access to sensitive trade secrets due to having an 'in' to critical systems. Furthermore, the system is simply not scalable beyond a certain number of people - if a

large development company has 1000 people, the process of adding/removing users from various programs quickly becomes a massive time sink for an individual user.

Some managers might prefer to keep the *discretionary access control* system, but apply significant management approval processes around it to limit the risk. The more process that is added for a user to gain access to (for example) a development environment equals lost time and work-hours that could have been spent doing something more productive. In an environment with users changing roles and projects frequently, this can come with a significant cost.

An additional way to approach these issues is to identify a group of people who each control access to a different software system, and disperse the administrative duties among multiple individuals; however, this comes with its own challenges. If various systems are controlled by different admins, it becomes more difficult to implement a standardized policy company-wide. A situation could arise where the administrator applies different levels of scrutiny for different requests; for example, if one works with one user on a personal basis, the admin may therefore be more comfortable addressing their requests sooner than another who may be remote or work in a different office. The resulting unevenness could result in frustration on the part of colleagues on the slower end of receiving permissions, but more concerning from a security perspective is the potential for uneven or lax application of security rules at all. If certain admins are going around policy for select users, the benefits of additional process around permissions can be lost with the issues around discretionary access control remaining in place.

What the above means is that a discretionary access control system may not necessarily be suitable for any company larger than a handful of people; therefore, an alternative system is both desirable and worth developing.

Requirements

The requirements for the system should be built primarily around the main roles; that of the HR administrator, that of the system administrator, and that of the end user (aka colleague).

The HR administrator will initiate the regular process. From their end will be a centralized location where user permissions are handled, accessible anywhere in the company. The HR administrator is expected to understand the business situation of when users need to be added and to know what roles they will need to be added to. It is unrealistic to expect the HR users to know any type of advanced code or SQL script writing for doing so, as it is not a commonly requested skill in that field; therefore, an intuitive user portal for this part of the process will be necessary. From the perspective of an HR admin accessing the system through the front end, I want to be able to add a new user, remove an old user, or update or otherwise change credentials of existing users. Additionally, the changes made from the HR admin should process through to the software systems on a regular basis; some delay is acceptable when not urgent, but otherwise user provisioning updates should be applied within 24 hours of the HR admin submitting the change through the portal. In addition to this, there will be situations where the HR admin must make the change off of regular business hours for security reasons; as an example, when a user with sensitive permissions exits the company, they must be deprovisioned immediately from critical accounts. Therefore, the system will require 24-hour accessibility, as well as an option to run the job immediately for urgent changes.

The second role to be accounted for is that of the system administrator. They should be able to update roles themselves (as in change their names or frequency), and make other changes outside of the usual HR process, such as adding new offices, departments and positions. Therefore, they should be able to have editing privileges to the provisioning tables. As system

administrators, they can be required to access the data via a SQL query or other command line; but their access must be secure as well due to the sensitivity of the system.

Lastly to be considered here is the end user being provisioned (referred to hereafter as the colleague). From their perspective, they should have the credentials set up for every necessary software for their role within 24 hours of beginning or being notified of a change; that means they should have access updated within 24 hours of joining, changing roles or submitting their resignation. Ideally, a notification should make users aware of any recent additions, changes, or removals (except for blacklistings) to the program.

Proposed Solution

My proposed solution will leverage multiple features of AWS Cloud, in particular AWS Lambda functions, a S3-supported SQL database, as well as bucketing and SNS notifications, to fulfill these requirements.

The flow of data through the solution will begin with a web app portal, accessible by the HR colleagues; this will offer a user-friendly way to add new colleagues or change their roles with a simple form. When a new colleague joins, the HR admin will enter a new page and be presented with a user interface to enter in all necessary information (see Appendix 2 for a minimalist mockup of what this could look like); enter in key information through a combination of freeform boxes for the names and dropdowns for the relevant department, geolocation and role. Furthermore, for a newly created user, a unique dummy identification key will be assigned automatically from amongst existing and unused identification numbers.

For permissions updates and removals, the HR administrator will be able to use the portal to search an existing name, ID or other fields (Appendix 3) and enter to edit their geolocation,

department, or role; or if necessary, apply a blacklist and remove the colleague from all functions.

After the HR administrator saves their changes, the information will trigger the first lambda job, the Database Updater. Lambda jobs are a good fit for most of the individual transformation tasks as they are lighter (serverless) and can minimize costs by only running when needed. This converts whatever change they made into a SQL query which is run against the SQL 'User' database, using RDS. This database will contain user information for all colleagues. For example, when a new user is added, an insert query will run to add a row to the database with the new information. For other cases whereby a user has their permissions updated or removed, the query will update their records accordingly (A set of dummy roles, 'INACTIVE' or 'BLACKLISTED', will be applied to records of users who depart or are removed from, to ensure records are kept for historical purposes). This will ensure that the SQL database is up to date with all changes.

Before moving to the next step, I wanted to mention the purpose of the administrator user flow. Administrators will have access to an admin portal, in order to make bulk updates; system administrators will also have access to the SQL database, but as part of the logic behind this system is security, we will attempt to minimize direct updating of the database. From here, administrators can add roles, titles, offices, teams, and make other changes outside the scope of the day-to-day process followed by the HR administrators. From the point of the change, another lambda job is activated to trigger a similar process for creating a SQL query that will run against the database.

The User Databases will consist of one table containing the user identifier number, their first and last name, the region they are assigned to (this should be understood as the office they

work out of), their specific team in the company, and that user's individual title. This will cover all colleagues who are in, or have been part of, the company. After this step, the 'Permissions File Query Runner' will be run on this database. This will be another lambda job off of which the various permissions files will be generated. For each given permissions file, the lambda will run a query to pull each user whose role corresponds to a given software, which will change for each service. For example, access to a project management software (such as JIRA) may be necessary for business analysts, quality assurance analysts and software developers, while for the development environment, only software developers might need access. Accordingly, the job will pull all colleagues that fit the appropriate user base for the software, and create an access provisioning file consisting solely of these users. One file will be generated for each software system to be provisioned. These files will be stored in an S3 bucket - the permissions file holder.

The reasoning for the bucket is to be a nestable source of past provisioning files. The intention will be to keep at least one year's backlog of files to trace back incidents or issues with the provisioning process; if a user is incorrectly provisioned, we can go back to the time of the incident and find the issue in the file. From the files in this bucket we will generate both the auto-provisioning files for the individual softwares, and send an SNS notification to colleagues.

The 'Individual Provisioning List Generator(s)' will convert the permissions files from the S3 (containing the colleagues raw user data) into whatever format is ingested by the platform; as different softwares will have different systems of giving access, this step will adjust that data into whatever format is accepted by the respective software; furthermore, some additional development may be required for the 'pull' aspect of these software provisioning rules, which I include in this step. The file will send any and all needed provisioning data to the software platform, which will automatically create, change and delete user profiles accordingly

to match this data based on this file, thus allowing the user to be provisioned to all necessary softwares with no additional human intervention after the HR administrator submitted the form on the web app.

Furthermore, from here, a separate check will be run to identify names that are being changed in the files. All users who are being provisioned into a file for the first time shall receive an SNS notification. This is to ensure that a user is aware when they are able to access a necessary piece of software. The SNS will use the colleague's email as the addresses, and generate individual messages for each user and piece of software combination.

Budget

For the budget for this project, I used the following estimates to cover the services included in the diagram: three EC2 instances, one to keep the admin portal running and two to keep the user portal running on a continuous basis; with a 3 year reservation on the assumption the company would be using the system for at least that long. The costs for the single S3 and the four Lambda services were negligible; higher was the RDS for PostgreSQL service, however this was necessary to have an updatable SQL database. The SNS system was set up to handle 1 million requests, HTTP/SQS notifications, and Email notifications per month; unless the firm scales up significantly or includes a tremendously large number of updates, this limit will not be exceeded or even approached (and even then additional SNS Services will not be a massive increase). Lastly, I included an instance of AWS App Runner to cover handling the web application. The 12-month total cost of this system ended at \$7780.92.

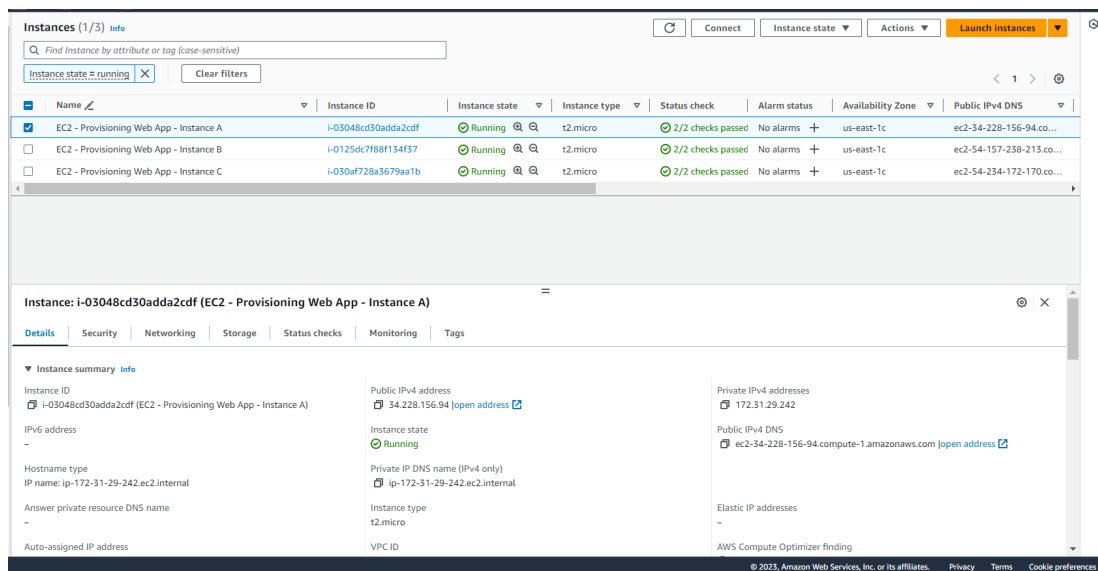
While this would appear substantial to upper management, I would refer to the extremely high cost of hacking cases, for example the Equifax Breach, which was partially due to the lax

security of using the credentials ‘admin’ and ‘password’, ultimately costing over \$300 million dollars in litigation and fines. The above system is a model of how to handle this process in a way to minimize human intervention points and avoid the complication with security and paperwork it would take to get everyone access to whatever they needed to work on. In doing this, it enables a more secure system and decreases the likelihood of these catastrophic security breaches.

Implementation

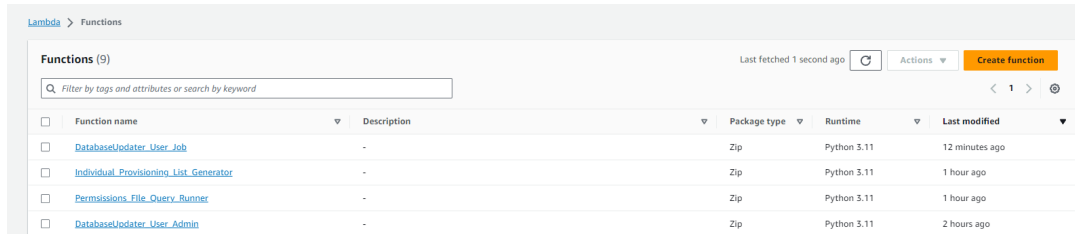
Below are screenshots of the setup versions of each system described in the solution proposal.

- *EC2 Instances.* These instances are used to keep the Human Resources system running on a continual basis, and to keep the admin portal running as well.



- *Lambda.* Lambda jobs in my implementation connect data to the next step. The updaters will be triggered by the portals and have a destination in the RDS database; the query runner is triggered by a daily cycle to pull from the database and deposit into the bucket,

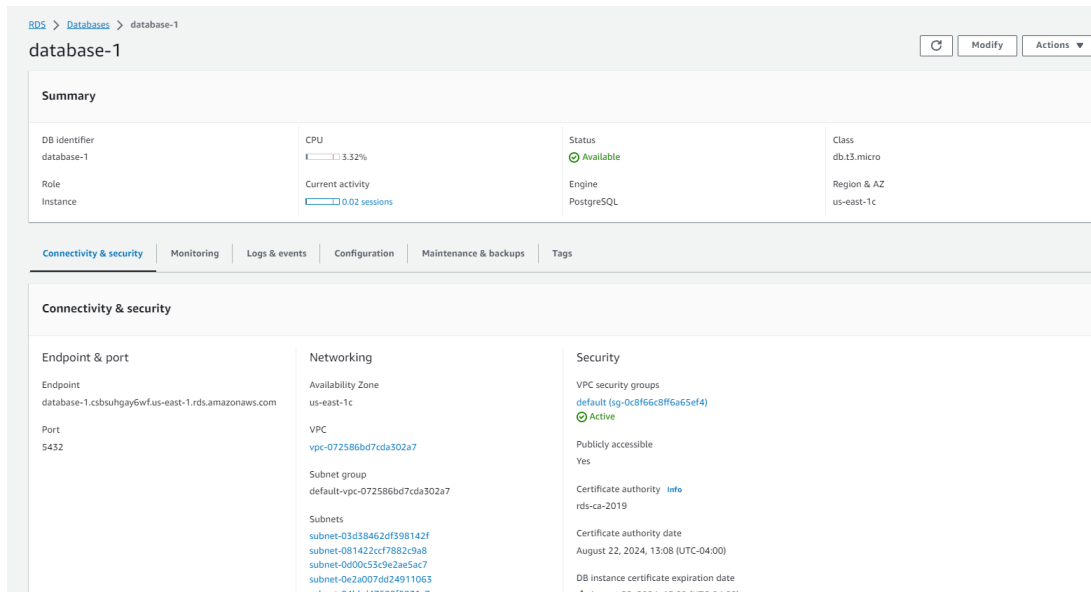
and the provisioning list generator is triggered by the appearance of a file on in the bucket (taking place immediately after the daily job completes).



The screenshot shows the AWS Lambda console 'Functions' page. It lists five functions: 'DatabaseUpdater_User_Job', 'Individual_Provisioning_List_Generator', 'Permissions_File_Query_Runner', and 'DatabaseUpdater_User_Admin'. Each function is a Zip package type using Python 3.11 runtime. The 'Last modified' times are 12 minutes ago, 1 hour ago, 1 hour ago, and 2 hours ago respectively.

Function name	Description	Package type	Runtime	Last modified
DatabaseUpdater_User_Job	-	Zip	Python 3.11	12 minutes ago
Individual_Provisioning_List_Generator	-	Zip	Python 3.11	1 hour ago
Permissions_File_Query_Runner	-	Zip	Python 3.11	1 hour ago
DatabaseUpdater_User_Admin	-	Zip	Python 3.11	2 hours ago

- *SQL RDS Database.* The RDS database will be a repository for all colleague information relevant to the provisioning process.



The screenshot shows the AWS RDS console details for an instance named 'database-1'. The instance is in the 'Available' state, running on a 'db.t3.micro' class. It is a PostgreSQL engine instance. The 'Connectivity & security' tab is selected, showing details about the endpoint, port, networking, and security groups.

Summary			
DB identifier database-1	CPU 3.32%	Status Available	Class db.t3.micro
Role Instance	Current activity 0.02 sessions	Engine PostgreSQL	Region & AZ us-east-1c

Connectivity & security		
Endpoint & port Endpoint database-1.csbiuhgawfowf.us-east-1.rds.amazonaws.com Port 5432	Networking Availability Zone us-east-1c VPC vpc-072586bd7cda302a7 Subnet group default-vpc-072586bd7cda302a7 Subnets subnet-03d38462df398142f subnet-081422ccf7882c9a8 subnet-0d00c53c9e2ae5ac7 subnet-0e2a007dd24911063 subnet-04b4d43c896f623a9	Security VPC security groups default (sg-0c8f6c8f6a65ef4) Active Publicly accessible Yes Certificate authority rds-ca-2019 Certificate authority date August 22, 2024, 13:08 (UTC-04:00) DB instance certificate expiration date 2025-03-30T13:08:00Z

- *S3.* The S3 bucket will be a temporary repository for the filtered provisioning files unique to each system. The file branch will consist of individual folders for each system, with individual files for the last 365 days stored within; the addition of a new file will trigger the Provisioning List Generator job to pull the most recent file.

Amazon S3 > Buckets

Account snapshot
Storage lens provides visibility into storage usage and activity trends. [Learn more](#)

[View Storage Lens dashboard](#)

Total storage ⌚ Pending	Object count ⌚ Pending	Average object size ⌚ Pending	You can enable advanced metrics in the "default-account-dashboards" configuration.
----------------------------	---------------------------	----------------------------------	--

General purpose buckets | Directory buckets

General purpose buckets (1) [Info](#)

Buckets are containers for data stored in S3. [Learn more](#)

[Refresh](#) [Copy ARN](#) [Empty](#) [Delete](#) [Create bucket](#)

Name	AWS Region	Access	Creation date
permissions-file-holder	US East (N. Virginia) us-east-1	Bucket and objects not public	December 12, 2023, 21:39:59 (UTC-05:00)

- *SMS Messaging Service*. This service will provide notification via email to each user. The endpoint given here is provisioningupdates@firm.com as a placeholder; the system will change the endpoint to the email of the user being provisioned.

Subscription to Provisioning_Updates created successfully.
The ARN of the subscription is `arn:aws:sns:us-east-1:555264689782:Provisioning_Updates:299e4956-93c9-4fc6-b28f-3feb51b7cd67`.

Amazon SNS > Topics > Provisioning_Updates > Subscription: 299e4956-93c9-4fc6-b28f-3feb51b7cd67

Subscription: 299e4956-93c9-4fc6-b28f-3feb51b7cd67 [Edit](#) [Delete](#)

Details

ARN <code>arn:aws:sns:us-east-1:555264689782:Provisioning_Updates:299e4956-93c9-4fc6-b28f-3feb51b7cd67</code>	Status ⌚ Pending confirmation
Endpoint <code>provisioningupdates@firm.com</code>	Protocol EMAIL
Topic Provisioning_Updates	
Subscription Principal <code>arn:aws:iam::555264689782:role/voclabs</code>	

Subscription filter policy | Redrive policy (dead-letter queue)

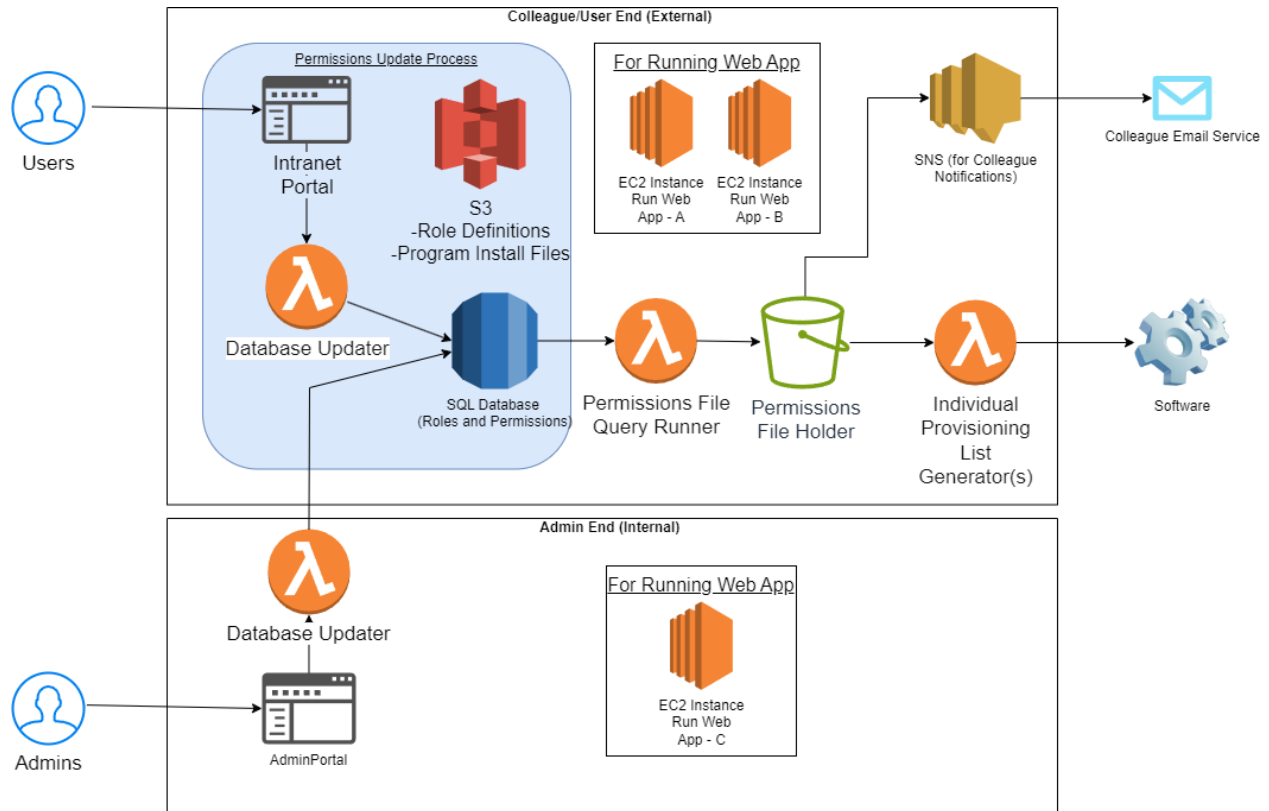
Subscription filter policy [Info](#)
This policy filters the messages that a subscriber receives.

No filter policy configured for this subscription.
To apply a filter policy, edit this subscription.

[Edit](#)

Appendix

1. Diagram



2. UI Mockup

a. Adding a New Colleague:

Automated Provisioning System - Add New Colleague

User ID:	123456 *	
First Name	John	Please enter colleague first name
Last Name	Smith	Please enter colleague last name
Region:	BAL ▼	Enter the office the colleague will be working out of here
Team:	Software ▼	Enter the team the colleague will be working on here
Title:	Developer ▼	Enter the colleague title

b. Searching Users:

Automated Provisioning System - Search Colleagues

Search Criteria:

User ID: --- *

First Name: --- Please enter colleague first name

Last Name: --- Please enter colleague first name

Region: PHI ▾ Enter the office the colleague will be working out of here

Team: --None Selected-- ▾ Enter the team the colleague will be working on here

Title: -- None Selected -- ▾ Enter the colleague title

Search Results:

User ID	First Name	Last Name	Region	Team	Title
000123	Abe	Gerald	PHI	Software	Developer
000124	Beatrice	Holden	PHI	Software	Quality Assurance
000125	Cedric	Irvin	PHI	Software	Product Manager
000126	Diane	Jacobs	PHI	Marketing	Customer Service
000127	Edward	Kline	PHI	Marketing	Marketing Manager
000128	Francine	Lollard	PHI	Payroll	Finance Manager

c. Updating Existing Users:

Automated Provisioning System - Update User

Search Criteria:

User ID: 000125 *Cannot be Edited*

First Name: Cedric Please click to select/edit colleague first name

Last Name: Irvin Please click to select/edit colleague last name

Region: PHI ▾ Please click dropdown to select office

Team: Marketing ▾ Please click dropdown to select team

Title: Product Manager ▾ Please click dropdown to select title

3. Budget Calculator

aws pricing calculator

Feedback Language: English Contact Sales Create an AWS Account

AWS Pricing Calculator > My Estimate

My Estimate edit

Export Share

Estimate summary

Upfront cost: 0.00 USD

Monthly cost: 648.41 USD

Total 12 months cost: **7,780.92 USD**
Includes upfront cost

Getting Started with AWS

Get started for free Contact Sales

My Estimate Duplicate Delete Move to Create group Add support Add service

Find resources

Service Name	Status	Upfront cost	Monthly cost	Description	Region	Config Summary
Amazon EC2	✓	0.00 USD	56.50 USD	-	US East (Boston)	Tenancy (Shared Instances), Operating system (Lin...
Amazon Simple Storage Service (S3)	✓	0.00 USD	0.05 USD	-	US East (Ohio)	S3 Standard storage (2 GB per month), S3 Standar...
AWS Lambda	✓	0.00 USD	0.00 USD	Lambda Functions	US East (Ohio)	Architecture (x86), Architecture (x86), Invoke Mode...
Amazon RDS for PostgreSQL	✓	0.00 USD	498.25 USD	RDS	US East (Ohio)	Storage volume (General Purpose SSD (gp2)), Stor...
Amazon Simple Notification Service (SNS)	✓	0.00 USD	20.71 USD	-	US East (Ohio)	Requests (1 million per month), HTTP/HTTPS Not...
AWS App Runner	✓	0.00 USD	72.90 USD	-	US East (Ohio)	Container compute size (1 vCPU), Minimum provis...

Works Cited

<https://www.cnbc.com/2017/09/14/equifax-used-admin-for-the-login-and-password-of-a-non-us-database.html>