rm rf root - Task Tracker 222

Maintenance Manual

4th December 2022

Amrin Sandhar
Michelle Chan
Brian Frey
Anish Gautam
Visoth Cheam
Noel Ian Paulite
Tobechi Maduchukwu
Marouf Zaman

Table Of Contents

1.0 Introduction

Task Tracker 222 is an application intended to be used for the purpose of scheduling events for users. As opposed to the document-style emails currently in use, the approach taken by this application allows for easy updates and edits to events, as well as a more readable format.

Events are served to recipients on the basis of whether or not the recipient is included in the participants list. This means that a recipient can be added as a participant not only individually, but also by rank, flight, workcenter, and team. The ability to add both individually and by group allows for a greater degree of flexibility than was currently available with the presently used system.

A dynamic calendar view, with an included intraday summary, allows for a quick “at a glance” visualization for the end user to quickly get a picture of how their schedule will look for any given week.

Management of both users and events was made easier with preset lists of both groups and locations that can be searched and drawn from. All functions required for management can be found on the frontend, requiring minimal technical knowledge for proper usage and operation.

2.0 Implementation Tools & Technologies

2.1 IDE, Programming languages, Libraries, Frameworks

a. IDE
    i. IntelliJ IDEA Ultimate - Note: this program is not free. Other sufficiently featured IDEs compatible with Java should work for the purpose of making changes, adjustments, and maintaining the application.
b. Programming Languages
    i. Backend
        1. Java - The backend of this application is written exclusively in Java.
    ii. Frontend
        1. Javascript - Used heavily for adaptive features on the frontend.
        2. HTML - Not a programming language, defines front-end markdown.
        3. CSS - Not a programming language, handles front-end styles.
c. Libraries
    i. Backend
        1. JUnit - Used for unit testing.
        2. AWS - Used for all features that interface with AWS services.
    ii. Frontend
        1. JQuery - Used for general frontend DOM manipulation.
        2. Chosen - Used for advanced, searchable selection fields.
        3. TimePicker - Used for time selection fields.
d. Frameworks
    i. Spring Boot - An extension of the Spring Framework, Spring Boot is a comprehensive package that allows for MVC (model, view, controller) development, including an integrated server. The entire backend is structured around the use of Spring Boot's functionality.
    ii. Maven - Although not strictly a framework, Maven is used for dependency management.

2.2 Web services

a. Cognito - Cognito is an AWS service used in order to handle user authentication. This takes the burden of security away from the application developer by providing APIs for authentication. Any management necessary that isn't possible from within the application can be done on the Cognito dashboard on AWS.

2.3 Database

a. AWS RDS PostgreSQL - Deployed and managed through AWS, the database used is a PostgreSQL instance in RDS.

2.4 APIs

a. No extra web services outside of the two listed in the previous two sections were used.

3.0 Runbook

3.1 Server goes down

a. Restart the service - The first thing to try is a simple restart of the service through the service on which it is hosted. If this does not succeed, proceed to the next option.
b. Check the console - If the primary service has ceased to work, check the console for errors. Spring prints detailed errors to the console, as well as stack traces in the event an error has occurred. Identify the error condition and resolve it (for example, hypothetically removing a malformed input from the database, even though this has never caused a crash in our testing). Once the error condition has been cleared, restart the service.

3.2 Database becomes corrupt

a. For preventative purposes, we suggest setting up regular, automated backups through Amazon's RDS control panel. From this control panel, the database may be reverted to a previous backup in the event they become corrupt.
b. For further reading, we suggest referencing Amazon's guide on **Backing up and restoring an Amazon RDS DB instance**. *We feel it would be a disservice to describe the steps in this document, as these interfaces and the relevant information are subject to regular change, and thus, problems relating to them should be resolved by referencing the official documentation that is kept up to date.*

3.3 Third Party services go down

a. Database
    i. If the database service (RDS) is not working, verify that your environment variables are correct. For further reading, please reference section **4.1d**.
    ii. For further reading, we suggest referencing the official **Amazon Relational Database Service (Amazon RDS) User Guide**.
b. Cognito
    i. If the authentication service (Cognito) is not working, verify that your environment variables are correct. For further reading, please reference section **4.1d**.
    ii. Cognito can be managed through the dashboard on AWS. In this control panel, error statuses can be found and used to resolve the issue causing the outage.
    iii. For further reading, we suggest referencing Amazon's guide on **Troubleshooting Amazon Cognito identity and access** and **Managing error responses** for Cognito. Just as with the RDS documentation, we feel it would be a disservice to detail the steps here, as they are subject to change.

3.4 How to set up development environment

a. An original copy of the source code will be provided to the organization. It is suggested that this copy is committed and pushed to GitHub under the organization's collection of repositories. The rest of this section is under the assumption this step has been completed.
b. Clone a copy of the repository to your local machine. This can be done with the command **git clone <insert repository link here>** using the git bash command line.
c. Open the source directory of the project in your IDE of choice. Jetbrains IntelliJ IDEA is recommended.
d. This project is very lean on outside dependencies. As such, the aforementioned steps should be sufficient to begin making changes to the code base.

3.5 Make changes to code base

a. The use of git is encouraged when making changes to the code base. The typical hierarchy of git branches used in this project were **main ← dev ← <all feature branches>**. It is recommended to first ensure that **dev** is up to date with **master**, then branch off of **dev** using the command **git checkout -b <branch name here>**.
b. To further elaborate upon hierarchy, **master** should be used for a stable release of the product, whereas **dev** should be the experimental branch that contains recent changes.
c. If an uncommitted change needs to be rolled back, use the command **git reset <path to file>**.
d. If it is desired to revert to a previous commit, ensure that the working tree is clean with the command **git status**. If the tree is clean, use the command **git revert <desired commit hash>**.
e. To commit changes, first add the files that are desired to commit by using either **git add <path to file>,** if adding individual files, or **git add -A**, if adding all files. Commit with **git commit -m "<commit message here>"**, using a descriptive commit message. Push changes to GitHub if desired with the command **git push origin <branch name>**.
f. The ideal way to merge changes back to the primary branches is through the use of **pull requests** on GitHub. Before doing this process, ensure that the desired branch is pushed to the remote repository. When ready, navigate to the repository on GitHub and proceed to the next step.
g. Navigate to the **branches** menu and click the **New Pull Request** button. Set your **base** branch as the branch that will receive the changes (usually **main** or **dev**), then set your **compare** branch to the branch that holds the new features. Write a comment if desired, then click **Create pull request.**

h. Navigate to the **Pull requests** tab. Select the desired pull request. Resolve any conflicts, if present. Once conflicts are resolved (if any), **merge** your changes. With this step, merging is complete!

4.0 Deployment

4.1 Host Machine

a. Note - Sections **4.2** and **4.3** should be completed before following the instructions in **4.1**.
b. The client will be provided with a final version of the single-source executable in the form of a **JAR file**. In the event the client has made changes to the codebase and needs to build a new executable, navigate to the source directory of the project and use the command **mvn clean install**. Once the build is complete, navigate to the **/target** directory of the project and retrieve your newly-minted jar file.
c. The aforementioned jar file should be placed in the directory from which its execution will occur. Ensure at least **Java 11** is present on the system hosting the application. Execute the jar file with the command **java -jar <name of jar file>**.
d. Upon the first execution of the application, a **/config** directory will be made within the same directory as the jar file. Within this directory, there are five files detailing the different ranks, flights, workcenters, teams, and locations, as specified by the client at the time of development. If desired, any of these files may be modified as the user sees fit - they are merely line-separated text files that serve as unordered lists.
e. For the application to work properly, the following **environment variables** must be set:
    i. **AWS_REGION** : The AWS region used by your services
        1. This will likely take the value of **us-west-1**
        2. See section **4.3.1e**
    ii. **USER_POOL_ID** : The ID of the user pool as defined by your instance of Cognito.
        1. See section **4.3.1e**
    iii. **ACCESS_KEY** : The access key associated with your instance of Cognito.
        1. See section **4.3.1g**
    iv. **SECRET_KEY** : The secret key associated with your instance of Cognito.
        1. See section **4.3.1h**
    v. **CLIENT_ID** : The client ID associated with your instance of Cognito.
        1. See section **4.3.1f**
    vi. **CLIENT_SECRET** : The client secret associated with your instance of Cognito.
        1. See section **4.3.1f**
    vii. **COGNITO_CLIENT_NAME** : The client name for your user pool in Cognito.
        1. See section **4.3.1f**
    viii. **COGNITO_ISSUER_URI** : The issuer uri associated with your instance of Cognito.
        1. See section **4.3.1i**
    ix. **DATABASE_URL** : The url of the database as defined by your RDS instance.
        1. See section **4.2.4a**

x. **DATABASE_USERNAME** : The database username in RDS.
    1. See section **4.2.4b**

xi. **DATABASE_PASSWORD** : The database password in RDS.
    1. See section **4.2.4c**

xii. **SUBDOMAIN** : The subdomain of your services.
    1. See section **4.2.4j**

4.2 Amazon Web Services - RDS

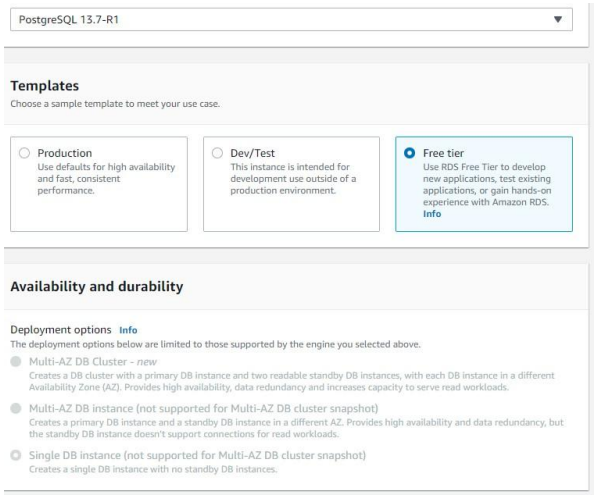
4.2.1 Table Creation

a. See section 6.1 in this document for scripts used to create each table (Drill and User)

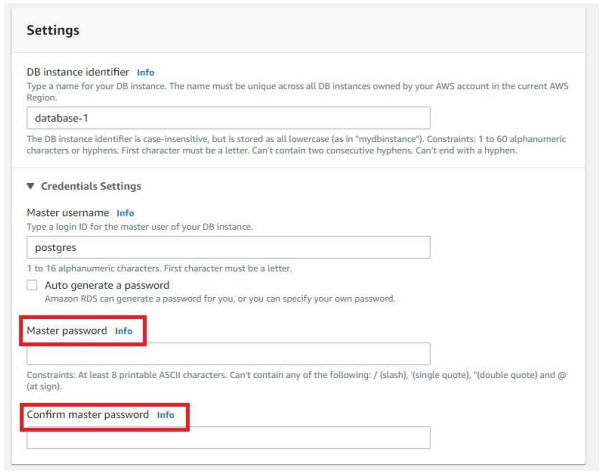
4.2.2 Set up an RDS instance:

a. Click **Create Database**

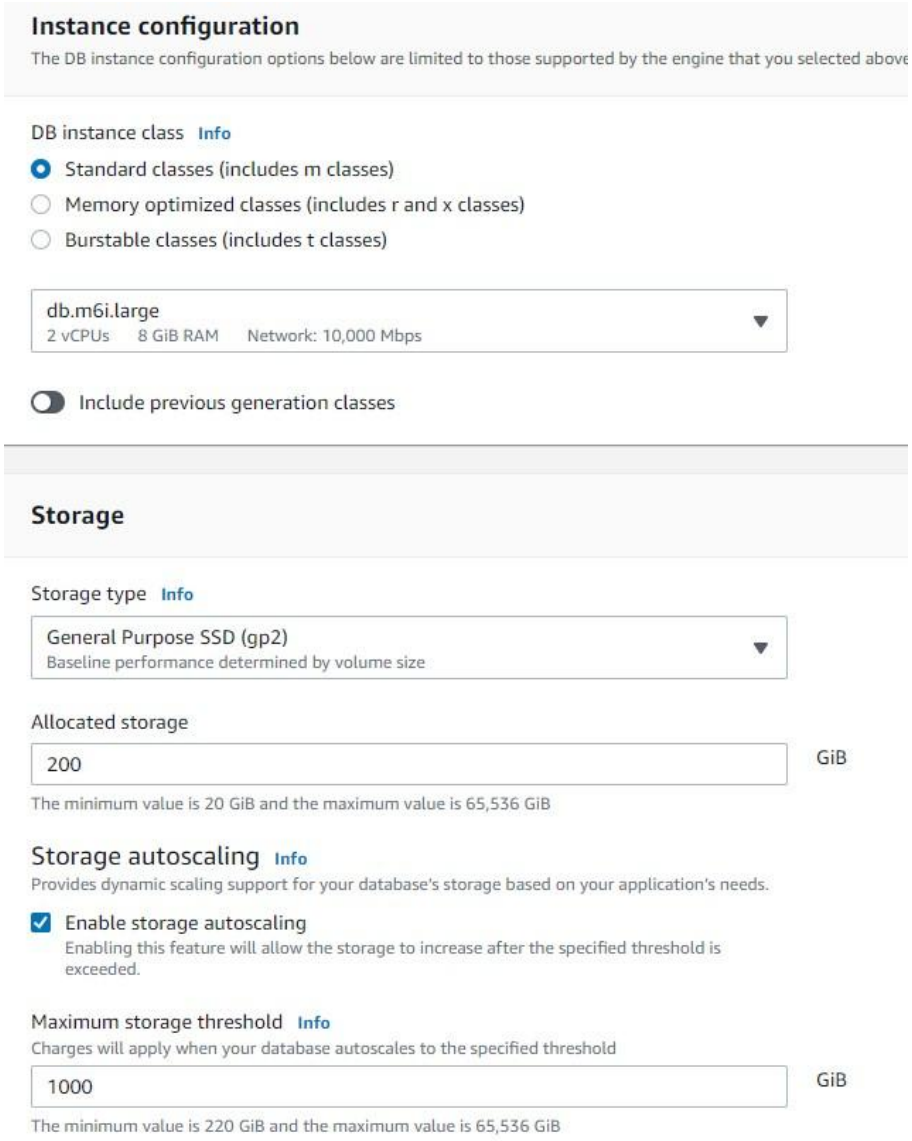
b. Click **Free Tier**

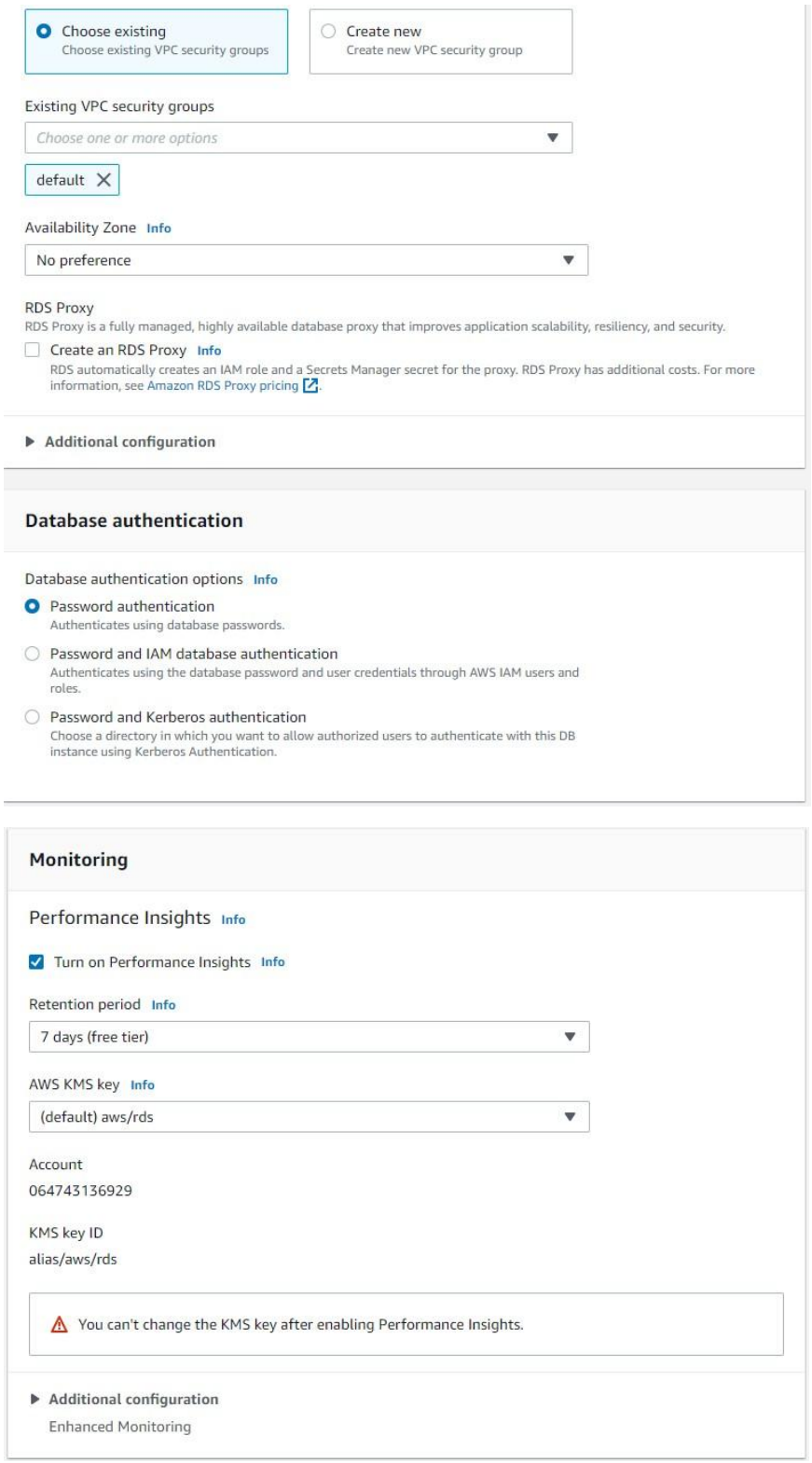


c. Enter the **master password** for the database instance

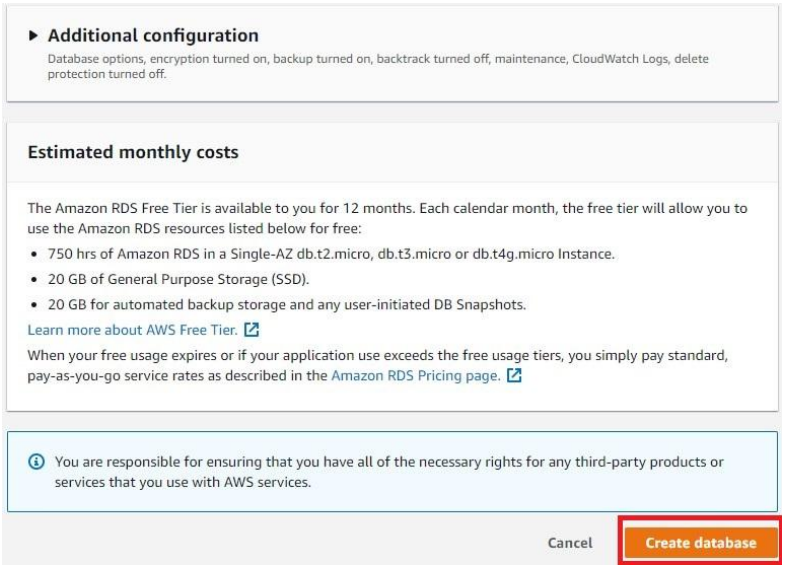


d. **Fill in your information** according to the following images:




Choose existing
Choose existing VPC security groups
Create new
Create new VPC security group
Existing VPC security groups
Choose one or more options
default
Availability Zone Info
No preference
RDS Proxy
RDS Proxy is a fully managed, highly available database proxy that improves application scalability, resiliency, and security.
Create an RDS Proxy Info
RDS automatically creates an IAM role and a Secrets Manager secret for the proxy. RDS Proxy has additional costs. For more information, see Amazon RDS Proxy pricing.
Additional configuration
Database authentication
Database authentication options Info
Password authentication
Authenticates using database passwords.
Password and IAM database authentication
Authenticates using the database password and user credentials through AWS IAM users and roles.
Password and Kerberos authentication
Choose a directory in which you want to allow authorized users to authenticate with this DB instance using Kerberos Authentication.
Monitoring
Performance Insights Info
Turn on Performance Insights Info
Retention period Info
7 days (free tier)
AWS KMS key Info
(default) aws/rds
Account
064743136929
KMS key ID
alias/aws/rds
You can't change the KMS key after enabling Performance Insights.
Additional configuration
Enhanced Monitoring

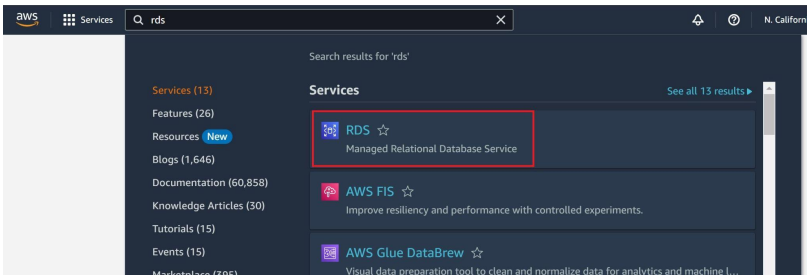
e. Click **Create Database**



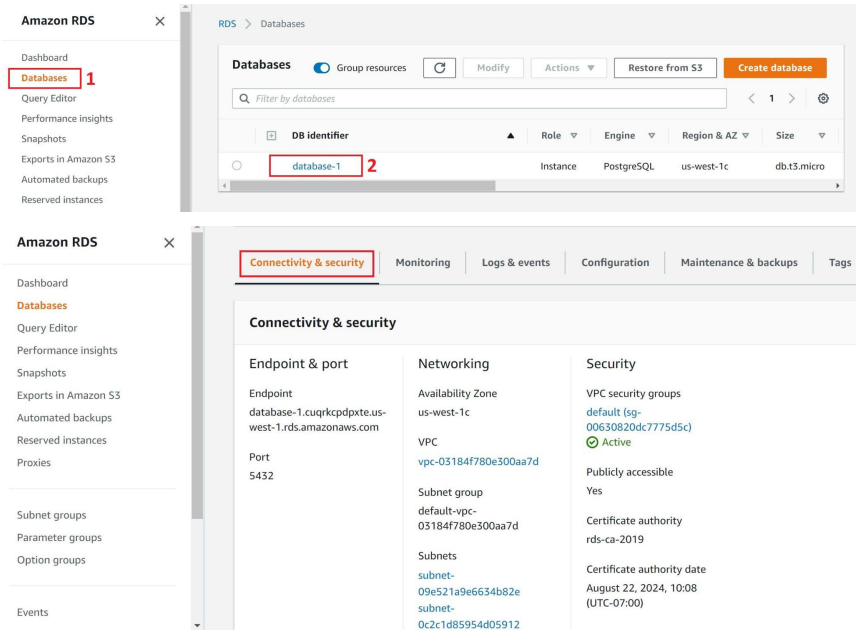
f. Database is **created**.

4.2.2 View database instance information:

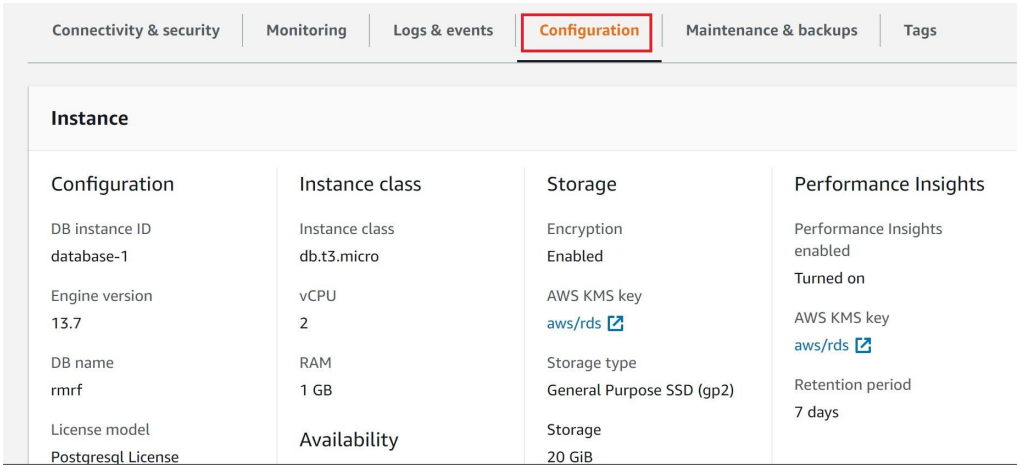
a. Log into the AWS console, click **search**, and search **RDS**



b. To view the database instance information, first click **database**, then click **database-1**

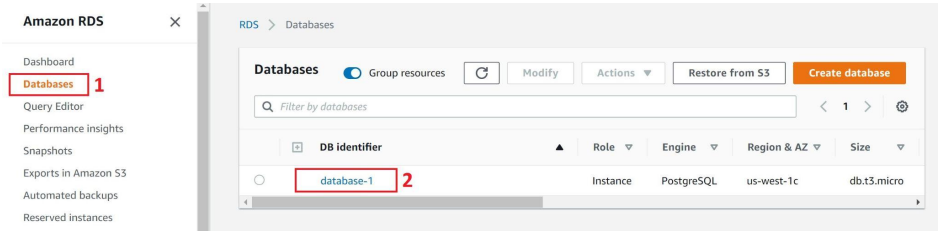


c. To view the database name, click **Configuration**

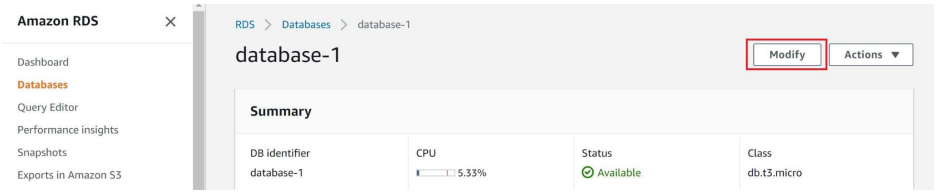


4.2.3 Change the database password:

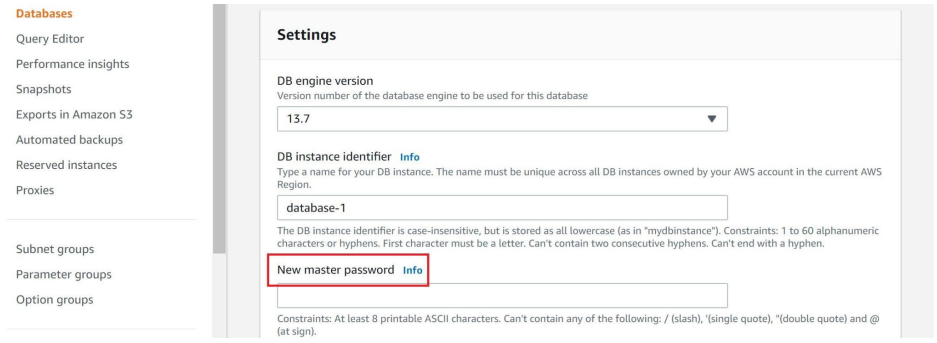
a. Click **database**, then click **database-1**



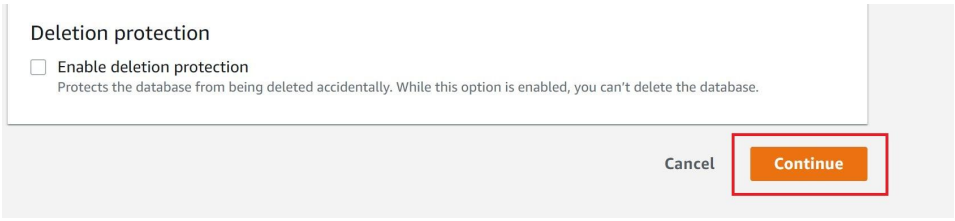
b. Click **modify**



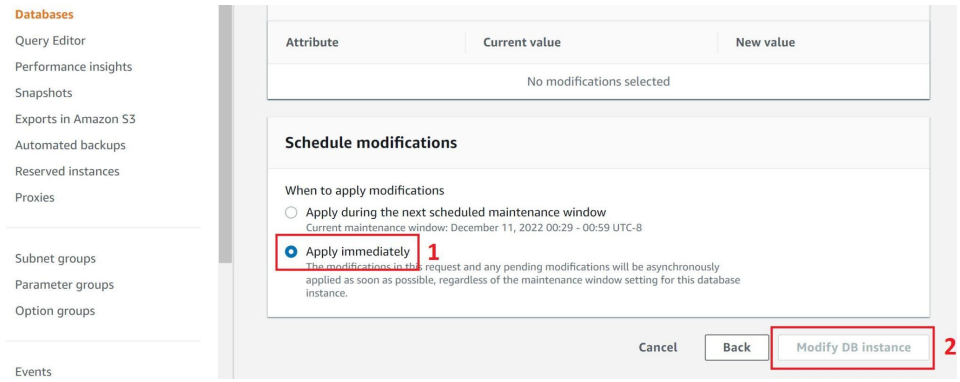
c. Fill **new master password**



d. Scroll to the bottom and click **continue**



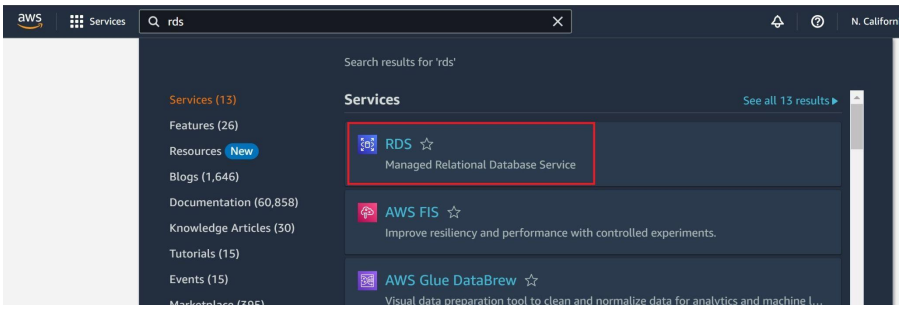
e. First, click **Apply immediately**, then click **Modify DB instance**



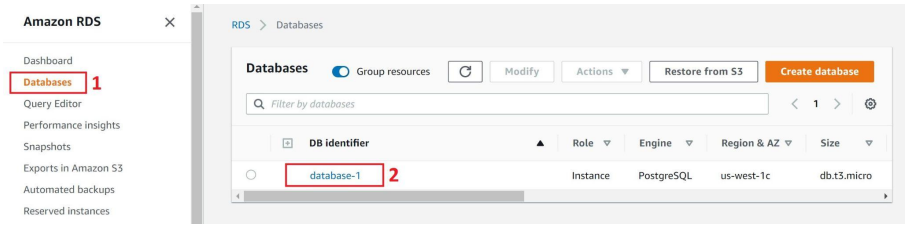
f. Password change is complete

4.2.4 Retrieve Cognito Credentials for Environment Variables

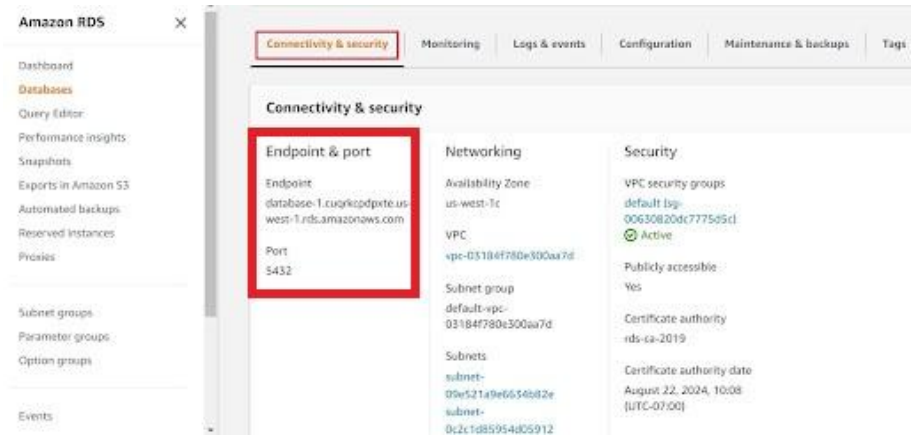
a. For **DATABASE_URL**:
    i. To find **endpoint** and **port number**
        1. Log into the AWS console, click **search**, search for **RDS**, then click **RDS**



        2. Click **database**, then click **database-1**

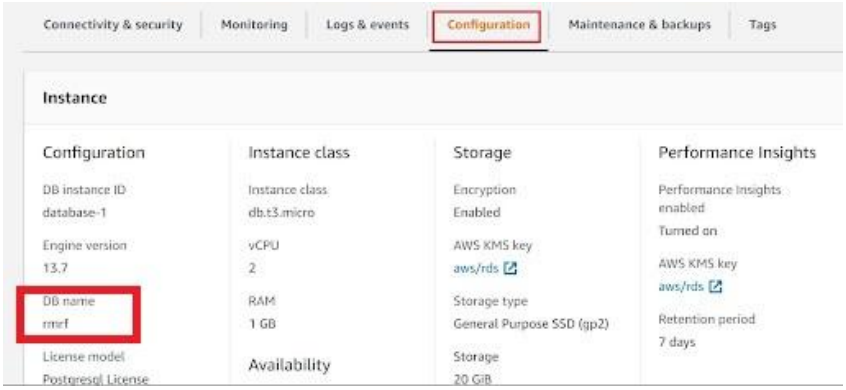


        3. Scroll down, click **Connectivity & Security** and look for **Endpoint & Port Number**



ii. To find **DB name**:

1. On the same navigator bar as the previous step, click **Configuration** and look for **DB name**



iii. Set **DATABASE_URL** equal to
postgresql://{Endpoint}:{portNumber}/{DB name}
with the appropriate fields replaced

b. For **DATABASE_USERNAME**:

i. The database username is the **DB name** found in the previous step (**4.2.4a.ii**)

c. For **DATABASE_PASSWORD**:

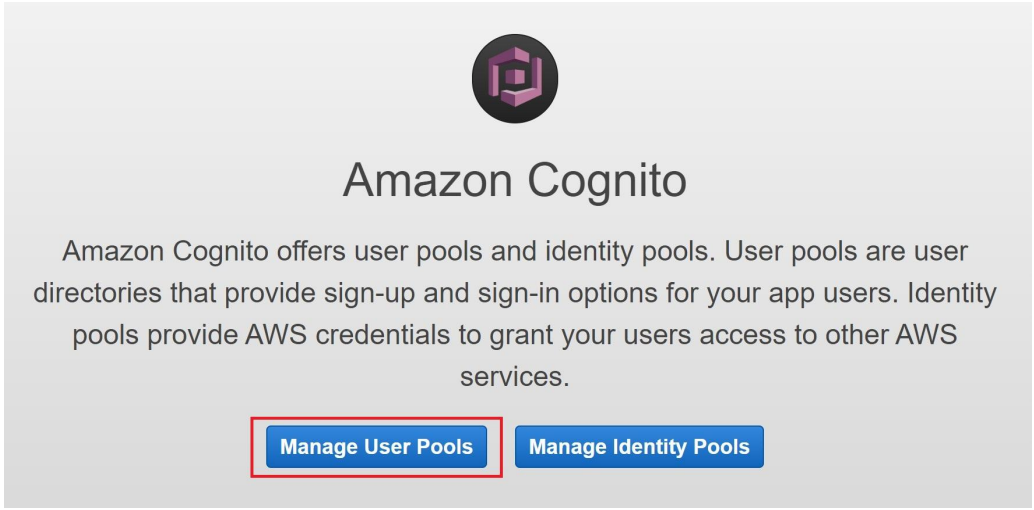
i. The database password is the **master password** that was set when creating the database. If this master password was not recorded, change the database password and record it (see **4.2.3**).

4.3 Amazon Web Services - Cognito

4.3.1 Set up AWS Cognito

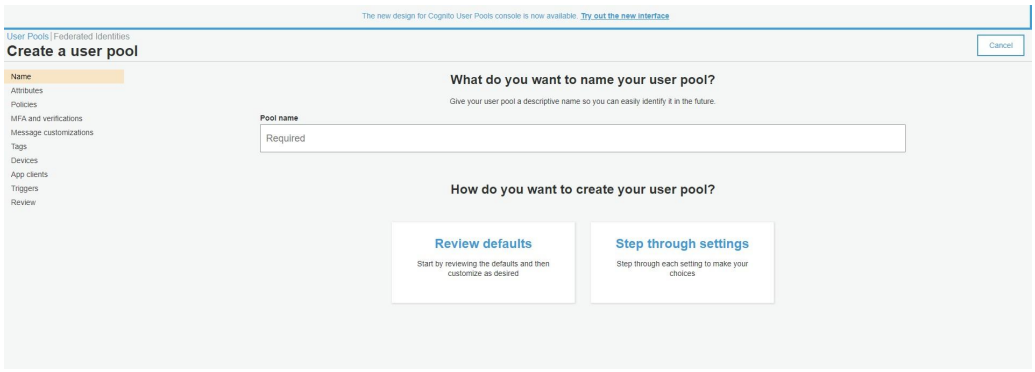
a. Log into the **AWS Console**
b. Click **search**, search for **Cognito**, then click on **Cognito (**inside where the red rectangle shows in the following figure)

c. The Cognito console will be shown. Click **Manage User Pools**

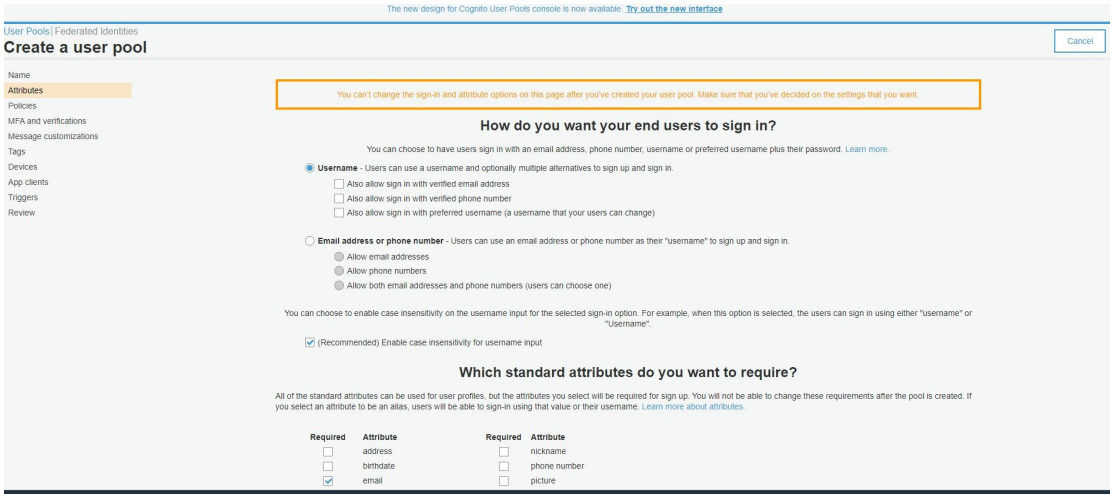


d. Click **Create User Pool**

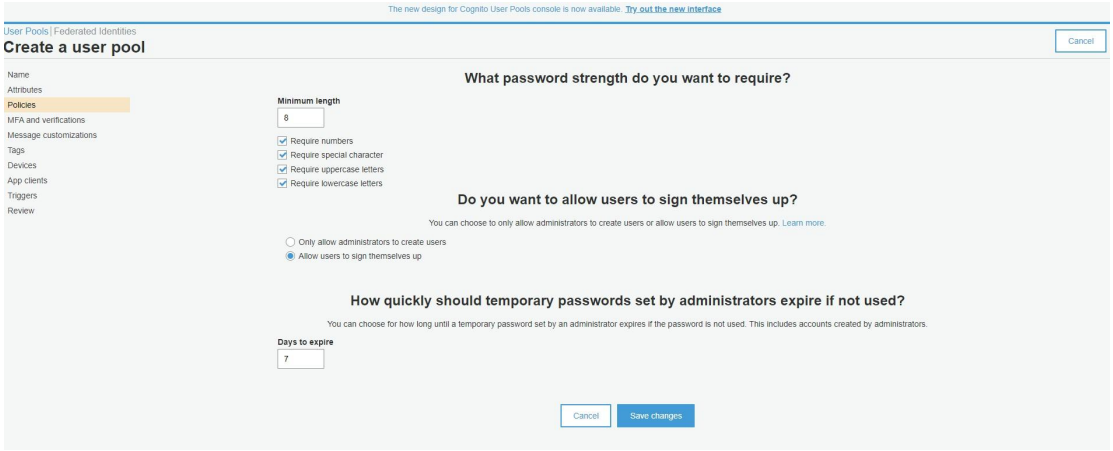
e. Click **Name**, then enter the pool name



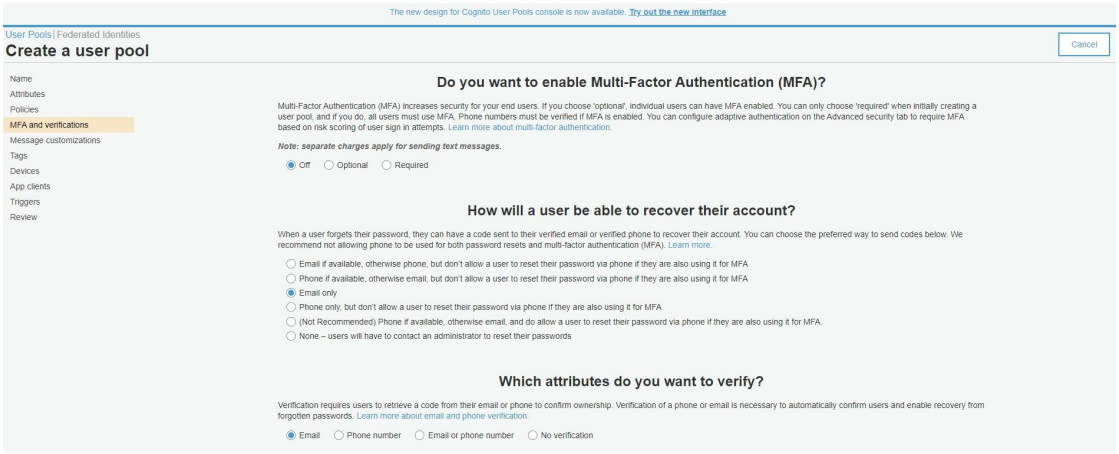
f. Click **Attributes**, then fill the information as below, or adjust it depending on your needs

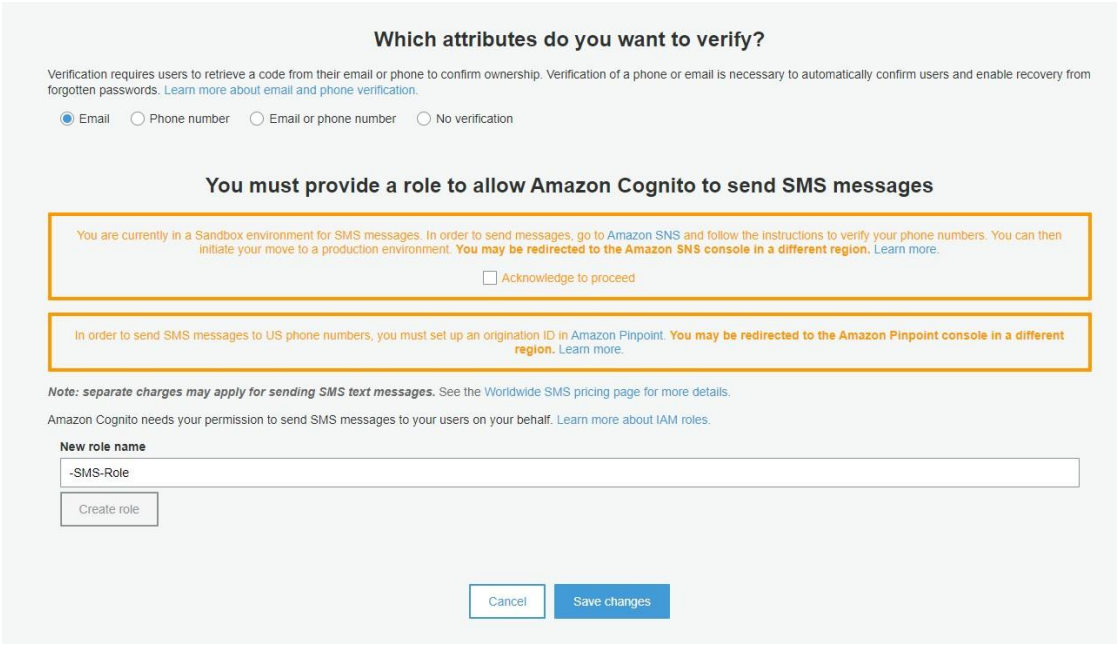


g. Click **Policies**, then fill the information as below, or adjust it depending on your needs

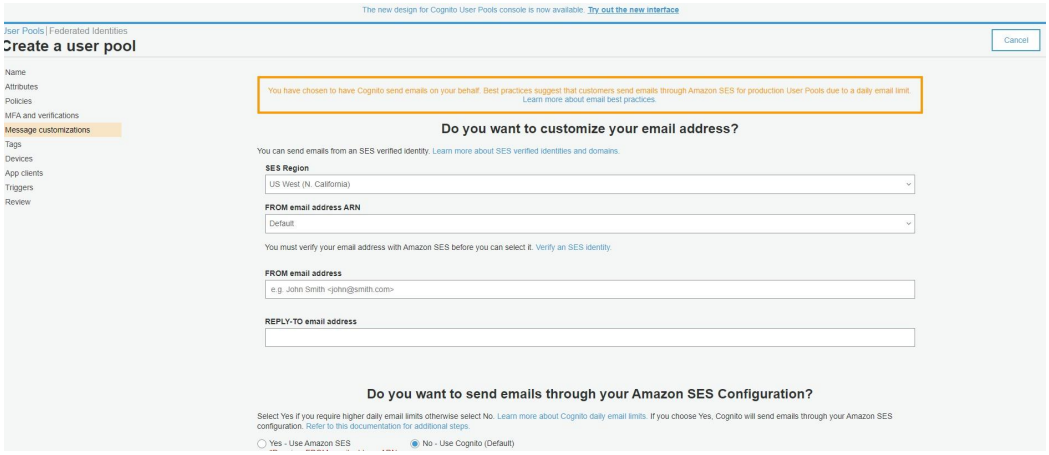


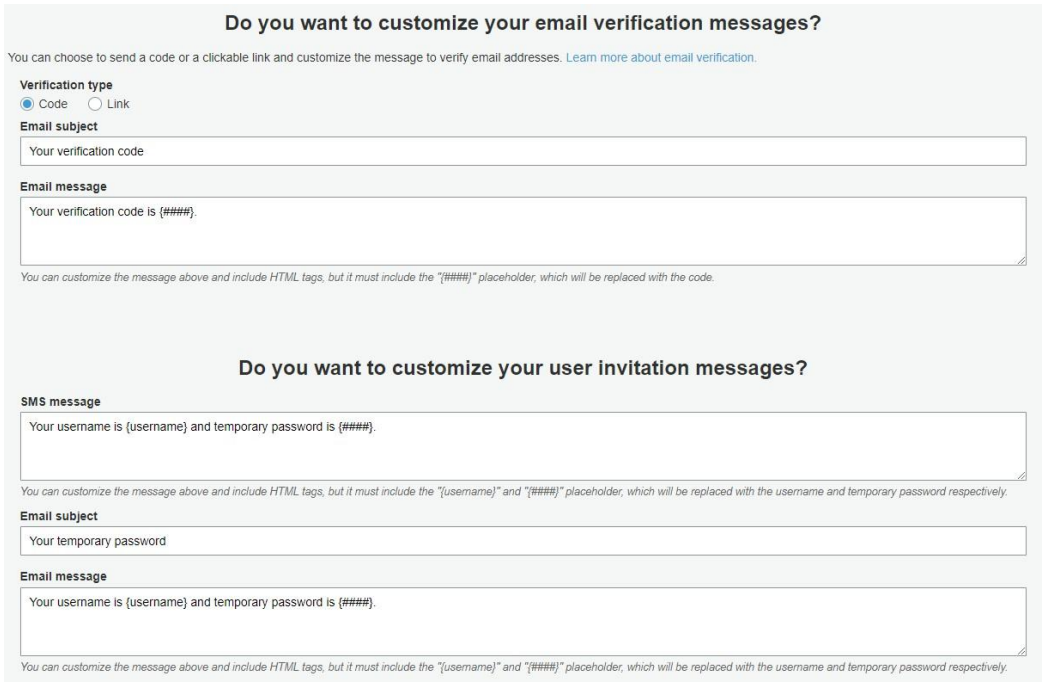
h. Click **MFA and verification**, then fill the information as below, or adjust it depending on your needs



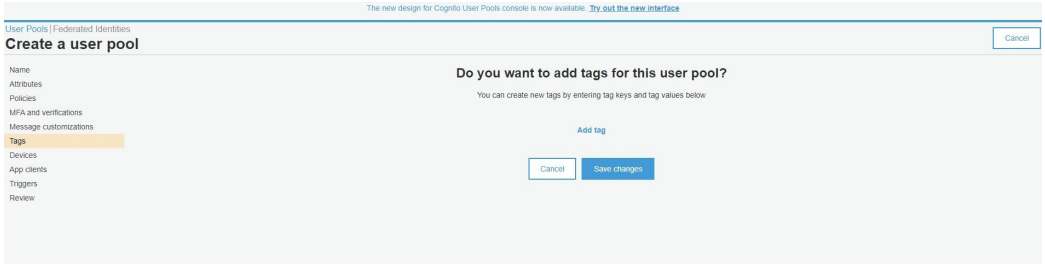


i. Click **Message customization**. You can leave it default, or adjust it depending on your needs

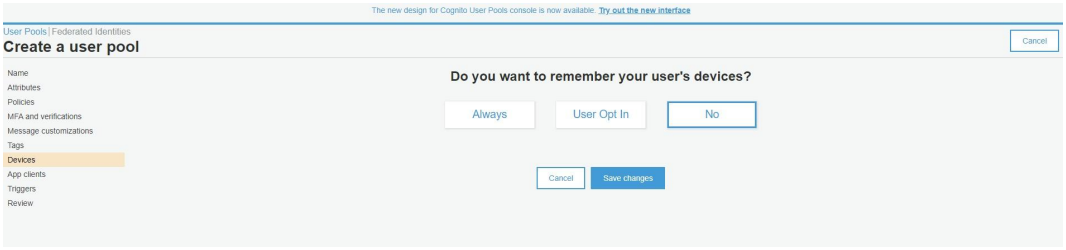




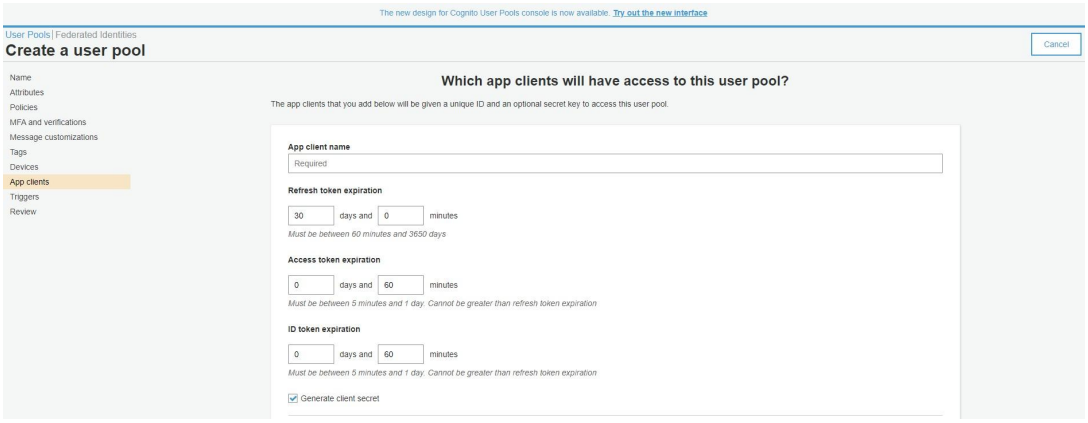
j. Click **Tags**. You can leave it as default, or adjust it depending on your needs

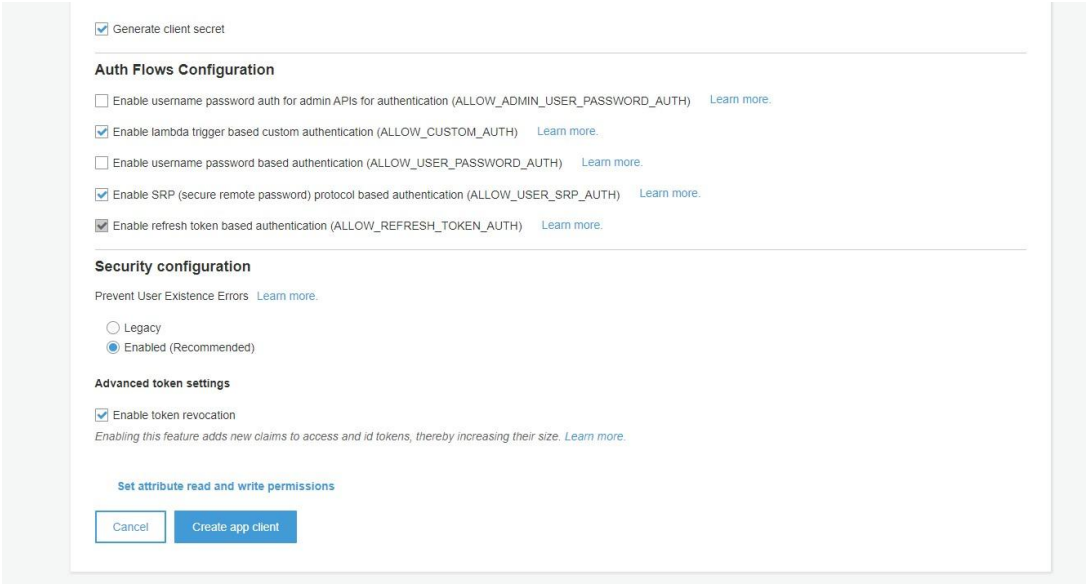


k. Click **Devices**. You can leave it as default, or adjust it depending on your needs

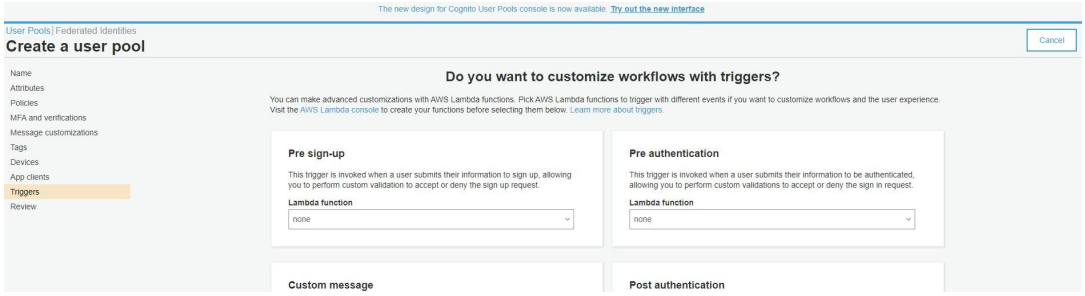


l. Click **App Clients** and enter **App client name**. It is important to remember this step, as the application configuration needs it. You can leave it default, or adjust it depending on your needs.

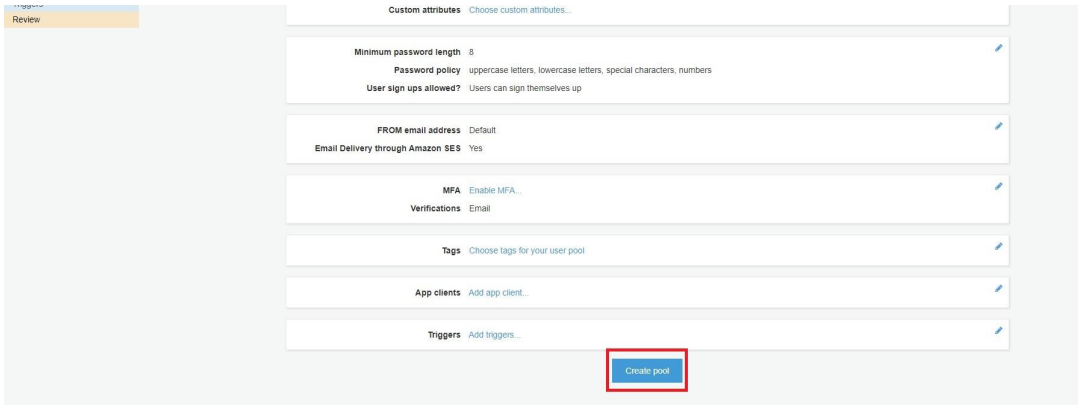




m. Click **Trigger**. You can leave it as default, or adjust it depending on what you need. It is required if you are using the AWS Lambda server



n. Click **Review**, then click **Create Pool**



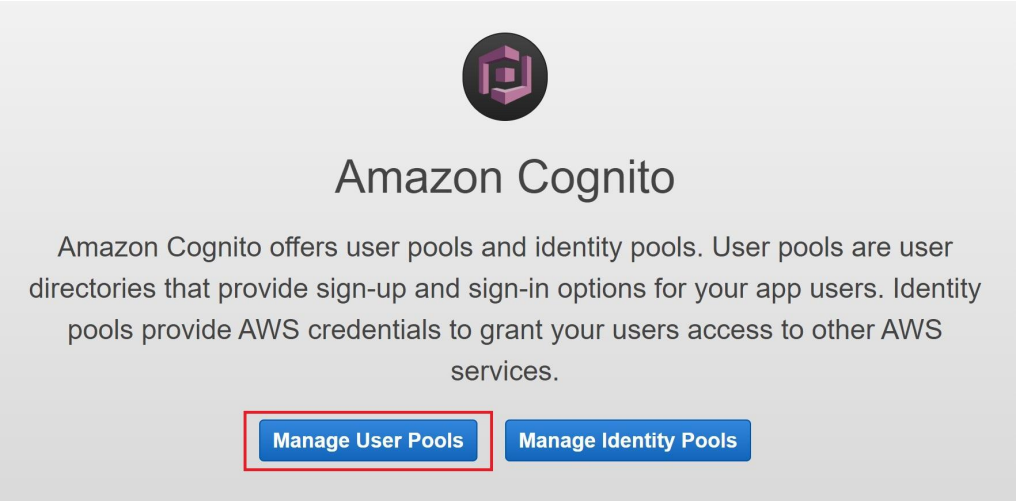
o. Cognito setup is **complete**.

4.3.1 Retrieve Cognito Credentials for Environment Variables

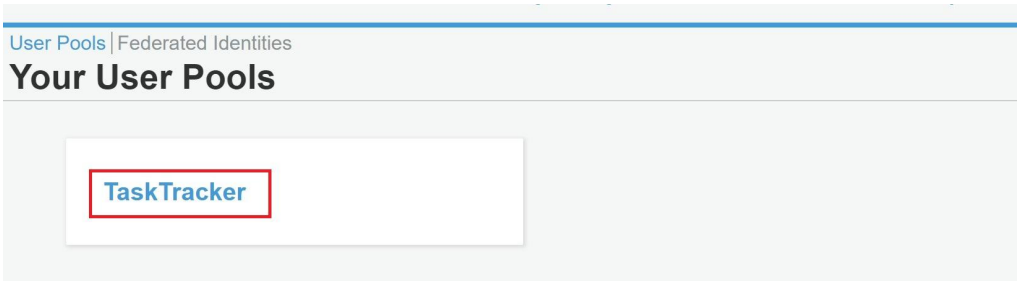
a. Log into the AWS Console

b. Click **search**, then search for **Cognito**. Select **Cognito** as pictured below

c. Click **Manage User Pools**

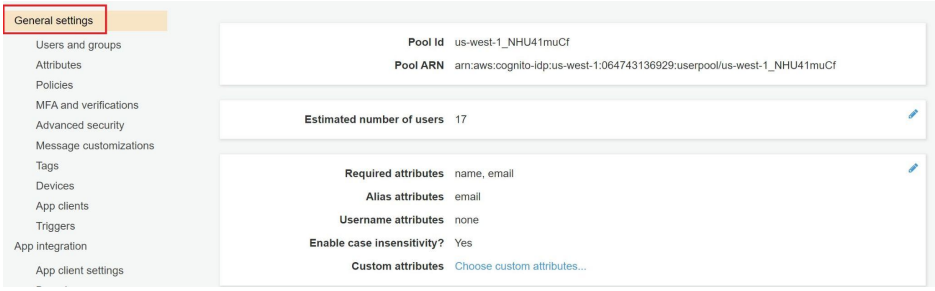


d. Click **TaskTracker**



e. For general information like **USER_POOL_ID** and **AWS_REGION**

i. Click **General Settings**

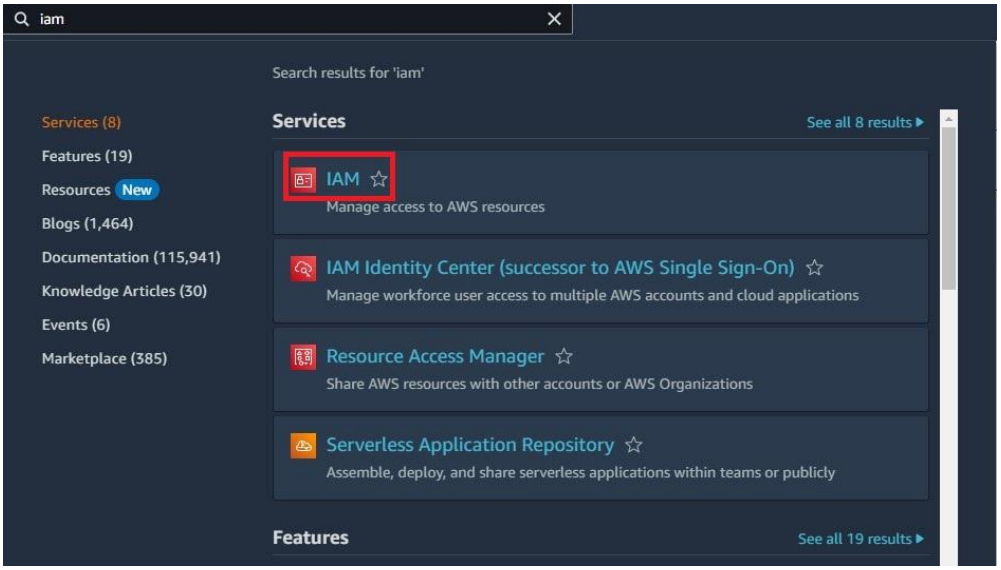


f. For **CLIENT_ID, CLIENT_SECRET,** and **COGNITO_CLIENT_NAME**

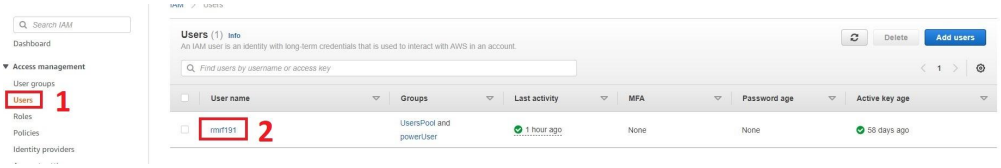
i. Click **App Clients**, then click **Show Details**

g. For **ACCESS_KEY**

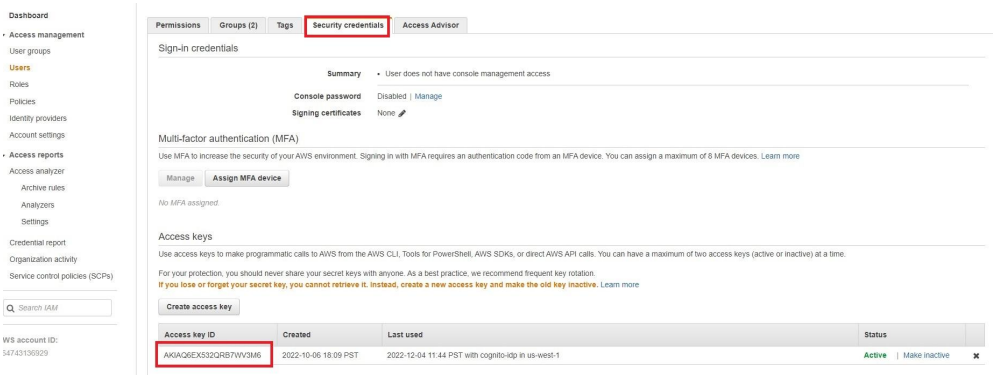
i. Log into the AWS console, click **Search**, then search **IAM**. Click **IAM**.



ii. Click **Users**, then click **rmr191** (or your equivalent), then click **IAM**

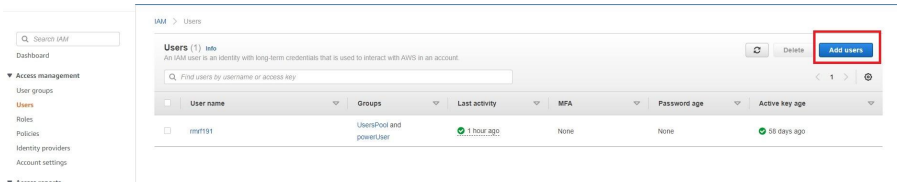


iii. Click **Security Credentials**

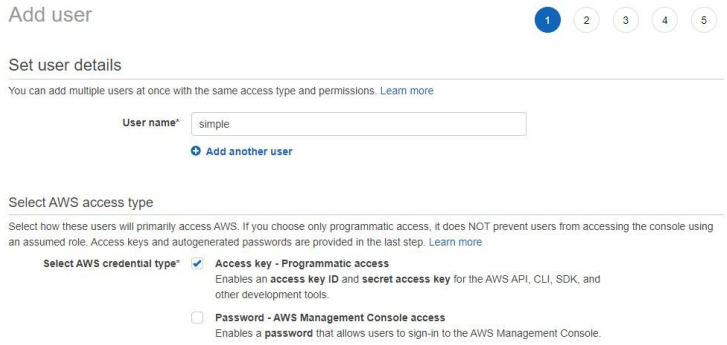


h. For **SECRET_KEY**, there is no way to find it on the user information board because AWS IAM only generates it one time. Every time you create an IAM user, you need to save the secret key in the last step of the process, as in the following example:

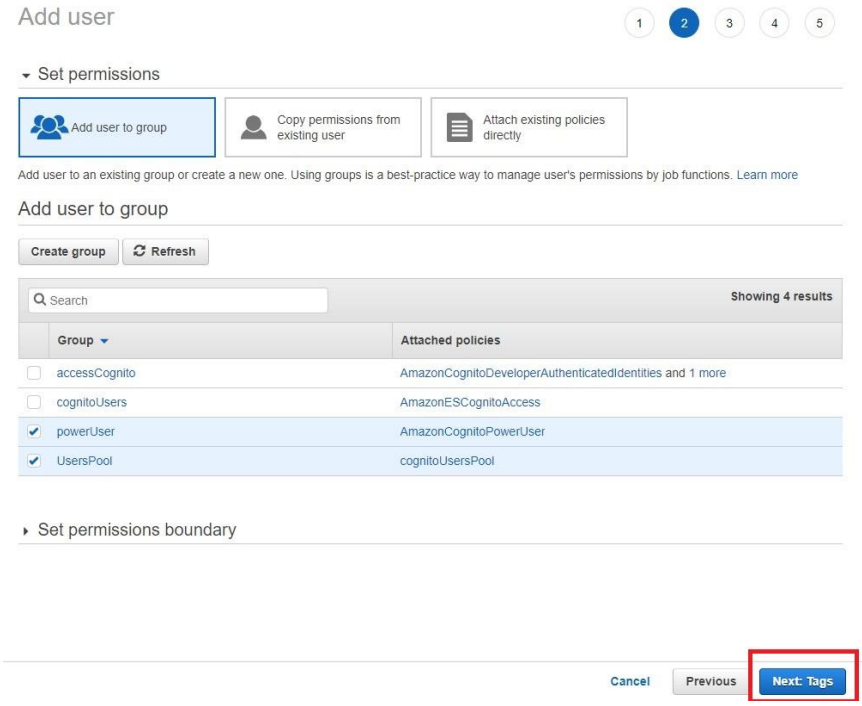
i. Click **Add Users**



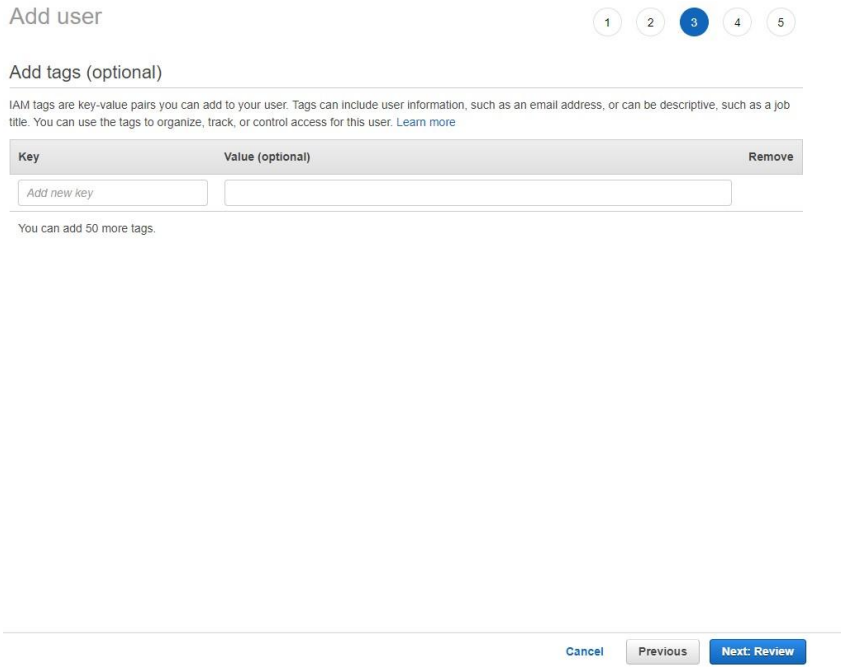
ii. Fill in **User name**, then click **Access Key**, then click **Next permissions**



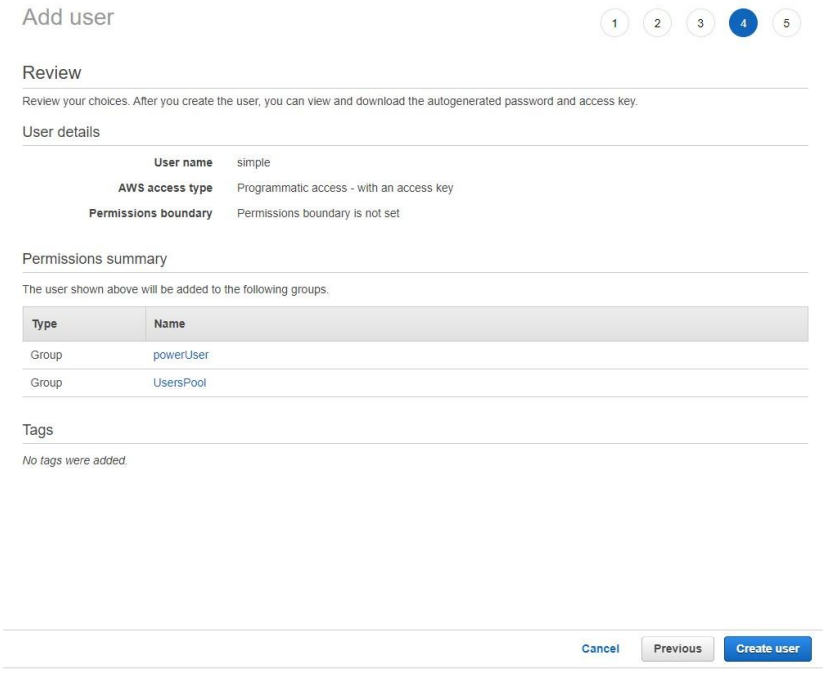
iii. Create or add **Group**, then click **Next: Tags**



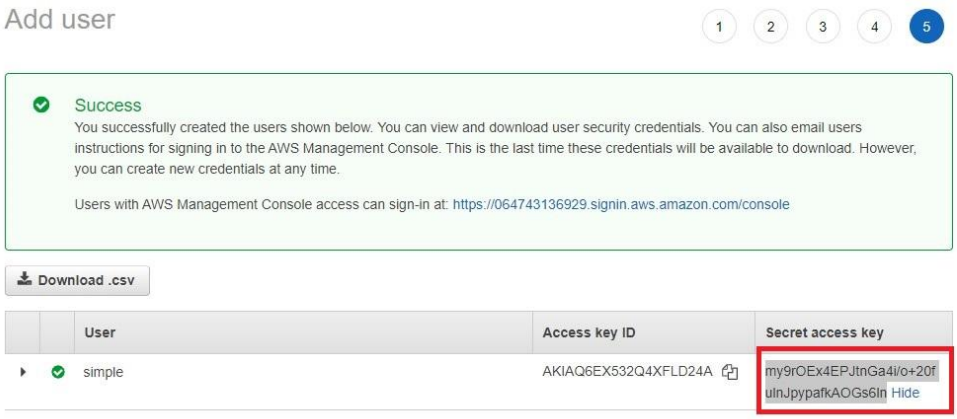
iv. Leave as default, or adjust based on needs, then click **Next: Review**



v. Review, then click **Create user**

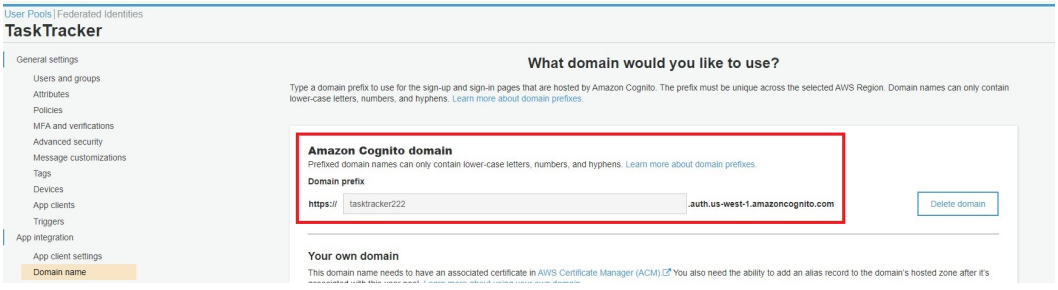


vi. **Save or copy** the **secret access key** somewhere safe before closing



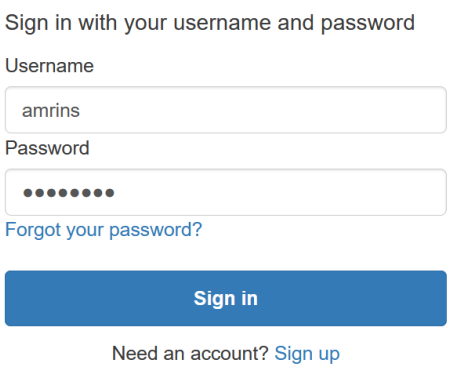
i. For **COGNITO_ISSUER_URI**:
   i. Go to https://docs.aws.amazon.com/general/latest/gr/cognito_identity.html, and find the **Endpoint.**
   ii. Set **COGNITO_ISSUER_URI** to **https://{endpoint}/{pool-id}**

j. For **SUBDOMAIN**:
   i. On the **Cognito Dashboard**, navigate to **Domain name**. This value will be used for the **SUBDOMAIN** environment variable.



5.0 User Interaction

5.1 Authentication

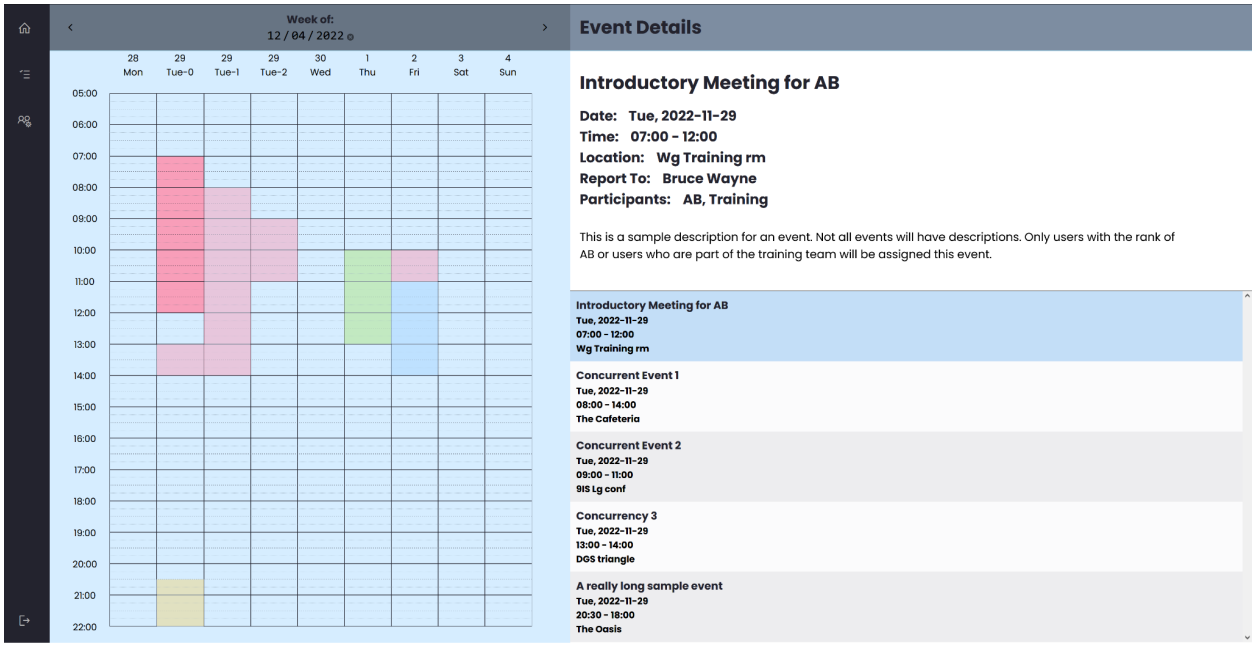


a.

i. Auth (including logout) is configured in the file **/src/main/java/com.rmrfroot.tasktracker222/configurers/SecurityConfiguration** in the method **configure()**

ii. User pool (for Cognito) is configured in the file **src/main/java/com/rmrfroot/tasktracker222/awsCognito/CreatePoolClient.java** in the methods **createCognitoClient()**, **getUserInfo()**, and **deleteUserByUsername()**

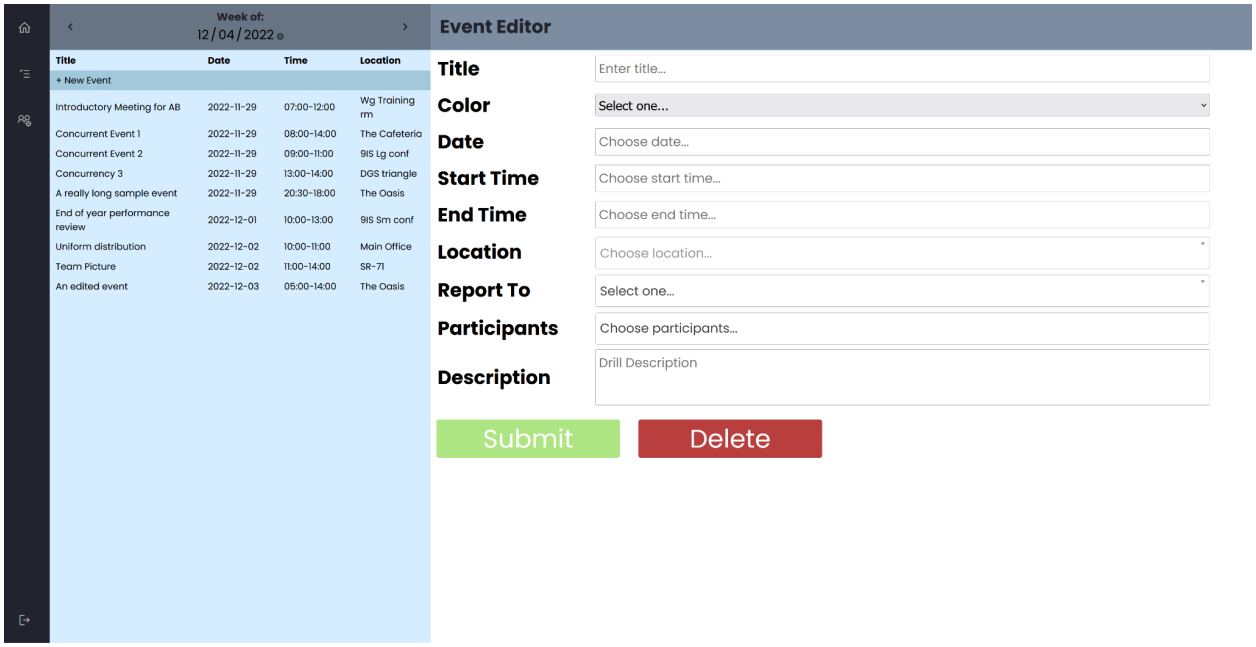
5.2 Drill Schedule



a. The drill schedule page is served by the controller in the file **src/main/java/com/rmrfroot/tasktracker222/controllers/DrillController.java** in the **drillScheduleRecipient()** method
b. The template used for the drill schedule page is the file **src/main/resources/templates/DrillScheduler.html**
c. The stylesheet used for the drill schedule page is the file **src/main/resources/static/js/user-management.js**
d. The procedural generation for the calendar and the intraday summary is handled in the file **src/main/resources/static/js/drill-schedule-generator.js**
e. The rendering of events on the calendar, including concurrency handling, is in the file **src/main/resources/static/js/drill-schedule-render.js**
f. The selection of events (i.e. bringing an event into focus on the calendar and intraday summary, as well as displaying details of the event) is handled in the file **src/main/resources/static/js/drill-schedule-selection.js**
g. The selection of drills shown to the user are calculated in the file **src/main/java/com/rmrfroot/tasktracker222/services/DrillDaoImpl.java** in the method **findDrillsInWeekOfDateByID()**, which takes the user's ID and the currently selected week as inputs
h. The algorithm handling concurrency (i.e., calculating when extra columns are needed if there are two concurrent events) can be found in the file **src/main/java/com/rmrfroot/tasktracker222/services/DrillDaoImpl.java** in the method **findConcurrencyInDay()**

i. Drill entities are defined in the file
src/main/java/com/rmrfroot/tasktracker222/entities/Drill.java

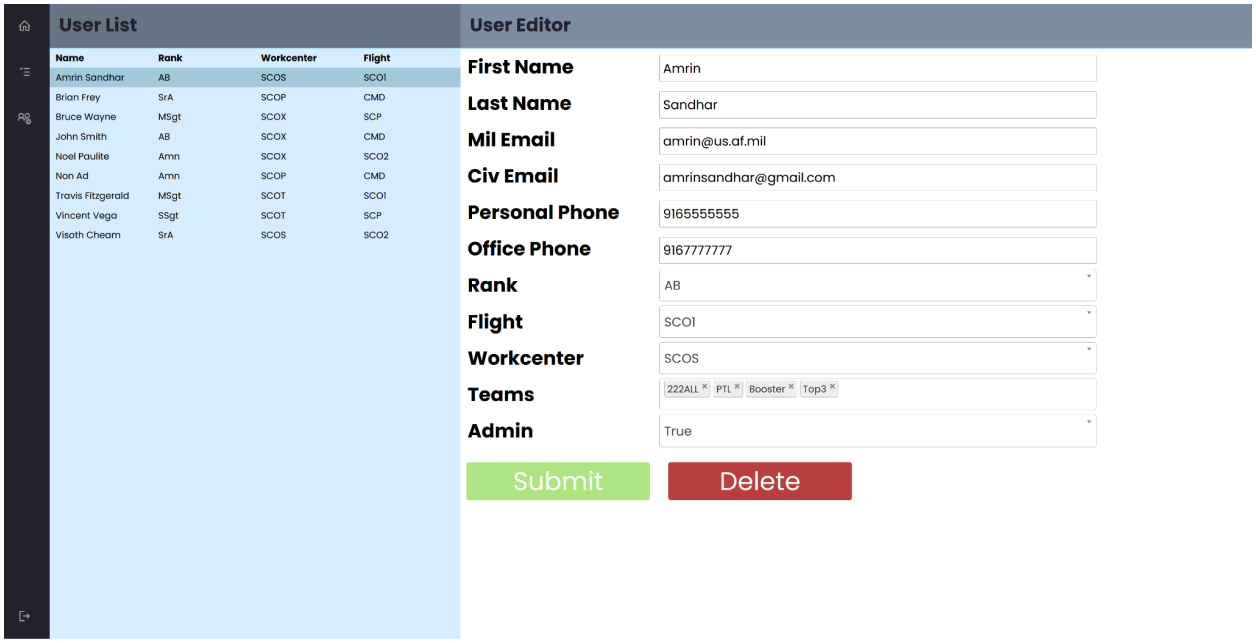
5.3 Drill Management



a. The drill management page is served by the controller in the file **src/main/java/com/rmrfroot/tasktracker222/controllers/DrillController.java** in the method **drillScheduleManager()**
b. The template used for the drill manager page is the file **src/main/resources/templates/DrillManagement.html**
c. The stylesheets used for the drill management page are **all the files** found in the **src/main/resources/static/drill-management** directory
d. The functions used for formatting and dynamically updating fields for drill management can be found in the file **src/main/resources/static/js/drill-management.js**
e. Submissions on the drill management page (new drill or edit drill postmapping) is served by the controller in the file **src/main/java/com/rmrfroot/tasktracker222/controllers/DrillController.java** in the method **editDrill()**
f. Submissions on the drill management page (delete drill postmapping) is served by the controller in the file **src/main/java/com/rmrfroot/tasktracker222/controllers/DrillController.java** in the method **deleteDrill()**
g. Submissions of new drills or edits to existing drills to the database are handled in the **src/main/java/com/rmrfroot/tasktracker222/services/DrillDaoImpl.java** file in the **save()** and **update()** methods respectively

h. Deletions of drills in the database are handled in the **src/main/java/com/rmrfroot/tasktracker222/services/DrillDaoImpl.java** file in the **deleteById()** method
i. Drill entities are defined in the file **src/main/java/com/rmrfroot/tasktracker222/entities/Drill.java**
j. Groups (i.e. rank, flight, workcenter, teams, locations) are defined in the file **src/main/java/com/rmrfroot/tasktracker222/entities/Group.java**

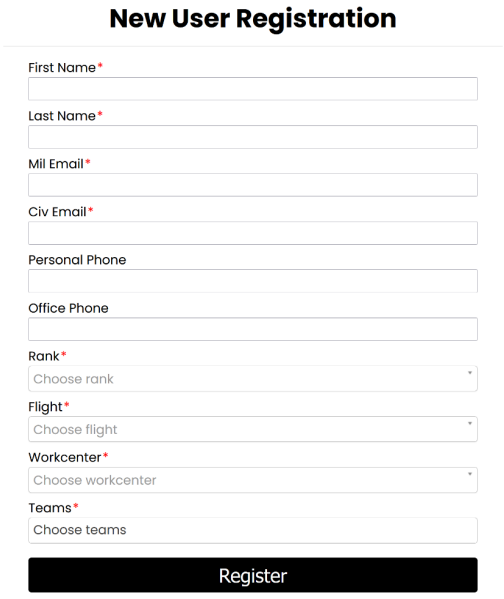
5.4 User Management



a. The user management page is served by the controller in the file **src/main/java/com/rmrfroot/tasktracker222/controllers/UserController.java** in the method **userManager()**
b. The template used for the user manager page is the file **src/main/resources/templates/UserManagement.html**
c. The stylesheets used for the user management page are **all the files** found in the **src/main/resources/static/user-management** directory
d. The functions used for formatting and dynamically updating fields for user management can be found in the file **src/main/resources/static/js/user-management.js**
e. Submissions on the user management page (edit user postmapping) is served by the controller in the file **src/main/java/com/rmrfroot/tasktracker222/controllers/UserController.java** in the method **userEditSubmit()**
f. Submissions on the user management page (delete user postmapping) is served by the controller in the file **src/main/java/com/rmrfroot/tasktracker222/controllers/UserController.java** in the method **userEditDelete()**
g. Submissions of user edits to the database are handled in the **src/main/java/com/rmrfroot/tasktracker222/services/UsersDaoServiceImpl.java** file in the **update()** method

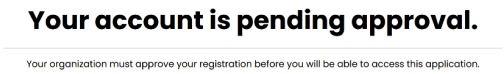
h. Deletions of users in the database are handled in the **src/main/java/com/rmrfroot/tasktracker222/services/UsersDaoServiceImpl.java** file in the **deleteById()** method
i. User entities are defined in the file **src/main/java/com/rmrfroot/tasktracker222/entities/User.java**
j. Groups (i.e. rank, flight, workcenter, teams, locations) are defined in the file **src/main/java/com/rmrfroot/tasktracker222/entities/Group.java**

5.5 User Registration



a. The user registration page is served by the controller in the file **src/main/java/com/rmrfroot/tasktracker222/controllers/UserController.java** in the method **addUser()**
b. The template used for the user registration page is the file **src/main/resources/templates/RegistrationForm.html**
c. The stylesheets used for the user registration page are **all the files** found in the **src/main/resources/static/registration** directory
d. The functions used for formatting and dynamically updating fields for user registration can be found in the file **src/main/resources/templates/RegistrationForm.html** in the inline <script> tag found at the bottom of the <body>
e. Submissions on the user registration page (new user postmapping) is served by the controller in the file **src/main/java/com/rmrfroot/tasktracker222/controllers/UserController.java** in the method **saveUser()**
f. Submissions of user edits to the database are handled in the **src/main/java/com/rmrfroot/tasktracker222/services/UsersDaoServiceImpl.java** file in the **save()** method
g. User entities are defined in the file **src/main/java/com/rmrfroot/tasktracker222/entities/User.java**
h. Groups (i.e. rank, flight, workcenter, teams, locations) are defined in the file **src/main/java/com/rmrfroot/tasktracker222/entities/Group.java**

5.6 Pending Approval



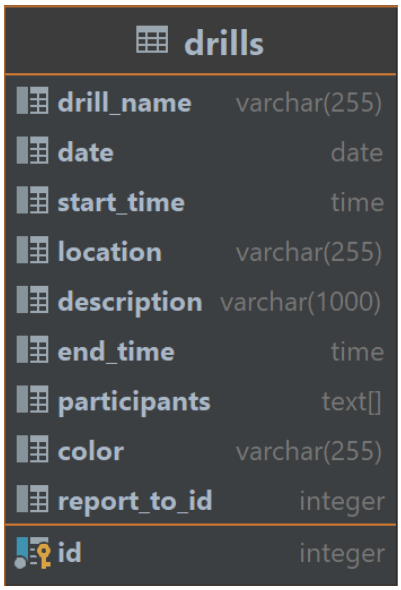
a. The pending approval page is served by the controller in the file **src/main/java/com/rmrfroot/tasktracker222/controllers/UserController.java** in the method **pendingApproval()**
b. The template used for the user registration page is the file **src/main/resources/templates/PendingApproval.html**

6.0 Database

6.1 ERDs

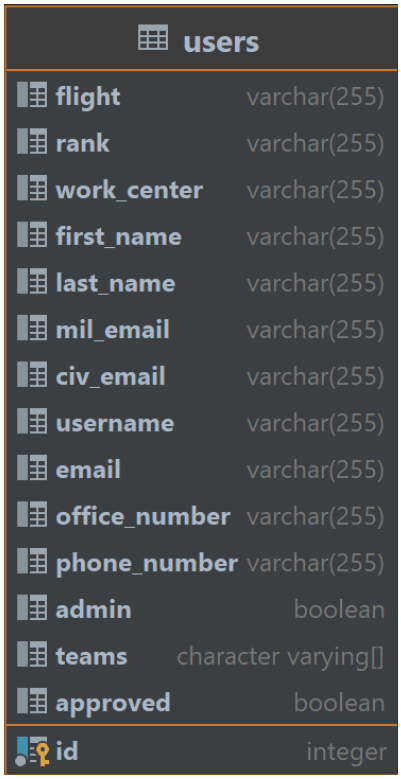
Note : There is no relational usage of the database. Only two, independent, tables are used.

a. Drills



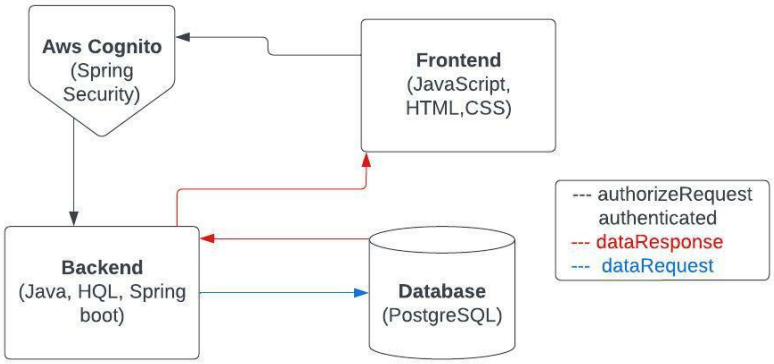

```
create table drills
(
    id           serial
        primary key,
    drill_name   varchar(255),
    date         date,
    start_time   time,
    location     varchar(255),
    description  varchar(1000),
    end_time     time,
    participants text[],
    color        varchar(255),
    report_to_id integer
);
```

b. Users




```
create table users
(
    id              serial
        primary key,
    flight          varchar(255),
    rank            varchar(255),
    work_center     varchar(255),
    first_name      varchar(255),
    last_name       varchar(255),
    mil_email       varchar(255),
    civ_email       varchar(255),
    username        varchar(255),
    email           varchar(255),
    office_number   varchar(255),
    phone_number    varchar(255),
    admin           boolean,
    teams           character varying[],
    approved        boolean
);
```

6.2 Other diagrams showing data flow



a.