

Lab 09

AWS RDS Review

Software Studio

DataLab, CS, NTHU

2021 Spring

Get content from GitLab

- Pull and checkout to the new branch from GitLab

- `git pull origin <branch>`
- `git checkout <branch>`
 - `weathermood_no_redux -> server-db` branch
 - `weathermood-server_no_redux -> db` branch

Outline

0. PostgreSQL Tutorial – Set Up a Local Database
1. Creating RDS Database
2. EB/RDS Connection – Security Group Setting
3. Set Up *weathermood* DB in RDS
4. Application Setting and Deploy

0. PostgreSQL Tutorial

– Set Up a Local Database

PostgreSQL Tutorial – Set Up a Local Database

- **Step 1: Connect to PostgreSQL Database Server**

- `psql -U postgres` : connect to db server as **User *postgres***

```
Joker@Joe-Macbook-Pro ~ INSERT psql -U postgres
psql (13.2)
Type "help" for help.
```

```
postgres=# |
```

PostgreSQL Tutorial – Set Up a Local Database

- **Step 2: Create Database *weathermood***

- **CREATE DATABASE *weathermood***

```
postgres=# CREATE DATABASE weathermood;
CREATE DATABASE
postgres=# \l
                                         List of databases
   Name    |  Owner   | Encoding | Collate      |   Ctype    | Access privileges
-----+-----+-----+-----+-----+-----+
  Joker   |  Joker   | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 postgres | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
template0 | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/postgres      +
                                         |           |           |           |           | postgres=CTc/postgres
template1 | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/postgres      +
                                         |           |           |           |           | postgres=CTc/postgres
 weathermood | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
(5 rows)
```

PostgreSQL Tutorial – Set Up a Local Database

- **Step 3: Connect to the database**

- **\c <dbname> <username>**: connect to the database with username
- **\dt**: List available tables

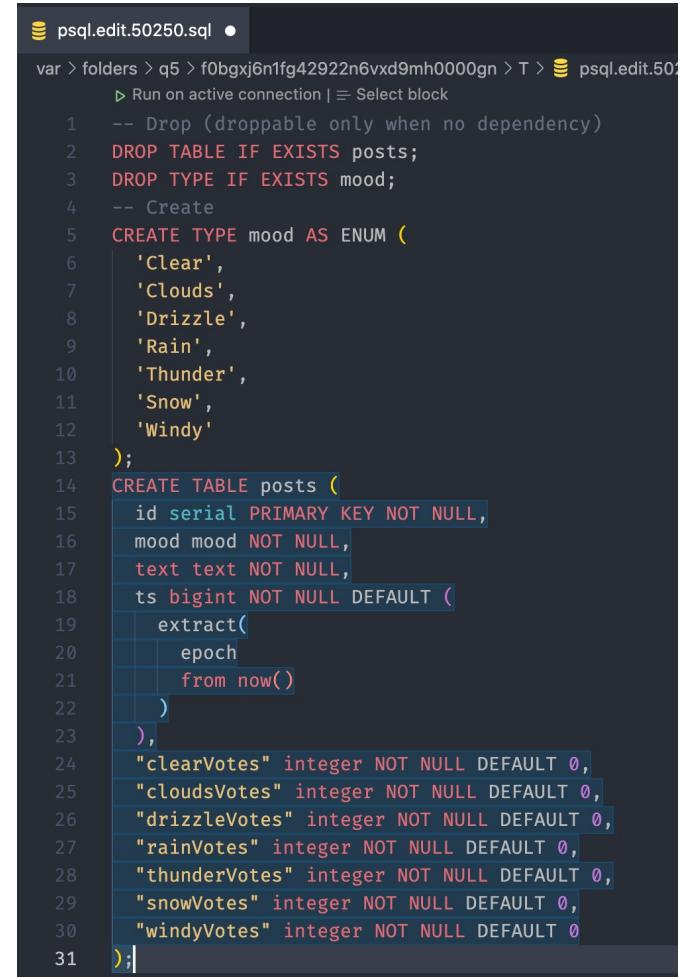
```
postgres=# \c weathermood
You are now connected to database "weathermood" as user "postgres".
weathermood=# \dt
Did not find any relations.
weathermood=# |
```

PostgreSQL Tutorial – Set Up a Local Database

• Step 4: Create a table

- `\e`: edit command in our own editor
- `DROP TABLE IF EXISTS <table name>`
- `CREATE TABLE <table name>`

```
weathermood=# \e
NOTICE:  table "posts" does not exist, skipping
DROP TABLE
NOTICE:  type "mood" does not exist, skipping
DROP TYPE
CREATE TYPE
CREATE TABLE
```



The screenshot shows a PostgreSQL SQL editor window titled 'psql.edit.50250.sql'. The code is as follows:

```
psql.edit.50250.sql
var > folders > q5 > f0bgxj6n1fg42922n6vxd9mh0000gn > T > psql.edit.50250.sql
-- Drop (droppable only when no dependency)
DROP TABLE IF EXISTS posts;
DROP TYPE IF EXISTS mood;
-- Create
CREATE TYPE mood AS ENUM (
    'Clear',
    'Clouds',
    'Drizzle',
    'Rain',
    'Thunder',
    'Snow',
    'Windy'
);
CREATE TABLE posts (
    id serial PRIMARY KEY NOT NULL,
    mood mood NOT NULL,
    text text NOT NULL,
    ts bigint NOT NULL DEFAULT (
        extract(
            epoch
            from now()
        )
    ),
    "clearVotes" integer NOT NULL DEFAULT 0,
    "cloudsVotes" integer NOT NULL DEFAULT 0,
    "drizzleVotes" integer NOT NULL DEFAULT 0,
    "rainVotes" integer NOT NULL DEFAULT 0,
    "thunderVotes" integer NOT NULL DEFAULT 0,
    "snowVotes" integer NOT NULL DEFAULT 0,
    "windyVotes" integer NOT NULL DEFAULT 0
);
```

PostgreSQL Tutorial – Set Up a Local Database

- **Step 5: Check the table**

- **\d <table name>**: Describe a table such as column, type, etc.

```
weathermood=# \dt
              List of relations
 Schema | Name   | Type  | Owner
-----+-----+-----+-----
 public | posts  | table | postgres
(1 row)

weathermood=# \d posts
                                         Table "public.posts"
      Column      | Type   | Collation | Nullable | Default
-----+-----+-----+-----+-----+
      id          | integer|           | not null | nextval('posts_id_seq'::regclass)
      mood         | mood    |           | not null |
      text         | text    |           | not null |
      ts           | bigint  |           | not null | date_part('epoch'::text, now())
      clearVotes   | integer |           | not null | 0
      cloudsVotes  | integer |           | not null | 0
      drizzleVotes | integer |           | not null | 0
      rainVotes    | integer |           | not null | 0
      thunderVotes | integer |           | not null | 0
      snowVotes    | integer |           | not null | 0
      windyVotes   | integer |           | not null | 0
Indexes:
 "posts_pkey" PRIMARY KEY, btree (id)
```

PostgreSQL Tutorial – Set Up a Local Database

- **Step 6: INSERT some test data**

- `INSERT INTO <table name> (column 1, column 2, ...)`**

- VALUES (value1, value2, ...)**

```
weathermood=# INSERT INTO posts (mood, text)
weathermood-#     SELECT
weathermood-#         'Clear',
weathermood-#         'word' || i || ' word' || (i+1) || ' word' || (i+2)
weathermood-#     FROM generate_series(1, 100) AS s(i);
INSERT 0 100
```

PostgreSQL Tutorial – Set Up a Local Database

- Step 7: Check the data

- `SELECT * FROM posts ORDER BY id DESC LIMIT 5;`

PostgreSQL Tutorial – Set Up a Local Database

- **Step 8: Create Index**

- **CREATE INDEX <idx_name> ON <table_name> USING [method]**

```
-- Extensions
CREATE EXTENSION IF NOT EXISTS pg_trgm;
-- Drop (droppable only when no dependency)
DROP INDEX IF EXISTS posts_idx_text;
DROP INDEX IF EXISTS posts_idx_ts;
CREATE INDEX posts_idx_ts ON posts USING btree(ts);
CREATE INDEX posts_idx_text ON posts USING gin(text gin_trgm_ops);|
```

PostgreSQL Tutorial – Set Up a Local Database

Step 9: Connect application to database – 1

- Set up environment variables
 - `NODE_ENV = development`
 - `PG_USERNAME = postgres`
 - `PG_HOSTNAME = localhost`
 - `PG_PORT = 5432`
 - `PG_DB_NAME = weathermood`

Windows

```
SET NODE_ENV=development|
```

Mac

```
export NODE_ENV=development|
```

PostgreSQL Tutorial – Set Up a Local Database

Step 9: Connect application to database – 2

- DB Connection settings

- config.js

```
switch (process.env.NODE_ENV) {  
  case 'development':  
    process.env.DB_URL = `postgres://${process.env.PG_USERNAME}@${process.env.PG_HOSTNAME}:  
    ${process.env.PG_PORT}/${process.env.PG_DB_NAME}`;
```

- model/posts.js

```
if (!global.db) {  
  const pgp = ...require('pg-promise')();  
  db = pgp(process.env.DB_URL);  
}
```

PostgreSQL Tutorial – Set Up a Local Database

Step 10: Define API to interact with DB - 1

- Define model to access database using SQL
 - server/model/posts.js

```
function list(searchText = '', start) {
  const where = [];
  if (searchText) where.push(`text ILIKE '%$1:value%'`);
  if (start) where.push('id < $2');
  const sql = `
    SELECT *
    FROM posts
    ${where.length ? 'WHERE ' + where.join(' AND ') : ''}
    ORDER BY id DESC
    LIMIT 10
  `;
  return db.any(sql, [searchText, start]);
}
```

PostgreSQL Tutorial – Set Up a Local Database

Step 10: Define API to interact with DB - 2

- Set up router to receive request and query
 - server/router/posts.js

```
// List
router.get('/posts', function (req, res, next) {
  const { searchText, start } = req.query;
  postModel
    .list(searchText, start)
    .then((posts) => {
      res.json(posts);
    })
    .catch(next);
});
```

PostgreSQL Tutorial – Set Up a Local Database

Step 10: Define API to interact with DB - 3

- Client send request based on the defined API
 - client/api/posts.js

```
export function listPosts(searchText = '', start) {  
  let url = `${postBaseUrl}/posts`;  
  let query = [];  
  if (searchText) query.push(`searchText=${searchText}`);  
  if (start) query.push(`start=${start}`);  
  if (query.length) url += '?' + query.join('&');  
  
  console.log(`Making GET request to: ${url}`);  
  
  return axios.get(url).then(function (res) {
```

PostgreSQL Tutorial – Set Up a Local Database

- Step 1: Connect to PostgreSQL Database Server
- Step 2: Create database *weathermood*
- Step 3: Connect to the database
- Step 4: Create a table
- Step 5: Check the table
- Step 6: INSERT some test data
- Step 7: Check the data
- Step 8: Create Index
- Step 9: Connect application to database
- Step 10: Define API to interact with DB

PostgreSQL Tutorial – Set Up a Local Database

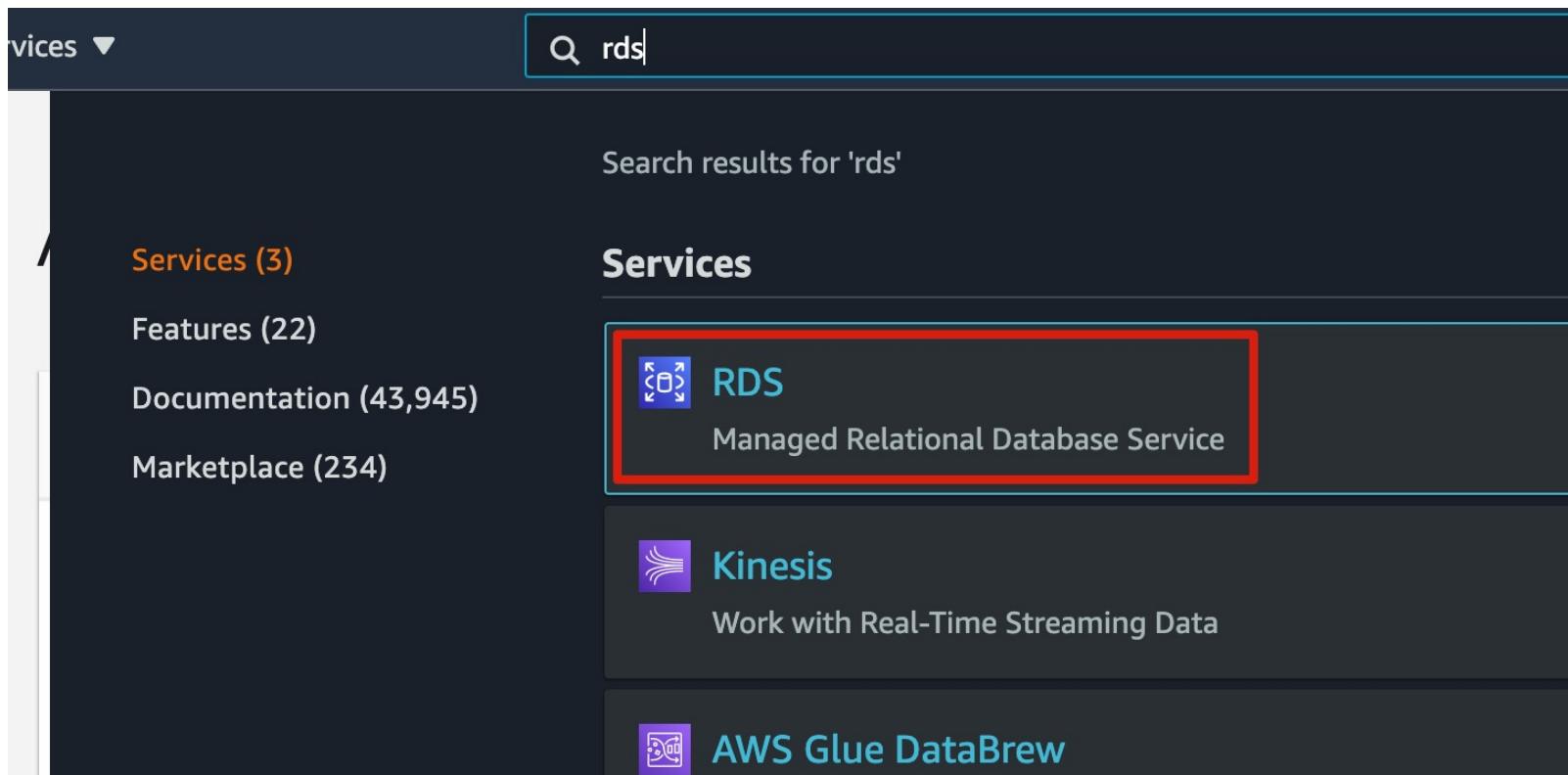
Some useful tricks:

1. Run 2 servers to avoid copy *dist* from client to server during development
2. Write a script to automate the setting of environment variables
 - **Reminder:** the setting would only be available in this shell

1. Creating RDS Database

Creating RDS Database

Step 1: Find RDS service



Creating RDS Database

Step 2: Start creating database

The screenshot shows the Amazon RDS Dashboard. On the left, there is a sidebar with the following menu items:

- Dashboard
- Databases
- Query Editor
- Performance Insights
- Snapshots
- Automated backups
- Reserved instances
- Proxies

The "Dashboard" item is highlighted in orange. On the right, there is a main content area with the following content:

Amazon Aurora
Amazon Aurora is a MySQL- and PostgreSQL-compatible storage engine with built-in support for multi-model data storage, 6-way replication across three availability zones, and automatic failover.

Create database (This button is highlighted with a red box.)

Or, [Restore Aurora DB cluster from S3](#)

Resources

You are using the following Amazon RDS resources in the [AWS CloudFormation stack](#):

Creating RDS Database

Step 3: Database settings - 1

- Method: Standard create
- Engine type: PostgreSQL
- Version: 12.5-R1 (default for free tier)

Choose a database creation method [Info](#)

Standard create
You set all of the configuration options, including ones for availability, security, backups, and maintenance.

Easy create
Use recommended best-practice configurations. Some configuration options can be changed after the database is created.

Engine options

Engine type [Info](#)

<input type="radio"/> Amazon Aurora 	<input type="radio"/> MySQL 	<input type="radio"/> MariaDB 
<input checked="" type="radio"/> PostgreSQL 	<input type="radio"/> Oracle 	<input type="radio"/> Microsoft SQL Server 

Version

PostgreSQL 12.5-R1 ▾

Creating RDS Database

Step 3: Database settings - 2

- Template: Free tier
- Master username: postgres
(recommended)

Templates
Choose a sample template to meet your use case.

Production
Use defaults for high availability and fast, consistent performance.

Dev/Test
This instance is intended for development use outside of a production environment.

Free tier
Use RDS Free Tier to develop new applications, test existing applications, or gain hands-on experience with Amazon RDS.
[Info](#)

Settings

DB instance identifier [Info](#)
Type a name for your DB instance. The name must be unique across all DB instances owned by your AWS account in the current AWS Region.

The DB instance identifier is case-insensitive, but is stored as all lowercase (as in "mydbinstance"). Constraints: 1 to 60 alphanumeric characters or hyphens (1 to 15 for SQL Server). First character must be a letter. Can't contain two consecutive hyphens. Can't end with a hyphen.

Credentials Settings

Master username [Info](#)
Type a login ID for the master user of your DB instance.

1 to 16 alphanumeric characters. First character must be a letter

Auto generate a password
Amazon RDS can generate a password for you, or you can specify your own password

Creating RDS Database

Step 3: Database settings - 3

- DB instance class: default
- Storage: default, only uncheck storage autoscaling

DB instance class

DB instance class [Info](#)
Choose a DB instance class that meets your processing power and memory requirements. The DB instance class options below are limited to those supported by the engine you selected above.

Standard classes (includes m classes)
 Memory optimized classes (includes r and x classes)
 Burstable classes (includes t classes)

db.t2.micro
1 vCPUs 1 GiB RAM Not EBS Optimized

Include previous generation classes

Storage

Storage type [Info](#)
General Purpose (SSD)

Allocated storage
20 GiB
(Minimum: 20 GiB, Maximum: 16,384 GiB) Higher allocated storage [may improve](#) IOPS performance.

Storage autoscaling [Info](#)
Provides dynamic scaling support for your database's storage based on your application's needs.

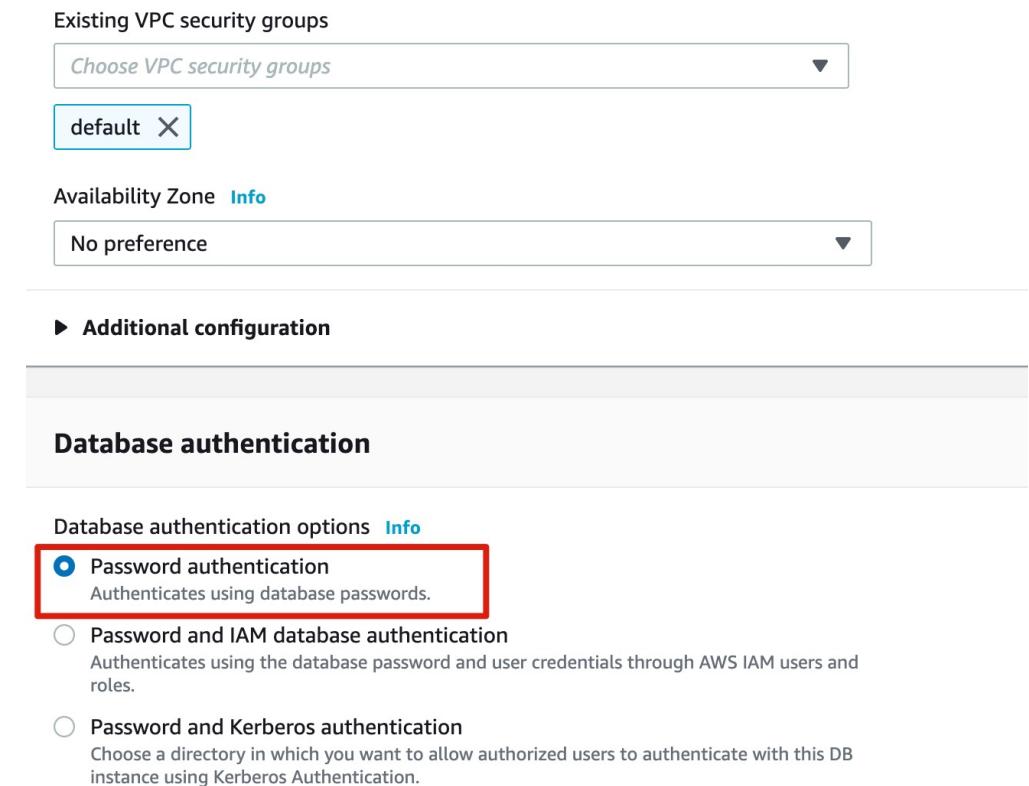
Enable storage autoscaling
Enabling this feature will allow the storage to increase once the specified threshold is exceeded.

Creating RDS Database

Step 3: Database settings – 4

- Database authentication:

Password authentication



Creating RDS Database

Step 3: Database settings – 5

- Review: make sure you are using **free tier**

Estimated monthly costs

The Amazon RDS Free Tier is available to you for 12 months. Each calendar month, the free tier will allow you to use the Amazon RDS resources listed below for free:

- 750 hrs of Amazon RDS in a Single-AZ db.t2.micro Instance.
- 20 GB of General Purpose Storage (SSD).
- 20 GB for automated backup storage and any user-initiated DB Snapshots.

[Learn more about AWS Free Tier.](#)

When your free usage expires or if your application use exceeds the free usage tiers, you simply pay standard, pay-as-you-go service rates as described in the [Amazon RDS Pricing page](#).

 You are responsible for ensuring that you have all of the necessary rights for any third-party products or services that you use with AWS services.

Cancel

Create database

Creating RDS Database

Step 4: Check if DB successfully created

- Reminder: it might take few minutes to launch

The screenshot shows the AWS RDS Databases page. At the top, a blue banner displays the message: "Creating database **weathermood-demo**. Your database might take a few minutes to launch." and a "View credential details" button. Below the banner, the navigation path is "RDS > Databases". The main interface has a header with "Databases", "Group resources" (which is turned off), and several buttons: "Modify", "Actions", "Restore from S3", and a prominent orange "Create database" button. There is also a "Filter databases" search bar and a pagination control with page number "1". The main table lists one database entry:

DB identifier	Role	Engine	Region & AZ	Size	Status
weathermood-demo	Instance	PostgreSQL	us-east-1a	db.t2.micro	Creating

2. EB/RDS Connection – Security Group Setting

EB/RDS Connection – Security Group Setting

Step 1: Database details checking

- Security groups
- Public Accessibility

RDS > Databases > weathermood-demo

weathermood-demo

Modify

Summary

DB identifier weathermood-demo	CPU 3.39%	Status Available	Class db.t2.micro
Role Instance	Current activity 0 Sessions	Engine PostgreSQL	Region & AZ us-east-1a

Connectivity & security Monitoring Logs & events Configuration Maintenance & backups Tags

Connectivity & security

Endpoint & port	Networking	Security
Endpoint weathermood-demo.cqsiyg8sz0kj.us-east-1.rds.amazonaws.com	Availability zone us-east-1a	VPC security groups default (sg-16c59435) (active)
Port 5432	VPC vpc-f9485a83	Public accessibility Yes
	Subnet group	

EB/RDS Connection – Security Group Setting

Step 2: EB security group setting - 1

- EB -> Environments -> Configuration -> Instance

The screenshot shows the AWS Elastic Beanstalk configuration interface for the environment 'weathermood-server-dev'. The left sidebar lists environments, applications, and various configuration options. The 'Configuration' link under the 'weathermood-server-dev' environment is highlighted with a red box and the number '2'. The main pane displays configuration categories: Software, Instances, and Capacity. The 'Instances' category is currently selected and has its 'Edit' button highlighted with a red box and the number '3'. The 'Software' category shows environment properties like NODE_ENV, RDS_DB_NAME, etc. The 'Capacity' category shows details like AMI ID, instance type, and scaling settings.

Category	Options	Actions
Software	Environment properties: NODE_ENV, RDS_DB_NAME, RDS_HOSTNAME, RDS_PASSWORD, RDS_PORT, RDS_USERNAME Log streaming: disabled Proxy server: nginx Rotate logs: disabled X-Ray daemon: disabled	Edit
Instances	EC2 security groups: awseb-e-ywwwsqwzhx-stack-AWSEBSecurityGroup-1KGXY03ZSCWZ, default IMDSv1: enabled IOPS: container default Monitoring interval: 5 minute Root volume type: container default Size: container default	Edit
Capacity	AMI ID: ami-018e5831cb11c140f Environment type: single instance Instance type: t2.micro Scaling cooldown: 360 seconds Time-based Scaling:	Edit

EB/RDS Connection – Security Group Setting

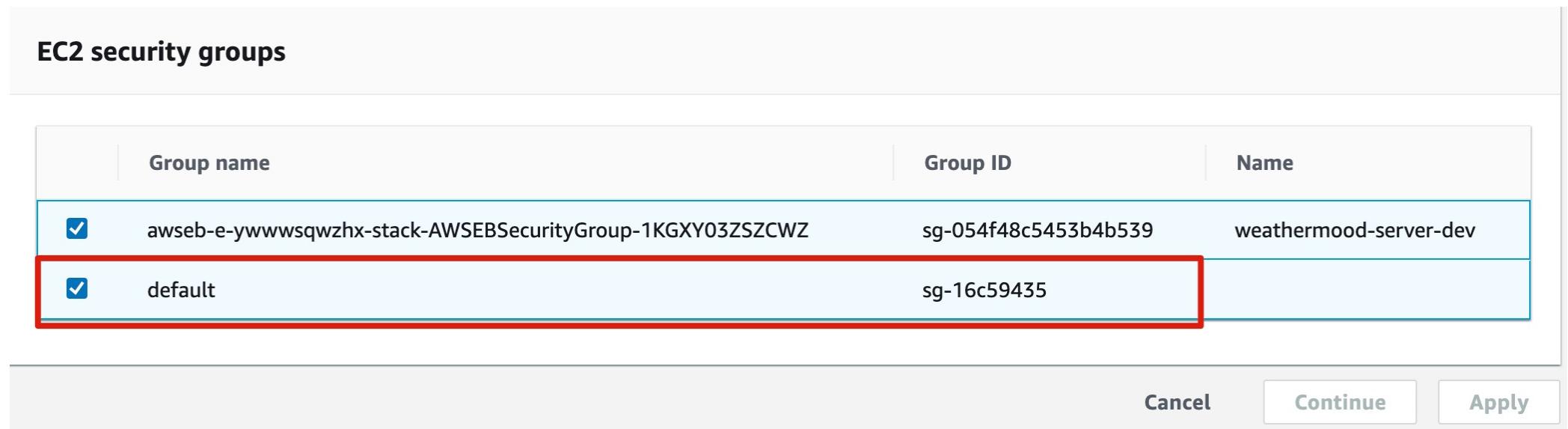
Step 2: EB security group setting - 2

- EC2 security groups: add the one from your DB

EC2 security groups

Group name	Group ID	Name
<input checked="" type="checkbox"/> awseb-e-ywwwsqwzhx-stack-AWSEBSecurityGroup-1KGXY03ZSZCWZ	sg-054f48c5453b4b539	weathermood-server-dev
<input checked="" type="checkbox"/> default	sg-16c59435	

Cancel **Continue** **Apply**



EB/RDS Connection – Security Group Setting

Step 3: Security group inbound rules setting - 1

- RDS -> Databases -> Click security group

The screenshot shows the Amazon RDS console interface. On the left, there's a sidebar with navigation links: Dashboard, **Databases**, Query Editor, Performance Insights, Snapshots, Automated backups, Reserved instances, Proxies, Subnet groups, Parameter groups, Option groups, and Custom Availability Zones. The main area displays details for a database instance named "weathermood-demo". Key information includes:

DB identifier	CPU	Status	Class
weathermood-demo	3.39%	Available	db.t2.micro
Role	Current activity	Engine	Region & AZ
Instance	0 Sessions	PostgreSQL	us-east-1a

Below this, a navigation bar has tabs: **Connectivity & security** (which is active), Monitoring, Logs & events, Configuration, Maintenance & backups, and Tags.

The **Connectivity & security** section contains three columns:

Endpoint & port	Networking	Security
Endpoint weathermood-demo.cqsiyg8sz0kj.us-east-1.rds.amazonaws.com	Availability zone us-east-1a	VPC security groups default (sg-16c59435) (active)
	VPC	

EB/RDS Connection – Security Group Setting

Step 3: Security group inbound rules setting - 2

- Inbound rules -> Edit Inbound rules

The screenshot shows the AWS Security Groups console. At the top, there is a search bar with the text "search: sg-16c59435" and a "Clear filters" button. Below the search bar is a table header with columns: Name, Security group ID, Security group name, VPC ID, and Description. A single row is listed: Name is "sg-16c59435", Security group ID is "sg-16c59435", Security group name is "default", VPC ID is "vpc-f9485a83", and Description is "default VPC security gr...".

Below the table, the security group "sg-16c59435 - default" is selected. The navigation tabs at the bottom are "Details" (highlighted with a red box and number 1), "Inbound rules" (highlighted with a red box and number 1), "Outbound rules", and "Tags".

The "Inbound rules (3)" section contains three rows:

Type	Protocol	Port range	Source	Description - optional
PostgreSQL	TCP	5432	140.114.85.17/32	-
PostgreSQL	TCP	5432	sg-16c59435 / default	-
All traffic	All	All	36.224.53.217/32	-

A red box with number 2 surrounds the "Edit inbound rules" button located at the bottom right of the "Inbound rules" section.

EB/RDS Connection – Security Group Setting

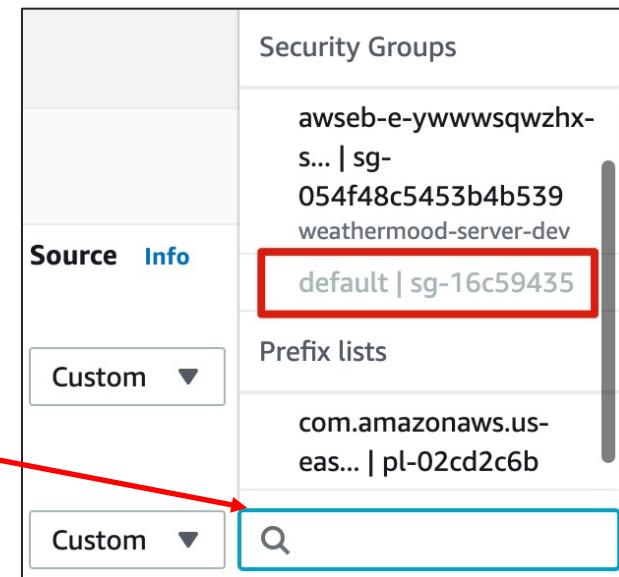
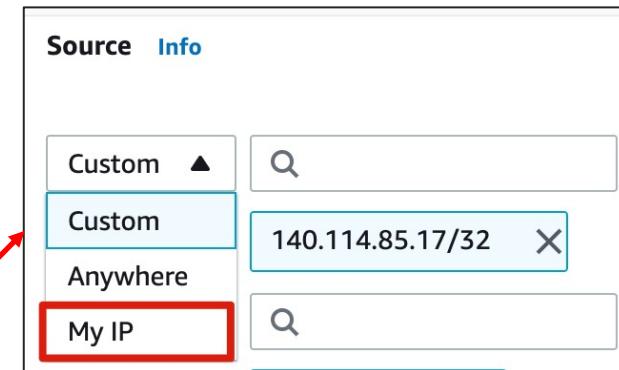
Step 3: Security group inbound rules setting - 3

- Add 2 rules:
 - Type: PostgreSQL
 - Source: My IP, Security Group

Inbound rules [Info](#)

Type Info	Protocol Info	Port range Info	Source Info	Description
PostgreSQL	TCP	5432	Custom ▼ 140.114.85.17/32 X	
PostgreSQL	TCP	5432	Custom ▼ sg-16c59435 X	
All traffic	All	All	Custom ▼ 36.224.53.217/32 X	

[Add rule](#)



3. Set Up *weathermood* DB in RDS

Set Up *weathermood* DB in RDS

Step 1: Create *weathermood* DB

- `createdb -h <rds-endpoint> -p 5432 -U postgres <DB Name>`

Step 2: Connect to *weathermood* DB

- `psql -h <rds-endpoint> -p 5432 -U postgres -d <DB Name>`

Options:

-h: host

-p: port

-U: username

-d: dbname

```
Joker@Joe-Macbook-Pro ~ ➤ INSERT ➤ createdb -h weathermood-demo.cqsiyg8sz0kj.us-east-1.rds.amazonaws.com -p 5432 -U postgres weathermood
Password:
Joker@Joe-Macbook-Pro ~ ➤ INSERT ➤ psql -h weathermood-demo.cqsiyg8sz0kj.us-east-1.rds.amazonaws.com -p 5432 -U postgres -d weathermood
Password for user postgres:
psql (13.2, server 12.5)
SSL connection (protocol: TLSv1.2, cipher: ECDHE-RSA-AES256-GCM-SHA384, bits: 256, compression: off)
Type "help" for help.

weathermood=> |
```

Set Up *weathermood* DB in RDS

Step 3: Define Schema / Create Table

- Method 1: Migrate schema
 - `pg_dump -h <dev-server> -U <dev-user> --no-owner --schema-only -c weathermood > db.dump`
 - `psql -h <rds-endpoint> -U <res-user> weathermood < db.dump`
- Method 2: Connect to remote psql server first and manually create

```
weathermood=> CREATE TABLE posts (
weathermood(>     id          serial PRIMARY KEY NOT NULL,
weathermood(>     mood         mood NOT NULL,
weathermood(>     text         text NOT NULL,
weathermood(>     ts           bigint NOT NULL DEFAULT (extract(epoch from now())),
weathermood(>     "clearVotes" integer NOT NULL DEFAULT 0,
weathermood(>     "cloudsVotes" integer NOT NULL DEFAULT 0,
weathermood(>     "drizzleVotes" integer NOT NULL DEFAULT 0,
weathermood(>     "rainVotes"   integer NOT NULL DEFAULT 0,
weathermood(>     "thunderVotes" integer NOT NULL DEFAULT 0,
weathermood(>     "snowVotes"   integer NOT NULL DEFAULT 0,
weathermood(>     "windyVotes"  integer NOT NULL DEFAULT 0
weathermood(> );|
```

Set Up *weathermood* DB in RDS

Step 4: Generate dummy data

```
weathermood=# INSERT INTO posts (mood, text)
weathermood-#     SELECT
weathermood-#         'Clear',
weathermood-#         'word' || i || ' word' || (i+1) || ' word' || (i+2)
weathermood-#     FROM generate_series(1, 100) AS s(i);
INSERT 0 100
```

4. Application Setting and Deploy

Application Setting and Deploy

Step 1: Add environments variables on EB environments

- Method 1: Using EB CLI
 - **eb setenv NODE_ENV=production, RDS_HOSTNAME=...**
 - Reminder: you also need **RDS_PASSWORD** for the DB_URL, check *server/config.js*
- Method 2: Using AWS console: EB -> Environments -> Configuration -> Software

The screenshot shows the AWS Lambda Configuration page for the environment 'weathermood-server-dev'. On the left, there's a sidebar with links like 'Go to environment', 'Logs', 'Health', 'Monitoring', 'Alarms', 'Managed updates', 'Events', and 'Tags'. The main area has a search bar at the top. Below it is a table with three columns: 'Category', 'Options', and 'Actions'. Under the 'Category' column, it says 'Software'. In the 'Options' column, there's a list of environment properties: 'Environment properties: NODE_ENV, RDS_DB_NAME, RDS_HOSTNAME, RDS_PASSWORD, RDS_PORT, RDS_USERNAME', 'Log streaming: disabled', 'Proxy server: nginx', 'Rotate logs: disabled', and 'X-Ray daemon: disabled'. To the right of the table is an 'Edit' button, which is highlighted with a red box.

Category	Options	Actions
Software	Environment properties: NODE_ENV, RDS_DB_NAME, RDS_HOSTNAME, RDS_PASSWORD, RDS_PORT, RDS_USERNAME Log streaming: disabled Proxy server: nginx Rotate logs: disabled X-Ray daemon: disabled	Edit

Application Setting and Deploy

Step 2: Configuration before deploy

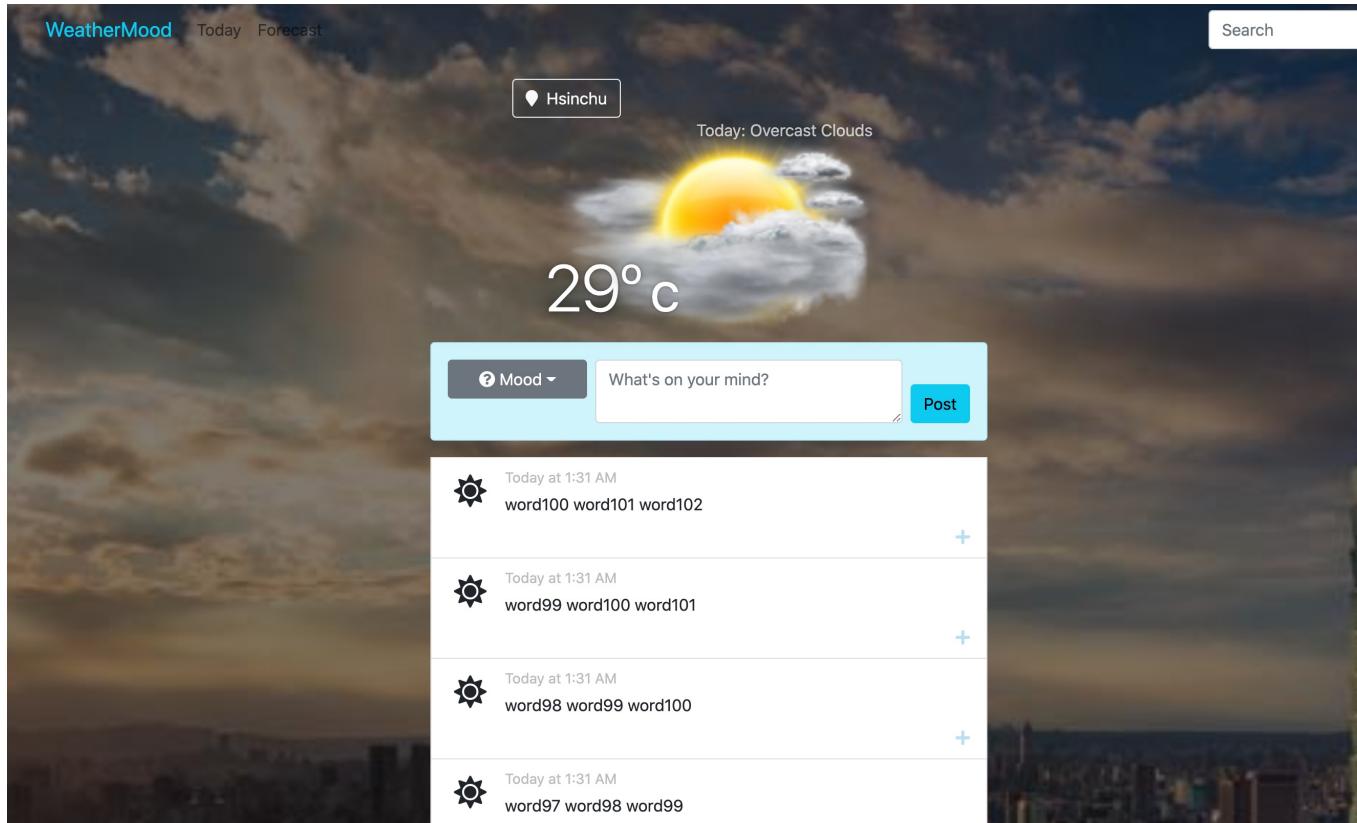
- Change `postBaseUrl` to your server
- Change `OpenWeatherAPI Key` to your key
- Build client project and copy `dist` to the server project

Step 3: Deploy to EB

- Commit before deploy
- `eb deploy <environment>`
- **Reminder:** You need to specify the environment this time

Application Setting and Deploy

Step 4: Check in your browser



Application Setting and Deploy

Some useful tricks:

- **eb console**: Open the EB console in browser
- **eb list**: List all environments
- **eb printenv**: Show the environment variables