**RDS**

In today's lab, we will migrate from MongoDB to PostgreSQL and from hosting a database on the same instance as our application server to hosting it separately using Amazon RDS. These two changes are each important in their own right.

**All your eggs in one basket**

Our implementation of Sparkify 1 suffers from a problem: since we are using one instance for everything, when one application goes down, it's possible that it will take the rest of the system down with it. This also makes debugging difficult. In a system as simple as ours, it's easy to tell where the problem comes from (e.g., MongoDB running out of space), but in a more complex architecture, that may not be so obvious. The solution is to partition our architecture so that each instance does only one thing. To that end, we will now want 3 instances:

* One EC2 instance to parse the data and insert it into our database
* One RDS instance to host our database
* Another EC2 instance to pull data from our database and present it to the user

Note that one of the effects of partitioning our architecture in this way means that our software dependencies are also isolated. At this point, you can stop your EC2 instance. We will spin up two new ones after we've launched our new database management system (DBMS).

**New feature in Postgres 9.2: Mongo**

Ever since version 9.2, PostgreSQL has supported the JSON datatype. This gives us the flexibility of storing the semistructured data that we enjoy in MongoDB without having to give up the functionality of Postgres and the relational data model in general.

**Getting Started with Amazon RDS**

For today's lab, start by [Creating a PostgreSQL DB Instance and Connecting to a Database on a PostgreSQL DB Instance](http://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/CHAP_GettingStarted.CreatingConnecting.PostgreSQL.html). (To stay on the [free tier](https://aws.amazon.com/rds/free/), you'll want to do a Single-AZ deployment on a db.t2.micro instance with no more than 20 GB storage.)

**Important:** Be sure to use Postgres 9.6 or later.

**DDL**

There are two parts of the Structured Query Language: the data definition language (DDL) and the data manipulation language (DML). Most of the time we are focused on the DML which provides us with the CRUD operations: INSERT, SELECT, UPDATE, DELETE, before we do any of that, we need a data model to manipulate. MongoDB lets us off easy by not requiring any DDL statements. You refer to a collection and if it doesn't already exist, MongoDB automagically creates it for you. (This can lead to a proliferation of orphan collections due to typos.) Postgres and other RDBMSs require you to CREATE your tables before you put anything into them. For this project, we will keep it simple. You need only [CREATE](https://www.postgresql.org/docs/current/static/sql-createtable.html) a table with a single column of the [JSON](https://www.postgresql.org/docs/current/static/datatype-json.html) datatype. Later we will work on normalizing this data to make it more useful for Online Analytical Processing (OLAP).

To do this using [pgAdmin](https://www.pgadmin.org/docs/pgadmin4/4.19/index.html" \t "_blank):

1. Connect to your RDBMS. The easiest way to do this is to click "Add New Server" under "Quick Links".
   1. Name it whatever you want (I pick the same name I used in AWS for simplicity).
   2. Go to the "Connection" tab
   3. Enter the Host address. This is the URL provided by AWS which should be [your database name].[some hash].[your region (*i.e.*us-east-1)].rds.amazonaws.com. This appears as your "endpoint" (not including :5432) in your RDS configuration detailed-schedule
   4. Leave the port (5432) and maintenance database (postgres) alone
   5. Fill in the user name and password you specified when launching your database. You may (if you wish) save your password.
   6. You may ignore Role and SSL mode for now.
   7. Save.
2. Create your table
   1. You'll now see your RDS server on the left under Servers. Expand it.
      * You'll now see Databases, Login/Group Roles, and Tablespaces under your server. Click "Databases"
        + You'll now see your database along with "postgres" under Databases. Ignore "postgres" (that is the maintenance database) and click on the one you created in AWS.
          - You'll now see Casts, Catalogs, Event Triggers, Extensions, Foreign Data Wrappers, Languages, and Schemas. These are all advanced topics to be covered on your own at a later date. For now, click "Schemas"

For now, there should just be one Schema: "public". Expand it.

Now we have Collations, Domains, *etc. &c.*

Right click on "Tables" and select Create -> Table...

Name your table raw\_events

Leave Owner, Schema, and Tablespace default and go to the "Columns" tab

Hit the "+" on the right to create a new column

* Name: status
* Data type: json
* Length and Precision do not apply to json data
* For now, go with the default and allow NULL and do not set a primary key. We will fix this when we normalize our data tomorrow.

**Pipeline**

Once your database has been created and your data model defined, you will need to populate your database. You can start with the code you used to populate MongoDB. There are a few things to watch out for when connecting to RDS.

1. Don't forget to put all your credentials in your api\_cred.yml file (and not in your code).
2. In order to convert from a Python dict to a Postgres Json datatype, you will want to import the Json class from psycopg2.extras.
3. The [execute](http://initd.org/psycopg/docs/cursor.html#cursor.execute) method expects a tuple or list.
4. Don't forget to [commit](http://initd.org/psycopg/docs/connection.html#connection.commit) your transaction after your insert. You may commit after every insert or, to save time, you may want to commit after every 100 or so inserts.

**Optional:** This time, try making this instance using the [Amazon Linux AMI](https://aws.amazon.com/amazon-linux-ami/) instead of Ubuntu. Of course, your user name will not be ubuntu this time, but ec2-user. The next thing you will notice is that you will be using [yum](https://en.wikipedia.org/wiki/Yellowdog_Updater,_Modified) instead of [apt](https://en.wikipedia.org/wiki/Advanced_Packaging_Tool) to install Linux packages. Just as before, you will want to start by updating everything (sudo yum update instead of sudo apt-get update). Unlike Ubuntu, you may need to install gcc-c++ in order for some python packages to work. Some packages may also have slightly different names (*e.g.* python3-psycopg2 instead of python-psycopg2). Of course, you can always search for packages just as you could with apt (*e.g.* yum search python postgres)

**N.B.:** You may find that RDS will not accept a connection from your EC2 instance (even though it may from your laptop). This is because RDS automatically allows connections from the IP address that created it (*i.e.* your laptop) but no where else. By now you should know how to fix this. The important thing is to change the inbound rules on the security group for your RDS instance to allow the source to be anywhere (or, if you want to be more restrictive, you can set it to just the security group you are using for your EC2 instance).

**Querying**

Now that you have data streaming into your database, trying querying it. Take a look at the [PostgreSQL JSON Tutorial](http://www.postgresqltutorial.com/postgresql-json/) to see how. Start by selecting out the name of each song. *Hint*: The [json\_array\_elements](https://www.postgresql.org/docs/current/static/functions-json.html" \t "_blank) function may come in handy.

Finally, implement a similar web application to that of Sparkify 1, but use the Postgres database instead of the MongoDB one.

**Submission:** Please submit your code and screenshots of the web application as a single .zip file. In the submission instructions, include the URL/public IP address of the user-facing web application, as well as the names of corresponding instances on your EC2 account.