Setting up an AWS EC2 Instance to support EngagemoreCRM*

John Wooten Ph.D.

March 13, 2020

Version: 1.0

Abstract

The document describes the steps taken to install a webhost server on Amazon Web Services. This proceeds from setting up a free-tier account, creating the instance, establishing the necessary database, and setting up the backup procedures. The software scripts that provide the backups and management of the logs are included in the Appendices[1].

*Software developed under contract with EngagemoreCRM

[1] `https://texblog.org/2008/04/02/include-source-code-in-latex-with-listings/`

Contents

List of Figures

List of Tables

1 Track Changes

Date	*Editor*	*Comment*	*Version*
200311	J. Wooten	Initial Draft	1.0

Table 1: Table of Changes

2 Introduction

3 Creating AWS EC2 Instance

I have been developing a webhook for a client to coordinate between a shopping cart service and a new CRM service. The shopping cart is used to sell various categories and levels of the CRM service and keeps track of all the credit cards, payments, cancellations, and allows for affiliate relationships. When a customer wishes to use one of the services of the CRM site, they are directed to the shopping cart where they pick out their desired level and enter their payment information. The shopping cart then sends a notification to a webhook you provide. The webhook I developed needed to track the product that was bought, then if it?s a new account on the CRM, create that account and save the shopping cart customer id, and the CRM account id along with the product type.

I chose to write the webhook in php (more on this later), and store the data using MySQL. I did the initial development on my own local apache server running on a Mac mini. I started by creating a GitHub project at GitHub.com and cloned the empty project to my local server in the web server area. On my Mac that was /Library/Server/Web/Data/Sites/Default/. I wrote it so that the server addresses, account information, and passwords were all in an external config.ini.php file. I added the config.ini.php filename to the .gitignore file so it would not be checked in. After doing this I committed the webhook project and pushed it to GitHub. I ran a lot of tests using my local server and local mysql server, then decided the webhook needed to be on something more robust and with better bandwidth and scalability if this project was to grow.

So, I went to aws.amazon.com and (either create an account or) log into my account. You should look at security and create a key pair. Name it and download one part to your local computer. I?m on a Mac so I put my key file into /.ssh/mykey.pem. Set the protection on this file and directory to 700 to only allow yourself to use, read, or change the files here.

If you don?t have an instance then create a small free EC2 one. I chose the smallest Linux 64 bit ARM type. Add security resources for SSH on port 22 and give it access to 0.0.0.0/0 unless you want to restrict it. I travel and need to ssh in from various places, so I used the above pattern. I?m protected because you have to have the pem file to ssh in. then http resources on port

80, 0.0.0.0/0;::0, which allows http access from all IP4 and IP6 addresses. You can put a pattern here to restrict it to a particular subset, also. Assign an elastic-ip and associate it with the instance.

Once you do, then ?spin up? the instance by clicking on it, noting the elastic-ip and then click on Actions/State/Start The instance should start up. after it has started, then from a terminal window, follow the below after ssh ec2-user@elastic-ip and login.

The below is the best example I could assemble of how to get an apache server established that would interpret php and use a MySql server to run a webhook. If you click on Instances and Launch an Instance, you will be given choices. Here is what I chose:

```
> sudo\ yum\ -y\ install\ httpd $(installs the httpd daemon)
> sudo\ yum\ -y\ install\ php $ (installs php)
> httpd\ -v $ (check the version apache httpd daemon)
> php\ -v $(check the version of php installed)
> sudo\ service\ httpd\ start $(start the apache server)
> sudo\ usermod\ -a \ G\ apache\ $ec2-user add ec2-user to
    apache)
> exit $( exit ssh session and then reconnect , to activate the
    above)
> groups $( See what groups you are in, should include apache )
> sudo\  chown \ R\ $ec2-user:apache\ /var/www (allow these
    groups access to the web server area www)
> sudo\ chmod\ 2775\ /var/www $(set permissions for user and
    group to allow writing and reading)
> find\ /var/www\ -type\ d \ exec\ sudo\ chmod\ 2775\ {}\ \;$ (
    set directories all to this)
> find /var/www\ -type \  -exec\ sudo\ chmod\ 0664\ {}\ \; $(
    files to this)
> echo ''<?php phpinfo(); ?>'' > /var/www/html/phpinfo.php$
> sudo\ rm\ -rf\ /etc/httpd/conf.d/welcome.conf $
(remove the default welcome page so that your php page can be
    accessed)
> sudo\ service\ httpd \ estart $(restart the apache server)

In your browser
     http://elastic-ip/phpinfo.php$
> sudo\ yum\ install\ git $
> cd\ /var/www/html$
> sudo\ git\ clone\ https://github.com/your-area/your.git$
```

This provided what I wanted. I had searched for how to get the server,

php, mysql all set up and found an exhausting complex array of choices. I tried several. They all failed for various reasons. With technical help from support at AWS, I took what they had and modified it somewhat (they provided examples that didn?t work as they had been done by someone with root access! But, it was a helpful start). The below worked.

ssh to your ec2-user@your-elastic-ip, then

The free tier gives you 750 free hours per month on a small EC2 instance. That?s 24 hours a day for 31 days. But, each time you start, the minimum time charged is 1 hour. After setup, it?s better to just leave it running. You also get a free amount of disk space. Watch your consumption here. The fee if charged is low, so 2.15/750 Hrs so a year would cost about $25. You can also look at the Billing section under your account and set alerts if you go above a certain amount.

4 Setting up MySQL database on AWS EC2 Instance

```
> sudo yum install mysql-server
> sudo chkconfig mysqld on
> sudo service mysqld start
> /usr/libexec/mysql55/mysqladmin -u root password ?passwd?

> sudo chkconfig mysqld on
> sudo chkconfig httpd on
> sudo chkconfig ?list (check for mysqld and httpd)

> mysql -u root -p
> create database users\_db;
> use users\_db;
> source users\_db.sql ( give file path if needed ) should see
    create statements
> show tables;
> exit;
```

5 Remote MySQL access to AWS EC2

In the AWS console add the rule to allow mysql access to port 3306 from 0.0.0.0/0 ssh to the instance

```
> mysql -u root -p
   (enter root password)
> mysql> CREATE USER 'new-userid'@'%' IDENTIFIED BY 'passwd';
   Query OK, 0 rows affected (0.00 sec)

> mysql> GRANT ALL PRIVILEGES ON . TO ''new-userid''@''\%'';
> mysql$>$quit;

After completing the permissions described, from a remote
   terminal session try:

> mysql -h elastic-ip -u userid -p
   (enter passwd from CREATE USER)

You should be connected to the mysql database on AWS.
Try:

> show databases;
   ( should show several including users\_db )
> use users\_db;
> show tables;
> select * from users limit 10;
> exit

Before the above script will work, you must ensure that the rcp
   command will work from the AWS EC2 instance. I did this by:


>  ssh-keygen (on the AWS EC2 instance, saving key in .ssh/id\
   _rsa)
> ssh-copy-id remote\_userid@remote\_server (copies the key)
> ssh remote\_userid@remote\_server ( test that you can ssh to
   the server)
Now, we create the crontab entries that will cause the mysql
   dumps to be made on a particular schedule. To do this:


>  crontab -e (then enter the following)

# Minute        Hour    Day of Month    Month    Day of Week
```

```
            Command
# (0-59)        (0-23)      (1-31)          (1-12)     (0-6)
    0             3           *               *          *
        /home/userid/bin/backup daily >> /home/userid/logs/daily
    0             4           *               *          0
        /home/userid/bin/backup weekly >> /home/userid/logs/
        weekly
    0             5           1               *          *
        /home/userid/bin/backup monthly >> /home/userid/logs/
        monthly
    0             0           1               1          *
        /home/userid/bin/backup yearly >> /home/userid/logs/
        yearly


(then close the crontab file with shift-Z shift-Z

You can inspect the file by typing:

> crontab -l ( and you should see the above )
```

6 Backup Plan for AWS

Once you have your AWS EC2 instance up and running, saving data into your MySQL database, there should be a backup plan to prevent data loss. While AWS offers some that can be set up from the EC2 console, they can cost money as they save their data often to an S3 repository. I?ve found S3 repositories to be a bit awkward to mess with unless your dataset is very large. So, instead I created a bash script to control the backups and configured crontab and logrotate to handle it the standard Unix way.

```
~/backups/daily
         /weekly
         /monthly
         /yearly
 ~/logs/daily
       /weekly
       /monthly
       /yearly
```

Now, with that log structure, I created the following bash script /bin/backup:

```
#!/bin/bash
#
# Run a mysql dump and attach a date to the dumpfile for use by
    restore
#
# Usage: backup [daily|weekly|monthly|yearly], default daily
#
# directories ~/backups/daily, etc. should exist and be writable
#
# The remote server and the directory on it ~/backups/aws must
# exist and be writable.
#
PERIOD='daily'
if [  "$#" -gt 0  ]; then
   PERIOD=$1;
fi
DATE=`date +%Y%m%d`

/usr/bin/mysqldump -u root -ppass-word database > /home/userid/
    backups/$PERIOD/database-$DATE.sql
if [ "$?" -ne 0 ]; then
        echo "$DATE backup FAILED to $PERIOD because $?"
else
        echo "$DATE backup SUCCEEDED to $PERIOD with size: `du /
            home/userid/backups/$PERIOD/database-$DATE.sql`"
/usr/bin/rsync -avz --ignore-existing --progress --rsh "ssh -l
    remote-user-id"  /home/userid/backups/$PERIOD/database-$DATE.
    sql remote-server:backups/aws > /dev/null
    if [ "$?" -ne 0 ]; then
        echo "$DATE rcp FAILED to destination because: $?"
    else
        echo "$DATE rcp SUCCEEDED to destination"
    fi
fi
```

The above crontab entries will produce a daily backup of of the database(s) you entered in the /bin/backup bash script and place it into the /backups/-daily directory with a filename like database-20200301.sql, append the record of the backup to /logs/daily, and then copy the file to your remote server into /backups/aws. I recommend that you provide at least daily backups of your remote server, also.

7 Use logrotate to Manage Backup Volume on AWS EC2

The previous post showed a way to use a simple bash script to create daily, weekly, monthly, and yearly mysql backups using crontab. It also provided for storing a copy of the backups on a remote server as part of the script. Now, after monitoring the script for at least a few days and verifying that indeed the logs are appearing in the directories and are not zero length, you should consider adding some logrotate features that trim the log files periodically. I created the following file, /etc/logrotate.d/myproject:

```
/home/userid/backups/daily/*.sql {
rotate 14
daily
}
/home/userid/backups/weekly/*.sql {
rotate 8
weekly
}
/home/userid/backups/monthly/*.sql {
rotate 3
monthly
}
/home/userid/backups/yearly/*.sql {
rotate 3
yearly
}

This keeps 14 daily backups, 8 weekly backups, 3 monthly backups
    and 3 yearly backups and runs them as dictated in the
   logrotate.conf file. You can adjust the numbers and
   frequencies if desired. Now, it is necessary to edit the /etc
   /logrotate.conf file. I used the following :


> cat /etc/logrotate.conf
# see "man logrotate" for details
# rotate log files weekly
weekly

# keep 4 weeks worth of backlogs
rotate 4
```

```
# create new (empty) log files after rotating old ones
#create if you uncomment this you get 0 length files!

# use date as a suffix of the rotated file
#dateext ( I didn't want the file names changed, backup creates
    the names )

# uncomment this if you want your log files compressed
#compress (they were small and I didn't think I needed this)

# RPM packages drop log rotation information into this directory
include /etc/logrotate.d

# no packages own wtmp and btmp -- we'll rotate them here
/var/log/wtmp {
    monthly
    create 0664 root utmp
        minsize 1M
    rotate 1
}

/var/log/btmp {
    missingok
    monthly
    create 0600 root utmp
    rotate 1
}
# system-specific logs may be also be configured here.
```

The only changes I made were to eliminate the addition of the date to the files, since I handled that in my script and to eliminate the compression. I periodically check the /logs files to see that backups are occurring and that they are being copied to my remote server. I also periodically check my remote server /backups/aws directory to see that indeed the files are copied there.

Being absolutely certain that the system will get me if it can, I do hourly TimeMachine backups on my remote server (It?s a mac-mini running OS X 10.12.6 connected to a TimeCapsule). In addition, I use Carbon Copy Clone(CCC) to produce a nightly clone of the entire mac-mini disk drive to another external drive which is bootable. Every three months, I also do a separate clone to another drive which I keep in a fire-proof safe. All of my laptops, desktops, and servers are similarly backed up and in addition most of the codes and scripts I use are maintained in github.

Told you I was paranoid!