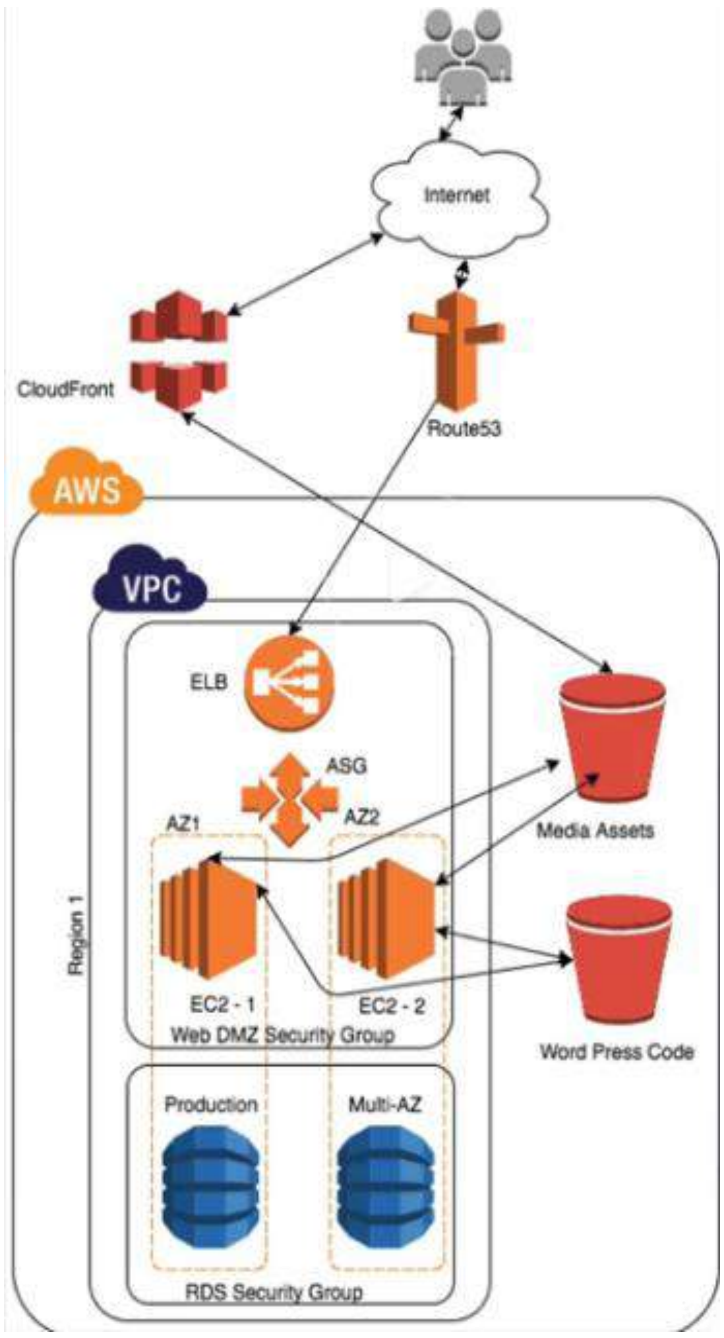
<u>Build a Fault Tolerant WordPress Site</u>

Here is a Diagram of the Architecture for our project:



Concepts covered in this project include: EC2, Security Groups, Auto Scaling, Multi-AZ, RDS, RDS fail-over, IAM roles, S3, S3 bucket policies, Application Load Balancers, Route 53, CloudFront, and AMI.

Architecture Description: Route 53 will connect to an elastic load balancer in the cloud. Our EC2 instances will be behind an auto scaling group. We will have EC2 instances in separate availability zones.

We will also have RDS instances in separate availability zones. We will have two S3 buckets. One S3 bucket will have our media assets for our wordpress site, and the other is going to have our code for WordPress. We will serve our photo files from our Wordpress site through CloudFront.

(For this lesson, I will be using N. Virginia Region. Go to the AWS management console and select the region at the top right of the screen)

<u>Step 1) Let's create our S3 buckets.</u>

First let's create the bucket that will contain our code.

Go to the AWS console. Click **Services**. Click **S3**.

Click **Create Bucket**

Bucket name: wordpress-code-bucket-2020 ←give your bucket a name similar to this so you can easily identify it as the bucket that will be storing your code.

Region: US East (N. Virginia)

Click **Create**

Now let's create another S3 bucket which will contain our media files

Click **Create Bucket**

Bucket name: wordpress-media-bucket-2020 ←give your bucket a name similar to this so you can easily identify it as the bucket that will be storing your media files.

Region: US East (N. Virginia)

Click **Create**

Our EC2 instances will be using these buckets, and so will CloudFront

<u>Step 2) Let's create a CloudFront Distribution</u>

Click **Services.** Click **CloudFront.**

Then click **Create Distribution**

★Click the **Get Started** button that is associated with creating a Web Distribution. (We do **NOT** want a RTMP distribution which is used for media files using adobe flash)

Origin Domain Name: wordpress-media-bucket-2020.s3.amazonaws.com ← select the media bucket we just created. This bucket is going to be our origin.

Origin ID: S3-wordpress-media-bucket-2020 ← this Origin ID will be automatically generated once you select the Origin Domain Name. Leave it as such.

Restrict bucket access: ✓NO ←Make sure to select No

Viewer Protocol Policy: ✓HTTP and HTTPS ← Make sure HTTP and HTTPS is selected

Scroll down and click **Create Distribution**

(This will take some time to deploy. Give it 15 minutes or so. Be patient)

<u>Step 3) Let's configure the security groups that we will need.</u>

Click **Services** and click **VPC**. Make sure to select your region as N. Virginia.

First we will create a security group for our EC2 instances (web servers)

Click **Security Groups** in the left margin.

Click **Create security group.**

Security Group Name: WebServerSGforWPsite ←give it a very specific name so you know that it is the security group for our web server for our Wordpress site.

Description: WebServerSGforWPsite ←for the description just put the name of the security group

VPC: vpc-7dd2ee07 ← ★We are just going to use the **default** VPC for this lesson and it will look something like this.

Then click **Add rule** button three times to add three **Inbound Rules** and configure them like so:

Type	Protocol	Port Range	Source
HTTP	TCP	80	Custom 0.0.0.0/0
HTTP	TCP	80	Custom ::/0
SSH	TCP	22	Custom 0.0.0.0/0

Then scroll down and click **Create Security Group**

Now let's create a security group for our RDS instances (database servers)

Click **Security Groups** in the left margin

Security Group Name: DatabaseServerSGforWPsite ←give it a very specific name so you know that it is the security group for our database server for our Wordpress site.

Description: DatabaseServerSGforWPsite ←for the description just put the name of the security group

VPC: vpc-7dd2ee07 ← ★We are just going to use the **default** VPC for this lesson and it will look something like this.

Then click **Add rule** button two times to add two **Inbound Rules** and configure them like so:

Type	Protocol	Port Range	Source
MYSQL/Aurora	TCP	3306	MY IP 172.88.102.35/32 ← ★★★Your IP will be different than mine
MYSQL/Aurora	TCP	3306	Custom sg-09928b2338e8be590 ←★★★select **WebServerSGforWPsite**

★ (for the first inbound rule select source as **MY IP** which is the IP for your computer…It will then automatically prompt the IP address for your computer which will look something like 172.88.102.35/32 However your IP will of course be different!!! To check that it prompted correctly you can go to google and type 'my ip address' and it will tell you what yours is.

★ (for the second inbound rule select the source as the security group for your web servers which we named WebServerSGforWPsite, which will then automatically prompt the corresponding Security Group ID number, which will look something like sg-09928b2338e8be590. Your Security Group ID number will of course be different when you do the lesson yourself)

<u>Step 4) Let's provision RDS</u>

Click **Services**. Click **RDS** which is located under Database.

Scroll down beneath the Resources section to the Create Database database section

And under Create Database click the **Create Database** button (which is located next to the Restore from S3 button) ★★★Do **NOT** click the Create database button that is associated with Amazon Aurora!!! Be careful not to do that because we want an RDS database and **NOT** an Aurora database!!!

Choose a database creation method: ✓Standard Create ← select Standard Create

Engine Options

Engine Type: MySQL ← ✓select the MySQL database

Version: MySQL 8.0.20 ← ✓select the latest version of MySQL

Templates: Dev/Test ← ✓select Dev/Test

DB instance identifier

DB instance identifier: wordpressproject ← give it a name to easily identify it

Master username: wordpressproject 🡐 for simplicity name it the same as the identifier

Master password: wordpressproject 🡐for simplicity name it the same as the username

Confirm password: wordpressproject 🡐 confirm the password

DB instance class: ✓Burstable classes (includes t classes) 🡐 select Burstable classes

✓db.t2.micro 🡐 select db.t2.micro

Storage type: ✓General Purpose (SSD) 🡐 select General Purpose (SSD)

Allocated Storage: ✓20 🡐 select 20

Storage autoscaling: ✓Enable Storage Autoscaling 🡐 select Enable Storage Autoscaling

Maximum storage threshold: ✓1000 GiB 🡐 select 1000 GiB

Multi-AZ deployment: ✓Create a standby instance (recommended for production usage) 🡐select Create a standby instance (recommended for production usage)

Virtual Private Cloud (VPC): ✓Default VPC (vpc-7dd2ee07) 🡐 select the default VPC it will look like (vpc-7dd2ee07) but yours will be different of course

Click **Additional connectivity configuration**

Publicly accessible : ✓No 🡐select No as we do not want it publicly accessible (RDS will not assign a public IP address to the database. Only Amazon EC2 instances and devices inside the VPC can connect to your database.)

Existing VPC Security Groups: ✓DatabaseServerSGforWPsite 🡐select the DatabaseServerSGforWPsite which is the security group that we created earlier. ★★★ Also, make sure to **uncheck** the default security group because we do **not** want to use that one in this case. Just to be extra clear…The only VPC Security Group that you will be selecting here will be the **DatabaseServerSGforWPsite**

Database port: ✓3306 🡐 select 3306 (this is the TCP/IP port that the database will use for application connections)

Click **Additional Configuration**

Initial database name: wordpressproject 🡐 For simplicity, name it wordpressproject

★★★if you forget to specify an initial database name, RDS will not create a database. So when you provision your wordpress instance it will have no database to connect into. Therefore, make sure to specify an initial database name!

Monitoring: **uncheck** Enable Enhanced Monitoring ← uncheck the box next to Enable enhanced monitoring. We do **not** need enhanced monitoring for this project.

Click **Create database**

(It might take up to 15 minutes to launch so be patient)

<u>Step 5) Let's create a Role (so that our EC2 instances can communicate to S3)</u>

Click **Services.** Then click **IAM** which is located under Security, Identity, & Compliance.

Click **Roles** in the left margin.

Click **Create Role**

Select type of trusted entity: ✓AWS service ← select AWS service

Choose a use case: ✓EC2 ← select EC2

Click **Next: Permissions**

Filter policies: s3 ← type s3 into the filter policies search bar

Policy name: ✓AmazonS3FullAccess ← select AmazonS3FullAccess as the policy

Then click **Next: Tags**

(we don't need to add any tags)

Click **Next: Review**

Role name*: S3forWPsite ← give it a name such as S3forWPsite

Then click **Create role**

<u>Step 6) Let's provision an EC2 instance</u>

Click **Services**. Click **EC2**.

Then click the **Launch instance** button

Click **Select** for the Amazon Linux 2 AMI (which is usually at the very top)

Check the box next to General purpose t2.micro which is free tier eligible

Click **Next: Configure Instance Details**

IAM role: ✓S3forWPsite ← make sure to select the IAM role as S3forWPsite which is the role we created in Step 5.

Scroll down to the **Advanced Details** section

★★★ Type or Copy the bootstrap script below and paste it into the **User data** text box

```
#!/bin/bash
sudo su
yum update -y
yum install httpd php php-mysql –y
amazon-linux-extras install php7.2 -y
cd /var/www/html
echo "healthy" > healthy.html
wget https://wordpress.org/latest.tar.gz
tar -xzf latest.tar.gz
cp -r wordpress/* /var/www/html/
rm -rf wordpress
rm –rf latest.tar.gz
chmod -R 755 wp-content
chown -R apache:apache wp-content
wget https://s3.amazonaws.com/bucketforwordpresslab-donotdelete/htaccess.txt
mv htaccess.txt .htaccess
chkconfig httpd on
service httpd start
chkconfig httpd on
```

(The above bootstrap script will elevate our privileges to root. Then it will update our operating system. It will then install apache and php and php-mysql. It will open up the /var/www/directory. It will create a file called healthy.html, which we will use to run a health check on our elastic load balancer. It will then get the latest version of WordPress. Then it will remove the install directory. It will change the permissions so that we can work with WordPress in our directory. Then we will use an HTaccess to allow us to do a URL rewrite, which will allow us to serve content out of CloudFront rather than doing it out of our S3 buckets. Then we will start Apache. Then we will make sure that our Apache service will start back up in case our EC2 instance restarts.)

Once you have typed or pasted in the bootstrap script click **Next: Add Storage**

(We will leave the storage as the default)

Click **Next: Add tags**

Key: Name 🡐type Name as the Key

Value: MyGoldenImage 🡐type MyGoldenImage as the Value

Click **Next: Configure Security Group**

✓Select an **existing** security group

✓WebServerSGforWPsite 🡐 select WebServerSGforWPsite which is the security group we created in step 2 of this project

Click **Review and Launch**

Then click **Launch**

✓Create a new key pair

Key pair name: WordPressSiteKeyPair 🡐 give your key pair a name

Click **Download Key Pair**

Then save the file somewhere on your computer so that you have it

Click **Launch Instance**

★ Once your instance called MyGoldenImage is **Running** make sure that your cloudfront distribution is **Enabled** and make sure that your RDS instance has a status of **Available** before you proceed.

<u>Step 7) Let's SSH into our EC2 instance to make sure apache is installed and make sure wordpress is installed</u>

Click Services click EC2 and click running instance

Select ✓MyGoldenImage

Click **Actions** then click **Connect**

Select ✓EC2 Instance Connect (browser-based SSH connection)

Then click **Connect**

Once you are connected into your instance type the following:

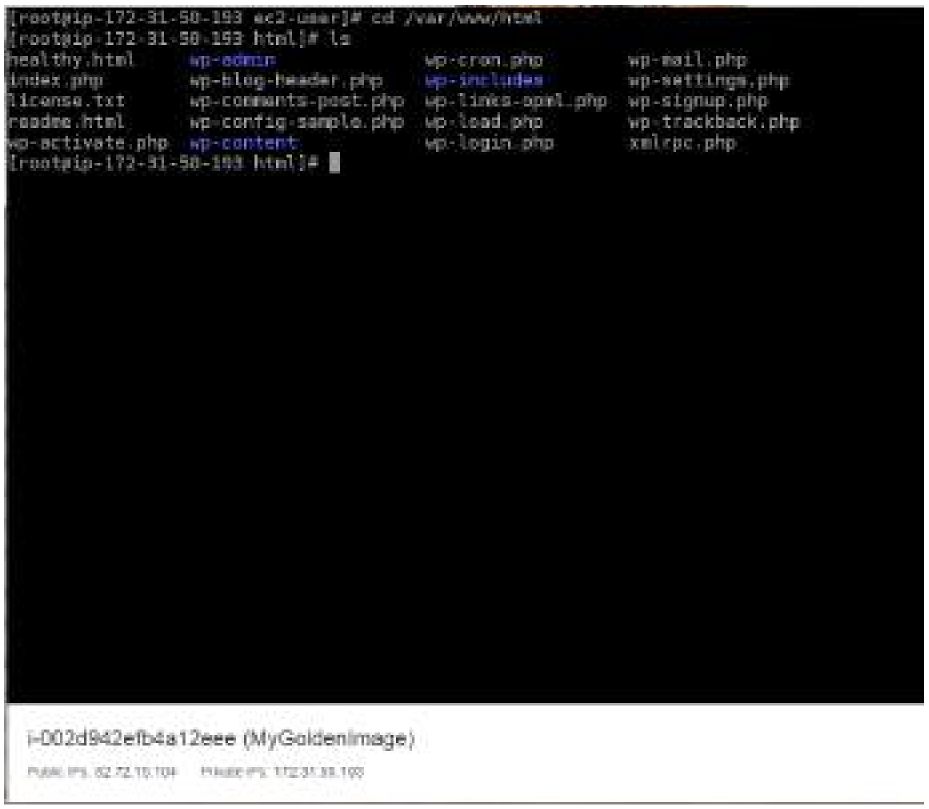
sudo su (and hit enter)

clear (and hit enter)

cd /var/www/html (and hit enter)

ls (and hit enter) 🡐 that is a lower case L by the way…it is NOT an uppercase i

(Below is a screenshot of what your screen should now look like…as it will have a list of the workpress files, etc.)



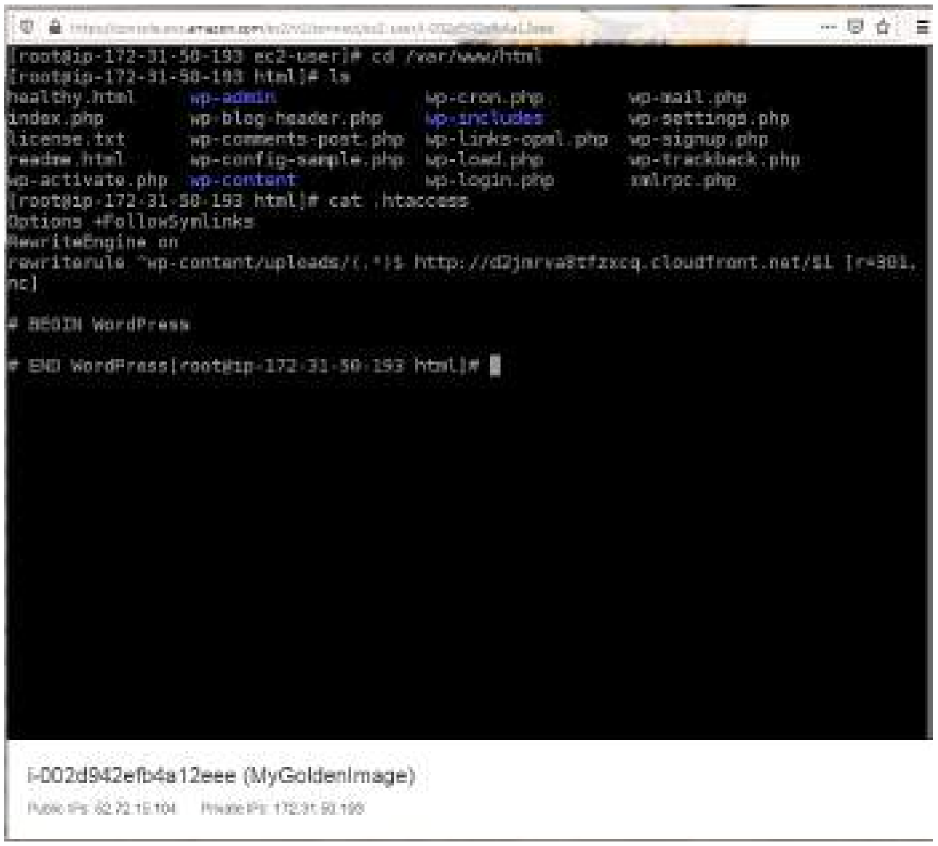
```
[root@ip-172-31-58-193 ec2-user]# cd /var/www/html
[root@ip-172-31-58-193 html]# ls
healthy.html     wp-admin              wp-cron.php           wp-mail.php
index.php        wp-blog-header.php    wp-includes           wp-settings.php
license.txt      wp-comments-post.php  wp-links-opml.php     wp-signup.php
readme.html      wp-config-sample.php  wp-load.php           wp-trackback.php
wp-activate.php  wp-content            wp-login.php          xmlrpc.php
[root@ip-172-31-58-193 html]# 
```

i-002d942efb4a12eee (MyGoldenImage)

Public IPs: 52.72.15.104   Private IPs: 172.31.58.193

Now type the following:

cat .htaccess (and hit enter) ←there is a space between cat and .htaccess



```
[root@ip-172-31-50-193 ec2-user]# cd /var/www/html
[root@ip-172-31-50-193 html]# ls
healthy.html     wp-admin               wp-cron.php        wp-mail.php
index.php        wp-blog-header.php     wp-includes        wp-settings.php
license.txt      wp-comments-post.php   wp-links-opml.php  wp-signup.php
readme.html      wp-config-sample.php   wp-load.php        wp-trackback.php
wp-activate.php  wp-content             wp-login.php       xmlrpc.php
[root@ip-172-31-50-193 html]# cat .htaccess
Options +FollowSymlinks
RewriteEngine on
rewriterule ^wp-content/uploads/(.*)$ http://d2jmrva8tfzxcq.cloudfront.net/$1 [r=301,
nc]

# BEGIN WordPress

# END WordPress[root@ip-172-31-50-193 html]#
```

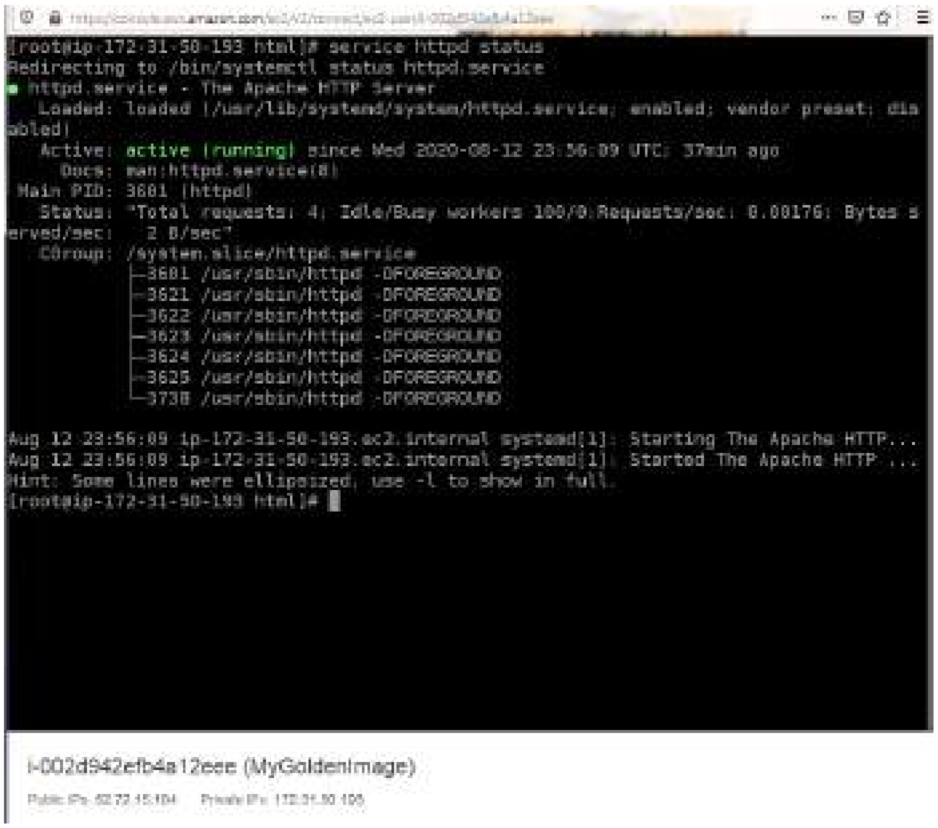
i-002d942efb4a12eee (MyGoldenImage)

Public IPs: 52.72.15.104   Private IPs: 172.31.50.193

Now type the following:

clear (and hit enter)

service httpd status (and hit enter) ←you will see that apache is active (running)



```
[root@ip-172-31-50-193 html]# service httpd status
Redirecting to /bin/systemctl status httpd.service
● httpd.service - The Apache HTTP Server
   Loaded: loaded (/usr/lib/systemd/system/httpd.service; enabled; vendor preset: dis
abled)
   Active: active (running) since Wed 2020-08-12 23:56:09 UTC; 37min ago
     Docs: man:httpd.service(8)
 Main PID: 3601 (httpd)
   Status: "Total requests: 4; Idle/Busy workers 100/0;Requests/sec: 0.00176; Bytes s
erved/sec:   2 B/sec"
   CGroup: /system.slice/httpd.service
           ├─3601 /usr/sbin/httpd -DFOREGROUND
           ├─3621 /usr/sbin/httpd -DFOREGROUND
           ├─3622 /usr/sbin/httpd -DFOREGROUND
           ├─3623 /usr/sbin/httpd -DFOREGROUND
           ├─3624 /usr/sbin/httpd -DFOREGROUND
           ├─3625 /usr/sbin/httpd -DFOREGROUND
           └─3738 /usr/sbin/httpd -DFOREGROUND

Aug 12 23:56:09 ip-172-31-50-193.ec2.internal systemd[1]: Starting The Apache HTTP...
Aug 12 23:56:09 ip-172-31-50-193.ec2.internal systemd[1]: Started The Apache HTTP ...
Hint: Some lines were ellipsized, use -l to show in full.
[root@ip-172-31-50-193 html]#
```

i-002d942efb4a12eee (MyGoldenImage)

Public IPs: 52.72.15.104   Private IPs: 172.31.50.193

Now type the following:

clear (and hit enter)

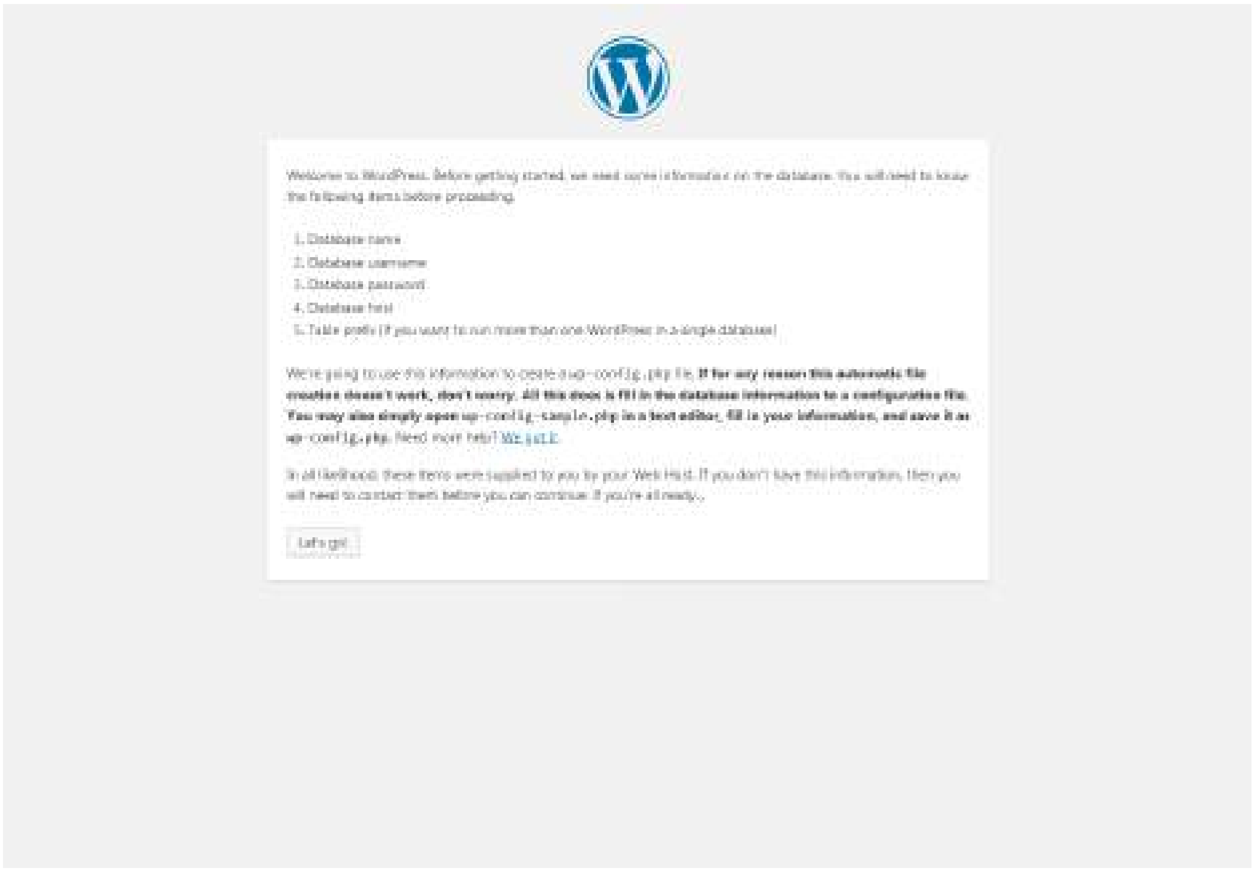
★★★Now close out of the terminal window.

Step 8) The WordPress Installation screen

Now return to the AWS console. Click Services and Click EC2. Then click Running instances. Select the instance we just created called MyGoldenImage. Scroll down and click the description tab. Within the description find the **IPv4 Public IP** for your instance and copy it into your clipboard. It will look something like the following: 3.91.18.25 ← yours will be different when you do this project

Paste that IPv4 Public IP into a new browser tab and hit enter.

We will see the WordPress Installation screen. Below is a screenshot of what you will see.



Welcome to WordPress. Before getting started, we need some information on the database. You will need to know the following items before proceeding.

1. Database name
2. Database username
3. Database password
4. Database host
5. Table prefix (if you want to run more than one WordPress in a single database)

We're going to use this information to create a `wp-config.php` file. **If for any reason this automatic file creation doesn't work, don't worry. All this does is fill in the database information to a configuration file. You may also simply open `wp-config-sample.php` in a text editor, fill in your information, and save it as `wp-config.php`.** Need more help? We got it.

In all likelihood, these items were supplied to you by your Web Host. If you don't have this information, then you will need to contact them before you can continue. If you're all ready…

Click the **Let's go** button

Database Name: wordpressproject

Username: wordpressproject

Password: wordpressproject

Database Host: 🡐★★★Here you must enter your RDS endpoint (see below to find out where it is)

Table Prefix: wp_ 🡐leave this as wp_ (it should already be defaulted as such)

★★★In the Database Host section you would have a local host if you were running MySQL on your EC2 instance. However, we are running MySQL on an RDS instance! So we need to enter in our RDS endpoint **instead** of localhost. To retrieve that endpoint, go to the AWS management console and click **Services**. Then click **RDS.** Then click **DB instances.** And then click on the RDS instance we created which is called **wordpressproject.** Then click the **connectivity & security** tab and copy the endpoint which will look something like the following:

wordpressproject.cgkclpiyi8qj.us-east-1.rds.amazonaws.com 🡐keep in mind your endpoint will be different

Click **Submit**

It will say sorry but I can’t write the wp-config.php file

And that you can create the wp-config.php file manually.

Step 9) Creating the wp-config.php file

So what you need to do is copy everything in the big text box which will contain something **similar** to the code below:

```
<?php
/**
 * The base configuration for WordPress
 *
 * The wp-config.php creation script uses this file during the
 * installation. You don't have to use the web site, you can
 * copy this file to "wp-config.php" and fill in the values.
 *
 * This file contains the following configurations:
 *
 * * MySQL settings
 * * Secret keys
 * * Database table prefix
 * * ABSPATH
 *
 * @link https://wordpress.org/support/article/editing-wp-config-php/
 *
 * @package WordPress
 */
// ** MySQL settings - You can get this info from your web host ** //
/** The name of the database for WordPress */
define( 'DB_NAME', 'wordpressproject' );
/** MySQL database username */
define( 'DB_USER', 'wordpressproject' );
/** MySQL database password */
define( 'DB_PASSWORD', 'wordpressproject' );
/** MySQL hostname */
define( 'DB_HOST', 'wordpressproject.cgkclpiyi8qj.us-east-1.rds.amazonaws.com' );
/** Database Charset to use in creating database tables. */
define( 'DB_CHARSET', 'utf8mb4' );
/** The Database Collate type. Don't change this if in doubt. */
define( 'DB_COLLATE', '' );
/**#@+
 * Authentication Unique Keys and Salts.
 *
 * Change these to different unique phrases!
 * You can generate these using the {@link https://api.wordpress.org/secret-key/1.1/salt/ WordPress.org secret-key service}
 * You can change these at any point in time to invalidate all existing cookies. This will force all users to have to log in again.
 *
 * @since 2.6.0
 */
define( 'AUTH_KEY',         'NuEjM)T]D_[O#*){LaIrqUW+>cPfsoLV3rYrZIoa+wXa^u.;vq5]ctH^AMakN]x@' );
define( 'SECURE_AUTH_KEY',  'QBcf@0#1?QV^q`O=eqgg?&TMFh<;Vfbb6:!sBpxM~n!G_N59r_hlU1*xUf?]hHgE' );
define( 'LOGGED_IN_KEY',    'C:oN/_e :h.]Zi6I[ Iukkc;9=a7(z>gkK}C5EUlSD7DMKf5:H9m!49pa4Nf:PaG' );
define( 'NONCE_KEY',        'O?&:mw+aUVUN5e}ZzCZQ<f,P2o23Cl+,CTGXDis+OF}=9_Y<D>Vw>P<j9iJU$:sC' );
define( 'AUTH_SALT',        'YND6cm`qH^+gv}LKk5RN7Cn[>?tNQc2$w]^-9JooQcsV.HEs,6{5[@[{Q*1HC]4E' );
define( 'SECURE_AUTH_SALT', 'u+(0S;zw0n Tb=eCIi21f`G4c4OfJZ7LY}V^4/s(+05IEd+ktwS/L4{*O2oQnst@' );
define( 'LOGGED_IN_SALT',   ')?U/SaZh;p+d#AtLWrW-Q.,Jz}y8M9.<M~XA~R?%A/}`/6X6`S))r[s`p0<sVyFY' );
define( 'NONCE_SALT',       'u8P Rex<>9#&Z@P+CtVaz}dbUI$%TLk5_UR?Cz`,P6$h)bri.Sv8lJZSS3-MA285' );
/**#@-*/
/**
 * WordPress Database Table prefix.
 *
 * You can have multiple installations in one database if you give each
 * a unique prefix. Only numbers, letters, and underscores please!
 */
$table_prefix = 'wp_';
/**
 * For developers: WordPress debugging mode.
 *
 * Change this to true to enable the display of notices during development.
 * It is strongly recommended that plugin and theme developers use WP_DEBUG
 * in their development environments.
 *
 * For information on other constants that can be used for debugging,
 * visit the documentation.
 *
 * @link https://wordpress.org/support/article/debugging-in-wordpress/
 */
define( 'WP_DEBUG', false );
/* That's all, stop editing! Happy publishing. */
/** Absolute path to the WordPress directory. */
if ( ! defined( 'ABSPATH' ) ) {
	define( 'ABSPATH', __DIR__ . '/' );
}
/** Sets up WordPress vars and included files. */
require_once ABSPATH . 'wp-settings.php';
```

Once you have copied all of that code into your clipboard return to the AWS console. Click **Services** and click EC2 and click **Running instances.** Select **MyGoldenImage** and then click **Actions** and click **Connect.** Select ✓EC2 Instance Connect (browser-based SSH connection)

And then click **Connect**

Type the following:

sudo su (and hit enter)

clear (and hit enter)

cd /var/www/html (and hit enter)

nano wp-config.php (and hit enter)

Then paste all of that php code that wordpress gave us into the wp-config.php that we just created

Then hit control x in order to exit and type y in order to save the file and then hit enter. Hit enter one more time for good measure.

★Then close the terminal window

Then go back to our Word press installation page and click **Run the installation**

(The Welcome page should prompt immediately! If it does not prompt then it could be that you haven't opened up the security group on MySQL port 3306)

So now we are ready to install wordpress it has already created our database inside RDS

Site Title: wordpressproject

Username: wordpressproject

Password: wordpressproject

✓Confirm use of weak password

Your Email: 🡐 enter the email address that you would like to use

Search Engine Visability: **uncheck** the box where it says Discourage search engines from indexing this site. ← just leave the box unchecked

Then click **Install WordPress**

Then click **Log In**

(Give it some time to load)

Username: wordpressproject ←type in your username

Password: wordpressproject ←type in your password

Select remember me (if you would like)

Then click **Login**

You will notice your url tab will say the IP address for MyGoldenImage and then /wp-admin/ So it will look something like this: 3.91.18.25/wp-admin/ ← yours will have a different IP address though when you do this project.

<u>**Step 10) Let's add our first post**</u>

Click **Posts** in the left margin

Click **Add New**

Title: I'm a Cloud Architect ←or just type anything you want

Screenshot the photo below and save the file on your computer as George-architect.jpg

(or save any jpg file that you want)

Underneath the Title click the **Add block** icon

Then click the Image icon (because we want to upload an image)

Click the **Upload** button and choose the George-architect.jpg file (or whatever jpg file that you want to upload)

Now screenshot the photo below and save it on your computer as Seal-architect.jpg (or save whatever jpg file that you want to upload)

Underneath the photo that you just uploaded into the body of your post, click on the add block icon

Then click the Image icon (because we want to upload another image)

Click the **Upload** button and choose the Seal-architect.jpg file (or whatever jpg file that you want to upload)

Now that you have added a title to your post and uploaded two jpg files, click the **publish button** and then click **publish** to confirm that you want to publish your post.

Once you have published your post go ahead and view your post

Now let's SSH back into our MyGoldenImage instance

Go to the AWS console and click **services** and click **ec2** and then click **running instances**. Select **MyGoldenImage** and then click **Actions** and click **Connect.** Select ✓EC2 Instance Connect (browser-based SSH connection)

And then click **Connect**

Type the following:

sudo su (and hit enter)

clear (and hit enter)

cd /var/www/html (and hit enter)

ls (and hit enter) ← this is a lower case l and not an uppercase i…ls is short for list

cd wp-content (and hit enter)

ls (and hit enter) 🡐 this is a lower case l and not an uppercase i…ls is short for list

cd uploads (and hit enter)

ls (and hit enter)🡐 (it will give us the current year which is 2020)

cd 2020 (and hit enter)

cd 08 (and hit enter) 🡐 ★★★08 is in reference to the month of the year. I am doing this in August (the 8th month of the year) Hence, I typed cd 08. So for example, if you are doing this project in September, then you would type in cd 09 because that is the ninth month of the year. Make sure to type the proper month!!!

ls (and hit enter) 🡐this will give you a list of the files that you uploaded to your post. It will list George-architect.jpg as well as Seal-architect.jpg (or the names of whatever files you chose to upload)

So these jpg files are now saved on our EC2 instance. They are in the cloud sitting on an EBS volume somewhere in a data center in N. Virginia.

Congrats on creating your site! In the next section of this lesson we will add resilience to your site!

Now let's add more resilience by typing following:

cd /var/www/html (and hit enter)

clear (and hit enter)

What we wanna do is make it so that everytime somebody uploads a file to our wordpress site, that file is also stored in S3 for redundancy. And eventually what we are going to do is we are going to force CloudFront to serve those files using the CloudFront distribution, rather than using the images on our EC2 instance, because then the site will load a bit faster. So the very first thing we want do is make sure our S3 role is working.

So now type the following in the terminal:

aws s3 ls and hit enter 🡐 this will display all of your s3 buckets (if not make sure the role is attached to your ec2 instance which gives u s3 access)

Next thing we are going to do is copy across those files

aws s3 cp --recursive /var/www/html/wp-content/uploads s3://wordpress-media-bucket-2020 (then hit enter) 🡐 ★to be clear cp has a space after it and then there are two hyphens before the word recursive. Also, my S3 media bucket is called wordpress-media-bucket-2020 but you will have named your media bucket something else. So in the command above, make sure to replace wordpress-media-bucket-2020 with the name of your S3 media bucket!!! The command we just typed means to copy all the files and folders within the directory called /var/www/html/wp-content/uploads to the S3 bucket that will be serving as our CloudFront origin. So once you have hit enter it will upload our jpg files to our S3 media bucket.

The next thing we wanna do is add redundancy. We could potentially lose our MyGoldenImage EC2 instance at any time, so what we want is a full copy of our website in our S3 code bucket. As you recall, I named my S3 word bucket wordpress-code-bucket-2020 but you will have named yours something else. So we are going to copy the entire /var/www/html directory into the our S3 code bucket. That way if we lose our EC2 instance, we can have an auto scaling group with a bunch of EC2 instances that pull down the code from the S3 Bucket as soon as they boot up. So in summary, we are going to copy the entire /var/www/html directory to our S3 code bucket.

Let's type the following into the command line:

aws s3 cp --recursive /var/www/html s3://wordpress-code-bucket-2020 (and hit enter) ←★Make sure to replace wordpress-code-bucket-2020 with the correct name of your code bucket. The command will copy all the files to our S3 Bucket and that will be done in just a couple of seconds. To be clear, there is a space after cp and then there are two hyphens before recursive.

So now we have a full copy of our WordPress site in S3

To test that, let's type the following into the command line:

aws s3 ls s3://wordpress-code-bucket-2020 (and hit enter) ← ★make sure to replace wordpress-code-bucket-2020 with the correct name of your s3 code bucket. The command will will show us all of our WordPress code.

Now type the following into the command line:

ls (and hit enter) ← Just to be clear, that is a lower case L and not an uppercase i. You will see that we have a file called healthy.html which we created with our bootstrap script.

cat healthy.html (and hit enter) ←just to make sure we have a file in there (once we hit enter it responds back with the word healthy which means we do have the file in there)

(so we can use our elastic load balancers to do the health check off of that healthy.html file.)

Now let's check on our .htaccess file (which is currently a hidden file)

Type the following into the command line:

nano .htaccess (and hit enter) ← Here we have what is called a URL rewrite rule and this essentially allows our website to serve our images out of CloudFront instead of serving the images out of EC2.

So your screen will say the following:

rewriterule ^wp-content/uploads/(.*)$ http://d2jmrva8tfzxcq.cloudfront.net/$1 [r=301,nc]

We want to replace the url above with our cloudfront domain name.

Go to the AWS console and click Services and Click CloudFront and then copy the domain name of your cloudfront distribution into your clipboard. Your CloudFront domain name will look something like this: d1y18ip3jbskfc.cloudfront.net 🡐 however yours will be different

★★★Now replace the d2jmrva8tfzxcq.cloudfront.net with the domain name of your cloud front which in my case is d1y18ip3jbskfc.cloudfront.net 🡐yours will be different.

So that line should look like the following:

rewriterule ^wp-content/uploads/(.*)$ http://d1y18ip3jbskfc.cloudfront.net/$1 [r=301,nc]

Now hit control x to exit and then just type y and hit enter to save the file (So that is now doing a URL rewrite to our CloudFront distribution)

So now we want to make sure that our S3 bucket is up to date. And we could copy the entire directory again or we could use the sync command.

So in the command line type the following:

clear (and hit enter)

aws s3 ls (and hit enter) 🡐this will list our buckets

aws s3 sync /var/www/html s3://wordpress-code-bucket-2020 (and hit enter) 🡐 And that will sync the one file that has changed. Don't forget your bucket name is NOT wordpress-code-bucket-2020. So make sure to enter your S3 code bucket name.

So you should now be able to see upload: ./.htaccess to s3://wordpress-code-bucket-2020/.htaccess 🡐keep in mind your S3 code bucket name is going to be different than wordpress-code-bucket-2020.

So now that the file is up to date.

We have done our URL rewrites and we've synced up our http access file to S3, it's still not gonna work becuz we have to tell apache that we are going to allow url rewrites.

Type the following into the command line:

clear (and hit enter)

cd /etc/httpd (and hit enter) ←this is where apache is installed
ls (and hit enter) ← this is a lower case L by the way and **not** an uppercase i
cd conf (and hit enter)
ls (and hit enter) ← We can now see our httpd.conf configuration file and since we are going to make a change to this file, it is ALWAYS wise to make a backup copy before we do that.
cp httpd.conf httpd-copy.conf (and hit enter) ←that will just create a backup file
ls (and hit enter) ← this will list the files including the backup file we just created
nano httpd.conf (and hit enter)

★★★Now what we are going to do is allow url rewrite rules...there are 362 lines and so u are going to need to scroll down on your screen and find where it says the following:

```
#AllowOverride controls what directives may be placed in .htaccess files.
#It can be "ALL", "None", or any combination of the keywords:
#   Options FileInfo AuthConfig Limit
#
AllowOverride None
```

★★★Instead of AllowOverride None, change it to say AllowOverride All

(By changing it to say AllowOverride All, it means we are allowed url rewrites. And a url rewrite will just rewrite our URL to be using the CloudFront distribution instead of using our public IP address forward slash wpcontent forward slash uploads, etc.)

Now hit control x to exit and hit Y to save it and hit enter. So we have just successfully written that file the way we want it.

Since we changed the configuration var, let's restart the service even though our change should have taken effect immediately.
Type the following into the command line:
service httpd restart (and hit enter)

Now close the terminal window.

Now go to the AWS console, then click **S3**, then click on your media bucket. As you recall mine is called wordpress-media-bucket-2020 but you will have named yours something different. Then click the **permissions** tab and then click the **Block public access** tab. Then click **Edit** and then **uncheck** the box where it says **Block all public access**. Make sure all the boxes are unchecked because we do want public access. Then click **Save**. Then type confirm. And then click **Confirm**.

Now click the **Bucket Policy** tab and paste in the following JSON code:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "PublicReadGetObject",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "s3:GetObject",
      "Resource": "MY ARN HERE/*"
    }
  ]
}
```

★★★Now on line 9 of the above code make sure to replace where it says MY ARN HERE with the actual ARN for your media bucket bucket. You can find the ARN for your bucket just to the right of where it says Bucket policy editor. My media bucket is called wordpress-media-bucket-2020. So my JSON code on line 9 will look like the following:

```
"Resource": "arn:aws:s3:::wordpress-media-bucket-2020/*"
```

★★★Your ARN will be different since your bucket name will be different, so make sure to change line 9 of your JSON code accordingly

Once you have added the appropriate JSON code click **Save**
Once you hit save, it will say Public on the permission tab as well as on the Bucket Policy tab.

We are now good to go. However, it will take CloudFront some time to propagate. (Give it 10 minutes at least) Now let's test to see that our CloudFront distribution is distributing our files.

Go to the AWS console and click CloudFront. Then copy the Domain Name of your CloudFront distribution into your clip board and paste it into a new url tab but do **NOT** press enter yet! At the end of the domain name type /2020/08/george-architect.jpg
Altogether it will look something like the following:

d1y18ip3jbskfc.cloudfront.net/2020/08/george-architect.jpg ←★★★of course when you do this project your cloudfront domain name is going to be different and your year and month may very well be different. Also, you may have saved your jpg image as something else.

<u>Now Let's create an Application Load Balancer</u>

We will move our EC2 instance behind an application load balancer

Go back to the AWS management console. Then click **EC2.** Then click **Load balancers** in the left margin. Create **Load Balancer**.

Then click **Create** an Application Load Balancer.
Name: ALBforWPsite ← named it my ALBforWPsite
Scheme: ✓internet-facing ←select internet-facing
IP address type: ✓ipv4 ←select ipv4

Load Balancer Protocol	Load Balancer Port
HTTP	80

VPC: ✓vpc-fe674d98 (172.31.0.0/16) (default) 🡐 select the default VPC, your numbers & letters will look different

Availability Zones: ✓select all six availability zones (if you did this project in another region you might have fewer availability zones. So just select all of them. We want use of as many AZ's as possible.)

Click Next: Configure Security Settings
Then just click next: configure security goups

Select an existing security group

✓WebServerSGforWPsite🡐select the security group we created for our web server. This is the only securitygroup we want selected so make sure all others are unchecked.

click **Next: Configure Routing**

Target group: ✓New target group 🡐 select New target group

Name: ✓WordPressInstances 🡐 type in a name for the target group such as WordPressInstances

Target type ✓Instance 🡐 select Instance as the target type
Protocol: ✓HTTP 🡐 select HTTP
Port: ✓80 🡐 select 80

Health Checks
Protocol: ✓HTTP
Path: /healthy.html 🡐 type /healthy.html as the path

Click **Advanced health check settings**
Port ✓traffic port 🡐 select traffic port
Healthy threshold: 2 🡐 type 2 into the box
Unhealthy threshold: 3 🡐 type 3 into the box
Timeout: 5 seconds 🡐 type 5 into the box
Interval: 6 seconds 🡐 type 6 into the box
Success codes: 200 🡐 type 200 into the box

Click Next: Register Targets
Scroll down to the **Instances** section
Select ✓MyGoldenImage
Then Click **Add to registered**
Click **Next: Review**
Click **Create**

(Wait for the state of your Load Balancer to say **state: active** and then proceed. Hit refresh after a few minutes to check on it)

Optional Step
Now let's go to Route 53. (If you have a domain name you want to use or if you want to purchase one to use go ahead and do so)

Click **Services** Click **Route 53** then click **Hosted zones**. Then click **praysmr.com** (which is the domain name that I have already purchased…yours of course will be different…or feel free to skip this step if you do not wish to use a domain name that you purchased)

Now click Create record
Routing policy: ✓Simple routing
Then click **Next**
Click Define simple record
Record Name: just leave as the naked domain name so don't type anything into the box

Value/Route traffic to: ✓Alias to Application and Classic Load Balancer ← select this
Choose Region: US East (N. Virginia)[us-east-1] ← assuming you did the project in N. Virginia
Choose load balancer: ✓Then select the load balancer that we just created. It will look something like the following dualstack.ALBforWPsite-1049975346.us-east-1.elb.amazonaws.com but of course yours will be a bit different.

Record type: ✓A – Routes traffic to an IPv4 address and some AWS resources ← select A which is short for Alias

Evaluate target health: ✓No ← select no to keep things simple

Click **Define simple record**

Then scroll down and click **Create Records** (That will add the A record that we just created to the existing records. Don't touch those existing records by the way)

DNS will take some time to propagate so be patient.

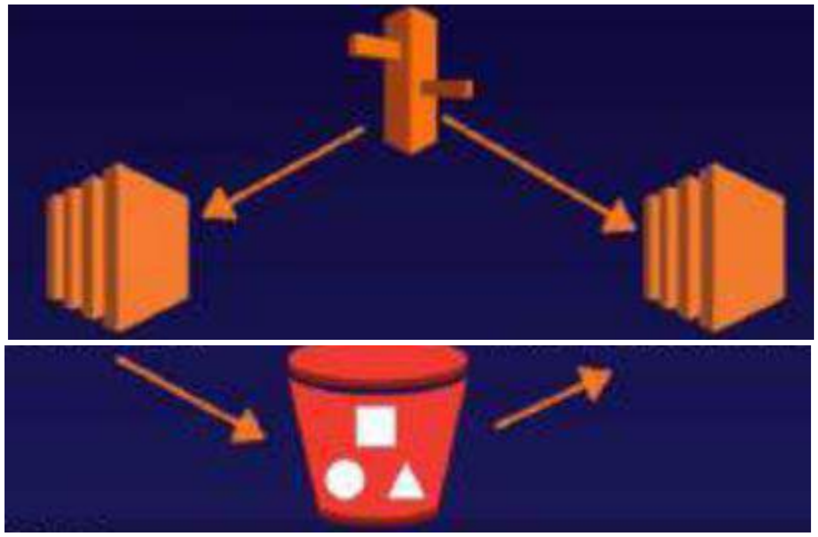
Let's check to make sure that MyGoldenImage is a registered target for our Target group called WordPressInstances

Click **Services**. Click **EC2**. Click **Target Groups** in the left margin. Click **WordPressInstances.** Then click the **Targets** tab and MyGoldenImage should be a registered target (if not then add it as a registered target)

Now in a new URL tab. Type the name of your domain. In my case, it is praysmr.com
Then hit enter. Our wordpress site should now be visible!

Right click on one of your jpg images and click Copy image address
Then paste that address into a new url tab and hit enter. You will notice that it rewrites to cloudfront.

Let’s add resilience and autoscaling.
The architecture will look as follows:


52.201.160.242 http://praysmr.com

The ip address is the ip of our current ec2 instance (of course yours will be different) This is going to be our writer node. So hypothetically every time the “marketing team” wants to go and write a new blog article for our WordPress site, they are going to navigate directly to this IP address (which is the ip on the left side of the above diagram), this ec2 instance will be configured to push any changes to S3 (see the bucket at the bottom of the diagram)

And then we are going to have a fleet of ec2 instances and their job is to pull the S3 bucket looking for changes (see the right hand side of the diagram)

When people visit our domain, praysmr.com (see the very top of the diagram), then route 53 is going to send them to the fleet of ec2 instances (see right side of the diagram), it is **NOT** going to send them to our writer node, it is just going to send our visitors to the read nodes only.

Let’s proceed with setting this up.

Click **Services** and click **EC2** and then click **running instances**.
Select MyGoldenImage and click Actions and then click **Connect**.

Connection Method: ✓ EC2 instance Connect (browser-based SSH connection) ←select this
Then click **Connect**

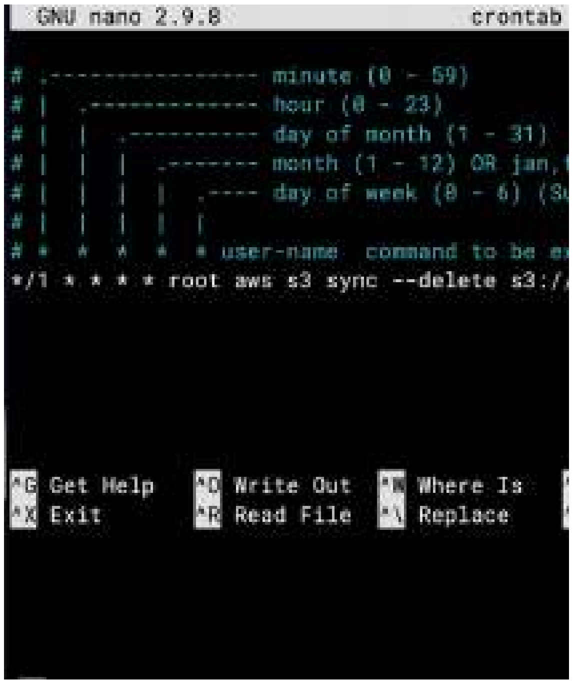
We are now SSH’d into our instance. Now type the following into the command line:
sudo su (and hit enter)
clear (and hit enter)
cd /var/www/html (and hit enter) ← there is a space between cd and /var/www/html
cd /etc (and hit enter) ← there is a space between cd and /etc
nano crontab (and hit enter) ← we are going to edit the crontab file. The Crontab is the scheduled tasks for windows. It is a way of running commands at a particular time (every minute, every hr, every day, every month, etc.

★On the line directly underneath where it says **# * * * * * user-name command to be executed** type the following command:

*/1 * * * * root aws s3 sync --delete s3://wordpress-code-bucket-2020 /var/www/html (and hit enter)

★In case it is hard to see, in the command above there are two hyphens before the word delete. Also, there is a space before /var/www/html There is also a space after */1   and there is a space after each subsequent asterisk.
★Also, In the command above my bucket is called wordpress-code-bucket-2020, but yours will be named something else. So in the command above make sure to enter your s3 code bucket name instead of mine.

(see the below screenshot for reference)



```
  GNU nano 2.9.8                              crontab

# .---------------- minute (0 - 59)
# |  .------------- hour (0 - 23)
# |  |  .---------- day of month (1 - 31)
# |  |  |  .------- month (1 - 12) OR jan,
# |  |  |  |  .---- day of week (0 - 6) (Su
# |  |  |  |  |
# *  *  *  *  * user-name  command to be e
*/1 * * * * root aws s3 sync --delete s3:/




^G Get Help    ^O Write Out   ^W Where Is
^X Exit        ^R Read File   ^\ Replace
```

The first star means the command is going to run every minute. Then we want four more stars because we want it to run every minute of every hour of every day of the month of every month as well as every day of the week. We want it to run at root level. It is going to run an aws command. We are going to be using s3. And what we are going to be doing is we are going to be syncing to S3. And we want this to be a perfect copy. So we want it to delete any files off of our EC2 instance that are not in our bucket. So we always want this to be a perfect copy of our bucket. Therefore, we are going to sync our s3 code bucket to our /var/www/html directory.

So the command is going to look for any changes in s3 and then it will go ahead and download those changes. If we have deleted files in s3 it will also delete the files from our ec2 instance, which is why we used the delete marker.

Now hit ctrl and x together. And then type y and hit enter, which will save the change we made to our crontab file.

Now we are back at our etc directory in the terminal.
Type the following into the command line:
service crond restart (and hit enter) 🡐 crond is not a typo in case u were wondering.
Now close the terminal window.

Now let's test what we have done.
Create a very simple text file using a text editor such as notepad.
Just type the word Test
And then save the file on your computer as test.txt or something similar.
Then go to the AWS console and click **Services** and then click **S3**.
Then click your S3 code bucket. Mine is called wordpress-code-bucket-2020, but yours has a different name.
Then click **Upload.** Then click **Add files.** Then choose the file called **test.txt** and then **upload** the file.

Click **Services** and click **EC2** and then click **running instances**.
Select MyGoldenImage and click Actions and then click **Connect**.

Connection Method: ✓ EC2 instance Connect (browser-based SSH connection) 🡐select this
Then click **Connect.**

Now type the following into the command line:

sudo su (and hit enter)
clear (and hit enter)
cd /var/www/html (and hit enter) 🡐 there is a space between cd and /var/www/html
cd /etc
service crond restart (and hit enter) 🡐(restarting the crond service will automatically just rerun all the commands we are basically just forcing the issue here instead of waiting for the commands to run.)

cd /var/www/html (and hit enter)
ls (and hit enter) 🡐 this is a lowercase L and not an uppercase i. You will see the test.txt file is listed.
cat test.txt (and hit enter) 🡐it will respond back with test which is the text we added to the file.

Now close out of the terminal window

Now let's create an AMI (Amazon Machine Image) This is going to serve as the golden template for our Wordpress Service in our autoscaling group.

Go to the aws console. Click **Services**. Click **EC2** and click **running instances**

Select **MyGoldenImage** (the name of our ec2 instance) then click **Actions** and then click **Image** and then click **Create Image**

Instance ID: i-0bfc87641abac78e8 ←it will look something like this. But yours will be different of course.
Image name: WordPressReadNode ← name it like so
Image Description: This is the default read node for WordPress ← type this
No reboot ← ✓ check this box

Leave the Root Volume as it is.
Scroll down and click **Create Image**

Now click **AMIs** in the left margin
And check that the AMI that we created says Status: Available

Click **Services** and click **EC2** and then click **running instances**.
Select MyGoldenImage and click Actions and then click **Connect**.

Connection Method: ✓ EC2 instance Connect (browser-based SSH connection) ←select this
Then click **Connect.**

Now type the following into the command line:

sudo su and hit enter
cd /etc (and hit enter) ← this takes us to our etc directory
clear (and hit enter)
nano crontab (and hit enter)

★★★We need to now **change** this line:
*/1 **** root aws s3 sync --delete s3://wordpress-code-bucket-2020 /var/www/html

★★★And instead have it look like this:
*/1 **** root aws s3 sync --delete /var/www/html s3://wordpress-code-bucket-2020

★★★And we will also add a line below that:
*/1 * * * * root aws s3 sync --delete /var/www/html/wp-content/uploads s3://wordpress-code-bucket-2020

★★★So just to be clear … we altered the first command and then added a second command

★ So underneath the line that says **# * * * * * user-name command to be executed** it will now look like:
*/1 **** root aws s3 sync --delete /var/www/html s3://wordpress-code-bucket-2020
*/1 * * * * root aws s3 sync --delete /var/www/html/wp-content/uploads s3://wordpress-media-bucket-2020

★ The first command is for our write node as it is the reverse of what we did before (which was for our read node)
★The second command syncs to our S3 media bucket, so that if any images are uploaded to this EC2 instance it is going to be distributed throughout CloudFront as well.
★Make sure to put the correct names of your s3 buckets as yours are different than mine!!!

Now hit ctrl and x together
Then type y and hit enter

Now that we are back to the regular terminal command line type the following:

cd /var/www/html (and hit enter)
echo "This is another TEST" > anothertest.txt (and hit enter) ←there is a space after " and a space after >
service crond restart (and hit enter) ←this is not necessary but forces the issue
service httpd status ← check that apache is running since we just did a reboot

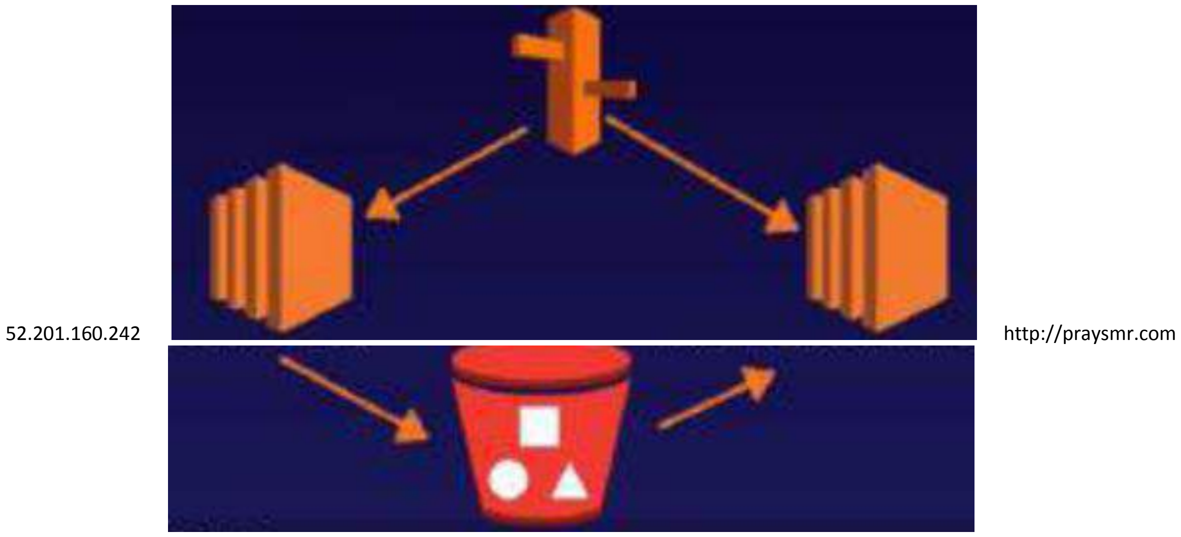
Now let's go back to S3 and see if our anothertest.txt file has been sent to our S3 bucket

Click Services then click **S3** then click into your S3 code bucket. Mine is of course called wordpress-code-bucket-2020 but yours you will have named something different.

And there we should see the anothertest.txt file that we just created

---

Here is a reminder of the architecture of our site



★We want it to have one write node (see the far left of the diagram) and this is basically a single ec2 instance that our blogging team will write articles to, and that's what our EC2 instance (MyGoldenImage) is right now. The cron command that we **just** set up is taking all the local assets and it is pushing it and syncing it to S3 every minute. So if someone goes into this EC2 instance and they do a new blog, it will automatically be pushed out to S3 within a minute. (You might not want to put the ip for our write node behind a domain name. If you do you would want to put it behind something like write.praysmr.com because you definitely don't want the general public trying to access this ec2 instance. You want to keep it as your master writer node (which is represented by the ip at the left side of our diagram)

★So what we did earlier with the **first** cron tab or cron job that we setup, we actually gave our ec2 instance (represented by the ip on the left side of our diagram) the ability to **pull** files from s3, and then we took an AMI of the EC2 instance having given it that ability!!! So **that** is the AMI that we are now going to use to build out the portion of our website that is represented by the right side of our diagram. We are going to use **that** AMI and put it into an autoscaling group and that autoscaling group is going to sit behind our application load balancer, and then when people go and visit our website, route 53 (which is represented by the top of the diagram) is going to send them to our application load balancer and then it's going to go on to our EC2 instances that are behind that (referring to the right side of the diagram) and those ec2 instances (represented by the right side of the diagram) will be pulling S3 every minute trying to download a local copy of their file.

Let's return to the AWS console. Click **Services** and click **EC2**. Click **Launch Configurations** which is located in the left margin. Click **Create launch configuration**

Name: WordPressLaunchConfiguration ← Give it a name like so

AMI: ✓WordPressReadNode ← select the AMI that we created earlier called WordPressReadNode

Instance type: ✓t2.micro ← select t2.micro and then click **Choose**

IAM instance profile: ✓S3ForWPsite ←select S3ForWPsite, which is the IAM role that we created earlier. This is a very important step because we need our instances to have access to S3.

Click **Advanced details**
★In the User data section copy and paste or manually type the following bootstrap script as text:

```
#!/bin/bash
yum update –y
aws s3 sync --delete s3://YOUR_S3_BUCKET_NAME /var/www/html
```

★★★On the third line of the bootstrap script, make sure to replace where it says YOUR_S3_BUCKET_NAME with the name of your S3 code bucket!!! As you recall my bucket is **wordpress-code-bucket-2020**

So my bootstrap script will look like the following:

```
#!/bin/bash
yum update –y
aws s3 sync --delete s3://wordpress-code-bucket-2020 /var/www/html
```

★★★Also just to be clear, there is a space before /var/www/html

The bootstrap script is syncing and using the delete marker. It is going to sync s3 to your /var/www/html directory. As soon as your ec2 instances are launched in an autoscaling group, it is going to run a yum update, and then it's going to download the latest from your s3 code bucket.

IP address type: ✓Only assign a public IP address to instances launched in a subnet with auto-assign public IP enabled (default) ← choose Only assign a public IP address to instances in a subnet with auto-assign public IP enabled (default)

EBS volumes: Leave the storage as default because all we need is 8 GiB and a general purpose (SSD) volume type. So no need to click anything here. Just leave it as is.

Security Groups:
✓Select an existing security group

✓WebServerSGforWPsite ← select the security group that we created in the beginning of this project. It is the **only** security group that you want to have selected here.

Key pair options:

✓Choose an existing key pair
✓WordPressSiteKeyPair ← select the key pair that we used before
✓I acknowledge that I have access to the selected private key file

Then click **Create launch configuration**

So now our launch configuration has been created. Now we can create our auto scaling group.
Click **Auto Scaling Groups** in the left margin. Then click **Create Auto Scaling Group**.

Auto Scaling group name: WordPressReaderNodes ← name it like so
Then click Switch to **launch configuration**
Launch configuration: ✓WordPressLaunchConfiguration ← select the configuration that we recently created

Then click **Next**

VPC: ✓Default ← leave this as the default VPC
Subnets: ✓us-east-1a
✓us-east-1b
✓us-east-1c
✓us-east-1d
✓us-east-1e
✓us-east-1f

Select **every** subnet. N. Virginia has 6 subnets, so select **all** of them. Other regions have less. If you are doing this project in another region than just select **all** the subnets that there are available.

Then click **Next**

Load balancing:
✓Enable load balancing ← select Enable load balancing
✓Select Application Load Balancer or Network Load Balancer ← select this since we have an application load balancer that we want to utilize.

Choose a target group for your load balancer: ✓WordPressInstances ← select the target group that we created earlier called WordPressInstances

Health check type: ✓ELB ← select ELB because we want our ELB to be monitoring
Health check grace period: 60 seconds ← choose 60 seconds

Then click **Next**

Group size:
Desired capacity: 2 ← select 2
Minimum capacity: 2 ←select 2
Maximum capacity: 2 ←select 2
(So the minimum group size is 2 and the maximum group size is 2 and the desired group size is 2)

Scaling policies: ✓**None** ← select none ⋯ this is if we wanted to autoscale based on cpu utilization. So we will leave it as none.

Instance scale-in protection
Uncheck the box next to Enable instance scale-in protection. We do not need it for this project.

Click **Next**

We do not need to add notifications so click **Next**
Tags:

Key: Name ← type Name as the key
Value: WP-ReaderNode-ASG ← type WP-ReaderNode-ASG

Then click **Next**
Then click **Create Auto Scaling Group**

Now click Target Groups in the left margin
We now want to take our write node out of that target group...and the reason for that is we don't particularly want it getting read traffic. (Also, keep in mind that we have already connected up our domain name (mine is praysmr.com) to Route 53)

Click the **WordPressInstances** target group
Then click the Targets tab
✓Select **MyGoldenImage** (aka our writer node) and click **Deregister**
(So now our only registered targets are our two reader nodes)

Click **Services,** click **EC2** and you can see our three instances. MyGoldenImage (aka our writer node) as well as two nodes called WP-ReaderNode (which are our reader nodes)
Disclaimer: If your writer node shares an availability zone with one of your reader nodes this is just fine for our project however, one way to have prevented this would have been to put the writer node in it's own autoscaling group. Just food for thought.

Click on the name section of MyGoldenImage and change the name to WP-WriterNode just so it is even more clear as to what this instance is.

Type your domain name into a new url and hit enter. (Mine is praysmr.com but yours will be different assuming you opted to buy a domain)

If the page loads then it means the reader nodes are working.

Now at the end of your domain name add /wp-admin/ (and hit enter)
Mine looks like praysmr.com/wp-admin/

Upon pressing enter, the ip of the writer node will show up in the url tab which lets us know that this is our writer node.

Username: wordpressproject ← enter your username
Password: wordpressproject ← enter your password

Then click **Log in**

Click **Posts**
Then click **Add New**

Title: This is a Test ←or just type anything you want

Screenshot the photo below and save the file on your computer as wordpress-test.jpg

(or alternatively just save any jpg file that you want)

Underneath the Title click the **Add block** icon

Then click the Image icon (because we want to upload an image)

Click the **Upload** button and choose the WordPressTest.jpg file (or whatever jpg file that you want to upload) ★★★ and be advised once you select the file chances are it will say:

This image has an empty alt attribute; its file name is wordpress-test.jpg

So upon uploading the image there seems to be an issue...however, a url rewrite is writing to cloudfront automatically, but it is going to take some time for this image to propagate through cloudfront it might take up to ten minutes at least so be patient!!! ★★★Regardless click **Publish** and then click **publish** (to confirm that you want to publish)

★★★Now take a 10 min break and come back and do a refresh and if still not there come back a bit later (because CloudFront and DNS propogation do take time)

Now in the url tab type the name of your domain (mine is praysmr.com) and hit enter and you can see the post there.

So it is definitely being URL rewritten to our CloudFront distribution

Mission Accomplished: we now have a highly available and fault tolerant website.

Now let's test the resiliency. In the AWS console click **Services** and then click **EC2**.

Click **running instances**.

Select one of our two Reader Nodes then click **Actions** and then hover over **instance state**

and then click **Terminate.** And then click **Yes, Terminate.**

Then wait for the instance state to say terminated.

Now click Target groups in the left margin. Then click **WordPressInstances** (which is the name of our target group) Then click the targets tab and you will see we now only have one registered target.

Open up a new url tab and go to your domain name again. (Mine is praysmr.com) and it is still serving our content even though we lost one reader node.

Now return to the AWS console and click **Services** and click **EC2.** Click **Auto Scaling Groups** in the left margin. Select WP-ReaderNode-ASG (which is the name of our Auto scaling group and then click the **activity** tab. You will see there that one of our instances was terminated and that it is generating a new instance to take it's place. Click **EC2 Dashboard** in the left margin. Then click **Running instances.** You will now see that a new instance is now running and has taken the place of the terminated instance.

Click **Target Groups** in the left margin. Then click **WordPressInstances** which is the name of our target group. Then click the Targets tab and you will see that a new instance is a registered target and took the place of the one that terminated.

Open a new url tab and test your domain name again. (Mine is praysmr.com) And you will see that it loads just fine including our latest post which as you recall we created after we took our AMI. So our AMI had already been deployed in yet it was able to go to S3 and download our latest post. So we have a self-healing WordPress site!

Now let's simulate a failure of our RDS instance.

In the AWS console click **Services** and then click **RDS.** Click **DB instances**.

We will force a failover by doing a reboot.

✓Select our DB instance named wordpressproject

and then click the **Actions** button and then click **Reboot**

✓Select Reboot with Failover and then click **Reboot**

The status will now say Rebooting and this will take your website down for about maybe one or two minutes because you are failing over from one availability zone to another. So if you visit your website during this time you might get a 504 Gateway Time-out because it is in the process of failing over.

★★★ Now Clean Up. Delete all of the resources your created. First Delete your RDS instance. Then delete your autoscaling group. Although, first you have to remove your registered targets from your target group and then delete your application load balancer. Then delete your target group. Then go to ec2 and you will see that we only have one instance left and that is our writer node. Go ahead and delete that. Then delete your S3 buckets. Then disable your CloudFront Distribution and delete your CloudFront distribution. It take a few minutes to disable it so be patient. Delete your CloudFront Distribution.

AMI Cheat Sheet

-Amazon Machine Image (AMI) provides the information required to launch an instance

-AMIs are region specific, if you need to use an AMI in another region you can copy an AMI into the destination region via **COPY AMI**

-You can **create an AMI** from an existing EC2 instance that's either **running** or **stopped.**

-Community AMI are free AMIs maintained by the community

-AWS Marketplace free or paid subscription AMIs maintained by vendors

-AMIs have an **AMI ID**. The same AMI (such as an Amazon Linux 2) will vary in both AMI ID and options (such as architecture options) depending on the region. So they are not exactly the same in the different regions.

-An AMI holds the following information:

> **-**A template for the root volume for the instance (EBS Snapshot or Instance Store template) e.g. an operating system, an application server, and application data.
>
> **-**Launch permissions that control which AWS accounts can use the AMI to launch instances.
>
> -A block device mapping that specifies the volumes to attach to the instance when it's launched.

EC2 Auto Scaling Groups Cheat Sheet

-An ASG is a collection of EC2 instances grouped for scaling and management

-Scaling Out is when you add servers

-Scaling In is when you remove servers

-Scaling Up is when you increase the size of an instance (e.g. updating Launch Configuration with larger size)

-Size of an ASG is based on a **Min, Max,** and **Desired Capacity**

-Target Scaling policy scales based on when a target value for a metric is breached e.g. Average CPU utilization exceeds 75%

-Simple Scaling policy triggers a scaling when an alarm is breached.

-**Scaling Policy with Steps** is the new version of Simple Scaling Policy and allows you to create steps based on escalation alarm values

-Desired Capacity is how many EC2 instances you want to ideally run

-An ASG will always launch instances to meet minimum capacity

-Health checks determine the current state of an instance in the ASG

-Health checks can be run against either an ELB or the EC2 instances

-When an Autoscaling Group launches a new instance it uses a Launch Configuration which holds the configuration values for that new instance (such as the AMI, instance type, role)

-Launch Configurations cannot be edited and must be cloned or a new one created

-Launch Configurations must be manually updated by editing the Auto Scaling Group Settings

---

ELB Cheat Sheet

-There are three Elastic Load Balancers: **Network, Application** and **Classic** Load Balancer

-A Elastic Balancer must have **at least two** Availability Zones.

-Elastic Load Balancers **cannot go cross-region**. You must create one per region.

-ALB (Application Load Balancer) has **Listeners, Rules** and **Target Groups** to route traffic.

-NLB (Network Load Balancer) use **Listeners**, and **Target Groups** to route traffic

-CLB (Classic Load Balancer) use **Listeners** and EC2 instances are **directly registered** as targets to CLB

-Application Load Balancer is for HTTP(S) traffic and as the name implies it is good for Web Applications

-Network Load Balancer is for TCP/UDP is good for high network throughput (such as for video games)

-Classic Load Balancer is legacy and its recommended to use ALB or NLB

-Use X-Forwarded-For (XFF) to get original IP of incoming traffic passing through the ELB

-You can attach Web Application Firewall (WAF) to ALB but not to NLB or CLB

-You can attach Amazon Certification Manager SSL to any of the Elastic Load Balancers for SSL

-ALB has advanced Request Routing rules where you can route based on subdomain header, path and other HTTP(S) information

-Sticky Sessions can be enabled for CLB or ALB and sessions are remembered via Cookie

---

Question

You've been tasked with replicating your production VPC in another region for disaster recovery purposes. Part of your environment relies on EC2 instances with pre-configured software. What steps would you take to configure the instances in another region?

A) None of these.
B) Create AMIs of the instances and deploy them in the new Region
C) Create AMIs of the instances and copy them to the new Region for deployment.
D) Write the IAM permissions for the new Region to use the AMIs from the original Region.

Explanation:
The AMIs must be copied to the new Region prior to deployment.
Resources
Answer: C

Question

You are creating an RDS database for your production environment and it needs to be highly available and continue to function in the event of an outage to the Primary database. Which of the following options will best meet this requirement?

A) Multi-AZ deployment
B) Multi-region deployment
C) Cross-region deployment
D) Read replicas

Explanation:
Multi-AZ deployment involves the creation of a standby replica in a different Availability Zone (AZ) from the primary database. A standby replica cannot serve read traffic, it is used to synchronously replicate data from the primary database. AZs are isolated from one another to prevent failure from spreading to them all. So, if the location of the primary database has issues, Amazon RDS automatically fails over to the standby replica. Read replicas are used to scale out to cater for high volumes of read requests - not automated failover. Multi-region deployment is not a valid RDS option and Cross-region deployments enable support for scaling of Read replicas and can be used for cross-region DR, but don't support the automatic failover due to a Primary DB outage.
Answer: A

Question

You have a 3-tier application that you want to deploy into AWS - this application is accessed by users around the world over the Internet. The design calls for an Application Load Balancer, EC2 Instances for the application software and another set of EC2 Instances to run the custom relational database system for the application. Also, periodically you want the instances to be able to download updates from the internet, via an already-deployed NAT gateway & Internet Gateway in the public subnet. Which of the below deployments would you recommend, keeping in mind that you want to keep costs and complexity to a minimum?

A) Place the ALB in a private subnet inside the VPC. Deploy the application EC2 instances into the same private subnet, and the EC2 instances required for the database into a different private subnet
B) Place the ALB in a private subnet inside the VPC. Deploy the application EC2 instances into a different private subnet, and the EC2 instances required for the database into a the same subnet as the application instances
C) Place the ALB in a public subnet inside the VPC. Deploy the application EC2 instances into the same public subnet, and the EC2 instances required for the database into a private subnet
D) Place the ALB in a public subnet inside the VPC. Deploy the application EC2 instances into a private subnet, and the EC2 instances required for the database into a different private subnetSelected

---

Explanation:
Placing the ALB in a private subnet would generally make it inaccessible to users outside your organization, so these two options can be discounted. Of the remaining two options, although both could work in theory, one has you deploying the application servers into the same public subnet as the ALB - this would mean having to attach a public IP to them in order to allow them to download updates, or using some custom routing at the OS which increases complexity. The recommended architecture is to deploy the ALB into the public subnet, and the application & database tiers into different private subnets.
Answer: D