**Highly Available Web Application Workshop**

In this workshop, we will explore how to configure AWS VPC, Amazon Aurora RDS, Amazon EC2, Amazon EFS, and Amazon ElastiCache to build a highly available, auto-scaling multi-tier web application.

In this series of labs you will create a highly available, scalable deployment of the Wordpress web application. You will deploy the Wordpress application in such a way that the application server, database, and file server can scale independently of one another. You will also deploy the application’s components into two availability zones to distribute it and guard against failure of any one availability zone. The Wordpress application will be deployed in a stateless fashion so that we can add or remove web application servers in response to the requests flowing into the system.

To create this scalable, HA web application you will use various AWS services in a series to create an architecture similar to the full scale reference architecture linked below. You will first create a virtual network spread across multiple availability zones in your region of choice. You will then deploy a highly available relational database across those availability zones using Amazon RDS.

With your database deployed the next step will be to create a database caching layer using Amazon ElastiCache. This will provide you with a cache around your database for frequently run queries, improving HTTP response time performance and reducing strain on your RDBMS. With the data tier created you will then begin to create the application tier.

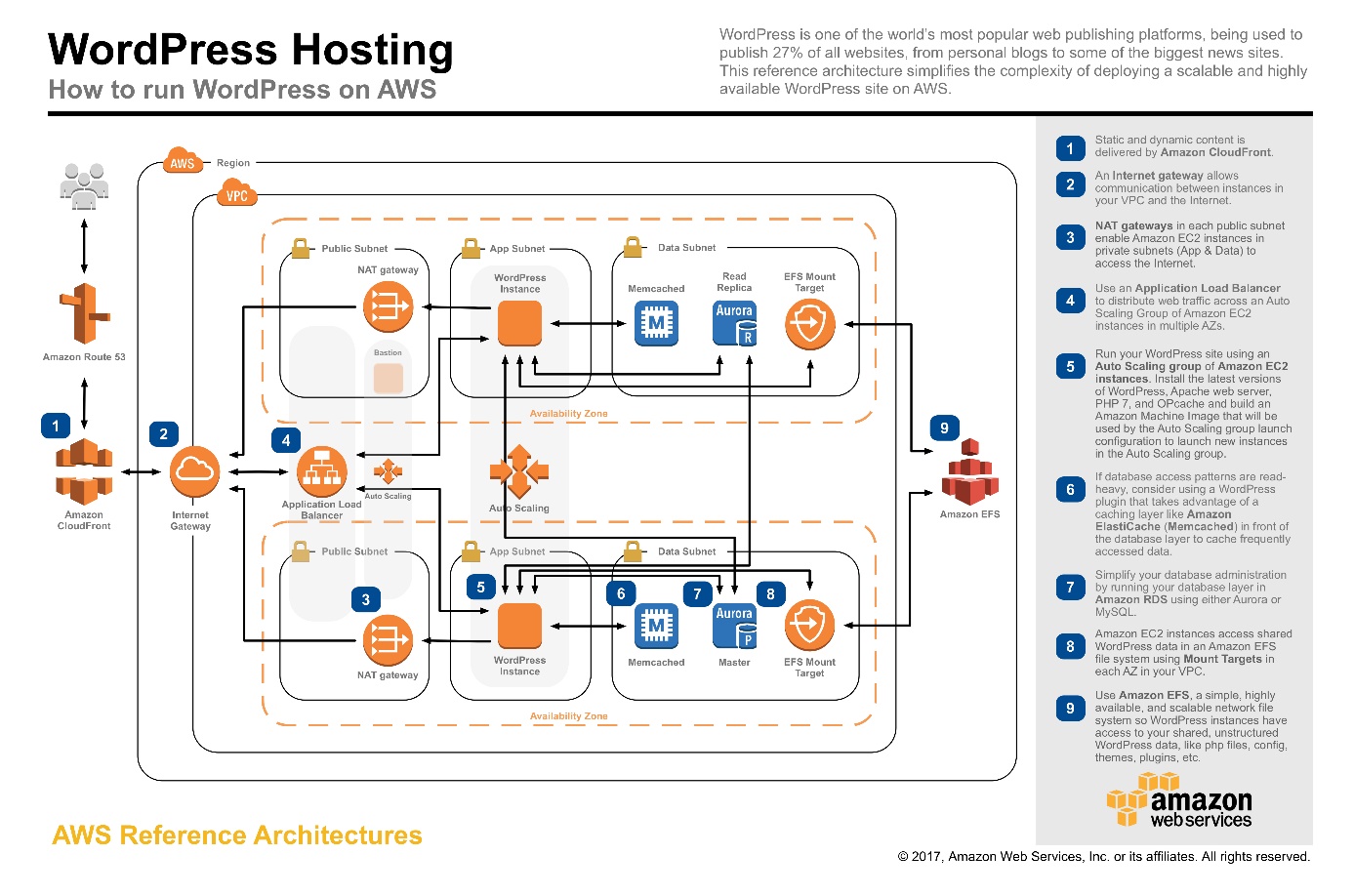
You will provision a shared storage layer powered by NFS. Using Amazon EFS you will create an NFS cluster across multiple availability zones. Then you will create a load-balanced group of web servers that will automatically scale in response to load to complete your application tier.

These labs are based upon the materials developed as a reference architecture by AWS. The reference architecture is available as a GitHub repository. These labs are based upon the materials developed as a reference architecture by AWS. The reference architecture is available as a GitHub repository here: <https://github.com/hosniah/aws-refarch-wordpress>

* **PREREQUISITES:**

The Wordpress deployment will be created in an AWS Virtual Private Cloud (VPC) which creates a virtual, software-defined network across AWS availability zones (AZs). An availability zone represents the logical grouping of one or more physical data centers which are fault isolated from one another, providing you with a way to create resilient, fault-tolerant applications and architectures.

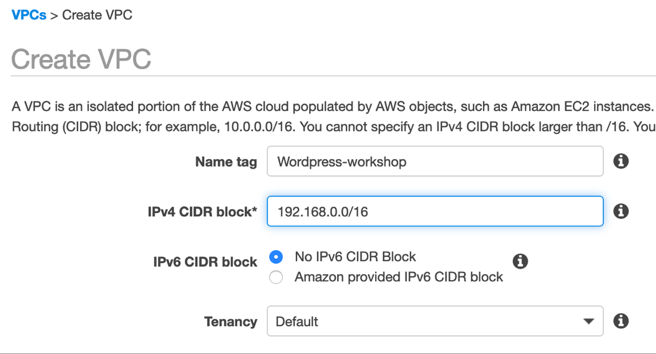
In the first lab you will create your own VPC in your AWS account and use the subnets and security groups you create to provision and protect your web application.



* **LAB 1: CONFIGURE THE NETWORK**

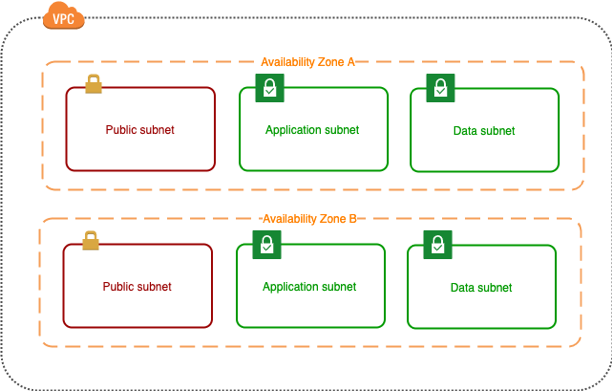
As a starting point for the workshop you will need to login to your AWS account, select the region of your choice and create a new VPC.

To do this click on **Your VPCs** on the left hand side of the console and click **Create VPC**. Enter a name for your VPC and a CIDR range such as the one below. When you’re fiinished click **Create**.



### **Create public and private subnets in the new VPC**

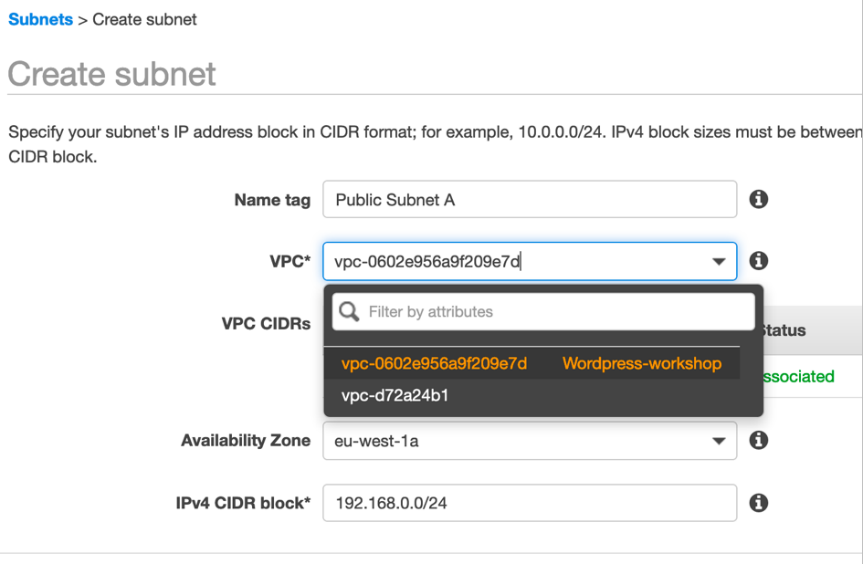
Once the VPC has been created, the next step is to create the subnets that will be used to host the application across two different Availability Zones. We are going to create six subnets in total, three for each AZ, as shown in the following diagram:



The first pair of subnets, Public, will be accessible from the Internet and contain load balancers and NAT gateways. The second pair, Application, will contain application servers and your shared NFS filesystem. Your application servers will be able to communicate with the Internet via the NAT gateways but will only be addressable from the load balancers.

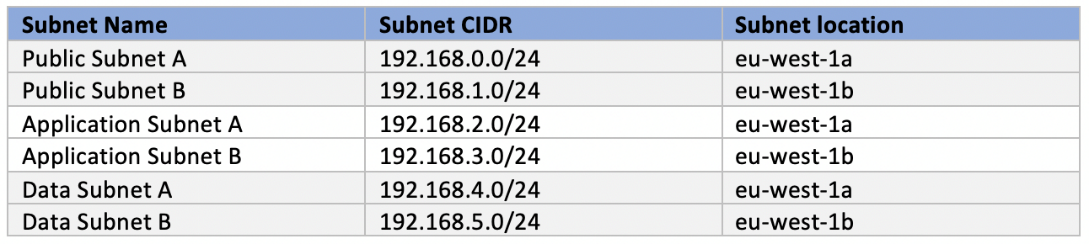
Finally the Database pair of subnets will hold your active / passive relational database and data cache. It will be accessible to other resources in the VPC but will have no access to the Internet and cannot be addressed by the Internet or the load balancers.

To create each of the six subnets please select **Subnets** on the left of the AWS VPC console, then click on **Create subnet** and use the details in the table below to define the characteristics of each of your subnets. Make sure to always select the **Wordpress-workshop** VPC when creating the subnets.



* *The screenshots in this lab were taken from a deployment in the Ireland (eu-west-1) region, if you are building in a different AWS region please just ensure that you create your subnets in 2 different availability zones in the same region, such as us-west-2a and us-west-2b.*

For each subnet specify a name and a CIDR range for the subnet. Be sure and create a public, application, and data subnet in each of two availability zones as detailed in the table below.

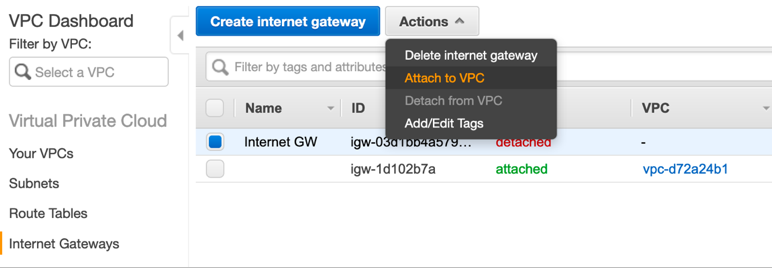


At this point all the correct subnets have been created and they can route network traffic between them. In the next set of steps you will create an Internet Gateway, allowing communication between your VPC and the Internet. You will also configure your routing tables to only allow internet communication with your public subnets and not the private application or data subnets.

### **Create an Internet Gateway and set up routing**

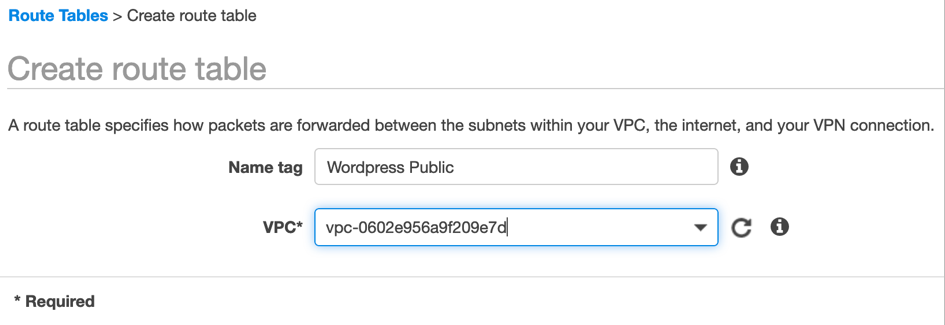
The following steps will allow connectivity from the Internet to the public subnets and also connectivity from the private subnets to the Internet via NAT gateways.

First you need to create a new Internet Gateway (IGW) from your VPC dashboard and attach it to the Wordpress-workshop VPC. Start by clicking **Internet Gateways** on the left hand side of the VPC console and then click the **Create internet gateway** button. Enter a name for your IGW such as WP Internet Gateway and click **Create**. After the IGW has been created you need to associate it with your VPC by attaching it to your VPC:

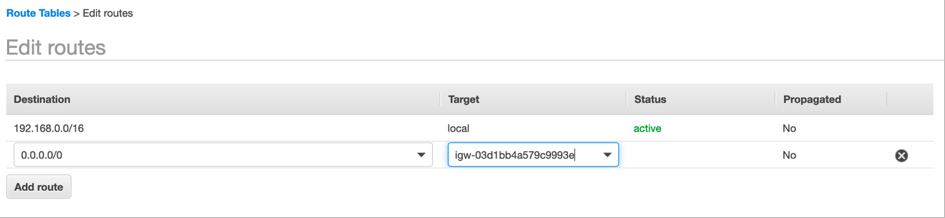


The gateway will be used by instances and services hosted in the public subnets (e.g. Public Subnet A and Public Subnet B) to communicate over the Internet.

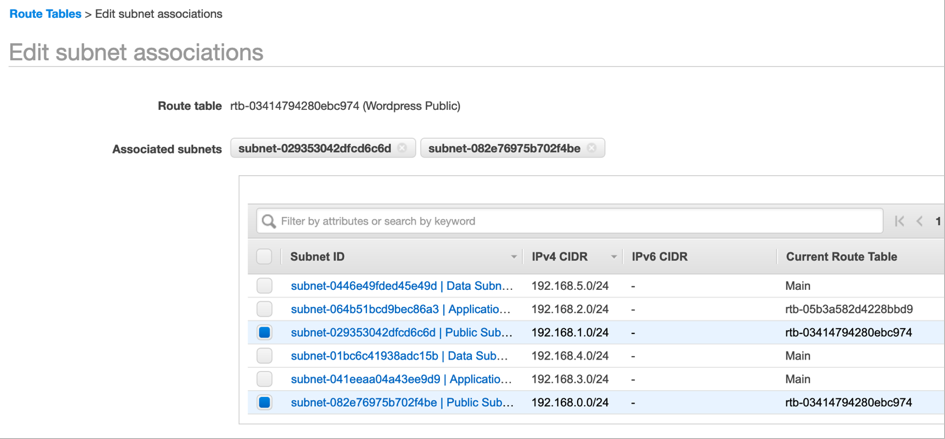
Once the gateway is created you will need to create a new routing table and associate it with the public subnets. Create a new route table by selecting **Route Tables** in the left-hand menu of the console.



After creating the routing table select it from the **Route Tables** section of your VPC dashboard, then click on **Edit routes** and add a default route via the Internet Gateway created in the previous step:



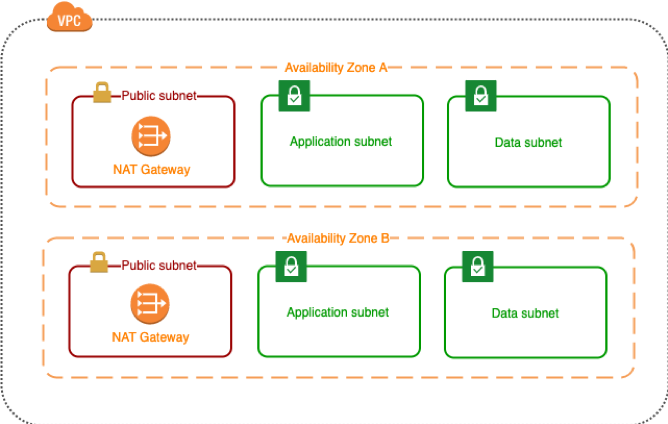
Finally, you need to associate the newly created routing table with the public subnets. To do that, click on the respective route table, then click on **Subnet Associations** and select the two public subnets created earlier:



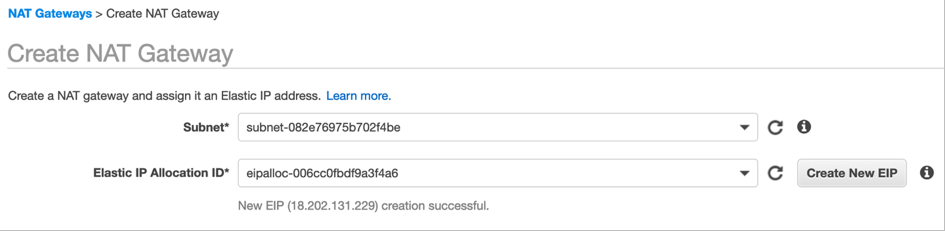
### **Create NAT gateways across the public subnets**

The Wordpress instances will need to be able to connect to the Internet and download application or OS updates so we’re going to create two NAT gateways, one for each availability zone where the application is deployed.

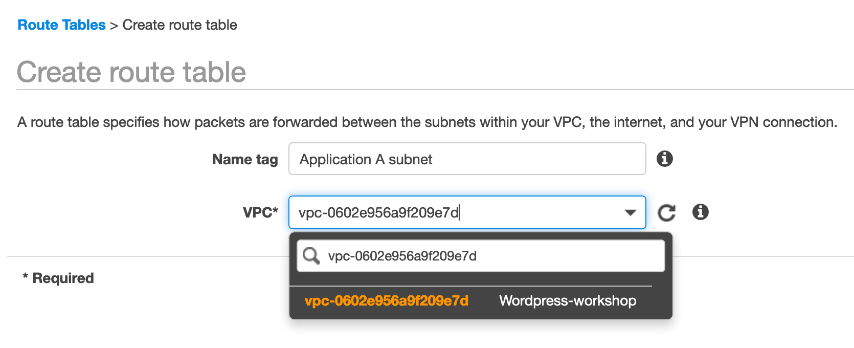
To do this you will create a NAT Gateway for each availability zone, then create a routing table for the application subnet in each availability zone, update the routing table with a path to the NAT gateway, and then associate it with the application subnet.

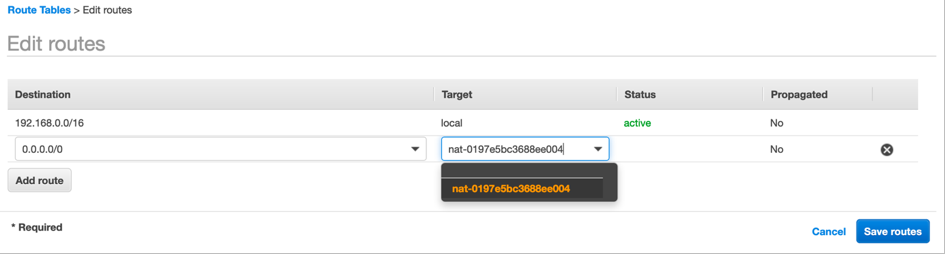


Go to the VPC dashboard in your account, select NAT Gateways and create one gateway in each of the two public subnets (i.e. Public Subnet A and Public Subnet B) Always make sure you have selected the correct public subnet when creating the gateway.

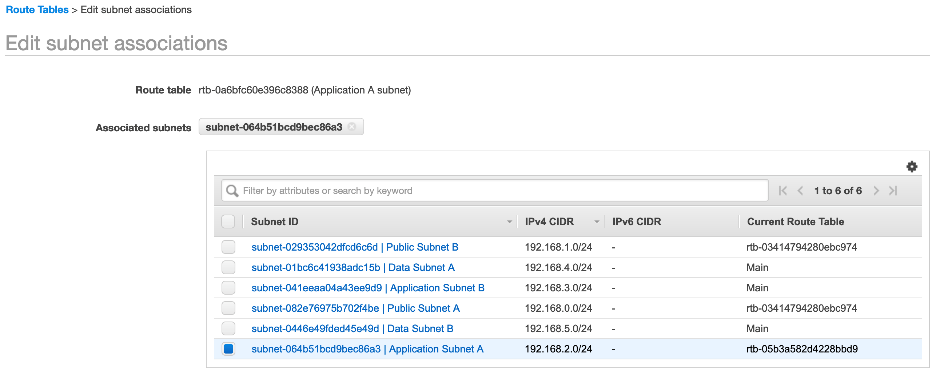


Now we need to create route tables for each of the two Application subnets and use the NAT gateways created earlier as the default gateway:



Edit the route table and add the default route via the NAT gateway in Application subnet A:

Associate the route table with Application Subnet A:



Repeat the last three steps to also create a route table for Application Subnet B which uses the NAT gateway deployed in the second availability zone.

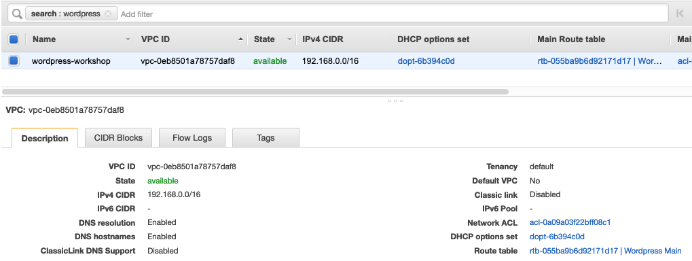
### **Verify your configuration**

You have now created a virtual private cloud network across two availability zones within an AWS region. You have created six subnets, three in each availability zone, and have configured a route so that the internet can communicate with resources in the public subnets and vice versa.

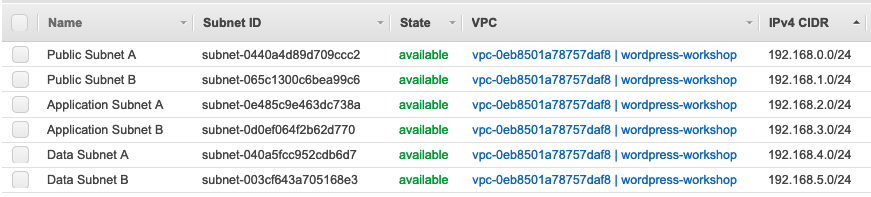
The application subnets have been configured, via routing table, to communicate with the internet via NAT gateways in the public subnets, and the data subnets can only communicate with resources in the six subnets, but not the internet.

* Please compare your own configuration based on the screenshots below and move along when you have verified your setup.

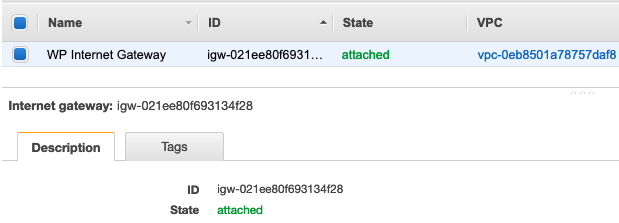
**VPC:**



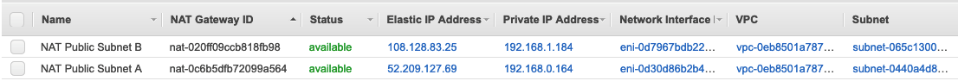
**Subnets**



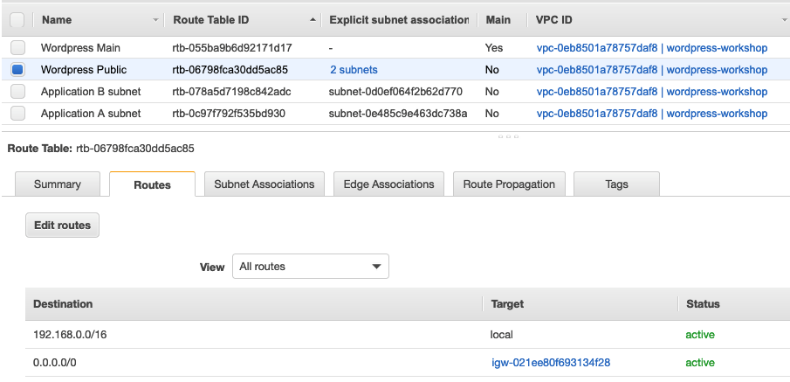
**Internet Gateway**



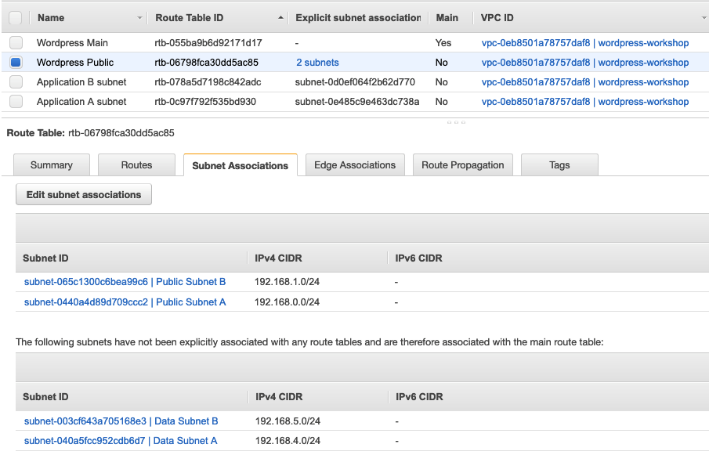
**NAT Gateway**



**Routes > Internet Gateway**



**Routes > Subnet Associations**



# BUILD THE HA DATA TIER

Now that you have created a virtual network across multiple data centers you will create a resilient, cached, highly-available data tier designed to support your Wordpress installation. To do this you will, over the next 2 labs, create an active / passive database deployment using Amazon Relational Database Service (RDS) and then add caching to the database using the managed caching service Amazon ElastiCache.

# LAB 2: SET UP YOUR RDS DATABASE

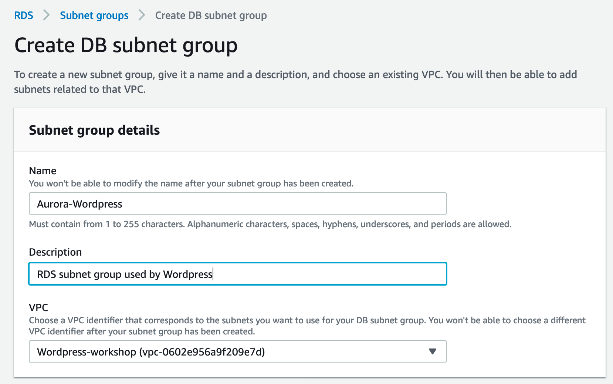
### **Create database security groups**

Visit the [AWS VPC console](https://console.aws.amazon.com/vpc/home) and create 2 security groups. The first security group should be named something like WP Database Client SG and the second security group should be named WP Database SG. With both groups created edit the **Inbound Rules** of the WP Database SG security group and create a rule of type MySQL / Aurora which allows traffic on port 3306 from the WP Database Client SG.

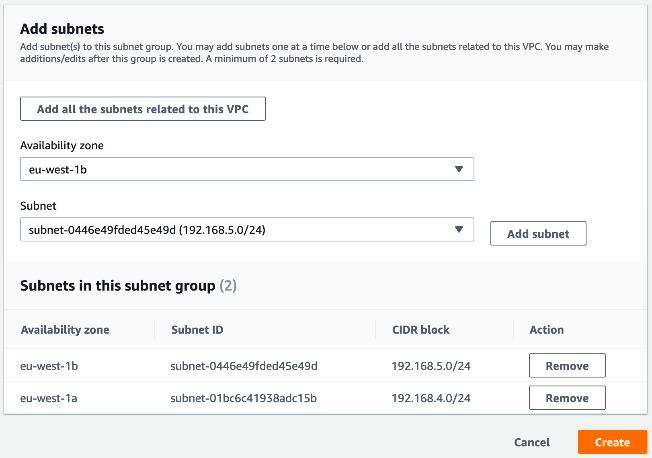
### **Create an RDS subnet group**

Amazon RDS provides managed database deployments. When you use Amazon RDS to deploy a database in a highly available fashion it will create 2 instances in 2 different availability zones. To do this, when you create a database deployment you specify a subnet group which tells RDS in which subnets it can deploy your database instances.

To create a DB subnet group browse to the [Amazon RDS console](https://console.aws.amazon.com/rds/home), click on **Subnet groups** in the panel on your left, click on the **Create DB Subnet Group** button and use the following details:

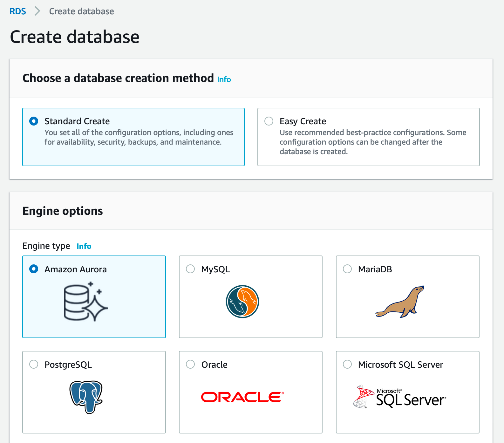


Scroll down and add the two **Data subnets** created earlier (one for each AZ) to your new subnet group. Please note, in order to get the ID of the data subnets, you can open a second tab and navigate to the **Subnets** section of the VPC console. From the list of subnets select the one you are interested in. On the bottom of the screen you will then be able to copy the **SubnetID** by clicking the Copy to clipboard icon that appers when hovering over the Id.

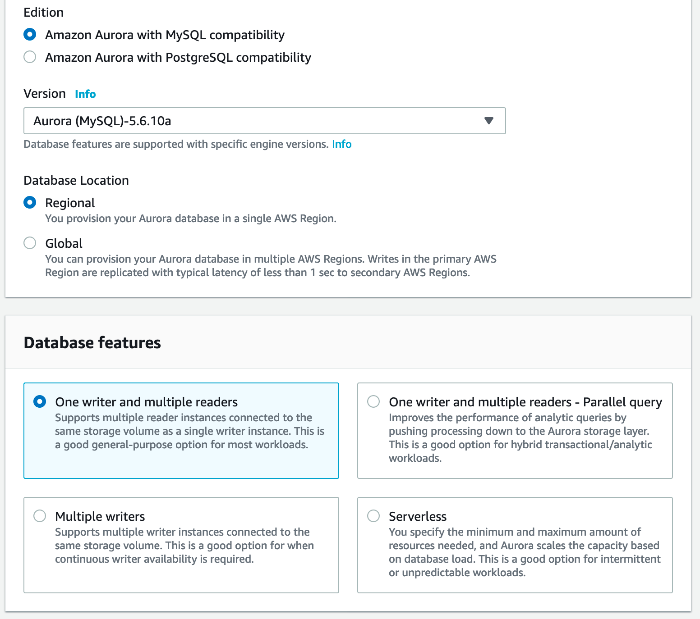


### **Create the Aurora database cluster**

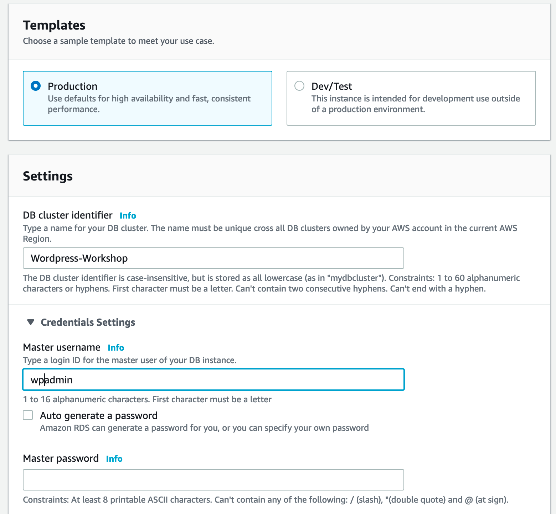
Once the subnet group has been created you are ready to launch your RDS-managed database. Click **Databases** on the left and click **Create database** and use the following details:

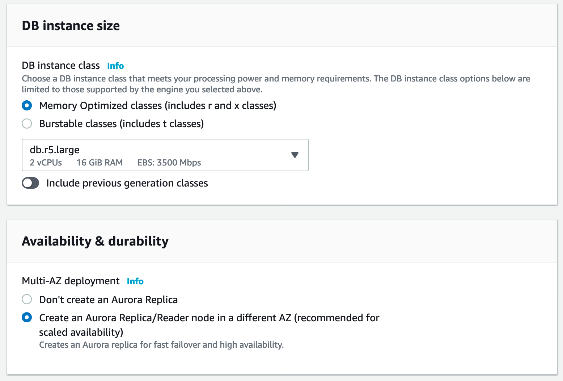


Select the MySQL-compatible Aurora engine:

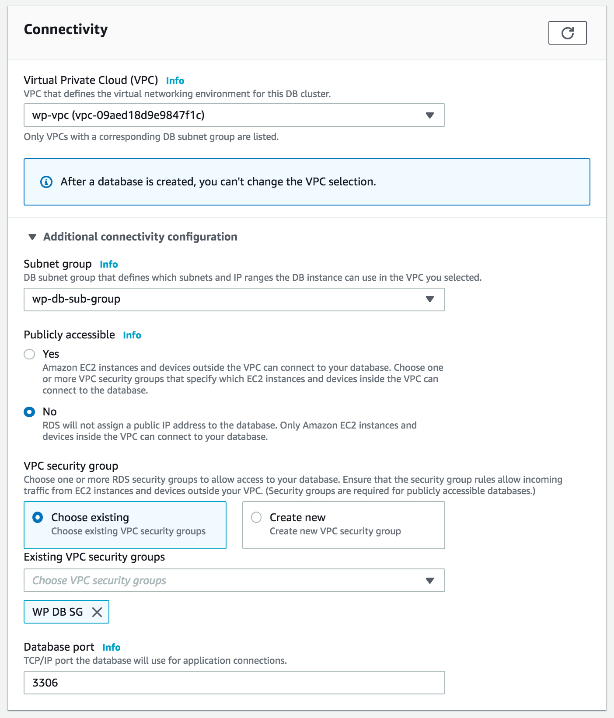


When prompted about the Master password, you can either click on Auto generate a password or create your own – in either case please make sure you write down the password as it will be required a few steps later when setting up the DB connectivity of the Wordpress instances.

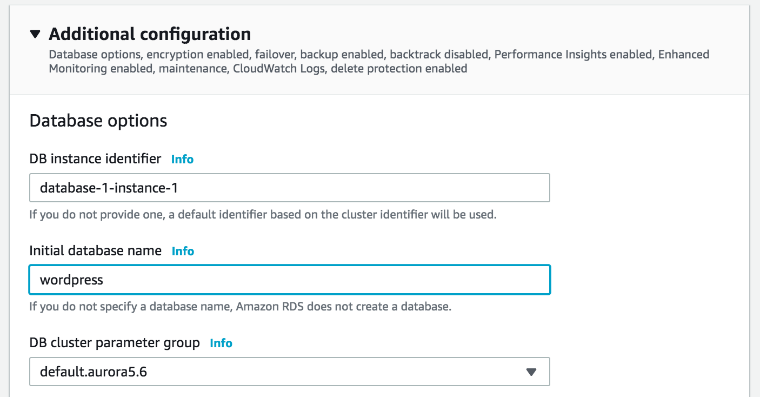


Select the DB instance size together with a Multi-AZ deployment, required for high availability:

In the connectivity section, make sure you select the correct VPC, together with the DB subnet group, and security group created earlier:



Expand **Additional configuration** and specify an **Initial database name** of wordpress.

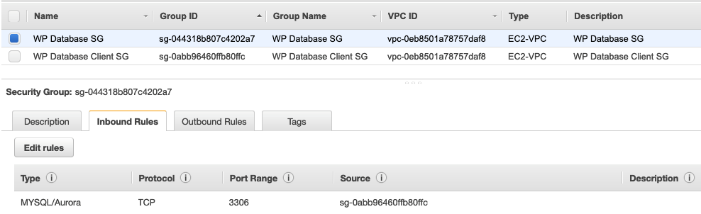
Finally, click on **Create database** to start building the cluster: 

The database will take a few minutes to be provisioned and made available.

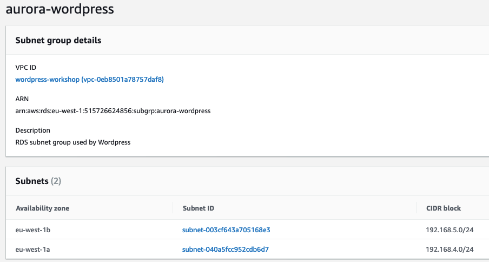
### **Verify your configuration**

You’re active / passive database should now be available and running in two different availability zones, waiting for connections from any EC2 resource with the client security group associated to it. Please compare your own configuration based on the screenshots below and move along when you have verified your setup.

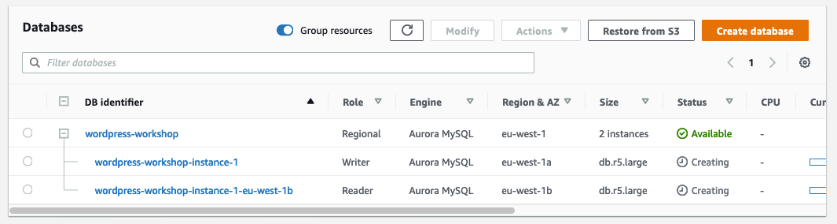
**Security Groups**



**Subnet group**



**Database setup**



If your database shows ‘creating’ for a very long time, click the ‘refresh’ button to update the screen.

# LAB 3: SET UP ELASTICACHE FOR MEMCACHED

Memcached is a server-side caching mechanism that reduces the database load by caching common database queries, leading to a faster loading Wordpress website.

In this lab you will create a managed deployment of Memcached. To start you will need to create a client and a server security group to protect your Memcached instances.

### **Create cache security groups**

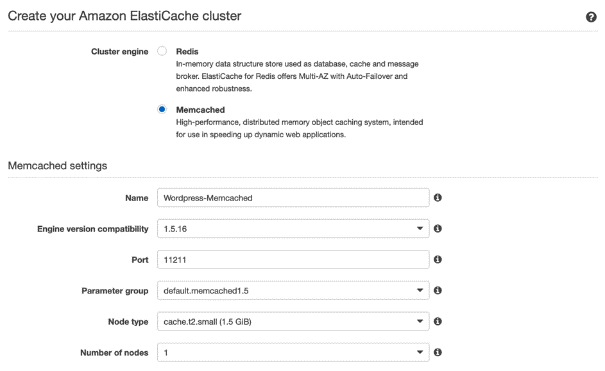
Visit the [AWS VPC console](https://console.aws.amazon.com/vpc/home) and create 2 security groups. The first security group should be named something like WP Cache Client SG and the second security group should be named WP Cache SG.

With both groups created edit the **Inbound Rules** of the WP Cache SG security group with type Custom TCP Rule allowing traffic on port 11211 from the WP Cache Client SG.

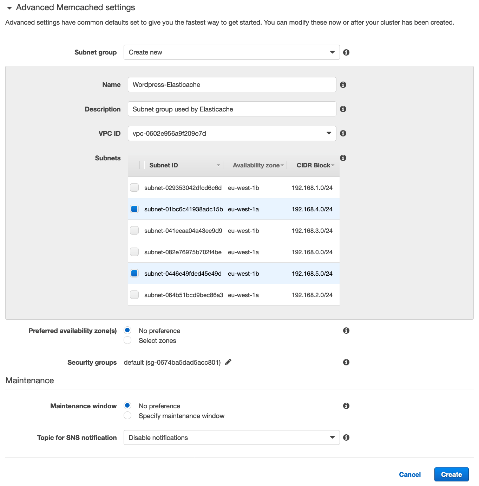
Again, use autocompletion to select the security group in the source field of the security group. Now you are ready to create your Memcached instance.

### **Create an ElastiCache Memcached instance**

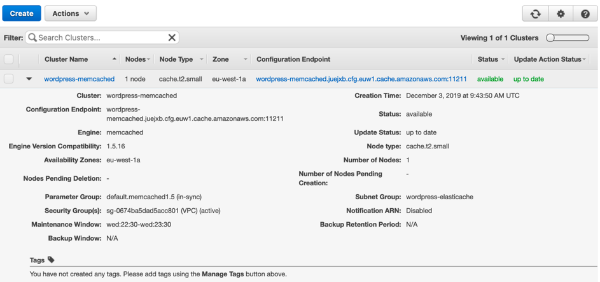
To create an ElastiCache instance open the [Amazon ElastiCache console](https://console.aws.amazon.com/elasticache/home) and click on **Memcached**, and click on **Create** to spin up your first cluster. Ensure that **Memcached** is selected and you have given your cache a name.



Then click on Advanced Memcached settings and make sure you create a new subnet group consisting of the Data subnets of both availability zones:



* *Edit the security group and select the security group you created earlier for your cache instance. Deselect the default security group, leaving only your WP Cache SG selected.*

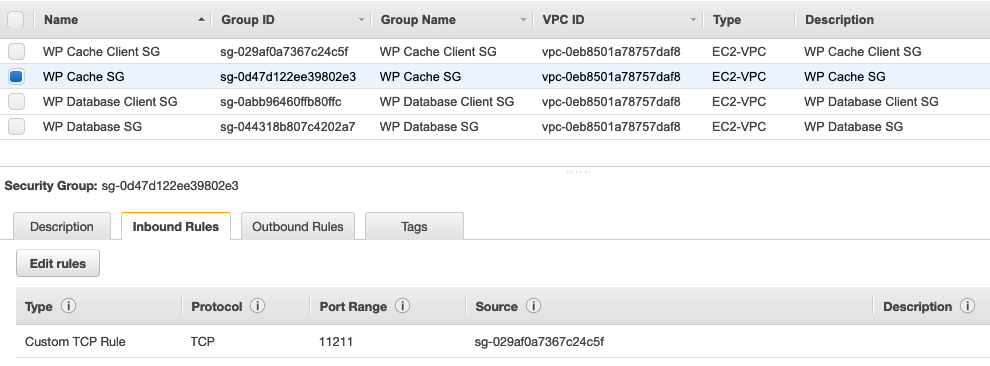
Finally, click on Create to start building your cluster. Once the cluster has been built, click on it to expand it and write down the Configuration Endpoint parameter as you will need it a few steps later:

After a few minutes your Memcached instance will show with a status of Available.

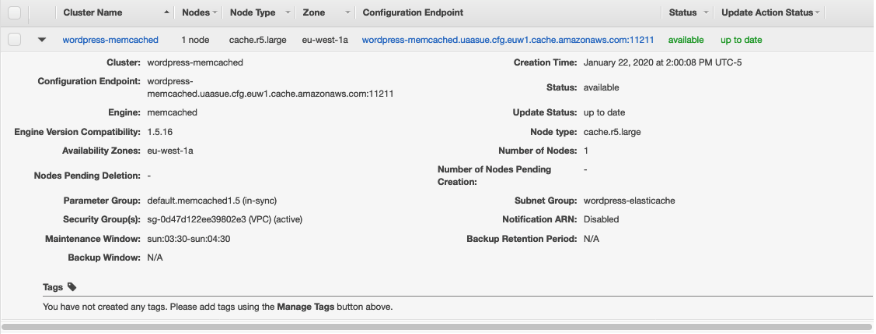
### **Verify your configuration**

Please compare your own configuration based on the screenshots below and move along when you have verified your setup.

**Cache security groups**



**Memcached setup**



# LAB 4: CREATE THE SHARED FILESYSTEM

Amazon Elastic Filesystem (EFS) provides you with a managed NFS cluster which is compatible with NFS v4.1. In this lab you will create an EFS cluster that will provide a shared filesystem for your web application servers.

### **Create filesystem security groups**

Visit the [AWS VPC console](https://console.aws.amazon.com/vpc/home) and create 2 security groups. The first security group should be named something like WP FS Client SG and the second security group should be named WP FS SG.

With both groups created edit the **Inbound Rules** of the WP FS SG security group and add a rule of type NFS which allows traffic on port 2049 from the WP FS Client SG.

Again, use autocompletion to select the security group in the source field of the security group. Now you are ready to create your EFS cluster.

### **Create the EFS cluster**

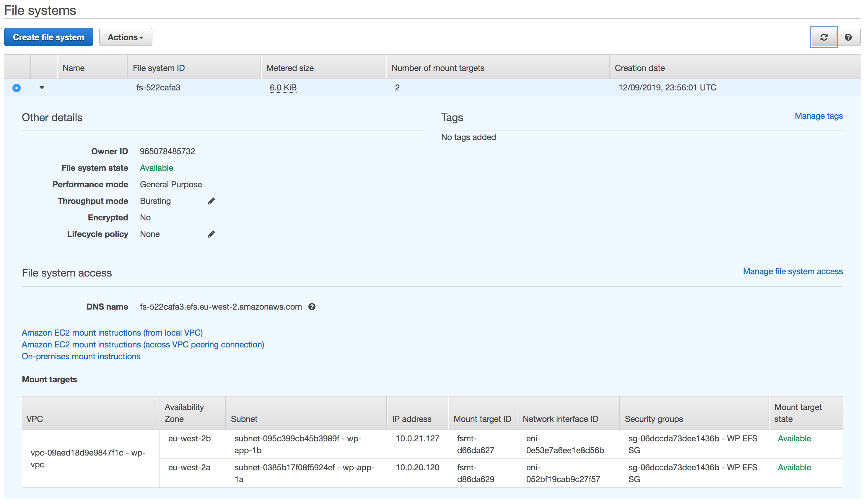
To create an EFS cluster visit the [Amazon EFS console](https://console.aws.amazon.com/efs/home) and click **Create file system**.

From the VPC drop down select your VPC and click **Customize**. Accept the defaults for File system settings and click **Next**.

For Network access, under Mount targets, choose the two subnets you created for the application tier (application subnet A and B). For each mount target, on the right under Security groups, associate the WP FS SG security group created above to each mount target and remove the association with the default security group. Click **Next**.

Accept the defaults on the next screen for File system policy by clicking **Next** and then confirm the file system creation by clicking **Create**.

This will create two mount targets in your subnets and after a few minutes the targets will reach a state of Available.



# BUILD THE APPLICATION TIER

You have created a shared filesystem, a central HA database, and a caching layer to improve application response times. It is now time to deploy the application itself in a highly available fashion. To do this you will create an EC2 AutoScaling group which will add and remove servers to a fleet of application servers in response to network traffic. You will also deploy a load balancer to distribute traffic across these instances as they are added and removed.

# LAB 5: CREATE THE LOAD BALANCER

To distribute traffic across your Wordpress application servers you will need a load balancer. In this lab you will create an OSI layer 7 Application Load Balancer.

### **Create load balancer and application security groups**

Visit the [AWS VPC console](https://console.aws.amazon.com/vpc/home) and create a security group for the load balancer. The security group should be named something like WP Load Balancer SG.

Again, use autocompletion to select the security group in the source field of the security group.

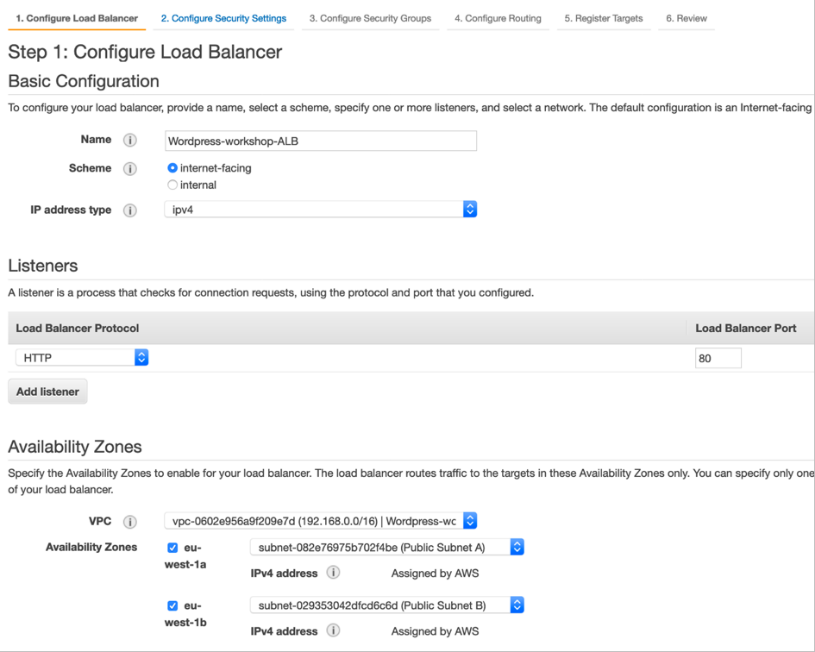
Edit the **Inbound Rules** for **WP Load Balancer SG** and allow HTTP traffic on port 80 from 0.0.0.0/0 (indicating from anywhere).

### **Create a load balancer**

A load balancer distributes incoming application traffic across multiple targets, such as EC2 instances, in multiple Availability Zones, increasing the availability of the Wordpress platform.

From the EC2 console click **Load Balancers** on the left-hand menu and then click **Create Load Balancer**. Under Application Load Balancer click **Create**.

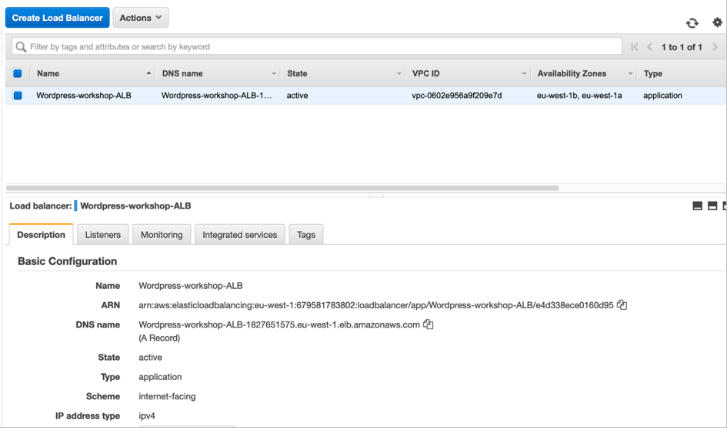
Give your load balancer a name and under **Availability Zones** select your VPC. Then tick the checkbox for both availability zones and select your public subnets as created in the first lab.



Click **Next: Configure Security Settings**, and then **Next: Configure Security Groups**.

Select the **WP Load Balancer SG** created earlier and click **Next: Configure Routing**.

Give the target group a name and click **Next: Register Targets**. Click **Next: Review** without defining any targets and then click **Create**.



Make a note of the DNS name created for your load balancer as you will need this in the following steps.

# LAB 6: CREATE A LAUNCH CONFIGURATION

You have created a software-defined network across multiple fault-isolated data centers, deployed an HA active / passive MySQL database, a managed Memcached instance, and a distributed NFS cluster for shared storage. In this lab you will deploy an HA application server running PHP to use these resources as part of a scalable Wordpress installation.

### **Create a security group for the Wordpress servers**

Visit the [AWS VPC console](https://console.aws.amazon.com/vpc/home) and create a security group for the Wordpress servers. The security group should be named something like WP Wordpress SG.

Edit the **Inbound Rules** for **WP Wordpress SG** and only allow HTTP traffic on port 80 from the **WP Load Balancer SG**.

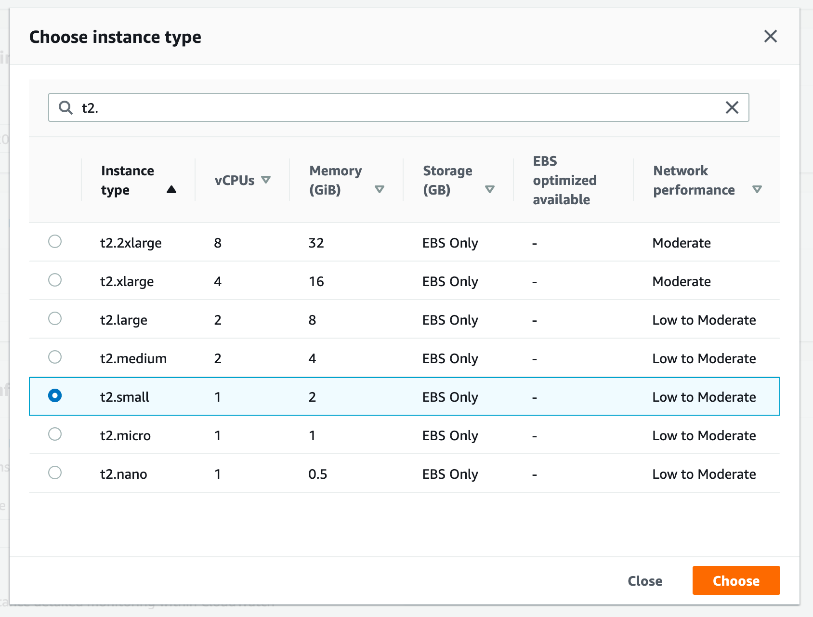
### **Create a launch configuration for the Auto Scaling Groups (ASG)**

A launch configuration is an instance configuration template that an Auto Scaling Group uses to launch EC2 instances. When you create launch configurations you need to specify configuration information for the instances, including the ID of the Amazon Machine Image (AMI), the instance type, a key pair, one or more security groups, and a block device mapping.

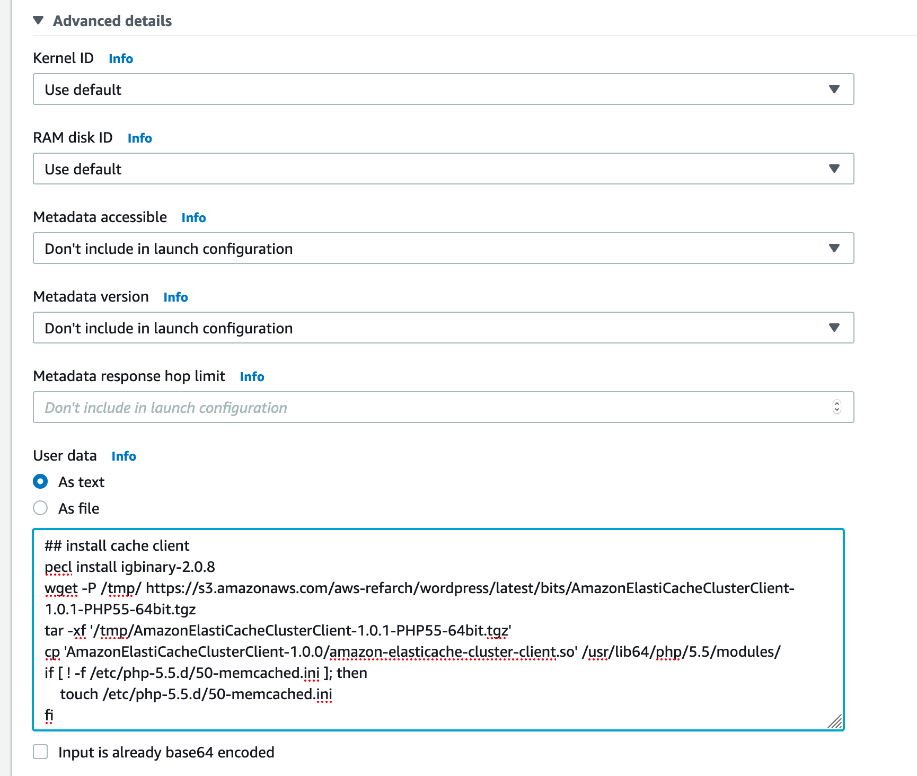
Select **Launch configurations** in your EC2 dashboard, then click on **Create launch configuration**. Choose the **Amazon Linux AMI** by typing the appropriate (based on region) AMI ID into the **AMI** dropdown.

| Region | Amazon Linux AMI ID |
| --- | --- |
| Ireland (eu-west-1) | ami-0eab41619a08cc289 |
| London (eu-west-2) | ami-02001468f37a0519e |
| Frankfurt (eu-central-1) | ami-08806ac26c006c18c |
| N. Virginia (us-east-1) | ami-01151c46ffec20ad3 |
| Oregon (us-west-2) | ami-03ef0decd4b2b21b7 |
| Ohio (us-east-2) | ami-09c18af55c950e78f |
| Sydney (ap-southeast-1) | ami-035b8bea67aeafba8 |

Select the instance type:



Under *Additional configuration* expand *Advanced details* and use the script below to populate the User Data field as text.



**Update the variables in the Bash script below with the values from your environment.**

* EFS\_MOUNT

This should be set to the DNS hostname of the Elastic Filesystem deployed in the previous lab. To obtain the DNS name for your EFS system visit the EFS console and from the detail page for your filesystem click *Attach*. You will see the DNS hostname towards the bottom of the popup, a value similar to fs-5a59585f.efs.us-west-2.amazonaws.com.

* DB\_NAME

This is the name of the database which Wordpress should use to store its data. If you entered the default values in Lab 2 this should be a value of wordpress. To confirm visit the details page for your RDS database and look for *DB name* under *Configuraiton*.

* DB\_HOSTNAME

This is the hostname of your database Writer instance. To obtain this visit the details page for your RDS database and look under *Connectivity & Security*. Use the *Writer* type instance hostname, a value such as wordpress-db.cluster-cytbfylghhjz.us-west-2.rds.amazonaws.com.

* DB\_USERNAME

This will be the database username you specified in Lab 2. It can be found as *Master username* under *Configuration* on the details page for your RDS instance.

* DB\_PASSWORD

This is the password for the database user created in Lab 2.

* LB\_HOSTNAME

This is the hostname for your application load balancer. To determine the hsotname visit the EC2 console and click on your load balancer to access its details page. Under *Basic Configuraiton* you will see a field labeled *DNS name*. It will have a value such as wordpress-lb-678399122.us-west-2.elb.amazonaws.com.

#!/bin/bash -xe

EFS\_MOUNT="<YOUR-EFS-HOSTNAME>"

DB\_NAME="wordpress"

DB\_HOSTNAME="<YOUR-DB-HOSTNAME>"

DB\_USERNAME="<YOUR-DB-USERNAME>"

DB\_PASSWORD="<YOUR-DB-PASSWORD>"

WP\_ADMIN="wpadmin"

WP\_PASSWORD="WpPassword$"

LB\_HOSTNAME="<YOUR-ALB-HOSTNAME>"

yum update -y

yum install -y awslogs httpd24 mysql56 php55 php55-devel php55-pear php55-mysqlnd gcc-c++ php55-opcache

mkdir -p /var/www/wordpress

mount -t nfs4 -o nfsvers=4.1,rsize=1048576,wsize=1048576,hard,timeo=600,retrans=2 $EFS\_MOUNT:/ /var/www/wordpress

## create site config

cat <<EOF >/etc/httpd/conf.d/wordpress.conf

ServerName 127.0.0.1:80

DocumentRoot /var/www/wordpress/wordpress

<Directory /var/www/wordpress/wordpress>

Options Indexes FollowSymLinks

AllowOverride All

Require all granted

</Directory>

EOF

## install cache client

pecl install igbinary-2.0.8

wget -P /tmp/ https://s3.amazonaws.com/aws-refarch/wordpress/latest/bits/AmazonElastiCacheClusterClient-1.0.1-PHP55-64bit.tgz

tar -xf '/tmp/AmazonElastiCacheClusterClient-1.0.1-PHP55-64bit.tgz'

cp 'AmazonElastiCacheClusterClient-1.0.0/amazon-elasticache-cluster-client.so' /usr/lib64/php/5.5/modules/

if [ ! -f /etc/php-5.5.d/50-memcached.ini ]; then

touch /etc/php-5.5.d/50-memcached.ini

fi

echo 'extension=igbinary.so;' >> /etc/php-5.5.d/50-memcached.ini

echo 'extension=/usr/lib64/php/5.5/modules/amazon-elasticache-cluster-client.so;' >> /etc/php-5.5.d/50-memcached.ini

## install wordpress

if [ ! -f /bin/wp/wp-cli.phar ]; then

curl -o /bin/wp https://raw.githubusercontent.com/wp-cli/builds/gh-pages/phar/wp-cli.phar

chmod +x /bin/wp

fi

# make site directory

if [ ! -d /var/www/wordpress/wordpress ]; then

mkdir -p /var/www/wordpress/wordpress

cd /var/www/wordpress/wordpress

# install wordpress if not installed

# use public alb host name if wp domain name was empty

if ! $(wp core is-installed --allow-root); then

wp core download --version='4.9' --locale='en\_GB' --allow-root

wp core config --dbname="$DB\_NAME" --dbuser="$DB\_USERNAME" --dbpass="$DB\_PASSWORD" --dbhost="$DB\_HOSTNAME" --dbprefix=wp\_ --allow-root

wp core install --url="http://$LB\_HOSTNAME" --title='Wordpress on AWS' --admin\_user="$WP\_ADMIN" --admin\_password="$WP\_PASSWORD" --admin\_email='admin@example.com' --allow-root

wp plugin install w3-total-cache --allow-root

# sed -i \"/$table\_prefix = 'wp\_';/ a \\define('WP\_HOME', 'http://' . \\$\_SERVER['HTTP\_HOST']); \" /var/www/wordpress/wordpress/wp-config.php

# sed -i \"/$table\_prefix = 'wp\_';/ a \\define('WP\_SITEURL', 'http://' . \\$\_SERVER['HTTP\_HOST']); \" /var/www/wordpress/wordpress/wp-config.php

# enable HTTPS in wp-config.php if ACM Public SSL Certificate parameter was not empty

# sed -i \"/$table\_prefix = 'wp\_';/ a \\# No ACM Public SSL Certificate \" /var/www/wordpress/wordpress/wp-config.php

# set permissions of wordpress site directories

chown -R apache:apache /var/www/wordpress/wordpress

chmod u+wrx /var/www/wordpress/wordpress/wp-content/\*

if [ ! -f /var/www/wordpress/wordpress/opcache-instanceid.php ]; then

wget -P /var/www/wordpress/wordpress/ https://s3.amazonaws.com/aws-refarch/wordpress/latest/bits/opcache-instanceid.php

fi

fi

RESULT=$?

if [ $RESULT -eq 0 ]; then

touch /var/www/wordpress/wordpress/wordpress.initialized

else

touch /var/www/wordpress/wordpress/wordpress.failed

fi

fi

## install opcache

if [ ! -d /var/www/.opcache ]; then

mkdir -p /var/www/.opcache

fi

# enable opcache in /etc/php-5.5.d/opcache.ini

sed -i 's/;opcache.file\_cache=.\*/opcache.file\_cache=\/var\/www\/.opcache/' /etc/php-5.5.d/opcache.ini

sed -i 's/opcache.memory\_consumption=.\*/opcache.memory\_consumption=512/' /etc/php-5.5.d/opcache.ini

# download opcache-instance.php to verify opcache status

if [ ! -f /var/www/wordpress/wordpress/opcache-instanceid.php ]; then

wget -P /var/www/wordpress/wordpress/ https://s3.amazonaws.com/aws-refarch/wordpress/latest/bits/opcache-instanceid.php

fi

chkconfig httpd on

service httpd start

Then under *Security Groups*, select the following security groups:

* WP Cache Client SG
* WP DB Client SG
* WP EFS Client SG
* WP Wordpress SG

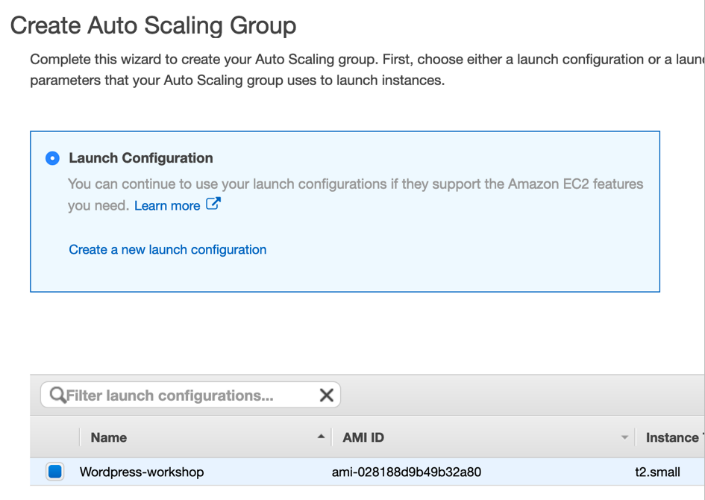
Click Create launch configuration to review and submit the final configuration. You can disregard warnings about being able to SSH into the server and can also choose *Proceed without keypair* as you will not need to remotely access these servers.

# LAB 7: CREATE THE APP SERVER

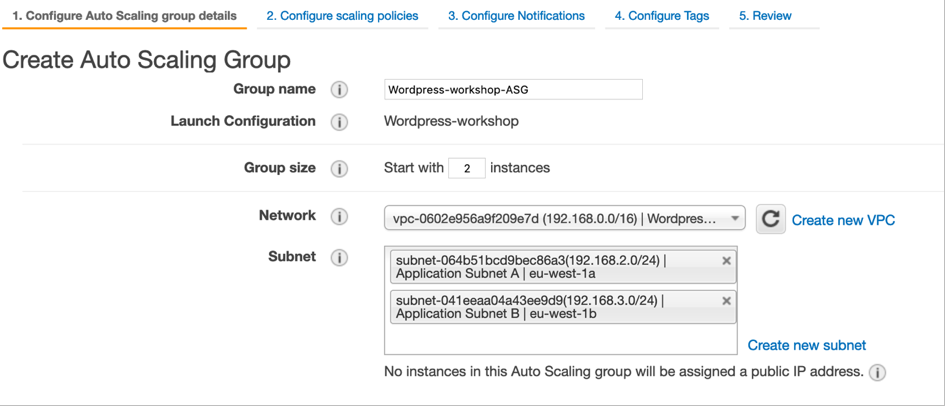
In this lab you will use the load balancer and launch configuration from the previous 2 labs to create an auto scaling fleet of Wordpress application servers.

### **Create the ASG for the back-end web servers**

Once you have created the launch configuration you can proceed to creating the Autoscaling group for the Wordpress web servers. To do that select **Auto Scaling Groups** in your EC2 dashboard, click on **Create Auto Scaling Group** and select the previously created launch configuration:

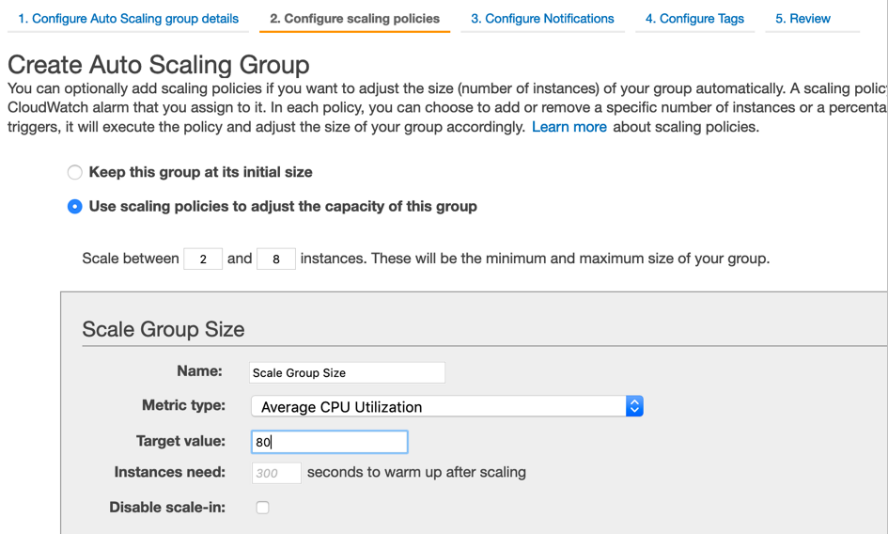


In the next screen make sure that the correct VPC is selected, together with the correct subnets for the web servers (i.e. Application Subnet A and Application Subnet B):



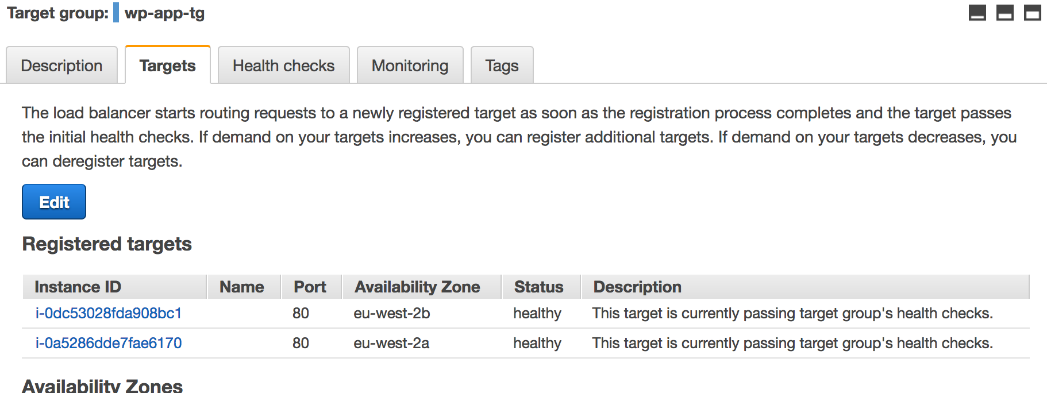
Under Advanced Details tick the box next to Load Balancing and for Target Groups select the target group you created earlier.

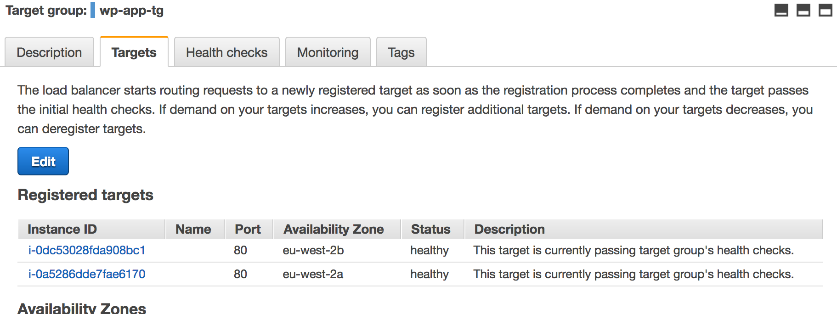
Click Next: Configure scaling policies and configure the scaling policies as follows:



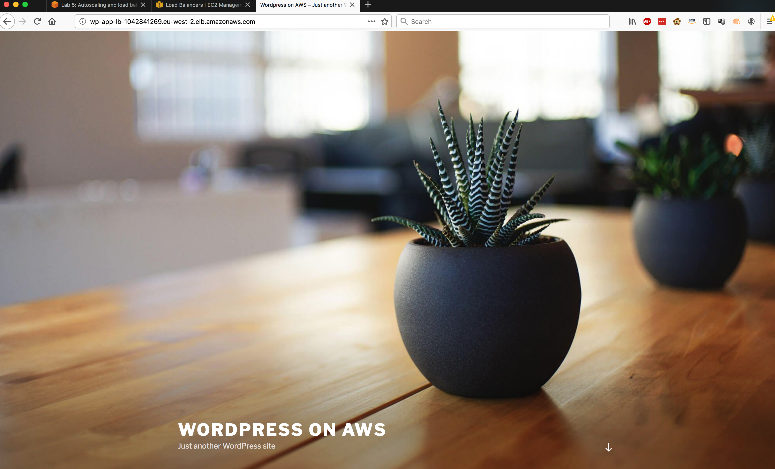
Click through and accept the remaining defaults to complete the creation of your auto scaling group.

The autoscaling group will now begin creating the desired number of EC2 instances based on the launch configuration you created. As the systems come online the target group is updated with the instance details for your EC2 instances and the load balancer will begin distributing traffic across the instances. As instances are added or removed the autoscaling group and load balancer will work in concert with one another to ensure that only healthy instances receive traffic.





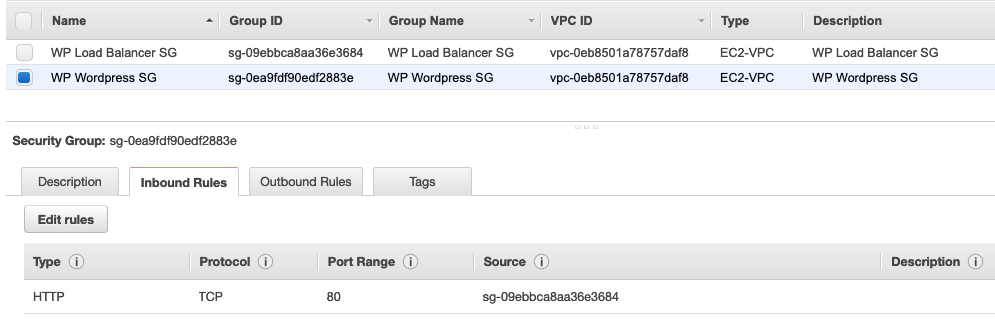
When your targets are deemed healthy in your target group you can open the DNS name for your Application Load Balancer in your web browser to view your newly created Wordpress installation.

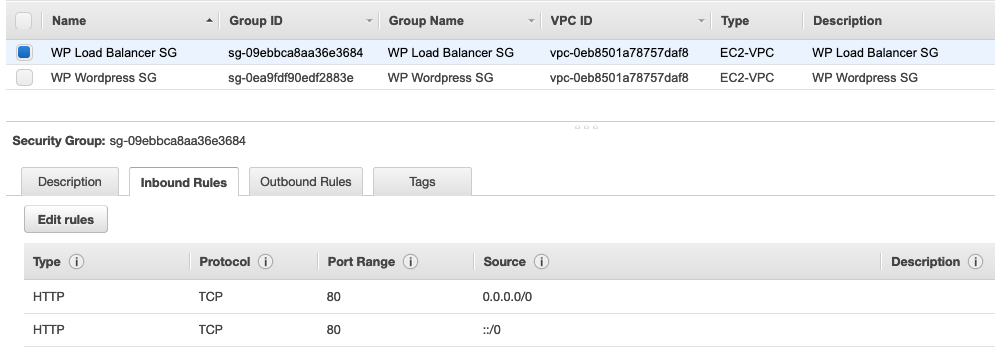


### **Verify your configuration**

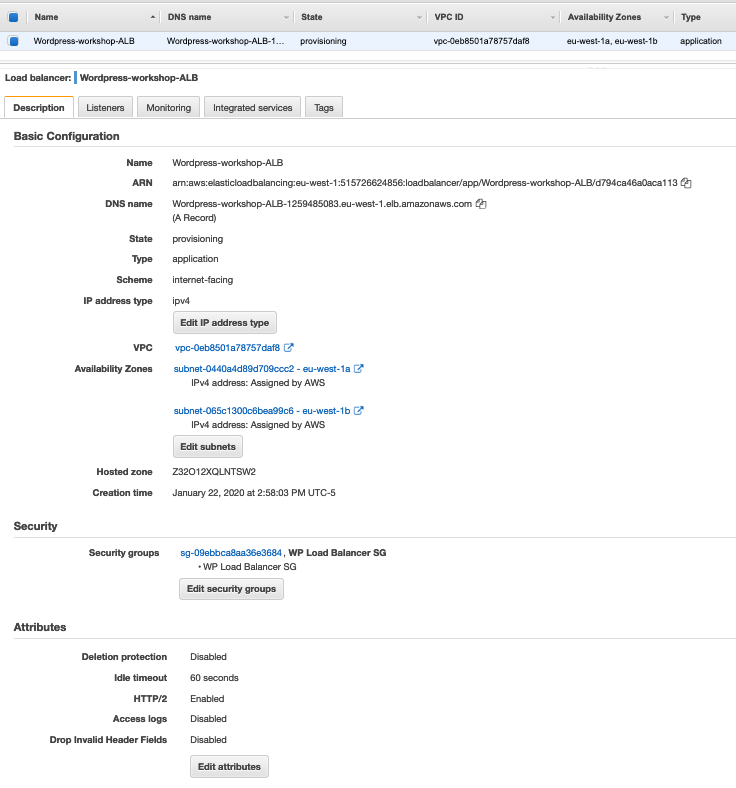
Please compare your own configuration based on the screenshots below and move along when you have verified your setup.

**Security Groups**





**Application Load Balancer**



**Launch config**

A sample configuration for the launch configuration user data should look like this:

EFS\_MOUNT="fs-6f6d4ba4.efs.eu-west-1.amazonaws.com"

DB\_NAME="wordpress"

DB\_HOSTNAME="wordpress-workshop.cluster-cwwfqfv4xcel.eu-west-1.rds.amazonaws.com"

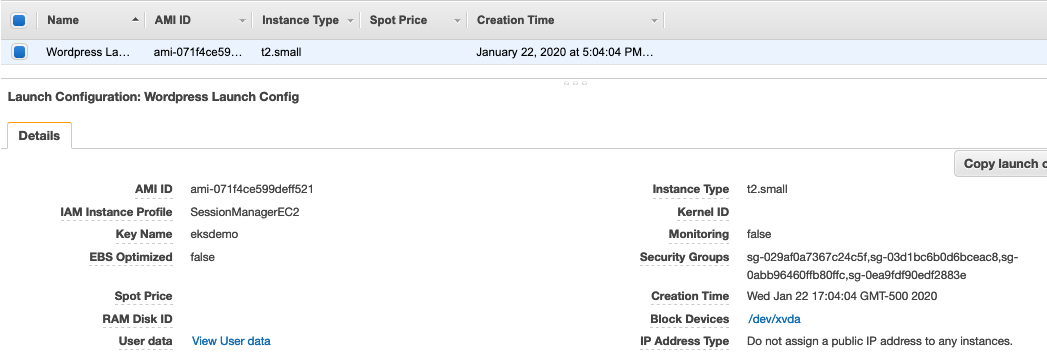
DB\_USERNAME="wpadmin"

DB\_PASSWORD="my-secret-password"

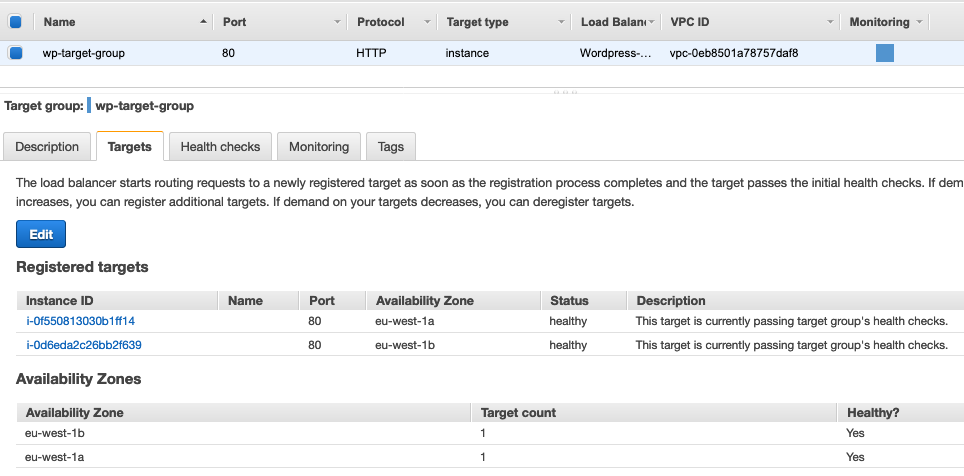
WP\_ADMIN="wpadmin"

WP\_PASSWORD="my-secret-password"

LB\_HOSTNAME="Wordpress-workshop-ALB-1258285083.eu-west-1.elb.amazonaws.com"



**Target group**



# LAB 8: OPTIONAL ADD CACHING

The following lab is optional.

You may have noticed that you have not yet configured Wordpress to use the Memcache deployment created in the earlier labs. To improve your site’s response time and to reduce the strain on your backend database, Wordpress can be configured to use Memcache as a caching layer for common requests.

To do this you can follow the [Wordpress / Memcached instructions](https://aws.amazon.com/elasticache/memcached/wordpress-with-memcached/" \t "_blank) on the AWS website. The necessary plugins and libraries are installed by the user data earlier in Lab 6, however you need to manually configure them through the Wordpress Web UI to use your Memcached deployment.

<https://aws.amazon.com/elasticache/memcached/wordpress-with-memcached/>