

# The new Developers Blog

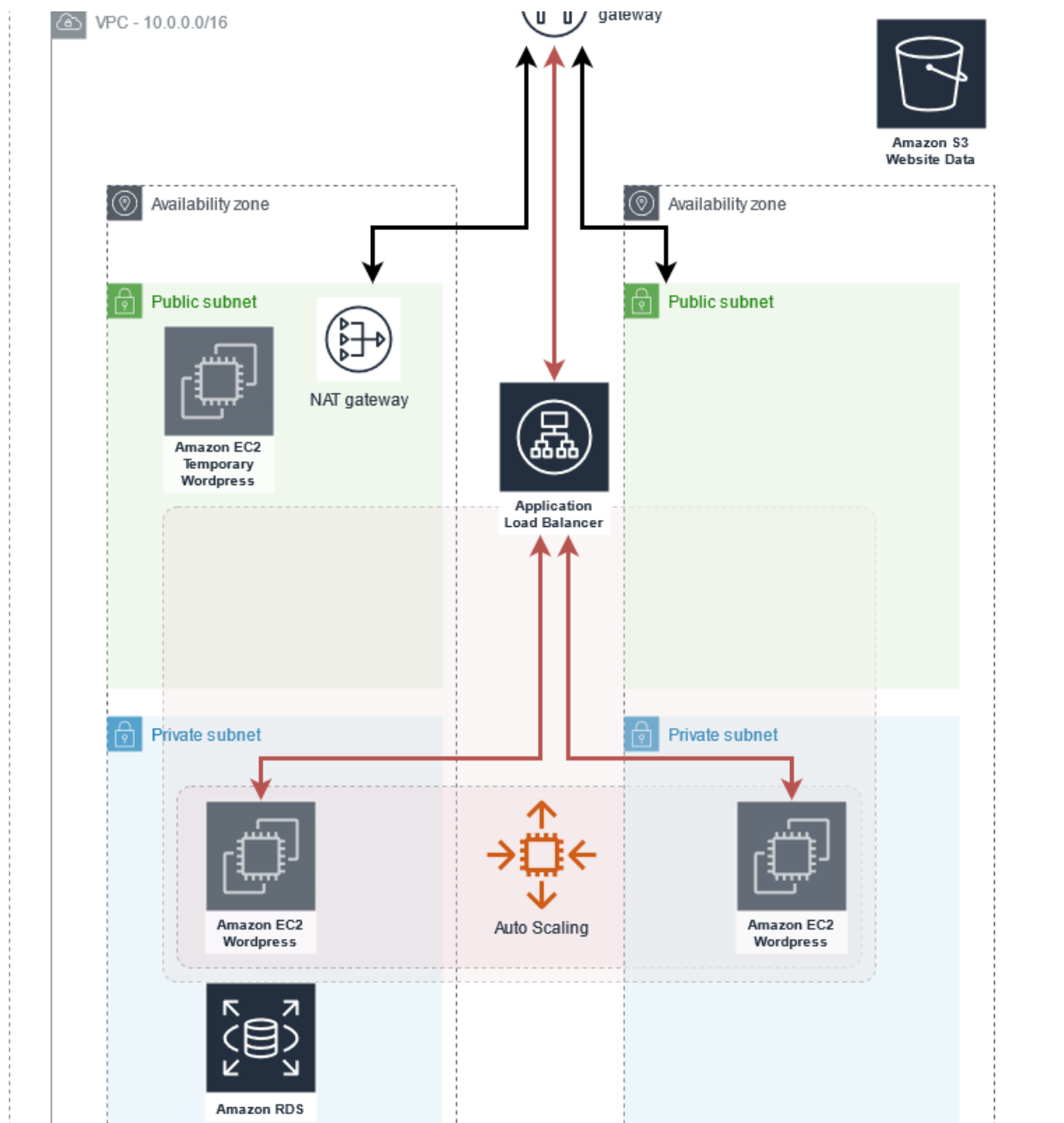
Your company, an ice-cream manufacturer called “Glacier To Eat”, is very proud of their cloud journey. They want to share all the benefits and challenges they had in a developers blog. This blog will also be used to increase the publicity of “Glacier To Eat” as an IT-focused employer and to impress talents.

Your task is to deploy and manage this blog. Because your team leader expects that there will be a huge demand for that blog content, he asks you to consider the following requirements:

- The webserver has to be decoupled from the database, to increase scalability.
- The webserver has to be autoscaling, to meet the expected high demand.
- Due to the rightsizing principle, we choose a small instance type (t2.micro, t3.micro) for our instances.
- We use an S3 Bucket to store the website data
- We use an Application Load Balancer to spread the load of our application

You have three possibilities to complete this project. Choose the Expert Path to have the maximum learning experience. You will try to apply your existing knowledge and you fill knowledge gaps on your own.

Choose the Learners Path if you need some hints in addition to the requirements. The Guided Path is a step to step guide, if you really have no clue what you have to do.



## Expert Path

### Task 1: Initialize the Network

Set up a new VPC with two public subnets in different AZs (e.g A and B) and two private subnets in the same AZs like the public ones (e.g. also A and B).

Hint 1: Use an Internet Gateway.

Hint 2: Use a NAT Gateway

Hint 2: Use two different Routing Tables with a Route to the Internet Gateway and one to the NAT Gateway.

```
$ aws configure
AWS Access Key ID [*****T7Y5]: Enter the Access Key from
Lab Details
AWS Secret Access Key [*****JI84]: Enter the Secret
Access Key from Lab Details
Default region name [us-west-2]: us-west-2
Default output format [None]: json
```

## Task 2: Set Up an RDS Instance

Set up an RDS Instance for our wordpress blog. Choose the MariaDB Engine and a db.t3.micro Instance type.

Hint 1: Use the security group and open the mysql port for the Wordpress Security Group..

Hint 2: You need 2 Security groups. One for the EC2 Instance and one for the RDS Instance

## Task 3: Set Up a Wordpress Instance

Wordpress:

Note: Copy all commands in this step to an external file to use these as userdata initialization for new EC2-Instances

Set up an Amazon Linux EC2 Instance with httpd and php7.4.. And install wordpress on this instance

Hint 1: You can get the latest release of wordpress under

<https://wordpress.org/latest.tar.gz>

## Task 4: High Availability Setup

Create an S3 Bucket and upload the `/var/www/html/wp-content` folder to the bucket. Create an Autoscaling Group using the User Data of the commands you used in the previous task. Add a load balancer to your Setup.

Hint: You have to configure a load balancer attribute so that the login sessions will work accordingly

# Learners Path

## Task 1: Initialize the Network

Set up a new VPC with two public subnets in different AZs (e.g A and B) and two private subnets in the same AZs like the public ones (e.g. also A and B).

Hint 1: Use an Internet Gateway.

Hint 2: Use a NAT Gateway

Hint 2: Use two different Routing Tables with a Route to the Internet Gateway and one to the NAT Gateway.

```
$ aws configure
AWS Access Key ID [*****T7Y5]: Enter the Access Key from
Lab Details
AWS Secret Access Key [*****JI84]: Enter the Secret
Access Key from Lab Details
Default region name [us-west-2]: us-west-2
Default output format [None]: json
```

1. Create the VPC with two subnets, each in different AZSs
2. Create an internet gateway and attach it to the VPC
3. Adjust the Route table to forward internet traffic from the public subnets to the internet gateway
4. Create two private subnets, each in a different AZ. Use the same AZs like you used before for the public subnets.
5. Create a route Table for the private Subnets.
6. Add a NAT-Gateway to your public subnet and adjust the Route table of the private subnets to route internet traffic to your NAT-Gateway.

## Task 2: Set Up an RDS Instance

Set up an RDS Instance for our wordpress blog. Choose the MariaDB Engine and a db.t3.micro Instance type.

Hint 1: Use the security group and open the mysql port for the Wordpress Security Group..

Hint 2: You need 2 Security groups. One for the EC2 Instance and one for the RDS Instance

## Task 3: Set Up a Wordpress Instance

Wordpress:

Note: Copy all commands in this step to an external file to use these as userdata initialization for new EC2-Instances

Set up an Amazon Linux EC2 Instance with httpd and php7.4.. And install wordpress on this instance

Hint 1: You can get the latest release of wordpress under

<https://wordpress.org/latest.tar.gz>

1. Create a key pair, store it in a file and adjust the permissions.
2. Create an instance in a public subnet using the Amazon Linux AMI, t3.micro instance type and the wordpress security group you created in the previous Task.
3. Wait a few minutes for the instance to be ready. Then connect to the instance via SSH.
4. Install php7.4 and a web server.
5. Download Wordpress and copy the files and folders to the Web servers directory.
6. Find out the RDS Endpoint and note it
7. Check if your wordpress page is reachable.
8. Open this url in your browser and configure wordpress.
9. Follow the installation steps

## Task 4: High Availability Setup

Create an S3 Bucket and upload the /var/www/html/wp-content folder to the bucket. Create an Autoscaling Group using the User Data of the commands you used in the previous task. Add a load balancer to your Setup.

1. Create an S3 Bucket. Hint: Use the Account ID as prefix for unique naming.
2. Create an IAM Role which allows an EC2 instance to connect to S3
3. Create an instance profile based on this role and associate it with your EC2 Instance.
4. Connect to the wordpress instance and upload the web servers data to your s3 bucket.
5. Create an Autoscaling group Launch Configuration from your Wordpress EC2 Instance you created in Step 4 (use the commands you copied to that file in the user data)
6. Create an Application Load Balancer and attach it to the Autoscaling Group
7. Note the load balancers URL
8. Adjust the Wordpress Configuration to use the Load Balancer and upload this file to s3
9. Create a target group and a listener for your loadbalancer and attach it to your Auto Scaling group.
10. Activate Load Balancer Stickiness to use always the same instance (Session awareness)
11. Fetch the Load Balancers public IP and open it in your browser.

# Guided Path

In Canvas go to Modules and find the Lab 1-[CF]-Lab - Sandbox Environment in the Instructor Course Dashboard section.

Follow the Description. Read carefully and do not click the “AWS” button.

## Task 1: Initialize the Network

Set up a new VPC with two public subnets in different AZs (e.g A and B) and two private subnets in the same AZs like the public ones (e.g. also A and B).

Hint 1: Use an Internet Gateway.

Hint 2: Use a NAT Gateway

Hint 2: Use two different Routing Tables with a Route to the Internet Gateway and one to the NAT Gateway.

```
$ aws configure
AWS Access Key ID [*****T7Y5]: Enter the Access Key from Lab Details
AWS Secret Access Key [*****JI84]: Enter the Secret Access Key from Lab Details
Default region name [us-west-2]: us-west-2
Default output format [None]: json
```

Create the VPC with two subnets, each in different AZSs

```
$ vpcId=$(aws ec2 create-vpc --cidr-block "10.0.0.0/16" --output text --query "Vpc.VpcId" --tag-specifications 'ResourceType=vpc,Tags=[{Key=Name,Value=MyVPC}]')

$ subnet1Id=$(aws ec2 create-subnet --cidr-block "10.0.0.0/24" --vpc-id $vpcId --availability-zone us-west-2b --output text --query "Subnet.SubnetId" --tag-specifications 'ResourceType=subnet,Tags=[{Key=Name,Value=MyVPC-Subnet1}]')

$ subnet2Id=$(aws ec2 create-subnet --cidr-block "10.0.1.0/24" --vpc-id $vpcId --availability-zone us-west-2a --output text --query "Subnet.SubnetId" --tag-specifications 'ResourceType=subnet,Tags=[{Key=Name,Value=MyVPC-Subnet2}]')
```

Create an internet gateway and attach it to the VPC

```
$ igwId=$(aws ec2 create-internet-gateway --output text --query "InternetGateway.InternetGatewayId")
```



```
$ aws ec2 attach-internet-gateway --vpc-id $vpcId
--internet-gateway-id $igwId
```

**Adjust the Route table to forward internet traffic from the public subnets to the internet gateway**

```
$ routeTableId=$(aws ec2 describe-route-tables --filter
Name=vpc-id,Values=[$vpcId] --query
"RouteTables[0].Associations[0].RouteTableId" --output text)
```

```
$ aws ec2 create-route --destination-cidr-block 0.0.0.0/0
--gateway-id $igwId --route-table-id $routeTableId
```

**Create two private subnets, each in a different AZ. Use the same AZs like you used before for the public subnets.**

```
$ subnet3Id=$(aws ec2 create-subnet --cidr-block "10.0.2.0/24"
--vpc-id $vpcId --availability-zone us-west-2b --output text --query
"Subnet.SubnetId" --tag-specifications
'ResourceType=subnet,Tags=[{Key=Name,Value=MyVPC-Private-Subnet3}]')
```

```
$ subnet4Id=$(aws ec2 create-subnet --cidr-block "10.0.3.0/24"
--vpc-id $vpcId --availability-zone us-west-2a --output text --query
"Subnet.SubnetId" --tag-specifications
'ResourceType=subnet,Tags=[{Key=Name,Value=MyVPC-Private-Subnet4}]')
```

**Create a route Table for the private Subnets.**

```
$ routeTable2Id=$(aws ec2 create-route-table --vpc-id $vpcId --output
text --query "RouteTable.RouteTableId")
```

**Add a NAT-Gateway to your public subnet and adjust the Route table of the private subnets to route internet traffic to your NAT-Gateway.**

```
$ eIpId=$(aws ec2 allocate-address --output text --query
"AllocationId")
```

```
$ natGwId=$(aws ec2 create-nat-gateway --allocation-id $eIpId
--subnet-id $subnet1Id --output text --query
"NatGateway.NatGatewayId")
```

```
$ aws ec2 create-route --destination-cidr-block 0.0.0.0/0
--gateway-id $natGwId --route-table-id $routeTable2Id
```

```
$ aws ec2 associate-route-table --route-table-id $routeTable2Id
--subnet-id $subnet3Id
```

```
$ aws ec2 associate-route-table --route-table-id $routeTable2Id
--subnet-id $subnet4Id
```



## Task 2: Set Up an RDS Instance

Set up an RDS Instance for our wordpress blog. Choose the MariaDB Engine and a db.t3.micro Instance type.

Hint 1: Use the security group and open the mysql port for the Wordpress Security Group..

Hint 2: You need 2 Security groups. One for the EC2 Instance and one for the RDS Instance

```
$ rdsSgId=$(aws ec2 create-security-group --description "Database Security Group" --vpc-id $vpcId --group-name RDSSecurityGroup --query "GroupId" --output text)
```

```
$ wordpressSgId=$(aws ec2 create-security-group --description "Wordpress Security Group" --vpc-id $vpcId --group-name WordpressSecurityGroup --query "GroupId" --output text)
```

```
$ aws ec2 authorize-security-group-ingress --group-id $rdsSgId --protocol tcp --port 3306 --source-group $wordpressSgId
```

```
$ aws ec2 authorize-security-group-ingress --group-id $wordpressSgId --protocol tcp --port 22 --cidr 0.0.0.0/0
```

```
$ aws ec2 authorize-security-group-ingress --group-id $wordpressSgId --protocol tcp --port 80 --cidr 0.0.0.0/0
```

```
$ subnetGroupName=$(aws rds create-db-subnet-group --db-subnet-group-name wordpressdb --db-subnet-group-description "Subnet Group for Wordpress" --subnet-ids $subnet1Id $subnet2Id --output text --query "DBSubnetGroup.DBSubnetGroupName")
```

```
$ rdsInstanceId=$(aws rds create-db-instance --db-subnet-group $subnetGroupName --db-name wordpress --db-instance-identifier wordpress --db-instance-class db.t3.micro --engine mariadb --no-multi-az --no-publicly-accessible --vpc-security-group-ids $rdsSgId --master-username admin --master-user-password wordpress --allocated-storage 20 --query "DBInstance.DBInstanceIdentifier" --output text)
```

## Task 3: Set Up a Wordpress Instance

Wordpress:

Note: Copy all commands in this step to an external file to use these as userdata initialization for new EC2-Instances

Set up an Amazon Linux EC2 Instance with httpd and php7.4.. And install wordpress on this instance

Hint 1: You can get the latest release of wordpress under

<https://wordpress.org/latest.tar.gz>

Create a key pair, store it in a file and adjust the permissions.

```
$ aws ec2 create-key-pair --key-name "myKeyPair" --output text  
--query "KeyMaterial" > myKeyPair.pem
```

```
$ chmod 600 myKeyPair.pem
```

Create an instance in a public subnet using the Amazon Linux AMI, t3.micro instance type and the wordpress security group your created in the previous Task.

```
$ wordpressInstanceId=$(aws ec2 run-instances --image-id  
"ami-0cf6f5c8a62fa5da6" --key-name myKeyPair --instance-type t3.micro  
--security-group-ids $wordpressSgId --subnet-id $subnet1Id  
--associate-public-ip-address --tag-specifications  
'ResourceType=instance,Tags=[{Key=Name,Value=wordpress}]' --query  
"Instances[0].InstanceId" --output text)  
  
$ publicIpWordpress=$(aws ec2 describe-instances --filter  
Name=instance-id,Values=[$wordpressInstanceId] --output text --query  
"Reservations[0].Instances[0].NetworkInterfaces[0].Association.Public  
Ip")
```

Wait a few minutes for the instance to be ready. Then connect to the instance via SSH.

```
$ ssh ec2-user@$publicIpWordpress -i myKeyPair.pem
```

Install php7.4 and a webserver.

```
$ sudo yum -y update
```

```
$ sudo yum install amazon-linux-extras -y  
$ sudo amazon-linux-extras enable php7.4  
$ sudo yum clean metadata
```

```
$ sudo yum install httpd jq -y
$ sudo amazon-linux-extras install -y php7.4
```

**Download Wordpress and copy the files and folders to the Web servers directory.**

```
$ sudo wget https://wordpress.org/latest.tar.gz

$ sudo tar xvzf latest.tar.gz

$ sudo rsync -avP ~/wordpress/ /var/www/html/

$ sudo rm latest.tar.gz

$ cd /var/www/

$ sudo chown -R apache:apache html

$ sudo systemctl restart httpd
$ sudo systemctl enable httpd
$ exit
```

**Find out the RDS Endpoint and note it**

```
$ aws rds describe-db-instances --query
"DBInstances[0].Endpoint.Address" --output text
```

**Check if your wordpress page is reachable.**

```
$ echo "http://$publicIpWordpress/wp-admin/setup-config.php"
```

**Open this url in your browser and configure wordpress.**

Click “Continue”



English (United States)

Afrikaans

العربية

العربية المغربية

অসমীয়া

Azərbaycan dili

گۆنئی آذربایجان

Беларуская мова

Български

বাংলা

བོད་སྐད་

Bosanski

Català

Cebuano

Čeština

Cymraeg

Dansk

Deutsch (Österreich)

Deutsch (Schweiz)

Deutsch

Deutsch (Sie)

Deutsch (Schweiz, Du)

Continue

Click “Lets Go”



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...

Let's go!

Enter the information in the input form. Use the Database Host



Below you should enter your database connection details. If you're not sure about these, contact your host.

|               |  |  |
|---------------|--|--|
| Database Name | <input type="text" value="wordpress"/>                     | The name of the database you want to use with WordPress.                               |
| Username      | <input type="text" value="admin"/>                         | Your database username.  |
| Password      | <input type="password" value="wordpress"/>                 | Your database password.  |
| Database Host | <input type="text" value="f.us-west-2.rds.amazonaws.com"/> | You should be able to get this info from your web host, if localhost doesn't work.     |
| Table Prefix  | <input type="text" value="wp_"/>                           | If you want to run multiple WordPress installations in a single database, change this. |

Follow the installation steps



## Task 4: High Availability Setup

Create an S3 Bucket and upload the /var/www/html/wp-content folder to the bucket. Create an Autoscaling Group using the User Data of the commands you used in the previous task. Add a load balancer to your Setup.

```
$ accountId=$(aws sts get-caller-identity --query "Account" --output text)
```

Create an S3 Bucket. Hint: Use the Account ID as prefix for unique naming.

```
$ aws s3 mb s3://$accountId-wordpress
```

Create an IAM Role which allows an EC2 instance to connect to S3

```
$ cat << EOF > trust-policy.json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "ec2.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

EOF

```
$ wordpressRoleName=$(aws iam create-role --role-name WordpressS3Role
--assume-role-policy-document file://trust-policy.json --query
"Role.RoleName" --output text)
```

```
$ aws iam attach-role-policy --role-name $wordpressRoleName
--policy-arn arn:aws:iam::aws:policy/AmazonS3FullAccess
```

Create an instance profile based on this role and associate it with your EC2 Instance.

```
$ instanceProfileName=$(aws iam create-instance-profile
--instance-profile-name WordpressInstanceProfile --query
"InstanceProfile.InstanceProfileName" --output text)
```

```
$ aws iam add-role-to-instance-profile --role-name $wordpressRoleName
--instance-profile-name $instanceProfileName
```

```
$ aws ec2 associate-iam-instance-profile --instance-id  
$wordpressInstanceId --iam-instance-profile Name=$instanceProfileName
```

**Connect to the wordpress instance and upload the webserver's data to your s3 bucket.**

```
$ ssh ec2-user@$publicIpWordpress -i myKeyPair.pem
```

```
$ accountId=$(curl  
http://169.254.169.254/latest/dynamic/instance-identity/document | jq  
-r ".accountId")
```

```
$ aws s3 sync /var/www/html s3://$accountId-wordpress
```

```
$ exit
```

**Create an Autoscaling group Launch Configuration from your Wordpress EC2 Instance you created in Step 4 (use the commands you copied to that file in the user data)**

```
$ cat << EOF > user-data.txt  
#!/bin/bash  
yum -y update  
yum install amazon-linux-extras -y  
amazon-linux-extras enable php7.4  
yum clean metadata  
yum install httpd jq -y  
amazon-linux-extras install php7.4  
accountId=$(curl  
http://169.254.169.254/latest/dynamic/instance-identity/document | jq  
-r ".accountId")  
aws s3 cp s3://$accountId-wordpress /var/www/html  
sudo chown -R apache:apache html  
systemctl restart httpd  
systemctl enable httpd  
EOF
```

```
$ aws autoscaling create-launch-configuration  
--launch-configuration-name wordpress --instance-id  
$wordpressInstanceId --user-data file:///user-data.txt  
--security-groups $wordpressSgId
```

```
$ az1=$(aws ec2 describe-subnets --filter  
"Name=subnet-id,Values=$subnet3Id" --query  
"Subnets[0].AvailabilityZone" --output text)
```

```
$ az2=$(aws ec2 describe-subnets --filter
"Name=subnet-id,Values=$subnet4Id" --query
"Subnets[0].AvailabilityZone" --output text)
```

### Create Application Load Balancer and attach it to the Autoscaling Group

```
$ lbArn=$(aws elbv2 create-load-balancer --name wordpress-lb --type
application --subnets $subnet1Id $subnet2Id --security-groups
$wordpressSgId --output text --query
"LoadBalancers[0].LoadBalancerArn")
```

### Note the load balancers URL

```
$ aws elbv2 describe-load-balancers --name wordpress-lb --query
"LoadBalancers[0].DNSName" --output text
```

### Adjust the Wordpress Configuration to use the Loadbalancer and upload this file to s3

```
$ ssh ec2-user@$publicIpWordpress -i myKeyPair.pem
$ sudo vim /var/www/html/wp-config.php
```

### Find the following line

```
define( 'WP_DEBUG', false );
```

After this line add the following lines and replace LB\_URL with your actual load balancers dns name

```
define( 'WP_HOME', "http://LB_URL" );
define( "WP_SITEURL", "http://LB_URL" );
```

```
* @link https://wordpress.org/support/article/debugging-in-wordpress/ ^M
*/ ^M
define( 'WP_DEBUG', false ); ^M
define( 'WP_HOME', "http://wordpress-lb-1854823885.us-west-2.elb.amazonaws.com" )
define( "WP_SITEURL", "http://wordpress-lb-1854823885.us-west-2.elb.amazonaws.com" );
^M
/* That's all, stop editing! Happy publishing. */ ^M
^M
```

```
$ accountId=$(curl
http://169.254.169.254/latest/dynamic/instance-identity/document | jq
-r ".accountId")
$ sudo aws s3 sync /var/www/html s3://$accountId-wordpress
$ exit
```

Create a target group and a listener for your loadbalancer and attach it to your autoscaling group.

```
$ tgArn=$(aws elbv2 create-target-group --name wordpress-tg
--protocol HTTP --port 80 --health-check-protocol HTTP
--health-check-port 80 --health-check-enabled --health-check-path /
--matcher "HttpCode=200-399" --vpc-id $vpcId --output text --query
"TargetGroups[0].TargetGroupArn")

$ aws elbv2 create-listener --load-balancer-arn $lbArn --protocol
HTTP --port 80 --default-actions "Type=forward,TargetGroupArn=$tgArn"

$ aws autoscaling create-auto-scaling-group --auto-scaling-group-name
wordpress --launch-configuration-name wordpress --min-size 2
--max-size 3 --vpc-zone-identifier $subnet3Id,$subnet4Id

$ aws autoscaling attach-load-balancer-target-groups
--auto-scaling-group-name wordpress --target-group-arns $tgArn
```

Activate Loadbalancer Stickiness to use always the same instance (Session awareness)

```
$ aws elbv2 modify-target-group-attributes --target-group-arn $tgArn
--attributes Key=stickiness.enabled,Value=true
```

Fetch the Load Balancers public IP and open it in your browser.

```
$ aws elbv2 describe-load-balancers --name wordpress-lb --query
"LoadBalancers[0].DNSName" --output text
```