Design of Highly Available, Fault Tolerant, Scalable Environment for WordPress Application with Amazon Web Services

By,

Shreyas Sadanand Bhandare
Graduate Student – Rutgers University
Department of Electrical and Computer Engineering

In this article, we will start building our environment from the basics!

You've got your one year free access of AWS Free Tier Cloud services and you want to create highly available, fault tolerant, scalable environment for a WordPress web application. Follow the procedures with and try out stuff hands on with your own AWS account.

Step 1: Create IAM Users, Groups and Attach Policies
First we will create some IAM (Identity and Access Management) roles for accessing our AWS components.

It is always a good practice to **never** access your AWS account with root credentials. For that, you need to create some IAM users and groups just like a software development company has. Go to IAM and create multiple developers, multiple admins and create groups of developers and admins.

As we are not writing any development code, we will stick with the admin users for now. Give administrative access to admin group by attaching IAM administrative access policy.

If you want to trace your user's login activities to AWS console, launch a Trail with **CloudTrail** for users in within the root account and enable SNS topic for the same trail. Create an email subscription for that SNS topic. While creating a Trail, you need to create **S3 Bucket** to push the trail logs.

Step 2: Create RDS instance
Relatively small step, but creating RDS instance is important for database connection with your EC2 instance webserver.

Amazon Relational Database Service (Amazon RDS) makes it easy to set up, operate, and scale a relational database in the cloud. It provides cost-efficient and resizable capacity while managing time-consuming database administration tasks, freeing you up to focus on your applications and business. Amazon RDS provides you six familiar database engines to choose from, including Amazon Aurora, PostgreSQL, MySQL, MariaDB, Oracle, and Microsoft SQL Server.

Select MySQL, MS SQL Server or any other database as your choice. For simplicity, pick MySQL as an example. Use Multi-AZ Deployment for high availability and consistent performance across all the availability zones.

Select appropriate DB class according to your database size needs. Free Tier usually is t2 micro instance with 1GB RAM. Chose database name, username and password for your DB. Chose default VPC for your database instance. For this project, we will consider all our instances will be in default VPC so that communication can be easier. Finally, launch the multi-AZ RDS instance and you are all set!
(Optional) Create route 53 hosted zone 'internal' for your RDS instance since we want to use database internally through private DNS. Add CNAME of RDS Instance public DNS to mysql.internal. This is a good practice if you are launching new RDS instances. You only need to change in your Route 53 configurations.

Step 3: Create and Launch EC2 Instance for webserver
We will be automatically launching the instance with auto-scaling. But for starters, let's create a new basic EC2 instance which will be our webserver.

Amazon Elastic Compute Cloud (Amazon EC2) is a web service that provides resizable compute capacity in the cloud. Amazon EC2's simple web service interface allows you to obtain and configure capacity with minimal friction. It provides you with complete control of your computing resources and lets you run on Amazon's proven computing environment.

To start with, select AMI of your choice, for our purpose we will select Ubuntu 14.04 LTS. We will chose instance type to be T2.Micro. Note that T2 instances are designed to provide moderate baseline performance and the capability to burst to significantly higher performance as required by your workload. They are intended for workloads that don't use the full CPU often or consistently, but occasionally need to burst. T2 instances are well suited for general purpose workloads, such as web servers, developer environments, and small databases.

Chose default VPC and create a new IAM roles to communicate your EC2 Instance with S3 Bucket. For now just create the role. We will see how this role helps to upload and download data from your bucket to EC2.

Lastly, assign default security group to your instance, tag your instance and launch it. Add http and ssh rules to your default security groups.

After launching, ssh into the webserver with your terminal, and run 'sudo apt-get update'. Install required packages with 'sudo apt-get install apache2 php5 php5-mysql php5-curl'. If you go to public DNS of this webserver, you will see default PHP page. Now we have to change this to run our WordPress application page.

Go to /var/www directory and download WordPress application with wget. Unzip the downloaded file. We will see how to configure WordPress page in the next steps.

Step 4: (Optional) Configure Domain Name to Public DNS of Webserver with Route 53
Amazon Route 53 is a highly available and scalable cloud Domain Name System (DNS) web service. It is designed to give developers and businesses an extremely reliable and cost effective way to route end users to Internet applications by translating names like www.example.com into the numeric IP addresses like 192.0.2.1 that computers use to connect to each other. Amazon Route 53 is fully compliant with IPv6 as well.
Create a new hosted zone for your own domain name and add your webserver public DNS as CNAME to it. If you don't have a domain name, continue with default public DNS of webserver for accessing pages. In next step we will configure **ELB** (elastic loadbalancer) for our webserver instance and route the traffic to it.

Step 5: Create Elastic Loadbalancer
Elastic Load Balancing automatically distributes incoming application traffic across multiple Amazon EC2 instances. It enables you to achieve fault tolerance in your applications, seamlessly providing the required amount of load balancing capacity needed to route application traffic.

Create a new ELB for balancing web traffic to your EC2 Instances. While creating you can add all the availability zones in your region so that, load balancer will balancer the traffic amongst different AZs. This is crucial for fault tolerant environment design.

Attach default security group to your loadbalancer. Make sure it is the same as your EC2 and RDS instance. If not, then create a new security group with similar configurations.

Routing configurations are the main part while creating ELB. You can create a new Routing Group for the ELB and add your desired instances in that group. For our purpose add your EC2 webserver instance to the ELB routing target group.

Finally launch the Elastic Loadbalancer. Go to elastic loadbalancer's public DNS and it will route you to your EC2 webserver's default page.
(Optional) If you have your domain name configured, you might want to add your ELB public DNS as an Alias record to your hosted zone. So whenever you go to your domain name say 'example.com', AWS DNS server will direct you to loadbalancer which will indeed route you to your EC2 webserver instance.

Step 6: Configure EC2 Webserver Instance
In this step, let's configure our Webserver instance to serve WordPress application. First and foremost ssh into your EC2 Webserver Instance.

Before anything else, type 'nslookup mysql.internal' to check RDS instance public DNS associated with mysql.internal. If that works, move on and follow the next steps. If not, check your Route 53 hosted zone for mysql.internal

Secondly, got to /var/www/wordpress and edit wp-config.php file and change dbname, username, password and hostname (mysql.internal) and save the file changes. Next, Install MySQL mysql-client. Try using mysql client command 'mysql -hmysql.internal -uwordpressappdb -p' with password (same as username). Check if you are logged in to the mysql terminal that means you are connected to RDS Instance.

Third, go to /etc/apache and edit 000-default.conf to change default directory from /var/www/html to /var/www/worpress and it will load wordpress app page on connecting to loadbalancer (or EC2 webserver for that matter). Make sure you have wp-config.php in /var/www/wordpress directory with database connection details.

This will set up default WordPress application webpage. Go to public DNS (or your own domain name) of loadbalancer in the web browser and you can find WordPress app page loaded. Enter details such as site title, username, password, and email to it. After finishing up, log into WordPress to design website.

Step 7: W3 Total Cache Plugin for WordPress
W3 Total Cache improves the user experience of your site by increasing website performance, reducing download times via features like content delivery network (CDN) integration.

You should be still sshed into EC2 Webserver machine. Download W3 Total Cache plugin for WordPress (to upload data to AWS S3 bucket) with wget and put it in wp-content folder and activate it in the application.

Change owner of /var/www/wordpress directory to www-data. (User and group both). Edit wwp-config.php file to add W3 plugin, add .htacess and run commands to install w3 total cache plugin correctly.

Go to public DNS of loadbalancer and see your WordPress, change CDN type of W3 cache plugin to Amazon CloudFront.

Step 8: Create CloudFront Distribution
Amazon CloudFront is a global content delivery network (CDN) service that accelerates delivery of your websites, APIs, video content or other web assets.

We will create a CloudFront distribution for our WordPress applications with connection to S3 bucket. Start with creating Web CloudFront distribution. Select origin domain as S3 bucket. Create a bucket that will store your WordPress application webpages and serve it to different edge locations.

Select alternate domain name is your own domain name. Configure it to CloudFront distribution public DNS later in Route 53. If you don't have it and use loadbalancer public DNS, leave it blank.

Go to your WordPress application and configure CloudFront distribution to WordPress app and export media, plugins and everything on it.

Create new IAM User and attach full S3 bucket access and full CloudFront access policy to it. Add this User's key and secret password to WordPress app.

Step 9: Bootstrapping
Before auto-scaling, we have to specify that whenever new instance is fired with auto-scaling, we have to download same webpages from S3 to that instance. Create a new bucket for that say 'buck-bootstrap'. Attach full s3 bucket access policy to EC2-S3 Communication Role created before.

ssh into EC2 webserver and install python-pip. After that do 'pip install awscli'. That will install AWS Command Line Interface to it.

In AWS EC2 dashboard, right click on webserver EC2 and create image. Wait for Image to be available. Delete current EC2 Webserver and create new EC2 Webserver from Image (AMI).

Copy /var/wordpress folder to S3://buck-bootstrap folder using AWS CLI tools and delete the instance (now you have no webservers! Just the image)

Step 10: Auto-Scaling
There are two essential parts of Auto-Scaling. First, creating Launch Configurations and second, creating Auto-Scaling group.

A launch configuration is a template that an Auto Scaling group uses to launch EC2 instances. When you create a launch configuration, you specify information for the instances such as the ID of the Amazon Machine Image (AMI), the instance type, a key pair, one or more security groups, and a block device mapping. If you've launched an EC2 instance before, you specified the same information in order to launch the instance.

In our Launch configurations, we will be specifying what configurations our EC2 webservers will need when they are fired up. Everything else will be same as we created EC2 Instance except the script to get S3 bucket data for bootstrapping WordPress app. Make sure that Launch Configurations are creaed properly because you can't edit them just like EC2 Instances. Otherwise you have to start with a whole new launch configuration.

Once this is done we will move to create Auto-Scaling group. We can edit auto-scaling group later so don't worry about that. Remember add our created loadbalancing group. Set actions and alarms for increase scale and decrease scaling. Define minimum capacity, maximum capacity.

Try going to loadbalancer public DNS and it will launch minimum capacity EC2 webserver instances and serve the traffic from both of them by round robin method. If your auto scaling capacity is 2 and you terminate 1 instance, your auto-scaling group will launch 1 more instance in few minutes.

Step 11: Create Failover DNS

Creating Route 53 Failover: Creating Failover CloudFront Distribution with S3 bucket static page. (don't use only S3 static hosting because it needs bucket name same as your DNS hosted zone you need your S3 bucket index.html to be available over all the edge locations via CloudFront.). Create a new CloudFront distribution pointing to S3 Failover page bucket. Add Record set to your domain name as secondary record type and failover routing policy.

Step 12: Review: How Everything Works Together!

We have successfully created a scalable, highly available, fault tolerant environment for WordPress application with Amazon Web Services.

- You application will start with no EC2 instances launched.
- Auto-Scaling group will then launch minimum capacity Webserver instances with specified launch configurations.
- Auto-Scaling group will also fire up/down the instances according to set alarms.
- Loadbalancer will balance the traffic across these instances by round robin method.
- Whenever a request for webpage is made, our CloudFront distribution will make sure the data is cached near its edge location.
- If any resource failure occurs, our failover DNS CloudFront distribution will show default static error page from S3 bucket

References:

[1] http://docs.aws.amazon.com/

[2] https://aws.amazon.com/