MyBBAWS Infrastructure

Observations

This project deploys a Multi-Tier Architecture on a single AWS region; the region is intentionally not explicitly specified in the template so the region in which the stack is launched and will be used.

I assumed an externally registered domain name (not via Route53); the automation template outputs the AWS-generated DNSName of the main (WWW) load balancer and a CNAME record should be defined to point to this name.

Prior to the launch of this template into a stack, you will have to create an EC2 KeyPair in the AWS account (for SSH access).

Overview

This is a complete AWS CloudFormation template

Running the stack

To run this project in an AWS Account perform the following:

1. Create an EC2 KeyPair (required for SSH access, can't be automated by CF);
2. Launch a stack from this template with CloudFormation;
3. Go to the URL in the "WWWBalancerDNSName" output variable for the live MyBB application.

What's could be improved

1. The current template does not cover uploading user-generated files to an S3 bucket and delivering them via CloudFront. We can use AWS EFS or OpenStack Manila like implementation for sharing the file storage and management.

2. We could add external appliance like **FortiGate or PaloAlto** for more security and vulnerability protection

3. Add **Jumpbox** for external access to Web and DB Servers

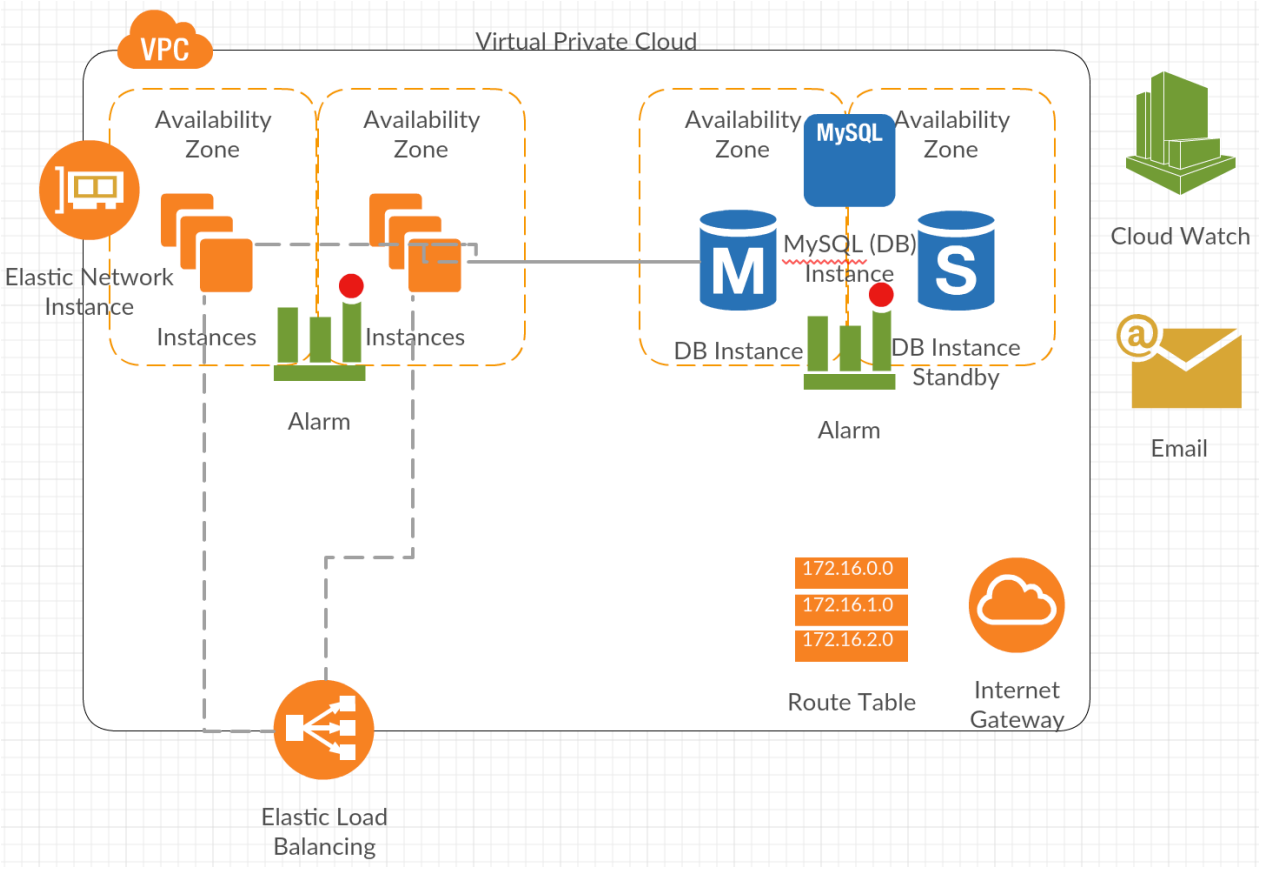
4. Add support for **ELK stack** i.e., AWS Elastic Search with Kibana for accessing infrastructure, Platform and Application Logs

5. Additional external monitoring tools could be added for **APM like AppDyamics or NewRelic.**

6. MyBB app can also be deployed in **AWS ECS** to spin each tier in separate containers.

7. Add support for **JGroups_S3_PING** to discover and synchronize the user generated files upload from EC2 instance to S3.

Overall Architecture Design



Security considerations

1. The entire infrastructure is encapsulated into a VPC; all Internet traffic goes through:
    a. **VPCInternetGateway**: gateway to Internet for VPC components (PublicSubnet*).

2. There are two main Security Groups which separate the internet-facing parts of the architecture, meaning the web-servers (and associated components), from the private parts, meaning the database cluster mainly.
    a. **PublicSecurityGroup**: HTTP/HTTPS/SSH access permitted from outside.
    b. **SecureSecurityGroup**: Database, access permitted only from web stack to DB stack.

3. There are two Subnets associated with each group (in distinct availability zones):
    a. **PublicSubnetA/B**: web servers stack, the subnets are publicly accessible.
    b. **SecureSubnetA/B**: database stack, the subnets are not publicly accessible.

4. The routing for these subnets is as follows:

    a. **PublicRouteTable**: opens traffic from the public subnets to the Internet.
    b. **SecureRouteTable**: ensures privacy for the secure subnets.

5. Currently the secure subnets have no access to the outside world (saw no point yet).

Scalability considerations

1. **Web Cluster**: the web servers are governed by an AutoScalingGroup and sit under an ElasicLoadBalancer instance for load-balancing and fault-tolerance.
    a. *Vertical scaling*: web servers can be upgraded to larger memory/compute/storage capacities without downtime.
    b. *Horizontal scaling*: The AutoScalingGroup implements policies for scaling up or down based on CPU usage metrics of the nodes (implemented via CloudWatch Alarms). This is

2. **Database Cluster**: The RDS Aurora cluster has a master (writer) instance and also a read replica at this time.
    a. *Vertical scaling*: instances can be upgraded to larger memory/compute/storage capacities without downtime.
    b. *Horizontal scaling*: up to 15 read replicas can be added without downtime; to scale write operations partitioning would be an option.

3. **Networking infrastructure**: All networking (glue) components such as ELBs, ASGs, InternetGateway, VPC Router (and probably many more) are scaled-out by the AWS ecosystem.

Availability considerations

1. **Web Cluster**: Both the ASG and ELB instances which govern the web servers currently **span over two Availability Zones** (although probably all should be used).

2. **Database Cluster**: The RDS Aurora deployment has better fail-over behavior (less downtime) than an analogous installation of ELB + RDS/MySQL instances; although downtime is still possible (in some cases of fail-over), it is much less likely (usually 100 - 200 seconds).

3. **DDoS by attack or failure**: I tried as much as possible to avoid any SPoFs (single-point-of-failure) to keep this infrastructure robust and my intent was that DDoS attacks would either:
    a. be avoided; mostly by encapsulation (VPC, subnets, traffic rules etc.) or...
    b. translate to high loads which AWS can take without experiencing service outage; obviously this is still bad, but manageable.

Monitoring and alerts

1. The ELB for the web servers commits logs into an S3 Bucket (every 5 minutes). Following CloudWatch alarms are defined:
    a. **Scaling up alarm**: send email on ASG events (when CPU usage is over 90%).
    b. **Scaling down alarm**: send email on ASG events (when CPU usage is under 50%).
    c. **Billing alarm**: send email when costs exceed a threshold USD amount (1000).
2. Alarms will notify the "OperationalEMail" address when triggered.

Further improvements

The current state of the automation template reflects what I managed to build in the time I had available, and so there are a few aspects which could (and probably should) be improved before going live with it (note that some are more important than others):

1. **Add S3/CloudFront support for uploaded files**: I intentionally did not use S3FS for storage and retrieval of uploaded files. I would definitely avoid this option if possible and use the

2. AWS PHP SDK to write the files straight to S3. Here's why:

    a. Using a FS abstraction is more difficult to control and debug, errors are less likely to be visible and we can't enforce retries or other type of error handling.
    b. Reading those files would take them from S3, through the EC2 machines to the user; this complicates the architecture badly, will perform badly and denies use of CloudFront.

3. I've taken a look over the MyBB codebase and it should not be difficult to add support for S3 file uploads, which easily extends to support **downloading files via CloudFront CDN**. This would perform and scale better. Add **CLoudWatch Alarms** for:
        i. **Abnormal Bandwidth usage** (signs of attack);
        ii. **ELB latency**;
        iii. **Database instance failures**.

4. Add **CNAME parameter** to the template so that the MyBB code will use the custom domain name instead of the public endpoint generated for the AWS balancer. The template outputs the "WWWBalancerDNSName" variable which is the public DNS-resolvable name to the main ELB instance; this should be CNAME'd to a custom domain name or handled via Route53.

5. Define **Network ACLs** for controlling traffic at subnet level; currently rules are enforced only by security groups.

6. **Remove Public IPs in PublicSubnets**: Currently I set "MapPublicIpOnLaunch" to true on public subnets for a quick way to debug the launch configuration; a **bastion** server should be used.

7. Define clear **Stack update and delete policies** to avoid downtimes during updates and to retain data (logs, databases etc.) after stack operations.

8. **Track CloudFormation calls via CloudTrail**: For additional security and as a measure of historical reference, CloudTrail should be used to log all CloudFormation API calls into a selected S3 bucket. The benefit is good and the costs should be negligible.

9. **Restrict SSH access**: By default the "SSHAllowedSources" parameter is set to "0.0.0.0/0" to allow any SSH connections to the nodes from any public IP address; this setting should be considerably more restrictive.

10. Add **parameter for MyBB Admin credentials**; currently they are hardcoded (see "Evaluation access" section).

11. **Split template** into separate stacks: (A) vpc, (B) web servers and (C) database; this would allow each stack to suffer updates and maintenance with less consequences to the other stacks; it would also make things more complex, of course.

12. **Custom AMI** or **make cloud-init scripts more private** (S3 or CodeCommit): The automation template (along with the documentation) are revision-controlled on a public **GitHub repository**; this repository is also used (for simplicity) to host the *cloud-init* installation script for MyBB 1.8.6 (including MyBB sources). It would be preferable that this whole thing would be turned into a new AMI instance or at least the codes would be placed in a more controlled environment (S3 or CodeCommit).

13. Use **all Availability-Zones** in the region; currently only two AZs are used.

14. **Outbound access for DB servers via NAT**: It could be useful to allow outbound access to the Internet for the machines residing in the private subnet of the VPC (for updates and such) by using a **NAT Gateway **.

15. Add an **ElasticCache/Memcache** deployment to improve performance.

16. **Time-based scaling**: Currently, the Auto-Scaling Groups resize solely based on CloudWatch performance metrics; rules for timed scaling could be defined to optimize costs.

17. **HTTPS support**: For a production environment I would enable HTTPS support and even try to force traffic through it (redirect "http://" traffic by HTTP 301 to "https://").