Classic Solutions Architecture Discussion

Solution architecture

How do you use all these components to make them all work together into one architecture.

We will study solution architecture and how to come up with them via case studies.

WhatsTheTime.com

Let people know what time it is. We don't need a database because every EC2 instances know the time.

We want to start small and downtime is acceptable, then scale vertically and horizontally with no downtime.

Initial solution

We start with a public t2.micro EC2 instance, and user will ask what is the time, it just spit back the current time. We attache an elastic IP address so the IP address of the EC2 instance is static.

Now users starts to come into our app, now our t2.micro can't keep up with the load, maybe we should scale it vertically and make it into m5 instances. We have to stop our app, and change our EC2 instance size to be m5. **We have downtime when upgrading our app**. This isn't great.

Now even more people come in, we scale it horizontally to three EC2 instances. But users needs to know about the IP address of those horizontally scaled EC2 instances.

To fix this, we can leverage Route 53. Set a A record to point to those three EC2 instances. Now users can access the time API just through a common endpoint, api.whatisthetime.com. With TTL of 1 hour.

Now if we are going to make an upgrade and take down one of the instances that the Route 53 points to, it is going to make some users suffer because the TTL is 1 hour. They won't be routed to other EC2 instances that are still up! They will be unhappy!

How do we remediate this?

We make our EC2 instances private, and front it with elastic load balancer with health checks. Route 53 need to have an Alias record that point to the ELB resource. Now it is working properly, no downtime for nay user because of health checks.

Now manually launching groups is tedious, we can have an auto scaling group to scale on-demand. We set min, max, and desire count of the instances.

But what happens if the ELB that we fronted with is in a availability zone that just had an earthquake? Our application will still go down! To solve this, we can deploy our ELB in multi-AZ, say AZ 1-3. Our auto scaling will group will also launch instances in different AZ. Now it is highly available great!

Now after optimizing the architecture, you will switch to thinking about cost saving. You can reserve capacity for cost saving! Reserving minimum capacity of our auto scaling group we can save lots of money!

Now this is good architecture. We are considering 5 pillars for a well architect ed applications: Cost (reserved instances for optimized cost + ASG), performance (Vertical scaling, ELB, adapt performance over time), reliability (Route 53, multi-AZ deployment), security (Security group to link ELB to EC2), operational excellence.

MyClothes.com (stateful app)

Now let's try to make a stateful app. This e-com web app will have a shopping cart and we need some place to store all these user informations. We want to keep our web app as stateless as possible, and user should not lose their shopping cart when they refresh their pages.

Details such as address should be in the database.

We will have the same set up from the Whatsthetime app with ELB, ASG, Multi-AZ, and Route 53. Now whenever the user add to chart, the page refreshes and they lose their data because they are talking to different EC2 instances from before they are redirected. How do we remediate this? We introduce stickiness so that user will be always talking to the same EC2 instances.

But if the EC2 instance is terminated data will still be lost, so stickiness isn't a complete solution.

We introduce server session, we set a cookie called session_id, and use ElastiCache to store user session. So that as long as the user have the same session_id, it will retrieve the same data for that user.

Now we can also introduce Amazon RDS to store user data in a database. But now there is too many reads what do we do? We add read replicas to the RDS, we can have up to 5 read replicas. We can also then add on top of it lazy read with ElastiCache, but this pattern require code change to your repository, however, it is more efficient since frequently accessed data will be in ElastiCache and doesn't need to hit RDS constantly.

Now how do we make it Multi AZ? In order to survive disasters. Route 53 we don't have to worry about it since it is highly available already. ELB make it multi-AZ, ASG make it multi-AZ, ElasticCache also have multi-AZ if you use Redis, RDS you can do multi-AZ to have a standby instance if the master goes down.

For security group, we restrict traffic only from the resources that it is fronted with.

MyWordPress.com (stateful app)

We want to make a scalable WordPress website, and they should have the capability to upload and access picture upload.

RDS layer with multi-AZ for handling user data. Or we can have Aurora MySQL to scale better than RDS.

Storing images we will do it with EBS initially. Image will have to go through ELB to EC2 then EC2 will store it into EBS volume. Problem is when we start scaling horizontally, what if image is stored into one of the EBS volume but not the other? Then user won't be able to access the image that they have uploaded to one of the EBS volume.

To solve this we will use EFS instead of EBS, it scales automatically as you use more storage, and the storage is shared between all many EC2 instances.

Instantiating applications quickly

For EC2 instances, we can use a golden AMI (or also called custom AMI), which contains pre-installed software and packages that is needed to run your applications. All your other EC2 instances can be created from this AMI at a much faster rate.

You can also use user data for dynamic configurations.

Elastic Beanstalk will combine both golden AMI and user data to quickly spin up your applications

For RDS databases restoring it from a snapshot is much faster.

EBS volumes also restoring it from a snapshot will be much faster.

Elastic Beanstalk

So far the architecture is ELB with EC2 in multi-AZ, then we have RDS and ElastiCache for caching frequently read data and session data.

Most of the application follow these type of structure, and if we are going to deploy many application then it is going to be a pain to deploy these manually for every application!

Most of the application follow these same architecture, ALB + ASG mainly, so Elastic Beanstalk provide this one way of deploying the code without you having to worry about provisioning all these resources yourself. All you have to do is write the code.

Elastic Beanstalk is going to use all of the component that we have seen before, it is a managed services so it will automatically handle capacity provisioning, load balancing, scaling, application health monitoring. We still have full control over the configuration but it puts less burden on the developer.

Beanstalk itself is free, but the resources that it manages will not be! Will be priced accordingly.

Beanstalk support lots of platforms! Python, Java, you name it, even if your platform isn't on it by default, you can just create it yourself.

Components

Application: The collection of Elastic Beanstalk components

Application version: An iteration of your application code

Environment: Collection of AWS resources running application version (running only one version at a time)

Tier: Web server environment tier vs worker environment tier, you can make multiple environment

So the process is you create application, upload it with a version number, and you launch into an environment, then you can upload updated version and relaunch it with the updated version.

Web server tier: The traditional architecture that we know, EC2 instances is managed by auto scaling group and is fronted with ELB, it will be deployed under a DNS name from ELB.

Worker environment: No clients accessing EC2 instances directly, the EC2 instance will be consuming SQS messages that comes from SQS queues. You push messages to the SQS queue to kick start the process.

Deployment modes

Single instances: Good for development, an EC2 instance with Elastic IP

High availability with load balancer: This is good for production, this is the traditional architecture in AWS that we see already. EC2 instances managed by ASG in multi-AZ, fronted by ELB.

Revision #2
Created 2023-02-15 20:43:39 UTC by Tamarine
Updated 2023-02-17 01:42:11 UTC by Tamarine