Classic Solutions Architecture Discussions

Uday Manchanda

August 22, 2021

1 WhatsTheTime.com

- Stateless Web App - allows people to know the time
- Start off with t2.micro, decide to vertically scale to m5.large. Creates downtime. Users not happy.
- Scale horizontally, more instances. No downtime when scaling. However users need to keep track of each instance's IP address
- Leverage Route53. Set up an A record with TTL of 1 hour. Users can reach domain name and R53 will respond with a list of IP addresses which correspond to the IP addresses of the instances.
- The problem is that if one of the EC2 instances goes down, the users sending requests to that instance will not be able to connect for the duration of the TTL.
- Use a load balancer with health checks. Set up SGs so that HTTP traffic flows between ELB and EC2 instances.
- WE can still use DNS but we use an Alias record. Which will point from R53 to the ELB. Users connect to the ELB, which then redirects traffic to the EC2 instances.
- With health checks we can minimize or eliminate downtime.
- Add an ASG since adding or removing individual instances is hard to do. The EC2 instances will be managed by the ASG. Set up rules/alarms to scale in/out depending on the amount of load.
- Lets say there's a disaster and AZ1 goes down.
- Implement multi-AZ. Make the ELB be multi AZ. Assume 3 AZs. The ASG will span across those 3 AZs.
- Reserve capacity to save money. If we know that two instances must be running at all times, we can reserve a third instance in AZ3.

1.1 Well Architected Application

- Cost
- Performance
- Reliability
- Security
- Operational Excellence

2 MyClothes.com

- Stateful Web app - allows people to buy clothes online with a shopping cart. Users should not lose their shopping cart and their details stored in a DB. Details stored in a DB.
- Start off with ASG, R53, Multi AZ ELB
- ELB talks to EC2 instances. New request from user to ELB hits a different instance so the shopping cart is lost. Solution: enable ELB stickness on the load balancer itself.
- This allows every request between the ELB and EC2 to go to the same EC2 instance. However, if an EC2 instance gets terminated, this creates a problem because the shopping cart will get lost.
- Solution: user cookies. Let the user (browser) store the contents of the shopping cart, not the EC2 instance.
- When the browser talks to the ELB, it knows what's in the shopping cart so when it talks to the EC2 instances, all of them know what's in the shopping cart.
- This is stateless - EC2 instances don't need to know what happened before because the browser will tell it.
- However, the HTTP requests are getting bigger. and there is a security risk since the cookies can get altered. EC2 instances should validate the content of the cookies.
- Cookies must be small (¡4 KB)
- Solution: server session. Send the session id from R53 to user in the cookies that the browser sends to the ELB.
- Create an elasticache cluster in the background. Send session id, talk to EC2 instance, say we're gonna add ”thing” to the shopping cart

- EC2 instance will add cart content into ElastiCache and the id to retrieve. Cart content will be a session id
- When the user sends another request, the session id will get sent to another EC2 instance. EC2 instance will lookup that EC2 instance
- ElastiCache stores/retrieves session data. (Alternative is DynamoDB)
- Store user data with RDS. Stuff like address, name, CC. Long term stuff. When we talk to our EC2 instance it will talk to that RDS instance to retrieve user information.
- Scaling reads with RDS master (writes) and RDS read replicas. So when the EC2 wants to lookup infomration from the DB it can read from the read replicas.
- Scaling reads an alternative way with write through. User talks to EC2, EC2 looks at cache and asks for information.
- If information is not present, cache will read from RDS and add it to the cache. Next time, when the information is present, the EC2 instance (and other ones) will get a cache hit since they will be able to read from the cache. Less traffic on RDS.
- Surive disasters with Multi AZ. Enable multi AZ on ELB, RDS, and ElastiCache.
- Security groups: Allow HTTP/HTTPS traffic on 0.0.0.0/0 to ELB. on the EC2 side, we restrict it to only allow from ELB. Restrict traffic to ElastiCache SG from EC2 SG. Same with EC2 and RDS.

2.1 Summary

- ELB stick sessions
- WEb clients for storing cookies and making our web app stateless
- Elasticache for storing sessions and caching data. Multi AZ.
- RDS for storing user data and read replicas for scaling reads. Multi AZ.

3 MyWordPress.com

- Stateful, fully scalable web app. Access and display picture uploads. User data and blog content stored in MySQL.
- Same architecture from previous section, replace RDS with Aurora.
- Storing images with EBS. EBS volume attached to EC2 instance. Problem with scaling.

- Image stored in EBS1 will not be available in EBS2.
- Solution: store with EFS Network File SYstem Drive. Creates ENI (elastic network interface) into each AZ which can be used for all our EC2 instances to access the EFS drive.

4 Elastic Beanstalk

- Most web apps have the same architecture (ALB+ASG), most devs just want their code to run across different apps and envs
- Beanstalk is a developer centric view of deploying an application on AWS
- Uses all the components: EC2, ASG, ELB, RDS
- Essentially a managed service. Infrastructure handled by AWS, developer handles code
- Application: collection of Elastic Beanstalk components (envs, versions, configs)
- Environment: Collection of AWS resources running an application version. Tiers: Web Server Env and Work Environment Tier
- Web Server Tier vs Worker Tier
  - Web Tier: Web Environment around ELB
  - Worker Tier: Worker Environment around SQS Queue