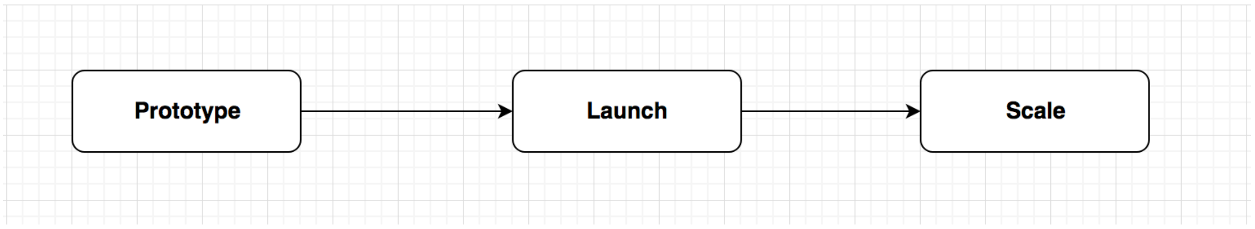
System Design



Assumptions.
The system has already passed the Prototype phase and hugely appreciated in the Launch, and thats why it is able to attract the funding.

By this time the architect must have realized some of the insights of the webapp.

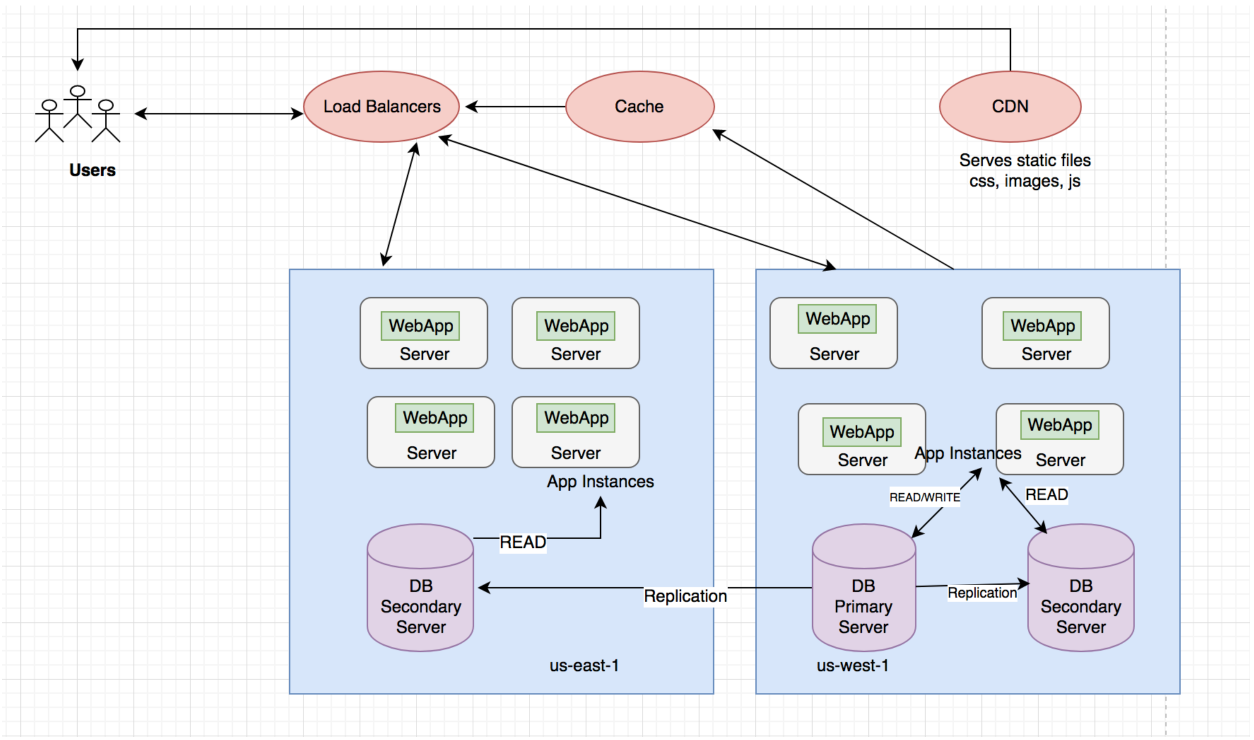
Stage - 1
It is very important to understand the nature / workload of the web app, since it is a Social media based.The following criteria it fulfills.
1. There are many Read and Writes(It can be posts, images, videos, updates).
2. There can be abrupt increase in traffic because of some scheduled event (Some Music concert, Elections etc).
3. There can be sudden increase in traffic because of some unschedule event (Some disaster, some breaking news).
4. General trend of traffic over the weekend or holidays as people have free time, and the following day as people start posting about it.

Stage - 2
Once you understand the data completely, We have to start thinking about the Data store strategies and the different tradeoff which can be.
1. Performance (Latency tolerance)
2. Durability (Data Loss tolerance)
3. Consistency (Weird result tolerance)
4. Availability (Downtime tolerance)

Stage - 3
Scaling
Specialized Choices
Use transactional data store for - Consistent data
Use Memchache for the - Static, Non changing data
Redis for the - Processing pipeline
Ways to scale
Replication
Cache
Shard

Based on the following inputs, We can use AWS and its different available services to design the above stages to build a complete solution.

- ELB with autoscaling could help to provide flexible ec2 instances with uncertainty around when and how much this demand. Cloud-front and S3 are charging on the traffic and storage you actually used. You would only pay for what you demand.
- EBS with snapshots could recover the state of web or app servers from disaster. RDS with snapshots could recover the data of database from disaster.
- RDS could provide flexible performance on various cpu, memory, iops and network. It also provides managements from console, cli and api.
- ELB with autoscaling could help effective distribution of load. It would scale out or in automatically and immediately.
- Cloud-front could serving static media from anywhere on the earth with very low latency by hundreds edge locations. Route 53 and EC2 in the region which is closest to the customers also help to provide quality access.



- Health check of ELB would provide transparent failover by switch traffic to the working ec2 instances. Moreover RDS in master-slave mode could also provide high availability by switching to the slave when the master is down automatically.
- VPC with Security Group of ELB, EC2 and RDS could protect data in transit. Encryption, versioning, snapshots and IAM on EBS, RDS and S3 could protect data at rest.
- IAM could provide flexible permissions for everyone in the delivery team. With properly configure, resources could be access only from who really needed and permitted. For example, the DBA would only access the RDS, the ops would only access the ec2, manager could access all resource.
- Glacier provide long-term, secure, durable object storage for data archiving. Object Lifecycle Management of S3 is really meeting the requirement.

- Cloudformation could help to build a full environment with a resource template. So compose the template which describe blueprint architecture at first, manage and replicate the resource as many as you want later.

Detailed discussion of the Architecture.

• This social media website being dynamic in nature always gets the request such as user register and static request to fetch some image, posts or any other content. Dynamic requests would be past through the web server (apache) and be processed by the app server (PHP). They require amount of computing resources. EC2, ECS and Lambda all three could provide computing resources. A dynamic request always ends up with persisting or querying something which is processed by a database (MySQL). While there are plenty services could work as database in aws, we got only 2 solutions under restriction of migration cost. Between hosting a MySQL on EC2 with EBS and RDS, we choose RDS because of hosting a high performance MySQL cluster would cost a lot on EC2, EBS, maintaining and optimization. Otherwise, RDS takes nearly zero effort to both scale up for better performance and replicate for more availability. Static requests always served by a web server (apache) which heavily depend on the performance of the EBS. Compare to the expensive high IOPS EBS, using S3 and Cloud-front to serve the static contents would be cheaper and more efficient. Run a EC2 cluster in the region which is closer to the customer would speed up the connection. At last, we should use Cloudwatch to metric the performance of our whole system. With help of monitoring, we could get an insight on our bottleneck or runtime issue. For example, We could use autoscaling to solving the lack of computing resources.

- Route 53 for name resolving. When customer start to request the domain, Route 53 response with a record of an aws ELB for dynamic request or redirect to the Cloudfront for static content. Together with Route 53, we could easily combine with other aws services and migrate later.

- Putting the static content into S3 and mapping them to Cloudfront automatically deliver them to edge locations. It means the customer would spend minimal time to see the website open in the browser. The company could develop features to upload static contents to S3 in their webapp later instead of uploading them manually. The S3 provide rich api and sdks to help the development. Compare to storing static content into EBS which is hard to scale and keep the ec2 stateful, S3 with Cloudfront is simpler to manage and scale and faster to access.

- Consider the performance between Beanstalk with S3 and self-composed EC2 with EBS for a typical LAMP webapp, we decide to go with Beanstalk which is easier configure, consume, scale and deploy. Elastic Beanstalk is perfect match a standard LAMP webapp. It provide mature configuration relative to a self composed architecture with VPC, ELB, autoscaling, security group, ec2 and ebs, which brings 2 benefits, separate the dynamic and static request, focus on the actual delivery other than infrastructure. The PaaS (Beanstalk) would greatly help a startup to get rid of the complexity of infrastructure and smoothly migrating to cloud. When dynamic requests from customers come, ELB would choose a health ec2 to serve. If all ec2 servers in an az stop working, ELB would redirect the requests to the ec2 servers in another az. When traffic is growing, autoscaling would start more ec2 servers to handle the requests. Vice versa, autoscaling would terminate ec2 servers to meet the minimal requirement. Instead of deploying to each ec2 instance or making snapshot or ami manually, the Beanstalk also greatly simplified the shipment of new version, upload the zip directly from the console or to the S3, just one click to finish the deployment. The Beanstalk would deliver the app to each working ec2 without any effort at once. The Beanstalk dashboard is more friendly for a newbie than Cloudwatch. It provide enough key metrics and easy to understand and operate than amounts of views and panels scattered in the services. Beanstalk causes less security issues than common ec2 service combination on key pair and AMI managements. At last, Upgrading the operation system and platform software is safe and smooth through the dashboard.

- RDS for data storage. The ec2 servers would read and write the master mode RDS in one az, another slave mode RDS in different az is standby to synchronize the data and prepare to serve when the master is down. When the master is up again, sync the latest data from the slave and then switch back the connections. Managing snapshots of RDS is a simple and reliable way for backup and recover data. With proper setting, for example, a high profile master and a low profile slave combination which meets the balance of ROI and availability.

- IAM for identify and access management. In the previous "Requirement" part, there comes with different environments. So the company need to have one root account for each environment and different IAM setting within. It is recommended to not manage with root account. Instead, designing roles with access token or MFA for people who need to access the resources. As developers need to deploy webapp over cli and editors would upload files to S3 over console, then the manager should give minimal permissions to the role and relate it with the specific members.

- Cloudtrail could record the people who operating the resource. We could use it to audit the operations and track member for security issues.

- VPC with security groups are provisioning protect from unauthorized access and threatens.T here are thress security groups in the architecture, one for ELB and one for ec2. They restrict the access from Internet and only transfer data between ELB and EC2 instances in the vpc by default. The EC2 instances behind the ELB are not connected with Internet directly to avoid threatens. The last security group is for RDS which restric the access except from authorized ec2 instances. The traffic in the aws are all encrypted by SSL/TLS. The company could purchase and apply certificates on their domain to provide extra security on the Internet by https.

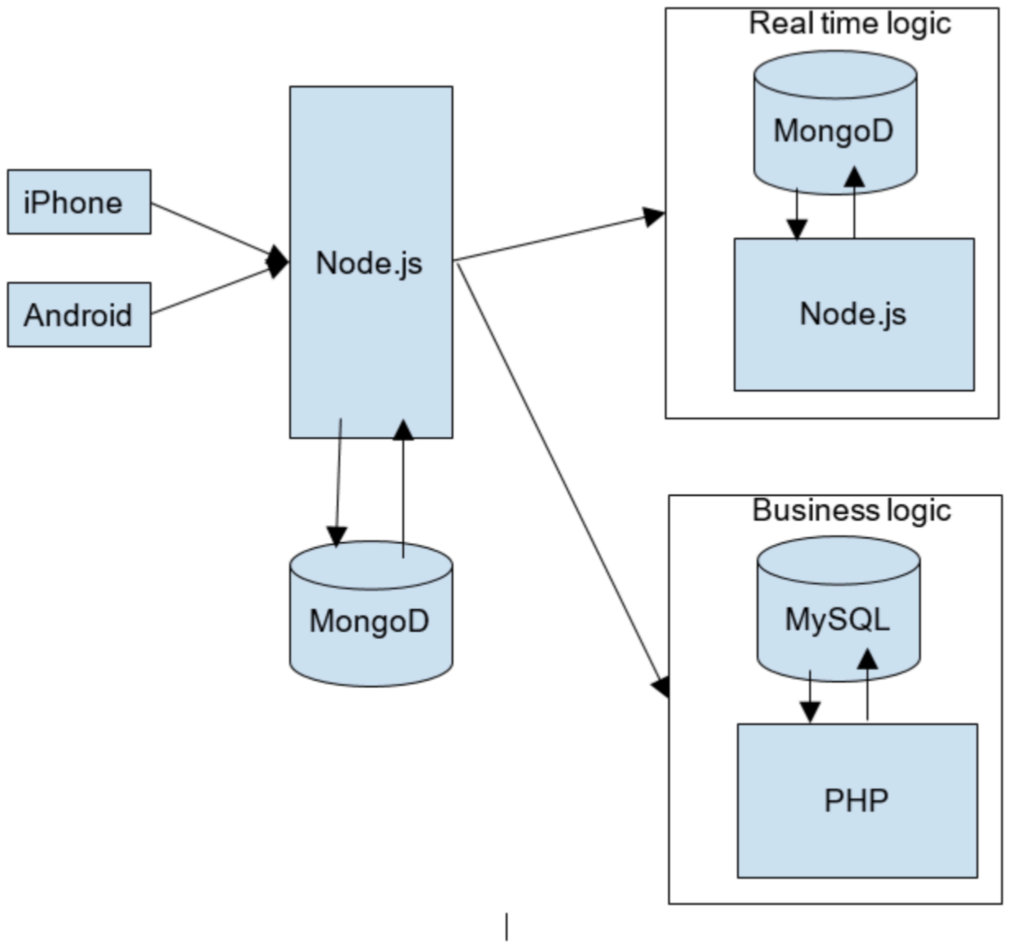
- Together with Cloudwatch and Cloudformation, you could find any issue of the system at a very first time and reproduced it in a separate environment at once.

- Trusted Advisor could help to revise the security issues.

- We can effectively use the Amazon Code pipeline for the CI/CD automation and the step functions for the micro service architecture.

- Static contents in S3 could only be access through https. Together with versioning, encryption and IAM, they are protected from losing, leaking and unauthorized operation. They would be archived to Glacier by Object Lifecycle Management. They take fewest cost to maintain the data.

A simple highly scalable mobile architecture looks something like this



With combination of above services of aws, we could build a manageable, secure, scalable, high performance, efficient, elastic, highly available, fault tolerant and recoverable architecture for the company.

R SANTOSH KUMAR