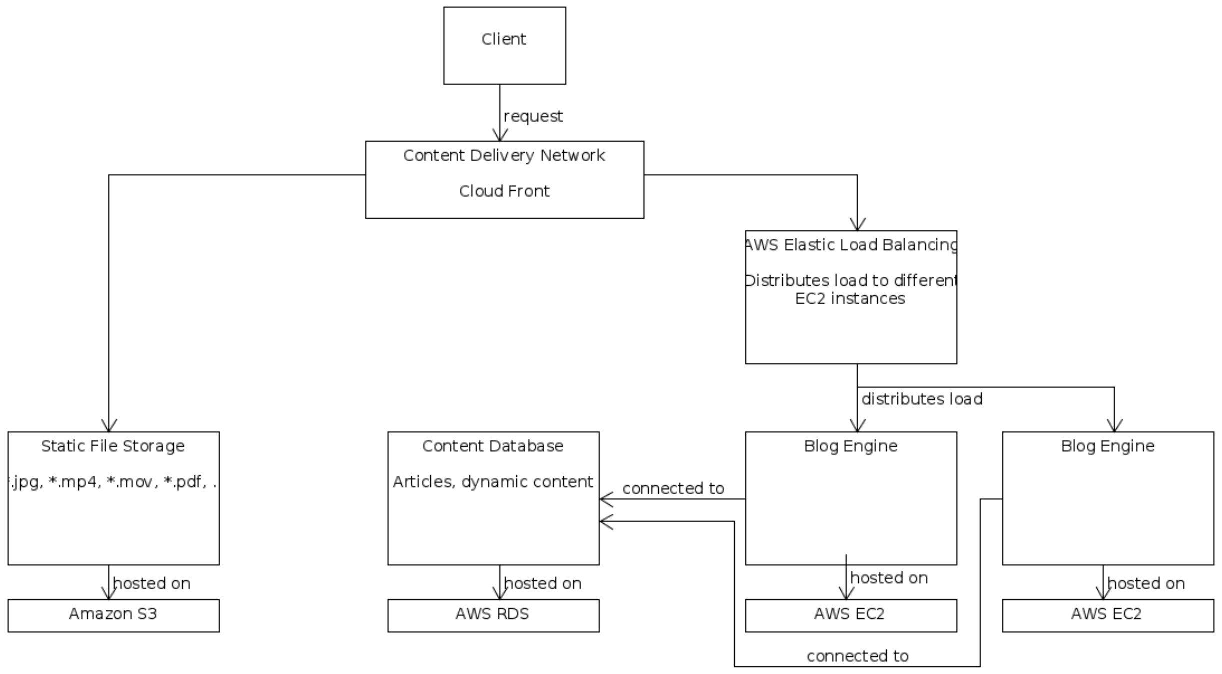
Assignment 1 - Group 2

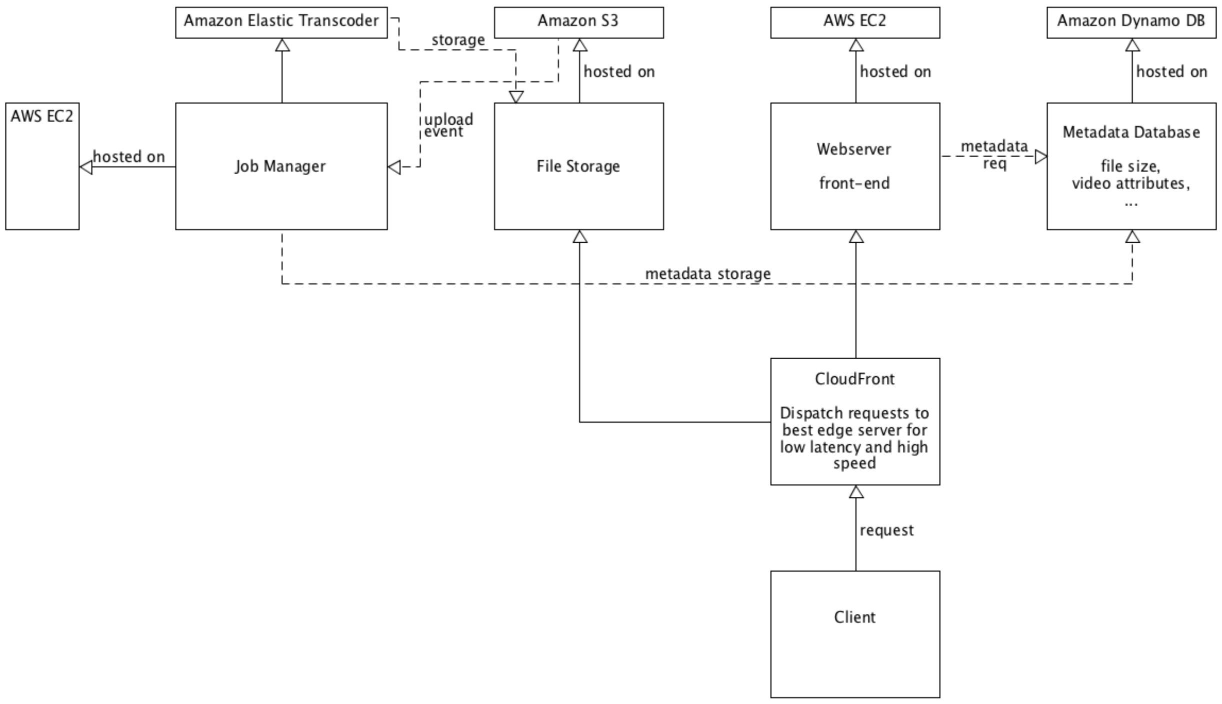
Felix Ebinger	2719182
Thomas Reinhardt	2635525
Julian Ziegler	2556772

1.2.a - Blog as three-tier Web application



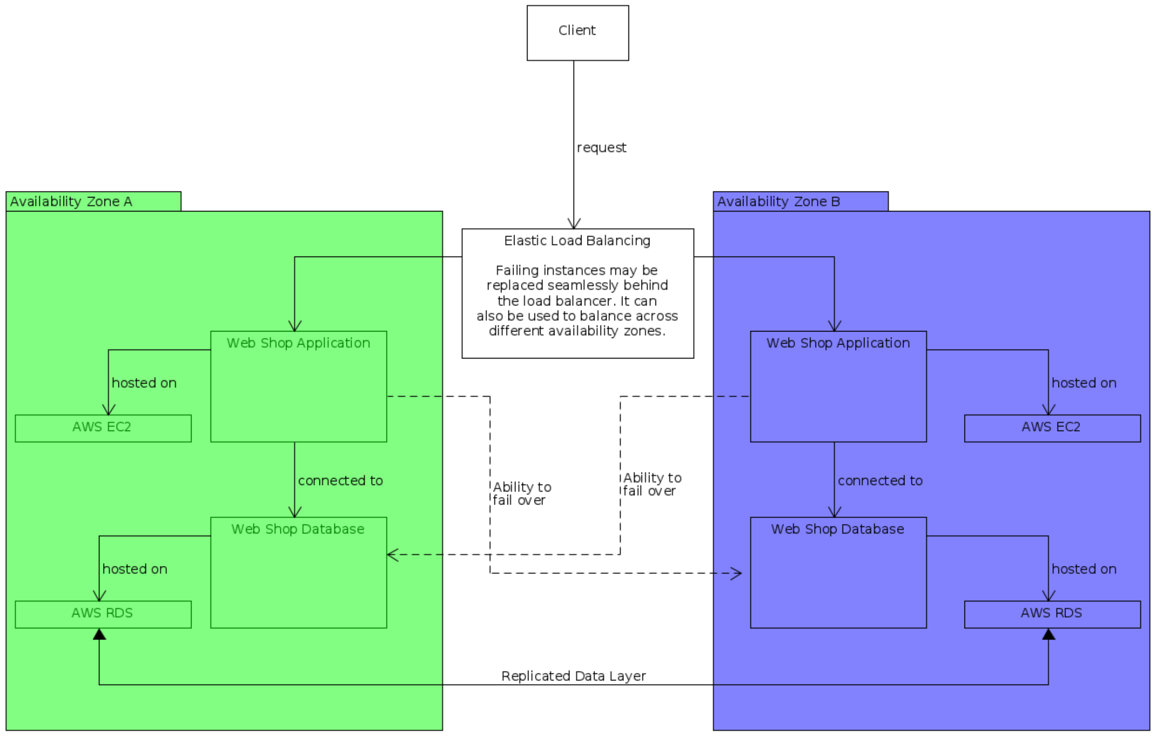
The blogging engine is hosted on different EC2 instances. The load is distributed by the load balancer Elastic Load Balance. All blogging engines use the same content database hosted on AWS RDS. The static files like images or videos are hosted on Amazon S3. All data are delivered by the CDN CloudFront.

1.2.b - Backend for video sharing platform



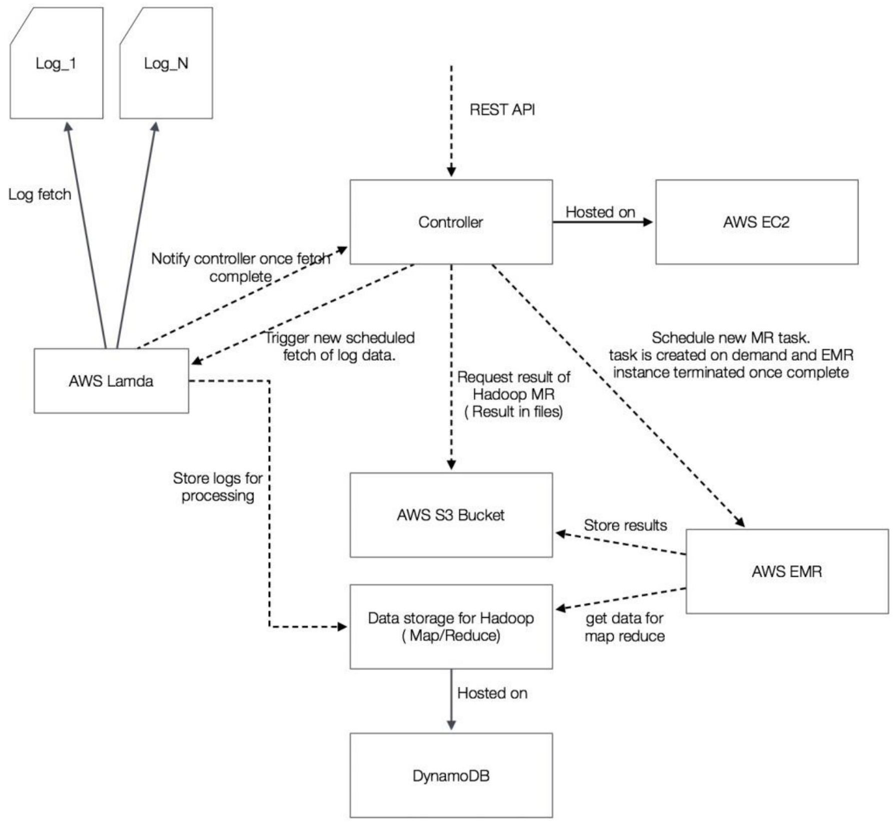
A webserver hosted on EC2 instances provides the client front-end for operations such as viewing or uploading videos. Necessary metadata is queried from DynamoDB by the front-end. An uploaded video is stored with Amazon S3, which in turn triggers a Job Manager to process the newly uploaded video. This Job Manager is queuing jobs, which are then processed by Amazon Elastic Transcoder. The transcoded videos are stored on S3, while the corresponding metadata is inserted into DynamoDB.

1.2.c - Zero-downtime Web shop



The web shop application is hosted on EC2 instances. The database of the web shop is hosted on AWS RDS instances. Both parts are duplicated in at least two different availability zones. So it's possible that each part or even a complete data center may fail and the web shop can still be used. The different database instances are synchronized. The load is distributed by the load balancer Elastic Load Balancing. The load balancer is also able to detect failing instances and distributes the traffic to the remaining health instances.

1.2.d - Log analyzer for a Web application


Log_1
Log_N
REST API
Log fetch
Controller
Hosted on
AWS EC2
Notify controller once fetch complete
Trigger new scheduled fetch of log data.
Schedule new MR task. task is created on demand and EMR instance terminated once complete
AWS Lamda
Request result of Hadoop MR (Result in files)
Store logs for processing
AWS S3 Bucket
Store results
AWS EMR
Data storage for Hadoop (Map/Reduce)
get data for map reduce
Hosted on
DynamoDB

For the architecture of our log analyzer, we assume that we do not need a elastic cluster for other than the MapReduce module. It is assumed that log files only need to be analyzed in a (fixed) time interval, e.g. once every day.

Controller:
The architecture is designed to use a central controller hosted on a AWS EC2 instance). It used user interaction and task scheduling / triggering. A user can add websites to analyze, or request results or meta/status data via a REST-Full API. The controller schedules new MapReduce tasks in a pre-defined time interval. It will trigger a AWS Lambda instance to fetch log files from the defined web servers and create a new Amazon EMR instance for analyzing.

Amazon Lambda:
Cost effective, time-triggered service by amazon: ".. runs [...] code in response to events and automatically manages the compute resources ...". Runs job to fetch logs from all known sources. Loads files into a NoSQL DB. Runs job only on external trigger (from controller) "... pay only for the requests served and the compute time required to run your code."

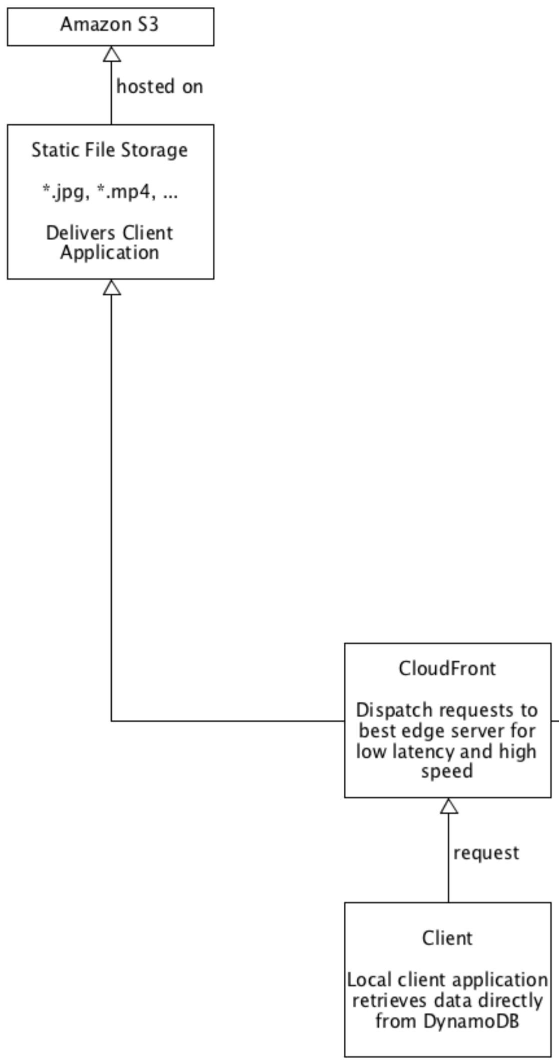
Amazon DynamoDB:
NoSQL DB for storing (unprocessed) log files. Filled by Lambda job, used by EMR. Only raw files are stored, results of MapReduce stored elsewhere. Functions as "kind of" HDFS (EMR uses Hadoop).

Amazon Elastic MapReduce (Amazon EMR):
MapReduce service by AWS. EMR with a Hadoop instance is created by controller once all logs have been fetched. Raw data is loaded from DynamoDB, processed data, i.e. results are stored in a S3 bucket. EMR terminated itself after all MapReduce tasks are finished (available a an option for EMR instance).

S3 bucket:
Storage area for MapReduce results. Results are accessed by controller and presented to user.

This architecture only uses 2 permanently active resources (EC2 and S3). All other components are created on demand and stopped once they finish their tasks. This enables a low-cost but always available analyzing service. A user can access results / manage used web applications at any time.

1.2.e - Blog as two-tier Web application


Amazon S3
hosted on
Static File Storage
*.jpg, *.mp4, ...
Delivers Client Application
Amazon Dynamo DB
hosted on
Content Database
Articles, Comments, ...
CloudFront
Dispatch requests to best edge server for low latency and high speed
request
Client
Local client application retrieves data directly from DynamoDB

Client requests are dispatched to their best edge location through CloudFront CDN. Amazon S3 delivers a client application written with AWS SDK. This client application is querying (and inserting) data directly using services like Amazon S3 or Amazon DynamoDB, without the need of an additional webserver in between.

1.3. & 1.4 – AWS Account & WordPress

Done

1.5. – Discussions

Now you have multiple WordPress instances. However, each instance has a separate address. What would you have *to do to put a load balancer in front of the WordPress instances, so that all instances can be accessed using a single address?*

→You may use Elastic Load Balancing as load balancer in front of the WordPress instances. It can be used to distribute incoming traffic between EC2 instances. You can select the EC2 instances you want to add to the load balancer. You can use the load balancer by visiting the IP address or DNS name of the load balancer. Additionally you may also use Auto Scaling to launch or terminate EC2 instances automatically based on the needs of your application.

Does it make sense at all to "load-balance" between these WordPress instances? Remember that currently each WordPress instance has its own separate database in the background.

→ No, it does not. The current approach provides two different blogs, operating on their own database respectively. However, two WordPress instances operating both on the same database could balance the webserver load for a single website. The database would remain a bottleneck though.