Cloud Distribution Network - Tunesheap

Victor Hawley

Robert Lis

Idea

- Concurrently serve “large” files to large number of users who request them.
- Every popular service on the internet requires an infrastructure that can deal with such loads.
- Provide an easy-to-use API (that also scales) to request these files and their metadata.
- Build a system that can easily scale even further during periods of high usage.

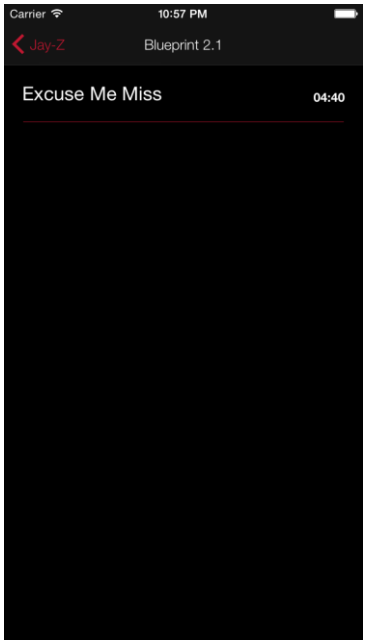
Problem

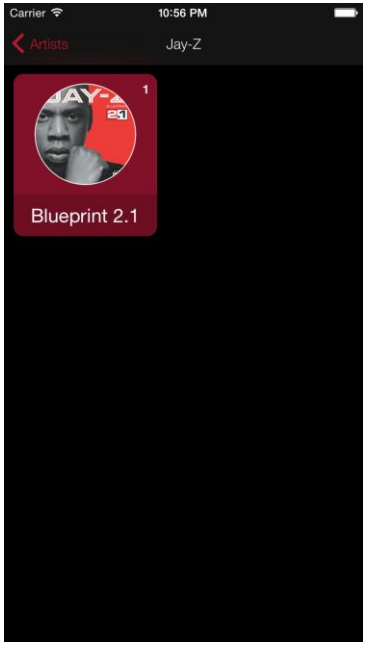
- Single servers cannot easily handle an influx of requests simultaneously.
- We have a working API to interface with the data.
- How do we scale the API to work at a larger scale (if our service becomes popular)?

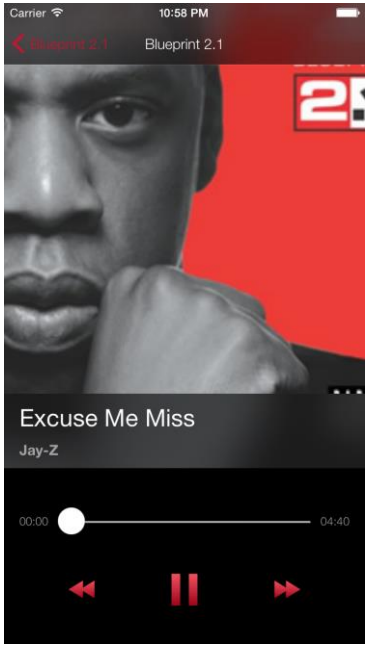
- Single server averages:
  - 7.3 secs/successful request at 1000 simultaneous requests (100% success rate)
  - 25.7 secs/successful request at 10000 simultaneous requests (52.5% success rate)
  - This isn’t good enough!

Our project - Tunesheap

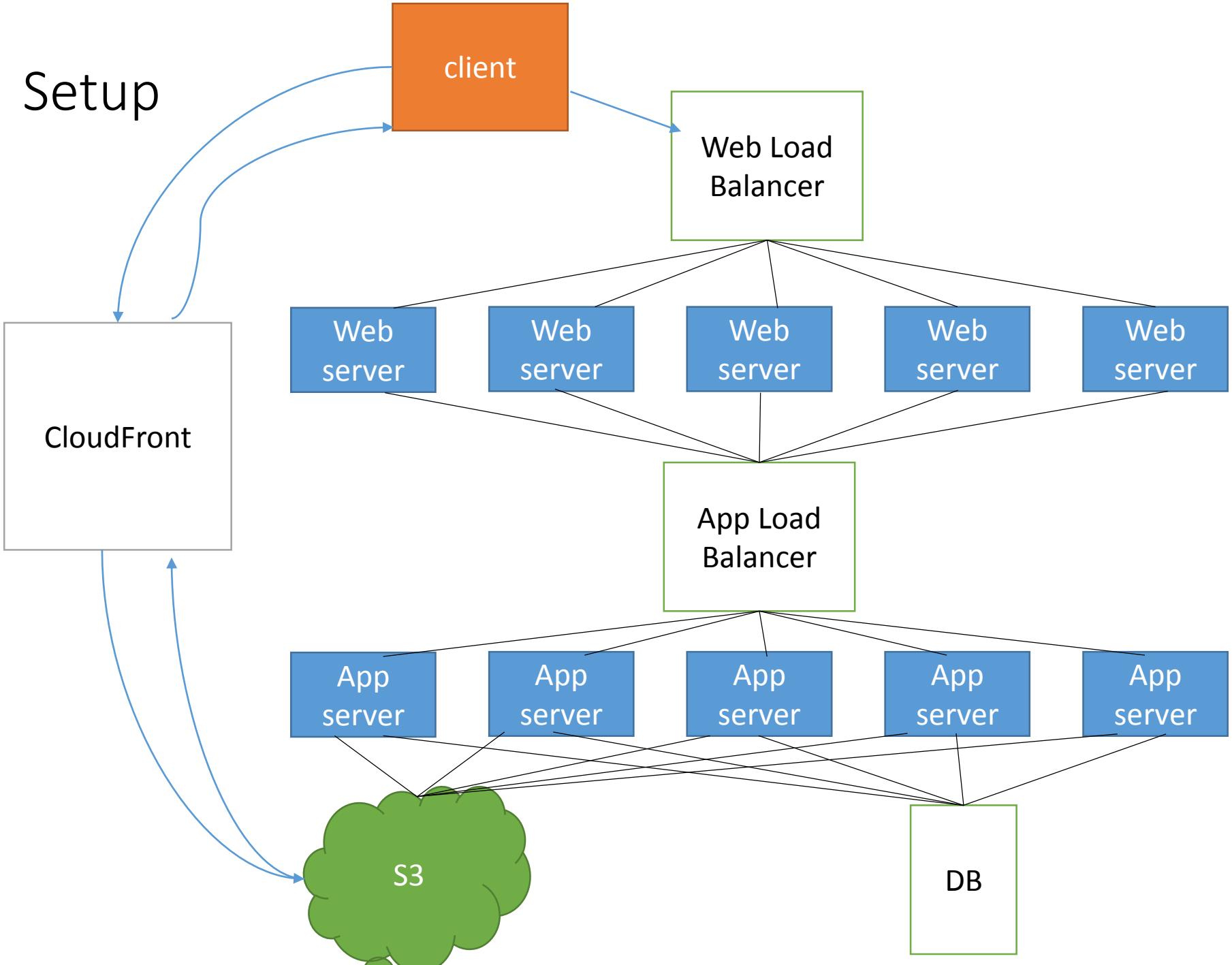
- Music streaming service (similar to Spotify)
- RESTful API (Ruby on Rails) for clients to interface with songs and their metadata (JSON objects).
- Amazon (EC2, S3, CloudFront, RDS) to scale the simple implementation to something that can handle an influx of requests.
- iOS client as a proof-of-concept client.







Setup

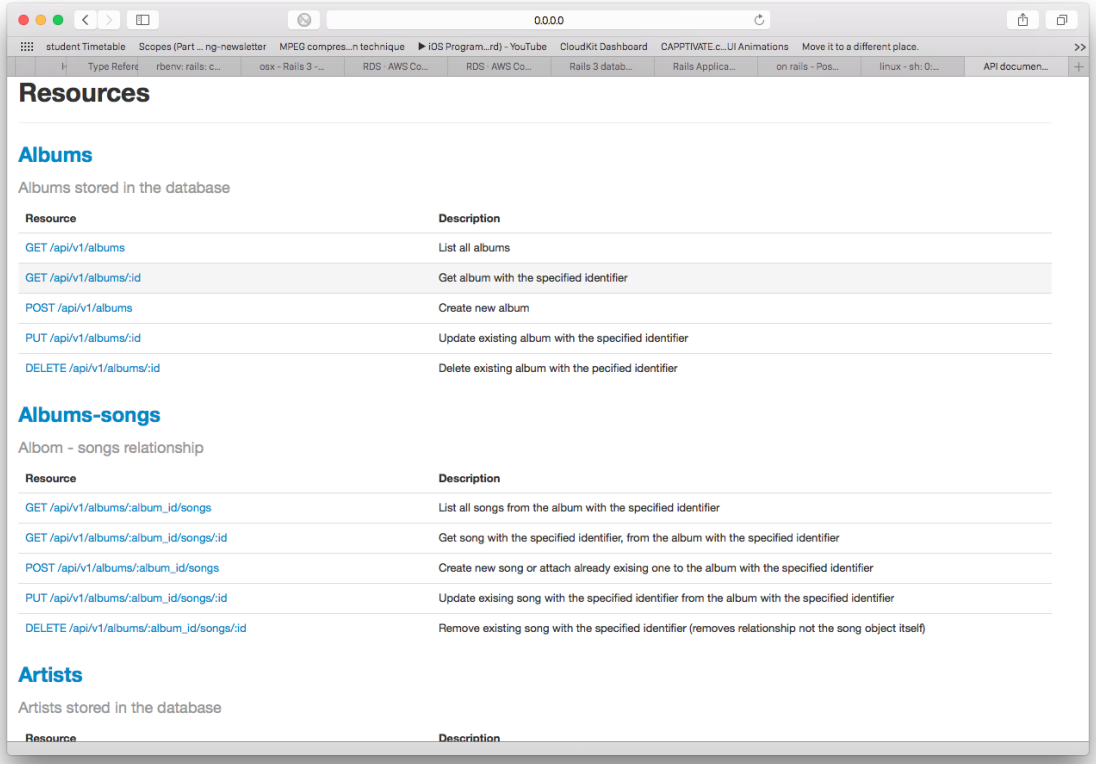

client
Web Load Balancer
CloudFront
Web server
Web server
Web server
Web server
Web server
App Load Balancer
App server
App server
App server
App server
App server
S3
DB

Setup (in detail)

- EC2 load balancer to split up web requests among multiple web servers.
- Web servers configured with nginx to communicate with the app servers
- Another load balancer for the app servers.
- Ruby on Rails API running on for app servers.
- S3 for song storage
- CloudFront to speed up retrieval of songs.
- Can add more systems as needed.

- Puppet and custom OS images used for deployment of machines.

Example API call



`GET: /api/v1/artists`

Response:


```
{
  "artists": [
    {
      "id": 4,
      "name": "Jay-Z",
      "country": "USA",
      "description": "New York native. ",
      "dob": "1969-12-04 00:00:00 UTC",
      "website": "www.rocafella.com",
      "picture_url": "https://tunesheap-content.s3.amazonaws.com/4-artist-picture"
    }
  ]
}
```

Testing/Conclusion

- Python script utilizing work_queue and condor to send HTTP requests and time the results.
- **Measuring scaled version of the app vs. an implementation using a single server**
- Each individual request's results are used to calculate the total performance (aggregate time for all requests to finish, including overlap)
- We are testing a wide range of the amount of requests and amount of workers performing those requests

- Still finalizing the infrastructure of the system, but major improvements are expected.

What's next?

- Finalize infrastructure and gather data
- memcached
- Elastic search
- Scale the database.

Questions?