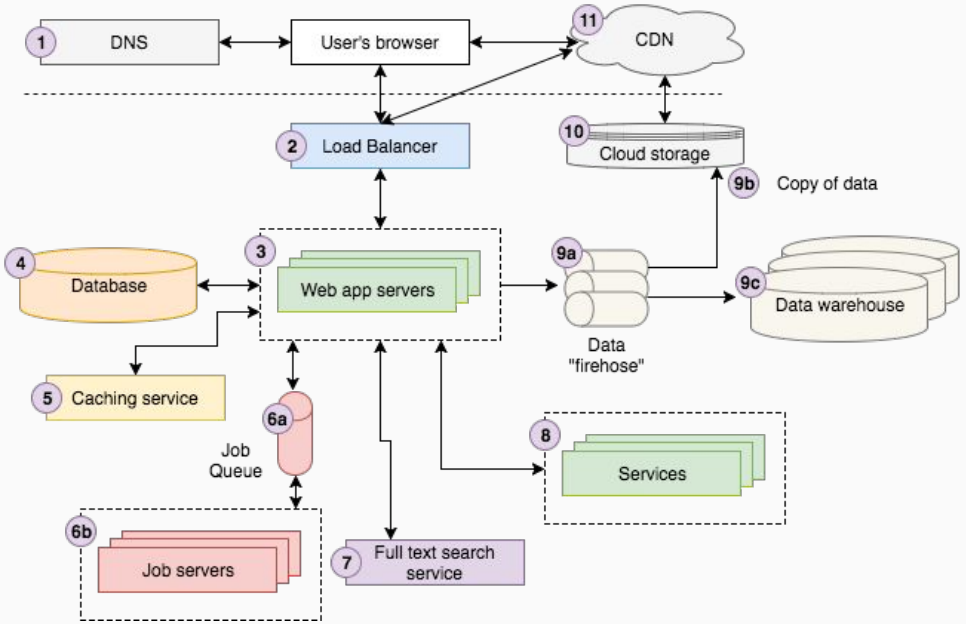
Web Architecture 101

웹 개발자로 시작할 때 알았으면 했던 기본적인 웹 아키텍처 개념들

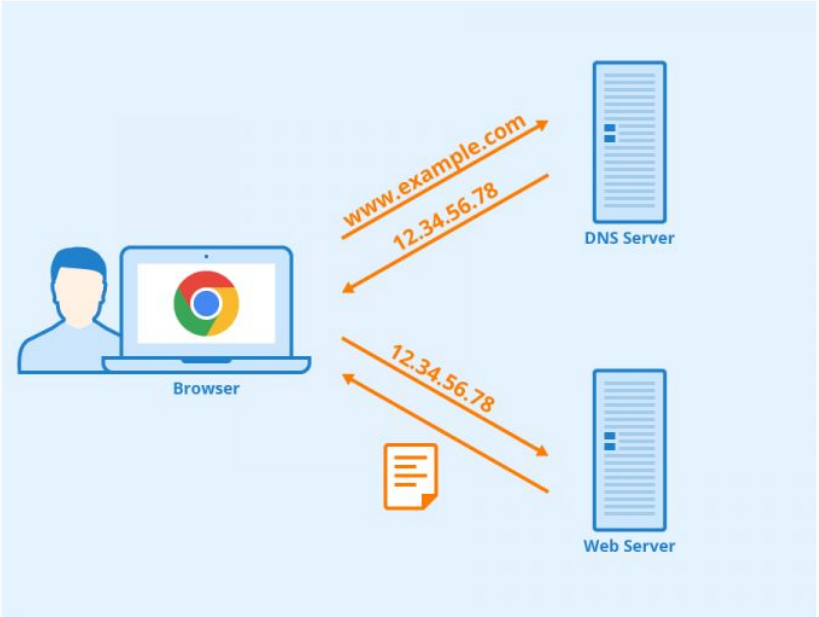
모던 웹 애플리케이션 아키텍쳐 개요



사용자가 구글에서 “짙고 아름다운 안개와 숲 속의 햇살”을 검색한다. 첫 번째 결과는 Storyblocks, 우리의 메인 사진 저장 및 벡터 사이트에서 나온다. 사용자는 결과를 클릭하고 브라우저는 사진 상세 페이지로 이동한다. 보이지 않지만, 브라우저 내부에서는 DNS 서버에 Storyblocks에 어떻게 접속할 수 있는지 물어본 후 Storyblocks에 접근을 시도한다. 브라우저의 요청은 우리의 로드 밸런서에 도착하고, 서비스를 운영하기 위해 동작 중인 10여 개의 서버 중 하나를 랜덤하게 선택해서 요청을 처리한다. 웹 서버는 캐싱 서비스에서 필요한 이미지 정보를 가져온 후 더 필요한 정보는 데이터베이스에 요청한다. 사용자에게 전달한 이미지의 컬러 프로필이 아직 만들어지지 않았음을 인지한 후 “컬러 프로필” 잡(job)을 잡 큐에 보낸다. 잡 서버는 큐에 추가된 것들을 비동기적으로 처리한 후 데이터베이스에 결과를 적절히 업데이트한다. 다음, 우리는 전체 텍스트(full-text) 검색 서비스에 사진의 제목을 전달해서 비슷한 사진들을 찾으려고 한다. 사용자가 Storyblocks의 멤버로 로그인했다면 그의 계정 정보를 계정 서비스에서 가져온다. 일련의 작업들이 끝난 후, 데이터 firehose에 페이지 뷰 이벤트를 발생시켜서 우리의 클라우드 스토리지 시스템에 기록하고, 그 정보는 분석가들이 비즈니스와 관련된 질의에 답하는 데 사용할 수 있도록 데이터 저장소(warehouse)에서 사용된다. 서버는 이제 HTML로 화면을 렌더링한 후 로드 밸런서를 통해서 사용자의 브라우저로 보낸다. 페이지는 CDN에 연결된 우리의 클라우드 스토리지 시스템에서 가져오는 자바스크립트와 CSS 파일을 포함하고 있다. 그래서 브라우저는 그 콘텐츠를 가져오기 위해 CDN에 접속한다. 마지막으로, 브라우저는 사용자가 볼 수 있는 콘텐츠를 렌더링한다.

1.DNS

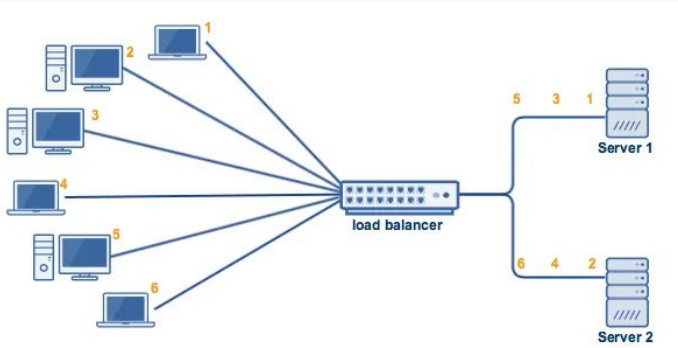
DNS는 “Domain Name Server”의 약자며
월드 와이드 웹(WWW)이 가능하도록 만드는 기반 기술이다.
가장 기본적인 레벨의 DNS는 도메인 이름(예. google.com)에서
IP 주소(예. 85.129.83.120)로의 키/값 조회를 제공한다.
도메인에서 IP 주소를 찾기 위해서는 DNS가 필요하다.



2.Load Balancer

수평적 확장은 더 많은 장치를 새로 추가하는 것이고,
수직적 확장은 이미 사용하고 있던 장치의 성능을 높이는 것이다.

수직적 확장은 언젠가 한계에 부딪힌다.



로드 밸런서는 수평적 확장이 가능하도록 만드는 마술이다.
로드밸런서는 들어오는 요청을 복제/미러링 된 수많은 애플리케이션
서버 중의 하나로 연결하고 서버의 응답을 다시 클라이언트로 보낸다.
모든 서버는 특정 요청을 같은 방식으로 처리해야 하며,
로드 밸런서는 이들 서버에 과부하가 걸리지 않도록 들어오는 요청을
적절히 분배해주는 일을 한다

개념적으로 로드 밸런서는 꽤 직관적이다.

3.web application server

사용자의 요청이 들어오면 핵심 비즈니스 로직을 실행하고 그 결과를 HTML에 담아 브라우저로 보낸다.
이 일을 하기 위해서는 보통 데이터베이스, 캐싱 계층, 잡 큐, 검색 서비스, 기타 마이크로 서비스,
데이터/로그 큐(queue) 등 무척 다양한 백엔드 인프라와 데이터를 주고받아야 한다.
사용자의 요청을 처리하기 위해 최소 2번, 보통 그보다 많은 횟수로 로드 밸런서에 연결될 것이다.
앱 서버 구현은 특정 언어(Node.js, Ruby, PHP, Scala, Java, C# .NET, …)를 선택하고 그에 맞는 웹 MVC
프레임워크(Express for Node.js, Ruby on Rails, Play for Scala, Laravel for PHP, …).를 선택해야 한다는
사실을 알아야 한다.
하지만 이들 언어와 프레임워크에 대해 깊이 알아보는 것은 이 글의 범위를 뛰어넘는다.

4.database server

모든 모던 웹 애플리케이션은 정보를 저장하기 위해 한 개 이상의 데이터베이스를 사용한다.
데이터베이스는 데이터 구조를 정의하고, 새로운 데이터를 삽입하고, 데이터를 찾고,
데이터를 수정하거나 삭제하고, 데이터로 연산을 수행하는 등의 일을 한다.
대부분은 웹 앱 서버는 잡 서버의 역할을 하는 데이터베이스 서버와 직접 통신한다.
거기에 더해서 각각의 백엔드 서비스는 애플리케이션의 다른 영역과 분리된 자신만의
데이터베이스를 가지고 있을 수 있다.

5.cashing service

캐싱 서비스는 정보를 거의 O(1) 시간 안에 찾을 수 있는 단순한 키/값 데이터 저장소를 제공한다. 애플리케이션은 캐싱 서비스를 통해 자원이 많이 소모되는 연산의 결과를 다시 계산하지 않고 캐시에서 가져옴으로써 효율을 높인다. 애플리케이션은 데이터베이스의 쿼리 결과, 외부 서비스 호출 결과, 주어진 URL의 HTML 등을 캐시에 저장한다.

- 구글은 일반적인 검색어인 'dog'나 'Taylor Swift'를 사용한 검색을 매번 실행하기보다는 결과를 캐시 한다.
- 페이스북은 당신이 로그인할 때 포스트 데이터, 친구 목록 등 많은 데이터를 캐시 한다.
- Storyblocks는 React 서버 사이드 렌더링으로 생성된 HTML, 검색 결과, 검색어 입력 자동완성 결과 등을 캐시 한다.

가장 널리 사용되는 캐싱 서버 기술 2개는 Redis와 Memcache다.

6.job queue, job server

거의 모든 웹 애플리케이션은 사용자의 요청에 대한 응답과는 직접적인 관련이 없는 작업을 백그라운드에서 비동기적으로 실행할 필요가 있다. 예를 들어 구글은 검색 결과를 얻기 위해 인터넷에서 데이터를 크롤링하고 인덱싱할 필요가 있다. 이는 당신이 검색을 요청할 때마다 실행되지 않으며 구글의 검색 엔진은 비동기적으로 웹을 크롤링하고 있다.

비동기적인 작업을 가능하게 하는 여러 가지 아키텍쳐가 있지만 가장 널리 사용되는 것은 “잡 큐” 아키텍처다. 이는 2개의 컴포넌트로 구성된다. “잡”으로 이루어진 큐, 그리고 큐에 들어있는 잡을 실행하는 1개 이상의 잡 서버가 그것이다.

잡 큐는 비동기적으로 실행될 잡 목록을 저장하고 있다. 대부분 애플리케이션이 결국에는 우선순위가 적용된 큐가 필요하게 되지만 가장 단순한 것은 first-in-first-out(FIFO) 큐다. 앱이 정기적인 일정이나 사용자에 의해 발생한 잡을 실행할 필요가 생기면, 그에 알맞은 잡을 큐에 추가하기만 하면 된다.

잡 서버는 잡을 처리 한다. 그들은 잡 큐를 가져와서 할 일이 있는지 확인하고, 있다면 큐에서 잡을 뽑아내서 실행한다.

7.full text search service

대부분은 아니지만 많은 웹 앱이 사용자가 텍스트 입력(보통 '쿼리'라고 불리는)을 입력을 하면 검색을 하고 가장 '관련 있는' 결과를 보여주는 기능을 제공한다. 이 기능을 가능하게 하는 것은 보통 "전체 텍스트 검색"이라고 불린다. 전체 텍스트 검색은 쿼리 키워드를 포함하는 문서를 빠르게 찾기 위해 역 인덱스(inverted index)를 활용한다.

Documents (Photo titles)	
id	**title**
1	Man running in the mountains
2	Mountains with snow
3	Man running marathon

Inverted Index	
keyword	**photo_ids**
man	1, 3
running	1, 3
mountains	1, 2
snow	2
marathon	3

오늘날 가장 인기 있는 전체 텍스트 검색 플랫폼은 Elasticsearch지만 Sphinx 또는 Apache Solr 같은 다른 선택지도 있다.

8.service

앱이 특정 규모에 도달하면 별도의 애플리케이션으로 분리해서 운영하기 위해 '서비스'가 생기게 된다.
외부에는 노출되지 않지만, 앱과 다른 서비스와는 연동된다.Storyblocks을 예로 들자면 몇 개의 운영, 계획 서비스를 하고 있다.

- 계정 서비스는 우리의 모든 사이트의 사용자 정보를 저장해서 교차 판매 기회를 더 쉽게 제공하고, 더 일관적인 사용자 경험을 가능하게 한다.
- 컨텐츠 서비스는 우리의 모든 비디오, 오디오, 이미지의 메타데이터를 저장한다. 또 콘텐츠 다운로드 인터페이스와 다운로드 이력을 보여주는 기능을 제공한다.
- 결제 서비스는 고객이 카드로 결제할 수 있는 인터페이스를 제공한다.
- HTML → PDF 서비스는 HTML을 PDF로 변환하는 간단한 인터페이스를 제공한다.

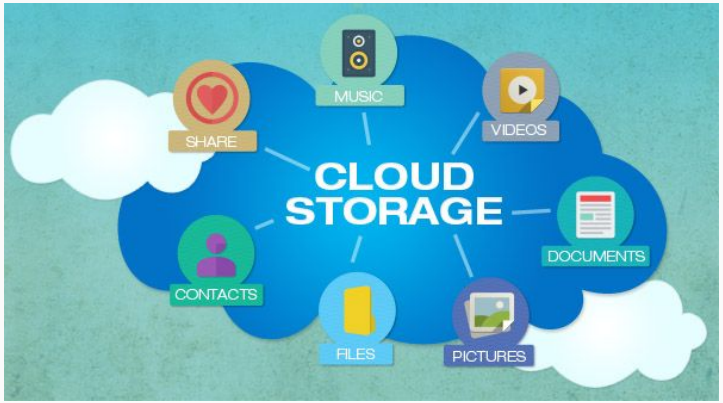
9.data

최근 거의 모든 앱은 특정 규모에 도달하면 데이터를 제어, 저장, 분석하기 위해 데이터 파이프라인을 사용한다. 전형적인 파이프라인은 3개의 주요 단계를 가진다.

1. 앱은 보통 사용자 상호작용으로 발생한 데이터를 데이터 'firehose'라 불리는 곳으로 전달한다. 그것은 데이터를 받아들이고 처리할 수 있는 스트리밍 인터페이스를 제공한다. 가공되지 않은 원시 데이터는 변형되거나 (transformed) 추가 정보와 함께(augmented) 다른 firehose로 전달된다. AWS Kinesis와 Kafka는 이러한 작업을 위한 대표적인 기술이다.
2. 원시 데이터와 최종 데이터는 모두 클라우드 스토리지에 저장된다. AWS Kinesis는 원시 데이터를 AWS의 클라우드 스토리지(S3)에 저장할 수 있도록 매우 쉽게 사용할 수 있는 'firehose'로 불리는 설정을 제공한다.
3. 변형/추가된 데이터는 종종 분석을 위해 데이터 웨어하우스(DW)에서 로드된다. 우리는 AWS Redshift를 사용한다. 큰 기업에서는 Oracle이나 기타 독점적인 웨어하우스 기술을 사용하고 있지만, 스타트업 업계에서는 RedShift를 많이 사용하고 있으며 점유율도 계속 오르고 있다. 만약 데이터가 충분히 축적되었다면 Hadoop같은 NoSQL MapReduce 기술이 분석을 위해 필요하게 될 것이다.

10.cloud storage

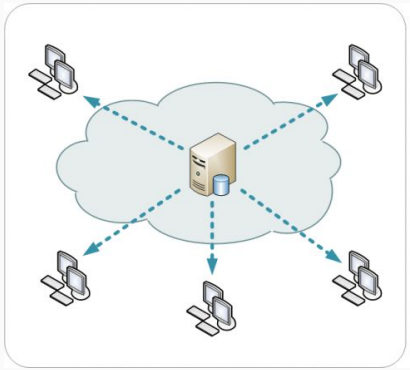
AWS에 의하면 “클라우드 스토리지는 인터넷을 통해 데이터를 저장, 접근, 공유할 수 있는 단순하고 확장성 있는 방법”이다.
당신은 로컬 파일 시스템에 저장할 수 있는 거의 모든 것을 RESTful API를 사용해서 HTTP를 통해 클라우드에 저장하고 접근할 수 있다.
아마존의 S3는 현재로써는 가장 인기 있는 클라우드 스토리지이며 Storyblocks에서는 비디오, 사진, 오디오 데이터, CSS, 자바스크립트, 사용자 데이터 등을 저장하기 위해 S3에 크게 의존하고 있다.

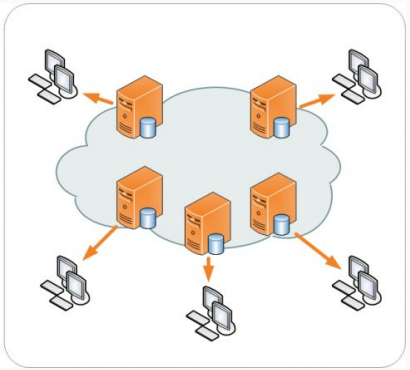


11.CDN

CDN(Contents Delivery Network)은 지리적 물리적으로 떨어져 있는 사용자에게 컨텐츠 제공자의 컨텐츠를 더 빠르게 제공할 수 있는 기술을 말합니다.

기본적으로 사용자가 원격지에 있는 서버(Origin Server)로 부터 Content(예. Web Object, Video, Music, Image, Document 등)를 다운로드 받을때 가까이 있는 서버에서 받는 것보다 시간이 오래 걸리므로, 사용자와 가까운 곳에 위치한 Cache Server에 해당 Content를 저장(캐싱)하고 Content 요청시에 Cache Server가 응답을 주는 기술입니다.





모던 웹 애플리케이션 아키텍쳐 개요