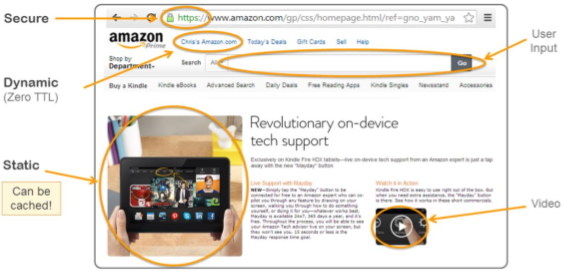
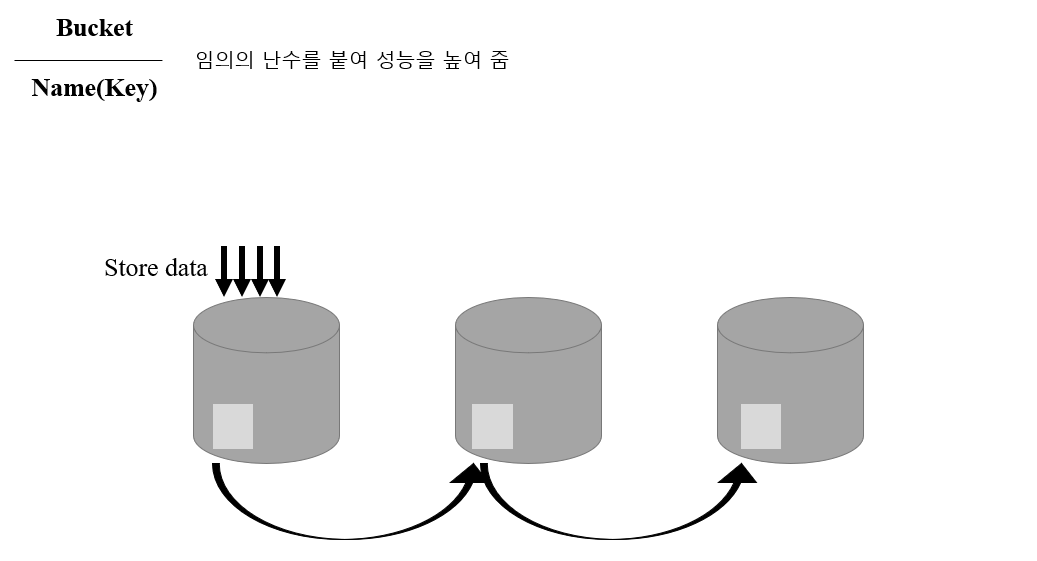
1. 파트1: 웹 기반 애플리케이션 작동 방식
   1. 웹 기반 애플리케이션의 현실
      1. 애플리케이션을 사용할 수 없게 되면
         * 매출 손실로 이어짐
         * 고객 출성도와 브랜드 이미지에 타격 입음
      2. 성능
         * 높은 페이지 뷰
         * 더 나은 고객 경험
         * 높은 전환율
   2. 모든 컨텐츠 제공(사진 출처: 강의 자료)



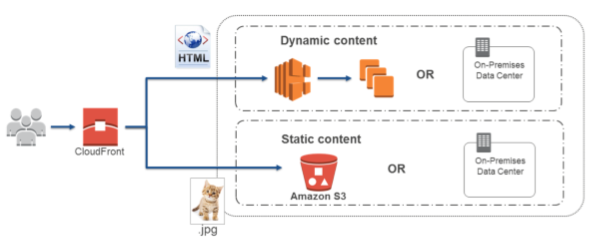
1. 파트2: 정적 자산을 Amazon S3에 저장
   1. 웹 스토리지 패턴
      1. 문제
         * 웹 서버에서 대규모 파일을 전송하는 것은 네트워크 지연 시간 측면에서 문제가 될 수 있음
         * 사용자 생성 콘텐트는 모든 웹 서버에 분산되어야 함
      2. 솔루션
         * 정적 자산 파일을 Amazon S3에 저장하고 파일을 Amazon S3에서 제공
         * Amazon S3에 저장된 객체는 사용자가 직접 액세스 가능(설정 - 퍼블릭)
   2. Amazon S3에 저장하는 이유
      1. 내구성이 높음
         * 리전 내 여러 시설에 객체를 **중복 저장**
         * **무결성 확인**
         * **모든 버전을 보존, 검색 및 복원** 가능
      2. 일관성 모델  
         일관성 🡪 쓰는 것과 읽혀 지는 것이 동일해야 한다.
         * 새로운 PUT에 대한 **쓰기 후 읽기** 일관성 제공
         * 덮어쓰기 PUT 및 DELETE에 대한 **최종 일관성** 제공
   3. 최대한 활용하려면 어떻게 해야 하는지
      1. 리전 선택
         * 물리적 **거리**에 따라 달라짐
         * 컴퓨팅 및 다른 AWS 리소스와 코로케이션하면 성능에 영향을 줌
      2. 이름 지정 체계에 주의
         * 버킷의 일관된 성능을 원하는 경우
         * 일상적으로 초당 ---개 PUT.LIST/DELETE 또는 초당 \_\_\_개 GET 요청을 초과하는 버킷 성능을 원하는 경우
   4. 버킷에 지속적으로 로드가 걸린다면?

임의의 난수를 붙여 성능을 높여 줌. 이 임의의 난수는 DynamoDB에 저장됨.

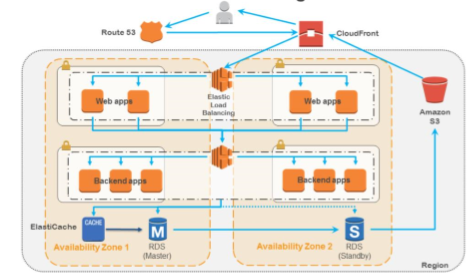
**보조 인덱스**를 유지하고 모든 키 이름 해시! DynamoDB 🡪



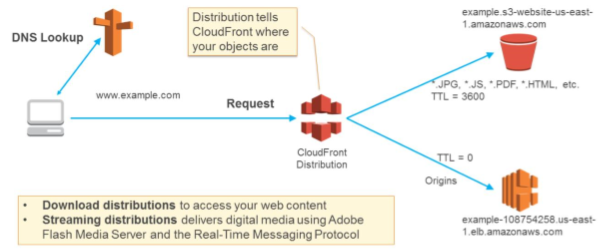
1. 파트3: 자주 액세스하는 자산을 CloudFront에서 제공(사진 출처: 강의 자료)
   1. 모범 사례: 캐싱 사용: 아키텍처의 **여러 계층**에서 캐싱 구현, **성능을 높**일 수 있음.
   2. 정적 재사용 가능한 콘텐츠 개시



* 1. AWS 클라우드 아키텍처: 호스팅

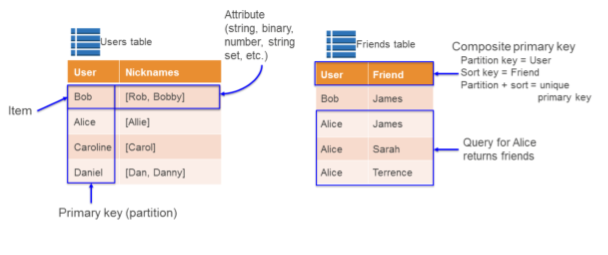


* 1. 어떻게 도움이 되는지
     1. CND(콘텐츠 전송 네트워크)
        + 전 세계에 캐시 됨.
        + 지리적 근접성은 짧은 지연시간을 의미
     2. 주요기능
        + 리전 엣지 캐시
  2. CloudFront 배포



* 1. 웹 사이트 속도 높이는 방법
     1. 캐시 제어 헤더 사용
     2. 만료 기간: 오리진의 파일에 캐시 제어 헤더를 설정하여 설정
  2. 콘텐츠 만료 방법
     1. Time To Live(TTL)
     2. 객체 이름 변경
     3. 객체 무효화

1. 파트4: 비관계형 데이터를 Amazon DynamoDB와 같은 NoSQL 데이터베이스에 저장
   1. 모범 사례: 적절한 DB 솔루션 선택
      1. 다양한 관계 데이터베이스 엔진, SQL 솔루션, 데이터 웨어하우징 옵션 및 검색 최적화 데이터 스토어 중에서 선택
      2. 고려사항
         * 읽기 / 쓰기 요구
         * 총 스토리지 요구 사항
         * 일반적인 객체 크기 및 이러한 객체에 대한 액세스의 특성
         * 내구성 요구 사항
         * 지연시간 요구 사항
         * 지원해야 하는 최대 동시 사용자 수
         * 쿼리의 특성
         * 필요한 무결성 제어의 강도
   2. NoSQL DB
      1. 일부 유형의 애플리케이션의 경우 관계형 데이터베이스의 대안이 될 수 있음.
      2. 대용량 데이터를 높은 가용성으로 처리 가능
      3. 다양한 구현 및 데이터 모델로 범주가 광범위
      4. 분산 내결함성의 일반 기능 지원
      5. 사용 중인 앱이 트랜잭션 지원, ACID 준수, 조인 SQL이 필요한지?
      6. 이들 모두 또는 일부, 아니면 데이터 모델의 일부가 없어도 실행 가능한지?
      7. 유연성, 가용성, 확장성 및 성능 향상
   3. **NoSQL로 기능 이동: 사용사례(시험에 잘 나옴)**
      1. **순위표 및 점수 매기기**
      2. **클릭스트림 또는 로그 데이터의 빠른 수집**
      3. **임시 데니터 필요(장바구니 데이터)**
      4. **핫 테이블**
      5. **메타데이터 또는 룩업 테이블**
      6. **세션 데이터**
   4. 사용 사례: 소셜 네트워크(사진 출처: 강의 자료)



* 1. **Amazon DynamoDB 일관성(중요)**
     1. **AWS 리전 세 개의 시설에 데이터를 동기식 복제**
     2. **읽기 일관성: 성공적으로 쓰기, 업데이트된 데이터 항복의 읽기 작업의 방법 및 시간**
     3. **읽기 요청 시, 최종적 또는 강력한 일관된 읽기 중에서 원하는 읽기 지정 가능**
  2. 모범 사례
     1. 항목 크기 작게 유지
     2. 메타데이터는 DynamoDB에 그림, 대용량 BLOB는 Amazon S3에 저장
     3. 시계열 데이터 저장 시, 일별, 주별, 월별 등의 테이블 사용
     4. 조건부 또는 낙관적 동시성 제어(OCC) 업데이트 사용
     5. 핫 키와 핫 파티션 사용 X

1. 파트5: 관계형 데이터를 Amazon RDS에 저장
   1. Amazon Aurora: 복원력이 뛰어난 설계
      1. 99.99% 가용성
      2. 즉각적 장애 복구
      3. DB 재시작해도 캐시 계층 보존되어 읽기 응답 개선
      4. 즉각적인 프로모션을 위해 설계된 읽기 전용 복제본
2. 웹 콘텐츠는 어떻게 저장하는지
   1. 정책 자산 🡪 Amazon S3
   2. 자주 액세스하는 자산 🡪 Amazon CloudFront
   3. 비관계형 데이터 🡪 Amazon DynamoDB와 같은 NoSQL
   4. 관계형 데이터 🡪 Amazon RDS