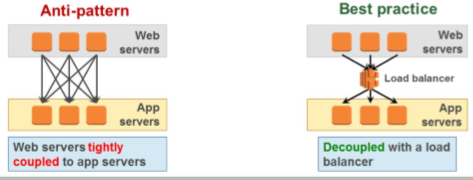
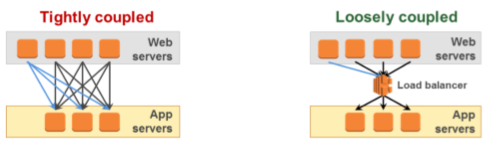
1. 파트1: 구성요소를 느슨하게 결합(사진 출처: 강의 자료)
   1. 모범 사례: 느슨하게 결합

구성 요소 하나의 변경이나 장애나 다른 구성 요소에 영향을 주지 않



* 1. 결합 해제

시스템이 더 느슨하게 결합될수록 더 쉽게 확장할 수 있음.



시스템이 더 느슨하게 결합될수록 더 쉽게 확장되며 내결합성도 강화 가능

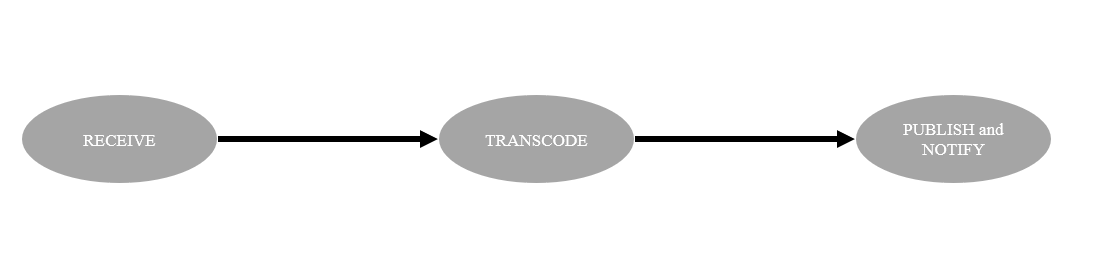


1. 파트2: 느슨한 결합 전략
   1. 모범 사례: 서버가 아니라 서비스를 설계 🡪 EC2 가 아닌 관리형 서비스 사용
      1. 관리형 서비스와 서버리스 아키텍처는 환경에 더 뛰어난 **안정성**과 **효율성** 제공
      2. 모범 사례
         * 서버리스 솔루션이 필요할 때 프로비저닝 됨.
         * 메시지 대기열이 애플리케이션 간의 통신 처리
         * 정적 웹 자산은 Amazon S3와 같은 외부에 저장
         * 사용자 인증 및 사용자 상태 스토리지는 관리형 AWS 서비스에서 처리
   2. 서비스 지향 아키텍처(SOA) 구현
      1. 애플리케이션 요소가 통신 프로토콜을 통해 **다른 구성요소에 서비스를 제공**하는 아키텍처 접근 방식
      2. 서비스는 구현 세부정보가 더 이상 관련되지 않는 **기능이 자체 포함**되어 있음.
   3. 마이크로 서비스
      1. SOA 내 **더 작고 정밀하며 독립적인 프로세스**
      2. 각 프로세스는 **하나의 작은 작업**을 수행하는데 주력함.
      3. **언어에 구애를 받지 않는 API**를 사용하여 서로 통신함
      4. **느슨한 결합**되어 있음!(사진 출처: 강의 자료)

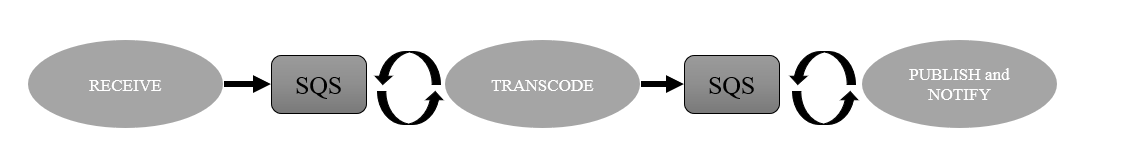


* + 1. 장점
       - 기능을 개별 수성 요소로 분할
       - 반복할 파트가 더 작아 짐.
       - 테스트할 영역 축소
       - 변경에 따른 위험 감소의 이점
       - 수평적 확장 가능한 개별 파트 사용
  1. 모범사례
     1. 수성 요소를 오류 없이 변경
     2. 간단한 **API 사용**
     3. 기술에 구애를 받지 않도록 유지
     4. 장애를 염두에 두고 설계
     5. 환경 모니터링
     6. 서버를 **상태 비저장** 모드로 처리!!
        + 상태 비저장 모드를 사용하면 Auto Scaling을 훨씬 더 쉽게 인스턴스를 추가하고 제거 가능

1. 파트3: 구성 요소 간에 쉽고 안정적으로 통신하려면 무엇을 사용해야 합니까?
   1. Amazon Simple Queue Service(SQS)
      1. 완전관리형 메시지 대기열 서비스
      2. 처리량과 관계없이 모든 볼륨의 메시지를 전송 가능
         * 메시지 🡪 형식과 관계없이 최대 **256KB**까지 작성 가능
         * Amazon SQS 🡪 같은 대기열에서 여러 리더 및 라이터를 지원
         * 대기열 🡪 처리 대기 중인 메시지를 위한 리포지터리(저장소)
   2. Amazon SQS 대기열 유형
      1. 표준 대기열
         * 최소한 1회 전송
         * 최선의 노력으로 순서화 🡪 순서는 보장 X
      2. FIFO 대기열
         * 정확한 1회 처리 🡪 중복 도입 X
         * 제한된 처리량 🡪 초당 최대 300건의 전송, 수신, 삭제
   3. 강결합 시스템 및 느슨한 결합 시스템
      1. 강결합 시스템

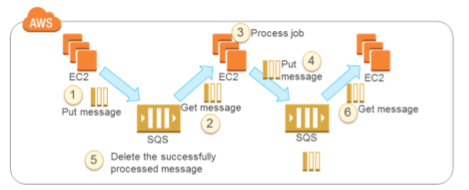


* + 1. 느슨한 결합 시스템

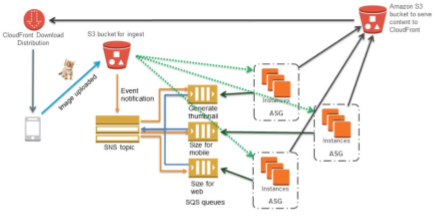


* 1. Amazon SQS를 통한 느슨한 결합(사진 출처: 강의 자료)

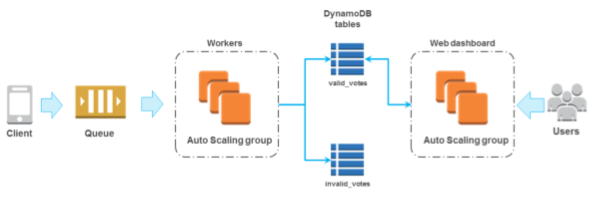
대기열 체인 패턴을 사용하면 비동기식 처리 가능



* 1. Amazon SQS의 이점
     1. 확장성: 잠재적으로 수백만 건의 메시지
     2. 안정성: 모든 메시지가 여러 데이터 센터 및 여러 서버에 중복으로 저장됨
     3. 쉬운 사용: 발신 메시지 수신, 송신 메시지 발신
     4. 동시 읽기 / 쓰기
     5. 보안: API 자격 증명 필요
  2. Amazon SQS의 주요 기능
     1. 가시성 제한 시간
        + 여러 구성 요서가 같은 메시지를 처리하는 것을 방지
        + 메시지 수신하면 처리하는 동안 ‘잠김’ 상태가 되며 이는 다른 구성 요소에서 해당 메시지를 처리하는 것을 방지함.
     2. Dead Letter Queues
        + **처리되지 못한** 메시지 대기열
        + 사용 이유 두 가지
          1. 성공적으로 처리되지 못한 메시지를 열외 분리 가능
          2. FIFO가 활성화된 상태이고 메시지를 처리할 수 없는 경우 메시지 흐름 복원
  3. Amazon SQS 사용 사례
     1. 작업 대기열
     2. 배치 작업 버퍼링
     3. 요청 오프로딩
     4. 팬아웃
     5. Auto Scaling
  4. Amazon Simple Notification Service (SNS)
     1. 알림을 설정 및 운영하고 구독 서비스 기타 어플리케이션에 알림 전송 가능
     2. **Amazon SNS 구독자 유형(\*시험문제\*)**
        + 이메일(일반 또는 JSON)
        + HTTP / HTTPS
        + SMS(문자 서비스) 클라이언트
        + Amazon SQS 대기열
        + 모바일 푸시 메시징
        + AWS Lambda 함수
     3. Amazon SNS의 특성
        + 단일 게시 메시지
        + 순서는 보장되거나 관련되지 않음
        + 회수 안 됨
        + HTTP / HTTPS 재시도
        + 메시지당 최대 **256KB**
  5. 앤드 투 앤드 시나리오: 이미지 처리 – S3 이벤트 알림(사진 출처: 강의 자료)



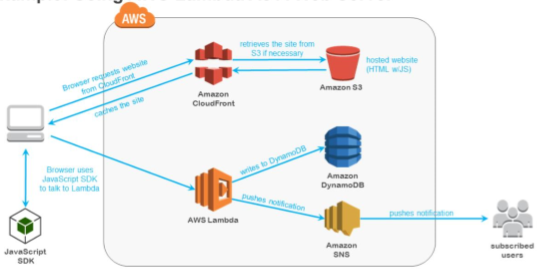
1. 파트4: 느슨한 결합 및 Amazon DynamoDB
   1. 느슨하게 결합된 인프라와 함께 DynamoDB 사용
      1. DynamoDB는 높은 처리량으로 **처리 결과를 저장 및 검색**하는데 적합한 솔루션
      2. DynamoDB
         * 고가용성
         * 내결함성
         * 완전관리형
      3. 느슨하게 결합된 시스템은 DynamoDB와 같은 관리형 NoSQL 데이터베이스 솔루션과 원활하게 연동
   2. Amazon SQS 및 DynamoDB의 예제 패턴(사진 출처: 강의 자료)



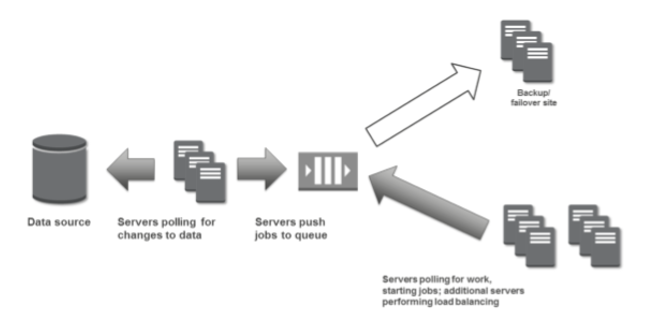
1. 파트5: Amazon API Gateway
   1. : Amazon API Gateway
      1. 데이터 비즈니스 로직, 또는 기능에 액세스할 수 있도록 하는 API 생성 가능
      2. 완전관리형 🡪 확장성이 높음
      3. 워크로드 처리 가능
         * Amazon EC2
         * AWS Lambda
         * 모든 웹 애플리케이션
   2. 기능
      1. 다양한 번과 단계의 API 호스팅 및 사용
      2. 개발자에게 API 키 생성하여 배포
      3. 요청 제한, 모니터링하여 백엔드 보호
      4. AWS Lambda와 긴밀히 통합
      5. AWS Marketplace와 통합
   3. 이점
      1. 관리형 캐시 사용하여 API 응답 저장
      2. Amazon CloudFront를 통해 지연 시간 단축 및 분산 서비스 거부(DDoS) 보호
      3. iOS, Android 및 JavaScript용 SDK 생성
      4. 요청/응답 데이터 변환
2. 파트6: 서버리스 아키텍처: 모든 인스턴스가 꼭 필요합니까?
   1. AWS Lambda를 사용하여 인프라를 결합 해제
      1. 고가용성과 제한된 비용으로 **데이터를 처리**하는데 적합한 솔루션
      2. **기존 서버** 대체
   2. 서버리스 컴퓨팅
      1. 이벤트 발생 후 수 밀리 초 내에 코드 시작
      2. 이럴 때 고려
         * 간단한 함수 또는 처리 애플리케이션을 실행하는데 전체 인스턴스를 사용하고 있는 경우
         * HA, 조정, 배포 또는 관리에 대해 신경 쓰고 싶지 않은 경우
   3. 함수 트리거(사진 출처: 강의 자료)



* 1. 사용 방법
     1. 코드 업로드(.zip형식)
     2. 실행 빈도 지정(예약된 함수) / 이벤트 소스 파악(이벤트 중심 함수)
     3. 필요한 컴퓨팅 리소스 지정(메모리 128MB ~ 3GB)
     4. 제한 시간 기간 지정
     5. 액세스해야 하는 리소스의 VPC 지정(해당하면)
     6. 함수 시작
  2. 리소스 크기 조정
     1. 함수는 **100밀리초에서 5분 사이**
     2. **프리티어**: 100만 건의 무료 요청 및 월별 4백만 GB/s의 컴퓨팅 시간
  3. 예(사진 출처: 강의 자료)
     1. 웹 서버로 사용



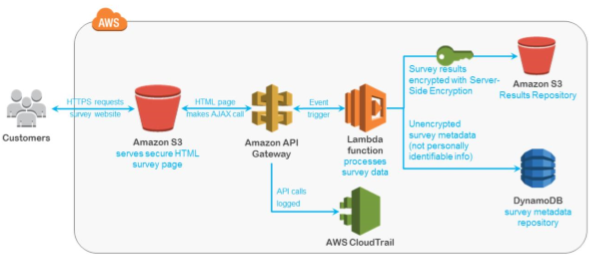
* + 1. 일반적인 데이터 처리 솔루션



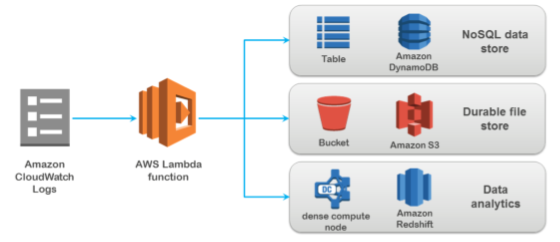
* + 1. Lambda 사용 시



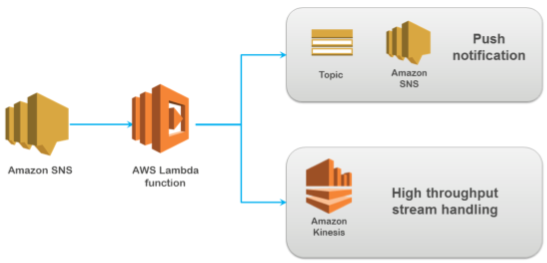
1. 파트7: 결합 해제 예(사진 출처: 강의 자료)
   1. 안전한 설문 조사 제공



* 1. Amazon CloudWatch Logs 처리



* 1. 실시간 메시지 처리 워크플로



* 1. CRUD 백엔드 워크플로

