Cloud & AWS

Cookie

Contents

-> 클라우드 기본 개념 소개

-> 클라우드 서비스

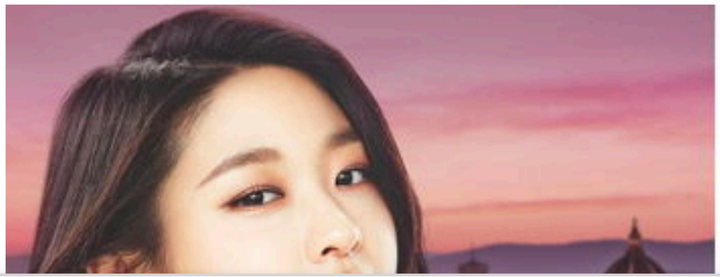
-> AWS

-> AWS 실습

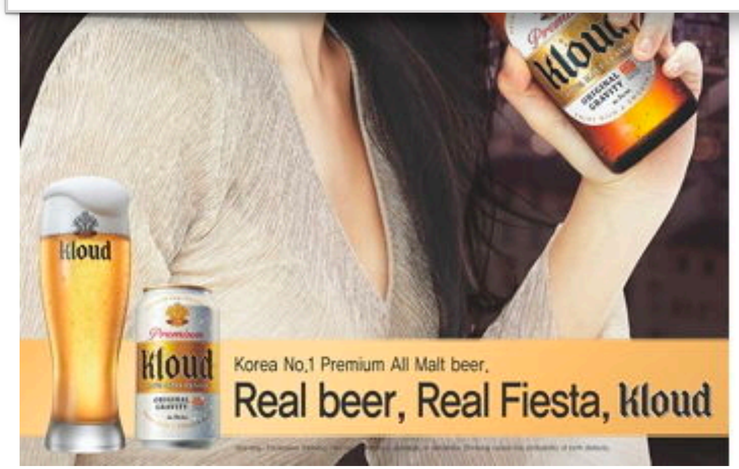
클라우드?

마시는건가..?

클라우드?



설현이 찍은 맥주 광고가 먼저 떠올랐다면
당신은 정상이다!



클라우드?

하지만 구름 io나 Google CloudPlatform
아마존 웹 서비스가 먼저 떠올랐다면
당신은 이미 휠에 근접했다 할 수 있다.

Google Cloud Platform

클라우드?

요새 이런게 엄청 많이 보인다. 잘 나간다는 서비스들은
전부 구름 모양 아이콘에 서비스 이름만 붙여놨다.

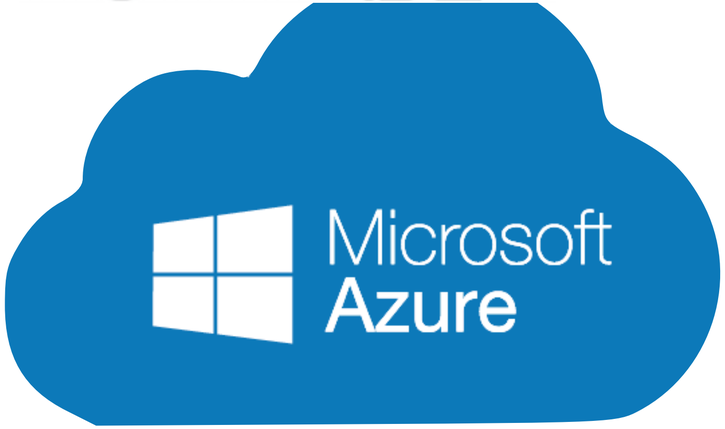
Google Cloud Platform

클라우드?

심지어 멀쩡한 로고가 있어도 구름
모양 아이콘 하나씩은 꼭 만들어 놓는다.

Google Cloud Platform

클라우드?



Google Cloud

클라우드?

-> 정말 많이 들어봤고.. 써본적도 몇 번 있는 것 같은데 **도대체 클라우드가 뭔데**?

-> 인터넷에 찾아봐도 이름만큼이나 **뜬구름 잡는 소리**밖에 안나온다.

클라우드

-> 클라우드 = 구름, 어디서든 보인다.

-> 클라우드 서비스 = 어디서든 구름 쳐다볼 수 있듯이 어디서든 똑같이 볼 수 있고 쓸 수 있다.

ex) 영화를 DVD로 본다?

* 위 클라우드에 대한 설명은 김성회의 G식 백과 채널의 영상 내용을 정리한 것입니다.

영화를 DVD로 본다? X

왜? 다른 장소에서 보려면 DVD 플레이어랑 DVD를 들고다녀야 하거든!

* 위 클라우드에 대한 설명은 김성회의 G식 백과 채널의 영상 내용을 정리한 것입니다.

구름은 들고 다니는게 아니야!

* 위 클라우드에 대한 설명은 김성회의 G식 백과 채널의 영상 내용을 정리한 것입니다.

넷플릭스 유튜브?

클라우드에 가깝다. 어디서든 볼 수 있으니깐!

* 위 클라우드에 대한 설명은 김성회의 G식 백과 채널의 영상 내용을 정리한 것입니다.

웹 하드에 있는 영화는?

비슷한데 미묘하게 약간 다르다. 어디서든 접속되는 건 맞는데 귀찮게 다운받아야하고, 내 컴퓨터에서 재생해야하기 때문!

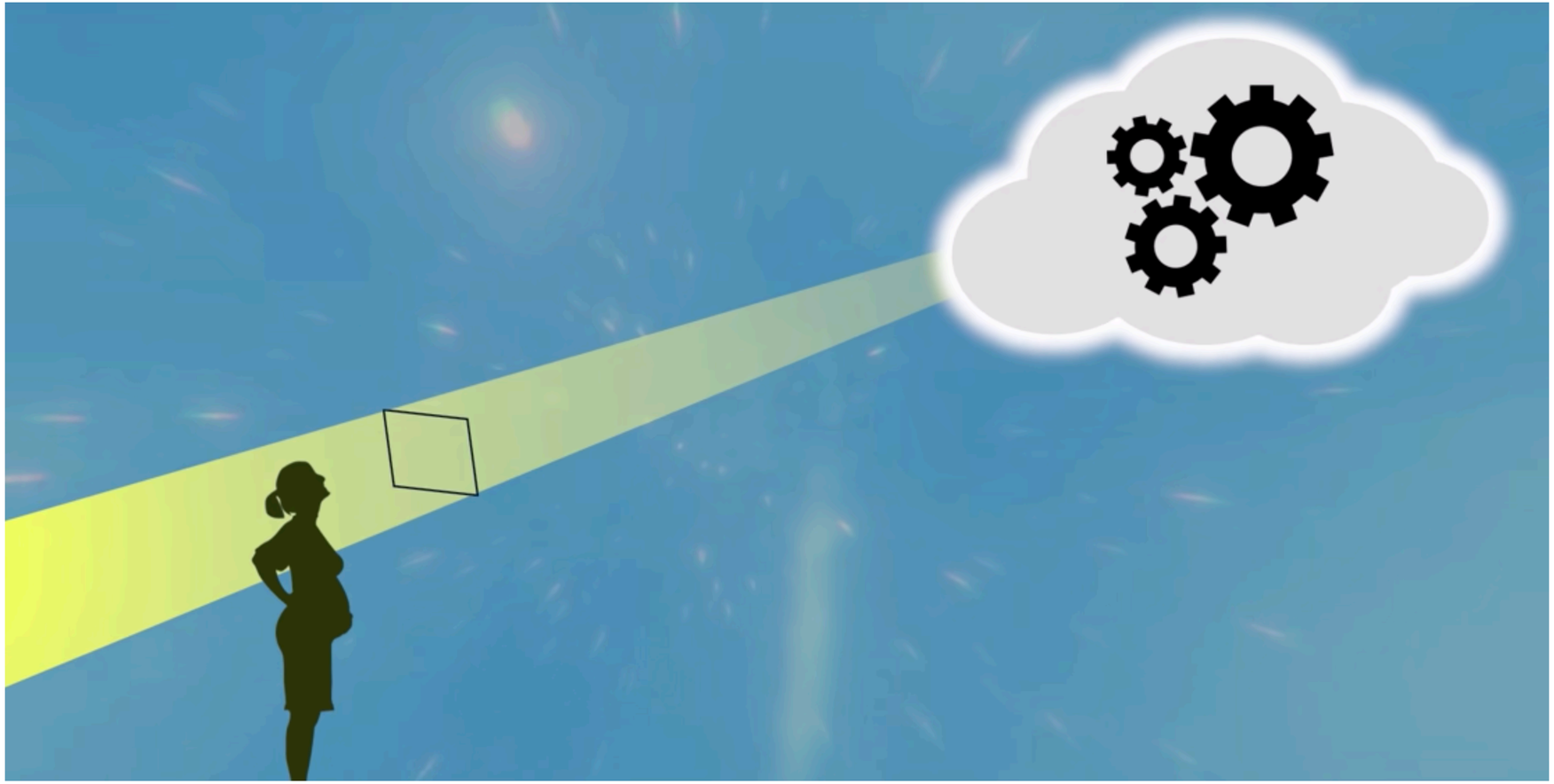
* 위 클라우드에 대한 설명은 김성회의 G식 백과 채널의 영상 내용을 정리한 것입니다.

구름에 손을 넣어서 뭔가를 꺼낼 수 있나요?
구름은 멀리서 보는거죠!

* 위 클라우드에 대한 설명은 김성회의 G식 백과 채널의 영상 내용을 정리한 것입니다.

구글 드라이브는?

얘는 클라우드라고 할 수 있다! 어디서든 접속되고 다운받지 않아도 바로 실행되고 다 알아서 되니깐!



클라우드는 원래 뜬구름이야!

구름 속에서 모든 처리를 마친다.
유저에게는 완성된 그림만 쏴준다!

* 위 클라우드에 대한 설명은 김성회의 G식 백과 채널의 영상 내용을 정리한 것입니다.

클라우드 - 요점

1. 어디서든 장소의 제약 없이
2. 파일을 다운로드 할 필요 없이
3. 내 디바이스는 보여주기만

* 위 클라우드에 대한 설명은 김성회의 G식 백과 채널의 영상 내용을 정리한 것입니다.

클라우드 - 장점

ex) 구글의 STADIA가(클라우드 게임 플랫폼) 상용화 된다면

1. 클라이언트 프로그램이 별도로 필요하지 않다.
   => 게임 설치를 할 필요가 없음. Active X 따위의 잡다한 프로그램을 깔지 않아도 됨.

2. 디바이스의 제약이 없어진다.
   => 인터넷 브라우저가 되는 모든 기기에서 바로 접속이 가능. 핸드폰, 아이패드, 노트북 심지어 옛날 옛적 윈도우 97에서까지!

3. 장소의 제약이 없어진다.
   => 컴퓨터로 하던 게임 똥싸러 가서도 핸드폰으로 계속 할 수 있다.

4. 보안 수준이 높아진다.
   => 외부에 노출된 프로그램이 없으므로 모든 유저들이 회사 내부 보안 수준의 보호를 받을 수 있게 된다.

* 위 클라우드에 대한 설명은 김성회의 G식 백과 채널의 영상 내용을 정리한 것입니다.

클라우드 - 장점

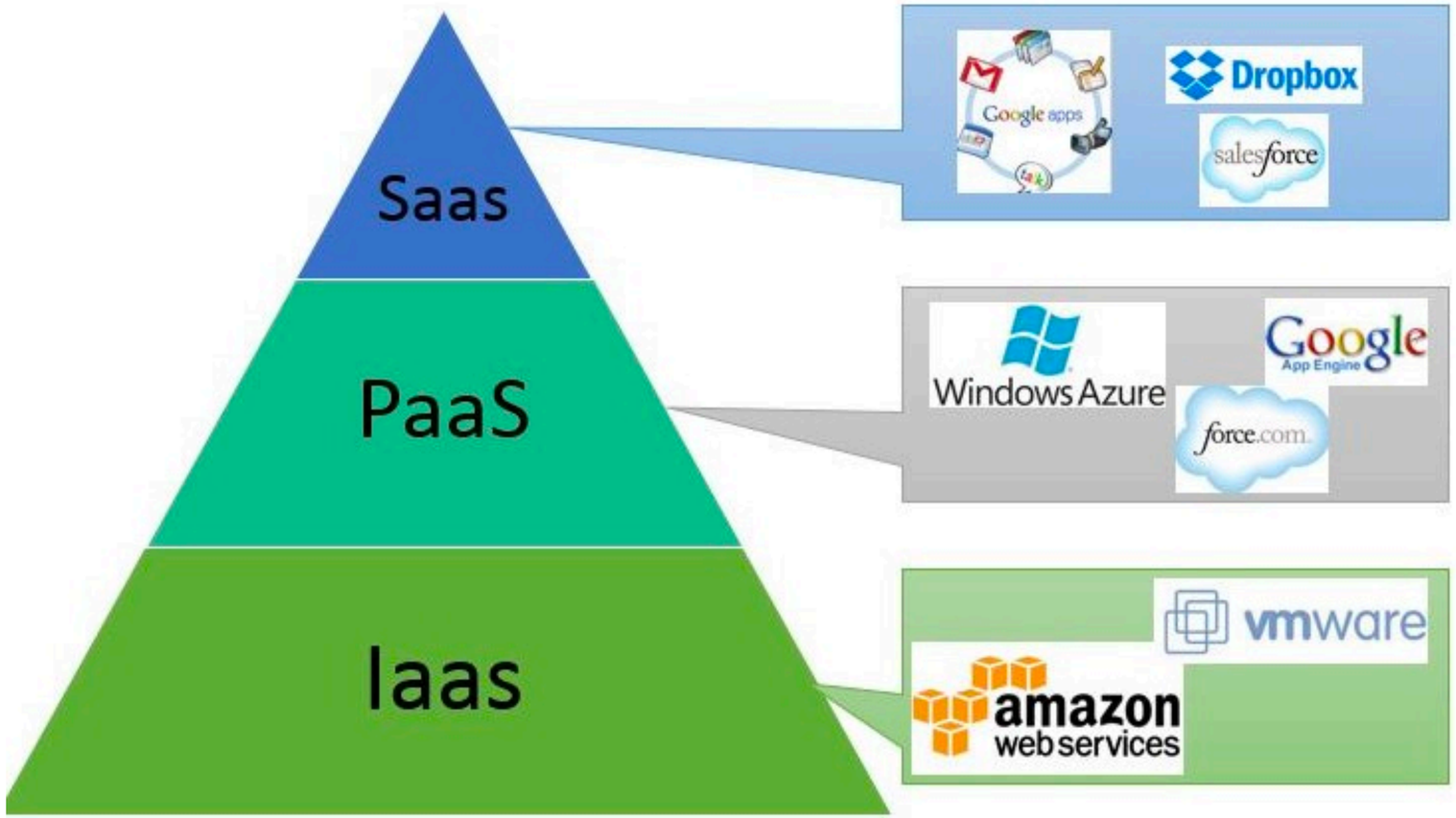
ex) 아마존 웹 서비스를 사용할 때를 생각해보면?

1. 네트워크 장비, 전용망 설치, 비상전력발생기 등등을 설치할 필요가 없다.
   => AWS에 카드 등록하고 EC2 인스턴스 생성하기를 누르면 컴퓨터가 생긴다!

2. 디바이스의 제약이 없어진다.
   => 터미널 접속만 가능한 디바이스라면 어디서든 접속할 수 있다. 내 컴퓨터의 운영체제가 OSX이던 윈도우던 리눅스던 아무 상관이 없다. 심지어는 안드로이드 핸드폰으로도 접속이 가능.

3. 장소의 제약이 없어진다.
   => 컴퓨터로 하던 코딩을 똥 싸러 가서 핸드폰으로 계속 할 수 있다…? 아니! 내가 지금 인터넷이 불안한 산골짜기 시골마을에 있던 KT 기지국 옆에 있던 내 서버는 안정적으로 잘 돌아간다.

4. 보안 수준이 높아진다.
   => 아마존 웹 서비스의 보안이 뚫리지 않는 한 내 프로그램이 해킹당할 위험은 거의 없다. (내 인증 정보를 실수로 흘리지만 않는다면.. ㅎㅎ)



클라우드 서비스

좀 자세하게 들어가볼까요

Software as a Service (SaaS)

-> **응용 소프트웨어를 서비스로 제공**하는 모델

-> 사업자가 **Dropbox, Gmail** 등의 애플리케이션을 제공하고 사용자는 인터넷에서 이를 바로 사용

-> 프로그램과 데이터가 **모두 서버에 저장**되어 언제 어디서든 손쉽게 서비스 이용 가능

Platform as a Service (PaaS)

-> **개발 환경을 서비스**로 제공하는 모델

-> platform 상의 자원을 활용하여 새로운 프로그램 제작 가능 (**구글 앱 엔진, 페이스북 플랫폼** 등)

-> 완전한 **개발, 배포 환경이 제공**됩니다.

-> 서버, 저장소, 네트워크 연결 등의 인프라뿐만 아니라 빌드, 테스트, 배포, 업데이트 등 **프로그램 수명 주기**까지도 쉽게 관리할 수 있습니다.

-> **소프트웨어 라이선스, 개발 도구 등 기타 리소스**를 관리해야하는 복잡성이 없어집니다. 사용자는 개발하는 응용 프로그램과 서비스 자체만을 관리하고 그 밖의 모든 항목은 서비스 제공자가 관리합니다.

Infrastructure as a Service (IaaS)

-> **컴퓨터 H/W 시스템을 서비스**로 제공하는 모델

-> **컴퓨팅 자원 및 각종 하드웨어 자원**을 사용할 수 있도록 제공. (**Amazon EC2**)

-> 사용자는 물리 서버의 구매 및 유지 비용에 대한 부담 없이 필요한 순간 바로바로 사용할 수 있음.

-> 네트워크 트래픽 변동, 하드웨어 장애 등 소규모 개발팀으로 해결하기 어려운 문제들은 대신하여 처리해 줌.

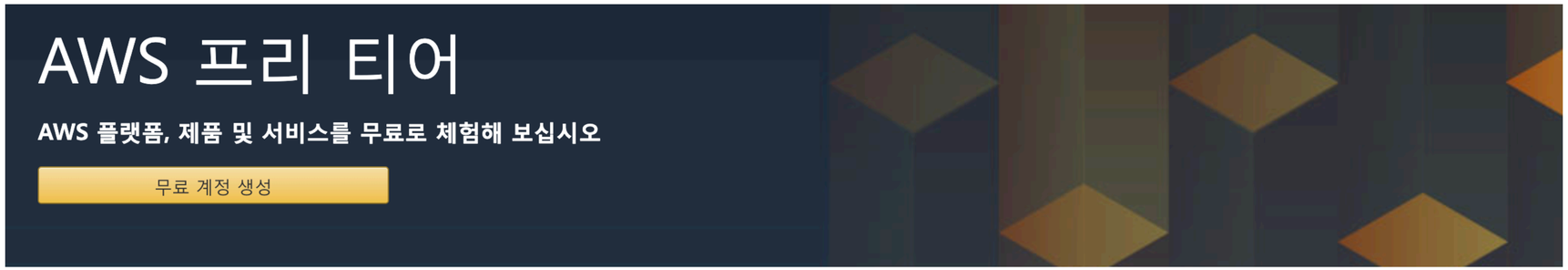
Infrastructure as a Service (IaaS)

-> Cloud Computing : 사용자가 서비스 제공자를 통해 빠르고 손쉽게 IT 자원들을 사용하고 사용한 만큼 요금을 지불하는 컴퓨팅 방식

-> 인스턴스의 선택폭이 다양하며 필요(ex 트래픽의 증가)에 따라 유연하게 확장하고 수축할 수 있다.

-> 비가 오나 눈이 오나 하드웨어 관리와 시스템 안정성에 대해 걱정하지 않아도 된다. 지진, 해일이 발생하고 온 도시가 마비 돼도 내 서버는 끄떡없음. (저도 압니다, 페이스북 데이터 센터에서는 비가 와서 데이터가 날라갈 뻔한 적이 있다죠..? 그래도 어쩌겠습니까.. 믿고 맡겨야지)

AWS



오퍼 유형

프리 티어를 이용해 60가지가 넘는 제품을 체험하고 AWS에 구축할 수 있습니다. 사용하는 제품에 따라 세 가지 유형의 프리 티어 오퍼가 제공됩니다. 각 제품에 대한 자세한 내용은 아래를 참조하십시오.

언제나 무료

이 프리 티어 오퍼는 만료되지 않으며 모든 AWS 고객이 사용 가능

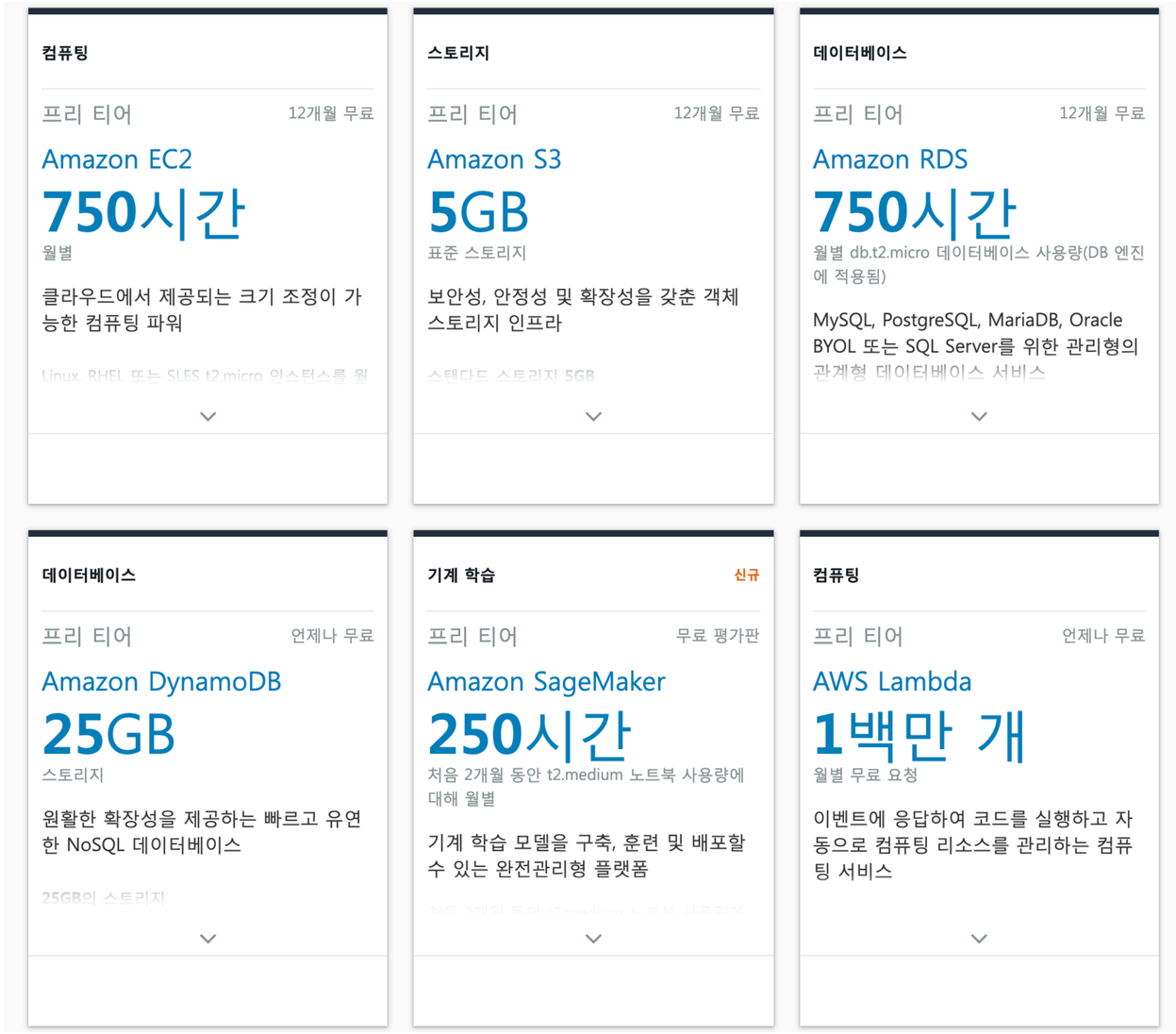
12개월 무료

처음 AWS에 가입한 날부터 12개월 동안 사용 가능

평가판

다양한 소프트웨어 솔루션에서 단기 무료 평가판 사용 가능

신규 가입을 하면 1년간 상당히 많은 서비스를 무료로 이용할 수 있다.
좋은 서비스는 많이 많이 이용하자
~~프리티어가 끝났으면 구글 계정을 하나 생성하고 가입하면 됨~~


컴퓨팅
프리 티어
12개월 무료
Amazon EC2
750시간
월별
클라우드에서 제공되는 크기 조정이 가능한 컴퓨팅 파워
스토리지
프리 티어
12개월 무료
Amazon S3
5GB
표준 스토리지
보안성, 안정성 및 확장성을 갖춘 객체 스토리지 인프라
데이터베이스
프리 티어
12개월 무료
Amazon RDS
750시간
월별 db.t2.micro 데이터베이스 사용량(DB 엔진에 적용됨)
MySQL, PostgreSQL, MariaDB, Oracle BYOL 또는 SQL Server를 위한 관리형의 관계형 데이터베이스 서비스
데이터베이스
프리 티어
언제나 무료
Amazon DynamoDB
25GB
스토리지
원활한 확장성을 제공하는 빠르고 유연한 NoSQL 데이터베이스
기계 학습
신규
프리 티어
무료 평가판
Amazon SageMaker
250시간
처음 2개월 동안 t2.medium 노트북 사용량에 대해 월별
기계 학습 모델을 구축, 훈련 및 배포할 수 있는 완전관리형 플랫폼
컴퓨팅
프리 티어
언제나 무료
AWS Lambda
1백만 개
월별 무료 요청
이벤트에 응답하여 코드를 실행하고 자동으로 컴퓨팅 리소스를 관리하는 컴퓨팅 서비스

뭔가 혜택이 짱짱하다..

Amazon Web Service

컴퓨팅

-> Elastic Compute Cloud (EC2)

-> Lambda

스토리지

-> Simple Storage Service (S3)

-> S3 Glacier

데이터베이스

-> DocumentDB (DocumentDB)

-> Relational Database Service (RDS)

-> DynamoDB

-> ElasticCache

네트워킹

-> Route53

-> Cloud Front

-> API Gateway

-> VPC

보안 및 자격 증명

-> IAM

-> AWS Single Sign-On

-> Cognito

-> VPC

-> Web Application Firewall (WAF & SHIELD)

Amazon Web Service

컴퓨팅

-> Elastic Compute Cloud (EC2)

-> Lambda

스토리지

-> Simple Storage Service (S3)

-> S3 Glacier

데이터베이스

-> DocumentDB (DocumentDB)

-> Relational Database Service (RDS)

-> DynamoDB

-> ElasticCache

Elastic Compute Cloud (EC2)

-> 컴퓨팅 파워를 제공

-> 컴퓨터 쇼핑을 하듯이 운영체제, CPU, Memory, Disk 용량, Network Bandwidth 등을 고려하여 컴퓨터를 선택할 수 있다.

-> 성능이 부족한 것 같으면 CPU, Memory 등 추가 장착도 가능

-> 컴퓨터가 배달되기를 기다릴 필요 없이 SSH 원격접속으로 컴퓨터를 활용하면 됨!

Amazon Web Service

컴퓨팅

-> Elastic Compute Cloud (EC2)

-> Lambda

스토리지

-> Simple Storage Service (S3)

-> S3 Glacier

데이터베이스

-> DocumentDB (DocumentDB)

-> Relational Database Service (RDS)

-> DynamoDB

-> ElasticCache

Elastic Compute Cloud (EC2)

-> 가격 정책이 다양함

On-Demand
Spot Instance
Reserved
Dedicated Host

Amazon Web Service

컴퓨팅

-> Elastic Compute Cloud (EC2)

-> Lambda

스토리지

-> Simple Storage Service (S3)

-> S3 Glacier

데이터베이스

-> DocumentDB (DocumentDB)

-> Relational Database Service (RDS)

-> DynamoDB

-> ElasticCache

Elastic Compute Cloud (EC2)

-> 가격 정책이 다양함

On-Demand - 가장 기본적인 방법. 미리 지불해야하는 금액은 따로 없으며 서버가 떠있는 시간만큼 다달이 비용을 지불하면 된다. 필요할 때 키고 다 쓴 뒤에 끄면 되므로 큰 고민 없이 사용하기에 좋다. 사용한 시간은 시간 단위로 계산되는데 Linux 인스턴스에 한해서 초 단위로 계산이 된다.
안정적으로 서버가 떠있어야 하고 컴퓨팅 파워의 소모량이 정확히 예측하기 어려운 경우 사용하기 적합하다.
Spot Instance
Reserved
Dedicated Host

Amazon Web Service

컴퓨팅

-> Elastic Compute Cloud (EC2)

-> Lambda

스토리지

-> Simple Storage Service (S3)

-> S3 Glacier

데이터베이스

-> DocumentDB (DocumentDB)

-> Relational Database Service (RDS)

-> DynamoDB

-> ElasticCache

Elastic Compute Cloud (EC2)

-> 가격 정책이 다양함

On-Demand

Spot Instance - AWS이 그 때 그 때 남는 자원을 싼 값에 판매. 항상 물량이 나와있어 구매를 할 수 있는건 아니지만 구매 가능한 경우 최대 90% 까지 싼 가격에 구입할 수 있다. 가격은 수요와 공급에 의해서 결정된다고 함.
시간 단위로 판매되며 Linux에 한해서 초단위로 구매가 가능
항상 떠있을 필요는 없지만 큰 규모의 컴퓨팅 파워가 필요한 경우에 적합

Reserved

Dedicated Host

Amazon Web Service

컴퓨팅

-> Elastic Compute Cloud (EC2)

-> Lambda

스토리지

-> Simple Storage Service (S3)

-> S3 Glacier

데이터베이스

-> DocumentDB (DocumentDB)

-> Relational Database Service (RDS)

-> DynamoDB

-> ElasticCache

Elastic Compute Cloud (EC2)

-> 가격 정책이 다양함

On-Demand

Spot Instance

Reserved - 장기간 대여료를 약간 할인된 가격에 선불로 구매할 수 있음. 컴퓨팅 파워의 소모량이 큰 폭으로 변화하지 않고 예측 가능할 때 적절. 스팍스의 AWS 인스턴스들이 이걸로 구매가 되어있음!

Dedicated Host

Amazon Web Service

컴퓨팅

-> Elastic Compute Cloud (EC2)

-> Lambda

스토리지

-> Simple Storage Service (S3)

-> S3 Glacier

데이터베이스

-> DocumentDB (DocumentDB)

-> Relational Database Service (RDS)

-> DynamoDB

-> ElasticCache

Elastic Compute Cloud (EC2)

-> 가격 정책이 다양함

On-Demand
Spot Instance
Reserved
Dedicated Host - EC2의 물리 서버를 구매하여 사용. On-Demand와 같은 가격이 붙음. 물리 서버를 굳이 사용하는 이유는 기존 서버에서만 사용할 수 있는 소프트웨어 라이센스 문제 때문이라는데.. 특이 케이스가 아니면 굳이 알 필요는 없을 듯.

Amazon Web Service

컴퓨팅

-> Elastic Compute Cloud (EC2)

-> Lambda

스토리지

-> Simple Storage Service (S3)

-> S3 Glacier

데이터베이스

-> DocumentDB (DocumentDB)

-> Relational Database Service (RDS)

-> DynamoDB

-> ElasticCache

Elastic Compute Cloud (EC2)

-> 가격 정책이 다양함

*초단위 가격 정책

원래는 EC2 인스턴스 가격이 시간 단위로만 매겨졌는데 사람들이 1시간보다도 더 짧은 시간동안 사용하는 경우가 많자 2017년 부터 초 단위로 가격이 매겨주기 시작함. 인스턴스를 사용하기 위해서 짧은 시간동안 EC2 인스턴스를 열었다가 닫아도 사용한 시간을 초 단위까지 계산해줌. 미니멈은 1분.

*부자 랭킹 1등도 해본 베조스의 아마존

18년 1분기 아마존 매출의 11%, 이익의 70%를 AWS가 기여했다고 함.

https://www.mk.co.kr/news/it/view/2018/04/268456/

Amazon Web Service

컴퓨팅

-> Elastic Compute Cloud (EC2)

-> Lambda

스토리지

-> Simple Storage Service (S3)

-> S3 Glacier

데이터베이스

-> DocumentDB (DocumentDB)

-> Relational Database Service (RDS)

-> DynamoDB

-> ElasticCache

Elastic Compute Cloud (EC2)

-> 인스턴스 유형

- 범용 인스턴스
- 컴퓨팅 최적화 인스턴스
- 메모리 최적화 인스턴스
- 스토리지 최적화 인스턴스

-> 각 유형의 목적은 이름에서 바로 알 수 있다.

-> 엄청난 고성능이 필요한 경우가 아니면 범용 인스턴스로 충분하다. 최적화 인스턴스들은 가장 낮은 사양부터 가격이 어마무시하다.

Amazon Web Service

컴퓨팅

-> Elastic Compute Cloud (EC2)

-> Lambda

스토리지

-> Simple Storage Service (S3)

-> S3 Glacier

데이터베이스

-> DocumentDB (DocumentDB)

-> Relational Database Service (RDS)

-> DynamoDB

-> ElasticCache

Elastic Compute Cloud (EC2)

-> 범용 인스턴스 타입

- A 타입
- T 타입
- M 타입

뒤에 숫자는 버젼 정보라고 보면 됨.

Amazon Web Service

컴퓨팅

-> Elastic Compute Cloud (EC2)

-> Lambda

스토리지

-> Simple Storage Service (S3)

-> S3 Glacier

데이터베이스

-> DocumentDB (DocumentDB)

-> Relational Database Service (RDS)

-> DynamoDB

-> ElasticCache

Elastic Compute Cloud (EC2)

-> 범용 인스턴스 타입

- **A 타입** - ARM 아키텍쳐에 최적화된 인스턴스. 비용 효율이 높다고 한다. 하지만 아쉽게도 서울리젼에서는 구매할 수 없다.
- M 타입
- T 타입

Amazon Web Service

컴퓨팅

-> Elastic Compute Cloud (EC2)

-> Lambda

스토리지

-> Simple Storage Service (S3)

-> S3 Glacier

데이터베이스

-> DocumentDB (DocumentDB)

-> Relational Database Service (RDS)

-> DynamoDB

-> ElasticCache

Elastic Compute Cloud (EC2)

-> 범용 인스턴스 타입

- A 타입
- **M 타입** - 컴퓨팅, 메모리 및 네트워크 리소스를 균형있게 제공한다. **고정된 성능**을 제공받을 수 있음.
- T 타입

Amazon Web Service

컴퓨팅

-> Elastic Compute Cloud (EC2)

-> Lambda

스토리지

-> Simple Storage Service (S3)

-> S3 Glacier

데이터베이스

-> DocumentDB (DocumentDB)

-> Relational Database Service (RDS)

-> DynamoDB

-> ElasticCache

Elastic Compute Cloud (EC2)

-> 범용 인스턴스 타입

- A 타입
- M 타입
- **T 타입** - 기본 수준의 CPU를 제공하지만 순간 확장 버스트 기능을 제공한다. 순간 순간 트레픽이 치솟는 서비스에 적합하다고 볼 수 있다.

  비용이 가장 저렴한 범용 인스턴스 유형으로 프리티어가 여기에 해당*

성능 순간 확장 인스턴스 (T타입)

-> 다른 타입의 인스턴스는 고정된 성능을 제공하는 반면, 성능 순간 확장 가능 인스턴스(T3, T3a, T2)는 평소에는 기본 수준의 CPU 성능을 발휘하다가 **기준 성능 이상**으로 요청이 많아지면 성능을 버스트하는 기능을 제공함. **기준 성능**과 **버스트 기능**은 **CPU 크레딧**에 의해 좌우된다.

-> CPU 크레딧 ?
1 크레딧은 CPU 하나를 1분동안 100% 성능으로 사용할 수 있음.
CPU 2개를 1분동안 50%의 성능으로 사용하거나
CPU 2개를 2분동안 25%의 성능으로 사용하는 것도 동일

-> 크레딧 획득
인스턴스는 밀리세컨드 단위로 일정량의 크레딧을 획득. CPU의 사용량이 크레딧 획득속도보다 높으면 크레딧이 소모되고 CPU의 사용량이 낮으면 크레딧이 누적됨

성능 순간 확장 인스턴스 (T타입)

인스턴스 유형	시간당 지급되는 CPU 크레딧	누적 가능한 최대 지급된 크레딧*	vCPUs	vCPU당 기준 성능
t2.nano	3	72	1	5%
t2.micro	6	144	1	10%
t2.small	12	288	1	20%
t2.medium	24	576	2	20%**
t2.large	36	864	2	30%**

*여기서 주목해야할 점!
t2.nano와 t2.micro, t2.micro 인스턴스의 CPU 갯수가 모두 1개로 동일하다.
참고로 순서대로 가격이 더 비쌈!

성능 순간 확장 인스턴스 (T타입)

-> 기준 성능 ?
t2.micro 인스턴스에 1개의 CPU가 배정되어있다고는 하나 CPU를 100% 활용할 수 있는 건 아님.
CPU 한개를 100% 사용한다고 할 때 한 시간동안 필요한 CPU 크레딧의 갯수는 ? 60개
t2.micro에 주어지는 시간당 크레딧은 단 3개
따라서 t2.micro 인스턴스의 기준 성능은 3개 / 60개 = 5% 이다.

-> 뭐야, 5% 밖에 못 쓴다고? 그럼 나머지 95%는 어떡해?
CPU의 사용량이 5%를 넘어가면 자동으로 크레딧이 소모되기 시작한다.

-> T3 및 T3a 인스턴스는 `unlimited`로 시작하도록 기본 설정되어 있습니다. T2 인스턴스는 `standard`로

성능 순간 확장 인스턴스 (T타입)

TMI…

-> 크레딧이 모두 소모되면 CPU 성능이 기준 성능으로 제한됩니다. Unlimited 모드에서는 CPU의 성능이 제한되지 않지만 제한을 넘어서 사용한 경우 추가 비용이 청구됩니다.

-> T2 인스턴스는 `standard`로, T3 및 T3a 인스턴스는 `unlimited`로 시작하도록 기본 설정 되어 있습니다.

-> 더 자세한 정보를 알고 싶다면 AWS 다큐먼트 (https://docs.aws.amazon.com/ko_kr/AWSEC2/latest/UserGuide/burstable-performance-instances.html)를 참고하세요!

Amazon Web Service

컴퓨팅

-> Elastic Compute Cloud (EC2)

-> Lambda

스토리지

-> Simple Storage Service (S3)

-> S3 Glacier

데이터베이스

-> DocumentDB (DocumentDB)

-> Relational Database Service (RDS)

-> DynamoDB

-> ElasticCache

Elastic Compute Cloud (EC2)

-> 범용 인스턴스 타입

그럼 버젼은 뭘로 선택해야할까?

T3 > T3a > T2
M5 > M5a > M4

새로운 인스턴스 타입이 개발되면서 높은 숫자를 부여함.

최신 인스턴스가 비용 효율, 성능 면에서 훨씬 더 우수함! AWS에서도 꼭 필요한 경우가 아니면 최신 세데 인스턴스를 사용하도록 권장하고 있음. 그러니 프리티어가 아니라면 최신 세대 인스턴스를 사용하도록 하자~
*T3의 가격 대시 성능이 T2에 비해 최대 30% 가량 높다고 한다.

Amazon Web Service

컴퓨팅

-> Elastic Compute Cloud (EC2)

-> Lambda

스토리지

-> Simple Storage Service (S3)

-> S3 Glacier

데이터베이스

-> DocumentDB (DocumentDB)

-> Relational Database Service (RDS)

-> DynamoDB

-> ElasticCache

Elastic Compute Cloud (EC2)

-> 범용 인스턴스 타입

t2.micro	1	변수	1GiB	EBS 전용	시간당 0.0144 USD
t2.small	1	변수	2GiB	EBS 전용	시간당 0.0288 USD
t2.medium	2	변수	4GiB	EBS 전용	시간당 0.0576 USD
t3.small	2	변수	2GiB	EBS 전용	시간당 0.026 USD
t3.medium	2	변수	4GiB	EBS 전용	시간당 0.052 USD

우리가 자주 사용하게 될 T 타입 인스턴스의 가격은?
t2.micro가 공짜가 아니면 ? 30일 기준 30.3456 USD (대략 3만 5천원)
t2.medium은 ? T2.micro의 4배, 대략 14만원 정도
현재 아라와 OTL이 돌고있는 서버 t2.medium!
t3.medium으로 절약할 수 있는 돈은 ? 한달 4 USD.. ~~알아보길 잘했나?~~

Q & A

Amazon Web Service

컴퓨팅

-> Elastic Compute Cloud (EC2)

-> Lambda

스토리지

-> Simple Storage Service (S3)

-> S3 Glacier

데이터베이스

-> DocumentDB (DocumentDB)

-> Relational Database Service (RDS)

-> DynamoDB

-> ElasticCache

Lambda

-> 작성한 프로그램만 업로드. 별도의 세팅이나 컴퓨팅 자원 관리를 할 필요가 없다. (서버를 띄울 필요가 없다.)

-> 코드를 업로드하기만 하면, Lambda에서 높은 가용성으로 코드를 실행 및 확장하는 데 필요한 모든 것을 처리.

-> 프로그램이 실행된 만큼만 비용 청구

ex) Express의 controller 영역만 람다에 업로드. 서버를 실행시켜놓을 필요가 없음. API 요청이 들어오면 Lambda에서 코드를 실행해서 결과값 반환. 서버가 켜져있지 않기 때문에 유휴 상태에 대한 비용을 지불할 필요 없이 코드가 실행된 시간에 한해서만 비용을 지불

Amazon Web Service

컴퓨팅

-> Elastic Compute Cloud (EC2)

-> Lambda

스토리지

-> Simple Storage Service (S3)

-> S3 Glacier

데이터베이스

-> DocumentDB (DocumentDB)

-> Relational Database Service (RDS)

-> DynamoDB

-> ElasticCache

Lambda

-> 컴퓨팅에 사용할 메모리를 선택 -> 메모리의 사이즈에 맞게 CPU와 기타 리소스가 할당됨

-> 높은 메모리를 할당 할수록 비쌈, CPU 용량이 높아짐

-> 요청 건수, 컴퓨팅 사용 시간으로 비용 계산

가격, 프리티어 (메모리 128MB, 미국 동부 버지니아 기준)

-> 요청당 0.0000002 USD (백만건 당 0.2 USD)

-> 초당 0.00000208 USD (하루기준 1.8 USD)

-> 프리티어 : 요청 1백만 건, 320만 초 (대략 37일)

-> 다른 서비스 프리티어와는 다르게 1년 제한없이 무제한임

Amazon Web Service

컴퓨팅

-> Elastic Compute Cloud (EC2)

-> Lambda

스토리지

-> Simple Storage Service (S3)

-> S3 Glacier

데이터베이스

-> DocumentDB (DocumentDB)

-> Relational Database Service (RDS)

-> DynamoDB

-> ElasticCache

Simple Storage Service (S3)

-> 서버 파일 저장소 끝판왕

-> 용량 무제한

-> te레픽 제한 없음

-> 다양한 접근 제어 가능

-> 초기 Dropbox는 사실 S3에 파일을 쉽게 올려 주는 GUI 였음

Amazon Web Service

컴퓨팅

-> Elastic Compute Cloud (EC2)

-> Lambda

스토리지

-> Simple Storage Service (S3)

-> S3 Glacier

데이터베이스

-> DocumentDB (DocumentDB)

-> Relational Database Service (RDS)

-> DynamoDB

-> ElasticCache

S3 Glacier

-> 장기간 보관을 위한 저비용 저장소

-> 파일 접근에 수 분에서 수 시간까지 소요될 수 있는 대신 비용이 아주 저렴함

-> 더 저렴한 **S3 Glacier Deep Archieve**도 있음. 일년에 1~2번 정도 접속하는데 적합하다고 함.

Amazon Web Service

컴퓨팅

-> Elastic Compute Cloud (EC2)

-> Lambda

스토리지

-> Simple Storage Service (S3)

-> S3 Glacier

데이터베이스

-> DocumentDB (DocumentDB)

-> Relational Database Service (RDS)

-> DynamoDB

-> ElasticCache

데이터 베이스

-> DB 서버 구동을 위해 위해 최적화된 클라우드 컴퓨팅

-> MySQL과 같은 Relational Database를 위한 RDS

-> Mongodb와 같은 Document DB를 위한 서비스

-> AWS의 전매 특허 NoSQL DB인 Dynamo DB

-> In memory data store로 1밀리초 미만의 응답시간을 보장하는 ElasticCache. Redis와 Memcached 사용 가능.

-> 못 들어보신 분들도 있으시겠지만 DynamoDB는 정말 엄청납니다!

데이터 베이스

-> DynamoDB는 성능이나 인프라가 정말 좋습니다.

-> 근데 배우기가 너무 너무 어렵고

-> 진짜 너무 어렵습니다.

Amazon Web Service

네트워킹

-> Route53 - DNS 관리

-> Cloud Front - DDOS 방어, SSL 인증, CDN 등. CloudFlare와 동일하다고 봐도 됨.

-> API Gateway - AWS의 기능들에 Web API 형태로 접근할 수 있도록 해줌. Lambda와 함께 사용해서 백엔드 서버를 구축할 수도 있음.

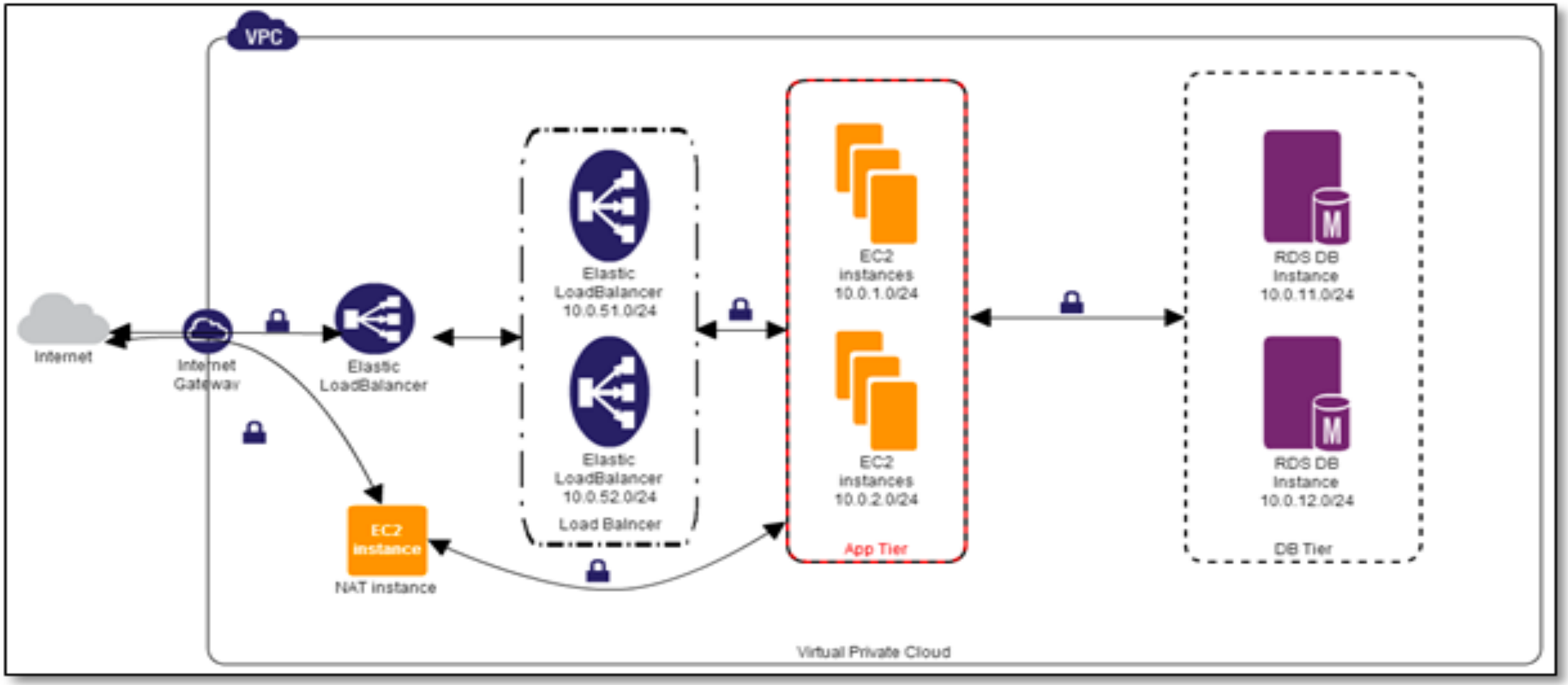
-> VPC - AWS내에서 프라이빗 네트워크 구축을 가능하게 해줌. ex) Bastion Host

보안 및 자격 증명

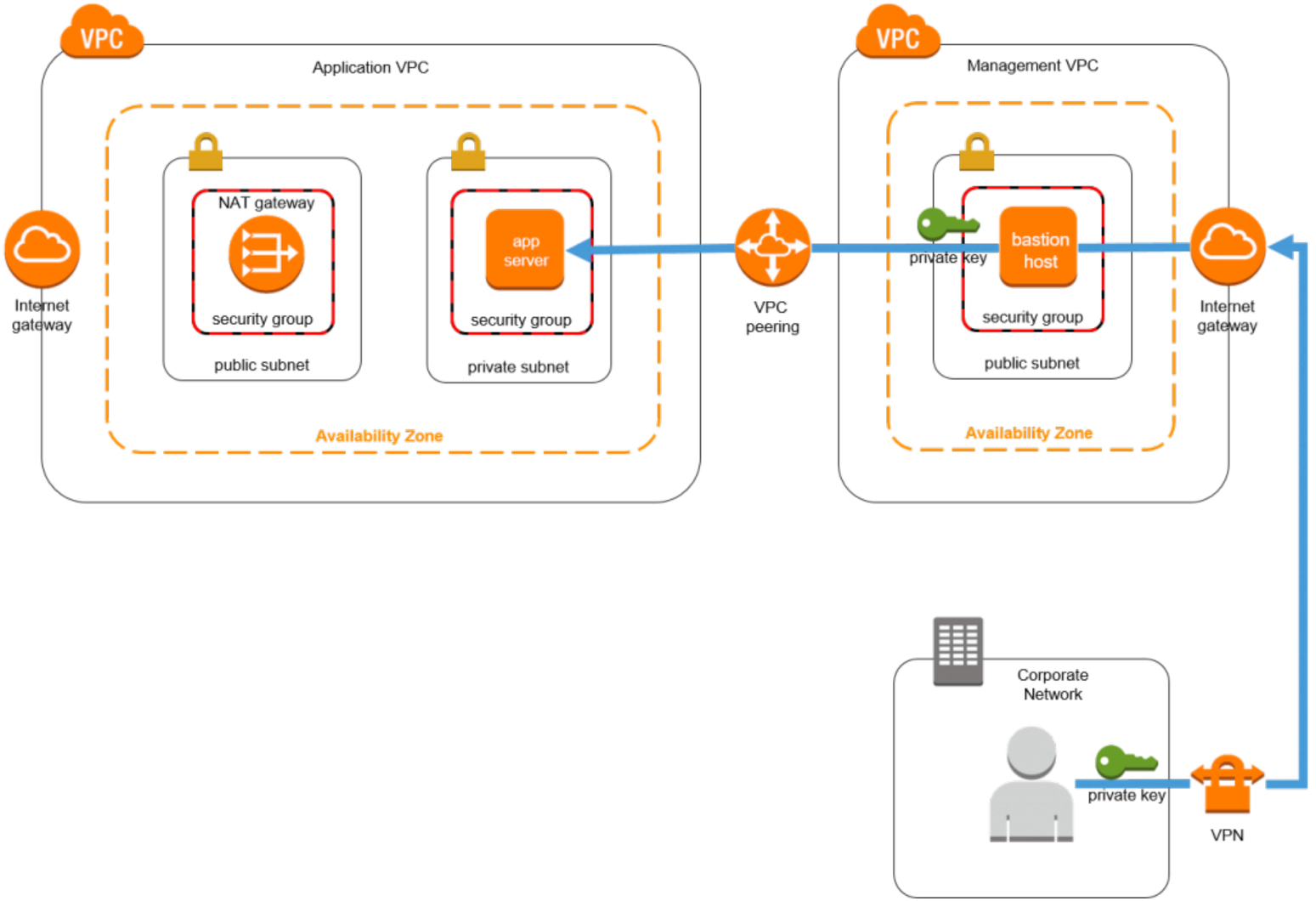
-> IAM - AWS 서비스에 대한 유저레밸 또는 그룹 레밸에서의 접근 권한을 관리해줌. ex) Loopy에게 준 AWS 시크릿키는 S3에만 접근할 수 있는 IAM 유저의 시크릿키였기 때문에 github 퍼블릭 레포지토리에 등록되어도 큰 문제 없이 넘어갈 수 있었음.

-> Web Application Firewall (WAF & SHIELD) - AWS 서비스에 방화벽을 설치. Cloud Front와 Load Balancer에 설치할 수 있음.

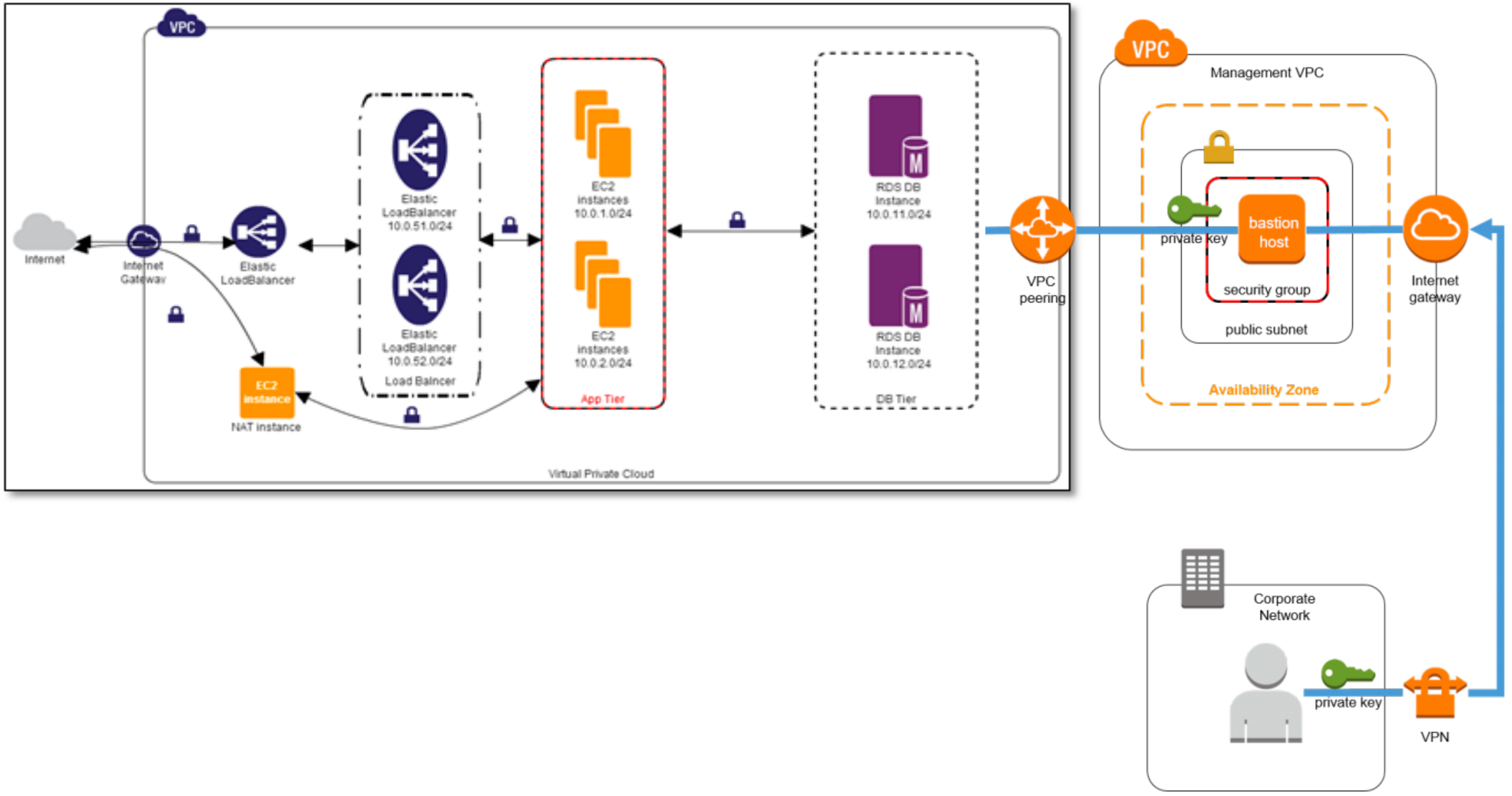
VPC - Load Balancer



VPC - Bastion Host



VPC - All In One



Amazon Web Service

AWS 관리

-> Cloud Watch - 로그 남기기

-> Cloud Trail - AWS 접속 / 사용 기록

-> Cost Explorer - 비용 청구 정보 시각화

-> Cost Management - 결제 대금, 결제 정보, 명세서 등 관리

-> AWS Budgets - 예산 관리

AWS

실습

AWS 실습

-> console.aws.amazon.com

-> (이메일 입력창이 뜨지 않는다면) 루트 계정으로 로그인

-> 로그인

-> 서비스 -> IAM

Identity and Access Management 소개

IAM 사용자 로그인 링크:

https://635949589444.signin.aws.amazon.com/console

사용자 지정

IAM 리소스

사용자: 2

역할: 2

그룹: 1

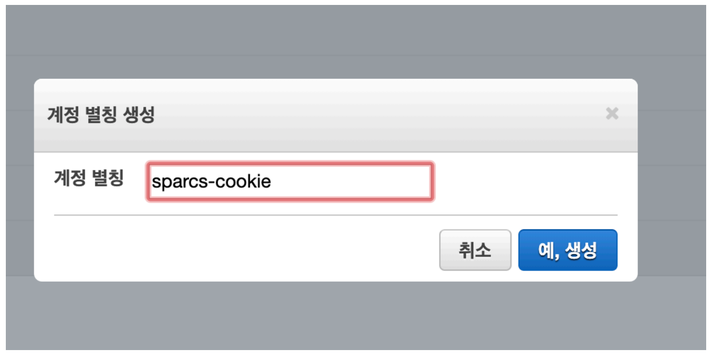
자격 증명 공급자: 0

고객 관리형 정책: 0

보안 상태

3/5 완료

- 루트 액세스 키 삭제
- 루트 계정에서 MFA 활성화
- 개별 IAM 사용자 생성
- 그룹을 사용하여 권한 할당
- IAM 비밀번호 정책 적용


계정 별칭 생성
계정 별칭
sparcs-cookie
취소
예, 생성

Identity and Access Management 소개

IAM 사용자 로그인 링크:

https://sparcs-cookie.signin.aws.amazon.com/console | 사용자 지정

IAM 리소스

사용자: 2

역할: 2

그룹: 1

자격 증명 공급자: 0

고객 관리형 정책: 0

보안 상태

3/5 완료

- ✓ 루트 액세스 키 삭제
- ⚠ 루트 계정에서 MFA 활성화
- ✓ 개별 IAM 사용자 생성
- ✓ 그룹을 사용하여 권한 할당
- ⚠ IAM 비밀번호 정책 적용

AWS 계정 설정 (cont.)

-> IAM 그룹 생성하기
그룹 이름 : ConsoleAdmin
접근 권한 :

1. AmazonEC2FullAccess
2. AmazonRoute53FullAccess
3. AmazonDocDBFullAccess
4. ElasticLoadBalancingFullAccess
5. AWSLambdaFullAccess
6. AmazonAPIGatewayAdministrator
7. AmazonS3FullAccess

   or

1. AdministratorAcces

-> IAM 그룹 생성하기 2
그룹 이름 : S3Admin
접근 권한 : AmazonS3FullAccess

-> IAM 유저 생성하기
사용자이름 : [nickname or anyname]
엑세스 유형 : AWS Management Console

-> IAM 유저 생성하기 2
사용자 이름 : S3Admin
엑세스 유형 :
AWS Management Console 액세스,
프로그래밍 방식 액세스
credentials.csv 파일 저장, 안전한 곳에 보관!
까먹지 안도록 파일 이름 변경 ex)[aws계정이름]_s3_admin.csv

IAM 유저로 로그인

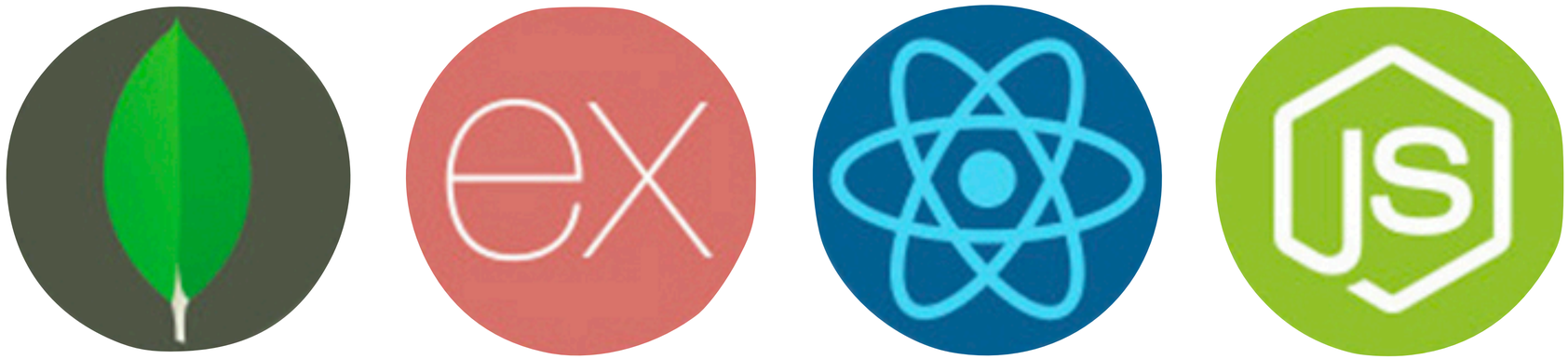
-> 로그아웃

-> console.aws.amazon.com

-> **계정 별칭**에 아까 설정한 계정 별칭 입력

-> IAM 유저 이름, 비밀번호로 로그인

MERN 스택



M E R N

-> 우리가 항상 쓰는 그 스택!

Amazon Web Service

웹 사이트 구동을 위해 필요한 서비스

-> Elastic Compute Cloud (EC2)

-> Simple Storage Service (S3)

-> DocumentDB (DocumentDB)

-> Route53

-> Cloud Front

-> Elastic Load Balancing (ELB)

-> Auto Scaling Group

-> Lambda

-> API Gateway

Amazon Web Service

웹 사이트 구동을 위해 필요한 서비스

-> Elastic Compute Cloud (EC2)

-> Simple Storage Service (S3)

-> DocumentDB (DocumentDB)

-> ~~Route53~~

-> ~~Cloud Front~~

-> Elastic Load Balancing (ELB)

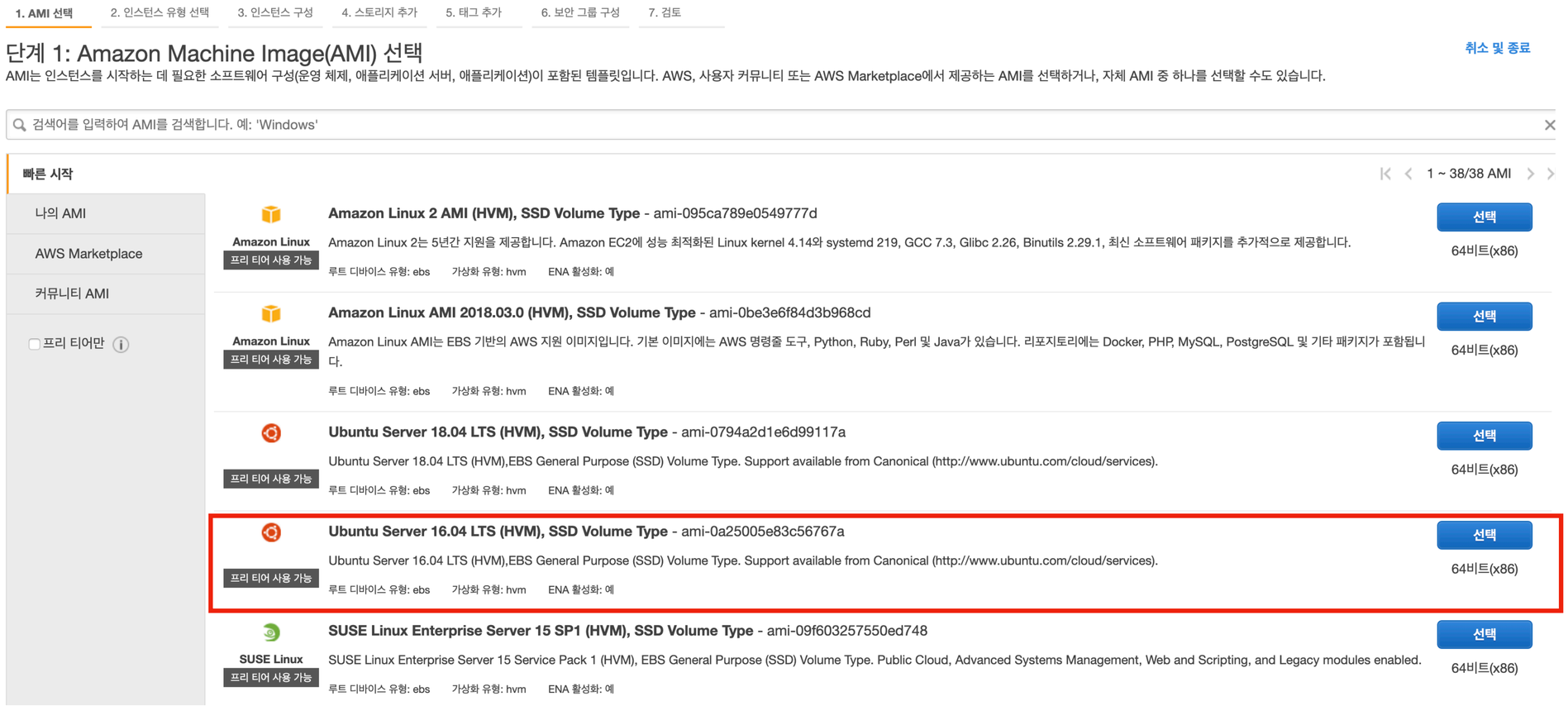
-> Auto Scaling Group

-> ~~Lambda~~

-> ~~API Gateway~~

EC2

-> 서비스 -> EC2 -> 인스턴스 -> 인스턴스 시작 ->



*** Ubuntu Xenial 선택**

EC2

-> 서비스 -> EC2 -> 인스턴스 -> 인스턴스 시작 -> Ubuntu Xenial ->

1. AMI 선택 | 2. 인스턴스 유형 선택 | 3. 인스턴스 구성 | 4. 스토리지 추가 | 5. 태그 추가 | 6. 보안 그룹 구성 | 7. 검토

단계 2: 인스턴스 유형 선택

Amazon EC2는 각 사용 사례에 맞게 최적화된 다양한 인스턴스 유형을 제공합니다. 인스턴스는 애플리케이션을 실행할 수 있는 가상 서버입니다. 이러한 인스턴스에는 CPU, 메모리, 스토리지 및 네트워킹 용량의 다양한 조합이 있으며, 애플리케이션에 사용할 적절한 리소스 조합을 유연하게 선택할 수 있습니다. 인스턴스 유형과 이 인스턴스 유형이 컴퓨팅 요건을 충족하는 방식에 대해 자세히 알아보기

필터링 기준: 모든 인스턴스 유형 | 현재 세대 | 열 표시/숨기기

현재 선택된 항목: t2.micro (Variable ECU, 1 vCPUs, 2.5 GHz, Intel Xeon Family, 1 GiB 메모리, EBS 전용)

	그룹	유형	vCPUs	메모리 (GiB)	인스턴스 스토리지 (GB)	EBS 최적화 사용 가능	네트워크 성능	IPv6 지원
☐	General purpose	t2.nano	1	0.5	EBS 전용	-	낮음에서 중간	예
☑	General purpose	t2.micro 프리 티어 사용 가능	1	1	EBS 전용	-	낮음에서 중간	예
☐	General purpose	t2.small	1	2	EBS 전용	-	낮음에서 중간	예
☐	General purpose	t2.medium	2	4	EBS 전용	-	낮음에서 중간	예
☐	General purpose	t2.large	2	8	EBS 전용	-	낮음에서 중간	예
☐	General purpose	t2.xlarge	4	16	EBS 전용	-	보통	예
☐	General purpose	t2.2xlarge	8	32	EBS 전용	-	보통	예
☐	General purpose	t3.nano	2	0.5	EBS 전용	예	Up to 5 Gigabit	예
☐	General purpose	t3.micro	2	1	EBS 전용	예	Up to 5 Gigabit	예
☐	General purpose	t3.small	2	2	EBS 전용	예	Up to 5 Gigabit	예
☐	General purpose	t3.medium	2	4	EBS 전용	예	Up to 5 Gigabit	예
☐	General purpose	t3.large	2	8	EBS 전용	예	Up to 5 Gigabit	예
☐	General purpose	t3.xlarge	4	16	EBS 전용	예	Up to 5 Gigabit	예

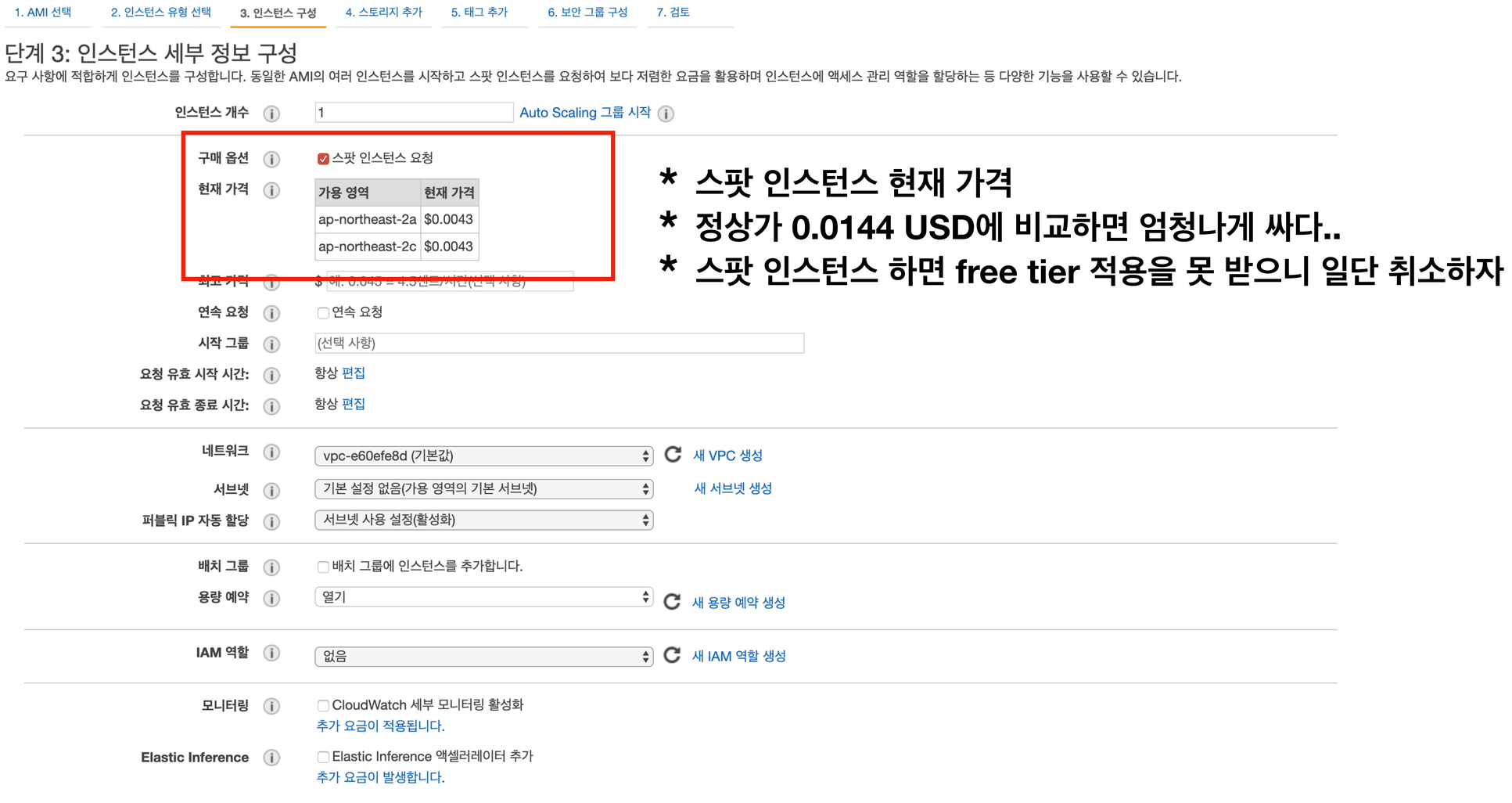
*** Free tier 선택**

EC2

-> 서비스 -> EC2 -> 인스턴스 -> 인스턴스 시작 -> Ubuntu Xenial -> t2.micro ->

EC2

-> 서비스 -> EC2 -> 인스턴스 -> 인스턴스 시작 -> Ubuntu Xenial -> t2.micro -> 스팟 인스턴스?

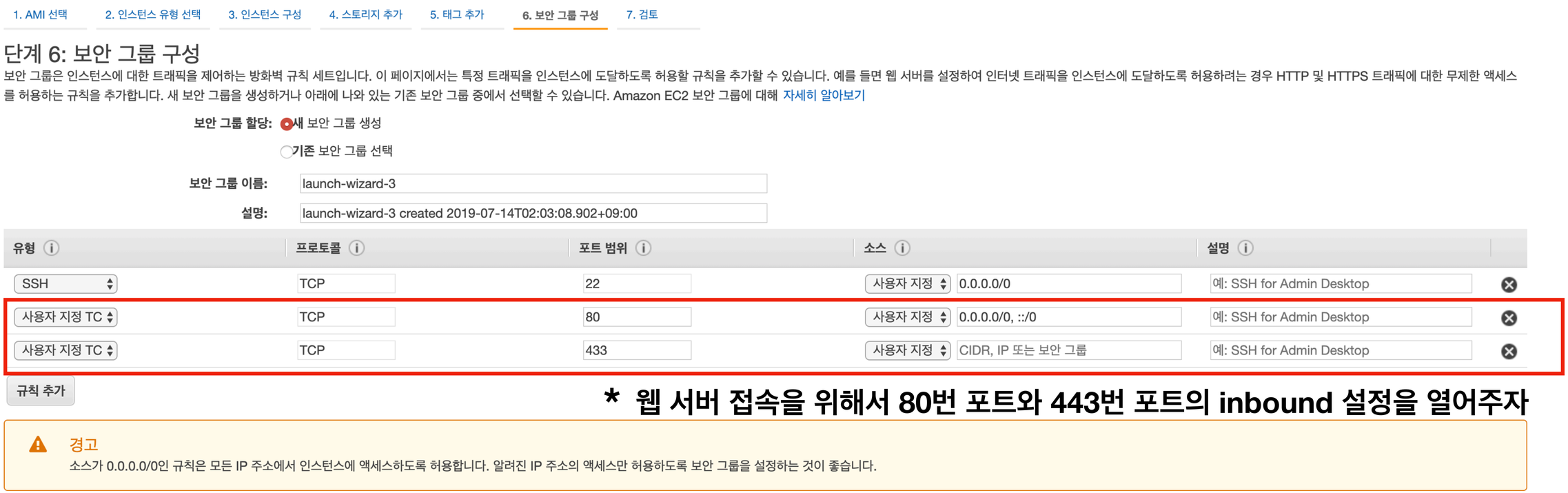


* 스팟 인스턴스 현재 가격
* 정상가 0.0144 USD에 비교하면 엄청나게 싸다..
* 스팟 인스턴스 하면 free tier 적용을 못 받으니 일단 취소하자

EC2

-> **스토리지 -> 태그**는 쭉쭉 넘어가자

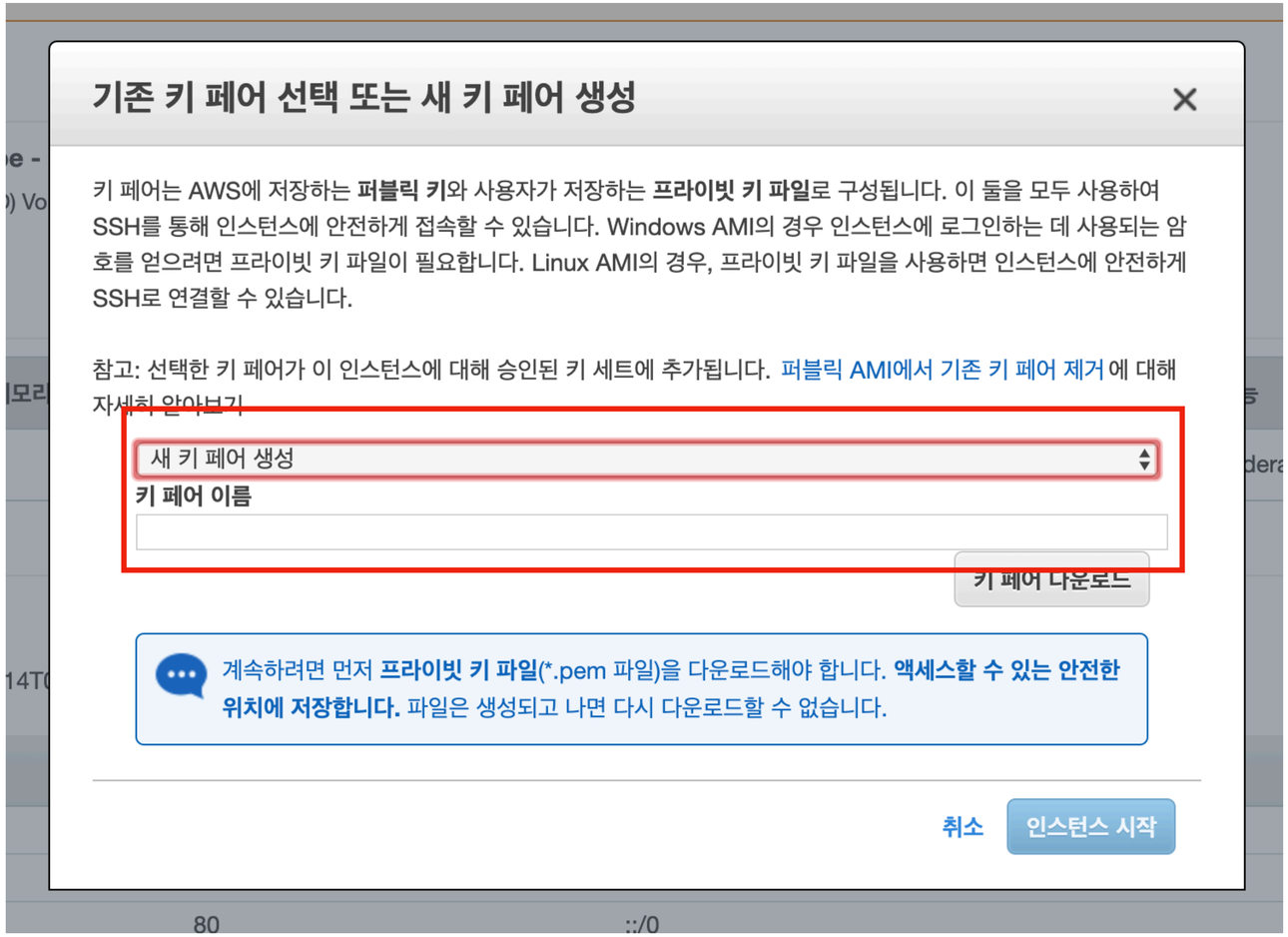
-> 서비스 -> EC2 -> 인스턴스 -> 인스턴스 시작 -> Ubuntu Xenial -> t2.micro -> ~~스팟 인스턴스?~~ -> 스토리지 -> 태그 -> 보안 그룹 구성 ->



*** 웹 서버 접속을 위해서 80번 포트와 443번 포트의 inbound 설정을 열어주자**

EC2

-> 서비스 -> EC2 -> 인스턴스 -> 인스턴스 시작 -> Ubuntu Xenial -> t2.micro -> ~~스팟 인스턴스~~? -> 스토리지 -> 태그 -> 보안 그룹 구성 -> 시작하기



* Ssh 접속을 위한 비대칭 키 생성을 하는 단계다.
* 이름 짓는게 어려우면 [nickname]_toy 로 저장
* 저장된 pem 파일은 ~/.ssh 폴더에 저장한다.

EC2

-> 새롭게 생성된 EC2 인스턴스에는 동적 IP가 할당되어있다.

-> 이 동적 IP는 인스턴스의 상태가 변할 때 마다 새로 배정되므로 정적 IP를 할당하자. *이전 네트워크 세미나에서 사용한 인스턴스 IP가 뜬금없이 카이버라인 서비스를 가리키게 된게 이것 때문.

-> Elastic IP (탄력적 IP) -> 새 주소 할당 -> 주소 연결 -> 방금 생성한 instance의 private IP에 연결

*Elastic IP도 돈이 나간다. 조금 특이한게 EC2 프리 티어 인스턴스에 연결이 되어있으면 돈이 안나가는데 연결되지 않은 상태로 놀고 있으면 돈이 나간다. 인스턴스 종료 후 IP가 더이상 필요하지 않으면 Elastic IP도 꼭 릴리즈 하도록 하자!

-> 인스턴스 -> 인스턴스 클릭 -> IPv4 퍼블릭 IP 복사 -> ssh ubuntu@[public_ip] -i [다운로드한 pem 경로 ex)~/.ssh/cookie_toy.pem]

EC2

*꿀팁

Linux 사용자면 ~/.ssh/config 파일을 열어서 아래 내용을 넣어두면 ssh 접속을 `ssh toy` 명령어로 간단하게 할 수 있다.

```
Host toy
    HostName [public_ip]
    User ubuntu
    IdentityFile ~/.ssh/[nickname]_toy.pem
```

EC2

-> NVM 설치
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.34.0/install.sh | bash
설치가 완료되면 안내 메시지에 따라서 마지막 세 줄을 복사해서 실행하거나 ssh 연결을 종료하고 재연결

-> Node 설치
nvm install --lts
설치가 완료되면 node -v 명령어로 설치가 잘 되었는지 확인
npm install -g yarn 으로 패키지 관리자 설치

-> Create React App으로 리엑트 어플리케이션 생성
npx create-react-app my-react-app
완료되면 package 설치 :
cd my-react-app
yarn && yarn build

-> express generator로 express 서버 생성
npx express-generator myserver
완료되면 package 설치 :
cd myserver
yarn

-> Node Process Manager인 PM2 설치
npm install -g pm2
설치가 완료되면 home 폴더로 이동 후

* 오래 걸리는 작업이니 터미널을 두 개 띄워놓고 진행하자

EC2 - 서버 배포

-> 서버를 구동해보자!

-> Myserver 폴더로 들어가서 서버를 pm2로 실행

DEBUG=myserver:* pm2 start bin/www

pm2 list 명령어로 서버가 잘 작동하고 있는지 체크

pm2 logs 명령어로 로그 체크

-> Express 서버가 3000번 포트에 돌고 있다. -> http://[public_ip]:3000 에 접속해보자

EC2 - 서버 배포

-> http://[public_ip]:3000 에 접속해도 아무일도 일어나지 않는다. (?)

-> EC2의 방화벽 설정에 3000번 포트가 세상에 열려있지 않기 때문

-> EC2 콘솔 -> 보안 그룹 -> launch-wizard-1 -> 인바운드 -> 편집 -> 규칙추가 -> 사용자 지정 TCP, 3000번 포트, 소스 0.0.0.0/0 -> 저장

-> 다시 접속해보자

-> 웰컴 투 익스프레스가 잘 보인다

EC2 - 프론트 배포

-> 이쯤에서 react 앱 빌드가 끝났을거다. 즉 ~/my-react-app/build 폴더에 리엑트 빌드 파일들이 생성되어있음. (없다면 my-react-app 디렉토리 안에서 yarn build 명령어 다시 실행)

-> React 배포는 어떻게 ?
1. express로 static file을 서빙한다. X (express는 static file 서빙에 매우 비효율적임)
2. nginx로 static file을 서빙한다. O (nginx는 static file 서빙에 매우 강하다)

-> Nginx 설치
sudo apt update
sudo apt install nginx -y
설치가 완료되면
sudo service nginx status 명령어로 상태 확인

-> nginx는 기본적으로 80번 포트를 먹고 있다. 그리고 80번 포트의 접근은 instance를 생성할 때 미리 열어두었다!
http://[public_ip] 에 접속하여 확인해보자

EC2 - 프론트 배포

-> Nginx 설정을 하자
cd /etc/nginx

-> 기존 설정 파일을 날리자
sudo rm sites-enabled/default

-> 새로운 설정 파일을 만들자
sudo vi sites-available/my-react-app

-> sudo ln -s /etc/nginx/sites-available/my-react-app /etc/nginx/sites-enabled/my-react-app

-> sudo nginx -t

-> sudo service nginx reload

-> http://[public_ip]에 접속해서 테스트해보자

```
server {
    listen 80;
    server_name _;

    root /home/ubuntu/my-react-app/build
    index index.html

    location / {
        try_files $uri $uri/ =404;
    }
}
```

EC2 - 배포 완성

-> 서버의 모든 라우트 앞에 /api를 붙이자

vi myserver/app.js

// 수정할 내용들

```
app.use('/api', indexRouter);
app.use('/api/users', usersRouter);
```

-> Nginx에서 /api로 시작하는 요청은 서버로 보내자

```
server {
    listen 80;
    server_name _;

    root /home/ubuntu/my-react-app/build
    index index.html

    location / {
        try_files $uri $uri/ =404;
    }
    location /api {
        proxy_set_header X-Forwarded-For $remote_addr;
        proxy_set_header Host $http_host;
        proxy_pass "http://127.0.0.1:3000";
    }
}
```

S3 - AWS CLI

-> S3를 써서 프론트를 배포해보자

-> AWS cli 를 설치해야함 (https://docs.aws.amazon.com/ko_kr/cli/latest/userguide/cli-chap-install.html)
python이 있어야함 -> EC2 Ubuntu Xenial에는 python3가 기본적으로 깔려있음
pip3 (파이썬 패키지 매니저)가 있어야함 -> sudo apt install python3-pip

-> LOCALE 설정이 망가져있음
export LC_ALL="en_US.UTF-8"
export LC_CTYPE="en_US.UTF-8"
sudo dpkg-reconfigure locales

-> AWS CLI를 다운로드
pip3 install awscli --upgrade --user
which aws => aws를 찾을 수 없음
echo 'PATH="/home/ubuntu/.local/bin:$PATH"' >> .bashrc
exec “$SHELL"

S3 - AWS CLI

-> AWS CLI 설정을 해줘야함

-> Credential (IAM 유저 생성하고 발급받은 credentials.csv 파일 안의 access_key 와 secret_key)
home directory로 가서

```
mkdir .aws
vi .aws/credentials

[default]
aws_access_key_id=xxx
aws_secret_access_key=xxx
```

-> Config
vi .aws/config

```
[default]
region=ap-northeast-2
output=json
```

S3 - AWS CLI

여러 가지 명령어를 활용해보자

-> aws s3 mb s3://[unique_bucket_name] (ex: aws s3 mb s3://2019-sparcs-wheel-cookie-bucket)

-> aws s3 ls

-> aws s3 cp my-react-app/README.md s3://2019-sparcs-wheel-cookie-bucket

-> aws s3 mb s3://2019-sparcs-wheel-cookie-bucket2

-> aws s3 mv s3://2019-sparcs-wheel-cookie-bucket/README.md s3://2019-sparcs-wheel-cookie-bucket2

-> aws s3 rb s3://2019-sparcs-wheel-cookie-bucket2

-> aws s3 rb s3://2019-sparcs-wheel-cookie-bucket

S3

-> AWS 콘솔 -> 서비스 -> S3 -> 버킷 만들기 -> Unique한 버킷 이름 (ex: wheel-seminar-cookie) -> 서울 리젼 선택 -> 다음 -> 옵션구성 : 다음 -> 권한설정 : 퍼블릭 엑세스 차단 선택 해제 => 다음, 만들기

-> 생성된 버킷 선택 -> 속성 -> 정적 웹 사이트 호스팅 -> 인덱스 문서 : index.html, 오류 문서 : index.html -> 저장

-> 권한 -> 버킷 정책 -> 정책 생성기 ->

S3 - 정책

Step 1: Select Policy Type

A Policy is a container for permissions. The different types of policies you can create are an IAM Policy, an S3 Bucket Policy, an SNS Topic Policy, a VPC Endpoint Policy, and an SQS Queue Policy.

Select Type of Policy S3 Bucket Policy *** S3 선택**

Step 2: Add Statement(s)

A statement is the formal description of a single permission. See a description of elements that you can use in statements.

Effect Allow Deny

Principal * *** 별 (*)**

Use a comma to separate multiple values.

AWS Service Amazon S3 All Services ('*')

Use multiple statements to add permissions for more than one service.

Actions 1 Action(s) Selected All Actions ('*')

Amazon Resource Name (ARN) *** GetObject 선택**

::<bucket_name>/<key_name>.

- GetEncryptionConfiguration
- GetInventoryConfiguration
- GetLifecycleConfiguration
- GetMetricsConfiguration
- GetObject
- GetObjectAcl
- GetObjectLegalHold

lid. You must enter a valid ARN.

S3 - 정책

Step 2: Add Statement(s)

A statement is the formal description of a single permission. See a description of elements that you can use in statements.

Effect ◉ Allow ○ Deny

Principal *

Use a comma to separate multiple values.

AWS Service Amazon S3 ☐ All Services ('*')

Use multiple statements to add permissions for more than one service.

Actions 1 Action(s) Selected ☐ All Actions ('*')

Amazon Resource Name (ARN) arn:aws:s3:::wheel-sen

*** arn:aws:s3:::<bucket_name>/***

ARN should follow the following format: arn:aws:s3:::<bucket_name>/<key_name>.
Use a comma to separate multiple values.

Add Conditions (Optional)

Add Statement

*** Add Statement 클릭**

S3 - 정책

You added the following statements. Click the button below to Generate a policy.

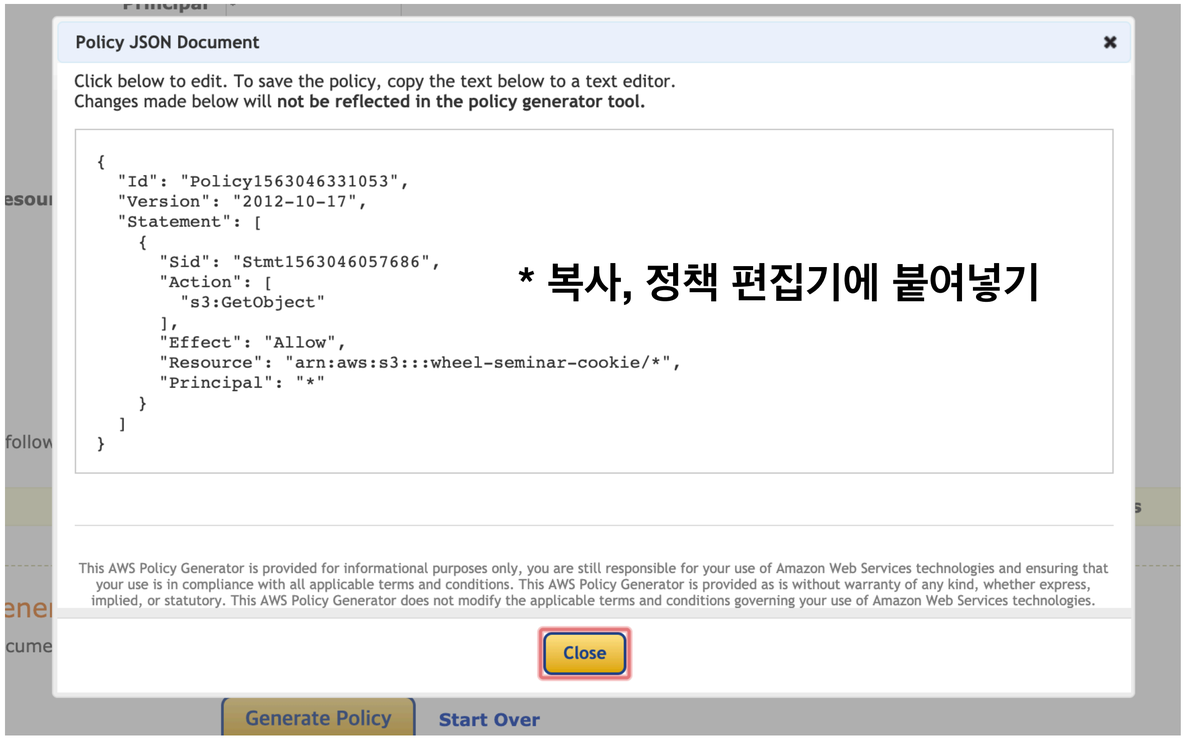
Principal(s)	Effect	Action	Resource	Conditions
• *	Allow	• s3:GetObject	arn:aws:s3:::wheel-seminar-cookie/*	*None*

Step 3: Generate Policy

A *policy* is a document (written in the Access Policy Language) that acts as a container for one or more statements.

Generate Policy Start Over

*** Generate Policy**



*** 복사, 정책 편집기에 붙여넣기**

S3

-> EC2 인스턴스로 돌아가서

-> aws s3 sync my-react-app/build s3://[방금_정책_설정한_버킷]

-> 버킷의 주소로 들어가보자
https://[bucket_name].s3.ap-northeast-2.amazonaws.com

Document DB - 생성

*주의 DocumentDB는 Free Tier가 아님.

*서울 리젼의 가장 싼 인스턴스 r5.large 의 가격은 시간당 0.334$, 월 240$, 후딱하고 지우자

-> Document DB는 **저장**(스토리지)과 **처리**(컴퓨팅)를 각각 **클러스터**와 **인스턴스**가 나누어 처리한다.

-> **클러스터**는 데이터를 관리하는 저장소이다.

-> **인스턴스**는 클러스터의 데이터를 읽고 쓰는 역할을 한다.

-> 한 클러스터는 최소 1개에서 16개 까지의 인스턴스를 가질 수 있다.
자세한 내용은 (https://docs.aws.amazon.com/ko_kr/documentdb/latest/developerguide/db-clusters-understanding.html)

-> AWS 콘솔 -> 서비스 -> Amazon Document DB(클러스터) -> 생성 -> 인스턴스 개수 1개, 고급 설정 - Delete Protection 설정 해제 -> 만들기

Document DB - 접속

Document DB 접속하기

* mongodb-client 설치 - (https://docs.aws.amazon.com/ko_kr/documentdb/latest/developerguide/getting-started.connect.html)

-> EC2 접속

-> sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv A2930ADAE8CAF5059EE73BB4B58712A2291FA4AD5

-> echo "deb [arch=amd64,arm64] https://repo.mongodb.org/apt/ubuntu xenial/mongodb-org/3.6 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-3.6.list

-> sudo apt-get update

-> sudo apt-get install -y mongodb-org-shell

-> Amazon DocumentDB 퍼블릭 키 다운로드 (TLS 연결을 하기 위해서 필수)
wget https://s3.amazonaws.com/rds-downloads/rds-combined-ca-bundle.pem

-> DocumentDB **클러스터** 상세 정보 -> 연결 -> mongo 쉘 연결 방식 복사 -> EC2에서 실행해보자

Document DB

-> after 5000ms milliseconds, giving up. 에러 발생 (실패하는게 정상)

왜 ?
Document DB 인스턴스를 생성했을 때 이 인스턴스에 대한 네트워크 접근 권한이 누구에게도 없다!
그 설정은 어디있을까? EC2의 보안그룹 설정에 default 라는 이름으로 생성되어있음

보안그룹?
AWS에서 네트워크 통신을 관리하기 위한 방화벽 설정을 말한다. EC2 콘솔에서 손쉽게 수정할 수 있음.

-> EC2 보안그룹 -> default 보안그룹 -> 인바운드 설정 -> 추가 [유형 : 모든 트레픽, 소스 : EC2의 보안그룹 (launch-wizard-1 또는 EC2의 보안그룹)] 추가

-> Document DB 연결 재시도

-> 연결 성공!

Document DB - 놀이터

```
db.inventory.insert(
   {
      "SKU" : "38220349",
      "Description" : "14 oz. whole wheat bread",
      "Vendor" : {
         "Name" : "Example Bakery, LLC.",
         "Street" : "321 No. Gilbert",
         "City" : "Anytown",
         "State-Provence" : "CA",
         "Country" : "USA",
         "Phone" : "000-555-0100",
         "Contacts" : [
            {
            "LName" : "Stiles",
            "FName" : "Jonathan",
            "FriendlyName" : "JJ",
            "Role" : "Sales",
            "Phone" : "000-555-0111"
            }
         ],
      },
      "Stock" : {
         "OnHand" : "4",
         "OrderWhenBelow" : "6",
         "OrderQuantity" : "8"
      },
      "UnitPrice" : {
         "Wholesale" : "1.71",
         "Retail" : "2.39"
      }
   }
)
```

```
db.inventory.find( {} ).pretty()
```

```
db.inventory.update(
    { "SKU" : "38220349" },
    {
      $set: { "UnitPrice" : { "Wholesale" :
"1.87", "Retail" : "2.69", "DayOld" : "1.89" } }
    }
)
```

*** 이것 저것 해보자. 하지만 여기서 끝이 아니니 조금만!**

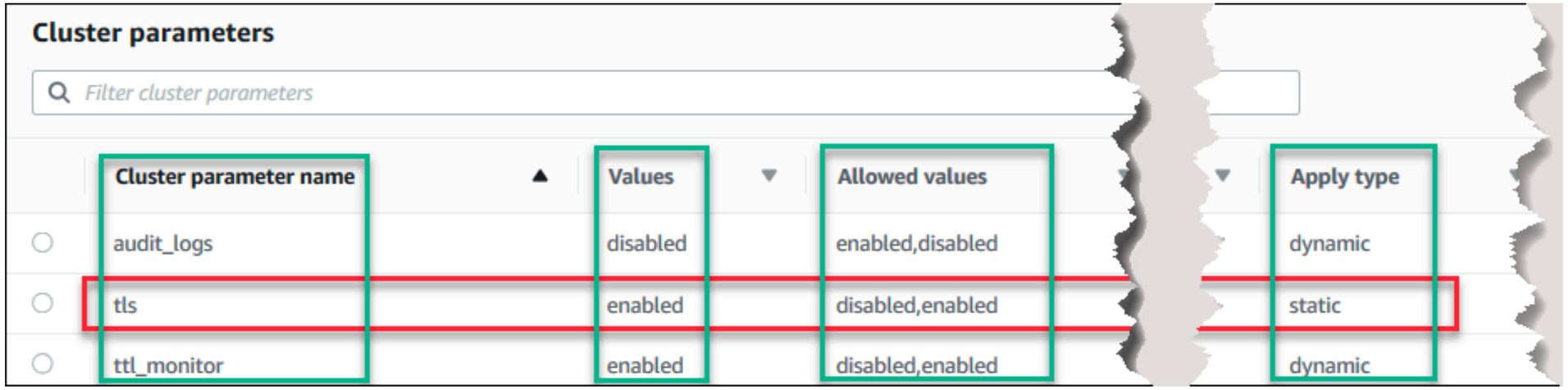
Document DB - Parameter Group

-> Document DB 인스턴스를 생성하면 기본 설정으로 TLS 설정이 켜져있음.

-> 서버에서 Document DB에 연결하는 실습을 할건데 이게 켜져있으면 너무 귀찮음! TLS 설정을 끄는 실습부터 해보자.

-> AWS 콘솔 -> DocumentDB -> Parameter Group -> 생성 -> 아무 이름이나 입력하고 생성 -> 생성된 paramter group 클릭 -> TLS 설정을 disabled 로 편집

* apply type이 static이라고 되어있는 parameter 값은 변경 후 instance를 재부팅해야 적용이 된다.



DocumentDB - Node 연결

재부팅을 기다리는 동안.. NodeJS 서버 세팅을 해보자

-> EC2 인스턴스 접속

-> 홈 디렉토리 또는 자기가 좋아하는 폴더에서
git clone https://github.com/jungdj/express-mongo-boilerplate

-> cd express-mongo-boilerplate

-> yarn

-> Document DB 주소를 환경 변수로 넣어줘야한다.
cp config/.env.example config/.env.development
vi config/.env.development
아래 값을 채워넣자
==>> MONGODB_CONNECTION_URL=mongo://<username>:<password>@docdb-2019-……
docdb.amazon.com/<만들고 싶은 db이름>
* 위 값은 AWSConsole > DocumentDB > Cluster 안에서 확인할 수 있다. **제일 뒤에 :27017 지워야함 주의!**

-> pm2 start ecosystem.config.js --env development

DocumentDB - Node 연결

-> 콘솔창을 하나 더 열어서 서버가 잘 작동하는지 테스트해볼 수 있다.

-> curl -X GET http://localhost:6001/user/list

-> curl -X POST http://localhost:6001/user -d '{ "firstName": "dongjin",
"lastName": "Jung" }’

-> curl -X GET http://localhost:6001/user/list

DocumentDB - Node 배포

-> 서버의 모든 라우트 앞에 /api를 붙이자

vi src/app.js

// 수정할 내용들

~~app.use('/', routes);~~

app.use('/api', routes);

-> Nginx에서 /api로 시작하는 요청은 서버로 보내자

```
server {
    listen 80;
    server_name _;

    root /home/ubuntu/my-react-app/build
    index index.html

    location / {
        try_files $uri $uri/ =404;
    }
    location /api {
        proxy_set_header X-Forwarded-For $remote_addr;
        proxy_set_header Host $http_host;
        ~~proxy_pass “http://127.0.0.1:3000”;~~
        proxy_pass "http://127.0.0.1:6001";
    }
}
```

Elastic Load Balancer

with Auto Scaling Group

AWS의 꽃

대규모 서비스 배포 준비를 해보자!

ELB

들어가기 전에 일단 이거부터 해두자 (오래걸려서)

EC2 콘솔 -> 인스턴스 -> 인스턴스 선택 -> 작업 -> 이미지 -> 이미지 생성 -> 이름 마음대로

ELB

대기업의 시작점

ex) ELB가 만약 회사 인사팀이라면..

- 공평한 업무 분담 (10분에 한 번씩 일 잘하고 있나 물어본다. 절대 농땡이 부릴 수 없다.)
- 유연한 고용 / 해고 (일이 많아지면 채용, 일이 적어지면 바로 해고)
- 사내 건강 관리 (주기적인 건강검진. 건강하지 않으면 바로 해고)
- 인터넷에 연결할 수 있는 방법이 ELB를 통하는 것 외에 아무것도 없어서 신고도 못한다.

~~나였으면 벌써 ㅈX..~~

ELB

Elastic Load Balancing (ELB)

시스템에 가해지는 부하를 여러대의 시스템으로 분산.

대규모 시스템의 시작점

ex) 서비스에 ELB를 띄우면..

- ~~공평한 업무~~ 트래픽 분산 (컴퓨팅 파워가 충분한 컴퓨터를 찾아 요청을 분산해준다.)
- 유연한 ~~고용 / 해고~~ 자원 확보 (컴퓨팅 파워가 부족하면 인스턴스를 추가, 요청이 줄어들면 다시 인스턴스를 삭제한다.)
- 사내 건강 관리(**Health Check**) (주기적으로 죽은 프로세스가 없는지 체크한다. 문제가 발생한 인스턴스는 삭제)
- 인터넷에 연결할 수 있는 방법이 ELB를 통하는 것 외에 아무것도 없어서 ~~신고도 못한다.~~ 안전하다.

Elastic Load Balancer

-> 현재까지 세팅해놓은 인스턴스의 snapshot을 찍어야함 (똑같은 시스템을 여러개로 복제할 거니깐!)

-> 시스템 부팅 되면서 Node 서버가 바로 실행 되어야함 (Node Process를 systemd (시스템 데몬)으로 생성, 부팅 시 해당 systemd이 실행되도록 enable)

이 복잡한 과정을 pm2가 간단하게 해결해준다.

-> pm2 startup

-> 실행 결과를 복사 붙여넣기 해서 실행

-> pm2 save (현재 돌고 있는 pm2 process들을 저장)

Elastic Load Balancer

EC2 콘솔 -> Load Balancer -> 로드 밸런서 생성 -> Application Load Balancer ->

1. 밸런서 구성

체계 : 내부 인터넷 연결 (원래 보안상 내부를 쓰는 것이 좋으나, 이 옵션을 선택하면 ELB에 접근하기 위해 API Gateway나 Route53 등의 다른 서비스를 한 번 거쳐야한다.)

가용 영역 : 3개 모두 체크. 분리된 3 개의 지역 중 어디에 인스턴스를 배치할지 결정하는 것

***가용 영역은 다른 가용 영역에서 장애가 발생할 경우 격리되도록 설계된 AWS 데이터 센터 내 물리적인 위치**

-> 다음

2. 보안 설정 구성

-> 패스

3. 보안 그룹 구성

기존 보안 그룹 선택 -> EC2에서 사용하고 있는 보안그룹 선택

-> 다음

4. 라우팅 구성 - 다음 페이지 사진 참조

대상 그룹

대상 그룹 ⓘ 새 대상 그룹

이름 ⓘ targets

대상 유형
- (●) 인스턴스
- ( ) IP
- ( ) Lambda 함수

프로토콜 ⓘ HTTP

포트 ⓘ 80

상태 검사

프로토콜 ⓘ HTTP

경로 ⓘ /api/hc

▼ 고급 상태 검사 설정

포트 ⓘ
- (●) 트래픽 포트
- ( ) 재정의

정상 임계 값 ⓘ 2

비정상 임계값 ⓘ 2

제한 시간 ⓘ 2 초

간격 ⓘ 5 초

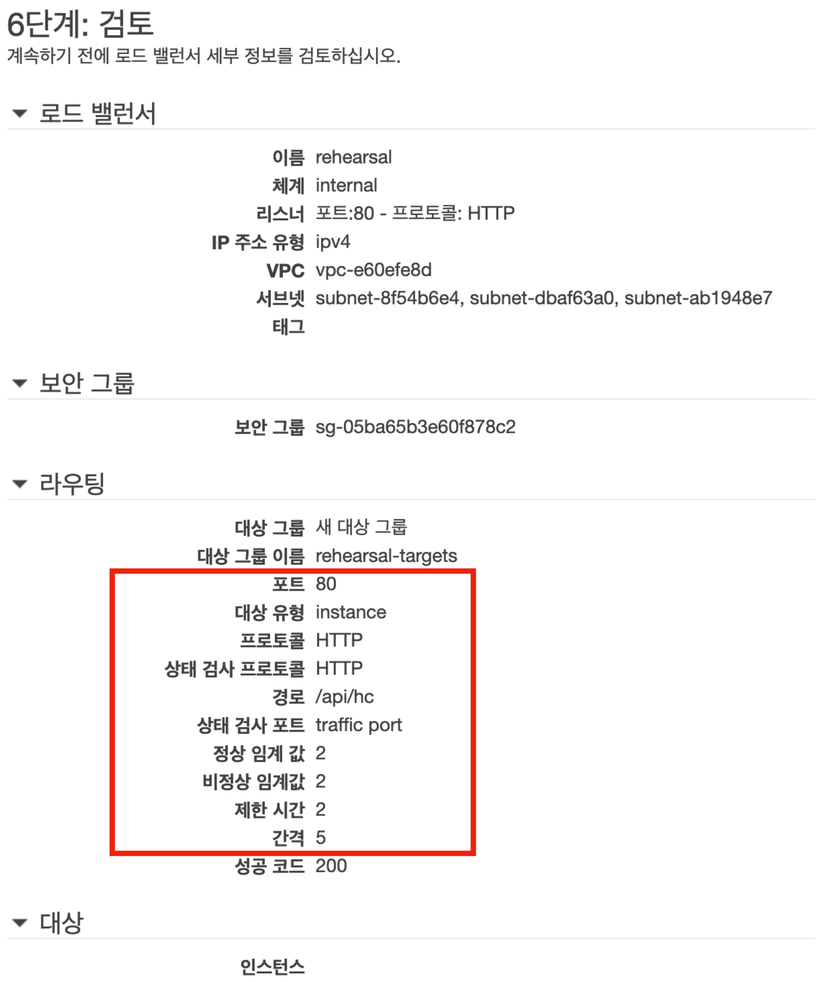
성공 코드 ⓘ 200

Elastic Load Balancer

5단계: 대상 등록

-> 패스

6단계: 검토



6단계: 검토

계속하기 전에 로드 밸런서 세부 정보를 검토하십시오.

▼ 로드 밸런서

이름	rehearsal
체계	internal
리스너	포트:80 - 프로토콜: HTTP
IP 주소 유형	ipv4
VPC	vpc-e60efe8d
서브넷	subnet-8f54b6e4, subnet-dbaf63a0, subnet-ab1948e7
태그	

▼ 보안 그룹

보안 그룹	sg-05ba65b3e60f878c2

▼ 라우팅

대상 그룹	새 대상 그룹
대상 그룹 이름	rehearsal-targets
포트	80
대상 유형	instance
프로토콜	HTTP
상태 검사 프로토콜	HTTP
경로	/api/hc
상태 검사 포트	traffic port
정상 임계 값	2
비정상 임계값	2
제한 시간	2
간격	5
성공 코드	200

▼ 대상

인스턴스

ELB - 성공?

-> 로드 벨런서 -> 설명 (기본 구성) - DNS 이름 ex) internal-rehearsal-1260646136.ap-northeast-2.elb.amazonaws.com

ELB - 성공?

아직 안 됨

대상 인스턴스를 추가 안했으니까

Auto Scaling Group

Auto Scaling Group

-> EC2 콘솔 -> Auto Scaling Group -> 시작 구성 생성 -> 내 AMI -> [만들어둔 AMI] -> 쭉쭉… 보안 그룹 구성

-> 규칙 추가, Custom TCP Rule, 80번 포트, source : ELB 대상그룹의 보안 그룹

-> 쭉쭉 … -> **만들기 -> 갖고 있는 pem 으로 생성**

-> Auto Scaling 그룹 생성 -> 그룹 이름 마음대로, **그룹크기 2,** 서브넷 3개 모두 추가, **고급 세부 정보 => 대상 그룹 (방금 만든 대상그룹), 상태 유예 검사 : 10초**

-> **조정 정책 구성**
1 ~ 3 개 사이에서 조정
평균 CPU 사용률 : 5 (*기준성능, 피피티 앞부분 자료 참고)
인스턴스 필요 시간 : 0

-> **쭉쭉… -> 생성**

END