ICT 인턴십
9월 2주차 과제

- Restful API, MSA, DevOps 보고서 -

Issue Date : 2018/09/18

KCC정보통신
인턴 신지영

- 목 차 -

1. Restful API

1) Restful API란?

- REST(REpresentational State Transfer)
  : 웹에 존재하는 모든 자원에 고유한 URI을 부여해 활용하는 Software Architecture

- API(Application Programming Interface)
  : 운영체제 및 프로그래밍 언어가 제공하는 기능을 제어할 수 있게 해주는 인터페이스,
  웹 API는 다른 서비스에 요청을 보내고 응답을 받기 위해 정의된 명세
  ex) 블로그 API, 구글 지도 API 오픈 API 등

- Restful한 API
  : REST 특징을 지키며 API를 제공하는 것

- 왜 필요한가?
  : 애플리케이션 분리 및 통합 이슈 해결, 다양한 클라이언트의 등장에 대응

1)-1. REST의 구조

REST = 자원(URI) + 행위(HTTP Method) + 표현(Representations)
= 리소스 + 메소드 + 메세지
= 대상 자원 + 자원에 대한 행위 + 행위의 내용

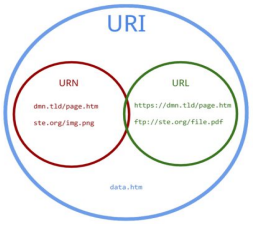
HTTP POST, http://myweb/users/

```
{
        "users":{
                "name" : "shin"
        }
}
```

---

① 자원(Resource)

- 메소드를 수행할 대상
- URI(Uniform Resource Identifier)
  : 인터넷에 있는 자원을 나타내는 유일한 주소
  ex) https://www.google.co.kr/search?q=uri
  · URL(Uniform Resource Locator) : 네트워크 상에서 자원이 어디 있는지를 알려주기 위한 규약
  ex) https://www.google.co.kr/search
  · URN(Uniform Resource Name) : 콘텐츠의 리소스 위치에 영향 받지 않는 유일무이한 이름



<그림 1. URI 구조>

② 행위(Method)

- 자원에 대한 행위
- HTTP Service for Restful API (CRUD)

HTTP Method	CRUD	기능
HTTP POST	Create	생성
HTTP GET	Read	조회
HTTP PUT	Update	수정
HTTP DELETE	Delete	삭제

③ Message

- 메소드의 내용을 정의한 것
- 메세지 포맷으로는 json, xml 등이 있음

2) 특징

1)-1. REST의 특징

① Uniform Interface
: Client가 특정 플랫폼·언어·기술에 종속되지 않아, HTTP를 사용하는 모든 플랫폼에서 요청 가능
② Stateless
: HTTP를 따라 갖는 속성으로 상태정보를 따로 저장하지 않고, 들어오는 요청만을 각각 단순 처리
③ Casheable
: 웹의 기존 인프라를 그대로 사용 가능하여, HTTP의 장점 중 하나인 캐싱 기능 적용
④ 자체 표현 구조
: 동사(Method) + 명사(URI)의 구조로, 어떤 메서드에서 무슨 행위를 하는지 쉽게 파악 가능

---

ex) Restful URI (X) : http://aaa.bbb.net/news/view.do?ncd=3421128
Restful URI (O) : http://aaa.bbb.net/**boards**/**1**/**posts**/**406**
→ resource = **게시판들** 중 **첫번째 게시판**의 **글들** 중 **406번째** 글

---

⑤ Client-Server 구조
: REST Server(API 제공)와 Client(사용자 인증 및 컨텍스트 관리)의 역할이 명확히 분리
⑥ 계층형 구조
: Client 입장에선 REST API 서버만 호출하지만, 순수 비즈니스 로직을 수행하는 API 서버에
(사용자 인증 + 암호화 + 로드밸런싱)을 추가하여 다중 계층으로 구성 가능

1)-2. 단점

: 명시적인 표준이 없음 / DB Table의 RESTful한 설계 필요(JSON을 그대로 저장하는 MongoDB가 유리)

3) REST 예제

기능	HTTP Method	HTTP URI & BODY
모든 회원 정보 조회	GET	http://www.aaa.com/users
특정 회원 정보 조회	GET	http://www.aaa.com/users/shin
회원 정보 검색	GET	http://www.aaa.com/users?query=xxx

회원 등록	POST	http://www.aaa.com/users
		{
		“name” : “shin”,
		...
		}

회원 삭제	DELETE	http://www.aaa.com/users/shin

특정 회원 정보 수정	PUT	http://www.aaa.com/users/shin
		{
		“name” : “shin”,
		“address” : “seoul”
		}

- GET과 POST의 차이점 : “GET은 가져오는 것(조회), POST는 수행하는 것(등록)”
→ 보안성 : GET < POST / 속도 : GET > POST
- POST와 나머지 명령어의 차이점 : POST는 idempotent 하지 않은 명령어
→ idempotent X = 여러 번 수행하면 결과값에 변화가 있음
idempotent O = 여러 번 수행해도 결과값이 변하지 않음

2. MSA(Micro Service Architecture)

1) Monolithic Architecture?

1)-1. 개념

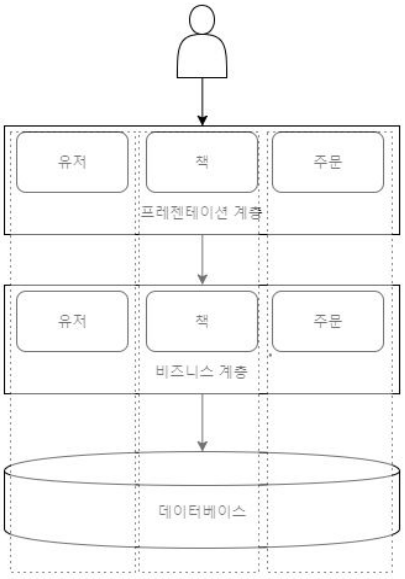
: 단일 응용 프로그램에 모든 비즈니스 로직이 묶여있는 구조

1)-2. 장·단점

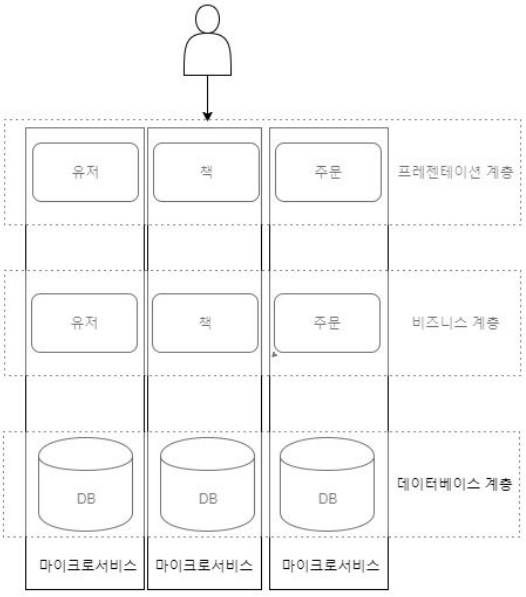
- [장점]
  ⓐ 기술의 단일화 가능
  ⓑ 배포 및 테스트 용이

- [단점]
  ⓐ 빌드 및 배포시간이 길어짐
  ⓑ 개발 언어에 종속적
  ⓒ 하나의 서비스가 모든 서비스에 영향을 줌

2) Microservice Architecture란?



<그림 2. MLA구조>



<그림 3. MSA구조>

- Micro Service = small + API로 타 서비스와 연계 + 자율적 + 한가지 일을 잘 함
- Microservice Architecture
  : 단일 응용 프로그램을 나누어 작은 서비스의 조합으로 구성하는 방법

2)-1. 장·단점

- [장점]
  ⓐ 빌드 및 테스트 시간 단축
  ⓑ 상황별 유연한 기술 적용
  ⓒ 하나의 서비스가 다른 서비스에 영향을 미치지 않음

- [단점]
  ⓐ HTTP 메소드 사용으로 인한 성능 이슈 발생
  ⓑ 트랜잭션의 번거로움
  ⓒ 관리 항목 증가

2)-2. 왜 필요한가?

ⓐ Frontend / Backend 분리하여 각각 독립적인 개발 가능
ⓑ 코드의 양을 줄임
ⓒ Working Group별 Product 생성
ⓓ Micro-Service 단위로 구현의 자율성 부여(Polyglot)
ⓔ Micro-Service 단위의 재사용, 자유로운 리팩토링

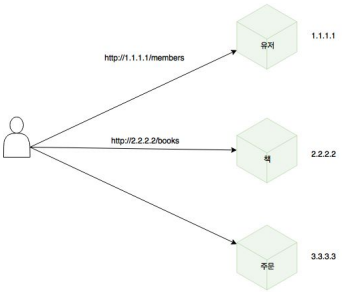
3) API GATEWAY

3)-1. 개념

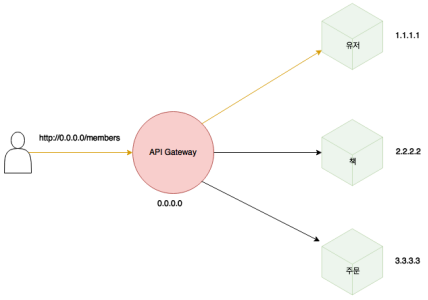
: API 서버 앞단에서 모든 API 서버들의 End-Point를 단일화하여 묶어주는 서버
→ Micro Service에서 RESTful 기반의 Request, Response에 대해 중앙에서 관장해 주는 역할

3)-2. 기능

① API 요청을 한 곳에서 받아서 해당 서비스로 이동(라우팅)시킴



<그림 4. API Gateway 없는 구조>

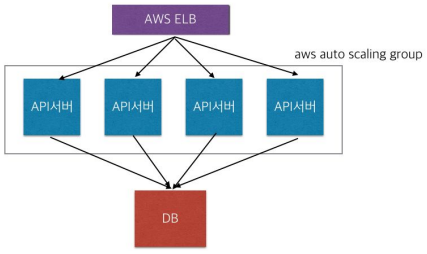


<그림 5. API Gateway 있는 구조>

② 보안 강화 : Gateway는 public으로, 각 서비스들은 private로 구축
③ 각 서비스의 공통 로직 처리 : 로깅, CORS(Cross-Origin Resource Sharing), 보안 및 인증, 모니터링 등

4) Microservice Architecture 사례

4)-1. 배달의 민족

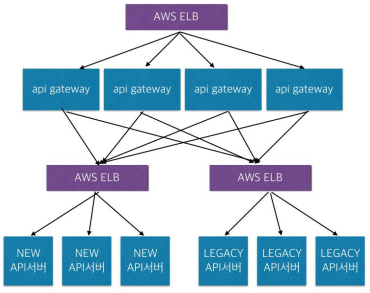


① 과거 API System Architecture

- Monolithic Architecture 구조
- AWS 사용
- DB의 도메인 분리 필요

→ 일부 수정하면, 전체를 배포해야 함
→ 서비스의 유연한 운영이 어려움

<그림 6. 과거 배달의 민족 API 구조>



② 현재 API System Architecture

- 기존에 사용했던 AWS ELB에 API Gateway 구성
- 2개의 ELB를 추가 생성하여,
  각각 기존의 LEGACY API 서버와 새로운 API서버로 구성
- Netflix의 GATEWAY인 'Zuul'처럼 동적 Filter 정의

→ 실시간으로 이슈에 대응 가능

<그림 7. 현재 배달의 민족 API 구조>

3. DevOps

1) DevOps란?

- DevOps = 'Development' + 'Operations'

= 소프트웨어 개발 운영

→ 개발과 운영을 합쳐, 한 조직내에서 서비스를 독립적으로 개발 및 운영할 수 있는 협업 체계이자 개발 문화

- 등장 배경

· 개발자 & IT/Ops 전문가는 서로 다른 목표(대개 경쟁하는 목표), 부서 리더십, 주요 성과 지표를 가짐

· 그 결과 두 조직은 사일로화되어 각자의 분야나 투입 시간, 릴리스 실패, 고객 불만에만 신경 씀

· 두 조직간의 소통 부재는 최종 결과물의 완성도에 악영향을 미침

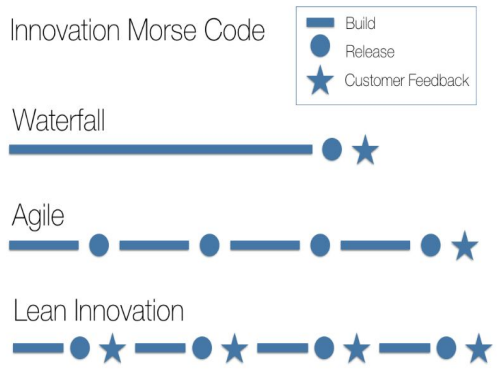
→ 개발과 운영이 분리되어있어 발생하는 문제점을 개선하기 위해 나온 방법

- 목적 : 신속하게 고성능 소프트웨어를 제공하는 것

- Agile, Lean 등의 개발 방식과 함께 시작된 확장된 범위의 개발 방식

1)-1. 다양한 개발 방식

:



① Waterfall

: 요구사항 분석 → 설계 → 개발 → 테스트 → 배포 → Market Feedback

(일이 순차적으로 진행되고, 다시 되돌아가기 힘든 구조)

② Agile

개발 → 배포 → 개발 → 배포 → … → Market Feedback

(요구사항에 맞춘 간단한 프로토타입을 만들고, 피드백에 따라 발전시켜나감)

③ Lean Start Up

: Agile에서 발생되는 '코드의 낭비'를 줄인 방식

(최소한의 프로토타입을 만들고, 쓸모없는 기능은 과감히 pivot해나감)

<그림 8. 다양한 개발 방식>

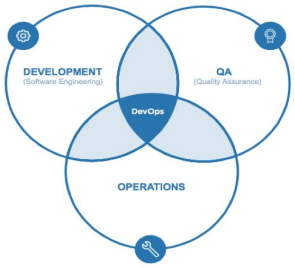
1)-2. 각 개발 방식의 Best Target

① 사용자의 문제점과 솔루션을 정확히 파악하고 있는 경우 : Waterfall

② 사용자의 문제점은 명확하지만, 솔루션은 모호한 경우 : Agile

③ 사용자의 문제점과 솔루션 둘 다 모호한 경우 : Lean

2) DevOps의 Best Model



① Cross Functional Team

: 한 팀 내에서 서비스와 관련된 모든 업무(기획, 운영, 개발, 영업 등)를 할 수 있는 구조로 팀을 구성

② Widely Shared Metrics

: 팀 전체가 기준으로 삼을 수 있는 서비스 관련 공통 지표(Metric)가 필요

③ Automating Repetitive Tasks

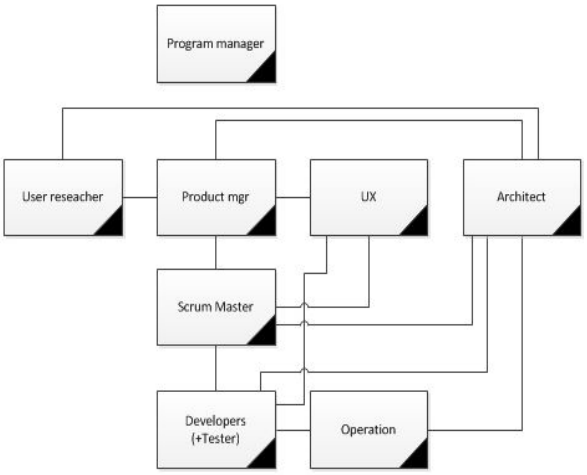
: 반복적인 작업을 툴을 이용해서 자동화

<그림 9. DevOps 방식>

④ Post Mortems
: 장애나 이슈가 있을때, 처리 후에 해당 내용을 전체 팀과 공유

⑤ Regular Release
: 정기적으로 릴리즈 주기를 설정

3) DevOps기반 Team



<그림 10. DevOps Team 조직 구조>

3)-1. 조직 구조

① Program Manager
: 전체 서비스(개발, 운영, 전체 서비스 기획)를 관장

② Product Manager
: 서비스를 기획 & 요구 사항 정의 & 우선 순위 지정

③ UX
: PM과 교류하며, UX 디자인을 프로토타입~개발 단계까지 정의
& 사용자의 피드백에 맞춰 지속적으로 UX를 개선

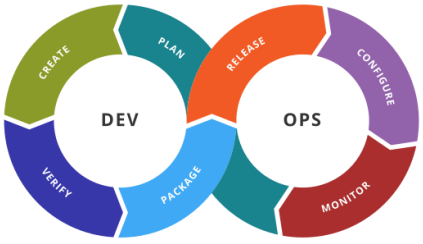
④ Scrum Master(=Project Leader)
: 실제 개발팀을 이끔 & 일정관리, 개발 리소스 관리등을 담당

⑤ Developers(+Tester)
: 소프트웨어 개발 & 테스트

⑥ Contents Writer/Technical Writer
: 서비스의 컨텐츠 작성 & 리뷰 & 다국어 번역

⑦ User Researcher
: PM보다 먼저, 주요 제품 로드맵을 정의 & 시장 상황 분석
& 수익 구조 및 비즈니스 모델을 정의

3)-2. 개발 Cycle



<그림 11. DevOps Cycle>

: 아이디어 → Test Case 작성 → 개발 → Test → 통합 → 배포 → 모니터링 → Feedback 수집 → (반복)

→ 기존 Waterfall 방식의 Cycle
: 개발팀과 운영팀이 각각 진행한 업무를 서로에게 던지고 잊어 버리는 (fire & forget) 형태
→ DevOps 방식의 Cycle
: 각 팀의 업무를 서로 던지는 것이 아니라, 과정 내내 같이 수행
요구 사항을 개발팀에 넘겨도, 개발팀과 계속 협의를 하면서 요구 사항을 구체화 및 개선

3)-3. DevOps Team 구성 시 주의사항

① DevOps Team을 따로 만들지 말 것
: 기존의 개발 및 운영팀으로 구성해 추가적인 업무를 만들지 말 것
② DevOps 엔지니어를 채용하지 말 것
: 문화적 변화는 수동적으로 시킨다고 되는 것이 아니므로, 경영진들 스스로 DevOps에 관심을 갖고
자발적으로 DevOps 문화를 구성해야 성공적인 변화가 가능함
③ 소규모 스타트업 기업에 유리. 조직이 큰 경우 인내심을 가지고 차근차근 적용해 나가야 함

4. 참고자료

1) Restful API

1)-1. 홈페이지

① Restful API Tutorial 사이트 / https://www.restapitutorial.com/lessons/httpmethods.html
② 'brainbackdoor' tistory 블로그 / https://brainbackdoor.tistory.com/53
③ '망나니개발자' tistory 블로그 http://mangkyu.tistory.com/46
④ 위키백과 'API' / https://ko.wikipedia.org/wiki/API

1)-2. 사진

① 그림 1 / https://danielmiessler.com/study/url-uri/

2) MSA(Micro Service Architecture)

2)-1. 홈페이지

① 'AlwaysPr' tistory 블로그 / http://alwayspr.tistory.com/19
② '배달의 민족 기술' 블로그 / http://woowabros.github.io/r&d/2017/06/13/apigateway.html
③ '월간 지앤선' tistory 블로그 / http://monthly-jiandson.tistory.com/12

2)-2. 사진

① 그림 2, 3, 4, 5 : 'AlwaysPr' tistory 블로그 / http://alwayspr.tistory.com/19

3) DevOps

3)-1. 홈페이지

① '조대협의 블로그' 블로그 http://bcho.tistory.com/817

3)-2. 사진

① 그림 8 : https://www.insightstudios.co/blog/2016/2/17/how-lean-innovation-is-different-from-the-status-quo
② 그림 9 : https://sdtimes.com/report-leaders-need-address-data-center-constraints-devops-evolution/