기술 구조도 설명

1. 구조 설계 목표

• 기능 요구와 비기능 요구를 반드시 만족시켜야 한다.

• 최대한 완숙한 오픈 소시 기술로 원가 절약한다.

• 구조가 간단함, 안전하게 업그레이드 및 확장, 신축한다.

• 전면적으로 정확하게 시스템 구조의 범위를 확정하며 정의 내부 시스템과 외부 시스템의 경계와 인터랙션 협의을 명확하게 한다.

• 각 서브 시스템의 구체적인 책임, 서브 시스템 간의 인터랙션 협의와 과정을 확정한다.

• 시스템의 개발, 배정과 유지 보수 플랫폼 및 해당 규범을 확정한다.

• 시스템에게 후속 코딩, 테스트, 유지보수를 위한 원칙, 기초와 규범을 설계해 준다.

2. 구조 설계 원칙

• **기능성 요구와 비기능성 요구를 만족시킨다.** 이는 소프트 웨어 시스템 가장 기본적인 요구이며 구조 설계 시 따라야 하는 제일 기본적인 원칙이다.

• **실용성 원칙.** 각 소프트 웨어 시스템을 유저에게 교부해 사용 시 실용적이여야 한다. 실제 문제를 해결해 주면서 시스템이 안정적이고 성능이 양호해야 한다.

• **중복사용 가능 원칙**. 표준 모듈은 중복 사용 가능한 미들웨어 혹은 공통 컴포넌트로 설계해 최대하게 개발 인원의 작업 효율성을 높인다.

• **단일 책임 원칙.** 각 서브시스템이 하나의 특정 기능을 환성하는데에 집중해 겹치거나 결합하지 않으며 시스템 간에 설계된 양호한 API 를 통해 인터랙션을 한다.

• **확장 가능 원칙.** 서비스 제공하는 방향으로 생각한다. 업무 모듈 서브 시스템이 서비스를 제공하는 방식을 통해 대외적으로 API 를 제공한다. 추후 업무 확정 필요 시 직접 서비스를 확장하면 된다. 시스템이 업무량의 크기에 따라 유연하게 배정한다. 즉 집중식 배정이나 분산식 배정 다 가능하다.

3. 상세 설명

3.1 앱 단 및 웹 단 서버 통신이 https 를 사용한다.

1) 인증 서버. 앱 단에 신뢰하는 CA 기관 인증서가 설치한다. 첫 단계에서 서버가 CA 기관을 통해 다운받은 서버 인증서를 제공한다. 만약 해당 서버 인증서의 CA 기관을 인증하게 되면 앱 단 신뢰받은 CA 기관 리스트에 저장해 서버 인증서의 정보와 현재 방문한 사이트(도메인)일치하면 앱 단은 서버 단이 신뢰가 가능하는 것을 인정해 서버 인증서에서 서버 공개 키를 획득하며 후속 프로세스에 적용한다. 아니면 앱 단은 요청을 보내지 않는다.

2) 협상 회화 키. 앱 단은 서버 인증을 완료하고 서버 공개 키를 받은 다음에 해당 공개 키를 이용해 서버와 암호화 통신을 진행한다. 협상을 통해 2 개 회화 키를 조성한다. 앱 단에서 서버 단에게 발송하는 데이터를 암호화하는 앱 단 회화 키와 서버 단에서 앱 단에게 보낸 데이터를 암호화하는 서버 단 회화 키이다. 현재 존재한 서버 공개 키가 암호화 통신하는 전제 하에 2 개의 대칭 키의 원인을 협상해야 한다. 왜냐하면 비대칭 암호화 상대적으로 더 복잡함으로 데이터 전송하는 과정에 대칭 암호화를 사용하면 컴퓨터 자원을 절약할 수 있다. 이 외에 회화 키가 랜덤으로 생성된 것으로 매번 협상할 때 다른 결과가 나와서 안전성도 상대적으로 높은 편이다.

3) 암호화 통신. 이 때 앱 단과 서버 두 측 다 이번 통신의 회화 키를 갖게 되고 그 이후로 전송한 모든 http 데이터가 회화 키를 통해 암호화한다. 이렇게 하면 네트워크에 기타 유저가 앱 단 과 서버 단 간의 전송하는 데이터를 훔치거나 수정하기가 어렵다. 그러므로 데이터의 완전성과 은밀성을 보장해 준다.

3.2 앱 단 인증서 위조를 방지하기 위해 앱 단 인증서 잠금 기술을 사용한다.

만약 유저 핸드폰에 악성 인증서를 설치되어 있으면 중간자 공격하는 방식으로 유저 통신을 도청하거나 equest 혹은 response 중에 데이터를 수정할 수 있다.

• 핸드폰 중간자 공격 과정:

1) 앱 단 실행 시 데이터를 전송하기 전에 앱 단과 서버 단에서 핸들링을 해야 한다. 핸들링하는 과정에서 쌍방 데이터를 암호화해서 전송하는 패스워드 정보를 확정한다.

2) 중간자가 이 과정에서 앱 단 포스트한 서버 핸들링 정보를 차단한 후에 앱 단을 모방해 서버에게(자신의 지원한 암호화 규칙을 서버에게 보낸다)포스트한다. 서버는 이 중에서 한 조의 암호 알고리즘과 해쉬 알고리즘을 골라 지신의 신분 정보를 인증서 형식으로 앱단에게 돌려 보낸다, 인증서에 사이트 주소, 암호화 키, 인증서의 발표 기관 등 정보가 들어 있다.

3) 이때 중간자가 서버 단이 앱 단에게 돌려 보내는 인증서 정보를 차단해 자신의 인증서 정보로 교체한다.

4) 앱 단이 중간자의 response 를 받은 후에 중간자의 인증서로 암호화 데이터 전송하는 것을 진행한다.

5) 중간자가 앱 단 요청하는 데이터를 받은 후에 자신의 인증서로 해독한다.

6) 데이터를 도청 혹은 수정 요청을 거친 다음에 앱 단 데이터 포스트 암호화를 요청하는 것을 모방한다. 이는 중간자 공격 전부 과정입니다.

• 방어 방법:

1) 공개 키 잠금

앱 단 apk 에 인증서 공개 키를 코딩해 https 통신 시 서버 단에게 전송하는 인증서 공개 키와 apk 거 일치하는지 체크한다.

2) 인증서 잠금

측 앱 단이 발급한 공개 키 인증서가 모바일 앱 단에서 저장해 https 통신할 때 서버 단에서 가져오는 게 아니라 앱 단 코드에서 고정적으로 인증서 정보를 가져 간다.

3.3 앱 게이트웨이거 해당 앱 API 의 로딩밸런신과 용단능력 구비하며 실패 재시도 진행이 가능하다.

1) 로딩 밸런싱

API 의 포스트 스트레스가 어느 순간에 단일한 서버 사례의 처리 능력을 초과할 수도 있다. 이 때는 우리가 여러 개 서버 사례를 시동해야 한다. 포스트 스트레스 다 하나의 사례에 집중적으로 전송하는 것을 피하기 위해 게이트웨이 레이어에서 폴링 책략을 시행한다. 측 매번 포스트할 때 각각 다른 서버 사례로 분산시킨다.

2) 공유 용단

백단 서버가 이상이 생길 때 이를 외부 층에게 던지지 않고 서버가 자동적으로 대운하는 것을 원한다. 어느 서버가 이상이 생기면 직접 우리에게 사전 설치한 정보 리턴해 준다.

자체 정의한 fallback 방법을 통해 이는 지정한 route 에게 해당 route 의 방문 문제가 생길 때 용단 처리를 구현한다.

3) 실패 시 재시도

가끔 네트워크 혹은 기타 원인 때문에 서버가 잠사 사용 불가능하게 된다. 이때는 우리가 서버에 대해 재시도 가능하는 것을 원한다.

우리가 자체 정의 재시도 책략을 통해 만약 어느 서버가 이상이 생기면 기타 서버 사례를 호출하거나 지정 정보를 리턴한다. 재시도가 어떤 상황에 시동하는 것은 문제가 있다. 예를 들어 스트레스 과다하면 하나의 사례가 정지된 경우 공유가 데이터를 측시 다른 사례로 이전하게 되어 최종적으로 모든 사례가 다운을 시키는 것을 초래할 수도 있다.

3.4 캐시 클러스터가 주요 백단과 API 에게 서비스한다.

1) 게이트웨이 레이어

주로 입증, 용단, 로딩밸런싱, 실패 재시도를 처리한다.

2) 앱 서비스 레이어

업무 로직 처리 담당한다.

3) 베이스서비스 레이러

베이스 데이터의 증가, 삭제, 수정, 조회, 그리고 팀 단말기 작업 처리를 제공한다.

3.5 앱 레이어 멀티 데이터 송신부를 사용해 데이터베이스를 방문한다.

1) 두개 혹은 여러 개 데이터베이스가 관련성이 없고 각자 독립적으로 존재한다. 실제로 이런 경우는 두 개 의 프로젝트로 나눠서 개발할 수 있다. 예를 들면 게임 개발에서 하나의 데이터베이스가 플랫폼 데이터베이스이고 별도로 플랫폼 아래 게임에게도 해당하는 데이터베이스가 있다.

2) 두개 혹은 여러 개 데이터베이스가 master-slave 의 관계이다. 예를 들면 mysql 에 master-master 하나가 구축하면 그는 여러 개의 slave 를 갖고 있다. 혹은 MHA 가 구축한 master-slave 복제를 사용한다.

3) spring 세팅 파일로 직접 여려 개의 데이터 송신부와 세팅한다.

관련성이 없는 두 개의 데이터베이스인 경우 직접 spring 세팅 파일에서 여러 개의 데이터 송신부를 세팅해 따로 작업 세팅을 진행한다.

4) AbstractRoutingDataSource 와 AOP 로 한 멀티 데이터 송신부 세팅. 기본적인 원리는 우리가 DataSource 류 ThreadLocalRountingDataSource 하나를 정의해 AbstractRoutingDataSource 를 상속한다. 그 다음에 세팅 파일에서 ThreadLocalRountingDataSource 에게 master 와 slave 의 데이터 송신부를 주입해 AOP 를 통해 유연하게 세팅한다. 어떤 데에 master 데이터 송신부를 선택하거나 어떤 데에 slave 데이터 송신부를 선택해야 한다.

3.6 공공 서비스를 통해서만 블록체인과 IPFS 를 호출할 수 있다.

파일이 IPFS 네트워크에 저장해 데이터가 각 네트워크 노드로 분산된다. 만약 파일을 찾으려면 파일 저장시 리턴받은 해쉬값을 사용해 IPFS 네트워크에서 프라그먼트를 찾아 다시 조립하면 된다.

3.7 데이터베이스 주종(主从)복제가 GTID 방식을 사용한다.

GTID 개요:

1) 전국 작업 표시: global transaction identifieds

2) GTID 작업은 전국적으로 유니크하고 하나의 작업은 하나의 GTID 와 대응한다.

3) 하나의 GTID 가 하나의 서버에서 한번만 집행한다. 중복 집행으로 인한 데이터 혼란 혹은 주종(主从)불일치를 방지한다.

4) GTID 가 classic 복제 방법을 대체해 더이상 binlog+pos 로 복제를 시동한 게 아니라 master_auto_postion=의 방식을 사용해 자동적으로 GTID 과 매칭해 복제를 진행한다.

5) MySQL-5.6.5 부터 1 지원, MySQL-5.6.10 후 완비한다.

6) 전통적인 slave 단, binlog 는 시작하지 않은 것이지만 GTID 에서 slave 단의 binlog 가 반드시 시작된 상태이다. 목적은 집행해던 GTID 를(강제로)기록하는 것이다.

GTID 의 구성 부분 :

앞부분은 server_uuid: 뒷부분에는 일련번호이다.

예를 들면 server_uuid: sequence number

7800a22c-95ae-11e4-983d-080027de205a:10

UUID: 각 mysql 사례 유니크 ID, slave 로 전송하게 돼서 근원 ID 로 이해할 수도 있다。

Sequence number: 각 MySQL 서버에서 전부 다 1 부터 증가하는 일련번호이다. 하나의 수치가 하나의 작업과 대응한다.

전통 복제보단 GTID 의 장점:

1) 더 간단하게 failover 구현한다. 옛날처럼 log_file 과 log_Pos 를 찾을 필요가 없다.

2) 더 간단한 주종(主从)복제를 구축한다.

3) 전통 복제보단 더 안전적이다.

4) GTID 는 연속적으로 구멍이 없는 것이어서 주종(主从)데이터베이스에 데이터 충돌이 나면 널을 추가하는 방식으로 넘어갈 수 있다.

GTID 의 작업 원리 :

1) master 데이터 업데이트할 때 사무 앞에서 GTID 를 생성한고 공동으로 binlog 에서 기록된다.

2) slave 단의 i/o 스레드가 변경한 binlog 를 로컬의 relay log 에 코딩하다.

3) sql 스레드가 relay log 에서 GTID 를 취득하고 난 후에 slave 단의 binlog 에 기록이 있는지에 대해 비교한다.

4) 만약 기록이 있으면 측 해당 GTID 의 사무 이미 집행완료되고 slave 가 넘어간다.

5) 만약 기록이 없으면, slave 가 relay log 에서 해당 GTID 의 작업을 집행해 binlog 에서 기록한다.

6) 파싱하는 과정에 기본 키가 있는지 판단하게 된다. 만약 없으면 이급 인덱스 혹은 전체 스캔을 사용한다.