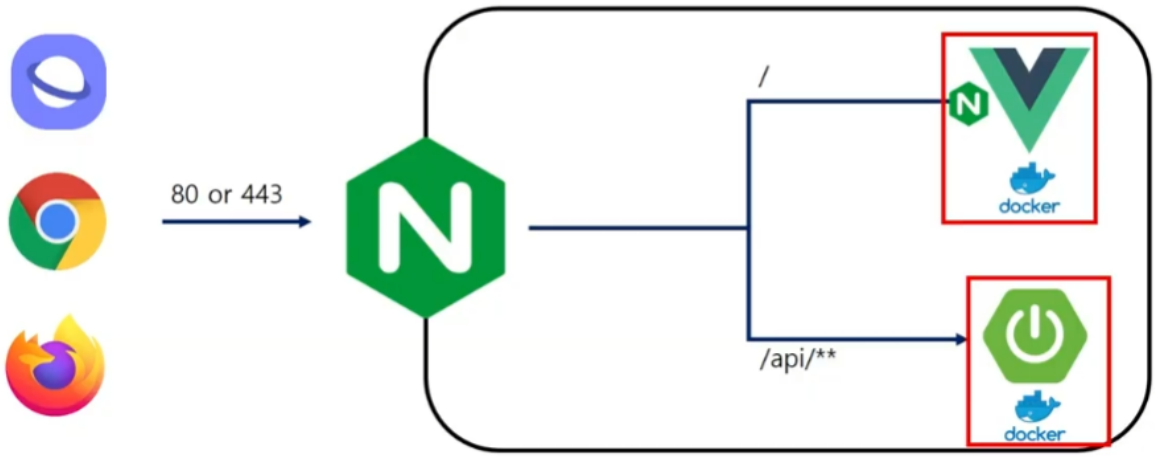
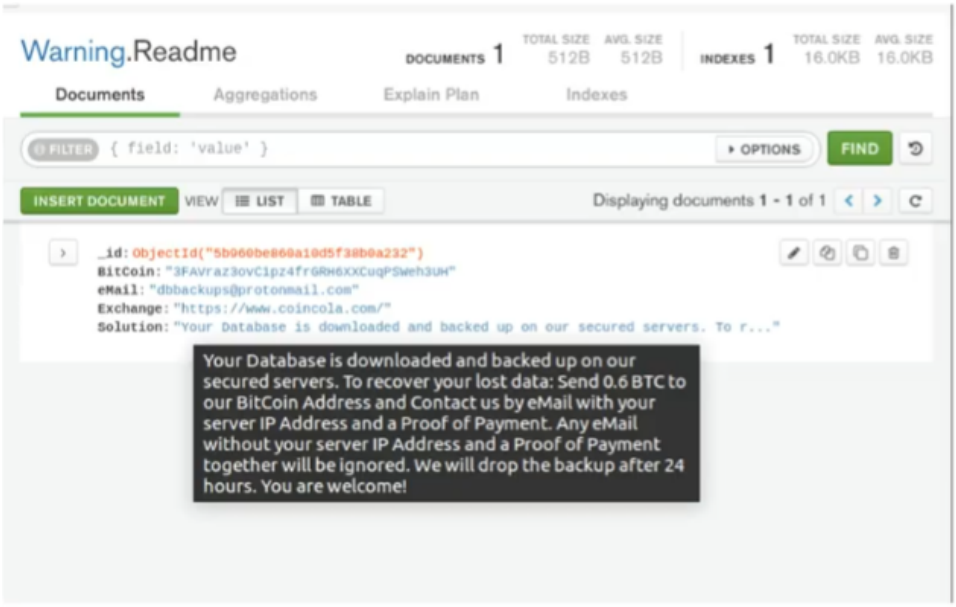
특화 프로젝트 AWS 배포하기

Date	@2021/09/28

공통 프로젝트 배포 구조



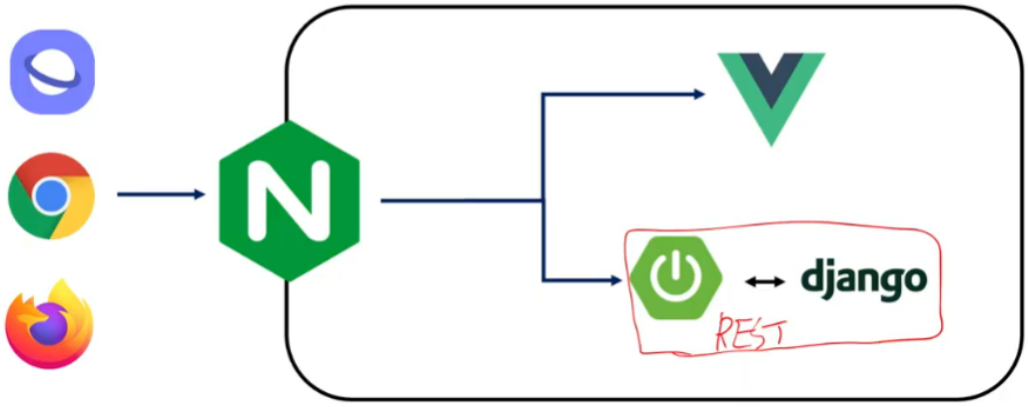
PW 보안을 잘 못한 경우 ...



특화 프로젝트 변화점

- Python 활용(AI, 빅데이터)
- 개발자가 제어할 수 없는 대상과의 통신(블록체인, IoT)
- 공통 프로젝트 대비 복잡한 구조

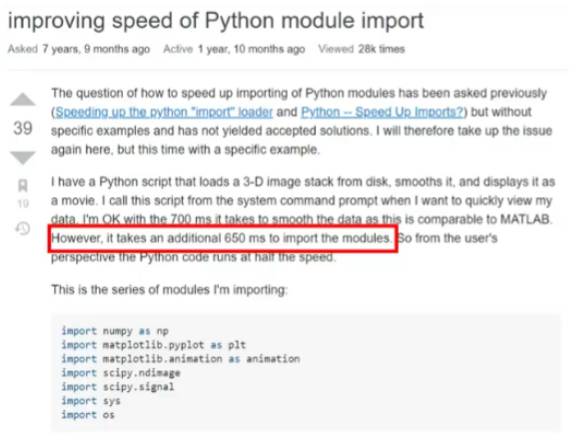
Django 외의 다른 방법은?

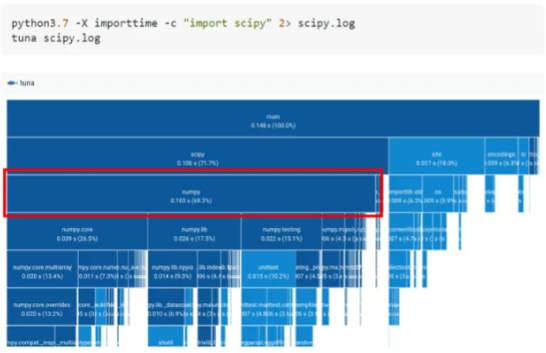


- 학습은 반드시 Python을 활용해야 한다.

- 하지만 REST 서버도 반드시 Python을 써야하는 건 아니다
- 보다 빠른 구현을 원하면 Python+Django 조합을 사용해보고 Spring 학습을 원한다면 REST 서버를 Spring으로 구성하는 것도 좋다

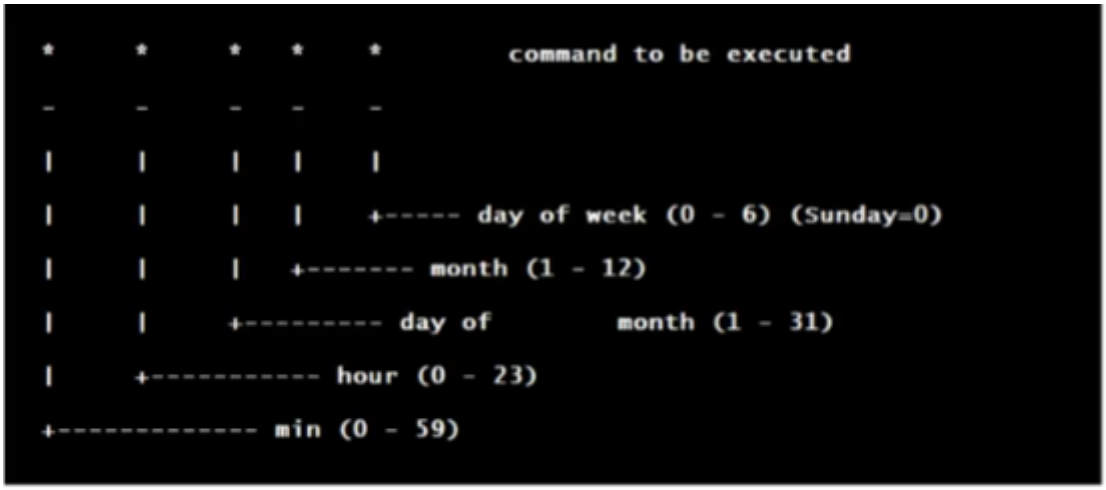
Java 코드로 .py를 실행하면?





- Jython이나 Command line executor로 .py 파일을 실행시키는 것도 가능하지만, Python 파일을 실행시킬 때 의존성 있는 라이브러리를 import 하는 시간이 오래 걸린다.
- 이전 기수의 경우에는 import에만 3초 이상 필요한 경우도 있었음
- Django 서버를 실행시켜놓고 Spring에서 REST 호출을 하면 빠른 실행 가능
- 유저로부터 얻게 되는 추가적인 데이터는 스케쥴링릉 통해서 정기적으로 학습 데이터에 반영할 수 있도록 한다.
- 매번 손으로 학습시키기보다 정기적인 스케쥴링을 통해 학습을 진행하고 결과를 백엔드 서버에 반영하고, 스케쥴링의 결과는 MM을 통해서 받아본다.

스케쥴링



"크론표현식" 이라는 키워드를 잘 기억해서 Report 형식으로 내용을 확인하자.

CronTrigger Tutorial | Quartz.NET

cron is a UNIX tool that has been around for a long time, so its scheduling capabilities are powerful and proven. The CronTrigger class is based on the scheduling capabilities of cron. CronTrigger uses "cron expressions", which are able to create firing schedules such as: "At 8:00am every Monday through Friday"

https://www.quartz-scheduler.net/documentation/quartz-3.x/tutorial/crontrigger.html#special-characters

블록체인

- public blockchain의 경우에는 어디서나 접근이 가능함
- 완전한 탈중앙화된 action을 원한다면 FE에서 blockchain에 접근하는 것이 바람직함
- 그러나 유저 단위로 Key와 주소를 관리하는 서비스의 경우에는 서비스의 백엔드에서 Key와 주소를 관리하고 blockchain에 접근하는 방식도 가능함(ex 암호 화폐 거래소)

Smart Contract

- 이더리움의 Smart Contract을 배포하는 경우에 유의할 점들

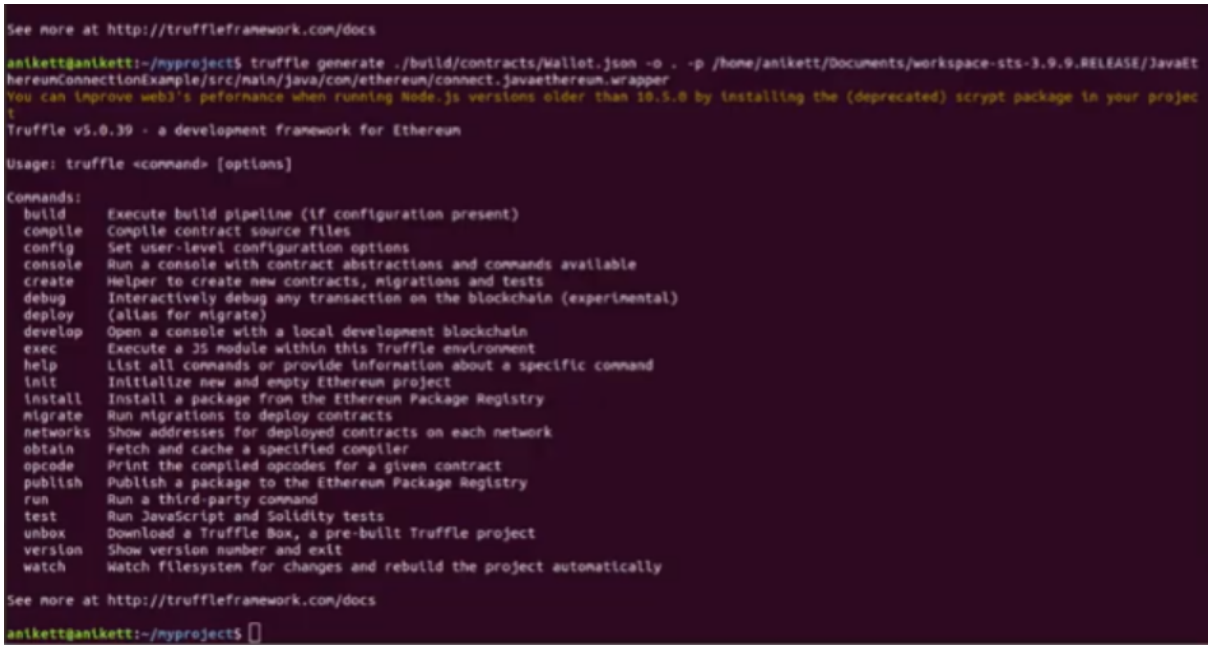
- solidity 파일의 내용(주석이나 공백 포함)이 바뀌는 경우에는 abi와 컨파일된 바이너리를 다시 생성해야 함
- abi는 일치하더라도 바이너리가 변경된다면 호출에 실패함
- 특히 Smart Contract을 호출하기 위해 생성한 Wrapper class(.java 파일)의 경우에는 solidity 파일이 변경된 경우 새롭게 생성해야하기 때문에 이 과정을 수동으로 반복하기 보다 jenkins를 이용해서 자동화 해두면 좋음

```
var solc = require('solc'); //contract Compile
var fs = require('fs'); //file 시스템

app.get('/smartcontract', function(req,res){
  //File Read
  var source = fs.readFileSync("./contracts/HelloWorld.sol", "utf8");

  console.log('transaction...compiling contract .....');
  let compiledContract = solc.compile(source);
  console.log('done!!' + compiledContract);

  var bytecode = '';
  var abi = '';
  for (let contractName in compiledContract.contracts) {
     // code and ABI that are needed by web3
     abi = JSON.parse(compiledContract.contracts[contractName].interface);
     bytecode = compiledContract.contracts[contractName].bytecode; //컨트랙트 생
     // contjson파일을 저장
  }
})
```

.sol 파일 변경 시에 빌드의 흐름

```
const MyContract = new web3.eth.Contract(abi);

let deploy = MyContract.deploy({
               data: bytecode,
               from: send_account}).encodeABI();
```

생성된 bytecode와 abi를 이용해서 deploy



Wrapper class를 생성

- Web3j + solc
- Web3j + truffle
- Gradle
- Maven

IoT (시뮬레이터)

- IoT 제어 시뮬레이터는 윈도우 환경에서 동작한다.
- 우리는 ubuntu 환경의 AWS instance로 진행할때, localhost가 아닌 어디로 진행해야 할지 확인해야한다.

개인 local PC의 public IP를 확인해서 진행하자

포트포워딩으로 접속하기

: 외부 접속이 가능하도록 세팅

- 공유기가 있다면 외부에 있는 AWS instance에서 시뮬레이터까지 연결하는 것이 가능하다. 시뮬레이터에서 사용하는 포트를 PC의 포트로 포워딩 해주면 세팅 끝!
- 남은 PC가 있다면 향후에 간단한 포트폴리오 사이트를 유지하거나 NAS 서버를 구축하는 것도 가능하다.
- 수시로 brute force 방식의 공격이 들어오니 비밀번호는 항상 어렵게 설정하자!

Sub PJT3 부터는 ROS 대신 UDP를 활용

- Sub PJT3부터는 ROS 대신 UDP를 활용해서 통신할 예정
- 프로토콜과 구현부를 수정하면 된다.
- 서버에서 시뮬레이터에 다양한 명령 조합 수행이 가능하다.
- 스케줄러를 활용해서 다양한 기능을 개발할 수 있다.

  (ex. 오후 0에 000 해줘!)