**\* 목차**

1. **aws을 이용한 서버 구축 및 금액**
2. **NodeJs 활용 – 디버그법**
3. **Mysql 연동 + Mybatis**
4. **Redis 연동**
5. **Git / git-hup 이용한 형상관리**
6. **Jenkins 이용한 배포**
7. **NodeJS 성능에 대한 건**
8. **EJS을 이용한 화면 구성 건**

**1.aws을 이용한 서버 구축 및 금액**

**- AWS** 사이트 접속

-루트 사용자 이메일을 사용하여 로그인 선택

**ID/pwd :** **midstar@naver.com !Dl~~~**

**MFA코드 : 안드로이드 Authenticator**

**Name : jang minseok**

**탄력적IP:**[**43.201.220.132**](https://ap-northeast-2.console.aws.amazon.com/ec2/home?region=ap-northeast-2#ElasticIpDetails:PublicIp=43.201.220.132)

* **putty 접속** :

C:\Users\TOURSOFT\Google 드라이브\중요자료

key : ubutu-micro.ppk

왼쪽 카테고리에서 Connection > SSH > Auth> Credeintial

로 이동합니다

"Private key file for authentication" 섹션에서 "Browse" 버튼을 클릭하고 앞서 생성한 .ppk 파일 을 선택합니다

왼쪽 카테고리에서 "Session" 🡪 Saved Sessions"에 원하는 이름을 입력하고 "Save" 버튼을 클릭하여 설정

* **AWS ppk 파일 생성**

**EC2대시보드로 이동 🡪 네트워크 및 보안🡪 키페어 🡪 키 페어생성**

* **AWS 인바운드 규칙 생성**

**EC2대시보드로 이동 🡪 네트워크 및 보안🡪 보안그룹 🡪 해당보안규칙 선택후**

* **인바우드 규칙 편집 🡪 새규칙 생성**

**-SFTP 접속**

**아래 그림처럼 key 파일에 ppk 파일 선택후 해당 ip 넣고 프로토콜을 SFTP 선택후 저장**

텍스트, 스크린샷, 디스플레이, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

**호스트(Host): AWS Ubuntu 인스턴스의 퍼블릭 DNS 또는 IP 주소 입력.**

**예: ec2-203-0-113-25.compute-1.amazonaws.com 또는 203.0.113.25**

**포트(Port): 22 (기본 SSH 포트)**

**프로토콜(Protocol): SFTP - SSH File Transfer Protocol 선택**

**로그온 유형(Logon Type): 키 파일(Key file) 선택**

**사용자(User): ubuntu (Ubuntu 인스턴스의 기본 사용자 이름)**

**키 파일(Key file): .ppk 파일의 경로를 지정합니다.**

**예: C:\path\to\your-key.ppk**

* **AWS 인스턴스 신규 생성은 ubuntu로 생성 법**

**EC2 🡪 인스턴스 🡪 오른쪽 위쪽 메뉴에 인스턴스 시작 선택**

* **Quick Start 🡪 ubuntu로 선택해서 실행하면됨**
* **AWS 청구 금액**

[**과금 정보 및 비용 관리**](https://us-east-1.console.aws.amazon.com/costmanagement/home#/home) **🡪 청구 및 결제 (이번달 예상금액과 이전달 것 청구 내역 )**

**AWS 요금 계산기 이용하여 예상 비용 예측 가능**

**Linux medium으로 1년 선결제시 467.76USD 🡪 약 64~70만원/년간 미만정도**

**2. NodeJS 활용 + 아파치 연동**

* **아파치 설치**

**sudo apt install apache2 -y # Ubuntu**

**sudo systemctl start apache2 # Ubuntu**

* **NodeJs 설치**

**sudo apt update**

**sudo apt install nodejs npm**

**node -v**

**npm -v**

**source ~/.bashrc**

* **아파치 + NodeJS 연동**

**Apache와 Node.js를 연동하여 특정 경로(예: /api)의 요청을 Node.js 애플리케이션으로 전달하는 방법을 설명해드리겠습니다:**

**Apache 모듈 활성화:**

**먼저, 필요한 Apache 모듈을 활성화합니다.**

**text**

**sudo a2enmod proxy**

**sudo a2enmod proxy\_http**

**Apache 설정 파일 수정:**

**/etc/apache2/sites-available 디렉토리의 해당 사이트 설정 파일(예: 000-default.conf)을 수정합니다.**

**text**

**sudo nano /etc/apache2/sites-available/000-default.conf**

**프록시 설정 추가:**

**VirtualHost 블록 내에 다음 설정을 추가합니다.**

**text**

**<VirtualHost \*:80>**

**# 기존 설정들...**

**ProxyRequests Off**

**ProxyPreserveHost On**

**ProxyVia Full**

**<Proxy \*>**

**Require all granted**

**</Proxy>**

**ProxyPass /api http://localhost:3000**

**ProxyPassReverse /api http://localhost:3000**

**# 기타 설정들...**

**</VirtualHost>**

**여기서 http://localhost:3000은 Node.js 애플리케이션이 실행되는 주소와 포트입니다.**

* **nodemon 사용:**

**nodemon은 파일 변경을 감지하고 자동으로 서버를 재 시작하는 도구입니다.**

**npm install -g nodemon**

**nodemon appAPI.js**

* **vscode 에서 디버그 방법**

**F5 key 누르면 디버그 시작**

**(다른 node 실행되고 있으면 안됨)**

**F10 🡪 한스텝씩 / F11 🡪 함수 안으로**

**F5 🡪 다음 breakpoint 까지**

**launch.json 파일 있어야 하면 설정에 맞게 수정함.**

**3. Mysql + Mybatis 연동**

**설치 - sudo apt install mysql-server**

**1.접속 :sudo mysql**

**2.사용자 생성**

**CREATE USER 'midstar'@'localhost' IDENTIFIED BY 'dlalwjd5';**

**3.권한설정**

**CREATE USER 'midstar'@'%' IDENTIFIED BY 'dlalwjd5';**

**GRANT ALL PRIVILEGES ON \*.\* TO 'midstar'@'%'; --로컬 / 원격 모두 접근가능**

**4. 설정 적용**

**FLUSH PRIVILEGES;**

**5. 시작/종료**

**sudo systemctl start mysql**

**sudo systemctl stop mysql**

**--부팅시 자동 시작**

**sudo systemctl enable mysql**

**-- 부팅시 자동시작 안함**

**sudo systemctl disable mysql**

**sudo systemctl start mysql**

**sudo systemctl enable mysql**

**6.CREATE DATABASE midstardb; database생성**

**db사용 : use midstardb;**

**접속: mysql -u midstar -p**

**7.외부 접속 허용**

**MySQL 설정 파일 편집: sudo nano /etc/mysql/mysql.conf.d/mysqld.cnf**

**bind-address 줄을 찾아 주석 처리하거나 0.0.0.0으로 변경:**

**# bind-address = 127.0.0.1**

**8.Mybatis 연동**

**npm install mybatis-mapper 설치**

**맵퍼 파일 생성**

**mybatisMapper.createMapper(['path/to/your/mapper.xml']);**

* **Mybatis 문법에 맞게 진행**

**User.js 파일 참고**

**9.윈도우용 접속 tool**

**Dbeaver 로 접속**

**4. Redis 설치**

**1. 설치 sudo apt install redis-server**

**2. Redis 서비스 시작:**

**sudo systemctl start redis-server**

**sudo systemctl restart redis-server**

**3. redis 설치 확인:**

**redis-cli ping --> PONG**

**4.설정 수정 – 외부 접속 허용**

**sudo nano /etc/redis/redis.conf**

**bind 0.0.0.0**

**5. 윈도우용 접속 tool - redis-insight**

**https://redis.io/redis-enterprise/redis-insight/**

**requirepass dlalwjd5**

**6. 프로그램 연동**

**User.js 파일 참고**

**5. git 활용 및 git-hup 사용법 – vscode 연동**

**- git 설치 : Git 공식 웹사이트(**[**https://git-scm.com/)에**](https://git-scm.com/)%EC%97%90)**접속합니다**

**웹사이트 하단에서 Windows용 Git 설치 파일을 다운로드합니다**

**다운로드한 설치 파일을 실행합니다**

설치 중 "Adjusting your PATH environment" 단계에서 "Git from the command line and also from 3rd-party software"를 선택합니다.

* **VSCode 설치**

VSCode 공식 웹사이트(https://code.visualstudio.com/)에서 Windows용 설치 파일을

설치 파일을 실행하고 기본 옵션으로 설치 진행.

* **Git–hup 가입**

[**https://github.com/jangminseok-window /**](https://github.com/jangminseok-window%20/) **midstar.jang@gmail/!dl~~가입**

**repositories 선택하여 New로 저장소 생성**

**- vscode – git / git-hup 연결법**

**기본적인 push 방법**

**변경사항을 커밋합니다:**

**git add .**

**git commit -m "커밋 메시지"**

**원격 저장소로 push합니다:**

**git push origin <branch-name>**

**여기서 <branch-name>은 push하려는 브랜치 이름입니다 (예: main, master 등)1.**

**새로운 브랜치를 원격 저장소에 push**

**새로 생성한 로컬 브랜치를 원격 저장소에 push하려면:**

**git push -u origin <new-branch-name>**

**-u 옵션은 로컬 브랜치와 원격 브랜치를 연결합니다1.**

**원격 저장소 추가**

**원격 저장소가 아직 설정되지 않았다면:**

**원격 저장소를 추가합니다:**

**git remote add origin <remote-repository-URL>**

**추가된 원격 저장소를 확인합니다:**

**git remote -v**

**이후 push 명령을 실행합니다1.**

**강제 push (주의 필요)**

**로컬 변경사항을 강제로 push해야 할 경우 (주의: 원격 저장소의 내용을 덮어쓸 수 있음):**

**git push --force origin <branch-name>**

**6. Jenkins 배포 관리**

**1.젠킨스 다운로드**

젠킨스 공식 웹사이트(https://jenkins.io/download/)에서 Windows용 설치 파일(.msi)을 다운로드

**2.설치 파일 실행**:

다운로드한 jenkins.msi 파일을 실행

**Jenkins 실행 및 브라우저에서 접근:**

**설치가 완료되면, Jenkins는 자동으로 실행됩니다.**

**3. Jenkins는 로컬에서 http://localhost:8080 주소로 접근**

**초기 설정 페이지가 나타나면, Administrator password를 입력하라는 메시지가 나옵니다. 이 비밀번호는 Jenkins 설치 중 생성되며, 파일에서 찾을 수 있습니다.**

**비밀번호 파일 위치:**

**C:\Users\TOURSOFT\AppData\Local\Jenkins\.jenkins\secrets\initialAdminPassword**

**id/ pwd : admin/!Dlal~~**

**--- 로컬에서 공용 ip없는 상태에서 jenkins 활용법 ---**

**https://ngrok.com/ 구글 계정으로 연동함**

**webhook사용해보자**

**https://ngrok.com/**

**복구 문구 gmail 에서 찾아볼것**

**D:\ngrok>ngrok config add-authtoken 2moraT9n2a114QOYK7tHExe8bwX\_4QC6pzUv785E2gLnqfCJS**

**Authtoken saved to configuration file: C:\Users\TOURSOFT\AppData\Local/ngrok/ngrok.yml**

**Forwarding https://4e07-112-217-239-226.ngrok-free.app ->** [**http://localhost:8080**](http://localhost:8080)

**jangminseok-window/nodeWeb --> setting --> webhook 설정해야함.**

**jenkins openssh 서버에서 aws 접속하려면**

**ssh -i C:\Users\toursoft\.ssh\ubuntu-micro.pem ubuntu@ec2-3-39-228-155.ap-northeast-2.compute.amazonaws.com**

**파일인데 ppk로 만들어진 privatekey를 puttyken를 이용해서 pem 형식으로 변환해야함**

* **Jenkins nodeWeb 배포 설정 확인하여 설정하면 됨**

**7.NodeJS 성능에 대한 건**

**Node.js의 성능에 대한 걱정은 이해할 만합니다.**

**하지만 실제로 Node.js는 많은 경우에 Java와 비슷하거나 때로는 더 나은 성능을**

**보여줄 수 있습니다.**

1. **JIT 컴파일: Node.js는 V8 엔진을 사용하며, 이는 Just-In-Time (JIT) 컴파일을 수행합니다.**

**이는 JavaScript 코드를 실행 시점에 기계어로 컴파일하여 성능을 크게 향상시킵니다.**

1. **비동기 I/O: Node.js의 비동기, 이벤트 기반 아키텍처는**

**I/O 작업이 많은 애플리케이션에서 특히 효율적입니다.**

**이는 대규모 동시 연결을 처리하는 데 유리합니다.**

1. **싱글 스레드 모델: Node.js의 싱글 스레드 모델은 컨텍스트 스위칭 오버헤드를 줄여 특정 상황에서 멀티 스레드 모델보다 효율적일 수 있습니다.**
2. **메모리 효율성: Node.js는 Java에 비해 일반적으로 메모리 사용량이 적습니다.**
3. **최적화된 성능: 많은 벤치마크에서 Node.js는 Java와 비슷하거나 때로는 더 나은 성능을 보여주었습니다.**

**물론 CPU 집약적인 작업에서는 Java가 더 나은 성능을 보일 수 있습니다.**

**그러나 대부분의 웹 서비스는 I/O 작업이 주를 이루므로,**

**Node.js는 충분히 경쟁력 있는 성능을 제공할 수 있습니다.**

**결론적으로, Node.js의 스크립트 언어 특성이 반드시 성능 저하로 이어지지는 않습니다.**

**오히려 많은 웹 서비스 시나리오에서 Node.js는 뛰어난 성능을 보여줄 수 있습니다.**

1. **EJS을 이용한 화면 구성**

EJS (Embedded JavaScript) 템플릿 엔진을 사용할 때, 일반적으로 파일의 간단한 수정사항이 있을 경우에도 Node.js 서버를 재시작해야 변경 사항이 적용됩니다. 하지만 이 문제를 해결할 수 있는 몇 가지 방법이 있습니다:

Nodemon 사용:

Nodemon은 파일 변경을 감지하고 자동으로 서버를 재시작해주는 도구입니다.

text

npm install -g nodemon

nodemon app.js

캐시 비활성화:

EJS의 캐시를 비활성화하면 매 요청마다 템플릿을 다시 읽어들입니다.

javascript

app.set('view cache', false);

개발 모드에서 캐시 비활성화:

javascript

if (process.env.NODE\_ENV !== 'production') {

app.set('view cache', false);

}

라이브 리로드 도구 사용:

브라우저 자동 새로고침 기능을 제공하는 도구들을 사용할 수 있습니다.

서버 사이드 렌더링 대신 클라이언트 사이드 렌더링 고려:

React, Vue, Angular 등을 사용하면 서버 재시작 없이 UI 업데이트가 가능합니다.

핫 리로딩 구현:

Webpack과 같은 도구를 사용하여 핫 리로딩을 구현할 수 있습니다.

1. **nodeJS 소스 코드 보호 관련 사항**

네, 맞습니다. Node.js 애플리케이션의 소스 코드를 완벽하게 보호하는 것은 현재로서는 매우 어렵습니다. 이는 JavaScript의 인터프리터 언어 특성과 Node.js의 실행 방식 때문입니다. 하지만 이 상황을 다음과 같이 이해하고 대응할 수 있습니다:

현실적인 접근:

완벽한 보호보다는 "충분한" 보호에 초점을 맞춥니다.

대부분의 경우, 코드 난독화와 적절한 보안 조치로 충분할 수 있습니다.

* Webpack과 UglifyJS 사용 / Terser 사용: 구체적으로 확인필요함.

비즈니스 로직 보호:

가장 중요한 비즈니스 로직은 별도의 서비스로 분리하여 API를 통해 접근하게 합니다.

이 방식으로 핵심 알고리즘이나 민감한 로직을 보호할 수 있습니다.

보안의 다층화:

코드 보호, 네트워크 보안, 접근 제어 등 여러 층의 보안을 구현합니다.

단일 방어선이 아닌 여러 방어선을 구축합니다.

지속적인 업데이트와 모니터링:

정기적인 보안 감사와 업데이트를 통해 새로운 취약점에 대응합니다.

비정상적인 접근이나 사용 패턴을 모니터링합니다.

법적, 계약적 보호:

라이선스 및 법적 문서를 통해 지적 재산권을 보호합니다.

직원 및 협력업체와의 계약에 기밀유지 조항을 포함시킵니다.

대안 기술 고려:

특별히 민감한 부분에 대해서는 컴파일 언어로 작성된 마이크로서비스 사용을 고려할 수 있습니다.

클라우드 서비스 활용:

서버리스 함수 등 클라우드 서비스를 활용하여 코드 노출을 최소화할 수 있습니다.

교육과 인식 제고:

개발팀과 운영팀에 보안의 중요성을 교육하고 인식을 높입니다.

결론적으로, 완벽한 보호는 어렵지만 다양한 방법을 조합하여 실질적인 위험을 크게 줄일 수 있습니다. 또한, 소스 코드 보호뿐만 아니라 전반적인 애플리케이션 보안에 초점을 맞추는 것이 중요합니다. 기술의 발전에 따라 더 나은 보호 방법이 계속 개발될 것이므로, 최신 보안 동향을 지속적으로 파악하고 적용하는 것이 좋습니다.