# KSB 인공지능 프레임워크 설치

제공된 KSB 툴박스 docker image를 사용하지 않고, 직접 Host PC에 KSB 인공지능 프레임워크를 설치하기 위해서 아래의 오픈소스 프레임워크들을 설치합니다. 사용자 csle 계정을 생성한 후 아래 프로그램들을 설치합니다.

## [Host PC] csle 사용자 계정(권한 : administrator) 생성하기

Host PC에 기존 Ubuntu 사용자와 설치환경을 분리하기 위해 아래의 명령으로 csle 계정을 추가 생성합니다.

```
sudo adduser csle
    Adding user 'csle' ...
    Adding new group 'csle' (1001) ...
    Adding new user 'csle' (1001) with group 'csle' ...
    Creating home directory '/home/csle' ...
    Copying files from '/etc/skel' ...
    새 UNIX 암호 입력:
    새 UNIX 암호 재입력:
    passwd: password updated successfully
    Changing the user information for csle
    Enter the new value, or press ENTER for the default
    Full Name []:
    Room Number []:
    Work Phone []:
    Home Phone []:
    Other []:
    Is the information correct? [Y/n] Y
```

/etc/sudoers를 열어 csle 계정에 administrator 권한을 추가합니다.
"경고: 읽기 전용 파일을 고치고 있습니다" 경고는 무시하고 저장하고 나옵니다.

```
sudo vi /etc/sudoers
    # User privilege specification
    root    ALL=(ALL:ALL) ALL
    # 맨 아래줄에 내용을 추가합니다.
    #includedir /etc/sudoers.d
    csle    ALL=(ALL) NOPASSWD: ALL
```

# [Host PC] hostname을 csle1으로 변경하기

docker 컨테이너가 호스트 pc의 네트워크를 동일하게 사용하는 host 모드로 동작하기 위해 docker 컨테이너와 호스트 pc의 호스트네임을 동일하게 합니다.

```
sudo vi /etc/hostname
csle1
```

PC를 재부팅하여 csle 계정으로 로그인합니다.

# [Host PC] /etc/hosts 수정하기

KSB 웹툴킷에 쉽게 접근하기 위해 아래와 같이 /etc/hosts 내용을 수정합니다. (host pc의 IP는 10.0.2.15로 가정합니다. 자신의 PC IP에 맞게 설정합니다.)

```
127.0.0.1       localhost
# 아래 주석처리. Hdfs 연동시 문제가 생김.
#127.0.1.1      csle1

# The following lines are desirable for IPv6 capable hosts
::1     ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters

# 아래의 내용 추가
10.0.2.15     csle1 master
```

# [Host PC] java-8-oracle 설치하기

Host PC에 Ubuntu 16.04를 처음 설치한 경우 java 8을 설치합니다. 아래의 내용을 터미널에 복사하여 설정가능합니다.

```
sudo apt-get update && \
sudo apt-get install -y python-software-properties && \
sudo add-apt-repository -y ppa:webupd8team/java && \
sudo apt-get update && \
echo "oracle-java8-installer \
shared/accepted-oracle-license-v1-1 select true" \
| sudo debconf-set-selections && \
sudo apt-get install -y oracle-java8-installer
```

터미널을 다시 시작한 후, 아래의 명령으로 java 설치 여부를 확인합니다.

```
csle@csle1:~$ java -version
java version "1.8.0_181"
Java(TM) SE Runtime Environment (build 1.8.0_181-b13)
Java HotSpot(TM) 64-Bit Server VM (build 25.181-b13, mixed mode)
```

가끔 오라클에서 jdk 버전을 업데이트 하면 아래와 같은 에러가 발생하며 jdk가 설치되지 않는 경우가 있습니다.

```
접속 download.oracle.com (download.oracle.com)|23.35.220.157|:80... 접속됨.
HTTP request sent, awaiting response... 404 Not Found
2018-10-17 11:08:56 ERROR 404: Not Found.

download failed
Oracle JDK 8 is NOT installed.
dpkg: error processing package oracle-java8-installer (--configure):
 설치한 post-installation 스크립트 하위 프로세스가 오류 1번을 리턴했습니다
처리하는데 오류가 발생했습니다:
 oracle-java8-installer
E: Sub-process /usr/bin/dpkg returned an error code (1)
```

해결방법은 오라클 사이트에 방문하여 최신 버전(2018.10.21. 기준 : 8u191)을 jdk-8u191-linux-x64.tar.gz 파일을 다운받습니다.
https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

그리고, 아래의 스크립트에서 다운받은 파일의 위치와 버전 부분만을 수정한 후, 터미널에 복사

하면 직접 설치가능합니다.

```
sudo mkdir /usr/lib/jvm
cd /usr/lib/jvm
sudo tar -xvzf /home/csle/jdk-8u191-linux-x64.tar.gz  # 수정 필요
sudo mv jdk1.8.0_191/ java-8-oracle/  # 수정 필요

sudo truncate -s0 /etc/environment
echo "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:\
/sbin:/bin:/usr/games:/usr/local/games:/usr/lib/jvm/java-8-oracle/bin:\
/usr/lib/jvm/java-8-oracle/db/bin:/usr/lib/jvm/java-8-oracle/jre/bin" |\
sudo tee -a /etc/environment >/dev/null && \
echo "J2SDKDIR=/usr/lib/jvm/java-8-oracle" | \
sudo tee -a /etc/environment >/dev/null && \
echo "J2REDIR=/usr/lib/jvm/java-8-oracle/jre*" | \
sudo tee -a /etc/environment >/dev/null && \
echo "JAVA_HOME=/usr/lib/jvm/java-8-oracle" | \
sudo tee -a /etc/environment >/dev/null && \
echo "DERBY_HOME=/usr/lib/jvm/java-8-oracle/db" | \
sudo tee -a /etc/environment >/dev/null

sudo update-alternatives --install \
"/usr/bin/java" "java" "/usr/lib/jvm/java-8-oracle/bin/java" 0
sudo update-alternatives --install \
"/usr/bin/javac" "javac" "/usr/lib/jvm/java-8-oracle/bin/javac" 0
sudo update-alternatives --set java /usr/lib/jvm/java-8-oracle/bin/java
sudo update-alternatives --set javac /usr/lib/jvm/java-8-oracle/bin/javac
update-alternatives --list java
update-alternatives --list javac
```

# [Host PC] KSB 관련 프로그램 설치하기

Host PC에 아래의 명령으로 KSB 관련 프로그램을 설치합니다. 아래의 내용을 터미널에 복사하여 설정가능합니다.

```
sudo apt-get update && \
sudo apt-get install -y --no-install-recommends apt-utils curl bc jq && \
sudo apt-get install -y ssh locales wget git vim rsync locales \
filezilla python3-pip && \
sudo apt-get install -y net-tools && \
pip3 install kafka-python
```

# [Host PC] SSH port 변경 및 root 로그인 가능하도록 SSHD config 수정하기

아래의 명령을 수행하여 포트 정보를 2243로 수정하고, root 로그인을 허용합니다.

```
sudo sed -ri 's/^Port 22/Port 2243/g' /etc/ssh/sshd_config
sudo sed -ri \
's/^PermitRootLogin prohibit-password/PermitRootLogin yes/g' \
/etc/ssh/sshd_config
```

# [Host PC] SSH config 수정하기 (known_hosts에 호스트 저장 질문을 하지 않도록 설정)

아래의 명령을 수행하여 ssh_config의 기존 내용을 모두 삭제하고 설정을 추가합니다. 아래의 내용을 터미널에 복사하여 설정가능합니다.

```
sudo truncate -s0 /etc/ssh/ssh_config
echo "Host localhost" \
| sudo tee -a /etc/ssh/ssh_config >/dev/null
echo "StrictHostKeyChecking no" \
| sudo tee -a /etc/ssh/ssh_config >/dev/null
echo "Host 0.0.0.0" \
| sudo tee -a /etc/ssh/ssh_config >/dev/null
echo "StrictHostKeyChecking no" \
| sudo tee -a /etc/ssh/ssh_config >/dev/null
echo "Host 127.0.0.1" \
| sudo tee -a /etc/ssh/ssh_config >/dev/null
echo "StrictHostKeyChecking no" \
| sudo tee -a /etc/ssh/ssh_config >/dev/null
echo "Host csle*" \
| sudo tee -a /etc/ssh/ssh_config >/dev/null
echo "StrictHostKeyChecking no" \
| sudo tee -a /etc/ssh/ssh_config >/dev/null
echo "UserKnownHostsFile=/dev/null" \
| sudo tee -a /etc/ssh/ssh_config >/dev/null
echo "Host master" \
| sudo tee -a /etc/ssh/ssh_config >/dev/null
echo "StrictHostKeyChecking no" \
| sudo tee -a /etc/ssh/ssh_config >/dev/null
echo "UserKnownHostsFile=/dev/null" \
| sudo tee -a /etc/ssh/ssh_config >/dev/null
sudo service ssh restart
```

# [Host PC] Hadoop 2.7.3 설치하기

아래 링크를 참고하여 사용자의 PC 혹은 클러스터 환경에 맞게 설치합니다.

참고링크 : https://hadoop.apache.org/docs/r2.7.3/
다운로드 링크 : https://archive.apache.org/dist/hadoop/common/hadoop-2.7.3/

**[KSB 툴박스를 이용한 설치 방법]**
KSB 툴박스를 동작시킨 상태에서 docker 컨테이너로부터 hadoop 설정 파일을 복사하여 설치할 수도 있습니다.
먼저 도커에 접속하는 방법을 참고하여 접속합니다. ( Filezilla를 이용하여 KSB 툴박스 docker 컨테이너에 접속하기 )
도커에 있는 hadoop-2.7.3 폴더를 Host PC의 /home/csle 폴더에 복사합니다.
또한, /home/csle/data 폴더의 hdfs 관련 정보저장 폴더를 /home/csle 폴더에 복사합니다.

그리고, ssh 포트 정보를 'hadoop-env.sh'파일에 추가합니다.

```
vi /home/csle/hadoop-2.7.3/etc/hadoop/hadoop-env.sh
export HADOOP_SSH_OPTS="-p 2243"
```

마지막으로 심볼릭링크을 설정합니다.

```
ln -s /home/csle/hadoop-2.7.3 /home/csle/hadoop
```

# [Host PC] spark-2.3.0-bin-hadoop2.7 설치하기

아래 링크를 참고하여 사용자의 PC 혹은 클러스터 환경에 맞게 설치합니다.

참고링크 : https://spark.apache.org/downloads.html
다운로드 링크 : https://spark.apache.org/downloads.html

**[KSB 툴박스를 이용한 설치 방법]**
KSB 툴박스를 동작시킨 상태에서 docker 컨테이너로부터 spark 설정 파일을 복사하여 설치할 수도 있습니다.
먼저 도커에 접속하는 방법을 참고하여 접속합니다. ( Filezilla를 이용하여 KSB 툴박스 docker 컨테이너에 접속하기 )
도커에 있는 spark-2.3.0-bin-hadoop2.7 폴더를 /home/csle 폴더에 복사합니다.

그리고 심볼릭링크을 설정합니다.

```
ln -s /home/csle/spark-2.3.0-bin-hadoop2.7 /home/csle/spark
```

# [Host PC] hbase 1.2.4(optional) 설치하기

아래 링크를 참고하여 사용자의 PC 혹은 클러스터 환경에 맞게 설치합니다.

참고링크 : https://hbase.apache.org/book.html
다운로드 링크 : https://archive.apache.org/dist/hbase/1.2.4/

**[KSB 툴박스를 이용한 설치 방법]**
KSB 툴박스를 동작시킨 상태에서 docker 컨테이너로부터 hbase 설정 파일을 복사하여 설치할

수도 있습니다.
먼저 도커에 접속하는 방법을 참고하여 접속합니다. ( Filezilla를 이용하여 KSB 툴박스 docker 컨테이너에 접속하기 )
도커에 있는 hbase-1.2.4 폴더를 /home/csle 폴더에 폴더에 복사합니다.
그리고, ssh 포트 정보를 'hbase-env.sh'파일에 추가합니다.

```
vi /home/csle/hbase-1.2.4/conf/hbase-env.sh
export HBASE_SSH_OPTS="-p 2243"
```

그리고 심볼릭링크을 설정합니다.

```
ln -s /home/csle/hbase-1.2.4 /home/csle/hbase
```

# [Host PC] kafka 2.11-0.10.0.1(optional) 설치하기

아래 링크를 참고하여 사용자의 PC 혹은 클러스터 환경에 맞게 설치합니다.

참고링크 : https://kafka.apache.org/
다운로드 링크 : https://kafka.apache.org/downloads

**[KSB 툴박스를 이용한 설치 방법]**
KSB 툴박스를 동작시킨 상태에서 docker 컨테이너로부터 kafka 설정 파일을 복사하여 설치할 수도 있습니다.
먼저 도커에 접속하는 방법을 참고하여 접속합니다. ( Filezilla를 이용하여 KSB 툴박스 docker 컨테이너에 접속하기 )
도커에 있는 kafka_2.11-0.10.0.1 폴더를 /home/csle 폴더에 복사합니다.
그리고 심볼릭링크을 설정합니다.

```
ln -s /home/csle/kafka_2.11-0.10.0.1 /home/csle/kafka
```

# [Host PC] zookeeper 3.4.9(optional) 설치하기

아래 링크를 참고하여 사용자의 PC 혹은 클러스터 환경에 맞게 설치합니다.

참고링크 : https://zookeeper.apache.org/

다운로드 링크 : https://archive.apache.org/dist/zookeeper/zookeeper-3.4.9/

**[KSB 툴박스를 이용한 설치 방법]**
KSB 툴박스를 동작시킨 상태에서 docker 컨테이너로부터 zookeeper 설정 파일을 복사하여 설치할 수도 있습니다.
먼저 도커에 접속하는 방법을 참고하여 접속합니다. ( Filezilla를 이용하여 KSB 툴박스 docker 컨테이너에 접속하기 )
도커에 있는 zookeeper-3.4.9 폴더를 /home/csle 폴더에 복사합니다.
그리고 심볼릭링크을 설정합니다.

```
ln -s /home/csle/zookeeper-3.4.9 /home/csle/zookeeper
```

# [Host PC] bashrc 설정 추가하기

csle 계정의 bashrc 파일을 업데이트 합니다.

```
echo "export JAVA_HOME=/usr/lib/jvm/java-8-oracle" >> ~/.bashrc
echo "export PATH=\$PATH:\$JAVA_HOME/bin" >> ~/.bashrc
echo "export PATH=/opt/apache-maven-3.3.9/bin:\$PATH" >> ~/.bashrc
echo "export HADOOP_HOME=/home/csle/hadoop-2.7.3" >> ~/.bashrc
echo "export HADOOP_PREFIX=\$HADOOP_HOME" >> ~/.bashrc
echo "export PATH=\$PATH:\$HADOOP_PREFIX/bin" >> ~/.bashrc
echo "export PATH=\$PATH:\$HADOOP_PREFIX/sbin" >> ~/.bashrc
echo "export HADOOP_CONF_DIR=\$HADOOP_HOME/etc/hadoop" >> ~/.bashrc
echo "export YARN_CONF_DIR=\$HADOOP_HOME/etc/hadoop" >> ~/.bashrc
echo "export HADOOP_MAPRED_HOME=\${HADOOP_PREFIX}" >> ~/.bashrc
echo "export HADOOP_COMMON_HOME=\${HADOOP_PREFIX}" >> ~/.bashrc
echo "export HADOOP_HDFS_HOME=\${HADOOP_PREFIX}" >> ~/.bashrc
echo "export YARN_HOME=\${HADOOP_PREFIX}" >> ~/.bashrc
echo "export HADOOP_COMMON_LIB_NATIVE_DIR=\${YARN_HOME}/lib/native">>\
~/.bashrc
echo \
"export HADOOP_OPTS="-Djava.library.path=\$YARN_HOME/lib/native"" >>\
~/.bashrc
echo "export ZOOKEEPER_HOME=/home/csle/zookeeper-3.4.9" >> ~/.bashrc
echo "export ZOOKEEPER_PREFIX=\$ZOOKEEPER_HOME" >> ~/.bashrc
echo "export ZOO_LOG_DIR=\$ZOOKEEPER_HOME/logs" >> ~/.bashrc
echo "export PATH=\$PATH:\$ZOOKEEPER_HOME/bin" >> ~/.bashrc
echo "export SPARK_HOME=/home/csle/spark-2.3.0-bin-hadoop2.7" >>\
~/.bashrc
echo "export PATH=\$PATH:\$SPARK_HOME/bin" >> ~/.bashrc
echo "export HBASE_HOME=/home/csle/hbase-1.2.4" >> ~/.bashrc
echo "export PATH=\$PATH:\$HBASE_HOME/bin" >> ~/.bashrc
echo "export KAFKA_HOME=/home/csle/kafka_2.11-0.10.0.1" >> ~/.bashrc
echo "export PATH=\$PATH:\$KAFKA_HOME/bin" >> ~/.bashrc
echo "export \
MONGODB_HOME=/home/csle/mongodb-linux-x86_64-ubuntu1404-3.4.1" >>\
~/.bashrc
echo "export PATH=\$PATH:\$MONGODB_HOME/bin" >> ~/.bashrc
echo "export LANGUAGE=ko_KR.UTF-8" >> ~/.bashrc
echo "export LANG=ko_KR.UTF-8" >> ~/.bashrc
echo "export PATH=/home/csle/anaconda3/bin:\$PATH" >> ~/.bashrc
echo "export ACTIVATOR_HOME=/home/csle/activator-dist-1.3.12" >>\
~/.bashrc
echo "export PATH=\$PATH:\$ACTIVATOR_HOME/bin" >> ~/.bashrc
echo "export PATH=\$PATH:/home/csle/KBE" >> ~/.bashrc
echo "export KSB_HOME=/home/csle/ksb-csle" >> ~/.bashrc
echo "export PYTHONPATH=/home/csle/ksb-csle/pyML/:\$PYTHONPATH" >>\
~/.bashrc
echo "export PYTHONPATH=./:/home/csle/ksb-csle/ksblib:\$PYTHONPATH" >>\
~/.bashrc
echo "export PYSPARK_PYTHON=/home/csle/anaconda3/bin/python" >>\
```

```
~/.bashrc
echo "export PYSPARK_DRIVER_PYTHON=/home/csle/anaconda3/bin/python" >>\
~/.bashrc
echo "export PATH=\$PATH:/home/csle/anaconda3/bin/" >> ~/.bashrc
source ~/.bashrc
```

# [Host PC] Anaconda3 설치하기

가상환경을 이용하여 tensorflow cpu버전을 설치하기 위해 anaconda3를 홈페이지
(https://repo.anaconda.com/archive/)에서
"Anaconda3-4.2.0-Linux-x86_64.sh"버전을 다운받습니다.
아래의 명령을 이용하여 csle 홈 폴더에 anaconda3를 설치합니다.

```
bash Anaconda3-4.2.0-Linux-x86_64.sh -b -p /home/csle/anaconda3
```

# [Host PC] 기타 라이브러리 설치하기

pyspark에서 사용하는 python에 관련 라이브러리를 설치합니다.

```
conda update -y conda  && \
conda install -c conda-forge protobuf  && \
pip install py4j  && \
pip install --upgrade pip  && \
pip install hdfs  && \
conda install -y hdfs3 libhdfs3 -c conda-forge
```

# [Host PC] tensorflow 설치하기 (cpu버전)

tensorflow 설치를 위한 가상환경을 설치합니다.

```
conda update -y conda && \
conda create -y -n tensorflow python=3.5
```

```
source activate tensorflow
pip install "tensorflow==1.6.0"  && \
pip install setuptools  && \
pip install matplotlib  && \
pip install pandas  && \
pip install scikit-optimize  && \
pip install hdfs  && \
conda install -y hdfs3 libhdfs3 -c conda-forge
source deactivate
```

# [Host PC] KSB 툴박스 파일 설치하기

KSB 프레임워크를 호스트 서버에 직접 설치하는 경우에도 ksb_toolbox_v1.tar.gz 파일이 필요합니다.
다운로드 페이지로부터 KSB 툴박스 파일을 다운로드합니다.

- ksb_toolbox_v1.tar.gz : KSB 툴박스 파일

csle 사용자 계정의 home에 ksb_toolbox_v1.tar.gz 압축을 해제합니다. 결과적으로 /home/csle/ksb-csle 폴더가 생성됩니다.

```
csle@csle1:~$ tar zxvf ksb_toolbox_v1.tar.gz -C /home/csle/
```

/home/csle/ksb-csle 폴더는 다음의 하위 폴더들로 구성됩니다.

- /bin : KSB 프레임워크 shell 스크립트 파일 저장 폴더
- /components : tensorflow train 예제 프로그램 저장 폴더
- /conf : KSB 프레임워크 환경설정 파일 저장 폴더
- /docker : KSB 툴박스 docker 컨테이너 실행 스크립트 저장 폴더
- /examples : CLI(Command Line Interface)를 통해 워크플로우 생성 및 submit 할 수 있는 프로젝트 폴더
- /jars : KSB 프레임워크 1.0 버전의 jar 파일 저장 폴더
- /ksblib : python 전처리 라이브러리 폴더
- /kubernetes : 쿠버네티스 환경설정 폴더
- /logs : log 저장 폴더
- /pyML : autoML python 프로젝트 폴더

- /tools: 예제 테스트를 위한 프로그램(jmeter, hadoop, kafka, maven, .ssh)

# [Host PC] ksb-csle config 파일 설정하기

/home/csle/ksb-csle/conf/ksb.conf 파일을 사용자 PC에 맞게 수정합니다.

```
csle {
    user {
        home = "/home/csle"      # 사용자 home 디렉토리를 지정합니다.
    }
    home = "/home/csle/ksb-csle" # KSB home 디렉토리를 지정합니다.
    submit.cli = "true"
    scheduler.initialization = "false"
}

servers {
    gateway {
        baseUrl = "http://KSB_GATEWAY:SERVICEPORT"
    }
    kubernetes {
        enable = "false"
        baseUrl = "http://SERVICENAME.ksb.local:30100"
        imgRepositoryUrl = "docker.io"
        kubernetes_yaml_path = "/kubernetes"
        masterIp = "csle1" # Kubernetes 마스터 도메인 이름 혹은 IP를 지정합니다.
        masterPort = "2243" # Kubernetes SSH 포트 번호를 지정합니다.
        nodesIp = "csle2, csle3"
        # Kubernetes 노드들의 도메인 이름 혹은 IP들을 지정합니다.
        user = "csle"       # Kubernetes 서버의 사용자 ID를 지정합니다.
        externalKafka {
            brokerHostName = "csle1"
            # 외부 Kafka broker서버 도메인 이름을 지정합니다.
            brokerHostIp = "192.168.0.5"
            # 외부 Kafka broker서버 IP을 지정합니다.
          }
    }
```

```
    tensorflow {
        enable = "true"
        python_path = "/anaconda3/envs/tensorflow/bin/python"
        # 사용자 PC의 tensorflow python 위치를 지정합니다.
        python_code_project_path = "/analysis"
    }
    pyspark {
        python_path = "/anaconda3/bin/python"
        # 사용자 PC의 python 위치를 지정합니다.
        python_code_project_path = "/pyML"
    }
    spark {
        home = "/spark"  # 사용자 PC의 spark home 위치를 지정합니다.
        bin = "/spark/bin/spark-submit"
        master = "local[*]" # spark master 모드를 지정합니다.
        deploy = "client"   # spark deploy 모드를 지정합니다.
        logging = "true"
        autoIncreaseModelPath = "true"
    }
    yarn {
        home = "/hadoop/bin" # 사용자 PC의 yarn 위치를 지정합니다.
    }
    postgres {
        db = "csledb"
        uri = "jdbc:postgresql://localhost:5432/csledb"
        user = "csle"
        password = "csle1234"
    }
    webtoolkit {
        db = "ui"
        uri = "jdbc:postgresql://localhost:5432/uidb"
        user = "csle"
        password = "csle1234"
    }
    hadoop {
        home = "/hadoop/" # 사용자 PC의 hadoop 위치를 지정합니다.
        master = "csle1"  # 사용자 PC의 hadoop master 이름을 지정합니다.
        port = "9000"     # 사용자 PC의 hadoop master port를 지정합니다.
        hdfs {
            activated = "true"
            baseDir = "/user/"
            modelPath = "/model"
            datasetPath = "/dataset"
        }
        webhdfs {
            port = "50070"
```

```
            baseDir = "/webhdfs/v1"
        }
    }
    hbase.zookeeper.quorum = "localhost"
    kbe {
        serverIp = "localhost"
        serverPort = "9876"
        serverType = "jena"
    }
}
```

# [Host PC] startService.sh, stopService.sh 생성하기

/home/csle에 startService.sh, stopService.sh 파일을 생성합니다.

```
cat <<EOT >> ~/startService.sh
#!/bin/bash
sudo service ssh restart
sudo service postgresql restart

bash /home/csle/zookeeper-3.4.9/bin/zkServer.sh start
ssh csle@master -p 2243 "cd /home/csle/zookeeper-3.4.9/bin; \
./zkServer.sh start"
ssh csle@csle1  -p 2243 "cd /home/csle/zookeeper-3.4.9/bin; \
./zkServer.sh start"
start-dfs.sh
start-yarn.sh
start-hbase.sh
/home/csle/kafka/bin/kafka-server-start.sh \
/home/csle/kafka/config/server.properties &
/home/csle/ui_of_csle/apache-tomcat-7.0.81/bin/catalina.sh start &
/home/csle/start-mongo.sh &
sleep 5
cd /home/csle/ksb-csle/bin
/home/csle/ksb-csle/bin/startKnowledge_service.sh localhost 9876
EOT
chmod +x /home/csle/startService.sh
cat <<EOT >> ~/stopService.sh
#!/bin/bash
stop-hbase.sh
bash /home/csle/kafka/bin/kafka-server-stop.sh
stop-yarn.sh
stop-dfs.sh
bash /home/csle/zookeeper-3.4.9/bin/zkServer.sh stop
/home/csle/ui_of_csle/apache-tomcat-7.0.81/bin/catalina.sh stop
/home/csle/stop-mongo.sh
cd /home/csle/ksb-csle/bin
/home/csle/ksb-csle/bin/stopKnowledge_service.sh
sudo service postgresql stop
EOT
chmod +x /home/csle/stopService.sh
```

# [Host PC] KSB 툴박스 docker 이미지내 SSH 키를 host pc에 복사하기

KSB 툴박스 docker 컨테이너의 인증키를 공유하기 위해 ksb-csle/tools 폴더에 있는 .ssh 폴더를 csle 홈에 카피합니다.

```
cp -r ~/ksb-csle/tools/.ssh/ /home/csle/
```

# [Host PC] SSH 재시작하기

```
sudo service ssh restart
```

# [Host PC] KSB 웹툴킷을 사용하기 위한 사용자 계정 추가하기

KSB 웹툴킷에 저장된 다양한 예제를 사용하기 위해서는 웹툴킷 사용자(id: ksbuser@etri.re.kr)에 대한 Ubuntu 및 Hadoop hdfs 계정 추가가 필요합니다.

## Ubuntu 사용자 계정 추가하기

```
csle$csle1:  sudo adduser ksbuser_etri_re_kr

[sudo] password for csle:
'ksbuser_etri_re_kr' 사용자를 추가 중...
새 그룹 'ksbuser_etri_re_kr' (1011) 추가 ...
새 사용자 'ksbuser_etri_re_kr' (1011) 을(를)
그룹 'ksbuser_etri_re_kr' (으)로 추가 ...
'/home/ksbuser_etri_re_kr' 홈 디렉터리를 생성하는 중...
'/etc/skel'에서 파일들을 복사하는 중...

새 UNIX 암호 입력: # ksbuser_etri_re_kr를 입력하세요
새 UNIX 암호 재입력: # ksbuser_etri_re_kr를 입력하세요
passwd: 암호를 성공적으로 업데이트했습니다

새로운 값을 넣거나, 기본값을 원하시면 엔터를 치세요
이름 []:
방 번호 []:
직장 전화번호 []:
집 전화번호 []:
기타 []:
정보가 올바릅니까? [Y/n] Y
```

## sudo 권한 부여하기

```
csle$csle1: sudo usermod -aG sudo ksbuser_etri_re_kr
```

## 계정 추가 확인하기

```
csle@csle1:~$ su - ksbuser_etri_re_kr

ksbuser_etri_re_kr@csle1:~$: sudo ll
합계 32
drwxr-xr-x  2 ksbuser_etri_re_kr ksbuser_etri_re_kr 4096  6월 30 08:37
 ./
drwxr-xr-x 14 root               root               4096  6월 30 08:36
 ../
-rw-r--r--  1 ksbuser_etri_re_kr ksbuser_etri_re_kr  220  6월 30 08:36
 .bash_logout
-rw-r--r--  1 ksbuser_etri_re_kr ksbuser_etri_re_kr 3771  6월 30 08:36
 .bashrc
-rw-r--r--  1 ksbuser_etri_re_kr ksbuser_etri_re_kr  655  6월 30 08:36
 .profile
-rw-r--r--  1 ksbuser_etri_re_kr ksbuser_etri_re_kr    0  6월 30 08:37
 .sudo_as_admin_successful
-rw-r--r--  1 ksbuser_etri_re_kr ksbuser_etri_re_kr 8980  6월 30 08:36
 examples.desktop
```

## Hadoop Hdfs 사용자 계정 추가하기

사용자 환경에 맞게 설치되어 있는 Hadoop Hdfs를 가동한 후, 아래 명령을 이용하여 hdfs에 ksbuser_etri_re_kr 사용자 폴더를 추가합니다.
만약 Hadoop 2.7.3 설치하기에서 도커 컨테이너로부터 /home/csle/data 폴더를 복사한 경우, 이미 사용자 폴더가 생성된 hdfs을 복사한 상태이므로 이 절차를 생략합니다.

```
su - csle
hdfs dfs -mkdir -p /user/ksbuser_etri_re_kr/dataset
hdfs dfs -mkdir -p /user/ksbuser_etri_re_kr/model

hdfs dfs -chown ksbuser_etri_re_kr:supergroup  /user/ksbuser_etri_re_kr/
hdfs dfs -chown ksbuser_etri_re_kr:supergroup \
/user/ksbuser_etri_re_kr/dataset
hdfs dfs -chown ksbuser_etri_re_kr:supergroup \
/user/ksbuser_etri_re_kr/model
```

# [Host PC] KSB 웹툴킷 설치하기

KSB 웹툴킷 설치하기 페이지를 통해 KSB 웹툴킷을 설치합니다.

# [Host PC] KSB 지식베이스 엔진 설치하기

KSB 지식베이스엔진(KSB-Knowledge Engine)을 사용하기 위한 절차 및 예제를 소개합니다.

## Play framework Activator 설치하기

KSB 지식베이스 엔진은 Play framework 기반 엔진이며, 이를 실행하기 위해서는 Play framework Activator를 사전에 설치해야 합니다.

```
# Play framework Activator 다운로드
wget https://downloads.typesafe.com/typesafe-activator/1.3.12/\
typesafe-activator-1.3.12.zip

# 압축해제
unzip typesafe-activator-1.3.12.zip

# Activator bin 경로를 환경변수에 등록
export PATH="/home/csle/activator-dist-1.3.12/bin:$PATH"
```

## KSB 지식베이스 엔진 시작 및 종료 하기

KSB 지식베이스 엔진을 시작하려면, 해당폴더 (/home/csle/ksb-csle/bin)로 이동하여, startknowledge_service.sh 스크립트를 ip address와 port 번호와 함께 실행합니다.

```
# KSB 지식베이스 실행 스크립트가 있는 경로로 이동
cd /home/csle/ksb-csle/bin
# KSB 지식베이스 엔진 시작
#(./startknowledge_service.sh ip_address port_number)
./startknowledge_service.sh 127.0.0.1 9876
```

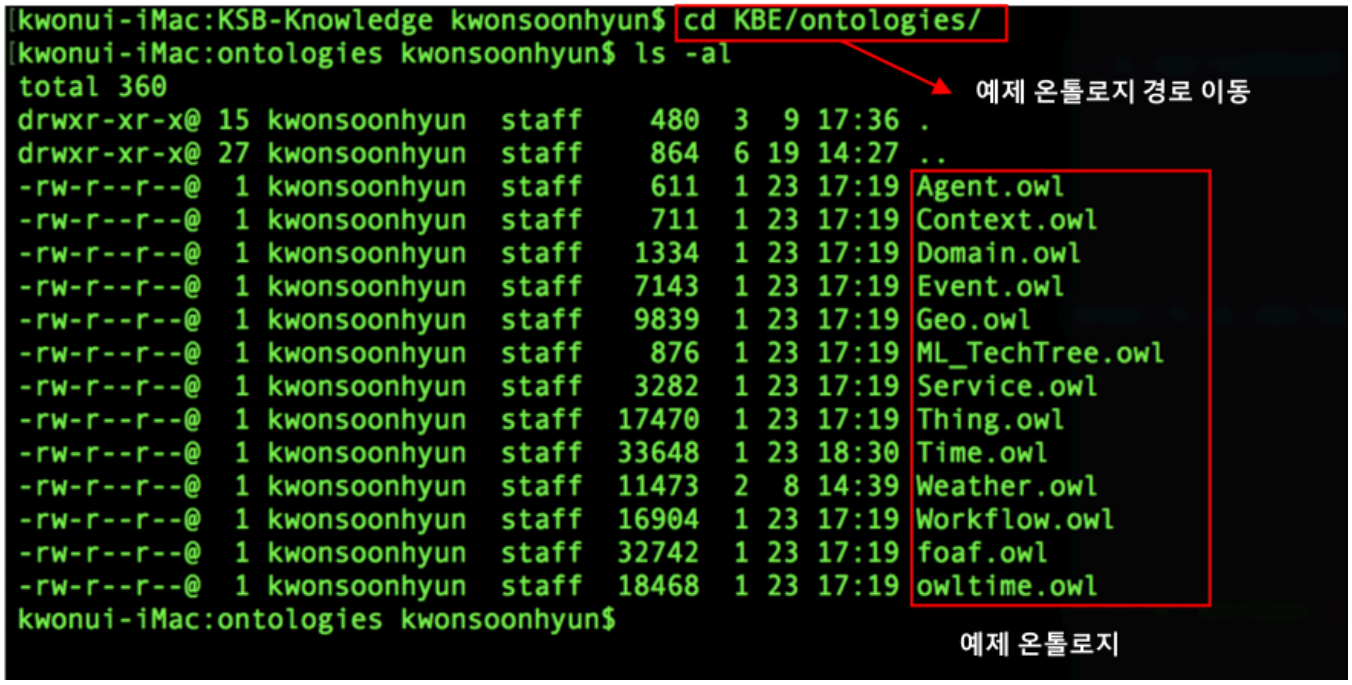
KSB 지식베이스 엔진을 종료하려면, 해당폴더 (/home/csle/ksb-csle/bin)로 이동하여, stopKnowledge_service.sh 스크립트를 실행합니다.

```
# KSB 지식베이스 엔진 종료
./stopKnowledge_service.sh
```

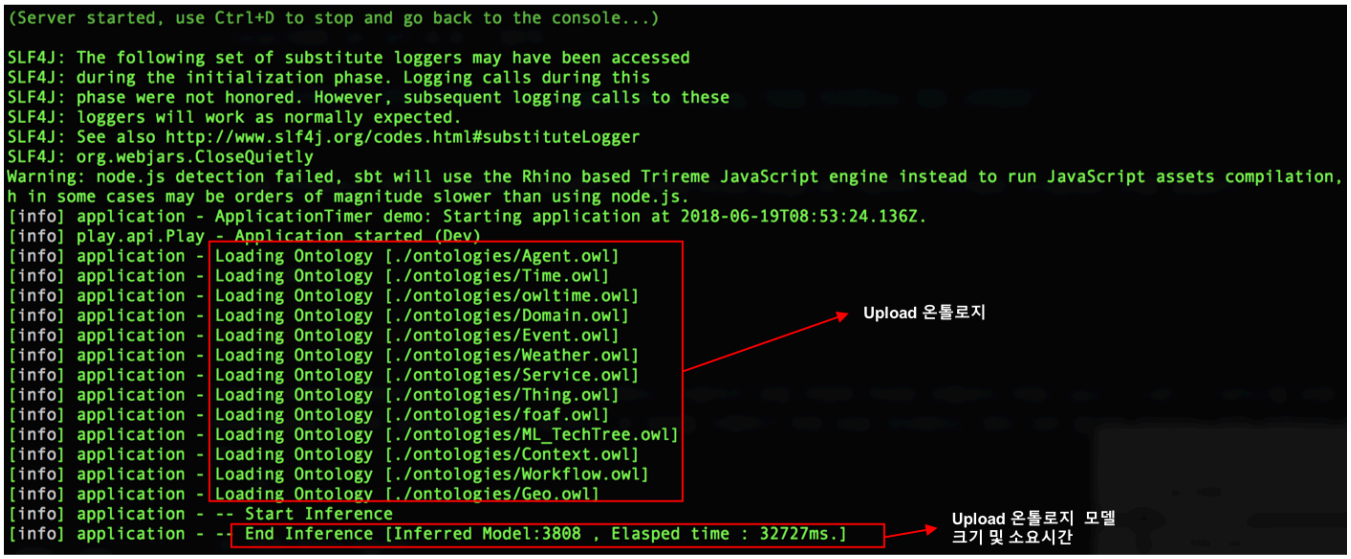
## 온톨로지 파일 Upload 하기

KSB 지식베이스 엔진이 Start 되어 있는 상황에서, 사용자가 로컬 파일에 작성된 온톨로지 파일(.owl, .rdf, .rdfs)을 KSB 지식베이스 엔진에 Upload 하는 기능을 설명합니다. 본 예제에서는 KSB 지식베이스 엔진 내부 폴더에 제공하는 공통 온톨로지 파일을 이용합니다.

```
# 예제 수행 온톨로지 확인을 위해 해당 경로로 이동
cd /home/csle/ksb-knowledge/KBE/ontologies
```

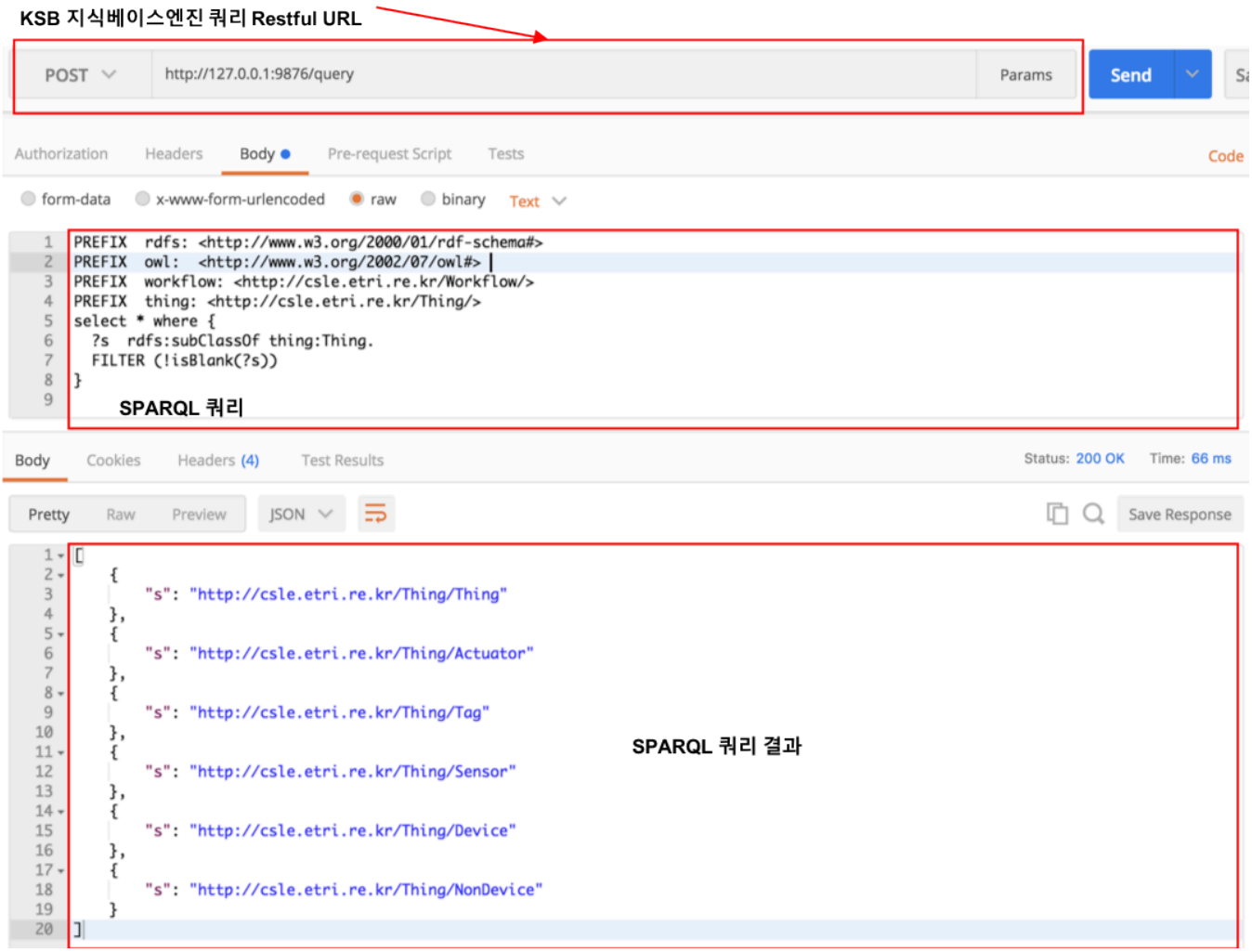



```
# Curl 명령어 restful request 수행 (url: /loadOntologies, parameter: 온톨로지 경로)
curl -X POST http://127.0.0.1:9876/loadOntologies -H 'Cache-Control: \
no-cache' -H 'Content-Type: text/plain' -d ./ontologies
```




KSB 지식베이스 엔진에 Upload된 온톨로지는 KSB 지식베이스 엔진이 제공하는 쿼리 Restful API( `http://127.0.0.1:9876/query` )를 통해 확인할 수 있습니다. 이를 위해 Restful Client(본 예제에서는 Postman)를 실행하고, URL과 질의 쿼리 내용을 아래와 같이 입력하여 실행합니다.

```
# 본 쿼리는 SPAQRL 1.1의 형태의 질의이며,
# 이는 온톨로지 thing:Thing 클래스의 서브 클래스를 검색하기 위한 쿼리입니다.
PREFIX  rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX  owl:  <http://www.w3.org/2002/07/owl#>
PREFIX  workflow: <http://csle.etri.re.kr/Workflow/>
PREFIX  thing: <http://csle.etri.re.kr/Thing/>
select * where {
  ?s  rdfs:subClassOf thing:Thing.
  FILTER (!isBlank(?s))
}
```




# IoE 사물 등록

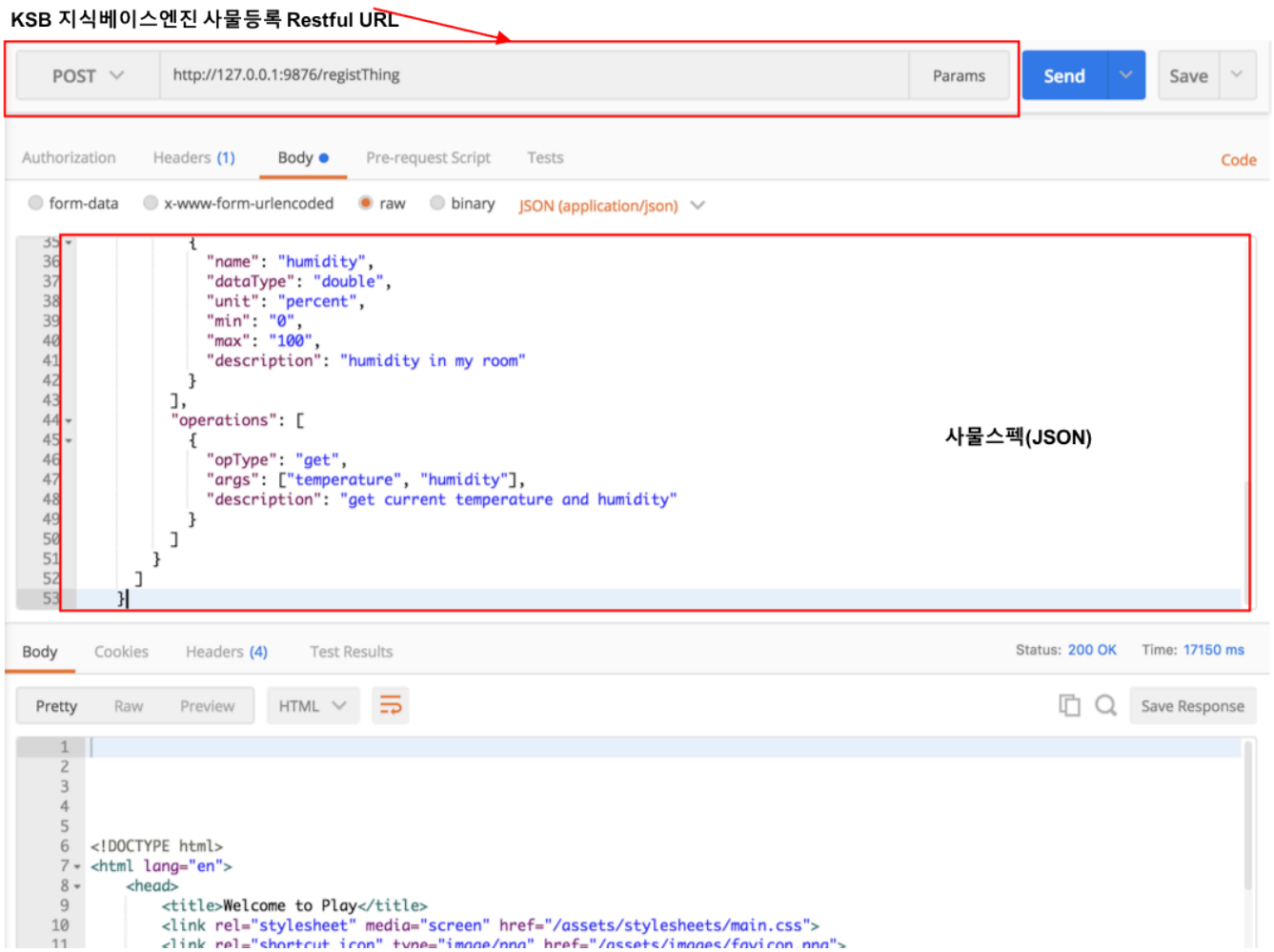
KSB 지식베이스 엔진은 IoE 정보를 지식화하여 정보의 연계, 추론을 통한 지식정보를 사용자에게 제공하는 기능이 있습니다. IoE 표준(oneM2M)의 사물 스펙을 지식화하는 기능을 제공합니다.

본 예제에서는 Json 포맷의 사물 스펙을 사용합니다.

```
# Json 포맷의 사물 스펙(예제로 사용한 IoE 사물의 id는 a1234이며,
# indoor temperature and humidity 센서입니다.)
{
      "version": "1.0",
      "id": "a1234",
      "registrationTime": "20170102112233",
      "properties": {
        "name": "indoor_env_1",
        "owner": "foo@gmail.com",
        "description": "indoor temperature and humidity",
        "gid": "global_unique_id_1",
        "model": "modelA",
        "coordinates": {
          "latitude": 33.3,
          "longitude": 122.2,
          "altitude": 0
        },
        "location": "seoul korea",
        "users": ["bar@gmail.com"],
        "tag": ["my room", "temperature", "humidity"],
        "madeBy": "companyA",
        "hostedBy": "companyB"
      },
      "resources": [
        {
          "name": "sensors",
          "description": "temperature and humidity in my room",
          "attributes": [
            {
              "name": "temperature",
              "dataType": "double",
              "unit": "celsius",
              "min": "-80",
              "max": "80",
              "description": "temperature in my room"
            },
            {
              "name": "humidity",
              "dataType": "double",
              "unit": "percent",
              "min": "0",
              "max": "100",
              "description": "humidity in my room"
            }
          ],
          "operations": [
```

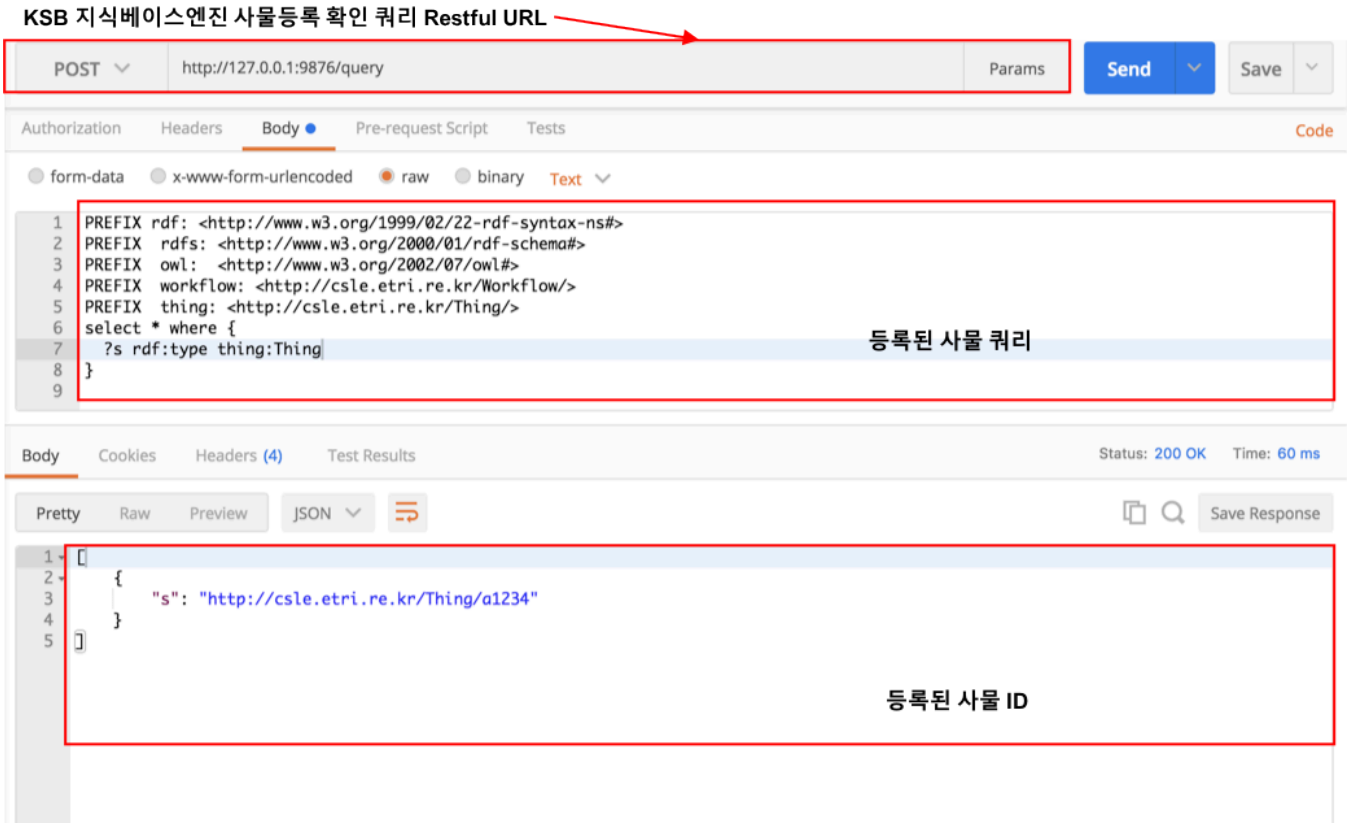
```
          {
            "opType": "get",
            "args": ["temperature", "humidity"],
            "description": "get current temperature and humidity"
          }
        ]
      }
    ]
  }
```

KSB 지식베이스 엔진이 제공하는 사물등록 Restful API ( `http://127.0.0.1:9876/registThing` )를 통해 KSB 지식베이스 엔진에 사물 스펙정보를 등록할 수 있습니다.




다음과 같은 쿼리 Restful API을 수행하여 결과를 확인합니다.

```
# 본 쿼리는 SPAQRL 1.1의 형태의 질의이며,
# 이는 온톨로지 thing:Thing 클래스의 인스턴스를 검색하기 위한 쿼리입니다.
PREFIX  rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX  owl:  <http://www.w3.org/2002/07/owl#>
PREFIX  workflow: <http://csle.etri.re.kr/Workflow/>
PREFIX  thing: <http://csle.etri.re.kr/Thing/>
select * where {
  ?s rdf:type thing:Thing.
}
```




# IoE 사물 센서데이터를 통해 지식추론하기

KSB 지식베이스 엔진은 IoE 디바이스의 센서데이터를 이용하여, 상황을 추론할 수 있는 기능을 제공합니다.
본 예제의 시나리오는 앞에서 등록된 온도센서(thing id : a1234)로부터 온도가 28도 이상이면 더운 상황(HotContext)을 추론하고,
추론된 정보를 통해 Hue(thing id : S_HUE_light_3)를 파란색으로 켜는 동작을 하는 간단한 시나리오 입니다.

이를 위해 Hue의 제어 명령을 등록합니다. (Restful API:

`http://127.0.0.1:9876/addDeviceControl`)

```
# Json 포맷의 사물제어 스펙(id S_HUE_light_3의 Hue)
[
   {"id":"hue_turnon_Blue",
     "thingId":"S_HUE_light_3",
     "resourceId":"controller",
     "operationId":"setState",
     "controlContents":[
        {"name":"on","value":"true"},
        {"name":"bri","value":"254"},
        {"name":"hue","value":"46920"},
        {"name":"sat","value":"254"}
       ]
   },
  {"id":"hue_turnon_Red",
   "thingId":"S_HUE_light_3",
   "resourceId":"controller",
   "operationId":"setState",
   "controlContents":[
      {"name":"on","value":"true"},
      {"name":"bri","value":"254"},
      {"name":"hue","value":"65535"},
      {"name":"sat","value":"254"}
      ]
  },
  {"id":"hue_turnon_Green",
    "thingId":"S_HUE_light_3",
    "resourceId":"controller",
    "operationId":"setState",
    "controlContents":[
       {"name":"on","value":"true"},
       {"name":"bri","value":"254"},
       {"name":"hue","value":"25500"},
       {"name":"sat","value":"254"}
      ]
   }
]
```

추론 규칙을 적용하기 위해, KSB-Knowledge/KBE/rules 폴더의 "KBE.rues" 파일을 열고 아래와 같이 추론 규칙을 추가합니다.

```
# 온도센서의 센싱값이 28이상이면 이벤트 Temp_HOT을 생성합니다.
[domain_temperature_hot:
         (?e event:hasEventValue ?v),
         (?v thing:hasType ?t),
         (?t thing:attributeName ?name),
         equal(?name, "temperature"),
         (?v thing:hasValue ?value),
         ge(?value, 28)
     ->
    (?e event:hasEventStatus domain:Temp_HOT)
 ]
 # 이벤트 Temp_HOT이면 더운 상황(HotContext)을 생성합니다.
 [HotContext_context:
    (?e event:hasEventStatus domain:Temp_HOT)
    -> (?e rdf:type domain:HotContext)
 ]
 # 더운 상황(HotContext)이면 Hue를 파란색으로 켭니다.
 [HotContext_service:
           (?e rdf:type domain:HotContext)
     ->
    (?e service:derives domain:hue_turnon_Blue)
 ]
```

본 예제를 실행하기 위해 온도센서(thing id: a1234)의 값을 32.6으로 설정하고, Restful API( `http://127.0.0.1:9876/recommendDeviceControl` )를 실행합니다.

```
# 온도센서(thingId : a1234)의 센싱값이 32.6으로 설정합니다.
{
      "thingId": "a1234",
      "resourceId": "sensors",
      "time": "20170330103347",
      "values": [
         {
            "name": "temperature",
            "value": "32.6"
         }
       ]
    }
```

Restful API를 실행한 결과는 다음과 같습니다.

```
# 상황(context)가 "HotContext"이고, 상황에 따른 서비스(service)가
# "hue_turnon_Blue"이며 그것에 해당하는 사물 제어정보를 리턴합니다.
[
    {
        "thingId": "S_HUE_light_3",
        "resourceId": "controller",
        "context": [
            "HotContext"
        ],
        "service": "hue_turnon_Blue",
        "controls": {
            "sat": "254",
            "hue": "46920",
            "bri": "254",
            "on": "true"
        }
    }
]
```

**KSB 지식베이스엔진 추론을 통한 사물제어**