CISE Sim

CISE Sim is an application capable to send and receive messages conform to the CISE Protocol. CISE Sim is able to simulate the behavior both of a CISE Node and of a CISE Adaptor. It is able to: - send CISE messages using a template as a base xml message and automatically creating XML fields like messageId, correlationId, creation date and signature needed for the message to be accepted by the receiving endpoint. - receive messages answering with synchronous acknowledgements. The CISE sync ack are supporting the success case and a subset of the errors that can be reported by a CISE node (i.e. access right matrix errors are not supported)

Requirements

The application must be run on a GNU/Linux operative system and requires to have installed the following software: - Java 1.8

How to run CISE Sim

Tar.gz archive distribution

CISE Sim is distributed in a tar.gz archive, therefore it needs to be unpacked in a folder by launching the following commands from the UNIX shell:

```
...$ mkdir -p /my/installation/path
...$ tar -xvzf cise-sim-*.tar.gz -C /my/installation/path --strip-components=1
...$ cd /my/installation/path
```

After unpacking the archive, you can use the `sim` shell script to manage the application.

Application lifecycle

Start in foreground

To run the application in foreground type:

```
...$ ./sim run
```

Start in background

To run the application in background with the command:

```
...$ ./sim start
```

The start command uses internally the UNIX nohup command to launch the application that it is possible to safely close the UNIX shell that started the simulator.

The application sever will start showing a banner and some information in the log file which is located in the `logs` directory. You can see the running logs by typing:

```
...$ tail -f logs/sim_stdout.log
```

Stop from the background

To stop a CISE Sim previously started in background just launch the `stop` command.

```
...$ ./sim stop
```

Other sim commands

Launching `sim` without parameters displays an help in console output:

```
...$ cd /cisesim/installation/path
...$ ./sim
Usage: sim COMMAND
sim server lifecycle manager (starting, stopping, debugging).
COMMAND
    start       starts the simulator in a detached shell using nohup command.
    run         starts the simulator in foreground.
    stop        stops the simulator running in background.
    restart     restart the simulator running in background.
    debug-start starts the simulator in a detached shell launching the application
                in debug mode (port 9999).
    debug-run   starts the simulator in foreground launching the application
                in debug mode (port 9999).
    status      show the current status the simulator (started or stopped).
```

These commands can be used to check the application status and to run the application in debug mode.

Docker distribution

It also exists a docker distribution of cise-sim: **ec-jrc/cise-sim:latest**
From now on this documentation suppose that the image is in the host Docker image repository

Configuration persistence

To obtain the configuration persistence it's necessary to create in the host a set of directories, put them in a preferred location (for example /srv/cise-sim-volumes).
Those directories will map the following cise directories:
- /srv/cise-simulator/**conf** - /srv/cise-simulator/**logs** - /srv/cise-simulator/**msghistory**

```
mkdir /srv/cise-sim-volumes/conf
mkdir /srv/cise-sim-volumes/logs
mkdir /srv/cise-sim-volumes/msghistory
```

Put in the directory **conf** the configuration files.
logs and **msghistory** are the location where the cise-simm will put the output for logs and messages managed.

Launch docker container

To create and run the container we suggest using a simple docker compose file, where map the server ports and the volumes. For example :

```
version: '3.3'
services:
  cise-sim:
    image: jrc-ispra/cise-sim:latest
    volumes:
      - /srv/cise-sim-volumes/conf:/srv/cise-simulator/conf
      - /srv/cise-sim-volumes/conf:/srv/cise-simulator/logs
      - /srv/cise-sim-volumes/conf:/srv/cise-simulator/msghistory
    ports:
      - 8280:8080
      - 8281:8081
```

Be warned that in this example the configurable ports and directory are still the default.

Accessing the CISE Sim

It is possible to interact with the CISE Sim through a user interface. To access the CISE Sim web interface just launch a browser and go to the url: `http://localhost:8080/` .

When the simulator application is running it provides also and endpoint where different counterparts (CISE Nodes or CISE Adaptors be tested) can send their messages to.

The following endpoints are available: - REST: `http://localhost:8080/api/messages` - SOAP: `http://localhost:8080/api/soap/messages`

Only the protocol configured in the sim.properties file is available to receive messages.

Configuration

Before connecting to a CISE node or adaptor, the CISE Sim needs to be properly configured.

The `conf/` directory contains two configuration files:

- `sim.properties`
- `config.yml`

sim.properties

This file is the main configuration file that allows to configure the parameters related to the protocol used by the application, the endpoint to send the messages to and the signature details.

To allow the CISE Sim to send message to the CISE Node, the node administrator needs to create a participant specifically for it. The node administrator should also provide the java key store file generated together with the participant specifying:

- the password to access the Java Key Store
- the alias to the key pair included in the Java Key Store
- the password to access the key pair included in the Java Key Store

  The keystore file should be substituted to the one present in the `conf` directory.

Configuration properties

Parameter	Description
simulator.name	This name is displayed on the CISE Sim web interface. It has no impact on the functioning of the application. It can be used to display which system the CISE Sim is impersonating.
destination.protocol	The protocol used by the CISE Sim to send messages to the destination.url. Allowed values are: SOAP and REST
destination.url	The destination URL where the CISE Sim will send the messages to. The url may point to a SOAP and to a REST interface according to the destination.protocol property.
templates.messages.directory	Specifies a path relative to the directory where the CISE Sim is unarchived. This path indicates where to find the templates loaded in the dropdown on the web interface. *WARNING* Currently the path is accepted only relatively to the sim installation directory.
signature.keystore.filename	the filename of the keystore contained in the `conf/` directory.
signature.keystore.password	the password to access the keystore.
signature.privatekey.alias	the alias that identify the *key pair* in the keystore that will be used to sign the CISE message. Example: `cisesim-nodeex.nodeex.eucise.ex`.
signature.privatekey.password	is the password to access the key pair.
app.version	The CISE Sim application version. This value is set by the build system and it has an informative purpose.
history.repository.directory	Specifies a path relative to the directory where the CISE Sim will write the xml format files of the messages received and sent. The name has a specific format, specified in the next chapter. *WARNING* Currently the path is accepted only relatively to the sim installation directory.
history.gui.maxthmsgs	Maximum number of threads that will be shown in the user interface.

Archived message file name format

The messages are archived in files. The single file contains the xml format of the message, and the name format is:
`Timestamp_TypeName_Direction_Uuid`

|Parameter|Description| |---|---| |Timestamp|Timestamp when the message was processed,following the format : yyyyMMdd-HHmmssSSS |TypeName|Type of the message (i.e. PULLREQUEST, FORWARD, etc.). For the Acknowledge messages, SYNCH are the ones received/sent synchronously after the message was sent/received |Direction|RECV for message received, or SENT for message sent |Uuid|Uniq identifier Some examples:
20200611-120029798_PULLREQUEST_SENT_31fb100d-dd13-450d-858b-d410a5f2c345
20200611-120029808_ACKSYNCH_RECV_184e0b37-bdb0-4efd-b993-ac18abd1f7ec

config.yml

The config.yml is needed to configure the application server used to run the CISE Sim application. Here is possible to configure the tcp port the CISE Sim will use to for serving the UI, to receive incoming messages:

```
server:
  # Protocol and port of simulator web interface
  applicationConnectors:
    - type: http
      port: 8080
```

While the logging information can be found mostly under the `logging.loggers` configuration:

```
logging:
  level: INFO
  loggers:
    "io.dropwizard.bundles.assets": INFO
    "eu.cise.sim.api": INFO
    "org.eclipse.jetty.server.handler": WARN
    "org.eclipse.jetty.setuid": WARN
    "io.dropwizard.server.DefaultServerFactory": WARN
    "io.dropwizard.bundles.assets.ConfiguredAssetsBundle": WARN
```

References and examples

sim.properties content

```
#
# CISE Sim (1.2.2-beta)
#

# This name is displayed on the CISE Sim web interface.
# It has no impact on the functioning of the application.
# It can be used to display which system the CISE Sim is impersonating.
simulator.name=eu.eucise.ex.cisesim-nodeex

# The protocol used by the CISE Sim to send messages to the destination.url.
#
# Allowed values are:
# - SOAP
# - REST
#
destination.protocol=SOAP

# The destination URL where the CISE Sim will send the messages to.
# The url may point to a SOAP and to a REST interface according to the
# destination.protocol property.
destination.url=http://your-endpoint.url.cise.ex:8080/api/soap/messages

# Directory relative to installation where the server looks for message templates
templates.messages.directory=templates/messages

# Signature configuration
signature.keystore.filename=cisesim-nodeex.jks
signature.keystore.password=cisesim
signature.privatekey.alias=cisesim-nodeex.nodeex.eucise.ex
signature.privatekey.password=cisesim

# The CISE Sim application version.
# This value is set by the build system and it has an informative purpose.
app.version=1.2.2-beta

# Repository information
history.repository.directory=msghistory
history.gui.maxthmsgs=10
```

config.yml content

```
#
# CISE sim application configuration (1.2.2-beta)
#

# The server section allows to modify the port used by the CISE Sim.
# To receive CISE messages and to serve the UI of the CISE Sim application
# The admin port is used to gather statistics of the communication
server:
  applicationConnectors:
    - type: http
      port: 8080
  adminConnectors:
    - type: http
      port: 8081

# The logging section is used to tweak the logging level of the simulator.
# To write the log in a file instead of the standard output is possible to
# uncomment the "type file" section.
logging:
  level: INFO
  loggers:
    "io.dropwizard.bundles.assets": INFO
    "eu.cise.sim.api": INFO
    "org.eclipse.jetty.server.handler": WARN
    "org.eclipse.jetty.setuid": WARN
    "eu.cise.sim.engine.Dispatcher": DEBUG
    "io.dropwizard.server.DefaultServerFactory": WARN
    "io.dropwizard.bundles.assets.ConfiguredAssetsBundle": WARN
    "org.wiremock": INFO
  appenders:
    - type: console
      threshold: ALL
      queueSize: 512
      discardingThreshold: 0
      timeZone: UTC
      target: stdout
#    - type: file
#      currentLogFilename: ./logs/sim.log
#      archivedLogFilenamePattern: ./logs/sim-%d{yyyy-MM-dd}.log.gz
#      archivedFileCount: 5
#      timeZone: UTC

# The assets section is required to serve the UI pages
# Given is a configuration internal to the application
# the following section should not be changed.
assets:
  mappings:
    /assets: /
  overrides:
    /base/: ${project.basedir}/cise-sim-react
    /base/static:  ${project.basedir}/cise-emulator-react/dist
```

Console output eu.cise.dispatcher.example

As a reference, when starting up the CISE Sim the output should be similar to the following one:

```
 $ ./sim run
CISE sim
Java path:    /home/gicappa/.jabba/jdk/amazon-corretto@1.8.222-10.1/bin/java
Java version: "1.8.0_222"
18:56:24.961 [main] DEBUG org.apache.cxf.common.logging.LogUtils - Using org.apache.cxf.common.logging.Slf4jLogger for logging.

================================================
INFO  [2019-12-02 17:56:26,552] org.apache.cxf.wsdl.service.factory.ReflectionServiceFactoryBean: Creating Service {http://www.cise.
INFO  [2019-12-02 17:56:26,875] org.apache.cxf.endpoint.ServerImpl: Setting the server's publish address to be /messages
INFO  [2019-12-02 17:56:26,883] io.dropwizard.server.ServerFactory: Starting EmulatorApp
     _________________ _______   ______ _______  ___
    / ____/  _/ ___// ____/  / ___//  _/  |/  /
   / /    / / \__ \/ __/     \__ \ / // /|_/ /
  / /____/ / ___/ / /___    ___/ // // /  / /
  \____/___//____/_____/   /____/___/_/  /_/

// CISE Sim // Ver: 1.3.0-ALPHA //

INFO  [2019-12-02 17:56:26,937] org.eclipse.jetty.server.Server: jetty-9.4.z-SNAPSHOT; built: 2018-06-05T18:24:03.829Z; git: d5fc052
INFO  [2019-12-02 17:56:27,252] io.dropwizard.jersey.DropwizardResourceConfig: The following paths were found for the configured res

     POST    /api/messages (eu.cise.sim.api.rest.MessageResource)
     GET     /api/ui/history/latest/{timestamp} (eu.cise.sim.api.history.HistoryResource)
     GET     /api/ui/history/message/{uuid} (eu.cise.sim.api.history.HistoryResource)
     POST    /api/ui/messages (eu.cise.sim.api.messages.UiMessageResource)
     POST    /api/ui/messages/discovery/send (eu.cise.sim.api.messages.UiMessageResource)
     GET     /api/ui/messages/discovery/values (eu.cise.sim.api.messages.UiMessageResource)
     POST    /api/ui/messages/incident/send (eu.cise.sim.api.messages.UiMessageResource)
     GET     /api/ui/messages/incident/values (eu.cise.sim.api.messages.UiMessageResource)
     GET     /api/ui/service/self (eu.cise.sim.api.rest.UIServiceResource)
     GET     /api/ui/templates (eu.cise.sim.api.rest.TemplateResource)
     GET     /api/ui/templates/{templateId} (eu.cise.sim.api.rest.TemplateResource)
     POST    /api/ui/templates/{templateId} (eu.cise.sim.api.rest.TemplateResource)


INFO  [2019-12-02 17:56:27,262] io.dropwizard.setup.AdminEnvironment: tasks =

    POST    /tasks/log-level (io.dropwizard.servlets.tasks.LogConfigurationTask)
    POST    /tasks/gc (io.dropwizard.servlets.tasks.GarbageCollectionTask)

INFO  [2019-12-02 17:56:27,271] org.eclipse.jetty.server.AbstractConnector: Started application@76b305e1{HTTP/1.1,[http/1.1]}{0.0.0.
INFO  [2019-12-02 17:56:27,271] org.eclipse.jetty.server.AbstractConnector: Started admin@6be766d1{HTTP/1.1,[http/1.1]}{0.0.0.0:8081
INFO  [2019-12-02 17:56:27,272] org.eclipse.jetty.server.Server: Started @2514ms
================================================

INFO  [2019-12-02 17:56:27,272] com.roskart.dropwizard.jaxws.JAXWSEnvironment: JAX-WS service endpoints [/api/soap]:

    /api/soap/messages ({http://www.cise.eu/accesspoint/service/v1/}CISEMessageServiceSoapImpl)
```

How to build the CISE Sim

To build the CISE Sim from the source code is required to install: - jdk 1.8 - nodejs 6.4.1+ - npm 10.11.0+

on your workstation.

There are three steps to build the simulator:

- install JavaScript dependencies
- build with npm ReactJS frontend
- build with maven the CISE Sim Java code

JavaScript install and build

To install the JavaScript dependencies is just enough to launch an npm command to install dependencies and to build an artifact of the front end:

```
$ cd cise-sim-react
$ npm install
$ npm run build --scripts-prepend-node-path=auto
```

Java API server build

As usual, it is possible to build the API server through maven build tool.

```
$ cd <root-path-of-cise-sim-code>
$ mvn clean install -U -P ciBuild
```

After the build succeeds, the final artifact can be found in `<root-path-of-cise-sim-code>/target` directory.