JChem Microservices

JChem Microservices provide microservices in small separate modules for different areas of ChemAxon functionalities like chemical dataset searching, conversion between chemical file formats.

JChem Microservices contain common microservice modules and modules providing ChemAxon functionalities.

Common modules:

- config
- discovery
- gateway

JChem modules:

- Calculations Web Services
- DB Web Services
- IO Web Services
- Markush Web Services
- Structure Checker Web Services
- Structure Manipulation

- Calculations Web Services

Download

Download **jws-installer** - appropriate for your operating system - from

https://chemaxon.com/products/jchem-engines/download

Software requirements

- Windows (64-bit), Linux, Mac OS
- Java 8

Installation

Windows

- unzip *jws_windows_x64_<version_number>.zip* or
- run *jws_windows_x64_<version_number>.exe* as Adminstrator

Linux

run `jws_unix_<version-number>.sh`

or

On Debian, Ubuntu

```
sudo dpkg -i jws_linux_<version-number>.deb
```

On Redhat, Centos

```
sudo yum install jws_linux_<version-number>.rpm
```

Mac OS

run `jws_macos_<version-number>.dmg`

Upgrade

Upgrade is the same process as installation.

The content of the *config* folders and the *data* folder of the DB Web Services module will not be overwritten.

However, special care must be taken in case of **DB Web Services**.

If the data structure version number of the new DB Web Services is higher than that of the existing old DB Web Services, the `updateMode` configuration parameter controls how the upgrade will be performed when DB Web Services service is started:

```
updateMode: EXIT|DROP|REINDEX
```

The available options are as follows.

EXIT: the service start process exits if any change is detected. Afterwards `updateMode` parameter must be set to DROP or REINDEX to start, or you should migrate back to the previous version.

DROP: the existing table and index data will be dropped.

REINDEX: the existing tables will be reindexed.

Uninstall

Depends on the operating system and on the installation mode you applied.

Windows:

run `C:\Program Files\ChemAxon\JChem Micro Services\uninstall.exe`

Linux:

run `<Installation folder>/ChemAxon/JChem Micro Services/uninstall`

or

On Debian, Ubuntu

```
sudo dpkg -r jws
```

On Redhat, Centos

```
sudo yum erase jws
```

Licenses

Put the license file under **jws/license/** folder with name: **license.cxl** or apply any of the options listed on the Installing to servers page.

- For using **DB Web Services**, " *JChem Microservices DB* " license is required.
- For using **IO Web Services**, "*JChem Microservices IO*" license is required.
- For using **Structure Checker Web Services**, "*JChem Microservices Structure Checker*" license is required.
- For using **Structure Manipulation**, "*JChem Microservices Structure Manipulation*" license is required.
- For using **Calculations Web Services**, "*JChem Microservices Calculations*" license is required.
- For using **Markush Web Services**, "*JChem Microservices Markush Enumeration*" license is required.

Logging

By default, the log files of all the modules are separately stored in **jws/logs/** folder. The place of log files and the logging level can be modified in the **application.properties** file of each service.

Health status

/actuator/health endpoint on the gateway reports the health status of the whole microservices system. It reports *Status 2xx* if all services are alive and work well.

The individual health status of the modules can be seen on the **<module_name>/actuator /health** endpoint.

Why does JChem Microservices have so many modules?

The idea behind JChem Microservices is to provide a solution with High-Availability, Robust and modern Web Service where highly used parts can be scaled up and rarely used parts can be switched down. For this we provide the opportunity to set up different arts of the software on many machines. All of the services can be configured by a central configuration service, they all register into a discovery service and have a common entry point, which should be the only visible service in your network.

Description of the modules

Config

Spring Cloud Configuration service is provided.

Run the service in command line in folder **jws/jws-config/**

`jws-config-service --start` (on Windows)

`jws-config-service start` (on Linux)

or

`run-jws-config.exe` (on Windows)

`run-jws-config` (on Linux)

Default configuration

common-config/application.properties	
# You can set all settings here for all applications	

eureka.client.serviceUrl.defaultZone=${EUREKA_URI:http\://localhost\:8761}/eureka/
 	
eureka.client.enabled=true
 	
spring.servlet.multipart.max-file-size=15MB
 	
spring.servlet.multipart.max-request-size=15MB	

# LOG config: 	
logging.level.root=WARN
logging.level.com.chemaxon.webservices.license=INFO	

```
# DB config:
initOnStart=AUTO
updateMode=EXIT

search.wallTimeLimitSeconds=3600

com.chemaxon.zetor.settings.indexDir=data/chemical-data/store
com.chemaxon.zetor.settings.scheme=mapdb
com.chemaxon.zetor.settings.forcePurge=true

com.chemaxon.zetor.types[0].version = 1
com.chemaxon.zetor.types[0].typeName = sample
com.chemaxon.zetor.types[0].typeId = 1
com.chemaxon.zetor.types[0].tautomer = OFF
com.chemaxon.zetor.types[0].standardizerAction = aromatize

com.chemaxon.zetor.types[1].version = 1
com.chemaxon.zetor.types[1].typeName = taumol
com.chemaxon.zetor.types[1].typeId = 2
com.chemaxon.zetor.types[1].tautomer = GENERIC
com.chemaxon.zetor.types[1].standardizerAction = aromatize

com.chemaxon.zetor.additional.scheme=mapdb
com.chemaxon.zetor.additional.indexDir=data/extra-data/
```

If DB Web Services are run as part of a microservices system, specify here its configuration parameters.

If DB Web Services are run as standalone web application, specify its configurations in its own module's application.properties file.

See explanations here

# Communication settings of gateway: 	
 	
ribbon.ReadTimeout=30000
 	
hystrix.command.default.execution.isolation.thread.timeoutInMilliseconds=25000000
	You can set up various timeouts in JChem MicroServices. We use the Spring Cloud Stack with solutions from Netflix. Ribbon load balancer is responsible for the communication between the services. You can set up its timeout settings with the ribbon prefix, like: ribbon.ReadTimeout. This property controls when to interrupt waiting for an answer between ChemAxon services. You can also setup a guard with Hystrix circuit breaker with properties like: execution.isolation.thread.timeoutInMilliseconds which will interrupt your request if it does not get an answer during the timeout.

config/application.properties

```
server.port=8888

logging.file=../logs/jws-config.log

eureka.client.enabled=true



spring.profiles.active=native

spring.cloud.config.server.native.searchLocations=./common-config
```

config/bootstrap.properties

Discovery

Eureka, the Netflix Service Discovery Server and Client is provided.

Run the service in command line in folder **jws/jws-discovery/**:

`jws-discovery-service --start` (on Windows)

`jws-discovery-service start` (on Linux)

or

`run-jws-discovery.exe` (on Windows)

```
run-jws-discovery  (on Linux)
```

Default configuration

config/application.properties
server.port=8761
logging.file=../logs/jws-disocvery.log

config/bootstrap.properties
spring.cloud.config.failFast=true
spring.cloud.config.uri=${CONFIG_SERVER_URI:http\://localhost\:8888/}
spring.cloud.config.retry.initialInterval=3000
spring.cloud.config.retry.multiplier=1.2
spring.cloud.config.retry.maxInterval=60000
spring.cloud.config.retry.maxAttempts=100

Gateway

Spring Cloud Gateway is provided.

Run the service in command line in folder **jws/jws-gateway/**:

```
jws-gateway-service --start (on Windows)

jws-gateway-service start (on Linux)
```

or

```
run-jws-gateway.exe (on Windows)

run-jws- gateway  (on Linux)
```

Default configuration:

config/application.properties
server.port=8080
logging.file=../logs/jws-gateway.log
eureka.client.enabled=true

config/bootstrap.properties
spring.cloud.config.failFast=true
spring.cloud.config.uri=${CONFIG_SERVER_URI:http\://localhost\:8888/}
spring.cloud.config.retry.initialInterval=3000
spring.cloud.config.retry.multiplier=1.2
spring.cloud.config.retry.maxInterval=60000
spring.cloud.config.retry.maxAttempts=100
spring.cloud.config.retry.maxAttempts=100

Message limitations in Gateway

Our Gateway is a simple Zuul Proxy (https://github.com/Netflix/zuul), developed by Netflix. By default it has a smart algorithm to retry failed requests on different backend instances. For this it keeps each request in memory until it is answered successfully or it cannot be retried on any other servers. This leads us to a limitation where the sent message is kept in memory so if we want to send lots of data at once (like indexing a large batch of molecules), we have to grant enough memory for Zuul to keep our whole request in memory during the process. You can set Gateway's maximum memory as any Java process in the appropriate `.vmoptions` file. (If you execute the service with `jws-gateway-servcie`, then use `jws-gateway-service.vmoptions`; if you execute the service with `run-jws-gateway`, then use `run-jws-gateway.vmoptions`.) The default setting is `-Xmx256m`, which means maximum 256 MB memory.

To protect Gateway from overloading its memory, we have introduced a filter to reject every message where `Content-Length` header is greater than 1/3 of the available maximum memory. With default memory settings this will lead you to reject every request with body greater than 85 MB.

You can also specify directly how many bytes do you allow in a request. For this set `gateway.max-message-size=<number_of_bytes_to_allow>` to the number of bytes you want to allow in a request. (Set this as any other property in Config server.)

If a request is too large, then a HTTP 413 error will be returned by the Gateway server.

Calculations Web Services

DB Web Services

IO Web Services

Markush Web Services

Structure Manipulation

Docker image example

A docker image of JChem Microservices is provided as an example.