Copper Monitoring GUI

The Copper Monitoring GUI is used for monitoring applications built with the Copper Engine. It connects through JMX to a Copper Engine application and provides information about the current state of the application's engines and workflows. Copper Monitoring is able to connect to more than one application at the same time. If engines from different applications are part of same application cluster (they share an 'engineClusterId') they will be grouped together for better visualization and ease of use.

Features:

1. Get the overhead view of your application:

- Real-time statistics visualized
- Number of active/broken workflows per engine
- Workflow details and interaction
- Audit Trail display
- Workflow Repository display
- Processor Pool display

2. Broken Workflows (with state of ERROR or INVALID)

- Display details (with stacktrace) of broken workflows
- Display error and last wait Strack Traces
- Display source code of the workflow (with highlights on the last waiting line and error line)
- Display the workflow's deserialized state
- Open workflow in pop-up window
- Filter workflows by class name, state or created and modification date
- Restart broken workflows (one, filtered or all)
- Delete broken workflows (one, filtered or all)

3. Waiting Workflows (with state of WAITING)

- Display details (with stacktrace) of waiting workflows
- Display source code of workflow (with highlights on the last waiting line and error line)
- Display last wait Strack Traces
- Display the workflow's deserialized state
- Open workflow in pop-up window
- Filter workflows by class name or created and modification date
- Delete waiting workflows (one, filtered, or all)

4. Workflow Repository info

- Display type, source, and last build results of the repository
- Display list of classes within the repository and their details
- Display source code of the repository classes

5. Processor Pools info

- Display processor pool statistics
- Processor pool actions: Resume, Suspend, Resume Deque and Suspend Deque

6. Audit Trail

- Filter audit trail based on time, log level, class, id, etc.

7. Collect and show Statistics for all engines

- The GUI can collect statistics and store them in Local Storage, but only while the page is running in a browser
- The GUI can alternatively give a user custom generated settings to use with Telegraf and Influx DB so that statistics can be collected continuously.
- Statistics will be visualized in real-time from either data source

Installation:

Download the Copper Monitoring source code here:
https://github.com/copper-engine/copper-monitoring

Or pull a Docker image with the command:
`Docker pull copperengine/copper-monitoring`

Set up - from Source Code:

To build and zip the source code for Docker to use, simply run the following commands:
Linux: `./start.sh --dockerize`
Windows: `./start --dockerize`
This will build and store the zipped source code in the /docker folder.

Next, check the default Dockerfile, as you may want to change certain environmental variables such as which credentials are required to log into the GUI, or what the default JMX credentials are.
Note: The Environment Variable `MONITORING_AUTH` sets the required credentials to log into the GUI. The default username / password is 'admin' : 'admin'.

To then create a Docker image from the zipped source code, run the following command:
`docker build -t copper-monitoring ./`
In this case, 'copper-monitoring' is a user defined variable name for the image.

To create a Docker container from the image, run the following command:
`docker create -p {nnnn}:8080 copper-monitoring`
Where {nnnn} is a port of your choice to forward to the container.

Finally, run your container with the command:
`docker start {id}`
Where {id} is the ID of the container.

NOTE on 'localhost': By default, inside a Docker container one is not able to access the host machine's 'localhost' address or any services hosted there. If you would like to access services hosted on your machine's localhost, take one of the following approaches based on your OS:

Linux: Use the following alternate command to create your docker image:
`docker create --network=”host” copper-monitoring`
You can not forward ports in this mode, however this ‘--network’ tag will allow you to directly access your machine’s localhost. Access the Copper Monitoring application now through its default port, ‘8080’.

Windows: Instead of ‘localhost’ use ‘host.docker.internal’ and it will be directed there.

Set up - from Docker Image:

Pull the Docker image with:
`Docker pull copperengine/copper-monitoring`

Then you may edit your Environment Variables through a file. A default Environment Variable file can be found at the following link. Note: The Environment Variable `MONITORING_AUTH` sets the required credentials to log into the GUI. The default username / password is ‘admin’ : ‘admin’.
https://github.com/copper-engine/copper-monitoring/blob/master/docker/env-vars/env.list

Build your Docker container including the Environment Variables with the following command:
`docker create -p {nnnn}:8080 --env-file {path to env.list} copper-monitoring`
Where {nnnn} is a port of your choice to forward to the container.

Finally, run your container with the command:
`docker start {id}`
Where {id} is the ID of the container.

NOTE on ‘localhost’: By default, inside a Docker container one is not able to access the host machine’s ‘localhost’ address or any services hosted there. If you would like to access services hosted on your machine’s localhost, take one of the following approaches based on your OS:

Linux: Use the following alternate command to create your docker image:
`docker create --network=”host” copper-monitoring`
You can not forward ports in this mode, however this ‘--network’ tag will allow you to directly access your machine’s localhost. Access the Copper Monitoring application now through its default port, ‘8080’.

Windows: Instead of ‘localhost’ use ‘host.docker.internal’ and it will be directed there.

Connecting to your Application:

The application you would like to monitor should have open JMX ports. Do this by setting the Virtual Machines options, take the following for example:

```
-Dcom.sun.management.jmxremote=true
-Dcom.sun.management.jmxremote.authenticate=false
-Dcom.sun.management.jmxremote.port=1099
-Dcom.sun.management.jmxremote.ssl=false
```

If you want to protect your exposed JMX port with credentials, use this alternative option:

`-Dcom.sun.management.jmxremote.authenticate=true`

And set your desired credentials in `Java/jdk/jre/lib/management` with the `jmxremote.password` file. For more information, see:

https://docs.oracle.com/javase/8/docs/technotes/guides/management/agent.html

Finally, your Copper application must have a JMX Exporter that will export your MBeans. For Copper Monitoring to work properly, you should export the following MBeans: engine, database, workflow repository, processor pool, audit trail query.

Securing the Connection with SSL:

For a secured JMX connection between your application and the Copper Monitoring server, add the following Virtual Machine options to your application:

```
Dcom.sun.management.jmxremote.ssl=true
-Djavax.net.ssl.keyStore={path to keystore_monitoring}
-Djavax.net.ssl.keyStorePassword={password}
-Djavax.net.ssl.trustStore={path to truststore_monitoring}
 -Djavax.net.ssl.trustStorePassword={password}
```

And add the following Virtual Machine options to the Copper Monitoring server:

```
-Djavax.net.ssl.trustStore={path to truststore_copper_app}
-Djavax.net.ssl.trustStorePassword={password}
```

For a secured connection between the Copper Monitor and your browser, enable HTTPS by adding the following Environment Variables to the Copper Monitor:
`KEYSTORE_LOC={path to keystore_monitoring}`
`KEYSTORE_PASS={password}`
`TRUSTSTORE_LOC={path to truststore_monitoring}`
`TRUSTSTORE_PASS={password}`
`HTTPS_ENABLED=true`

When using a Docker container, the same Virtual Machine options and Environment Variables must be applied.

If you are using an existing Docker image from Dockerhub, apply them using the Environment Variable file when building your Docker container, as described above in the section “Set up - from Docker Image”. The appropriate Virtual Machine options and Environment Variables are commented out in the default file, which you can find here:
https://github.com/copper-engine/copper-monitoring/blob/master/docker/env-vars/env.list
To apply the file properly and create the necessary folder for a secure connection, use the following command to create your container:
`` `docker create -p {nnnn}:8080 -v {path to your security files}:/app/certs --env-file {path to env.list} {image}` ``
Otherwise, if building a new Docker image from the source code, you can edit the Dockerfile directly.
The ‘-v’ tag will bind your local security files to the Docker container so that it can use them for the secure connections.

Troubleshooting:

If you can’t log into Copper Monitoring (the default credentials are admin:admin) make sure that the Environment Variable `` `MONITORING_AUTH` `` is set and use those credentials.

If the Copper Monitor cannot connect to your application, make sure you are connecting to the correct “host:port” and make sure your application has that port exposed. Try to connect to your application with JConsole (using your application’s host and port) to check whether or not your MBeans are properly exported. Your MBeans should be visible in JConsole if they are properly exported.

Copper Starter:

To test or demonstrate the Copper Monitor and its functionality, you can make use of the Copper Starter application. It is a simple demo application using the Copper Engine, and it can simulate broken workflows. Run a second instance with the Environment Variable `COP_STARTER_VERSION` to ‘V2’ to simulate a second group of engines to see how they cluster together. After container startup it will start running 2 workflows every 5 minutes. Workflows firstly will
be in running state, then in waiting state and finally to broken state.

Connect to the Copper Starter on port 1099, and a second connection to the ‘V2’ version on port 1098.

Download the source code from here:
https://github.com/copper-engine/copper-starter

Or pull a Docker container with the following command:
`docker pull copperengine/copper-starter`
Create and run the container with the following command:
`docker run -p 1099:1099 copperengine/copper-starter`