OCCIware demo - Linked Data server on Docker optimized for analytics

Introduction

This demo showcases OCCIware Studio deploying a complete, working Ozwillo Datacore cluster (one Java and 3 mongo replica nodes) on Docker both locally and on a remote Open Stack VM, and developing a custom OCCI extension (including designer and connector) for Linked Data that allows to publish data projects and let them use a specific mongo secondary rather than the whole cluster (typically for read-heavy queries such as for analytics). This last point is achieved by visually linking OCCI Resources across Cloud layers: from Linked Data as a Service (LDaaS) to Infrastructure as a Service (IaaS).

It has been originally shown at EclipseCon France 2016 on June the 9th (with versions 1.0 of Docker images and 2cada878ecaf901fb7750d65b6cda66815467ff2 of Datacore) in the One Cloud API to rule them all talk.

In Fall 2016, it has been updated, improved and enriched, notably with demonstration of the connector's OCCI HTTP API, being provided by the erocci runtime through erocci-dbus-java and managed by the OCCInterface web admin UI.

During Summer 2017, the demonstration has been fully dockerized, facilitating its reproduction by anyone.

Requirements

Java 8, Maven 3, OCCIware Studio (latest in the ecore repository), VirtualBox (latest, tested with version 5.0.40), Docker & Docker-Machine (latest, respectively tested with versions 17.03.1-ce and 0.4.1).

You might want to take a look at the latest presentation on Occiware made by Marc Dutoo: Occiwareposs 2016 - An extensible standard xaas cloud consumer platform. In order to better understand the slides, you may want to watch the video of the presentation on Youtube.

Installation of the requirements

Install Docker

Follow the recommended steps in the Docker Documentation - Installation for Ubuntu page.

Install Docker Machine

Follow the recommended steps in the Install Docker Machine page.

Install Virtualbox, java and maven

```
sudo apt-get install virtualbox maven default-jre default-jdk
```

Install the CloudDesigner

> **Note**: To ease the installation process and allow you to copy the following commands verbatim, we recommend that you clone every repository (including the one containing this README.md) for this demo **IN THE SAME PARENT DIRECTORY**.

Clone + build locally the Datacore, so that the LinkedData connector will be able to find the Datacore client in the local maven repository (This step will no longer be necessary when the datacore is posted on Maven-Central):

```
git clone https://github.com/ozwillo/ozwillo-datacore.git
cd ozwillo-datacore
mvn clean install
```

Install the OCCI-Studio following the indications in the "If you want to play around with the sources" section of the official documentation page (don't forget to read the warning below). Follow the instructions of parts "1.3. The not so easy and long way : starting from an existing Eclipse installation" and "2. The “guest” Eclipse: the OCCI-Studio Project".

Repository structure

This repository is divided into four subfolders:

- "docker" contains the docker files for each project involved in executing the linked data demonstration. If you want to rebuild the docker image for a particular application, you simply have to execute the following commands in its corresponding subfolder, replacing the mentions between angle brackets:

```
sudo docker build -t <your-dockerhub-username>/<the-project-name>:1.0 .
sudo docker push <your-dockerhub-username>/<the-project-name>:1.0
```

If you don't already have an account on the Docker Hub, create one, and then log in in your terminal with the following command:

```
sudo docker login
```

- "erocci" contains an integration of the demonstration with the erocci backend (no longer maintained).
- linkeddata_extension contains a copy of the latest version of the linkeddata extension that can also be found in the ecore repository.
- "mongo-dumps" contains cleaned up mongodb dumps of the datacore database: **base_fixed.tar.gz** contains the data and models for the org_1 and geo_1 (which is a bare minimum to create a new project on the Datacore), and **energy_fixed.tar.gz** builds upon base_fixed.tar.gz by adding the **energy_0**'s project required data and models (organization, persids, energy contracts and energy consumptions). If you want to take a peek into the data, uncompress the one you want to use in the same folder, and restore it with the restoration tool provided by mongodb, for example:

```
cd mongo-dumps/
tar xzf energy_fixed.tar.gz
mongorestore --db datacore energy_fixed/dump/datacore/
```

- "ozwillo-datacore-occiware" is the CloudDesigner project that contains the models to execute the demo. You can open it in the CloudDesigner you installed in the previous section by following the instructions below.

- "nodemcu" contains the source files to flash the NodeMCU and instructions on how to use it with the Blynk demo component.

Building and running the project

1. In the CloudDesigner (the "Guest Eclipse"), import the ozwillo-datacore-occiware project, which is in the subfolder of the same name in the occiware-ozwillo repository : File > Import... > General > Projects from Folder or Archive > Next > Directory > OK > Finish. In the file tree on the left, find the representations.aird file, click on the '>' on its left in order to see its sub-elements (OCCI extensions) in the file tree, then again to see OCCI diagram kinds, and again to see OCCI configurations. Then open the following OCCI configurations by double-clicking on them:
    - ozwillo-datacore-cluster: this will be displayed in the Docker designer,
    - Mytest: this will be displayed in the Linked Data designer.
    - energy: this will be displayed in the Linked Data designer.
2. Before going any further, you must know that you can open the Properties panel by clicking on Window > Show view > Properties. When you explore the graphs, it will allow you to know what is inside the boxes, their configuration, and to edit it.

    > **Important Performance Note**: Since the **Properties panel** is being listened to by model elements through EMF, having it always open can lead to **tremendous lags** when using the Docker studio for example. As long as the performance issues are not solved, **we recommend that you open it only when you need to see the details of a particular element.**

3. In the Docker designer of ozwillo-datacore-cluster.docker, right-click on the linkeddatadevlocal VM , and click on Docker Execute > Start (We don't use Startall since we want to control the start order of the VMs to ensure they have the proper IPs set). Wait until the VM has been created (the loading screen will disappear, and its block will become green).
4. Now shut down the VM so that we can change its alloted RAM : right-click on the linkeddatadevlocal VM , and click on Docker Execute > Stop. You will need to open the Virtualbox GUI and setup the alloted RAM: right-click on the VM > Configuration > System > Slide the memory cursor to 4096Mo > Validate with OK. Then go start again the VM in the Docker Designer as explained in step 3.
5. Start each of the following containers manually from the Docker-Studio **in the following order** (right-click on the container block, then click on Docker Execute > Start):
    - ozwillo-mongo-1, then 2, and finally, 3
    - ozwillo-datacore-1
    - occinterface-1
    - martserver-linkeddata-1

    > **Important Note:** Since on the first start, the docker images are being downloaded, it might take quite a while (up to 5-10 minutes for the biggest images like smilelab/martserver-linkeddata:1.0 or smilelab/ozwillo-ozenergy:1.0) for the download to complete. The interface will freeze during the entire duration, but don't worry, it's ok.

> If not on the first start, and you removed all containers for whatever reason, and are trying to create them anew, be careful not to boot the datacore before the ozwillo-mongos are ready, or they won't load the energy data. To check that, simply ssh into the ozwillo-mongo-1, start the mongo shell, and use 'show dbs' to check if the datacore db exists or not.

6. Open the Virtualbox GUI and setup the redirection of the required ports: right-click on the VM > Configuration > Network > Advanced > Port Redirection > Click the Add button (right of the window) > fill in the info and do OK > OK > Close Virtualbox window when you are finished:

Name	Protocol	Host IP	Host Port	Client IP	Client Port
datacore	TCP	127.0.0.1	8088		8088
martserver	TCP	127.0.0.1	8081		8081
occinterface	TCP	127.0.0.1	3000		3000
ozenergy	TCP	127.0.0.1	8080		8080

If you want to change the default ozwillo-mongo-master-with-energy-data image that is attributed to ozwillo-mongo-1 for the ozwillo-mongo-master-with-base-data image, in order to test data/model importation, you will also need to set up the port redirection to allow loading the data from your local computer to the container within the VM.

Name	Protocol	Host IP	Host Port	Client IP	Client Port
mongo-1-debug	TCP	127.0.0.1	30000		27017

In case you would like to debug the applications using remote debugging, you might want to also set the following port redirections:

Name	Protocol	Host IP	Host Port	Client IP	Client Port
remote-martserver	TCP	127.0.0.1	8000		9000
remote-datacore	TCP	127.0.0.1	8500		8500
remote-ozenergy	TCP	127.0.0.1	8000		8000

> **Note**: (LATER it would be better to be able to do it automatically using the OCCI configuration of the VM, see issue #132 on Github).

> **Note**: If you want to use the ozwl-occi-data Generic VM, you would need to connect to the Ozwillo vpn,

> locally modify the "MARTSERVER_URL" constant in "/org.occiware.clouddesigner.occi.linkeddata.connector/src/org/occiware/clouddesigner/occi/linkeddata/connector/LdnodeConnector.java" to the IP address of the VM, and then, when you've started the CloudDesigner ("Guest Eclipse") and opened the Docker config model, complete the following fields in the properties of the VM box: IP Adress, SSH Key, SSH User. Then do as detailed for the local VirtualBox VM. For the SSH Key, don't forget to add it to the VM's authorized keys (if you don't know what this sentence means please go to https://www.digitalocean.com/community/tutorials/how-to-set-up-ssh-keys--2):

~~~
```bash
ssh-copy-id root@10.28.7.17
```
~~~

> **Note**: If you want to use the IoT extension, you will need to use the ozwl-occi-data Generic VM : see above for instructions.

> **Note**: If you want to use the OW2 Open Stack VM, in the Docker Studio on said VM (linkeddatadevlocal in the ozwillo-datacore-cluster.docker configuration), first set its "username" and "password" attributes (if you don't have any, you must ask 0W2 at https://jira.ow2.org/browse/SERVICEDESK). Then do as detailed for the local VirtualBox VM.

Testing the project

Once the project has been built and is running, you can do the following actions to test its functionnalities.

Playing around with "Mytest"

1. In your browser, go to the Datacore Playground at http://localhost:8088/dc-ui/index_old.html (http://localhost:8088/ may not redirect you properly). The top dropdown box should list all existing data projects. If you select for instance the "geo_1" project, its "project portal" should be displayed in the central colored textarea, and clicking on its first (eponymous) link should display the project's configuration in JSON(-LD) format.

2. Go to the "Mytest" file you opened earlier. Do the "Publish" action on the "geo_1" project (Right-Click on its box > Publish), in the Mytest file. It should set its "dcmp:frozenModelNames" property to ["*"] (and similarly "Unpublish" should set it to []), which can be seen in the Datacore Playground in said project's configuration (to find the project configuration in the Datacore, select the oasis.meta project > click on "all their resources" that is on the "dcmpv:PointOfView_0" line > find the "geo_1" project and click on its name). You can also do things in reverse, that is, click on "EDIT" in the Datacore Playground and switch the "dcmp:frozenModelNames" property to ["*"], save it by clicking on "PUT", and then go to the CloudDesigner, right-click on the "geo_1" project > CRUD operations > Retrieve, to see the model be updated.

3. Actually, you can do all the same updates from either the CloudDesigner, the Datacore or the OCCInterface, which connects to the MartServer. For that, go to http://localhost:3000/, and it will open the OCCInterface. In the top dropdown menu, select "http://martserver-linkeddata-1:8081", and once you see it has loaded (text appears in the text area below), click on the "Select Kind" menu in the top > "http://occiware.org/linkeddata#" > "ldproject - LDProject". Then, click on the "EDIT" button. Remove all content (which should just be "{}") from the text area and

copy-paste the following code:

```
{
  "kind": "http://occiware.org/linkeddata#ldproject",
  "attributes": {
    "occi.ld.project.name": "geo_1"
  }
}
```

Click on the "POST" button and you should obtain a result like this:

```
{
  "id" : "urn:uuid:8bd3146b-9e60-4440-aaea-24cf188a28fb",
  "kind" : "http://occiware.org/linkeddata#ldproject",
  "mixins" : [
  ],
  "attributes" : {
    "occi.ld.project.name" : "geo_1",
    "occi.ld.project.lifecycle" : "draft",
    "occi.ld.project.robustness" : "cluster"
  },
  "actions" : [
    "http://occiware.org/linkeddata/ldproject/action#publish",
    "http://occiware.org/linkeddata/ldproject/action#unpublish",
    "http://occiware.org/linkeddata/ldproject/action#update"
  ],
  "location" : "http://martserver-linkeddata-1:8081/ldproject/8bd3146b-9e60-4440-
aaea-24cf188a28fb"
}
```

As you can see, the state of the "geo_1" project is perfectly described in the OCCI model presented in the OCCInterface, just as it was on the Datacore when you clicked on the "POST" button.

Don't hesitate to click on "ACT..." > "publish" to change "occi.ld.project.lifecycle" to "published", and check on the Datacore that your modifications have been reflected by hitting the Datacore Playground's "GET" button.

4. In the Linked Data Studio, do the "Update" action on the "geo_analytics_1" project. It should create it, meaning it should be listed in the Datacore Playground's project dropdown list after refreshing the page. Its project configuration (if displayed in the Datacore Playground as said) should be the same as the one set in the OCCI configuration. Especially, the geo_analytics_1's project "dcmpv:name" property should be set to the value of the "occi.ld.project.name" attribute ("geo_analytics_1"), and its "dcmp:localVisibleProjects" property should be a list of URIs of projects linked by LDProjectLinks in the OCCI configuration.

   Note: For now the Update action is merely doing a "get" then "create" or "merge and update" according to whether the LDProject already exists. LATER, the Update action could probably be replaced by properly implementing the CRUD > POST action, possibly improved by implementing a "Synchronize" action getting back the current state of the configuration, maybe automatically executed, with "occi.ld.project.version" being -1 for not yet created projects.

5. In the Datacore Playground, select the "geo_analytics_1" project as current project in the top dropdown box. Then copy-paste the sample analytics query below (*) in the Playground's URL bar and execute it using the "?" debug button. Find "serverAddress" (use Ctrl+F in your browser, that will be faster) in the results: its "host" attribute should be set to the host of the Compute that is linked to the OCCI LDProject through a LDDatabaseLink (the ozwillo-mongo-2 secondary MongoDB host in the demo configuration), and to the primary MongoDB host otherwise (ozwillo-mongo-1 in the demo configuration), as is the case when selecting another project (such as "geo_1"). This proves that the LdDatabaseLink works !

(*) sample analytics query FOR NOW:

```
/dc/type/dcmo:model_0?dcmo:isHistorizable=false
```

Playing around with "energy"

Now that you got acquainted with the Datacore Playground, the OCCInterface and the CloudDesigner, let's explore the OzEnergy project. Go to the "energy" file you opened earlier.

1. Just like with "Mytest", all previously presented actions are available: you can again try to play around with linked data projects properties in either of the interfaces, be it the CloudDesigner, the Datacore Playground or the OCCInterface.

2. What is also interesting to see here, is the appearance of the LDNode. The LDNode is an object to configure the deployment of the system via the MartServer. By default, the ozenergy project will fetch its data from ozwillo-mongo-1, but for the sake of performance, we want it to fetch the data only from a replica: ozwillo-mongo-3. In order to tell it to do that, you will need to upload the configuration that has been specified in the "energy: LDnode" box to the MartServer **before** starting the ozwillo-ozenergy-1 container, so that it will be able to fetch the right configuration on boot. To do so, simply right-click on the box > CRUD Operations > Create. You can then go to the OCCInterface to check that the model has been properly uploaded by clicking on "Select Kind" > "http://occiware.org/linkeddata#" > "ldnode - LDNode".

> **IMPORTANT NOTE:** If you are using the preprod demonstration, don't forget to locally modify the "MARTSERVER_URL" constant in "/org.occiware.clouddesigner.occi.linkeddata.connector/src/org/occiware/clouddesigner/occi/linkeddata/connector/LdnodeConnector.java" to the IP address of the VM : "private static final String MARTSERVER_URL = "http://10.28.7.17:8081/";".

3. Before starting the container, you might want a mean to verify that the ozenergy container actually fetches its data from ozwillo-mongo-3, and not ozwillo-mongo-1. For that you will need to connect to the ozwillo-mongo-3 container and check the database's logs. To do so, simply execute the following commands:

```
# SSH into the VM
docker-machine ssh occiwareozwillodevlocal
# SSH into the container
docker exec -it ozwillo-mongo-3 /bin/bash
# Activate the logging of all queries
mongo datacore --eval "db.setProfilingLevel(2)"
# Display logs
tail -f /var/log/mongodb/mongod.log
```

4. Start the ozwillo-ozenergy-1 container in the Docker-Studio (right-click on the container block, then click on Docker Execute > Start). Again, it might take a while on first start, since the Docker Image must be first be downloaded. What you expect to see in the log lines of the ozwillo-mongo-3 container are queries to Energy Consumption data, like the following:

```
2017-07-28T10:13:21.036+0000 [conn1725] getmore
datacore.energy_0.enercons:EnergyConsumption_0 cursorid:369832659846
ntoreturn:0 keyUpdates:0 numYields:3 locks(micros) r:86196 nreturned:8666
```

```
reslen:4194364 113ms
```

If you do, then it means, data is effectively being read by ozwillo-ozenergy-1 from ozwillo-mongo-3.

5. Once the logs stop displaying lines like the one presented above, it means that ozwillo-ozenergy-1 has successfully loaded. You can then go to your favourite web browser, and open "http://localhost:8080/", which will lead you to the ozenergy homepage. Clicking on "My Consumption" will lead you to graphs and tables detailing "your" consumption (which is, in dev mode, just sample data), aggregated as sums or averages by time period. Clicking on "Overview" will show you the same kind of aggregations, but at a city's scale.

> Note: Please be aware that any changes to the configuration of the ldnode will only be taken into account when the ozwillo-ozenergy-1 container is restarted, since these configuration elements must be given on the project's start.

To rewrite the extension from scratch

> **Important tip**: BEWARE, docker-machine doesn't allow hyphens in its VM names (better in v2).

If you prefer rewriting the Linked Data extension from scratch rather than using the complete version provided by the demo archive, do as in the EclipseCon slides, with the following hints.

Extension definition linkeddata.occie

Reuse the provided one, or define it using the Extension designer like this:

```
extension linkeddata : "http://occiware.org/linkeddata#"
import "http://schemas.ogf.org/occi/core#/"
import "http://schemas.ogf.org/occi/infrastructure#/"
import "http://schemas.ogf.org/occi/platform#/"
kind ldproject extends core.resource {
        title "LDProject"
        attribute occi.ld.project.name : core.String
        attribute occi.ld.project.published : core.Boolean = "false"
        attribute occi.ld.project.robust : core.Boolean = "true"
        action publish ()
        action unpublish ()
        action update ()
}
kind lddatabaselink extends core.link {
        title "LDDatabaseLink"
        attribute occi.ld.dblink.database : core.String = "datacore"
        attribute occi.ld.dblink.port : core.Number = "27017"
}
kind ldprojectlink extends core.link {
}
```

Extension connector

Reuse the provided code.

Extension designer

Reuse the provided one, or use the generated one and add:

- A copy of the Docker Designer's container Container.
- A Drop Container that targets it, to allow drag'n'dropping Docker containers in the LinkedData Designer.

Docker configuration occiware-datacore-cluster.docker.occic

Reuse the provided one, or define it using the Docker designer like this (only showing the local VirtualBox, but others ex. remote OW2 OpenStack can be created just the same way):

```
configuration
use "http://occiware.org/occi/docker#/"
resource "6df690d2-3158-40c4-88fb-d1c41584d6e4" : docker.machine_VirtualBox {
        state name = "ozwillodatacoredev"
        state occi.core.id = "6df690d2-3158-40c4-88fb-d1c41584d6e4"
        link "da58c05f-fe96-4fc8-aab9-3e9232aab767" : docker.contains target
        "9dcd39ac-4451-44eb-ac05-227951c23d40" {
                state occi.core.id = "da58c05f-fe96-4fc8-aab9-3e9232aab767"
        }
        link "a92ea98c-ca50-467b-a97c-bba9cc483322" : docker.contains target
        "71405ae7-2402-455c-9c96-0de3e4fc39a6" {
                state occi.core.id = "a92ea98c-ca50-467b-a97c-bba9cc483322"
        }
        link "177a2374-1194-41b0-9f34-0c873b3400bf" : docker.contains target
        "55936644-6215-495d-967f-7d453be484a5" {
                state occi.core.id = "177a2374-1194-41b0-9f34-0c873b3400bf"
        }
        link "fe219da5-79d0-477b-ae3d-f3c976c70d6a" : docker.contains target
        "cc806100-d62a-488f-ac13-eb9b20d2914e" {
                state occi.core.id = "fe219da5-79d0-477b-ae3d-f3c976c70d6a"
        }
}
resource "9dcd39ac-4451-44eb-ac05-227951c23d40" : docker.container {
        state occi.core.id = "9dcd39ac-4451-44eb-ac05-227951c23d40"
        state name = "ozwillo-mongo-1"
        state image = "mdutoo/ozwillo-mongo"
        state occi.compute.hostname = "ozwillo-mongo-1"
}
resource "71405ae7-2402-455c-9c96-0de3e4fc39a6" : docker.container {
        state occi.core.id = "71405ae7-2402-455c-9c96-0de3e4fc39a6"
        state name = "ozwillo-mongo-2"
        state image = "mdutoo/ozwillo-mongo"
        state occi.compute.hostname = "ozwillo-mongo-2"
}
resource "55936644-6215-495d-967f-7d453be484a5" : docker.container {
        state occi.core.id = "55936644-6215-495d-967f-7d453be484a5"
        state name = "ozwillo-mongo-3"
        state image = "mdutoo/ozwillo-mongo"
        state occi.compute.hostname = "ozwillo-mongo-3"
}
resource "cc806100-d62a-488f-ac13-eb9b20d2914e" : docker.container {
        state occi.core.id = "cc806100-d62a-488f-ac13-eb9b20d2914e"
        state name = "ozwillo-datacore-1"
        state image = "mdutoo/ozwillo-datacore:latest"
        state ports = "8080:8080"
        state occi.compute.hostname = "ozwillo-datacore-1"
        link "bf49c770-453c-4a16-a7f0-4f8bae191706" : docker.link target
        "9dcd39ac-4451-44eb-ac05-227951c23d40" {
                state occi.core.id = "bf49c770-453c-4a16-a7f0-4f8bae191706"
        }
        link "fe41e935-801c-41a4-8e2f-5b29daa978ce" : docker.link target
        "55936644-6215-495d-967f-7d453be484a5" {
                state occi.core.id = "fe41e935-801c-41a4-8e2f-5b29daa978ce"
        }
        link "29e9dbc4-44db-48b3-af1d-7d0123bcf3ec" : docker.link target
        "71405ae7-2402-455c-9c96-0de3e4fc39a6" {
                state occi.core.id = "29e9dbc4-44db-48b3-af1d-7d0123bcf3ec"
        }
}
```

LinkedData configuration Mytest.linkeddata.occic

Reuse the provided one, or define it using the LinkedData designer like in the EclipseCon slides.