Installation, Deployment and customisation manual

Cloudwatch2 Clustering web application

Version 1.1

19 February 2016

Michel Drescher, Oxford e-Research Centre

University Oxford

[Michel.Drescher@oerc.ox.ac.uk](mailto:Michel.Drescher@oerc.ox.ac.uk)

Introduction 2

System architecture 2

Deployment scenarios 2

Scenario 1: All-in-one deployment 2

Scenario 2: PostgreSQL and Tomcat on different hosts 2

Installation and deployment 3

Assumptions 3

Database commissioning 3

Installing PostgreSQL 3

Configuring PostgreSQL 3

Application server commissioning 6

OPTIONAL: Remove default web applications 6

Web Application commissioning 7

# Introduction

This document provides information on system architecture, installation and customization of the Cloudwatch2 clustering web application.

The intended audience are system administrators, and web application designers.

Essentially, the document is split in two parts; while perhaps not interesting for web application designers, the following section provides a high-level overview of the architecture of the application, which major subsystems are needed, and which software development tools and frameworks are used.

Following are deployment instructions for system administrators to get the software up and running.

Finally, the section on customizing and designing the application’s look provides web designers with an overview of the structure of the web pages and where to find the style sheet documents that control the application’s appearance.

# System architecture

The Cloud clustering web application is implemented as a Java EE 6 web application. It does not use all features available on the Java EE platform, but the most essential ones. Fortunately, the system requirements are quite simple and straightforward, as described below.

The system requirements are quite simple:

* PostgreSQL 9 RDBMS
* Apache Tomcat 8 Web Application container
  + Requires Java Runtime Environment, or a JDK (when running in debug mode)

The web application itself is deployed on Tomcat.

## Deployment scenarios

There are essentially only two reasonable scenarios – either works fine in a virtualized or containerized scenario. In fact, the entire development environment (except for the Eclipse IDE) is containerized on top of Linux LXC.

As far as the web application is concerned, both scenarios are supported with no preference for either over the other – choose the one that fits best in your environment.

### Scenario 1: All-in-one deployment

Straight-forward and most simple: PostgreSQL and Apache Tomcat are installed on the same host.

Database connections from Apache Tomcat are made from “localhost” only, which simplifies hardening database access. Easy to accomplish in an already virtualized infrastructure with minimal impact on other services deployed on the same infrastructure.

### Scenario 2: PostgreSQL and Tomcat on different hosts

This scenario sees the database server and the web application server deployed on different hosts, again virtualization/containerization being implementation details as far as the web application is concerned.

This scenario suits infrastructures with database and Tomcat services already being deployed for shared access by application services.

# Installation and deployment

Depending on the existing local infrastructure some services may not need installing, or configuration steps need to be adapted from the assumptions made in this document.

## Assumptions

The following assumptions were in place at the time of writing this document:

* Control over commissioning as many hosts as necessary
  + Bare metal, virtualised or containerized infrastructure
* Hosts commissioned with Debian or derivatives
  + Local systems are Linux containers on top of Ubuntu 14.04.3
* SSH access or console access to the commissioned hosts
* Root access to the commissioned hosts is required for software installation
  + PaaS scenarios have not been tested!
* DNS, DHCP etc. may be present, but are not a requirement
* Administrative access to PostgreSQL and Tomcat service instances

Apply the installation instructions contained herein to your local environment – do not slavishly follow them!

## Database commissioning

The web application uses Java Database Connectivity (JDBC) to access data in the database. With that, it is possible to substitute a MySQL database for the default PostgreSQL, or an Oracle instance, etc.

However, the correct drivers for the underlying database need to be bundled with the web application.

**In case you choose not to commission a PostgreSQL database, IMMEDIATELY contact me vie E-Mail (see above) so that I can update the deployment bundle with the corresponding driver!**

Installing and commissioning a database system are straightforward, so the instructions will not give specific commands, but describe what needs to be accomplished using the respective RDBMS. However, the instructions refer to the dev env environment services

### Installing PostgreSQL

1. **Log into the database host, and install a vanilla PostgresSQL 9 service.**  
   Consult the local OS’s manual, and the PostgreSQL manual for specific instructions.  
   Result:   
   A PostgreSQL instance is listening on port 5432 on the host, and is reachable.  
   Information:  
   – Note the IP address or DNS name of the host, as it is needed later on.  
   – Note the port on which the database is listening for connections.

### Configuring PostgreSQL

This step requires adding a PostgreSQL role to the database service, configuring host-based access, add a database catalog owned by the configured user.

1. **Log into the database host as root, and switch to user ‘postgres’.**Depending on your local environment, you may not be allowed direct remote root access. In that case, log in as a common user with admin rights, and switch to the root user afterwards.  
   Then switch to user ‘postgres’ (or the user underwhose id the PostgreSQL service is running). Consult the PostgreSQL manual for details.  
   Result:   
   The current command shell runs under the PostgreSQL’s default system user.
2. **Choose a database name, and a set of database role names.**To fully commission the database service, a database name, and several role names are needed for a database layout with basic security. While Postgres uses database access using roles only, they can be used to mimic database users and database groups as follows:  
   – Choose a database name, e.g. ‘clustering\_db’  
   – Choose a group role name, e.g. conveniently named ‘clustering’ (any name is good)  
   – Choose at least one user role, e.g. ‘webapp’.  
   – Optionally, choose additional user roles as required (see below), e.g. ‘michel’.  
   Information:  
   – Make a note of the database name  
   – Make note of the group role, and all the user roles for future use.
3. **Configure host-based access to the database service.**This step depends on the deployment scenario, i.e. whether database and web container are deployed on the same, or on different hosts. This document presumes different hosts with different IP addresses for the database and the web app container.  
   Further, it presumes that there will be two users allowed to connect to the database; one user is dedicated to the web application, and another user to direct database access from remote, e.g. for maintenance purposes. Upon login, either user role will inherit the associated group role’s permissions and access rights.  
   On linux systems, the PostgreSQL HBA configuration typically resides in “/etc/postgresql/9.3/main/pg\_hba.conf” where “9.3” denotes the Postgres database version. In pg\_hba.conf, scroll to the end of the file and add the following configuration snippet. Adapt to your choosing from step 3:  
   # TYPE DATABASE USER ADDRESS METHOD  
   hostssl clustering\_db webapp samenet md5  
   hostssl clustering\_db michel all md5  
     
   This configuration allows the db user ‘webapp’ to connect to database ‘clusteringdb’ from anywhere in the same subnet (i.e. Trust-IT premises), using encrypted database connections. It also allows a user ‘michel’ to connect from anywhere in the internet using encrypted connections to the same database. Both users will need to authenticate using their respective passwords, which are checked using md5 to avoid transferring clear-text passwords.  
   This configuration can be altered as follows:  
   – Substitute “samenet” with “127.0.0.1/32” for localhost connections only (deployment scenario 1)  
   Result:  
   – One user role is configured to log into the server and connect to a database from either localhost, the same network segment, or anywhere from the Internet.  
   – A second user role is configured to connect to the same database from anywhere on the internet (optional)
4. **Create database user and group roles, and the database**Connect to the database service using the ‘psql’ command, as user ‘postgres’.  
   Create the group role using the following command, adapting it to the role name chosen in step 3:  
   **“**create role clustering;”  
     
   Now create the user roles as follows. Adapt the command to the names chosen in step 3.  
   **“**create role webapp login password 'webapp’ inherit;   
   create role michel login password 'michel' inherit;  
   grant clustering to webapp;  
   grant clustering to michel;  
   **”**  
     
   Of course, choose better passwords! :-)  
     
   Finally, create the database and configure it to be owned by the group role (e.g. ‘clustering’):  
   **“**create database clustering\_db owner clustering;”  
     
   Note: You will not be able to verify this setup, unless:  
   – One of the created users happen to also exist as a local OS system user, or  
   – You configure PostgreSQL to allow local connections with MD5 authentication for both users.  
   Either is not covered in these instructions. You may, however, install the psql client (or any other suitable db client) elsewhere and test connections from there.  
     
   Result:  
   – A database group role exists.  
   – Two user roles exist, and both are members of the group role.  
   – A database owned by the group role exists.
5. **Create the db schema and load initial content**The installation bundle contains a zip file with four SQL scripts, named:  
   – schema\_create.sql  
   – schema\_drop.sql  
   – data\_insert.sql  
   – data\_delete.sql  
     
   Assuming you are still running the psql tool as user ‘postgres’, connect to the web app’s database:  
   **“**\c clustering\_db” Note the missing semicolon here!  
     
   Now copy and paste the contents of “schema\_create.sql” either line by line or in its entirety for execution. Do the same with “data\_insert.sql”.  
     
   Result:  
   The database schema and contents are initialized. Verify with typical “select \* from foo;” statements.

This concludes commissioning the database server.

## Application server commissioning

The Apache Tomcat 8 application server is designed so that it can be downloaded, extracted, and started from any existing directory in the local filesystem – can’t get easier than that! However, the installation directory of the application server needs to be made known also to the web ui designers.

This section presumes that the target host already has a suitable Java environment installed. Your host OS documentation provides information on how to do this.

1. Download the Apache Tomcat 8 from here: <http://tomcat.apache.org/download-80.cgi>
2. Transfer the ZIP file to the target host, and unpack in the target installation directory.
3. Cd into the installation directory, further referred to as $TOMCAT\_HOME
4. Cd further into $TOMCAT\_HOME/bin
5. If necessary, grant execution rights to the scripts appropriate to your environment, i.e. .bat files on Windows, and .sh files for Linux and Mac OS X.
6. Start Tomcat by invoking ./startup.sh (or similar for Windows)
7. Verify that you can access Tomcat at the address <http://host_ip:8080/>

Result:

Tomcat is up and running.

Information:

– Tomcat host’s IP address

– Tomcat’s installation directory $TOMCAT\_HOME

### OPTIONAL: Remove default web applications

Tomcat comes out of the box deployed with some web applications, e.g. an application server managing application and other things. To reduce the possible attack and exploit surface, you can remove all of these as they are not needed for this deployment.

1. Open a shell to Tomcat’s host.
2. Stop Tomcat:  
   $ cd $TOMCAT\_HOME/bin  
   $ ./shutdown.sh
3. Cd into Tomcat’s web application deployment directory  
   cd $TOMCAT\_HOME/webapps
4. Delete all contents   
   rm –rf \*
5. You may start Tomcat again, if you wish:  
   $ cd $TOMCAT\_HOME/bin  
   $ ./startup.sh

## Web Application commissioning

The installation bundle contains a “web application archive”, a .war file.

This file contains everything that is needed to deploy the clustering web application on the Tomcat Application container.

However, before the application can successfully run, its configuration needs to be adapted to the local environment.

### Configuring the web application

1. Download the .war file, and transfer it onto Tomcat’s host machine into an empty directory of your choice, e.g. “CW2ClusteringWebApp“. Cd into that directory in a shell.
2. Unpack the .war file:  
   $ unzip CW2ClusteringWebApp.war
3. Delete the WAR file as it is not needed anymore (WAR files can potentially be deployed as is)  
   $ rm CW2ClusteringWebApp.war
4. The file “persistence.xml” contains database connection information that needs to be changed:  
   $ cd ./WEB-INF/classes/META-INF  
   nano persistence.xml  
     
   Three properties defined in that file need to be changed as described below. These properties are prefixed with “javax.persistence.jdbc.”.  
   A fourth property, “javax.persistence.jdbc.Driver” MUST be left alone!
5. Edit the “.user” property to match the webapp’s database user role configured in step 5 (e.g. “webapp”)
6. Edit the “.password” property to contain the user’s password in cleartext.
7. Edit the “.url” property as fllows.  
   The URL has the format “jdbc:postgresql://<db\_host>:<port>/<db>”  
   – Substitute <db\_host> with the database host’s IP address or DNS name (step 1)  
   – Ditto for <port> (default 5432, step 1)  
   – Substitute <db> with the created database’s name (e.g. “clustering\_db”, step 5)
8. Using the defaults, the corresponding snippet in persistence.xml would look like this (presuming a database host address of 10.0.0.10):  
   <properties>  
    <property   
    name="javax.persistence.jdbc.url"   
    value="jdbc:postgresql://10.0.0.10:5432/clustering\_db"/>  
    <property   
    name="javax.persistence.jdbc.user"   
    value="webapp"/>  
    <property   
    name="javax.persistence.jdbc.password"  
    value="webapp"/>  
    <property   
    name="javax.persistence.jdbc.driver"  
    value="org.postgresql.Driver"/>  
   </properties>

### Deploying the web application

Configured like so, deploying the web application simply comprised of copying the directory where it was extracted and configured into Tomcat’s webapps directory:

1. Open a shell on the Tomtac host. Then execute the following commands:  
   $ cd CW2ClusteringWebApp  
   $ cd ..  
   $ cp –R CW2ClusteringWebApp $TOMCAT\_HOME/webapps  
     
   NOTE: Make sure that the directory, and not its content is copied!
2. If Tomcat is running, the application should be automatically picked up by Tomcat. Otherwise, start Tomcat as described in step 18.
3. Verify that everything went fine by opening the following URL in your browser:  
    http://<host>:8080/CW2ClusteringWebApp/index.xhtml

This concludes the installation instructions for system administrators.

If you made it until here, well done, congratulations!

## Providing access for UI designers

The final step is to decide on how to provide access to the design part for the UI designers. This can be done in a variety of ways depending on the local environment.

As far as the web application is concerned, all it requires is that the following CSS files are updated so that they are picked up by the application:

* about.css
* analysis.css
* login.css
* manage.css
* original.css
* scores.css
* template.css

All CSS files reside in one common directory:

$TOMCAT\_HOME/webapps/CW2ClusteringWebApp/resources/default/css

UI designers require access to these files to overwrite them regularly to test the outcome of their changes. How that is done remains your decision.

The following section(s) will provide information for Web UI designers.

# Customising the look of the web application

This section provides information for UI designers to customize the look and feel of the web application.

The web application makes heavy use of CSS v3 and applies classes and ids to UI elements the need styling.

However, it may be that not all elements have been properly tagged, or tags are missing or outright wrong and could be improved. In this very likely case, please provide feedback as to what you would need changing, and a next revision will hopefully implement your wishes.

However, to be able to design the UI, ask the system administrator for the following information:

* **Access URL for the web service.**This URL should be similar to   
   http://<host>:8080/CW2ClusteringWebApp/index.xhtml   
  where <host> may be a domain name or an IP address.
* **Username and password for the protected parts of the application.**The default values should be “user123” and “pass123” but these might have changed.
* **Access details to the CCS documents.**You will need to retrieve and upload a number of CSS files in order to develop the UI of the application.  
  How exactly this is done depends on the local environment; please consult with your

## Structure of the application

The application essentially consists of six different “pages” that allow the user to perform a number of actions.

Four pages can be accessed by clicking on one of the provided menu buttons, one page allows a user to log in before being allowed to manage the underlying data, and the sixth page is the “landing page” one can access via the “Cloud landscape clustering tool” title.

All pages are built using a template providing a header, the menu bar, a content area, and a footer. The CSS selectors have been spread over seven files, for each artefact one, i.e.:

* about.css
* analysis.css
* login.css
* manage.css
* original.css
* scores.css
* template.css

including one for the template.

To update the application, it simply suffices to update the CSS file on the server and reload the page; the application has been configured to invalidate the CSS pages after 1 second. A production version will change this behavior once the CSS styling is finished.

## Providing feedback and updated style sheets

Simply write me an E-Mail with your feedback, and attach any updated style sheet files.

I will then prepare a new version and send it back to the system administrator for re-deployment.