Cloud-TM

Specific Targeted Research Project (STReP)

Contract no. 257784

Companion document for deliverable D2.2: Preliminary Prototype of the RDSTM and RSS

Date of preparation: 10 June 2010
Start date of project: 1 June 2010
Duration: 36 Months

Contributors

Emmanuel Bernard, Red Hat
Joao Cachopo, INESC-ID
Mark Little, Red Hat
Francesco Quaglia, CINI
Paolo Romano, INESC-ID
Vittorio A. Ziparo, ALGORITHMICA
Manik Surtani, Red Hat
Sanne Grinovero, Red Hat
Fabio Cottefoglie, ALGORITHMICA

Table of Contents

1 Introduction

This document accompanies Deliverable D2.2, Preliminary Prototype of the Cloud-TM platform. Its aim is to overview the current architecture of the prototype and to document how to set it up in order to develop applications running on top of it.

As planned in the DoW this preliminary prototype does not include neither dynamic reconfiguration mechanisms, nor the Autonomic Manager which are going to be developed later on during the project. On the other hand, it contains already a preliminary version of some important building blocks of the Cloud-TM Data Platform, namely:

- Fénix Framework and Hibernate OGM, which are two alternative implementations of the Object Grid Mapper, based, respectively, on the Domain Modelling Language (DML) and on the Java Persistence API (JPA) approach;
- Infinispan, which serves as the Cloud-TM's Distributed Software Transactional Memory layer and allows persisting its state to a wide range of heterogeneous storage systems (ranging from local file systems/databases to Cloud-oriented storages).

We remind to Deliverable D2.1 for a detailed description of the architecture of these components, but, for self-containment, we provide in the following a brief overview of their main functionalities.

Note that, in order to simplify the development of the project's pilot applications, the Cloud-TM's Data Platform has been integrated with TorqueBox, a popular open-source framework for the development of web applications based on RubyOnRails 2 technology.

In order to demonstrate the prototype, and verify the successful integration of the different components of the Data Platform, we developed a simple demo, in the form a web application. that allows testing three basic operations: i) definition of transactions; simple retrieval of objects; create/read/update/delete (CRUD) operations.

Clearly the platform's API will undergo minor changes, as new functionalities and services are provided during the project. Nevertheless, the development of this demo has allowed also to identify integration issues at an early stage, and enabled Algorithmica to start developing the Cloud-TM's pilot applications even earlier than originally planned.

The structure of this document is the following. We start, in Section 2, by overviewing the architecture of the Cloud-TM Preliminary Prototype, and illustrating the key functionalities provided by each of its main components. Next, in Section 3, we describe the content of the package containing the platform's prototype, and provide instructions on how to install it and test it by running the demo application.

1.1 Relationship with other deliverables

The Cloud-TM Preliminary Prototype, which this document accompanies and illustrates, has been developed on the basis of the Cloud-TM architecture described in deliverable "D2.1 Architecture Specification Draft". As a consequence, the deliverable

depends also on the user requirements gathered in deliverable “D1.1 User requirements report”, and takes into account the technologies identified in the deliverable “D1.2 Enabling technologies report”.

2 Architectural Overview of the Cloud-TM Preliminary Prototype

The high level architecture of CLOUD-TM preliminary prototype is depicted in Figure 1. In the following we briefly overview the main modules composing the prototype, discussing what functionalities they currently support. An extensive description of the internal mechanisms that are already integrated in these modules, and of their planned developments, is provided in Deliverable D2.1.

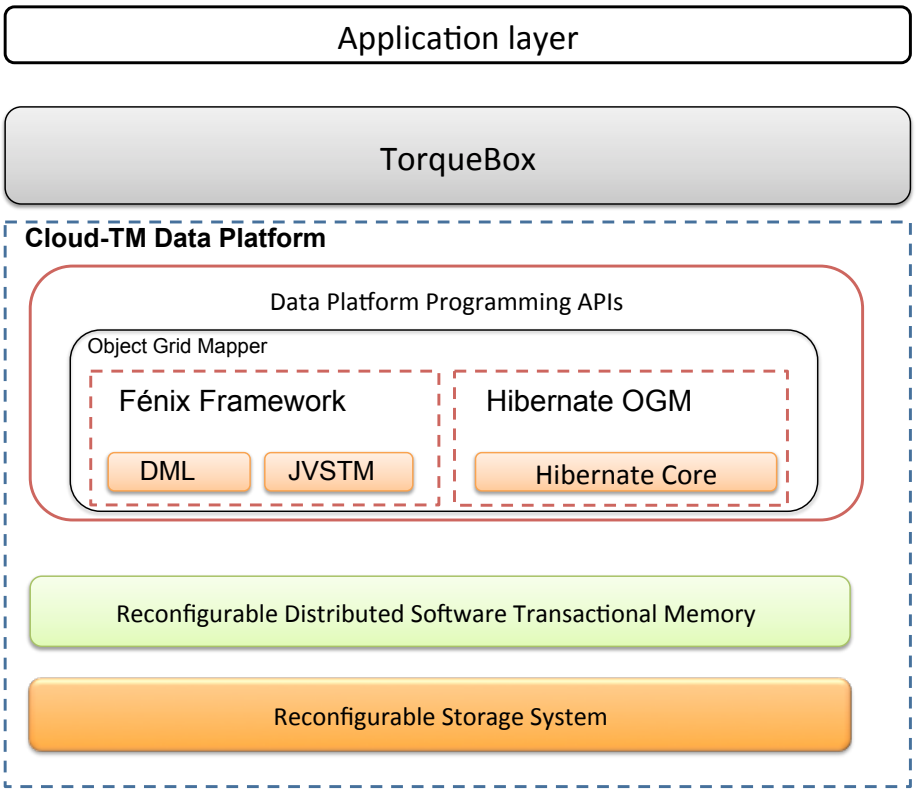

Figure 1: Architecture of the CLOUD-TM preliminary prototype

2.1 TorqueBox

As described in deliverable D6.1, "1st Periodic Management Report ", it has been decided to develop the pilots as web applications using Ruby on Rails (RoR), a popular open source web framework that allows to write powerful applications in a very elegant and compact way. In particular, it has been agreed to use the JBoss implementation of RoR, namely TorqueBox, which allows for deploying applications developed using RoR into the JBoss J2EE Application Server.

This required integrating TorqueBox programming model with the Cloud-TM's Data Platform, and specifically with the APIs provided by the Cloud-TM's Object Grid Mapper. In fact, RubyOnRails 2 (the version used by Algorithmica) already comes with its own domain modelling language, namely ActiveRecord, that relies on relational databases and is highly coupled to the framework. Nevertheless, shortly after the beginning of the project Rails 3 was released. This new version allows to replace the data model and provides a high level interface (ActiveModel) that allows to integrate into Rails different domain modelling languages. Thus, the work carried out up to date has allowed to use Rails 3 without ActiveRecord support allowing for a clean and easy integration with the Data Platform Programming APIs, and specifically with the Object Grid Mapper module, which is described in the following.

2.2 Object Grid Mapper

The *Object Grid Data Mapper* which is responsible for handling the mapping of object oriented data domain to the $< key, value >$ pair data model, which is employed by the underlying Reconfigurable Distributed Software Transactional Memory. Currently, the prototype is equipped both with the *Fénix Framework* and *Hibernate OGM*. The former is able to specify, through a Domain Model Language (DML), the structure of an object-oriented domain model and to support transactions by relying on the *Java Versioned Software Transactional Memory* module. The latter is capable of storing data in a NoSQL data grid exploiting the pre-existent *Hibernate Core*, which exposes standard Java Persistence API.

The internal architectures of these two modules are described in detail in Deliverable D2.2. In this document, therefore, we describe only how Fénix and OGM were integrated into TorqueBox.

Another issue addressed is that, for both Fénix and OGM interfaces, it is required to access a Java API from Ruby code. TorqueBox is based on JRuby[1], a pure Java implementation of the Ruby language. For this reason, it is possible to write (J)Ruby code that transparently uses Java APIs. In the following we describe how this feature has been used to access the Data Platform Programming API, within the context of a TorqueBox application.

Fénix Framework. In order to integrate Fénix into TorqueBox, we need to enable access from TorqueBox to the Fénix domain model. To this end, we added to the standard Rails directory structure the " lib/fenix" folder that contains all the Fénix libraries, configuration files and generated domain classes. When the application is deployed, a script copies these files into a TorqueBox Service Archive (SAR file) specifically created for the Fénix Framework. The SAR file is generally used for deploying a service component in the application server without dependencies on other components. When application server starts, the component will be deployed and will start running independently.

In " lib/fenix_loader.rb", we developed code that initializes the Fénix framework and that provides some shortcuts for accessing the Fénix public API that are then used

[1] `http://www.jruby.org/`

at runtime to define transactions and manipulate the domain model. Currently, within TorqueBox, we can define transactions, navigate and perform CRUD operations on the Fénix domain model.

Hibernate OGM At the time of writing, Hibernate OGM is a relatively less stable technology with respect to Fénix, which created additional issues to achieve integration with TorqueBox. For the time being, we have implemented a fairly simple, but fully working, interface that allows for specifying the domain model and the operations on the model at the Java level. Thus, it is currently not possible to define transactions at the JRuby level. Any transactional operation is embedded in a Java method that is then invoked from JRuby.

Ongoing work is primarily aimed at allowing a more fine-grained access to the Hibernate OGM API from JRuby and to span appropriately the Hibernate EntityManager (EM) lifecycle. Regarding the latter issue, currently we open one EM per persistence action. Despite a thorough performance testing is still to be done, this approach is deemed to be a performance anti-pattern for JPA. To this end, we are currently investigating the possibility to have the RoR app running on TB to interact via stateful session bean(s) encapsulating the JPA conversation.

2.3 Reconfigurable Distributed STM and Storage System

In the lower layers of the architecture lie the Reconfigurable Distributed Software Transactional Memory and the Reconfigurable Storage System.

Both these functionalities are provided, in this prototype, by *Infinispan* (v. 5.0 Pagoa). Infinispan is a in-memory transactional data grid, which meets several of the base requirements of the CLOUD-TM platform and throughout the following phases of the project will be enriched with self-tuning and autonomic reconfiguration mechanisms.

As described in the deliverable D2.1 Architecture Specification Draft, currently this version supports the following operational modes.

- **Standalone (non-clustered) mode:** in this mode, Infinispan acts as a Software Transactional Memory. Unlike classical STMs, however, Infinispan does not ensure serializability (or opacity), but, in order to maximize performance, it guarantees more relaxed consistency models. Specifically, it ensures the ISO/ANSI SQL isolation levels Read-committed and Repeatable-read isolation levels, in addition to what is called Write skew anomaly avoidance.

- **Full replication mode:** in this mode, Infinispan is deployed over a set of interconnected nodes, each one holding a replica of the same key set. This mode is mainly intended for small scale deployments, given that the need to propagate updates to all the nodes in the system hampers the global scalability of this operational mode. The replication mechanism employed by Infinispan is based on the usage of the classical two phase commit, which ensures that, in order for a transaction to be committed, it can acquire locks successfully across all the nodes in the platform (and thus ensuring replica-wide agreement on the transaction serialization order).

- **Partial replication mode:** in this mode every Infinispan replica stores a subset of the data globally managed by the system; every $\langle key, value \rangle$ pair is replicated (using a Consistent Hashing scheme to distribute keys among nodes) over a set of k nodes, thus leading to the possibility to meet availability and fault tolerance requirements without the drawbacks that are endemic to the full replicated mode. Upon the commit of a transaction, only the nodes which store keys affected by it are involved in the two-phase commit.

The availability of the above operational modes allows already for deploying, and preliminary testing, of applications both in private and public Clouds, also thanks to the availability of predefined configuration files supporting a wide range of deployment scenarios. The replication/distribution algorithms currently used at this stage are not adaptive, as adaptive, self-tuning functionalities will be developed in the following of the project.

In addition to serve as the initial RDSTM of the Cloud-TM platform, Infinispan supports also the abstraction of Reconfigurable Storage System: by means of a modular plugin-based layer, in fact, Infinispan can persist/retrieve data from a range of heterogeneous stable storages, including local file-systems and cloud-oriented storages. Also in this case, adaptive behaviors are not yet available of course, and will be provided in the following of the project.

3 Setting up the prototype

In this section we describe the content of the package and the necessary steps to set up

3.1 Structure and Content of the Package

The content of the package is structured as follows

```
cloudTM_preliminary_prototype
  |_src
        |_RDSTM_and_RSS
              |_infinispan-4.2.2
              |_infinispan-5.0.0CR4
        |_object_grid_mapper
              |_Fenix
              |_Hibernate_OGM
demo
  |_lib
        |_torquebox
              |_1.0.1
                    |_torquebox-dist-1.0.1-bin.zip
              |_2.x.incremental.95
                    |_torquebox-dist-2.x.incremental.95-bin.zip
  |_src
        |_fenix_demo
        |_ogm_demo
doc
  |_D2.2-CompanionDocument.pdf
```

- The *cloudTM_preliminary_prototype* folder contains the source code of the preliminary prototype of the Cloud-TM platform.
- The *demo* folder contains the source code of the demo application and the Torquebox binaries needed for its compilation and deployment.
- The *doc* folder contains this companion document.

3.2 Installing and running the prototype

In order to show how to setup the preliminary prototype of the Cloud-TM platform, and demonstrate its functioning, we have prepared a simple demo application that uses the two APIs currently provided by the OGM, namely Fenix Framework and Hibernate OGM.

The demo application is a dummy rails application intended for experimenting on the integration of the Hibernate OGM framework[2] with JRuby[3] and TorqueBox[4].

The application shows a web page with a draggable ball. When dragging the ball, the client fires asynchronous Ajax requests to the server, that update position of the

[2] http://community.jboss.org/en/hibernate/ogm
[3] http://www.jruby.org
[4] http://torquebox.org/

ball within a transaction. As there are many asynchronous requests, conflicts are often generated (and logged).

In the following we provide instructions on how to set-up the demo using first Hibernate OGM and, next, the Fénix Framework

3.2.1 Hibernate OGM

This application requires Torquebox 2.0 (http://torquebox.org) as Hibernate OGM has some dependencies with JBossAS 7. In order to run the application into a Jboss/-Torquebox application server follow these steps:

1. install TorqueBox v2.x.incremental.95
   (a) Get Torquebox binary from demo/lib/torquebox/2.x.incremental.95/torquebox-dist-2.x.incremental.95-bin.zip
   (b) Ensure you have Java 6

       $ java -version
       java version "1.6.0_07"
       Java(TM) SE Runtime Environment (build 1.6.0_07-b06-153)
       Java HotSpot(TM) 64-Bit Server VM (build 1.6.0_07-b06-57, mixed mode)

   (c) Unzip the binary distribution somewhere handy:

       $ unzip -q torquebox-dist-2.x.incremental.95-bin.zip

   (d) Set the following environment variables:

       $ export TORQUEBOX_HOME=your_folder/torquebox-2.x.incremental.95
       $ export JBOSS_HOME=$TORQUEBOX_HOME/jboss
       $ export JRUBY_HOME=$TORQUEBOX_HOME/jruby

2. Install the needed gem libraries: open a shell, cd to the project folder and run

   jruby -S bundle install

   If you don't have the bundler library in your system you must install it before the aforementioned command

   jruby -S gem install bundler

   Note: if you upgrade Torquebox remember to repeat step 3 after removing the Gemfile.lock file.

3. Deploy the application into TorqueBox by executing the following command in the application root folder:

   jruby -S rake torquebox:deploy

4. Run TorqueBox:

   jruby -S rake torquebox:run

5. Open the browser at http://localhost:8080/ogm_demo to see the demo.

3.2.2 Fenix Framework

In order to run the application into a Jboss/Torquebox application server follow these steps:

1. Install TorqueBox v1.0.1
   (a) Get Torquebox binary from demo/lib/torquebox/1.0.1/torquebox-dist-1.0.1-bin.zip
   (b) Ensure you have Java 6

       $ java -version
       java version "1.6.0_07"
       Java(TM) SE Runtime Environment (build 1.6.0_07-b06-153)
       Java HotSpot(TM) 64-Bit Server VM (build 1.6.0_07-b06-57, mixed mode)

   (c) Unzip the binary distribution somewhere handy:

       $ unzip -q torquebox-dist-1.0.1-bin.zip

   (d) Set the following environment variables:

       $ export TORQUEBOX_HOME=your_folder/torquebox-1.0.1
       $ export JBOSS_HOME=$TORQUEBOX_HOME/jboss
       $ export JBOSS_CONF=all
       $ export JRUBY_HOME=$TORQUEBOX_HOME/jruby
       $ export PATH=$JRUBY_HOME/bin:$PATH

2. Copy all contents of lib/fenix into jboss (requires $JBOSS_CONF to be set):

   jruby -S script/fenix_install.rb

3. Install needed gem libraries: open a shell, cd to the project folder and run

   jruby -S bundle install

   If you don't have the bundler library in your system you must install it before the aforementioned command

   jruby -S gem install bundler

4. Deploy the application into TorqueBox by executing this command in the project folder:

   RAILS_ENV=torquebox jruby -S rake torquebox:deploy

5. Run TorqueBox:

   jruby -S rake torquebox:run

6. Open the browser at http://localhost:8080/fenix_demo to see the demo.

4 Conclusions

In this document we have presented the Cloud-TM preliminary prototype of the RDSTMA and RSS components. The prototype is based on the integration of several components. At the core of the prototype we have Infinispan that represents a preliminary version of the RDSTM and of RSS, already implementing three operational modes but still with no support for adaptive and self-tuning features. On top of that, we have two interfaces of the Data Platform Programming API, namely a DML interface based on the Fénix Framework and a JPA interface based on Hibernate OGM. These components have been integrated in TorqueBox, a web application platform that will be used for the development of the pilot applications.

Finally, we provided instructions on how to install and set-up the prototype, as well as on how to test it by means of a demo application.