Ref. Ares(2013)129796 - 01/02/2013

Building the PaaS Cloud of the Future

Immigrant PaaS Technologies: Experimental Prototype of Software Components and Documentation

D7.3.2

Version 1.0

WP7 – Immigrant PaaS Technologies

Dissemination Level: Public

Lead Editor: Steve Strauch, University of Stuttgart

31/01/2013

Status: Final

The research leading to these results has received funding from the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 258862

Seventh Framework Programme

FP7-ICT-2009-5

Service and Software Architectures, Infrastructures and Engineering

This is a public deliverable that is provided to the community under a Creative Commons Attribution 3.0 Unported License: http://creativecommons.org/licenses/by/3.0/

You are free:

to Share — to copy, distribute and transmit the work

to Remix — to adapt the work

Under the following conditions:

Attribution — You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).

With the understanding that:

Waiver — Any of the above conditions can be waived if you get permission from the copyright holder.

Public Domain — Where the work or any of its elements is in the public domain under applicable law, that status is in no way affected by the license.

Other Rights — In no way are any of the following rights affected by the license:

Your fair dealing or fair use rights, or other applicable copyright exceptions and limitations;

The author's moral rights;

Rights other persons may have either in the work itself or in how the work is used, such as publicity or privacy rights.

Notice — For any reuse or distribution, you must make clear to others the license terms of this work. The best way to do this is with a link to this web page.

For a full description of the license legal terms, please refer to:

http://creativecommons.org/licenses/by/3.0/legalcode

Contributors:

Vasilios Andrikopoulos, University of Stuttgart
Nicolas Chabanoles, BONITASOFT
François Exertier, Bull
Rouven Krebs, SAP
Benoit Pelletier, Bull
Guillaume Porcher, Bull
Ricardo Jimenez-Peris, UPM
Simon Riggs, 2ndQ
Steve Strauch, University of Stuttgart
Kathryn Woodcock, 2ndQ

Internal Reviewer(s):

Craig Sheridan, FLEXIANT
Stéphane Carrié, FT
Johannes Wettinger, University of Stuttgart

Version History

Version	Date	Authors	Sections Affected
0.1	07/12/2012	Steve Strauch (USTUTT)	Initial version containing structure and planned content
0.2	08/12/2012	Steve Strauch (USTUTT)	Content regarding Extension of Apache ServiceMix and partner responsibilities have been added.
0.3	21/01/2013	Steve Strauch (USTUTT)	Integration of contributions from involved WP7 partners
0.4	25/01/2013	Steve Strauch (USTUTT)	Final fixes before 4CaaSt internal review
0.5	29/01/2013	Craig Sheridan (FLEXIANT), Stéphane Carrié (FT), Johannes Wettinger (USTUTT), Steve Strauch (USTUTT)	Integration of internal review feedback. Correction of grammar, spelling, and wording.
0.6	30/01/2013	Johannes Wettinger (USTUTT), Steve Strauch (USTUTT)	Preparation of final version and final fixes and corrections.

Copyright © USTUTT and other members of the 4CaaSt consortium 2013

1.0	31/01/2013	Vasilios Andrikopoulos (USTUTT), Johannes Wettinger (USTUTT), Steve Strauch (USTUTT)	Final version

Copyright © USTUTT and other members of the 4CaaSt consortium 2013

Table of Contents

Executive Summary 10
1. Introduction 11
1.1. Purpose and scope 11
1.2. Document Overview 11
2. Prototype Description 12
2.1. PostgreSQL and Repmgr 12
2.2. PgCloud 12
2.3. Java Open Application Server – JOnAS 14
2.4. Performance Isolation Benchmarking 16
2.4.1. Architecture of the SoPeCo 16
2.4.2. Architectural Overview of the Solution 17
2.4.3. Deployment 18
2.5. Bonita Open Solution – BOS 18
2.6. Extension of Apache ServiceMix for Multi-Tenant Aware Administration and Management 18
3. Components Management 21
3.1. PostgreSQL and Repmgr 21
3.1.1. Requirements 21
3.1.2. Installation 21
3.2. Installation Procedure of PgCloud in a VM with the Chef PgCloud Cookbook 24
3.3. Java Open Application Server – JOnAS 25
3.3.1. JOnAS Agent 25
3.3.2. JOnAS Multi-Tenant Aware Application Server 26
3.4. Performance Isolation Benchmarking 26
3.4.1. Implementing an own Measurement Environment Controller 27
3.4.2. Installation and Start of the Benchmark 27
3.5. Bonita Open Solution – BOS 27
3.6. Extension of Apache ServiceMix for Multi-Tenant Aware Administration and Management 28
3.6.1. Installation 29
3.6.2. Deployment 31
3.6.3. Build From Source Code 31
4. User Guide 32
4.1. PostgreSQL and Repmgr 32
4.1.1. Creating Some Sample Data 32
4.1.2. Setting up a Repmgr User 32
4.1.3. Clearing the PostgreSQL Installation on the Standby 33
4.1.4. Testing Remote Access to the Master 33

Copyright © USTUTT and other members of the 4CaaSt consortium 2013

4.1.5. Cloning the Standby 33
4.1.6. Setup Repmgr Configuration File 33
4.1.7. Registering the Master and Standby 34
4.1.8. Monitoring and Testing 34
4.1.9. Simulating the Failure of the Primary Server 35
4.1.10. Promoting the Standby to be the Primary 35
4.1.11. Bringing the former Primary up as a Standby 35
4.1.12. Restoring the Original Roles of Prime to Primary and Standby to Standby 35
4.2. PgCloud 36
4.3. Java Open Application Server – JOnAS 36
4.3.1. JOnAS Agent 36
4.3.2. Multi-Tenant Aware JOnAS 37
4.4. Performance Isolation Benchmarking 38
4.5. Bonita Open Solution – BOS 39
4.6. Extension of Apache ServiceMix for Multi-Tenant Aware Administration and Management 40
4.6.1. General Information 40
4.6.2. JBIMulti2 Web Service Interface 40
4.6.3. Access Configured Tenant Endpoint on SOAP MT Binding Component 41
4.6.4. Configure Hermes JMS to test JMS MT Binding Component 41
4.6.5. Configure Mozilla Thunderbird to test E-Mail MT Binding Component 41
4.7. FAQ 42
5. Conclusion 44
6. References 45

Copyright © USTUTT and other members of the 4CaaSt consortium 2013

List of Figures

Figure 1. Example Setup of Repmgr with three PostgreSQL Nodes. 12
Figure 2. PgCloud Overall Architecture 14
Figure 3. Overview of JOnAS Agent Services Used by Chef Scripts 15
Figure 4. Overview of JOnAS Extended for Multi-Tenancy 15
Figure 5. SoPeCo Architecture 16
Figure 6. SoPeCo Extensions 17
Figure 7. Overview of the Architecture of Apache ServiceMix Extended for Multi-Tenant Aware Communication and Multi-Tenant Aware Administration and Management [4]. . 19

Copyright © USTUTT and other members of the 4CaaSt consortium 2013

List of Tables

Table 1. API Operations for Asynchronous Task Management 36
Table 2. API Operations for JOnAS Instance Lifecycle Managment 37
Table 3. API Operations for Application Lifecycle Management 37
Table 4. Overview of Benchmark Configuration Properties 39

Copyright © USTUTT and other members of the 4CaaSt consortium 2013

Abbreviations

4CaaSt	Building the PaaS Cloud of the Future
API	Application Programming Interface
BC	Binding Component
BOS	Bonita Open Solution
EAR	Enterprise Archive
ID	Identifier
JAR	Java Archive
JDBC	Java Database Connectivity
JDK	Java Development Kit
JOnAS	Java Open Application Server
JPA	Java Persistence API
OS	Operation System
OSGi	Open Services Gateway initiative framework
PGXS	PostgreSQL Extension System
RDBMS	Relational Database Management System
Repmgr	Replication Manager for PostgreSQL clusters
REST	Representational State Transfer
RP	Reporting Period
SA	Service Assembly
SoPeCo	Software Performance Cockpit
SUT	System Under Test
WAR	Web application ARchive
URL	Uniform Resource Locator
UUID	Universally Unique Identifiers
WP	Work Package

Copyright © USTUTT and other members of the 4CaaSt consortium 2013

Executive Summary

The goal of the 4CaaSt project is to provide the PaaS Cloud of the future. The aim of work package 7 within 4CaaSt is to make proven immigrant platform technologies cloud-aware. Therefore, we are focusing on technologies stemming from the following four domains and which are reflected in the structure of the tasks accordingly: relational databases (Task 7.1), application servers (Task 7.2), composition frameworks and engines (Task 7.3), and integration technologies such as enterprise service bus (Task 7.4).

This deliverable and the corresponding prototypes provide the second version and thus extended and improved versions of the prototypical implementation of WP7 components and their documentation. We concentrate in this document on the delta compared to D7.3.1 [2] only and provide references to other deliverables and documents wherever possible.

In particular, in this document we provide the prototype descriptions (Section 2), installation and setup information (Section 3), as well as user guides and FAQ (Section 4) for the following immigrant PaaS technologies extended for Cloud-awareness in RP2:

- *Repmgr* – a Replication Manager for PostgreSQL clusters allowing monitoring and management of replicated PostgreSQL databases as a single cluster.
- *PgCloud* – a middleware for scalable database replication extended for replica discovery, fault-tolerance, replica fail-over, and elasticity.
- *Java Open Application Server (JOnAS)* – an open source Java EE and OSGi application server extended for enabling interaction via REST interface and multi-tenant service handling.
- *Performance Isolation Benchmarking* – tooling to measure performance isolation
- *Bonita Open Solution* – an open source BPM solution extended for multi-tenancy regarding data and configuration isolation.
- *Apache ServiceMix* – an open source Enterprise Service Bus extended for multi-tenant aware administration and management

Copyright © USTUTT and other members of the 4CaaSt consortium 2013

1. Introduction

This document is based on the former WP7 deliverables D7.3.1 [2] and D7.2.2 [3]. Deliverable D7.3.1 reported and provided the first version of the WP7 prototypes and documentation. Based on this we provided new requirements and corresponding design for extension and improvements in D7.2.2.

Thus, this document represents the documentation of the realization of the concrete requirements, specification, and design to achieve cloud-awareness of immigrant PaaS technologies focusing on the improvements and adaptations compared to the prototypes delivered with D7.3.1. The corresponding runtime artefacts and components of the prototypes are referenced within this document and delivered together with this document.

There will be one update of this document and the corresponding prototypes in project M39. This update will provide the prototypical implementation and documentation of the final version of the cloud-aware WP7 prototypes integrated into the 4CaaSt platform.

1.1. Purpose and Scope

This is the second version of the prototypical implementations and documentations of the immigrant PaaS technologies to be made cloud-aware in WP7. This document provides the descriptions of the prototypes, information on how to build, install, and setup each of the prototypes, and guidelines how to use each prototype including FAQ. Therefore, this document provides essential information enabling project internal usage, e.g., in WP8 concerning realization of 4CaaSt use cases, and project external usage of WP7 building blocks.

Moreover this document together with D7.2.2 build the basis for the final development and integration cycle further extending the current versions of WP7 prototypes and specifically focusing on finishing the integration of all WP7 building blocks into the 4CaaSt until the end of the project.

1.2. Document Overview

The remainder of this document is structured as follows: the prototype descriptions of the second version of WP7 components extended for Cloud-awareness is presented in Section 2. The sequence of the components presentation in this document follows the order of the WP7 tasks the corresponding components belong to. Information on how to build, install, and set up the prototypical implementations is provided in Section 3. A user guide for each prototype focusing on the description of the graphical user interface and the API depending whether the corresponding prototype provides both or only one of them, is contained in Section 4 in addition to FAQ. Finally, Section 5 concludes this document.

Copyright © USTUTT and other members of the 4CaaSt consortium 2013

2. Prototype Description

In this section we present an overview of the different WP7 prototypes focusing on the improvements and extensions that have been done during RP2.

2.1. PostgreSQL and Repmgr

Replication Manager for PostgreSQL clusters (Repmgr) allows monitoring and management of replicated PostgreSQL databases as a single cluster. Repmgr includes two components:

- repmgr: command-line tool that performs tasks and then exits
- repmgrd: management and monitoring daemon that watches the cluster and can automate remote actions.

Based on RP1, Repmgr has been enhanced including the following key features:

- Automatic failover configuration and execution of a user supplied script to manage complex situations, whilst still allowing for manual intervention where necessary
- Cleaner failover server identification
- Cluster clean-up of monitoring table or disabling the monitoring facility completely

For the purpose of demonstration, Figure 1 shows a Repmgr setup with three PostgreSQL nodes.

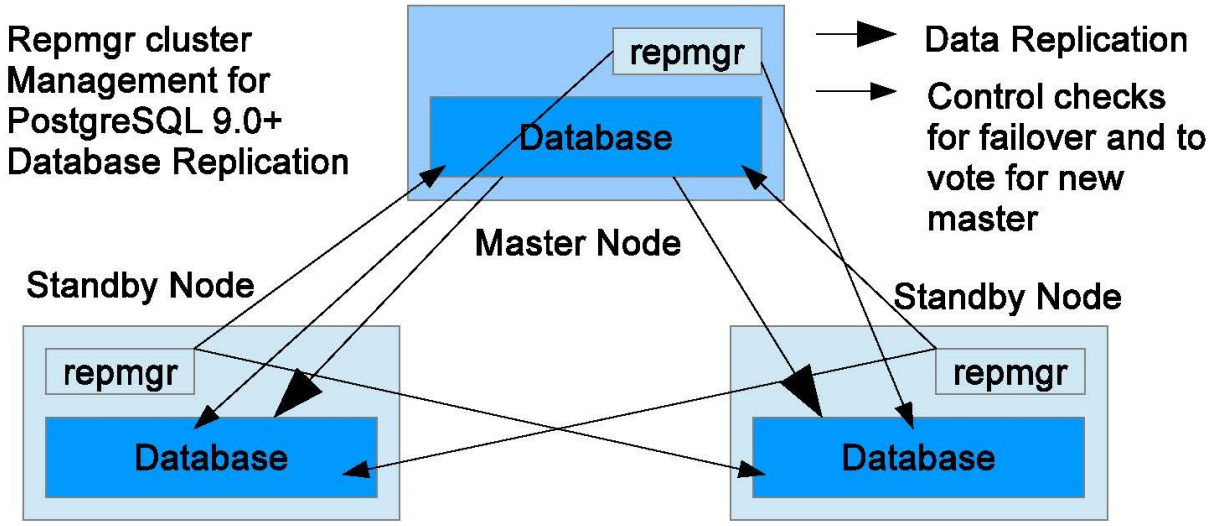

Figure 1. Example Setup of Repmgr with three PostgreSQL Nodes.

2.2. PgCloud

In the second period, the functionality of PgCloud has been extended and improved. These extensions and improvements are related to replica discovery, fault-tolerance, replica fail-over, and elasticity. Additionally, Chef scripts for the deployment of PgCloud on the 4CaaSt platform have been developed to automate its installation and deployment. The overall architecture is depicted in Figure 2.

PgCloud provides full transparency. A client connects to PgCloud through the PgCloud JDBC driver that encapsulates all the functionality required at the client side for replica management.

Replica discovery enables clients to connect to a replicated database without knowledge of the actual number of replicas neither their IP addresses. The JDBC driver uses IP multicast to discover dynamically the available replicas. Due to IP multicast being unreliable, we have developed a reliable discovery protocol that periodically retries to send discovery messages and can work consistently despite receiving only a fraction of answers from working replicas.

Copyright © USTUTT and other members of the 4CaaSt consortium 2013

The discovery protocol also obtains information about the load of the different replicas. In this way, the client can perform some load balancing at connection time. The clients select the replica to connect to by choosing one randomly with a probability inverse to the relative individual load of each one (e.g., with two replicas and a replica with 75% of the load and another replica with 25% of the load, the first one will be chosen with a probability 0.75 and the second with a probability 0.25).

When a client connects to a replica, a TCP connection is established between the client and the selected replica via the connection server. The connection server spawns a client session thread that reads from the TCP connection. On the client side the connection is maintained by the request dispatcher that submits requests from the client to the selected replica.

The client session thread forwards received SQL statements to the transaction manager. The transaction manager executes the requests using the DB session manager. The DB session manager maintains a pool of processes each of them with a connection to PostgreSQL instance. PosgreSQL requires each client to be located on a different process and it spawns a postmaster instance on a separate process for handling each DB connection. Therefore, for each database connection a pair of processes is maintained, a DB client process on PgCloud and a postmaster process on the PostgreSQL side. The PostgreSQL that we use has been extended with two services required to perform the replication in an efficient manner: one to obtain the updates performed by a transaction and another one to apply the update performed by a transaction. The DB session manager keeps a pool with a constant number of connections. In this way, the overhead of creating and destroying sessions is avoided which is pretty expensive due to the underlying creation of processes on PgCloud and PostgreSQL. Requests that cannot be executed wait queued till a DB client process becomes free when it terminates the transaction it was executing.

When a local transaction is completed, its writeset (the updated performed by it) is obtained by means of service implemented in PostgreSQL. If the writeset is null, this means that the transaction is read-only. If this is the case the transaction is committed and the reply sent back to the client thorough the corresponding client session thread. If it performs any update, then it is an update transaction. In this case, the writeset is multicast to all replicas by the update propagation manager using the group communication module. More concretely, it uses a total order reliable multicast that guarantees that all replicas deliver the messages in the order. When the writeset is received by a replica, a certification process is run by the certifier thread. It checks that the writeset does not overlap with any of the updates of any previous committed transaction that is concurrent to the one being committed. If the certification is successful, the transaction will be committed. If the transaction is local, then it is simply committed through the corresponding DB client process that was handling it. If the transaction is remote, then a new transaction is started and run at a DB client process that applies the received updates using the service implemented in PostgreSQL and commits the transaction. If the certification was unsuccessful, then the transaction is aborted in the case is local or ignored in the case it is remote. After committing or aborting a transaction, the local replica replies to the client with the transaction outcome through the client session thread.

Copyright © USTUTT and other members of the 4CaaSt consortium 2013

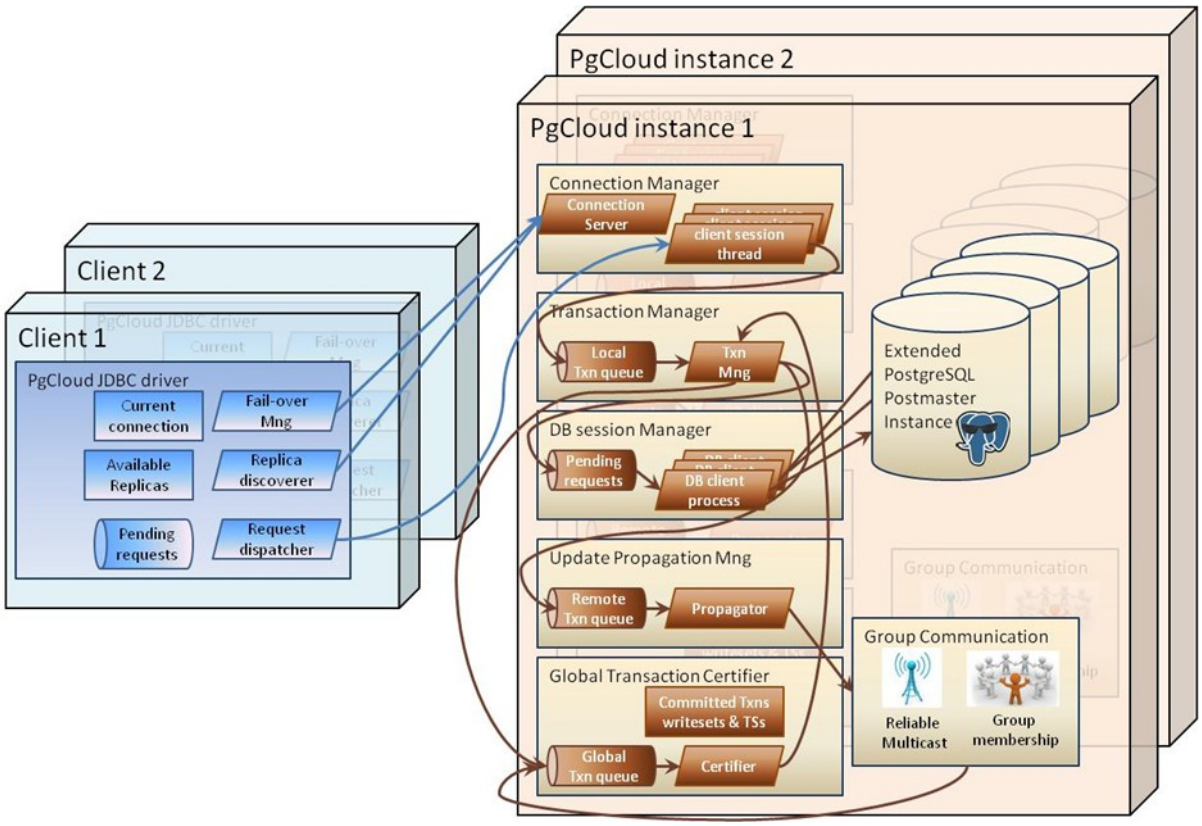

Figure 2. PgCloud Overall Architecture

In case of failures the state does not get lost because all replicas apply all the updates. On the client, side, however, the connection is lost to the failed replica. In this case, the fail-over manager checks whether the failure is actual. If not, it re-establishes the connection. If yes, then it tries to connect to one of the known replicas. If no known replica is available, it triggers again the discovery process. If there was a transaction that was not committed, the transaction has been implicitly aborted by the replica failure. If the transaction was in the process to be committed, that is, the client submitted the commit request, then it is checked whether the transaction was actually committed by the other replicas or not. If not, the transaction is aborted and the fact is notified to the client. If it was committed, this is communicated to the client as well.

For elasticity, monitoring of the computational resources available at each replica is performed and reported periodically to all other replicas by means of multicast. One replica, the first one in the view of the reliable multicast group, takes care of measuring whether the system is able to cope with the incoming load. If not, a new replica is provisioned. This replica does not have a database copy. For this reason, one of the replicas (we will name it recoverer) takes a snapshot of its current version of the database and send this snapshot to the new replica. After sending the snapshot, the recoverer forwards all the received writesets to the newly provisioned replica. This replica stores the writesets in a queue and when the snapshot is installed it starts applying the writesets. When the provisioned replica catches up and the queue of writesets is empty it finalizes the transfer protocol with a handshake in which it is determined, which is the last writeset to be received from the recoverer and which writesets will be received from the replicas as a regular replica. When this process is completed the new replica will be kept up-to-date through the regular update propagation process. The functionality that still needs to be added is the dynamic load balancing that will re-distribute the load across replicas and the process to determine when to scale-down.

2.3. Java Open Application Server – JOnAS

In RP2 we extended JOnAS with respect to two main aspects. We completed the JOnAS REST management agent interface to fulfil the operations of provisioning, control, and deployment. The JOnAS agent services can be used by the Chef script as illustrated in Figure 3.

Copyright © USTUTT and other members of the 4CaaSt consortium 2013

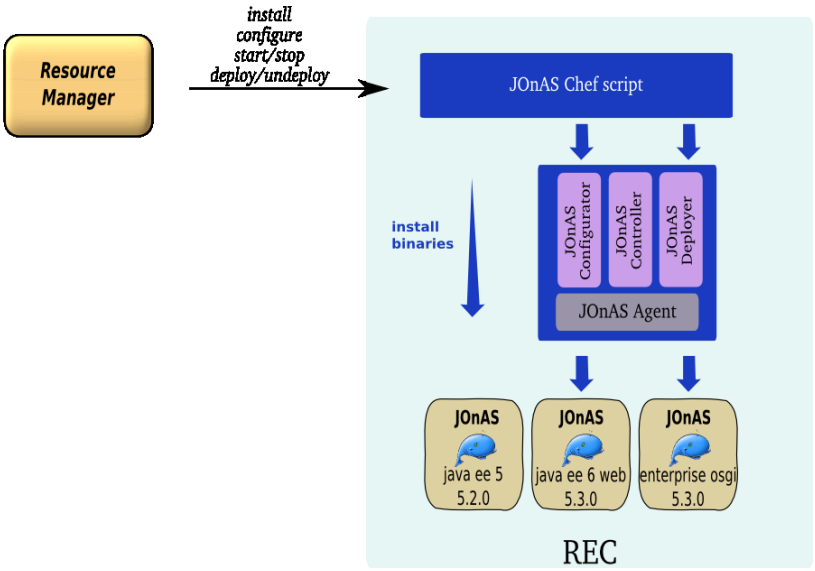

Figure 3. Overview of JOnAS Agent Services Used by Chef Scripts

Moreover, we added a new multi-tenant service handling a tenant context and enhancing the following services towards multi-tenant awareness: Persistence (JPA), Management (JMX), Registry (JNDI), and Logging. The tenant context is set in the meta-data of the Java EE archive and is identified in the URL. The persistence service is based on EclipseLink (Figure 4).

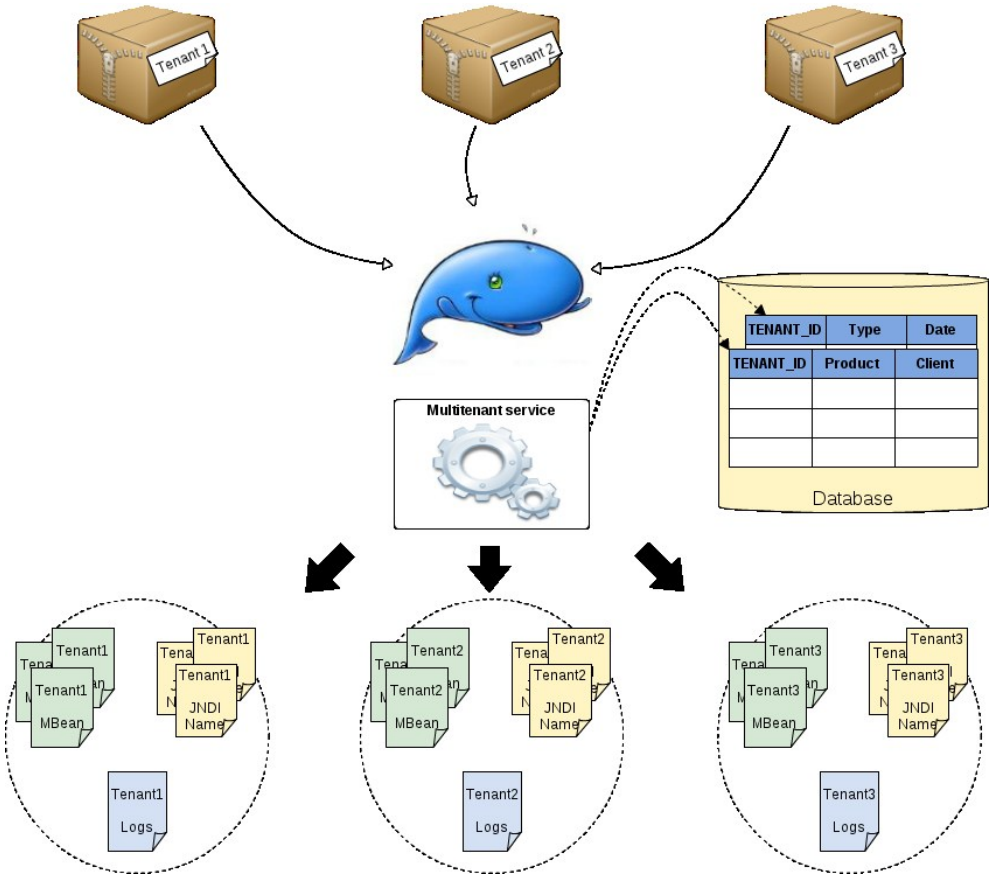

Figure 4. Overview of JOnAS Extended for Multi-Tenancy

The specification and design of the extensions are described in D7.2.2 [3].

Copyright © USTUTT and other members of the 4CaaSt consortium 2013

2.4. Performance Isolation Benchmarking

In this section a brief overview of the architecture is presented. The main goals for the concrete isolation benchmarks implementation were:

- Generically - it can be used for any system which have shared environments
- Decoupled - the framework is decoupled from the actual target systems
- Reusable - the framework allows repetition of the measurement scenarios with similar settings and reuse of existing components.

The actual implementation is based on the Software Performance Cockpit. The Software Performance Cockpit (SoPeCo) has a plug-in based architecture, which allows reusing functionalities (e.g., data storage, communication with the System Under Test) and conceptual elements. One of the conceptual requirements already fulfilled is the calculation of new workload configurations based on previous experiment results. This is an essential part to support the metrics we use to quantify performance isolation (cf. [10], and the Scientific Report to be published).

The SoPeCo has a generic meta-model configuration, which makes the framework decoupled from the System Under Test (SUT). The framework itself communicates with the target system via an abstract interface which allows communication with various systems. However, the SoPeCo had to be enhanced regarding several aspects because it did not support concrete Performance Isolation and shared systems according to our definition.

2.4.1. Architecture of the SoPeCo

Figure 5 describes the most important components of the Software Performance Cockpit. The architecture consists of eight major components. These components are explained in the following.

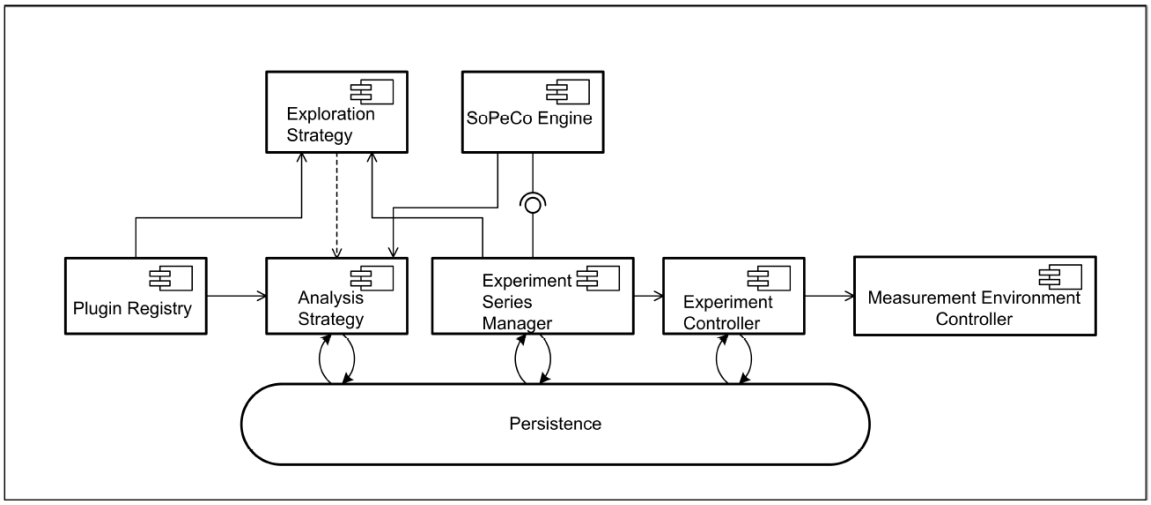

Figure 5. SoPeCo Architecture

The SoPeCo Engine reads the configuration file, which includes the model for the measurements configuration, loads all the required components to prepare the framework for the execution of measurements, initializes constant parameter values used to configure the system (parameters are derived by the configuration), and generates a set of experiment series according to the defined measurement scenario.

The Experiment Series Manager is responsible for the management and instantiation of components needed for executing an experiment series (i.e., a set of experiments).

An Exploration Strategy is an algorithm to define experiments that should run within one experiment series and follow a defined algorithm for the efficient measurement of a large parameter space.

Copyright © USTUTT and other members of the 4CaaSt consortium 2013

The Experiment Controller is the point where experiments are executed on the SUT. The experiments in an experiment series (which was defined by the Exploration Strategy) is executed by the Experiment Controller.

The Persistence is the central data store of the Software Performance Cockpit responsible for the persistence of the measured data.

The Analysis Strategy is responsible for analyzing measured data. Various statistical analysis methods are implemented in the Software Performance Cockpit framework. The selection of an appropriate statistical method depends on the measurement goals. Statistical analyses are basically mathematical functions that analyze sample datasets (measured data) to understand the dependency between independent parameters (e.g., workload) and dependent parameters (e.g., response time) and it creates a statistical model based on the observed dependency. The statistical model can be applied to predict results for unmeasured data ranges.

At the Plugin Registry all the extendable components (e.g., Exploration Strategy, Analysis Strategy) are made available for the framework. All the SoPeCo Extension Artifacts are saved in the Plugin Registry repository.

The Measurement Environment Controller is responsible for the communication between the Software Performance Cockpit framework and the SUT. It communicates with the Experiment Controller, executes the experiments on the SUT and gathers measured data. Thus it provides the control for environmental set-ups, controls the load generation and measures data. For each type of SUT at least one Measurement Environment Controller has to be implemented.

2.4.2. Architectural Overview of the Solution

This section describes the architecture of the additional implementation. There are three main components to be enhanced: The Measurement Environment Controller, the Exploration Strategy, and the Analysis Strategy.

The SoPeCo can be applied to a wide range of problems. However, this makes the usability for somebody who wants to apply it for a new scenario more complicated. As we will only apply it for the sake of performance isolation and shared systems we applied an adapter pattern for the three components. The purpose of these adapters is to convert the comprehensive SoPeCo information into a light weighted version specialized to our requirements. Figure 6 shows the high level view on the extensions where the focus is on the relevant aspects needed for implementing own system adapters.

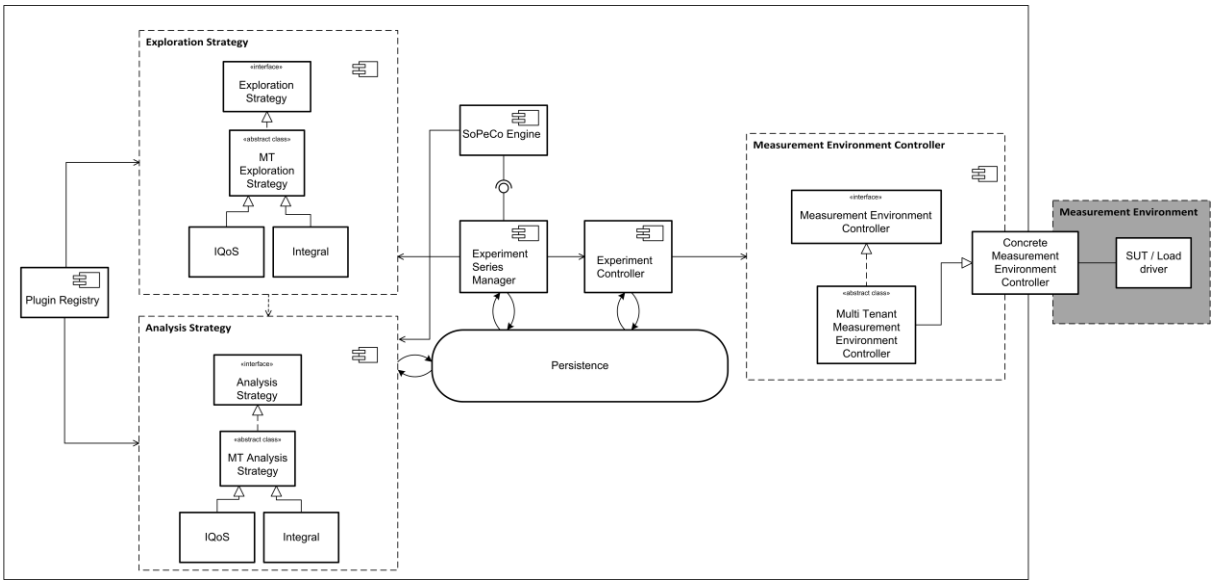

Figure 6. SoPeCo Extensions

Multi-Tenant Measurement Environment Controller: The MT Measurement Controller is the super class for the concrete measurement environment controller. It is the user exit to

adapt the framework for an own system. The superclass is the user's endpoint to the adapter enabling the SoPeCo to support Performance Isolation.

Exploration Strategy Extensions: Various metrics for measuring performance isolation are possible. Thus the MT Exploration Strategy also realizes the discussed Adapter to make the implementation of a specific one easier. At this time, two types are supported. The MT Analysis Strategy follows the same approach.

2.4.3. Deployment

In the current implementation everything has to be deployed on one single host and is run as one bundle. However, the concrete SUT should usually run on a separate system and the deployment of a concrete Load driver referenced by the specific implementation of the Measurement Environment Controller is usually also deployed on a separate machine to avoid performance influences.

2.5. Bonita Open Solution – BOS

Bonita Open Solution (BOS) has been extended in RP2 to support on demand multi-tenancy. A management module has been included to BOS to allow tenants lifecycle management. From now on a tenant can be created, enabled, disabled, or deleted. The multi-tenancy brings two main benefits:

- Data isolation at database level. The default strategy for data isolation is to use in PostgreSQL a discriminant column that let the system know to which tenant a value belongs to.
- Configuration isolation. Each tenant has its own configuration files. Every tenant can declare the implementation of each service it uses among a catalogue of pre-packaged service or even brings a new custom implementation.

BOS offers a Web based and a programmatic User Interface to administrate tenants and to interact with application such as the BonitaShop ticket booking application. The user interface consists in a WAR file deployed in a container. The container can be either Tomcat or JOnAS. All ticket booking orders are stored in the PostgreSQL database.

Every time the 4CaaSt platform requires a tenant creation, a call to BOS API is issued to create a tenant inside BOS. This action produces the two following significant deliverables:

- A new tenant is created in database to receive all business data in an isolated manner
- Configuration files are generated for the tenant based on a template.

Moreover in order to reduce resource consumption on a cloud environment the database schema has been updated. The size of columns of each table has been reviewed in order to select the data type that best fit the possible values and the functional requirements.

2.6. Extension of Apache ServiceMix for Multi-Tenant Aware Administration and Management

In RP2 we extended Apache ServiceMix (hereafter referred to as ServiceMix) [1] to support multi-tenant aware administration and management. In addition, we integrated these results with the previous results from RP1, i.e., the extension of Apache for multi-tenant aware communication. The information on how to setup, configure, and use multi-tenant aware communication is available in D7.3.1 [2]. The specification and design of the extension for multi-tenant aware administration and management is described in D7.2.2 [3]. Details on the integration of both approaches are described in [4].

Copyright © USTUTT and other members of the 4CaaSt consortium 2013

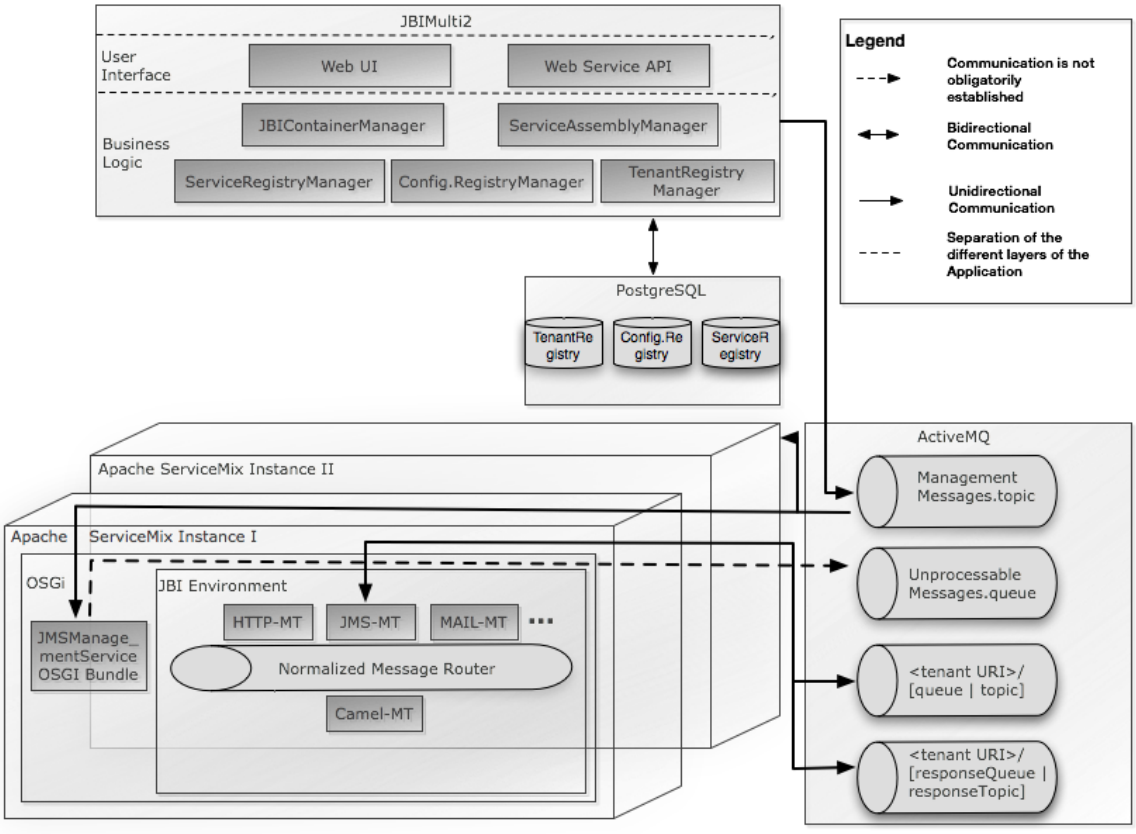

Figure 7. Overview of the Architecture of Apache ServiceMix Extended for Multi-Tenant Aware Communication and Multi-Tenant Aware Administration and Management [4].

An overview of the architecture of extended ServiceMix integrating the results from RP1 and RP2 is shown in Figure 7. The main components are the JBIMulti2 Web application, a PostgreSQL server with three databases, ActiveMQ is used as message broker, and the cluster container potentially several instances of the extended ServiceMix.

The JBIMulti2 Web application consists of a presentation layer containing a Web UI and a Web service API and the business logic layer implementing the managers for manipulating the underlying resources, i.e. the PostgreSQL databases or the ServiceMix instances via ActiveMQ. The PostgreSQL server hosts three databases for storing tenant information, configuration information and data on the services registered and integrated via ServiceMix in a tenant aware manner. As the JBIMulti2 Web application might manipulate more than one resource at a time and to ensure a consistent state between the resources distributed transactions are used. As the number of ServiceMix instances in the cluster might increase the number of participants in the distributed transaction increases and raises the complexity. Thus, we decoupled the different ServiceMix instances from the distributed transaction by using ActiveMQ as a message broker. Thus, the message broker is participating in the distributed transaction using reliable messaging and afterwards another distributed transaction is run between the message broker and the different ServiceMix instances involved in the transaction.

Additionally, ActiveMQ is used for integrating services via the ServiceMix using JMS. Therefore, a topic or a queue is created per tenant for the incoming as well as outgoing messages from ServiceMix.

The communication between JBIMulti2 and the multiple instances of ServiceMix is established by a unidirectional connection to a JMS topic each of the ESB instances subscribe to. In each of the ServiceMix instances JMSManagement OSGi bundle, which consumes the topic's management messages, which contain lifecycle instructions, e.g., install and uninstall SA, and the JBI BCs to deploy, as message payload. In case the deployment is not successful, the instruction message is stored in a queue of unprocessed messages. In case of successful deployment, the tenant-aware endpoint configuration is packed in a SA is installed in ServiceMix isolated from other tenants by modifying the SA to

Copyright © USTUTT and other members of the 4CaaSt consortium 2013

reference it to a specific tenant. This reference is realized by inserting tenant context in an XML file saved in each of the SAs, which contains the tenant user and tenant Universally Unique Identifier (UUID), and the tenant's URL. We use this information for authenticating the incoming tenant's requests to the ESB.

3. Components Management

This chapter contains information regarding build, deployment, configuration, and start-up of each of the prototypical implementations of WP7 cloud-aware building blocks developed and extended in RP two. Additionally, the links are provided, where the required software artefacts per component can be downloaded. The software artefacts are either publicly available, or they are available in the 4CaaSt project subversion repository. For the source code and binary of the performance isolation benchmark tooling the dissemination level is not decided so far. Thus, neither the link to the source code nor to the binary can be provided yet.

3.1. PostgreSQL and Repmgr

This section describes how to compile, deploy, and configure Repmgr. The source code, release notes, and further information required are available at: http://repmgr.org

3.1.1. Requirements

Repmgr works with PostgreSQL versions 9.0 and superior. Repmgr is currently aimed for installation on UNIX-like systems that include development tools such as *gcc* and *gmake*. It also requires that the *rsync utility* is available in the *PATH* of the user running the Repmgr programs. Some operations also require PostgreSQL components such as *pg_config* and *pg_ctl* to be in the *PATH*.

3.1.2. Installation

In order to install and use Repmgr and Repmgrd follow these steps:

1. Build Repmgr programs
2. Set up trusted copy between Postgres accounts, needed for the *STANDBY CLONE* step
3. Check your primary server is correctly configured
4. Write a suitable *repmgr.conf* for the node
5. Setup *repmgrd* to aid in failover transitions

3.1.2.1. Build Repmgr Programs

Both methods of installation will place the binaries at the same location as your Postgres binaries, such as *psql*. There are two ways to build it. The second requires a full PostgreSQL source code tree to install the program directly into. The first instead uses the PostgreSQL Extension System (PGXS) to install. For this method to work, you will need the *pg_config program* available in your *PATH*. In some distributions of PostgreSQL, this requires installing a separate development package in addition to the basic server software.

3.1.2.1.1. Build Repmgr Programs using PGXS

If you are using a packaged PostgreSQL build and have *pg_config* available, the package can be built and installed using PGXS instead following Listing 1:

```
tar xvzf repmgr-1.0.tar.gz
cd repmgr
make USE_PGXS=1
make USE_PGXS=1 install
```

Listing 1. Commands for Building and Installing Using PGXS

This is preferred to building from the *contrib* subdirectory of the main source code tree.

If you need to remove the source code temporary files from this directory, it can be done using the command shown in Listing 2.

```
make USE_PGXS=1 clean
```

Listing 2. How to Remove Source Code Temporary Files

In the following we provide information on building on RedHat Linux variants.

3.1.2.1.2. Build Repmgr Programs using Full Source Code Tree

In this method, the repmgr distribution is copied into the PostgreSQL source code tree, assumed to be at the *${postgresql_sources}* for this example. The resulting subdirectory must be named *contrib/repmgr*, without any version number, see Listing 3.

```
cp repmgr.tar.gz ${postgresql_sources}/contrib
cd ${postgresql_sources}/contrib
tar xvzf repmgr-1.0.tar.gz
cd repmgr
make
make install
```

Listing 3. Copying Repmgr Distribution into PostgreSQL Source Code Tree

If you need to remove the source code temporary files from this directory, it can be done as shown in Listing 4.

```
make clean
```

Listing 4. Remove Repmgr Source Code Temporary Files from this Directory

Notes on RedHat Linux, Fedora, and CentOS Builds

The RPM packages of PostgreSQL put *pg_config* into the *postgresql-devel package*, not the main server one. And if you have a RPM install of PostgreSQL 9.0, the entire PostgreSQL binary directory will not be in your *PATH* by default either. Individual utilities are made available via the alternatives mechanism, but not all commands will be wrapped that way. The files installed by Repmgr will certainly not be in the default *PATH* for the postgres user on such a system. They will instead be in */usr/pgsql-9.0/bin/* on this type of system.

When building Repmgr against a RPM packaged build, you may discover that some development packages are needed as well. The following build errors can occur, see Listing 5.

```
/usr/bin/ld: cannot find -lxslt
/usr/bin/ld: cannot find -lpam
```

Listing 5. Possible Errors During Build Process

Please install the following packages to correct those as shown in Listing 6.

```
yum install libxslt-devel
yum install pam-devel
```

Listing 6. Packages to be Installed to Overcome Build Errors

If you are building Repmgr as a regular user, then doing the install into the system directories using *sudo*, the syntax is hard. *pg_config* won't be in *root's path* either. The following recipe should work (Listing 7)

```
sudo PATH="/usr/pgsql-9.0/bin:$PATH" make USE_PGXS=1 install
```

Listing 7. Packages to be Installed to Overcome Build Errors

Copyright © USTUTT and other members of the 4CaaSt consortium 2013

3.1.2.2. Set up Trusted Copy Between Postgre Accounts

Initial copy between nodes uses the *rsync* program running over SSH. For this to work, the postgres accounts on each system need to be able to access files on their partner node without a password.

First generate a SSH key, using an empty passphrase, and copy the resulting keys and a matching authorization file to a privileged user on the other system as shown in Listing 8.

```
[postgres@node1]$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/var/lib/pgsql/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /var/lib/pgsql/.ssh/id_rsa.
Your public key has been saved in /var/lib/pgsql/.ssh/id_rsa.pub.
The key fingerprint is:
aa:bb:cc:dd:ee:ff:aa:11:22:33:44:55:66:77:88:99 postgres@db1.domain.com
[postgres@node1]$ cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
[postgres@node1]$ chmod go-rwx ~/.ssh/*
[postgres@node1]$ cd ~/.ssh
[postgres@node1]$ scp id_rsa.pub id_rsa authorized_keys user@node2:
```

Listing 8. Configuration of Authentication for Synchronization Between Different Nodes

Afterwards, please login as a user on the other system, and install the files into the Postgres user's account utilizing the commands in Listing 9.

```
[user@node2 ~]$ sudo mkdir -p ~postgres/.ssh
[user@node2 ~]$ sudo chown postgres.postgres ~postgres/.ssh
[user@node2 ~]$ sudo mv authorized_keys id_rsa.pub id_rsa ~postgres/.ssh
[user@node2 ~]$ sudo chmod -R go-rwx ~postgres/.ssh
```

Listing 9. Installing Required Files into Postgre User Account on Different Node

Now test that SSH in both directions works. You may have to accept some new known hosts in the process.

3.1.2.3. Confirm Software was Built Correctly

You should now find the Repmgr programs available in the subdirectory where the rest of your PostgreSQL installation is at. You can confirm the software is available by checking its version using the commands in Listing 10.

```
repmgr --version
repmgrd --version
```

Listing 10. Checking the Version of the Repmgr

You may need to include the full path of the binary instead, such as in the following RHEL example in Listing 11.

```
/usr/pgsql-9.0/bin/repmgr --version
/usr/pgsql-9.0/bin/repmgrd --version
```

Listing 11. Checking the Version of the Repmgr Using the Full Path

In case you are using Debian please use the following commands, Listing 12.

Copyright © USTUTT and other members of the 4CaaSt consortium 2013

```
/usr/lib/postgresql/9.0/bin/repmgr --version
/usr/lib/postgresql/9.0/bin/repmgrd --version
```

Listing 12. Checking the Version of the Repmgr Under Debian

Below this binary installation base directory is referred to as PGDIR.

3.1.2.4. Primary Server Configuration

PostgreSQL should have been previously built and installed on the system. Listing 13 provides a sample of changes to the *postgresql.conf* file:

```
listen_addresses='*'
wal_level = 'hot_standby'
archive_mode = on
archive_command = 'cd .'        # we can also use exit 0, anything that
                                # just does nothing
max_wal_senders = 10
wal_keep_segments = 5000        # 80 GB required on pg_xlog
hot_standby = on
/usr/lib/postgresql/9.0/bin/repmgrd --version
```

Listing 13. Primary Server Configuration

In addition to the configuration in Listing 13 you need to add the machines that will participate in the cluster in *pg_hba.conf* file. One possibility is to trust all connections from the replication users from all internal addresses, such as shown in Listing 14.

```
Host  all         all   192.168.1.0/24    trust
host  replication all   192.168.1.0/24    trust
```

Listing 14. Configuration of Machines Participating in the Cluster

A more secure setup adds a Repmgr user and database, just giving access to that user (Listing 15).

```
Host  repmgr      repmgr      192.168.1.0/24    trust
Host  replication all         192.168.1.0/24    trust
```

Listing 15. More Secure Configuration of Machines Participating in the Cluster

If you give a password to the user, you need to create a *.pgpass file* for them as well to allow automatic login. In this case you might use the MD5 authentication method instead of trust for the Repmgr user.

Please don't forget to restart the database server after making all these changes.

3.2. Installation Procedure of PgCloud in a VM with the Chef PgCloud Cookbook

PgCloud requires a 32bits Linux OS, so we have created a new VM following these steps:

1. Create a new VM instance in Flexiant Cloud using as disk image the Ubuntu 10.04 32 bits image. We are naming the new VM '*pgcloudrepl*', and we will use chefserverA as chef server.
2. Configure the new VM:

   2.1 Start the VM

2.2 */etc/hostname* replace the default ubuntu text with the VM name (suggestion: use the same node name set when creating the node in Flexiant cloud in the previous step *'pgcloudrepl'*)

2.3 */etc/hosts* add the node name and IP in the first line of the file

2.4 Create /etc/chef folder ($ *sudo mkdir /etc/chef*), and copy there the */etc/chef/validation.pem* file from the chefserverA node by using the scp command

2.5 Configure sudo so it can be called by ubuntu user without typing the password (change the */etc/sudoers* file as explained in http://www.davidverhasselt.com/2008/01/27/passwordlesssudo/)

3. Install chefclient in the new VM following the instructions in http://wiki.opscode.com/display/chef/Installing+Chef+Client+on+Ubuntu+or+Debian using the RubyGems option. Modify the */etc/chef/client.rb* file so the chefserverurl param points to chefserverA IP, that is, change line *#45* as follows:
   - *chef_server_url "http://109.231.67.236:4000"*
4. Now, there are two possible paths to follow:

   4.1 Registering the node in the chef server, with the recipes it must run:

   4.1.1 In chefserverA, as ubuntu user, run the following commands:

   $ knife node run_list add pgcloudrepl

   'recipe[install_pgcloud_cookbook::Deploy_PIC]'

   $ knife node run_list add pgcloudrepl

   'recipe[install_pgcloud_cookbook::Start_PIC]'

   4.1.2 Now, you can run $sudo chefclient from the new VM command line. This will run the recipes Deploy_PIC and Start_PIC of the install_pgcloud_cookbook.

   4.2 Running the chefclient straight ahead in the VM, setting the recipes to run using the “o” flag:

   $ sudo chefclient

   “install_pgcloud_cookbook::Deploy_PIC,

   install_pgcloud_cookbook::Start_PIC”

3.3. Java Open Application Server – JOnAS

This section describes how to compile, deploy, and configure both the JOnAS agent and the JOnAS multi-tenant service.

3.3.1. JOnAS Agent

In order to install JOnAS agent binary, select a location and un-compress it using the commands available in Listing 16.

```
cd <mydir>
wget
http://repository.ow2.org/nexus/content/repositories/releases/org/ow2/jonas
/agent/jonas-agent-assembly/5.3.0-M7/jonas-agent-assembly-5.3.0-M7-
bin.tar.gz
tar xvfz  jonas-agent-assembly-5.3.0-M7-bin.tar.gz
```

Listing 16. Download and Install JOnAS Agent Binary

In case you want to build it, please check out the source at OW2 gitorious: git://gitorious.ow2.org/ow2-jonas/jonas-agent.git and execute the Maven command provided in Listing 17.

```
mvn clean install
```

Listing 17. Launch the Build Process of JOnAS Agent Binary

3.3.2. JOnAS Multi-Tenant Aware Application Server

In order to install JOnAS version 5.3 latest binary, select a location and un-compress it using the commands available in Listing 18.

```
cd <mydir>
wget
http://repository.ow2.org/nexus/content/repositories/releases/org/ow2/jonas
/assemblies/profiles/legacy/jonas-full/5.3.0-M7/jonas-full-5.3.0-M7-
bin.tar.gz
tar xvfz  jonas-full-5.3.0-M7-bin.tar.gz
```

Listing 18. Download and Install Multi-Tenant Aware JOnAS

For the multi-tenant aware deployment please ensure that the multi-tenant service is enabled in the *$JONAS_ROOT/conf/jonas.properties* file by including the property *multitenant* as shown in Listing 19.

```
jonas.services  jtm,db,resource,ejb3,web,ear,depmonitor,multitenant
```

Listing 19. Enable Multi-Tenant Aware Deployment in Configuration File

In order to propagate the tenant ID to the database you have to use EclipseLink by setting the JPA implementation to EclipseLink 2.3 as shown in Listing 20.

```
jonas.service.ejb3.jpa.provider    eclipselink2.3
```

Listing 20. Configuring the JPA Implementation for EclipseLink

To enable logging of the tenant IDs add *%T* to the wanted handler (tty, logf, …) in *trace.properties* (Listing 21).

```
handler.tty.pattern  %T %d : %O{1}.%M : %m%n
```

Listing 21. Configuration for Tenant ID Monitoring

For futher information, please see the JOnAS configuration guide available at: http://jonas.ow2.org/JONAS_5_3_0_M7/doc/doc-en/html/configuration_guide.html

You can check out the source code available at OW2 SVN forge and launch the build process using the commands provided in Listing 22.

```
svn checkout svn://svn.forge.objectweb.org/svnroot/jonas jonas
cd jonas
mvn clean install
```

Listing 22. Launch the Build Process of Multi-Tenant Aware JOnAS

3.4. Performance Isolation Benchmarking

In this section we provide an overview on how to implement an own Measurement Environment Controller to quantify the isolation of a shared system. Furthermore, a description on how to install the benchmark, how to configure an Experiment Series and running the benchmark is provided.

3.4.1. Implementing an own Measurement Environment Controller

The Measurement Environment Controller is the connection to the actual benchmark for a specific system. To use our existing framework, which automates the measurement of Performance Isolation, one has to connect a system by implementing such a Controller. The runtime JAR file includes the abstract class `MultiTenantMeasurementEnvironmentController` in package `org.sopeco.engine.measurementenvironment.isolation` which has to be implemented for this reason. The following methods must be provided:

```
void initialize(ParameterCollection<ParameterValue<?>> initializationPVs) throws
RemoteException
```

This method initializes the measurement environment controller with a list of initialization values. These values will be configured in the description of the experiment series and contain information like IP addresses for load drivers etc. The method is called once before an experiment series. The implementation is thought to setup the system environment and to make it ready for the actual experiments.

```
void setObservationParameters(ParameterCollection<ParameterDefinition>
observationParameters)
```

This method sets the parameter that should be observed and returned after each execution of runExperiment. This is needed in cases where various parameters are under investigation.

```
Collection<ParameterValueList<?>> runExperimentOnSUT(TenantCollection<Tenant>
tenants, ExperimentTerminationCondition terminationCondition
```

This method runs a single experiment on the measurement environment. As a result, for every observation parameter a list of observed values is returned. These observed values correspond to the QoS metric of interest for calculating the isolation value. Usually it consists of the observed response times.

The input is a set of Tenant objects. It corresponds to a collection of input value assignments which specify the workload to be forwarded to the load driver. Furthermore it contains information like, tenant specific identifier for the SUT or the endpoints for a tenants requests.

The terminationCondition should be checked on a regular basis if its state is terminated. If this is the case the experiment has to be stopped.

3.4.2. Installation and Start of the Benchmark

Make sure, JDK1.7 is installed. To install the prototype download the jar file and save it on your local directory.

Make sure your implementation of the `MultiTenantMeasurementEnvironmentController` is available on the class path.

Make sure the listed jar files are also available on the classpath.

Run the jar file via java command line and add an additional parameter which points to the file describing your measurement setup as absolute path.

3.5. Bonita Open Solution – BOS

The installation procedure described in D7.3.1 [2] is still accurate as the improvements made on multi-tenancy are bundled in the same artefact as the other modules.

Nevertheless, we updated configuration files to make them cleaner and more readable. All configuration related to the platform has been gathered in Platform sub-directory. All tenant specific configuration has been gathered under a sub-directory which name is the ID of the tenant newly created.

In order to install the prototypes here are the required steps:

1. Download Bonita Open Solution version 5.9.1 bundled with Tomcat available at http://www.bonitasoft.com/products/BPM_downloads.

Copyright © USTUTT and other members of the 4CaaSt consortium 2013

2. Unzip BOS-Tomcat.zip bundle
3. Update the *bonita-platform.xml* file to point to your database server. (Optional as an embedded database server is also packaged in the zip)
    1. If you change default configuration you may need to add the JDBC drivers of the targeted DBMS into the *TOMCAT_HOME/lib* folder of the tomcat installation.
4. Start the tomcat server by using the *startup.(sh/bat)* script provided in the *TOMCAT_HOME/bin* folder.

Once Tomcat has been started please verify logs that the Bonita Web application has been started successfully. To do so you can open your Web browser to the following URL: http://localhost:8080/bonita and use the following login/password: install/install.

Multi-tenancy API is now available. It can be called through HTTP using any client, e.g. curl.

3.6. Extension of Apache ServiceMix for Multi-Tenant Aware Administration and Management

This section describes how to compile, deploy, and configure extended ServiceMix. The source code, artifacts, and further information required are available at:

https://svn.forge.morfeo-project.org/4caast/trunk/WP7/ExtendedApacheServiceMix/release-RP2/

In order to install the application, follow strictly the order of the configuration described in this section. The following middleware components are used for setting up the whole ServiceMix environment:

- *Apache Tomcat 7.0.23* hosts the ESBPerformanceEchoService.war and the ESBPerformanceEchoServiceWithResponse.war. These are used for testing individually the SOAP HTTP multi-tenant configured endpoints. The first is used for the performance evaluation.
- *JOnAS 5.2.2* hosts the JBIMulti2 management application, currently named jbimulti2-application-1.0.ear.
- *PostgreSQL 9.1.1* hosts the “serviceRegistry” database, “tenantRegistry” database, and “configurationRegistry” database. They are accessed by the JBIMulti2 management application.
- *Apache ServiceMix 4.3.0* wires the Taxi Scenario Web applications together. Furthermore, internally it runs an Apache ActiveMQ 5.3.1 messaging broker. Management messages from the JBIMulti2 management application are targeted to this message broker. The OSGi blueprint bundle jbi.servicemix.jmsmanagement-1.0.jar listens for these messages to perform JBI component installations and service assembly deployments.
- *Apache ActiveMQ 5.2.0* for the creation of the external queues and topics where the messages are delivered from ServiceMix.

The following paths are references in this section:

- **$jonas5.2.2$** - JOnAS 5.2.2 installation directory
- **$postgre$** - PostgreSQL **bin** directory
- **$servicemix$** - Apache ServiceMix installation directory
- **$tomcat$** - Apache Tomcat installation directory
- **$activemq$** - ActiveMQ 5.2.0 **bin** directory

Copyright © USTUTT and other members of the 4CaaSt consortium 2013

3.6.1. Installation

In the following we provide step-by-step guidance for installing the whole environment. As we run the whole environment on an Ubuntu VM in the FlexiScale infrastructure from Flexiant, we assume that the environment will be installed under Linux.

3.6.1.1. Initialization of PostgreSQL Databases

As there is no binary for Linux, it first has to be built. For this purpose, additional C libraries might be necessary (refer to the given errors).

According to the official manual of PostgreSQL, the user postgres should be created for security reasons. Then a database location and three logical databases can be created. Please check before whether PostgreSQL is started. If it is not started please start PostgreSQL and follow the instructions in Listing 23.

```
/* if everything is configured already in you VM you can directly follow
this steps using the password wpFzCAqb for the postgres user */
su - postgres
$postgre$/initdb -D /usr/local/pgsql/data /* if necessary*/
$postgre$/postgres -D /usr/local/pgsql/data

/* --- or --- */

/* to create a log file in the postgres home directory */
$postgres_home_dir$ /usr/local/pgsql/bin/pg_ctl start –l logfile –D
/usr/local/pgsql/data/

/* otherwise you have to perform the following steps first */
adduser postgres
mkdir /usr/local/pgsql/data
chown postgres /usr/local/pgsql/data
su -postgres
$postgre$/initdb -D /usr/local/pgsql/data
$postgre$/postgres -D /usr/local/pgsql/data
$postgre$/createdb tenantRegistry
$postgre$/createdb serviceRegistry
$postgre$/createdb configurationRegistry
```

Listing 23. Initialization of PostgreSQL databases

3.6.1.2. Configuration of JOnAS 5.2.2

Ensure that this instance of JOnAS is running on port 9000. It hosts the JBIMulti2 Application. The deployment is described in Section 3.6.2. Add the options shown in Listing 24 to the JOnAS executable in $jonas5.2.2$/bin/:

```
JONAS_OPTS="
...
-Deasybeans.dynamicinterceptors=true\
-Xms1024m\
```

```
-Xmx1024m\
-Dfile.encoding=UTF-8\
-server\
"
```

Listing 24. JOnAS configuration options

Change the following entry of $jonas5.2.2$/conf/carol.properties as shown in Listing 25 to prevent conflicts with Apache ServiceMix on same machine.

```
JONAS_OPTS=carol.jrmp.url:rmi://localhost:1097
```

Listing 25. JOnAS configuration adaptation to prevent conflicts with ServiceMix

Change the following entries of $jonas5.2.2$/conf/jonas.properties to configure database connections as shown in Listing 26.

```
jonas.services: jtm,db,resource,ejb3,jaxws,web,ear,depmonitor,dbm
jonas.service.dbm.datasources: TenantRegistry,ServiceRegistry,
ConfigurationRegistry
```

Listing 26. JOnAS configuration of database connections

Create three files named TenantRegistry.properties, ServiceRegistry.properties, and ConfigurationRegistry.properties in $jonas5.2.2$/conf/ analog to the file PostgreSQL1.properties with the following contents (Listing 27 shows an example for Tenant Registry):

```
...
datasource.name jbimulti2/tenantRegistry
datasource.url jdbc:postgresql://localhost:5432/tenantRegistry
datasource.classname org.postgresql.Driver
datasource.username postgres
datasource.password wpFzCAqb
datasource.mapper rdb.postgres
...
```

Listing 27. JOnAS - creation of data source property files

Put the PostgreSQL JDBC driver to $jonas5.2.2$/lib/ext as shown in Listing 28.

```
wget http://jdbc.postgresql.org/download/postgresql-9.1-901.jdbc3.jar
```

Listing 28. JOnAS – configuration of PostgreSQL JDBC driver

Start the JOnAS instance after copying the packages (for deployment) which are described in Section 3.6.2.

3.6.1.3. Initialization of Tomcat

No changes in the configuration of Tomcat are needed. Start the Tomcat server after copying the packages (for deployment) which are listed for Tomcat in the Section Deployment to the $tomcat$/webapps directory.

3.6.1.4. Initialization of ServiceMix

No changes in the configuration of ServiceMix are needed. Copy the packages (for deployment) listed in Section 3.6.2 for ServiceMix in the $servicemix$/deploy directory. Start then the ServiceMix server by running the command ./servicemix under the $servicemix$/bin directory. For performance optimization, the Java heap memory assigned to ServiceMix can be modified in the $servicemix$/bin/servicemix script. However, it is not necessary unless a bad performance is observed.

A list of the commands available in ServiceMix is available in the following URL: http://servicemix.apache.org/kernel/41-commands.html

3.6.1.5. Initialization of ActiveMQ

No changes in the configuration of ActiveMQ are needed. Start the ActiveMQ server. The ActiveMQ Web console can be accessed under the URL http://localhost:8161/admin/index.jsp.

3.6.2. Deployment

Put the JBIMulti2 management application into $jonas5.2.2$/deploy by using the command in Listing 29. Note: If the PostgreSQL databases (tenantRegistry, configurationRegistry, and serviceRegistry) were created from scratch in the system when following this manual, you have to build the JBIMulti2 application with the hibernate.hbm2ddl.auto value set to **create**, and deploy it. This happens because the schema in the registries must be created by the application. Once this is done (or if the databases were already created in the system), this value should be set to **validate**, stop JOnAS, copy the EAR to the $jonas5.2.2$/deploy directory and start. From this point use this last EAR if you don't want to loose the data in the database each time the EAR is redeployed (or JOnAS restarted). You can also use the built packages located in bin/ServiceMix_MTBCs/JBIMulti2/<createSchema|validateSchema>

```
deploy: jbimulti2-application-1.0.ear

deploy: console.war
```

Listing 29. JBIMulti2 deployment in JOnAS

Put the echo backend service for testing the multi-tenant SOAP over HTTP BC in the $tomcat$/webapps directory following the instructions in Listing 30.

```
deploy: ESBPerformanceEchoService.war

deploy: ESBPerformanceEchoServiceWithResponse.war
```

Listing 30. Deployment of echo backend service

The deployment of the tenant aware endpoint configuration ZIPs (the SAs which pack the SUs which contain the endpoint configuration), as well as the JBI BCs, are done by the user of a specific tenant through the JBIMulti2 application. We have packed different SAs for the different extended BCs, which pack the SUs containing the endpoint configuration stored in the bin/ServiceMix_MTBCs/Tenant_SAs.

3.6.3. Build From Source Code

All artefacts introduced in the scope of the JBIMulti2 project are contained in a single Maven project hierarchie. Build all artifacts by executing the following command in the root directory src/ with the following command in Listing 31.

```
mvn clean install -Dmaven.test.skip
```

Listing 31. Building all JBIMulti2 artefacts using Maven

To execute the unit tests of the business logic on a developer machine, run the PostgreSQL database, as described previously in Sections 3.6.1.1 and 3.6.2. Then call the mvn test command in the directory src/application/logic/accesslayer/. To create Eclipse projects from the Maven project hierarchy call the following command in the root directory src/ as shown in the following Listing 32.

```
mvn -DdownloadJavadocs=true -DdownloadSources=true

org.apache.maven.plugins:maveneclipse-plugin:2.6:clean

org.apache.maven.plugins:maven-eclipse-plugin:2.6:eclipse
```

Listing 32. Deployment of echo backend service

Copyright © USTUTT and other members of the 4CaaSt consortium 2013

4. User Guide

This chapter provides information and updates on the user guidelines provided in D7.3.1. Moreover, we provide in the FAQ solutions for well-known challenges when dealing with the WP7 prototypes in Section 4.7.

4.1. PostgreSQL and Repmgr

Full and complete details on how to use Repmgr are available at: http://repmgr.org

The following guidelines assume you've already followed the steps in "Installation Outline" to install Repmgr and Repmgrd on the system.

A normal production installation of Repmgr will usually involve two different systems running on the same port, typically the default of 5432, with both using files owned by the Postgres user account. This walkthrough assumes the following setup:

- A primary (master) server called "node1," running as the "postgres" user who is also the owner of the files. This server is operating on port 5432. This server will be known as "node1" in the cluster "test".
- A secondary (standby) server called "node2," running as the "postgres" user who is also the owner of the files. This server is operating on port 5432. This server will be known as "node2" in the cluster "test".
- Another standby server called "node3" with a similar configuration to "node2".

The Postgres installation in each of the above is defined as $PGDATA, which is represented here as */var/lib/pgsql/9.0/data*

4.1.1. Creating Some Sample Data

If you already have a database with useful data to replicate, you can skip this step and use it instead. But if you do not already have data in this cluster to replicate, you can create some like sample data following the instructions in Listing 33.

```
createdb pgbench

pgbench -i -s 10 pgbench
```

Listing 33. Creation of Sample Data

Examples below will use the database name *pgbench* to match this. Substitute the name of your database instead. Note that the standby nodes created here will include information for every database in the cluster, not just the specified one. Needing the database name is mainly for user authentication purposes.

4.1.2. Setting up a Repmgr User

Make sure that the "standby" user has a role in the database, "*pgbench*" in this case, and can login. For "node1" please execute the query as shown in Listing 34.

```
createuser --login --superuser repmgr
```

Listing 34. Creation of Repmgr User

Alternately you could start psql on the pgbench database on "node1" and at the node1b# prompt type, as shown in Listing 35.

```
createuser --login --superuser repmgr
```

Listing 35. Creation of Repmgr User Remotely

The main advantage of the latter is that you can do it remotely to any system you already have superuser access to.

4.1.3. Clearing the PostgreSQL Installation on the Standby

To setup a new streaming replica, start by removing any PostgreSQL installation on the existing standby nodes.

Stop any server on "node2" and "node3". You can confirm the database servers running using a command shown in Listing 36.

```
ps -eaf | grep postgres
```

Listing 36. Check for Running Database Servers

Now you can look for the various database server processes: server, logger, wal writer, and autovacuum launcher. Go to "node2" and "node3" database directories and remove the PostgreSQL installation using the commands as shown in Listing 37.

```
cd $PGDATA
rm -rf *
```

Listing 37. Remove PostgreSQL Installation

This will delete the entire database installation in */var/lib/pgsql/9.0/data*. Be careful that *$PGDATA* is defined here; executing *ls* to confirm you're in the right place is always a good idea before executing *rm*.

4.1.4. Testing Remote Access to the Master

On the "node2" server, first test that you can connect to "node1" the way Repmgr will by executing (Listing 38).

```
psql -h node1 -U repmgr -d pgbench
```

Listing 38. Testing Remote Access to the Master

4.1.5. Cloning the Standby

With "node1" server running, we want to use the clone standby command in Repmgr to copy over the entire PostgreSQL database cluster onto the "node2" server. Execute the clone process with (Listing 39).

```
repmgr -D $PGDATA -d pgbench -p 5432 -U repmgr -R postgres --verbose
standby clone node1
```

Listing 39. Starting the Clone Process in Repmgr

Here "*-U*" specifies the database user to connect to the master as, while "*-R*" specifies what user to run the *rsync* command as. Potentially you could leave out one or both of these, in situations where the user and/or role setup is the same on each node.

NOTE: you need to have *$PGDIR/bin* (where the PostgreSQL binaries are installed) in your *path* for the above to work. If you don't want that as a permanent setting, you can temporarily set it before running individual commands such as shown in Listing 40.

```
PATH=$PGDIR/bin:$PATH repmgr -D $PGDATA ...
```

Listing 40. Adapting the Path Variable

4.1.6. Setup Repmgr Configuration File

Create a directory to store each *repmgr* configuration in for each node. In that, there needs to be a *repmgr.conf* file for each node in the cluster. For each node we'll assume this is stored in */var/lib/pgsql/repmgr/repmgr.conf* following the standard directory structure of a RHEL system. It should contain the properties shown in Listing 41.

```
cluster=test
node=1
```

```
node_name=earth
conninfo='host=node1 user=repmgr dbname=pgbench'
```

Listing 41. Content of Repmgr Configuration File

On "node2" create the file */var/lib/pgsql/repmgr/repmgr.conf* with the content shown in Listing 42.

```
cluster=test
node=2
node_name=mars
conninfo='host=node2 user=repmgr dbname=pgbench'
```

Listing 42. Content of Repmgr Configuration File on Node2

The STANDBY CLONE process should have created a *recovery.conf* file on "node2" in the *$PGDATA* directory with the following content (Listing 43).

```
cluster=test
node=2
node_name=mars
conninfo='host=node2 user=repmgr dbname=pgbench'
```

Listing 43. Content of Repmgr Configuration File on Node2

4.1.7. Registering the Master and Standby

First, register the master by using the commands in Listing 44 on "node1".

```
repmgr -f /var/lib/pgsql/repmgr/repmgr.conf --verbose master register
```

Listing 44. Registering the Master

Then start the "standby" server. You could now register the standby by typing the command provided in Listing 45 on "node2".

```
repmgr -f /var/lib/pgsql/repmgr/repmgr.conf --verbose standby register
```

Listing 45. Registering the Standby

However, you can instead start Repmgrd (Listing 46), which will automatically register your standby system. And eventually you need Repmgrd running anyway, to save lag monitoring information. Repmgrd will log the deamon activity to the listed file.

```
repmgrd -f /var/lib/pgsql/repmgr/repmgr.conf --verbose >
/var/lib/pgsql/repmgr/repmgr.log 2>&1
```

Listing 46. Starting Repmgrd to Register the Standby Automatically

You can watch what it is doing by using the command in Listing 47.

```
tail -f /var/lib/pgsql/repmgr/repmgr.log
```

Listing 47. Check the Status During Auto-Registration of Standby

Hit control-C to exit this tail command when you are done.

4.1.8. Monitoring and Testing

At this point, you have a functioning primary on "node1" and a functioning standby server running on "node2". You can confirm the master knows about the standby, and that it is keeping it current, by looking at *repl_status*, see Listing 48.

```
postgres@node2 $ psql -x -d pgbench -c "SELECT * FROM
repmgr_test.repl_status"
```

```
-[ RECORD 1 ]-------------+-----------------------------
primary_node               | 1
standby_node               | 2
last_monitor_time          | 2011-02-23 08:19:39.791974-05
last_wal_primary_location | 0/1902D5E0
last_wal_standby_location | 0/1902D5E0
replication_lag            | 0 bytes
apply_lag                  | 0 bytes
time_lag                   | 00:26:13.30293
```

Listing 48. How to Check the Replication Status

Some tests you might do at this point include:

- Insert some records into the primary server here, confirm they appear very quickly (within milliseconds) on the standby, and that the *repl_status* view advances accordingly.
- Verify that you can run queries against the standby server, but cannot make insertions into the standby database.

4.1.9. Simulating the Failure of the Primary Server

To simulate the loss of the primary server, simply stop the "node1" server. At this point, the standby contains the database as it existed at the time of the "failure" of the primary server. If looking at *repl_status* on "node2", you should see the *time_lag* value increase the longer "node1" is down.

4.1.10. Promoting the Standby to be the Primary

Now you can promote the standby server to be the primary, to allow applications to read and write to the database again, by using the command in Listing 49.

```
repmgr -f /var/lib/pgsql/repmgr/repmgr.conf --verbose standby promote
```

Listing 49. How to Check the Replication Status

The server restarts and now has read/write ability.

4.1.11. Bringing the former Primary up as a Standby

To make the former primary act as a standby, which is necessary before restoring the original roles, use the commands in Listing 50 on node1.

```
repmgr -D $PGDATA -d pgbench -p 5432 -U repmgr -R postgres --verbose --
force standby clone node2
```

Listing 50. How to Check the Replication Status

Then start the "node1" server, which is now acting as a standby server. Make sure the record(s) inserted the earlier step are still available on the now standby (prime). Confirm the database on "node1" is read-only.

4.1.12. Restoring the Original Roles of Prime to Primary and Standby to Standby

Now restore to the original configuration by stopping "node2" (now acting as a primary), promoting "node1" again to be the primary server, then bringing up "node2" as a standby with a valid *recovery.conf* file. Stop the "node2" server using the command shown in Listing 51.

```
repmgr -f /var/lib/pgsql/repmgr/repmgr.conf standby promote
```

Listing 51. Start Node1 as Primary Again

Now the original primary, "node1" is acting again as primary. Start the "node2" server and type the command provided in Listing 52 on "node1".

```
repmgr standby clone --force -h node2 -p 5432 -U postgres -R postgres --
verbose
```

Listing 52. Start Node2 as Standby Again

Verify the roles have reversed by attempting to insert a record on "node2" and on "node1". The servers are now again acting as primary on "node1" and standby on "node2".

4.2. PgCloud

PgCloud offers a JDBC standard interface through its JDBC driver. All JDBC core v2.0 API operations (defined in the java.sql.Connection interface) are supported by the driver [6]. Pause and resume of transactions is not available. The isolation level supported is 'TRANSACTION_SERIALIZABLE'.

4.3. Java Open Application Server – JOnAS

This section provides a brief introduction on how to use the JOnAS agent as well as the Multi-Tenant Aware JOnAS.

4.3.1. JOnAS Agent

JOnAS agent is running on top of the JOnAS micro-server profile. To start it, run the command shown in Listing 53.

```
jonas start
```

Listing 53. Start JOnAS Agent

JOnAS agent provides a REST interface with the following operations. A subset of the API operations (long time operations) is executed asynchronously. The operations provided by the REST API can be categorized into three different groups: asynchronous task management, JOnAS instance lifecycle management, and application lifecycle management. An overview on each category is provided in the following sections.

4.3.1.1. Asynchronous Tasks Management

Table 1 provides an overview of the asynchronous operations offered by the API with respect to task management.

Operation	Request	Body	Response
Tasks list	GET /task		200 OK XML TaskList
Describe a task	GET /task/${taskid}		200 OK XML Task
Cancel a task	POST /task/${taskid}/action/cancel		200 OK

Table 1. API Operations for Asynchronous Task Management

4.3.1.2. JOnAS Instance Lifecycle Management

Table 2 provides an overview of the operations offered by the API with respect to JOnAS instance lifecycle management.

Operation	Request	Body	Response
Create new JOnAS instance	PUT /server/${servername}	topology.xml describing the configuration	202 Accepted XML Task
Start a JOnAS instance	POST /server/${servername}/action/start		202 Accepted XML Task
Stop a JOnAS instance	POST /server/${servername}/action/stop		202 Accepted XML Task
Get list of JOnAS instances	GET /server		200 OK XML ServerList
Describe a JOnAS instance	GET/server/${servername}		200 OK XML Server
Delete a JOnAS instance	DELETE /server/${servername}		204 No content

Table 2. API Operations for JOnAS Instance Lifecycle Managment

4.3.1.3. Application Lifecycle Management

Table 3 provides an overview of the operations offered by the API for application lifecycle management.

Operation	Request	Body	Response
Get list of applications	GET /server/${servername}/app		200 OK XML AppList
Describe an application	GET /server/${servername}/app/${appname}		200 OK XML App
Deploy an application	POST /server/${servername}/app/action/deploy	<media application>	202 Accepted XML Task
Undeploy an application	POST /server/${servername}/app/${appname}/action/undeploy		202 Accepted XML Task

Table 3. API Operations for Application Lifecycle Management

4.3.2. Multi-Tenant Aware JOnAS

Set the tenant-ID in the JOnAS specific deployment descriptor of your application. For example, for an EAR, put the tenant-ID in the *jonas-application.xml* file as shown in Listing 54.

Copyright © USTUTT and other members of the 4CaaSt consortium 2013

```
<jonas-application
xsi:schemaLocation="http://www.objectweb.org/jonas/ns
http://jonas.ow2.org/ns/jonas-application_5_3.xsd">
  <tenant-id>4seasons</tenant-id>
</jonas-application>
```

Listing 54. Setting the Tenant-ID for an Application to be Deployed in JOnAS

Once the application is deployed, it is available on the context root specified within the application with the tenant-ID prefix. For example, the url could be as specified in Listing 55.

```
http://localhost:9000/4seasons/javaee5-earsample
```

Listing 55. Sample for a Tenant-Specific Application URL

All the application data are prefixed with the tenant-ID:

- Log: the tenant-ID appears for each message.
- Registry: a tenant-ID prefix is set for each JNDI entry.
- Database: a column tenant-ID is added for each table.
- Mbeans: the tenant-ID appears in the Mbean name.

For a detailed user guide, refer to http://jonas.ow2.org/xwiki/bin/download/JPaaS/Multitenancy/multitenantguide.

4.4. Performance Isolation Benchmarking

The user can configure the benchmark with help of the configuration file referenced at the start of the benchmark environment. To configure the experiment series the referenced file has to follow the java properties format. The table shows an overview of the properties interpreted by the application. The result of the measurement is printed on the standard output of the operating system.

Property	Description	Type
`environment.controller`	The full qualified classname of the Measurement Environment Controller implementation	string
`environment.controller.*`	Implementation specific properties used to setup the Measurement Environment Controller. These parameters are directly transferred to the implementation. Possible information is user names for the SUT or the IP Address of the load driver, warm up times time for one measurement.	string
`analysis.metric`	The type of metric to be used for the calculation of the isolation value. At the moment only QoSAnalysis is supported. In the future also IntegralMetrics will be supported.	enum(QoSAnalysis, IntegralMetrics)
`tenant.*`	The wildcard identifies a concrete tenant using a system. There is no value directly assigned to this property. The value at the wildcard	numeric value starting with 1 and increases by 1

	is used to group various parameters to one tenant.	
`tenant.*.classification`	Classifies the tenant as a: 1. Disruptive on which increase the load to influence the performance of the others in a negative way. 2. An abiding one whose QoS are measured to calculate the influence.	enum(Abiding, Disruptive)
`tenant.*.workload.min`	abiding tenant: If the QoSAnalysis is used this value is ignored. In the IntegralMetrics case this is the lower bound of reduced workload before the measurement stops. disruptive tenant: For both metrics this value describes the reference workload for the disruptive tenant.	Integer
`tenant.*.workload.max`	abiding tenant: If the QoSAnalysis is used this value describes the workload for this tenant which is constant for the whole measurement. In the IntegralMetrics case this is the upper bound of the workload which corresponds to a perfect isolation. disruptive tenant: For both metrics this value describes the maximum disruptive load of the tenant	Integer
`tenant.*.workload.stepwidth`	Defines the step width used to adjust the workload during the measurement.	Integer
`tenant.*.id`	A technical identifier used by the Measurement Environment Controller to identify a tenant at the SUT	numeric value or string

Table 4. Overview of Benchmark Configuration Properties

4.5. Bonita Open Solution – BOS

As stated in D7.3.1 [2] the BOS Web User interface allows users to interact with the prototype in a manner depending on his/her profile. The system administrator can easily

manage in an aggregated view all the tenants currently deployed on the system. The end user has access to the forms of all the applications to interact with deployed applications.

In order to log on a particular tenant, a user must know its URL, e.g. http://localhost:8080/bonita/console/homepage?tenant=1 to access the tenant 1. A user belongs to a single tenant only, which means that a couple login/password is only valid on a single tenant. A user trying to access a tenant he/she does not belong to, will be rejected.

Once logged on a tenant, a user with some privileges can for instance

- Add users, groups, roles
- Re-organize the menus
- Change the look and feel of the portal
- Deploy new applications on the platform.

4.6. Extension of Apache ServiceMix for Multi-Tenant Aware Administration and Management

For utilizing the whole extended ServiceMix environment please refer to the SoapUI [7] projects provided in:

https://svn.forge.morfeo-project.org/4caast/trunk/WP7/ExtendedApacheServiceMix/release-RP2/miscellaneous/SoapUI_TestSuits

4.6.1. General Information

1. The service assemblies of the http-mt, jms-topic-mt, email-mt binding components are stored (maven generated zip and base 64 zip) in the bin/ServiceMix_MTBCs/JBI_BCs folder.
2. The service assemblies which will be deployed for testing the above binding components are stored (maven generated zip and base 64 zip) in the /bin/ServiceMix_MTBCs/Tenant_SAs/ for testing the extended BCs individually.
3. The service assemblies must be attached in the SOAP test project which contains the requests which are forwarded to the JBIMulti2, <TaxiScenario-soapui-project-remote-v1.1>. It is located in miscellaneous/SoapUI_TestSuits/JBIMulti2_WS-Interface.
4. Messages examples for SOAP, JMS and Email can be found in <messageExample.txt> in the miscellaneous/messageSamples folder, and in the SoapUI test suits located in miscellaneous/SoapUI_TestSuits.

4.6.2. JBIMulti2 Web Service Interface

Once the JBIMulti2 management application is deployed, three Web services are available. The corresponding WSDL files can be retrieved at the links shown in Listing 56.

```
http://<VM IP Address>:9000/jbimulti2WebService/SystemAdminService?wsdl
http://<VM IP Address>:9000/jbimulti2WebService/TenantAdminService?wsdl
http://<VM IP Address>:9000/jbimulti2WebService/TenantOperatorService?wsdl
```

Listing 56. Deployment of echo backend service

Refer to the TaxiScenario-soapui-project.xml SoapUI project for a sequence of example requests. They install JBI components, create two tenants, and finally deploy the required service assemblies to completely integrate the Taxi Scenario Web applications.

The SoapUI test case located in the miscellaneous/SoapUI_TestSuits/JBIMulti2_WS-Interface directory contains the SOAP management operations, e.g. create tenant, create user, deploy JBI component, deploy SA, etc.

Copyright © USTUTT and other members of the 4CaaSt consortium 2013

Note: The SOAP Messages in the SOAP Messages examples file should be modified by the requester with the values of their tenant, user and optional values. The WSDL file in the SOAP Provider should be adapted to the backend service that the user wants to invoke through the ESB. Furthermore, the IP address in the SoapUI test suite must be changed.

4.6.3. Access Configured Tenant Endpoint on SOAP MT Binding Component

The SOAP Binding Component can be accessed under the following multi-tenant URL: http://**<VM IP Address>**:8193/tenant-services/**<registered tenant URL>**/**<Service Name>**/**<Endpoint Name>**/main.wsdl

For example:

http://187.212.86.2:8193/tenant-services/taxicompany.example.org/httpSoapConsumer/TaxiProviderHttpSoapConsumerEndpoint/main.wsdl

4.6.4. Configure Hermes JMS to test JMS MT Binding Component

Set up values of Hermes JMS Session for sending message:

- Brokername = default
- Servicename = service:jmx:rmi:///jndi/rmi://<VM IP Address>:1900/jmxrmi
- Serviceurl = tcp://<VM IP Address>:61616

Set up values for topic:

- topicName = <tenant_uri_without_http>/ConsumerTopic (e.g. taxicompany.example.org/ConsumerTopic)
- queueName = <tenant_uri_without_http>/ConsumerQueue (e.g. taxicompany.example.org/ConsumerQueue)

In ServiceMix execute **activemq:list**.

The JMS MT TOPIC or QUEUE Provider will create one topic or queue per tenant, which is described by its URI. The tenant URI is appended to the ServiceMix default topic or queue specified name in the xbean file. To see the name execute in ServiceMix activemq:bstat –jmxlocal | grep taxicompany. This will show the queues or topics created with contains the value taxicompany. To see the full list of queues and topics, execute the same command without the grep.

A JMS Topic or Queue Consumer has to be created to consume the messages from the JMS providers. Note: the provider endpoint is configured to create a topic or queue in the ActiveMQ server which runs outside ServiceMix (on port 65000). The Provider Topic will look like the consumer topic name (named Provider instead of Consumer). It is always recommended to create (or just set the connection URI value) the backend queues in an external ActiveMQ server, in order to be able to see the message in the ActiveMQ Web console. In both cases, the external ActiveMQ server URI should be set in the xbean file.

Please refer to the <messageExample.txt> for message's examples.

4.6.5. Configure Mozilla Thunderbird to test E-Mail MT Binding Component

Two email accounts are created for both of the tenants (Taxi Company and Taxi Company B) for testing. Those are:

- User: taxiserviceprovider@gmail.com - Password: 4caast2012
- User: taxiserviceproviderb@gmail.com - Password: 4caast2012

Copyright © USTUTT and other members of the 4CaaSt consortium 2013

The email account where the BC listens should be modified in the <xbean.xml> file of the service unit.

The email BC Consumer will poll the emails from the account and create the normalized message. The email BC Provider will de-marshal and send the email to the "mailUserContact" key's value of the optional entry.

Refer to the <messageExample.txt> file for email content examples. To include the tenantContextKey into an e-mail, you need to set a custom header named "tenantContextKey". If you use Mozilla Thunderbird, read the following link to see how you can include a custom header:

http://email.about.com/od/mozillathunderbirdtips/qt/Add_an_Arbitrary_Custom_Header_to_Mozilla_Thunderbird.htm

4.7. FAQ

How can I manage an instance of Apache ServiceMix?

Apache ServiceMix provides a SSH interface at <VM IP Address>:8101. The default username is smx and the default password is also smx. Listing 57 shows a subset of provided commands:

```
osgi:list ---> shows all OSGi bundles
osgi:ls ---> shows all OSGi services
osgi:uninstall ---> uninstalls a OSGi bundle
osgi:start ---> starts an OSGi bundle
osgi:stop ---> stops an OSGi bundle
jbi:list ---> shows all JBI components, JBI shared libraries, and service assemblies
activemq:query --jmxlocal ---> shows attributes of components in ActiveMQ, such as topics or queues
activemq:purge --jmxlocal ---> deletes all messages in a queue
```

Listing 57. Useful Commands to Manage ServiceMix Instances

The JMS Management Service deployed on Apache ServiceMix seems not to retrieve any JMS messages, what is the problem?

Some problems occured, where the JMS Management Service OSGi bundle could not receive management messages. Firstly, the dead letter queue jbimulti2.queueUnprocessableMessages should be purged. If the problem still exists, restarting the OSGi bundle might help.

Why can't I access the PostgreSQL master node from the standby node in Repmgr?

Possible sources for a problem here include:

- Login role specified was not created on the master
- The database configuration on the master is not listening on a TCP/IP port. That could be because the *listen_addresses* parameter was not updated, or it was but the server wasn't restarted afterwards. You can test this on the master (node1) itself the way shown in Listing 58.

```
psql -h node1 -U repmgr -d pgbench
```

Listing 58. Checking Master Configuration

The "*-h*" parameter forces a connection over TCP/IP, rather than the default UNIX socket method.

Copyright © USTUTT and other members of the 4CaaSt consortium 2013

- There is a firewall setup that prevents incoming access to the PostgreSQL port (defaulting to 5432) used to access "node1". In this situation you would be able to connect to the "node1" server on itself, but not from any other host, and you'd just get a timeout when trying rather than a proper error message.
- The *pg_hba.conf* file does not list appropriate statements to allow this user to login. In this case you should connect to the server, but see an error message mentioning the *pg_hba.conf*.

How do I see the lag between master and standby servers within Repmgr?

Repmgrd helps monitor a set of master and standby servers. You can see which node is the current master, as well as how far behind each is from current. To activate the monitor capabilities of Repmgr you must include the option --monitoring-history when running it as shown in Listing 59.

```
repmgrd --monitoring-history --config-file=/path/to/repmgr.conf &
```

Listing 59. How to Monitor Master and Standby Servers

To look at the current lag between primary and each node listed in *repl_node*, consult the *repl_status view* using the following command (Listing 60).

```
psql -d postgres -c "SELECT * FROM repmgr_test.repl_status"
```

Listing 60. How to Inspect the lag Between Primary and Each Node

This view shows the latest monitor info from every node:

- *replication_lag*: in bytes. This is how far the latest xlog record we have received is from master.
- *apply_lag*: in bytes. This is how far the latest xlog record we have applied is from the latest record we have received.
- *time_lag*: in seconds. How many seconds behind the master is this node.

Copyright © USTUTT and other members of the 4CaaSt consortium 2013

5. Conclusion

This deliverable provides the documentation for the prototypical implementations of the immigrant PaaS technologies extended for cloud-awareness and integration into the 4CaaSt platform in WP7 during RP2 of the 4CaaSt project. We focus on providing information on the architectures of the prototypes (Section 2), component management including build, installation, setup and configuration (Section 3), and a user guide as well as FAQ for each of the prototypes (Section 4). The specification and design of the prototypes can be found in previous deliverable D7.2.2 [3].

Based on this prototype deliverable, we will work on deliverable D7.2.3 (due M39) to analyse the achievements of the second iteration of WP7 prototype development and identify the pending requirements in order to achieve the goal of Cloud-aware immigrant technologies until the end of the project.

The final version of this deliverable will be provided with D7.3.3 (M39). These will realize a final set of requirements to be fulfilled to enable Cloud-awareness for proven immigrant PaaS technologies. In particular we focus on finalizing the integration of the WP7 building blocks into the 4CaaST platform, which has been started in RP2 (e.g., integration with 4CaaSt deployment based on Chef, see Section 3.2). The goal is to enable seamless collaboration and interaction with other work packages to ensure coverage of the whole building block lifecycle including offering in the 4CaaSt marketplace, dynamic scalability and deployment, monitoring, as well as accounting and billing.

Copyright © USTUTT and other members of the 4CaaSt consortium 2013

6. References

[1] Apache Software Foundation: Apache ServiceMix. http://servicemix.apache.org.

[2] 4CaaSt Consortium: D7.3.1 Immigrant PaaS Technologies: Experimental Prototype of Software Components and Documentation, Version 1.0, January 30, 2012.

[3] 4CaaSt Consortium: D7.2.2 Immigrant PaaS Technologies: Components Design and Open Specification, Version 1.0, August 2, 2012.

[4] Gómez Sáez, Santiago: Integration of Different Aspects of Multi-Tenancy in an Open Source Enterprise Service Bus, Student Thesis No. 2394, Institute of Architecture of Application Systems, University of Stuttgart, 2013.

[5] FP7 Grant Agreement - Annex II General Conditions Grant Agreement No. 258862

[6] Oracle: JDBC core API, Version 2.0. http://www.oracle.com/technetwork/java/download-141179.html#spec.

[7] SmartBear Software. soapUI. http://www.soapui.org.

[8] Apache Software Foundation: Apache ServiceMix. http://servicemix.apache.org.

[9] Apache Software Foundation: Apache Camel. http://camel.apache.org.

[10] Rouven Krebs, Christof Momm, and Samuel Kounev. Metrics and Techniques for Quantifying Performance Isolation in Cloud Environments. Proceedings of the 8th ACM SIGSOFT International Conference on the Quality of Software Architectures (QoSA 2012)

All links have been last checked on January 31, 2013

Copyright © USTUTT and other members of the 4CaaSt consortium 2013