Abstract:

The goal of CloudScale is to aid service providers in analysing, predicting and resolving scalability issues, i.e., support scalable service engineering. The project extends existing and develops new solutions that support the handling of scalability problems of software-based services.

This deliverable presents the work performed during the second year within Work Package 5, which was to develop an open access application well suited to showcase the benefits of the tools and methods developed in the project. The showcase application is called CloudStore.

This document explains the realization of the cloud-enabled version of CloudStore, the definition of an expected usage evolution, and a measurement method for the capacity and elasticity of a system, as well as a set of tools for the deployment of cloud applications, and the distributed generation of load for them. CloudStore has gained recognition as an asset in the cloud community.

Dissemination level

<table>
<thead>
<tr>
<th>PU</th>
<th>Public</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>CO</td>
<td>Confidential, only for members of the consortium (including Commission Services)</td>
<td></td>
</tr>
</tbody>
</table>
CloudScale consortium

CloudScale (Contract No. FP7-317704) is a Collaboration Project (STREP) within the 7th Framework Programme, Call 8, Priority 1.2 (Cloud Computing). The consortium members are:

SINTEF ICT (SINTEF, Norway)
NO-7465 Trondheim, Norway
www.sintef.com

Project manager: Richard T. Sanders
richard.sanders@sintef.no
+47 930 58 954

Technical manager: Gunnar Brataas
gunnar.brataas@sintef.no
+47 472 36 938

SAP AG (SAP, Germany)
69190 Walldorf, Germany
www.sap.com

Contact: Roozbeh Farahbod
roozbeh.farahbod@sap.com

Ericsson Nikola Tesla (ENT, Croatia)
Krapinska 42
10000 Zagreb, Croatia
www.ericsson.com/hr

Contact: Darko Huljenic
darko.huljenic@ericsson.com

XLAB, Slovenia
Pot za Brdom 100
1000 Ljubljana, Slovenia
www.xlab.si

Contact: Jure Polutnik
jure.polutnik@xlab.si

University of Paderborn (UPB, Germany)
Zukunftsmile 1
33102 Paderborn, Germany
www.uni-paderborn.de

Contact: Steffen Becker
steffen.becker@uni-paderborn.de

Technische Universität Chemnitz (TUC, Germany)
Straße der Nationen 62
09107 Chemnitz, Germany
www.tu-chemnitz.de

Contact: Steffen Becker
steffen.becker@informatik.tu-chemnitz.de

1 UPB withdrew from the Consortium in October 2014, as the research team there relocated to TUC. TUC was not a partner before October 2014.
# Table of contents

CloudScale consortium.................................................................................................................. ii
Table of contents .......................................................................................................................... iii
List of figures ...................................................................................................................................... iv
Executive summary ........................................................................................................................ 1

1 Introduction ................................................................................................................................. 3
   1.1 CloudScale motivation and background ............................................................................. 3
   1.2 Summary ............................................................................................................................... 4
   1.3 Structure of this document ................................................................................................. 4
   1.4 Relationships with other deliverables ............................................................................... 4
   1.5 Contributors ....................................................................................................................... 4
   1.6 Acronyms and abbreviations ............................................................................................ 4

2 Work performed and definitions ............................................................................................... 5
   2.1 Progress and achievements in year two ............................................................................. 5
   2.2 CloudStore for scalability measurement .......................................................................... 5
   2.3 Metrics ................................................................................................................................. 6
   2.4 Usage evolution .................................................................................................................. 6

3 CloudStore ............................................................................................................................... 9
   3.1 Current implementations .................................................................................................... 9
   3.2 Deployment ...................................................................................................................... 10

4 Load generation ....................................................................................................................... 14
   4.1 JMeter scripts ................................................................................................................... 14
   4.2 Distributed JMeter ........................................................................................................... 16

5 Measurements results ............................................................................................................... 17
   5.1 Load Generation ............................................................................................................... 17
   5.2 Quality Thresholds .......................................................................................................... 17
   5.3 Work .................................................................................................................................. 18
   5.4 Configuration parameters .............................................................................................. 18
   5.5 Capacity ............................................................................................................................. 18
   5.6 Elasticity ............................................................................................................................ 20

6 CloudScale tools results .......................................................................................................... 23
   6.1 Dynamic Spotter on MT CloudStore ............................................................................... 23

7 Open repository and resources ............................................................................................... 25
   7.1 Distributed JMeter ........................................................................................................... 25
   7.2 Showcase .......................................................................................................................... 25
   7.3 Deployment Scripts .......................................................................................................... 26
   7.4 AWS ................................................................................................................................. 26
   7.5 OpenStack ....................................................................................................................... 27

8 Conclusion/Further work .......................................................................................................... 29

References ..................................................................................................................................... 30
List of figures

Figure 1: The results of CloudScale ................................................................. 3
Figure 2: Load evolution ............................................................................... 7
Figure 3: Threshold evolution ...................................................................... 8
Figure 4: CloudStores Meta Data Access Mechanism ................................. 10
Figure 5: Showcase deployment structure .................................................. 11
Figure 6: Choose provider page of web service for deploying showcase .......... 12
Figure 7: Configure frontend page of web service for deploying showcase ........ 13
Figure 8: Probabilities for different scenarios according to TPC-W specifications .......... 14
Figure 9: Example of linear JMeter scenario ................................................. 15
Figure 10: CloudScale JMeter scenario ....................................................... 16
Figure 11: JMeter scenario used during test ................................................. 17
Figure 12: SLO for 14 TPC-W operations ................................................... 17
Figure 13: JMeter scenario for elasticity ..................................................... 21
Figure 14: CPU Utilization for 1300 VU ...................................................... 21
Figure 15: CPU Utilization for 2500 VU ..................................................... 22
Figure 16: CPU Utilization for 3600 VU ..................................................... 22
Figure 17: Deployment of MT CloudStore and DS on Monsoon .................. 23
The main objective of work package 5 is to develop a showcase that lends itself well to demonstrate the benefits of the CloudScale tools and methods for (i.e. prediction, analysis, metrics, measurement, anti-patterns, etc.), offering a convincing story line to support dissemination and providing a platform for exploitation. The showcase aims to be realistic, while completely open, thus overcoming any issues that the use-case partners (SAP and ENT) have regarding confidentiality in connection with the evaluation work in WP4.

To this end, the project has developed a showcase we call CloudStore. This is a free and open scenario definition and implementation of a typical web application, with some inherent scalability and performance issues that demonstrate the merits of CloudScale. CloudStore also allows one to measure and compare different cloud providers, architectures and deployment in terms of capacity, elasticity and costs. CloudStore is based on the functional and non-functional specification of TPC-W\(^2\); giving it a different name than the well-known benchmark is intentional, i.e. to avoid confusion as to what it is.

The work performed during the second year of the Showcase work package focused on the CloudStore implementations, creating a distributed load platform, defining the cloud-related metrics we measured during this second year and their methodologies, and facilitating the usage of the Showcase as a tool for testing both the CloudScale tools and cloud applications and their deployments.

The CloudScale tools where applied to analyse CloudStore. The Showcase implementations were migrated to both private and public clouds, namely to OpenStack and Amazon’s cloud platforms, and used to measure and compare different deployment options. To that end a methodology of measurement for capacity and elasticity was defined, including a Usage Evolution schema that mimics the expected changes in the user load in real life of our on-line book sell application. Given the methodology defined to test these metrics, in conjunction with a “usage evolution” to be used as scenario, we performed the measurement steps to provide with a comparison of the different deployment setups for the CloudStore application.

A set of tools to facilitate this test were created, including a load generation that is distributed in order to produce any kind of volume given the necessary resources, a set of scripts for the deployment in both aforementioned private and public clouds, and a user interface for the flexible and rapid definition and instantiation of a 3-tier application on the cloud.

All these tools are made available through the project’s GitHub repository\(^3\), and include instructions of how to make use of them.

---

\(^2\) TPC-W (transactional web e-Commerce benchmark) is a web server and database performance benchmark, proposed by the Transaction Processing Performance Council, see [www.tpc.org/tpcw](https://www.tpc.org/tpcw)

\(^3\) [https://github.com/CloudScale-Project/Showcase](https://github.com/CloudScale-Project/Showcase)
1 Introduction

1.1 CloudScale motivation and background

Cloud providers theoretically offer their customers unlimited resources for their applications on an on-demand basis. However, scalability is not only determined by the available resources, but also by how the control and data flow of the application or service is designed and implemented. Implementations that do not consider their effects can either lead to low performance (under-provisioning, resulting in high response times or low throughput) or high costs (over-provisioning, caused by low utilisation of resources).

CloudScale provides an engineering approach for building scalable cloud applications and services. Our objectives are to:

1. Make cloud systems scalable by design so that they can exploit the elasticity of the cloud, as well as maintaining and also improving scalability during system evolution. At the same time, a minimum amount of computational resources shall be used.

2. Enable analysis of scalability of basic and composed services in the cloud.

3. Ensure industrial relevance and uptake of the CloudScale results so that scalability becomes less of a problem for cloud systems.

CloudScale enables the modelling of design alternatives and the analysis of their effect on scalability and cost. Best practices for scalability further guide the design process.

The engineering approach for scalable applications and services will enable small and medium enterprises as well as large players to fully benefit from the cloud paradigm by building scalable and cost-efficient applications and services based on state-of-the-art cloud technology. Furthermore, the engineering approach reduces risks as well as costs for companies newly entering the cloud market.
1.2 Summary

This document expresses the achievements in Work Package 5 reached during the second year of the project, with a brief mention to the state of the Showcases at the beginning of this period.

1.3 Structure of this document

The deliverable explains the CloudStore (the extended TPC-W specifications for scalability comparison) and its role within the project, the metrics that were used during year two, the usage scenarios that helped us standardize and perform those measurements, the measurement results, and the tools created to that end.

1.4 Relationships with other deliverables

The measurement methods (D1.2) and validation activities (D4.2) referred to in this document relate to the following deliverables:

- D1.2 Design support, second version: Defines the measurement methodology
- D5.1 First version of Showcase was the predecessor of D5.2, but differs greatly in structure and form due to recommendations from the first review. Essential parts are the same, however, in that D5.2 continues the Storyline defined in D5.1 into the Usage Evolution definition. New to D5.2 are the software implementation and deployment, which were developed following the plan for Phase 2 defined in D5.1, as well as the measurements on said elements. The results of using some of the CloudScale tools on the CloudStore code base are included here; however, due to events beyond our control a number of the results of tool use on CloudStore are not included in this document, as explained in the chapter 6. The background information on TCP-W is not repeated in D5.2; see D5.1 for this.

1.5 Contributors

The following partners have contributed to this deliverable:

- XLAB
- SINTEF
- SAP
- UPB
- ENT

1.6 Acronyms and abbreviations

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWS</td>
<td>Amazon Web Services</td>
<td></td>
</tr>
<tr>
<td>RDS</td>
<td>Amazon Relational Data Base</td>
<td></td>
</tr>
<tr>
<td>TPC</td>
<td>Transaction Processing Performance Council</td>
<td></td>
</tr>
<tr>
<td>MVC</td>
<td>Model View Controller Pattern</td>
<td></td>
</tr>
<tr>
<td>SOA</td>
<td>Software as a Service</td>
<td></td>
</tr>
<tr>
<td>VM</td>
<td>Virtual Machine</td>
<td></td>
</tr>
</tbody>
</table>
2 Work performed and definitions

2.1 Progress and achievements in year two

From the CloudStore versions produced in year one, we created a cloud version that was deployed and tested on both a private OpenStack cloud and the Amazon public cloud. The cloud-enabled application can be also set to scale automatically given the necessary platform features, such as Amazon’s Elastic Beanstalk. Additionally, a Multitenant version of the CloudStore was produced, though measurements and analysis on it will be performed during the third year, which will focus on cost.

In order to measure the properties of each CloudStore deployment we created a set of tools and definitions such as an engine for distributed load generation that follows the TPC-W definition for load generation, a usage evolution definition of how load shall vary at different time scales, and a standard for measuring the first two cloud metrics of the project: capacity and elasticity.

This measurement standard based on the usage evolution definitions were used measure the current implementations of the CloudStore application under different deployment environments in order to achieve the project’s first results in measuring benefits of different cloud providers and deployment architectures.

Finally, other achievements include the migration and documentation of the CloudStore applications and the load generation to the open GitHub repositories.

2.2 CloudStore for scalability measurement

As explained in D5.1, the TCP-W standard consists of a detailed specification for an on-line shop, together with non-functional requirements such as elements size and time responses, as well as expected user behaviour and loads.

Although the standard was designed for benchmarking implementations and deployments of the desired functionalities, we have taken its specifications to create a standard for the measurement of implementation and deployment scalability and elasticity of an application under the name of CloudStore.

Besides the functional requirements, we took the non-functional requirements, such as the percentage of errors and time responses, in order to produce a Service Level Agreement that must be respected by the implementation and deployment to be tested. This will provide a realistic scenario for evaluating the scalability, elasticity and implied costs of different cloud solutions using a simplified yet realistic example of an e-commerce web site.

In order to measure metrics related to the scalability and elasticity of the implementation, we define load variations that will benchmark the behaviour of the solutions upon varying loads, thus exposing important indicators such as the response speed, efficiency, and general cost of the varying number of services instances being created and destroyed.

The load changes, named Usage evolutions (see Section 2.4), can reflect variations at an hourly, daily or even longer scales, and can mimic the expected variations to the desired degree in order to compare costs for different implementation in order to choose the most cost-effective under the defined circumstances, or simply obtain a projection of the future infrastructure costs.
2.3 Metrics

We use the following metrics for measuring CloudStore:

- Capacity: This metric finds the maximum load with a given quality threshold and with a given amount of work. The metric described in more detail in Section 2.3.1 in D1.2. The evaluation plan for how to measure this metric is described in Chapter 3 in D1.2. This metric is the basis for the scalability with respect to cost metric described in Section 2.3.2.

- Number of SLO violations: This elasticity metric count the number of SLO violations during a given time interval and with a given load evolution. Both the quality thresholds and the work are also fixed. For more details see in D1.2, Section 2.3.4 for the metric and Chapter 4 for the evaluation plan for how to measure this metric.

2.4 Usage evolution

This section describes the long term (3 years) usage evolution of the CloudStore Showcase based on the bookstore storyline in D5.1. The usage evolution developed herein for the definition of the expected capability and scalability of the CloudStore will also serve the cost analysis of the system in year three.

We first describe some assumptions before we consider the evolution of load for the CloudStore showcase during the three-year duration of the showcase. Afterwards, the evolution in terms of work is described, before we elaborate on the evolution in terms of quality thresholds.

2.4.1 Assumptions

We start with these assumptions:

- We make a distinction between the real time scale of this CloudStore scenario and the time scale when actually running this scenario. We use the following conversion factor: 1 year in real life corresponds to 1 day when running the scenario.

- Using Usage Evolution, we are able to model the following changes in load, work and quality thresholds:
  - Stable: unchanged. Described by two parameters, the level and the duration.
  - Linear: with a certain slope. Described by three parameters, the current level, the slope and the duration of the slope. A very fast slope will be quite similar to a step function. However, a real step function is not possible to produce in practice. We will always have a more or less steep increase.
  - Exponential: in our case we use the special kind of exponential case where we have an accumulative increase for each time unit, e.g. we can multiply the load by 1.1 each month: for the second month we will then have a value 1.21 and after one year we will have the value $1.1^{12} = 3.14$. Described by four parameters: the start value, the increase per time unit, the time unit used and the duration of the exponential increase.

- What we are able to model will then determine how the load, work and quality threshold evolution can look like.

- We also take the following cloud limitations into account:
  - AWS/EC2 requires approximately 3 minutes to initialize new resources
  - The billing period is 1 hour, which means we cannot distinguish cost on a smaller time scale. This has implications for how short we can make the actual scenarios.
  - As SLO for the CloudStore operations, we will base ourselves on the 90 percentile response times with the limits from the TPC-W specification.
2.4.2 Load evolution

We use the following assumptions for the load evolution:

- We use only one parameter to represent load evolution. This parameter represents the number of CloudStore users in the system at a given point in time for the three years. As defined in the TPC-W specifications, at any given time some of these users will be actively interacting with the CloudStore, while other users will be idle.
- We indicate load levels using four variables $A$, $B$, $C$ and $D$.

The load varies as follows:

- In the first semester we start with load $A$, and then have a linear increase to load $B$.
- After that semester we start with load $B$ and have an exponential increase for one year until we reach load $C$ in the middle of the second year.
- For the rest of the second year we have a linear increase from $C$ to $D$.
- In the third year the load is stable at $D$.

This is depicted in Figure 2. Note that this figure does not take into account yearly variations like Christmas peaks, summer vacation load, nor does it take weekly variations into account.

![Figure 2: Load evolution](image)

2.4.3 Work evolution

We have two work parameters:

- Number of books
- Number of customers

The evolution of both these work parameters corresponds to the load evolution in Figure 2. The only differences are the values of the parameters $A$, $B$, $C$ and $D$. These values must be set individually for each of the two work parameters.
2.4.4 Threshold evolution

We have the following assumptions for the evolution of quality thresholds:

- This threshold is measured in terms of the total weighted average response time for the different CloudStore operations. Therefore it is only measured in terms of one single number.

This quality threshold is stable with the value of $E$ seconds from the start until the start of the second year. The quality threshold linearly decrease to the value $F$ until in the middle of this third year. For the final half-year, the response time quality threshold is stable with the value $F$.

![Figure 3: Threshold evolution](image-url)
3 CloudStore

3.1 Current implementations

In year two we started from the “modernized” version (phase 1) of the TPC-W standard application which already made use of the Spring framework, and migrated it to a cloud deployment. No particular new functionality was added to the implementation, and due to the fact that the modernized version already implemented the component decoupling following the MVC and SOA patterns, we had to mainly resolve some issues regarding the distributed storage and database.

We moved the database to a Master-Slave replication architecture that was setup both on Amazon’s RDS and in our private OpenStack infrastructure and the required drivers.

Additionally, a Multi-tenancy version of the CloudStore was produced by SAP, which will allow us to compare the costs of having different deployments for several CloudStore instances against having a single CloudStore multitenant instance that can be personalized and offered to third parties. This costs comparison will be performed during year three.

3.1.1 Multi-Tenant CloudStore

Cloud environments reduce data centre operating costs through resource sharing and economies of scale. Infrastructure-as-a-Service is one example that leverages virtualization to share infrastructure resources. However, virtualization is often insufficient to provide Software-as-a-Service applications due to the need to replicate the operating system, middleware and application components for each customer. To overcome this problem, multi-tenancy has emerged as an architectural style that allows sharing a single Web application instance among multiple independent customers.

3.1.1.1 Multi-tenancy

In general, we distinguish three major approaches to separate a tenants data from the data persisted by other tenants (Wang et al. [1]and Chong et al. [2]). The dedicated database system provides a separate dedicated database for each tenant and has the best isolation at the cost of the highest overhead. In a dedicated table/schema approach, every tenant uses the same database management system, but separate tables or schemas. The highest degree of sharing, respectively efficiency, is established by sharing the same tables and schemas. To differentiate the data, a column with the tenant id is added to each table. This approach also has the largest consequences on the application or platform. If the platform provides an abstraction of the database, it might handle the additional tenant id in a transparent way (e.g., EclipseLink [3]). Koziolek [4] presents a high level architecture for MTAs based on observations he made about existing offerings. In general, this architecture reflects a Web Application Architecture with an additional Meta data storage for the tenant specific information (e.g., customization, database id, tenant name, SLAs).

3.1.1.2 Implementation

We decided to use the shared table approach because of its widely accepted usage. Consequently, the primary key has to be a combination of the tenantId and the entity specific id field. This was realized by making the CloudStore JPA compatible and using EclipseLink as persistence manager. EclipseLink has integrated support for Multi-tenancy. In addition to the existing CloudStore implementation the tenantId, retrieved from the meta-data manager, is added to every existing native SQL statement to ensure data isolation and thus privacy of the data.

The CloudStore access the tenant meta-information by using a factory, which returns an interface abstracted reference to a tenant object. This makes the CloudStore portable for different tenant metadata access mechanisms. It would also enable the CloudStore to be deployed on environments without multi-tenancy support by implementing the access to the required information in a specific platform adapter (see Figure 4: CloudStores Meta Data Access Mechanism).
We implemented a Tomcat Valve which extracts the tenant id and thus the name from the servers’ hostname used to access the application and the necessary adapter. This enables the CloudStore to run on any servlet engine supporting the used Frameworks.

![Diagram of CloudStores Meta Data Access Mechanism](image)

**Figure 4: CloudStores Meta Data Access Mechanism**

### 3.1.1.3 Future Work

In the first steps, we outlined the characteristics of a multi-tenant system and used this knowledge to develop a multi-tenant version of the CloudStore as showcase. In the next project iteration we will leverage the opportunities given by a representative multi-tenancy enabled application to compare the operational costs between a shared application and separate virtual machines for each tenant. This enables us to show, in which situations Multi-tenancy outperforms alternative sharing approaches. Although the measurement results may have a relation to the capacity of a given system the interpretation of costs should be in the focus of this investigation. Therefore its relevance for the third year of the project is more significant.

### 3.2 Deployment

In phase 2 we prepared four versions of CloudStore, two versions for public clouds and two versions for private clouds. One of these two versions is SQL version and the other one is noSQL version. For each cloud environment we prepared deployment architecture as shown in Figure 5.
3.2.1 Public cloud

Our choice for testing a public cloud provider was the widely popular Amazon Web Services. In this phase we deployed only the SQL version of showcase on Amazon Web Services using its EC2 IaaS service, since problems were detected with DynamoDB.

3.2.1.1 SQL version

This version of CloudStore for public clouds is an adapted version of modernized version of CloudStore for usage of Amazon Web Services. We configured showcase to work with multiple databases. Instead of using a single database instance as in phase 0 and phase 1, we changed the CloudStore deployment to make use of the MySQL master-slave replication provided by Amazon’s RDS service. The “Relational Database Service” (RDS) is an Amazon's solution to scale relational databases which currently supports only master-slave setup.

The main difference between master-slave and master-master replication is that with master-slave replication all write operations go to the master instance, while read operations are distributed across slaves by round-robin or other balancing strategy. With master-master replication all operations are evenly distributed across all database instances regardless of operation.

3.2.1.2 noSQL version

On AWS we didn't test the noSQL version yet, because of development of new tools and some problems with testing of SQL version. Next year we will look into possibilities to deploy showcase with DynamoDB service since this is the only noSQL service that Amazon is offering.
3.2.2 Private cloud

For a private cloud environment we used our own OpenStack private cloud infrastructure. We use OpenStack Grizzly version, which doesn't support auto-scaling and monitoring services, so we deployed showcase on a cluster.

For the deployment of the frontend component of CloudStore we used 2 Ubuntu virtual machines with Tomcat version 7 installed and one instance for load balancer. We used HAProxy free software to load-balance traffic between these two virtual machines. Measurements on OpenStack will be performed in year 3, since OpenStack currently lacks of auto-scaling.

During this period no scalability nor elasticity measurements were performed, though these are expected to be carried out in year three, after we solve the problem of auto-scaling in OpenStack.

3.2.2.1 SQL version

For deployment of the MySQL cluster we used Galera cluster for MySQL. Using Galera we set up master-master cluster and not master-slave as we used with AWS.

3.2.2.2 NoSQL version

We used MongoDB as the noSQL database in private cloud and deployed it with 1 router instance and 5 shard instances. We used sharding because it support deployments with very large data sets and high throughput operations and because high query rates can exhaust the CPU capacity of the server.

3.2.3 Deployment scripts

In this phase we also prepared deployment scripts using Python and Boto Python interface for Amazon Web Services. Scripts are available on projects GitHub account4. We also prepared Ubuntu VM with all necessary software to run and test the showcase on AWS. This includes the deployment scripts and JMeter software for load testing.

Because our partners had troubles with prepared VM, we also developed two web-services for easier deployment and testing of showcase. One web-service is for deploying showcase on AWS and OpenStack and the second one is a service for running a distributed JMeter on AWS. In both web-services the backend is written in Django framework and the frontend is written in AngularJS JavaScript Framework.

![Choose provider](image)

**Figure 6: Choose provider page of web service for deploying showcase**

4 [http://github.com/CloudScale-Project/Showcase](http://github.com/CloudScale-Project/Showcase)
**Configure frontend**

We require at least these permissions for EC2:
- RunInstances
- StartInstances
- StopInstances
- TerminateInstances
- CreateKeyPair
- DeleteKeyPair

You can simulate your permissions [here](#).

Amazon access key

Amazon secret key

Do you want autoscaling?
- Yes
- No

Instance type

---

**Figure 7**: Configure frontend page of web service for deploying showcase
4 Load generation

4.1 JMeter scripts

In order to test different showcase deployments we prepared JMeter scenario script according to TPC-W specifications. TPC-W specifications define three different scenarios for crawling the showcase, shopping mix, browsing mix and ordering mix. We only focused on browsing mix and took into account the probabilities for browsing mix.

<table>
<thead>
<tr>
<th>Web Interaction</th>
<th>Browsing Mix (WIPSb)</th>
<th>Shopping Mix (WIPS)</th>
<th>Ordering Mix (WIPSso)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Browse</td>
<td>95 %</td>
<td>80 %</td>
<td>50 %</td>
</tr>
<tr>
<td>Home</td>
<td>29.00 %</td>
<td>16.00 %</td>
<td>9.12 %</td>
</tr>
<tr>
<td>New Products</td>
<td>11.00 %</td>
<td>5.00 %</td>
<td>0.46 %</td>
</tr>
<tr>
<td>Best Sellers</td>
<td>11.00 %</td>
<td>5.00 %</td>
<td>0.46 %</td>
</tr>
<tr>
<td>Product Detail</td>
<td>21.00 %</td>
<td>17.00 %</td>
<td>12.35 %</td>
</tr>
<tr>
<td>Search Request</td>
<td>12.00 %</td>
<td>20.00 %</td>
<td>14.53 %</td>
</tr>
<tr>
<td>Search Results</td>
<td>11.00 %</td>
<td>17.00 %</td>
<td>13.08 %</td>
</tr>
<tr>
<td>Order</td>
<td>5 %</td>
<td>20 %</td>
<td>50 %</td>
</tr>
<tr>
<td>Shopping Cart</td>
<td>2.00 %</td>
<td>11.60 %</td>
<td>13.53 %</td>
</tr>
<tr>
<td>Customer Registration</td>
<td>0.82 %</td>
<td>3.00 %</td>
<td>12.86 %</td>
</tr>
<tr>
<td>Buy Request</td>
<td>0.75 %</td>
<td>2.60 %</td>
<td>12.73 %</td>
</tr>
<tr>
<td>Buy Confirm</td>
<td>0.69 %</td>
<td>1.20 %</td>
<td>10.18 %</td>
</tr>
<tr>
<td>Order Inquiry</td>
<td>0.30 %</td>
<td>0.75 %</td>
<td>0.25 %</td>
</tr>
<tr>
<td>Order Display</td>
<td>0.25 %</td>
<td>0.66 %</td>
<td>0.22 %</td>
</tr>
<tr>
<td>Admin Request</td>
<td>0.10 %</td>
<td>0.10 %</td>
<td>0.12 %</td>
</tr>
<tr>
<td>Admin Confirm</td>
<td>0.09 %</td>
<td>0.09 %</td>
<td>0.11 %</td>
</tr>
</tbody>
</table>

Figure 8: Probabilities for different scenarios according to TPC-W specifications

We also took into account the waiting time between operations, which is set to 7 seconds.

In our distribution of JMeter we also included JMeter Plugins\(^5\). With these plugins it is possible to support different kind of scenarios (spikes, linear,) which is not possible with regular JMeter distribution. Example of linear scenario is shown in figure 6.

\(^5\) http://www.jmeter-plugins.org
4.1.1 Organization of JMeter scenario

The main entry point to JMeter scenario is a Test Plan node. Every Test Plan must contain at least one Thread Group. In our case we used the Ultimate Thread Group from JMeter Plugins. Everything under the Thread Group will be executed in every thread. From Figure 9 you can see that we execute some BeanShell code in every thread. After executing some BeanShell code for choosing which page we visit with certain probability, we actually execute the request with Http Logic Controller. At the end we register some listeners to collect results from logic controllers.
4.2 Distributed JMeter

Details of the implementation of the distributed JMeter architecture for scalable load generation, and it’s utilization with the defined scripts

During load-testing CloudStore we found the need for distributing JMeter, since running a single instance in one machine quickly runs out of RAM. JMeter runs out of memory when we try to run too many concurrent users, because it creates a new thread for each user. To that end, we developed a web service which runs distributed JMeter which anybody can use. The current version of the Distributed JMeter support uploading a JMeter scenario, specifying the desired EC2 instance type, the number of virtual users (VU) and the host on which the CloudStore is accessible. Currently we limited the EC2 instance type just to t2.medium, because it's not too costly, and we measured only for this type of instance how many VU can handle. We measured that one t2.medium instance can handle around 300 VU (we found that number by bisection), with 2GB Java heap size for JMeter. We didn't use the master-slave setup, because JMeter has problems with too many slaves and collecting the results of the tests. Instead of that, we run \( \frac{\text{number of VU}}{300} \) JMeter masters and collect the results from all instances merging them into one single file, which can be then downloaded and analysed.
5 Measurements results

This section describes the results of the pilot’s Capacity and Elasticity measurements for the CloudStore running on the Amazon Cloud, in accordance to the scalability evaluation plan described in D1.2 and the Usage evolution defined in section 2.4 of this document. We describe the measurement setup in detail. Afterwards we describe the results, and found bottlenecks.

5.1 Load Generation

Figure 11 shows an example JMeter scenario that has been used during measurements. Warm-up time is 2 minutes to fill the cache on frontend node and ramp-up time is 3 minutes where we start other virtual users (VU) and then we hold that load for 10 minutes.

![Figure 11: JMeter scenario used during test](image)

The warm-up VU is calculated as C/4 where C is the end number of VU or threads. E.g.: We want to test the system for 1500 VU, C = 1500, C/4 = 375. In this case the 1125 users will be started during startup time, in 3 minutes (where C refers to the Capacity)

5.2 Quality Thresholds

![Figure 12: SLO for 14 TPC-W operations](image)

Figure 12 shows the 14 operations in the TPC-W specification as well as their SLO. In CloudStore we have implemented 12 out of 14 TPC-W operations. The operations Admin Request and Admin Confirm are seldom used and are not yet implemented. The SLO in this figure is the maximum
response times for each operation in seconds. After a test we compare how many response times are under the SLO requirements with total number of operations of each type. At least 90% of all response times for each operation must be under the value that is specified in table above, otherwise we mark it as an SLO violation. We look only into responses that have response code in HTTP header equal to 200, because for other response codes (500) response times are around zero.

5.3 Work

There are two work parameters in CloudStore: number of books and number of customers. We use 10000 books. According to the TPC-W specifications we can choose between 1000, 10000, 100000 and 1000000 books in the database. 1000 books represent a small website and it is not appropriate for serious testing of scalability. 10000 books are more appropriate for testing scalability. It puts more work on the database and shows how well the architecture of application is designed, which is hard to know with a small database. 100000 or 1000000 books would give a considerable longer time to populate the database. A bigger database would probably lead into longer response times, but not so much longer if the database is properly indexed.

According to Section 4.3 in the TPC-W specification, we start with 288000 customers. Afterwards, new customers are registered by the Customer Registration operation. There are no operations for removing customers. Given an average of 3 second response time and 7 second think time, we will have 6 operations per minute per thread. With 0.85% of these operations as New Customers operations and with 2000 threads, we get approximately 170 (2000 * 10 * 0.85 / 100) new customers per minute, which is approximately 1700 new customers for the whole hold time of 10 minutes, which is negligible compared to 288000 customers. Number of customers can therefore be viewed as constant.

5.4 Configuration parameters

For the RDS m3.large instance type, which has allowed maximum connections of 600, we primarily used the connection pool size of 500, but later used 100, which gives slightly more capacity. The value 100 is also better because if use for example 5 frontend instances, that means 500 connections to database which is still lower than the limit of 600. If we would use the value 500 for each, we would already exceed the 600 limit with only 2 frontend instances.

In Tomcat we had to increase the default values of parameters maxThreads which determines the maximum number of simultaneous requests that can be handled and acceptCount which is maximum queue length for incoming connection requests when all possible request processing threads are in use, so the Tomcat isn’t a bottleneck. We also needed to customize the settings of mpm_worker Apache module so that Apache does not become bottleneck.

5.5 Capacity

5.5.1 Resource Space and Investigation Bottlenecks

Table 1 shows what configurations in the resource space we used for measurements. This table shows the largest VU that fulfills the SLO, rounded down to the nearest hundred. We did the measurements three times in a row for statistical significance. The hold time was 10 minutes each time.

The first four rows in table shows the measurements we performed with db.m3.large instance types for RDS and t2.small instance types for EC2. With these measurements we found out that the bottleneck was the database in most cases. In the case where the database wasn’t the bottleneck, we found that the limits for Tomcat and Apache were set too low. With these findings in mind we used the larger

---

6 Apache Tomcat configuration documentation: [http://tomcat.apache.org/tomcat-5.5-doc/config/http.html](http://tomcat.apache.org/tomcat-5.5-doc/config/http.html)
instance type *db.2xlarge* for RDS, and modified the settings for Tomcat and Apache as described in previous section.

To find out if the size of connection pool could be a bottleneck, we did the first two measurements in the table for *db.m3.large* for RDS and same for *db.m3.2xlarge* instance type. After that we did measurements for 1, 2 and 3 frontend instances for each RDS instance type.

<table>
<thead>
<tr>
<th>RDS slaves, # and type</th>
<th>Connections</th>
<th>Frond Ends</th>
<th>Max VU</th>
<th>Front end CPU Util.</th>
<th>RDS CPU Util.</th>
<th>Modified Tomcat &amp; Apache</th>
</tr>
</thead>
<tbody>
<tr>
<td>5, large</td>
<td>100</td>
<td>1</td>
<td>700</td>
<td>No data</td>
<td>No data</td>
<td>NO</td>
</tr>
<tr>
<td>5, large</td>
<td>500</td>
<td>1</td>
<td>700</td>
<td>No data</td>
<td>No data</td>
<td>NO</td>
</tr>
<tr>
<td>1, large</td>
<td>100</td>
<td>1</td>
<td>400</td>
<td>No data</td>
<td>No data</td>
<td>NO</td>
</tr>
<tr>
<td>1, large</td>
<td>500</td>
<td>1</td>
<td>400</td>
<td>No data</td>
<td>No data</td>
<td>NO</td>
</tr>
<tr>
<td>5, large</td>
<td>500</td>
<td>1</td>
<td>700</td>
<td>45%</td>
<td>30%</td>
<td>NO</td>
</tr>
<tr>
<td>5, large</td>
<td>500</td>
<td>2</td>
<td>1400</td>
<td>50%</td>
<td>80%</td>
<td>NO</td>
</tr>
<tr>
<td>5, large</td>
<td>500</td>
<td>3</td>
<td>1600</td>
<td>35%</td>
<td>90%</td>
<td>NO</td>
</tr>
<tr>
<td>5, large</td>
<td>500</td>
<td>4</td>
<td>1700</td>
<td>30%</td>
<td>95%</td>
<td>NO</td>
</tr>
<tr>
<td>5, 2xlarge</td>
<td>100</td>
<td>1</td>
<td>1300</td>
<td>80%</td>
<td>20%</td>
<td>YES</td>
</tr>
<tr>
<td>5, 2xlarge</td>
<td>500</td>
<td>1</td>
<td>1200</td>
<td>90%</td>
<td>50%</td>
<td>YES</td>
</tr>
<tr>
<td>5, 2xlarge</td>
<td>100</td>
<td>1</td>
<td>1300</td>
<td>80%</td>
<td>20%</td>
<td>YES</td>
</tr>
<tr>
<td>5, 2xlarge</td>
<td>100</td>
<td>2</td>
<td>2500</td>
<td>100%</td>
<td>40%</td>
<td>YES</td>
</tr>
<tr>
<td>5, 2xlarge</td>
<td>100</td>
<td>3</td>
<td>3600</td>
<td>90%</td>
<td>40%</td>
<td>YES</td>
</tr>
</tbody>
</table>

**Table 1: Measurement results**

We were limited to a maximum of 20 EC2 instances in total. In our distributed JMeter environment one JMeter instance gives approximately 300 VUs. For 3600 VMs we therefore needed approximately 12 JMeter instances. In addition we also need one JMeter master instance and 3 front end instances (see last row in the measurements table). Together this gives 16 EC2 instances. Testing with 4 front ends is therefore not possible with our current set up.

### 5.5.2 Estimate Time to Measure

Table 2 shows an estimate of the amount of human and computer time used to perform measurements.
### Table 2: Amount of time used for measurements

Producing the measurements reported here took more time than we planned, because of searching a capacity, finding bottlenecks and fixing bugs. First we had a connection pooling size at 500, which was a sound value when we measured with the database as a bottleneck. However we did not manage to make the front end a bottleneck with this configuration, so we changed it to 100, but then Tomcat and Apache became the bottleneck. This also required us to tweak the Tomcat and Apache parameters `maxThreads`, `acceptCount` and `mpm_worker` module in order to obtain a soundly elastic system that would automatically scale with the load we could produce. The database also appeared as the bottleneck while analysing the multi-tenant version of the CloudStore with the Dynamic Spotter, as seen in Section 6.1.

#### 5.6 Elasticity

For measuring elasticity we used 6 RDS db.m3.2xlarge instances to make sure that database didn’t become a bottleneck. We deployed the showcase in an auto-scaling group with a scale-up policy that adds one new instance if average CPU Utilization > 70% in a 60 seconds time frame. The scale down policy removes one instance if the average CPU Utilization falls below 40% for 60 seconds. For the cool-down property we used the value 0, in order to avoid a too slow scale. The cool-down property tells EC2 how many seconds it has to wait until the next scaling operation. For Tomcat and Apache we used same settings as with measurements for capacity.

In order to measure elasticity, we used a slightly different scenario (Figure 13). We extended the ramp-up time because of the EC2 provisioning speed of new instances which is between 2-3 minutes. We were measuring elasticity as the number of SLO violations with auto-scaling enabled. For elasticity measurements we used the capacity values 1300, 2500 and 3600. Different values mean different speed of load increase.

---

7 Apache Tomcat configuration documentation [http://tomcat.apache.org/tomcat-5.5-doc/config/http.html](http://tomcat.apache.org/tomcat-5.5-doc/config/http.html)
Table 3: Percentage of SLO violations

<table>
<thead>
<tr>
<th>No. virtual users</th>
<th>No. all requests</th>
<th>No. unsuccessful requests</th>
<th>Percentage of SLO violations</th>
</tr>
</thead>
<tbody>
<tr>
<td>1300</td>
<td>171283</td>
<td>10388</td>
<td>6%</td>
</tr>
<tr>
<td>2500</td>
<td>327577</td>
<td>17591</td>
<td>5%</td>
</tr>
<tr>
<td>3600</td>
<td>472735</td>
<td>17259</td>
<td>3%</td>
</tr>
</tbody>
</table>

There were some SLO violations (Table 4) which are minimal and expected. They all happen in time when a new EC2 instance is available but CloudStore application is not deployed yet in Tomcat, which takes around 30s. That’s why we got some 500 response codes from Tomcat which counts as SLO violations.

Below we are showing the CPU Utilization of EC2 instances. It is very nicely shown when a new virtual machine is added and when removed, and also how the CPU utilization distribute when the newly added virtual machine starts handling requests.
Figure 15: CPU Utilization for 2500 VU

Figure 16: CPU Utilization for 3600 VU
6 CloudScale tools results

All CloudScale tools were applied on the CloudStore application in year two. Below we present the results from using the Dynamic spotter. Due to events beyond our control, the results of the use of the Analyser, Extractor and Static Spotter are not included. This is due to events related to the migration of University of Paderborn personnel to the University of Chemnitz. These results will available in D5.3.

6.1 Dynamic Spotter on MT CloudStore

The Dynamic Spotter was used to analyse the CloudStore implementation in year one as well; in year two SAP used it to analyse the multi-tenant implementation of the showcase. As part of our validation the multi-tenant version of CloudStore was used on the SAP internal Cloud Environment Monsoon to validate the functionality of the Dynamic Spotter.

The Monsoon Cloud Environment enables individual developers with access to infrastructure and software via APIs and tools. It enables teams to share easy-to-configure and reproducible work environments, Simplifies and encourage sharing of infrastructure automation code across SAP, and lower operations cost and time to delivery for professional high volume and high complexity productive installations by providing a high performance IaaS and automation cloud using virtualization and bare-metal as well as different automation providers.

6.1.1 Goals

The goal of the experiment was to validate the applicability of the Dynamic Spotter on a cloud hosted application. Furthermore, we wanted do test the multi-tenant version of the Cloud Store and the load driver if they work in a cloud-based infrastructure. Thus the subsequent questions rose:

1. Can the multi-tenant CloudStore run on Monsoon
   a. Which steps are required to install the CloudStore?
   b. Does the CloudStore run without any problems?
2. Can the Dynamic Spotter be used in the Monsoon Cloud environment
   a. Which steps are required to setup the Dynamic Spotter?
   b. Can the Dynamic Spotter be executed in the Monsoon environment?
   c. Does the Dynamic Spotter find bottlenecks?

![Figure 17: Deployment of MT CloudStore and DS on Monsoon](image)
6.1.2 System Deployment

Figure 17 shows the deployment of the application and the required instrumentation on the Monsoon Cloud as well as the components of the Dynamic Spotter on a local test environment within the SAP corporate network segment. For more details about the different components we refer to Section 3.1.1 and D2.2.

The Application Server and the Database Server were of a small instance type running with 1 core and 2GB memory each. The Test Harness Server had 4 cores with 8GB memory.

6.1.3 Results

In the following we discuss the results of the experiment by answering the questions we elaborated in the previous Goals section.

Can the multi-tenant CloudStore run on Monsoon?

We were able to install the multi-tenant CloudStore on Monsoon. After installing a JVM, Tomcat and MySQL database the war file with the current information for the database connection was deployed. After filling the database with test data no additional steps were required to run the CloudStore on the monsoon database. However, to use the multi-tenancy features name aliases were entered to the name servers to identify various tenants accessing the application by the host name they are calling.

After these steps we were able to connect to the CloudStore with different hostnames and using the implemented functionality. The various tenants were identified by the hostnames used to call the application.

Can the Dynamic Spotter be used in the Monsoon Cloud environment?

For installing the Dynamic Spotter we first had to install JMeter as load driver for the multi-tenant CloudStore. After this step, Dynamic Spotter could be installed by following the documentation already provided. An Instrumentation/Measurement Agent was connected to the JVM hosting the Tomcat server, and consequently the application. Another Measurement Agent was deployed at the database server to monitor the resource utilization. However, in first experiment the Dynamic Spotter was not able to instrument the CloudStore. Dynamic Spotter did not find suitable entry points into the application to start with its instrumentation. This is because CloudStore uses the Spring Framework and Dynamic Spotter uses certain patterns to find entry points into the application which were not used by the Spring Framework. After adding the XX Annotation to the list of relevant patterns Dynamic Spotter successfully instrumented the application for further experiments.

After the installation process we were able to run Dynamic Spotter against an application hosted in the Monsoon Cloud environment.

The Dynamic Spotter identified no anti-patterns in the code, but observed the database becoming a bottleneck. Therefore, no anti patterns were identified but the clear advice was outlined to increase the database size to achieve a better performance. Most probably the major problem seems to be a potentially bad way to access the data, because the problem already arises at very low load. However, the heuristic required to identify this potential anti pattern is not yet supported by Dynamic Spotter.
7 Open repository and resources

The source code for deployment scripts, distributed JMeter web service and CloudStore is available on our GitHub project and is organized in the following directory structure:

- application/ - CloudStore source code
- deployment-scripts/ - deployment scripts
- distributed-jmeter/ - Django application for distributed JMeter web service

7.1 Distributed JMeter

In distributed-jmeter/ directory is a Django application for running distributed JMeter on AWS. If you want to run this application on your server you need to first provide the AWS credentials in config.ini.production and config.ini file.

You can run Django application locally or upload it on server and run it from there. If you want to run it locally you need to install the requirements in requirements.txt file with pip:

$ pip install -r requirements.txt

Otherwise, if you want to install it on server, you can do this by editing fabfile.py file which is Fabric run file. Set env.hosts to the IP or DNS name of server and env.user to the username of the user on server. Then run the following command:

$ fab new deploy:install=True

This command will install all needed software on server with Debian based Linux distribution to run distributed JMeter web service.

You should also change the server_name variable in conf/nginx.conf file to the name of your server. In conf/gunicorn.conf file set the user and group variable to the user and group which have permissions to the Django application. In conf/supervisor.conf also change the user and group variables to the user that have permissions to the Django application. Also change the user in paths of directory, command and environment variables.

7.2 Showcase

In order to build showcase you need to download Eclipse Java EE edition from their website. After you have installed and opened Eclipse you will need to import the CloudStore project from our GitHub. You will also need to download and setup the Apache Tomcat software inside Eclipse in order to run the showcase. If you want to run showcase from Eclipse you need to change row:

<import resource="classpath:${eu.cloudscale.datasource}.xml" />

in application/src/main/resources/app-context.xml file if you want to run MySQL version. You will also need to change the database url, port, username and password in file application/src/main/resources/database/database.aws.hibernate.properties to the settings of your database.

If you want to run the noSQL version from Eclipse you will need to change the import row from previous example to the:

<import resource="classpath:mongodb.xml" />

and set the values in application/src/main/resources/database.aws.mongodb.properties to ones with your MongoDB setup.

If you want to build the showcase from the command line, you don't have to anything above, just change directory the application/ and run:
Copy target/showcase-1.0-SNAPSHOT.war to the Tomcat webapps/ directory and restart Tomcat.

### 7.3 Deployment Scripts

Deployment scripts are written in Python using Boto Python interface to Amazon Web Services and python-novaclient package for OpenStack Nova API.

#### 7.3.1 Requirements

Before running scripts you need to install few tools on system where you running the scripts. Since scripts are written in Python you will need a Python installed and pip tool for installing and managing Python packages. All Python requirements are in the requirements.txt file and you can install them by executing the following command:

\$ pip install -r requirements.txt

Beside Python requirements you will need to install Java, Maven and MySQL. On Debian based Linux distributions you can do this with apt tool:

\$ apt-get install openjdk-7-jdk maven mysql-client

Deployment scripts are designed as standalone self-contained scripts and organized in three layers:

- infrastructure
- platform
- software

We provided scripts for Amazon Web Services and for OpenStack. Each script on each layer can be run independently from any other script. Despite independency between scripts, scripts needs to be run in certain order to properly deploy showcase on AWS or OpenStack. This execution flow is included in install.sh file which can be run from Linux terminal as:

\$ sh install.sh

If the user wants to run every script separately he just need to provide the path to the config.ini file as an argument to the script.

// Add image of scripts execution flow for AWS and OpenStack

### 7.4 AWS

Before running any script, user needs to edit the config.ini file. In that file there are settings for credentials to access to AWS or OpenStack, settings for database and settings for CloudStore.

#### 7.4.1 Configuration file

As mentioned in section 3, we are using AWS EC2 service for virtual machines and RDS service to setup the SQL database. Config file is therefore separated into two sections according to these two services. The section for EC2 includes settings for:

- aws_access_key_id
- aws_secret_access_key
- instance_type = t2.small
- key_name = cloudscale
- key_pair =
- region = eu-west-1
- availability_zones = eu-west-1a
- ami_id = ami-33a27444

![CloudScale](image-url)
and the section for **RDS** include settings for:

- `generate_dump_path`
- `instance_type = db.t1.micro`
- `num_replicas`
- `master_identifier`
- `generate_type = dump`
- `region = eu-west-1`
- `master_identifier = cloudscale-master`
- `replica_identifier = cloudscale-replica`
- `database_name = tpcw`
- `database_user = root`
- `database_pass = password`
- `driver = com.mysql.jdbc.ReplicationDriver`

Some settings have default values and it’s better to leave them as they are, except `instance_type` settings.

### 7.4.2 Infrastructure scripts

The infrastructure layer include Python scripts for creating the virtual machine on EC2 service, creating the AMI (Amazon Machine Image) image from EC2 instance, creating the auto-scalability group and launch configurations and finally the script for clearing the instances, database, auto-scalability groups, etc. These tasks are in scripts and their naming is self-explanatory:

- `aws-create-instance.py`
- `aws-create-autoscalability.py`
- `aws-create-ami.py`
- `aws-remove-all.py`

### 7.4.3 Platform scripts

On the platform layer we only have a script for setting up the MySQL instances on RDS. Beside that script we also have the complete dump of the MySQL database that script import when it creates the instance.

### 7.4.4 Software scripts

On software layer you will find the complete source code of showcase, Apache virtual host configuration file, Bash script for setting up the virtual machine and Python script for deploying the showcase on virtual machine. Below is the complete list of all scripts and they are self-explanatory:

- `showcase/`
- `cloudscale-apache-virtualhost.conf`
- `cloudscale-vm-setup.sh`
- `deploy-showcase.py`

### 7.5 OpenStack

Before running any scripts, user needs to edit the config.ini file. In that file there are settings for credentials for access to OpenStack, settings for MySQL and MongoDB database and settings for CloudStore.

#### 7.5.1 Configuration file

Config file is separated into three sections: OPENSTACK, MYSQL and MONGODB. In section OPENSTACK are common settings for authentication to OpenStack and settings for disk image.
Section OPENSTACK includes following settings:

- username
- password
- tenant_name
- auth_url
- image_name = Ubuntu 12.04
- instance_type = 512MB-1CPU-0GB
- key_name = key-pair
- key_pair = key_pair
- image_username = ubuntu
- database_type = mysql

Section MYSQL includes following settings:

- generate_type = dump
- generate_dump_path = software/rds-tpcw-dump_latest.sql
- instance_type = 2GB-2CPU-10GB
- database_name = tpcw
- database_user = root
- database_pass = password
- num_replicas = 5
- driver = com.mysql.jdbc.ReplicationDriver

Section MONGODB includes following settings:

- generate_type = dump
- generate_dump_path
- instance_type = 2GB-2CPU-10GB
- database_name = tpcw
- database_user = root
- database_pass = password
- num_replicas = 5

7.5.2 Infrastructure scripts

Infrastructure layer includes Python scripts for creating virtual machines on OpenStack, creating images from virtual machines, creating security groups, adding keys and script for clearing instances, images and security groups.

These tasks are in scripts and their naming is self-explanatory:

- openstack_create_mysql_instances.py
- openstack_create_mongodb_instances.py
- openstack_create_showcase_instances.py
- openstack_create_balancer_instance.py
- openstack_remove_all.py

7.5.3 Software scripts

On software layer you will find the complete source code of showcase, Apache virtual host and mysql configuration files and Bash scripts for installing and starting software on virtual machines. Below is the complete list of all scripts and their names are self-explanatory:

- install-apache-tomcat.sh
- install-load-balancer.sh
- install-mongodb.sh
- install-mysql-galera.sh
- install-mysql-galera-import-dump.sh
- my.cnf
- start-mysql-galera-first-node.sh
- start-mysql-galera-other-nodes.sh
8 Conclusion/Further work

The main objective was to develop a showcase to demonstrate the benefits of the CloudScale tools and methods, offering a convincing story line to support dissemination and providing a platform for exploitation. CloudStore is a free and open scenario definition and implementation of a typical web application, with some inherent scalability and performance issues that demonstrate the merits of CloudScale. It provides an example cloud application which allows us to define best practices and procedures for the measure and comparison of different desirable variables between implementation, architecture, deployment, configuration, and cloud provider options for a given application.

In this second year we provided a cloud-enabled version of the CloudStore, and performed the defined procedures for the measurement of an application’s Capacity and Elasticity as defined in D1.2.

This study allowed us to detect several problems with our preliminary set-up for the CloudStore deployment, and a nice comparison of what elasticity we could expect given different configuration and deployment parameters.

This is, after all, the main purpose of the CloudScale showcase as close-to-real-life example of a cloud application, and the methodology for comparing cloud providers, architectures, deployments and configuration options in order to improve not only an application’s capacity, but also its elasticity.

The identified database bottleneck is coherent also with the results of the Dynamic Spotter on the multi-tenant version of the CloudStore. As explained in chapter 6, the results from the use of the other tools will be included in the next version.

The CloudStore application has shown itself to be an asset for other projects as well as for its original intended use. CloudStore is a free and open implementation of a typical web application, and allows one to measure and compare different cloud providers, architectures and deployment in terms of capacity, elasticity and costs. CloudStore has been presented at a number of conferences and collaboration events, and has gained recognition in the cloud community

Additional results of this year’s work include a distributed version and a web interface for JMeter, with which we were able to produce the necessary load for the defined measurements and deployment scripts for both Amazon and OpenStack to simplify and speed up the deployment of cloud applications to both platforms. The source code for the deployment scripts, the distributed JMeter and the CloudStore implementation itself were kept and shared through the open-source GitHub source code management site, through which is being shared with the public.

In the next year we will focus in other desirable aspects such as scalability efficiency, but mostly the end goal of cost efficiency. To that end we will include a private OpenStack cloud in order to compare it to the Amazon public cloud in different terms of scalability, and also costs. Some work towards that direction has already been taken, such as with the deployment scripts for OpenStack and some preliminary test, though the lack of auto-scaling in the latest version of OpenStack means additional work will be needed in order to run the measurements in it. We will also work on the deployment of the noSQL version of the CloudStore on OpenStack, and investigate its scalability properties compared to the relational deployment. This will allow us to make a cost comparison for CloudScale following the expected usage, load and threshold evolutions defined in this document.

Additionally, a full report of the results of the tools applied to the CloudStore will be included in D5.3, the last version of this deliverable.

CloudScale has been awarded a stamp of recognition from the OCEAN project [www.ocean-project.eu]. The distinction is denoted "Reviewed by OCEAN Open Cloud". CloudScale is currently one of the 20 out of 74 projects currently in the Open Cloud Directory that has received such a distinction. The OCEAN Open Cloud label recognizes innovative assets, new concepts, architecture documentation and/or re-usable open source cloud components. The asset is found at http://www.ocdirectory.org/dataset/cloudstore.
References


