From a Rack to a Data Center: Development-Production Pipeline and Cluster Management in Service Oriented Architecture Scenarios

SERGIO CUCINELLA

Master of Science Thesis
Stockholm, Sweden 2013
TRITA-ICT-EX-2013:197
From a Rack to a Data Center: Development-Production Pipeline and Cluster Management in Service Oriented Architecture Scenarios

SERGIO CUCINELLA

Master of Science Thesis
Software Engineering of Distributed Systems
Examiner: Dr. Jim Dowling, KTH/SICS
Supervisor: Umut Alp, Truecaller

TRITA-ICT-EX-2013:197
Abstract

The choice of a system architecture for a provided service is one of the hardest tasks that enterprises are facing nowadays. In particular the biggest challenge is to design an agile, self-adaptable and innovative distributed infrastructure that can scale up globally according to the growth of user demand and that can keep low the overall costs.

In the last years the Service Oriented Architecture paradigm has clearly become the most used approach to design distributed systems. The reasons of its popularity are connected to the benefits that SOA brings, the most relevant are: loosely coupled architecture, seamless connectivity and interoperability of components, reuse of existing assets and applications, appropriate for integrating legacy systems, parallel and independent service development and easy and effective system maintenance.

The Swedish startup True Software Scandinavia AB built the world’s largest collaborative phone directory, Truecaller. It counts over 23 million of users that perform more than half a billion searches each month. The current infrastructure of Truecaller was not designed to hold this demand, therefore a re-engineering of the system architecture is required in order to provide a high quality user experience.

This research project presents the benefits that Truecaller would gain by migrating towards SOA and suggests new designs for both the internal system and the REST API that aim to cover all challenges previously mentioned. An important part of this project is also the definition of a structure and rigorous pipeline based on the concept of the Continuous Integration that guides the service lifecycle from development to production.

In conclusion, this thesis contributes by designing and developing Truecaller Cluster Management Service, a cluster management service that supports software engineers and administrator with all tasks related to the management of services and their configurations that is planned to be released as an open-source project.

The research also provides a roadmap that gives indications of how the current system should evolve and be extended.
Acknowledgements

The last two years have been very intense but full of moments of great joy and satisfaction. This Master’s Programme has come to an end and the long-awaited moment of the graduation has arrived, therefore I would like to thank the people that have been important for me in this journey.

Foremost, I would like to express my sincere gratitude to Truecaller that gave me the responsibility of designing and developing such critical and sensitive components. It was a big project, spanning a lot of different areas in computer engineering and I have learned a lot with it. Particular thanks go to my supervisor, Umut Alp, who has been listening to all my thoughts and ideas, sometimes bizarre, always providing me with his valuable pieces of advice. Without his mentorship, patience, help, insights and commentaries on every version of my work, this thesis would have been far from what it is today. I am thankful to Alan Mamedi and Nami Zarringhalam for making me feel a member of the team since the first day and for giving me the great opportunity to work in a fantastic startup where I can develop my skills in one of the most important period of my life. I also thank Alexander Flodin, Ilia Mashkov and Lorin Scraba.

A particular gratitude goes to my academic supervisor and examiner, Prof. Jim Dowling, for his continuous support, numerous pieces of advice and valuable feedbacks. I could have never thought of having a better advisor and mentor for my thesis.

I’m also deeply grateful to my entire family for believing in me all the time, giving me the possibility to pursue my higher education abroad and supporting me by any means they could.

I thank my girlfriend Nadya for her unconditional love, understanding, patience and support in all these years, but especially in the last months.

Finally, I would like to thank Federico, Giannario, Alexandru, Julio, Paris, Lars, Mariya and Mario for the stimulating discussions, the sleepless nights when working on assignments before their deadlines and for all the fun we had together. Without you all, these years would have never been so wonderful!
# Contents

## Contents

List of Figures .................................................. 8
List of Acronyms and Abbreviations ....................... 10

## 1 Introduction

1.1 Presentation of the project .......................... 13
1.2 Aims and objectives for this thesis .............. 15
1.3 Thesis structure ..................................... 15

## 2 Service Oriented Architecture

2.1 Introduction to SOA ............................... 17
2.2 System components ................................ 19
2.3 Benefits ............................................. 20
2.4 Limitations ......................................... 22
2.5 Web Services ...................................... 25
2.5.1 RESTful Web Services ......................... 25
2.5.2 REST vs. SOAP ................................. 28

## 3 Truecaller Architecture

3.1 System architecture evolution .................... 31
3.2 Migration to SOA .................................. 38

## 4 Truecaller Development-Production Pipeline

4.1 Development with Continuous Integration ....... 45
4.2 Automated Building and Deployment ............. 48

## 5 Truecaller Cluster Management Service

5.1 Design ............................................. 55
5.2 Implementation .................................... 60
List of Figures

3.1 Truecaller System Architecture schema in December 2012 .......... 33
3.2 Truecaller System Architecture schema in March 2013 ............ 34
3.3 Truecaller System Architecture linking after the moving to a data center in July 2013 ........................................ 36
4.1 Representation of the Git flow [15] ........................................ 49
4.2 Representation of Truecaller new development flow .............. 50
5.1 Screenshot of the deploy section of Truecaller Cluster Management web interface ............................................. 56
5.2 Current Architecture of Truecaller Cluster Management Service . 60
5.3 Representation of redeployment flow of a Truecaller service .... 63
6.1 Truecaller System Architecture linking after the purchase of hard-
   ware load-balancers, routers and firewalls .......................... 70
6.2 Truecaller Cluster Management Service Architecture after the mi-
   gration of the data store to MySQL .................................... 72
# List of Acronyms and Abbreviations

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>BPEL</td>
<td>Business Process Execution Language</td>
</tr>
<tr>
<td>CI</td>
<td>Continuous Integration</td>
</tr>
<tr>
<td>CMS</td>
<td>Content Management System</td>
</tr>
<tr>
<td>CORBA</td>
<td>Common Object Request Broker Architecture</td>
</tr>
<tr>
<td>DB</td>
<td>Database</td>
</tr>
<tr>
<td>DNS</td>
<td>Domain Name System</td>
</tr>
<tr>
<td>ESB</td>
<td>Enterprise Service Bus</td>
</tr>
<tr>
<td>HATEOAS</td>
<td>Hypermedia As The Engine Of Application State</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hypertext Transfer Protocol</td>
</tr>
<tr>
<td>IDL</td>
<td>Interface Description Language</td>
</tr>
<tr>
<td>IT</td>
<td>Information Technology</td>
</tr>
<tr>
<td>JDI</td>
<td>Java Debug Interface</td>
</tr>
<tr>
<td>JMS</td>
<td>Java Messaging Service</td>
</tr>
<tr>
<td>JSON</td>
<td>JavaScript Object Notation</td>
</tr>
<tr>
<td>JVM</td>
<td>Java Virtual Machine</td>
</tr>
<tr>
<td>MOM</td>
<td>Message-Oriented Middleware</td>
</tr>
<tr>
<td>OS</td>
<td>Operating System</td>
</tr>
</tbody>
</table>
PoC  Proof of Concept
QoS  Quality of Service
RC  Release Candidate
RDBMS  Relational Database Management System
REST  REpresentational State Transfer
ROA  Resource Oriented Architecture
ROI  Return on Investment
SOA  Service Oriented Architecture
SOAP  Simple Object Access Protocol
SoC  Separation of Concerns
SQL  Structured Query Language
TCM  Truecaller Cluster Management
TDD  Test Driven Development
TSDB  Time Series Database
UDDI  Universal Description Discovery and Integration
URL  Uniform Resource Locator
WADL  Web Application Definition Language
WSDL  Web Services Description Language
XML  Extensible Markup Language
XP  Extreme Programming
Chapter 1

Introduction

1.1 Presentation of the project

In the world of internet applications, the speed with which the application performs is a very important factor from the user’s perspective [22] [23]. The application’s ability to scale, in matters like number of users, computed data or peak usage will decide if this new application will bloom to a well known and used service or it will be just one of the many applications around. Other aspects like service availability and reliability are of a remarkable importance for both customers and providers; it is in the expectations of every customer to be able to use the service without interruptions. On the other hand, as a consequence of providing a reliable and available service, less resources need to be invested in the system development and maintenance.

The correct design and choice of the system architecture for a provided service will influence its performance and, thus, its success. Knowing how to design a system architecture and API infrastructure, that are able to evolve over time without massive changes in order to handle the continuously growing customer demand, may proactively avoid most of the future problems.

Every software deployed as a service requires an underlying infrastructure that can scale up with the increasing demand of usage. This is even more important when the data it collects and produces is highly interconnected. The Swedish startup True Software Scandinavia AB is facing extraordinary growth thanks to its product Truecaller. It is a trusted collaborative global phone directory available on all major mobile OS and on the web [9]. Truecaller counts over 23 million of users that currently perform more than half a billion searches each month [12]. In order to provide the best service
quality to their customers, Truecaller’s engineering team is currently working on increasing system’s scalability by a system architecture and the API infrastructure redesign.

In accordance with the engineering best practices, it is of the utmost importance to analyse the system, come up with the most significant factors affecting the system performance, then measure them and finally optimize them to have the most throughput out of the system. To approach such a complex problem, the system can be modelled or approximated as a combination of factors acting on data flow. Unfortunately there is not a single method to model such big software projects, and the evolution of new technologies makes the situation even harder. As a result, each system should be analysed in its own context and this analysis should be translated to a model, so that the development and maintenance routines can monitor and optimize this model.

According to Gartner Group reports, nowadays, 55% of global IT budgets are spent on infrastructure and operations costs [13] and almost the totality of new mission-critical operational applications and business processes are designed according to the Service Oriented Architecture (SOA) paradigm. Furthermore, IT organizations among the Global 1000 companies will broker (aggregate, integrate and customize) two or more cloud services for internal and external users from the current 5% up to 30% by 2014 [11]. In the last years, many enterprises worldwide that needed to optimize their business flexibility, efficiency and software evolution costs have chosen to migrate towards a Service Oriented Architecture [7].

Service-based architectures are really powerful infrastructures since they easily adapt to any sort of demand and are suitable for building distributed applications; services can be deployed in a distributed manner and can be composed together in order to build applications features [3]. Flexibility is just one of the benefits that supporters of Service Oriented Architecture praise, example of other very promising advantages are its cost-efficiency, predisposition for integrating legacy systems.

Another benefit of service based environments is that final consumers, or more in general services, do not require at development time the knowledge of where to find other useful services required for performing their business. This is possible due to the existence of some repository called Service Registry. Every Service Provider located in a Service Oriented Architecture, once its service is ready to handle incoming requests, has to publish and announce it
in a registry so that others can start querying it. Services require a specific configuration in order to be executed, this information could be present in different repositories, local or remote, and stored in different formats like XML or properties. The providers of system configurations are called Configuration Services. The possibility to change the configuration of a service without the necessity of redeploying or restarting it and to monitor their effects allows to provide always the most efficient and reliable service.

1.2 Aims and objectives for this thesis

The objective of this thesis is to analyse which system key metrics and overall design principles are involved while migrating a system architecture and API infrastructure towards a service based configuration and to expose the existing solutions and recommendations. Furthermore, it presents the choice that has been made to suit the most the needs that True Software Scandinavia AB has. The aim is to restructure the current Truecaller platform in a more modern service oriented system way so that the possible bottlenecks are isolated, a better overall system availability and reliability is ensured and that the customers demand is met. Finally it describes the design and implementation of a simple but efficient cluster management service.

1.3 Thesis structure

Chapter 2 provides the background knowledge relevant to this thesis, which allows the reader to understand it in depth. Furthermore, it introduces the state of art of Service Oriented Architecture. Chapter 3 analyses the current architecture used by True Software Scandinavia AB and presents the benefits that a migration to SOA would bring. Chapter 4 describes the new development flow and the procedures for deploying services in the staging and production environments defined at Truecaller. Chapter 5 presents the Truecaller Cluster Management Service and provides details related to its design goals and requirements as well as its implementation. Finally Chapter 6 concludes this thesis with an overview of the decisions taken to fulfil the requirements of True Software Scandinavia AB, the benefits that the migration brought and possible future work.
Chapter 2

Service Oriented Architecture

This chapter provides the reader with the background knowledge relevant for following entirely the discussion proposed in this thesis. The first paragraph presents an introduction to the Service Oriented Architecture. The second paragraph describes the components that are part of a service based environment. The benefits and limitations that come from the adoption of such architecture are discussed respectively in the third and fourth paragraph.

2.1 Introduction to SOA

Building complex applications over distributed platforms, such as grid architectures or clusters, is difficult and has led to an increasing interest in modular software interfaces based on services [3]. Service Oriented Architecture (SOA) is a paradigm for designing, developing, deploying and managing systems based on the concept of separation of concerns. It defines interfaces in terms of protocols and functionality, instead of a simple API, and how to integrate various applications in a Web-based environment. SOA is composed of a structured collection of coarse-grained Service Providers (Services), Service Consumers (Consumers) and Service Registries (Registries). The benefits that are generally recognized to SOA are its cost-efficiency, business agility, adaptability and leverage of legacy systems [5]. Fundamental challenge of open systems, that Service Oriented Architecture addresses, is to choreograph the behaviours of all interacting parties. As a consequence all of them can apply their local policies autonomously, but at the same time achieve effective and coherent cross-enterprise processes. The term choreography describes the interactions happening at the interface of the services from the perspective of an external observer. The term orchestration refers as well to the composition of processes from existing services, but it deals with the internals of the processes.
SOA introduces new approaches to the system design and development, therefore they are significantly different from traditional ones. The main differences that can be identified between traditional approaches and the ones based on services are:

- In traditional systems, system components are tightly coupled, on the other hand in service oriented systems service providers and consumers are loosely coupled.

- Semantics are shared explicitly during the design phase for traditional systems, on the other hand in service oriented systems semantics are shared without much communication between the engineers that develop services and the ones that develop consumers.

- In traditional systems it is known a priori the set of users and usage patterns that it will have, on the other hand in service oriented systems they are potentially unknown.

- In traditional systems the organization has the ownership of all components on the other hand in service oriented systems the components can potentially be owned by multiple organizations.

- In traditional software development the reuse is a well known concept but it is used only for convenience, on the other hand in service based software development the reuse is essential because complex services are built by assembling and integrating simple ones. As a consequence systems, shift from being implemented as single monolithic structures to as a series of components that cooperate to provide the wanted functionality.

- Service based software development replaces code generation, proper of traditional software development, with a combination of service discovery, selection, and engagement. Some or all of these steps might occur at runtime.

- In traditional software development there are long release cycles, on the other hand in service based software development there are short release cycles mainly due to the capability of rapidly adapting to changing business needs.

- In traditional systems legacy subsystems are very hard to integrate, on the other hand in service oriented systems they can be leveraged with
potentially minimal change to existing systems.

Nowadays, the most common way of implementing Service Oriented Architecture is without any doubt through Web Services [5] [61]. This is mainly due to the fact that they are based on a set of largely accepted and widely used standards such as HTTP [37], XML [29], SOAP [55], WSDL [62], UDDI [59] and BPEL [24]. Besides the traditional standards-based implementation approach (through SOAP) of developing Web Services, another one that is gaining popularity is that of RESTful Web Services [2]. The REpresentational State Transfer (REST) is an architectural style that represents a more lightweight implementation of Web Services, based on the definition of resources that provide the same interfaces, as defined by HTTP [5]. Both approaches are widely used, present advantages and disadvantages and in some cases the use of one over the other is more appropriate [20].

Other examples of technologies that can be used for developing service based systems are Message-Oriented Middleware (MOM), Publish-Subscribe technologies (e.g. Java Messaging Service (JMS)), and Common Object Request Broker Architecture (CORBA). Many software houses provide their own solutions for supporting the development in this environment.

2.2 System components

The three major components of a Service Oriented Architecture are: Service Providers (Services), Service Consumers (Consumers) and Service Registries (Registries).

Services

The Services are the building blocks for the development of large scale distributed applications since they provide high level abstraction. In particular, they abstract the infrastructure level of an application, that makes more efficient the usage of grid resources and facilitates utility computing, especially when redundant services can be used to achieve fault tolerance [3]. Services are loosely-coupled, discrete and remotely accessible software modules that perform self-contained and reusable business functionalities (e.g. number lookup, user validation) and that communicate using standardized interfaces, data formats, and access protocols (predominantly, but not exclusively, message-based) [8]. Service interfaces are separate from service implementation, which makes them platform-independent. Service information are commonly published and announced on registries.
Registries
The Registries contain basic information about available services such as description, specification (contract), documentation; and should include additional information such as classification, usage history, test results, and performance metrics. They can be as simple as directory of services categorized by type but they can be also more sophisticated and categorize services according to a predefined ontology, with quality of service (QoS) and binding information. They are usually centralised components, known to both publishers and consumers [8].

Consumers
The Consumers, other applications and systems that may or may not belong to the same administrative domain of the services, discover services in the registries, by querying them for services with desired characteristics, and then invoke the functionality that they provide. Consumers can invoke the services either directly over the network (e.g. via synchronous, direct, request-reply connections) or via a middleware component such as an Enterprise Service Bus (ESB); it is important that they have a robust exception handling in the event that services are no longer available. They can be either the final clients of that service or they can create new value by building more sophisticated and complex functionalities by composing or putting services together and becoming a provider on their own for other consumers. An example of service composition can be an order processing application that uses services such as customer lookup, credit check, and item lookup that are derived from a number of sources inside and outside the enterprise. It is challenging for consumer to guarantee after the composition QoS properties like performance, security, or properties that are common in distributed systems such as the support for transactions, especially when services are distributed across multiple organizations [5].

2.3 Benefits
The interest around Service Oriented Architecture is growing constantly from enterprises and organizations worldwide hoping to benefit from its acclaimed advantages. The most significant benefits of SOA adoption are [3] [5] [6] [19]:

- Loose coupling: Component parts of a Service Oriented Architecture are loosely coupled and location transparent, therefore enterprises can plug
in new services or upgrade existing ones with almost zero effort. System-level data and state consistency is insured by components interactions specified by the high-level contractual relationships.

- Cost-Efficiency: The reuse of existing assets and applications with the consequent elimination of any redundancy is at the base of the cost, development time and maintenance time reduction that SOA adoption brings. As an example, considering that three applications have their own number lookup functionality, if a service with that specific functionality would be implemented and used by all three applications, that service would be the only implementation to maintain besides the fact that it could be used by even more applications without any additional development.

- Business agility: The ability to quickly provide new functionality, by assembling existing services, offers enterprises the possibility to move and/or enter with the same speed into new channels of business and increase customer satisfaction. It is also very important to notice that it dramatically reduces business risks and exposure in favour of an increased business visibility. If we consider again the example of an order processing application that uses a set of services to implement part of its functionality, if the enterprise would enter the education business, it would count on a series of services already used by the order processing application, such as customer lookup and credit check. All services specifically created to support this new line of business, like room reservation, would increase the company portfolio and could be used by other applications.

- Implementation independence: Services of a Service Oriented Architecture are accessed by consumers in a standard way through the selected SOA infrastructure using the service interface. As long as the interface remains the same, the underlying services logic implementation can be written in any language and, together with the technologies supporting them, can be updated, adapted and replaced according to the changing needs and technologies without affecting the interaction with existing consumers.

- Leverage of legacy systems: SOA is currently the best option available for systems integration and leverage of legacy systems; legacy systems functionality can be exposed and accessed by consumers through a standard service interface. The hiding of legacy platform diversity from service consumers is also very helpful for incremental migration of legacy
systems components towards services without creating any disruption and for simplifying the integration process during mergers and acquisitions.

- Flexible configurability: Component parts of a Service Oriented Architecture are bound to each other and configured late in the process. This means that system configuration can change dynamically according to requirements without losing correctness.

- Granularity: The modelling of services at a coarse granularity helps in highlighting the high-level qualities that the service can provide. Furthermore, it allows to reduce dependencies among services as well as the amount of messages exchanged (few messages of greater significance).

- Development independence: Since every service deals with a specialized aspect and scope they can be developed in parallel and independently. This allows to increase the productivity and rapidly meeting changing business demands by speeding up the introduction and implementation of new products. As a consequence, they have easier and more effective maintenance, better scalability and graceful evolutionary changes.

2.4 Limitations

It is predictable, that a system re-engineering and adoption of a new architecture may have some negative aspects as well. A migration to or in general the adoption of a Service Oriented Architecture is not exempt for having some arguable limitations and disadvantages. Generally speaking, all kinds of distributed system development are affected by the common problem that what works well in a controlled and constrained development environment does not usually work once deployed in the production environment, mainly due to the number of standards and sometimes immaturity of both standards and products that support the system execution [5].

Another important aspect to consider is that several case studies, articles and publications that support and describe the business value provided by Service Oriented Architecture are sponsored by vendors, while only few enterprises have documented and discussed aspects of SOA implementation in practice together with the real benefits of reusability, increased agility and cost reduction [4].

The most relevant limitations, issues and disadvantages that may affect Service Oriented Architectures are [5] [6] [7] [8]:


2.4. LIMITATIONS

- **Initial investment:** Service Oriented Architecture adoption requires inevitably a large upfront investment for technology, development and deployment. Therefore, it could take a lot of time before the enterprises could have a Return on Investment (ROI).

- **Coarse granularity:** The biggest drawbacks that can derive from this type of granularity are, in the first place, that services cannot be easily optimized for efficiency, secondly that it is nearly impossible to test and validate in a complex service every combination of every condition, besides the fact that can affect response time and overall machine load, finally since services provide functionalities to several consumers this can lead to implementation inefficiency caused by complex and chaotic control structures.

- **Service interoperability and integration:** Integrating services in heterogeneous environment can be a really hard task, especially when there are many standards involved (e.g. web services that exchange SOAP messages over HTTP, encapsulating XML data). Furthermore, it requires specific skills.

- **System immaturity:** Majority of the systems that enterprises have may not be mature enough to expose their functionalities as services before re-engineer them and this obviously involve considerable investments.

- **Wrong strategy:** If during a migration to a Service Oriented Architecture the wrong strategy is chosen, then the result can be an expensive collection of random services that are never used.

- **Evolutionary development:** Adding continuously new functionalities to existing services can compromise the correct operation of the entire system.

- **Development tools:** Many vendors provide tools to allow enterprises to migrate and adopt SOA but majority of them are not mature enough and based on evolving standards. As a consequence, enterprises need to keep updating the existing code until those standards are consolidated.

- **Network connections:** When architecture and services are highly interconnected even a partial network failure may turn into a disruption of the entire system. Despite the fact that current technologies allow to cope with system downtimes, many enterprise architectures do not have enough redundancy and therefore are not sufficiently robust.
• Bugs: Due to the extensive service reuse, a bug or corruption present in a very used service may take down the entire system.

• Single point of failure: Service Oriented Architectures may contain some centralized components (e.g. Service Registry). Those components may introduce security and reliability risks, and may mine the system’s scalability and performance.

• Internet protocols: Majority, if not the totality, of Service Oriented Architectures are based on Internet protocols that are known to be unreliable, not able guarantee delivery or its order. Therefore it is required an extra effort to build services that ensure message delivery in a timely fashion.

• Security: The openness of services, due to the adoption of open standards, to other services and applications trigger some security issues. WS-Security appears still quite immature.

• Application ownership: Given the high service reuse in Service Oriented Architectures, it is really hard to define the boundaries of applications’ ownership and who needs to solve something when an issue appears. This introduces new problems in the service management.

• Lack of expertise: Service Oriented Architectures are based on a number of new standards and technologies and there are not that many experts available in the market. Understanding the underlying technologies is critical for those enterprises that are planning to migrate. Furthermore designing services at the right level of abstraction is also really challenging and requires specific skills and expertise.

• Loose coupling: It is a really admirable and useful engineering principle but it dramatically increases the system complexity.

• Metadata explosion: The number of messages that services exchange during their execution can grow really high and therefore managing services metadata is another important challenge.

• Testing limitations: In order to minimize the migration time to Service Oriented Architecture solutions it would be rational to be able to manage multiple integrations at the same time. If testing single new integrations might not be extremely complex, testing multiple integrations at the same time is currently almost impossible since there are no methods and tools for assessing the effects that each integration has on the entire system.
2.5 Web Services

The two most popular implementation approaches for Web Services are [2]:

1. The one based on the standard protocol Simple Object Access Protocol (SOAP), called also WS-*, with a clear reference to the prefix used in the various standards defined by the W3C [6], that is message passing based and it is used for the invocation of remote services. Its goal is to bring in web-based application a remote calls approach (Remote Procedure Call) [17] proper of protocols such as CORBA, DCOM, and RMI.

2. The one called REpresentational State Transfer (REST) inspired by the architectural principles of the Web and that is focussed on the definition of resources, on how to locate them on the Web and how to transfer them. Furthermore it is lightweight, highly efficient and scalable.

2.5.1 RESTful Web Services

REST is neither a technology nor an architecture, but it is an architectural style that does neither refer to a concrete and well defined standard nor it is a standard approved by a standardization body. It defines a set of design principles and methodologies (design pattern) that distributed systems should have for meeting certain characteristics, as an example being highly scalable. Its definition appeared for the first time in 2000 in Roy Fielding’s (one of the main authors of HTTP specifications) doctoral thesis called: “Architectural Styles and the Design of Network-based Software Architectures” [1]. This thesis analyses some principles at the base of several software architectures including the ones of software architecture that would allow to use the Web as a platform for distributed computing. REST principles are not necessarily related to the Web, they are abstract principles, but the Web is one concrete example.

RESTful Web Services are growing in popularity mainly because of the fact that the Web can be considered, without any change or additional system, a distributed computing platform according to the principles of REST. This is in total opposition to the SOAP-based Web Services.

The principles defined by REST are [10]:

Identification of resources: The resources are objects on which operations can be performed and represent the basic elements on which RESTful Web services are based, unlike the SOAP-oriented Web Service which
are based on the concept of remote call. They are uniquely identified by a URI.

**Explicit use of HTTP methods:** Every resource can be invoked by using the HTTP methods (POST, GET, PUT, DELETE). The usage of these standardized methods makes trivial the interaction with resources, thing that does not happen in non RESTful Web Services since it is required to know the methods’ names. REST establishes a one-to-one mapping between the typical CRUD operations (Create - Creates a new resource, Read - Gets an existing resource, Update - Updates a resource or changes its state, Delete - Deletes a resource) and HTTP methods.

**Self-described resources:** The resources are conceptually different from their representations returned to the consumers. As an example, a Web Service does not send to the client directly a database record, but instead an encoded representation. The ability of REST to handle the resource in multiple representation (e.g. HTML, XML, JSON) give the possibility to create both a Web API (Web Service) and a Web UI.

**Links between resources:** REST principles imposes that all resources are related to each other through hyperlinks. This principle is also known as Hypermedia As The Engine Of Application State (HATEOAS) and focuses on how to manage an application state. This means that every client can find all information related to a specific resource either directly or through hyperlinks in its representation.

**Stateless communication:** The interactions between clients and servers are stateless, as well as in the HTTP protocol. This means that there is no relation between previous, current and future requests. The fact that REST is based on stateless communication, does not mean that application do not have a state; it is client’s responsibility to manage the applications’ states. The main reason for this approach is the scalability, it is quite expensive, in terms of server resources, to keep sessions state and with the increase of number of clients this cost can be unaffordable. If on the one hand this lead to some sort of performance degradation due to the increased amount of data sent in the requests, on the other hand it encourages the usage of caching and clusters, since there are no constraints on the current session and no need to synchronize the session data with an external application, that optimizes the overall system performance.

This resource centric approach inspired by REST principles gave the birth to a new naming convention for these types of architectures: Resource Ori-
2.5. WEB SERVICES

REST-oriented Architecture (ROA); in opposition with Service Oriented Architecture (SOA) that is proper of service centric approaches.

Although REST principles include stateless communications, it does not mean that RESTful Web Services do not have any state. As mentioned previously REST stands for REpresentational State Transfer, and this emphasizes the importance of state management in a distributed system. The state that RESTful Web Services take into account is therefore the one related to resources and the one of the entire application. Unlike majority of the web applications, where the application state is often managed by the server along with the communication state, application state in a RESTful architectures is the result of collaboration between client and server, each with their roles and responsibilities. As an example we can consider how to handle a "cart" in common e-commerce applications. There are two possibilities that match the RESTful approach: the client manages the cart or the cart is managed on the Web Service as a resource. In the first case, the client will have a data structure in which will keep track of the items the user is interested in and they are sent along with the request for a new order. In the other case the Web Service provides the cart resource that records all the items chosen by the user. Being the cart a resource, it means that it is persistent, not associated with the current session, accessible by its URI at any time and can be managed via HTTP methods.

Another important aspect is how RESTful Web Services manage the State Transfer, meaning how an application moves from one state to another. The HATEOAS, one of REST principles, proposes the usage of hypermedia links (hyperlinks) as the basic mechanism to move an application from one state to another. It conceptually corresponds to a finite-state machine where the states are represented by the state of individual resources and the transitions from one state to another are triggered by hypermedia links. This principle aims to encourage not only the use of links to represent complex resources, but also to define any other relationship between resources and to control the permitted transitions, from one state to the other, of the application. The HATEOAS principle is also at the base of loose coupling between clients and RESTful Web Services. As an example, if a Web Service rearranges the relationships between resources, the client is able to find everything it needs in the received representations. Potentially the only thing a client requires is only the initial URI of the resource. The application state is then updated as the client follows the hypermedia links embedded in the successive representations of the resources. Unfortunately majority of the RESTful Web Services now available do not take advantage of this principle.
2.5.2 REST vs. SOAP

Although the goal of both REST and SOAP is to use the Web as a computing platform, their vision and the suggested solution are totally different. Regarding the vision, on the one hand, REST proposes a vision of the Web centred on the concept of resource and it is supported by a ROA (Representational State Transfer Architecture); a RESTful Web Service manages a set of resources on which a client can ask the canonical operations proper of the HTTP protocol. On the other hand, SOAP Web Service emphasizes the concept of service and it is supported by a SOA (Service-Oriented Architecture); a SOAP-based Web Service exposes a set of methods that can be called remotely from a client.

The Simple Object Access Protocol defines a data structure for the messages exchanged between applications, reproposing in some aspects part of what was already done by the HTTP protocol. SOAP uses HTTP as a transport protocol, but is neither limited nor bound to it, since it may use other transport protocols as well [17]. In opposition to the HTTP protocol, SOAP specification does not address topics such as security or addressing; for this reason two specific standards have been defined: WS-Security and WS-Addressing.

SOAP does not take full advantage of the HTTP protocol, since it uses it as a simple transport layer protocol, on the other hand REST uses the full potential of HTTP as an application layer protocol.

If, on the one hand, it is clear that the approach adopted by SOAP-based Web Services derives from the interoperability technologies that are essentially based on remote procedure calls, such as DCOM, CORBA and RMI. Therefore this approach can be intended as a sort of adaptation of these technologies to the Web. On the other hand, REST approach takes advantage of all Web characteristics, emphasizing its inclination as a platform for distributed computing. Therefore it is not necessary to add any extra infrastructure on the Web to allow remote applications to interact.

Furthermore, SOAP-based Web Services require the standard Web Services Description Language (WSDL) to define the service interface. This is a further evidence of the attempt to adapt to the Web the interoperability approaches based on remote calls. In fact, WSDL is an Interface Description Language (IDL) of a software component. If, on the one hand, WSDL promotes the usage of tools for client code auto-generation in specific programming language, on the other hand, it introduces a strong dependency between client and server. REST does not explicitly provide any method to describe how to inter-
act with a resource, since the HTTP protocol already provides it. Similar to WSDL standard there is the Web Application Definition Language (WADL) [60], an XML-based language that defines resources, operations and exceptions provided by a RESTful Web Service. WADL has been submitted to the W3C [63] for standardization in 2009, but as of the time of writing this thesis it is not a standard. The Web Application Definition Language is a bit in opposition to the HATEOAS principle since the first provides a static view of a Web Service, while the second poses in the hypermedia links in the resource representation the definition of contract, that provides a more dynamic vision and a loose coupling between client and server.

SOAP-based Web Services have explicit support for an "advanced" transactional feature while REST does not have any [17]. In particular, the standard WS-Atomic Transactions supports distributed, two-phase commit transactional semantics over SOAP-based services. It is possible, though complex, to introduce transaction handling in RESTful Web Services by using a "Transaction" resource that is created, updated and finally deleted by the client and implementing a specific control logic.

SOAP-based Web Service creates a complex and redundant infrastructure on top of the Web for achieving functionalities that the Web already provides. The advantage of this type of services is that they define an independent standard and that their infrastructure can be based without problems on different protocols. On the other hand, REST intends to use the Web as architecture for distributed computing, without adding extra unnecessary infrastructures, but it is currently works only over HTTP. Furthermore, since REST has a performance impact with a better support to caching, lightweight requests and responses, and therefore lighter response parsing, it allows to significantly reduce the traffic.

In conclusion, both REST and SOAP can be used to implement similar functionality and have advantages and disadvantages when it comes to building Web Services. After this comparison, my opinion is that, for most service scenarios, REST is simpler and more appropriate than SOAP. On the other hand, it is better to use SOAP if the application requires some features that are proper and specified in SOAP.
Chapter 3

Truecaller Architecture

This chapter initially presents a detailed analysis of the current architecture adopted by True Software Scandinavia AB for providing the Truecaller service. The first paragraph illustrates how Truecaller system architecture evolved over time, pointing out all the limitations and problems encountered along the way. In order to make Truecaller service scale proportional to the increase in the number of users, certain design decisions are made and this part tries to explain these in further details. In the second paragraph is described how the migration towards a service based architecture has been approached and are presented the technologies used by Truecaller for building the new services.

3.1 System architecture evolution

Truecaller system architecture has evolved relatively much since last year. The main reason of topology and configuration changes are due to the rapid growth that Truecaller is experiencing and therefore to provide all customers with the best experience possible.

The first real system architecture adopted by Truecaller is dated May 2012. It was a fairly simple configuration composed by 4 servers:

- 1 Apache Web Server and Memcached;
- 2 DB Servers;
- 1 Backup Server.

The Web Server was handling the requests for both the application backend, the website et alia. Furthermore, on the same machine it resided also the
memcached and cron jobs responsible for data manipulation and update. The DB were two MySql databases, one configured as Master and the other as Slave. This architecture was able to serve 80 requests/sec peak, with a max capacity of 150 requests/sec. The Internet traffic at that time was 3Mbps.

The major problems affecting this configuration were three:

1. The machines were distributed in three different shared racks;
2. All the machines had a public IP address and were directly connected to the ISP switches;
3. The internal network speed was 100Mbit.

Only couple of months later, in July 2012, the architecture was upgraded again. In addition to the initial 4 servers, the architecture presented:

- 2 Switches (24 port gigabit);
- 2 Net Servers.

The main purpose of this upgrade was to address the previously presented issues and to introduce a load-balancing infrastructure. In fact, all machines were moved into the same rack, the net servers had one ethernet interface connected to the ISP network and one connected to the switches, they were the only machines to have a public IP address and all the rest resided in the private network, finally the two switches allowed to upgrade the network speed from 100Mbit to 1Gbit.

The Net Servers were used as routers, load-balancers (HAProxy), firewalls (iptables) and for network services (time, dns, monitoring, etc.). An important aspect is that they were a high-availability pair running in Active-Passive mode; according to this mode of operation, the primary load-balancer dispatches the incoming requests to the appropriate servers while the second load-balancer operates in listening mode, it constantly monitor the primary load-balancer and it is ready at any time to step in and take over the system load balancing in case of failure of the primary load-balancer. Ideally the switch runs transparently to the rest of the system with no downtime.

In the following months the architecture grew in terms of number of machines in order to cope with the increase of traffic. In December 2012 the configuration included 12 machines residing in one rack and looked, as depicted in 3.1 as follows:
3.1. SYSTEM ARCHITECTURE EVOLUTION

- 2 Net Servers;
- 2 Switches;
- 3 Apache Web Servers;
- 4 DB Servers;
- 1 Backup Server.

Out of the three Web Servers, two of them were handling the requests for the application back-end as well as the memcached and cron jobs responsible
for data manipulation and update. The third Web Server was instead in charge of the website et alia. The DB were four MySql databases, two configured as Master and two as Slave.

This architecture was able to serve 500 requests/sec peak, with a max capacity of 1000 requests/sec. The Internet traffic at that time was 30Mbps and the internal traffic was 1.5Gbps.

In order to guarantee a better fault-tolerance, in this architecture has been introduced a network partition, that creates two redundant subsystems. Specifically, each switch is connected to an Apache Web Server, a DB Master Server and a DB Slave Server.

Figure 3.2: Truecaller System Architecture schema in March 2013
In the middle of February 2013 Truecaller released its new major update, Truecaller 3, which helped Truecaller in strengthening its brand and popularity; as a consequence, the incoming traffic to Truecaller system increased dramatically. Furthermore, the website was rebuilt using a popular Content Management System (CMS) which included new security questions. For these reasons, it was necessary to revise the infrastructure and scale horizontally. This led to the architecture shown in Figure 3.2 that started to be deployed shortly after, in March 2013, and that was composed by 28 machines residing in two racks:

- 2 Net Servers;
- 2 Switches;
- 5 Apache Web Servers;
- 4 Web Application Servers;
- 11 DB Servers;
- 1 Memcached Server;
- 1 Staging Server;
- 2 Backup Servers.

In this configuration three of the Web Servers were exclusively dedicated to the requests handling of the application back-end and the other two were in charge of the website et alia. The DB were eleven MySql databases, four configured as Master and seven as Slave.

This architecture was able to serve 700 requests/sec peak, with a max capacity of 1200 requests/sec. The Internet traffic at that time was 40Mbps and the internal traffic was 2Gbps.

The Net Servers were used, as before, as routers, load-balancers (HAProxy), firewalls (iptables) and for network services (time, dns, monitoring, etc.), but in this new architecture were converted from Active-Passive High-Availability mode to Active-Active. Specifically, the primary load-balancer is in charge of routing and load-balancing the live Truecaller service requests, the secondary one instead is in charge of monitoring and load-balancing the low-priority requests like the ones directed to the developers’ tools and staging environment.
The security question posed by the usage of a popular CMS for the website were addressed by creating an insecure DMZ network exclusively for the website et alia. As a result, the architecture includes two networks: the original API network and the insecure network. All hosts included in the insecure DMZ network are considered 'Internet' by the API network by means of firewall rules.

One of the biggest problem Truecaller architecture experienced was the location in which the servers were physically deployed. The server hall is in fact quite small and used mainly for small volume websites. The facility is connected to a single ISP, that is not a Tier 1 ISP, and it lacks some high-level security countermeasures. It is evident that this was a big limitation for Truecaller architecture’s scalability, reliability and availability. In order to address this important limitation, the previously described infrastructure has been moved in July 2013 to a real data center with better physical security and resource redundancy like electricity, network connectivity, etc.
3.1. SYSTEM ARCHITECTURE EVOLUTION

The only change that the architecture received is related to the connection with the public networks. As part of the moving strategy, Truecaller management decided to outsource the routing, firewalling and load-balancing to a specialized company; the racks switches are still handled by Truecaller. In 3.3 it is illustrated this new interconnection.

This architecture is able to serve 1000 requests/sec peak, with a max capacity of 1800 requests/sec. The Internet traffic at this time is ~50Mbps and the internal traffic is ~4Gbps.

After one year of observation, it has been identified a growth patterns, network wise, that indicates a ~12x increase per year in capacity needs. By the end of 2013 is it expected an increase in number of machines (~40 machines distributed in 3 racks), the architecture should serve ~3000 requests/second and the Internet traffic should be ~100Mbps.
3.2 Migration to SOA

The current codebase of Truecaller is mainly based on a LAMP (Linux, Apache, MySql, Php) stack; in 3.1 it has been explained that the back-end application runs on a number of Apache Web Server instances that grew over time: one in May 2012, two since December 2012 and three since March 2013.

The legacy codebase as well as the API protocols, used for the communication between mobile clients and servers, are not designed with good abstractions and clean structure. This makes it very hard to maintain and add new features as the product line evolves.

In addition to this, some limitations proper of Php and Apache stack like lack of shared memory, lack of real connection pooling, overhead of process initialization and poor memory management, are also becoming more and more visible road blockers to scalability for Truecaller’s current and expected growth.

The back-end application uses the database also as a queuing data store directly, and there are periodic jobs that query the database and consume the written jobs according to some specific logics. Writing and managing async tasks in a relational database is a general anti-pattern, which might work for small systems, but does not scale as the load increases. Truecaller’s current async tasks from Php are handled in a similar fashion, which brings several problems, for example database replication lag due to heavy write contention on queue tables, hard to maintain queue state and little visibility of job progress. Some operations on the other hand are done in synchronous fashion, just because there is not an established persistent queuing system.

In order to avoid that the heavy load floods the databases, the current codebase uses memcached as a wrapper on top of most SQL operations; only option due to Php lack of a real shared memory. This caching approach works perfectly, but in any case the real problem is the fact that the application logic still executes and consumes all other resources. This ends up with heavy CPU and memory usage on Php code and a lot of traffic to the memcached servers.

The Php back-end system relies on a configuration system defined inline in code, in addition to some configuration defined in databases retrieved and cached by the Php scripts. The Apache Web Server instances are designed to be stateless, but they all share the same configuration in a big monolithic application architecture composed of extremely coupled parts. Furthermore there are several copies of almost the same codebase in versioning systems that
are manually deployed on the different instances as production and staging environments. This brings extra effort to synchronize these systems and extra-risk of failing to do so.

There is no real process for change management today in terms of application code and configuration changes. Engineers make code changes on the test environment and push them to live if all tests pass. There is also no proper unit testing or scenario based functional testing involved. All changes are tested manually and this brings a lot of problems when concurrency is high, like in the production environment. Such poor code structure and almost inexistent test coverage inevitably bring hard-to-spot bugs and performance bottlenecks.

The current back-end application servers handle all incoming requests in a stateless manner, and although they share master data source (MySql databases) and caching (memcached) layers, they can run independently. Each incoming request is distributed to the servers in a Round Robin scheme by the load-balancers.

A similar approach is taken for database machines as well. Each database is kept in a minimum one-master-one-slave setup and point in time backup is taken from these machines in order to maximize data security. If a master node goes out of circulation for hardware failure reasons, one of the replicas can be promoted to be the new master during the time needed to rebuild the old master. The overall service quality might degrade due to less resources serving the load, but a total failure and therefore downtime should not happen.

As it has been explained previously in 2.3 all these issues and limitations can be addressed and resolved very easily adopting a service based architecture.

Nowadays, customers have rapidly evolving needs, want to enjoy services cross-platforms (from desktops, laptops and mobile devices), and expect 24/7 connectivity and reliability. As a consequence, service providers, in order to gain customer fidelity, need to release updates faster and more frequently, without compromising on quality; this is always very important because otherwise customers could switch to competitors or other alternatives. On the other hand, frequent updates cause downtime resulting in service disruption and lost revenue.

In order to be always in step with the times, innovative and provide the best service to the customers Truecaller engineering team decided in the beginning of 2013 to try to migrate to a Service Oriented Architecture and evaluate the outcome and its potential benefits.

The migration strategy has been planned in the following phases:
First
Design, development and testing of a Proof of Concept (PoC) that can illustrate the type of work required for the implementation and deployment of services. Specifically for Truecaller project it has been chosen to develop a new functionality required for the new major version and plug it in the existing monolithic Php application;

Second
Evaluation of the of the Proof of Concept;

Third
Definition of a roadmap based on priorities (core business functionalities first) for the migration of all functionalities included in the monolithic source code, in case of a positive result from the evaluation;

Fourth
Application of the roadmap.

The importance of a phased approach is that it allows an incremental migration and during this transition the core business works as usual. Furthermore it is also less risky compared to a complete migration of the codebase at the same time, mainly because it is easier to identify and resolve problems when the code involved is limited. Finally it limits the investment costs.

Among the several possibilities for implementing and expose services, as described in 2.1 Truecaller choice fell on RESTful Web Services. In particular the choice was to build RESTful Web Services using the Java Framework Dropwizard [27]. This framework stands out from others because of its lightweightness, ease of use and active community behind, characteristics well appreciated by software engineers. Dropwizard uses [28]:

- Jetty [40] - high-performance web server;
- Jersey [45] - the JAX-RS (the Java API for RESTful Web Services) reference implementation that allow to write clean, testable classes which gracefully map HTTP requests to simple Java objects;
- Jackson [41] - very powerful JSON parser and generator as well as a sophisticated object mapper;
- A number of other useful libraries like JDBI [43], Guava [34], Metrics [50] and Logback [39].
3.2. **MIGRATION TO SOA**

Very important aspect of this configuration is that Jetty is run neither as a heavy Application Server nor as a Servlet Container, but instead is run in embedded mode and the whole Web Server is executed as a single process JVM instance. This dramatically minimizes the administrative side hassles related to deployment at first and then monitoring of each Web Service.

The clear visibility of performance metrics with minimal overhead, ensured with the adoption of Metrics library, makes it very easy to spot bottlenecks and focus on the needed optimizations rather than operating in the dark. The Metrics library allows to have measurements like counters or more in general an instantaneous measurement of a value (e.g. the number of pending jobs in a queue), rate of events over time (e.g. requests per second), statistical distribution of values in a stream of data like minimum, maximum, mean, 75th, 90th, 95th, 98th, 99th, and 99.9th percentiles. Regarding the last type of measurement, the library actually does a statistical sampling in order to be memory efficient, therefore the values are not real values based on the combination of all samples, but an approximation of it.

Database access is critical in these kind of products since data play a fundamental and central role in every service. In Truecaller services database access is done through managed connection pooling; it is an expensive process to open and close a database connection for every database operation, therefore connection pools improve the performances by allowing multiple clients to use a pool of shared and reusable connection objects. These services take advantage of JDBI [43], a lightweight SQL convenience library built on top of JDBC [42], to write the sql prepared statements and object mappings. Unlike object relational mapping solutions, JDBI library does not create extra overhead for serialization and deserialization of data structures, but rather follows a lightweight easy to intercept logic to execute sql queries against the given database connection from the connection pool.

The adoption of Dropwizard together with the other libraries, allows Truecaller engineering team to easily build modular applications on a modern JVM based stack as well as monitor and optimize them.

The developed Proof of Concept clearly showed that a service based approach would successfully address and solve the initial problems of Truecaller back-end. Therefore the natural next step in the roadmap would be to continue developing decoupled services on top of this new set of libraries and migrating other parts of the system onto it.

Since in Truecaller current architecture there are a lot of code and database structure design decisions, which limit the scalability of the system, the en-
CHAPTER 3. TRUECALLER ARCHITECTURE

gineering team needs to refactor and rewrite them with a cleaner design and scalability in mind. An important part of the design is to create modular services so that the possible bottlenecks are easy to spot and optimize in an isolated service, preferably without degrading the user experience as a whole. It should be noted that this is not a one-to-one code migration from Php to Java, but instead there are numerous required changes and improvements in the way services are designed.

Since the whole codebase is going to be migrated to the new JVM based system, there is going to be more options to cache depending on the access and change characteristics of data. One big improvement would be using object-based caching, which is not happening in Php code today. This is not suitable for every type of data structure due to garbage collection latencies when the heap memory used gets large. But there are some data structures, which are very slowly changing, and thus a very good fit for in-memory caching.

This new caching option will not only suppress the load on databases, but will also cut down the size of code execution paths for the system resulting in decreased CPU and Ram usage, better latency and better throughput.

Memcached will still be one of the major components of the system in order to keep the load to the database servers at a reasonable level at all times. Memcached deployments in a clustered fashion might also provide extra benefit for overall system scalability.

These JVM based Web Services allow to ensure high availability of the platform by increasing redundancy at low cost due to their lightweightness and optimization; in order to ensure the best user experience any service should not stop working if a single server or application crashes. Redundancy also simplifies maintenance and upgrades since parts of the platform can be taken off-line without affecting the service availability.

In order to ensure an equal load distribution on all instances of a service, it has been assigned a unique URL per service and then the load-balancer is responsible to resolve it in Round Robin fashion and forward all the request to all endpoints with no operational bias involved.

Another important aspect is that the new codebase will be decorated with instrumentation logic, so that real-world metrics out of the execution in the production environment can be retrieved and examined at any time. They also help the change management since the effects of any change can be measured and compared with previous versions. This instrumentation is kept at a very minimal overhead level thanks to the libraries and methods used, yet it gives to the engineers very good insight into what is happening in the system.
This way, the system engineers can clearly visualize how important measures like latency and throughput fluctuate throughout the lifetime of the service. These metrics are currently kept in memory in the JVM for easy access and presented to the engineers through a JSON API.

The migration of Truecaller platform towards a more modern Service Oriented Architecture provides improved scalability and maximum capacity, the isolation and resolution of all possible bottlenecks and risks, a better overall system availability and maintenance and a more stable creation of new services. The new Truecaller architecture in combination with the JVM based web services should hold over 100 million registered users and over 10 thousand simultaneous users.
Chapter 4

Truecaller
Development-Production Pipeline

This chapter presents the new Development-Production pipeline adopted at Truecaller. The first paragraph illustrates the concept of Continuous Integration (CI) and its benefits. The second paragraph instead provides a detailed description of the new pipeline.

4.1 Development with Continuous Integration

The Continuous Integration is a well known Software Engineering practice that is applied in contexts where software development is done through a versioning system. In other words, every member of a team integrate and align their work to a shared environment (mainline) frequently, leading to multiple integrations per day \cite{CI}. This concept was originally proposed in the context of Extreme Programming (XP) \cite{XP}, as a countermeasure to the difficulties of integrating software’s modules developed independently over long periods of time; referred to as "integration hell" in early descriptions of XP.

One of the decisive aspects of Continuous Integration that convinced Truecaller engineering team to adopt it in their development flow is the ability to deploy quickly, safely and with minimal impact to production traffic. It is in general a big challenge to avoid problems completely and their frequency increase considerably as the product grows more complex and as the team gets larger.
Some common non-solutions to this problem are:

**More manual testing**

It is impossible to address and test all possible scenarios and to identify all problems, mainly due to the fact that these tests are usually not run in the production environment. Furthermore this approach definitely does not scale with complexity.

**More up-front planning**

It is really hard to determine if the up-front planning is too little or too much. The second scenario is usually more likely to happen and it easily degenerates in focussing on non-real issues. Even more problems related to non-essential requirements will appear.

**More automated testing**

Automated testing is fundamental in any software development process and the larger code coverage it has the more useful it is. Regardless of code coverage ratios, there is no way to ensure that a feature given to real end users will survive, because no automated tests are as brutal, random, malicious, ignorant or aggressive as real users could be.

**Code reviews and pairing**

This is a really good practice but only for increasing code quality, prevent defects and educate engineers.

**Ship more infrequently**

This approach definitely reduces the downtime that problems can produce, but it does not assure that no problem will appear. This will push to ship even less frequently that is absolutely not good.

Despite these are solutions to real problems, to resolve the type of problems identified previously the adoption of Continuous Integration and Deployment is necessary.

Another important benefit of CI is that it is cheaper to have it more than not to have it. It does not automatically remove all bugs and problems, but the regular integration of the new changes allows to find and remove them easily, this mainly because if something goes wrong the amount of changes to analyse is extremely small. In return there is more time to invest in feature development. In opposition, the longer is the periods between integrations the more difficult is to identify and resolve problems. Big integration problems can easily take a product development off the roadmap, or cause its entire
4.1. DEVELOPMENT WITH CONTINUOUS INTEGRATION

failure.

In Truecaller the new services are structured as multi-module Maven projects. In particular, every project has either 2 or 3 modules:

**Service module** - that includes all the application process related code, Jersey resources and service configuration;

**Client module** - that exposes the service interfaces to the consumers and includes the response objects;

**Core module** - that include all business logic and data access classes (e.g. Data Access Objects) shared among the Service and Client modules and potentially other services as well.

There are few reasons that make multi-module Maven projects suitable for Truecaller solution. Firstly, it allows to clearly apply the Separation of Concerns principle (every module includes exclusively the logic and objects related to a specific scope and nothing more) and therefore it gives conceptual clarity. Secondly, it allows to easily apply the software reuse by adding module decoupling. Finally, it provides easier maintainability in long term projects due to smaller component sizes which does one thing at a time, instead of one large interconnected codebase.

The Continuous Integration was originally designed to be complementary to other practices, especially related to Test Driven Development. According to TDD every software developer should setup and run automated tests before integrating their contributions in the mainline, in order to ensure that the changes do not introduce errors in the existing software. For this reason, the CI is often applied in environments in which there are systems for automatic building and/or automatic testing; e.g. Jenkins.
4.2 Automated Building and Deployment

The aim of setting up a structured and rigorous pipeline, that guides the lifecycle of services from their development to their deployment, is to establish a well known and tested process for every engineer involved in all the phases of the pipeline.

As it has been explained in 4.1, the key part of the Continuous Integration lies in the ease of periodical merge of changes applied to the codebase of a specific product. This part of the process involves a certain central repository used and shared by all the engineers contributing to a particular component of the system. Truecaller choice regarding the versioning system fell on the adoption of Git [32]; in particular to create a repository for each project. Git is well known to be great for branching and merging, since they are extremely cheap and simple actions, and therefore a perfect match for development flows based on CI.

There are several development flows that can be applied in accordance to Git, the most famous are: the Git flow [15] and the GitHub flow [14].

As shown in 4.1 the peculiarities of the Git flow are:

- The repository holds two main branches with an infinite lifetime: master and develop;
- The HEAD of the master branch contains the codebase in a stable production state;
- The HEAD of the develop branch contains the latest developed change;
- Whenever the source code need to be changed, a feature breach is created out of the develop branch. At the completion of the feature the changes are merged back into the develop branch;
- When all features planned for a release are ready, from the develop branch is created a release branch. In the release branch can be applied patches for identified bugs, these patches are merged into the develop branch;
- When the codebase in the release branch is ready for production, it is merged into both the master and the develop branches.
4.2. AUTOMATED BUILDING AND DEPLOYMENT

Figure 4.1: Representation of the Git flow [15]
CHAPTER 4. TRUECALLER DEVELOPMENT-PRODUCTION PIPELINE

On the other hand the characteristics of the GitHub flow are:

- The repository holds one main branch with an infinite lifetime: the master branch;
- The HEAD of the master branch contains source code that can be deployed;
- Whenever the source code needs to be changed, a feature breach is created out of the develop branch. At the completion of the feature the master opens a pull request (a code review request). After the code is reviewed and approved the changes are merged back into the master branch and can be deployed.

Figure 4.2: Representation of Truecaller new development flow
The development flow designed for Truecaller is a mix of the two previously presented flows with some minor changes. As illustrated in 4.2 the new Truecaller flow is:

1. The software developer clone the master branch (the development branch) on his/her workstation;

2. The software developer applies some changes or develops a new piece of functionality;

3. The software developer tests the new code;

4. The software developer commits locally the new code;

5. Steps 2, 3, 4 may be repeated until all planned enhancements are ready;

6. The software developer submits the changes for review;

7. A senior software architect reviews the submitted changes and suggests improvements, if any;

8. The software developer applies the suggested improvements, if any, merge back into the development branch the changes and push to upstream;

9. When all enhancements planned for a release are ready, a senior software architect merges the master into the RC branch and push to upstream;

10. The commit on the RC branch triggers a post-receive Git Hook that starts a proper Jenkins job;

11. The started Jenkins job pulls the latest revision from the RC branch, starts the Maven building process for all artifacts involved (that produces a set of deployable JAR files) and runs both unit and integration tests;

   - If the started Jenkins job has an upstream dependency, this scenario appears whenever a project depends on another one, the other job is executed at first and upon its completion (successful or not - depending on the specified rule) the initial job continues its execution.

12. If all tests are successful the Jenkins job archives the built JAR file(s) in the Nexus local repository. On the other hand, if there are any tests that fail, a message error is sent to a selected group of engineers and administrators that will delegate the resolution of the problem(s) to a software developer.
Therefore the main characteristics of Truecaller new development flow are that the master branch is not the production one, but the development one instead. There are no feature branches but a cloning of the main branch instead. The HEAD of the Release Candidate (RC) branch contains source code that can be deployed. A Jenkins server job, triggered by a Server-Side Git Hook, creates Continuous Integration builds and runs automated testing routines. The Nexus local repository stores and serves the JAR files of the services and the dependency components in intranet.

It is a design choice that every new JVM based service is built as single uber-JAR file; this is a super file that contains all of the .class files required to run your service, dependencies included. Having a single deployable JAR file per service simplifies the deployment procedure a lot and the process for promoting services form the staging to the production environment, usually complex because the two environments could have installed different libraries’ versions. The creation of this uber-JAR file is possible thanks to a Maven plugin called maven-shade \[49\]. With few changes in the <build><plugins> section of the service pom.xml file it is possible to configure Maven to do the following things during its package phase \[28\]:

- Produce a pom.xml file which does not include dependencies for the libraries that are included in the JAR file;
- Exclude all digital signatures from signed JAR files. Required otherwise Java considers the signature invalid and won’t load or run the JAR file;
- Collate the various META-INF/services entries in the JAR files instead of overwriting them. Required for the correct functioning of Jersey;
- Set the specified class as the JAR’s MainClass. Required for being able to run the JAR file using ”java -jar”.

Once the JAR files are archived in Nexus, they can be easily accessed and downloaded to hosts where they can be run. Initially all services are run in the staging environment, so that a selected target group can test the new functionalities and assess the quality of the product. In the new pipeline of Truecaller, the desired behaviour of development lifecycle is spending just enough time in the staging environment for testing the functionality and then passed to the next phase which is the production environment.

The change to a more modern way of designing and implementing services, together with a more structured and dynamic development-production
pipeline and a more controlled and organized change management guarantees that the services deployed in the Truecaller cluster are highly available, reliable and high quality.
Chapter 5

Truecaller Cluster Management Service

This chapter presents Truecaller Cluster Management (TCM) Service, a cluster management service developed at Truecaller in order to simplify all procedures related to the management of services and configuration that is going to be released as an open-source software. The first paragraph explains the reasons that led to the idea phase of such a service and presents its design requirements. The second paragraph provides an insight on how this service has been developed.

5.1 Design

Since the number of services is going to increase as new features are built, so will the complexity of service and configuration management. In order to help the administrators to deploy, manage and monitor services and also the engineers to monitor performance metrics, it has been considered essential and strategic to develop a service that encapsulates and expose all of these features.

In Truecaller there are a lot of services that are deployed several times a day. From the beginning, the goal has been to design and develop a service that would have the least number of steps, people and procedures involved for deploying any service on any node of Truecaller’s cluster.

TCM is the result of several months of work to address the obstacles of making the deployment as easy and painless as possible. Thanks to the clean and immediate web interface, as shown in 5.1, TCM allows one entitled person to deploy any service with a few clicks and more importantly in a few seconds. The alternative procedure used to involve few people among developers and
administrators and takes tens of minutes.

The deployment is definitely a very important process in the everyday working activities at Truecaller. Truecaller Cluster Management Service will help Truecaller to get more features out faster with less problems and hassles and will be fundamental to keep the same speed with the continuous team expansion.

Besides the deployment, the redeployment and the undeployment, Truecaller Cluster Management Service allows to see the whole set of nodes and services existing in the system, to see the list of all services deployed on a specified node as well as the list of nodes on which a specific service is running, to see the Health Checks (Dropwizard specific runtime tests used to verifying a service’s behaviour in its production environment) of a specific service, to see the snapshot of a specific service’s Metrics, to see the Configuration of a specific service and most importantly to change on the fly the Configuration of a specific service.
5.1. DESIGN

Before proceeding with an in house solution and reinventing something that is already existing, few available softwares have been examined, e.g. Chef [20], Puppet [54], Fabric [31], Capistrano [25]. As a result of the analysis the best option was to develop an ad-hoc solution for Truecaller since no other software had the basic characteristics required by the specifications. These characteristics are:

- Complexity proportional to Truecaller’s system;
- Web based;
- Possibility to dynamically change services’ configuration;
- Possibility to check a snapshot of a service Metrics;
- Adaptable to Truecaller’s network;
- Easy to use.

As it has been already said, TCM does much more than simply deploy services in Truecaller infrastructure, in facts it manages services and their configurations. In order to do so, it need to be aware of two basic entities: Application and Node.

An Application is a service, it has been chosen this name because it is the one used as a convention in Dropwizard 0.7. An application is identified by an Artifact Id and a set of ten ports: Port, Admin Port, JDI Port and seven reserved Ports; only the listening port is actually chosen all the rest instead are assigned incrementally. A Node instead is a host that resides in the system. It is identified by a Name, that is its domain name.

Applications and Nodes are in relation with each other, specifically an N to N relation since an Application can run on several Nodes and in a Node can be deployed several Applications.

The first version of TCM Service provides the following functionalities:

- Insert a new Application (Only Applications that have already a JAR file archived on Nexus can be inserted). Requires the following info (all must be unique in the DB):
  - Artifact Id (String max 128 characters);
  - Port (Integer between 30020 and 31000).
• Delete an existing Application (It is impossible to delete deployed Applications).

• Insert a new Node. Requires the following info (all must be unique in the DB):
  – Name (String max 128 characters).

• Delete an existing Node (It is impossible to delete Nodes with running Applications).

• Show the list of all existing Applications;

• Show the list of all existing Nodes;

• Show the list of all Applications deployed on a specified Node;

• Show the list of Nodes on which a specific Application is running;

• Map an Application with a Node. It is required for deploying an Application. Requires the following parameters:
  – Node name;
  – Application. Once selected should send the following info:
    * Artifact Id.

• Unmap an Application with a Node. Requires the following parameters:
  – Node name;
  – Application. Once selected should send the following info:
    * Artifact Id.

• (Re-)Deploy a specific Application on a specific Node. Upon successful deploy it is updated the table of Relations including information about Application’s version deployed and timestamp. Requires the following parameters:
  – Node name;
  – Repository (Optional);
  – Group id (Pre-filled with 'com.truecaller');
  – Application version (Optional);
  – Application (If does not exists yet allow to insert a new Application). Once selected should send the following info:
5.1. DESIGN

- Artifact Id;
- Port;
- Admin Port.

- Undeploy a specific Application from a specific Node. Upon successful undeploy it should update the table of Relations. Requires the following parameters:
  - Node name;
  - Application. Once selected should send the following info:
    - Artifact Id.

- Get and show the Health Checks of a specific Application from a specific Node. Requires the following parameters:
  - Node name;
  - Application. Once selected should send the following info:
    - Admin Port.

- Get and show the snapshot of a specific Application’s Metrics from a specific Node. Requires the following parameters:
  - Node name;
  - Application. Once selected should send the following info:
    - Admin Port.

- Get and show the Configuration of a specific Application from a specific Node. Requires the following parameters:
  - Node name;
  - Application. Once selected should send the following info:
    - Port.

- Set the Configuration, and then show the new one, of a specific Application from a specific Node. Requires the following parameters (This is a POST method):
  - Node name;
  - Application. Once selected should send the following info:
    - Port;
  - JSON object "config" that holds the new values (Since it is a POST method, this information has to be sent as a Form Parameter).
5.2 Implementation

Separation of concerns and abstraction layers have been the main focus in the development of TCM. For this reason, it has been split in three major components:

**UI** - Service that is aware of all Applications and Nodes. This is the only component that can be directly accessed, via a web interface, by the personnel;

**Control** - Extension that exists in every Service in form of dependency. It allows to get and update the Service configuration from UI;

**Node** - Service that resides in every host of the system and is deployed as part of the server setup procedure. It can either act as a mere proxy between UI and all running Applications (it forwards the requests) or (re-)deploy and undeploy Applications by using native system calls.

Truecaller Cluster Management Service Architecture is illustrated in Figure 5.2:

![Figure 5.2: Current Architecture of Truecaller Cluster Management Service](image)
5.2. IMPLEMENTATION

It is important to highlight that all components are single-module Maven projects based on Dropwizard. Furthermore, as it has already been written, UI is the only component that receives the commands directly for the administrators via the web GUI and therefore is the only active component. On the other hand, Node and Control passively react on the received API calls.

It is understandable that every component has a different lifecycle, in fact the Control component is bound to a Service lifecycle, the Node component is strictly linked to a server lifetime, while the UI component has a lifecycle that does not depend on anything else. Although the UI component plays a key role in Truecaller Cluster Management, the first version of the service does not include any redundancy for this component. The main motive for this choice is the fact that no real services actually depend on the UI component. If the UI component fails to operate successfully, then the only disruption to the service level is about cluster configuration management and monitoring. The services will keep running as usual and the UI component will need to be fixed in isolation not causing any downtime for other services.

The information related to existing Applications, Nodes and their Relation are stored in a database. The first version of TCM Service relies on a file based H2 database \[38\] in order to reduce external dependency as much as possible. Although a general failure of the UI component is not considered a service level disruptive problem, a file based database might have resulted in the loss of all data in case of a disk failure. Since this is not an acceptable design, two alternatives were proposed: either to create a new scheduled job that would periodically create a backup copy or to migrate this schema as well to the MySql database since it is already backed-up, replicated and secured. The second option seemed the most reasonable and therefore the next version of TCM is going to use the MySql database as storage. The adoption of the MySql database will also allow to introduce redundancy of the UI component and therefore improve its availability.

All Applications running on a Node are executed by a special App user that has a limited set of privileges. Since some Applications are exposed to the Internet, this ensures a good level of security since, even if an Application gets compromised, the attacker will not gain automatically full privileges on the host. Another important security aspect is that all sensitive functionalities listen on the Admin Port, port that due to firewall rules is accessible exclusively from Node (not even UI can access it), therefore even if the service exposes its public API to the Internet the crucial features are subjected to a restricted access. Finally, another security enforcement consists in the authentication;
Ui needs to authenticate itself against Node before it can actually pass the command to execute, same thing applies for Node against Control.

The TCMService has been developed as part of Truecaller Common library and is going to be branded and released as an open-source project. Information on how and where to find the source code will be published on Truecaller website: http://www.truecaller.com.

The Artifacts Ids involved in the service are:

- tc-common-core - base dependency for all Truecaller services. Inside the package com.truecaller.common.control there are all the classes responsible for the Control component implementation;
- tc-common-node - Artifact Id that implements the Node component;
- tc-common-ui - Artifact Id that implements the UI component.

All Truecaller services have inside their base package (in the form of com.truecaller.ArtifactId.Module) either two or three packages. The two default packages are: app and jersey, while the optional one is db.

The app package contains two classes:

1. The Application class - It is the main class of the uber-JAR file and takes care of the bootstrap of the whole service;
2. The Configuration class - It contains all the settings and configuration values concerning the Application.

The jersey package instead may contain a variable number of classes. These classes are the Jersey Resources, that are the endpoints waiting for incoming requests.
Figure 5.3: Representation of redeployment flow of a Truecaller service
CHAPTER 5. TRUECALLER CLUSTER MANAGEMENT SERVICE

As it might be understandable the db package exists only in those services that require a database access. Inside the db package there are other three packages:

1. dao - It contains all the Data Access Objects, classes that create an abstraction layer between the database and the business logic so that database characteristics are not exposed;

2. entity - It contains all the Entity Objects, classes that abstract the data model of a database;

3. manager - It contains all the classes that hold the business logic for querying the databases through the DAOs.

Specifically tc-common-node has, in its base package (com.truecaller.common.node) only the packages app and jersey. On the other hand, tc-common-ui has in its base package (com.truecaller.common.ui), as it would be expected, all three packages.

One of the most interesting features provided by TCM Service is without any doubt the one related to the (re-)deployment of Applications (Artifacts). It is interesting mainly because it is designed to support 100% service availability even during the redeployment phase. The whole flow, as depicted in 5.3 is the following:

1. An administrator logs into the UI web interface;
2. Selects the re-deploy entry of the menu;
3. Fills in all the relevant fields;
4. Selects the re-deploy button that trigger the UI REST API for (re-)deployment;
5. The UI service receives the requests and calls the REST API for (re-)deployment on the selected Node;
6. The Node service receives the REST API call and invokes the proper bash script;
7. The script downloads the JAR file correspondent to the version number received, sends a SIGTERM to the running process (if any), runs the new JAR and returns the result to the Node;
8. The Node returns to the UI service the result of the (re-)deployment;
9. The UI service receives the result and shows it, through the web interface, to the administrator and update accordingly the Relation table.

It is possible to run the new JAR file immediately after firing the Termination Signal on the old process since, as soon as Jetty receives it, the Connector.close() method is called and it closes the server socket so no new connections will be accepted. Existing connections will continue their execution until the end of the set delay, moment in which it is called Connector.stop(). Since the first thing that Jetty does after receiving a SIGTERM is to close the socket by running immediately the new JAR we allow the new process to register to the port so that it can start to handle the incoming requests right away.
Chapter 6
Conclusions and Future Work

This thesis project focuses on the benefits that any company can gain by migrating their system architecture from an old legacy configuration to a modern one based on services. SOA-based systems are really convenient for building complex applications over distributed platforms since it is based on the concept of SoC.

There are several differences between the traditional design and development approaches and the ones based on services, among the most important we can find that in the traditional systems, the components are tightly coupled with each other while for the service-oriented systems they are loosely coupled. Another very important difference is that in the traditional software development the reuse is applied only for convenience while in the one based on services the reuse is essential because the composition of simple services is frequently used for creating more complex ones. Finally, it is relatively easy in service-oriented systems to expose functionalities provided by legacy systems, while in traditional systems it is known to be very complex and costly due to the amount of changes required to the existing systems.

If, on the one end, the most significant benefits derived from the adoption of Service Oriented Architecture are its cost-efficiency, business agility, adaptability and implementation independence. On the other hand, some limitations and disadvantages associated to it are: large initial investment, complex service interoperability and integration, hard to fully test and arguable security due to the adoption of open standards.

Currently, the most preferred ways of implementing Service Oriented Architectures are through SOAP or REST Web Services. Lately the second ones are receiving a lot of attentions because they rely on the Web standards and
protocols and because of their lightweightness, highly efficiency and scalability.

An integral part of this thesis is the redesign of Truecaller system architecture; the new proposed infrastructure addresses completely all problems related to availability and scalability encountered in the last months. The project focuses also on the assessment of real benefits that Truecaller would gain by migrating the current monolithic Php Applications towards modern JVM based Web Services. It has been seen that with the adoption of this new Application stack the bottlenecks are easily isolated and, as a consequence, a better overall system reliability is ensured. This new scalable architecture supports also a rapid user growth, it should hold over 100 million registered users and over 10 thousand simultaneous users.

The goals of the newly designed platform are:

- Scalability - Expand the service platform when the customer base grows or usage increases;
- High availability - The service should continue to work even if a server crashes or is undergoing maintenance;
- Security - The entire service platform and customer database should be secured;
- Excellent user experience - The solution should seamlessly provide a palette of services to the users.

A new structured and rigorous pipeline is defined. The development flow, based on [CI], requires that all software engineers frequently integrate and align their work with the shared repository. This allow every engineer involved in the development to have always the latest changes and to use all the time for delivering new enhancements instead of fighting with merging procedures.

The adoption of Git as versioning system simplifies even more the development process since the clone, branch and merge procedures are extremely cheap, simple and efficient actions.

The new development flow includes also the automation of the building and testing processes; this is possible thanks to the adoption of a Jenkins server. Jenkins’ jobs, that are automatically triggered by Server-Side Git Hooks, pull the latest revision from the repository, build the uber-JAR files, run unit and integration tests and upon successful completion of all the previous steps archive the produced JAR files in the local Nexus repository.
Finally, Truecaller Cluster Management Service, a cluster management service, is developed in order to simplify all procedures related to the management of services and configuration and is going to be released as an open-source software. At Truecaller the development line is continuously active and on a daily basis there is the need to deploy the updates in both the staging and production environments. TCM helps the administrators with their operations tasks (e.g. deploy, redeploy or undeploy a service) as well as engineers by providing them, as an example, the possibility to change service configurations on the fly and monitor their effects.

The redesign of the system architecture, the adoption of a more modern approach for designing and developing services, together with a more structured and dynamic development-production pipeline and a more controlled and organized change management guarantees to Truecaller to have high quality, highly available, reliable and secure services deployed in its cluster.

In order to continue improving the scalability, reliability, availability and security of Truecaller System Architecture the following enhancement can be applied:

- Use a common service provider paradigm for network topology - Access, Distribution, Core;
- Upgrade backbone capacity from 1G to 10Gbps;
- Buy hardware load-balancers, routers and firewalls and apply the linking shown in 6.1;
- Restructure DB architecture and apply horizontal sharding;
- Deploy a catastrophe recovery architecture;
- Separate networking concerns (dedicated firewalls, dedicated routers, dedicated load-balancers, etc.) for improving DDoS resilience;
- Support ~10x yearly increase for at least 2 years without major hardware upgrades, which means:
  - 60-70000 requests / second;
  - ~4Gbps Internet traffic;
  - ~200Gbps internal traffic.
Figure 6.1: Truecaller System Architecture linking after the purchase of hardware load-balancers, routers and firewalls
There are some new functionalities that can be introduced in order to improve both the JVM based Services and TCM. The most relevant are the following:

- **Brand and release as an open-source project** Service as soon as the codebase reaches a high level of quality, security and documentation. Information on how and where to find the source code will be published on Truecaller website: [http://www.truecaller.com](http://www.truecaller.com);

- **Introduce a new service component** that periodically reports its own Metrics to a central service that persists them in a Time Series Database (TSDB). This way, it would be much more visible how the trends are going in a service and what is affected after each code/configuration change;

- **Introduce a more efficient point-to-point protocol** for the services communication like Protocol Buffers [53], Kryo [47] or Thrift [58] instead of JSON messages passing over HTTP;

- **Introduce message queues and asynchronous processing** implemented by using message passing systems like HornetQ [36] instead of using MySql and Cron Jobs;

- **Change in TCM the way of storing information** from a file based H2 database to a MySql database and introduce redundancy of the UI layer, as shown in 6.2;

- **Introduce automatized management and monitoring tasks** over TCM REST APIs;

- **Introduce an automatized capacity planning service**;

- **Change in TCM the port used by the get and set configuration functionalities** from the standard service listening Port to the Admin Port in order to raise a security precaution;

- **Introduce in TCM the possibility to start, restart and stop a certain Artifact Id**;

- **Introduce in TCM the possibility to deploy a certain Artifact Id to a cluster or set of clusters** instead of single machines;

- **Introduce in TCM a feature of automatic rollback** in case of errors in the newly deployed JAR version;
• Introduce in TCM the possibility to tail specific log files like the one of the requests or the one of the Application. An initial version could simply request for a certain number of latest lines and periodically refresh it automatically;

• Introduce in TCM the possibility to fetch the log files size;

• Introduce in TCM the possibility to fetch I/O statistics;

• Integrate a REST API explorer like Swagger [56] into TCM;

• Introduce in TCM the notion of user and roles so that functionalities are provided only to the ones that have the right privilege;

• Apply in the new Development-Production pipeline more rigorous testing processes;

Figure 6.2: Truecaller Cluster Management Service Architecture after the migration of the data store to MySql
Bibliography


BIBLIOGRAPHY


