XRAY for JIRA Cloud

André Miguel Pereira Rodrigues

Thesis to obtain the Master of Science Degree in
Information Systems and Computer Engineering

Supervisor: Prof. Rui António dos Santos Cruz

Examination Committee

Chairperson: Prof. Daniel Jorge Viegas Gonçalves
Supervisor: Prof. Rui António dos Santos Cruz
Member of the Committee: Prof. Alberto Manuel Rodrigues da Silva

June 2018
Acknowledgments

I would like to thank my parents for their continuous support and encouragement, to my amazing girlfriend for always giving me strength to finish my masters thesis. I would also like to acknowledge my dissertation supervisor Prof. Rui António dos Santos Cruz for his support, insight and sharing of knowledge that has made this Thesis possible. To all my friends and colleagues with whom I could share and discuss ideas and to Xpand IT for the amazing opportunity.
Abstract

XRAY for JIRA is an “app” that extends the functionality of JIRA, an “issue and project tracking” tool from Atlassian, to allow the user to create tests, execute the tests either manually or automatically using external tools and then view the executions in reports or gadgets to have a better overview of how their project is doing in terms of tests. The Development of XRAY for Cloud was deemed necessary due to the focus on JIRA Cloud and to the high client demand for XRAY for JIRA Cloud environment. This thesis addressed the challenge of creating a JIRA Cloud compatible version of XRAY for JIRA, as an “Atlassian Connect” add-on implemented in JavaScript using the development framework “Atlassian Connect Express”. After the development and near the start of its beta phase, the infrastructure to support it was created using Amazon Web Services and MongoDB Atlas service. XRAY for Cloud in this infrastructure was extensively tested in order to ensure that it was able to operate correctly and efficiently when subjected to a large amount of requests also guaranteeing fault tolerance, high availability and high performance.

Keywords

XRAY, JIRA, Jira Cloud, Test Management, Issue Tracking, Continuous Integration.
Resumo

XRAY para o JIRA é uma “app” que estende a funcionalidade do Jira, um sistema de gestão de problemas e gestor de projectos da Atlassian, para permitir aos utilizadores criarem tests, executarem os tests tanto manualmente como automaticamente utilizando ferramentas externas e depois verem estas execuções tanto em reports como em gadgets. O desenvolvimento do XRAY para ambiente Cloud foi considerado necessário devido ao foco que tem sido posto no “Jira Cloud” e devido ao elevado número de clientes que tem pedido XRAY para “Jira Cloud”. Esta tese fala dos desafios de criar uma versão do XRAY para Jira compatível com a versão do “Jira Cloud”, implementado em JavaScript utilizando a framework “Atlassian Connect Express”. Após o desenvolvimento ter sido acabado e perto do início da fase de beta a infra-estrutura para suportar a “app” foi criada utilizando Amazon Web Services e MongoDB Atlas Service. O XRAY para ambiente Cloud nesta infra-estrutura for testado extensivamente para garantir que funciona correctamente e eficientemente sobre uma grande quantidade de pedidos, garantindo tolerância a falhas, alta disponibilidade e elevado desempenho.

Palavras Chave

XRAY, JIRA, Jira Cloud, Gestão de Testes, Gestão de Problemas, Integração Contínua.
# Contents

1 Introduction ............................... 1
   1.1 Motivation ............................ 3
   1.2 Objectives ........................... 7
   1.3 Process and Management .......... 7
   1.4 Thesis Outline ....................... 8

2 Background .............................. 9
   2.1 Introduction ........................ 11
   2.2 Current Competition ............... 12
   2.3 Atlassian Jira ....................... 13
      2.3.1 Jira Server ..................... 13
      2.3.2 Jira Cloud ..................... 14
   2.4 Technology .......................... 15
      2.4.1 Database Technologies ......... 15
         2.4.1.A Discussion: Why MongoDB? 18
         2.4.1.B Discussion: Multi-Tenant Data Store? 18
      2.4.2 Frameworks ..................... 20
         2.4.2.A Spring Boot: ................ 21
         2.4.2.B Node.JS: .................... 22
         2.4.2.C Spring Boot vs. Node.JS: . 22
   2.5 Tests and Evaluation ............... 23

3 Implementation ......................... 25
   3.1 Idea .................................. 27
   3.2 Conceptual Implementation ........ 27
   3.3 Technical Implementation .......... 30

4 Infrastructure Setup & Deployment ... 35
   4.1 Overview ............................ 37
   4.2 Components .......................... 38
## List of Figures

1.1 Jira Project: (a) Concepts and (b) Workflow ................................................. 6

2.1 Jira Architecture ................................................................. 13

2.2 Jira Server plug-in Architecture .................................................. 14

2.3 Jira Cloud Connect plug-in Architecture ............................................. 15

2.4 XRAY for Jira Server Database Model ................................................. 19

2.5 Benchmark for Spring Boot ........................................................... 22

2.6 Benchmark for Node.JS ............................................................... 23

3.1 NodeJS Event Loop ........................................................................ 28

3.2 Jira Server plug-in Architecture ...................................................... 28

3.3 Project Structure ............................................................................ 31

4.1 XRAY Cloud Deployment ................................................................. 38

4.2 Releases ......................................................................................... 39

5.1 XRAY for Jira Cloud - Request Response Time .................................... 45

5.2 XRAY for Jira Cloud - Request Throughput ....................................... 46

5.3 XRAY for Jira Cloud - Workflow ...................................................... 47

5.4 XRAY for Jira Cloud - Jest Integration Tests ...................................... 47

5.5 XRAY for Jira Cloud - Jest Unit Tests ............................................... 48

5.6 XRAY for Jira Cloud - Sentry ............................................................ 49
Acronyms

ACID  Atomicity, Consistency, Isolation and Durability
API   Application Programming Interface
BPM   Business Process Management
CMMI  Capability Maturity Model Integration
GUI   Graphical User Interface
HTML  Hypertext Markup Language
ID    Identity Document
IaaS  Infrastructure as a Service
IO    Input/Output
IT    Information Technology
JQL   JIRA Query Language
JSON  JavaScript Object Notation
JWT   JSON Web Token
OLTP  Online Transaction Processing
OS    Operating System
PaaS  Platform as a Service
QA    Quality assurance
RDBMS Relational Database Management System
REST  Representational State Transfer
<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SaaS</td>
<td>Software as a Service</td>
</tr>
<tr>
<td>SQL</td>
<td>Structured Query Language</td>
</tr>
<tr>
<td>UI</td>
<td>User Interface</td>
</tr>
<tr>
<td>URL</td>
<td>Uniform Resource Locator</td>
</tr>
<tr>
<td>NPM</td>
<td>Node Package Manager</td>
</tr>
<tr>
<td>ACE</td>
<td>Atlassian Connect Express</td>
</tr>
<tr>
<td>PaaS</td>
<td>Platform as a Service</td>
</tr>
</tbody>
</table>
# Contents

1.1 Motivation .................................................. 3  
1.2 Objectives .................................................. 7  
1.3 Process and Management .................................. 7  
1.4 Thesis Outline ............................................. 8
1.1 Motivation

With the fast development of processing and storage technologies and the success of the Internet, computing resources have become cheaper, more powerful and more available than ever before. This technological trend has enabled the realization of a new computing model called cloud computing, in which resources are provided as general utilities that can be leased and released by users through the Internet in an on-demand fashion. Cloud computing has emerged as a new paradigm for hosting and delivering services over the Internet. Cloud computing is attractive to business owners as it eliminates the requirement for users to plan ahead for provisioning, and allows enterprises to start from the small and increase resources only when there is a rise in service demand. Cloud computing is composed of five essential characteristics, three service models, and four deployment models [1, 2].

Essential Characteristics:

- **On-demand self-service** - A consumer can unilaterally provision computing capabilities, such as server time and network storage, as needed and automatically without requiring human interaction with each service provider.

- **Broad network access** - Capabilities are available over the network and accessed through standard mechanisms that promote use by heterogeneous thin or thick client platforms (e.g., mobile devices, tablets, laptops, and workstations).

- **Resource pooling** - The provider’s computing resources are pooled to serve multiple consumers using a multi-tenant model, with different physical and virtual resources dynamically assigned and reassigned according to consumer demand. There is a sense of location independence in that the customer generally has no control or knowledge over the exact location of the provided resources but may be able to specify location at a higher level of abstraction (e.g., country, state, or data-center). Examples of resources include storage, processing, memory, and network bandwidth.

- **Rapid elasticity** - Capabilities can be rapidly and elastically provisioned, in some cases automatically, to quickly scale out and rapidly released to quickly scale in. To the consumer, the capabilities available for provisioning often appear to be unlimited and can be purchased in any quantity at any time.

- **Measured Service** - Cloud systems automatically control and optimize resource use by leveraging a metering capability at some level of abstraction appropriate to the type of service (e.g., storage, processing, bandwidth, and active user accounts). Resource usage can be monitored, controlled, and reported providing transparency for both the provider and consumer of the utilized service.
Service Models:

- **Cloud Software as a Service (SaaS)** - The capability provided to the consumer is to use the provider’s applications running on a cloud infrastructure. The applications are accessible from various client devices through a thin client interface such as a web browser (e.g., web-based email). The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, storage, or even individual application capabilities, with the possible exception of limited user-specific application configuration settings.

- **Cloud Platform as a Service (PaaS)** - The capability provided to the consumer is to deploy onto the cloud infrastructure consumer-created or acquired applications created using programming languages and tools supported by the provider. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, or storage, but has control over the deployed applications and possibly application hosting environment configurations.

- **Cloud Infrastructure as a Service (IaaS)** - The capability provided to the consumer is to provision processing, storage, networks, and other fundamental computing resources where the consumer is able to deploy and run arbitrary software, which can include operating systems and applications. The consumer does not manage or control the underlying cloud infrastructure but has control over operating systems, storage, deployed applications, and possibly limited control of select networking components (e.g., host firewalls).

Deployment Models:

- **Private cloud** - The cloud infrastructure is operated solely for an organization. It may be managed by the organization or a third party and may exist on premise or off premise.

- **Community cloud** - The cloud infrastructure is shared by several organizations and supports a specific community that has shared concerns (e.g., mission, security requirements, policy, and compliance considerations). It may be managed by the organizations or a third party and may exist on premise or off premise.

- **Public cloud** - The cloud infrastructure is made available to the general public or a large industry group and is owned by an organization selling cloud services.

- **Hybrid cloud** - The cloud infrastructure is a composition of two or more clouds (private, community, or public) that remain unique entities but are bound together by standardized or proprietary technology that enables data and application portability (e.g., cloud bursting for load-balancing between clouds).

Cloud software takes full advantage of the cloud paradigm by being service oriented with a focus on statelessness, low coupling, modularity, and semantic interoperability. In the last couple of years,
cloud computing is getting more and more companies to heavily invest in it, one of those companies is Atlassian. Atlassian is an enterprise software group of companies founded in 2002 by Mike Cannon-Brookes and Scott Farquhar, that develops products geared towards software developers and project managers. It is best known for its issue tracking application, Jira. Although commonly used for software issue tracking, due to its advanced customization features, the web application is also highly suitable for other types of ticketing systems and project management. According to Atlassian, Jira is a “workflow management system that lets you track your work in any scenario” [3].

Traditionally, issue tracking systems have been largely viewed as simple data stores where software defects are reported and tracked as bug reports within an archival database. Currently the most advanced way of dealing with software bugs is to enter them into an issue tracking system. Issue trackers address the critically important task of tracking and managing issues and bugs that emerge during a project. Issue tracking tools such as Jira, Freshdesk, Bugzilla, Pivotal, and Mantis BT are a class of project management software that keeps track of various issues for project teams. These tools are also known as project tracking tools. In software projects, these tools are often referred to as “bug tracking tools” because software defects are the main issues in the context of software development. In other types of projects, issues often mean tasks. Issue tracking systems help organizations manage issue reporting, assignment, tracking, resolution, and archiving [4].

The basic functionalities that an Issue Tracker must ensure in Software Projects, are [5]:

- Share the information across the team;
- Have an instant overview of the state of the software;
- Expertly decide about releasing;
- Set and update the importance of individual fixes and adjustments;
- Have a recorded history of changes;
- What should be fixed or created;
- What the bug symptoms and appearances are, what actually doesn’t work;
- Who reported the request, who confirmed, analyzed, implemented the solution and verified it;
- When the request was reported, when it was fixed and when verified;
- What led to the decision to choose one way of fixing instead of another;
- How long it took to handle the request.
To easily understand Jira we must know what means “Issue”, the fine-grained concept inside Jira. Due to Jira advanced customization features, different organizations use Jira to track different kinds of issues. Depending on how the organization is using Jira, an issue can represent a software bug, a project task, a help desk ticket, a leave request form, among others. Besides the concept of “Issue” there are other important Jira concepts, such as Project, Component and Workflow, as illustrated in Figures 1.1(a) and 1.1(b).

(a) Jira Concepts. Source: [6]

(b) Jira Workflow.

Figure 1.1: Jira Project: (a) Concepts and (b) Workflow

Jira allows to extend its functionality by installing plug-ins to extend the platform. The Jira installations
come with a set of pre-installed plug-ins (from Atlassian) but later other plug-ins from external companies can be installed through a marketplace provided by Atlassian. XRAY for Jira is one of those plug-ins that extends the Jira platform. It was created on December 2013 by XpandIT, a Portuguese global company specialized in strategic planning, consulting, implementation and maintenance of enterprise software, fully adapted to the customers needs. They have services and products in several areas such as Business Intelligence, Big Data, Business Process Management (BPM) and Enterprise Middleware and Collaborative Platforms. XpandIT stands out for its innovative approach fully supported by tools, processes and agile methodologies, fully mapped with Capability Maturity Model Integration (CMMI).

XRAY for Jira was created to provide for Jira users a way to perform Test Management directly from inside Jira. To achieve this objective, XRAY allows the users to create test suites that can later be executed manually or automatically using various continuous integrations tools. This executions can then be use in Jira gadgets or generate reports to provide a better overview of how the project is performing in terms of tests.

1.2 Objectives

The plug-ins for Jira are divided into three types: (a) plug-ins Type-1, (b) plug-ins Type-2, and (c) Atlassian Connect plug-ins (Jira Cloud). Only Atlassian Connect plug-ins can be used/installed in instances of Jira Cloud. Because of Atlassian bet and investment in cloud products, much of the Jira Server customers are migrating to Jira Cloud and as such there is a need to create a new version of XRAY for Jira (plug-in Type-2) as an Atlassian Connect plug-in so that it can be installed on Jira Cloud and thereby being sold to new Jira cloud customers or to customers who have migrated from server version to cloud version.

It was therefore the objective of the work for this thesis, to develop a prototype of XRAY, but for the Jira Cloud platform.

1.3 Process and Management

The development of Jira for Jira Cloud was planned and managed using Jira as Issue Tracking tool and ticketing system. All documentation is created/updated at the Confluence (Atlassian tool) and will be available to the public. Nowadays it is crucial to develop software over a version control system, and in the case of XRAY, Bitbucket was used, which is also a tool owned by Atlassian. During the development, a specific methodology was adopted, the XPAGILE, a methodology based on agile principles and simultaneously mapped with CMMI level 2. Bamboo (Atlassian tool) was used as continuous integration tool to automate the release management creating a continuous delivery pipeline and as continuous Testing
tool where the automated tests can be defined and executed based on specific triggers, e.g., commit, release or even manual trigger.

This development is being done in a team environment. Initially the team started with just 2 members: Bruno Conde, who is the Project Manager of XRAY for Jira Cloud and responsible for deciding how features should be implemented and which frameworks should be used, although typically these decisions are always discussed and decided by the team, and myself, as developer of XRAY for Jira Cloud, responsible for developing the features and also participate in discussions related with feature and framework decisions. Later two other members joined the team: Sérgio Vieira, developer of XRAY for Jira Cloud responsible, like myself, for the development of new features and frequently responsible for developing complex aggregate queries for MongoDB, and José Sousa, developer of XRAY for Jira Cloud responsible for the development of new features and due to his preexisting knowledge, responsible for configuring many of the external systems used in XRAY for Jira Cloud.

1.4 Thesis Outline

The remaining of the document is organized as follows.

Chapter 2 begins by describing XRAY for Jira progress in Atlassian marketplace and discussed the expected competition of XRAY for Jira Cloud on Jira Cloud marketplace. Will present the Jira Cloud and Jira Server architectures, their advantages and disadvantages according to the customer’s point of view as well as their respective markets. After that is identified the available technologies and they advantages and disadvantages. Finally, at the end will be discussed and planned the tests and the maintenance strategy for XRAY.

Chapter 3 contains a detailed description of XRAY for Jira Cloud implementation. It begins by making an analyze of the XRAY for Jira Server modules and understanding what could be possibly used in cloud version and which changes were needed to do it. All the major components will be described as well as the required communication between them to get all working as expected.

Chapter 4 presents and explains the cloud computing platform architecture where the XRAY for Jira Cloud will be running. Will be described all the components, their communication, configuration, security rules and strategies to assure high availability, scalability, great performance and security.

Chapter 5 reports and analyzes the difficulties in each project phase and how they were overcome, what went well and what could have been approached differently and an overview of the performed tests and the process and maintenance strategies that were followed during development.

Finally, Chapter 6 concludes this dissertation, by resuming the key results achieved and discussing possible directions for future work.
2 Background

Contents

2.1 Introduction ......................................................... 11
2.2 Current Competition .............................................. 12
2.3 Atlassian Jira ..................................................... 13
2.4 Technology ......................................................... 15
2.5 Tests and Evaluation ............................................. 23
This chapter will start by making a brief overview on the progress and development of XRAY for Jira in the Atlassian Marketplace, and subsequently present the Jira Cloud and Jira Server architectures, the advantages and disadvantages of each one according to the customer’s point of view as well as their respective markets. It will then analyze the current XRAY for Jira Cloud competition on Jira Cloud marketplace. The chapter concludes with the technological issues concerning the development of XRAY for Jira Cloud, the database model that best suits this use case, the appropriate development frameworks and the reasons for the choices made.

2.1 Introduction

As explained earlier, Jira is a highly flexible Issue tracking software that allows to extend its functionalities by installing plug-ins developed by external companies, e.g. XRAY for Jira, which are obtained in the marketplace Atlassian. XRAY for Jira was first developed and released through the Atlassian marketplace in 2013. That first version of the plug-in was compatible with versions 5.2 to 6.1.4 of Jira server. Currently XRAY for Jira has a total of 2,511 active customer installations all over the globe, with the biggest clients in countries such as the United States of America, Germany and United Kingdom.

In addition to the new versions of its server products, Atlassian has been increasingly focusing on their cloud versions. This ambition has to do especially with the ease and speed a customer can begin enjoying Jira, the facilities of integration with other Atlassian cloud products and the fact that customers do not have to worry about buying and maintaining the infrastructure.

Developing a plug-in for Jira Cloud brings many limitations versus developing a Type-2 plug-in, meaning that, there are some features already implement in XRAY for Jira Server that currently are not be possible to be implemented in XRAY for Jira Cloud. The reasons for these limitations are that an Atlassian Connect plug-in is much more limited to the data it can access when it is running in the cloud, which means that the Jira Cloud Server is going to be hosted in a different place from the XRAY for Jira cloud Servers, and the only way for the plug-ins to communicate with the Jira server is by Representational State Transfer (REST) Application Programming Interface (API), making every operation to take extra time and adding the risk of requests being lost. Another big reason for the limitations present in the Atlassian Connect plug-ins are that Atlassian does not want foreign code to be running inside their servers, so features like custom fields types, JIRA Query Language (JQL) functions, or configuration of screens that were available in the Jira Server, are no longer available for Jira Cloud plug-ins to create or extend. All these limitations have forced us to rethink how XRAY for Jira Cloud could be implemented and to not include key features that were available in XRAY for Jira Server in the Cloud version.

These limitations are currently prescribed by Atlassian plug-ins for the type Atlassian Connect but may in the future become available, which also forces us to implement and design the final solution
keeping in mind that these features might be possible to implement in the future and if so, they shouldn’t impact the application in a significant way.

2.2 Current Competition

In the XRAY for Jira version for Server, there are many competitors in the Atlassian Marketplace, being the ones with the larger customer base, excluding XRAY for Jira:

- Zephyr for Jira–Test Management
- Behave Pro for Jira Agile
- SynapseRT–Test management & Quality assurance (QA) in Jira
- Test Management for Jira, previously known as Kanoah Tests
- TestFLO–Test Management for Jira

Even though XRAY for Jira has many competitors in Jira Server the same does not happen in the Cloud version because not all of the previously mentioned Test Management add-ons have a Jira Cloud version and even if they do they are lite versions, which end up just being Cloud plug-ins with a very limited amount of features when compared to server versions of those plug-ins.

The only real competitor for XRAY for Jira Cloud is Zephyr. Zephyr is one of the first Test Management plug-ins for Jira Server, and currently the only Test Management plug-in with more customers than XRAY on the Atlassian Marketplace. Like XRAY for Jira, Zephyr allows the users to perform all their Test Management directly from Jira by allowing to create Tests and executions which can later be used in gadgets and reports to extract Metrics and to provide the users with a quick overview of the state of the project. The biggest difference between XRAY for Jira and Zephyr are on automated Tests, something that Zephyr does not currently fully support. Even though Zephyr can do continuous integration, it is not as specific and simple as XRAY’s and the tools provided to users to perform those tasks are sold separately while in XRAY for Jira they are either inside the plug-in, for example the REST API, or are free tools for other platforms such as Bamboo, Jenkins or Maven which will then complement the Jira plug-in.

In general even though some solutions already exist in the Atlassian marketplace for Jira Cloud for Test Management, they are not as complete or powerful as XRAY for Jira will be, but the fact that they already exist will make it harder for XRAY for Jira to grow its customer base in the Jira Cloud side of the Atlassian marketplace.
2.3 Atlassian Jira

Jira is available in two distinct options: Jira Server and Jira Cloud SaaS. An overview of general Jira architecture is shown in Figure 2.1.

![Figure 2.1: Jira Architecture. Source: [7]](image)

2.3.1 Jira Server

Jira Server was the first solution released and currently still is the one with more clients. The reason why most of the customers still continue to use Jira Server is related essentially to the following:

- Source code control;
- More flexibility and customization options;
- More third-party plug-ins available from marketplace;
- Upgrades control;
- No admin restrictions;
- Full access to databases.

An overview of the Jira Server plug-ins architecture is shown in Figure 3.2.
2.3.2 **Jira Cloud**

Jira Cloud, formerly known as OnDemand is where Atlassian is investing more. The main reason for that is the simplicity in environment setup, the little or no concern with infrastructure issues, and, of course, the speed and reliability of cloud-based applications [8]. The most important points of Jira Cloud are:

- High scalability and availability.
- Own infrastructure not needed.
- No additional hardware requirements and associated costs.
- No additional work on the own Information Technology (IT) department.
- No maintenance tasks needed.
- Systems can be used quickly and in a short term.

The Jira Cloud connect plug-ins architecture is shown in Figure 2.3.

The Jira Cloud marketplace is still smaller than its server counterpart, and with more customers starting to migrate from Jira Server to Jira Cloud, it is imperative that XRAY for Jira is released as soon as possible to the Cloud market so it can ensure that the clients previously using XRAY for Jira on Jira server can continue to do so in the Cloud version, and to make sure to capture all the new possible clients that are looking for a Test Management Solution in the Jira Cloud.
2.4 Technology

During XRAY for Jira Cloud development some technological challenges were found. These challenges are related to several factors: the differences between the plug-ins and Connect Frameworks (used to development Jira Server and Jira Cloud add-ons respectively), the lack of experience from the developers, the addition of new developers during the development and also with the fact that the new architecture has brought concerns and big limitations that did not exist before. In Jira Cloud add-ons, vendors are responsible for setting up the platform where the add-on will be running, and must manage all the add-on data and settings.

2.4.1 Database Technologies

Database technologies are typically of type Relational or Non-Relational.

Relational databases can also be called Relational Database Management Systems (RDBMSs) or Structured Query Language (SQL) databases. The most popular are Microsoft SQL Server, Oracle Database, MySQL, Postgres and IBM DB2. These RDBMSs are mostly used in large enterprise scenarios, with the exception of MySQL, which is mostly used to store data for web applications, typically as part of the popular LAMP stack (Linux, Apache, MySQL, PHP/Python/Perl).

Non-relational databases are also called NoSQL databases, wit the most popular being MongoDB, DocumentDB, Cassandra, Coachbase, HBase, Redis, and Neo4j. These databases are usually grouped into four categories: (a) Key-value stores; (b) Graph stores; (c) Column stores, and (d) Document stores.

1. **Relational databases**: The reasons for the dominance of relational databases are simplicity, robustness, flexibility, performance, scalability and compatibility in managing generic data. For large databases, especially the ones used for web applications, the main concern is scalability [10].

![Figure 2.3: Jira Cloud Connect plug-in Architecture. Source: [9]](image-url)
As more and more applications are created in environments that have massive workloads (e.g., Amazon), their scalability requirements can change very quickly and grow very large. Relational databases scale well, but usually only when that scaling happens on a single server (scale-up). When the capacity of that single server is reached, a scale-out is required in order to distribute the load across multiple servers, moving into so-called distributed computing. This is when the complexity of relational databases starts to cause problems with their potential to scale [11], as when trying to scale to hundreds or thousands of servers, the complexities become overwhelming. The characteristics that make relational databases so appealing are the very same that also drastically reduce their viability as platforms for large distributed systems.

Relational-model databases can be tweaked and set up to run large-scale read-only operations through data warehousing, and thus potentially serve a large amount of users. But data warehouses are distinct from typical databases in that they are used for more complex analysis of data. This differs from the transactional database paradigm, whose main use is to support operational systems and offer day-to-day, small scale reporting.

2. Non-Relational Databases:

   (a) **Key-value stores:** These databases pair keys to values. An analogy is of a file system where the path acts as the key and the content acts as the file. There are usually no fields to update, instead, the entire value other than the key must be updated if changes are to be made. It scales well but it can limit the complexity of the queries and other advanced features. Examples are: Dynamo, MemcacheDB, Redis, Riak, FairCom c-treeACE, Aerospike, OrientDB, MUMPS, HyperDex, Azure Table Storage.

   (b) **Graph stores:** These databases excel at dealing with interconnected data. Graph databases consist of connections, or edges, between nodes. Both nodes and their edges can store additional properties such as key-value pairs. The strength of a graph database is in traversing the connections between the nodes. But they generally require all data to fit onto one machine, limiting their scalability. Examples include: Allegro, Neo4J, InfiniteGraph, OrientDB, Virtuoso, Stardog, Sesame.

   (c) **Column stores:** Relational databases store all the data in a particular table rows together on-disk, making retrieval of a particular row fast. Column-family databases generally serialize all the values of a particular column together on-disk, which makes retrieval of a large amount of a specific attribute fast. This approach lends itself well to aggregate queries and analytics scenarios where run-range queries can run over a specific field. Examples include: Accumulo, Cassandra, Druid, HBase, Vertica.
(d) **Document stores:** These databases store records as documents, where a document can generally be thought of as a grouping of key-value pairs (it has nothing to do with storing actual documents such as a Word document). Keys are always strings, and values can be stored as strings, numeric, booleans, arrays, and other nested key-value pairs. Values can be nested to arbitrary depths. In a document database, each document carries its own schema unlike an RDBMS, in which every row in a given table must have the same columns. Examples: Lotus Notes, Clusterpoint, Apache CouchDB, Couchbase, MarkLogic, MongoDB, OrientDB, Qizx, Cloudant, Azure DocumentDB.

NoSQL databases are increasingly used in “big data” and real-time web applications. They became popular with the introduction of the web, when databases went from a maximum of a few hundred users on an internal company application to thousands or millions of users on a web application. NoSQL systems are also called “Not only SQL” to emphasize that they may also support SQL-like query languages. Many NoSQL stores compromise consistency in favor of availability and partition tolerance. Some of the reasons that block adoption of NoSQL stores include the use of low-level query languages, the lack of standardized interfaces, and huge investments in existing SQL. Also, most NoSQL stores lack true Atomicity, Consistency, Isolation and Durability (ACID) transactions or only support transactions in certain circumstances and at certain levels. Finally, RDBMSs are usually much simpler to use as they have Graphical User Interfaces (GUIs) where many NoSQL solution use a command-line interface.

The NoSQL databases Motivations regarding web applications, can be the following:

- Simplicity of design;
- Better horizontal scaling to clusters of machines—NoSQL databases automatically spread data across servers without requiring application changes (auto-sharding), meaning that they natively and automatically spread data across an arbitrary number of servers;
- Finer control over availability;
- Schema-on-read instead of schema-on-write;
- Speed;
- Cost.

RDBMSs suffer from no horizontal scaling for high transaction loads (millions of read-writes), while NoSQL databases solve high transaction loads but at the cost of data integrity and joins [12]. The bottom line for using a NoSQL solution is the case of an Online Transaction Processing (OLTP) application having thousands of users and a very large database requiring a scale-out solution and/or is using JavaScript Object Notation (JSON) data, in particular if this JSON data has various structures. The
additional benefit is high availability, as NoSQL solutions store multiple copies of the data. However, in order to achieve better performance, data consistency might be sacrificed, as well as the ability to join data, use SQL, and do quick mass updates.

2.4.1.A Discussion: Why MongoDB?

Initially, since Jira Cloud provides add-ons with some storage in the form of entity properties, not having a database was a possibility, but after some more research a limitation was found, related with the character limit of each entity property, which meant that if using entity properties to store data the application would grow increasingly more complex since it would have to deal with the data being split along multiple entity properties. This option was quickly dropped.

XRAY for Jira is a very data intensive plug-in and since each client can have multiple users, a very high demand from the data is highly probable. Therefore, the choice falls for a NoSQL database engine, such as MongoDB, since this type of database can scale horizontally allowing the data (and subsequent requests) to be partitioned or sharded though multiple nodes [13]. The main concern about adopting a NoSQL database for this purpose is that some queries or the current database model may not work in such databases. Figure 2.4, illustrates the XRAY for Jira database model, which is quite complex. As is, this model could not be converted to work on MongoDB database model, but after some research and consulting with MongoDB specialists inside XpandIT it was possible to conclude that with a few changes to the model it would be possible to convert it for MongoDB.

2.4.1.B Discussion: Multi-Tenant Data Store?

After choosing a database model there is still the question about how the data for each individual client would be stored [14]. There are three choices for a Multi-Tenant architecture:

1. **All tenants in a single database**–Store all tenants in the same database, identifying each row/document by tenant Identity Document (ID);

   - **Advantages:**
     - This is a relatively simple strategy in terms of implementation. Of course, the code needs to filter by tenantID in each query, but we consider this not to add much complexity;
     - Horizontal scaling is done with sharding by tenantID.

   - **Disadvantages:**
     - Data security concerns;
     - Backup or restore individual tenants;
     - If using MongoDB, performance may degrade if indexes do not fit in memory.
2. **Collections per tenant**—Create a separate schema (in relational databases) or a separate collection (in noSQL databases) by tenant;

   - **Advantages:**
     - This is a relatively simple strategy in terms of implementation. Of course, the code needs to use the appropriate collection for searching tenants;
     - Backups and restore by tenant ID would be simpler than the previous solution.

   - **Disadvantages:**
     - Data security concerns;
     - Not ideal for scaling through sharding;
     - There is a limitation with the number of collections within a database. This only happens if we do not use the new WiredTiger storage component.

3. **Database per tenant**—Create a separate database by tenant.

   - **Advantages:**
– Best for isolating data for different tenants;
– Backups and restores are very easy.

• Disadvantages:
  – This is the most complex strategy in terms of implementation. We would have to manage
    the horizontal scaling by the different available nodes. This requires node discoverability,
    a control database and managing where a tenant database should be created. Balancing
    tenants could also be required;
  – Each database has a minimum of 32 MB of storage, doubling in size if there is any data
    in it.

Each option has its advantages and disadvantages [15] in the XRAY Context. Although there might
be even other advantages and disadvantages to consider in these approaches, they are not applicable
in the context of this project.

Therefore, a Database per tenant seems to fit better in this project as it allows to isolate the data
for different tenants, meaning that, even if something goes wrong with the data stores in the database
it would not affect the other clients and the problem would be easily solved with a restore which this
model also allows to be done in a fairly easy way. One other reason for this choice is that it allows XRAY
for Jira to start small with just one machine for database and then scale by doing sharding. As for the
disadvantage of this solution, is that it is more complex in terms of implementation. In a proof of concept
for this solution that was already realized, it was concluded that, in terms of implementation, all it takes
is to change the database.

2.4.2 Frameworks

To help in the development of Atlassian Connect plug-ins, Atlassian provides a framework available in
several languages such as JS Node, Java (Spring Boot), .NET, Play (Java), Play (Scala) among others
with several functionalities:

• Serve descriptor and add-on User Interface (UI)

• Handle add-on installation

• Persistent store

• Handle add-on request

• JSON Web Token (JWT) token handler [16]

• Crypto library
In case the development is made in another stack (e.g., Rails, Django, PHP), it is up to the developer to ensure those functionalities are available. Therefore, it is preferable to opt for using the Atlassian provided framework in the language the developer feels most confident, allowing a faster development, and in case of limitations or doubts, it facilitates the exchange of ideas or exposing problems in the community also using that framework.

Even though Atlassian provides a framework in those several languages, the only officially supported are Node.JS and Spring Boot (Java). The other frameworks are supported by communities not related with Atlassian, and so, for XRAY for Jira the choice was between Spring Boot and Node.JS.

**Non-blocking Input/Output (IO):** Over the past years, there has been an increase in concerns when building high demanding application servers. This basically means that, when performing IO tasks, there would be no need to wait for the results by blocking threads. The Operating System (OS) is responsible to signal the process when the IO operation completes. If threads are waiting for the IO tasks to complete, no other incoming requests can be answered, even if the server computation power is low. Non-blocking IO technologies/frameworks allow to maximize resources usage avoiding to scale horizontally when there is not such a need.

Most applications spend their entire time performing IO operations and XRAY for Jira cloud will fall into this category as well. There will be large amounts of IO requests either accessing the database or querying Jira for additional information.

### 2.4.2.A Spring Boot:

Spring Boot is based in servlet containers version before V3.1, namely Tomcat. Hence non-blocking IO is not yet supported natively. Spring v5 will support non-blocking IO but unfortunately:

1. v5 was expected to be released at the end of 2017;

2. the Atlassian connect bootstrap library does not support Spring v5 (nor non-blocking IO).

The main advantages in using Spring boot are:

- Java language, strongly typed;
- Very fast;
- Big community, existing applications, existing libs, and more mature overall;
- XRAY for Jira Server was implemented in Java which means developers already had more experience in Java.
2.4.2.B  Node.JS:

Node.js is built with non-blocking IO as a requirement. Every request will be processed in the same thread so this cannot be blocked in any way. IO operations are directly handled by IO signals or separate threads. Hence Node.js will minimize latency and maximize resource consumption [17]. The main disadvantages of Node.js are:

1. Javascript language, natively not a typed language although typescript can be used to a similar effect;
2. Very slow when compared with Java;
3. Non-blocking programming model. Even if this is an advantage, this programming model is more complex when compared with a simple thread-based web application.

The main advantages in using Node.js are:

- Working with JSON based models is very easy and non intrusive;
- It is more natural to use MongoDB with a dynamic language as Javascript;
- Non-blocking IO

![Response Time Graph](image)

**Figure 2.5:** Benchmark for Spring Boot

2.4.2.C  Spring Boot vs. Node.JS:

Before making a decision about the framework to use in XRAY for Jira Cloud some benchmark tests (Figures 2.5 and 2.6) were performed to test which one would handle better the requests that the server
would receive.

Figure 2.6: Benchmark for Node.JS

From the tests it can be observed that, when handling a small to medium amount of requests, Spring boot is the winner, mainly due to Java being faster than Javascript. But when in a more realistic scenario with a high amount of requests it can be observed that the time that Node.JS takes to respond to requests is more or less constant. In the Spring Boot case, not only the average time to respond to a request is twice as much as Node.JS’s but also the time is not constant, essentially due to garbage collections and other clean up tasks adjacent to Java language. Based on the test the conclusion for the best framework, based on XRAY for Jira requirements, would be Node.JS.

2.5 Tests and Evaluation

A very important piece in software development is the definition of a suite of tests which are applied at different stages of the product life-cycle to guarantee the quality, the consistency and the reliability of the product, to ensure customers satisfaction, to reduce maintenance costs and to also ensure that the application should not result into any failures, as those would become very expensive in later stages of the development cycle [18]. In the development of XRAY for Jira Cloud the following test suites were considered to be built:

- **Development Testing**: Development testing is the consistent application of software testing practices that involves synchronized application of defect prevention and detection strategies in order to reduce software development risks, time, and costs. To achieve that, in XRAY for Jira development, tools like ESLint [19] will be used.
• **Continuous Testing:** Continuous testing is the process of executing automated tests as part of the software delivery cycle to obtain immediate feedback of the global product status. In XRAY for Jira this will be achieved by having a set of automated tests that will perform actions in XRAY for Jira which will then be compared with the expected results.

• **Regression and Non-Regression Testing:** The intent of regression testing is to assure that in the process of fixing a defect, no existing functionality has been broken. Non-regression testing is performed to test that an intentional change has had the desired effect. In XRAY for Jira, this is achieved in two distinct ways: by having in the issue work-flow a testing stage where the QA team will be responsible for verifying the issue along with a separate testing project which will contain detailed manual tests, for each specific issue, which will be performed by the QA team. These tests will be specified in Jira using XRAY for Jira Server, and by having a set of automated tests and by comparing their results with the expected results (also used to assure Continuous Testing).

• **Integration Testing:** The main function or goal of Integration testing is to test the interfaces between the units/modules. Regarding XRAY for Jira, unit tests will be set up to verify the correct operation of all the modules as well as integration with other plug-ins and with the Jira Cloud instances.

• **Web Testing:** Web testing is the name given to software testing that focuses on web applications and it can help address issues before the system is revealed to the public. In XRAY for Jira, this kind of testing is achieved by using the Selenium tool [20] or the Robot Framework [21] that simulate events (like a click) and compares the generated Hypertext Markup Language (HTML) code with the expected result.
3 Implementation

Contents

3.1 Idea ................................................. 27
3.2 Conceptual Implementation .............................. 27
3.3 Technical Implementation ................................ 30
In this chapter the final implementation and architecture of XRAY for Jira Cloud as well as all the components and their interactions will be explained along with the major difficulties encountered.

3.1 Idea

The idea behind the implementation that was made, was to develop a web application that will integrate with another service/platform (in this case the Jira cloud platform), providing a set of new features to that platform. This web application should be as transparent as possible to the users of that system to give them feeling that they are only using the original platform when in fact they are using not only the platform, but the web application XRAY for Jira Cloud. Both the web application and the platform should communicate, with each other, in a secure way. In this case the communication channel used is REST APIs, running in complete isolation from each other. The platform/service should behave like a multi-process architecture that allows independent sandboxed apps to extends parts of it.

3.2 Conceptual Implementation

As mentioned before, XRAY for Jira Cloud has been implemented as an Atlassian Connect app. We have chosen Atlassian Connect Express, one of two official frameworks provided by Atlassian, to help in the development process using Node.js. Node.js, also known as Node, is a JavaScript runtime platform. It is based on Google’s runtime implementation (V8 engine). V8 and Node are mostly implemented in C and C++, focusing on performance and low memory consumption. Whereas V8 supports mainly Javascript in the browser, Node aims to support long-running server processes. Unlike in most other modern environments, a Node process does not rely on multi-threading to support concurrent execution of business logic, although mechanisms exist to allow Node to run on multi-thread [22]. It is based on an asynchronous I/O eventing model, shown in Figure 3.1 [23,24]. Node server process can be thought of as a single-threaded daemon that embeds the Javascript engine.

At the beginning we have started by analyzing the existing XRAY for Jira Server components to identify the elements that could possibly be reused in the Cloud version. We rapidly reached the conclusion that everything would have to be done from scratch due to the big differences in technologies, as the Server version used Java for the server side and backbone and JS for the interface while its Cloud counterpart used Node for the Server side and Handlebars and React for the interface, and even bigger differences in architecture, since now all apps data needs to be stored and managed outside of Jira, all the apps screens and dialogs must be rendered and sent to iframes, that will be presented to the end users by Jira, and all the data must now be retrieved and sent by REST APIs since the java API is no longer available in the Cloud Version.
The architecture of the Connect apps can be observed in Figure 3.2:

An Atlassian Connect app is a multi-tenant web application that operates remotely over HTTPS in a Sandboxed environment inside Atlassian cloud instances that can be written in any programming language and web framework, with two of them being officially supported. They have three major capabilities:

1. Add content in defined and limited Atlassian application UI places (inside iframes);
2. Using Atlassian REST APIs to access and send data;
3. Listen to WebHooks (events) fired by Atlassian application (which are not guaranteed to arrive).
As it was previously mentioned, with this new architecture and without the Java API, all data accesses are made using authenticated REST API calls. The mechanism used is a technology called JWT that authenticates the apps [16, 26]. When the app is installed, a security context is exchanged; this context is then used to create and validate all future JWT tokens, embedded in the API calls. The use of JWT tokens guarantees that:

- Atlassian application can verify that it is talking to the app and the same for the add (authenticity);
- The HTTP request has not be tampered with, which means none of the query parameters, path or HTTP method have been altered (integrity).

After understanding the major differences between plugins type-2 and the connect apps, the stack, the frameworks and technologies that are to be used in the development and the required components needed, are the MongoDB database, the Node Server and the handlebars and React to handle the client UI rendering. There were still be some questions and concerns left such as, How can the app be installed? Is there a way to run it and test it in a local environment and, if not, what are the alternatives? The Atlassian documentation gave us answers to all the questions.

Related to the first questions we found that the app needs a descriptor [27]. This descriptor is a JSON file that informs the Jira Cloud application about: where the app is hosted; which permissions the app requires (the permissions are called scopes); which modules the app intends to extend and which Web-Hooks it wants to receive and to where they should be sent. The scopes are a concept unique to Atlassian Cloud instances needed to improve security and transparency, by forcing the apps to declare what type of access they require from the Atlassian application, which will then define what data can be accessed and modified by the app and will also allow Jira Cloud administrators the possibility to review and approve, or not, those permissions.

Regarding local development, although Atlassian provides the developers with ways to run the Jira Cloud application in a local machine, we found that this approach is not ideal since both the app and the Atlassian application would be running in the same machine and we would rather develop on an environment much closer to the end environment. So we opted for using the free development instance that Atlassian provides the developers, which has a very easy setup:

1. **Registering a development version of Jira Cloud**—Atlassian provides a free 1 user tier cloud instance for developers.

2. **Enable development Mode**—In the apps page, enable the Development Mode option

3. **Install the app**—This can be done manually by specifying the app descriptor Uniform Resource Locator (URL) to the Atlassian application or since we decided to use the Atlassian Connect Express this step can be simply accomplished by specifying the user credentials and the Atlassian
application URL in a configuration file of our app which will then be used to inform our app where it should install itself.

Like it was originally intended with this approach it is possible to develop it in a way that almost perfectly simulates the production environment. In summary, to begin the development of the XRAY for Jira Cloud each developer has prepared a development environment, as mentioned above, created an Express project, a descriptor that follows the Atlassian specifications and started developing the business logic of the app.

### 3.3 Technical Implementation

Projects skeletons were created using the help of Atlassian Connect Express (ACE). ACE provides a vast library of middleware, tools and helpers that make it easier to build Atlassian apps, more specifically:

1. A way to speed the development loop and handling registration and de-registration on the target Atlassian applications at start-up and shutdown;

2. A file system watch, that detects changes to the app descriptor (Atlassian-connect.json), and that, when changes are detected, the app is re-registered with the host or hosted. We have later disabled this, as it was causing unexpected error, forcing us to have to develop MongoDB tables to ensure the app could be re-registered;

3. Automatic JWT authentication both on inbound and outbound requests;

4. Automatic persistence of host details (on the app database), like client key, host public key, host base URL and more;

5. Local tunnelling server (using ngrok) for testing and development purposes;

To create the project skeleton we needed to install Node.js and Node Package Manager (NPM), after what we installed Atlas-Connect (ACE client tool), created the project skeleton using it and then install all the project dependencies; this was achieved by running the following set of commands on a console:

1. `npm i -g atlas-connect`

2. `atlas-connect new xraycloud`

3. `npm install`

After the skeleton project created with added a new file, `credentials.json`, where we added the Atlassian application URL and the user credentials. At the start of the development process we started with the default project structure created by Atlas-Connect but then have to change it to something that
better suits our development and organization. In Figure 3.3 the final project structure is illustrated, along with the most important components which are marked and are responsible for:

![Figure 3.3: Project Structure](image)

1. The `node_modules` folder which contains all the server and client side dependencies and modules for our project, they are managed by NPM.

2. The `client` folder which contains all client logics and is sub divided as follows:
   - `css` – which contains all the css files;
   - `fonts` – which contains all the font files
   - `images` – which contains all the image files
   - `js` – which contains all the Javascript files in which the client logics lives in.

3. The `server` folder which contains all server logics along with the handlebar files used to render the views and is subdivided as follows:
• events – which contains all events used in the application;
• locales – which contains all internalization files;
• routes – which contains all the REST API modules used between the client and the server;
• services – which contains most of the server side logic;
• store – which contains all the repository files used to communicate with the database;
• util – which contains all utility functions and modules;
• views – which contains all the handlebar files used to render the views;

4. The atlassian-connect.json file is the add-on descriptor, previously mentioned;

5. The config.json file contains the configuration for each run-time environment the plugin runs in;

6. The credentials.json file is where the authentication data and URL of target instances are specified so that ACE can automatically register the add-on;

7. The package.json file holds several relevant metadata to the project as well as the list of server and client side dependencies;

8. The __test__ folder holds all the jest tests; these tests focus on testing and ensuring the React Components work as intended.

After understanding all that concepts, XRAY for Jira Cloud development was started. In atlassian-connect.json were defined all the Jira UI modules, scopes and routes that will be responsible to handle each UI module. This routes will receive Webhooks containing information about the Jira instance and have the job of rendering the module accordingly. To do that, the routes firstly verify the license and the JWT from the request using atlassian-connect-express middleware (JWT must be authenticated and authorized), then if they need additional data they can use the REST APIs to obtain data from Jira, although whenever possible REST API calls to Jira are kept on the client side to try to keep the server load to a minimum, or query the MongoDB database to obtain add-on related data. After that, the required context data is passed to a template engine file and finally that page is rendered and sent back as the request response.

All the views must import the Atlassian Connect JavaScript client library that establishes the cross domain messaging bridge with parent iframes and provides several methods and objects that could be used in views without making a trip back to the add-on server. There are some occasions where the client side needs to make REST API calls to the Server, to perform some actions; all these REST APIs are declared in separate routes to keep the code organized. To make these calls from the client to the server it is necessary that the request is authenticated and authorized; the behavior is almost the same, except that in this case the JWT does not exists in the request once it does not come from the Jira
instance. Luckily, ACE generates the JWT as a context variable and place it in our views (client-side) which the add-on can verify later using the middleware (server-side).

After declaring all the modules that will be extended, defining and implementing both the view routes and the action routes, implementing all the business logic and solving other features such as, centralized logging, interface internationalization (using 118n module from NPM) and performance and usage metrics collecting, the first version of XRAY for Jira Cloud was complete.
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1 Overview</td>
<td>37</td>
</tr>
<tr>
<td>4.2 Components</td>
<td>38</td>
</tr>
</tbody>
</table>
In order to make XRAY for Jira Cloud available to the public, that might be around the globe, it needs to be running in a scalable and high performant environment that ensures:

• High Availability – since XRAY for Jira Cloud will be available across to clients from any country in the world;

• Fault-tolerance – since XRAY for Jira Cloud should always be available and working;

• High Performant – since the app will receive a high number of requests and is expected to maintain great performance.

To ensure high availability the app will be running in several nodes, completely isolated from each other and preferably in different geographic locations, in order to reduce the latency the clients will experience, and thus increase their overall experience. Initially XRAY for Jira Cloud will be deployed in the US and the reason for this is that Atlassian Jira Cloud is also deployed in AMAZON AWS in the US region and we want to reduce the latency of the requests between XRAY and the Jira Cloud instance.

There should also exist a mechanism responsible for checking the nodes status, and in the case of one of them being down or answering incorrectly, immediately start another, thus providing fault tolerance to the system. That way the architecture could survive to catastrophes that affect only one location and to errors that affect only some nodes. Good performance should be achieved by dynamically allocate resources, starting new nodes, as they are required and release them afterwards, for example, launching more instances to answer a high amount of requests, and when the work load stabilizes remove some nodes in order to lower the hosting costs [28].

The first major decision was to choose what would be the best computing cloud hosting platform to implement and host the XRAY architecture. Despite the difficulty to make a comparison between cloud computing platforms due to the constantly arise of new services and changes in costs and performance, we can reduce our options to the two major existing platforms: Amazon Web Services (AWS) and Microsoft’s Azure. Considering price, performance, and support, these options are very similar. Due to the wide range of services (which may be useful in the future), platform maturity, the interest of our company to have a project running using their services and our company already having another project running using their services, the choice was AWS.

4.1 Overview

After defining the requirements, objectives and goals of XRAY for Jira Cloud the architecture chosen is the one illustrated in Figure 4.1. The decisions related with this architecture were made to guarantee the 3 points already mentioned above: high availability, fault-tolerance and high performance. In order to achieve these requirements the architecture needs to guarantee good horizontal scalability. In a
horizontal scaling model, instead of increasing the capacity of each individual actor (vertical scaling) in the system, we simply add more actors to the system.

This is a natural choice due to the database model that will be used in XRAY for Jira Cloud, since MongoDB already provides auto-sharding for horizontal scale-out and with its native replication and automatic leader election providing high availability which is one of the requirements of XRAY for Jira Cloud. XRAY for Jira Cloud will receive a high volume of concurrent requests that will need to answered in a quick manner and for that it is required to distribute them between different cores and even different servers. Horizontal scale fits perfectly in this scenario because it is easier and cheaper to add more nodes to the cluster, to handle the concurrent requests, than it is to make a single node powerful enough to handle them all by itself, which is what vertical scaling is.

4.2 Components

In XRAY for Jira Cloud we have decided to use an Amazon’s Elastic Beanstalk with the Docker Platform and Elastic Container Registry (ECR). Elastic Beanstalk is a PaaS where we provide the artifact, in this case a Docker image, and it managed deployments and scaling automatically [29]. ECR is a fully-managed Docker container registry that makes it easy for developers to store, manage, and deploy Docker container images. The reason we have decided to use Elastic Beanstalk is because it reduces our maintenance by providing upgrades, auto expanding and shrinking. Elastic Beanstalk also provides
deployment automation, health monitoring, log and metric collection and auto scaling. The reason we have decided to package our application using Docker is that it allows to run our application in a containerized environment and allows to customize the Node.js environment and package it with our dependencies, providing a greater control over our application without tying us to the constraints of a traditional Elastic Beanstalk environment. This architecture provides a cluster of nodes spread across multiple availability zones with a managed platform capable of, almost independently, running standard Docker images. The final XRAY for Jira Cloud architecture is composed by the following elements:

1. Elastic Beanstalk
2. Elastic Container Registry (ECR)
3. Docker Image

![Figure 4.2: Releases](image)

Elastic Beanstalk, as mentioned before, is responsible for updating, auto expanding and shrinking the cluster of nodes that represent the application. This component has smaller sub-components inside, the Elastic Load Balancer (ELB), which is responsible to balance the incoming request across the different nodes. Those nodes are being ran in Elastic Compute Cloud (EC2) which basically represents a virtual machine (VM). The Elastic Beanstalk contains two more components: the Simple Storage Service (S3), which is used to store the logs coming from the multiple nodes that are or were running; and a Cloudwatch, which is used for collecting metrics and for monitoring the health of the EC2s that are running the application.

The Elastic Container Registry is responsible for storing and managing the Docker versions, and is also responsible for deciding which Docker image should be used by the Elastic Beanstalk.
Each Docker Image lives inside the EC, contains the application and all the required dependencies for it to run. The way these Docker images are created are by using Bamboo, another Atlassian tool, and by implementing tasks that will provide capabilities to achieve continuous integration and deployment. The process for creating a Docker image is as follows: after Bamboo finishes building and testing the application it creates an artifact; any artifacts that has successfully passed the tests can be used to create a Release (illustrated in Figure 4.2. When creating a Release and deploying it to an environment, Bamboo with create the Docker image with the necessary artifacts, tag the Docker and push the Docker image to the ECR. To update the cluster with the new version of the application, on the deploy phase we create a new application version on the Elastic Beanstalk which will then trigger it to update all the nodes in the cluster to update to the new version.

Another amazing feature of Release is that not only it provides a snapshot of a specific version artifacts but it also stores information about which Jira issues are related with that Release, commit record and test and build metadata. This enables a much smoother reporting and tracking as the Release moves through our environments, and allows us to easily track changes between releases and fix bugs that possible arise.
Contents

5.1 Evaluation Questions ......................................................... 43
5.2 Tests ................................................................. 44
This chapter presents the primary business problems/concerns that were found during the development of XRAY for Jira Cloud as well as how they were resolved. For each of those concerns we will discuss how they were solved as well as which tests were performed to guarantee the issues are indeed solved as well as a brief explanation of why those particular tests were performed instead of some others.

5.1 Evaluation Questions

XRAY for Jira Cloud was designed to create a version of XRAY for Jira that was compatible with the Atlassian Jira Cloud application, assuring the same available features of its counterpart version 2.0. During the development of XRAY for Jira Cloud we decided to not implement all features (as of its server counterpart) since there was some room for improvement. Given the major differences regarding architecture and development resources between these two versions (Server and Cloud) it was predictable that some problems/questions would arise that would need to be analyzed and thought.

The main problems were related with the differences between the Jira Cloud and Jira Server applications, mainly the fact that Jira Cloud apps do not run in the same server as the Jira applications. For this reason, something that was not done for the Server version of XRAY needed to be done for its cloud version and that was an environment that had to be build to run the app. That environment had to be performant, secure (since it will handle sensitive client data) and scalable (needs to handle a huge amount of concurrent requests). All app data will now have to be stored in its own database and managed by XRAY for Jira Cloud, bringing new concerns such as data backup, data replication and permissions which now need to be considered. Since add-ons and Jira applications are now running on separated machines, all the data must be accessed via REST APIs instead of Java API as in the server version of the app. This brings security and performance constrains because data must be transferred over the network from one server to another, which, besides taking more time could also be subjected of forging attempts, that includes the extended UI modules that needs to be rendered fast to keep the user attention. With these concerns in mind, some questions emerged:

• In average, how long will a request take to be resolved, how long will the users need to wait for the XRAY modules to be displayed in the page? Is that time accepted and, if not, can it be reduced?;

• How can it be ensured that a feature has been correctly implemented? How can we guarantee and verify that it replaced\[id=RC\]does not doesn’t break old functionality?

• Can features be independently tested?

• What is the application reaction to continuous requests over an extended period of time?
• What happens in case of peak workload conditions caused by simultaneous importations? How much is the performance affected?

5.2 Tests

JavaScript testing is normally comprised of three types of tests [30]:

• Unit Tests – Testing of individual functions or classes by supplying input and making sure the output is as expected. These tests will be done using Jest Framework and Supertest Framework, for testing the REST API endpoints parameter validation.

• Integration Tests – Testing processes or components to behave as expected, including the side effects. These tests will also be done using Jest Framework.

• UI Tests – Testing scenarios on the product itself, by controlling the browser or the website, regardless of the internal structure to ensure expected behavior, also known as Functional Tests. These kinds of Tests are very brittle and typically require a high amount of maintenance to be relevant and up to date. Due to this, it was decided that they would be implemented after the Beta phase started since only then the interface of XRAY for Jira Cloud would be finalized and only then it would make sense to make this kind of tests.

The reason Jest was chosen as the Testing Framework is because, in terms of specification, it is very similar to others, such as Jasmine and Mocha, but what Jest has that other Frameworks lack is the ability to compare the result of a test with a snapshot. This is particularly useful when testing React components which XRAY for Jira Cloud has many, allowing the maintenance of tests to be faster and easier for the developers and simpler and more intuitive for the testers [31, 32].

Making the maintenance of Tests a lot simpler and faster was the main selling point of Jest Framework since this is typically a very big problem with tests since new features sometimes break the existing tests, which sometimes can be a good thing since it identifies bugs in the new features but sometimes the behavior was intentionally changed and this means that the tests also have to be changed. With traditional frameworks the developer would have to look at the test and change the parameters of the tests, which can still happen using Jest. But where Jest shines is with the test expected result, since it can just compare a snapshot of the result, all the developer needs to do is review the result and decide if the snapshot should be updated without the need to change or write additional code. This can also cause problems if the snapshot is incorrectly changed, but since all code changes made by a developer are reviewed by other developers before going to production this replaced[id=RC]should not shouldn’t happen too frequently.
In order to test the loading time of the modules extended by XRAY for Jira Cloud a software analytics tool called NewRelic [33] was used.

![Slowest average response time](image)

<table>
<thead>
<tr>
<th>Request Path</th>
<th>Response Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>get /view/page/execute-test</code></td>
<td>1,170</td>
</tr>
<tr>
<td><code>get /view/webpanel/test/test-preconditions-view</code></td>
<td>199</td>
</tr>
<tr>
<td><code>get /view/webpanel/test/test-test-plans-view</code></td>
<td>192</td>
</tr>
<tr>
<td><code>get /view/webpanel/test/testruns-view</code></td>
<td>187</td>
</tr>
<tr>
<td><code>get /view/webpanel/test/test-sets-view</code></td>
<td>184</td>
</tr>
<tr>
<td><code>get /view/webpanel/test/test-run-status-view</code></td>
<td>162</td>
</tr>
<tr>
<td><code>get /view/webpanel/test/test-details-view</code></td>
<td>158</td>
</tr>
<tr>
<td><code>get /api/internal/testRunStatus/testissueld</code></td>
<td>45</td>
</tr>
<tr>
<td><code>get /view/webpanel/testexec/tests-view</code></td>
<td>23</td>
</tr>
<tr>
<td><code>get /view/webpanel/testplan/testexecs-view</code></td>
<td>22.4</td>
</tr>
<tr>
<td><code>post /api/internal/issuelinks/sourceType/issueld/targetType</code></td>
<td>22.2</td>
</tr>
<tr>
<td><code>post /api/internal/testruns</code></td>
<td>20.7</td>
</tr>
<tr>
<td><code>get /view/webpanel/testplan/tests-view</code></td>
<td>19.6</td>
</tr>
<tr>
<td><code>get /view/webpanel/testexec/test-environments-view</code></td>
<td>19.3</td>
</tr>
<tr>
<td><code>post /api/internal/testplan/testPlanissueld/tests</code></td>
<td>19.2</td>
</tr>
<tr>
<td><code>post /api/internal/settings/miscellaneousOptions/projects</code></td>
<td>19.1</td>
</tr>
<tr>
<td><code>get /api/internal/settings/global/testEnvironments</code></td>
<td>18.3</td>
</tr>
<tr>
<td><code>get /view/webpanel/testexec/testplans-view</code></td>
<td>18.1</td>
</tr>
<tr>
<td><code>get /view/dialog/issue-picker/issue-picker-view</code></td>
<td>15.6</td>
</tr>
<tr>
<td><code>get /api/internal/testExec/testExecId/testPlansMissingTests</code></td>
<td>14.4</td>
</tr>
</tbody>
</table>

**Figure 5.1:** XRAY for Jira Cloud - Request Response Time

This tool was used to calculate the time a request takes to be resolved, on average (as illustrated in Figure 5.1). It has also available a Dashboard where the results could be consulted and organized by page load time or by throughput (as illustrated in Figure 5.2). This allows us to have an easy overview about request response times in XRAY for Jira Cloud and quickly spot potentially problems. One problem that we found fairly quickly was the fact that the requests were taking a long time to be completed and they were spending a big part of the time reading files from disk; this was due to the a misconfiguration in the Handlebars view engine, which was therefore corrected.
Figure 5.2: XRAY for Jira Cloud - Request Throughput

In Figure 5.1 are presented the request response time for each request that is made from the Jira extended modules by XRAY for Jira Cloud. From that figure, it is possible to be verified that the worst case is the XRAY for Jira Cloud execute-test module. This is due to the fact that this module requires data that is both in the database and in the Jira Cloud Server, causing it to have to make REST API calls to Jira Cloud Server and to fetch data from our database, and since the data is related with each other this process cannot be done in parallel, slowing even more the process. This process could possibly still be optimized, however. With a worst case of 1170 ms to resolve a request for the slowest module and the other modules taking less than 200 ms to resolve it can be concluded that XRAY for Jira Cloud achieves with success the task to resolve the requests as fast as possible. In Figure 5.2 it is also present the Throughput by route.
To ensure that new features are implemented correctly, do not break or negatively affect existing features, extra stages were added to the XRAY for Jira Cloud project workflow (Figure 5.3). The first group of stages, Writing QA Tests, Waiting for Tests and Testing, were added so our QA team can verify that the feature was correctly implemented and that it is not affecting other existing features. The other stage added is for the developers to quickly verify that the feature is correctly implemented and working in our staging environment. This extra step was added to catch potential errors that are not caught in our personal development instance of Jira Cloud but exist in the Staging environment. Additionally to the stages added in the workflow integration tests for the React UI Components were implemented to ensure that changes to the existing components do not break existing features. More of these types of tests are planned to be implemented after the Beta phase of XRAY for Jira Cloud have started because similar to the UI Tests, these tests should be implemented after the UI is finished to prevent, although
simple to do, to have to always update the JEST snapshots (Figure 5.4).

In order to respond to the question *Can features be independently tested?* unit tests were done to key server components and to ensure that the REST API is receiving and validating the parameters correctly, both ensuring that the REST endpoints are correct and also if someone tries to exploit or break the REST API that this will not happen, as this could be a serious problem because it would cause internal errors in the Node Servers and overload the Servers unnecessarily.

Figure 5.5: XRAY for Jira Cloud - Jest Unit Tests

These tests were done using Jest framework along with a module called Supertest (Figure 5.5). The reason Supertest is used is because it allows the simulation of the REST request but instead of actually making a REST call, it calls the function responsible for making the REST call along with all its respective middlewares. Jest was used again for its snapshot capability allowing us to take and compare the REST API responses thus facilitating the maintenance of the tests, since when a new parameter needs to be
added to a request we only need to review the snapshot and update it if its correct. The reason not
more unit tests were done to the server components is due to the fact that, in most of the cases, it is
necessary to make requests to the Jira Cloud application to fetch more information which in the unit
tests case would have to be mocked since unit tests should not make REST API calls, thus sometimes
making the test irrelevant, therefore only key server components that are not dependent on external
information were tested with Jest.

As for the questions **Whats is the application reaction to continuous requests over an extended
period of time?** and **What happens in case of peak workload conditions caused by simultaneous
importations? How much is performance affected?**, the answers to those questions will be obtained
during the Beta phase of XRAY for Jira Cloud and that is the main objective of the Beta to see how the
system behaves under intense workloads and to catch additional errors that were missed by the QA
team. In other to catch and easily view the errors that will arise during the Beta phase, two mechanisms
were put in place: AWS CloudWatch [34] and Sentry [35].

With AWS CloudWatch it is possible to track metrics, collect and monitor log files, set alarms, and
automatically react to changes in your AWS resources. Currently XRAY for Jira Cloud is using AWS
CloudWatch to collect and monitor log files, this way the files are all in one single place where they can
be easily filtered. Later, it is planned to use additional features of AWS CloudWatch, specially the set
alarms feature which will allow us to scale the application to better suit the workload present. Sentry
allows the collection of logs, but in this case, the logs are present in the client browser, and this gives
us information that typically is only available if requested to the clients. On top of that, if a client has a problem, since we get the logs right away, it is possible that by the time the client opens a support ticket or notices a problem, the development team is already working on solving it. In Figure 5.6 it is possible to see examples of how errors are displayed in Sentry.
The objective of this Thesis was to create XRAY for Jira Cloud, an app that could be used in Jira Cloud applications. The objective was achieved with success, an Atlassian Connect add-on (XRAY for Jira Cloud) was developed which has available all the features that the version 2.0 of XRAY for Jira Server currently offers with some extra improvements.

6.1 Conclusion

The entire development was done using Agile methodologies and best practices in a team environment, also, the project management was done using Jira itself as an Issue Tracker. One of the important aspects that was taken into account during development was to keep the XRAY for Jira Cloud interface as similar as possible with the Jira application so the user experience is as smooth as possible. Another important aspect was to have the same logic in both XRAY for Jira Cloud and its server versions so if clients were already used to one of them the transition would again be as smooth as possible.

In addition to the development itself, an entire infrastructure had to be designed to support the app. The requirements for this infrastructure were that it had to support multiple clients and respond effectively to a large amount of requests, allowed scalability, high availability and fault tolerance.

Despite the lack of knowledge and experience in the used technologies, NodeJS, MongoDB, AWS and the fact that Atlassian changed the Jira Cloud application interface twice during development, all objectives were fulfilled, the app can manage all the required configuration data from each client, along with all their tests information, calculate the status for a requirement, generate multiple reports based on the stored information and import test execution results using public REST API. The project is currently starting its Beta phase with real customers using the product.

6.2 Future Work

In the future, the tests will be improved, UI Tests will be implemented, the bugs resulting from the Beta phase of XRAY for Jira Cloud will be fixed. After the Beta phase ends the app will be released and the development cycle will start again where new features will be implemented and bugs will be fixed, with the objective of having all the features that the current XRAY for Jira Server has and possible with new features that might be relevant for the Jira Cloud application.
Bibliography


