Xporter for JIRA Cloud

Fábio Cristiano Martins Antunes
fabio.antunes@tecnico.ulisboa.pt

Instituto Superior Técnico, Lisboa, Portugal

October 2016

Abstract

Xporter for JIRA is a plugin that extends the functionality of JIRA, an Atlassian's issue and project tracking tool, providing to users an easy way to extract and format data from JIRA using customized templates and producing, as a result, personalized reports. JIRA is currently available in two versions, JIRA Server and JIRA Cloud, Xporter for JIRA was only compatible with JIRA Server. This thesis addressed the challenge of creating a JIRA Cloud compatible version of Xporter for JIRA, an Atlassian Connect add-on that is already available to the public, implemented with the help of a development framework called Atlassian Connect Express, used to create Atlassian connect add-ons in NodeJS. Xporter for JIRA Cloud was implemented mostly in JavaScript using only an external service developed in Java, responsible for the documents generation and that is shared between versions Server and Cloud of the add-on. After finishing the add-on development, all the infrastructure to support it was also created, using for this, the Amazon Web Services and MongoDB Atlas service. This infrastructure has been created in order to ensure that the add-on would be able to operate correctly and efficiently even when subjected to great amounts of work by scaling itself and also to guarantee fault tolerance and high availability, being deployed in different and independent regions.
Keywords: Xporter for JIRA Cloud, Issue Tracking Tool, JIRA, Cloud, Connect Framework, Web Application.

1. Introduction

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 is composed of five essential characteristics (On-demand self-service, Broad network access, Resource pooling, Rapid elasticity, Measured service), three service models (Software as a Service, Platform as a Service, Infrastructure as a Service), and four deployment models (Private Cloud, Community Cloud, Public Cloud, Hybrid Cloud)[1, 2, 3].

Every day that goes by, cloud computing is winning more companies that invest heavily in it, one of that companies is **Atlassian**, an enterprise software group of companies that develops products geared towards software developers and project managers. It is best known for its issue tracking application, **JIRA**(*"workflow management system that lets you track your work in any scenario."*)[4].

Traditionally, issue tracking systems have been largely viewed as simple data stores where software defects are reported and tracked within an archival database. Currently the most advanced way of dealing with bugs is to enter them into an Issue tracking system to address the critical and important task of helping organizations manage issue reporting, assignment, tracking, resolution, and archiving[5]. Some basic functionality that an Issue Tracker must ensure in Software Projects are for example: Share the information across the team, have an instant overview of the state of the software, have a recorded history of changes and know when the request was reported, fixed and verified [6].

To understand JIRA we must know what is the mean of *Issue*. Issue is the fine-grained concept inside JIRA, it could 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 concepts: Project, Component and Workflow as shown in Fig.1 and 2.

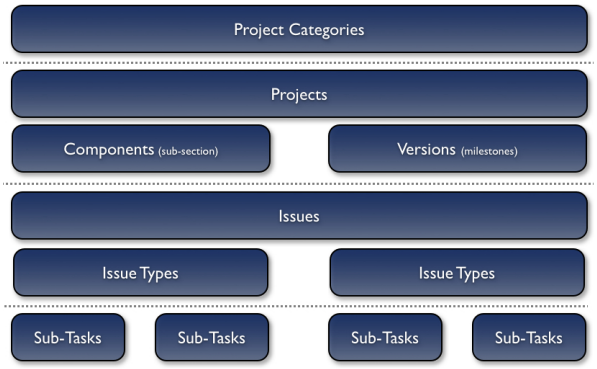

Figure 1: JIRA Components.

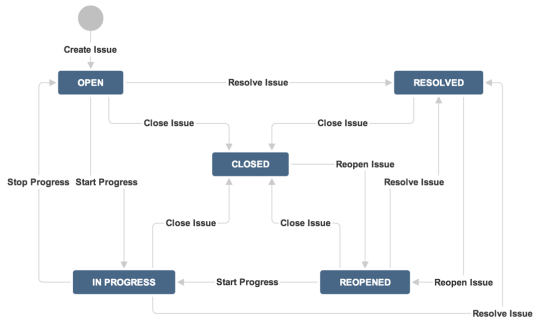

Figure 2: JIRA Workflow.

JIRA allows extending its functionality by installing add-ons to extend the platform. The JIRA installations come with a set of pre-installed plugins (from Atlassian) but later on can be installed other plugins from external companies through a marketplace provided by Atlassian [7]. Xporter for JIRA is one of that plugins that extends JIRA platform. It was created in 19th December 2011 by XpandIT, a Portuguese global company specialized in strategic planning, consulting, implementation and maintenance of enterprise software, fully adapted to the customers needs.

Xporter for JIRA was created to provide for JIRA users a way to generate custom reports in several formats, filled with data from JIRA issues. To create this custom reports, users must create templates (Word or Excel files) and write special placeholders that will be replaced by issues data as shown in Fig.3. The Fig.4 shows the possible output formats for each template type. With Xporter for JIRA it is also possible to export several issues at the same time using JIRA Bulk operation, or using loops defined in Xporter for JIRA templates.

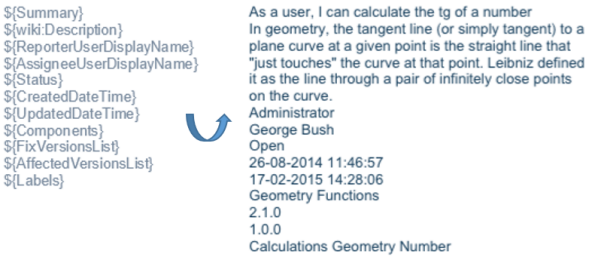

Figure 3: Xporter Template and Report

		Template Extensions						
		.DOCM	.DOCX	.DOTM	.DOTX	.RTF	.XLSX	.XLSM
Output Formats	.PDF	.pdf	.pdf	.pdf	.pdf	.pdf	.pdf	.pdf
	.DOCX	**.docm**	.docx	**.dotm**	**.dotx**	.docx		
	.SVG	.svg	.svg	.svg	.svg	.svg		
	.PNG	.png	.png	.png	.png	.png		
	.RTF	.rtf	.rtf	.rtf	.rtf	.rtf		
	.ODT	.odt	.odt	.odt	.odt	.odt		
	.XLSX						.xlsx	**.xlsm**

Figure 4: Xporter Templates Input/Output Matrix.

The plugins for JIRA are divided into three types: Plugins Type-1, Plugins Type-2 and Atlassian Connect plugins. Only the Atlassian Connect Plugins can be used/installed in JIRA Cloud instances [8]. Atlassian has been betting and investing in cloud products what have led to a migration of Server customers to JIRA Cloud, and as such, there is a need to create a new version of Xporter for JIRA (PluginType-2) as an Atlassian Connect plugin. That way it can be installed on JIRA Cloud instances and thereby be sold to new JIRA cloud customers or to customers who have migrated from server version.

2. Background

Xporter for JIRA development has started in 2011 and was released to public through Atlassian marketplace in 19th December 2011. Currently, Xporter for JIRA has a total of 1112 active customer installations being United States of America, Germany and United Kingdom the countries with the biggest Xporter for JIRA clients [9]. Atlassian has been increasingly focused on their cloud versions. This ambition is related, especially in the ease and speed with which a customer can setup a JIRA instance, the facilities of integration with other Atlassian cloud products and the fact that customers do not have to worry about buying and maintaining their own infrastructures. As already mentioned, only Atlassian Connect Plugins can be used/installed in JIRA Cloud, which leads to the need of developing an Atlassian Connect plugin version of Xporter for JIRA Cloud. It exists some limitations related to Atlassian Connect add-ons what can cause that some of the capabilities in server version may not be implemented in cloud version

as shown in Fig.5 and Fig.6.

UI Elements	JIRA Server	JIRA Cloud
Administration Web-Section	✓	✓
Administration Web-Items	✓	✓
Xporter Search Request View	✓	✓
Issue Detail Web-Panel	✓	✓
Xporter web-Item in Single-Issue-View	✓	✗ (ACJIRA-651)
Workflow Post Functions Views	✓	✓
Xporter for JIRA Bulk Operation	✓	✗

Figure 5: Xporter interface elements compatibility between JIRA Server and JIRA Cloud.

Functional Requirement	JIRA Server	JIRA Cloud
Xporter Global Settings	✓	✓
Manage Templates	✓	✓
Template Store	✓	✓
Permission Schemes	✓	✓
Post Function Authentication	✓	💡
License Management	✓	💡
Single Issue Export	✓	✓
Bulk Export using Search Request View in the Export menu	✓	✓
Bulk Export using Xporter for JIRA Bulk Operation	✓	✗
Create Document Post Function	✓	✓
Email Report Post Function	✓	✓

Figure 6: Xporter functional requirements compatibility between JIRA Server and JIRA Cloud.

2.1. Current Competition

Currently, in JIRA Cloud, the Xporter for JIRA Cloud competition comes down to two other plugins that aim to export data from JIRA:

Exporter Issues to CSV - This Atlassian Connect add-on focus is to allow the exportation of issue comments and issue transitions history to xlsx files. It does not allow custom reports, definition of permission schemes or data manipulation as Xporter for JIRA provides. It has 673 active client installations, so it has less than half of the Xporter for JIRA Server client installations [10].

Excel and All-In-One JIRA Reports - This add-on allows users to create insightful reports for daily use, executive reporting and other business purpose. This is achieved using drag-and-drop, within a user interface in JIRA, that allows to choose the fields to export and the way they are exported by choosing between a set of predefined template styles. Its limitation is related to the predefined template styles, that prevent the clients to have total control over their own templates. However, the interface provided is a very positive point because it allows great facility in the construction of templates that are not too complex. This plugin has 175 active client installations, so it stills in a initial growing phase [11].

In general we can see that none of the Xporter competitors allows customers to have a freedom and a power of customization of templates as high as the Xporter allows. Moreover, none of them allows data manipulation in the templates, definition of Post Functions to integrate exports with JIRA Workflows or the variety of exportation formats. Although, there are positive points in these other plugins, the fact that they provide an interface to build the templates can be a strong argument when choosing the plugin that clients will buy to export data from JIRA.

2.2. Atlassian JIRA

JIRA is available in two distinct options: **JIRA Server** and **JIRA Cloud** [12]. JIRA Server was the first being released and currently stills the one with more clients. The reason why most of the customers continues to use JIRA Server is related to the following points: Source code control, flexibility and customization options, more add-ons available from marketplace, upgrades control, no admin restrictions, full access to databases and privacy. An overview over JIRA Server architecture is shown in Fig. 7.

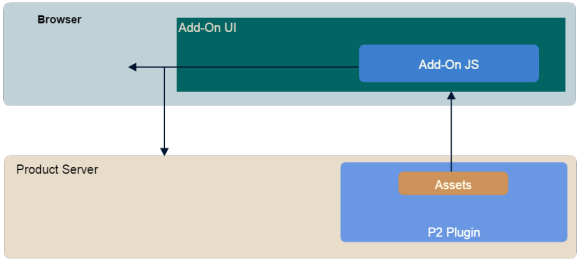

Figure 7: JIRA Server Architecture.

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. The most important points of JIRA Cloud are: High scalability and availability, own infrastructure not needed, no additional hardware requirements or associated costs, no additional work on the own IT department, no maintenance tasks needed and systems can be used quickly and short term [13, 14, 15]. An overview over JIRA Cloud architecture [16] is shown in Fig. 8.

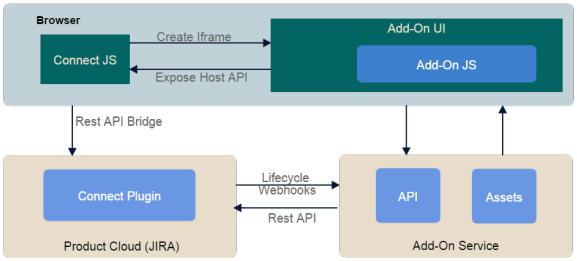

Figure 8: JIRA Cloud Architecture.

Xporter for JIRA Cloud must provide all the features available in Xporter for JIRA Server that are possible to implement in JIRA Cloud version. For that to be accomplished, additional logic and components will be needed, like the support infrastructure and the add-on database.

2.3. Technology

During Xporter for JIRA Cloud development were found some technological challenges. These challenges are related to several factors: the differences between the Plugins and Connect Frameworks, the lack of experience in the new development stack and also with the fact that the new architecture has brought concerns that didn't exist before. In JIRA Cloud add-ons vendors are responsible for setting up the platform where the add-on will be running, must manage all the add-on data and settings and the java API from *Plugins Framework* isn't available anymore.

To help in Atlassian Connect Add-ons [17] development, Atlassian provided a framework that could be used to help develop new add-ons and is available in several languages. This framework has several functionalities: Serve add-on descriptor and UI, handle add-on installation, persistent store, handle add-on requests, JWT token handler, crypto library, JSON and HttpClient libs. Our development team used the **NodeJS** version of the framework [18].

As stated earlier, the add-on data management must now be made out of JIRA. This requires the use of an external database which will be used and managed by the add-on. The chosen database to be used by Xporter for JIRA was MongoDB, an open-source non-relational database developed by MongoDB Inc [19]. The choice of MongoDB as the database was made taking into account factors such as: fast development, ease of scale horizontally (providing new levels of availability and scalability), natural mapping to object-oriented programming languages (like Javascript, that will be used in Xporter for JIRA Cloud development) and the massive community behind it [20].

3. Implementation

At the beginning we have split Xporter for JIRA Server in modules to understand what could be used in Cloud version or what needed to be adapted before we can use in our connect add-on.

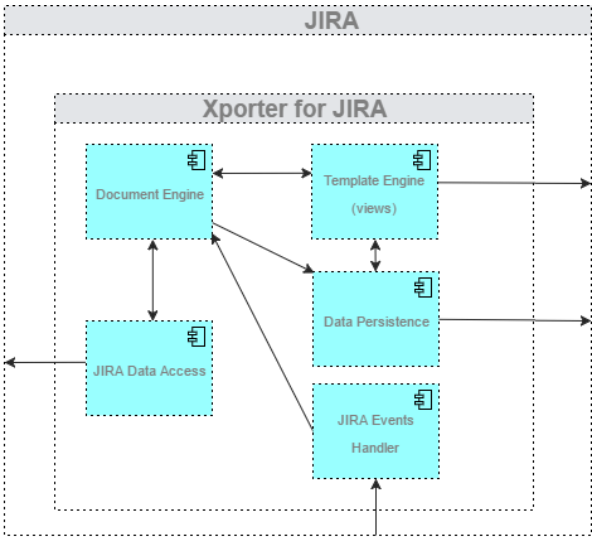

Figure 9: Xporter for JIRA components.

Given these modules, we quickly conclude that the document engine module could be used in the cloud version as an external service. Therefore, major changes were needed in the documents generation module because it was very dependent on other modules, especially in the module responsible for getting data from JIRA (using the java API from Plugins Framework). Considering these needs, a huge refactoring was made so that we could have a fully independent module responsible for generating documents that could be used in both server and cloud versions thus facilitating maintenance and coherence of the add-on on both platforms. The remaining modules would have to be developed from scratch due to considerable differences that the new architecture and the Connect framework would bring. For example, now all the add-on data needs to be stored and managed outside of JIRA, all add-on screens must be previously rendered and then sent to iframes presented to the end user and all the data must be retrieve by the REST API because there's no more a java API for that.

As its verified, without the Java API, all data accesses are made using calls through REST APIs. This means that a mechanism for authentication has to be used. Atlassian Connect uses a technology called JWT (JSON Web Token) to authenticate add-ons and also to ensure requests integrity [21, 22]. In order to an add-on be installed

it needs to declare itself with a descriptor. That descriptor is a JSON file that tells the Atlassian application about the add-on: where it is hosted, which modules it intends to use, and which scopes it will need. The scope is a concept that only exists in atlassian cloud instances and the main goal of it is to improve security by forcing the add-ons declaring what type of access they need from the Atlassian application what will define which REST API resources the add-on can use, giving the possibility to atlassian cloud application administrators to accept or decline those accesses. Local development could be done in two ways: using the Atlassian SDK to create a local cloud instance, running the add-on, tunneling it and install it (which works fine but both add-on and cloud instances were running on the same machine)[23] or by setting up a free 5 users tier cloud instance, enable the development mode in that instance running the add-on, tunneling it and install it[24]. Once we are using the Atlassian Connect Express framework the tunneling and installing steps are accomplish by simply specifying the URL, username and password from the JIRA application created in a configuration file of our add-on, and it will try to install itself in that instances.

3.1. Technical Implementation

So, firstly, NodeJS and NPM (Node Package Manager) must be installed, after that, its installed atlas-connect (ACE client tool) and created the project skeleton using it. In Fig. is shown the project structure created by atlas-connect.

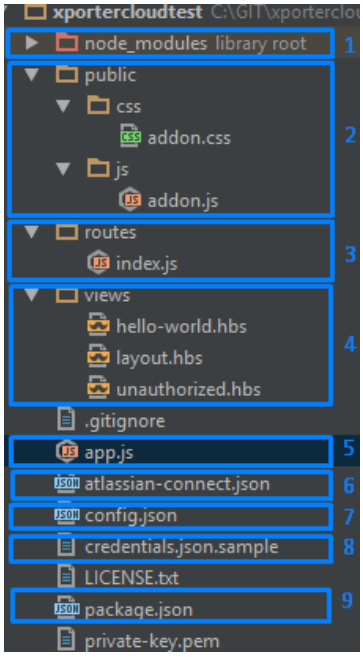

Figure 10: Connect Add-ons Structure.

1. The ***node_modules*** folder contains all the server-side dependencies of our project, they're manage by NPM.

2. The ***public*** folder is where are placed the project client-side dependencies. To manage this dependencies we will use Bower, a dependency manager tool for front-end components.

3. The ***routes*** folder is where all the files which define behavior related to the addon routes should be placed.

4. The ***Views*** folder is where all the template engine files should be placed. For Xporter for JIRA Cloud we decide to keep the default template engine, handlebars.

5. The ***app.js*** file is responsible to require some dependencies, create the express application, initialize and configure the components and to start the server were the application will run.

6. The ***atlassian-connect.json*** file is the add-on descriptor, previously mentioned.

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

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

9. The ***package.json*** file holds several relevant metadata to the project as well as the list of server-side dependencies.

After understanding all that concepts, Xporter for JIRA Cloud development has started. In *atlassian-connect.json* were defined all the JIRA UI modules that Xporter needed and defined the routes that will be responsible to handle each one. This routes will receive the 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 or query the mongodb database to obtain add-on related data. After that, they pass the required context data 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. Besides that kind of requests (from JIRA instance to the add-on) they had to be established new routes to handle requests from the iframe back to the add-on. The behaviour 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). The requests made back to the add-on are needed when a view needs additional data or for example in case of Xporter for JIRA Cloud, to send requests that will trigger an exportation or an upload/download of a template.

As stated before, the Document Engine module from Xporter for JIRA Server was re-factored to be used as a service of Xporter for JIRA Cloud. It is running in a tomcat server and the logic is written in java. Besides that, because of time restrictions and business/logistical reasons from Xpand IT, in the first versions, instead creating our own MongoDB cluster we're using MongoDB Atlas (mongodb as a service). The final Xporter for JIRA Cloud data flow is summarized in Fig. 11.

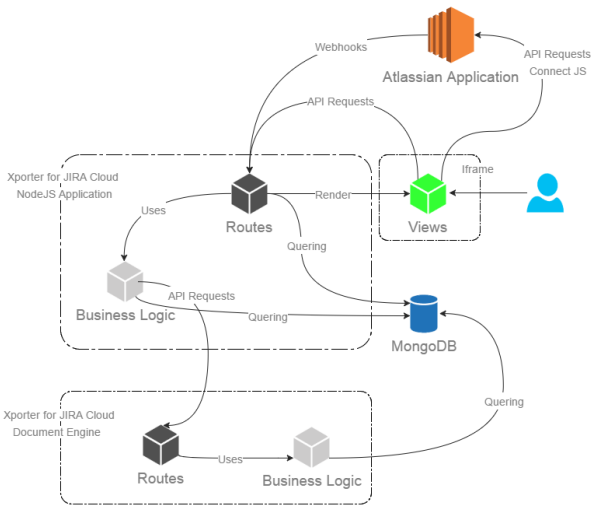

Figure 11: Xporter for JIRA Cloud data flow.

After declaring all the intended modules to be extended in the add-on descriptor, defined all the routes, implemented all the business logic and solved other required features like: logging, interface inter nationalization (using 118n module from NPM) and metric reports, the first version of Xporter for JIRA Cloud was completed.

4. Infrastructure Setup & Deployment

Due to the wide range of services (which may be useful in the future), platform maturity and the interest of our company to have a project running using their services, the choice of the platform that will hold the infrastructure was, AWS. All the components/resources will be attached to a dedicated virtual network in AWS [25]. A virtual private cloud (VPC) is a virtual network dedicate within AWS, it is logically isolated from other virtual networks in the AWS cloud and it could also be viewed like a networking layer for Amazon EC2. The final Xporter for JIRA Cloud Virtual Private Cloud is composed by several elements that together guarantee high availability, resources scaling, access restrictions and great performance.

4.1. Components

VPC Internet Gateway (1): An Internet gateway is a horizontally scaled, redundant, and highly available VPC component that allows communication between instances in the VPC with the Internet. It serves two purposes: Providing a target in the VPC route tables for Internet-routable traffic, performing network address translation (NAT) for instances that have been assigned public IP addresses[26].
NAT Gateway (1): Used to enable instances in a private subnet to connect to the Internet or other AWS services, but prevent the Internet from initiating a connection with those instances[27].
Elastic Load Balancer (2): Responsible to automatically distribute incoming application traffic, based on application or network level information, across multiple Amazon EC2 instances. It allows to achieve fault tolerance in applications, seamlessly providing the required amount of load balancing capacity needed to route application traffic. The Load Balancer also monitors the health of its registered instances and ensures that it routes traffic only to healthy instances. When the Load Balancer detects an unhealthy instance, it stops routing traffic to that instance, and then resumes routing traffic to that instance when it detects that the instance is healthy again. As such, this helps to ensure high performance and fault-tolerance.[28]
Elastic Cloud Compute (>3): A Web-based service that allows business subscribers to run application programs with resizable compute capacity in the cloud. The EC2 can serve as a practically unlimited set of virtual machines[29].
Availability Zone (2): Amazon cloud computing resources are housed in highly available data center facilities. Data center locations are called regions. Each region contains multiple distinct locations called Availability Zones. Availability Zone are engineered to be isolated from failures in other

Availability Zones, and to provide inexpensive, low-latency network connectivity to other zones in the same region. By launching instances in separate Availability Zones, applications are protected from failure of a single location, as it was done in Xporter for JIRA Cloud infrastructure [30].

Subnet (6): A subnet is a range of IP addresses in VPC. It is possible to launch AWS resources into a specific subnet. Usually a public subnet is used for resources that must be connected to the Internet, and a private subnet for resources that won't be connected to the Internet[31].

Auto-Scaling Group (2): Auto-Scaling Groups helps maintaining application availability and allows scaling EC2 capacity up or down automatically according to conditions you define. They help ensuring that are running the desired number of Amazon EC2 instances as well as automatically increase the number of Amazon EC2 instances during demand spikes to maintain performance and decrease capacity during lulls to reduce costs[32].

The data flow in the infrastructure architecture could be detailed in the following steps:

Handle Incoming Requests: In Xporter for JIRA VPC the access to Internet from instances is made using the Internet Gateway. The incoming traffic is handle by an *Internet-Facing Classic* Elastic Load Balancer (the public one)[33]. The DNS name of an Internet-facing load balancer is publicly resolvable to the public IP addresses of the nodes, therefore, Internet-facing load balancers can route requests from clients over the Internet to the most suitable EC2 instance based on CPU, memory and network informations. The public ELB needs to have *Cross-Zone Load Balancing* property enabled in order to distribute traffic to the *node-private-subnet* EC2 instances in any Availability zone. For that, it also needs to belong to each public subnet of each availability zone to properly route out the requests to the Internet Gateway. This public ELB has associated a trusted certificate (to decrypt the requests from Atlassian instances before route the requests to the instances), a security group that defines that will be only accepted TCP incoming traffic in port 443 (SSL) and a Health Check policy to ensure that all the registered instances are available.

Requests Processing: After a request is routed to one of the *node-private-subnet* EC2 instances (Xporter for JIRA Cloud NodeJS node) it is processed by Xporter for JIRA Cloud NodeJS Application. Those instances have a Security Group that restricts the inbound requests to only accept TCP requests to port 3000 and SSH requests on port 22 (to access the instances) and are associated to an Auto Scaling Group that is responsible to start or delete instances based on CPU and memory metrics. In the processing stage, additional requests could be made to the MongoDB database, the Atlassian cloud instance that issued the request or to one of the *webengine-private-subnet* EC2 instances (Xporter for JIRA Cloud Document Engine). For the first two cases, the request response is sent back using the NAT gateway. When a *node-private-subnet* EC2 instance makes a request to the Document Engine the request target is an Internal load balancer[34]. The DNS name of an internal load balancer is publicly resolvable to the private IP addresses of the nodes, therefore, internal load balancers can only route requests from clients with access to the VPC for the load balancer. This internal load balancer will then route the request to the most suitable *webengine-private-subnet* EC2 instance based on CPU, memory and network informations. While processing the request, additional requests could also be made to the MongoDB database, the Atlassian cloud instance that issued the request or to the *node-private-subnet* EC2 instance (as explained before the two first request must use the NAT gateway). *Webengine-private-subnet* EC2 instances have a Security Group that restricts the inbound requests to only accept TCP requests to port 8080 and SSH requests on port 22 (to access the instances) and as *node-private-subnet* EC2 instance, are also associated to an Auto Scaling Group that is responsible to start or delete instances based on CPU and memory metrics.

5. Results

Since cloud add-ons are not running in the same server as JIRA applications an environment had to be built to run the add-on, that needs to be scalable (to handle big amount of concurrent requests), secure (will handle sensitive data) and ensure performance. All add-on data has now to be saved in its own database, therefore, all the concerns related to it, like data backup, data replication and accesses permissions have now to be considered. Since add-ons and JIRA applications are now running on separated machines, all the data must be accessed using REST APIs instead Java API like in server add-ons, this brings security and performance constrains because data must be transferred over network from one server to another, what 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. In view of these concerns, emerge questions such as:

1. *In average, how long users need to wait in order to get their modules loaded (page load timing)? Is the amount of time acceptable?*
2. *What happens in case of peak workload conditions caused by simultaneous exportations? is*

the performance affected? How much?

3. *How can new features be validated ? How could be ensured that they don't break old functionalities?*

4. *What is the application reaction to continuous requests over an extended period of time?*

5. *Can Xporter for JIRA features be independently testable?*

6. *How is the database behaviour under normal and peak workload conditions in terms of memory, network and Process CPU?*

5.1. Tests

In order to test the loading time of the modules extended by Xporter for JIRA Cloud was used a software analytics tool called *newrelic*[35]. This tool was used to calculate the time difference between the request being made and the page is fully charged. In Fig. 12 are presented the average page load times for each route that handles the JIRA extended modules by Xporter for JIRA Cloud. As it is possible to verify the worst cases are the web-panel and manage permissions pages. In web panel, it is probably due to the required conditions checks that needs to be performed and in manage permissions page the problem could be related to complex code while resolving the permissions for that client. With a worst case of *3.520ms* to completely render a module can be concluded that Xporter for JIRA Cloud achieved with success the task to present the UI as fast as possible to the end users.

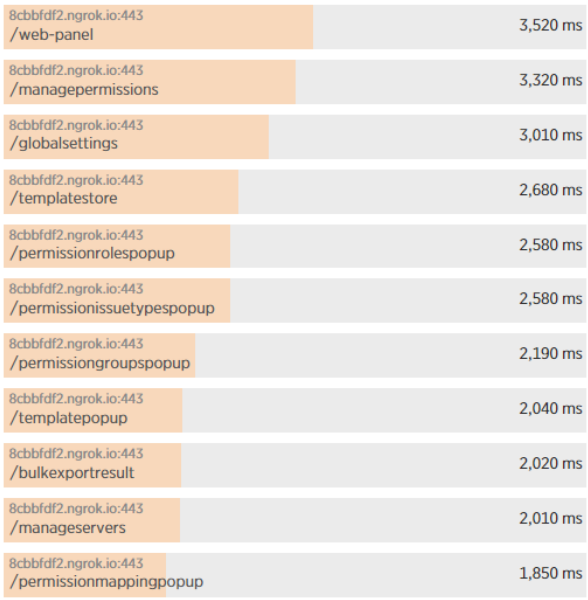

Figure 12: Xporter for JIRA Cloud - Page Loading Times.

To ensure that each bug fix or new features were properly implemented, are working as expected and did not affect/break any old functionality was used Testing stages in the Xporter for JIRA project Workflow. That way, in order to be resolved, each issue had to go through a stage in the Workflow where quality team members verified its functionality and ensured that no related feature has been affected. As Fig. 13 shows, after opening and start progress in a new issue, there are several possible paths to the closed state but all the paths that have passed by the resolved state had to necessarily passed also by the testing one, ensuring that way that no issues were resolved without passing by the QA team.

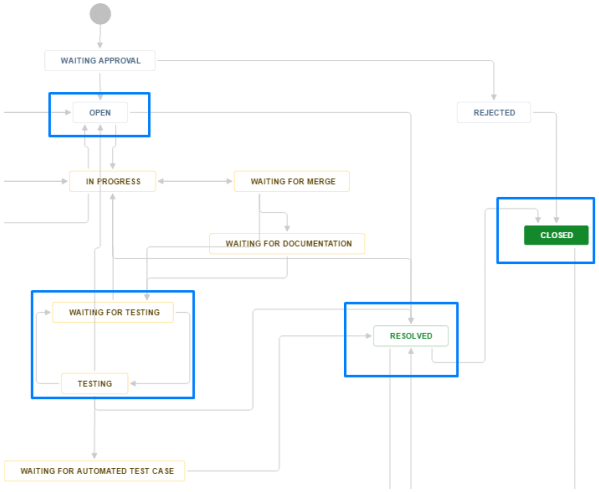

Figure 13: Xporter Cloud Project Workflow.

To properly test the results related to exportations performance was used *Jmeter*[36]. Within *Jmeter* were performed tests that tried to simulate the following scenarios: Normal/expected scenario of exportations, Peak workloads of simultaneous exportations and Big amount of exportations requests over large periods of time.

Scenario 1: To try simulate an expected/normal amount of work were made 900 exportations with 3 seconds of interval between each request. In this scenario, in terms of response time to the requests, there was an average of 3.8s with the worst case of time consuming of 18s to be completed. It was verified a slight increase in CPU utilization in EC2 instances and also in the instance of MongoDB, where there was a slight increase in the number of active connections.

# Samples	Average	Min	Max	Error %	Throughput	KB/sec
900	3858	1728	18407	0,00%	14,4/min	56,8
900	3858	1728	18407	0,00%	14,4/min	56,8

Figure 14: Requests Results - Scenario 1

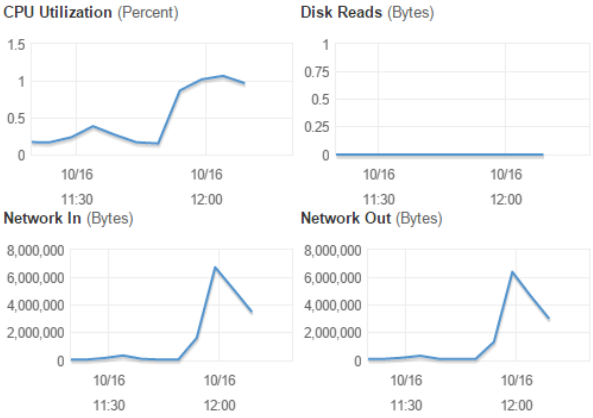

Figure 15: Node EC2 Metrics - Scenario 1

Scenario 2: In order to simulate peak workloads were made 900 exportations divided in 9 groups of concurrent requests. The response time to the requests has had an average of 14.4s with a worst case of 120s to be completed. This values are probably related to some connections timeouts that have happened. It was noticed a considerable increase in CPU usage in Webengine and MongoDB instances, where there also was a big increase in the number of active connections and in network usage.

# Samples	Average	Min	Max	Error %	Throughput	KB/sec
900	14454	1329	119743	0,00%	1,7/sec	396,3
900	14454	1329	119743	0,00%	1,7/sec	396,3

Figure 16: Requests Results - Scenario 2

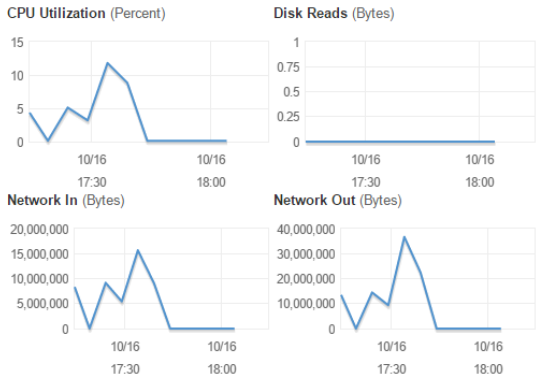

Figure 17: Webengine EC2 Metrics - Scenario 2

Scenario 3: In order to simulate continuous work were made 2271 exportations divided in groups of 42 concurrent requests. In terms of response time, there was an average of 20s with the worst case time consuming 43s to be completed. It was the scenario that caused the biggest increase in CPU and network usage in both EC2 instances and also in MongoDB instance, where there was a huge increase in the network activity and in the number of active connections.

# Samples	Average	Min	Max	Error %	Throughput	KB/sec
2271	20717	304	43333	7,49%	2,0/sec	1,1
2271	20717	304	43333	7,49%	2,0/sec	1,1

Figure 18: Requests Results - Scenario 3

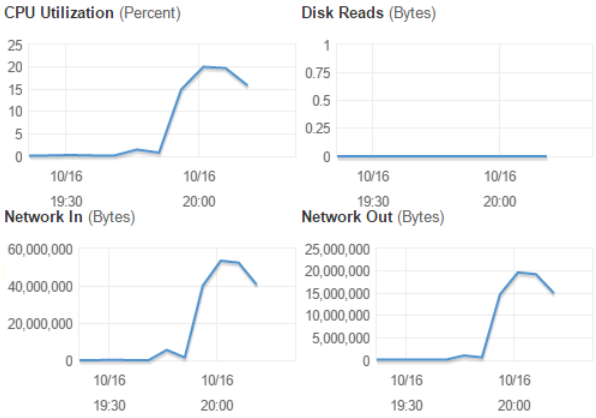

Figure 19: Node EC2 Metrics - Scenario 3

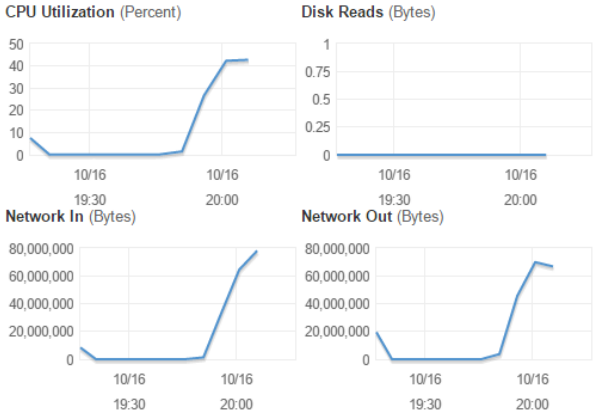

Figure 20: Webengine EC2 Metrics - Scenario 3

6. Conclusions

As originally proposed, the aim of this thesis was to create a version of Xporter for JIRA add-one, which has achieved with success, having been developed an Atlassian Connect add-on (Xporter for JIRA Cloud) that have available almost all the features that the version for JIRA Server currently offers. In addition to the development itself, it had also to be designed in AWS, the entire infrastructure that supports the add-on, ensuring that it is prepared to be used by multiple clients and respond effectively to large amounts of requests, allowing scalability, high availability, fault tolerance and of course the customer satisfaction. All the objectives have been fulfilled, the add-on can effectively manage all the required configuration data for each client, quickly displays front-end content to customers, and en-

sures high performance in the add-on most common task, the report generation. The project is already commercially available to clients through Atlassian Marketplace since 1st of August of 2016.

Acknowledgements
I want to thank my advisor Professor João Garcia, for all the support and attention provided throughout the Thesis.
Also, I want to thank my family and my best friends, all the strength they gave me during the course of the thesis.
Lastly, I would like to thank my co-workers for all the support and all the interesting ideas they shared with me and also to Xpand IT for the opportunity provided and confidence in my work.

References

[1] N. Leavitt. Is cloud computing really ready for prime time? *Technology News*, pages 1–6, 2009.

[2] P. Mell and T. Grance. The nist definition of cloud computing. *National Institute of Standards and Technology*, pages 1–7, 2011.

[3] D. Chappell. A short introduction to cloud platforms. *CHAPPELL & ASSOCIATES*, pages 1–13, 2008.

[4] J. Fisher, D. Koning, and A. P. Ludwigsen. Utilizing atlassian jira for large-scale software development management. *ICALEPCS San Francisco, United States*, pages 1–7, 2013.

[5] A.S.Syed Fiaz and S.Aarthi N.Devi. Bug tracking and reporting system. *International Journal of Soft Computing and Engineering*, pages 1–4, 2013.

[6] J. Janak. Issue tracking systems. *Masaryk University Faculty of informatics*, pages 1–106, 2009.

[7] The atlassian plugin ecosystem for jira. Accessed: 2016-09-28.

[8] What is atlassian connect? Accessed: 2016-09-28.

[9] Xporter - export issues from jira. Accessed: 2016-09-28.

[10] Exporter issues to csv and excel. Accessed: 2016-09-28.

[11] All-in-one jira reports. Accessed: 2016-09-28.

[12] Jira documentation. Accessed: 2016-09-29.

[13] Pros and cons of cloud vs. server. Accessed: 2016-09-28.

[14] Atlassian cloud or atlassian server? whats right for your organisation? Accessed: 2016-09-29.

[15] Atlassian cloud (formerly ondemand) - advantages and disadvantages. Accessed: 2016-09-29.

[16] Understanding atlassian in the cloud. Accessed: 2016-09-29.

[17] Installing in the cloud. Accessed: 2016-09-29.

[18] Express module for atlassian connect. Accessed: 2016-09-29.

[19] Mongodb. Accessed: 2016-09-29.

[20] Why is mongodb wildly popular? its a data structure thing. Accessed: 2016-09-29.

[21] Atlassian connect: Authentication. Accessed: 2016-09-29.

[22] Atlassian connect: Security. Accessed: 2016-09-29.

[23] Atlassian connect: Local development. Accessed: 2016-09-29.

[24] Atlassian connect: Development setup. Accessed: 2016-09-29.

[25] Amazon - vpc. Accessed: 2016-09-29.

[26] Amazon - internet gateway. Accessed: 2016-09-29.

[27] Amazon - nat gateway. Accessed: 2016-09-29.

[28] Amazon - elb. Accessed: 2016-09-29.

[29] Amazon - ec2. Accessed: 2016-09-29.

[30] Amazon - regions and availability zones. Accessed: 2016-09-29.

[31] Amazon - subnets. Accessed: 2016-09-29.

[32] Amazon - auto scaling groups. Accessed: 2016-09-29.

[33] Amazon - internet-facing load balancer. Accessed: 2016-09-29.

[34] Amazon - internal load balancer. Accessed: 2016-09-29.

[35] New relic. Accessed: 2016-09-29.

[36] Apache jmeter. Accessed: 2016-09-29.