XRAY for JIRA Cloud
André Miguel Pereira Rodrigues

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. 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.

Index Terms—XRAY, JIRA, Cloud, Test Management, Issue Tracking, Continuous Integration.

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 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].

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].

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.

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.

2 BACKGROUND

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.

2.1 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 thought 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.

2.2 Atlassian JIRA

JIRA is available in two distinct options: JIRA Server and JIRA Cloud Software as a Service (SaaS). JIRA Server was the first solution released and currently still is the one with more clients. The reason why most of the customers still continues to use JIRA Server is related 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.
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 [6]. 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 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 4.

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.

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 [8], crypto library, JavaScript Object Notation (JSON) and HttpClient libs. Our development decided to use the NodeJS version of the framework.

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 through multiple nodes [9]. 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.

3 IMPLEMENTATION

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 present 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.

As it was previously mentioned, with this new architecture and without the Java API, all data accesses are made using authenticated REST APIs calls. The mechanism used is a
technology called JWT that authenticates the apps [8], [10]. 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). In order for the app to be installed it needs a descriptor file. 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 addon 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 the we would rather develop on an environment much closer to the what the end environment will be. So we opted for using the free development instance that Atlassian provides the developers which has a very easy setup: textbf{Registering a development version of Jira Cloud - Atlassian provides a free 1 user tier cloud instance for developers}, Enable development Mode – In the apps page, enable the development Mode option, Install the app – This can be done manually by specifying the app descriptor URL to the Atlassian application or since we decided to use the Atlassian Connect Express this step can be simply accomplish by specifying the user credentials and the Atlassian application URL in a configuration file of our app which will then be used to inform out app where it should install itself.

3.1 Technical Implementation

To create the project skeleton we needed to install Node.js and Node Package Manager (NPM), after that we installed atlas-connect (ACE client tool), created the project skeleton using it and then install all the project dependencies. In Figure 5 is shown the final project structure along with the most important components which are marked and are responsible for:

![Figure 5. 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
XRAY FOR JIRA CLOUD

to render the views and is sub divided 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 are specified the authentication data and URL of target instances so that Atlassian Connect Express (ACE) can automatically register the add-on;

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

8) The `__test__` folder holds all the jest tests, this tests focus on testing and ensuring the React Components work as intended.

After understanding all that concepts, Xray for JIRA Cloud development has 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 its 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 that 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 this calls from the client to the server its 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 like: centralized logging, interface internationalization (using `i18n` module from NPM) and performance and usage metrics collecting, the first version of Xray for JIRA Cloud was complete.

4 INFRASTRUCTURE SETUP & DEPLOYMENT

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 three 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 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 Platform as a Service (PaaS) where we provide the artifact, in this case a Docker image, and it managed deployments and scaling automatically[65]. 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 us to run our application in a containerized environment and allows to customize the Node.js environment and package it with our dependencies, this gives us a greater control over our application without tying us to the constraints of a traditional Elastic Beanstalk environment. This architecture provides us with 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

Elastic Beanstalk like mentioned before is responsible for updating, auto expanding and shrinking the cluster of nodes that represent the application, this component has smaller components inside, the Elastic Load Balancer (ELB) which is responsible to balance the incoming request across the different nodes, this nodes are being ran in Elastic Compute Cloud (EC2) which basically represents a virtual machine (VM). The Elastic Beanstalk contains 2 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 lastly a Cloudwatch which is used for collecting metrics and for monitoring the health of the EC2s that are running the application. 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. Docker Image lives inside the EC, contains the application and all the required dependencies for it to run. The way this Docker images are created are by using Bamboo, another Atlassian tool, and by implementing tasks that will provide with 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. 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.

5 Results

The main problems were related with the differences between the Jira Cloud and Jira Server applications, mainly the fact that Jira Cloud addons don’t run in the same server as the Jira applications. For this reason, something that was not done for the Server version of Xray needed for be done for it is cloud version and that was an environment that had to be build to run the addon. 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 addon 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 Java API like in the server version of the addon, 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. 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 does not 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?

5.1 Tests

JavaScript testing is normally comprised of three types of tests [11]:

- 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 these kinds of tests.

In order to test the loading time of the modules extended by Xray for JIRA Cloud a software analytics tool called NewRelic [12] was used. This tool was used to calculate the time a request takes to be resolved, on average. It has also available a Dashboard where the results could be consulted and organized by page load time or by throughput. This allows 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.

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. 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 this types of tests are planned to be implemented after the Beta phase of Xray for Jira Cloud starts because similar to the UI Tests this tests should be implemented after the UI is finished to prevent, although
simple to do, to have to always update the JEST snapshots.

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 this both ensures that the REST endpoints are correct and also if someone tries to exploit or break the REST API that this wont happen this could be a serious problem because it would case internal errors in the Node Servers and overload the Servers unnecessarily. This tests were done using Jest framework along with a module called Supertest, the reason Supertest is used it 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 it is respective middlewares. 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 What 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 this 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 [13] and Sentry [14].

6 Conclusion

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. 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 it is server versions so if clients were already used to one of them the transition would again be as smooth as possible. 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 it is Beta phase with real customers using the product.

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.

References