Report on

Python Package for creation Of CI/CD Automation Tools

AND

Improvement of UI for company's Resource Reservation System and Flaky Test Analyzer WebApps

BY

Name of the Student:	**ID No.:**
B. Rishi Saimshu Reddy	2018A7PS0181H

AT

COHESITY

A Practice School-II Station of

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI (Dec 2021)

Acknowledgments

I would like to express my deepest appreciation to all those who provided me with the opportunity to take up this project at Cohesity which gave me insight and knowledge on not only the project field but also on the workings of an IT company. I owe my deep gratitude to my mentors **Gururaj Krishnamurthy, Peter Kirubakaran** and my Manager **Vinay Rao,** who took keen interest and guided me all along, during the course of this project by providing all the necessary and invaluable inputs.

I would also like to thank my college **BITS Pilani** for facilitating this opportunity to work in industries to gain exposure and work experience.

Table of Contents

About the Company and Its Product

Cohesity was founded by Mohit Aron in the year 2013. Long before Cohesity, he worked at Google where he created the web-scale file system which was infinitely scalable and continuously available. He brought the same values to the Cohesity Platform.

Cohesity is a scale-out solution for managing and enabling faster access of the ever growing size of secondary data of a company. This solution is made up of multiple x86 physical server nodes or as a Virtual appliance for Remote Office Branch Offices(ROBO) situations and directly from the cloud. This is the area where we see the major strengths of the cohesity platform against other competitors in the same field. With these resources present, Cohesity is a lot more than just a huge box of dumb storage. Once data is written to the platform there is global indexing and search allowing you to search across all stored data for anything you wish to find. Data stored across the Cohesity platform, not just for backup can be replicated to not only another physical Cohesity platform but to a virtual platform at a branch office or in the cloud, or any other type of S3 or NFS storage.

Part 1: Python Package for creation Of CI/CD Automation Tools

Problem Statement

Automation is at the core of CI/CD. It facilitates processing pipeline commands, inter-platform connectivity, and valuable integrations with 3rd party services. If you take a deeper look at the core, CI/CD Engineers will design and manage the automation that powers continuous integration and delivery operations.

The different members of the CI/CD team at cohesity have made various tools that automated many tasks and made the flow of development smoother for many engineers working in the company. It was observed that there was a lot of duplication between the code of the various tools in use. Thus, bringing with it a lot of issues raised by code duplication. This led to the proposal of making a python package containing all the various functions that are used and might be used in one place so that all the current tools and future ones will use this code. This removes the problem of code duplication and also adds the advantage of easy maintenance of the functions when there are changes or bugs.

Since I was allotted to the CI/CD team at Cohesity, I was asked to work on this project along with other members of the team for the duration of the internship.

Objectives of the project

The requirements from the project to be met are:

- The python package must include functions that help in building automation tools that interact with various third party applications like Jira, Gerrit, Jenkins.

- Make sure that functions works as expected and help in refactoring the present code

- Write this python package in a properly structured way such that it can be easily packaged and uploaded to the company servers ready to be downloaded and installed by the programmers for development

The expected outcomes at the end of the Project is to have a Python package that:

- Helps a programmer to avoid writing the code from scratch every time for frequently used functions while building new automation tools

- Have a Python library that helps in refactoring the cohesity codebase with the library functions and removing all the duplicate code and the dependencies that it carries thus improving the code quality

Literature Survey:

Why use Packages/Libraries? :

There will be times when one is planning to make a package where they deliberate on the need to put in time and make a library and maintain it over writing the required functions quickly then and there, but nobody gets ahead by reinventing the wheel unnecessarily. In other words, writing functions again and again is not only a waste of time and resources, but it bulks up the codebase and makes it harder to manage. The best programmers focus on getting things by reusing codes as much as possible.

It is common knowledge that thousands of man-hours go into creating packages, but this ensures that they're bug-free in all kinds of conditions, making them efficient, and adding features to them over time. Even if you recreate some functions of a package on your own, you will not be able to match all those things (or if you do, you will really be wasting a lot of time that you should have spent on getting other things done.

Yes, in the short-term, it seems easier to write your own functions rather than figuring out how a third-party package works, and setting things up appropriately for the package. However, in the long-term, the time saved by using the library will be worth more.

Few advantages of using Packages/Libraries :

- We can identify our components uniquely.
    - As a part of the process of creating a package, when you break down functionalities to make functions, you are enabling the ability to trackdown the details about what is happening and where and when it’s happening when you use a collection of these functions to create a new functionality in your application.
- Naming conflicts can be solved easily.
    - There may be cases where you need to have two functionalities but the same name for a function, this naming conflict can be solved by adding the two different implementations into different sub-packages and call them along with the sub package they belong to, thus avoiding the confusion
- Improves the Modularity of the application
    - With different independent functionalities made into functions and added into packages first, it helps in creating new functionalities by adding many functions like blocks. This nature of you application being modular helps in faster development of any functionality
- The readability of the application will be improved.
    - Breaking a complicated functionality and replacing the insides with smaller functions which are logically named based on what it does helps reduce the clutter, thus making it easier for a reader to understand what is happening inside the function.
- Maintainability of the application will be improved.
    - Maintenance of an application becomes faster because the functionalities of an application is a group of smaller functions defined in the package, this makes it easier to pinpoint the source of a bug and take care of it.

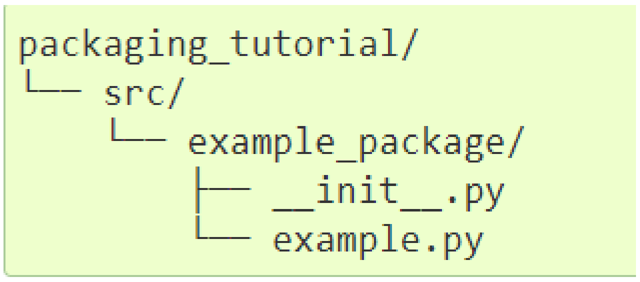
Python Package Structure:

All python packages have a similar structure. The following is a detailed dive into the specifics of the structure of a python package.

When your Python codebase keeps growing in size, it becomes unorganised over time more often than not. Having all the code in a single file makes it difficult to maintain as it grows. At this point, **modules** and **packages** help in organizing and grouping your content into files and folders.

- **Modules** are python code files that help to organise related functions or classes or any code block in the same file. The best programming practise is to break up a large python code file into modules of 300-400 lines of code each
- Grouping similar types of modules into one directory makes a **Package**. The package folder contains related modules and an **__init__.py** file that is used for optional package-level initialisation.
- Should the situation demand for our Python application, you can consider to group your modules in sub-packages such as **doc, core, utils, data, examples, test.**

So a snapshot of a sample python package structure would look something like the following :

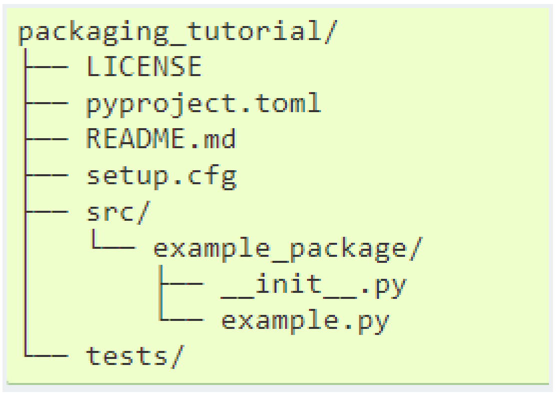


```
packaging_tutorial/
└── src/
    └── example_package/
        ├── __init__.py
        └── example.py
```

The above snapshot is the boat basic form of a python package, we need to add a few more files/folders that are important for making the python package be ready for distribution like:

- **tests/** directory that holds all the pytest files used for testing the proper functioning of the package's functions
- **pyproject.toml** tells package building tools (pip, build, etc) the requirements to build the project.
- **setup.cfg** is the package metadata that tells setuptools about the name and version of your package as well as which code files to include. Eventually much of this configuration may be able to move to **pyproject.toml**.
- Other files like a **README file**, **LICENSE** are added to give the installers the details about information they might need to know before/while using the package.

After which the structure of a package would look like the below figure. At this stage, the package is ready to be uploaded and distributed.



```
packaging_tutorial/
├── LICENSE
├── pyproject.toml
├── README.md
├── setup.cfg
├── src/
│   └── example_package/
│       ├── __init__.py
│       └── example.py
└── tests/
```

Here on, the package grows to become more useful by adding more functions or by adding new sub-packages itself to the package.

Project Plan:

The initial stage of the project that lasted about a week was spent on going through the codebase to make a note and realise in general what type of functions are being used often and for what purposes, post which the final structure and requirements of the python package was decided.

In the next stage, The functions were divided into sub-packages based on what it was dealing with and the work was started on building those functions. This stage lasted for three weeks, where basic functions like making API calls to gather information and do CRUD operations at the various third party applications(Jira, Gerrit, Jenkins) were written and added to the respective sub-packages.

With the basic functions ready, The next week was allocated for getting the directory upto shape of a python package. The Setup details were written after which the directory was packaged and uploaded to the Cohesity's artifactory.

The subsequent step from here would be the testing of the library before it can be released for company wide use in the production code. This task is allocated to be done after completing a few rounds of code review on the present package by the team members. This is expected to be done by the end of next month.

The Last stage is refactoring the existing Production codebase with the functions present in the above developed python package. This stage will take up a lot of the time since the codebase to be refactored is large. This cycle of writing new functions then testing then refactoring continues making the library more larger and robust.

Results:

At the time of writing this report, we have the python package in the proper structure that can be compressed and uploaded onto our company artifactory. We tested the installation of our python package after downloading from our servers and installing in a virtual environment and everything works as expected.

We wrote sample python programs/automation tools where we imported our installed python package to check if the package has the right structure and if all the functions called are being discovered properly in their respective sub-packages. Everything ran and gave the results as expected.

While refactoring some code to check the working of our package, we could reduce about 30 lines of code at least (not always) in many of the files we took up. This number is only expected to go up as we keep using it in other files and also adding more functions to the library

Conclusions:

In the process of building this python package for our internal use , there were many takeaways that have been a really valuable educational experience. The following are a few reasons why making a package and adding to it is more helpful in the long run than writing your functions then and there where needed.

The first important learning was the need of a library for regularly used functions. Just like how a function in a standalone program helps reduce the reuse of code while also making the code easy to debug and maintain, a package/library is the same but on a much grander scale. This project was the means through which I got a sense of this scale. Refactoring the code made us realise just how many times a particular function needed to be defined in many different tools. We noticed we could remove about 100-200 lines of code from each automation tool on average.

Maintenance of the code became very easy after refactoring code with the package we made. The errors we found in the function we wrote were easily fixed on the side of the package while the many places where the function was called could be overlooked without any problems.

Thus from experience it is recommended to make a package despite the initial setup taking some time, because it will not only help you skip writing code from scratch, but to write uncomplicated code that is also easy to debug and in general in the long run improve your efficiency in work output.

Key Challenges:

There were quite a lot of challenges and bugs that were faced while making this python package, which took time and some searching to find solutions and overcome those. The following are few main challenges faced amongst other small problems

The most challenging part of the project so far has been in building the structure of the package. The best way is to make it completely on your own from scratch such that it meets the project requirements and has nothing else extra. But this decision we initially made helped us realise that it is complicated setting the parameters for various package building tools so that we can start off with a basic structure package. Although opting to use widely available python package cookie cutters/ templates makes setting up a basic structure very easy, it comes with a lot of extra files. Nonetheless, it's more cost efficient to remove the extra files than to spend more time installing and setting up the required tools for building the packages by ourselves.

The other noticeable challenge faced was while the package was being tested to check whether it was working like we wanted it to. The Package had functions that were making API calls to various applications and since they depend on the internet for carrying them out, proper internet connection to the servers became a limitation. Searching for solutions made us land on the technique of mocking using Pytest. During the testing process, mocking returns the expected API call response instead of actually pinging the servers. This eliminates the dependency of the internet for testing while also reducing the number of API calls to the server thus reducing the load on the servers.

Part 2: Improvement of UI for company’s Resource Reservation System WebApp

Part 2.1 : Resource Reservation System

Problems statement

This issue came through an admin’s use case on the web app where the admin might have to create/update/delete reservation details of more than one resource as per a request by a user. As the process of fulfilling this request, the admin is required to enter a few details that would be updated in the reservation details of the resource. These details that the previous system asked the admin to enter while updating the reservation details individually were common for all the multiple reservations that the admin had to do as per the user’s request

Hence it was decided that there needs to a way such that the admin would be able to enter the details and make changes to the reservation details of all the resources only once

Objectives of the project

The requirements from the project to be met are:

- Change the UI (viewable only by Admins) to be able to select multiple resources to operate on
- A “select all” option to select filtered list of resources in one click
- New APIs that would take the list of resources instead of one resource and the new details common to all and make the required change in the data at the database.

The expected outcomes at the end of the Project:

- Enter common details only once for changing the reservation details of multiple resources in a single operation
- Improve the usability by reducing the number of steps required to carry out the request

Plan of Approach

1> Discuss with a few admin users about what the UI should be like to change the flow of reserving multiple resources

2> Go through code base and get to grips with the structure of the webapp and then get an idea of how to make the change

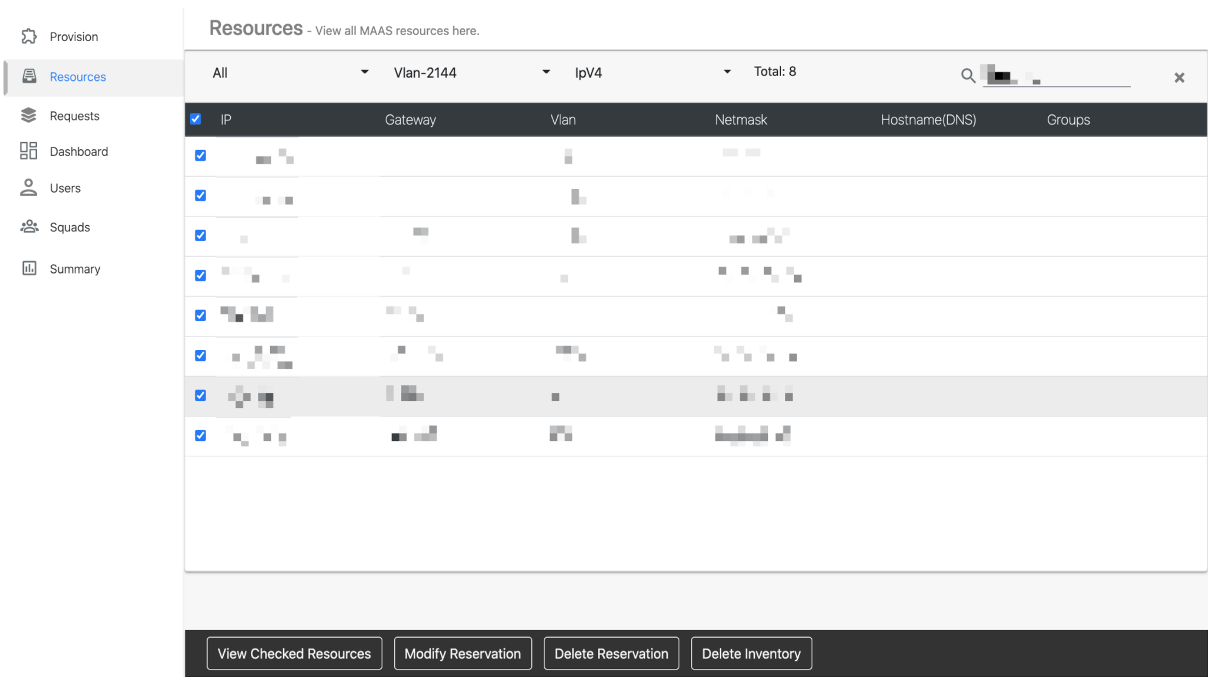
3> Make changes to the codebase such that it can be run locally and be used to test out the changes made

4> Implement the UI changes that were discussed and test out the features like the new UI appearing only for admins, “Select all” button.

5> Look at the existing apis that change the reservation details of a resource and referencing them to create new apis that would change the same details but for multiple resources at the same time

6> Test the apis to make sure the change is updating the database with new details for and the resources and the same is being reflected on the UI.
7> At present the change is to operate on the Static IPs, So the last step before pushing the change to the main server is to make it modular so that it can be applied to other resource types .

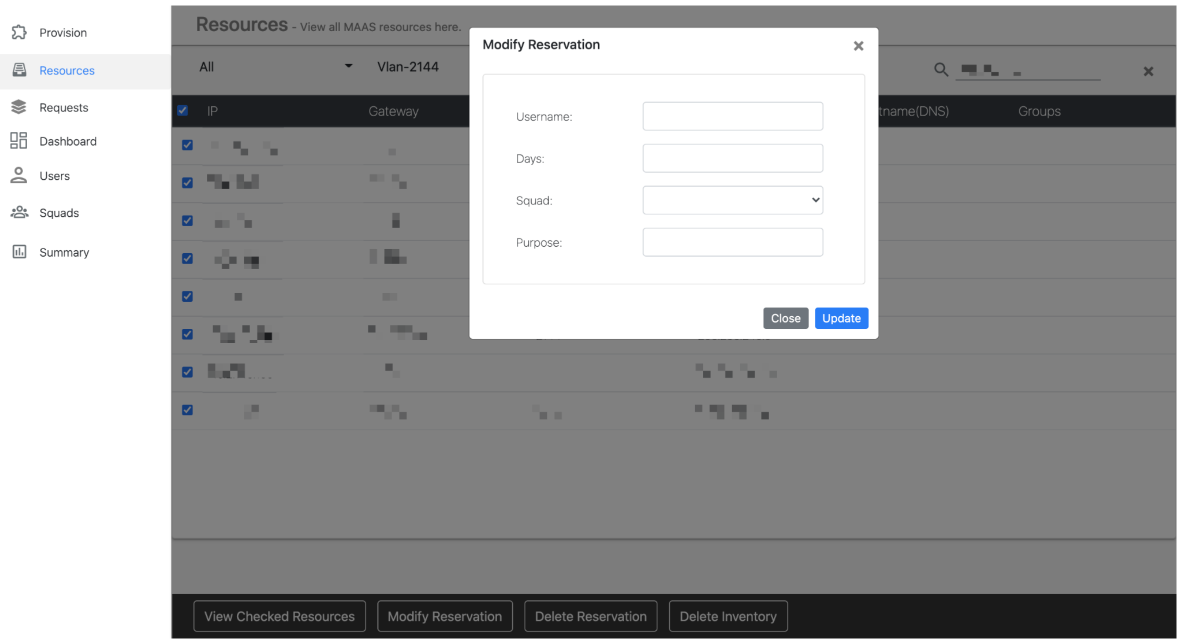
Few ScreenShots of the Completed Task



☑	IP	Gateway	Vlan	Netmask	Hostname(DNS)	Groups
☑	[illegible]		[illegible]	[illegible]		
☑	[illegible]		[illegible]	[illegible]		
☑	[illegible]	[illegible]	[illegible]	[illegible]		
☑	[illegible]	[illegible]	[illegible]	[illegible]		
☑	[illegible]	[illegible]		[illegible]		
☑	[illegible]	[illegible]	[illegible]	[illegible]		
☑	[illegible]	[illegible]	[illegible]	[illegible]		
☑	[illegible]	[illegible]	[illegible]	[illegible]		

(Parts of the picture have been blurred out for security purposes)

The checkboxes on the left of the table head and its rows, and the option bar at the bottom are the additions to the reservation system that are visible only by the admins. Checking the checkbox in table head selects all the visible rows in the table and vice versa. The selections stay until the admin carries out any of the operations available at the bottom


Resources - View all MAAS resources here.

Modify Reservation

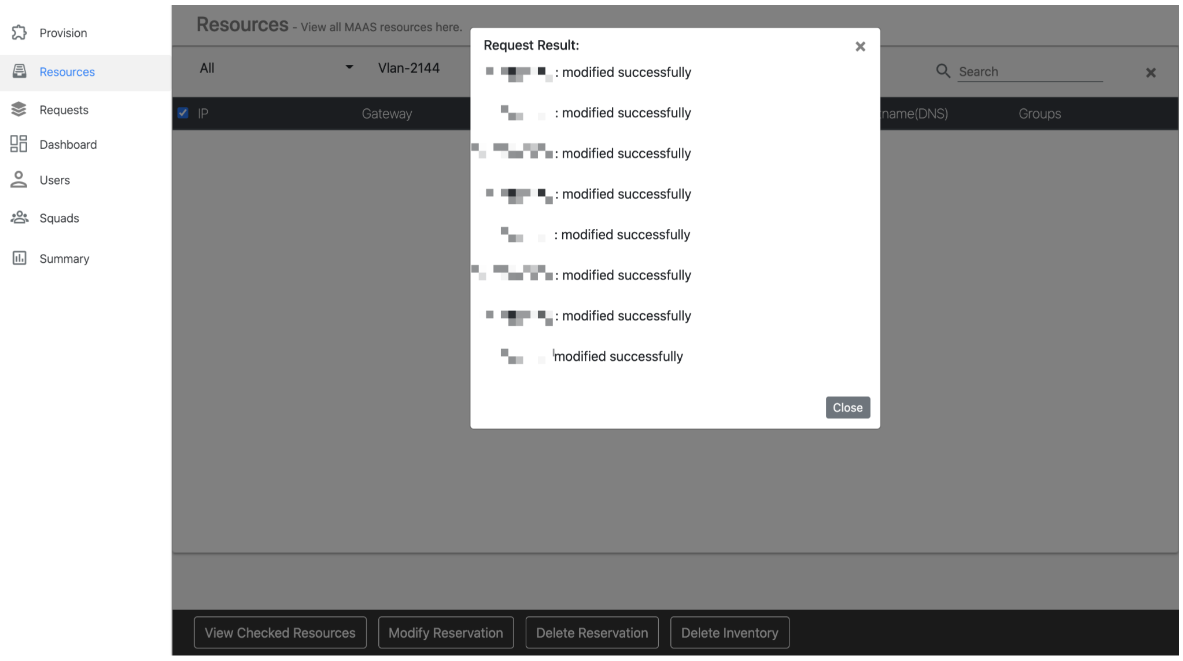
Username:

Days:

Squad:

Purpose:

On clicking either of the options at the bottom, a designated box appears with more options. After filling all the fields (should the operation require), on continuing… Another dialog box appears with the results of the operation for each of the resources (seen below). The change can be seen being reflected in the front-end when the page refreshes after closing this dialog box


Resources - View all MAAS resources here.

Request Result:

[redacted] : modified successfully

[redacted] : modified successfully

[redacted] : modified successfully

[redacted] : modified successfully

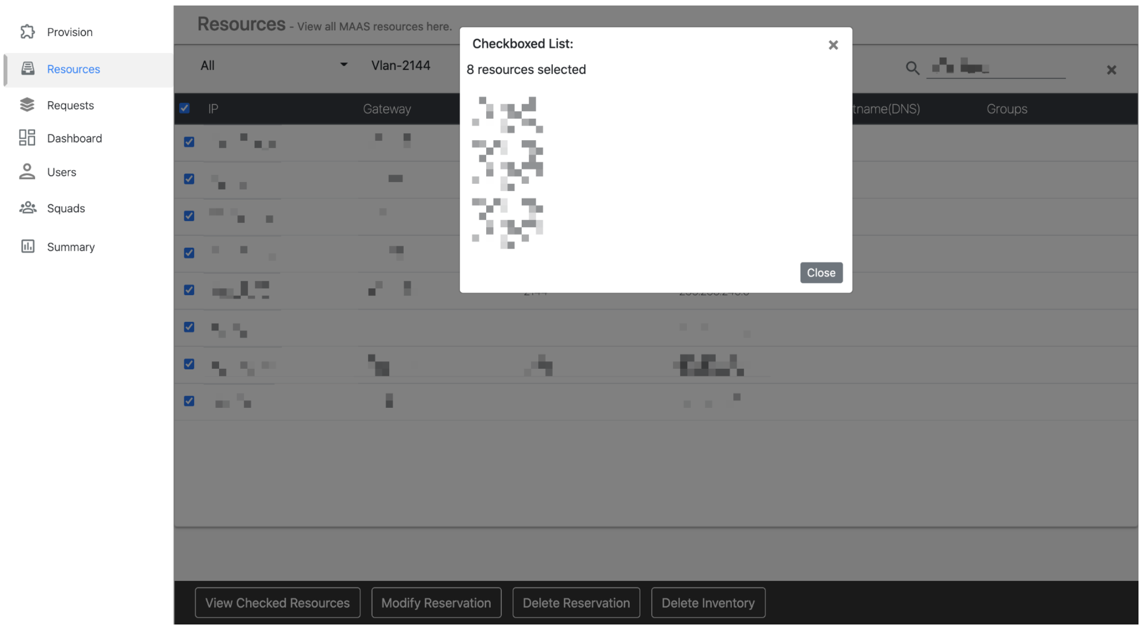
[redacted] : modified successfully

[redacted] : modified successfully

[redacted] : modified successfully

[redacted] modified successfully

To enhance the usability a bit more, I took the liberty to add a view checked resource button as seen below which would allow the admin (as the name suggests) view the selected resources before carrying out the bulk operation



Results Achieved and Conclusion

The above screenshots stand as proof of the results achieved which are:

- Showing the new changed UI only when accessing as an admin user
- “Select all” button feature helps in saving time spent by selecting all the filtered list of resources at once instead of selecting each resource individually.
- The APIs are working as expected and the change is being reflected in the UI.

To conclude, we can say that this change that enable an admin to perform bulk operations, the help that it provides is profound when dealing with huge number of resources

For example:

Let’s assume that an admin received a request to reserve 15 particular static IPs to be reserved under Username X of Squad Y for N days for “ABC” purpose

If we are entering those 4 fields for reserving 15 resources only once, that would save the admin from doing at least 56 extra steps when done individually.

This saves a lot of time spent by the admin in completing the request as they won’t have to let the page refresh and get new data after each update and then make the change for another resource

Part 2.2: Improvement of the UI for and Flaky Test Analyzer WebApp

Problem Statement

Flaky test Analyzer is a tool used inside the company that checks whether a test is flaky or not. To view the data it produces, a reactJS app was built. Flaky Test Analyzer's front-end displays the test trains, test suites and the tests all in tables.The problem that arises is due the sheer number of entries in the database that a query from the app fetches. This causes timeout more often than not. Even if the query fetches all the results successfully, the table displayed has so many rows which makes it difficult to read the table. Solution for this issue was to paginate the tables.

Objectives of the project

The requirements from the project to be met are:

- Paginate the table with a option to skip to a particular page number
- Query the database in a paginated sense so that loading a page doesn't take time
- Implement a "Go back" button as the present webapp doesn't allow a user to go back to the prev step
- Other UI changes to make the app more responsive and usable

The expected outcomes at the end of the Project:

- Load pages faster
- Make the webapp easy to get your information from.

Plan of Approach

Similar to the approach in the previous part, the plan of approach for this task is:

1> Go through the existing codebase of the webapp and decide the approach to implement paginated table

2> Write the new apis to query the database in a paginated way

3> Create a <div> that gives the user the option to jump to any particular page.

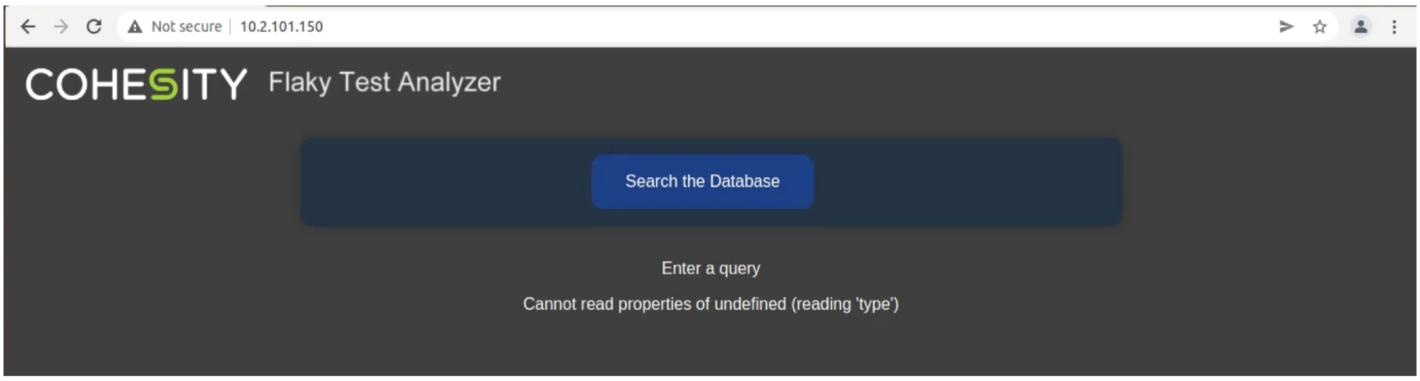
4> Collect feedback from the users and make the changes

5> push the change to the server and host the webapp with the new changes

Few ScreenShots of the Completed Task

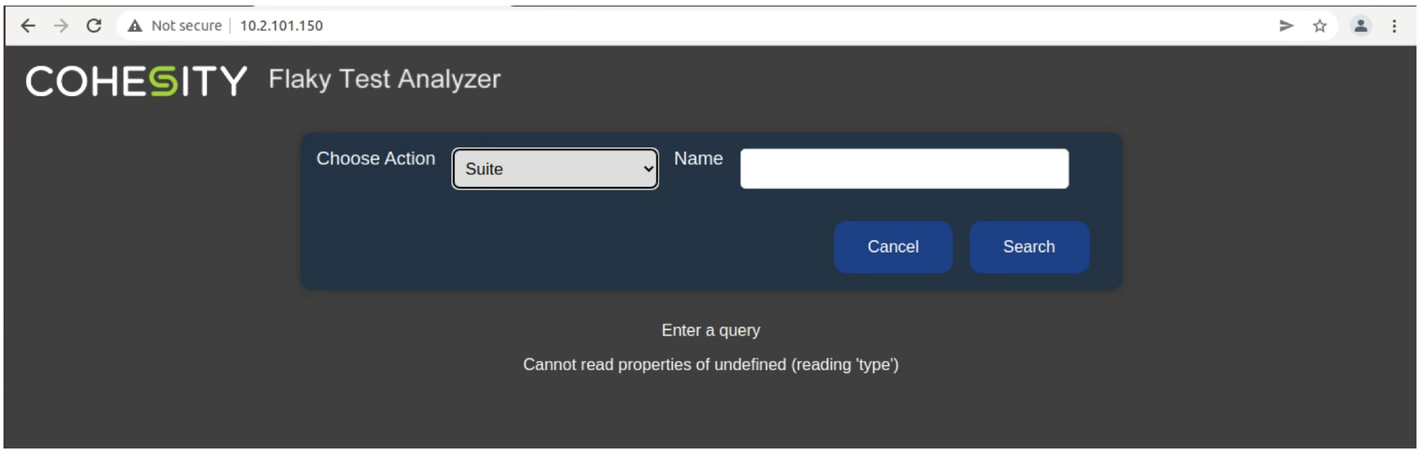
> Landing page and query form:

Previously



Enter a query

Cannot read properties of undefined (reading 'type')



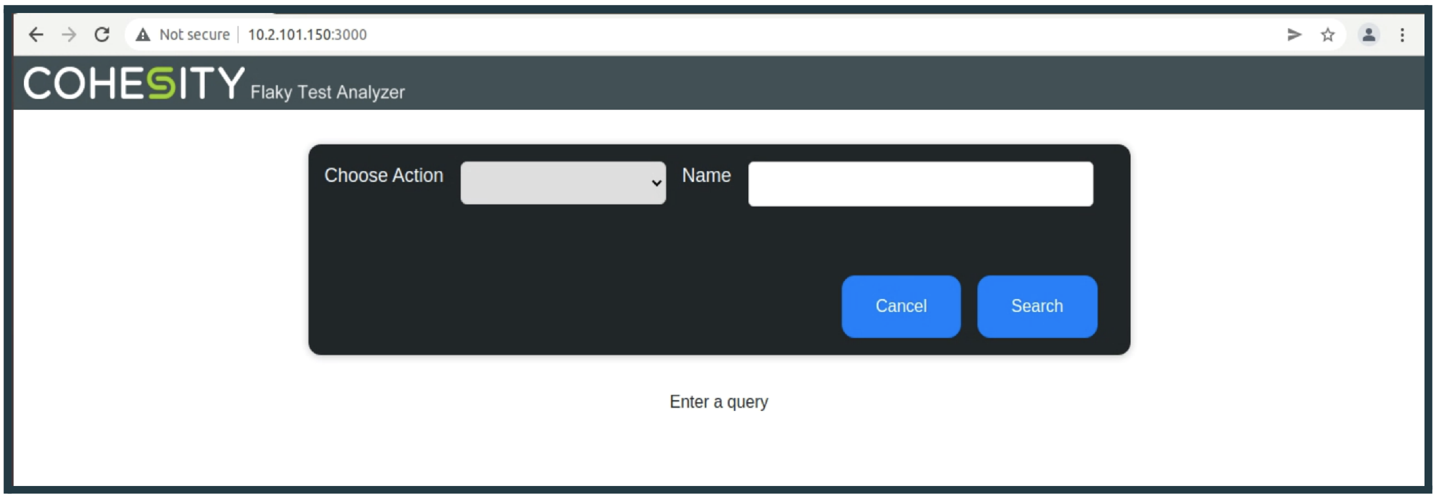
Choose Action: Suite

Name

Enter a query

Cannot read properties of undefined (reading 'type')

Now



Choose Action

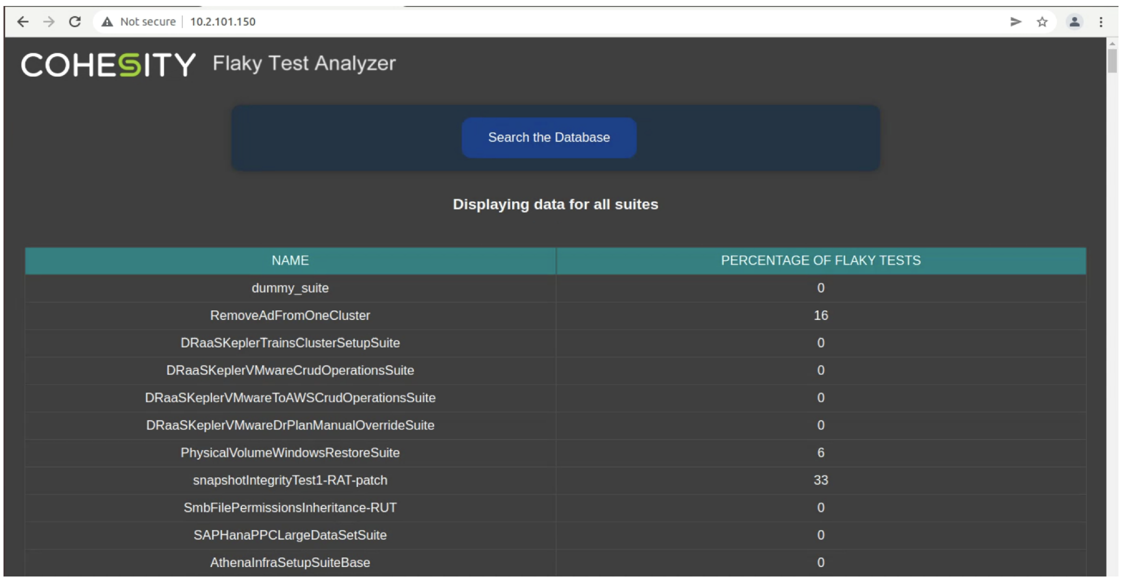
Name

Enter a query

On the landing page the user had nothing to do but press the button and then the form would appear. The landing page also showed errors even before querying started. All this was fixed and the new landing page is the form directly. The theme was also changed to match the other WebApps the team had built

> Result of a Query:

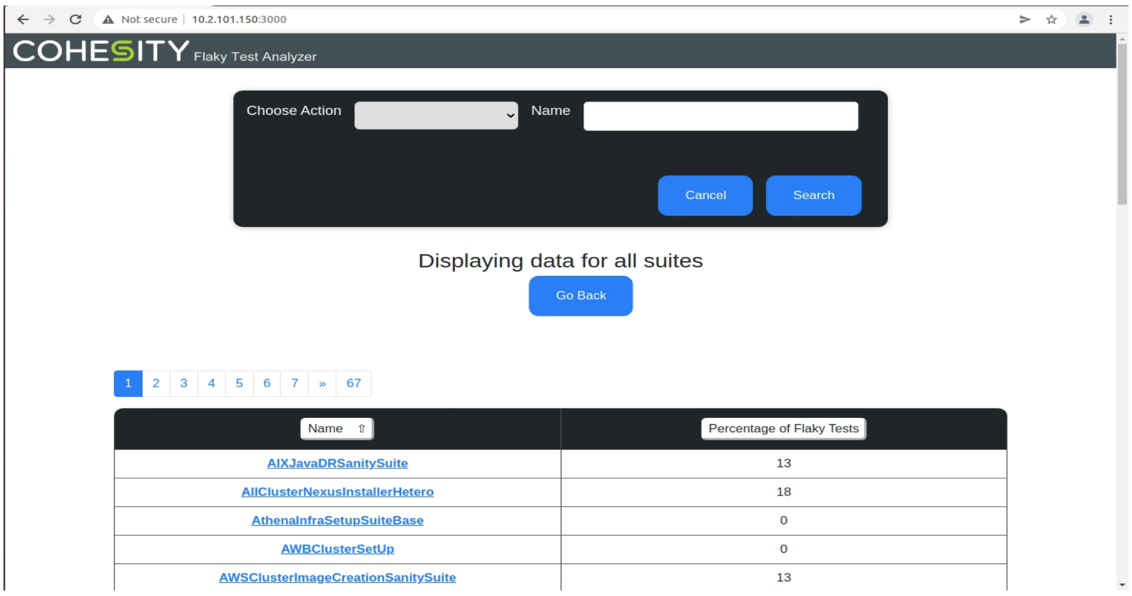
Previously



Displaying data for all suites

NAME	PERCENTAGE OF FLAKY TESTS
dummy_suite	0
RemoveAdFromOneCluster	16
DRaaSKeplerTrainsClusterSetupSuite	0
DRaaSKeplerVMwareCrudOperationsSuite	0
DRaaSKeplerVMwareToAWSCrudOperationsSuite	0
DRaaSKeplerVMwareDrPlanManualOverrideSuite	0
PhysicalVolumeWindowsRestoreSuite	6
snapshotIntegrityTest1-RAT-patch	33
SmbFilePermissionsInheritance-RUT	0
SAPHanaPPCLargeDataSetSuite	0
AthenaInfraSetupSuiteBase	0

Now



Displaying data for all suites

Name ⇧	Percentage of Flaky Tests
AIXJavaDRSanitySuite	13
AllClusterNexusInstallerHetero	18
AthenaInfraSetupSuiteBase	0
AWBClusterSetUp	0
AWSClusterImageCreationSanitySuite	13

Upon successful retrieval of data, all the data was presented in one table with too many rows. This made reading the table harder. After the change was done (as you see in the second image) only the first X number of rows have been queried and shown in the table and upon selecting any of the page numbers on top of the table, the rows pertaining to that page would be queried and shown. This removes the load of getting all the data at once, thus leading to loading of the page faster. To Improve the efficiency of the user in getting the required information the columns of all tables were added with the feature to sort in increasing/decreasing order alphabetically/numerically. The previous version did not allow the user to go back to the previous step and had to start a query from the first step, this was also implemented in this change to improve the usability of the webapp.

Key Results

The Results achieved so far:

1> The pages which took over 20 seconds to load earlier take less than a second to load right now

2> The paginated tables makes it easier to read the table and the page numbers lets you skip entries and go directly to the page required

3> The “Go back” button now enables the user to go back to prev page instead of starting the search from the fist step

Key Challenges

The only part of these two tasks that felt challenging was the integration of my code for the required changes with the existing code. During this I encountered quite an amount of problems and bugs taking a lot of time to investigate, find fixes and then actually fix them. Using GoLang for the APIs, ReactJS for the Front-end part for the first time definitely made matters a bit harder, but seeking help from my mentor and colleagues, we did finally solve a lot of those issues.

Key Learnings:

This internship till now has been one of the most valuable educational experiences for me as a student. On the technical side, I was introduced to the field of CI/CD. Me being a student interested in pursuing a career in Human-Computer-Interaction. Part of being in the CI/CD team is to help the in-house engineers to carry out their work smoothly and more efficiently by providing them with solutions and tools that fix their problems or automate them. I find this aspect of CI/CD team's work to align with the ideals of HCI. Since I joined the CI/CD team, I was given tasks to write scripts, APIs, etc because of which I had to learn about them and the languages to use for them. Until this point, I have learnt a lot more than I knew on shell scripting and development using python.

On the other hand, I also gained exposure to work life in the IT industry. The most important business learning has been the benefits of SCRUM. This is a framework that teams follow because it encourages them to learn from each other's experiences and self organize to solve problems and also look back at previous successes and failures to continuously improve themselves. The importance of daily standup meetings was the most fascinating thing I learnt after starting this internship. During a Code sprint, the daily standup meet gives every team member the information regarding the status of all the other members and also gives a team member to put their problems or doubts in front of the team to get help thereby saving a lot of time because of all that collective experience working on that issue and solving it quicker than individually. All this puts the team constantly updated on the status of the sprint and adapt as required.

References:

The following pages and blogs (not int order) were referred to while writing this report

- https://www.definetomorrow.co.uk/blog/2018/3/10/an-introduction-to-cohesity-vretreat
- https://circleci.com/blog/cicd-engineer/
- https://chercher.tech/python-programming/python-packages
- https://packaging.python.org/tutorials/packaging-projects/
- https://qr.ae/pGSKI8
- https://towardsdatascience.com/learn-python-modules-and-packages-in-5-minutes-bbdfbf16484e