20 years in a nutshell

## 👶 Heyy guys, I am Sindhu

Bit of context for this pic ❤️:

This was the first time I went out to celebrate after I landed my internship.

I am currently a SWE Intern in the Phantom 🧛 team of JSM Automation.

I am working in the backend team which has been such a big dream of mine.

This is my intro blog to help you all get to know  at least one thing about me .

I study at IIIT Allahabad in the IT- Business Informatics Branch.

Looking forward to a fun time 🎉 with my dream company

## 😉 Some Un-Fun Facts about Myself

- I **broke my foot** while running to catch my school bus and still went to school for some reason and suffered in pain the entire day . ⁉️
- I got low marks on an assignment in the 5th grade where I wrote about my role model- Abdul Kalam but **he died before they graded the test**, so I was marked wrong for all of my tenses. 😩💀
- I almost lost my eyesight **twice**-once by applying **eucalyptus oil** to my eyes and once someone shot me with a **nerf gun** near my eye. I survived successfully  ( I wear contacts though 🙃).

## My Professional Experience

I have previously worked at **Prutech Solutions** in Hyderabad. This internship was my first time in a corporate environment and in a lot of ways shaped my journey to reach Atlassian.

I learned how to document every little thing I was doing, understand the user requirements and schemas before actually implementing even a line of code. 🤯

I truly understood then the power of how a few lines of code can have such an impact and also have so much thought process and deliberation behind it.

This allowed me to understand problems in a much more different light and not just see them as something to finish but as something that can be implemented in the grander scheme.

## 👥 Me and my People

I am actually a US citizen (surprise !!), my family moved pretty early on so I actually never experienced being a NRI or anything.

I spent the majority of my life with my grandparents and family in a joint family setup and it was the best - always having someone to play with was amazingg!!.

All these people and my family have shaped me in ways that I can't fully explain.

My parents and them are my biggest support system 😍

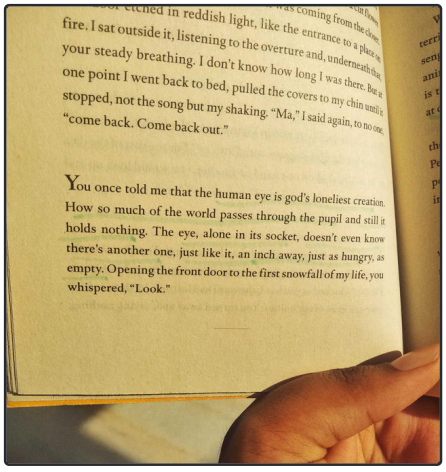
Now as I am more grown up, I value my friendships- especially my childhood friendships and my relationships with family and now with all my fellow interns and colleagues too :)

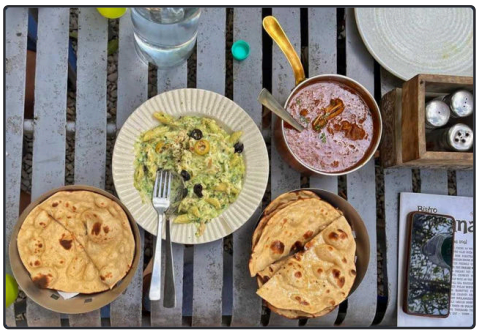
## 🪅 My Interests in a Box

| 😋 **Foodie Alert** | 🛬 **Places I've been** |
|---|---|
| I love trying out new food -from cafes to roadside chat places<br>1. My **Grandma's Coconut Egg curry**<br>2. My **Mom's Bean fry**<br>3. Turkish Milk Cake | I love travelling and just have got the freedom to fully plan trips on my own<br>1. Taj Mahal<br>2. Mussoorrie<br>3. Mysore<br>4. Panchgani |

| | |
|---|---|
| 4. Mutton Curry in some place in Rishikesh<br>5. My Chilli Oil Maggi<br>6. Summer fruits 🍉🥭<br><br>These are just the stuff of the top of my head lol | To more and more adventures in the future.<br><br>Maybe even might solo travel at some point 👱‍♀️ |
| **📚 Books I recommend** | **🎬 Movies I recommend** |
| I am a huge bookie 👌<br><br>1. Tuesdays with Morrie<br>2. Animal Farm<br>3. The Kite Runner<br><br>My **mom read me books every night** ever since a child, so this hobby has been inculcated in me since basically birth.<br><br>I also try to participate in the **Hyderabad book fair** and **Load the Box** program whenever i can.<br><br>Pls give the above books a try **especially the 1st** , they are life changing. 😁<br><br>Reach out to me :) to **keep me accountable for reading new books** since I have been slacking off on reading ever since I started college. | Also I watch so many tv-shows and movies🤓<br><br>1. The Office( US one).<br>2. Fleabag<br>3. The Family Man<br>4. Dhootha<br><br>1. Barfi -Hindi<br>2.C/O Kancherapalam - Telugu<br>3.Super Deluxe -Tamil<br>4.Shawshank Redemption-English<br><br>I have stayed primarily in Hyderabad and the film and **movie culture in the Andhra and Telangana states is unmatched.**<br><br>Everyone should atleast once experience watching a **movie in a single screen theatre** with thousands of fans hooting, throwing torn newspapers at the screen. The excitement and joy just transfers to you. 😝 |

## 🤹‍♀️ My Hobbies in Pics





## 👋 Get in Touch

💌 My email

@Sri Sindhu Veerathu

My Linkedin

sindhu.194

## Closing Thoughts

This first week has been a wonderful time for me , my mentor- @Gaurav Arora and the Early Careers Team- @Lakshmi K P and @Lubna Vaseeq have been great at making me feel more at place and helping me further improve myself.

My fellow interns are also super fun and collaborating and conversing with them has been a breeze.

---

New to Confluence? Check out the following resources to learn more.

- Confluence Essentials Guide
- Get Started Quickly with Confluence from Atlassian University

# Internship Documentation

| | |
|---|---|
| Name | *Sri Sindhu Veerathu* |
| Project Name | *Playbook Usage Tab [BE]* |
| Duration | May 19, 2025 - Jul 11, 2025 |
| Mentor | @Gaurav Arora |
| Buddy | @Harshita Goyal |
| Manager | @Sandeep Ravichandran @Uğur Turan |

## May 19, 2025 & May 20, 2025 - WEEK 1

**TASKS :**

1. Setup my laptop with the stated requirements like WARP , Okta Verify, and making sure my device is compliant and ready for further tasks.
2. Completed orientation with my recruiters- learning about Atlassian evolution,Atlassian Values, Atlassian products, the new Workspace, the Atlassian Foundation and last but not the least Community Guidelines.

**LEARNINGS :**

1. I ensured my laptop is compliant and secured by installing all the required tools , this step was crucial for secure access to the Atlassian systems and for meeting compliance standards
2. I learnt about the company's history and growth and how its products and culture has developed over time.
3. I got a deep dive into the company's core values that guide the decision making and behaviour of every individual in Atlassian.
4. I gained an overview into the main products of the company which is essential for understanding my future work.
5. I learnt about the new modern workspace which encourages collaboration. and also took a keen interest in the philanthropic initiatives of the company and its commitment to social responsibility.
6. I learnt the expected standards for behaviour within the Atlassian community, making sure I contribute to a respectful and inclusive environment.

## May 21, 2025

**TASKS :**

1. Completed product walkthrough which included learning about jsm, automation rules and the new playbook feature and my role in it. New Hire Onboarding for Playbook
2. Completed product setup which included creating a jsm site, making automation rules and gaining access to playbooks and setting up a few mock ones on my site, also included upgrading my jsm site to a premium subscription. New Hire Onboarding for Playbook
3. Attended a Sips and Tips session with my recruiter and my intern pod.

**LEARNINGS :**

1. Product Walkthrough helped me gain an insight into understanding JSM- it's concepts like projects, issues, work and request types and user groups.
2. Configuring automation rules was something i explored heavily- looking into all the different trigger, actions and conditions.
3. Playbook feature is something I prioritised to understand since my intern project is related to it- it is a collection of manually triggered automation rules and instructional rules that can be triggered independently and shows execution logs and in the future offers usage insights too.Its helpful for standardizing procedures -making them repeatable and easy to execute for the team.
4. Product Setup- creating a JSM site enabled me to explore and try out automation rules, playbooks etc.
5. I gained access to the Playbooks feature by upgrading my subscription and set up  mock playbooks to test out the features.
6. Today has allowed for me to have some hands-on experience in JSM and I understood about the rolling out ( essentially feature gates ) that are implemented for a new feature like Playbooks.
7. Partcipated in the Sips and Tips Sessions with Lakshmi and my intern pod- was a great bonding activity and helped me connect to ppl working across various teams.

**CHALLENGES :**

1. Faced some trouble in accessing playbooks on my JSM staging site since it didnt roll out to my site so after expressing this Gaurav helped me manually add my cloudId to gain access to the playbooks feature.

## May 22, 2025

**TASKS :**

1. Completed the software setup by installing IDE- IntelliJ, Bitbucket, Docker, Maven, Java, Postman, Homebrew, Github Copilot, Atlas, CLI, and also setting up the Bitbucket Cloud Account to gain access to Atlassian repositories. Software Setup
2. Completed Absorb Courses - Security Awareness Training for Newlassians, EEO Training for Newlassians (based outside the US), Public Company Policies for Newlassians, FY25 Privacy Awareness Training and Incident Reporting & Management - Foundations for Atlassians.
3. Attended the Debugging your Internship session.

**LEARNINGS :**

1. Installing Postman allowed me to test out some rest APIs and practice my knowledge in backend development.

2. By configuring my Bitbucket Cloud Account, I had the opportunity to go through the code base and gain a bit of understanding of all the work being done.
3. I went through the basics of GraphQL to prepare myself for coding in the further weeks.
4. The debugging session was very insightful and I left the meeting with a ton of tips and confident enough to complete my project successfully but also take help when I need it.

**CHALLENGES :**

1. Face a couple challenges in setting up the Bitbucket account- especially the SSH keys but with time I resolved these on my own by thoroughly going through all the Confluence Pages related to SSH keys.

**May 23, 2025**

**TASKS :**

1. Completed the playbook backend onboarding- cloning the automation-playbook repo, setting up nebulae and debugging and testing nebulae within a sandbox environment. Local Development, Debugging and Testing | STEP 2: Setup nebulae
2. Made my Intro Blog Confluence page for the Intern Bingo Challenge. - 20 years in a nutshell
3. Participated in the daily standup and made work items on the phantom scrum dashboard.

**LEARNINGS :**

1. By cloning the repository and branching it, I could make sufficient notes in order to understand the large codebase.
2. Setting up the nebulae and a sandbox environment gave some experience in preparing a backend development environment.
3. Intro blog was a fun experience since I could really let my personality shine through and connect to more like-minded individuals.
4. Daily Standup was a way for me to gain insights into how a whole team works together and all the varied tasks each one performs which helps the overall goal, made me want to contribute more towards this shared goal.

**CHALLENGES :**

1. I couldnt figure out how to add my work items on the phantom scrum dashboard - both Gaurav and Harshita helped me in setting that up and this allowed me to add more tasks for myself on the dashbaord to keep myself accountable.

**May 24, 2025**

1. Worked on the product onboarding exercise -faced some issues in rule creation specifically with assigning user roles and workflow transitions.
2. Worked on improving my knowledge on GraphQL.

**May 25, 2025**

1. Went through Git commands
2. Completed  Secure Code Training - Foundational Absorb Course

## May 26, 2025 - WEEK 2

**TASKS :**

1. Resolved my doubts in product onboarding exercise and successfully completed all its tasks. Onboarding - Playbooks for JSM - 101
2. Completed the Rules of Behavior Absorb Course
3. Completed playbook backend onboarding by setting up lanyard authentication, making my activation id and playbook files in the repository and setting up postman and importing environments to utilize it for graphql queries and mutation. Local Development, Debugging and Testing | Helpful link for Lanyard Authentication and ASAP token generation

**LEARNINGS :**

1. I had a couple doubts regarding the product onboarding exercise due to some of the ambiguity of the actions asked to perform and the inclusion of workflow transitions which was a bit out of scope but nevertheless explored and learnt from. I resolved these doubts from Gaurav and commented some doubts for clarification.
2. By setting up my activation id files- it was crucial for simulating the correct environment and enabling backend services to recognize my test setup.
3. I learnt that Lanyard authentication was essential for making authenticated requests to Atlassian service(ie through a secure port).

## May 27, 2025

**TASKS :**

1. Worked on testing GraphQL queries locally on postman.

**LEARNINGS :**

1. Faced issues in testing GraphQL queries locally :

- The GraphQL schema had some errors showing up which I resolved through going through documentation.
- I couldn't find the scope ID -mandatory for the queries , which I resolved with the help of Konica

2. I worked on increasing my knowledge in GraphQL.

**CHALLENGES :**

1. The only issue still remaining was that the query wasn't going through due to a 403 forbidden error stating I didn't have admin permission, I tried going through docs and even collaborated with Konica and Harshita to resolve this issue but we couldnt figure it out.

## May 28, 2025

**TASKS :**

1. Tested GraphQL queries locally on Postman.
2. Updated the onboarding documents with instructions regarding the issues I faced to prevent future confusion. (from Obtaining User Context Token onwards) Local Development, Debugging and Testing

3. Drafted the rough outline of GraphQL schema for the usage tab of automation playbooks. Copy of Usage Response
4. Reviewed the concepts of pagination for further utilization.

**LEARNINGS :**

1. Gaurav had resolved the issue of forbidden error, I realised it was cause of not adding a user context token to the lanyard authentication- so I edited the onboarding docs to prevent further confusion for other interns.
2. I reviewed the ERS schema and the playbook schema and drafted a rough one to understand the requirements for the Usage Tab.
3. I learnt the concepts of pagination to implement pagination for further GraphQL queries.

**May 29, 2025**

**TASKS :**

1. Completed FedRAMP - Change Management and Declaration of Atlassian Asset Possession Absorb Course.
2. Worked on refining the GraphQL schema with the frontend team to understand their requirements and what fields should be exactly implemented.
3. Went through the codebase to figure out the exact structure of implementation of Usage Tab code. https://hello.atlassian.net/wiki/spaces/~7120207183aad7e69a4168a399750eb8a0d514/pages/5363453890 Can't find link

**LEARNINGS :**

1. I showcased the GraphQL schema to Gaurav in our daily sync and to further refine it and align ourselves with the frontend I scheduled a sync with Vinit  and Snithik . We discussed about the possible requirements and the fields that should be implemented exactly.
2. To reach a consensus regarding the Usage Tab , we needed product design's team input on this  to finalise all the details.
3. This outlined the cross-team collaboration that is required from the start to develop something from scratch and was an insight into how big companies take the time to understand user requirements and plan each step before implementing. This was a way more detailed and methodical than what I was used to , so was a good learning point for me.
4. I read through and grasped knowledge about the flow and exact structure of the repository and targeted folders that needed to be modified for usage tab.

**May 30, 2025**

**TASKS :**

1. Discussed relevant docs required to query the DB as per our requirement [Analysis] Searching and Sorting in ERS https://developer.atlassian.com/platform/entity-relationship-store/node/model/#filters
2. Wrote code for the data transfer objects related with the usage tab and looked into implementation of pagination in it.

**LEARNINGS :**

1. Gained insight into the abstracted database and learnt that we cant directly query from it and ERS helps in abstracting it.  I learned to use its attributes (like Node) and filters to fetch data in the required format. The

documentation clarified how to use property filters  and how sorting works.

2. While working on the DTOs for the usage tab, I learned the importance of structuring them to match the data requirements for both backend and frontend. This involved understanding the schema and ensuring the DTOs facilitate smooth data transfer between layers.
3. I implemented pagination methods in the interface for the usage tab, which involved not just slicing data but also ensuring the backend supports efficient and secure data retrieval.

### Jun 2, 2025 - WEEK 3

**TASKS :**

1. Wrote the code for the interface and service implementation layer in regards to the usage tab.
2. Also wrote code for the payload part allowing it a connection to the repository.
3. Working on integration testing of the code.
4. Working on the aggregate query for playbooks.

**LEARNINGS :**

1. I learned how the service layer interacts with the repository to fetch and process data, and how to design methods that are both reusable and testable.
2. This process also highlighted the importance of aligning the service layer with the requirements from both frontend and product teams, ensuring the data returned matches what is needed for the usage tab.
3. I learned how to structure payload objects (DTOs) to facilitate smooth data transfer between the service and repository layers.
4. I gained practical experience in writing integration tests to validate the end-to-end functionality of the code.

### Jun 3, 2025

**TASKS :**

1. Completed Accessibility Fundamentals Absorb Course
2. Wrote the code for integration testing , but its failing due to dependency and localhost issues-need to get it resolved.
3. Merged my service and implementation layer package to the existing StepInstanceExecution one
4. Drafted the documentation of feasibility of certain criterias in the usage tabs. Feasibility -Usage Tab

**LEARNINGS :**

1. Till now I had made separate files for the usage tab code but by merging my service and implementation layer package into the existing StepInstanceExecution module, I learned how to integrate new features into a larger codebase. This process reinforced the importance of maintaining code consistency, understanding existing architecture, and ensuring that new code aligns with established patterns and practices.
2. While documenting the feasibility of various criteria for the usage tab, I learned how to assess technical requirements against platform limitations.
3. This also proved to be a very thought intensive exercise since I had to figure out how feasible each criteria is based on the exisiting codebase and the indexes available to us.

**CHALLENGES :**

1. I tried running and debugging the integration test code but it had a lot of dependency issues and wasnt connecting to localhost but this taught me the need for a stable local environment and proper dependency management.

These issues also emphasized the value of troubleshooting and collaborating with others to resolve blockers.

### Jun 4, 2025

**TASKS :**

1. Discussed the feasibility criterias with Harshita and looked into the implementation of these criterias especially for aggregative queries.
2. Worked on the rough code of implementation of these criterias.

**LEARNINGS :**

1. Since theres no aggregative (groupBy) property directly in the ERS Schema, we need to utilize indexes and hashValues to fetch all the data in a list and then take its size to find the count and then map the count and the list in a refernce object to perform ordering operations on it.
2. While some doubts were resolved during discussions with Harshita, I realized the importance of consulting with the product design team to finalize the approach and ensure alignment with the intended user experience and UI design.
3. Working on the rough code gave me hands-on experience in translating requirements into implementation, especially in handling data aggregation, pagination, and mapping results to response objects for the frontend.
4. An ERS HOT was raised regarding the pipelines for local testing having failed, so this was a potential cause for my integration tests failing , but this failed testing helped me in understanding that I need to check all the dependencies and how Docker networking works. It enabled me to carefully read the error logs to identify what is causing the failure.

### Jun 5, 2025

**TASKS :**

1. Got the rough code of some of the feasibility criterias reviewed by Harshita and discussed some of my doubts. Feasibility -Usage Tab
2. Resolved some doubts about the schema of the usage tab, with Vinit by scheduling a sync, posted my queries in his confluence page for the product team regarding the design and implementation of certain criterias.
3. Started working on the GraphQL schema for the Usage Tab.

**LEARNINGS :**

1. Since the schema is limited, selecting the correct index is crucial for efficient data retrieval. After fetching the data, I apply the necessary operations (like deduplication or mapping) in code to fulfill the requirements of each criterion.
2. Getting my rough code for some feasibility criteria reviewed by Harshita helped me identify areas for improvement and validated my approach and helped me discuss more of my doubts and catch issues early.
3. Scheduling a sync with Vinit and posting queries highlighted to me the benefit of being proactive and owning my work, reaching out with clear communication enabled me to grow at a better pace and helps in reaching my deadlines.
4. Me and Harshita have pondered and decided utilising a subsection of an index - if feasible would be appropriate for fetching all the criterias then applying the correct operations and queries on the data would be the best way of implementation.

## Jun 6, 2025

**TASKS :**

1. Finished the GraphQL Schema for the usage tab. GraphQL Schema for Playbook Usage Service

**LEARNINGS :**

1. Since a subset of the hash properties cannot be utilised ,the index needed to be carefully selected for efficient data retrieval.The chosen index was the same as the one used for playbook execution logs, ensuring consistency and leveraging existing data structures.
2. Collaborating with the frontend team and product design was essential to finalize the schema fields and ensure the API would meet UI and user requirements. This cross-team collaboration highlighted the importance of clear communication and iterative feedback in schema design.
3. The schema supports multiple filtering options (monthly, step status, playbook name) and implements pagination, which is crucial for handling large datasets and providing a responsive UI experience.
4. The controller checks for project existence and validates that the user has the necessary admin permissions before returning data, ensuring that sensitive usage insights are only accessible to authorized users.
5. Due to ERS limitations, some aggregation and filtering operations are performed in memory after fetching the data, rather than directly in the database query. This required careful handling to maintain performance and accuracy

## Jun 9, 2025 - WEEK 4

**TASKS :**

1. Got the GraphQL Schema reviewed by Harshita and Gaurav and discussed some of the implementation aspects
2. Got the git pull issue resolved
3. Made a controller object in the code for usage tab.

**LEARNINGS :**

1. Gaurav and Harshita have helped validate my thought-processes, approaches and to also catch potential issues and doubts.Being open to feedback and communicating clearly has benefited me throughout this internship.
2. I learnt to double-check all the setup cause my SSH keys config file wasnt added which was causing problems in applying git commands-resolved this by going through and throughly reviewing the concerned documentation.
3. I understood the separation of packages -the controllers focusing on request handling and the service classes handling business logic.This improves code maintainability and testability.

**CHALLENGES :**

1. I still am not able to perform testing locally due to a spring bean error and VM options error.Have been trying to resolve this issue.

## Jun 10, 2025

**TASKS :**

1. Worked on refining the controller code discussing the argument requirements with Snithik and how the general code is gonna flow.
2. Committed my controller code and put it up for reviewing.
3. Attended Phantom Sprint Planning.
4. Updated the GraphQL schema documentation.

5. Made the service interface layer and the outline of the service implementation layer.
6. Working on the repository method since this and the logic in the implementation layer go hand in hand.

### Jun 11, 2025

**TASKS :**

1. Finished the repository method for usage tab.
2. Had a sync with Gaurav where we reviewed the controller code.
3. Attached the graphql schema to the controller query and added the graphqlst content to the schema doc.
4. Had a sync with Sandeep and reviewed the progress and performance so far.
5. Tried resolving the integration testing issue that I'm facing with Harshita and Gaurav.
6. Uploaded my Atlassian value linkedin post for intern bingo challenge - "Through My Eyes": 4 Weeks at Atlassian 😄 | Sindhu Veerathu

### Jun 12, 2025

**TASKS :**

1. Updated the GraphQL Schema after Snehals review.
2. Discussing whether a new ARI is required for playbook usage response object.- talked with Harshita regarding this and we are thinking of utilising "playbook" ARI since it has cloudId, activationId and playbookId in it.
3. Had a sync with Vinit regarding the schema.- needs some additonal changes
4. Harshita and I discussed about the filters that need to be implemented in the repository, how pagination will work in usage tab and the map that I have to make for unique issues and unique agents.

### Jun 13, 2025

**TASKS :**

1. Added the filters in the repository code.
2. Added the validators and permission checkers in the controller code.
3. Made a new paginated response file for the usage tab for returning results from repository.
4. Finished the service impl code and tested it locally using mock data -since my debugger wasnt working.
5. Added a mapper function in the service layer implementation code.
6. Had a sync with Harshita regarding the ARI to be implemented in the schema - maybe need a new one.
7. Worked with other interns on designing posters for change activity for Atlassian Foundation. - link if you want to see :)

**CHALLENGES :**

1. Having trouble understanding if ARI is required and how to implement it.

### Jun 16, 2025 - WEEK 5

**TASKS :**

1. Completed the Spring framework Linkedin Course CertificateOfCompletion_Spring Framework in Depth.pdf
2. Resolved my doubts about ARI with Gaurav and got some of my serviceimpl code reviewed, talked about the next steps like pagination on the filtered data, mapper functions etc.

3. Edited the Atlassian resource identifier repository and made changes to add the playbook-step ARI.
4. Documented the new playbook-step ARI LDR: New ARI for Playbook Step
5. With the help of Harshita raised a pr for the new ARI-waiting for its approval.
6. Finalised some of the schema changes along with Snithik.

**LEARNINGS :**

1. Mehmet,Vinit and Gaurav agreed on making a new ARI called playbook-step which would help uniquely identify each step of a playbook in a log. This would be implemented in the Usage Tab of Playbooks.
2. With Harshita's help I learnt new git commands like git stash and fixed build failures in the pr-made me realise to review my code properly before raising a pull request.
3. Learnt that the owner for ownerAccountId is hydrated from external templates and contains many details like name and picture and is abstracted-this was a doubt that we had encountered about how to send picture.

**Jun 17, 2025**

**TASKS :**

1. Resolved the debugger issue in intellij locally-added a byte buddy configuration in debugger
2. Raised a draft pr for service implementation layer for Gaurav to review it.
3. Raised and merged pr for activation ids and workspace ARI. https://bitbucket.org/jira-service-management/automation-playbook/pull-requests/422 Connect your account
4. Raised a pr for controller code and its dependencies- need to be approved.
5. Resolved issues in the graphql schema considering notations.

**LEARNINGS :**

1. How I resolved the issues I have been facing with running tests locally.
   - Had to clear docker of any additional images, containers
   - .Resolve all the dependencies and maven checkstyle errors for running the sandbox.
   - To resolve micros dependency issues had to login into micros using atlas micros login.
   - Added a Byte Buddy VM option in the debugger configuration to overcome the exception.
2. Learnt how to raise a pr for previously committed specific files. This helps in maintaining a clean code which is easy to review and approve for the pr and also helps in thorough understanding of the code. For this I had to make a new branch for each piece of code that I wanted to raise a pr for, then add those files in the new branch from my spike branch and commit them and raise a pr.

**Jun 18, 2025**

**TASKS :**

1. Merged the pr for new ARI for Playbook Step-after approval from Gaurav,Vinit,Snehal and two codeowners of the repository. https://bitbucket.org/%7B02b941e3-cfaa-40f9-9a58-cec53e20bdc3%7D/%7Bef03708c-238d-43f9-b01b-cb6d3f180cf7%7D/pull-requests/813/overview Connect your account
2. Resolved comments and added commits in the pr of controller layer. https://bitbucket.org/jira-service-management/automation-playbook/pull-requests/424/overview Connect your account
3. Raised a pr for the graphql schema changes-resolved all the comments Vinit made. https://bitbucket.org/jira-service-management/automation-playbook/pull-requests/428/overview Connect your account

4. Worked on implementing pagination on top of the sorted results in the service implementation layer.
5. Worked on applying filters on the fetched data from the repository.

### Jun 19, 2025

**TASKS :**

1. Tried raising a pr in AGG for the new playbook usage grapql schema- faced build errors https://bitbucket.org/atlassian/graphql-central-schema/pull-requests/12062 Connect your account
2. Applied additional filters of playbook state, calculated successful and failed steps and worked more on mapper implementation.
3. Won the Goodie prize for the first row of Intern Bingo Challenge.
4. Raised a pr for the service and database layer code.

### Jun 20, 2025

**TASKS :**

1. Working on resolving the issues with AGG PR.
2. Restructing the service impl layer- to convert step aggregates to a method and add mappers for conversion of response object.
3. The new ARI will be available for use from next week- **I cannot see an ARI which should exist in this repository. What do I do?**

   Try to figure out who the resource owner of the ARI is, and *kindly* ask them to add their ARI to the ARI Registry YAML 😄

   We auto-generate classes based on what exists in the registry. Teams should be reflecting their ARI back into it, using the process defined here.

   If your ARI has recently been added to the registry, you will have to wait until the start of the next business week (AEST) for the changes to be reflected in this library. https://bitbucket.org/atlassian/ari-typescript/src/main/ Connect your account
4. Fixed the build failures in service layer pr, its ready for review. https://bitbucket.org/jira-service-management/automation-playbook/pull-requests/434/overview Connect your account
5. The controller layer pr has been approved.
6. Have to wait for ARI to be enabled to test out the mapper implementation and overall testing of the code.

**CHALLENGES :**

1. Having trouble fixing the build failures of the agg pr for a graphql schema for usage tab- the errors are stating default hydration,routing and ati errors. I tried editing the graphql pr to add default hydration but that cause build failures. Figured out that these build failures were occuring due to some incompatibility with versions, but not able to resolve it as of yet.

### Jun 23, 2025 - WEEK 6

**TASKS :**

1. Worked on updating the service implementation layer - simplifying the logic.

2. Worked on testing the logic and flow of the backend code through integration tests - only blocker is that since jiraPlaybookStep ARI is still not in action, I cant properly test the code till the full extent.
3. Working on fixing the code flow through debugging and making necessary changes
4. Worked on adding a filter utils file for a few filters in the repository code.
5. Communicated with Neel and Snithik regarding the intern project- Snithik about the progress of work in the frontend portion and other specifities and Neel on the progress on the agg pr since him and Snehal are actively working on that.

**Jun 24, 2025**

**TASKS :**

1. Added ownerAccountId in the results of jiraPlaybookStepUsage
2. Resolved the agg pr errors and modified code of playbook schema pr too - First rebased the new ari version changes from master and then added default hydration, polymorphic hydration, routing and entity tests to my branch of the central schema and then raised an agg pr through pipeline.
3. Playbook Schema pr is approved and merged. https://bitbucket.org/jira-service-management/automation-playbook/pull-requests/428 Connect your account

**Jun 25, 2025**

**TASKS :**

1. Merged controller layer pr. https://bitbucket.org/jira-service-management/automation-playbook/pull-requests/424 Connect your account
2. Added the new jiraPlaybookStepARI in service layer code to generate cursor.
3. Working on local debugging and testing of code properly since the related ARI is available now.
4. Resolved AGG pr build errors - added ari directive - pr is approved and merged https://bitbucket.org/atlassian/graphql-central-schema/pull-requests/12188/overview Connect your account

**Jun 26, 2025**

**TASKS :**

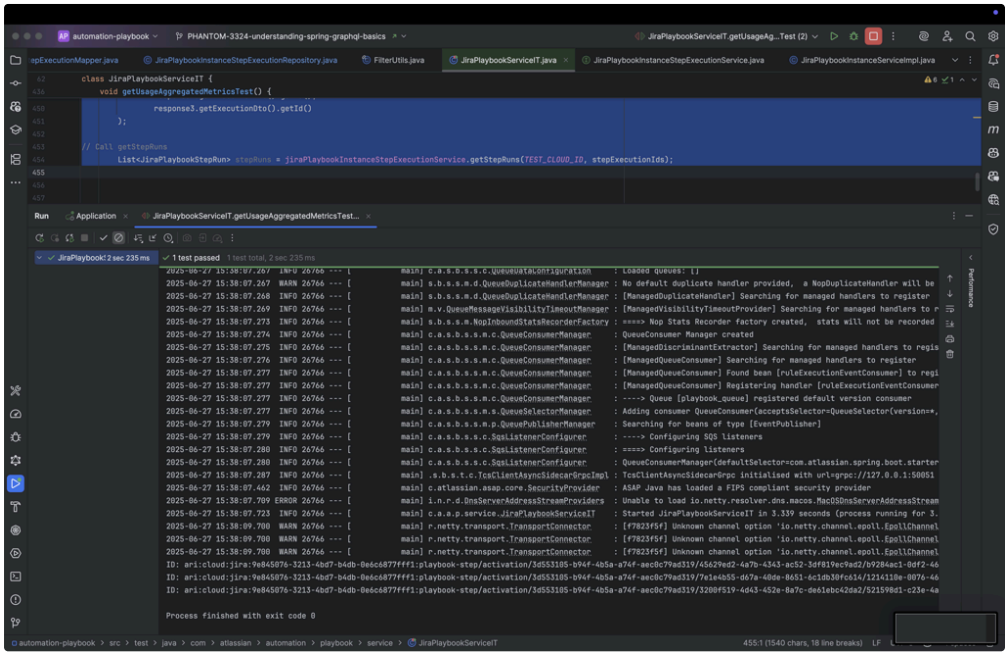
1. Faced some errors and inconsistencies being reflected on Neel's AGG pr , so had a sync and worked on resolving and syncing automation playbook and AGG pr. https://bitbucket.org/jira-service-management/automation-playbook/pull-requests/457 Connect your account
2. Synced with Harshita to resolve some issues I found during local testing.
3. Worked on locally testing pagination and filters in the code.

**Jun 27, 2025**

**TASKS :**

1. Worked on locally testing the code- the code is reflecting changes ie calculating the metrics.
2. Worked on testing the pagination and filtering of the metrics - pagination is working by page limit ,like if there are 3 step executions and the limit is 2, the console is reflecting 2, but for the next iteration with the next page cursor I'm having trouble getting the last execution to reflect on the console.

3. Have to work on reducing the complexity of the service implementation code.- regarding filtering by state of playbook and the fetching of ownerAccountId details.



## Jul 1, 2025 - WEEK 7

**TASKS :**

1. Resolved comments on service layer pr.
2. Discussed the issue with owner id being hydrated or not for frontend with Snithik
3. Worked on debugging and resolving issues with pagination.

## Jul 2, 2025

**TASKS :**

1. Resolved issues regarding pagination during testing.
2. Tested out filters in the code whether they are working properly.

## Jul 3, 2025

**TASKS :**

1. Added null checks in code.
2. Made the code more readable by encapsulating it in clear methods and writing multiple test cases with assert statements.
3. Fixed the doubt I had with Harshita -why the instructional rule eventhough being executed multiple times returned a count of 1- made the feature gate constant for this true so it can count both success and success_undone counts.
4. Reached out to product regarding the success step counts -how we should count it and reflect on the table.
5. Tried raising a pr - but a previous test was giving a false return so there was a pipeline error in my pr - talked to the owner of the test regarding it.

**Jul 4, 2025**

**TASKS :**

1. I made the code more clean,readable and modular- separating it into different methods,files etc.
2. Consulted with product regarding the parameters of the successful step counts for instructional rules and gained clarity.
3. Updated the pr with all the commits.

**Jul 7, 2025 - WEEK 8 !!**

**TASKS :**

1. Deployed my branch to staging to figure out any errors.
2. Faced errors regarding page limit-resolved it.
3. Faced errors regarding filter application-in the process of resolving it
4. Worked on final project presentation.

## Feasibility -Usage Tab

Monthly filter, step status filter and playbook name filter is there

Across step level display order is based on execution count.

Across the playbooks level- not decided

Add failed and successful step counts after execution count ( 2 extra columns)

Top 5 agents and issues-maybe

Doubts:

1. If a rule fails or is aborted, in some cases,steps may not complete, leading to missing stepEndTime or duration which can cause a null pointer error so maybe we need to filter out the rules where there is a missing stepEndTime.
2. Or in some cases when a step fails it still gives back a duration so should it also be involved in the calculation of Average Execution time or should we omit the failed steps ? - we are calculating the execution times even taking failed steps into consideration
3. Are we just keeping a track of the uniqueIssue and uniqueAgent count or even the names or ids and if yes, to what extent are we going to display it to the user - might require to be paginated if the extent is big. - future scope
4. Are we going to allow all the enums of StepStatus to be displayed or will we limit it to just Success or Failure? (success, failed, no actions performed, some errors)

**DEMO VIEW**

| Playbook Name | Owner | Step Name | Step Type | Count | Unique Agents | Unique Issue | Duration (Min,Max,Avg) |
|---|---|---|---|---|---|---|---|
| Playbook 1 | EFC | Step 1.2 | Instructional | 24 | 3 | 5 | - |
| Playbook 2 | ABC | Step 2.1 | Automation | 16 | 2 | 2 | 10 mins |
| Playbook 1 | EFC | Step 1.1 | Instructional | 12 | 5 | 3 | - |

**CRITERION**

**Unique Agent - keeping a count on the number of unique users/agents running a particular step of the playbook**

1. "triggeredByAccountId": {
   "type": "string",
   "description": "Identifier of the user who ran the step",
   "required": true,
   "allowedValues": [],
   "classification": []
   },- this field would be used to fetch the unique agents of each step
2. It is feasible to use the triggeredByAccountId field to track the number of unique users/agents running a particular step of the playbook.
3. Use the ERS API to fetch all jira-playbook-instance-step-execution records for a given stepId and playbookId.
4. Retrieve all matching record's triggeredByAccountId.
5. Aggregate the results in the service layer to count distinct triggeredByAccountId values.
6. No schema change is needed. Only need to implement the appropriate query or aggregation logic in the service/repository layer to count unique triggeredByAccountId values per step.

Complexities-

If there are many unique users executing a step, may need to paginate through large result sets to collect all triggeredByAccountId values.

```
final QueryRequest<JiraPlaybookInstanceStepExecutionResource> queryRequest =
QueryRequest.forClass(JiraPlaybookInstanceStepExecutionResource.class)
    .usingIndex(JiraPlaybookInstanceStepExecutionResource.INDEX_CLOUD_ID_PLAYBOOK_ID_STEP_ID)
    .withHash(HashValue.ofMultiPropertyHash(List.of(
        cloudId,
        playbookId,
```

```
        stepId
    )))
    .pageLimit(request.getPageLimit());

List<JiraPlaybookInstanceStepExecutionResource> executions =
jiraPlaybookInstanceStepExecutionRepository.getUsageByInstance(request);

Set<String> uniqueAgents = new HashSet<>();
for (JiraPlaybookInstanceStepExecutionResource execution : executions) {
    if (execution.getTriggeredByAccountId() != null) {
        uniqueAgents.add(execution.getTriggeredByAccountId());
    }
}
int uniqueAgentCount = uniqueAgents.size();

public class StepAgentCountDto { // Response Object(in playbook execution) for all the unique agent count
    private String stepId;
    private int uniqueAgentCount;
  }

List<StepAgentCountDto> response = stepIdToUniqueAgentCount.entrySet().stream()
    .map(entry -> new StepAgentCountDto(entry.getKey(), entry.getValue()))
    .toList();

then send this upto frontend.
```

**Unique Issue - keeping a count on the number of unique issues which ran a particular step of the playbook**

1. Utilizing “contextId” for this since contextId is a required string property that represents the unique identifier associated with the context of the step execution.
2. This is feasible as ERS supports indexed queries and pagination, can fetch only the required fields and counting can be done with a simple aggregation step.
3. Use the ERS search or query API to filter records by playbookId, stepId, and context: ISSUE.
4. Retrieve all matching records' contextId fields.
5. Deduplicate contextId values in your application code to count unique issues.
6. Aggregate the results in the service layer to count distinct contextId values.

Complexities-

If there are many issues which executed a step of a playbook, may need to paginate through large result sets to collect all contextId values.

```

final QueryRequest<JiraPlaybookInstanceStepExecutionResource> queryRequest =
QueryRequest.forClass(JiraPlaybookInstanceStepExecutionResource.class)
    .usingIndex("by-cloudId-activationId-scopeType-scopeId-context-contextId-stepStartTime")
    .withHash(HashValue.ofMultiPropertyHash(List.of(
        cloudId,
        activationId,
        scopeType,
        scopeId,
        "ISSUE",     // this is using context for filtering -doubtful if this should be used
        contextId
    )))
    .pageLimit(request.getPageLimit());

List<JiraPlaybookInstanceStepExecutionResource> executions =
jiraPlaybookInstanceStepExecutionRepository.getUsageByInstance(queryRequest);

Set<String> uniqueIssues = new HashSet<>();
for (JiraPlaybookInstanceStepExecutionResource execution : executions) {
    if (execution.getContextId() != null) {
        uniqueIssues.add(execution.getContextId());
    }
}

int uniqueIssueCount = uniqueIssues.size();


public class StepIssueCountDto { // Response Object(in playbook execution) for all the unique issue count
    private String stepId;
    private int uniqueIssueCount;
}

List<StepIssueCountDto> response = stepIdTouniqueIssueCount.entrySet().stream()
    .map(entry -> new StepIssueCountDto(entry.getKey(), entry.getValue()))
    .toList();

then send this upto frontend.
```

**Average Execution Time - calculate the average execution time of an automation step(not of instructional since it's just marked as completed)**

1. This is feasible since the ERS schema has all the required fields and can filter and group them accordingly for the calculation.These indexed fields are well suited for querying.
2. Query step execution records filtered by cloudId, projectId, and step type: Automation Rule
3. Group results by stepId.
4. Extract the duration field from each record.
5. Sum all durations.
6. Divide by the number of execution counts to calculate the average time.

Complexities-

There are both step start and end time and rule start and end time

If a rule fails or is aborted, in some cases,steps may not complete, leading to missing stepEndTime or duration which can cause a null pointer error so maybe need to filter out the rules where duration is null.

In a failed rule the duration is specified so should we take that into consideration while calculating the average execution time.

**Step Usage within Playbooks(Most and Least)**

? Step usage within playbooks (Most-least) For this one are we proposing to order all the steps in the playbook like using a sort key based on the execution count of each

? Or are we going to display on the UI like only the most and least executed ones and other all having a null field

We group by playbook Id , then step id and then order by execution count (descending order).

Need to discuss the display issues.

**Step Status breakdown - display whether the particular step was a success or a failure.**

1. This is feasible since all the enabled states are present in the ERS schema under "stepStatus", so just need to fetch it and map it to the displayable options.
2. For the cases of Success and No actions performed it will be categorized under the state of Success
3. For the cases of Some errors, Failure, Throttled, Loop and Aborted it will be categorized under the state of Failed.
4. When fetching step execution data, map each execution status to either Success or Failed.

| Data | Step | Complexity |
|---|---|---|
| Unique Agent (AccountId) | Execution level data | Low to Medium Complexity (needs pagination for scalability) |
| Execution time (Duration) | Average, Max and Min Duration | Medium Complexity(needs to filter out null durations and need to distinguish between rule and step duration |
| Step usage within playbooks (Most-least) | NOT FEASIBLE | Depends upon the definition |
| Unique Issue | Execution level data | Low to Medium (needs pagination for scalability) |
| Breakdown on step status | Execution level data | Low Complexity |
| Breakdown on Playbook Name | Playbook level data | Low Complexity |
| Breakdown on Step Type | Execution level data | Low Complexity |
| Breakdown on DateTime range | Execution level data | Low Complexity |
| Breakdown on Playbook state | Playbook level data | Low to Medium ( since filtering on this isnt available in the instance resource so have to fetch playbooks through another DTO) |

# GraphQL Schema for Playbook Usage Service

| | |
|---|---|
| **Status** | MERGED |
| **Drivers** | @Sri Sindhu Veerathu |
| **Reviewers** | ☑ @Sandeep Ravichandran<br>☑ @Gaurav Arora<br>☐ @Uğur Turan<br>☐ @Harshita Goyal<br>☐ @Vinit Jain<br>☐ @Prabhav Sharma |
| **Informed** | ITOps Phantom - JSM Automation ( @Arka Dutta @Chandra Bhan Giri (Deactivated) @Debangshu Banerjee @Gaurav Arora @Harshita Goyal @Kumar Utkarsh @Nakul Mohan @Pawan Agrawal @Sandeep Ravichandran @Shreya Jain @Snehal Gupta @Supraja Alleni @Uğur Turan @Vinit Jain ) |

The purpose of this document is to outline the GraphQL schema design for managing the Usage Tab of Jira Playbooks, enabling usage insights into the playbook executions.

For Usage Service this is the **demo view**

| Playbook Name | Owner | Step Name | Step Type | Count | Unique Agents | Unique Issue | Duration (Min,Max,Avg) |
|---|---|---|---|---|---|---|---|
| Playbook 1 | EFC | Step 1.2 | Instructional | 24 | 3 | 5 | - |
| Playbook 2 | ABC | Step 2.1 | Automation | 16 | 2 | 2 | 10 mins |
| Playbook 1 | EFC | Step 1.1 | Instructional | 12 | 5 | 3 | - |

There are **filtering options** also available based on

- **monthly filter** from latest to 3 months back

-**step status filter**

-**playbook name filter.**

Explored the Node Model of the ERS https://developer.atlassian.com/platform/entity-relationship-store/node/model/#filters .

For playbook usage service, since the database of the repository is abstracted, we need to **utilize only the attributes present in the Entity Relationship Store.**

From the Node model, there is a limitation on the hashProperties for each index

| | |
|---|---|
| Limitations/Constraints | `hashValue` must have the same number of values as the index's hash key. Each |

| | |
|---|---|
| | value must be of the same type as the index's corresponding hash property. Hash values must be supplied in the same order as the hash properties on the index. |

**Utilising a subset of the HashProperties is not possible** due to this limitation.

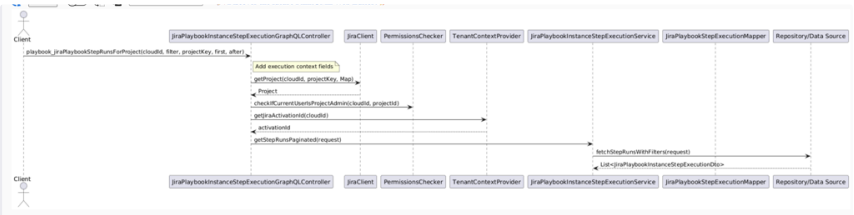
The best suited index for this is the same **index we use for the playbooks execution logs tab** which is

```
{
      "name": "by-cloudId-activationId-scopeType-scopeId-stepStartTime",
      "hashProperties": [
        "cloudId",
        "activationId"
      ],
      "rangeProperties": [
        "scopeType",
        "scopeId",
        "stepStartTime"
      ]
    }
```

After fetching the data through this index we can do aggregation and other **operations in memory** and also apply filters which are additional through **filtering options**.

This is the **data flow**

**Request → Controller → Service → Mapper → Repository → Service → Mapper → Controller→ Response**




## ARI

A new ARI is required for the Usage Tab which allows each log to be uniquely identified. The existing ARIs which have stepId for differentiation of each log are also dependent on contextId

So after deliberation, created a new ARI called **playbook-step**

```
ari:cloud:jira:{siteId}:playbook-step/activation/{activationId}/{playbookId}/{stepId}
```

This ARI is used to uniquely identify and reference a specific step within a playbook in Jira.

LDR: New ARI for Playbook Step

## SCHEMA- jira-playbook-usage.graphqlst

```
 type Query { // in jira-playbook-query-mutation.graphqlst
  """
  This will be used in Usage Tab in Admin view
  """
  playbook_jiraPlaybookUsageForProject(
    cloudId: ID!  @CloudID(owner: "jira") # Cloud ID of the Jira instance
    filter: JiraPlaybookUsageFilter,      # Filters to apply to the usage query
```

```
    sort: [JiraPlaybooksSortInput!] = [{by :totalExecutionCount , order : DESC}], # Sorting options, by execution counts
    projectKey: String!,                  # Key of the Jira project to fetch usage for
    first: Int = 20,                      # Number of results to return (pagination)
    after: String                         # Cursor for pagination
  ): JiraPlaybookUsageConnection          # Returns a paginated connection of playbook usage
    @lifecycle(name: "PlaybooksInJSM", stage: EXPERIMENTAL, allowThirdParties: false) # Metadata for API lifecycle
}
```

```
"""
Connection for paginated playbook step usage
"""
type JiraPlaybookStepUsageConnection implements QueryPayload & HasPageInfo {
  """
  List of edges containing step usage nodes and cursors
  """
  edges: [JiraPlaybookStepUsageEdge!]
  """
  Pagination information (hasNextPage, endCursor, etc.)
  """
  pageInfo: PageInfo!
  """
  List of errors, if any
  """
  errors: [QueryError!]
  """
  List of step usage nodes (shortcut for edges.node)
  """
  nodes: [JiraPlaybookStepUsage]
  """
  Indicates if the query was successful
  """
  success: Boolean!
  """
  True if there are more pages after this one
  """
  hasNextPage: Boolean
  """
  True if there are pages before this one
  """
  hasPreviousPage: Boolean
  """
  Cursor for the next page
  """
  nextPageCursor: String
}

type JiraPlaybookStepUsageEdge {
  """
  Cursor for this edge (used for pagination)
  """
  cursor: String!
  """
  The playbook step usage data for this edge
  """
  node: JiraPlaybookStepUsage
}

"""
The response object for usage tab
"""
type JiraPlaybookStepUsage implements Node @defaultHydration(field: "playbook_jiraPlaybookStepUsages", idArgument: "ids", batchSize: 90) {
  """
  Unique identifier for the playbook step
```

```
  """
  id: ID! @ARI(type: "playbook-step", owner: "jira")
  """
  Name of the playbook
  """
  playbookName: String
  """
  Account ID of the playbook owner
  """
  ownerAccountId: String @templateHiddenFromOverall
  owner: User @hydrated(service: "identity", field: "users", arguments: [{name : "accountIds", value : "$source.ownerAccountId"}], identifiedBy: "id", batchSize: 50)
  """
  Name of the step
  """
  stepName: String
  """
  Type of the step (enum)
  """
  stepType: JiraPlaybookStepType
  """
  Average execution duration for this step/playbook
  """
  avgExecutionDuration: Long
  """
  Minimum execution duration
  """
  minExecutionDuration: Long
  """
  Maximum execution duration
  """
  maxExecutionDuration: Long
  """
  Number of unique agents who executed
  """
  uniqueAgentCount: Long
  """
  Number of unique issues involved
  """
  uniqueIssueCount: Long
  """
  Total number of executions
  """
  totalStepExecutionCount: Long
  """
  Count of successful executions
  """
  successfulStepExecutionCount: Long
  """
  Count of failed executions
  """
  failedStepExecutionCount: Long
}

"""
The request filters for usage tab
"""
input JiraPlaybookStepUsageFilter {
  """
  Filter by playbook name (exact match)
  """
  name: String
  """
  Filter by playbook state (enum)
  """
  state: JiraPlaybookStateField
  """
  Filter for executions after this date-time
  """
  startTime: DateTime
```

```
  """
  Filter for executions before this date-time
  """
  endTime: DateTime
  """
  Filter by step type (enum)
  """
  stepType: JiraPlaybookStepType
  """
  Filter by step run status (enum)
  """
  stepStatus: [JiraPlaybookStepRunStatus!]
}

```

```
directive @defaultHydration( // in agg-shared-directives.graphqlst
  "The backing field for the data"
  field: String!
  "Name of the ID argument on the backing field"
  idArgument: String!
  "The name of the field in the result type used to match results to the input"
  identifiedBy: String! = "id"
  "The batch size"
  batchSize: Int
  "The timeout to use when completing hydration"
  timeout: Int! = -1
) on OBJECT | INTERFACE
```

---

## Request Layer

| Name | Definition |
|---|---|
| 1. FetchJiraPlaybooksStepUsageRequest.java | This DTO is used to pass query parameters from the API/controller layer to the service layer when requesting usage statistics or records for Jira playbooks. |
| 2. FetchJiraPlaybooksStepUsageFiltersRequest.java | This DTO encapsulates all the filter and sort parameters needed to query playbook usage data.<br>It Allows the backend to receive complex filter and sort criteria from the frontend or API clients in a structured way. |

## Request Layer Code

**1. FetchJiraPlaybooksStepUsageRequest.java**

```
public class FetchJiraPlaybooksStepUsageRequest {
    private String cloudId;
    private String activationId;
    private String scopeId;
    private String startTime;
    private String endTime;
    private String nextPageKey;
    private String previousPageKey;
    private int pageLimit;
}
```

**2. FetchJiraPlaybooksStepUsageFiltersRequest.java**

```
public class FetchJiraPlaybooksStepUsageFiltersRequest extends FetchJiraPlaybooksStepUsageRequest {
    private String playbookNameFilter;
```

```
    private JiraPlaybookStateField state;
    private String stepTypeFilter;
    private JiraPlaybookScopeType scopeType;
    private List<JiraPlaybookStepRunStatus> stepStatus;
    private String sortBy;
    private String sortOrder;
}
```

---

## Controller Layer

| Name | Definition |
| --- | --- |
| 1. JiraPlaybookStepUsage.java | This is a response or payload object (DTO) for representing usage statistics and details about playbook step executions in Jira.<br><br>It includes fields for playbook and step identification, user and context information, timestamps, status, and various usage metrics such as average execution duration, unique agent count, unique issue count, and total execution count. |
| 2. JiraPlaybookStepUsageConnection.java | This is a response or payload class for paginated usage data about Jira playbook step executions.<br><br>It contains a cursor and node for pagination and pagination metadata. It comprises of error reporting and a success flag for queries. |
| 3. JiraPlaybookStepUsageGraphQLController.java | It is a controller that exposes API endpoints (query mappings) for interacting with Jira playbook step executions. It acts as the entry point for frontend  to trigger step executions, fetch step run logs, and retrieve step run details.<br><br>In our case, we would utilise a new method for returning aggregated usage statistics for all steps in a specific playbook in a project.<br><br>These new queries would call service methods that aggregate and return usage data, which can then be displayed in the usage tab for analytics and reporting. |
| 4. JiraPlaybookStepUsageFilter.java | It is a controller responsible for handling GraphQL queries related to Jira playbook usage, specifically for the "usage" tab in your application's UI.<br><br>It accepts filters, pagination, and project details, validates project existence and permissions, builds a request with the provided parameters, delegates data fetching to the service layer, and returns a connection object containing the usage data and pagination information. |

| | |
|---|---|
| 5. PlaybookValidator.java | Helps validate the playbook input data.<br>Added a new method to help validate the date range i.e to not allow null values and also the end date being earlier than the start date.<br>This is utilized in JiraPlaybookStepUsageGraphQLController.java |

## Controller Layer Code

1. JiraPlaybookStepUsage.java

```
public class JiraPlaybookStepUsage implements Node {
    /**
     * Unique identifier for the step usage.
     */
    private String id;
    private String playbookName;
    /**
     * Account ID of the owner of the playbook.
     */
    private String ownerAccountId;
    private String stepName;
    private JiraPlaybookStepType stepType;
    /**
     * Average execution duration of the step (in milliseconds).
     */
    private Long avgExecutionDuration;
    /**
     * Minimum execution duration of the step (in milliseconds).
     */
    private Long minExecutionDuration;
    /**
     * Maximum execution duration of the step (in milliseconds).
     */
    private Long maxExecutionDuration;
    /**
     * Number of unique agents that executed this step.
     */
    private Long uniqueAgentCount;
    /**
     * Number of unique issues this step was executed on.
     */
    private Long uniqueIssueCount;
    /**
     * Total number of times this step was executed.
     */
    private Long totalStepExecutionCount;
    /**
     * Number of successful executions of this step.
     */
    private Long successfulStepExecutionCount;
    private Long failedStepExecutionCount;
}
```

2. JiraPlaybookStepUsageConnection.java

```
public class JiraPlaybookUsageConnection implements QueryPayload
{
    private List<JiraPlaybookUsageEdge> edges;
    private PageInfo pageInfo;
    private List<GraphQLError> errors;
    private List<JiraPlaybookUsage> nodes;
    private boolean success;
    private Boolean hasNextPage;
    private Boolean hasPreviousPage;
    private String nextPageCursor;
```

```
10
11      @Override
12      public boolean success() {
13          return success;
14      }
15
16      @Override
17      public List<GraphQLError> errors() {
18          return errors == null ? List.of() : errors;
19      }
20
21      @Data
22      @Builder
23      public static class JiraPlaybookUsageEdge {
24          private String cursor;
25          private JiraPlaybookUsage node;
26      }
27  }
```

3. JiraPlaybookStepUsageGraphQLController.java

```
1
2   @Controller
3   public class JiraPlaybookStepUsageGraphQLController {
4       private static final Logger log =
    LoggerFactory.getLogger(JiraPlaybookStepUsageGraphQLController.cla
    ss);
5       public static final int GET_PLAYBOOKS_USAGE_MAXIMUM_LIMIT =
    100;
6       private final JiraClient jiraClient;
7       private final TenantContextProvider tenantContextProvider;
8       private final PermissionsChecker permissionsChecker;
9
10      @Autowired
11      private JiraPlaybookUsageService jiraPlaybookUsageService;
12
13      @Autowired
14      public JiraPlaybookStepUsageGraphQLController(JiraClient
    jiraClient,
15
    TenantContextProvider tenantContextProvider,
16
    PermissionsChecker permissionsChecker) {
17          this.jiraClient = jiraClient;
18          this.tenantContextProvider = tenantContextProvider;
19          this.permissionsChecker = permissionsChecker;
20      }
21
22      @QueryMapping(name =
    "playbook_jiraPlaybookStepUsageForProject")
23      public JiraPlaybookStepUsageConnection
    getStepUsageForProject(@Argument String cloudId,
24
    @Argument Optional<JiraPlaybookStepUsageFilter> filters,
25
    @Argument String projectKey,
26
    @Argument Optional<Integer> first,
27
    @Argument Optional<String> after) {
28
29
    ExecutionContext.addCustomField(ExecutionContextCustomFields.GRAPH
    QL_MAPPING, "playbook_jiraPlaybookStepUsageForProject");
30
    ExecutionContext.addCustomField(ExecutionContextCustomFields.PLAYB
    OOK_PROJECT_KEY, projectKey);
31
    ExecutionContext.addCustomField(ExecutionContextCustomFields.FIRST
    , String.valueOf(first.orElse(null)));
```

```
        Project project = Optional.ofNullable(jiraClient.getProject(cloudId, projectKey, Map.of()))
                .orElseThrow(() -> new PlaybookException("Project not found", Errors.JIRA_NOT_FOUND_ERROR));

        String projectId = project.getId();
        if (projectId.isEmpty()) {
            throw new PlaybookException("Project ID not found for projectKey: " + projectKey, Errors.JIRA_NOT_FOUND_ERROR);
        }

        permissionsChecker.checkIfCurrentUserIsProjectAdmin(cloudId, projectId);
        PlaybookValidator.validateOrDefaultDateRange(filters);
        String nameFilter = filters.map(JiraPlaybookStepUsageFilter::getName)
                .filter(name -> !name.isBlank())
                .orElse(null);

        Long stepStartTimeFilter = filters.map(JiraPlaybookStepUsageFilter::getStartTime)
                    .map(Instant::toEpochMilli)
                .orElse(null);
        Long stepEndTimeFilter = filters.map(JiraPlaybookStepUsageFilter::getEndTime)
                .map(Instant::toEpochMilli)
                .orElse(null);

        int pageLimit = Math.min(first.orElse(GET_PLAYBOOKS_USAGE_MAXIMUM_LIMIT), GET_PLAYBOOKS_USAGE_MAXIMUM_LIMIT);
        FetchJiraPlaybooksStepUsageFiltersRequest request = FetchJiraPlaybooksStepUsageFiltersRequest.builder()
                .cloudId(cloudId)
                .playbookNameFilter(nameFilter)
                .nextPageKey(after.orElse(null))

                .state(filters.map(JiraPlaybookStepUsageFilter::getState).orElse(null))
                .startTime(stepStartTimeFilter == null ? null : String.valueOf(stepStartTimeFilter))
                .endTime(stepEndTimeFilter == null ? null : String.valueOf(stepEndTimeFilter))

                .stepStatus(filters.map(JiraPlaybookStepUsageFilter::getStepStatus).orElse(null))

                .stepTypeFilter(filters.map(JiraPlaybookStepUsageFilter::getStepType)
                        .map(String::valueOf)
                        .orElse(null))
                .pageLimit(first.orElse(pageLimit))
                .scopeType(JiraPlaybookScopeType.PROJECT)
                .scopeId(project.getId())

                .activationId(tenantContextProvider.getJiraActivationId(cloudId))
                .build();
        JiraPlaybookStepUsageConnection connection = jiraPlaybookUsageService.getPlaybookUsagePaginated(request);

        if (connection.getNodes().isEmpty()) {
            log.info("No step usage found for project:{}", project.getId());
        }

        return connection;
    }
}
```

4. JiraPlaybookStepUsageFilter.java

```
public class JiraPlaybookStepUsageFilter {
    /**
     * Name of the playbook step to filter by.
     */
    private String name;
    /**
     * State of the playbook to filter by.
     */
    private JiraPlaybookStateField state;
    /**
     * Start time for filtering step usage (inclusive).
     */
    private Instant startTime;
    /**
     * End time for filtering step usage (inclusive).
     */
    private Instant endTime;
    /**
     * Type of the playbook step to filter by.
     */
    private JiraPlaybookStepType stepType;
    /**
     * Status of the playbook step run to filter by.
     */
    private List<JiraPlaybookStepRunStatus> stepStatus;
}
```

5. PlaybookValidator.java

```
public static void validateDateRange(String startTime, String endTime) throws PlaybookException {
        if (startTime == null || endTime == null) {
            throw new PlaybookException("Start time and end time cannot be null", Errors.INVALID_INPUT_ERROR);
        }
        try {
            Instant start = Instant.parse(startTime);
            Instant end = Instant.parse(endTime);
            if (start.isAfter(end)) {
                throw new PlaybookException("Start time cannot be after end time", Errors.INVALID_INPUT_ERROR);
            }
            long days = ChronoUnit.DAYS.between(start, end);
            if (days > MAX_ALLOWED_DAYS_RANGE_FOR_PLAYBOOK_STEP_USAGE) {
                throw new PlaybookException("Date range cannot exceed 31 days", Errors.INVALID_INPUT_ERROR);
            }
        } catch (DateTimeParseException e) {
            throw new PlaybookException("Invalid date format", Errors.INVALID_INPUT_ERROR);
        }
    }

    public static void validateOrDefaultDateRange(Optional<JiraPlaybookStepUsageFilter> filters) {
        if (filters.isPresent() && filters.get().getStartTime() != null && filters.get().getEndTime() != null) {
            PlaybookValidator.validateDateRange(
                String.valueOf(filters.get().getStartTime()),
                String.valueOf(filters.get().getEndTime())
            );
        } else {
            Instant endTime = Instant.now();
            Instant startTime = endTime.atZone(java.time.ZoneOffset.UTC).minusMonths(1).toInstant();
```

```
29
   PlaybookValidator.validateDateRange(startTime.toString(),
   endTime.toString());
30         }
31     }
```

## Service Layer

| Name | Definition |
| --- | --- |
| 1. JiraPlaybookUsageService.java | It fetches a paginated list of aggregated playbook usage data (such as execution counts, unique users, average durations, etc.) across multiple playbooks or steps, based on the filters and pagination options provided in the request.<br>Returns a JiraPlaybookUsageConnection, which contains usage data nodes, pagination info, and error handling for the UI. |

## Service Layer Code

1. JiraPlaybookUsageService.java

```
1 public interface JiraPlaybookUsageService {
2     JiraPlaybookUsageConnection
  getPlaybookUsagePaginated(FetchJiraPlaybooksUsageFiltersRequest
  request);
3 }
```

## Service Layer Implementation

| Name | Definition |
| --- | --- |
| 1. JiraPlaybookUsageServiceImpl.java | It fetches a paginated list of aggregated playbook usage data (such as execution counts, unique users, average durations, etc.) across multiple playbooks or steps, based on the filters and pagination options provided in the request.<br>Returns a JiraPlaybookUsageConnection, which contains usage data nodes, pagination info, and error handling for the UI. |
| 2. JiraPlaybookStepUsageAggregator.java | JiraPlaybookStepUsageAggregator.java is a utility class responsible for aggregating playbook step execution data in the automation playbook system. Its main function, aggregateStepUsages, takes a list of step execution resources and groups them by playbook and step. For each group, it calculates various statistics such as total executions, successful and failed counts, unique agents, unique issues, |

| | |
|---|---|
| | average/min/max execution durations, and more.<br><br>This is utilized in JiraPlaybookUsageServiceImpl.java |

## Service Layer Implementation Code

1. JiraPlaybookUsageServiceImpl.java -

```
@Service
public class JiraPlaybookUsageServiceImpl implements JiraPlaybookUsageService {
    private final JiraPlaybookInstanceStepExecutionRepository jiraPlaybookInstanceStepExecutionRepository;
    private final JiraPlaybookStepExecutionMapper jiraPlaybookStepExecutionMapper;
    private final JiraPlaybookService jiraPlaybookService;
    private final ARIUtils ariUtils;

    private static final Logger log = LoggerFactory.getLogger(JiraPlaybookUsageServiceImpl.class);

    @Autowired
    public JiraPlaybookUsageServiceImpl(JiraPlaybookInstanceStepExecutionRepository jiraPlaybookInstanceStepExecutionRepository,

JiraPlaybookStepExecutionMapper jiraPlaybookStepExecutionMapper,
                                        JiraPlaybookService jiraPlaybookService,
                                        ARIUtils ariUtils) {
        this.jiraPlaybookInstanceStepExecutionRepository = jiraPlaybookInstanceStepExecutionRepository;
        this.jiraPlaybookStepExecutionMapper = jiraPlaybookStepExecutionMapper;
        this.jiraPlaybookService = jiraPlaybookService;
        this.ariUtils = ariUtils;
    }

    @Override
    public JiraPlaybookStepUsageConnection getPlaybookUsagePaginated(FetchJiraPlaybooksStepUsageFiltersRequest request) {

        List<JiraPlaybookInstanceStepExecutionResource> filteredResults;
        List<JiraPlaybookInstanceStepExecutionResource> results = new ArrayList<>();
        String nextPageKey = null;
        String endCursor = null;
        int remainingItems;
        boolean hasNextPage;
        JiraPlaybooksStepExecutionPaginatedResponse currentResponse;

        do {
            FetchJiraPlaybooksStepUsageFiltersRequest currentRequest = FetchJiraPlaybooksStepUsageFiltersRequest.builder()
                    .cloudId(request.getCloudId())
                    .nextPageKey(nextPageKey)

.playbookNameFilter(request.getPlaybookNameFilter())
                    .pageLimit(request.getPageLimit())
                    .scopeType(request.getScopeType())
                    .scopeId(request.getScopeId())
                    .state(request.getState())
```

```
                    .startTime(request.getStartTime())
                    .endTime(request.getEndTime())
                    .stepStatus(request.getStepStatus())
                    .activationId(request.getActivationId())
                    .stepTypeFilter(request.getStepTypeFilter())
                    .stepStatus(request.getStepStatus())
                    .build();
            currentResponse =
jiraPlaybookInstanceStepExecutionRepository.getUsagePaginated(cur
rentRequest);

            nextPageKey = currentResponse.getNextPageCursor();
            hasNextPage = currentResponse.getHasNextPage();

            if (currentResponse == null ||
currentResponse.getResults() == null ||
currentResponse.getResults().isEmpty()) {
                log.info("No playbook step execution results
found for the given filters. Response object: {}",
currentResponse);
                break;
            }

            List<String> currentPlaybookIds =
currentResponse.getResults().stream()
                    .map(response -> response.getPlaybookId() !=
null ? response.getPlaybookId().toString() : null)
                    .toList();

            List<JiraPlaybookDto> playbooks =
jiraPlaybookService.getPlaybooksByIds(request.getCloudId(),
currentPlaybookIds);

            Set<String> playbookIds;
            playbookIds = extractPlaybookIdsByState(playbooks,
request);

            if (playbookIds.isEmpty()) {
                log.info("No playbook IDs matched the requested
state filter. Playbooks considered: {}", playbooks);
                continue;
            }

            filteredResults =
currentResponse.getResults().stream()
                    .filter(filterresponse ->
playbookIds.contains(
                            filterresponse.getPlaybookId() !=
null ? filterresponse.getPlaybookId().toString() : null
                    ))
                    .toList();

            List<JiraPlaybookInstanceStepExecutionResource>
filteredByName = filteredResults.stream()
                .filter(filterResults ->
StringUtils.isBlank(request.getPlaybookNameFilter())
                        || (filterResults.getPlaybookName() !=
null &&
filterResults.getPlaybookName().equalsIgnoreCase(request.getPlayb
ookNameFilter())))
                .toList();

            List<JiraPlaybookInstanceStepExecutionResource>
filteredByStepType = filteredByName.stream()
                .filter(filterResults ->
request.getStepTypeFilter() == null
                        || (filterResults.getStepType() != null
&&
filterResults.getStepType().name().equals(request.getStepTypeFilt
er())))
                .toList();
```

```
87
88              List<JiraPlaybookInstanceStepExecutionResource>
    currentFilteredResults = filteredByStepType.stream()
89                      .toList();
90
91              if (currentFilteredResults.isEmpty()) {
92                  log.info("No playbook step executions matched all
    applied filters. Filtered results: {}", filteredResults);
93                  continue;
94              }
95
96              for (JiraPlaybookInstanceStepExecutionResource item :
    currentFilteredResults) {
97                  if (!results.contains(item)) {
98                      results.add(item);
99                  }
100             }
101         } while (hasNextPage);
102
103         remainingItems = request.getPageLimit();
104
105         if (results.isEmpty()) {
106             log.info("No results after filtering. Returning empty
    usage connection.");
107             return JiraPlaybookStepUsageConnection.builder()
108                     .nodes(new ArrayList<>())
109                     .edges(new ArrayList<>())
110
    .pageInfo(PageInfo.builder().hasNextPage(false).hasPreviousPage(f
    alse).endCursor(null).build())
111                     .success(true)
112                     .build();
113         }
114
115         List<JiraPlaybookStepUsage> aggregatedUsages =
    JiraPlaybookStepUsageAggregator.aggregateStepUsages(results);
116
117         setOwnerAccountIds(aggregatedUsages,request);
118
    aggregatedUsages.sort(Comparator.comparingLong(JiraPlaybookStepUs
    age::getTotalStepExecutionCount).reversed());
119
120         int startIndex = 0;
121         if (request.getNextPageKey() != null &&
    !request.getNextPageKey().isEmpty()) {
122             for (int i = 0; i < aggregatedUsages.size(); i++) {
123                 if
    (aggregatedUsages.get(i).getId().equals(request.getNextPageKey())
    ) {
124                     startIndex = i + 1;
125                     break;
126                 }
127             }
128         }
129
130         int endIndex = Math.min(startIndex + remainingItems,
    aggregatedUsages.size());
131
132         List<JiraPlaybookStepUsage> pagedUsages =
    aggregatedUsages.subList(startIndex, endIndex);
133
134         hasNextPage = endIndex < aggregatedUsages.size();
135         if (hasNextPage) {
136             endCursor = pagedUsages.getLast().getId();
137         }
138
139         updatePagedUsageIdsAndOwnersToARIFormat(pagedUsages,
    request);
140
141
    List<JiraPlaybookStepUsageConnection.JiraPlaybookStepUsageEdge>
```

```
edges = pagedUsages.stream()
.map(jiraPlaybookStepExecutionMapper::jiraPlaybookStepUsageToEdge)
                .toList();

        PageInfo pageInfo = PageInfo.builder()
                .hasPreviousPage(startIndex > 0)
                .hasNextPage(hasNextPage)
                .endCursor(endCursor)
                .build();

        return JiraPlaybookStepUsageConnection.builder()
                .nodes(pagedUsages)
                .edges(edges)
                .success(true)
                .pageInfo(pageInfo)
                .build();
    }

    private void setOwnerAccountIds(List<JiraPlaybookStepUsage> aggregatedUsages, FetchJiraPlaybooksStepUsageFiltersRequest request) {
        List<String> playbookIds = aggregatedUsages.stream()
                .map(JiraPlaybookStepUsage::getId)
.map(JiraPlaybookUsageServiceImpl::extractPlaybookIdFromCompositeId)
                .toList();

        List<JiraPlaybookDto> playbooks = jiraPlaybookService.getPlaybooksByIds(request.getCloudId(), playbookIds);
        Map<String, JiraPlaybookDto> playbookMap = playbooks.stream()
                .collect(Collectors.toMap(JiraPlaybookDto::getId, Function.identity()));

        for (JiraPlaybookStepUsage usage : aggregatedUsages) {
            JiraPlaybookDto playbook = playbookMap.get(extractPlaybookIdFromCompositeId(usage.getId()));
            String ownerAccountId = playbook != null ? playbook.getOwnerAccountId() : null;
            usage.setOwnerAccountId(ownerAccountId);
        }
    }

    private void updatePagedUsageIdsAndOwnersToARIFormat(List<JiraPlaybookStepUsage> pagedUsages, FetchJiraPlaybooksStepUsageFiltersRequest request) {
        pagedUsages.forEach(playbookStepUsage -> {
            playbookStepUsage.setId(
                    ariUtils.getJiraPlaybookStepARI(
                            request.getCloudId(),
                            null,
extractPlaybookIdFromCompositeId(playbookStepUsage.getId()),
extractStepIdFromCompositeId(playbookStepUsage.getId())
                    )
            );
playbookStepUsage.setOwnerAccountId(ARIUtils.getIdentityUserARI(playbookStepUsage.getOwnerAccountId()));
        });
    }

    public static String extractPlaybookIdFromCompositeId(String compositeId) {
        if (compositeId == null) {
```

```
            return null;
        }
        String[] parts = compositeId.split(":", 2);
        return parts.length > 0 ? parts[0] : null;
    }

    public static String extractStepIdFromCompositeId(String compositeId) {
        if (compositeId == null) {
            return null;
        }
        String[] parts = compositeId.split(":", 2);
        return parts.length > 1 ? parts[1] : null;
    }

    private Set<String> extractPlaybookIdsByState(List<JiraPlaybookDto> playbooks, FetchJiraPlaybooksStepUsageFiltersRequest request) {
        JiraPlaybookStateField state = request.getState();
        if (state != null) {
            return playbooks.stream()
                    .filter(playbook -> playbook.getState() == state)
                    .map(JiraPlaybookDto::getId)
                    .collect(Collectors.toSet());
        } else {
            return playbooks.stream()
                    .map(JiraPlaybookDto::getId)
                    .collect(Collectors.toSet());
        }
    }
}

```

2. JiraPlaybookStepUsageAggregator.java -

```
public class JiraPlaybookStepUsageAggregator {
    /**
     * Private constructor to prevent instantiation of this utility class.
     */
    private JiraPlaybookStepUsageAggregator() {

    }

    public static List<JiraPlaybookStepUsage> aggregateStepUsages(List<JiraPlaybookInstanceStepExecutionResource> resources) {
        class AggregationData {
            int totalExecutionCount = 0;
            int successfulCount = 0;
            int failedCount = 0;
            long totalDuration = 0;
            long minDuration = Long.MAX_VALUE;
            long maxDuration = Long.MIN_VALUE;
            String playbookName = null;
            String stepName = null;
            JiraPlaybookStepType stepType = null;
            Set<String> uniqueAgentIds = new HashSet<>();
            Set<String> uniqueIssueIds = new HashSet<>();

            void add(JiraPlaybookInstanceStepExecutionResource r) {
                totalExecutionCount++;
                if (r.getTriggeredByAccountId() != null) {

uniqueAgentIds.add(r.getTriggeredByAccountId());
                }
                if (r.getContextId() != null) {
                    uniqueIssueIds.add(r.getContextId());
```

```
                }
                if (r.getDuration() != null) {
                    totalDuration += r.getDuration();
                    minDuration = Math.min(minDuration,
r.getDuration());
                    maxDuration = Math.max(maxDuration,
r.getDuration());
                }
                if (playbookName == null && r.getPlaybookName() !=
null) {
                    playbookName = r.getPlaybookName();
                }
                if (stepName == null && r.getStepName() != null) {
                    stepName = r.getStepName();
                }
                if (stepType == null && r.getStepType() != null) {
                    stepType =
JiraPlaybookStepType.valueOf(r.getStepType().name());
                }
                if (r.getStepStatus() != null) {
                    switch (r.getStepStatus()) {
                        case SUCCESS, NO_ACTIONS_PERFORMED,
SUCCESS_UNDONE -> successfulCount++;
                        case FAILED, SOME_ERRORS, FAILURE,
THROTTLED, LOOP, ABORTED, QUEUED_FOR_RETRY, CONFIG_CHANGE, WAITING
-> failedCount++;
                        default -> { }
                    }
                }
            }

            JiraPlaybookStepUsage toJiraPlaybookStepUsage(String
playbookId, String stepId) {
                return JiraPlaybookStepUsage.builder()
                        .id(playbookId + ":" + stepId)
                        .playbookName(playbookName)
                        .stepName(stepName)
                        .stepType(stepType)
                        .totalStepExecutionCount((long)
totalExecutionCount)
                        .uniqueAgentCount((long)
uniqueAgentIds.size())
                        .uniqueIssueCount((long)
uniqueIssueIds.size())
                        .avgExecutionDuration(totalExecutionCount
== 0 ? 0 : totalDuration / totalExecutionCount)
                        .minExecutionDuration(minDuration ==
Long.MAX_VALUE ? 0 : minDuration)
                        .maxExecutionDuration(maxDuration ==
Long.MIN_VALUE ? 0 : maxDuration)
                        .successfulStepExecutionCount((long)
successfulCount)
                        .failedStepExecutionCount((long)
failedCount)
                        .build();
            }
        }

        Map<String, AggregationData> aggregationMap = new
HashMap<>();
        for (JiraPlaybookInstanceStepExecutionResource resource :
resources) {
            String playbookId = resource.getPlaybookId() != null ?
resource.getPlaybookId().toString() : "";
            String stepId = resource.getStepId();
            String key = playbookId + ":" + stepId;
            aggregationMap.computeIfAbsent(key, k -> new
AggregationData()).add(resource);
        }

        List<JiraPlaybookStepUsage> result = new ArrayList<>();
```

```
81          for (Map.Entry<String, AggregationData> entry :
    aggregationMap.entrySet()) {
82              String playbookId =
    JiraPlaybookUsageServiceImpl.extractPlaybookIdFromCompositeId(entr
    y.getKey());
83              String stepId =
    JiraPlaybookUsageServiceImpl.extractStepIdFromCompositeId(entry.ge
    tKey());
84
    result.add(entry.getValue().toJiraPlaybookStepUsage(playbookId,
    stepId));
85          }
86          return result;
87      }
88  }
89
```

---

## Mapper Layer

1. JiraPlaybookStepExecutionMapper.java

| Name | Definition |
| --- | --- |
| 1. JiraPlaybookStepExecutionMapper.java | This helps in generating the cursor with the help of the new PlaybookStep ARI. This helps in paginated retreival of data. |

## Mapper Layer Code

1. **JiraPlaybookStepExecutionMapper.java**

```
1  default JiraPlaybookStepUsageConnection.JiraPlaybookStepUsageEdge
   jiraPlaybookStepUsageToEdge(JiraPlaybookStepUsage
   jiraPlaybookUsage) {
2          return
   JiraPlaybookStepUsageConnection.JiraPlaybookStepUsageEdge.builder(
   )
3                  .cursor(generateCursor(jiraPlaybookUsage))
4                  .node(jiraPlaybookUsage)
5                  .build();
6      }
7
8  default String generateCursor(JiraPlaybookStepUsage
   jiraPlaybookUsage) {
9          return jiraPlaybookUsage.getId() != null ?
   JiraPlaybookStepARI.valueOf(jiraPlaybookUsage.getId()).getStepId()
   : "";
10     }
```

---

## Repository Layer

1. JiraPlaybookInstanceStepExecutionRepository.java

| Name | Definition |
| --- | --- |
| 1. JiraPlaybookUsageConnection getUsagePaginated(FetchJiraPlaybooksRequest request) | Builds a query to fetch a paginated list of playbook step execution resources using a specific index and hash properties (cloudId, activationId, scopeType, scopeId). |

| | |
|---|---|
| | Executes the query and wraps the results in a JiraPlaybookUsageConnection object, which includes the results, pagination info (hasNextPage, hasPreviousPage, nextPageCursor), and is suitable for GraphQL pagination. |
| 2. FilterUtils.java | This method builds a list of filter conditions for querying playbook step usage data based on the provided filter request. It adds conditions for playbook name, scope type, scope ID, start/end time, and step status (supporting multiple statuses with OR logic). All conditions are combined with AND logic and applied to the query request. |

## Repository Layer Code

1. JiraPlaybookInstanceStepExecutionRepository.java

```
public JiraPlaybooksStepExecutionPaginatedResponse getUsagePaginated(FetchJiraPlaybooksStepUsageFiltersRequest request) {
        final QueryRequest<JiraPlaybookInstanceStepExecutionResource> queryRequest = QueryRequest.forClass(JiraPlaybookInstanceStepExecutionResource.class)

.usingIndex(JiraPlaybookInstanceStepExecutionResource.INDEX_CLOUD_ID_ACTIVATION_ID_SCOPE_TYPE_SCOPE_ID_STEP_START)
                .withHash(HashValue.ofMultiPropertyHash(List.of(
                        request.getCloudId(),
                        request.getActivationId()
                )))
                .pageLimit(500)
                .nextPageKey(request.getNextPageKey());

        FilterUtils.addUsageFiltersIfRequired(queryRequest, request);

        QueryResponse<JiraPlaybookInstanceStepExecutionResource> queryResponse = fetchAllQuery(queryRequest);

        boolean hasNextPage = queryResponse.getNextPageKey() != null;
        boolean hasPreviousPage = request.getNextPageKey() != null;

        return JiraPlaybooksStepExecutionPaginatedResponse.builder()
                .results(queryResponse.getResults())
                .hasNextPage(hasNextPage)
                .hasPreviousPage(hasPreviousPage)
                .nextPageCursor(queryResponse.getNextPageKey())
                .build();
    }
```

2. FilterUtils.java

```
public static void addUsageFiltersIfRequired(QueryRequest<?> queryRequest, FetchJiraPlaybooksStepUsageFiltersRequest fetchJiraPlaybooksStepUsageFiltersRequest) {
        List<Condition> filters = new ArrayList<>();
```

```
3
4           if
(fetchJiraPlaybooksStepUsageFiltersRequest.getPlaybookNameFilter()
!= null
5                   &&
StringUtils.isNotEmpty(fetchJiraPlaybooksStepUsageFiltersRequest.g
etPlaybookNameFilter().toLowerCase())) {
6
filters.add(Condition.singlePropertyCondition("playbookNameLowerca
se", ComparisonOperator.CONTAINS,
7
fetchJiraPlaybooksStepUsageFiltersRequest.getPlaybookNameFilter().
toLowerCase()
8               ));
9           }
10          if
(fetchJiraPlaybooksStepUsageFiltersRequest.getScopeType() != null)
{
11
filters.add(Condition.singlePropertyCondition("scopeType",
ComparisonOperator.EQUALS,
fetchJiraPlaybooksStepUsageFiltersRequest.getScopeType()));
12          }
13
14          if
(StringUtils.isNotEmpty(fetchJiraPlaybooksStepUsageFiltersRequest.
getScopeId())) {
15
filters.add(Condition.singlePropertyCondition("scopeId",
ComparisonOperator.EQUALS,
fetchJiraPlaybooksStepUsageFiltersRequest.getScopeId()));
16          }
17
18          if
(fetchJiraPlaybooksStepUsageFiltersRequest.getStartTime() != null)
{
19              long from =
Long.parseLong(fetchJiraPlaybooksStepUsageFiltersRequest.getStartT
ime());
20
filters.add(Condition.singlePropertyCondition("stepStartTime",
ComparisonOperator.GREATER_THAN_OR_EQUALS, from));
21          }
22          if (fetchJiraPlaybooksStepUsageFiltersRequest.getEndTime()
!= null) {
23              long to =
Long.parseLong(fetchJiraPlaybooksStepUsageFiltersRequest.getEndTim
e());
24
filters.add(Condition.singlePropertyCondition("stepEndTime",
ComparisonOperator.LESS_THAN_OR_EQUALS, to));
25          }
26          if
(fetchJiraPlaybooksStepUsageFiltersRequest.getStepStatus() != null
&&
!fetchJiraPlaybooksStepUsageFiltersRequest.getStepStatus().isEmpty
()) {
27              List<JiraPlaybookStepRunStatus>  stepStatusFilter =
fetchJiraPlaybooksStepUsageFiltersRequest.getStepStatus();
28              List<Condition> statusFilterConditions =
stepStatusFilter.stream()
29                      .map(status ->
Condition.singlePropertyCondition("stepStatus",
ComparisonOperator.EQUALS, status))
30                      .toList();
31              filters.add(statusFilterConditions.size() == 1 ?
statusFilterConditions.getFirst() :
Condition.groupedCondition(LogicalOperator.OR,
statusFilterConditions));
32          }
33
```

```
        if (!filters.isEmpty()) {
            queryRequest.where(filters.size() == 1 ?
filters.getFirst() :
Condition.groupedCondition(LogicalOperator.AND, filters));
        }
    }
```

## Testing Layer

1. JiraPlaybookStepUsageServiceIT.java

| Name | Definition |
| --- | --- |
| 1. JiraPlaybookStepUsageServiceIT.java | This helps in generating the cursor with the help of the new PlaybookStep ARI. This helps in paginated retreival of data. |

## Testing Layer Code

1. **JiraPlaybookStepUsageServiceIT.java**

```
 @Test
    void getUsageAggregatedMetricsTest() {
        executeAllTestStepsForUsageTab();
        FetchJiraPlaybooksStepUsageFiltersRequest request =
FetchJiraPlaybooksStepUsageFiltersRequest.builder()
                .nextPageKey(null)
                .cloudId(TEST_CLOUD_ID)
                .activationId(TEST_ACTIVATION_ID)
                .state(JiraPlaybookStateField.ENABLED)
                .pageLimit(3)
                .build();
        JiraPlaybookStepUsageConnection metrics =
jiraPlaybookUsageService.getPlaybookUsagePaginated(request);

        assertEquals(3, metrics.getNodes().size());

assertThat(metrics.getPageInfo().getHasNextPage()).isTrue();

assertThat(metrics.getPageInfo().getEndCursor()).isNotNull();

        String nextPageKey =
metrics.getPageInfo().getEndCursor();
        while (nextPageKey != null) {
            request =
FetchJiraPlaybooksStepUsageFiltersRequest.builder()
                    .cloudId(TEST_CLOUD_ID)
                    .activationId(TEST_ACTIVATION_ID)
                    .state(JiraPlaybookStateField.ENABLED)
                    .pageLimit(1)
                    .nextPageKey(nextPageKey)
                    .build();
            JiraPlaybookStepUsageConnection metrics2 =
jiraPlaybookUsageService.getPlaybookUsagePaginated(request);

            assertEquals(1, metrics2.getNodes().size());

assertThat(metrics2.getPageInfo().getHasNextPage()).isFalse();

assertThat(metrics2.getPageInfo().getEndCursor()).isNull();

            nextPageKey = metrics2.getPageInfo().getEndCursor();
        }
    }

    @Test
```

```
    void getUsageAggregatedMetricsWithStepTypeFilterTest() {
        executeAllTestStepsForUsageTab();
        FetchJiraPlaybooksStepUsageFiltersRequest request = FetchJiraPlaybooksStepUsageFiltersRequest.builder()
                .nextPageKey(null)
                .cloudId(TEST_CLOUD_ID)
                .activationId(TEST_ACTIVATION_ID)
                .state(JiraPlaybookStateField.ENABLED)
                .pageLimit(5)
                .stepTypeFilter(String.valueOf(JiraPlaybookStepType.AUTOMATION_RULE))
                .build();
        JiraPlaybookStepUsageConnection metrics = jiraPlaybookUsageService.getPlaybookUsagePaginated(request);

        assertEquals(2, metrics.getNodes().size());
        assertThat(metrics.getPageInfo().getHasNextPage()).isFalse();
        assertThat(metrics.getPageInfo().getEndCursor()).isNull();
    }

    @Test
    void getUsageAggregatedMetricsWithPlaybookNameFilterTest() {
        executeAllTestStepsForUsageTab();
        FetchJiraPlaybooksStepUsageFiltersRequest request = FetchJiraPlaybooksStepUsageFiltersRequest.builder()
                .nextPageKey(null)
                .cloudId(TEST_CLOUD_ID)
                .activationId(TEST_ACTIVATION_ID)
                .state(JiraPlaybookStateField.ENABLED)
                .pageLimit(4)
                .playbookNameFilter(jiraPlaybookDto.getName())
                .build();
        JiraPlaybookStepUsageConnection metrics = jiraPlaybookUsageService.getPlaybookUsagePaginated(request);

        assertEquals(2, metrics.getNodes().size());
        assertThat(metrics.getPageInfo().getHasNextPage()).isFalse();
        assertThat(metrics.getPageInfo().getEndCursor()).isNull();
    }

    @Test
    void getUsageAggregatedMetricsWithScopeIdFilterTest() {
        executeAllTestStepsForUsageTab();
        FetchJiraPlaybooksStepUsageFiltersRequest request = FetchJiraPlaybooksStepUsageFiltersRequest.builder()
                .nextPageKey(null)
                .cloudId(TEST_CLOUD_ID)
                .activationId(TEST_ACTIVATION_ID)
                .state(JiraPlaybookStateField.ENABLED)
                .pageLimit(5)
                .scopeId(jiraPlaybookDto2.getScopeId())
                .build();
        JiraPlaybookStepUsageConnection metrics = jiraPlaybookUsageService.getPlaybookUsagePaginated(request);

        assertEquals(2, metrics.getNodes().size());
        assertThat(metrics.getPageInfo().getHasNextPage()).isFalse();
        assertThat(metrics.getPageInfo().getEndCursor()).isNull();
    }

    @Test
    void getUsageAggregatedMetricsWithScopeTypeFilterTest() {
        executeAllTestStepsForUsageTab();
```

```
93          FetchJiraPlaybooksStepUsageFiltersRequest request =
    FetchJiraPlaybooksStepUsageFiltersRequest.builder()
94                  .nextPageKey(null)
95                  .cloudId(TEST_CLOUD_ID)
96                  .activationId(TEST_ACTIVATION_ID)
97                  .state(JiraPlaybookStateField.ENABLED)
98                  .pageLimit(5)
99                  .scopeType(jiraPlaybookDto.getScopeType())
100                 .build();
101         JiraPlaybookStepUsageConnection metrics =
    jiraPlaybookUsageService.getPlaybookUsagePaginated(request);
102
103         assertEquals(4, metrics.getNodes().size());
104
    assertThat(metrics.getPageInfo().getHasNextPage()).isFalse();
105
    assertThat(metrics.getPageInfo().getEndCursor()).isNull();
106     }
107
108     @Test
109     void getUsageAggregatedMetricsWithStepStatusFilterTest() {
110         executeAllTestStepsForUsageTab();
111         FetchJiraPlaybooksStepUsageFiltersRequest request =
    FetchJiraPlaybooksStepUsageFiltersRequest.builder()
112                 .nextPageKey(null)
113                 .cloudId(TEST_CLOUD_ID)
114                 .activationId(TEST_ACTIVATION_ID)
115                 .state(JiraPlaybookStateField.ENABLED)
116                 .pageLimit(5)
117
    .stepStatus(Arrays.asList(JiraPlaybookStepRunStatus.SUCCESS,
    JiraPlaybookStepRunStatus.FAILED))
118                 .build();
119         JiraPlaybookStepUsageConnection metrics =
    jiraPlaybookUsageService.getPlaybookUsagePaginated(request);
120
121         assertEquals(4, metrics.getNodes().size());
122
    assertThat(metrics.getPageInfo().getHasNextPage()).isFalse();
123
    assertThat(metrics.getPageInfo().getEndCursor()).isNull();
124     }
125     @Test
126     void getUsageAggregatedMetricsWithDateTimeFilterTest() {
127 //        executeAllTestStepsForUsageTab();
128         JiraPlaybookInstanceStepARI
    jiraPlaybookInstanceStepARIAutomation =
    JiraPlaybookInstanceStepARI.from(TEST_CLOUD_ID,
    TEST_ACTIVATION_ID, jiraPlaybookId,
    jiraPlaybookAutomationRuleStepId, TEST_ISSUE_CONTEXT,
    TEST_ISSUE_ID);
129         JiraPlaybookInstanceStepARI
    jiraPlaybookInstanceStepARIInstruction =
    JiraPlaybookInstanceStepARI.from(TEST_CLOUD_ID,
    TEST_ACTIVATION_ID, jiraPlaybookId,
    jiraPlaybookInstructionalRuleStepId, TEST_ISSUE_CONTEXT,
    TEST_ISSUE_ID);
130         JiraPlaybookInstanceStepARI
    jiraPlaybookInstanceStepARIDifferent =
    JiraPlaybookInstanceStepARI.from(TEST_CLOUD_ID,
    TEST_ACTIVATION_ID, jiraPlaybookDto2.getId(),
    jiraPlaybookDto2.getSteps().getFirst().getStepId(),
    TEST_ISSUE_CONTEXT, TEST_ISSUE_ID);
131         JiraPlaybookInstanceStepARI
    jiraPlaybookInstanceStepARIInstruction2 =
    JiraPlaybookInstanceStepARI.from(TEST_CLOUD_ID,
    TEST_ACTIVATION_ID, jiraPlaybookDto2.getId(),
    jiraPlaybookDto2.getSteps().getLast().getStepId(),
    TEST_ISSUE_CONTEXT, TEST_ISSUE_ID);
132
    when(featureGateService.isEnabledForTenant(eq(DONE_UNDONE_INSTRUC
```

```
TIONAL_STEP), any())).thenReturn(true);

        JiraPlaybookInstanceStepExecutionDto executionDto =
jiraPlaybookInstanceStepExecutionService.executeJiraPlaybookInsta
nceStep(jiraPlaybookInstanceStepARIAutomation, List.of(),
TargetTransition.DONE).getExecutionDto();
        JiraPlaybookInstanceStepExecutionDto executionDto2 =
jiraPlaybookInstanceStepExecutionService.executeJiraPlaybookInsta
nceStep(jiraPlaybookInstanceStepARIAutomation, List.of(),
TargetTransition.DONE).getExecutionDto();
        JiraPlaybookInstanceStepExecutionDto
executionDto3Different =
jiraPlaybookInstanceStepExecutionService.executeJiraPlaybookInsta
nceStep(jiraPlaybookInstanceStepARIDifferent, List.of(),
TargetTransition.DONE).getExecutionDto();
        JiraPlaybookInstanceStepExecutionDto executionDto5 =
jiraPlaybookInstanceStepExecutionService.executeJiraPlaybookInsta
nceStep(jiraPlaybookInstanceStepARIAutomation, List.of(),
TargetTransition.DONE).getExecutionDto();
        JiraPlaybookInstanceStepExecutionDto executionDto6 =
jiraPlaybookInstanceStepExecutionService.executeJiraPlaybookInsta
nceStep(jiraPlaybookInstanceStepARIAutomation, List.of(),
TargetTransition.DONE).getExecutionDto();
        JiraPlaybookInstanceStepExecutionDto executionDto7 =
jiraPlaybookInstanceStepExecutionService.executeJiraPlaybookInsta
nceStep(jiraPlaybookInstanceStepARIDifferent, List.of(),
TargetTransition.DONE).getExecutionDto();
        JiraPlaybookInstanceStepExecutionDto executionDto8 =
jiraPlaybookInstanceStepExecutionService.executeJiraPlaybookInsta
nceStep(jiraPlaybookInstanceStepARIAutomation, List.of(),
TargetTransition.DONE).getExecutionDto();
        JiraPlaybookInstanceStepExecutionDto executionDto9 =
jiraPlaybookInstanceStepExecutionService.executeJiraPlaybookInsta
nceStep(jiraPlaybookInstanceStepARIInstruction2, List.of(),
TargetTransition.DONE).getExecutionDto();
        JiraPlaybookInstanceStepExecutionDto executionDto4 =
jiraPlaybookInstanceStepExecutionService.executeJiraPlaybookInsta
nceStep(jiraPlaybookInstanceStepARIInstruction, List.of(),
TargetTransition.DONE).getExecutionDto();
        JiraPlaybookInstanceStepExecutionDto executionDto10 =
jiraPlaybookInstanceStepExecutionService.executeJiraPlaybookInsta
nceStep(jiraPlaybookInstanceStepARIInstruction, List.of(),
TargetTransition.UNDONE).getExecutionDto();
        JiraPlaybookInstanceStepExecutionDto executionDto11 =
jiraPlaybookInstanceStepExecutionService.executeJiraPlaybookInsta
nceStep(jiraPlaybookInstanceStepARIInstruction, List.of(),
TargetTransition.DONE).getExecutionDto();
        FetchJiraPlaybooksStepUsageFiltersRequest request =
FetchJiraPlaybooksStepUsageFiltersRequest.builder()
                .nextPageKey(null)
                .cloudId(TEST_CLOUD_ID)
                .activationId(TEST_ACTIVATION_ID)
                .state(JiraPlaybookStateField.ENABLED)
                .pageLimit(5)

.startTime(String.valueOf(executionDto.getStartTime().toEpochMill
i()))

.endTime(String.valueOf(executionDto.getEndTime().toEpochMilli())
)
                .build();
        JiraPlaybookStepUsageConnection metrics =
jiraPlaybookUsageService.getPlaybookUsagePaginated(request);

        assertEquals(1, metrics.getNodes().size());

assertThat(metrics.getPageInfo().getHasNextPage()).isFalse();

assertThat(metrics.getPageInfo().getEndCursor()).isNull();
    }

```

```
161     protected void executeAllTestStepsForUsageTab() {
162         JiraPlaybookInstanceStepARI
    jiraPlaybookInstanceStepARIAutomation =
    JiraPlaybookInstanceStepARI.from(TEST_CLOUD_ID,
    TEST_ACTIVATION_ID, jiraPlaybookId,
    jiraPlaybookAutomationRuleStepId, TEST_ISSUE_CONTEXT,
    TEST_ISSUE_ID);
163         JiraPlaybookInstanceStepARI
    jiraPlaybookInstanceStepARIInstruction =
    JiraPlaybookInstanceStepARI.from(TEST_CLOUD_ID,
    TEST_ACTIVATION_ID, jiraPlaybookId,
    jiraPlaybookInstructionalRuleStepId, TEST_ISSUE_CONTEXT,
    TEST_ISSUE_ID);
164         JiraPlaybookInstanceStepARI
    jiraPlaybookInstanceStepARIDifferent =
    JiraPlaybookInstanceStepARI.from(TEST_CLOUD_ID,
    TEST_ACTIVATION_ID, jiraPlaybookDto2.getId(),
    jiraPlaybookDto2.getSteps().getFirst().getStepId(),
    TEST_ISSUE_CONTEXT, TEST_ISSUE_ID);
165         JiraPlaybookInstanceStepARI
    jiraPlaybookInstanceStepARIInstruction2 =
    JiraPlaybookInstanceStepARI.from(TEST_CLOUD_ID,
    TEST_ACTIVATION_ID, jiraPlaybookDto2.getId(),
    jiraPlaybookDto2.getSteps().getLast().getStepId(),
    TEST_ISSUE_CONTEXT, TEST_ISSUE_ID);
166
    when(featureGateService.isEnabledForTenant(eq(DONE_UNDONE_INSTRUC
    TIONAL_STEP), any())).thenReturn(true);
167
168         JiraPlaybookInstanceStepExecutionDto executionDto =
    jiraPlaybookInstanceStepExecutionService.executeJiraPlaybookInsta
    nceStep(jiraPlaybookInstanceStepARIAutomation, List.of(),
    TargetTransition.DONE).getExecutionDto();
169         JiraPlaybookInstanceStepExecutionDto executionDto2 =
    jiraPlaybookInstanceStepExecutionService.executeJiraPlaybookInsta
    nceStep(jiraPlaybookInstanceStepARIAutomation, List.of(),
    TargetTransition.DONE).getExecutionDto();
170         JiraPlaybookInstanceStepExecutionDto
    executionDto3Different =
    jiraPlaybookInstanceStepExecutionService.executeJiraPlaybookInsta
    nceStep(jiraPlaybookInstanceStepARIDifferent, List.of(),
    TargetTransition.DONE).getExecutionDto();
171         JiraPlaybookInstanceStepExecutionDto executionDto5 =
    jiraPlaybookInstanceStepExecutionService.executeJiraPlaybookInsta
    nceStep(jiraPlaybookInstanceStepARIAutomation, List.of(),
    TargetTransition.DONE).getExecutionDto();
172         JiraPlaybookInstanceStepExecutionDto executionDto6 =
    jiraPlaybookInstanceStepExecutionService.executeJiraPlaybookInsta
    nceStep(jiraPlaybookInstanceStepARIAutomation, List.of(),
    TargetTransition.DONE).getExecutionDto();
173         JiraPlaybookInstanceStepExecutionDto executionDto7 =
    jiraPlaybookInstanceStepExecutionService.executeJiraPlaybookInsta
    nceStep(jiraPlaybookInstanceStepARIDifferent, List.of(),
    TargetTransition.DONE).getExecutionDto();
174         JiraPlaybookInstanceStepExecutionDto executionDto8 =
    jiraPlaybookInstanceStepExecutionService.executeJiraPlaybookInsta
    nceStep(jiraPlaybookInstanceStepARIAutomation, List.of(),
    TargetTransition.DONE).getExecutionDto();
175         JiraPlaybookInstanceStepExecutionDto executionDto9 =
    jiraPlaybookInstanceStepExecutionService.executeJiraPlaybookInsta
    nceStep(jiraPlaybookInstanceStepARIInstruction2, List.of(),
    TargetTransition.DONE).getExecutionDto();
176         JiraPlaybookInstanceStepExecutionDto executionDto4 =
    jiraPlaybookInstanceStepExecutionService.executeJiraPlaybookInsta
    nceStep(jiraPlaybookInstanceStepARIInstruction, List.of(),
    TargetTransition.DONE).getExecutionDto();
177         JiraPlaybookInstanceStepExecutionDto executionDto10 =
    jiraPlaybookInstanceStepExecutionService.executeJiraPlaybookInsta
    nceStep(jiraPlaybookInstanceStepARIInstruction, List.of(),
    TargetTransition.UNDONE).getExecutionDto();
```

```
178         JiraPlaybookInstanceStepExecutionDto executionDto11 =
    jiraPlaybookInstanceStepExecutionService.executeJiraPlaybookInsta
    nceStep(jiraPlaybookInstanceStepARIInstruction, List.of(),
    TargetTransition.DONE).getExecutionDto();
179     }
180
```

---

## Input Types

### Enums

1. `enum JiraPlaybookStateField { ENABLED , DISABLED }`
2. `enum JiraPlaybookStepType { INSTRUCTIONAL_RULE , AUTOMATION_RULE }`
3. `enumJiraPlaybookScopeType { PROJECT , TEAM , GLOBAL }`

## Error Handling

### QueryError

- **fields**: Defines which fields the error pertains to.
- **message**: Describes the error message.
- **code**: A machine-readable code for the error.
- **details**: for query-specific errors details should be in the message.

Errors are handled by returning structured error objects in the `errors` field of query payloads, allowing clients to programmatically handle issues that arise during API operations.

# LDR: New ARI for Playbook Step

| **Status** | **MERGED** |
| --- | --- |
| Component | ARI for Playbook Step |
| Drivers | @Sri Sindhu Veerathu |
| Reviewer | @Gaurav Arora @Vinit Jain |
| Informed | |

**What is an ARI?**

An **ARI (Atlassian Resource Identifier)** is a globally unique identifier for an entity within the Atlassian ecosystem. It is used to represent all instances and representations of an entity across all Atlassian services and APIs.

ARI is crucial for **AGG (Atlassian Graph Gateway)** because it enables AGG to direct queries to the appropriate tenant identified by the ARI.

## Playbook Step

This ARI is used to uniquely identify and reference a specific step within a playbook in Jira.

It contains details about the product (Jira), the entity type (playbook-step), and the unique identifiers for the activation, playbook, and step.

```
ari:cloud:jira:{siteId}:playbook-step/activation/{activationId}/{playbookId}/{stepId}
```

Above ARI has below information:

1. which product owns this object - `jira`
2. which type of entity this object is - `playbook-step`
3. what is the internal id of this object - `stepId`

   Here `stepId` is step Id of playbook step.