Automated Replication of Internal Resource Type Permissions

A Research Paper submitted to the Department of Engineering and Society

Presented to the Faculty of the School of Engineering and Applied Science
University of Virginia • Charlottesville, Virginia

In Partial Fulfillment of the Requirements for the Degree
Bachelor of Science, School of Engineering

Emily Lu
Fall 2021

On my honor as a University Student, I have neither given nor received unauthorized aid on this assignment as defined by the Honor Guidelines for Thesis-Related Assignments

Advisors

Rosanne Vrugtman, Department of Engineering and Computer Science

Jim Cohoon, Department of Engineering and Computer Science

Automated Replication of Internal Resource Type Permissions

Emily Lu
Department of Computer Science
The University of Virginia
School of Engineering and Applied Science
Charlottesville VA USA
el6ct@virginia.edu

ABSTRACT

Manually replicating database modifications across multiple regions is not only time-consuming but can also lead to potential mistakes such as replicating in an incorrect region or creating/deleting a permission with the wrong configurations. To reduce such mistakes, automation of the replication workflow is crucial to improving the efficiency and consistency of future related operations within the organization. As an intern for Amazon Web Services, I created an automated workflow for creating and deleting internal permissions over certain resource types for the Resource Ownership Service (ROS) team by developing APIs and linking them to a Simple Queue Service (SQS) and Lambda. Through the workflow, users can create or delete internal permissions by calling designated APIs, and the resulting create or delete is then automatically replicated through all regions in which the corresponding resource type exists.

1 INTRODUCTION

Amazon Web Service (AWS) users are able to create and share resources with other individuals or organizations. Since resources can contain private or sensitive information, users can create, delete, and update permissions to restrict access to shared resources across multiple regions. However, manually creating and replicating those permissions can result in slips in security from mistakes such as accidentally replicating a permission in the wrong region, creating or deleting the wrong permission version inconsistently, and so on. Correcting such mistakes can take a large amount of valuable time away from developers, spending time searching through each region to find what could be a minor typo instead of being able to work on an actual project or resolving customer tickets. Therefore, automating the replication process not only reduces the likelihood of user error, but also reduces the amount of time needed to perform a simple create/ delete and to resolve any permission configuration mistakes, allowing developers to focus their time and efforts on more important objectives.

2 BACKGROUND

Given that AWS is a cloud computing service that serves as the foundation of many other large companies such as Netflix, the reliability of companies using AWS services is, to a certain degree, directly correlated with the reliability of AWS as a service [1]. Hence, AWS developers consistently receive a large influx of customer support tickets, and are constantly creating new features and tools for both internal and external organizations. Customer satisfaction is always the number one priority at Amazon as a whole, and thus despite the presence of many talented engineers working for AWS, working overtime is extremely common especially among smaller, understaffed teams. AWS engineers especially are pushed to work at a rapid pace to deliver results as soon as possible, making automation a crucial component to any workflow.

3 RELATED WORKS

With the exponentially growing tech industry, more companies are embodying the agile development method. Especially at AWS where their own products can directly impact the success of other large companies and customer satisfaction is the most important principle, developers are constantly pushing out new features and updates, and working with a high volume of customer support tickets [2]. Combined with the agile development culture, AWS developers are typically pressed for time, but fixing bugs and potential mistakes take up valuable time that could be spent on developing an actual product. A key component of agile development automation—more automation means reduced overhead, opening up more time for developing features and extensive testing. The need for increased automation in AWS' internal operations served as the basis of my internship project.

4 PROJECT DESIGN

My replication tool was designed to be simple and quick to use for users, allowing users to simply call the corresponding API command in command line and entering the proper permission parameters, and the workflow would then process the command, handling the actual creation/deletion and replication automatically.

4.1 Review of System Architecture

There are numerous resource types in AWS, and users can define custom managed permissions for each resource type. A managed permission consists of the following fields: permission name, service name, resource type, version, default version, and actions. Each resource type also has a pre-defined mapping of leader and

destination regions, which will be expanded upon in the breakdown of the replication workflow.

The overall workflow of the permission replication workflow consists of two main sub-components: the APIs that handle database mutations, and the Simple Queue Service (SQS) and Lambda that handle the replication workflow. Both the create and delete calls consist of a pair of APIs each—one API to route the call and one API to mutate the database. When the user enters a command line command for a permission create or delete, the user is calling the corresponding router API. Next, the router API will then evaluate if the call should be redirected to the replication workflow, the database API, or both. Within the replication workflow is the SQS queue which directly receives "messages" from the router APIs and triggers the proper Lambda function to handle the incoming message. Messages are generated by converting the user input into a JavaScript Object Notation (JSON) string. Each AWS region will have its own corresponding SQS queue and Lambda, such that every region acts independently of each other. When either a create or delete is called, the call will be routed to the corresponding resource type's leader region queue, which will then send a copy of the received message to all corresponding destination region queues. From there, each destination region queue will trigger its associated Lambda and modify the database.

4.2 Review of Replication Use Case Logic

Each resource type has a corresponding leader region, and one or more destination regions. Depending on which region the router API is called from, the behavior of the workflow will differ. The possible cases are as follows:

1. Routing API is called by the user from the leader region
2. Routing API is called by the user from a destination region
3. Routing API is called by a destination region's lambda

In the first case, the routing API will send the generated message to the leader region's lambda function and call the database mutation API and execute the corresponding create/delete in the current region. Similarly, the second case will send the generated message to the correct leader region lambda, but the database mutation API will not be called immediately. In case three, the routing API will call the database mutation API and execute the create or delete in the caller region. There will never be a case where the leader region's lambda will call the routing API, as the database execution is handled by case one and the leader region lambda will only send messages to destination region lambdas to prevent potential overhead.

4.3 Project Requirements

In addition to the use case logic specified previously, there are additional checks needed prior to modifying the database itself. Required checks include verifying the caller's account ID (which is an auto-generated parameter specified in the request when calling an API), as well as any user-defined parameters in each API call. In both the create and delete APIs, there is an "allow-list" that contains a list of account IDs that are authorized to call the corresponding API. Prior to executing a create or delete mutation in the database, the caller's account ID is checked against the allow-list to verify that the caller has permission to change the database—if the corresponding account ID is not found in the allow-list, their API request will be denied and terminated.

For a create API call, the user is required to specify a permission name, resource type, service name, default version, and a list of actions. When attempting to add an entry to the database, the permission name will be converted to an Amazon Resource Name (ARN), and a scan of the database will check whether there exists a permission with that ARN. If there does not exist a permission with the generated ARN, a new permission will be created with a default version number set to 1, otherwise the permission will be added into the database with a version number one greater than the largest existing version number. Service name, resource type, and actions will then be checked all together—resource types are a subset of a service, and actions are a subset of a resource type. The user-provided service name, resource type, and actions will be checked against a pre-defined mapping of the relationship between the three and verify that the user's combination exists in that mapping. In the event that the provided combination does not exist, the create request will terminate.

For a delete API call, the user is required to specify a permission ARN and a version number. When initiating a delete API call, the database of permissions is scanned to check for an existing permission with the provided ARN and version number. If the specified combination does not exist, the delete request is terminated. If there exists an entry with the provided parameters, two additional checks are made. The first check will determine whether the specified permission is the default version, such that any permission that is the default version cannot be deleted. The second check will scan the database to determine whether the provided permission is currently associated with any active resource types, such that permissions associated to an active resource type also cannot be deleted.

4.4 Challenges

As I worked on a primarily independent project, I faced many challenges throughout the development process such as understanding the team's existing structure and code base, design changes, dependency delays, and pipeline blockages. As a new intern at AWS, there were many new tools and resources that were foreign to me and learning how to use all of them in the span of three months proved to be very difficult. With a project heavily reliant on knowledge of the team's existing infrastructure, I had to constantly dig through the team's code base, ask many clarification questions, and study additional documentation on the side. Juggling a game of catch-up at the same time as working to complete a large, independent project in the span of merely three months, keeping a consistent pace of progress proved to be very difficult. Additionally, documentation was difficult to find, and

what documentation could be found tended to be confusing or lacking. The lack of clear and trackable documentation is a side effect of the sheer amount of code in Amazon's code base, and with short development periods and fast turnaround times, ensuring that the features are bug-free and functioning as expected take precedence over detailed documentation for every code snippet.

Throughout the entire development process, many changes were consistently being made to my design document. Some changes only modified certain fields and smaller features, while others could completely change the foundation of the implementation logic. Being able to adapt to and implement design changes quickly was another challenge as it created overhead and slowed down the pace of progression. Since my project involved many checks for authorized users, existing structures, and valid fields which were not to be defined in my implementation, without those structures pre-defined in an easy to access manner, many checks had to be hard-coded to serve as a temporary replacement for the full scope of its intended functionality. For example, an allow list of authorized users was hard-coded to only contain a few account IDs, and the mapping of valid services to resource types to actions was also hard-coded to only contain a small subset of all possible, existing combinations.

Lastly, many of my changes were being pushed through an existing pipeline, which contained many packages that other team members were also pushing changes to. Containing so many packages made it easy for the pipeline to fail and be blocked by a simple bug even in the smallest code changes, which would stop all subsequent updates to go through the pipeline until the bug was resolved and the push could successfully pass all integration tests at that stage. Furthermore, running the automated tests in each stage of the pipeline took long periods of time, and to go from one stage of the pipeline to the next, all the corresponding integration tests needed to pass. Hence, during the testing phase of my project, there were many blockages in the pipeline that prevented me from being able to test my code all the way through, and I had to improvise new testing methods even just days before my final presentation. Ultimately, I was able to find a sub-par testing method by writing a script to run the commands locally and at least test the workflow partially.

5 RESULTS

Internal teams at AWS can now create and delete internal permissions with a simple command line command, and have their request automatically replicated across all resource type regions. Although I was not able to test the workflow all the way through, each segment of the workflow is working as expected, and assuming that all segments work together as intended, what used to take up to twenty minutes to complete can now be completed in under five minutes. Not only does the replication workflow decrease the amount of time a user needs to create or delete a permission, but it also reduces the number of mistakes that can be made during a create or delete. Having an automated replication system ensures that the permissions for each resource type will be consistent across all applicable regions.

6 CONCLUSION

As the number of AWS users continues to increase, the amount of sensitive data that AWS contains will also increase. Especially with the rapid pace at which technology is growing, developers are pushed to constantly update their code base, creating new features and working to improve security and efficiency. Amidst such a fast-paced industry, automation is a valuable tool in reducing human error and overhead from repetitive procedures. Thus, this project works to reduce human error and save time for AWS developers, leaving them with more time to work on customer-related work and meet deadlines more easily. The project was structured in such a way that it would be easy to update and understand for other developers, keeping the same structural and architectural format as other tools for the developing team. Additionally, the resulting tool from the project was designed to be intuitive and easy to use, requiring minimal user involvement and reducing potential confusion in how the tool was intended to be used.

7 FUTURE WORK

With such a short three month internship, there were only so many features that I was able to fully implement. Many structures that were hard-coded as they were not already pre-defined in a separate database, and I was unable to successfully test the entire workflow end to end. Among those hard-coded components, the allow-list needs to be migrated to a separate internal system that contains a database of authorized account IDs. The existing internal system is linked to a front end component such that developers will be able to add themselves to a list of authorized account IDs with proper admin approval. Next, a formal definition of leader and destination regions needs to be made for each resource type region in a database such that future changes to the mappings will automatically be reflected in the workflow as well to avoid manual updates. The last hard-coded component consists of the mapping between valid services, resource types, and actions. The mapping of the relationship between the three types will need to be migrated to a formal database so that like the leader and destination regions, changes to the relationships will be automatically reflected in the workflow instead of needing manual updates. Lastly, more unit tests and integration tests need to be written to meet overall coverage requirements. Unit tests should be written for each sub-component of the overall workflow, namely each of the APIs and the SQS to Lambda communication. Integration tests should be written to test the entire workflow from end to end.

8 UVA PROGRAM EVALUATION

The most useful component of my program of study was Advanced Software Development, which introduced me to the workings of an agile development, one of the most used development cycles in large tech companies. Technical knowledge from other courses such as Computer Architecture did not directly translate over into my internship experience as the information was not applicable to the scope of my project. However other courses in the program helped me improve my skills in searching for relevant information needed to understand how to use new tools as well as common coding practices. Courses that could prove to be helpful and more relevant to a real working situation would be a course focused on developing and understanding common software application component architecture designs such as the relationship between an API and a database, and how they are synthesized.

REFERENCES

[1] Amazon Web Services. (2020). Netflix on AWS. Amazon. Retrieved October 22, 2021, from https://aws.amazon.com/solutions/case-studies/netflix/.

[2] Amazon. (2021, August). Amazon's Global Career Site. amazon.jobs. Retrieved October 22, 2021, from https://www.amazon.jobs/en/principles.