🔹 Aladdin Engineering – Observability (May 2024 – Present)

1. Filename Encoder App

What is URL-encoding, and why is it necessary for filenames?
URL encoding refers to the process of converting characters into a format that can be safely transmitted over the internet in a URL. It was necessary for filenames because another team in Private Markets department encountered an issue with the file transfers from the BlackRock network drive into their team project application. The issue was specifically related to the fact that the Pollers they used failed whenever the file names contained spaces or alphanumeric characters. And Url encoding became necessary solution because it encoded all the characters into alphanumeric+% character which the Poller could work with

- **How did you implement the encoding logic in Python? (Which libraries did you use?)**

  So for this project we used the os python library for directory identifying directories and their contents and moving files, urllib.parse for encoding filenames, shutil for copying the files, argparse for taking in command line arguments for the app, logging for logging important stages of the app run, time for pausing the app run for a fixed period. So the basic logic of this was the app would sit between the original source directory for the Poller and the a destination directory which would become Poller's new source directory and would contain the encoded filename versions of the files. Now the app would take files from the original source directory, move them to the a dummy "working directory" to ensure that we have the files we are working with in that period, copy them from working to an optional archive directory which would help to differentiate which apps have already been moved in the previous job, then url encode the files in the working directory and then move them to the destination directory

- **What were the possible edge cases you considered in the filenames?**

  The only real edge cases for this was if the file didn't exist for the encoding in which cases we skipped trying to encode and move altogether. A good example of this is if the name of the directory content is not actually a file but a directory instead.

- **How did you handle non-ASCII characters or special characters like spaces and slashes?**

  Urllib.parse.quote function call automatically handled the encoding of the special characters and non-ASCII which was the whole point of the project to begin with

- **How did you ensure the filenames remained compatible with downstream systems or file readers?**

  To ensure that the filenames remained compatible with our Poller for pickup, we added tests that confirmed that there were no special characters or non-ASCII characters in the filenames of the files after the encoding and move.

- **What did the automation pipeline look like—from receiving the file to final transfer?**
  Source->url encoding app job-> destination directory containing files with the url encoded filenames

- **How did you test the accuracy and completeness of the encoded filenames?**
  We used a regex to check if the encoded filenames contained alphanumeric characters or %

- **Was the encoding process idempotent? Could it be run multiple times without corrupting filenames?**

The nature of the app is such that the encoding is done only once during the transfer to the Poller source directory and so there is no scenario of double encoding. Hence the encoding process is idempotent. Because of this there was no filename corruption. If there was no need to run the encoding multiple times on the same file then there would be corruption but that didn’t apply to our case.

🧱 Architecture & Design

- **How did you structure the codebase? (Modular design? Configurable paths?)**

  The codebase was structured around a single main script, filename_url_encoder.py, which contained the core logic for both URL encoding and file transfer. While the design was not modular in terms of splitting functionality across multiple files or packages, the script was organized in a clean and logical way, separating concerns like argument parsing,encoding logic and file operations. The tool was configurable via command-line arguments, allowing users to specify:

  The source and destination directories for the file transfers.

  Whether or not to perform the URL encoding as part of the process.

  This made the tool flexible and adaptable to different environments or workflows without requiring any code changes. The project also included standard project files for CI/CD (Jenkinsfile, azure-pipelines-*.yml), documentation (docs/), and testing(test/), which contributed to maintainability and deployment readiness. While the core logic was contained in a single file, the use of command-line arguments and clear separation of logic within that file made the tool easy to use and extend.

- **Was the tool deployed as a script, web service, or part of a pipeline?**

  The tool was developed as a standalone Python script and committed to a shared repository within my team’s project space on BlackRock’s Azure DevOps environment.

While the script itself was not deployed as a web service, it was integrated into the company's standard CI/CD pipelines, which were maintained by other engineering teams. These pipelines automatically handled code quality checks, testing and deployment to pre-production environments. So while the tool was authored as a script, it was ultimately deployed and executed as part of a larger automated pipeline.

- **Did you implement logging or monitoring for the automation? If so, how?**
  We use an execution system called server_cron.pl which will automatically execute your python script using a predefined schedule. Then when it execute the python command, it will print out the log using a standard logger library/system that BLK implemented to all perl/java/python code. That logger will log the event every time it execute your program.

- **What version control or CI/CD processes did you follow for this project?**

  Since this was a first time project we create a new main repo and then create a branch repo called testing. I cloned this testing repo to my local computer environment via VSCode and then wrote the code including the test code and then gradually committed the changes to the remote version of the testing branch in Azure DevOps until we had all of the code ready to be deployed to preproduction. Before deploying we had to pass the quality and base pipelines and get approvals from leads during the pipeline runs before deploying the app to preproduction environments for testing.

- **What file systems or environments were involved in the transfers (e.g., internal shared drives, cloud storage)?**

  BlackRock Network drives, Poller source directory

- **Did the app have a CLI interface, UI, or configuration files?**

No it did not have any UIs but it did have the internal BlackRock CLI interface within the internal BlackRock servers where commands are put in to run this Python app periodically, also known as a job.

📊 Performance & Optimization

- How did you benchmark the “30% improvement in ingestion efficiency”?

- What was the baseline performance, and what bottlenecks existed before encoding?

  Before we create the URL encoding Python app, Pollers are used for file transfers only could work with filenames with only alphanumeric characters and % and not with the special characters. Which led to a lack of flexibility with naming and differentiating between crucial team files moved by the Poller from the BlackRock Network Drive

- Did encoding impact the size or readability of files?

  No the encoding did not impact the size or the readability of the files because the app only edited the filenames not the contents of the file. Neither did it enforce and storage limits on those files.

- How did you measure the success of the automation over time?
  We measured the success of the automation over time by first testing it with dummy source and location folders within the DEV and TST environments. Because we were going to periodically run the app as a job we created a cron server in both pre-production environments and test the job the cron server would on the app a couple of times before releasing to production. Once that was successful we released the app to the production environment and provided ways to manually disable the cron server runs the app job in case of any

incidents in other parts of the BlackRock production environment that could impact the running of the server.

💼 Business & Impact

- **Who were the users or teams impacted by the encoding app? Private Markets team were the primary users of this app as the encountered the Poller failing on special character filenames when working on their independent project.**

- **What specific ingestion system(s) was this tool preparing files for?**

  The specific ingestion system this tool was preparing files for was the BlackRock Poller system that takes action to perform file transfers and monitors those file transfers.

- **How did this tool reduce manual effort or improve data quality?**

  It saved the creators of those files any time they would have lead to use create unique filenames that were compatible with the BlackRock Poller.

- **Did this project affect SLAs or reduce processing errors?**

  This process totally eliminated errors encountered by the Poller when processing special character filenames.

- **Was this a one-off fix or a foundational tool used repeatedly?**

  It was a foundational tool being used repeated after the Poller continuously picks up files that are continually moved into its Poller source directory. That is why are URL encoder app had to be run

periodically ie as the job rather than once manually.

🛡️ Security & Compliance

- **Were there any compliance or security constraints when automating file transfers?**

  The compliance and security constraints we faced we mainly on eliminating any pieces of code that might expose the filenames, and make sure BlackRock copyright logo was present in all of the app files.

- **How did you ensure that sensitive filenames or metadata weren't exposed during encoding or transfer?**

  In order to ensure that no sensitive filenames or metadata were exposed during the encoding process we made sure to eliminate all log or print statements which involved outputting the filenames. Also we designed our test to track the number of files available after the url encoder encodes and moves the files from the source directory to the destination (Poller source directory). The made sure we didn't have to do the same names to confirm if the encoding and the moves were successful.

🧩 Collaboration & Communication

- **Who were the stakeholders? Did you receive any specific requests or feedback?**
  The stakeholders were members of a team in the Private Equity department whose main interest was to implement the app to run this url encoding solution. However the option of an archive folder to

store files with previously encoded filenames was one of their requests in the project implementation process.

- **How did you gather requirements for the app?**

  We gathered requirements for the app through the Outlook work email chain involving me, my engineering lead for my team, a peer engineer and stakeholders. Information relation to changing requirements and considerations were shared in the email changing, architecture diagrams to enhance understanding of the file transfer process was also shared in the email chain.

- **Did you demo or document the tool for others?**

  Yes we documented the app the implementation and workflow using Confluence wikis which is a required standard practice for engineers or developers after creating and before release software components to production.I had demo meetings with my engineering leads and stakeholder representatives to demonstrate how the cron server run the job on the app in pre-production.

- **Did you pair with a DevOps team or SRE for deployment?**

  We paired with the DevOps team to resolve any issues related to the running of the pipelines or how to locate and run the cron server in the pre-production environment.

🛠️ Troubleshooting & Challenges

- **What was the most difficult bug or unexpected behavior you encountered?**
  The most difficult or unexpected behavior I encountered was when I encountered errors in my unittesting saying the certain folders were not found and that's when I had to really understand how absolute and relative paths especially in Linux environments worked. That's because I had mistakenly used relative paths as arguments for some function calls of moving the files instead of absolute paths. I ended

up correcting then once I understood the difference between these types of paths.

- **How did you debug transfer failures or encoding issues?**

  By creating a bug fix branch of the repo, cloning that to my local machine, fixing the issue, committing to remote version of the repo in ADO and merging to the main branch for deployment to preproduction.

- **What fallback mechanisms or error handling did you implement?**

  I implemented error handling using try/except blocks to catch issues like missing files or encoding errors. When an error occurred, the script logged a clear message with file name and error type, which helped with debugging. In cases where a file couldn't be processed, the tool skipped it and continued with the rest, ensuring the process didn't halt entirely. While I didn't implement retries or backup paths, the logging provided enough visibility to manually address any issues.

🚀 Growth & Reflection

- **If you had more time, what would you improve or refactor in the app?**

  I would make a scalable version of it that would take in a list of original source directories and then perform the url encoding and move of the files in them. Also I would have created a component of the code that checks for if the app run has been manually killed for monitoring purposes.

- **Would you design it differently for larger-scale deployments?**
  For larger scale deployments, I would make a scalable version of it that would take in a list of original source directories and then perform the url encoding and move of the files from them.

- **What did this project teach you about file systems, encoding, or automation?**

  It taught me a lot about how paths work especially relative vs absolute paths. It taught me about how Pollers work and their limitations especially with filenames they can process which in turn affect which file moves they can make.

- **How did this project help you get comfortable with BlackRock's internal tools and engineering standards?**

  This project was actually a great introduction to how things work at BlackRock. I got hands-on with the internal development workflow- starting with cloning my repo locally and pushing changes back to Azure Devops. I also had to go through the process of requesting permissions to run the company's quality pipelines, which helped me understand the checks and standards in place before code can move forward. Once everything passed, I used an internal tool to release the code from ADO to the DEV environment. On top of that, I got familiar with the internal web interface used to configure cron jobs, which was how we schedule the Python app to run. Overall, it gave me a solid foundation in both the tooling and the expectations around code quality and deployment.

---

2. CPAdmin Page Enhancement

🧠 Interview Questions for CPAdmin Page Enhancement

Here's a full set, grouped by theme:

🔧 Technical Implementation

- What kind of configuration data was being stored or edited through the CPAdmin page?

  The kind of configuration data stored and edited on the CPAdmin page is the appserver configuration data to govern/ control who the appserver runs it's associated app once or periodically.

- How did you implement the JSBootstrap editable table? Was it a custom build or a library component?

  We built the editable table using a custom Node.js setup, but we did leverage the Bootstrap Table library for the core functionality. I created the table element directly in the Javascript file and gave it an ID - something like myTable. Then, using jQuery, I referenced it with $('#myTable') and initialized it using the .bootstrapTable() function. From there, I passed in all the necessary configuration - like column definitions, editable fields, and any custom features we heeded. So while the table itself was built into our app in a custom way, we used the Bootstrap Table plugin to handle the interactive and editable features.

- What backend framework or architecture did you use for the Java logic?

  My task was to build the backend logic that supported a dynamic, editable table on the frontend. This table allowed users to assign Kubernetes clusters to their applications and manage those assignments directly through the surface. To structure the backend, we used the Model-View-Controller architecture. The controller handled incoming requests from the frontend and passed them to the service layer, which contained the core business logic– such as validating cluster assignments and applying internal rules. The service layer then communicated with the data access layer, which handled all the database interactions. We used the Spring framework to support features like dependency injection, caching, and scheduling. We also integrated internal tools for configuration management and secure data access. This setup helped us keep the code modular and easy to maintain, which made testing and debugging much smoother. IN the end, the feature was successfully deployed to production and made it much easier for developers to manage their app server configuration through a clean, user- friendly interface.

- How was data validation handled on the frontend and backend?

- Did the page support real-time updates or require manual saving?

The page required manual saving using a save button at the top of the page and the saved changes were only written in the database representation of the table once approval was given, After approval is given the next reload of the page would contain the updated data in the table.

- How were the configuration updates stored—database, flat file, or other?

  They were the immediately stored in firm's large database.

- What challenges did you face while integrating frontend and backend?
  One significant challenge we encountered was dynamically populating dropdown options for certain columns in the editable without hardcoding the values directly into the frontend or backend. Initially, we attempted to use internal APIs available through BlackRocks's software artifactory to retrieve these values. However those APIs were unreliable at the time and frequently failed to return consistent results. As a workaround, we created a centralized repository in Azure DevOps where the dropdown options were stored in a structured file. Once this repository was deployed to BlackRock's preproduction environment, we were able to access the dropdown values programmatically. Specifically, we used environment variables to store the file paths or content, and a Python utility function would retrieve and parse the values at runtime. This solution allowed us to maintain flexibility and avoid hardcoding, while also ensuring that the dropdown data could be updated independently of the application code. It was a good example of adapting to infrastructure limitations while still delivering a scalable and maintainable solution.

🛠️ Systems/Architecture

- What was the role of this configuration data in the Kubernetes cluster assignment process?
  So currently, application runs on ACP platform does not have a central place to manage applications's configuration. In this case the assignment configuration. And CPAdmin is our existing configuration management system which exten the service to ACP platform. It help the owners in managing configuration of the application, serve as a trust of source for any automation services that utilize the configuration data, it also help as a trust of source of monitoring system in which owners can get an update on how/where their applications running on ACP platform.
- How did the CPAdmin page fit into the broader pipeline for scheduled app runs?
  The CPAdmin page takes in the configuration information for the server to control the scheduled running of the app. Some of these configuration information include startup command process pattern shutdown command sleep delay, memory limit, etc. This information is then sent to the actual server code that initiates commands based on the

configuration information to perform the scheduled runs of the app associated with the server.

- Were changes to the config data reflected immediately in downstream systems?

  Well not immediately . Only after some senior approvals were they reflected immediately in the downstream systems.

- How did you secure or permission access to the page?
  There are other teams responsible for this but I believe secure access was ensured by approvals from some production operations senior members to even grant access to page for a developer planning to use it.

📈 Impact & Efficiency

- What problems existed with the previous version of CPAdmin (or lack thereof)?

  There were no problems with the previous version fo the CPAdmin. We just needed to include a feature on the page that allows users to enter server configuration information for the internal apps they build and want to deploy to some Kubernetes clusters within BlackRock's cloud system.

- What improvements did your version deliver? (e.g., less manual editing, fewer errors, easier visibility?)

  Our improvements just added a new feature to the CPAdmin page that enable developers to specify configurations related to the deployment of their app to Kubernetes.

- How many users interacted with the page across the firm?
  About 2000 developers/engineers across the firm

🔒 Security & Access Control

- Did you implement role-based access for editing or viewing cluster configurations?

We used Boolean switches to determine which server names enabled user access to our new cluster assignment table feature of the page. In the backend we were able to access a list of server names from the preproduction and production environment and sent them over to the frontend code and the frontend code would determine if that cluster assignment section was visible based on whether the server name of the current page it was rendering was in this list of approved servers.

- How was data integrity ensured if multiple users were editing?
  Data integrity was maintained through an approval mechanism built into the system. When a user made changes and submitted them by clicking the save button, those updates entered a pending state and required approval before becoming active. During this period, other users were prevented from making additional edits to the same data. This ensured that only one version of the data could be modified and approved at a time, avoiding conflicts and preserving consistency across the system.

🔁 Collaboration & Deployment

- Who were the primary stakeholders for this tool?

  The primary stakeholders for this tool was the ACP team which managed the Aladdin Container Platform (ACP) which is the company's internal containerization framework built on top of the Cloud Native Platform, and their Kubernetes K8s layer which are both in turn built on top of the Microsoft Azure Cloud Ecosystem.

- How did you gather requirements or receive feedback?

  We gathered broader requirements and feedback via Outlook from the stakeholder team that needed the project and we go some minor feedback especially on the table's functionality on the CPAdmin page from the stakeholder team via teams group chat.

- How was the tool deployed—did it go through a CI/CD pipeline?

  Yes it went through azure devops quality and publish pipelines within which there were checks and tests in the pipeline runs before being released to the preproduction environments, usually DEV first.

- Did you write documentation or conduct user training?
  Yes I wrote confluence wiki documentation regarding the inclusion of this new table feature for server Kubernetes cluster assignment configuration and what kind of information can be entered into that table.

🧩 Edge Cases & Debugging

- What were some edge cases you had to account for (e.g., malformed configs, missing cluster mappings)?

  First edge case we considered was when the mapping object containing the table data was empty, then in that case we don't load anything into the bootstrap table. Another crucial edge cases was when cluster assignment modes did not have any corresponding values for the appropriate server site. In that case we don't display the mode and its corresponding version of the table data in the page.

- What was the most complex bug you encountered during development?
  The most complex bug i encountered during development was where I had tried to make three columns (region, distro and segment depend on each other) so distro depended on region values and segment values in turn depended on distro values. Specifically the problem was getting the right drop down values for the segment depending on the distro value and after weeks of hitting dead ends in researching a solution using AI tools like Copilot especially, I implemented the clever idea of assigning a function to the segment key of the table.

🚀 Growth & Reflection

- How did this project improve your full-stack skills?
  It really helped me to understand the implementation of MVC model of full-stack applications. More specifically it answered and made me understand what had always puzzled me about full-stack applications on how necessary data was transferred from storage or database to backend logic to frontend logic.
- If you were to redesign the config editing experience, what would you change?
  I would say if I had more time to complete the project, I would modularize the code a bit more that it was. Because I added new code to an already existing code that wasn't modularized, a better and faster way in my opinion was not to try to modularize my code integration which would mean modularizing the already existing code especially the frontend code which would not be very possible given the time constraints we had. Although I did have one separate js file for some of the frontend logic for my code integration but that was the only appropriate modularization that I felt could be implemented.
- What did this project teach you about internal tooling at scale?

  It taught me about how important data is and how tools used to receive and transfer data for the running of non-data related tools is crucial. This especially goes for teams like mine which is an observability team and it made sense that we were responsible for web pages like CPAdmin because a good amount of configuration data that is being entered and used to run the servers was data that we used to assess the health of those servers and the apps they run.

---

3. ServiceNow AI Inventory

🧱 System Design & Architecture

- What types of data models did you create in ServiceNow to represent AI/ML applications?

  To represent AI/ML application in ServiceNow, I worked with three key data models. First, I extended the existing Business Applicaition table by adding new fields that captured whether an application had AI/ML capabilities, along with related metadata such as model type or AI functionality. Second, I created a new table called the AI Use Case table. This was designed to store detailed information about each AI/Ml use case, including its purpose, data sources, and compliance attributes. Third I created Business Application- AI Use Case Mapping table to model the one-to-many relationship between business applications and their associated AI use cases. This allowed a single application to be linked to multiple AI/ML use cases in a structured and scalable way. These data models enabled centralized tracking, governance, and reporting of AI/ML integrations across the organization, and supported cross functional collaboration between engineering, compliance, and business teams.

- How did you decide on the schema for tracking AI capabilities and use cases?

  We were tasked with enabling centralized tracking and governance of AI/ML capabilities across business applications within the organization. To do this effectively, we needed to design a scalable and structured schema in ServiceNow that could support both technical and compliance needs. Our goal was to created a data model that could capture relevant AI/ML information support one-to-many relationships between applications and use cases, and be easily maintained over time. We started by reviewing the existing Business Application table and decided to extend it with new fields that captured AI-specific attributes — such as whether the application used AI/ML, the type of model, and any compliance considerations. To avoid overloading that table with too much detail, we created a separate AI Use Case table to store more granular information about each use cases, including its purpose, data sources and risk classification . We then introduced a mapping table to establish a one-to-many relationship between business applications and their associated AI use cases. Throughout the process, we collaborated closely —- myself, my team lead, and key stakeholders— to ensure the schema aligned with both technical requirements and governance standards. The result was a flexible and scalable data model that enabled effective tracking, compliance reporting, and cross-team collaboration around AI/ML integrations.

- Did the system have relationships between applications, capabilities, departments, or risk levels?

  The system did include relationships between business applications and AI use cases, which were modeled through a mapping table to support a one-to-many relationship. However there were no formal or structured relationships defined between applications and specific capabilities–those were not tracked as distinct entities in the schema. In terms of departments, while users from teams across the firm could interact with the system, the AI Approval and AI Governance groups were the primary stakeholders responsible for managing and approving the data entered into the AI-related fields. As for risk levels, there was a dedicated field in the AI Use Case table to capture the relevant AI/ML risk associated with each use case. However, this was a standalone field and not part of the relational model - meaning it wasn't linked to a separated risk table or taxonomy.

- How scalable or flexible is the data model if new types of AI tools are introduced?

  The data model is quite scalable and flexible. For example, the name of the AI tool or model used is captured as a field on the Business Application form, and the dropdown options for that field can be easily updated to include new tools or models as they are introduced. Additionally, the AI Use Case table is designed to accommodate an unlimited number of entries, so new capabilities or use cases can be added without any structural changes. These use cases can then be linked to one or more relevant Business Applications through the mapping table which supports a one-to-many relationship This design allows the model to evolve alongside the organization's AI landscape, without requiring major schema changes. It also supports future enhancements like tagging categorization, or integration with risk scoring systems if needed

---

⚙️ Implementation & Automation

- How did you implement the automated data imports? (e.g., ServiceNow Scheduled Jobs, IntegrationHub, REST APIs?)
  We used ServiceNow's Transform Map feature to implement the data imports for both the AI Use Case table and the Business Application- AI Use Mapping table. This allowed us to import structured data—typically from Excel files where each row represented a record and each column mapped to a specific field in the target table. The transform Map enabled us to define field mappings, apply transformation logic where needed, and ensure that new entries were created with the appropriate data in each related form. This approach was efficient, required minimal custom scripting, and ensured consistency across imported records. While we didn't use Integration Hub or REST APIs for this particular use case, the Transform Map approach was well-suited for bulk onboarding of

structured dat and could easily be extended with Scheduled Jobs if we needed to automate recurring imports in the future.

- What sources did you pull the inventory data from (e.g., CMDB, Excel, internal databases)?

  I pulled the inventory data from the Microsoft Excel files. These files contained structured information about AI/ML use cases, which I then imported into ServiceNow using the Transform Map feature. Each row in the Excel file represented a record, and the columns were mapped to the corresponding fields in the AI Use Case and mapping tables. This approach allowed for efficient bulk data entry and ensured consistency across the imported records.

- What scripting or logic did you use to clean and validate the imported data?

  In this case, we didn't use any scripting or in-platform logic within ServiceNow to clean or validate the data. All data validation and cleanup were handled prior to the Excel upload, typically by the data owners or stakeholders responsible for preparing the inventory.ServiceNow's Transform Map feature to import it into the appropriate tables. Since the data was already cleaned, the import process focused on mapping fields correctly rather than applying additional validation logic.

- How often did the automation run, and what happened if an import failed?

  There were no scheduled import runs for this process. Instead, data was imported manually – whenever a user was ready, they would upload the Excel file directly into ServiceNow. In the event of an import failure, ServiceNow's built-in import set tracking handled error reporting. The platform automatically logs each import attempt and provides detailed information about its success or failure, including any errors encountered during the process. This allowed users or administrators to quickly identify and resolve issues without needing custom error-handling scripts.

📊 Tracking & Reporting

- What kinds of compliance or regulatory frameworks did the tool help address (e.g., internal AI use policy, AI ethics reviews)?

- How did teams access or query the data once it was populated?

- Did you create ServiceNow reports or dashboards for visualization?

🛡️ Security & Governance

- Were there role-based access controls on who could view or edit the AI inventory?
- How did this system help detect unauthorized or untracked AI tool usage?
- Did this tool integrate with any risk scoring or approval workflows?

📈 Business Impact

- What were the limitations of the previous tracking method, and how did this new system improve things?
- How did your solution impact audit readiness, compliance efforts, or internal transparency?
- Who were the key stakeholders (e.g., Security, Compliance, Engineering), and what feedback did you receive?

🧩 Challenges & Troubleshooting

- What was the biggest challenge in mapping real-world AI tools to standardized entries in the ServiceNow system?
- How did you deal with inconsistencies or outdated records during imports?

🚀 Growth & Reflection

- How did this project increase your understanding of ServiceNow as a platform?

- What did you learn about governance or operationalizing AI in a large organization?
- If you were to extend this project, what feature would you add (e.g., AI usage approval workflows, risk dashboards)?

---

4. Cusip Fund Retriever (Job Training Project)

💻 Technical Foundations

- How did you extract fund data related to CUSIPs? What was the structure of the SQL queries?

  In one of my recent projects, I was tasked with extracting fund-related data based on a list of CUSIPs for a specific reporting date. To approach this, I first clarified the data sources involved and ensured I understood the relationships between funds, portfolios, and securities. I designed a SQL query that joined three key datasets: one containing fund- CUSIP mappings, another with portfolio metadata, and third with security master details. I used subqueries to filter the data early — specifically by pos_date and the target CUSIPs — to improve performance and readability. The structure of the query allowed me to:

  Filter only the relevant CUSIPs and date from the based table. Join that with portfolio information to get human-readable names. Join the security master to enrich the data with tickers and descriptions. I also made sure the query was modular and easy to maintain by using parametrized inputs and abstracting table names through a database handler. This made the coded more reusable and secure. The final result was a clean, ordered dataset showing portfolio names, Cusips, tickers, and descriptions — ready for downstream reporting or analysis.

- Did you face any challenges with data formatting or query performance?

  In terms of data formatting, we didn't face significant challenges. The input was a text file containing CUSIP strings, which I parsed into a list in Python. BlackRock provided internal library functions that handled the formatting of that list into a structure suitable for SQL queries, so I didn't need to manually format the data for querying. However, one of the more complex challenges came when retrieving NAV prices from the query results. The returned data was deeply nested in a combination of class-based objects, dictionaries, and lists. I had to carefully inspect the structure and types of these nested elements to correctly extract the NAV values for each CUSIPs. On the query performance side, the main issue wasn't speed but access. I initially encountered permission issues when trying to submit SQL queries form Aqua Data Studio, which was external to BlackRock's database environment. I worked with the database team to resolve those access restrictions so I could test and validate the queries effectively.

- Which Python libraries did you use for email automation?
  For email automation, I used a combination for internal and standard Python libraries. Specifically, I used BlackRock's internal SMTP library to handle email delivery within the firm's infrastructure. For constructing the email content, I used Python's built-in email.mime.multipart and email.mime.text modules to format the message body and attachments. Additionally, I used BlackRock's internal user and SSO libraries to authenticate and manage user-specific email routing securely. This setup allowed the script to send formatted query results to the appropriate recipients in a secure and standardized way.
- How did you handle errors — such as missing data, bad connections, or failed sends?
  For errors that were external to my code–such as database permission issues or connectivity problems — I followed BlackRock's internal support process by submitting JIRA requests to the appropriate teams, like the database team for query access issues. In addition to formal ticketing, I also leveraged Microsoft Teams chats and group channels to communicate directly with individuals who had domain-specific knowledge. This helped accelerate troubleshooting and often provided valuable context that wasn't captured in documentation. For errors within the code itself — such as missing data or unexpected query results — I implemented validation checks to ensure the script could handle those gracefully. This helped prevent failed email sends and ensured the automation remained stable and reliable.
- What was the format of the output (CSV, HTML, PDF)? How did you ensure readability for the recipients?

  The output was formatted as an HTML table embedded directly in the body of the email. This format was chosen to ensure that the data — particularly NAV prices and associated fun information — was easy to read and interpret at a glance. The HTML table included clear column headers, consistent alignment, and basic styling to improve visual structure. This made it easier for recipients to quickly scan and understand the financial data without needing to open attachments or manually reformat anything. It was especially helpful for stakeholders who needed to review NAV values across multiple CUSIPs efficiently.

🏗️ Architecture & Deployment

- Was the tool designed to run manually or on a schedule?

  The tool was designed to be run manually using the Python command line. It was intended for on-demand use, so whenever someone had a specific list of CUSIPs and a target date, they could run the script themselves. It accepted a few command-line arguments — like the path to the file containing the CUSIPs, the date for which they wanted the fund data, and the email address where the results should be sent.This made it flexible and easy to use without needing to modify the code each time.

- Did you integrate the tool into any existing reporting pipeline?

  No the tool wasn't integrated into an existing reporting pipeline. It was designed to be run manually and used primarily for on-demand data retrieval and email delivery. The focus was on flexibility and ease of use, allowing users to run it whenever they needed specific fund data tied to CUSIPs

- How was this tool deployed or run within BlackRock systems? Did you follow any specific CI/CD or packaging processes?
  The tool was designed to be run manually, but I did follow BlackRock's internal CI/CD practices for code quality and deployment. I set up an Azure Repos pipeline for the project using and existing YAML template, which handled tasks like dependency installation and test execution. Reviewers typically tested the tool either by cloning the repository locally or by deploying it to BlackRock's DEV pre-production environment and running it there. This allowed them to validate the functionality in a controlled setting. I was able to pass the review process successfully, which confirmed that the tool met the required standards for production-readiness.

🧠 Learning & Familiarization

- What internal tools or libraries did you have to learn as part of this project?

- How did this project help you understand BlackRock's data environment and reporting workflows?

- What was your biggest technical or process takeaway from this project?

📈 Business Context

- What is a CUSIP, and why is it important in financial data analysis?

  A CUSIP stands for Committee on Uniform Securities Identification Procedures. It is a unique 9-character alphanumeric code used to identify securities such as stocks, bonds, mutual funds, and other instruments in the US and Canadian markets.

- How was the data retrieved used by other teams or departments?
- It is retrieved from BlackRock's large internal database either by direct querying via AquaDataStudio or using APIs or programming language libraries developed by BlackRock developers.

- Who would have used the emailed reports (e.g., analysts, portfolio managers)? Mostly analysts, but some portfolio managers do that too.

🧩 Challenges & Debugging

- What did you do when your SQL query didn't return the expected data?

- How did you test the automation end-to-end?

- Did you have to sanitize or anonymize any of the data?

🚀 Growth & Reflection

- What would you improve if this were a full production tool?

- How did this experience prepare you for your next project (e.g., Filename Encoder or CPAdmin)?

- If you had to extend the tool, what feature would you add?

🔹 Technology Support (May 2023 – April 2024)

1. VDI Imaging

- What tools or scripts did you use for imaging and provisioning the VDIs?

- What challenges did you face when setting up 100+ devices?

- Did you create any documentation or SOPs for the process?

2. Troubleshooting

- Describe a difficult hardware/software issue you diagnosed and resolved.
- How did you balance multiple support requests? Any ticketing tools or prioritization logic?
- How did you escalate issues you couldn’t solve?

3. Windows 11 & Microsoft Copilot Testing

- What was your process for testing the integration of new OS features?
- Did you report bugs or give structured feedback? To whom?
- How did Copilot integration affect performance or user workflows?

🏆 Award: 1st Place – Internal Data Science Competition (Team Markov Pain)

- What was the project topic and how did your team approach it?
- What technical tools or ML models did you use?
- What was your role specifically—data engineering, modeling, presenting?
- What feedback did you receive that helped you win?