Development of Authenticated Clients and Applications for ICICLE CI Services

Final Report for the REHS Program, June - August, 2022

Sahil Samar
Del Norte High School,
San Diego, CA, USA
Email: sahilsamar031@gmail.com

Mia Chen
Westview High School,
San Diego, CA, USA
Email: mialunachen@gmail.com

Jack Karpinski
San Diego High School,
San Diego, CA, USA
Email: jackadoo4@gmail.com

Michael Ray
JSerra Catholic High School,
San Juan Capistrano, CA, USA
Email: michael.ray@jserra.org

Archita Sarin
Mission San Jose High School,
Fremont, CA, USA
Email: archita.sarin@gmail.com

Christian Garcia (mentor)
Texas Advanced Computing Center,
Austin, TX, USA
Email: cgarcia@tacc.utexas.edu

Matthew Lange (mentor)
IC-FOODS,
Davis, CA, USA
Email: matthew@icfoods.org

Joe Stubbs (mentor)
Texas Advanced Computing Center,
Austin, TX, USA
Email: jstubbs@tacc.utexas.edu

Mary Thomas (mentor)
San Diego Supercomputer Center
La Jolla, CA, USA
Email: mpthomas@ucsd.edu

***Abstract*—The Artificial Intelligence (AI) institute for Intelligent Cyberinfrastructure with Computational Learning in the Environment (ICICLE) is funded by the NSF to build the next generation of Cyberinfrastructure to render AI more accessible to everyone and drive its further democratization in the larger society. We describe our efforts to develop Jupyter Notebooks and Python command line clients that would access these ICICLE resources and services using ICICLE authentication mechanisms. To connect our clients, we used Tapis, which is a framework that supports computational research to enable scientists to access, utilize, and manage multi-institution resources and services. We used Neo4j to organize data into a knowledge graph (KG). We then hosted the KG on a Tapis Pod, which offers persistent data storage with a template made specifically for Neo4j KGs. In order to demonstrate the capabilities of our software, we developed several clients: Jupyter notebooks authentication, Neural Networks (NN) notebook, and command line applications that provide a convenient frontend to the Tapis API. In addition, we developed a data processing notebook that can manipulate KGs on the Tapis servers, including creations of a KG, data upload and modification. In this report we present the software architecture, design and approach, the successfulness of our client software, and future work.**

1. Introduction

The *AI institute for Intelligent Cyberinfrastructure with Computational Learning in the Environment* (ICICLE) program aims to develop intelligent cyberinfrastructure with transparent and high-performance execution on diverse and heterogeneous environments, as well as to advance plug-and-play AI that is easy to use by scientists across a wide range of domains, promoting the democratization of AI. [1] [2] The deep AI infrastructure that ICICLE plans to utilize to sift through data relies on knowledge graphs (KGs) in which information is stored in a graph database that uses a graph-structured data model to represent a network of entities and the relationships between them. [3]

Finding a way to make KGs easily accessible and functional on high performance computing (HPC) systems is an important step in helping to democratize HPC. Thus a large focus of this project was contributing to the body of knowledge needed for hosting live, dynamic, and interactive services that interface with HPC systems hosting KGs for ICICLE based resources and services.

This report describes the results and lessons learned from the project we worked on as part of the San Diego Supercomputer Center (SDSC) [4] 2022 Research Experience for High School Students (REHS) program. [5],

During the course of the project, we learned about many technologies, in the following core areas: (1) Programming and Development - Bash scripting, Docker containers, Jupyter Notebooks, JupyterLab, Anaconda Environments, Machine learning, Neo4j Cypher Query Language, CLI Development, Github; (2) Tapis – instantiating Tapis Pods on remote HPC systems, exercising the TapisAPI in notebooks/python scripts; using Tapis to remotely run singularity/docker containers, using Tapis as an alternative to SSH for file transfer; and (3) HPC Systems – connecting to and running jobs on Expanse and Stampede, How to use the Slurm batch scheduler, How to navigate the Expanse storage system.

In the following sections, we describe the architecture and components of the system as follows: Section 2 describes the architecture of the Hello ICICLE system and

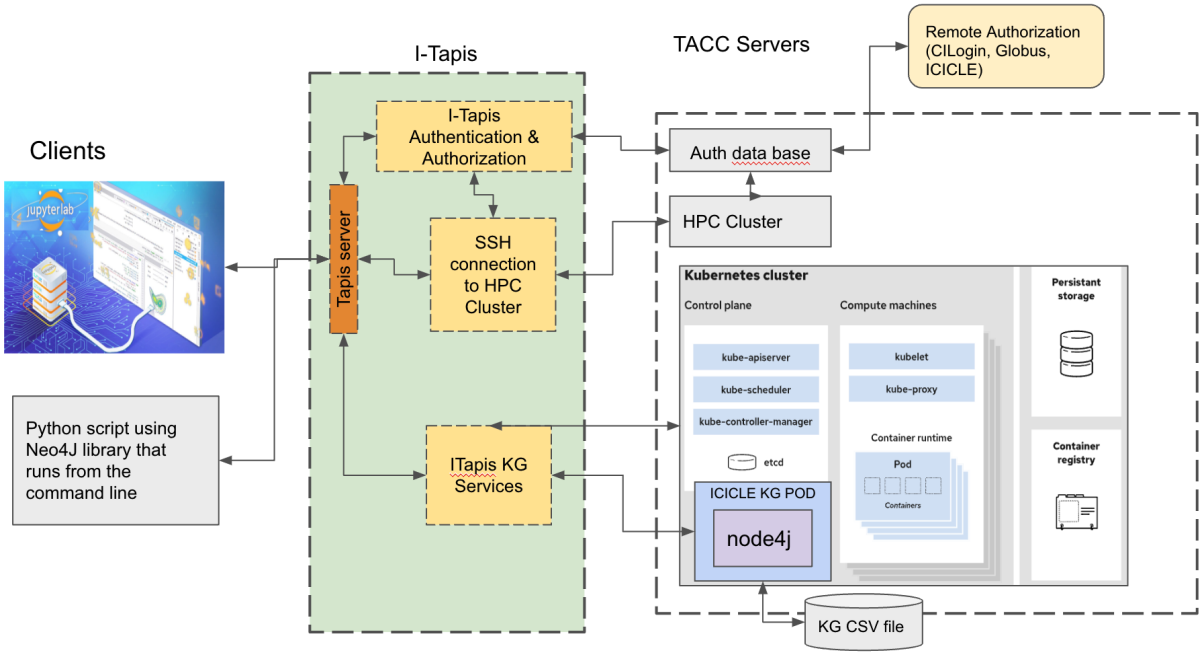

Figure 1. Hello ICICLE 3-tier middleware architecture, consisting of client, middleware and backend resources.

the various components of the sofware stack; Section 3 covers the methodology we used for creating Knowledge Graphs, developing Jupyter Notebook and Python Clients, and testing applications such as Neural Networks and Machine Learning with KGs; Section 4 summarizes the results of our project; finally, Section 5 provides a conclusion as well as an outline of further research to be done with the ICICLE project.

2. Architecture of the Hello ICICLE System

There are many ways to access HPC systems, but they are generally difficult and require a specialized understanding of computer science and system architecture. Many scientists and researchers who want to analyze large data sets do not have this expertise, and cannot utilize advanced computing resources to their fullest. As a result, there is a high demand for simpler, more user friendly interfaces with HPC systems. Many of these architectures are based on a 3-tier middleware architecture [6], which consist of a client, middleware and backend resources. The middleware layer often includes the Web app frameworks, and local databases and services. To meet this demand, we developed a system we initially referred to as *Hello ICICLE* that allows clients to create, edit, and query remote ICICLE services (see Figure 1). The system supports Jupyter Notebook and command line interface (CLI) clients, that use the Tapis Framework [7] to authenticate and run jobs on remote services and to access Neo4J generated Knowledge graphs [8]. The various components used in the Hello ICICLE system, what we learned about them, and how they are used are described in the sections below.

2.1. Hello ICICLE Clients

We designed two types of clients that can be run on the users local laptop, workstation or HPC system: Jupyter Notebooks and command line Python apps (CLI). Jupyter Notebooks are popular Web based applications that compartmentalize and simplify authentication and interfacing to jobs and services hosted on these HPC systems. [9], [10] We initially ran and tested the notebooks provided by the Tapis project. To create an ICICLE relevant test case, we focused on the deployment and use of Neo4j knowledge graph databases using pods hosted on Kubernetes clusters that are hosted via Tapis and run on TACC's *Cyclone* supercomputer. Figure 2 shows how user requests (e.g. the Jupyter Notebook) are routed to the appropriate Pod or service hosted in the Kubernetes cluster(green).

Many HPC systems users know how to run programs using command line interface (CLI) applications. Consequently, we developed a set of CLI applications that emulate the Jupyter Notebook KG functions. Results and outcomes of our clients are discussed in the Results section (§ 4).

2.2. Tapis Computing Platform

Tapis authentication is a multi-layered operation that constitutes the most difficult portion of this process. Figure 3 shows the backend architecture of the Tapis system, as well as the Tapis API. [11] In our project, we heavily leveraged the use of the Tapis API in order to create a Neo4j template database hosted on Tapis systems. User requests are routed through the Tenant Router, and authentication is done using Tokens and the Security Kernel; then, a user's request for a

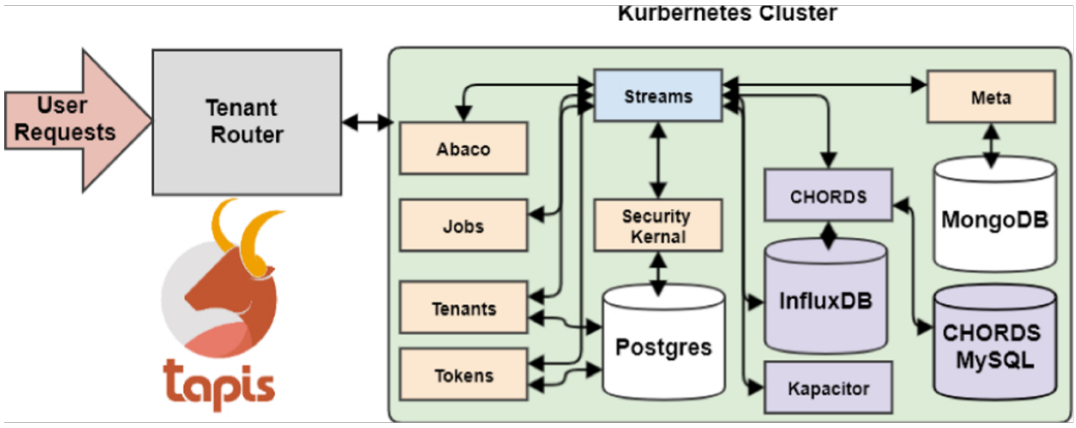

Figure 2. Figure shows how user requests (e.g. Jupyter Notebook) are routed to the appropriate Pod service hosted in the Kubernetes cluster(green). [11]

job is able to go through. The workflow for the end user is much simpler as these layers are handled by the Tapis API.

2.2.1. Tapis Authentication and Authorization. Tapis has provided a tutorial called "Hello, Tapis" in order to introduce researchers to the Tapis systems. [1] We have compiled a Jupyter Notebook [5] in which we simplified and elaborated parts of this tutorial and annotated each step with clear instructions.

For authentication and authorization via Tapis, we leveraged TACC accounts. The Jupyter Notebook starts with a request for input of the TACC username and password which will then be stored as variables in the notebook. Then, Tapis is imported and a python Tapis client is created for the user. After a call to the Tokens API, the Notebook prints out a string that includes a token that the user needs to set to the JWT variable in the shell using the command:

```
export JWT=<access token starting with ey...>
```

Then the user can run the command:

```
t.authenticator.get_userinfo()$
```

This command should return the info that is part of the user's TACC account. The next step is to access a new system. To do this you need to create a system variable which in our Notebook is used to access Stampede2 by someone with an allocation on it. To run it, edit cell 5 in the Notebook by replacing <stampede_username>in the code with your stampede username. Then after running a createSystem command, you have connected to the system. and there are some commands to check if you have connected correctly.

2.2.2. Using Tapis Pods.

There are a multitude of possibilities of what can be done once authenticated. Accessing a TACC machine like Stampede or Jetstream requires that HPC system to be registered with the TACC account. We cover this process in our "Tapis-Notebook" notebook. Most of our project, however, was focused on Tapis Pods. Tapis Pods are a Tapis service which allow for easy deployment and use of databases. They are able to import and export data from live databases. Tapis has an easy to use template to create a pod that will host Neo4J knowledge graphs.

Each Tapis pod hosts one knowledge graph. These are stored on TACC systems using Kubernetes. The creation of these Pods is simplified through the Tapipy library. It is simply done through this method of the tapis object, "t":

```
t.pods.create_pod(pod_id=pod_id,
    pod_template=pod_template,
    description=pod_description)
```

Each of the parameters should be defined prior to using them. The "pod_id" is a string that will be used to reference the pod. The "pod_template" should describe what is going into the pod. The pod_template that we used was a neo4j template, which is specified simply through assigning the parameter a string value of "neo4j". The third parameter provides some information about the pod.

2.2.3. Tapis Generated User Credentials. The next section describes the steps needed to create user credentials, which are used to access the pods. For example, each of the pods (described in Section 2.2.2) has a specific set of permissions, and one can have the same permissions as the owner with the "ADMIN" level. These permissions can be set using our Tapis console application or can be done through a few simple commands, which are also covered in the "tapis_auth" notebook. When pods are created, they have preset credentials associated with them. These credentials can easily be accessed from a TACC user's python Tapis object through the following code:

```
username,password=t.pods.get_pod_credentials
(pod_id = pod_id).user_username,t.pods.
        get_pod_credentials(pod_id=pod_id)
        .user_password
```

where "t" is the Tapis object. It is important to not print out these passwords, as it is currently a security threat since once logged in, the credentials can be accessed, but those credentials can be used by members without any other access to the pod. The current best practice is to do as done in the code snippet, where the credentials are stored in aliases that are referenced when the user needs to login

to a pod. Currently, the best supported and easiest way to create a template using pods is the Neo4j template. Once a Neo4j pod is created and the permissions have been set, the authentication flow is as follows:

1) Using python, create a Tapis object.
2) Login to TACC.
3) Get the ID of the pod you want to access.
4) Get the credentials of that pod.
5) Pass in those credentials to a Neo4j graph object.
6) Use the Py2Neo client package to generate the graph.

An example for generating the graph is shown below:

```
graph = Graph(f"bolt+ssc://{pod_id}.
pods.icicle.develop.tapis.io:443",
auth=(username, password), secure=True, verify=True)
```

In the above example, "icicle.develop.tapis.io" is the base url that was used to create the pod. The base url is specified in the creation of the Tapis object, like so:

```
t = Tapis(base\_url =
"https://icicle.develop.tapis.io",
username = username, password = getpass('password'))
```

After the Pod is completely registered and all of the above steps have been followed, authentication can be largely streamlined using the resources that we have made such as our console applications and Jupyter Notebooks. All of these resources essentially just require a TACC login (as of now), and handle all of the further authentication described in this section in the backend.

2.3. Neo4j Database Service

Neo4j is a graph database service which allows users to create complex visualizations of data by using node edges and relationships instead of columns and rows to arrange data. Each node holds data, and connects to one or more nodes through relationships. Both nodes and relationships can possess unique properties to describe the data. The Neo4j article, "From Graph to Knowledge Graph: How a Graph Becomes a Knowledge Graph", provides a simple definition of a KG to be "an interconnected dataset enriched with semantics so we can reason about the underlying data and use it confidently for complex decision-making." [12]

Neo4j allows for flexibility in storing relationship driven data. Where relationships or classifications play a significant role, Neo4j proves much simpler and more efficient than other services, such as SQL. Throughout our project, a main testing focus was using python and Tapis to manage Neo4j knowledge graphs remotely on Stampede.

3. Methodology

In order to test the architecture, we had to develop software and clients to access the different services that are part of the overall architecture (see Figure 1). These are described below.

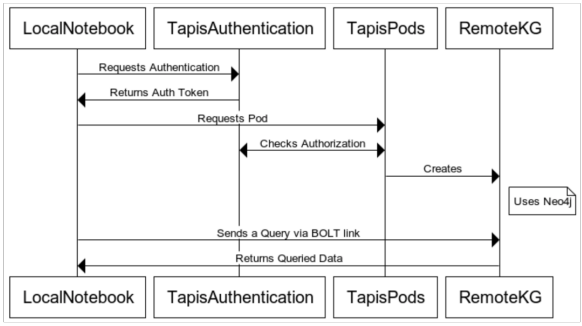

Figure 3. Sequence diagram showing access steps needed to authenticate the Jupyter Notebook client to the Tapis authentication system

3.1. Client Authentication Processes

The first step was the authentication. That only required a TACC username and password. However, there were multiple steps to follow to get the knowledge graphs to run on the Tapis Pods. In order to understand the authentication process that we went through to access our knowledge graphs, we created sequence diagrams of the Tapis architecture. These diagrams contained the steps that connected our laptop to the pods where the notebooks were run. Figure 3 shows the steps needed to authenticate the Jupyter Notebook client to the Tapis authentication system.

Once Tapis authenticates the TACC username and password from the client notebook, it returns a token that the user uses to request a pod. The software checks the authentication and if everything properly executes, it creates a pod on which the knowledge graph is remotely hosted. The KG runs on the pod using Neo4J. To query the data, the computer sends a query through the BOLT link and the remote KG returns the output.

3.2. Creating Knowledge Graphs

We initially used a Neo4j service to get acquainted with creating knowledge graphs. Using cypher, we each created our own knowledge graphs and then worked together on a larger set of data to efficiently host it on ICICLE systems. It was integral that we had a larger set of data in order to confirm that hosting the KG would be easy for scientists looking to just test their data. We found a way to write, query, and edit the KGs with Jupyter Notebooks using a Neo4j python package called "py2neo". We then automated the creation of knowledge graphs from .csv files via the LOAD CSV cypher command in a Jupyter Notebook. In the final product of our software, we exclusively used the Tapis pods service to create our KGs. After authentication on TACC, all that was left is to run the KGs. We created Jupyter Notebook clients to efficiently host remote KGs.

3.3. Accessing and Hosting Knowledge Graphs

As part of our project, we investigated several mechanisms for hosting KGs for public consumption. These included locally hosted graphs on our laptops, using the Neo4J browser and desktop apps, and the AuraDB and Bolt services. As we tested these methods, we determined that Kubernetes Pods (hosted on Tapis) were the optimal solution for our experiments. These methods are described below.

3.3.1. Local Host. Our project primarily used Neo4j's localhost service for testing purposes, which was useful given its simplicity and speed. While localhost services cannot operate on remote computers or be shared between team members, for much of the project it allowed us to quickly test cypher queries and programs without having to worry about networking.

3.3.2. Neo4j Browser and Desktop Apps. The Neo4j browser can be used for lightweight visualization of knowledge graphs. The Neo4j browser can be used to connect to a Tapis Pod with a Neo4j template. The credentials for authentication for this connection can be found using the same methods as described in the previous section. The Neo4j browser can be accessed through the Neo4j Desktop app. From there, a new connection can be created; the checkbox for using an encrypted connection must be ticked here to authenticate. Then, the credentials for the Pod simply have to be entered and the Neo4j Browser can open the project. Cypher queries can be visualized from here. The main drawback was that the browser and desktop apps were unable to host the KGs themselves.

3.3.3. AuraDB Hosting Service. AuraDB is Neo4j's own cloud service for hosting knowledge graphs. While it is useful for people who don't want to go to the trouble of manually setting up a bolt service, it isn't persistent, and not easily scalable due to the additional costs of hosting more than one knowledge graph. As such, for large scale or multi-project data hosting, AuraDB is not the ideal option. As such, while we initially used AuraDB for our database hosting, we quickly phased it out for other methods.

3.3.4. Bolt Portforwarding. After trying and realizing the limitations of cloud based solutions, Neo4j's Bolt service was our next option. We used our own laptops as servers, and exposed a Bolt URL for public access. While it did provide a cost free way to host our knowledge graphs, setup was difficult and unreliable. In the few instances we managed to set up bolt services, they were unstable (dropping connections, service errors), and not easily scalable, since each database had to be manually configured (if no other software is used). This made Bolt unsuitable for our intended use.

3.3.5. Tapis Pods. Tapis pods were our final solution, and fixed most of the aforementioned issues with other graph hosting services. As shown in Section II, after authenticating with the Tapis service, the user can create, or access a graph database in a Kubernetes cluster. Since the entire database creation and querying backend was streamlined by TACC, it takes minimal python code and minutes to accomplish the same as with hours configuring bolt. Thanks to its user friendliness and reliability, Tapis pods became the primary hosting service for our project.

3.4. Developing the Jupyter Notebook Clients

A key goal of our project was to create Jupyter Notebook Clients that allowed anyone to easily access Stampede2 and other Tapis supported HPC's. We wanted the Notebooks to be able to run from a local computer (laptop, workstation) and access remote HPC systems, so we used the Tapis Authentication system. Some of these Notebooks are a slight variation on the "Hello Tapis" tutorial. [13] Others allow people to access and visualize KGs using the py2neo, neo4j, and neo4jupyter python modules. These notebooks can be found in the project GitHub repository. [14]

To create these Notebooks as a team, we hosted them on our Github repository. [14] We used Visual Studio Code and Github Desktop to write and push edits to our Notebooks. For testing, we ran our Jupyter Notebooks using the browser, Visual Studio Code, and HPC environments.

All of this effort was in order to make HPC resources more accessible for computational scientists without needing to go through all the training that is currently required to use them.

3.5. Developing Python Clients

Not only did we create Jupyter Notebooks but we also made python clients that can be run from the command line. These clients access the KGs hosted on the TACC system using a few simple inputs. It does require your TACC username and passcode for authentication. It shows you all the pods that you have access to and asks which you'd like to use. It then allows you to easily access, create, delete, change permissions, get information, and query the knowledge graph without needing to know Cypher (Neo4j's query language).

4. Results

In the next sections we describe the projects that were developed by this project. Some of the projects were done in collaboration by the entire group, while some were done by individuals; but we were always helping each other with ideas and solutions. All of the software created by the students can be found in the project GitHub repository. [14]. In addition, some of our work (ICIConsole and TapisCL-ICICLE) will be included in the first ICICLE Software Components Catalog (VC3), which released around April 2023. [15] Those components can be found in the VC3 release github repository. [14]. Below, we highlight several of our projects.

4.1. Jupyter Notebook Clients

We created various Jupyter Notebooks that displayed important components in Authentication and Authorization, as well as setting up services like a Neo4j Knowledge Graph on a Tapis Pod. We explore how to populate blank templates of knowledge graphs, and went on to develop notebooks as clients to these knowledge graphs.

4.1.1. Tapis Authentication Notebook. We compiled authentication to Tapis as well as creation and authorization for Tapis Pods in this notebook. It forms the basis for many of our other applications.

4.1.2. Data Loading Notebook. In many cases, there is pre-existing data that must be brought into a knowledge graph. We showed how this can be done in our Data Loading notebook, where we displayed CSV data being loaded into a blank knowledge graph. The main requisite is having a link where the CSV is stored; we used Github for this. Then, we simply pass in the link to the Cypher function

`LOAD CSV WITH HEADERS`

and specify columns to node types using the

`MERGE`

keyword. We also did some scripting to automate the creation of relationships from CSV data.

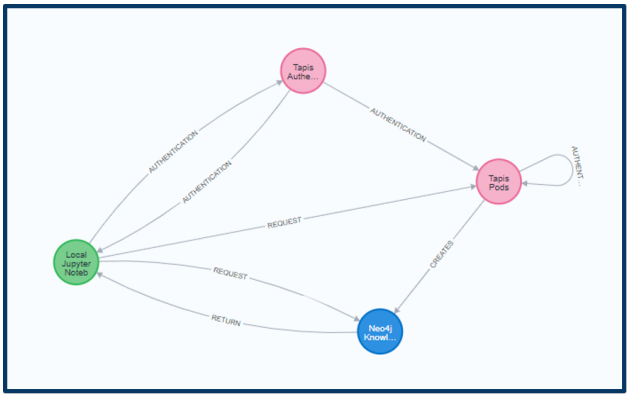

Figure 4. Figure kg-graph-visualization.png

4.1.3. Knowledge Graph Visualization of Architecture. This Knowledge Graph is an alternative visualization to our architecture diagram. It contains nodes representing the Tapis Authentication service, the Tapis Pods service, the Neo4j Knowledge Graph, and the Local Jupyter Notebook that queried the knowledge graph. It also contained relationships that displayed how the nodes interacted with each other. The goal of this project was to show how we could successfully host Neo4j Knowledge Graphs on Tapis Pods, and authenticate to them using local Jupyter Notebooks - we created a notebook for this so that the user could experience the point of view from the "Local Jupyter Notebook" node.

Figure 4 shows the different nodes and relationships in the KG.

4.2. Command-line Interfaces (CLIs)

We developed Command-line Interfaces (CLIs) which will allow users to connect to HPC systems using a simple terminal/shell application. The first step to any communication with a TACC system (cluster, Tapis service, etc.) is authentication. The prerequisite to this is that one must have an account registered with TACC already; this can be easily checked using the online TACC portal. Our console applications handle authentication to Tapis with the user's TACC account. We demonstrate the authentication process in the "tapis_auth" notebook described in Section 4.1.1; our CLIs simply transported that functionality to the console. We developed two separate CLIs. One had the goal of streamlining Tapis functions such as creation of pods (TapisCLICICLE), while the other had the goal of easily working with data in a Neo4j pod (ICICONSOLE). As other institutions begin using Tapis services, our applications will also be to authenticate to them.

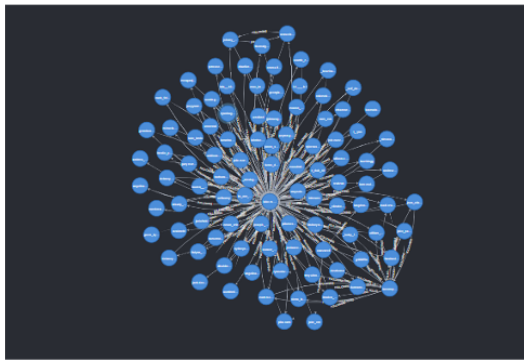

Figure 5. Instagram Web Scraper Knowledge Graph.

4.3. Application Examples

4.3.1. Instagram Web Scraper Knowledge Graph. We also developed an application to gather follower data from Instagram and feed that information into a Neo4j pod hosted on the TACC Tapis service. Since every user's data was stored in a separate JSon file, it offered an opportunity to explore different ways of loading information into Neo4j graphs. After uploading the data to the pod, we were able to visualize and query individual users and their relationships on the Tapis pod shown in Figure 5.

4.3.2. Neural Network Asteroid Classification. In order to demonstrate the capabilities and flexibility of Neo4j KG system we developed, we created a dense neural network to classify asteroids as hazardous or harmless using data from NASA's, "Nearest Earth Object," dataset. [16] This classification was done based on diameter, absolute magnitude, relative velocity, and estimated miss distance of an asteroid. Because of the data imbalance between hazardous and non-hazardous asteroids in the dataset, we achieved only about 75% accuracy during dataset testing. However, only 3.8%

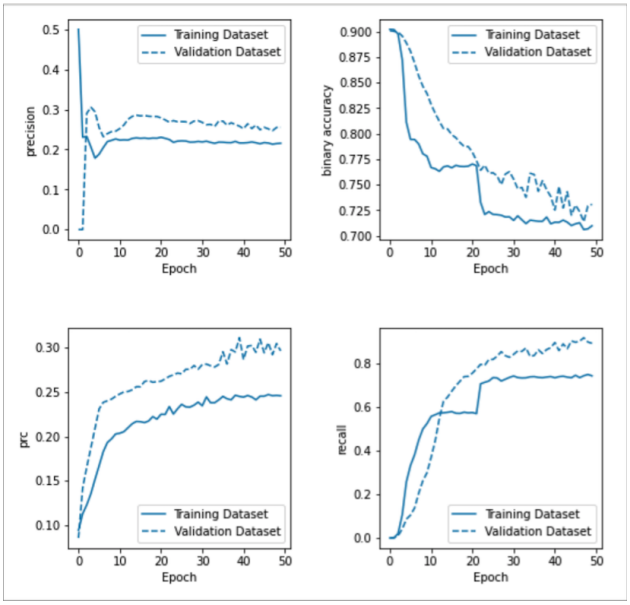

Figure 6. Train and test results for the ML Asteroid classification model.

of these predictions were false negatives (that is a hazardous asteroid was classified as non-hazardous).

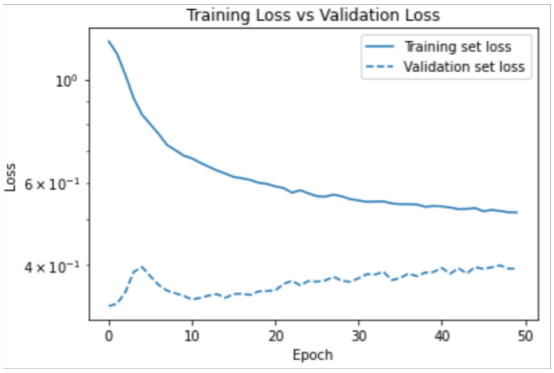

Figure 7. Training set losses for the ML Asteroid classification model

Figures 6 and 7 visualize the performance metrics of the machine learning model. Note that in Fig 6, PRC represents the area under the precision curve (AUPRC).

These deficiencies, however, proved valuable, since visualizing the model's faulty predictions gave us more opportunity to push our data loading program. Then we visualized these relationships in Figure 8.

In Figure 8, green nodes represent asteroids. Blue nodes represent the asteroids' classifications, as well as how the neural network classified them (hazardous or non-hazardous), and orange nodes represent the nature of the network's prediction (false negative, false positive, etc.)

While feeding a neural network's output into a knowledge graph isn't the most obvious usage of Neo4j, it demonstrates that it can present complex, high volume data in a unique visual medium using simple, short pieces of code; and it can also do so without extensive knowledge of the backend that goes into creating these visualizations. Hosting the knowledge graph on a Tapis pod during this project also demonstrated its efficacy for similar use cases.

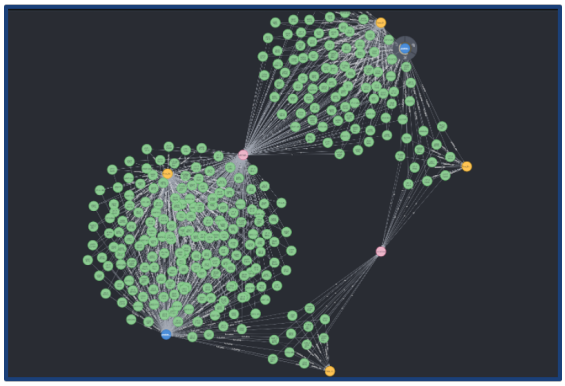

Figure 8. Training vs. Testing accuracy with Dropout Regularization.

5. Conclusions and Future Work

Throughout the course of our work, our team developed notebooks for stable, efficient authentication to ICICLE services hosted on TACC using Tapis, notebooks to host Neo4j Knowledge graphs, and command line applications to make accessing HPC systems more user-friendly. Through our research, we created the first instances of Jupyter Notebooks to connect to the iTapis/ICICLE system and access ICICLE-hosted KGs behind a Kubernetes Pods. Our interactive Jupyter Notebook Clients successfully address the issue of democratizing HPC systems by providing a simple and accessible way to create remotely hosted knowledge graphs by providing an easy plug-and-play that simplifies the authentication process. A key outcome of our efforts includes contributing our software to the upcoming ICICLE Software Components Catalog, which will be released around April, 2023. [14], [15]

Going forward, we will work on expanding the capabilities of our applications to ensure greater outreach of HPC resources. One of our primary goals is to develop our console applications to a production standard for easy access to Tapis resources. Additionally, we want to work with Visual Analytics within ICICLE to visualize our data outside of Neo4j.

6. Acknowledgment

As members of REHS 2022 Program, we are grateful to have been a part of such a large and impactful project, and we hope that the materials presented in this paper will be of use for ICICLE developers. The work on this project was carried out with the support of the San Diego Supercomputer Center Research Experience for Undergraduates (REHS) program. We also want to acknowledge the use of several NSF funded resources and services including: the AI Institute for Intelligent CyberInfrastructure with Computational

Learning in the Environment (ICICLE) (#2112606); SDSC Expanse project (#1928224); TACC Stampede System (# 1663578) and Tapis projects (#1931439); the NSF Track 3 Award: COre National Ecosystem for CyberinfrasTructure (CONECT) (#2138307); and finally, the Extreme Science and Engineering Discovery Environment (XSEDE) (NSF award #ACI-1548562).

References

[1] "Intelligent Cyberinfrastructure with Computational Learning in the Environment (ICICLE)." [Online]. Available: https://icicle.osu.edu/

[2] M. Parashar, "Democratizing Science Through Advanced Cyberinfrastructure," Computer, vol. 55, no. 9, pp. 79–84, 2022.

[3] "Knowledge Graph Wikipedia Page." [Online]. Available: https://en.wikipedia.org/wiki/Knowledge_graph

[4] "REHS Program Home Page." [Online]. Available: https://education.sdsc.edu/studenttech/rehs-home/

[5] "REHS Program Home Page." [Online]. Available: https://github.com/sdsc-hpc-students/REHS2022

[6] M. P. Thomas, C. Cheng, and J. E. Castillo, "Development of a Cyberinfrastructure-based Computational Environment for the General Curvilinear Coastal Ocean Model," in Congress on Computer Applications and Computational Science, no. Ci, 2010.

[7] Tapis, "Tapis Project Documentation." [Online]. Available: https://tapis.readthedocs.io/en/latest/

[8] Neo4J, "The Neo4J Graph Data Platform Project." [Online]. Available: https://neo4j.com/

[9] M. B. Milligan, "Jupyter as Common Technology Platform for Interactive HPC Services," in Proceedings of the Practice and Experience in Advanced Research Computing PEARC, vol. 9, no. 3, 2018, pp. 21–29.

[10] J. Stubbs, J. Looney, M. Poindexter, E. Chalhoub, G. J. Zynda, E. S. Ferlanti, M. Vaughn, J. M. Fonner, and M. Dahan, "Integrating Jupyter into Research Computing Ecosystems," in PEARC '20: Practice and Experience in Advanced Research Computing, Portland, OR, USA, 2020, pp. 91–98.

[11] S. B. Cleveland, A. Jamthe, S. Padhy, J. Stubbs, M. Packard, J. Looney, S. Terry, R. Cardone, M. Dahan, and G. A. Jacobs, "Tapis API Development with Python: Best Practices in Scientific REST API Implementation: Experience implementing a distributed Stream API," ACM International Conference Proceeding Series, pp. 181–187, 2020.

[12] "From graph to knowledge graph: How a graph becomes a knowledge graph." [Online]. Available: https://neo4j.com/blog/from-graph-to-knowledge-graph-how-a-graph-becomes-a-knowledge-graph/

[13] Tapis, "Hello Tapis Tutorial." [Online]. Available: https://tapis-project.github.io/tutorials/hello/hello/

[14] "Hello ICICLE Auth Clients, ICICLE VC3 Software Release." [Online]. Available: https://github.com/sdsc-hpc-training-org/hello_icicle_auth_clients

[15] "ICICLE Software Components Catalog - Release March, 2023." [Online]. Available: https://icicle.osu.edu/knowledge-transfer/cyberinfrastructure

[16] "NASA Near-Earth Object Surveyor Site." [Online]. Available: https://www.jpl.nasa.gov/missions/near-earth-object-surveyor