# Tapis Pods Service Exploration and Initial Performance Analysis

Christian Garcia, Joe Stubbs, Richard Cardone, Nathan Freeman
*Texas Advanced Computing Center*
*The University of Texas at Austin*
Austin, TX
(cgarcia, jstubbs, rcardone, nfreeman)@tacc.utexas.edu

***Abstract*—The Tapis Pods service is a novel open-source API within the Tapis platform which enables researchers to seamlessly manage Kubernetes containers, volumes, networking, and security at the Texas Advanced Computing Center (TACC). This paper explores the underlying operations, technologies, and workflows of the Tapis Pods service, showcasing its current implementation and effectiveness. Additionally, we discuss current and potential use cases, highlighting the service's unique features: such as management capabilities, persistent storage, sharing, and automatically encrypted networking. Initial performance measurements against local Docker containers and alternative cloud solutions demonstrate the Tapis Pods service's competitive performance, emphasizing its value as a general interface for deploying user-defined containers.**

***Index Terms*—containers-as-a-service, cloud-infrastructure, open-source, API, Kubernetes**

## I. Introduction

The Tapis Pods service is a hosted API within the Tapis [1] platform that manages Kubernetes [2] containers, volumes, networking, and security at the Texas Advanced Computing Center (TACC). Tapis's comprehensive suite of APIs equips researchers with the necessary tools to develop intricate and specialized workflows. These workflows include HPC system management, file handling, app integration, job scheduling, actor control, stream processing, and more. Tapis Pods extends the platform's capabilities by providing users with a robust interface to deploy and manage their long-lived containers, with automatically encrypted networking and flexible persistent storage options. For example, a user can instantiate a Neo4j [3] graph database, a custom Flask [4] server, or many user-defined applications. Users are able to configure accessible ports and their encrypted subdomain location using a single API call to the Pods service. This paper overviews the Pods service, its technologies, workflows, and performance.

Section 2 provides examples of use cases for the service. Section 3 offers an overview of the Pods service along with current usage. Section 4 details our design for testing performance against a local Docker [5] container and an alternative cloud solution. Section 5 presents initial performance findings. Section 6 discusses related work in the field. Section 7 outlines potential future work. Finally, Section 8 concludes the paper, summarizing the service and its role in research.

## II. Use Cases

This will be an overview of the Tapis Pods service's use cases, along with examples of the benefits a user might expect when using the Pods service.

### A. Encrypted Online Cloud Databases

The Pods service was initially designed to facilitate database creation and proxying. Many research groups have no simple method to deploy secure and collaborative database environments for multiple users outside of paid cloud services. The Pods service aims to alleviate deployment frustrations by providing a generic interface to deploy multiple templated environments. Currently the Pods service is capable of proxying connections for Neo4j, GraphDB [6], and PostgreSQL [7], three popular open-source databases, with a single line of code. The service is generic enough that users can easily deploy alternative databases and applications as well. Once deployed, the Pods service optionally routes user traffic directly to and from a deployed pod via encrypted TCP or HTTPS. As a result, users can continue to employ the interfaces they are accustomed to, such as regular database clients like psycopg2 for PostgreSQL, the Postgres's pgAdmin GUI, and Neo4j visualizers, which function as usual. This capability allows teams to collaborate on databases remotely and enables users to manage databases so that other pods can utilize them.

### B. Software Stack Deployments

Software stack deployments are a strong suit of the Pods service. The current trend of containerization for software applications makes deploying a wide range of software stack architectures with the Pods service possible. Users can create pods using any Docker image they prefer, making it easy to deploy different types of applications, such as simple web apps, complex multi-pod services, or production APIs.

Additionally, the Pods service offers robust pod management endpoints that allow users to easily shutdown, start, or a restart a pod. This feature provides a valuable mechanism for users to update their pod definitions and have their changes reflected with a quick restart. It's also worthwhile to note that the Pods service allows pinning of Docker container hashes or user-readable Docker tags. This allows users to easily update and upgrade the underlying Docker images behind their software stacks conveniently using the Pods service.

### C. Backups, Versioning, and Data Democratization

The Pods service also contains the notion of volumes and snapshots. Volumes allow users to write pod data to persistent block storage with read and write permissions while snapshots allow users to save a copy of volume data with read permissions. With this, users can persist data and additionally share live data between pods. This allows users to backup data, version data, publish data, and create complex multi-pod software stacks with data easily flowing between components. These objects are also easily shared between Tapis users and are able to made public to entire Tapis namespaces.

## III. Service Design and Implementation

The Tapis Pods service is a Python API using FastAPI [8], the Kubernetes Python client, and Tapis OAuth to provide an interface to manage pods, volumes, and snapshots. Users can interact with the service through HTTPS web requests or Tapis's Tapipy [9] python library, an auto-generating "live" API library with type checking and helpful hints.

### A. Explanation of Service Objects

Pods, equivalent to Kubernetes pods, can be defined via an API call or with one line of Python executed anywhere: "tapipy.Tapis().pods.create_pod(template='neo4j')". Users can specify the following: the Docker image to run, the command to run in the pod, ports to expose securely with TLS using TCP or HTTP protocols, volume mounting, expiration time, and requested hardware resources. In addition to pods, users can create and manage volumes and snapshots. Volumes are persistent block storage that can be mounted to pods, providing persistence and live data sharing between pods. Users can upload, list, and delete files attached to these volumes, offering flexible workflows. Snapshots are non-editable, persistent block storage that can be created and mounted, providing a snapshot of a volume at a specific point in time. Snapshots can be used for backups or sharing data from a volume or database backend, offering another powerful workflow function.

The service takes advantage of Kubernetes for deployment and object management. The Pods service simplifies the process and narrows the scope of Kubernetes. Whereas Kubernetes requires complex configuration, admin access, and understanding of the interface, Pods grants authenticated users the ability to simply define and deploy their applications in any Kubernetes environment the service is hosted in. For instance the Pods service can be easily deployed with a single command in a minikube [10] instance deployed locally on something as small as a Raspberry Pi [11]. Encrypted routing is also bundled with the Pods service, taking full advantage of Traefik proxy [12] with custom configurations required for database connection types such as Neo4j's Bolt and Postgres's StartTLS protocol.

### B. Service Architecture Overview

In this section, we'll explore and gain a better understanding of the Pods service architecture as depicted in Figure 1, starting at the top-left and moving clockwise.

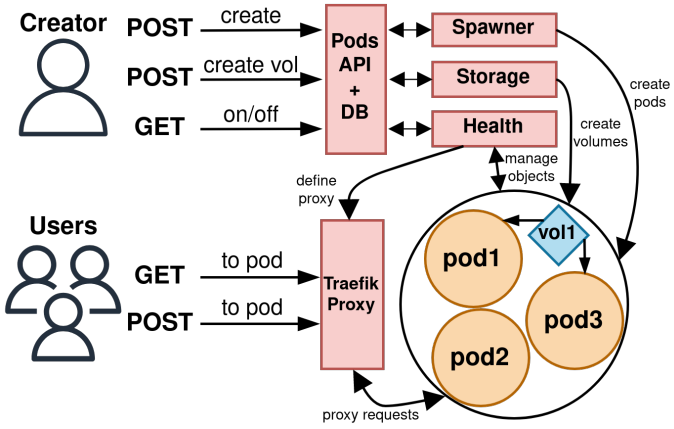


Fig. 1. An architecture diagram of the Tapis Pods service

Admins can create, share, and manage Pods through the Pods API. These changes are reflected in the PostgreSQL backend database, which contains object and administration tables. A RabbitMQ [13] message is submitted to request a new user-defined pod when a pod is created. The spawner container manages the RabbitMQ and creates requested objects. Kubernetes then takes over and creates the pod according to the user's provided definitions.

The Pods service uses a "health" container to regularly poll Kubernetes for pod status and state updates, which are then reflected in the PostgreSQL backend. The health container plays a crucial role in maintaining a clean working environment by managing the pruning of pods, volumes, and snapshots that do not have associations in the database. The process also flags objects in the database not present in Kubernetes. This functionality is critical to ensure optimal performance and a seamless user experience.

A critical task that the health process has is in the configuration of the Pod service's Traefik proxy. This proxy enables users to access their pods directly via the internet. While ensuring a healthy environment, the health process dynamically creates a proxy configuration file and updates the proxy when changes have been made. This means that if a user's pod, "mypod", went to "RUNNING" in the Kubernetes environment, health would grab the correct service IP address and update the proxy. This allows for a relatively short latency between deployment and access.

The final part of the diagram details that users can access a pod directly through the Traefik proxy provided by the Pods service. The Pods service can route a request to the correct pod by matching subdomain patterns and sniffing incoming requests' Server Name Identification (SNI). For example, a user can connect to "https://myvol.pods.tapis.io" to access an example "mypod" container on port 5000 using "HTTP", "TCP", "Bolt", or "StartTLS" protocols. The service uses an encrypted "HTTPS" connection without user input rather than "HTTP". This is made possible with the Traefik proxy dynamically allocating TLS certificates at runtime to ensure security for all proxy locations. This subdomain routing method allows

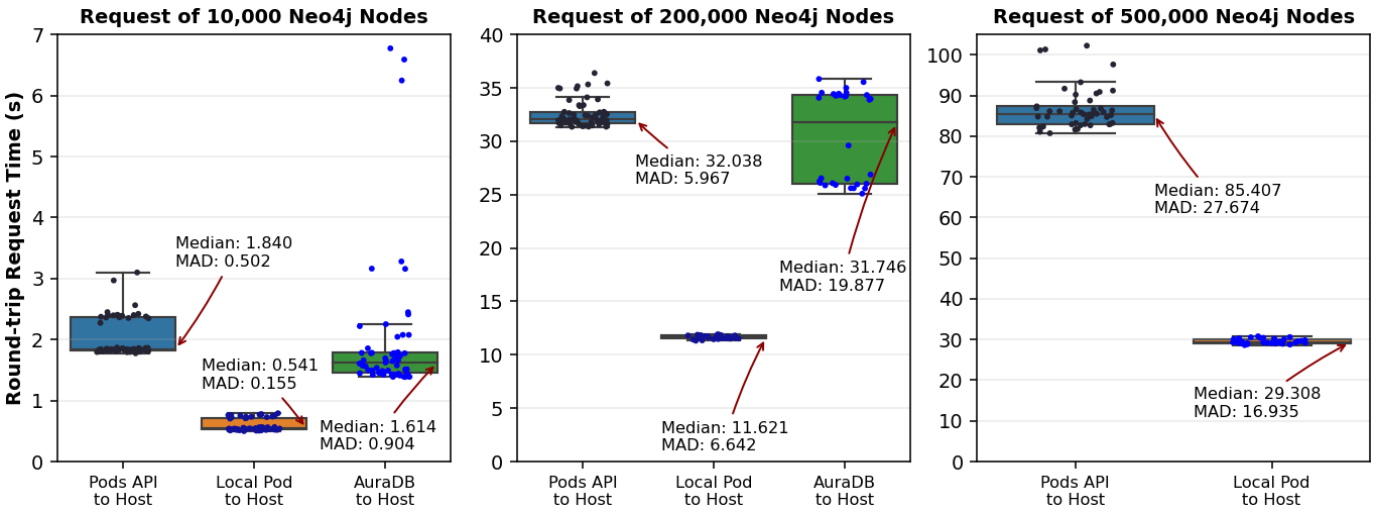


Fig. 2. Initial Neo4j performance results

users to deploy their image and use custom routes, such as "https://myvol.pods.tapis.io/mycustomroute", with minimal additional overhead while maintaining full encryption.

### C. Current Examples

*1) ICICLE - Components Catalog [14]:* The Components Catalog is a part of the ICICLE institute; a group that develops collaborative research tools. This groups makes use of the Pods service to deploy a Python Flask server pod and a Neo4j database pod that displays information from the Neo4j pod on the Flask server dashboard pod. With the Pods service's pod restart mechanism and Github action CI/CD, updating and redeploying the software stack is simplified. Whenever there are code updates, Github actions automatically build associated Docker images, push said Docker images, and restart the Flask server pod with the newly minted images. This streamlined process makes the Pods service an attractive option for deployment, especially for projects that require frequent updates and maintenance.

*2) ICICLE - Smart Foodsheds Visualization Dashboard [15]:* The Smart Foodsheds Project is an ICICLE institute initiative focused on creating a user-friendly graphical user interface (GUI) for retrieving and editing knowledge graphs stored as graph structures in Neo4j databases. The project utilizes a Vue.js frontend pod, Flask backend pod, and 3 Neo4j database pods, each of which is remotely managed by different members of the institute simultaneously. By leveraging the Pods service, the backend pod can connect to the three Neo4j database pods, while the frontend pod can connect to the backend pod. This multi-pod deployment allows for flexibility, encrypted networking, and scalability, while reducing the need for additional infrastructure management. Like the Components Catalog, the Smart Foodsheds Project uses Github actions for quick and efficient redeployments, enabling rapid development deployment cycles.

## IV. Performance Testing Overview

The Pods service provides a direct connection to user pods, however additional encryption and proxy layers may result in additional latency. To measure this latency, we will measure the round-trip request time when returning 10K, 200K, and 500K nodes from a Neo4j database instance. You can think of this as returning database rows in a traditional SQL database. This testing is intended to inform us only of initial service aptitude and is not an intensive breakdown of performance. A future paper will center around further detailed experimentation.

To evaluate the initial performance of the Tapis Pods service, we measured round-trip request time from a host machine to the Pods service and from a host machine to a local Docker container. We also collected metrics for round-trip request times from AuraDB [16] to a host machine for further comparison. AuraDB is a Neo4j-owned cloud service that enables remote access to Neo4j databases. The Pods service can be thought of as an open-source alternative to database cloud services and this will give us initial comparison points.

### A. Testing Setup

A detailed and documented repository to completely replicate our testing environment and procedure is located on Github at `tapis-project/pods_service_papers`. Our experiment architecture involves creating a Pods service Neo4j database, deploying a local Docker Neo4j database, and deploying a Neo4j database with Neo4j's AuraDB service. With our testing repository, we can quickly run and visualize the experiments, with each experiment being run 60 times to gather a complete picture of performance.

## V. Findings

The results of our initial testing can be found in Fig. 2. The figure features an x-axis detailing the experiment backend, a y-axis that denotes the round-trip request time in seconds, and

plot titles describing the experiment type. For each experiment, we employ both a box plot for analytics and a strip plot to facilitate an analysis of all data points. In all the experiments depicted in these figures, we report the median request time, accompanied by the Median Absolute Deviation (MAD) as a measure of dispersion. MAD provides a robust metric to describe data variability as opposed to standard deviation.

Fig. 2 shows the results of our differently scaled experiments in the three plots. Here we can see the difference in request timing depending on the three experiment backends. Note that the local container test proves helpful as a point of reference but not as a point of comparison. Inspecting the 10,000 node experiment, we see that the Pods service incurs a 0.226 second penalty compared to the offerings provided by AuraDB and a 1.299 second penalty compared to a locally deployed container. Inspecting the 200,000 node experiment, we can see that Pods service incurs a 0.292 second penalty compared to the offerings provided by AuraDB and a 20.417 second penalty compared to a locally deployed container. Finally, in the 500,000 node experiment, the Pods service incurs a 56.099 second penalty compared to a locally deployed container; note that AuraDB restricts node count to 200,000.

This initial testing reveals that the difference in median response time between AuraDB and the Pods service is sub-second even at such a preliminary stage with no additional optimization.

## VI. Related Work

The Pods service can be compared to several existing software systems. Prominent commercial examples of containers-as-a-service are Amazon's Elastic Container Service (ECS) [17] and Google's Kubernetes Engine [18]. Both solutions enable the use of custom container images, supported by defined storage, similar to the Pods service. However, the Pods service stands out as a more lightweight, open-source alternative that can be self-hosted.

Due to its versatility as a general deployment platform, Tapis Pods also directly competes with specialized cloud database solutions, such as Google's Cloud SQL [19] and Neo4j's AuraDB, which hosts Postgres and Neo4j databases, respectively. Our performance experiments reveal that the difference in median response time between AuraDB and the Pods service is sub-second, showcasing its competitive performance as a lighter-weight and simpler to use product.

## VII. Future work

Future improvements for the Pods service will focus on optimizing performance and user experience. Response times can be significantly reduced by streamlining proxy layering, which currently involves two proxies and then Traefik. Additionally, focusing on quality-of-life improvements, such as simplified data backup processes and integration with Tapis's UI component, will facilitate developer onboarding and offer users an intuitive GUI for even further simplification.

## VIII. Conclusion

In conclusion, this paper delves into the underlying architecture of the Tapis Pods service, detailing its key features and use cases. We performed a performance evaluation, facilitating an initial examination of system variance and a performance comparison with an established competitor in the cloud service domain, AuraDB. The comparison underscores the Tapis Pods service's capability to compete with enterprise-level paid offering with lower overhead and simpler controls.

Furthermore, we discussed related works and outlined future research and development avenues. This paper ultimately demonstrates the practicality and value of the open-source, user-friendly, and self-hostable Tapis Pods service, positioning the service as an appealing choice for a diverse array of applications within the research community.

## Acknowledgment

This material is based upon work supported by the National Science Foundation Office of Advanced Cyberinfrastructure, the Tapis Framework [1931439 and 1931575], and ICICLE [2112606].

## References

[1] J. Stubbs, R. Cardone, M. Packard, A. Jamthe, S. Padhy, S. Terry, J. Looney, J. Meiring, S. Black, M. Dahan, S. Cleveland, and G. Jacobs, "Tapis: An api platform for reproducible, distributed computational research," in *Advances in Information and Communication*, K. Arai, Ed. Cham: Springer International Publishing, 2021, pp. 878–900.

[2] (2023) Kubernetes. Last access: 2023-4-19. [Online]. Available: kubernetes.io

[3] (2023) Neo4j. Last access: 2023-4-19. [Online]. Available: neo4j.com/

[4] (2023) Flask. Last access: 2023-4-19. [Online]. Available: flask.palletsprojects.com/en/2.3.x/

[5] (2023) Docker. Last access: 2023-4-19. [Online]. Available: docs.docker.com/

[6] (2023) Graphdb. Last access: 2023-4-19. [Online]. Available: ontotext.com/products/graphdb/

[7] (2023) Postgresql. Last access: 2023-4-19. [Online]. Available: postgresql.org/docs

[8] (2023) Fastapi. Last access: 2023-4-19. [Online]. Available: fastapi.tiangolo.com

[9] (2023) Tapipy. Last access: 2023-4-19. [Online]. Available: github.com/tapis-project/tapipy

[10] (2023) Minikube. Last access: 2023-4-19. [Online]. Available: minikube.sigs.k8s.io/docs

[11] (2023) Raspberry pi. Last access: 2023-4-19. [Online]. Available: raspberrypi.com/

[12] (2023) Traefik. Last access: 2023-4-19. [Online]. Available: doc.traefik.io/traefik

[13] (2023) Rabbitmq. Last access: 2023-4-19. [Online]. Available: rabbitmq.com/documentation.html

[14] (2023) Components catalog. Last access: 2023-4-19. [Online]. Available: https://components.pods.icicle.tapis.io/data

[15] Y. Tu, X. Wang, R. Qiu, and H.-W. Shen, "An interactive knowledge and learning environment in smart foodsheds," *IEEE Comput. Graph. Appl.*, vol. PP, Apr. 2023.

[16] (2023) Neo4j auradb. Last access: 2023-4-19. [Online]. Available: neo4j.com/docs/aura

[17] (2023) Amazon elastic container service. Last access: 2023-4-19. [Online]. Available: aws.amazon.com/ecs/

[18] (2023) Google kubernetes engine. Last access: 2023-4-19. [Online]. Available: cloud.google.com/kubernetes-engine

[19] (2023) Google cloud sql. Last access: 2023-4-19. [Online]. Available: cloud.google.com/sql/docs/postgres