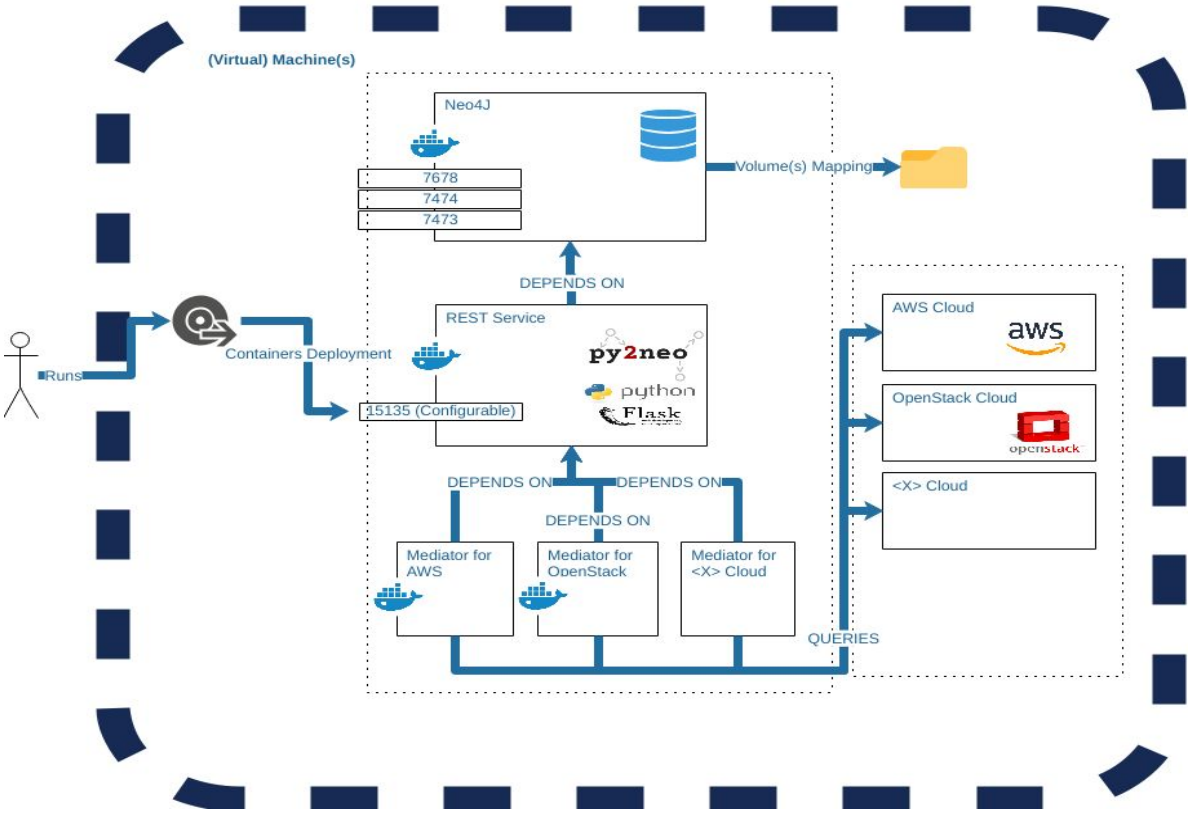
Storing OpenStack Infrastructure Data in Neo4J Graph Database—Technical Documentation

1. Introduction:

To keep track of components that form a working large-scale cloud environment, the resources provisioning/de-provisioning and faulty resources are difficult data management tasks for a cloud provider. Moreover, detecting failures efficiently and detecting used up resources are priorities for any cloud provider so that the appropriate action can be taken timely to keep the services running. The situation gets worse when provisioning/de-provisioning of resources occur frequently or in case of frequent failures. To solve these issues, we propose a solution that will provide real-time cloud infrastructure data in a user-friendly graph-based storage and visualization to keep track of those resources. It provides better planning of resources which will consequently save cost and avoid failures. Prior to this solution, these problems were demanding lots of scripting and manual work. This system can be used with any cloud environment that provides an API to access cloud infrastructure data and it can be deployed on a virtual or physical machine(s).

2. System Architecture



We propose to deploy the graph database and other components in Docker containers. We choose Docker for this purpose to make the components of the system reusable, and portable and simplify the required configurations. Moreover, We use docker-compose to deploy the required containers—Neo4J graph database, the REST service, and a mediator—to further simplify the deployment and configuration process.
In our implementation, we implemented a mediator for OpenStack to keep track of VMs and containers deployed on those VMs. For persistence, the data folder of the graph database is mapped to the system where it is deployed. The REST service, which has connection information to the graph database, interacts directly with the graph database.
In the following section, we will look into each component in detail.

3. Components

The system essentially contains three components—Neo4J graph database, the REST service, and a mediator—deployed inside Docker containers. The Graph database and the REST service are the essential components of the system, however, the user can provide its own mediator component.

- **Neo4J Graph Database**

  We choose the Neo4J graph database over other available relational or graph databases to store the data.

  [TODO: graph vs. relational and Neo4J vs other graph dbs for this problem]

  The data folder is mapped to a local folder of the containing (virtual) machine so that the data is retained in case the container crashes.

- **REST Service**

  It exposes REST service that interacts with the Neo4J database to perform CRUD (Create, Read, Update, Delete) operations. This layer is an abstraction to the Neo4J database that contains business-logic of the system and to avoid data format or validation errors that a user can make while performing CRUD operations.

- **Mediator**

It is a client module of the REST service. It queries and fetches data from a cloud infrastructure and sends it to the REST service to write it on the graph database. Mediator for more than one cloud type can be used.
The mediator can subscribe to several events of interests provided by the cloud environment. Upon the occurrence of any of those events, the graph can be updated immediately to reflect the change made to the cloud environment.

4. **Deployment**

To make the deployment easier, we use docker-compose to deploy the following containers:

- *Neo4J*
- *REST service*
- *Mediator*

*Deployment instructions are given in the **README** file associated with the Github repository.*

5. **Graph Data Structure**

A dependency graph, *which is a directed graph representing dependencies of several objects towards each other*, is created as a result of running the end-to-end workflow.

It is helpful to show dependencies in a graph structure, which will make easier to recognize failed component and its dependent components which will also fail. E.g. if a compute node in OpenStack is failed, the VM running on that node will also fail.

[TODO: how dependency graph works...]

[TODO: insert an image as an example]

6. **Conclusion**

By using this system, cloud providers can rapidly store and update the infrastructure data for better visualization, exploration, and planning. This will help the business in better resource allocation and deallocation which will result in reduced cost and more profit. Moreover, the graph's data can be used for other purposes in future.