HORIZON, A WEB-BASED USER INTERFACE FOR MANAGING SERVICES IN OPENSTACK: AN INTROSPECTION

Ashish Lingayat
Computer Engineering
MIT Academy of Engineering
Pune, India
ashishvijaylingayat@gmail.com

Avinash Singh
Computer Engineering
MIT Academy of Engineering
Pune, India
singhy05081996@gmail.com

Vinay Naik
Computer Engineering
MIT Academy of Engineering
Pune, India
naikvinay832@gmail.com

Ranjana R. Badre
Computer Engineering
MIT Academy of Engineering
Pune, India
rbadre@comp.maepune.ac.in

Anil Kumar Gupta
HPC Infrastructure and Ecosystems
C-DAC Innovation Park
Pune, India
akgupta5592@gmail.com

***Abstract*—Cloud computing, an emerging technology in today's world is making a significant impact on sharing resources. An open-source platform OpenStack used for provisioning cloud environment is gaining tremendous popularity. OpenStack introduces independent projects developed for providing specific services in the cloud. These services interact with each other to act as one cloud environment. The services are managed using commands and require knowledge of their usage. To make the management of the services more convenient OpenStack has designed and developed a user interface based project called Horizon. It is a web-based interface which interacts with other services to make them configurable and manageable through a dashboard. In this paper, we will discuss the present scenario of the services interacting with Horizon.**

***Index Terms*—Cloud computing, OpenStack, Horizon, User interface.**

I. Introduction

Cloud Computing is on-demand technology for delivering computing services such as networking, database storage, resource pooling, software application and other IT services through the internet via usage-based pricing. Five deployment model provided in cloud computing include private, public, hybrid, virtual private and community cloud. Services provided by cloud computing are Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Services (SaaS) [1].

The cloud organizations use OpenStack to spread out and supervise cloud-based infrastructure that footholds an array of use cases, including web hosting, big data projects, SaaS delivery or positioning high volume of containers. OpenStack is a stack of open source module that provides the framework for organizing and managing public and the private cloud infrastructure. The main characteristics offered by OpenStack are compatibility, scalability, and flexibility [2].

The component named OpenStack Horizon provides a dashboard for managing services in OpenStack. Horizon interacts with OpenStack Nova, OpenStack Swift, OpenStack Keystone and other services and all these services accessed via an OpenStack Application Programming Interface (API).The developer can extend the existing Dashboard by integrating more functionality to it [2].

A. Aim of this paper

This paper covers the information required for developing and working with Horizon dashboard (web-based interface) and improves the limitation of documentation. Using this information, it becomes easy to design and develop Horizon to meet user requirements. This paper also presents the benefits of using OpenStack Horizon instead of other interfaces.

II. Related Work

Cloud vendors provide limited support on common standards which makes it difficult for creating cloud applications [3]. Many cloud vendors are having closed source, and proprietary interfaces and APIs. Standardization of the interfaces and APIs is required to deal with this issue. Standards are developed for providing unified user experience across multiple platforms. Open Grid Forum has a standard called Open Cloud Computing Interface (OCCI) for managing cloud tasks by using API [4]. These standards provide interoperability to the applications [5].

OpenStack User Survey [6] shows the need for further development on OpenStack Horizon. Horizon being used 88% in production deployment, needs further developments in auto-scaling groups, Horizon for admins, and other features. Horizon is still not developed to the extent for the commercial offering. Lack of documentation is one of the most significant areas for consideration.

The OpenStack dashboard is customizable as per the requirement of the user. It can be designed and modified according to organizational hierarchy. Prajesh Anchalia et al. [7] show the sample customization of OpenStack Horizon.

III. OpenStack

NASA and Rackspace Hosting mutually launched a platform for cloud computing known as OpenStack in July 2010.

It is an open source cloud operating system, and provides a framework for various deployment models in the cloud. The openstack.org is licensed under Apache License. Projects that run under OpenStack infrastructure (to provide OpenStack software) must license under an OSI-approved license. OpenStack community releases updates regurarly every six months. The community organizes OpenStack Design Summit to assemble members for smooth development, working sessions, and convene plans. The release history of various version includes name starting alphabetically from Austin till Queens [8].

OpenStack develops projects implemented autonomously for achieving the objectives of the service. The services are controlled using two nodes Controller node and Compute node. Controller node centrally manages these projects that run on compute node. Compute node handles large stock of storage, computing and networking resources across the cloud environment. These resources are provisioned with the help of a dashboard. OpenStack eases the functionality of user by allowing it to fetch corresponding resources through the web-based interface [8].

IV. Projects in OpenStack

Fig. 1. shows the interaction of the users with the services in OpenStack. The OpenStack architecture is highly customizable, built with components that work autonomously called services. The commencing release of OpenStack included just two services: Nova (to manage compute resource pools); and Swift (the object storage system) [8]. As an IaaS framework that is extensible and API-driven to offer vendors and solution providers, a path for associating their computing, networking, and storage infrastructure plugins best suited to their environment. OpenStack is having a modular structure and received the various unique name for its components.

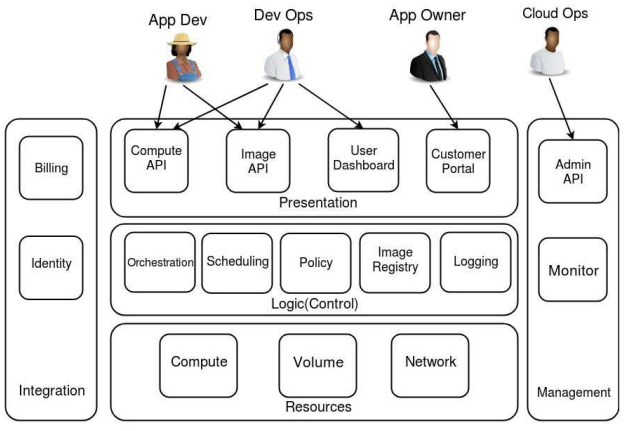

Fig. 1. Interation of users and services in OpenStack [9].

The services provided by OpenStack are categorized as per their functionality. Below is the list of all the present projects being developed for provisioning the respective services:

A. Workload provisioning

This category includes the projects being developed for provisioning workload.

Sahara: This project is used for data processing (Spark, Hadoop, and Storm) in OpenStack.

Magnum: Provides Container Orchestration Engine (Docker Swarm, Kubernetes, and Apache Mesos) for management of containers.

Trove: It provisions relational and non-relational database using Database as a Service.

B. Frontend (Web-based interface)

This provides a frontend for the services present in OpenStack.

Horizon: Provides a dashboard for managing services of OpenStack.

C. Orchestration

Orchestration of services is implemented by following projects.

Aodh: Triggers action based on the events generated by Ceilometer as defined in rules.

Heat: Based on the templates (in the form of text files) created orchestration of resources in infrastructure is implemented.

Senlin: It objective is to form a cluster of homogenous objects from the orchestration of similar objects.

Mistral: A YAML file-based implementation used for state management, parallelism, synchronization, correct execution order, and high availability for implementing workflow service.

Zaqar: A messaging service for multi-tenant cloud services providing communication between web and mobile developers.

D. Lifecycle of Application

Applications lifecycle is managed by using projects.

Solum: Used for automating software development lifecycle.

Murano: Using Heat it is used for publishing cloud-ready applications and enabling developers and cloud administrators to be displayed in the browsable catalog.

Freezer: it is used for disaster recovery, backup, and restore.

E. Shared services

The services sharing data across OpenStack are listed in this category.

Searchlight: Used for searching and indexing resources in OpenStack.

Barbican: It is used for storing secret data by acting as a Key Manager in OpenStack.

Keystone: Identity service used for authenticating users centrally in OpenStack.

Karbor: It is developed for protecting metadata and data of the applications deployed in OpenStack.

Glance: Repository for storing images in OpenStack.

F. Container infrastructure

This projects provision infrastructure for providing containers.

Kuryr: Acts as a bridge between networking and storage of OpenStack to networking and storage of containers.

G. Storage

This services centrally provide storage in OpenStack.

Swift: Provides object storage in the OpenStack cloud.

Manila: Shared filesystem in OpenStack.

Cinder: Block storage provisioning in OpenStack.

H. Compute

These projects execute the computing tasks.

Nova: Computing service of OpenStack.

Ironic: Used for providing bare metal machines.

Zun: launches and manages containers using OpenStack API

I. Monitoring tools

Other services are monitored by using these projects.

Monasca: Used to generate alarms and notifications for provisioning Monitoring as a Service.

Ceilometer: Used for billing purpose by implementing telemetry service.

Panko: Used for collecting data of a resource at a particular state and given time by providing indexing, event storage, and metadata.

J. Billing / Business logic

Cloud providers use this project for generating bills to the consumers.

CloudKitty: It is used for converting metrics to prices by using Rating as a Service.

K. Network Function Virtualization (NFV):

This project is used for virtualizing the network.

Tacker: Network services and Virtual Network Functions (NFV) are deployed and operated using this project.

L. Multi-region tools

This service helps in implementing tools into multiple regions.

Tricircle: Implemented in neutron for automating networking.

M. Optimization / Policy tools

For implementing policies and optimizing the services below services are used.

Vitrage: To the detect root cause of a problem in OpenStack Root Cause Analysis (RCA) is used in this project.

Watcher: It is implemented for resource optimization of service in OpenStack.

Congress: It is used for creating policies in services provided by OpenStack for provisioning Policy as a Service.

Rally: It is used for benchmarking and performance analysis for the code of OpenStack.

N. Networking

These services provision networking in the OpenStack cloud.

Neutron: Software Defined Network (SDN) for networking in OpenStack.

Octavia: Load balancing solution in OpenStack for containers, virtual machine, and bare metal.

Designate: Provides DNS service.

O. Deployment/Lifecycle tools

These tools are used for deployments and lifecycle management of services.

TripleO: Provides infrastructure for deploying OpenStack.

OpenStack-Ansible: Enables Ansible playbooks and roles for configuration and deployment in OpenStack.

Kolla: Developed for providing production-ready containers in OpenStack.

V. OpenStack Horizon and CLI

The difference between the OpenStack Horizon and CLI can be understood by comparing the process and method for launching an instance. Prerequisite for starting an instance requires virtual network, flavor, key pair, security group, and image to be created. The process for launching the instance is different for both the interfaces. Fig. 2. shows process of CLI for launching instance using commands and Fig. 3. shows the process for launching instance through Horizon using the web-based interface.

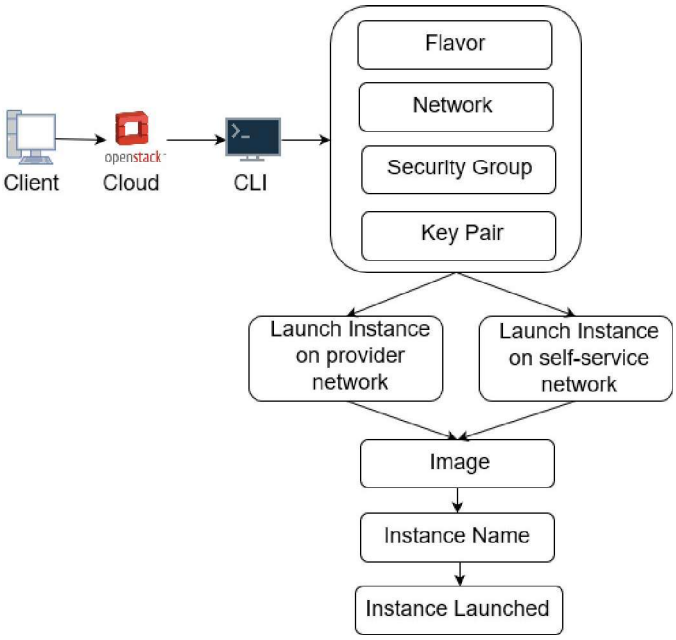

Fig. 2. OpenStack CLI for launching an instance.

CLI involves an additional overhead of remembering the commands, services, and the name of the services created. In Horizon the dashboard displays all the options available for launching the instance. The need of remembering the commands and name of the services created is eliminated.

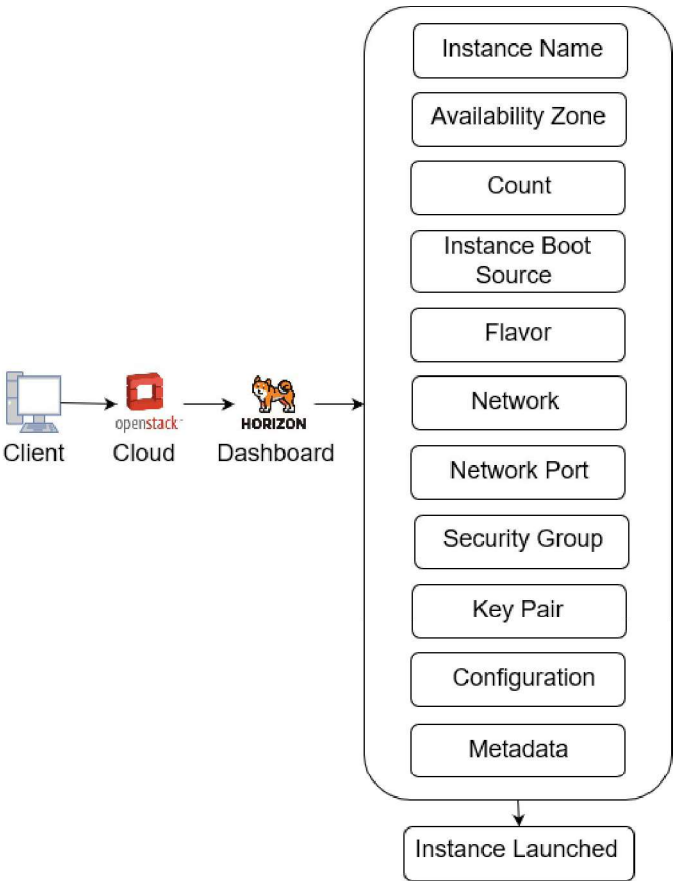

Fig. 3. Horizon dashboard for launching an instance.

Field validation ensures that the instance is started with no mistake.

VI. OpenStack Horizon

Horizon is the standard implementation of OpenStack's Dashboard [10]. It is a Django based web application and is hosted using an Apache web server. It provides web-based user interfaces to various OpenStack services like Nova, Swift, Keystone, Cinder, Glance, and many more [11]. The services are called using REST-based API frontend to make this services readily available to the user through a web browser.

Fig. 4. shows the interaction of services with OpenStack Horizon. Horizon plays an essential role in the center of the design and architecture of OpenStack. It handles three dashboards for providing core support and includes all the core applications of OpenStack [12]:

- User Dashboard
- System Dashboard
- Settings Dashboard

Horizon dashboard is extensible by using the *Dashboard* class for creating top-level navigation component of OpenStack. To create a new dashboard, developer needs to hook the created app with other dashboard and import it. For creating sub-navigation in dashboard, panel is used which invokes the

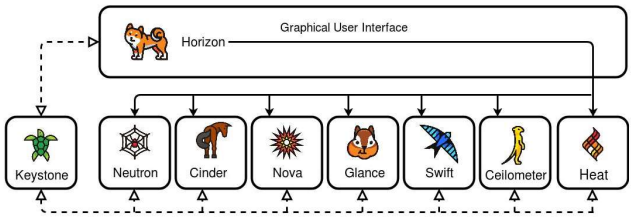

Fig. 4. Dashboard interface of OpenStack Horizon.

method *Panel*. Panel contains all the necessary components required for the interface. The items required for building a form are present in templates or additional tools to make the application consistent. By shipping these components together, it is assured that the changes are made most comfortably and makes it more stable. The components can be used by invoking them when required. This makes them reusable and saves time for redevelopment of the existing components and lets the developers focus on further development.

To support the angular work done in Horizon, it uses RESTful API and allows the code on the client side to make API call to the server without Django server-side rendering. This makes sure that problems like cross-origin resource scripting as well as leveraging authentication and authorization libraries already exists [13].

A. Working with OpenStack Horizon

Horizon project is open source under Apache license 2.0 to invite contributions from the developers. Horizon development is standardized to maintain code quality standards. It is entirely customizable to modify it as per the user requirements. Horizon dashboard can be branded by changing the styling of pages, logo, and many more according to the requirements of the organization willing to use it.

Horizon is developed using Python 2.7 or 3.5, and Django 1.11 or 2.0. The other platforms used for developing Horizon are JavaScript, AngularJS, ESLint, CSS, JavaScript and CSS libraries using xstatic, and HTML. Horizon is developed and tested in a virtual environment by hosting the service in *tox* (used for standard and automated packaging, testing, and release of python software in the virtual environment) which is officially supported by the OpenStack Foundation. The official code of Horizon is available at https://git.openstack.org/openstack/horizon and can be cloned for further contributions.

The services of OpenStack are deployable individually, but some of them require the essential components to be pre-installed. One such service required by Horizon is the Keystone. Keystone endpoint is used for authenticating the users to begin working with Horizon. The other services such as swift, glance, cinder, nova, and neutron are optional and are auto-detected by using the keystone endpoints. Use of plugins supports other services of OpenStack.

1) Horizon Structure: The structure of Horizon is divided into two components:

- **horizon**
  It contains the libraries and components that are required by Django.
- **openstack_dashboard**
  It includes a reference for the Django for using **horizon**.

2) Components of Project:

- Dashboard configuration Used to add new dashboard in Horizon.
- URL URL for accessing the dashboard.
- Template This is used for maintaining the HTML files.

3) Design of Application:

- Structure This defines the file system structure of the project.
- Dashboard class Defines the new dashboard.
- Panel class Used for including panel under the dashboard as sub-navigation.

VII. Demand for Horizon

The services handled traditionally, make use of Command Line Interface (CLI). Use of CLI requires the technical knowledge base of the commands. Getting familiar with these commands becomes very difficult for the users, making the service less convenient to use. To make the services convenient and intractable for the user, there is a need for a user interface. OpenStack makes this user interface available through a web-based dashboard named Horizon.

VIII. Existing project working with Horizon

The projects who are using Horizon for managing the services using the web-based interface are:

- **Ceilometer:**
  This project provides resource tracking, billing, and alarming capabilities for all the core components of OpenStack using Horizon.
- **Cinder:**
  Management of volumes, taking backups, snapshots, consistency group, and consistency group snapshot are the features available with Horizon.
- **CloudKitty:**
  Using this plugin user can define the rating policy without CLI. Information on usage and prediction cost of an instance can be obtained via Horizon.
- **Congress:**
  Writing policies and rules can be done using Horizon for governance of OpenStack.
- **Designate:**
  Using Horizon DNS domains, reverse DNS, and zones can be managed.
- **Freezer:**
  Freezer dashboard provides reporting, metrics, and multi-node backup synchronization which are advance settings provided through CLI.
- **Glance:**
  Images are created, updated, and deleted using the dashboard for Glance.
- **Heat:**
  Heat Orchestration template can be generated, and stacks are managed and launched using Heat dashboard.
- **Ironic:**
  The Ironic dashboard is used to manage and view drivers, ports, and bare metal nodes.
- **Karbor:**
  Checkpoints, managing protection plan, triggers, maintain operation logs, maintain protection providers, restores, and scheduled operations can be performed through Karbor dashboard.
- **Keystone:**
  Keystone service is utterly manageable by using dashboard.
- **Magnum:**
  Magnum dashboard is capable for managing cluster templates, clusters, and quotas.
- **Manila:**
  Using Manila dashboard, it can manage a share, manage the shared network, and manage security service.
- **Mistral:**
  Mistral dashboard can manage actions, and manage cron triggers.
- **Monasca:**
  It is used to monitor the health of Monasca specific components.
- **Murano:**
  Using Murano dashboard user can control and manage created environments, executing applications, and application catalog along with other resources present in OpenStack.
- **Neutron:**
  Almost all the components of Neutron can be managed by using Horizon dashboard.
- **Nova:**
  Nova services are entirely manageable using UI of Horizon.
- **Octavia:**
  Using Octavia dashboard user manages the load balancer provided by this project.
- **Sahara:**
  Data processing on clusters panel, plugins, and jobs panel are available in Sahara dashboard.
- **Searchlight:**
  Searchlight dashboard provides the search facility for searching and indexing resources in OpenStack.
- **Senlin:**
  Using Senlin dashboard users can manage clusters.
- **Solum:**
  Manage applications, and create an assembly from an application is done using Solum dashboard.
- **Swift:**
  The Swift project is utterly manageable through the dashboard.

- **TripleO:**
  Using TripleO dashboard user can manage plans, manage nodes, configure the deployment, assign nodes, deploy the Overcloud, and Post-Deployment.
- **Tracker:**
  Tracker dashboard is used for managing NFV.
- **Trove:**
  The dashboard of Trove provides capabilities for taking database backups, managing database clusters, configuring the database, and managing databases.
- **Vitrage:**
  The dashboard of Vitrage is used for topology, alarms, and entities.
- **Watcher:**
  Action plans, actions, audit templates, audits, goals, and strategies are managed using Watcher dashboard.
- **Zaqar:**
  Zaqar UI provides messaging groups, queues, pools, and pool flavors.
- **Zun:**
  The dashboard of Zun provisions clouds shell and container.

IX. CONCLUSION

This paper presents a study on the web-based interface present for managing the services in OpenStack. The services are operated using Command Line Interface or Dashboard. To remove the dependency of remembering the commands, OpenStack has developed a web-based user interface called Horizon (a self-service portal). It interacts with the services by making OpenStack API calls.

The GUI is an extensible Django web framework that allows plugin for third-party applications easily. The dashboard is highly configurable to suit the requirements of the organization and brand it accordingly. The user can view the available resources and monitor them using Horizon. The topology of the network can be seen graphically in the user interface. The demand of Horizon is extensive due to the benefits available.

To the limit of our knowledge Horizon seems to be less exploited. We have tried to compact the information in this paper to know about the current development of Horizon. There is immense scope for future development of Horizon since new services will get added, which will require an interface. The existing services also need to get highly manageable through Horizon to ease the task of the administrators, users, and providers.

ACKNOWLEDGMENT

The work was supported by the Centre for Development of Advanced Computing, Pune, India.

REFERENCES

[1] A. Kavanagh, "Openstack as the api framework for nfv: the benefits, and the extensions needed," *Ericsson Review*, vol. 2, 2015.

[2] R. Kumar and B. B. Parashar, "Dynamic resource allocation and management using openstack," *Nova*, vol. 1, p. 21, 2010.

[3] V. Munteanu, C. Şandru, and D. Petcu, "Multi-cloud resource management: cloud service interfacing," *Journal of Cloud Computing: Advances, Systems and Applications*, vol. 3, no. 1, p. 3, 2014.

[4] S. Ortiz, "The problem with cloud-computing standardization," *Computer*, vol. 44, no. 7, pp. 13–16, jul 2011.

[5] S. Fu, J. Liu, X. Chu, and Y. Hu, "Toward a standard interface for cloud providers: The container as the narrow waist," *IEEE Internet Computing*, vol. 20, no. 2, pp. 66–71, mar 2016.

[6] "Openstack user survey," openstack.org, Tech. Rep., 2017. [Online]. Available: https://www.openstack.org/assets/survey/April2017SurveyReport.pdf

[7] P. P. Anchalia, P. Gupta, and J. Shetty, "A customized dashboard for vm provisioning using openstack," in *Computational Intelligence, Communication Systems and Networks (CICSyN), 2015 7th International Conference on*. IEEE, 2015, pp. 177–182.

[8] "Overview of azure service fabric," Jan. 2018. [Online]. Available: https://docs.microsoft.com/en-us/azure/service-fabric/service-fabric-overview

[9] K. Pepple, "Openstack architecture the begin end." [Online]. Available: http://ken.pepple.info/

[10] P. P. Anchalia, P. Gupta, and J. Shetty, "A customized dashboard for vm provisioning using openstack," in *Computational Intelligence, Communication Systems and Networks (CICSyN), 2015 7th International Conference on*. IEEE, 2015, pp. 177–182.

[11] A. Awasthi and R. Gupta, "Multiple hypervisor based open stack cloud and vm migration," in *Cloud System and Big Data Engineering (Confluence), 2016 6th International Conference*. IEEE, 2016, pp. 130–134.

[12] "Horizon basics," Mar. 2018. [Online]. Available: https://docs.openstack.org/horizon/latest/contributor/intro.html

[13] "Horizon/restapi," Mar. 2018. [Online]. Available: https://wiki.openstack.org/wiki/Horizon/RESTAPI