Master of Science in Internetworking

Department of Electrical and Computer Engineering

Report for Capstone

Name of Capstone Project:

Comparison and Contrast of OpenStack and OpenShift

Mentor:

Dr. Mike MacGregor

Submitted By:

Geetha Nagini Vallabhaneni

TABLE OF CONTENTS

TABLE OF FIGURES

CHAPTER 1: INTRODUCTION TO CLOUD COMPUTING

Introduction of cloud computing:

Cloud computing provides organizations with computing resources (hardware or software) via the internet. Computing resources like servers, networking equipment, database, storage, and software can subscribe by the organizations from cloud service providers. The cloud service providers help companies to scale their computing resources on-demand and pay for the resources used. Instead of owning their computer infrastructure or data centers, businesses can lease access to software to storage from a cloud service provider. It helps businesses to save their capital expenditure on infrastructure. Most of the companies benefit from the pay-as-you-go model of cloud computing.

The basic concept in cloud computing is that cloud service providers take advantage of economies of scale to deliver IT services to customers on demand. Technology has significantly improved, and many more organizations have needs for computing that are elastic. Businesses cannot afford to offer IT facilities to satisfy the peak demand that happens infrequently. Cloud computing is the optimal way to meet these needs without incurring high costs in providing services.

Cloud computing is successful because of the availability of higher bandwidth for communications and the underlying technology virtualization. Using virtualization technology, each device(servers) is divided into many virtually to leverage the device's capacity. With the advent of virtualization, companies can access the service provider's resources over the internet. Cloud computing provides a vast number of services to which every type of organization can get access.

Definition of Cloud Computing:

According to, NIST (National Institute of Science and Technology) defines “cloud computing as a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or interaction. This cloud model comprises five essential characteristics, three service models, and four deployment models”[2]. The key features of cloud computing are explained.

Essential Characteristics of cloud computing: [2]

1.On-demand self-service: A customer can automatically provide computing capabilities unilaterally, such as server time and network storage, as necessary, without requiring human interaction with each service provider.

2.Broad area network access: Capabilities are accessible over the network and accessed via standard frameworks that allow heterogeneous thin or thick consumer platforms to use (e.g., mobile phones, tablets, laptops, and workstations).

3.Resource pooling: Using a multi-tenant model, the computing resources of the provider are pooled to support multiple clients, with different physical and virtual resources dynamically allocated and reassigned according to consumer demand. There is a sense of independence from the location in that the consumer typically has no power or knowledge of the exact location of the services offered but may define the location at a higher abstraction level (country, state, datacentre). Storage, processing, memory, and network bandwidth are examples of resources.

4.Rapid elasticity: Capabilities can be supplied and released elastically, in some cases automatically, to scale outward and inward according to demand quickly. The capabilities available for provisioning always seem limitless to the user and may at any time be appropriated in any quantity.

5.Measured service: By using a metering capability at some stage of abstraction relevant to the type of operation, cloud systems automatically monitor and optimize resource use (e.g., storage, processing, bandwidth, and active user accounts). The use of resources can be monitored, reported, and controlled, providing both the provider and the consumer of the service used with transparency.

Cloud deployment and service types:

Cloud computing has four cloud deployment types, namely: public, private, hybrid, community. These deployment models are briefly discussed in Chapter 2.

Software as a Service (SaaS), Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Backend as a Service (BaaS) are various cloud service types. These topics are discussed in Chapter 3.

Cloud services providers: Amazon, Microsoft Azure, IBM Cloud, Google Cloud Provider, RackSpace, Salesforce.com, VMWare, OpenStack, OpenShift, Apple, and Alibaba cloud.

Advantages of Cloud Computing:

- The cloud service providers make sure their service availability is 24*7. Hence there is no interruption in the service from cloud service providers.
- Pay-as-you-model of cloud computing benefits the companies from not owning to buy infrastructure.
- Scalability is another crucial advantage of cloud computing. Organizations can scale up or scale down their resources according to their demand and pay what is used.

- Companies can build applications faster instead of waiting to buy the necessary IT resources and software to deploy them. Cloud provides different platforms for companies to test and develop their applications.

CHAPTER 2: CLOUD DEPLOYMENT MODELS

Introduction:

NIST defines four cloud deployment models: public clouds, private clouds, community clouds, and hybrid clouds. The cloud deployment model is represented as where the deployment infrastructure is located and who controls the infrastructure. One of the most significant cloud deployment choices organizations make, which deployment model they will go with [2].

The most frequently deployed and used by both small and medium-sized enterprises are public clouds. Private clouds are mainly used by large corporations that need to supplement their data centers efficiently. Hybrid clouds offer companies a way to manage certain in-house services by themselves and use the public cloud for some of their customer-facing services. The Community Cloud serves the vertical market, such as healthcare and automotive, where users have some standard features in their applications.

Each cloud deployment model is useful for different needs of an organization. Every cloud deployment model has a different value proposition and various costs related to it. Therefore, a cloud model should be chosen carefully to fulfill the needs of the organization adequately. In certain situations, the choice of cloud computing model comes down to funds [2]. In any case, before deciding on a particular cloud deployment, organizations must have profound knowledge of different cloud deployment models.

In the following sections, each type of cloud deployment model is explained briefly.

Public Cloud:

Public cloud is defined as providing computing services by the cloud service providers over the internet. Public clouds are environments where everything is managed and handled by external cloud service providers. When we think about computer clouds, it is nothing but a public cloud, and it took some time to develop various kinds of cloud deployment models. The public cloud model is the most used cloud deployment model.

Small and medium businesses use the public cloud as their primary IT source. Large organizations use the public cloud as an add-on to their in-house computing sources. A public cloud model would allow the company to choose any cloud services (Iaas, Paas, Saas) required **[1]**. One of the advantages of cloud computing is the capacity to offer both computing resources and storage as the need arises. Many organizations need to provide on-demand data storage as the volume of data accumulates rapidly.

Small and medium businesses can support their computing hardware and software needs using a public cloud. Since they are a pay-as-you-go operation, the companies can focus on their core strengths and using the cloud services required when the need arises, and pay for what is used

only. There is an unavoidable fluctuation in demand of large corporations and thus, investing in computer hardware is not the right way to meet peak demand. In such cases, for large companies' public cloud is convenient to use. For their non-sensitive data, large companies may use the public cloud. One of the significant considerations of large organizations in terms of storage requirements is exponential data growth. Data centers are vital to large businesses' success, and the public cloud serves the computing needs of data centers [1].

Technology has advanced dramatically in facilitating information sharing and rapid distribution using social networks of geolocation data. Many studies estimate that there will be a rapid growth of data in larger organizations as time increases, and it will put a dent in corporate budgets to keep pace with this form of storage demand. The need to ensure the security, privacy, and availability of stored data is associated with data storage. Since Amazon's Web Services storage solutions provide security, privacy, backup, and availability aspects through their cloud service S3 (Simple Storage Service), several major companies use the S3 cloud service. The pricing model used by Amazon makes it attractive to consider cloud storage for all sorts of companies, as it costs less than 10 cents per gigabyte of storage [1].

To make their services highly accessible, most public cloud providers already have the hardware, software, and staffing in place. To have improved availability, they will charge a little as much for the service, but it would not be close to the cost of doing it internally. Users need to ask the provider what service is being offered. If increasing availability is an addition, a user must know this when measuring the cost [2]. We can also ensure that the availability we want is part of a service-level contract (SLA). SLA will give the user a degree of confidence that their requirements for availability will be fulfilled.

Be aware that although public clouds can increase availability, make sure users know what will be open. It will rely on the offering of the service. The application itself will be available in the SaaS contract offer. However, if PaaS or IaaS are provided, the application may not be available while the platform or infrastructure may be accessible. Application issues cannot be mitigated by using the public offering of IaaS or PaaS [2].

It is a fact using the virtualization principle; public clouds host several tenants on one server. Customers feel that the lack of control over the hardware is a cause for concern, combined with the risk of someone else accessing their data. Since it is over an internet connection to access the public cloud, consumers are restricted to the speed they get from their ISP. Businesses are historically able to depreciate their assets, including computer assets. However, cloud storage use prevents the client from depreciating any computer assets since they do not own the infrastructure. Customers often believe they are locked into a cloud service, and due to a lack of standards, the cost of switching to a new provider is high.

Consequently, due to storage formats, any customer switching from one provider to another provider would find it difficult to use their data. Therefore, due to storage formats, any customer switching from one provider to another provider would find it difficult to use their data. For small companies that prefer to use SaaS rather than PaaS and IaaS, this is a bigger problem. Medium and large organizations using PaaS and IaaS have a higher degree of control over the storage elements. It would not be that difficult to switch to another cloud provider [1]. Amazon Elastic Cloud Compute (EC2), Google App Engine, Blue Cloud by IBM, and Windows Azure Services Platform are examples of public clouds.

Private Cloud:

The idea of the private cloud implies some form of ownership. It is true that when a private cloud is used, the client has a higher degree of control. Compared to the public cloud, this is a costly service. From the infrastructure and the system management viewpoint, it will be affordable only for large companies.

Four types of private clouds are available. In a typical private cloud, the company hosts the cloud behind the corporate firewall in one of their data centers. The architecture is like that of an intranet, where the company can use the internet strategies, but the internal employees only access cloud content. The organization uses cloud technology but restricts users to its internal employees. The second kind of private cloud is one that a third party manages. In this scenario, the company still owns the infrastructure in one of its data centers, but a third party operates the facility. In this scenario, private cloud is known as ‘managed private cloud’, meaning that the infrastructure belongs to the company but is managed by someone else. In the third model, a cloud service provider provides the infrastructure and manages the infrastructure required, called the ‘hosted private cloud’. The customer's advantage in this situation is that the cloud service provider promises scalability, demand elasticity, and availability. There is more protection since the servers are not shared with other organizations. The 'Virtual Private Cloud (VPC)' is the fourth model. The VPC service is delivered in a multi-tenant environment by a conventional cloud service provider [1].

1.Private cloud - hosted and managed by company: One of the key advantages of cloud computing is the economies of scale from which the cloud service provider can use virtualization to leverage the servers. The company's advantage is that vast computing resources are available to use and pay for as required. The company is supposed to benefit in the form of no investment in infrastructure costs. Using the private cloud mentioned above, some of these advantages are compromised because it must invest in the infrastructure. Using cloud concepts such as virtualization, the use of the word 'cloud' in 'private cloud' is justified [1]. Thus, with the planned demand in mind, the private cloud over provisions the infrastructure

and sets it behind the corporate firewall. Access is restricted to the organization's employees, and a great deal of automation is offered in the form of allowing virtual servers on their physical servers. It can better meet compliance criteria and provide a higher degree of protection than in a public cloud since the company manages both the infrastructure and access to the systems. The company can handle the private cloud because it has a vast IT workforce committed to addressing the private cloud. Large companies such as IBM, Cisco, and Verizon can be examples of private cloud users.

2.Managed private cloud: In this model, third-party cloud service providers manage the organization's infrastructure. The managed cloud service provider utilizes technologies like virtualization, thereby eliminating the possibility of multi-tenancy. There is a general myth that the private cloud will be more expensive relative to the public cloud. Nonetheless, several studies have shown that this is not the case. Based on the type of service in the cloud, costs are different for each deployment model [1]. Some examples of managed cloud providers are Rackspace, Microsoft, Century Link, Indiqus.

3.Hosted private cloud: In the third model, the 'hosted private cloud', the provider of cloud services, provides the company with dedicated servers, thereby avoiding multi-tenancy issues. This service offers all the advantages of cloud computing, such as scalability, demand elasticity, availability, and security. Hosted private cloud is sometimes also known as the leased private cloud. The organization pays a slightly higher price for the use of dedicated servers in this model [1]. Since the cloud service provider could not expect to allocate these dedicated servers for other purposes during idle times, the customer's utilization rate would be considered 100 percent of the server time, irrespective of any lower percentage use. Amazon EC2 Dedicated, IBM SmartCloud Enterprise, and Rackspace Cloud hosted private cloud providers.

4.Virtual private cloud: The cloud service provider offers virtual servers on multi-tenant hardware with VPN (Virtual Private Network) connectivity to their customers in the fourth model, the 'Virtual Private Cloud'. Therefore, when accessing their virtual servers, the client has a certain added degree of security. Compared to the other three private cloud types, this service is much less costly but more expensive than the public cloud. In this case, the third party would be a provider of bonafide cloud services with all the cloud service benefits, such as scalability, the elasticity of demand, availability, and storage. Using a VPN link, the company accesses the cloud to ensure security [1]. Amazon VPC, VMware, Microsoft Private Cloud, IBM SmartCloud Enterprise Plus, and Rackspace RackConnect are virtual private cud service providers.

Hybrid Cloud:

Hybrid cloud is a natural evolution of the two service models described in the preceding two sections. For some of the computing requirements, the hybrid model uses both proprietary computing resources that the company directly manages and the public cloud, especially those with varying resource demands [4]. The typical hybrid cloud utilizes the infrastructure as a service type to create the public cloud portion of the hybrid cloud. Since the company controls the cloud infrastructure usage, they can move applications between on-site and off-site. The hybrid cloud provides the cloud benefits of scalability, availability, demand elasticity, and pay-as-you-go models by making the architecture public cloud component [1]. Some examples of hybrid cloud providers are Microsoft, Google, Cisco, NetApp, Amazon, VMware, Hewlett Packard.

As new applications emerge, most companies today need the advantages of a hybrid cloud, and they need a way to test their viability before incorporating them with their systems and processes. Hybrid cloud management should also be an extension of private cloud management such that applications can be seamlessly transferred between the two elements of the company's hybrid cloud. Organizations are typically interested in monitoring their resources. Hybrid cloud technology using the Infrastructure as a Service model can provide the control that companies want. Many companies choose to use the private cloud for security and control considerations, as described in the previous section. The cost-benefit advantage does not apply to small businesses since private cloud customers are mainly large corporations [1].

One of the hybrid cloud's main advantages is that the organization can retain sensitive applications on-premise and transfer the other applications to the public cloud. The company is willing to give up some control over its IT resources using this mixed strategy. It must be noticed that IT companies strive to develop all the resources they need. However, the IT divisions are bypassed by business units in the case of cloud technology in accessing the third-party tools they need to achieve their goals [1]. The ability to evaluate third-party applications has been introduced by this fact, thereby opening the potential to accept new methods. In this way, a hybrid cloud can be said to enable organizational IT to be future-proof by creating an ability to implement new techniques. This mixture of private and public clouds opens the possibilities for developers of applications to develop new approaches that could be tested on a public cloud and ultimately implemented by private clouds.

It is vital to have the tools for the various development groups, IT staff, and the Quality Assurance team to interact well to find acceptance of hybrid clouds. Eucalyptus, an open-source program, is one such method. Its compatibility with Amazon Web Services is the appeal of Eucalyptus (AWS). This is an essential plus since AWS controls most of the public cloud

offering, and a hybrid cloud requires the ability to work well with AWS standards. Hyper Table is the other open-source software available for the Hybrid Cloud. Google has created the HyperTable concept. The primary objective of Hyper Table is to incorporate a high capacity, scalable, and distributed storage framework. Database processing is the main technology envisioned for HyperTable. The database approach is already moving to NoSQL from regular SQL. Traditional database lookups include maintaining a hash table without ordering the data on a key itself. The data is stored in the HyperTable approach based on a specific field, such as the URL. For fast lookup, HyperTable uses a multi-dimensional information table. Using a compiled code, the HyperTable is implemented, thus providing more incredible performance speed. Cloud infrastructure that offers on-demand servers to companies that follow a hybrid cloud strategy would leverage the cloud to use the HyperTable. Organizations do not have to invest substantial money on HyperTable as an open-source software [1] [5].

There are some management control issues in the hybrid cloud environment of internal IT and vendors. Businesses often face a service lock-in because there are no universal standards for cloud technology. This problem is significantly reduced using the IaaS service model. New platforms and niche players are increasingly emerging in cloud computing. Still, the lack of portability from one cloud provider or vendor to another makes it challenging to take advantage of potential opportunities [1]. It is hard to forecast the speed of change in cloud computing. It is becoming increasingly evident that hybrid cloud companies will rely on it because it provides the required leverage over some of their internally owned systems and benefits from cloud technology for non-sensitive applications. Another big issue of hybrid technology concerns jurisdictional limits. Conflict controls and legal requirements are a constant problem, as the cloud provider in one jurisdiction may not be aware of the legal requirements in another country or area. Besides, the vendor's security controls may not be in line with the company's internal requirements.

Community cloud:

When companies in a specific sector such as automotive, oil, financial, and health care realized that they need unique applications that are not relevant to other industries, the community cloud idea emerged. The community cloud provides the advantages of public cloud computing but is limited to a specific industry segment and a hosted private cloud's security features. Since the community cloud members are leading companies in that sector, the cloud services are mutually respected and secured. For example, a community cloud for the health care industry may concentrate on HIPAA compliance and the associated need for patient data security and privacy. A conventional email service such as Gmail or Yahoo Mail would not provide certain email forms with exceptional protection. For example, a community cloud conversation between two doctors will not be part of the general email community. It would

be able to provide greater security for the information as a result. People may also be aware that individual financial firms only allow their confidential communications accessible via their network to which the client must log in. However, via a public email system such as Gmail or Yahoo Mail, the financial company would alert the customer of the available communication [1].

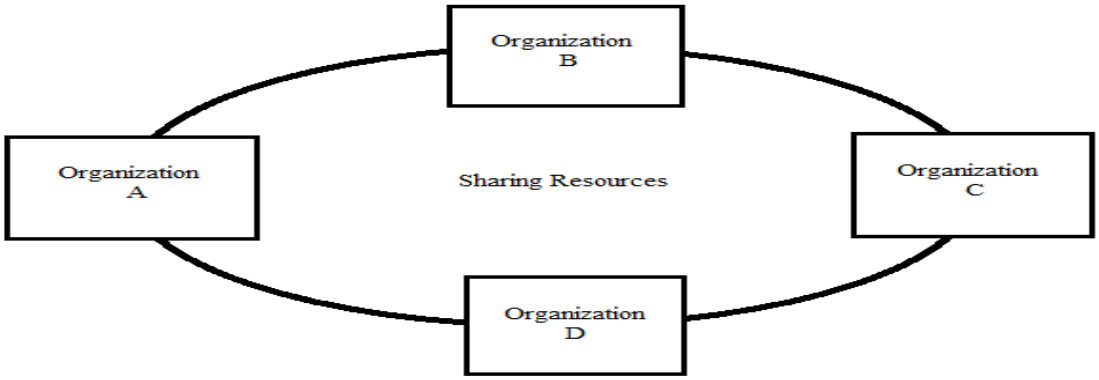

Figure 1 Federated cloud environment

Two basic types of Community Cloud models are Federated and Brokered. The underlying assumption is that companies only engage in this form of cloud in the same sector. A shared computing resource in one organization is accessible to another organization on-demand in the federated model. Figure 1 demonstrates a Federated cloud environment. A trusted third-party acts as the broker and interfaces with the different community cloud participants in the Brokered model. The broker is responsible for providing the industry sector's various services and providing all members with them. A Brokered community cloud is illustrated in Figure 2 [1].

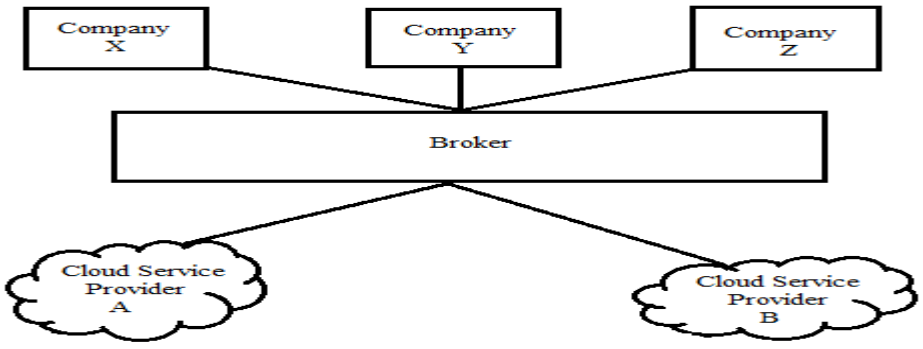

Figure 2 Brokered community cloud

The community cloud is a closed structure that is only open to users. The specific sectors listed above are broad enough to manage a group cloud. The in-depth study of the Community Cloud

concept shows that it has evolved from the techniques developed by vertical market segments, particularly the automotive sector. The essential advantage for community cloud organizations is that they would have more significant cost savings using applications needed in that sector. The Brokered model best serves this functionality than the Federated model [1].

The Federated community cloud model will allow member companies to share unused resources. However, the liability issues about such processing are not resolved, especially when there is a service interruption in the middle of a processing process. One example of Federated Community Cloud is a collaboration between the Mount Sinai Hospital of Toronto and the Canadian federal government to make available a fetal ultrasound application to the users of 14 area hospitals [1]. The benefits of a community cloud are seen in this example, hospitals produce 15 TB of data per year, and this growth is expected to rise 31 times by 2020. It becomes prohibitive to store this data internally if hospitals have such large quantities of data. Instead, the hospitals can save storage costs and share data with other hospitals using a community cloud.

In managing the accountability problems in the community cloud, the Brokered model is even better. The broker is responsible for contract settlements with the vendors, providing the participants with the necessary confidence to use the system, and resolving disputes. The broker would provide the members with the data required to fulfill their compliance obligations [1]. One question in the case of cloud services is how the cloud service provider can mitigate the risk. In a community cloud with a broker present, the participants need not be worried about this feature because the broker would take care of the risk mitigation. Besides, some additional value-added services will be offered by the broker. For example, in the financial sector, as it manages the purchase and selling of orders for stock traded in a stock exchange, communication needs to be low latent. Several Brokers serve the cloud computing community segment. Some include the United Health Care Optum Health Cloud for the healthcare industry, the International Gaming Technology IGT Cloud, and the global CFN financial services cloud.

Cloud computing is rapidly maturing, and many service providers are globally available. It offers suppliers an opportunity to differentiate themselves by concentrating their services on a particular sector in the community cloud. This accessibility of a vast network of vendors by industry in the community cloud segments provides an opportunity for industry members to take advantage of community cloud services with the added security features that are not present in a public cloud offering [1].

CHAPTER 3: CLOUD SERVICE MODELS

Introduction:

Cloud computing provides access to all necessary software via the Internet. Software as a Service (SaaS) is the only service that offers this feature, and this is the cloud's leading form of service. Platform as a Service (PaaS) can be chosen by medium-sized companies with the potential to provide computing expertise to their employees. By choosing different cloud provider platforms, PaaS offers the company the opportunity to select applications that fit their requirements best. Large and niche organizations decide to use only the cloud provider's infrastructure, thereby benefiting from the option of Infrastructure as a Service (IaaS).

Software as a service (SaaS):

Cloud computing can be summarized as SaaS, and SaaS offers to an enterprise both the server hardware and software without any complexities of operating an IT system. The simplest example of a SaaS service will be an organization's email. In 1996, Hotmail from Microsoft became the first such service. Email services from providers such as America Online and CompuServe were server-based before Hotmail. Because of its strength in that field, the cloud provider benefits from economies of scale in managing a vast infrastructure and can provide the requisite computing services at an affordable cost to customers; most of the customers are small and medium-sized enterprises (SMEs). SaaS leaves the provider in complete charge of the computing system. SaaS is also known as On-Demand software because companies pick the software they need from a host of cloud service providers' software. In 2003, IBM was the first to provide SaaS service. At the time, it was known as on-demand software. SaaS came into vogue in 2005 when Amazon introduced the Elastic Compute Cloud (EC2) [1].

Today, some big commercial SaaS providers are AWS, GCP, Microsoft Azure, SalesForce, Alibaba Cloud, and IBM.

Office applications like Microsoft Office, Adobe Photoshop, and other multimedia programs need constant upgrading for new features and patches and improvements for the software. Each of these components demands that time and energy be dedicated to management. It is a requirement for small and medium-sized companies to have a reliable IT resource, but it also detracts them from their core strengths [1]. Small companies need to remain consistent and continually develop their service to manage and expand their company. It would take a toll for them to divert their focus from their core strengths to managing the necessary IT infrastructure. This is where SaaS is convenient for small and medium-sized companies. In an organization, a service-oriented IT department will significantly benefit from shifting their attention from deploying the various software and maintaining them to managing the results of the different applications delivered by SaaS vendors.

SaaS has developed as a natural extension of the conventional distribution of software to companies. SaaS was a single-tenant service from the service provider in this early approach. This moved the software maintenance to the cloud provider but did not provide the user with the advantages of integration from running multiple cloud applications. SaaS helps the user function in a multi-tenant world with the growth of technology today, where the user can integrate the outcomes of many cloud applications [1].

Businesses know that SaaS is a strategic option which they should take to mitigate the risks of losing control over their applications. In the conventional model, software vendors offered their applications for a single-time charge, and consumers were responsible for installing their vendor updates. SaaS model, the software application is being provided over the cloud by the provider or a third party known as the aggregator. In this model, without the inconvenience of maintaining the program, the user pays an ongoing fee for using the software on a per-user basis [1]. The SaaS advantage also applies to large companies. However, in this situation, the company should have a software management system whereby a single group must clear all. There is a greater need for both on-premises applications and some SaaS applications in many companies. Until recently, organizations were worried about cloud-related security problems, and there was some resistance to SaaS applications. With the widespread use of SAS 70 audit features by cloud service providers, many organizations now seem to consider SaaS applications because they would have the requisite audit data to satisfy any requirements.

When it comes to using SaaS, companies find another advantage. Usually, to invest in capital expenditure, enterprises' budgeting process requires considerable lead time and often requires substantial IT time to execute the new application. Eighteen months is a fair approximation of this time. On the other hand, organizations typically spend less than two months introducing the new application using SaaS. The budgeting phase shifts to another region other than capital expenditure and accelerates the approval process. This simple implementation also has a potential drawback. An organization will identify unique applications it wants in the cloud, and they will be distributed rapidly. This could lead to the mushrooming of applications used by an organization, and it could be challenging to incorporate the outputs of these different applications. That is why companies should have a strategic vision for using SaaS to deploy their strategy as a quick solution [1].

We look at the Microsoft Office 365 suite of items' specifics, following up on the previous paragraph's theme. Microsoft's cloud service offers all the leading workplace management software products. The essential advantage of using Office 365 for an organization is that it comes with the applications that people are used to using on their own devices and provides storage for Office 365-generated documents. This role allows users to view their documents

from anywhere on different devices [1]. Today, access to documents on multiple devices is a must with the workforce being so mobile. Also, some of these devices, such as the iPhone, use different operating systems. An organization stands to benefit-cost and manage time by having the service offered by a cloud provider such as Microsoft to use Office 365 instead of controlling the program on all its workers' devices and from every location.

In considering cloud services, evaluate the pros and cons of maintenance of software and hardware. Small and medium-sized enterprises profit the most from SaaS. The SaaS technology services consist of more than just office productivity applications. For this purpose, a company may have to consider other aspects of Software as a Service (SaaS) related services in business services, such as inventory control and customer relationship management (CRM). On the cloud, Salesforce.com is the pioneer in CRM services. Businesses would have to determine the ease of interoperability between the various software applications they select on the cloud, an individual software. Office 365 could, therefore, not accommodate all businesses. One application that is worth noting in this regard is Zoho. Zoho has a free version that could be used for small and medium-sized companies like its main competitors, Salesforce.com and Google Apps. It is worth noting the approach taken by Zoho as it provides substantial integration of different types of applications such as Office products such as email, spreadsheet, inventory management, and business growth using simple tools to pursue customer leads. Big applications such as Salesforce.com and Google Apps also have such integration, but monthly at a per-user expense that can quickly add up for a small or medium-sized company [1].

More versatility in integrating multiple applications should be offered by web-based services such as SaaS. One such application for many businesses is web-based forms from which companies gather valuable information. Application software such as CRM can facilitate the integration of data collected via web-based services. The value of SaaS for SMEs was highlighted by the company being able to concentrate on its core strengths and rent IT services from a cloud provider. Because of the need to support multiple mobile devices for their employees, small and medium-sized businesses are faced with the problem of supporting applications on mobile devices. Companies like Zoho have recognized this need and build products that will help them remotely troubleshoot mobile devices for specialized applications. Apart from Salesforce.com and Zoho, BatchBook and SugarCRM are other similar CRM products. This software attempts to provide a niche service, such as seamlessly integrating with social media, to monitor customer leads via email [1].

The ability to incorporate vertical market applications is another significant benefit of cloud service. For example, cloud computing may reduce redundancy in implementing application software for healthcare industry organizations in health care or automotive enterprises [1].

Using economies of scale, a SaaS provider would provide all companies in the same sector, such as health care, with the necessary standard application software. Cloud computing can also help reduce redundancy in applying application software and its management in horizontal business applications such as payroll and customer relations management.

One of the significant concerns companies using Saas is data security. The service provider's employees would have direct access to these systems where the data is stored. The protection of the data at the software level is one way to mitigate this. The data at rest and the data in motion will have to be encrypted. This will prevent the provider from reading the data when stored or the provider's devices are stored and moving on the provider's network.

Platform as a service (PaaS):

PaaS offers a service that provides users with a platform to use for their computing needs. This framework is used for development in most instances. The development platform may simply be an operating system or a complete development platform that includes a web server and development libraries, depending on the provider. Compared to a SaaS user, the PaaS user must have adequate computer specialists to handle the platform they subscribe to. PaaS brings the same degree of flexibility regarding resource availability and demand elasticity that a cloud platform provides and suits the pay-as-you-go model [1] [3].

PaaS helps companies to develop and deploy Web applications without having to build their infrastructure. In general, PaaS services include development, integration, and testing facilities. When an organization moves to the implementation of PaaS, it will implement applications or services on the platform. In general, the vendor does not influence how the application or service is produced. Some Paas service providers offer load balancing facilities to their users to deploy their applications [3].

Some examples of Paas providers Google App Engine, Windows Azure, AWS Elastic Beanstalk, RedHat Open Shift, Oracle Cloud Platform, IBM Cloud Platform.

PaaS is well suited for designing, testing, and deploying new applications based on various platforms for large businesses and entrepreneurs. Because the cost of infrastructure uses the pay-as-you-go model, many developers can use their applications on several platforms. Given the ease of use for the end-user, applications for multiple concurrent users can be evaluated in an immersive way [1]. A great advantage for developers is this kind of load testing. Since all resources are accessible over the cloud, different user interfaces could be developed by developers. Since PaaS users build their apps on a test platform, testing may entail multiple users sharing applications while preparing for scalability and security. Another advantage of PaaS is that it enables the developer to form distributed teams that work on various aspects of their application simultaneously and grant different access levels to other users and monitor

their use patterns during the testing process. Thus, PaaS provides a scalable architecture for multi-tenants.

One of PaaS's main advantages is that it supports every application development's complete life cycle. PaaS features to keep the ability to collect user pattern logs and recognize any issues that arise when a real user attempts a new application. A developer must realize that the platform supports the application Lifecycle Management (ALM). The developer must recognize that it will be easy to incorporate future changes [1].

Significant aspects of being considered while choosing a PaaS provider:

1. Does the platform support multi-tenancy in architecture and applications?
2. What Application are Lifecycle Management applications supported?
3. What Applications are Programming Interfaces (APIs) supported?
4. Does the platform facilitate scalability?
5. What types of log data would be available for the user?
6. What programming languages are supported by the platforms?

Infrastructure as a Service (IaaS):

IaaS offers services such as computing power, storage, networking, and operating system [3]. Infrastructure as a Service (IaaS) provides the customer the same functionality as a service (PaaS), but the customer is entirely responsible for controlling the leased infrastructure. IaaS requires the organization to have the requisite individuals with comprehensive computing skills, unlike PaaS. IaaS is often referred to as "utility computing" as the organization requires computing resources but does not explicitly invest in it but acquires the resources to obtain a utility such as electricity and water. The IaaS users would have full responsibility for the system's security aspects, but the cloud providers would handle the physical security [1].

Examples of IaaS providers AWS, Microsoft Azure, IBM, GCP, Oracle cloud.

IaaS is usually used when a developer creates an application on a cloud service provider's virtual machines and customizes it by running it on several virtual machines to satisfy different clients' needs [1]. The large company can take advantage of virtual machines' availability and operate the VMs to run their specialized apps. IaaS is the costliest of the three services and is used by large companies. Large companies use IaaS as a supplement for their in-house computing. As mentioned above, using a VM environment, IaaS could also be deployed for some applications.

IaaS is useful for large companies to gain considerable computing resources without paying for it. Since the company owns the computing resources, it also has power over infrastructure. The primary reason for possessing greater computing power is to incorporate numerous

applications. For example, when SalesForce CRM is running on the SalesForce.com website, the hardware to integrate a part of the output from the SalesForce.com application to another application running in another cloud service provider may not belong to the company. These applications do not require to be in IaaS. They may be hosted PaaS or hosted SaaS applications. Thus, the company would have much need to test several integration scenarios, and it may need additional hardware that is unrestrained by corporate security policies [1].

The most challenging aspect of using IaaS will be security. Security elements typically require total control over the infrastructure. The company would have to cede the infrastructure's physical safety to the cloud service provider using IaaS. This is not a significant aspect, and because cloud service providers have a reputation to protect, they will be careful to provide the infrastructure with the requisite physical protection [1]. The company should then have the resources to hire their employees to tackle the requisite security plan. On the networking side, the real advantage of IaaS is the ability to deploy a variety of hardware from multiple locations to connect the organization's different segments spread across a wide geographical area. Since the service is a pay-as-you-go model, IaaS facilitates installing the requisite infrastructure.

To have burst capacity, organizations are also looking at IaaS providers. Only on some occasions do specific organizations need to be increased capacity. Customers do not want to spend on costly permanent solutions for that reason. Customers of an IaaS provider can add power temporarily. With IaaS, clients only pay when they need it for the increased capacity [3].

Backend as a service (BaaS):

Backend-as-a-Service is a cloud service model where developers outsource all aspects of a web or mobile application behind-the-scenes only to need to write and manage the frontend. For activities taking place on servers, such as user authentication, database maintenance, remote updating, push notifications (for mobile apps), cloud storage, and hosting, BaaS providers offer pre-written applications. The industry's move to outsource resources such as infrastructure, networks, applications, and the backend (i.e., IaaS, PaaS, SaaS, BaaS) has had a significant effect on the pace of marketing, user interface, affordability, and availability of the next central app. BaaS, like all members from the "as a service" family, builds on the premise that the product can be supplied on demand [6] [7].

BaaS vendor provides services like [7] :

- Storage: All BaaS providers will have data storage facilities. Some will provide schema-less, and others will be more structured abilities. Storage will be handled using APIs for any form of data. It needs the ability to query, sort, page, and filter.

More advanced features can include data connectors for on-premises data stores to sync cloud-based data with others.

- Dashboard: It is essential to manage data and users through some form of portal. While all the services can be performed through APIs, it is also time-saving and informative to have a UI. A nice feature is managing data, resources, users, authorization, and display reports.
- Database Management
- Cloud Storage for storing contents of user information.
- User authentication: In most applications, user provisioning and authentication are critical as apps need to control who can access what data and services are available. A typical example is that a doctor can only have access to their patients, not every patient in the program. A BaaS enables the user to configure user "groups" and then decide what information they should access through Access Control Lists (ACLs). Platforms need role-based user permissions and authorizations based on granular statements. 3rd party authentication is also helpful for sites such as Facebook, Twitter, Goggle, and others.
- Remote updating
- Analytics: The ability to monitor API calls is essential. This gives a viewpoint on who, when, and how much others are using BaaS. It is necessary to monitor results, failures, events, and the use of storage. A better user experience can be created by personalization for users.
- Hosting
- Push notifications.
- Geolocation

The list of service offerings should give what could be offered by a BaaS platform. This list is growing, and more redundant roles are being added. Think of the service providing in-app that involves custom coding and maintenance, making sense to shift the repetitive features to a standard backend framework [7].

BaaS helps developers to concentrate on writing application code for the frontend. They can integrate all the backend features they need through APIs (which are a way for a program to request another program) and SDKs (which are software building kits) provided by the BaaS provider without building the backend themselves. To keep the application going, they also do not have to handle servers, virtual machines, or containers. As a result, mobile apps and web applications (including single-page applications) can be developed and deployed more quickly [6].

One of the critical reasons developers prefer BaaS is that it takes time and resources to create their services, as processes need to be duplicated and then customized through multiple providers. We can quickly build on top with a consolidated BaaS, regardless of the operating system we connect. It is a part of the API-first movement that allows anything from a website to mobile apps to be developed more quickly on top of an API. Many like the BaaS strategy because it removes unnecessary stack setup and boilerplate repeat code, and the platform manages everything. An API backend, however, is not for all. It is data-centric, and it can generate more significant security risks if not handled correctly [7].

Some BaaS providers are AWS Amplify, Firebase, Back4App, and Parse.

CHAPTER 4: VIRTUALIZATION CONCEPTS

Virtual machines (VM):

A virtual machine is the virtual representation of a physical computer with its CPU, memory, storage, network interface. The virtual machine is an application or program running on a physical hardware system. It has its operating system, which is different from the physical machines' operating system. Virtual Machine is often called a guest, and the physical machine they run is known as the host.

A hypervisor is a software layer on top of the physical hardware machine that emulates the host's physical computing resources. It allocates processor, memory, storage to the guest operating system. Using the concept of virtualization, more than one virtual machine can be created on a physical computer or server, and each of them having its operating system. With the help of a hypervisor, each virtual machine is kept isolated from the other. Each VM has its virtual hardware mapped to physical hardware on the physical machine; therefore, the cost needed for physical machine and maintenance is reduced. A full justification for using VMs is server consolidation. When deployed to physical hardware, most operating system and device implementations only use a limited amount of the available physical resources. We can position several virtual servers on each physical server to increase hardware efficiency by virtualizing servers.

How do VMs work?

If a hypervisor is used on a physical machine or server, it allows a physical computer to isolate its operating system and its software from its hardware. Then, it can split into many "virtual machines" that are independent. Each of these new virtual machines can then independently run its own operating systems and applications while still sharing the physical computer's original resources (memory, storage, and RAM) that the hypervisor manages. From the physical environment to the VMs, resources are partitioned as needed. If the VM is in progress, the hypervisor schedules a request for the resources of the physical system to enable the operating system and applications on the virtual machine to access the shared pool of physical resources, and the user or software issues the instruction that requires additional physical environment resources [8] [9].

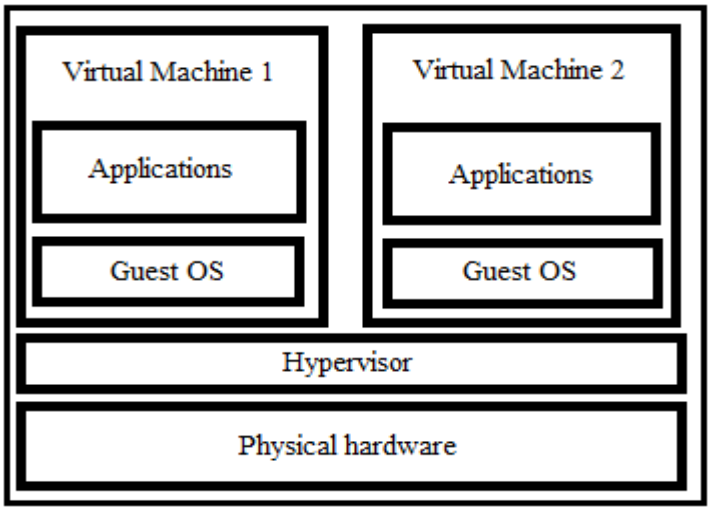

Figure 3 Virtualization on physical hardware

Advantages of Virtual Machines [8]**:**

- Resource utilization and improved ROI: Since several VMs run on a single physical device, customers do not; each time they want to run another OS need to purchase a new server, and with each piece of hardware they already have, they can get more returns.
- Scale: With cloud computing, several copies of the same virtual machine can easily be deployed to better support load increases.
- Portability: VMs can be performed on physical machines in a network. This allows workloads to be assigned to servers that have spare computing resources. VMs can also switch between on-premises and cloud environments, making them useful for hybrid cloud scenarios in which the data center and a cloud service provider share computing resources.
- Flexibility: It can be less difficult to build a VM than to install the OS onto a physical server. New environments can be built on demand by developers and software testers to manage new tasks when they arise.
- Security: When opposed to operating systems running directly on the hardware, VMs enhance protection in many ways. A VM is a file that an external program for malicious software can scan.

Hypervisor:

Until hypervisors entered the mainstream, only one operating system (OS) at a time could be run by most physical computers. This made them stable since requests from that one OS only had to be managed by the computing hardware. The downside to this technique was that it wasted energy because the operating system could not always use all the machine's power. A

hypervisor is a thin layer of software that runs on physical hardware that creates and runs virtual machines. The physical machine where the hypervisor is installed is known as the host machine, and the virtual machine installed on that physical machine is called a guest machine. The hypervisor, also known as Virtual Machine Monitor (VMM), manages the multiple VM's running on the host machine.

Hypervisor emulates host machine resources like CPU, memory, and storage to use the guest machine. It isolates VM's from each other; they do not interfere with each other; if one VM is crashed on the host machine, it does not affect the remaining VM's on the host machine. The hypervisor allocates resources to each virtual machine and schedules physical resources between VM's. In multiple VM's, hypervisors schedule requests from VM's to CPU of the host machine to be executed. Hypervisors allow switching VMs on various physical machines between hypervisors without stopping them, which can be useful for both fail-over and workload balancing.

There are two different types of hypervisors:

1. Type 1 - native or bare-metal hypervisors
2. Type 2 – hosted hypervisors

Type 1 – native or bare-metal hypervisors:

Type 1 hypervisors are software running directly on physical hardware or server; hence they are called bare-metal hypervisors. Type 1 hypervisors behave like an operating system and manage underlying hardware resources, as required by VM's. Since they have direct access to physical hardware, type 1 hypervisors are highly efficient. This also improves their security since there is nothing that an attacker might compromise between them and the CPU. In a data center or other server-based environments, this type of hypervisor is the most common.

Examples of type 1 hypervisors: VMWare vSphere with ESX/ESXi, KVM (Kernel-Based Virtual Machine), Microsoft Hyper-V, Oracle VM, Citrix Hypervisor.

Type 2 – Hosted hypervisors:

Type 2 hypervisors run on a physical machine with an operating system as an application or program are called hosted hypervisors. They are used in individual PC rather than server-based environments. Type 2 hypervisor is ideal for individual users who choose to run multiple operating systems on a personal computer. Type 2 hypervisors are useful for evaluating new applications and research projects.

Examples of type 2 hypervisors: Oracle VM VirtualBox, VMware Workstation, VMware Fusion.

Containers:

Containers can provide a virtual environment that brings together the application processes, metadata, and file system, all of which an application needs to run. However, containers do not need their own OSs, unlike virtual machines. Instead, they are only wrappers around a process of UNIX that communicates directly to the kernel to request and use the resources.

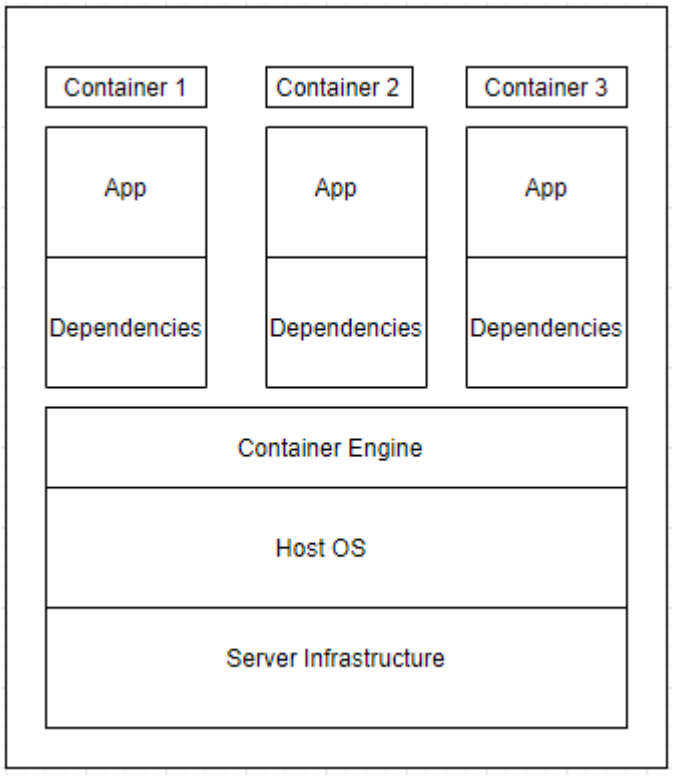

Figure 4 Container Architecture

Containers explicitly provide application and process separation, as seen from the above figure, where one application is unaware of another application's presence. Nevertheless, all processes run on the same kernel used by the OS and share it. How is this happening? To allow independent processes to run within a single Linux instance, containers use resource isolation features of the Linux kernel, such as control groups and namespace. This goes back to why, as VMs do, each application does not have its OS. This also means that than containers provide, virtual machines provide more splendid isolation [10]. That is what makes containers very lightweight, though, making them simple to ship and move around. More containers can run on a given hardware combination because of containers' lightweight design than to run VMs.

These containers are also referred to as LXCs or Linux containers. The idea of containers has been around forever, but due to Docker, it has only recently gained significant popularity. Docker is an open-source project that has introduced several modifications to Linux-based containers to make them more portable, easy to use, and versatile. It did this by introducing a

collection of utilities that allow portability and flexibility for the containers. These utilities allow users to build, ship, copy and run containers quickly [10].

Pods:

Pods are the most atomic unit of work in a cluster of Kubernetes. A Pod contains one or more containers installed together on the same machine, allowing local data sharing mechanisms (Unix sockets, TCP loopback devices, and even shared directories backed up by memory) to be used for faster communication. By preventing a network roundtrip, containers clustered in Pods achieve quicker communication and use shared resources such as file systems. The main feature of a Pod is that once deployed, the same IP address and port space are shared. The Pod deployed in Kubernetes is not an "image" or "disk" but an actual, tangible, CPU-cycle-consuming, network-accessible resource [11].

Functions:

The developer develops and runs custom event-driven code on stateless, ephemeral containers created and managed by a third party known as Function as a service. Using programming languages such as C#, Python, etc., FaaS can implement a feature built by a developer. With the support of triggers, these functions are executed based on an event-driven model. Function as a Service (FaaS) allows developers to isolate and host small functional blocks of complex multi-tiered frameworks as functions that can scale independently. This method is cost-effective since, rather than scaling the whole program, individual functions can be scaled based on their load. FaaS differs from the conventional monolithic design, packed as a single unit for the entire application. In comparison, as compared to a microservices architecture, FaaS goes one step further in breaking down the framework into small features [12].

Serverless Computing:

A modern approach to cloud computing that allows programmers to run event-driven functions in the cloud without the need to handle resource allocation or configure the runtime environment is serverless computing, also known as functions as a service. Instead, the cloud platform automatically handles dependencies, compiles code, configures the operating system, and manages resource allocation when a programmer deploys a serverless function. Serverless computing enables programmers to concentrate solely on application code, unlike virtual machines or containers, which require low-level system and resource management [13].

If demand for the function unexpectedly increases, the cloud platform will load another instance of the function on a new machine transparently and handle load-balancing without the programmer's intervention. Conversely, the platform will transparently terminate underutilized instances if demand for the feature falls [13]. If there is no demand for an extended time, the cloud provider will shut down all running instances of the function. Because the platform in

this way entirely handles the operating system and resource allocation, serverless computing is a language-level abstraction for cloud programming.

An economic benefit of serverless functions is that they only incur costs for the processing events spent on time. A function that is never invoked, therefore, does not incur any cost. By comparison, when they are idle, virtual machines incur costs, and they need idle power to accommodate unexpected rises in demand smoothly. Serverless computing enables cloud systems to optimize resource distribution more efficiently among many users, improving hardware and decreasing costs, e.g., reducing energy usage [13].

CHAPTER 5: OPENSTACK

Introduction to OpenStack:

OpenStack is an open-source cloud computing operating system that manages vast pools of computing, storage, and networking resources in a data center [15]. During the first months of 2010, OpenStack was created. Rackspace wanted to rewrite the infrastructure code running its offering of cloud servers and considered open sourcing the current code for cloud files. At the same time, Anso Labs (a NASA contract) released the beta code for Nova, a "cloud computing fabric controller" based on Python. Both efforts converged and established the basis for OpenStack. The first Concept Summit was held on July 13-14, 2010, in Austin, TX, and the project was formally announced on July 21, 2010, at OSCON in Portland, OR [15].

OpenStack is a collaborative initiative between Rackspace and NASA to make cloud computing more ubiquitous. It is deployed as Infrastructure-as-a-service (IaaS) in both public and private clouds. It utilizes integrated modules that manage different hardware types such as computer processing units, storage, and network resources via a data center. A dashboard is also available, giving administrators control while empowering their users via a web interface to provide resources [15]. Beyond standard infrastructure-as-a-service features, additional modules include orchestration, fault management, and service management, among other services, to ensure user application's high availability.

OpenStack is an open-source software project with many contributors from a wide range of businesses. Currently, Rackspace remains a significant contributor to OpenStack. Even contributions to the project today come from a wide variety of organizations, including conventional open-source contributors (Red Hat, IBM, and HP) and companies that are entirely dedicated to OpenStack (Mirantis and CloudBase). There are contributions in the form of drivers for specific infrastructure pieces (Cinder block storage drivers and Neutron SDN drivers), bug fixes, or new functionality in core projects [14].

OpenStack was initially developed to provide applications programming interface (API) compatibility with the Amazon Web Services. According to the November 2014 user survey, 44% of production deployments used EC2 Compatibility APIs to communicate with the framework [14]. As the platform's popularity has evolved, the OpenStack API has become a de facto standard of its own. OpenStack is written in Python, and it is deployed on Linux operating system. Many enterprise organizations use OpenStack clouds for underlying Infrastructure as a Service layer for one or more platforms as a Service or Hybrid Cloud deployments.

OpenStack offers a common platform for monitoring clouds of servers, storage, networks, and more. OpenStack is accessible from a web-based interface, a command-line interface (CLI),

and an application programming interface (API). This platform controls resources and does so without requiring a user to choose a single hardware or software manufacturer. With minimal effort, vendor-specific elements can be substituted. For a wide variety of people in IT organizations, OpenStack offers value [16].

OpenStack controls many commercial and open-source hardware and software, including a layer of cloud management on top of vendor-specific services. Repeated manual activities such as disk and network provisioning are automated with the OpenStack platform. The whole process of provisioning virtual machines and even applications can be automated using OpenStack architecture. Users can do anything from merely provisioning virtual machines (VMs) like AWS to running complex virtual networks and applications, all within an isolated tenant (project) [16]. Tenants, also known as projects, are the way that OpenStack isolates allocations of resources. Tenant isolation includes storage, network, and VM isolation, so that much more flexibility can be provided to end-users than in conventional virtual server environments. Imagine a quota of services being allocated to end-users to provision how and when they want it quickly.

OpenStack Components:

The framework of Open Stack is modular. OpenStack adopts a modular architecture to provide a set of core services that promote scalability and elasticity as core design principles. The OpenStack implementation includes a range of components that provide APIs for accessing infrastructure resources.

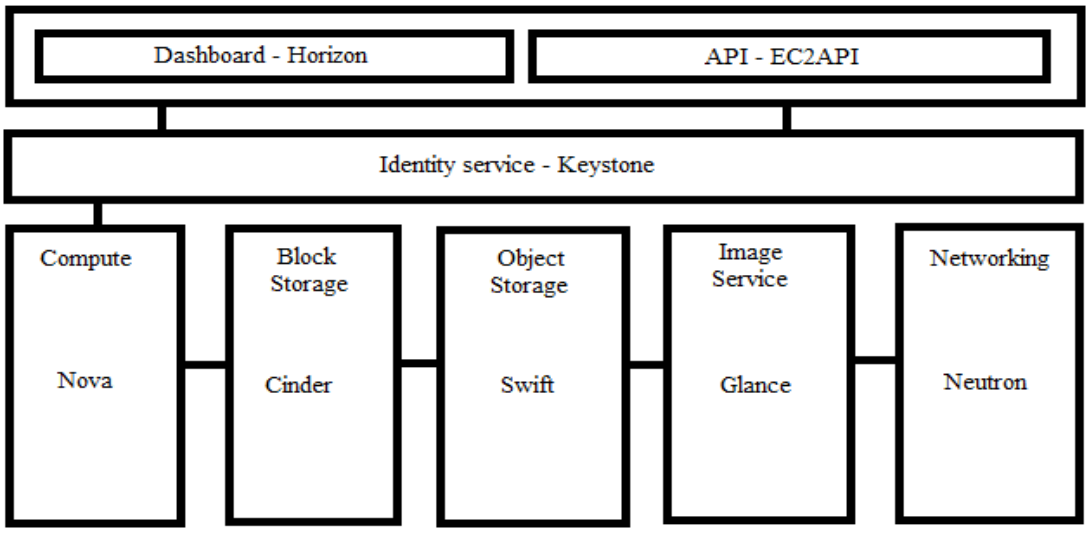

Figure 5 Components of OpenStack

OpenStack Compute (Nova):

OpenStack Compute (Nova) is one of the primary OpenStack elements. It provides the ability, depending on the configuration, to provision a virtual machine, an application container, or a physical device. It handles computing resources, such as creating, deleting, and managing scheduling. It is a resource automation application responsible for the virtualization of resources and high-performance computing. Nova supports the development of virtual machines, bare-metal servers and has minimal support for device containers. Nova runs as a collection of daemons on existing Linux servers to provide this service. Nova integrates with other OpenStack services to provide persistent block storage, encryption of disks, and bare-metal compute instances.

Nova requires services like Keystone, Glance, and Neutron for its basic functionality to provisioning virtual machines and image-based containers. Keystone is an identity service responsible for all forms of OpenStack service authentication and authorization. It is a directory-based service that maps the right resources to the correct user using a central repository. Horizon is responsible for providing a web-based interface for OpenStack and manages monitors and provision cloud resources.

The OpenStack image service known as Glance is a prerequisite for computing service. Glance image services provide the discovery, registration, and retrieval of images from a virtual machine (VM). Glance has a RESTful API that enables VM image metadata to be queried and the actual image to be retrieved. VM images accessible via Glance can be stored in several places, from basic file systems to object storage systems such as the OpenStack Swift project. OpenStack's Compute service initially supported networking, and some significant deployments still use the Nova service's networking features. The Neutron service is used for most current implementations.

We refer to provisioned computing nodes in OpenStack as instances and not as virtual machines. While this might seem like a semantic matter, for many reasons, it is a useful device. The first reason the deployment process is described-all computing in OpenStack is the instantiation of a Glance image with a defined hardware template, the flavor [14]. The flavor specifies the instantiated image's features-it typically represents several compute cores with a set amount of memory and storage. Storage can be provided by the Compute Service or can be provided by the Cinder Block Storage Service. At the same time, quotas are specified to restrict the number of cores, memory, and storage available to the given user(tenant).

The second reason why the term instance is useful is that OpenStack virtual machines usually do not have the same life cycle as conventional virtualization. Although virtual machines could be expected to have a multi-year life cycle like physical machines, we might expect instances

to measure a life cycle in days or weeks. Virtual machines are backed up and retrieved while instances are rescued or evacuated [14]. A resize operation could occur without downtime on a virtual machine, whereas a resize operation on an instance is a new deployment and a migration. This is because of the architectural variations between conventional virtual machines and their hypervisors and OpenStack. The legacy virtualization platforms assume resizing and modification behaviours in-place. In contrast, OpenStack cloud platforms anticipate redeploying the virtual machines or additional capacity by additional instances and not adding additional resources to the current virtual machines.

The third reason we find it beneficial to use the word instance is that the Compute service has developed to launch various computer types over the years. Some OpenStack deployments can only launch physical devices, while others may launch a mixture of physical, virtual, and container-based instances. Regardless of the computing provider, the same construct applies. Some of the distinctions between virtual machines and instances are becoming more blurred as the OpenStack Compute service introduces more enterprise functionality [14].

OpenStack Network (Neutron):

The last foundational service in OpenStack is Neutron, a network service. Neutron is an OpenStack project that offers "network connectivity as a service" between interface devices (e.g., vNICs) operated by other OpenStack services (e.g., nova). Neutron provides an API to build ports, routers, networks, and subnets. Add-on network services such as firewalls and load balancing are provided in some OpenStack deployments [14].

Neutron is an API frontend (and a collection of agents) that manages the user infrastructure of software-defined networking (SDN). It means that each tenant will build isolated virtual networks when an OpenStack implementation uses Neutron. It is possible to link each of these isolated networks to virtual routers to establish routes between them. A virtual router may have an external gateway connected to it [17]. Each instance can be granted external access by associating an instance with a floating IP on an external network. Neutron then places all the configurations to route the traffic sent to the floating IP address into a launched instance via these virtual network resources. The ability to provide networks and network services through software on demand is NaaS.

By default, to orchestrate the underlying virtualized networking infrastructure, the OpenStack distribution we will install uses Open vSwitch. Open vSwitch is a managed, virtual switch. Open vSwitch can be the infrastructure designed to separate virtual networks for tenants in OpenStack, as long as the nodes in a cluster have easy access to each other. Several vendor plugins allow substituting a physically controlled switch for Open vSwitch to handle the virtual networks. Neutron also can handle multiple network appliances using various plugins. In an

OpenStack implementation, Open vSwitch and a vendor's device may be used in parallel to control virtual networks. This is a perfect example of how OpenStack is constructed to give its users flexibility and choice [17].

Block Storage Service (Cinder):

OpenStack workloads are provided with traditional persistent storage through the Cinder block storage component. Cinder is the OpenStack Block Storage service that offers volumes to Nova virtual machines, Ironic bare metal hosts, containers, and more. Cinder volumes' life cycle is managed independently of computer instances, and volumes may be attached or detached to one or more computer instances to provide a file system-based storage backup store. Some of Cinder's ambitions are to be/have: [15]

- Component-based architecture: Incorporate new behaviors easily.
- Highly available: Scale to very severe loads of work.
- Fault-Tolerant: Isolated processes prevent failures from cascading.
- Recoverable: It should be quick to diagnose, debug, and rectify failures.
- Open Standards: Be a guide for community-driven API implementation.

Cinder manages screenshots as well. Users may take snapshots of the block volumes or the instances. These instances can also be used as a boot source for these snapshots. A comprehensive set of storage backends can be configured as the backup store for volumes and snapshots from Cinder. The Logical Volume Manager (LVM) is configured by default. GlusterFS and Ceph are two standard storage solutions that are based on software. For hardware appliances, there are also several plugins.

Object Storage Service (Swift):

It is an object storage service with high fault tolerance capabilities and supports Restful API to retrieve unstructured data objects. As a distributed network, it is often used to provide redundant storage within clustered servers. It is capable of handling petabytes of data successfully.

The Nova service offers ephemeral backup storage for device cases. This storage is temporary since the life cycle co-terminates with the computational instance's life cycle. When an instance terminates, the ephemeral storage associated with the instance is evacuated from the compute host on which it resides. The first kind of persistent storage offered in the OpenStack framework was object storage based on S3 by AWS [14].

Object storage is a simple system for content-only storage. Files are maintained without the metadata of a block file system. They are all containers and files. The files are just content. As part of its deployment, Swift has two layers: the proxy and the storage engine. The proxy is the

layer of the API. This is the service through which the end-user interacts. The proxy is configured to speak to the storage engine on behalf of the user, and the storage engine is a Swift storage engine by default. It is capable of distribution and replicating software-based storage. GlusterFS and Ceph are standard storage backends for Swift as well. They have similar delivery and replication capabilities to Swift's storage capabilities [17].

OpenStack Orchestration Service (Heat):

The orchestration part is Heat. Orchestration is the process of launching many instances that are meant to function together. Orchestration uses a file known as a "template," which determines what will be launched. Orders or dependencies may also be set up between the instances in this template. In these models, data that needs to be transferred between instances for configuration can also be specified [17]. Heat is also compatible with templates from AWS CloudFormation and, in addition to the template language of AWS CloudFormation, incorporates additional features.

One of these templates is written to describe a collection of instances to be launched to use Heat. When a template is launched, it generates a virtual resource set (instances, networks, storage devices) called a stack. When a stack is spawned, the configuration and the dependencies, shared configuration data, and post-boot configuration are coordinated through Heat.

OpenStack Telemetry Service(Ceilometer):

The Ceilometer is a component of telemetry. It gathers measurements of resources and is capable of monitoring the cluster. The Ceilometer was initially developed for billing users as a metering device. As it was being designed, there was a realization that it would be more useful than just billing and turned into a general telemetry device [17]. In an OpenStack implementation, ceilometer meters calculate the resources being used. It is called a sample when the Ceilometer reads a meter. Regularly, these samples are registered. A statistic is called a set of samples. Telemetry statistics can provide insights into how an OpenStack deployment's resources are used. The samples can be used for alarms as well. Alarms are nothing but monitors watching to satisfy specific criteria. Initially, these alarms were designed for Heat Autoscaling.

OpenStack Architecture:

Conceptual architecture:

Openstack is an entirely open-source combination of software projects initially developed by NASA and Rackspace, with such robustness and reliability as an IaaS solution for public and private cloud infrastructures. Its modular and highly configurable architecture allows this solution to be configured and customized to suit

the hardware resources available, whether professional or entry-level, with few or more computer nodes, especially in response to each case [18]. This enables companies to select from several complimentary services to satisfy various computing, networking, and storage needs. The Openstack conceptual architecture with all native software components, created by businesses and individual supporters, is depicted in Figure 5, demonstrating how they communicate with each other.

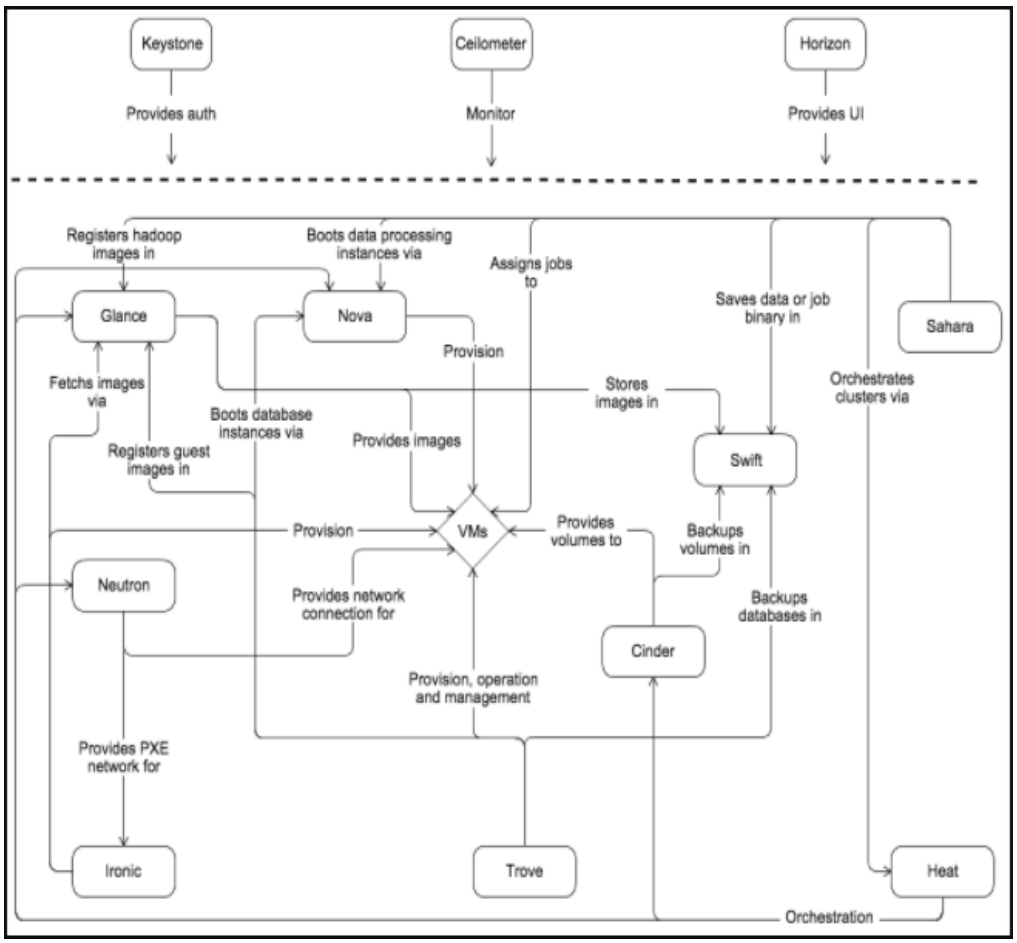

Figure 6 Conceptual architecture of OpenStack [15]

Logical architecture:

OpenStack consists of several independent sections, called OpenStack services, as seen in Conceptual Architecture. Authentication of all services through a shared identity service. Individual services, except where privileged administrator commands are required, communicate with each other through public APIs. OpenStack services are made up of many processes internally. All services have at least one API process that listens, pre-processes, and moves on to other parts of the infrastructure for API requests [15]. With the exception of the Identity service, the actual work is done by distinct processes. An AMQP message broker is used for communication between the processes of one service. The status of the service is stored in a database. We can

choose from many message broker and database solutions, such as RabbitMQ, MySQL, MariaDB, and SQLite when deploying and configuring the OpenStack cloud. Users can access OpenStack through the Dashboard's web-based user interface, through a command-line interface, and through tools such as browser plug-ins or curls to issue API requests. Several SDKs are accessible for applications. All these access methods eventually issue REST API calls to the different services of OpenStack. The following figure shows the architecture for an OpenStack cloud:

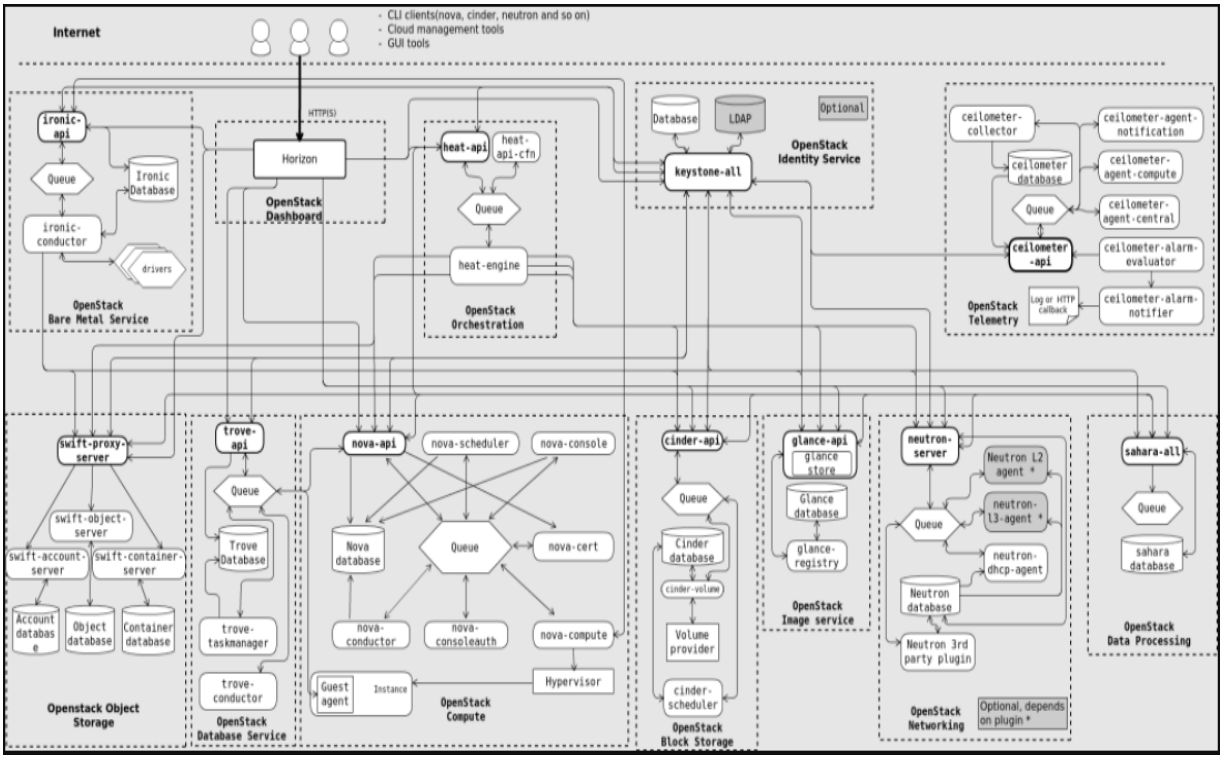

Figure 7 Logical architecture of OpenStack [15]

General-purpose architecture by RedHat:

General-purpose architecture can deploy a generally high-availability cloud and flexible architecture type that does not emphasize any single OpenStack component and is not limited to specific environments [19]. 80 percent of possible use cases are covered by this architecture category, including:

- Database
- The runtime environment of web application
- Environment for shared application development
- Testing Environment
- An environment that needs scale-out rather than scale-up additions

Three Controller Nodes and at least eight Compute Nodes are included in the architecture for this example. It uses OpenStack Object Storage for static objects and

all other storage needs, OpenStack Block Storage. The nodes use the Pacemaker(resource managing software) add-on for Red Hat Enterprise Linux and HAProxy(load balancer and proxy server) to ensure that the OpenStack infrastructure components are highly accessible.

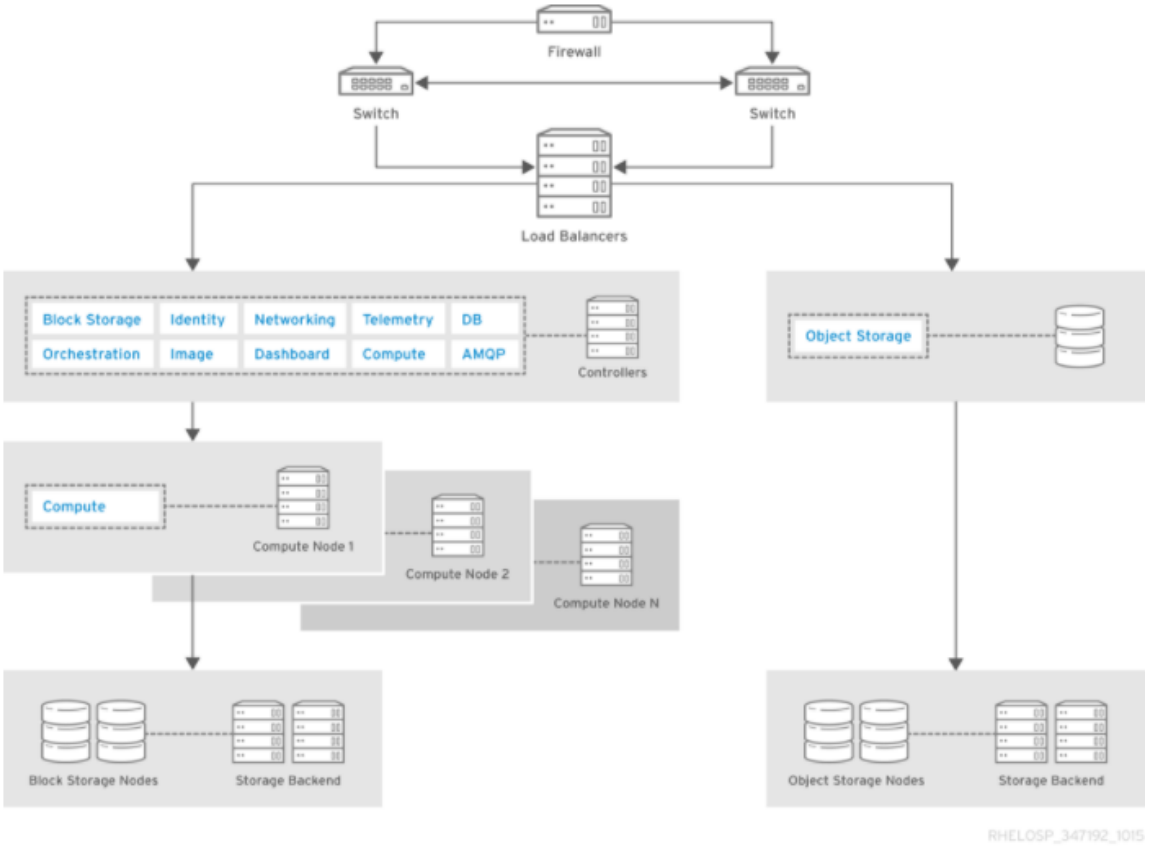

Figure 8 General-purpose architecture of OpenStack by RedHat [19]

Components in the above architecture: [19]

- For public-facing network links, firewalls, switches, and hardware load balancers.
- The OpenStack controller service runs Image, Identity, and Networking, together with the support services MariaDB and RabbitMQ. These services are configured on at least three controller nodes for high availability.
- Cloud nodes are designed with high availability with the add-on Pacemaker for Red Hat Enterprise Linux.
- Compute nodes use OpenStack Block Storage for instances where there is a need for persistent storage.
- OpenStack Object Storage is used to support static objects, such as images.

Description of each component in the above architecture: [19]

- Block storage provides persistent images for instances.
- Compute controller provides services for computing management and scheduling that run on the controller.
- The Dashboard is used as a web console for OpenStack management.
- Identity service for basic authentication for users and tenants and authorization.
- Image service stores images that can be used to boot instances and handle snapshots.
- MariaDB is a database for all components of OpenStack. MariaDB servers retain their data on enterprise shared storage, such as NetApp and Solidfire. When a MariaDB instance fails, another instance of MariaDB must be joined to the Galera cluster.
- Networking service controls hardware load balancing with plug-ins and networking APIs. If we want to grow OpenStack Object Storage, we should plan for network bandwidth. It is recommended to run OpenStack Object Storage on 10 GbE or higher network connections.
- Object storage is used for processes and archive logs from web application servers. Object Storage to transfer static web content from the OpenStack Object Storage container or backup images handled by OpenStack Image.
- Telemetry is used for monitoring and reporting for other services in OpenStack.

CHAPTER 6: OPENSHIFT

Introduction to OpenShift:

OpenShift is a family of software products developed by Red Hat for containerization. The OpenShift Container Framework is its flagship product, an on-site platform designed around Docker containers, organized and operated by Kubernetes on a Red Hat Enterprise Linux basis. The other family products provide this framework across various environments: OKD acts as the upstream community-driven platform, OpenShift Online is the software-as-a-service platform, and Openshift Dedicated is the managed service platform.

The Red Hat OpenShift Container Platform provides a hybrid cloud application platform for developers and IT organizations to deploy new and existing applications on a stable, scalable infrastructure with minimal configuration and overhead management. The OpenShift Container Platform offers various programming languages and frameworks, such as Java, JavaScript, Python, Ruby, and PHP. [20]

The OpenShift Container platform puts Docker and Kubernetes together and provides the management of these resources with an API. Based on Red Hat Enterprise Linux and Kubernetes, the OpenShift Container Platform offers modern enterprise-class applications with a more secure and scalable multi-tenant operating system while providing integrated application runtimes and libraries [20]. The OpenShift Container Platform helps organizations meet security, privacy, compliance, and governance requirements.

OpenShift runs applications in containers of Docker. Docker containers package up a piece of software in a full file system that includes all it requires to run: code, runtime, system tools, and system libraries - all that can be installed on a server. This ensures that it will always run the same, regardless of the context in which it is running. The open-source project OpenShift is developed on top of Kubernetes. The core container orchestration functionality is supported by Kubernetes, including features such as resilience to component failure, monitoring, automatic container rescheduling, and horizontal scaling [22].

The OpenShift umbrella is made up of three projects that coexist together. Although the three subprojects represent the users, developers, and the community in various ways, they contribute to bringing the OpenShift technology to the end-users, developers, and community members [22]. This project is defined in the following way [22]:

1. **OpenShift Origin:** OpenShift Origin is the open-source version of OpenShift provided by the Apache License 2.0 community. Openshift is an upstream feeder project for both OpenShift Online and OpenShift Enterprise.

2. **OpenShift Online**: OpenShift Online is the public-controlled edition of OpenShift. This Amazon EC2-based service runs a version of OpenShift Origin, which has been hardened and stabilized.
3. **OpenShift Enterprise**: OpenShift Enterprise is a wholly funded, private PaaS solution from Red Hat that can run on company hardware. By implementing OpenShift Enterprise, businesses may improve their agility when meeting their business application demands.

Ways to interact with OpenShift:

1. **RHC command line:** RHC client command-line tool is installed as a command-line interface on a local computer to talk to OpenShift. The RHC client command-line code is written in Ruby and is available as a Ruby gem. Using this technique is the most efficient way to access all of the features of OpenShift.
2. **Web console:** This is the best way to get started with OpenShift, as no computer program needs to be installed. The web console allows to log in and start developing applications.

Containers in OpenShift:

Container platform is an application platform that uses containers to create, deploy, serve, and orchestrate applications running within it. OpenShift uses two essential tools to support container applications: a container runtime to create Linux containers and an orchestration engine to control a cluster of servers running containers.

Container runtime: A container runtime works for container development and management on a Linux server. We may think of containers as discrete, portable, scalable application units. Containers carry all that the applications within them need to function. It contains all the libraries and code available for the application to operate appropriately any time a container is deployed [23]. Programs running within a container can only access the resources in the container. Applications in the container are isolated from other containers running on the host and from applications running elsewhere in the container. The following five types of resources are isolated for each container:

- Mounted filesystem
- Shared memory resources
- Domain name and hostname
- Networking resources
- Process counter

Docker is the service that is used to handle containers in OpenShift. Docker is an open-source, large, active project launched by Docker, Inc. Docker's resources to separate processes in containers are all available as part of the Linux kernel.[23] These resources include items such as SELinux, namespaces for Linux, and control groups (cgroups)(Control groups (cgroups) are used for limiting, accounting, and monitoring the resources (i.e., memory, CPU, I/O, checkpoint) of the process array). In addition to making these tools much easier to use, Docker has also introduced many features that have increased its popularity and growth:

- Portability
- Image reuse
- Application-centric API
- Ecosystem - Docker, Inc. maintains a free container image hosting environment for the public.

Container orchestration:

Although the docker engine manages containers by facilitating Linux kernel resources, it is limited to a single host operating system. Although it is fascinating to have a single server running containers, it is not a platform for creating robust applications. We have to deploy application containers across several servers to deploy highly accessible and scalable software. We need to use a container orchestration engine to orchestrate containers efficiently across multiple servers: an application that manages a container runtime across a cluster of hosts to provide a scalable application framework. OpenShift uses Kubernetes as its container orchestration engine. Kubernetes is a Google-launched open source project that was donated to the Cloud Native Computing Foundation in 2015.[23]

Kubernetes employs a master/node architecture. The Kubernetes master servers keep the server cluster information, and the nodes run the actual application workloads. Kubernetes is a useful open-source project. It is consistently one of GitHub's most active projects.

Interaction between components in different layers of OpenShift:

When deploying an OpenShift application, the request begins with the OpenShift API. We need to take a more thorough look at how these services work together to deploy the application to understand better how containers separate the processes inside them. A chain of dependencies is the relationship between OpenShift, Kubernetes, Docker, and, finally, the Linux Kernel. Deploying applications starts with components of the framework that are unique to OpenShift. The process is as follows [23] :

1. Using deploying source code and the builder image template defined, OpenShift will generate a custom container image. For instance, the PHP builder image is used by app-CLI and app-GUI.

2. This image is uploaded to the image registry in the OpenShift container.

3. A build configuration is created by OpenShift to record how an application is designed. This includes the image made, the builder image used, the source code's location, and other details.

4. OpenShift provides a configuration deployment to monitor deployments and install and upgrade applications. Replica numbers, update methods, application-specific variables, and mounted volumes are all specified in deployment configs.

5. OpenShift produces a deployment that describes a single version of an application that has been deployed. Each unique application deployment is associated with the configuration deployment portion of an application.

6. The internal load balancer of OpenShift is modified with an entry for the application DNS record, and this entry will be linked to a component that Kubernetes developed.

7. OpenShift generates a part of the image stream. An image stream in OpenShift monitors the builder's image, deployment configuration, and other components for changes. When a change is detected, image streams will cause the redeployment of applications to represent changes.

OpenShift, Kubernetes, the Docker, and Linux kernel are the automated workflow performed when deploying an application in OpenShift. The relationships and dependencies cover multiple resources, as defined in Figure 9.

When a developer uses the OpensShift-CLI command-line tool to generate source code and initiate the new application deployment, OpenShift generates a deployment config, image stream, and build config components. The build config generates a custom container image for the application with the required builder image and source code. This file is stored in the image registry of OpenShift. The deployment configuration portion generates a single application deployment for each application version. The image stream is generated and monitored in the internal registry for changes to the deployment configuration and associated images. The DNS route is generated and connected to an object from Kubernetes [23].

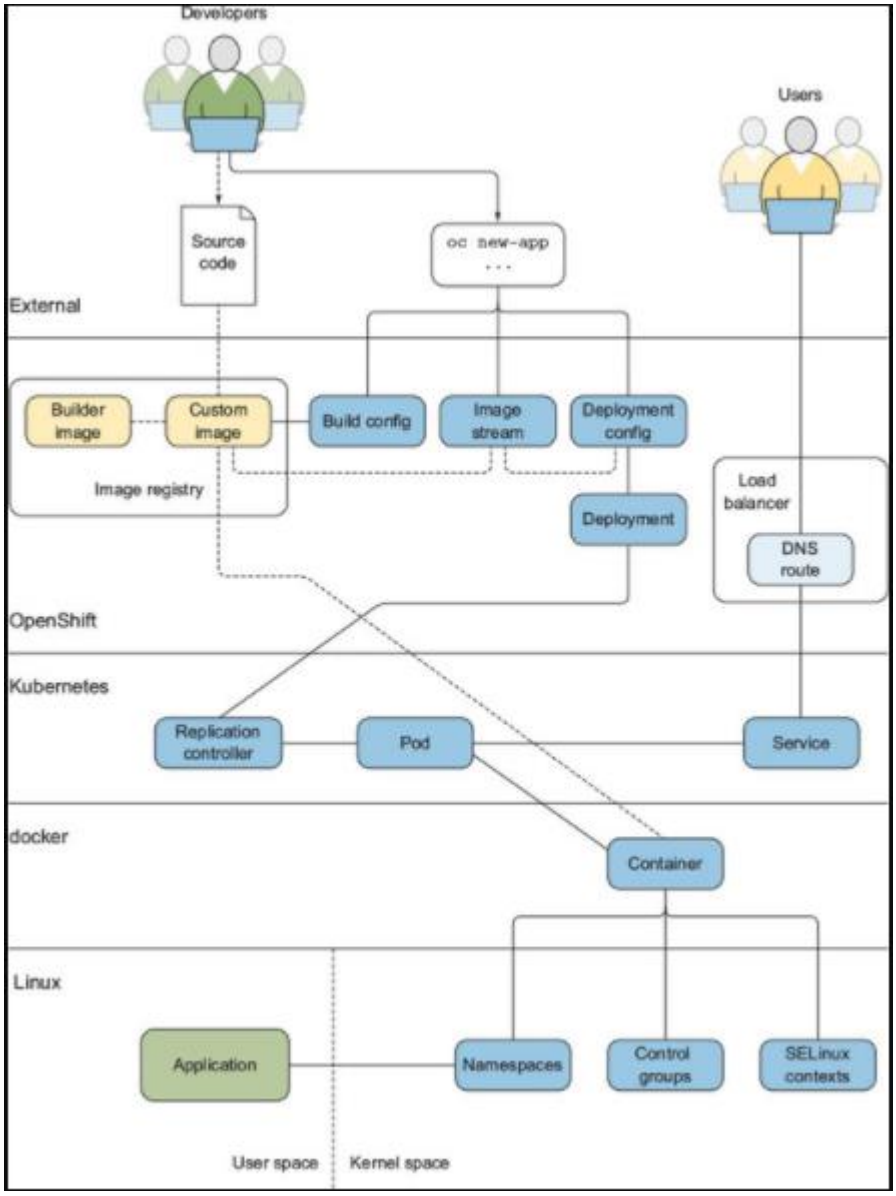

Figure 9 Components of OpenShift [23]

Developers and users connect with OpenShift and its services mainly. OpenShift works with Kubernetes to ensure user requests and continuous delivery of applications according to the developer's designs.OpenShift relies on both the Kubernetes and the dockers to get the program deployed to the customer. Kubernetes is used to schedule applications in various nodes in OpenShift. Kubernetes is the heart of OpenShift's orchestration engine. An OpenShift cluster is in several respects a Kubernetes cluster. Kubernetes will build many application components when first deployed in app-cli [24]:

- **Replication Controller**: Scale the application in Kubernetes as required. This component also ensures the desired number of Pods in the deployment configuration is held at all times. If too many Pods exist, the ReplicationController will stop any number of Pods exceeding the specified amount. When too few are present, the

ReplicationController starts extra Pods until the number is specified. If a Pod is failed, or if Pods are deleted or terminated, ReplicationController is responsible for re-creating the Pods that are failed or deleted to match the requested number Pods.

- **Service**: A service is a Kubernetes object that maps one or more incoming ports to a selected target set of Pods. They serve a microservice or a cluster program. Pods can find the resources on the cluster. Pods usually communicate with other applications or microservices through the Service object in the cluster.
- **Pods**: Represent the smallest scalable component in OpenShift. A Pod consists of one or several containers that share the namespaces and cgroups of Linux containers in a Pod share the same mount and network namespace (the same IP address and ports of TCP/UDP). Within a Pod, each container may have additional sub-isolations (i.e., different UTC namespaces). Using localhost, pods interact with each other.
- **Deployment**: The Deployment object is a declarative configuration specifying the desired state, quantity, and Pods version to deploy.

The replication controller decides how many pods are generated for initial application deployment and is connected to the OpenShift deployment component. The service displays all the pods used by a replication controller. In OpenShift, it offers a single IP address, as it is scaled up and down at various nodes in a cluster. The service is the internal IP address referenced in the route generated in the OpenShift load balancer. The relationship between deploys and replication controllers is to deploy, scale, and update applications. When changes to an implementation configuration are made, a new deployment is created, creating a new replication controller [23]. The replication controller will then create the desired number of pods in the cluster, where an application is deployed.

Kubernetes is used to orchestrate OpenShift containers. However, Kubernetes relies on Docker for each application to build containers. Docker is a container runtime mechanism. The application on a server that builds manages, and removes containers is a container runtime. A container runtime can act as an autonomous tool on a computer or a single server, but it is most effective when a tool like Kubernetes is organized around a cluster [23]. The newer version of OpenShift uses Container Runtime Initiative(CRI) as a container runtime. Kubernetes manages the Docker in order to build containers for the application. These containers use the customized base image as a starting point for the files visible in the container for applications. Finally, the container in the dock is attached to the Kubernetes pod.

The Docker uses Linux kernel components to isolate libraries and applications in the container image and other server resources. These kernel resources are the components that separate the applications in a container from all other applications in the application node. Docker uses

three Linux kernel components to isolate applications that run in containers it creates and to restrict access to host resources [23]:

- Linux namespaces— Provide isolation for the resources running in the container. Although the term is the same, this is a different idea than Kubernetes namespaces, which are loosely equivalent to an OpenShift project.
- Control groups (cgroups)— Have full, assured access limits for Cpu and memory on the application node.
- SELinux contexts— Prevent the container applications from inappropriately accessing resources on the host or in other containers. An SELinux context is a unique label added to a container's resources on the application node. This particular label prevents the container from accessing something that does not have a similar label on the host.

Applications in OpenShift are run and connected with these kernel components. They have the isolation that is seen from inside a container. A Linux server is split between the user area and the kernel area, two key resource classes. Applications run in the userspace. Any process not in the kernel is considered to be part of the Linux server user space. The kernel space itself is the kernel. Without special administrative privileges such as the root user, users cannot modify code in the kernel space [23]. Applications in a container run in the userspace, but the container separated components run in the kernel space. In other words, containers are separated by kernel elements, which cannot be changed from inside the container.

OpenShift Architecture:

Overview:

OpenShift Container Platform has a smaller, isolated units architecture focused on microservices that operate together. OpensShift runs on top of a Kubernetes cluster, a stable clustered key-value store, with data on objects stored in etcd. Those services are split by function [20] :

- REST APIs that expose each of the objects at the core.
- Controllers that read such APIs apply modifications to other objects and report or write the status back to the object.

Users call the REST API to update the system status. Controllers use the REST API to read and synchronize the other device's other parts with the desired user status. For instance, if a user asks for a build, they create a "build" object. The build controller sees the development of a new build and executes a process on the build cluster. When the build is complete, the controller uses the REST API to update the build object, and the user knows that the build is complete [20].

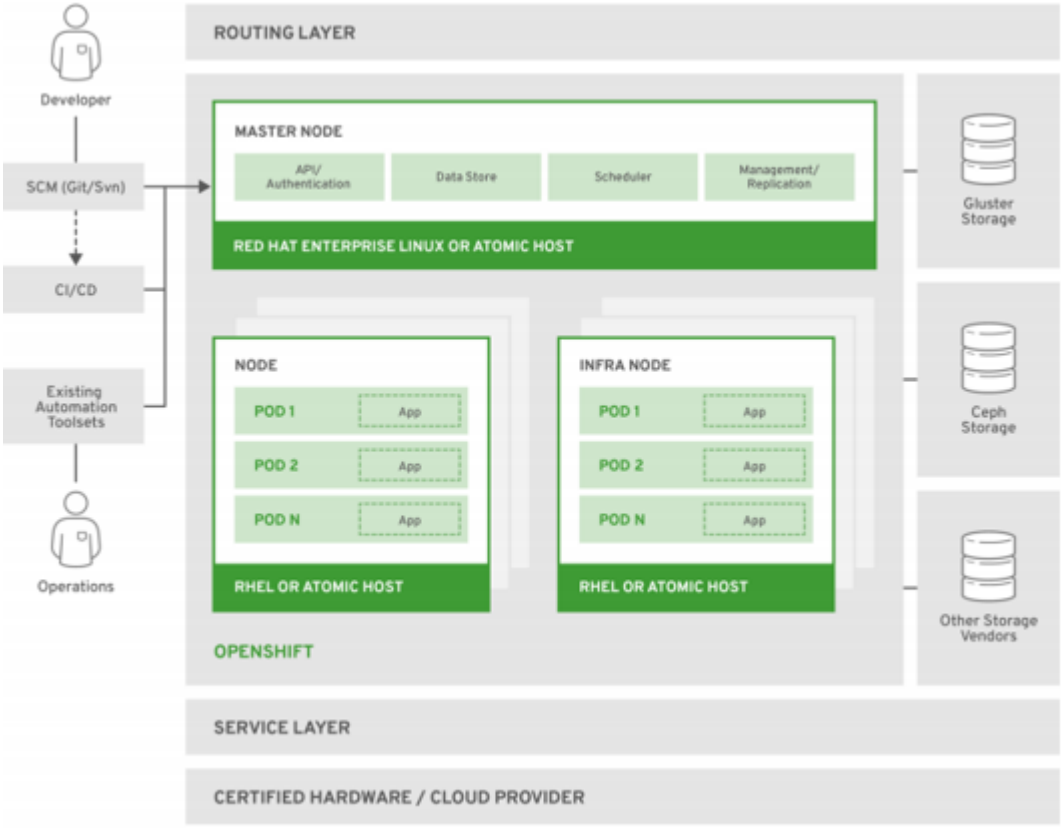

Figure 10 OpenShift Container Platform Architecture [20]

The controller pattern ensures that a significant part of the OpenShift Container Framework functionality is extensible. The way that builds is run and launched can be customized regardless of how images are handled or how deployments occur. The controllers execute the system's "business logic," take user behavior and transform them into reality. Various behaviors can be enforced by customizing these controllers or replacing them with their logic [20]. From the perspective of system administration, this also means that the API can be used to script a repeated schedule for typical administrative actions. These scripts are also controllers that monitor for and take action on changes. OpenShift Container Framework allows the cluster to be customized in a first-class way.

To make this possible, controllers exploit a reliable stream of system modifications to synchronize their system view with what users are doing. As soon as changes occur, this event stream moves changes from etcd to the REST API and then to the controllers so that modifications can ripple out very quickly and efficiently through the system. Because failures can happen at any moment, however, the controllers must also get the latest system status at start-up and check that everything is in the correct state. This resynchronization is essential because it ensures that even if anything goes wrong, the operator may reset the components affected, and before proceeding, the machine double checks everything. As the controllers can

always bring the system into sync, the system should eventually converge to the user's purpose [20].

Kubernetes Architecture:

Kubernetes manages containerized applications through a range of containers or hosts within the OpenShift Container network and offers deployment, maintenance, and application-scaling mechanisms. Runtime packages for containers, instantiate, and run container applications. A cluster of Kubernetes contains one or more masters and a collection of nodes. With Kubernetes as its basis, OpenShift inherits and expands all base constructs (Pods, Replication controllers, Services, and deployments) to the orchestration of containers from Kubernetes. Much of these extensions come from the addition of capabilities or functionalities that are not part of the Kubernetes foundation but needed to run the platform successfully [24]. Other extensions include the enforcement of prescriptive best practices to meet the business environments' stability and regulations.

The architecture of Kubernetes consists of the following main elements [24]:

Master nodes: The nodes are the master nodes, those host core elements like the kube-API-server, the kube-scheduler, and the kube-controller-manager, as well as the etdc database [24]:

- **Kube-api-server:** the part commonly called the Kubernetes API. It is the frontend API for the Kubernetes cluster control plane.
- **Kube-Scheduler:** This part handles the scheduling of Pods into nodes with due regard to resource specifications, policy limits, affinity or anti-affinity rules, and other filters.
- **Kube-controller-manager:** This part runs multiple services for the master controller. Some of the controllers are as follows:
    - **Node Controller:** This controller is responsible for the detection and response of node failures.
    - **Replication Controller:** This controller is responsible for ensuring the correct number of Pods run on the system as requested by an object.
    - **Controller Endpoints:** This handles the Endpoint Objects with the right Service and Pods.
- **etcd:** This is a key-value storage database commonly used by Kubernetes to store cluster configuration data (i.e., nodes, pods) and service discovery.

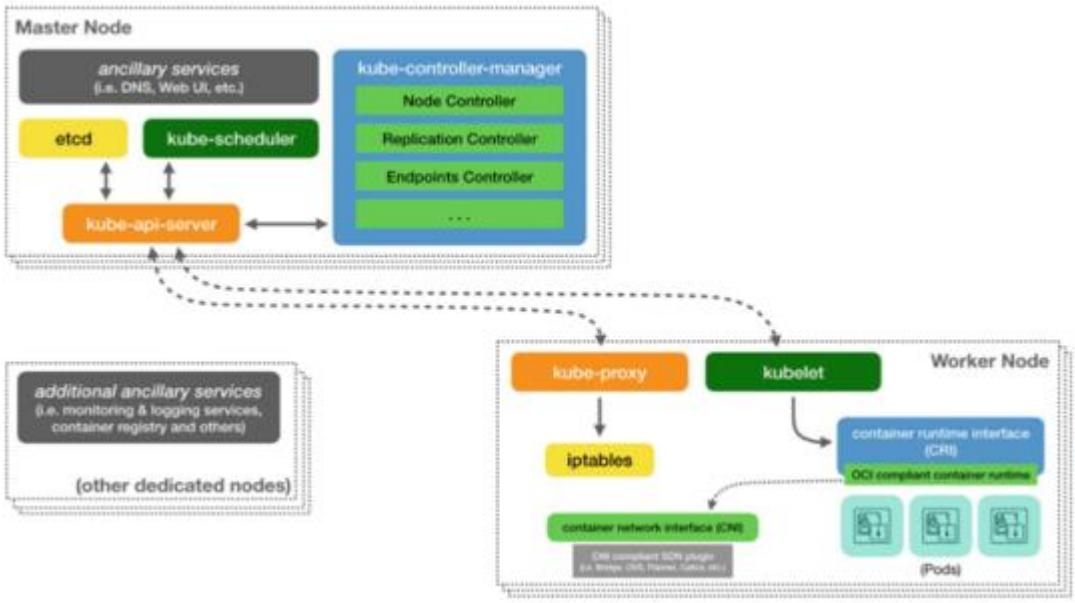

Figure 11 Elements of Kubernetes Architecture [24]

Worker nodes: Worker nodes host elements such as kubelet, kube proxy, and container runtime [24]:

- **kubelet:** The Kubernetes agent is also known as the node agent and runs on each node. The kubelet ensures that containers are started and continued to run as the container manifest specifies and updates the node accordingly.
- **kube-proxy:** Simple network proxy agent for Kubernetes that runs on any node. The kube-proxy resumes the host-defined network services move traffic to a related service, and balances traffic loads. This is achieved by managing the host's iptables rules.
- **A container runtime:** Any CRI-conforming runtime can run OpenShift container Initiative-compliant containers (i.e., Docker Daemon, CRI-O, containers).

Types of nodes in OpenShift architecture:

The architecture of OpenShift builds on top of Kubernetes and consists of three node types [24]:

- **Master Nodes:** Master nodes in OpenShift are Kubernetes Master Nodes that can have additional functionality, such as the self-service portal web console and developers and dashboards based on operations.
- **Infrastructure Nodes:** Kubernetes Worker Nodes are dedicated to hosting features such as OpenShift Routes and the internal registry of OpenShift.

- **App Nodes or Nodes:** Nodes are the Kubernetes worker nodes used to run OpenShift deployed microservices and containerized applications.

As a Kubernetes superset, there can be multiple integrated components from other Open Source projects within these nodes, beyond the Kubernetes components, that work together to increase Kubernetes features and capabilities and form the OpenShift Container Platform [24]. A specific emphasis of this integration is on promoting the use of developers and application owners.

Master Nodes:

The key control elements in the OpenShift control plane are the Master Nodes. This is the Kubernetes Master node and offers any Kubernetes Master's necessary services and offers many features built on top of the Kubernetes that build OpenShift.

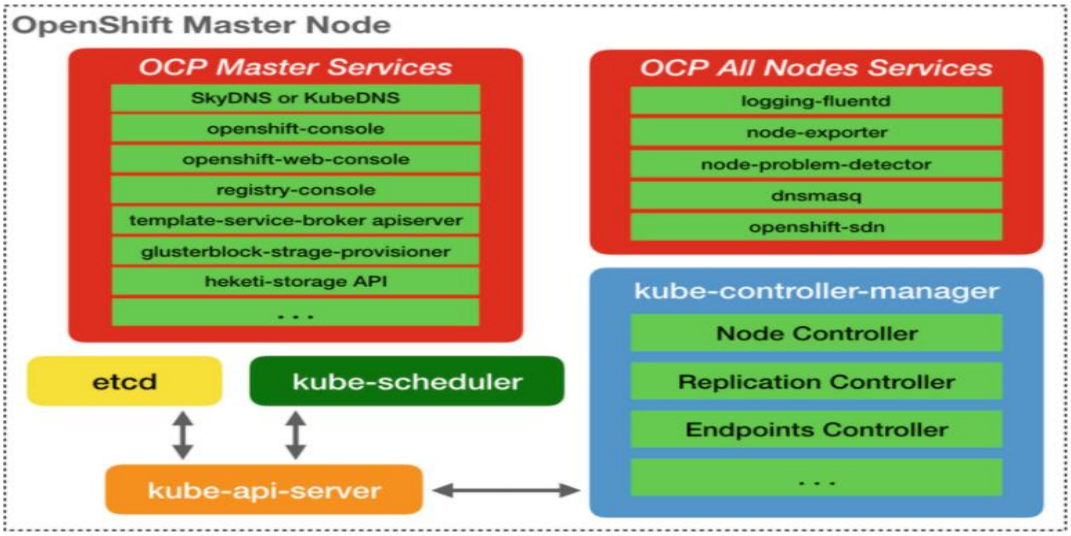

Figure 12 OpenShift Architecture - Master Node [24]

List of elements include in OpenShift master node [24]:

- **Kubernetes DNS:** SkyDNS as part of Kube-DNS was used with the OpenShift 3.x updates. This DNS service listens on port 8053 by default.
- **OpenShift Web Console:** this is the microservice for the development console or portal.
- **OpenShift Console:** This is the working console for the microservice.
- **Registry Console:** The microservice offers a simple web user interface to communicate with the internal container registry.

- **Additional APIs and consoles:** Many cluster services have their Web frontends and APIs. These APIs and frontends are supplied as containers in the master nodes by default. Examples include template service agents and the OpenShift storage container (glusterfs).
- **Node-exporter and kube-state-metrics:** Node-exporter and kube-state-metrics services are part of the Prometheus-based OpenShift cluster surveillance solution. The node-exporter agent collects and renders Prometheus available node hardware and OS metrics. The kube-state-metric agent transforms metrics of objects of Kubernetes (i.e., kubelet) into metrics that Prometheus uses.
- **Node-problem-detector:** This service runs in each node to identify and report multiple node problems to the API server.
- **dnsmasq:** This service is automatically installed on all nodes as part of the Kube-DNS service. Pods use their default DNS with the node. Dnsmasq will submit the query to the DNS at the Master Nodes of Kubernetes when receiving a name resolution request. If not, it will attempt recursive DNS onto the upstream DNS server configured initially on the node.
- **OpenShift-SDN:** Consists of collecting privileged containers offering an OpenShift Cluster Open vSwitch(OVS) Software Defined Network (SDN).

- **Fluentd:** In every node, the Fluentd service runs. Fluentd aggregates logs from the host node, including logs from the Pods, and sends them to the Infrastructure Nodes Elasticsearch (ES) database.

Infrastructure Node:

These are Kubernetes Worker Nodes that host vital elements for the proper operation of the OpenShift Cluster. Included are the Container Registry and the OpenShift Router.

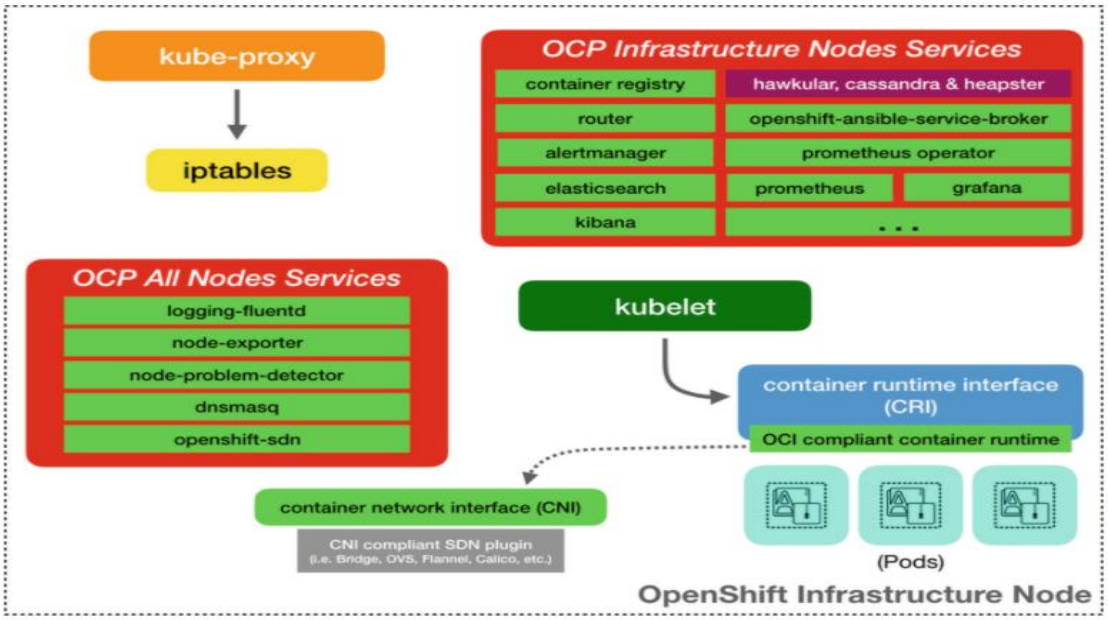

Figure 13 OpenShift Architecture - Infrastructure Node [24]

Like the Master Nodes, the Infrastructure Nodes resources' exact list depends entirely on the cluster's services. Of the services shown in the diagram, a few merit mention here [24]:

OpenShift Registry Container (OCR): The OpenShift registry container is a containerized registry service internally used by the container cluster. OpenShift Container Platform offers the OpenShift Container Registry (OCR), an integrated container image registry that enables the automatic provision of new image repositories on request. This allows users to transfer the resulting images in an integrated location for their application builds. When a new image is pushed into OCR, OpenShift Container Platform notifies the registry that a new image has been generated, transmitting all its information such as namespace, name, and image metadata. Without building and deployment integration, OCR can also be deployed as a stand-alone component that functions solely as a container image registry [20].

OpenShift Router: The OpenShift Router is used to expose an external client to a Kubernetes service via a fully qualified domain name(FQDN).

Elasticsearch (ES) and Kibana: Elasticsearch collects logs sent by each node to the Fluentd service. The Kibana Web UI communicates with data and generates displays and dashboards for aggregated data.

Prometheus, Grafana, and Prometheus Operator: the new OpenShift Monitoring and Metrics solution contains these components. These are used to gather information on the cluster's health and all its services and components. The Grafana web interface is used to create dashboards that represent the status of the managed elements.

Node or App Node:

Kubernetes Worker Nodes, or simply OpenShift Nodes, are used to run the workloads that are built in an OpenShift cluster. These include applications, microservices, or containerized applications. It includes popular OpenShift Services outside the Kubernetes Nodes elements to provide Pods with network access, DNS resolution, node monitoring, and log aggregation [24].

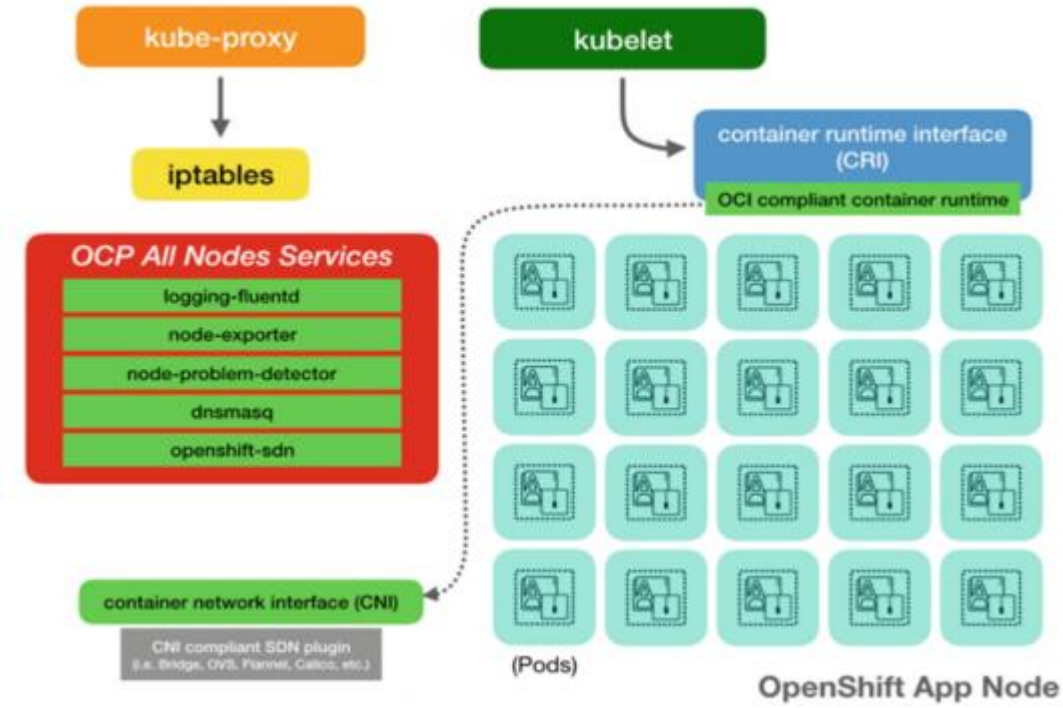

Figure 14 OpenShift Architecture - Node/App Node [24]

Networking:

Kubernetes ensures that pods can network together and assigns an IP address from an internal network for each pod. This makes sure that all containers act as though they were on the same host inside the pod. Giving and pod its IP address ensures that pods can be handled in port allocation, networking, classification, service discovery, load balancing, application configuration, and migration like physical hosts or virtual devices [20]. It is unnecessary to build connections between pods, and it is not recommended that users' pods talk to each other directly using the IP address. Instead, users can build a service and then communicate with the service.

OpenShift Container Platform DNS: For the frontend pods to connect to the backend services, environment variables for usernames, IPs, and more are generated. Suppose users are running multiple services, such as frontend and backend services for multiple pods. The deleted and recreated service will allocate a new IP address to the service and re-create the front pods for modified values of the service variable's IP environment. Also, before any of the front pods,

the backend service must be developed to ensure that the service IP is appropriately produced and that it can be given as an environmental variable to the front pods. For this purpose, the OpenShift Container Platform has a built-in DNS so that the DNS service and the IP/port service can access the services. The OpenShift Container Framework supports Split DNS by running a master SkyDNS that responds to service DNS queries. The master listens by design to port 53 [20].

OpenShift Container Platform SDN: The OpenShift Container Platform uses a software-defined networking (SDN) approach to provide a single cluster network that allows pod-to-pod communication across the OpenShift Container Platform cluster [20]. The pod network is developed and managed by OpenShift SDN, which uses Open vSwitch to configure an overlay network (OVS).

Routers: An Administrator of the OpenShift Container Platform can deploy routers to nodes in the OpenShift Container Platform cluster that allow external clients to use routes generated by developers. The OpenShift Container Platform routing layer is pluggable, with many router plug-ins offered and supported by default [20].

To locate the service and the endpoints enabling the service, a router uses the service selector. The OpenShift Container Platform uses router load balancing as both router and service have load balancing. A router senses and adapts its configuration accordingly to the relevant changes in its services' IP addresses [20]. For custom routers, this helps communicate API object changes to an external routing solution.

The request path begins with a host name's DNS resolution to one or more routers. The method suggested is to describe a cloud domain with a wildcard DNS entry pointing to multiple router instances backed by one or more virtual IP (VIP) addresses. Routes that use names and addresses outside the cloud domain enable individual DNS entries to be configured [20].

Routes between a series of routers can be shared. Administrators may set up cluster sharings, and users may set up sharings for their namespace project. Sharding enables several router groups to be specified by the operator. Only a subset of traffic serves each router in the group [20]. Routers of the OpenShift Container Platform provide external host name mapping and load balancing of service endpoints over protocols that pass directly to the router distinguishing information; the hostname must be present in the protocol so that the router can decide where it is to be sent. Router plug-ins presume that, by design, they will connect to host ports 80 (HTTP) and 443 (HTTPS). This implies that routers must be mounted on nodes that are not otherwise used by such ports.

Persistent Storage:

The OpenShift Container Platform uses the Kubernetes persistent volume (PV) framework to allow cluster administrators to provide a cluster with persistent storage. Developers may use persistent volume claims (PVCs) without detailed knowledge of the underlying storage infrastructure for requesting PV services. PVCs are project-specific and are built and used by developers to use a PV. A single project does not cover PV resources alone; it can be shared and demanded from any OpenShift Container Platform cluster project. However, after a PV is bound to a PVC, PV cannot be bound to additional PVCs. This results in a linked PV scope to a single namespace (that of the urgent project) [20].

PVs are specified by the PersistentVolume API object, which represents a part of the cluster's existing networked storage that the cluster administrator provided. It is a cluster resource, just like a cluster resource is a node [20]. PVs are volume plug-ins, such as volumes, but have a life cycle independent of any PV-using individual pod. Whether NFS, iSCSI or a cloud-provider-specific storage device, PV objects capture the storage implementation information.

PVCs are specified by a PersistentVolumeClaim API object, representing a developer's storage request. It is analogous to a pod in that node resources are consumed by pods, and PVCs consume PV resources. For instance, pods can request specific resource levels (e.g., CPU and memory), while PVCs can request specific storage capacities and access modes (e.g., they can be mounted read/write once or read-only several times) [20].

Ephemeral Storage:

The OpenShift Ephemeral System is a Technology Preview (TechPreview) capability that enables administrators to restrict and control the ephemeral local storage consumed by pods and containers running in a specific node. Without the Ephemeral system, Pods do not know how much local storage the Writable Layer container can consume, and Pods cannot request guaranteed local storage [24]. As a consequence, if the Node fails to store the Pods locally, all data stored in the ephemeral volume can be deleted.

CHAPTER 7: COMPARISON AND CONTRAST OF OPENSTACK AND OPENSHIFT

Similarities between OpenStack and OpenShift:

OpenStack and OpenShift are both open-source Linux-based cloud platforms and maintained by Red Hat Inc. Red Hat Inc. They will often release a different version of each platform, and recent releases are Red Hat OpenStack Platform 16.1 and OpenShift Container Platform 4.7.

Both the platforms provide deployment of their architecture in public, private or hybrid clouds. OpenStack utilizes REST APIs to create virtual machines, launch instances, add metadata to virtual machines and images, and create storage containers and objects. Whereas in, OpenShift all end-user applications and the cluster and users of the cluster is managed through REST APIs. Both of them use web-console/ web-interface for deployment and management of their technologies.OpenStack uses persistent storage type to store Virtual Machines (VM) images, and OpenShift uses the Kubernetes persistent volume (PV) to provide persistent storage.

Comparison of OpenStack and OpenShift:

OpenStack	OpenShift
OpenStack platform is built in Python language.	OpenShift platform is built in Go and Angular. Js languages.
It is an open-source platform maintained by Red Hat Inc. and others.	OpenStack is also an open-source platform maintained by Red Hat Inc.
OpenStack can be built on any underlying operating system avoiding vendor lock-in.	OpenShift platform is built on Red Hat Enterprise Linux or CoreOS.
OpenStack is used to deploy Infrastructure as a service(Iaas).	OpenShift platform is deployed for Platform as a service(Paas).
OpenStack provides limited support for containers and focuses instead on VMs focused on hypervisor KVM, Xen, and VMWare.	OpenShift, by comparison, is focused primarily on container operations and containers using Docker and Kubernetes.
OpenStack consists of a single-point platform controlled by distributed data centers. It does not rely on global resource clusters. The platform supports the distributed storage via third-party storage components Ceph and Glusterfs but does not rely on those components.	OpenShift relies heavily on distributed and globally controlled clusters via Kubernetes. It uses master nodes to manage distributed worker nodes that allow communication between resources. It uses networks.
OpenStack is compatible with Azure, AWS, and Google Cloud providers. Users can use OpenStack to create a single cloud on top of existing resources, whether on-site or in the cloud.	In comparison, OpenShift is a service platform (PaaS) that works with containerization independently of the cloud resources. This enables users to build cloud-based services and applications to host for distributed availability in any cloud resource.

Advantages of Openstack:

- Red Hat OpenStack Platform can accommodate today's virtual machines and bare-metal systems on a single scale.
- OpenStack increases business mobility, availability, and reliability by providing on-demand networks, resource pooling, self-support, and highly elastic and calculated services.
- Since OpenStack enables private, on-site cloud construction, it can help regulate compliance efforts. If a cloud is in a data center owned by organization, privileges, security measures, and security policies will be monitored. Personal data, financial information, and other confidential and regulated information policies are implemented.
- One big issue is the vendor's lock-in with a proprietary solution. OpenStack supports many proprietary technologies that can be used in the hypervisor and bare-metal environments in the smorgasbord. Its ability to work with commodity hardware allows users to select solutions based on a wider variety of costs and competencies.
- OpenStack improves time-to-market by providing business units with a self-service portal for accessing on-demand services and an API-driven framework for creating cloud-aware applications.
- The agnostic HDD/node failure feature provides reliable self-healing data redundancy, which protects against failures.
- OpenStack has scalable storage volume and performance. For example, when a website comes under increased pressure at particular times of the day, users only have to pay for additional resources when they are used.
- OpenStack is particularly useful because it is easy to automate tasks, and its automation is a key selling point compared to other alternatives. The platform provides built-in tools that significantly promote cloud management.
- OpenStack has an application programming interface (API) that enables other programs to tap into it. People may create apps that interface with OpenStack to perform tasks such as firing virtual machines. Anyone can build and upload an OpenStack solution, and anyone from around the world can use them.
- OpenStack is similar to Linux in that there are several distributions with different features but still the same core components. This promotes innovation because no one organization has universal software rights. The basic code is free, but vendors and resellers must all get started from the same location and show creativity to create a spin-off product that will bring something different to the market.

Advantages of OpenShift:

- A container framework such as Red Hat OpenShift allows the containerized applications to work on the infrastructure they are ideally suited to while remaining portable.
- Pods are one or more containers on one host. The CPU, memory, disk, and network bandwidth are allocated to each pod. Use a single pod to deploy a complete web application, complete with a private database case. To build different applications, use multiple pods.
- OpenShift allows the elasticity of the cloud by automatically scaling the horizontal pod as the application load increases. This removes the need for operations to increase the number of application instances manually.
- OpenShift is consists of a control plane (Kubernetes masters), persistent REST API object storage (etcd), and an infrastructure hosting framework (Kubernetes nodes). For fail-over and load-balancing scenarios, each piece of the network can be designed with several redundancies to remove the impact of hardware or infrastructure failure.
- OpenShift offers customers the ability to access the physical or virtual, public or private cloud, and hybrid cloud infrastructure. This gives IT the flexibility to deploy OpenShift to suit the current infrastructure best.
- OpenShift has a rich developer interface for web consoles with a responsive UI design so users can conveniently view it in a browser. Inside the web console, developers can build, modify and manage their applications and related resources.
- OpenShift provides several command-line tools that provide full access to the developer interface for developers that prefer operating from a command line. These tools are simple to use, and automatic interactions can be written.
- OpenShift's specific SELinux-based architecture enables users (developers or operations) to remotely execute commands or log into individual containers for platform applications via SSH. The logged-in user accesses only user processes, file systems, and log files. This helps users to access the applications they need to architect and handle.

Use Cases of OpenStack and OpenShift:

- OpenStack provides Infrastructure as a Service(IaaS); larger organizations can deploy cloud on-premises or data center utilizing their own infrastructure using OpenStack. Organizations can deploy public, private, or hybrid cloud according to the needs of the organization.

- OpenStack is being used for network functions virtualization (NFV), which includes separating a network's main functions such that they can be spread across several environments.
- OpenStack helps to run containers on a virtual machine or bare metal for an organization.
- Big data can also be deployed using Openstack, which provides scalable and elastic infrastructure.
- High-performance computing clouds can be built by taking advantage of different components in OpenStack.
- OpenShift helps to deploy and develop web applications, backend applications, microservices, and databases. Applications can be built in any language of developers' choice.
- OpenShift Container Platform can be used to deploy on-premises platform as a service(PaaS), applications can be built using containers in Dockers and maintained by using Kubernetes.
- OpenShift Online helps developers build a public cloud application development and hosting platform, allowing developers to concentrate on writing application logic by automating application provisioning, management, and scaling.
- Red Hat OpenShift Platform provides container-based systems management built on Red Hat OpenStack, VMware vSphere, or customers cloud services provider's infrastructure. This is the hybrid model for moving from VMs to containers, which is gradual as more applications come into being containers. Adding more abstraction layers to provide further flexibility and stability, underlying technology's value continues to decrease, enabling IT to concentrate more on applications, updates, and driving – not maintenance and management.

References:

[1] Srinivasan, S., 2014. Cloud computing basics. Springer.

[2] Mell, P. and Grance, T., 2011. The NIST definition of cloud computing.

[3] Rountree, D. and Castrillo, I., 2013. The basics of cloud computing: Understanding the fundamentals of cloud computing in theory and practice. Newnes.

[4] Bhattacharjee, R. (2009). An analysis of the cloud computing platforms. MIT MS Thesis, Cambridge

[5] HyperTable. (2013). http://www.hypertable.org. Accessed 11 Dec 2013.-hybrid cloud

[6] What is Baas? | Backend-as-a-service vas Serverless

What is BaaS? | Backend-as-a-Service vs. serverless - Bing

[7] Carter, Brian. (2016). Grow your own Backend-as-a-Service (BaaS) platform.

[8] IBM Cloud Learn Hub / Virtual Machines

https://www.ibm.com/cloud/learn/virtual-machines#

[9] Virtualization / Virtual Machines

https://www.redhat.com/en/topics/virtualization/what-is-a-virtual-machine

[10] Kocher, P.S., 2018. Microservices and containers. Addison-Wesley Professional.

[11] Garbarino E. (2019) Pods. In: Beginning Kubernetes on the Google Cloud Platform. Apress, Berkeley, CA.

[12] Vemula, R., 2019. Integrating Serverless Architecture: Using Azure Functions, Cosmos DB, and SignalR Service. Apress.

[13] Jangda, A., Pinckney, D., Brun, Y. and Guha, A., 2019. Formal foundations of serverless computing. Proceedings of the ACM on Programming Languages, *3*(OOPSLA), pp.1-26.

[14] Solberg, M., & Silverman, B. (2017). OpenStack for Architects. Packt Publishing.

[15] OpenStack Documentation

http://docs.openstack.org.OpenStack Docs: Overview

[16] Bumgardner, V.C., 2016. OpenStack in action (Vol. 5). Manning Publications Company.

[17] Radez, D., 2016. OpenStack Essentials. Packt Publishing Ltd.

[18] Rosado, T. and Bernardino, J., 2014, July. An overview of OpenStack architecture. In Proceedings of the 18th International Database Engineering & Applications Symposium (pp. 366-367).

[19] RedHat OpenStack Documentation Guide

https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/9/html-single/architecture_guide/index#Examples

[20] OpenShift Documentation OpenShift Container Platform

Overview | Architecture | OpenShift Container Platform 3.11

[21] Gulati, S., 2014. OpenShift Cookbook. Packt Publishing Ltd.

[22] Lossent, A., Peon, A.R. and Wagner, A., 2017, October. PaaS for web applications with OpenShift Origin. In Journal of Physics: Conference Series (Vol. 898, No. 8, p. 082037). IOP Publishing.

[23] Duncan, J. and Osborne, J., 2018. OpenShift in Action.

[24] Caban, W., Architecting and Operating OpenShift Clusters.