Lecture 12: March 22

Lecturer: Prashant Shenoy *Scribe:* ***Nikhil Agarwal Jasmine David***

12.1 Kubernetes (k8s)

Kubernetes is a cluster management system using containers as its fundamental abstraction. It is based on Google's Borg and Omega cluster schedulers.

Kubernetes uses another abstraction called 'pod.' A pod is an abstraction which contains on or more containers. Kubernetes deals with resources and allocate resources to pods. Kubernetes takes pods, put them on machines, and then allocate the resources of that machine to the pods. This means that Kubernetes doesn't deal with containers directly but instead deals with pods. However, we can put just one container inside a pod and then a pod is essentially very similar to container. How different components are put inside pods is a design choice for the application. A simple approach could be to put every component (like a database or a web server) and put it inside its own pod. We can replicate pods as well. For example, to run three instances of a web server, we can put it inside a pod and then replicate this pod a total of 3 times.

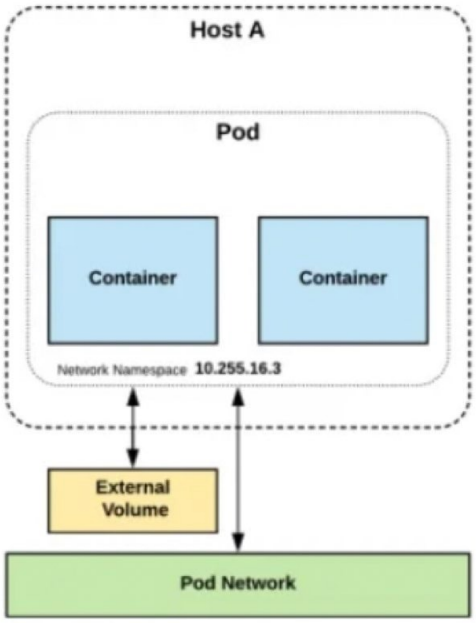


Figure 12.1: Kubernetes pod.

Question: Can pods span across machines

Answer: Yes pods can absolutely span across machines.

12.1.1 Kubernetes services

A Kubernetes service is how Kubernetes exposes pods to the outside world by giving them a public IP addresses through which clients like web browsers can interact with them

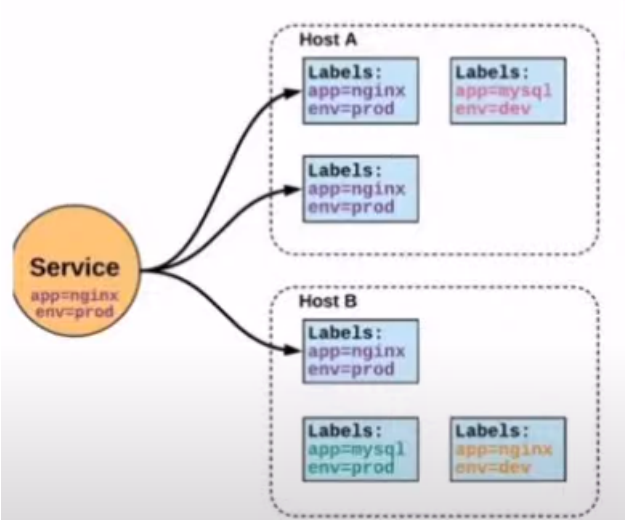


Figure 12.2: A Kubernetes service exposing a web server.

In Figure 12.2, the blue boxes are the pods (the containers are not shown in this figure) which is a replicated server exposed to the outside world through this service using a single IP address. In addition to exposing the web server, the service also performs functionalities like load balancing. Note that a service is a durable entity, i.e., it is kept around by Kubernetes until we decide we no longer need the service. A pod is a much more temporary abstraction, an ephemeral entity. This means that Kubernetes can terminate or migrate pods if needed. Also, the mapping of pods to machines can vary over time and the number of pods k8s creates can vary over time. A service however remains durable even if the pods it exposes are changed over time.

Question: When you talk about replication, are you talking about replicating containers or replicating pods?

Answer: In the Kubernetes world, you are dealing with [pods]

Question: Is it possible to run a database inside a pod?

Answer: A pod will allow you to run a container. You can run whatever code you want inside a container including a database.

Question: [Unclear] **Answer:** It says that each cluster has a static IP address. The pods network is private. Pods can crash or be moved but that is not visible to the user, who are dealing with the service. You want the service to have a stable and often public IP address that is not going to change. Thus the service IP address is static while pods are more transient.

Question: Do we call Kubernetes a platform? **Answer:** Kubernetes is an orchestration platform but not a platform as a service per se. It is essentially allocating containers but not determining what goes into the containers as a PaaS does. Closer to an IaaS than a PaaS.

Does Kubernetes decide the replication strategy or does the developer decide?

Answer: When a developer deploys an application to Kubernetes, they provide a config file. In the config file, they specify which ports should be replicated, which shouldn't, etc. and Kubernetes will act accordingly.

Question: Don't Kubernetes and Docker do similar things? If so, which do you recommend?

Answer: The actually are not similar. Docker is a set a packaging tools, allow developers to make containerized, portable code. If you wanted to deploy that code to a cluster, that Docker code could be put in communication with Kubernetes.

12.1.2 Kubernetes Control Plane

Control plane is the controller of the Kubernetes which manages the machines in the cluster. Here is a picture of different components of the control plane.

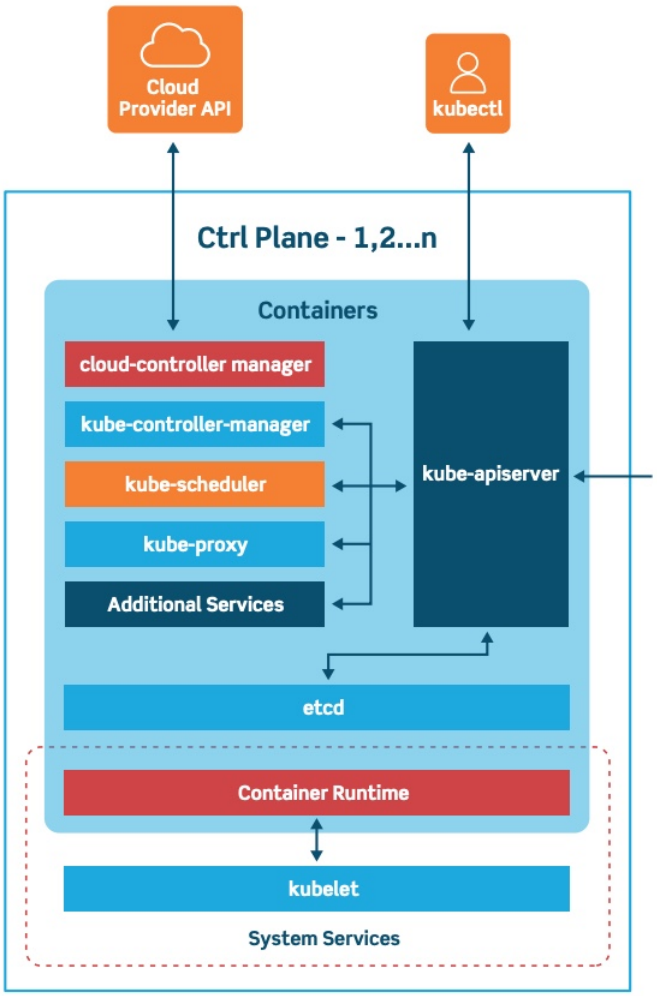


Figure 12.3: A Kubernetes control plane

The components of the control plane are as follows:

- **apiserver**: Exposes REST interfaces to the users using which they can pragmatically interact with the k8s controller for management, like starting a new application or requesting more resources. Please note that this is the controller API and not an application API.
- **etcd**: A key-value datastore which is kept on the disc because it stores the state of the k8s which needs to be persistent. It stores information like which pods are mapped to which machines, running status of the applications etc. It provides properties like high durability (RAFT consensus) and strong consistency. Replication of datastore can be done onto multiple nodes for fault tolerance consistently using something known as RAFT consensus.
- **controller-manager**: Used to create and replicate pods, monitor health of the pods and the nodes on which they are running, and to restart pods. For example, we can specify a replication degree of four and the controller-manager will replicate the pod on four machines and restart it on another machine in case of failure to maintain the replication degree. Note that this is done without developer intervention. Controller-manager ensure that failure recovery is built-in the system. This is not common in other cluster schedulers.
- **scheduler**: Scheduler is the entity that assigns pods to servers based on resource constraints and policies like choosing least loaded machine.
- **cloud-controller-manager**: This entity only comes into play if the Kubernetes is running on cloud machines. The cloud-controller-manager interacts with the interface of the cloud platform to manage the Kubernetes. s

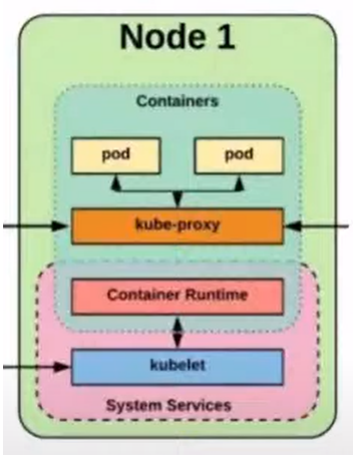


Figure 12.4: A Kubernetes node

Another important entity in the Kubernetes is the **k8s node** which is the actual machine on which the pods are running. There are several components within these nodes which are explained below:

- **kubelet**: On each of these nodes, a kubelet is running which monitors the health of the containers which are running on this machine and esures they are healthy. In case it detects a problem, it communicates with the control plane to take appropriate action about this incident.
- **kubelet proxy**: Kubelet proxy is the network manager which directs the request to the pods referenced over the network. This component also provides services like load balancing for incoming requests to cluster services.

- **container runtime**: The container runtime is the OS virtualization that is used by Kubernetes to run the containers. Kubernetes can use different kind of OS level virtualization as runtime like docker, cri-o etc.

Question: A node is a physical machine. If you have a physical machine, why have pods? Why not deploy containers directly on the physical machine? **Answer:** The reason we need a pod is because they provide isolation between different applications on shared nodes. Since multiple customers/users might share rental resources on the node, we need to isolate them with a logical machine abstraction–a pod. It is essentially a sandbox.

Question: How does Kubernetes deal with higher level traffic management in a dataset? **Answer:** Think of these as layers of abstraction. There is a network infrastructure for a data center with a set of rules. Whatever application you have is subject to those rules. You can connect the Kubernetes rules to data center rules.

Question: If pod is an abstraction which consists of containers, what do we mean when we say a pod has failed?

Answer: A pod fails when a container within it fails, which in turn fail if a process that runs within the container fails. Remember, pods and container are both abstractions and process is the entity which can fail. We can however, just take action on failed containers inside a pod and don't have to restart the whole pod. In the default case when a pod only has one container, if the container fails then the entire pod fails.

Question Why are we using a pod instead of a VM? **Answer** A pod is not a real thing – just a set of rules about a set of containers. It is lighter weight than a VM which requires an OS.

Question: Do the pods have their own IP address or the containers?

Answer: A pod has one IP address which is also the IP address of all the containers inside it. However, the containers host services at different ports at this IP address.

Question: Do the containers within a pod share the same namespace.

Answer: Yes, the namespaces are shared even though the resources are isolated within a container.

Question: How can the resources be isolated if the namespace is shared?

Answer: Namespace is what you can see and resources are what you can use. CPU, memory and other resources can be allocated at the container level.

12.1.3 containerd

containerd is what Kubernetes uses to start and stop containers (i.e., lifecyle management). It is a low-level container orchestation framework. It is it's own open-source project, separate from Kubernetes, that even other cloud providers use to offer Kubernetes-like services.

12.2 Data Centers

A data center is a large collection of servers and storage that provides these computing and storage resources to applications. They are also referred to as servers and storage farms. They can be large or small depending on the number of servers (or racks of servers) or storage units that they have. All companies like Google, Meta, Amazon etc. are running their applications on servers which are sitting in a data center somewhere.

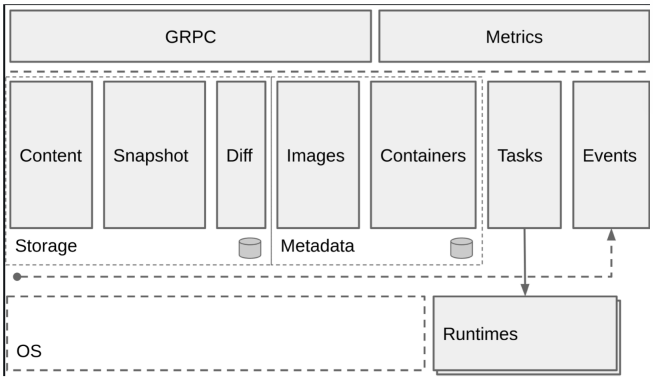


Fig courtesy https://github.com/containerd/containerd

Figure 12.5: containerd.

Figure 12.6: An aisle of a data center

These servers can be owned by these companies or by a different cloud provider. A company can create its own data center and use them or use the servers provided by cloud provider. Regardless of that, a data center can be used for variety of purposes like data processing, hosting web sites, hosting business apps, among many other functionalities.

12.2.0.1 Inside data center

Professor Shenoy notes that the above figure is misleading as it shows that containers are encapsulating pods, which is inaccurate. Ignore the green thing labeled containers and then the figure is accurate.

What we see in Figure 12.6 is an aisle of a data center. We can see racks of data servers which collectively form this aisle. One data center has many such aisles of racks of servers. Each of these servers provide compute and/or storage resources. To oversimplify, we can say it is like a supermarket where we have servers on the shelves instead of products. Because these servers generate a lot of heat, a lot of cooling infrastructure is also needed. In addition to these, power generators/converters/backups are also needed for

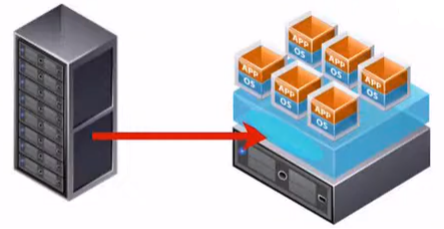


Figure 12.7: A virtual server

the servers and the coolers.

12.2.1 Data center architectures

Data-center architectures have seen a shift from traditional architecture to a modern day architecture. These are discussed below:

- **Traditional architecture**: A traditional architecture has applications running on physical servers. System administrators manage and monitor these servers manually. This architecture Storage Arrays Networks (SAN) or Network Attached Storage (NAS) to hold data. As a company or an individual in need of servers, you go and buy these servers from vendors, put them in a physical warehouse, and then run your application on these servers using a lot of manual configuration and decision making like which IP addresses are assigned to which servers and which applications run on which of these IP addresses.

- **Modern architectures**: Modern architectures automate a lot of the things in the traditional architecture. Instead of running applications on physical servers, they are ran on containers which are present on virtual machines mapped to physical systems. All the mapping from containers to virtual machines, and virtual machines to physical servers are done by an automatic controller instead of system administrator. For example, cluster managers like Kubernetes can be used for this purpose. Overloading can be managed using techniques like live migration. All these properties makes a modern architecture much more flexible and scalable.

12.2.2 Virtualization in Data Centers

In a **virtual server**, a hypervisor is layered over a server which allows different virtual machines or containers to be laid on top of it. These virtual machines can run different operating systems in them and these OS can run different applications inside them. An array of such servers is put on a rack in the data center.

The concept of **virtual desktops** is also becoming common these days where the actual desktops are placed on the virtualized data centers as explained above and the thin remote clients (with less resources) can connect to these desktops over a graphical interface and access them from anywhere. This architecture makes the actual desktop physically separated from the end users. These virtual desktops are generally used by workplaces for their employees.

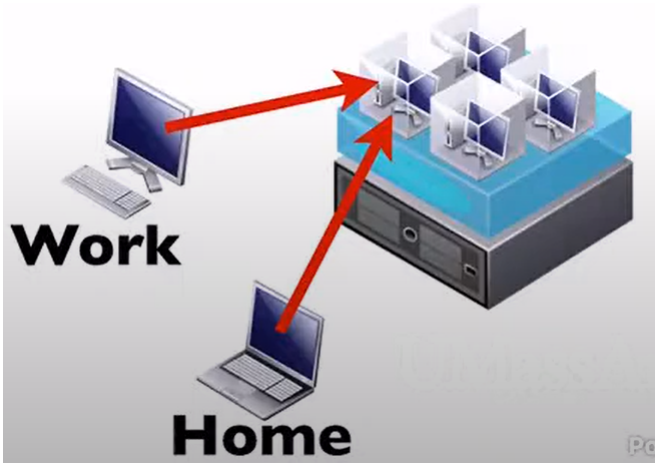


Figure 12.8: A virtual desktop

12.2.3 Server virtualization

Server virtualization is very similar to virtualization of any other machine. A virtualization layer like hypervisor 1 or hypervisor 2 can be used. This allows for a server to be sliced into different virtual machine. This is done primarily because applications require only a subset of resources available. Virtual machines can be created based on different resource needs and the application can then run on these VMs. All benefits discussed like rapid resource allocation, fast live migration are available here as well. For example if there are two VMs on server VM1 and VM2, VM2 can be live migrated to another machine if VM1 needs more resources. These functionalities are not available on a physical server. This is the main reason most data centers and cloud providers use a virtualization architecture.

Question: Does the term virtual private server refer to virtualization?

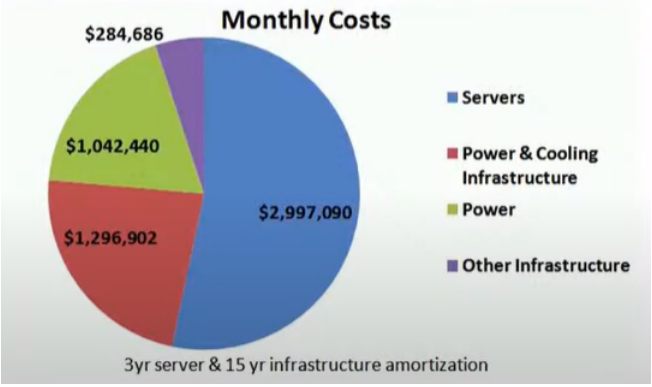
Answer: It is indeed a virtual server which allows private access only to the person/entity it is assigned to.

12.2.4 Data center costs

Running a data center is expensive. In addition to powering up the actual servers present, additional costs also need to be paid like cooling infrastructure. There is a lot of power consumption in the form of electricity in the data center. A lot of work has gone into reducing the power foot print of the data center which could be mesaure by PUE (Power Usage Effectiveness). This is calculated as Total power/IT power as we want IT power to be the major chunk of the total power. IT power is the power used for servers. Ideally this should be 1 as we would want no overhead. This used to be around 2, which means there was a 100% overhead on every unit of power consumed by the server. Nowadays, typically this metric is around 1.7 but google has managed to take it down to 1.1 in its own data centers.

12.3 Cloud computing

Cloud computing is the use of remote servers to run distributed applications. Mostly the servers are running remotely on the cloud and clients are the users of the distributed application. The actual use-case of cloud computing can be much more detailed though. The biggest revolution in the field has been cloud computing



platforms. The owners of the applications have been separated out from the owners of the cloud servers. The resources available on the cloud servers is provided to others (individuals or companies) as a service. The users can lease the remote resources from the data center as a service and pay the provider according to a planned policy. This decouples the applications development from the application deployment and IT infrastructure. There are many benefits of such a service like remote availability on the internet, high scalability (procure more resources on demand), high flexibility (pay only for shorter duration of usage times). In addition to these individual benefits, this service also creates a shared infrastructure which leads to a more economic use of total available resources. The focus has shifted from buying resources to leasing resources which has made all these benefits available to almost everyone.

12.3.1 The cloud stack

The services being operated on the cloud can be made available to the users in varying degrees of freedom and specificity. The lower stack of the cloud services we get, the more barebone hardware resources we get and the more we are in control of what and how to do things like deployment. The more up we get into the stack, the more things are already done and the lesser is our control on the allocation and use of the resources.

- **Infrastructure as a service (IaaS)**: In this the users get machines from the service providers. These can be the servers and/or storage units. What and how to do with these machines is decided by the user of the service. The metric for billing here is the scale and duration for which the resources are procured.

- **Platform as a service (PaaS)**: Not everyone will know how many resources/machines they need for their use-case. Platforms are better for such use-cases because the user just provide their app as a container and how to deploy it including decisions like scale of resources to use is decided by the platform provider. The metric for billing here is usually the number of requests served by the application.

- **Software as a service (SaaS)**: Neither the development, nor the deployment of the application service is done by the user. The user just has user level access to a software/application that is provided by the SaaS provider. Examples: GMail, google drive etc.

Figure 12.9: Cloud stack

12.3.2 IaaS: Amazon EC2

We are explaining IaaS by using the example of Amazon EC2 here. Remote servers and storage units are provided to the users as bare-bones. Billing is done for both servers and storage based on the configuration selected by the user. These range from smallest, medium, largest to other specific categories for both the servers and the storage units. The metrics for billing is generally CPU cores, memory (RAM), storage being provided. A sample billing policy has been mentioned below which has prices that can give a good idea about the actual billing. Note that all the resources are virtualized, and physical servers and disc are not allocated to the users. This allows for single servers and discs to be shared among multiple users.

12.3.3 Types of IaaS instances

In addition to different scale of the resources, the billing policy is also categorized on the basis of what is given as an abstraction in the service. These are described below:

- **On demand instances**: This is the most exclusive and expensive service provided. Here resources are requested and provided on the fly on demand. They are to kept with the user exclusively till the user decided to terminate this procurement. The billing is done by the minute and is generally the highest because of the services are provided exclusively and on a short notice. This model is generally

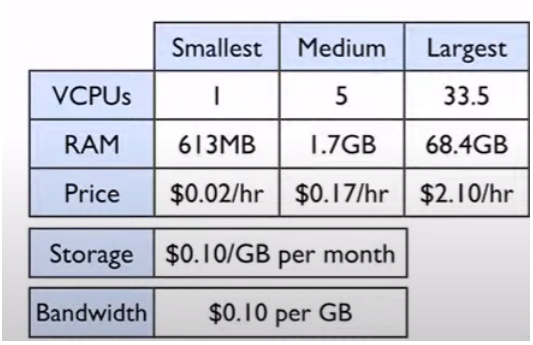


Figure 12.10: Billing policy

used by larger companies/firms for unforeseen required resources.

- **Reserved instances**: Reserved instances are procured for longer duration of time (like 1 year to 3 years) and for cheaper prices compared to the on demand model. This is so because the provider can bills user for all the time and has his resource busy during this long duration. This model is preferred for most of the work done by larger firms/companies.

- **Spot instances**: These are instances of the resources that are provided at large discounts because these contain resources which have not been procured as part of the first two models and are currently free. However, these resources can be revoked after a warning within a short period of time if needed by some user in the on demand model. This model can be used during times where low load of processing is required. (off-peak periods). Because of the sporadic availability in this billing model, it is generally used for lower priority tasks and background tasks like batch jobs, large data processing, mapreduce and spark jobs. This model is not used for hosting real time applications which impact end users like websites. To use spot instances, jobs generally have systems like checkpoints built in them so that progress can be saved before a restart happens.

Question: Do cloud providers switch off the servers during off peak periods?

Answer: Yes, this needs to be done to save electricity which the provider itself has to pay for. They don't necessarily need to be shut down and can be put in a low power mode where they can be started quickly.

12.3.4 PaaS Cloud

In platform as a service cloud, instead of providing bare-bones server and storage units, the provider provides a platform on which applications are deployed. The biggest difference with IaaS is that the development of the application is separated out from the deployment of the application. The application developers just build their applications in frameworks like .NET, Java, Python, Node etc. and it is the responsibility of the platform provider to take care of deployment and scalability of the application. Since hardware is not provided as a granularity here, the billing is done on the basis of the usage of bandwidth of the provider which is directly related to the rate of requests served by the application. This model is also getting popular in cloud providers (serverless computing) where servers are not directly provided to the users. User provide their application container to the cloud provider which is hosted by them just as in PaaS.

12.3.5 Serverless Computing

Serverless computing is a flavor of PaaS. While, there is of course a server behind the service, it is called serverless because the application developer does not deal with the servers because of the PaaS. Again, the platform can scale up or down automatically (elastic scaling). The platform can actually scale the application's resources down to zero when idle. Moreover, the application development process for Serverless Computing is unique in that it leverages Function-as-a-Service (FaaS).

- **FaaS :** Write code as a set of functions and deploy each function. Similarly, these functions can be chained together and the functions are often stateless. This structure of an application as a chain of functions is a more fine-grained structure than micro-services. An example of FaaS in the cloud is AWS Lambda.

Question: What does scale down to zero mean? Does that mean it can't receive any requests?

Answer: It means that there is no container or VM that has your code running in it–maybe they are paused. However, if a request comes, the resources will be scaled up.

12.3.6 Public, private, hybrid cloud

The cloud service which we have been discussing so far is known as public cloud which is available to the general public if they create an account with the provider and pay for it. This is called as the public cloud. However, many a times, companies are not comfortable with this kind of service as they don't want to share their disc time and servers with others. In this case, dedicated data centers or smaller parts of existing centers can be allocated for the use of just one company. All the benefits and services of virtualization are still available but the use is limited to a single entity to which the resources are dedicated. However, this might come at a higher cost because of the promised exclusivity.
A hybrid of these models is called the hybrid cloud in which the provider moves resources between the public pool and the private pool of servers. From the users perspective, this is done because their usage may have changed over time and they might want to reduce or increase the resources allotted to them. Accordingly, resources are moved to or from the public cloud. From the providers perspective, this is done for the more efficient use of resources so that they are kept busy as much as possible. There is also a financial incentive to move resources from public pool to hybrid pool. This usually happens when the clouds bursts, which means the user needs and starts using cloud servers and storage from what they had been allotted earlier.

Question: Why even a provision for something like a hybrid cloud?

Answer: Hybrid cloud is needed because the workload is dynamic and can be different from estimates because of daily and seasonal variations. Hybrid cloud provides a good workaround by moving resources on a need basis.

12.3.6.1 Cloud workloads

Cloud services can be used for a variety of purposes. Some of them have been discussed below:

- Client server: Cloud services can be used to host web servers and databases.
- Batch processing: Cloud services can be used for processing data as batches as need for business apps and payroll management.

- Data processing and analytics: Cloud services can be used for data intensive computing as needed for MapReduce and Spark.

- AI and Machine Learning: Recently, clouds have also been used for training machine learning model in which case servers with GPUs are being used.

- Specialized instances: Many of the highly specialized instances are conducted on supercomputers. Certain cloud providers now provide resources that can be used by such users which can be cheaper for them on a lease basis.

12.3.7 Cloud storage

Just like servers can be leased on the cloud platforms, storage can also be leased on the cloud platforms. Cloud platforms can give storage in the form of object storage in which any form of object like a file, a dataset etc. can be stored and retrieved using get and put methods. This storage is different from typical file systems that we deal with on most of the computers. Here, blobs of storage are used as the unit of storage. However, cloud platforms also have the provision for traditional disc storage/block storage on which a regular file system can be run. Note that in block storage, the disc is attached to a machine and only that particular machine can access that disc. A spin-off of this storage is the network file storage which is essentially a networked file system where the disc storage is shared among multiple machines sharing a network. There is also a provision for archival storage which is slower and cheaper in nature. It is mainly used for storing backups of data which are used infrequently. One notable use of cloud platforms as storage is Dropbox which doesn't provide any servers to the users and only provides them space. The user can simply upload their files/data in the Dropbox folder in one machine and it is automatically synchronized and made available on the other machines running the same Dropbox account using cloud as the intermediary where the files are uploaded. Other popular storage over cloud platforms include Google drive, One drive, Box etc. which essentially gives the user space on cloud accessible over the internet. Other cloud storage models are also available like cloud backups (iCloud), cloud media storage (Google photos) etc.

Question: Can we say that object storage is a key-value abstraction?

Answer: Yes, it is a key-value store. How that is stored internally depends. In some cases, it is stored on a file system. This is an implementation detail

12.3.8 Cloud orchestration

Cloud orchestration refers to the management of the cloud service. This is typically done via the cloud controller which is conceptually no different from a Kubernetes controller. Just like a k8s controller, it manages multiple customer requests using virtualization. VMs are created/migrated/terminated on servers to allocate and deallocate appropriate resources (like CPU, disc, network bandwidth) on the basis of the specific needs of the customer requests and the moving status of these requests. VMs can also be booted on the servers using specific images that meet specific customer needs. This cloud controller is very similar to k8s controller. IaaS platforms now provide containers and VMs. This container orchestration is very similar to what is done in Kubernetes but the difference is that it is used for third party customer requests.

Question: In the first part of the lab 1, if the number of requests cross the length of the bounded buffer, what needs to be done?

Answer: Ideally, one could just one wait. The producer thread will wait to put the request in the queue and all the incoming requests would be blocked at the socket level. This will be handled at the OS level and

need not be handled. However, requests can be dropped using a programmatic logic but this is not required for this lab.

Question: If the main server thread is blocked by a full bounded buffer queue, what happens to requests from other clients.

Answer: If the server is blocked in such a way, all the later incoming requests are blocked at the OS level. They are put in an OS buffer. This OS level buffer can fail if the load injected crosses a threshold but this would generally not happen.