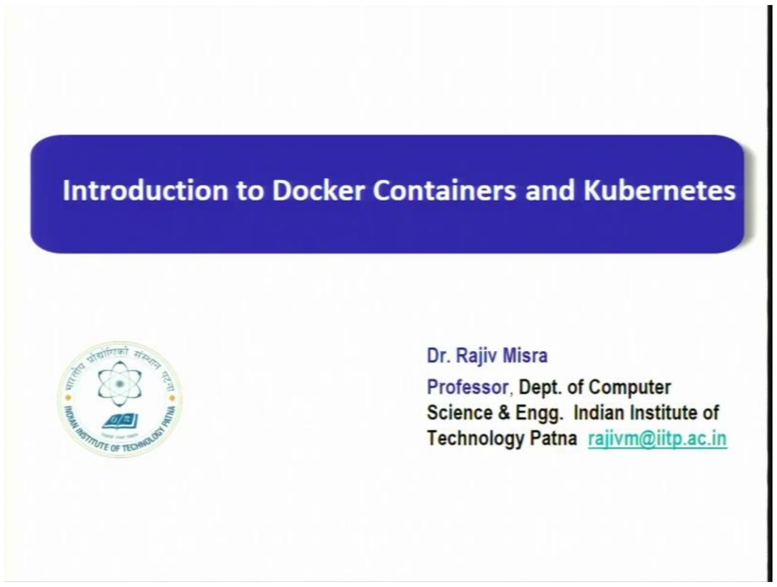
Foundation of Cloud IoT Edge ML

Professor Rajiv Misra
Department of Computer Science and Engineering
Indian Institute of Technology, Patna

Lecture 7
Introduction to Docker Containers and Kubernetes

Myself, Dr. Rajeev Mishra from IIT, Patna.

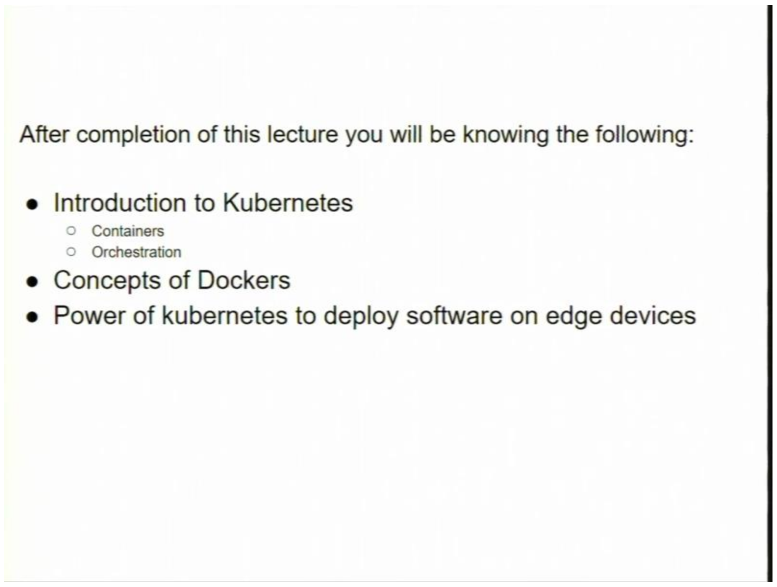
(Refer Slide Time: 00:17)


Introduction to Docker Containers and Kubernetes

Dr. Rajiv Misra
Professor, Dept. of Computer Science & Engg. Indian Institute of Technology Patna rajivm@iitp.ac.in

The topic of this lecture is introduction to Docker container and Kubernetes.

(Refer Slide Time: 00:27)


After completion of this lecture you will be knowing the following:

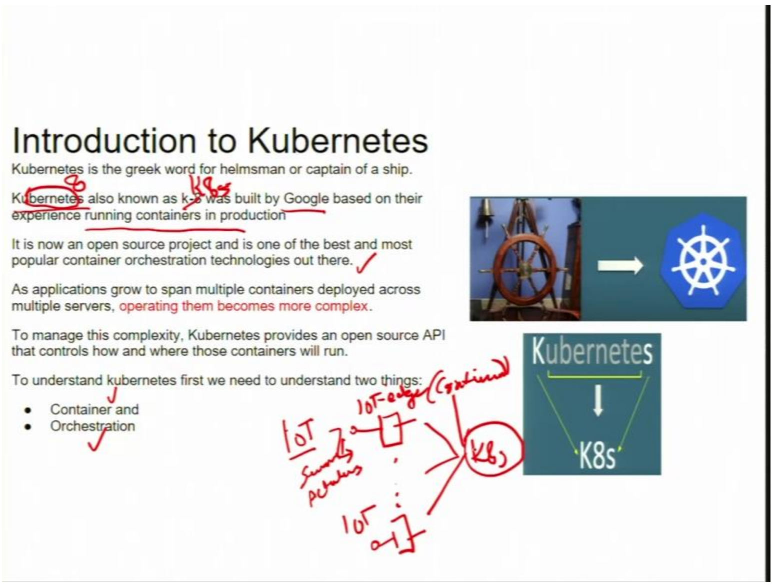
- Introduction to Kubernetes
  - Containers
  - Orchestration
- Concepts of Dockers
- Power of kubernetes to deploy software on edge devices

In this lecture, you will be knowing the following information. First, we will introduce to the Kubernetes. For that, we will require the background of container orchestration. We will also make you understand the concepts of Dockers and we will show you that due to the Kubernetes, you will be able to deploy and manage the softwares on the IoT edge. So, before I go ahead, let us set some motivation why we need to discuss Docker container and Kubernetes, three different technologies.

So, you know that IoT edge, to build the edge, you require this kind of technology which is called a Docker container. So, this Docker and container will allow the edge to run the functionality or capabilities of a cloud like, that is it may mimic the functionalities of the cloud at the edge. So, the enabling technology is Docker and container.

Now, to manage so many number of Docker and containers which are running over so many IoT devices, therefore manually it is not possible, therefore it requires an automation for this and therefore there is a need for container orchestration tool which is Kubernetes. So, Kubernetes in turn is used for the production. For example, if you are running a smart city application using IoT, then you require this kind of services also to run. So, with this motivation, let us go ahead and discuss about the Kubernetes.

(Refer Slide Time: 02:38)



Introduction to Kubernetes

Kubernetes is the greek word for helmsman or captain of a ship.

Kubernetes also known as k-8 was built by Google based on their experience running containers in production

It is now an open source project and is one of the best and most popular container orchestration technologies out there.

As applications grow to span multiple containers deployed across multiple servers, operating them becomes more complex.

To manage this complexity, Kubernetes provides an open source API that controls how and where those containers will run.

To understand kubernetes first we need to understand two things:

- Container and
- Orchestration

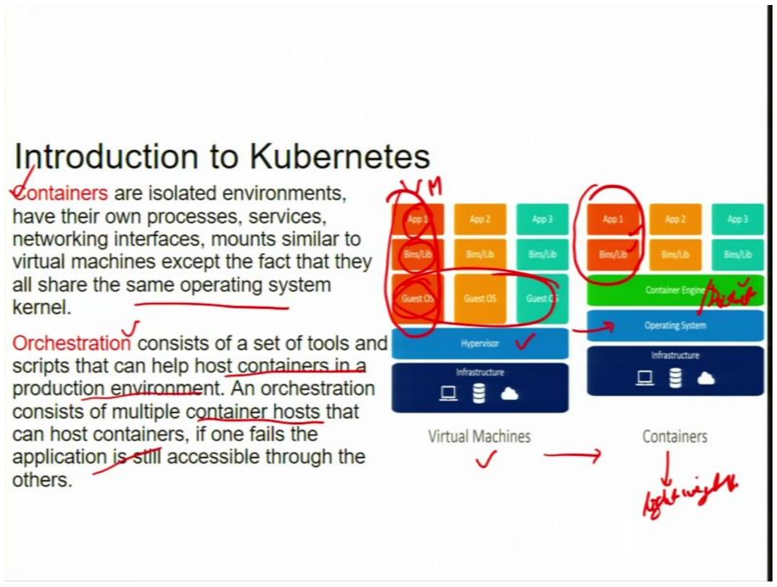
So, Kubernetes is a Greek word for a captain of a ship. So, Kubernetes is also known as K8S. So, K8S, so if you see the Kubernetes, so if you count the number of letters, they are 8 in number. So, K and S is retained and in between there are 8 letters. So, it is also known as K8S. So, this Kubernetes was built by Google and later on it was open sourced. This Kubernetes Google made for container orchestration or to use the containers in the production.

So, this open source project which is Kubernetes is one of the best and the most popular container orchestration technologies out there. The applications grow in size, therefore it spans over multiple containers which are deployed over multiple servers. Therefore, their orchestration or a lifetime management of these containers becomes a complex task manually to do it. To manage this complexity and automation with the help of Kubernetes which is an open source API controls how and where these containers will run.

Now, to understand this Kubernetes, we have to understand two concepts. One is about the container and the other is called orchestration. Now, before we go ahead, I will now let you aware about the use in the IoT. So, in the IoT parlance, this Kubernetes is for example, you have this kind of situation where you have an IoT device and so many IoT devices are now running and this IoT devices are having connectivity with the things, that is the sensors and actuators.

Now, to manage this particular IoT devices are also sometimes edge enabled. So, this IoT edge now is running the container or containerized applications. Now, these containers to manage the lifetime of a container, you require a service which is called a Kubernetes service and Kubernetes in turn will manage these containers so that if let us say that server or let us say some container is having an issue at a particular hardware, so the Kubernetes will automatically overcome from those kind of things. So, we will see all those details.

(Refer Slide Time: 05:48)



Introduction to Kubernetes

Containers are isolated environments, have their own processes, services, networking interfaces, mounts similar to virtual machines except the fact that they all share the same operating system kernel.

Orchestration consists of a set of tools and scripts that can help host containers in a production environment. An orchestration consists of multiple container hosts that can host containers, if one fails the application is still accessible through the others.

So, introduction to Kubernetes. Let us introduce two concepts to understand the background for Kubernetes. So, the containers is an isolated environment to have their own process, services, networking interface for an application that mount similar to the virtual machine except the fact that they all share the same operating system kernel.

Meaning to say that if you contrast with the virtual machine, so, in the virtual machine an application to run, it will be packed with the binaries and all dependencies which application needs to run and the operating system on which this application is running and this particular entire suite which comprises of the guest operating system, the binaries and libraries which is required to run the application, this is called virtual machine.

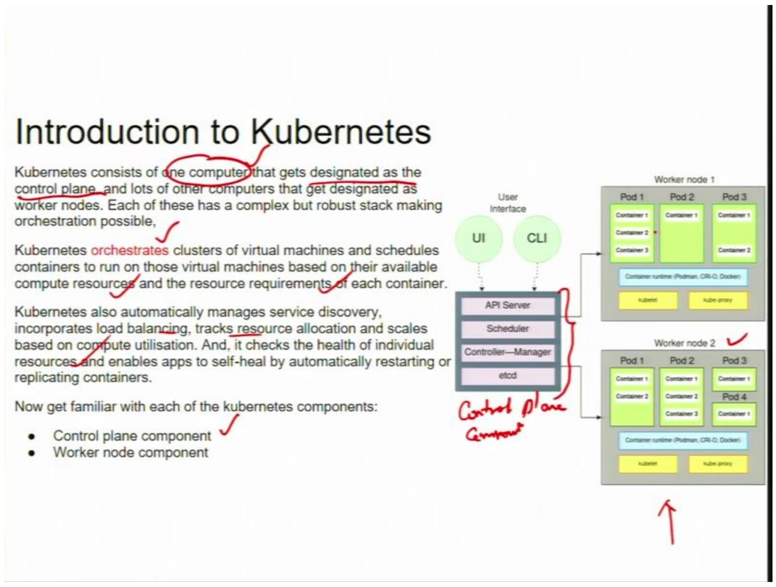
Now, virtual machine is supported to run on an hypervisor. Now, this entire scenario is now changed with the introduction of new technologies that is called containers. Now, containers do not carry the guest operating system along with the application. Therefore, you can see that in the container it is only the application and the related binaries.

So, therefore, this container is a lightweight and can be packed in many numbers compared to the virtual machines. Now, there is a container engine which is also popularly known as the docker and this docker supports running of these different containers and the operating system becomes a common entity for the container or a docker engine to run.

So, therefore, container is free from carrying the operating system and operating system becomes a common entity. So, this is a more efficient system or environment to run the application. Now, coming the orchestration. So, orchestration is a set of tools and scripts that can help the containers in a production environment.

So, an orchestration consists of multiple container hosts that hosts containers and if one of these applications is still accessible or through others if some containers are failed. So, that is what is the functionality of orchestration and Kubernetes is one of the orchestrator tool that is an open source developed by the Google that we will discuss at a later point of time in this further slides.

(Refer Slide Time: 08:54)



Introduction to Kubernetes

Kubernetes consists of one computer that gets designated as the control plane, and lots of other computers that get designated as worker nodes. Each of these has a complex but robust stack making orchestration possible,

Kubernetes orchestrates clusters of virtual machines and schedules containers to run on those virtual machines based on their available compute resources and the resource requirements of each container.

Kubernetes also automatically manages service discovery, incorporates load balancing, tracks resource allocation and scales based on compute utilisation. And, it checks the health of individual resources and enables apps to self-heal by automatically restarting or replicating containers.

Now get familiar with each of the kubernetes components:

- Control plane component
- Worker node component

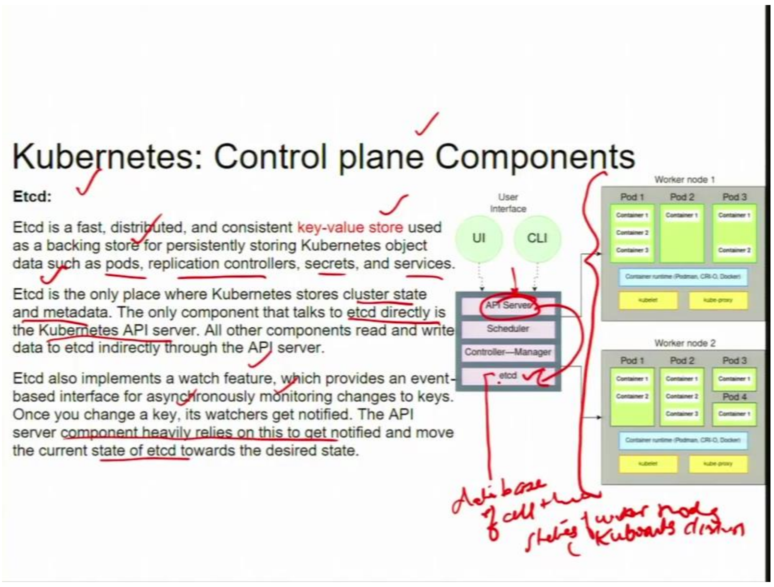
So, the Kubernetes consists of one computer that gets designated as a control plane. When we say computer may not be the physical computer, it may be the virtual machine as well. So, this particular Kubernetes has a designated system and that is called a control plane. Now, each of these has the complex, but robust stack making this orchestration possible.

So, Kubernetes orchestrates cluster of virtual machines. So, you can see here this is the cluster of virtual machines which is running these kind of worker nodes and schedules the container to run on those virtual machines based on their available compute resources and the resource requirement of each container.

So, the Kubernetes automatically manages the service directory, incorporates load balancing, tracks resource allocation, scales based on the compute utilization. It also checks the health of the individual resources, enables the application to self-heal automatically restarting or replicating the containers. You can get familiar with these Kubernetes components, that is the control plane component. So, this is the control plane component.

The other one is called the worker nodes. Now, let us go inside this details of this control plane components and then we will see the worker node component which will bring up this complete details of Kubernetes on the platform.

(Refer Slide Time: 10:45)



Kubernetes: Control plane Components

Etcd:

Etcd is a fast, distributed, and consistent key-value store used as a backing store for persistently storing Kubernetes object data such as pods, replication controllers, secrets, and services.

Etcd is the only place where Kubernetes stores cluster state and metadata. The only component that talks to etcd directly is the Kubernetes API server. All other components read and write data to etcd indirectly through the API server.

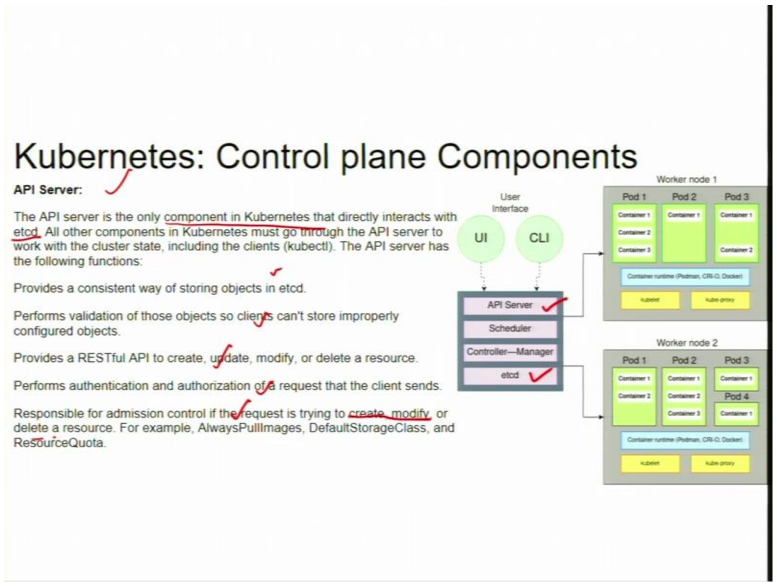
Etcd also implements a watch feature, which provides an event-based interface for asynchronously monitoring changes to keys. Once you change a key, its watchers get notified. The API server component heavily relies on this to get notified and move the current state of etcd towards the desired state.

So, Kubernetes control plane components comprises of Etcd. So, Etcd that is Etcd daemon also is a fast distributed and consistent key value store, used as the backing store for persistently storing Kubernetes object data such as parts, replication, controllers, secrets and services. Therefore, this Etcd is a database of all the states information of the worker nodes, that is the Kubernetes cluster.

So, Etcd is the only place where the Kubernetes stores the cluster state and metadata. So, this is one of the most important component of a Kubernetes which runs in the control plane. The only component that talks to the Etcd directly is the Kubernetes API server. So, all other components read and write data to the Etcd indirectly through API server. So, they have to now connect to the API server and through that they can access this Etcd.

So, Etcd is also implements a watch feature which provides a event based interface for asynchronously monitoring changes to the keys; once there is a change in the key its watchers get notified. So, the API server component heavily relies on this notified and moves to the current state of Etcd towards the desired state. So, the state of this entire Kubernetes cluster is maintained at Etcd and to access this Etcd or to modify that is done through API server.

(Refer Slide Time: 12:49)



Kubernetes: Control plane Components

API Server:

The API server is the only component in Kubernetes that directly interacts with etcd. All other components in Kubernetes must go through the API server to work with the cluster state, including the clients (kubectl). The API server has the following functions:

Provides a consistent way of storing objects in etcd.

Performs validation of those objects so clients can't store improperly configured objects.

Provides a RESTful API to create, update, modify, or delete a resource.

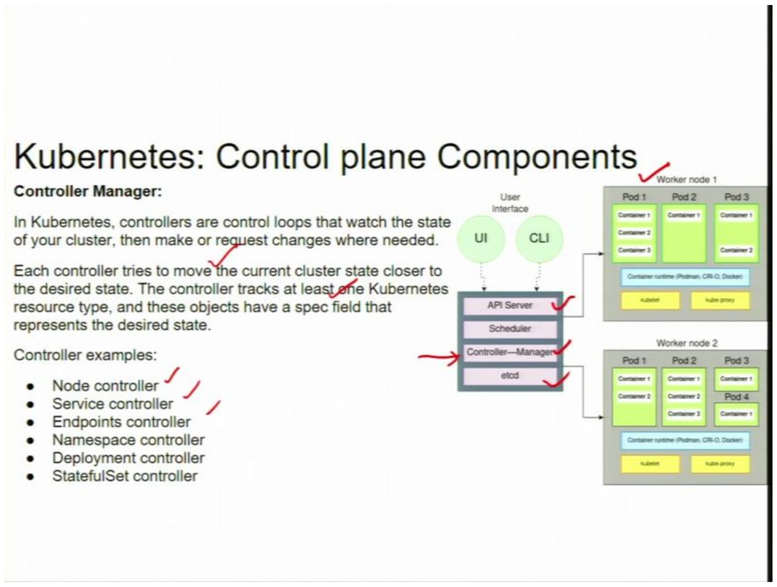
Performs authentication and authorization of a request that the client sends.

Responsible for admission control if the request is trying to create, modify, or delete a resource. For example, AlwaysPullImages, DefaultStorageClass, and ResourceQuota.

The other component of a control plane is API server. API server is the only component in the Kubernetes directly interacts with Etcd that we have already seen in the previous slide. So, the API server has the following functions. The first one is it provides the consistent way of storing the objects in Etcd, it performs the validation of these objects.

So, the client cannot improperly configure the objects, it provides the RESTful API to create, update or modify or delete the resources. So, these are the functionalities, it performs authentication and authorization of request that client sends and it is also responsible for the admission control if a request is to try, create, modify or delete a resource.

(Refer Slide Time: 13:41)



Kubernetes: Control plane Components

Controller Manager:

In Kubernetes, controllers are control loops that watch the state of your cluster, then make or request changes where needed.

Each controller tries to move the current cluster state closer to the desired state. The controller tracks at least one Kubernetes resource type, and these objects have a spec field that represents the desired state.

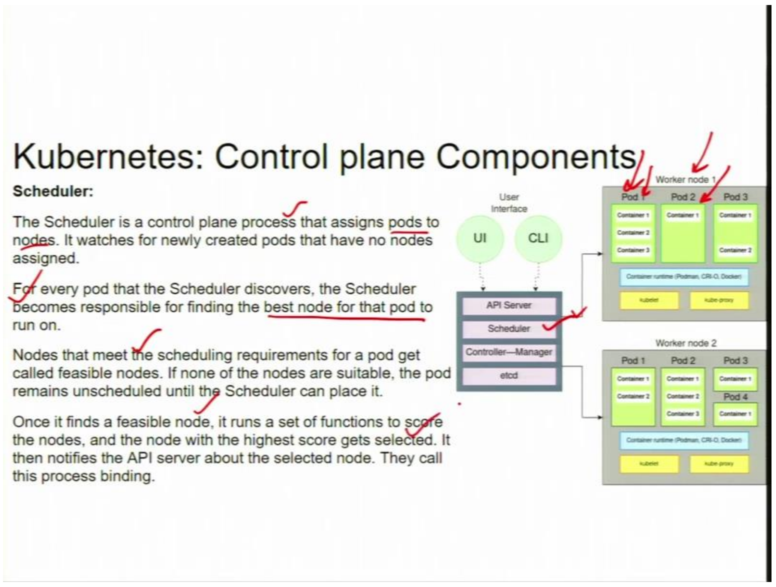
Controller examples:

- Node controller
- Service controller
- Endpoints controller
- Namespace controller
- Deployment controller
- StatefulSet controller

Now, we will look to the controller manager that is the third component of the control plane. So, in Kubernetes controllers are the control loop that watch the state of your cluster and then make the request changes wherever it is needed. Each controller tries to move the current cluster state closer to the desired state.

So, controller tracks at least one Kubernetes resource type and these objects have a specific field. So, controller types are mentioned over here that is the node controller, service controller, endpoint, namespace controller, deployment and statefulset controller.

(Refer Slide Time: 14:23)



Kubernetes: Control plane Components

Scheduler:

The Scheduler is a control plane process that assigns pods to nodes. It watches for newly created pods that have no nodes assigned.

For every pod that the Scheduler discovers, the Scheduler becomes responsible for finding the best node for that pod to run on.

Nodes that meet the scheduling requirements for a pod get called feasible nodes. If none of the nodes are suitable, the pod remains unscheduled until the Scheduler can place it.

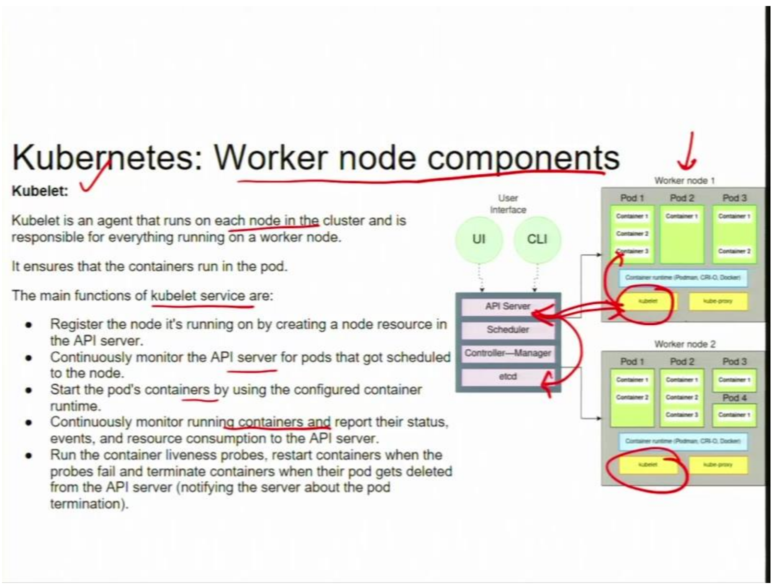
Once it finds a feasible node, it runs a set of functions to score the nodes, and the node with the highest score gets selected. It then notifies the API server about the selected node. They call this process binding.

Now, comes finally, the last component in the control plane called scheduler. So, scheduler is the control plane process that assigns parts to the nodes. So, when we say node we mean that the worker node and this becomes the cluster, Kubernetes cluster, part of the Kubernetes cluster. So, the unit of assignment is called the pods and the pods contains the cluster. So, therefore, the scheduler is a process that assigns pods to the nodes.

So, which pods are running in in these nodes that is the job of the scheduler. So, scheduler is very important it now creates the pods which is the unit of execution. So, for every pod the scheduler discovers, the scheduler becomes a responsible of finding the best node for that pod to run on depending upon what are the resource requirement for the pod and where that resource is available in which of the nodes.

So, that particular mapping is done by the scheduler because the scheduler knows the current state of all the nodes and when a pod is requesting to be assigned a particular node then this scheduler does that. So, once it finds a feasible node it runs a set of functions to score the nodes and the nodes with the highest score get selected, this is the method which scheduler uses.

(Refer Slide Time: 15:55)



Kubernetes: Worker node components

Kubelet:

Kubelet is an agent that runs on each node in the cluster and is responsible for everything running on a worker node.

It ensures that the containers run in the pod.

The main functions of kubelet service are:

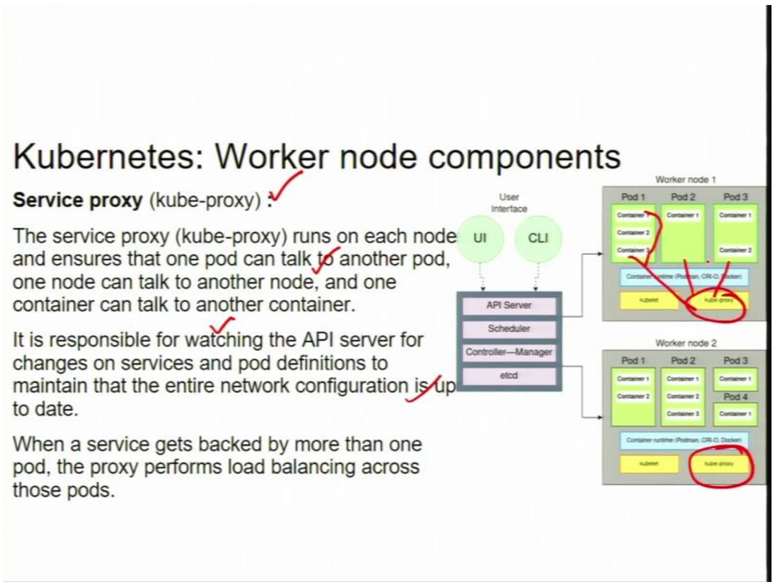
- Register the node it's running on by creating a node resource in the API server.
- Continuously monitor the API server for pods that got scheduled to the node.
- Start the pod's containers by using the configured container runtime.
- Continuously monitor running containers and report their status, events, and resource consumption to the API server.
- Run the container liveness probes, restart containers when the probes fail and terminate containers when their pod gets deleted from the API server (notifying the server about the pod termination).

Finally, now we will be looking up the worker node components. So, in the worker node if you see there is a component which is called a kubelet. So, kubelet is an agent that runs in each node you can see that here there is an example of two nodes - in every node this kubelet is running as an agent responsible for everything running on the worker node.

So, they kubelet controls the entire functionality of the worker node. It ensures that the containers run in a pod the main functions of a kubelet services are register the nodes it is running on by creating the node resource in the API server. So, it interacts with the API server. To do this continuously monitor the API server for the pods that got scheduled.

Start the pods containers by using the configured container runtime, continuously monitor the containers and report their status events and resource consumption to the API server. So, you can see that the API server used to update the state in the Etcd; for that the kubelet has to keep on continuously informing the API server about its current state in the worker node.

(Refer Slide Time: 17:16)



Kubernetes: Worker node components

Service proxy (kube-proxy) :

The service proxy (kube-proxy) runs on each node and ensures that one pod can talk to another pod, one node can talk to another node, and one container can talk to another container.

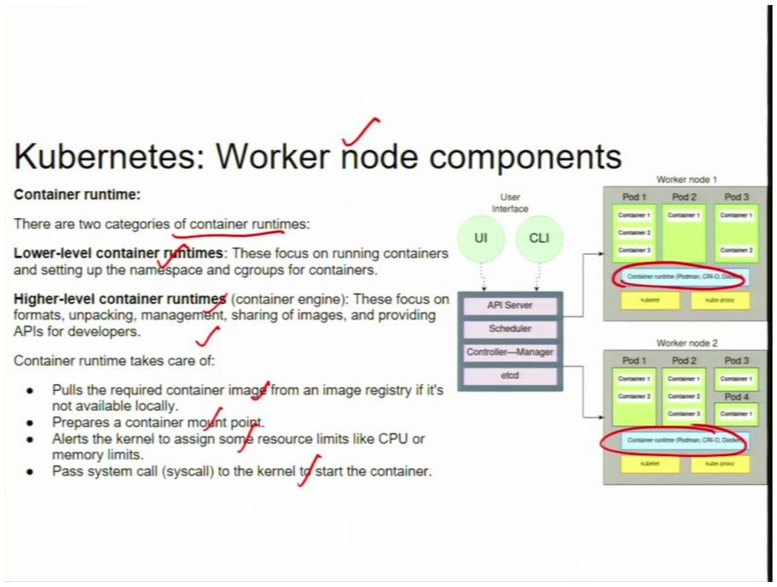
It is responsible for watching the API server for changes on services and pod definitions to maintain that the entire network configuration is up to date.

When a service gets backed by more than one pod, the proxy performs load balancing across those pods.

Then comes the kube proxy. The kube proxy also runs as a service in every node, worker node. So, the kube proxy runs on each node and ensures that one pod can talk to another pod and one node can talk to another node and one container can talk to another container. So, that means, kube proxy becomes a point of interaction or a communication just like you can think of a proxy server working in the internet for internet connectivity.

So, this kube proxy is responsible for watching the API server for changes on the services and the pod definition to maintain the entire network configuration up to date. So, this kube proxy is the mechanism by which all the nodes, all the containers they are properly networked and they will be able to communicate with each other.

(Refer Slide Time: 18:13)



Kubernetes: Worker node components

Container runtime:

There are two categories of container runtimes:

Lower-level container runtimes: These focus on running containers and setting up the namespace and cgroups for containers.

Higher-level container runtimes (container engine): These focus on formats, unpacking, management, sharing of images, and providing APIs for developers.

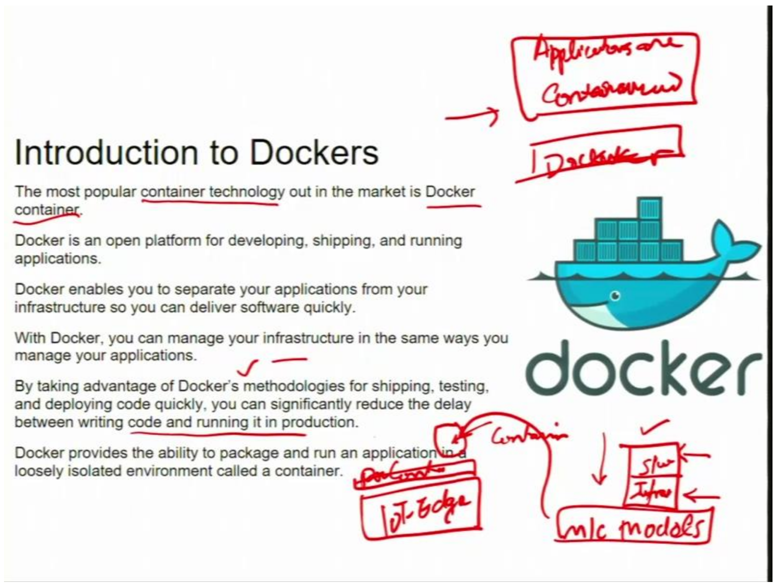
Container runtime takes care of:

- Pulls the required container image from an image registry if it's not available locally.
- Prepares a container mount point.
- Alerts the kernel to assign some resource limits like CPU or memory limits.
- Pass system call (syscall) to the kernel to start the container.

Similarly, now there is a thing which is called a container runtime. So, container runtime is also a worker node component. So, it has two categories of container runtime - one is that lower level container runtime focuses on running containers and setting up the name space; a high level container runtime focus on formats unpacking management and sharing of the images.

So, container runtime has to take care of pull the required container image from an image registry if it is not available locally, prepare a container mount point and alerts the container to assign some resources and pass system call to the kernel to start the container. So, therefore container runtime is almost like an operating system of a worker node which is now supporting these containers and the pods.

(Refer Slide Time: 19:15)



Introduction to Dockers

The most popular container technology out in the market is Docker container.

Docker is an open platform for developing, shipping, and running applications.

Docker enables you to separate your applications from your infrastructure so you can deliver software quickly.

With Docker, you can manage your infrastructure in the same ways you manage your applications.

By taking advantage of Docker's methodologies for shipping, testing, and deploying code quickly, you can significantly reduce the delay between writing code and running it in production.

Docker provides the ability to package and run an application in a loosely isolated environment called a container.

Now let us introduce about the dockers. So, the most popular container technology is called the docker container. So, docker is an open platform for developing shipping and running applications. So, docker enables you to separate your application from your infrastructure, so that you can deliver the software quickly. Now there are two important thing - in most of the cases there is a software and there is an infrastructure and the hardware.

Normally to make this management easy that is the management of infrastructure to be separated out of the software, this particular separation is done with this help of this particular technology. So, with docker you can manage your infrastructure in the same way as you manage your applications.

So, by taking the advantage of the dockers methodologies for shipping testing and deploying the code, you can significantly reduce the delay between the writing a code and running on the production. So, docker provides the ability to package and run the application in a loosely isolated environment which is called the container. So, let me explain you in more detail before I go ahead the use of docker. Now you know that the applications are containerized.

What do you mean by these applications are containerized? Now when you write an application and the developer develops on a particular computer that application has to run on different kind of environment and systems. Therefore, the lifetime of the application where it runs is not very clear at the time of development.

To make that application independent of the system or the dependencies therefore this application to make this independent of a system, so, application has to be or need to be containerized and the environment is to be infrastructure free. So, docker is the technology which gives these application make independent of the platform dependent.

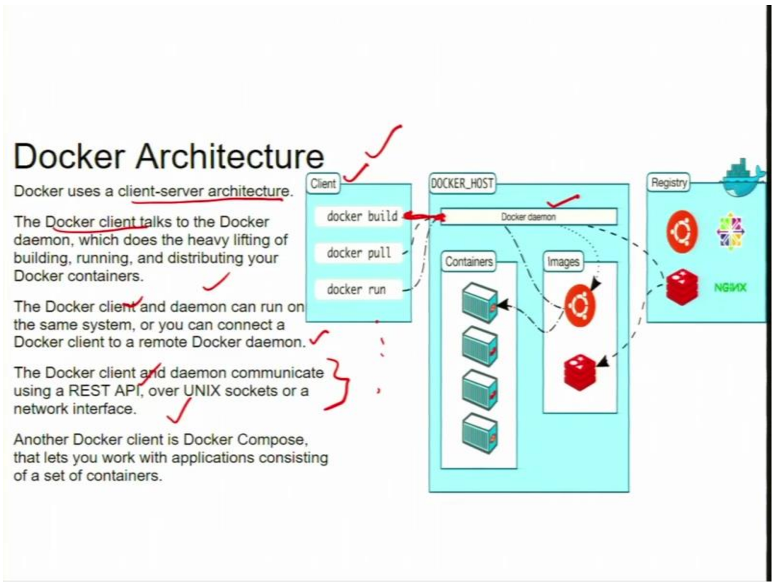
So, the applications are containerized. Containerized means that all the dependencies of the application such as the libraries and binaries, they are packed together and they are called containers. As far as the operating system is concerned if let us say the application is tied up with the operating system then without that operating system again the application will not be running it.

So, to make it independent the container you know that does not contain the operating system, it is out of that. Now this hardware and all the operating system dependencies to make it independent, the docker engine comes into the play. Once you install the docker engine then it will be the point where these containers will be running their applications.

So, therefore in an IoT application, IoT nodes or edge nodes that is also called an IoT edge will run this containers or a docker container. So, that means first docker will be running on this IoT edge hardware and on top of it the containers will be placed and they will be running.

Now if the machine learning you want to run on IoT edge then the machine learning models need to be containerized and once it is containerized then they can run as a container onto that IoT edge. So, therefore we are now discussing we have already told you about the container, now docker we are telling that to run the container you require the docker engine. So, docker engine is nothing but a kind of operating system for the containers.

(Refer Slide Time: 23:37)



Docker Architecture

Docker uses a client-server architecture.

The Docker client talks to the Docker daemon, which does the heavy lifting of building, running, and distributing your Docker containers.

The Docker client and daemon can run on the same system, or you can connect a Docker client to a remote Docker daemon.

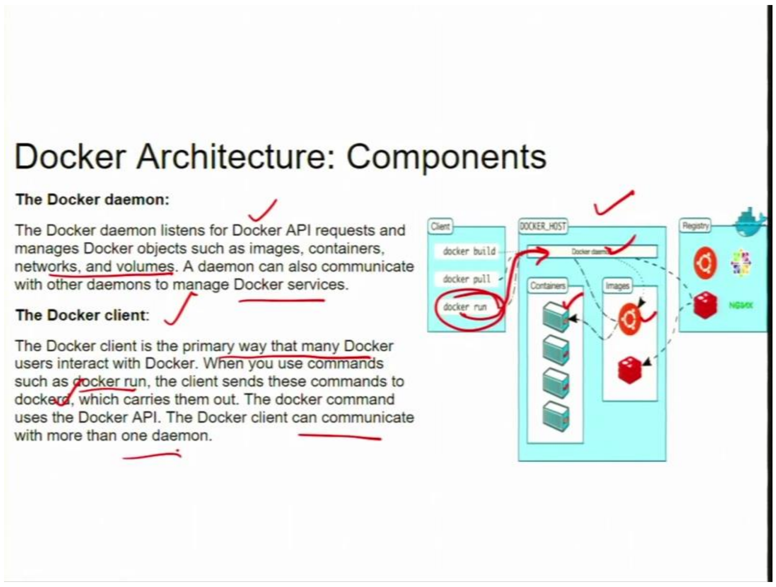
The Docker client and daemon communicate using a REST API, over UNIX sockets or a network interface.

Another Docker client is Docker Compose, that lets you work with applications consisting of a set of containers.

So, let us see the architecture of the docker. So, docker uses a client server architecture. So, there is a docker client. So, docker client talks to the docker daemon. So, here there is a docker daemon and here there is a docker client. Why the docker client talks to the docker daemon which does the heavy lifting of the building running and distributing your docker containers. So, docker client and the docker daemon can run on the same system and you can connect a docker client to a docker daemon.

So, docker client and docker daemon communicates with the help of rest API over the unique socket or on some network interface. So, this endpoint is explained over here as either the rest API over the unique socket or network interface. Now another docker client is the docker compose. So, and that lets you work in the application consisting of the set of containers.

(Refer Slide Time: 24:49)



Docker Architecture: Components

The Docker daemon:

The Docker daemon listens for Docker API requests and manages Docker objects such as images, containers, networks, and volumes. A daemon can also communicate with other daemons to manage Docker services.

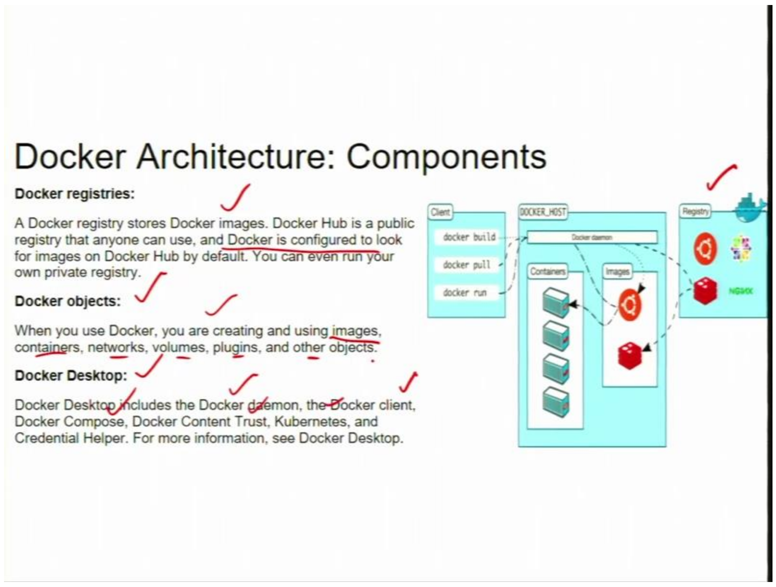
The Docker client:

The Docker client is the primary way that many Docker users interact with Docker. When you use commands such as docker run, the client sends these commands to dockerd, which carries them out. The docker command uses the Docker API. The Docker client can communicate with more than one daemon.

So, docker daemon then listens to the docker API request and manages the docker objects such as the images, containers, network and volume. So, the docker a daemon can also communicate with other daemons to manage the docker services.

So, the docker client is the primary way that the docker users interact with the docker and when you give a command such as docker run, then the client sends these commands to the docker daemon where it carries them out. So, when you say docker run at the docker client then this particular command will be now picked up by docker daemon and it will start the execution or with one or more docker daemon.

(Refer Slide Time: 25:53)



Docker Architecture: Components

Docker registries:

A Docker registry stores Docker images. Docker Hub is a public registry that anyone can use, and Docker is configured to look for images on Docker Hub by default. You can even run your own private registry.

Docker objects:

When you use Docker, you are creating and using images, containers, networks, volumes, plugins, and other objects.

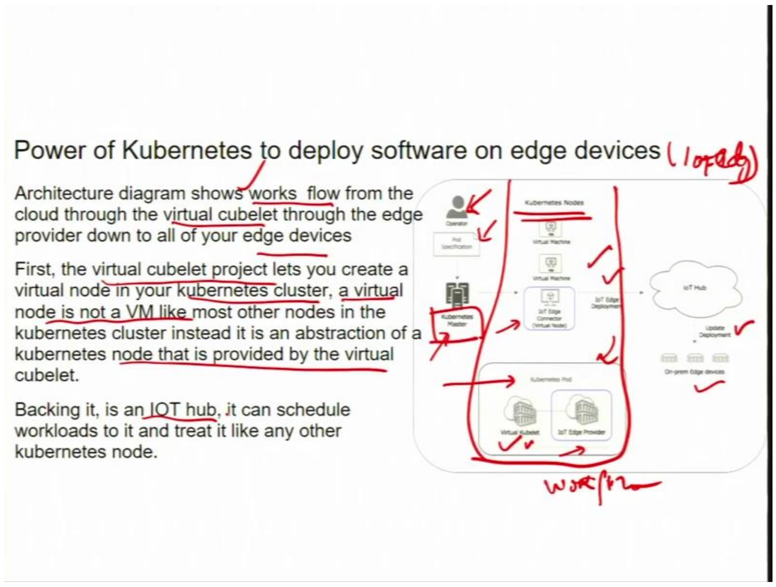
Docker Desktop:

Docker Desktop includes the Docker daemon, the Docker client, Docker Compose, Docker Content Trust, Kubernetes, and Credential Helper. For more information, see Docker Desktop.

So, docker registry is also a component, it stores the docker images. So, docker hub is a public registry that anyone can use it. So, docker is configured to look for the images on the docker hub by default and you can run, even run your own private registry. So, let us see about the docker objects. So, docker objects, so when you use the docker you are creating and using the images, containers, networks, volume, plugins and other objects.

So, these particular images is completely packed up all the dependencies of an application and the application and also often includes the containers, networks, volumes, plugin and other object. So, docker desktop includes the docker daemon, the docker client, docker compose, docker trust, kubernetes and other more things.

(Refer Slide Time: 26:54)



Power of Kubernetes to deploy software on edge devices

Architecture diagram shows works flow from the cloud through the virtual cubelet through the edge provider down to all of your edge devices

First, the virtual cubelet project lets you create a virtual node in your kubernetes cluster, a virtual node is not a VM like most other nodes in the kubernetes cluster instead it is an abstraction of a kubernetes node that is provided by the virtual cubelet.

Backing it, is an IOT hub, it can schedule workloads to it and treat it like any other kubernetes node.

Now, let us see that how kubernetes you can use to deploy the software on the edge devices. So, this also is applicable for an IOT edge. So, the architecture shows the flow from the cloud through a virtual cubelet. So, here you can see this particular illustration the workflow from the cloud through the virtual cubelets through the edge provider down to your edge devices.

So, this entire thing is explained here, the workflow. So, here you have to see that there is a operator and you have to, this operator specifies for the pod and then this particular pod specification will map to the kubernetes master and the kubernetes master in turn will find out a virtual machine on the kubernetes nodes.

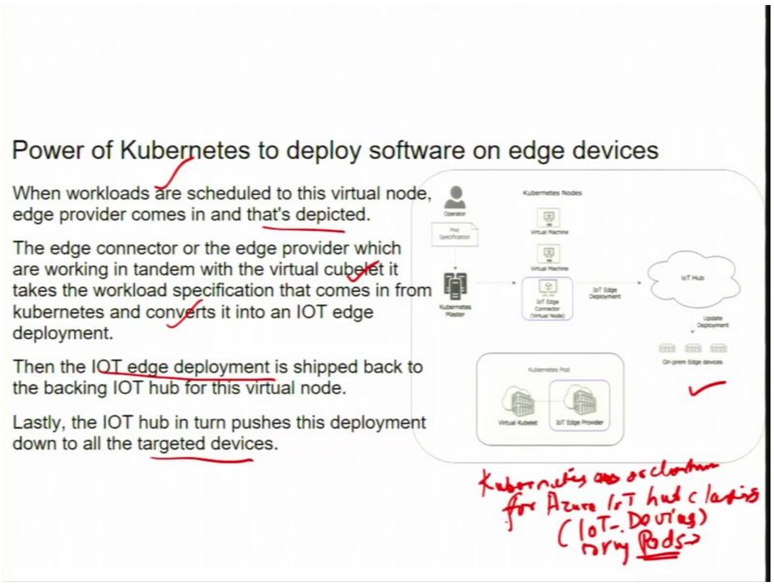
So, this you know that scheduler does this kind of mapping and then IOT edge is called a virtual node. If let us say there you want to install the IOT Azure IOT edge services, so, it requires that particular device to be first virtualized and run a virtual machine on top of it, that particular node becomes the virtual node where this IOT edge connector will run.

And then IOT edge deployment will happen with the help of IOT hub and this particular deployment on premise edge devices will be running these IOT edge and this kubernetes master will manage it. Now, these kubernetes nodes will run the parts in the nodes that we have seen in the previous diagram.

So, that pod is called a kubernetes pod. Kubernetes pod will have the virtual kubelet and this virtual kubelet in turn will manage that pod inside that kubernetes and IOT edge provider will be now using this orchestrator called kubernetes master. So, first the virtual kubelet project lets you create the virtual node in your kubernetes cluster and that kubernetes cluster consists of these kind of things or this is the nodes which are running as IOT nodes.

A virtual node is like the virtual machine like most other nodes in the kubernetes cluster instead it is an abstraction of a kubernetes node that is being provided by virtual kubelet. So, it is an IOT hub, it can schedule the workloads to it and treat it like any other kubernetes node.

(Refer Slide Time: 30:07)



Power of Kubernetes to deploy software on edge devices

When workloads are scheduled to this virtual node, edge provider comes in and that's depicted.

The edge connector or the edge provider which are working in tandem with the virtual cubelet it takes the workload specification that comes in from kubernetes and converts it into an IOT edge deployment.

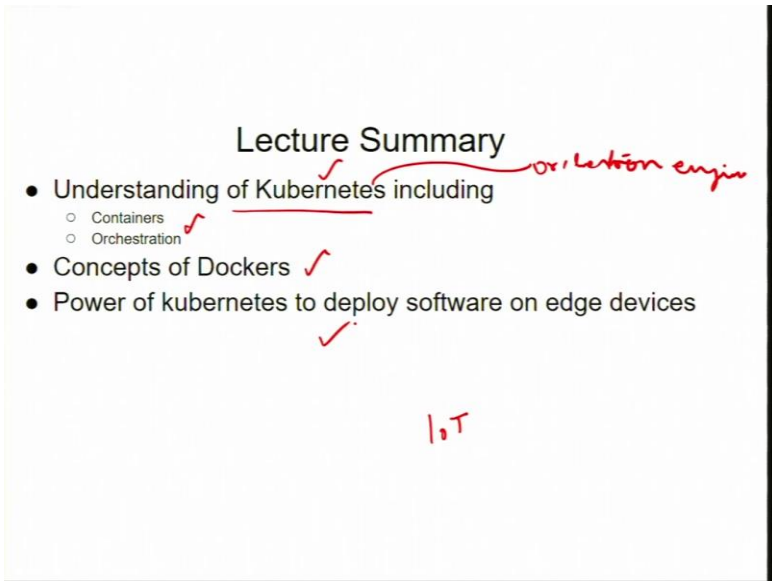
Then the IOT edge deployment is shipped back to the backing IOT hub for this virtual node.

Lastly, the IOT hub in turn pushes this deployment down to all the targeted devices.

So, when a workload are scheduled to this virtual node edge provider comes in, that is depicted over here. So, the edge connector or the edge provider which are working in tandem with the virtual kubelet it takes the workload specification that comes in from the kubernetes and converts it into an IOT edge deployment. Then this IOT edge deployment is shipped back to the backing of IOT hub for it is virtual node.

Lastly the IOT hub in turn pushes this deployment down to the target devices. So, therefore, what we have seen over here is the entire ecosystem of using the kubernetes as the orchestrator for managing the azure IOT hub clusters that is nothing, but the IOT devices which are running these kubernetes pods. And inside pod you know that containers are running and containers is nothing but containerized application which runs on the IOT edge device.

(Refer Slide Time: 31:50)



Lecture Summary

- Understanding of Kubernetes including
  - Containers
  - Orchestration
- Concepts of Dockers
- Power of kubernetes to deploy software on edge devices

orchestration engine
IoT

So, with this let us summarize the lecture. So, what we have seen is the use of kubernetes very much in the context of IOT. So, you know that kubernetes is an orchestrator, orchestration engine. Why this is needed is because so many IOT devices are running their software with the help of containers and this container orchestration that is a lifetime of a container requires continuous monitoring, states of these containers which are running inside a pod of a kubernetes cluster requires automation like kubernetes.

So, we have already discussed about the container. So, container is a process with the help of docker and container. We have seen that this particular application to make it platform independent and run on any of this IOT devices, we require the concept of a docker and container and kubernetes is automatic management that is at the production time. So, power of the kubernetes we have seen is to deploy the software on IOT edge devices. So, with this I conclude this lecture. Thank you very much.