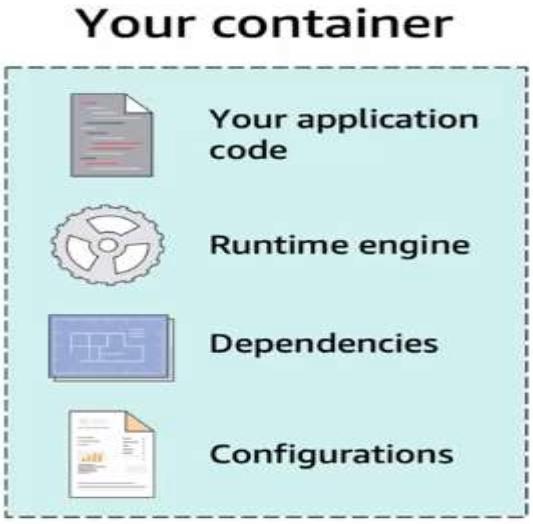
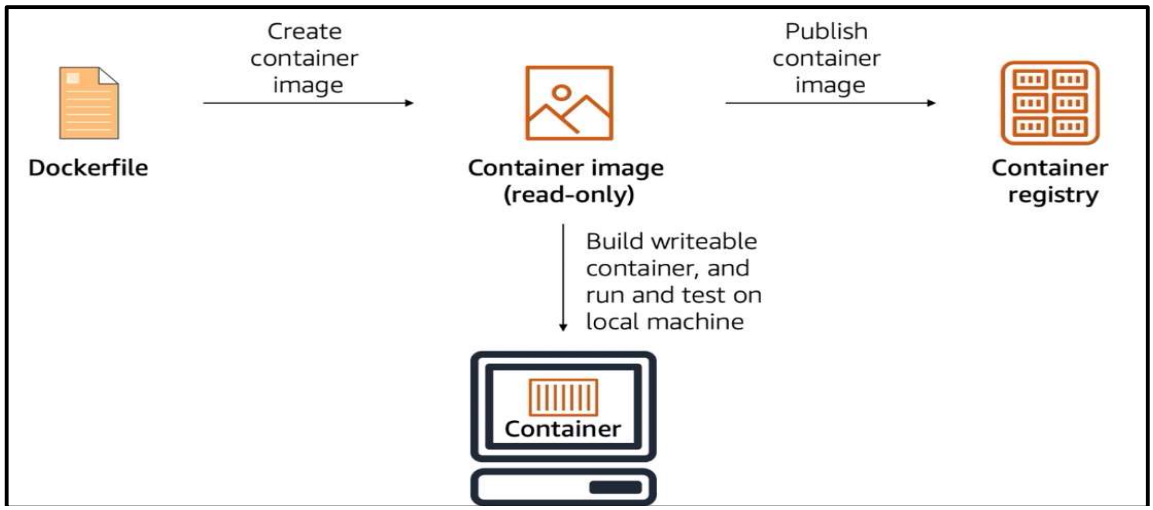
## Need for Containers




- Containers are a method of operating system virtualization that enables you to run an application and its dependencies in resource-isolated processes.
- A container is a lightweight, standalone software package.
- It contains everything that a software application needs to run, such as the application code, runtime engine, system tools, system libraries, and configurations.
- Containers can help ensure that applications deploy quickly, reliably, and consistently, regardless of deployment environment (Developer Env, Test env and production env).
- A single server can host several containers that all share the underlying host system's OS kernel.
- These containers might be services that are part of a larger enterprise application, or they might be separate applications that run in their isolated environment.
- **A container is created from a read-only template that is called an image.**

**How containers are created?**

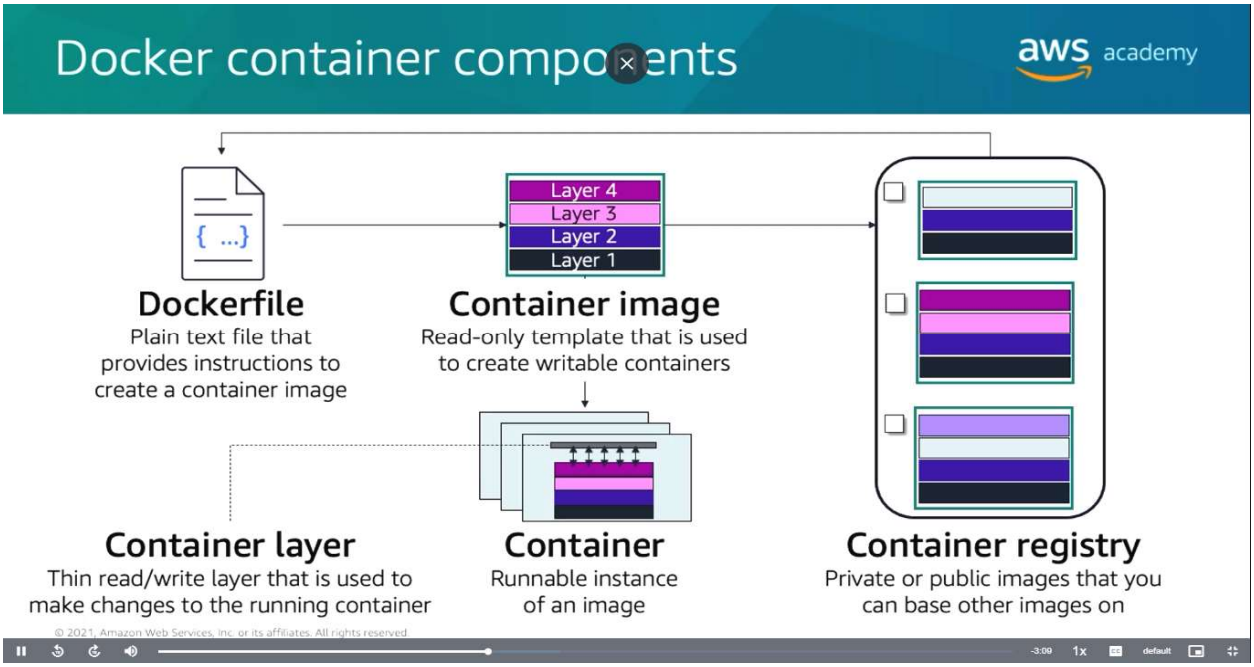
- A container is created from a read-only template that is called an image.
- Images are typically built from a Dockerfile, which is a plaintext file that specifies all the components that are included in the container.
- You can create images from scratch, or you can use images that others created and published to a public or private container registry.
- A container image is the snapshot of the file system that is available to the container.
- For example, you might have the Debian operating system as a container image. When you run this container, a Debian operating system is available to it.
- You can also package all your code dependencies in the container image and use it as your code artifact.
- Container images are stored in a registry.
- You can download the images from the registry and run them on your cluster.
- Registries can exist in or outside your AWS infrastructure




## Dockers

- Docker is a portable runtime application environment.
- You can package an application and its dependencies into a single, immutable artifact that is called an image.
- After you create a container image, it can go anywhere that Docker is supported.
- You can run different application versions with different dependencies simultaneously.
- These benefits lead to much faster development and deployment cycles, and better resource utilization and efficiency. All of these abilities are related to agility

## Docker Components / How container are created?



- Docker containers are created from read-only templates, which are called container images.
- Images are immutable and highly portable.
- You can port an image to any environment that supports Docker.
- Images are built from a Dockerfile, which is a plain text file that specifies all of the components that are included in the container.
- Instructions in the Dockerfile create layers in the container image.
- You can create images from scratch, or you can use images that others created and published to a public or private container registry.
- The image could also install a web server and your application, in addition to the essential configuration details to make your application run.
- A container is a runnable instance of an image.
- You can create one container or several containers based on that image.
- Docker file simple example

```
# Start with the Ubuntu latest image
FROM ubuntu:latest

# Output hello world message
CMD echo "Hello World!"
```

- To build your own image, you create a Dockerfile by using a simple syntax to define how to create the image and run it.
- Each instruction in a Dockerfile creates a read-only layer in the image.
- In this simple example, you start with the Ubuntu latest image that is already created for you, and hosted on Docker Hub or some other site.
- The only thing that you do is add a command to echo the message Hello World! after the container is instantiated.

## Docker CLI Commands

| Command | Description |
|---|---|
| `docker build` | Build an image from a Dockerfile. |
| `docker images` | List images on the Docker host. |
| `docker run` | Launch a container from an image. |
| `docker ps` | List the running containers. |
| `docker stop` | Stop a running container. |
| `docker start` | Start a container. |
| `docker push` | Push the image to a registry. |
| `docker tag` | Tag an image. |

| Command | Description |
|---|---|
| `docker logs` | View container log output. |
| `docker port` | List container port mappings. |
| `docker inspect` | Inspect container information. |
| `docker exec` | Run a command in a container. |
| `docker rm` | Remove one or more containers. |
| `docker rmi` | Remove one or more images from the host. |
| `docker update` | Dynamically update the container configuration. |
| `docker commit` | Create a new image from a container's changes. |

## AWS ECR

- Amazon Elastic Container Registry (Amazon ECR) is an AWS managed container image registry service.
- Amazon ECR supports private repositories, it means specified users or Amazon EC2 instances can access your container repositories and images.
- You can use your preferred CLI to push, pull, and manage Docker images.
- Components of AWS ECR:
  - ✓ Registry: An Amazon ECR private registry is provided to each AWS account; you can create one or more repositories in your registry and store images in them.
  - ✓ Authorization token: Your client must authenticate to Amazon ECR registries as an AWS user before it can push and pull images.
  - ✓ Repository: An Amazon ECR repository contains your Docker images.
  - ✓ Repository policy: You can control access to your repositories and the images within them with repository policies.
  - ✓ Image: You can push and pull container images to your repositories.

## Pushing to Amazon ECR

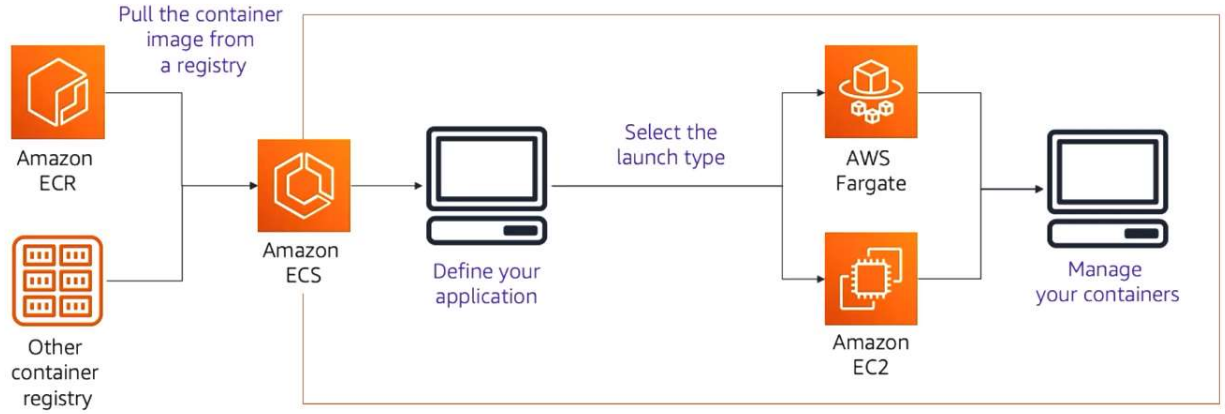
- You can push your Docker images to your private repositories.
- Amazon ECR requires that users must have the permissions to push images.
- You can push your container images to an Amazon ECR repository with the **docker push** command.
- The Amazon ECR repository must exist before you push the image.
- **To push a Docker image to an Amazon ECR repository**
  - ✓ Authenticate your Docker client to the Amazon ECR registry to which you intend to push your image. Authentication tokens must be obtained for each registry used, and the tokens are valid for 12 hours.
    ***aws ecr get-login-password --region region | docker login --username AWS --password-stdin aws_account_id.dkr.ecr.region.amazonaws.com***
  - ✓ Identify the local image to push. Run the **docker images** command to list the container images on your system.
    ***docker images***
  - ✓ Tag your image with the Amazon ECR registry, repository, and optional image tag name combination to use.

***docker tag e9ae3c220b23 aws_account_id.dkr.ecr.region.amazonaws.com/my-repository:tag***

- ✓ Push the image using the **docker push** command: ***docker push aws_account_id.dkr.ecr.region.amazonaws.com/my-repository:tag***

## AWS ECS

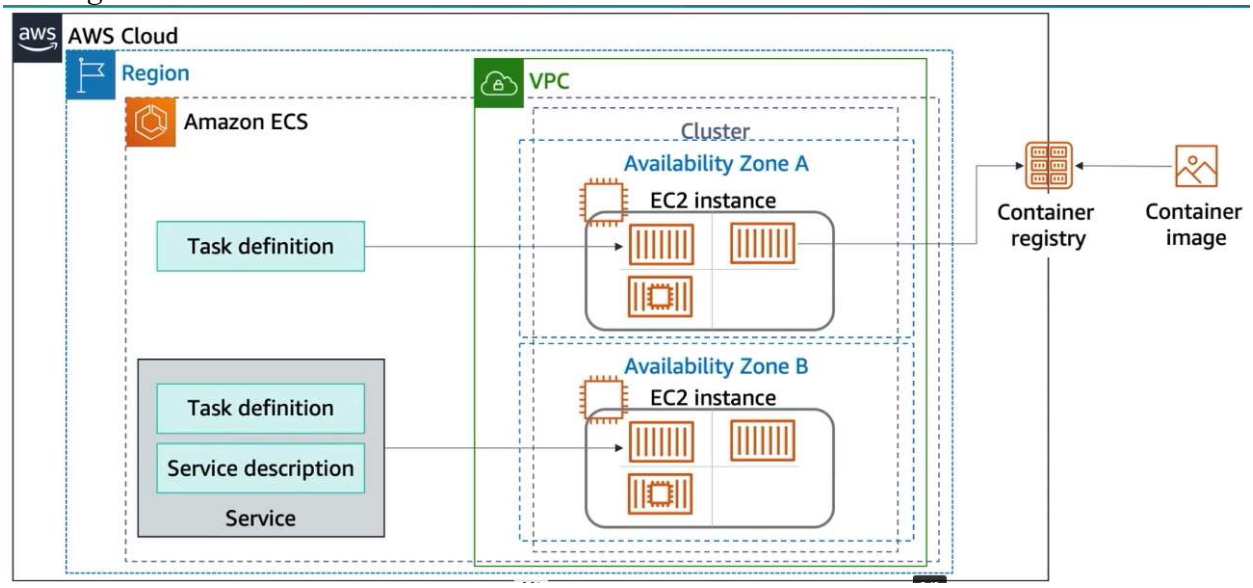
- Amazon Elastic Container Service (Amazon ECS) is a highly scalable and fast container management service.
- You can use it to run, stop, and manage containers on a cluster.
- With Amazon ECS, your containers are defined in a task definition that you use to run an individual task or task within a service.
- In this context, a service is a configuration that you can use to run and maintain a specified number of tasks simultaneously in a cluster.
- You can run your tasks and services on a serverless infrastructure that's managed by AWS Fargate.
- Alternatively, for more control over your infrastructure, you can run your tasks and services on a cluster of Amazon EC2 instances that you manage.




- Amazon ECS provides the following features:
  - A serverless option with AWS Fargate. With AWS Fargate, you don't need to manage servers, handle capacity planning, or isolate container workloads for security.
  - Integration with AWS Identity and Access Management (IAM). You can assign granular permissions for each of your containers.

## ECS Components

- An Amazon ECS cluster is a logical grouping of tasks or services. You can use clusters to isolate your applications.
- Container is a standardized unit of software development that holds everything that your software application requires to run. This includes relevant code, runtime, system tools, and system libraries. Containers are created from a read-only template that's called an *image*.

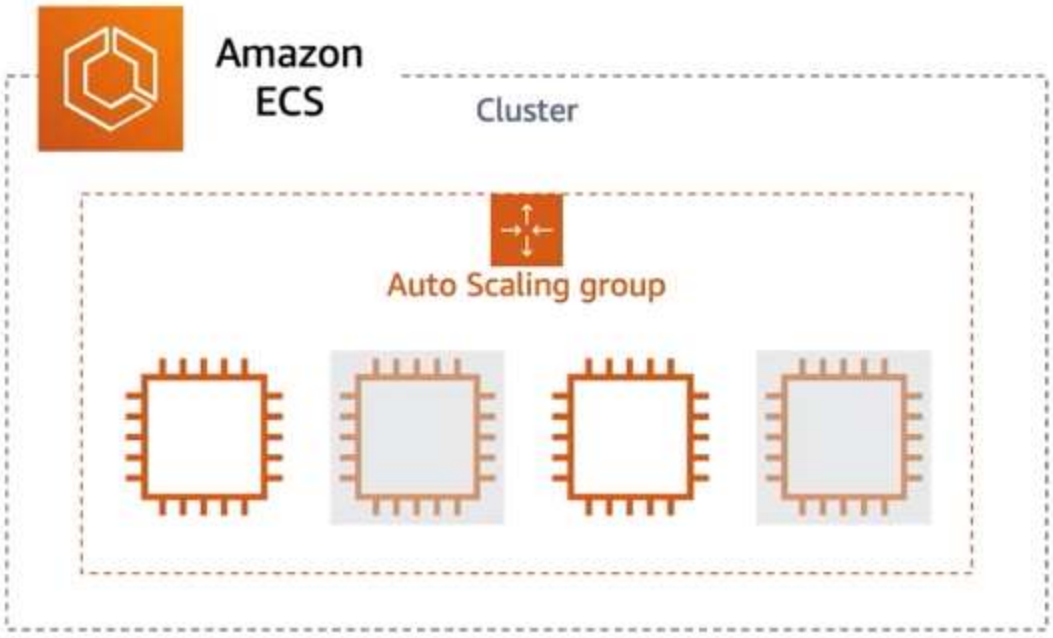



- A *task definition* is a text file that describes one or more containers that form your application. It's in JSON format. You can use it to describe up to a maximum of ten containers. The task definition functions as a blueprint for your application. It specifies the various parameters for your application. For example, you can use it to specify parameters for the operating system, which containers to use, which ports to open for your application, and what data volumes to use with the containers in the task.
- A *task* is the instantiation of a task definition within a cluster. After you create a task definition for your application within Amazon ECS, you can specify the number of tasks to run on your cluster. You can run a standalone task, or you can run a task as part of a service.
- You can use an Amazon ECS *service* to run and maintain your desired number of tasks simultaneously in an Amazon ECS cluster. How it works is that, if any of your tasks fail or stop for any reason, the Amazon ECS service scheduler launches another instance based on your task definition. It does this to replace it and thereby maintain your desired number of tasks in the service.

- The *container agent* runs on each container instance within an Amazon ECS cluster. The agent sends information about the current running tasks and resource utilization of your containers to Amazon ECS.

## ECS Auto-scaling

- *Automatic scaling* is the ability to increase or decrease the desired count of tasks in your Amazon ECS service automatically.




- Amazon ECS publishes CloudWatch metrics with your service's average CPU and memory usage. You can use these and other CloudWatch metrics to scale out your service (add more tasks) to deal with high demand at peak times, and to scale in your service (run fewer tasks) to reduce costs during periods of low utilization.

- Amazon ECS Service Auto Scaling supports the following types of automatic scaling:

    - Target tracking scaling policies— Increase or decrease the number of tasks that your service runs based on a target value for a specific metric. This is similar to the way that your thermostat maintains the temperature of your home. You select temperature and the thermostat does the rest.

- Step scaling policies— Increase or decrease the number of tasks that your service runs based on a set of scaling adjustments, known as step adjustments, that vary based on the size of the alarm breach.
- Scheduled Scaling—Increase or decrease the number of tasks that your service runs based on the date and time.

## ECS Rolling-Update

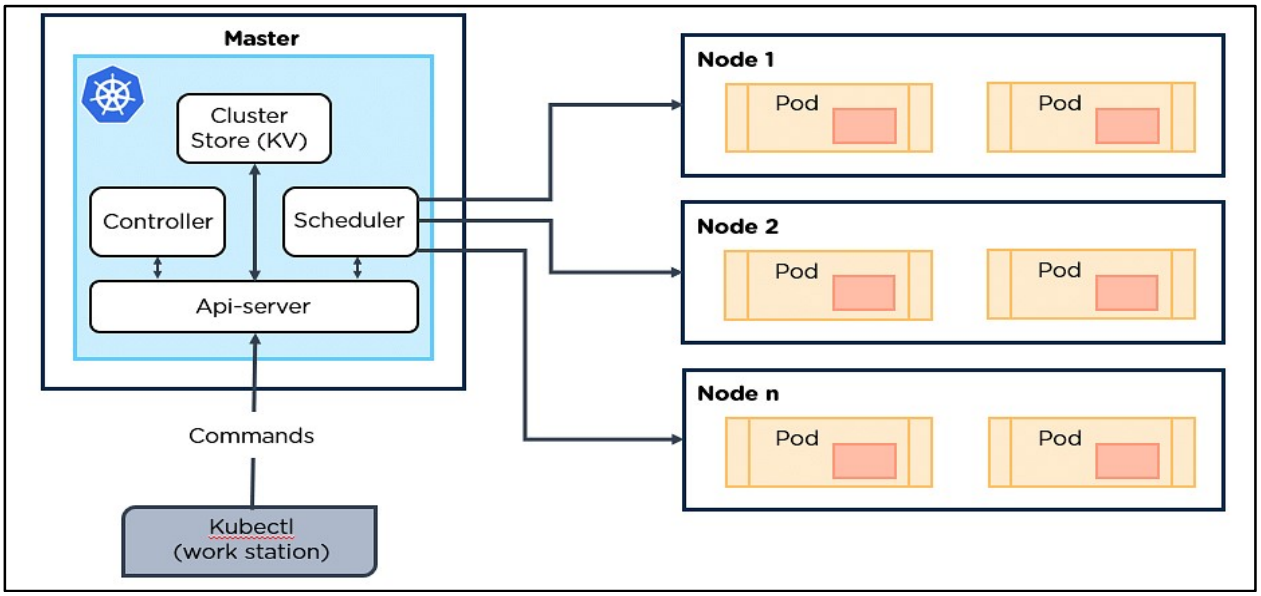
- When you start a service which uses the *rolling update* (ECS) deployment type, the Amazon ECS service scheduler replaces the currently running tasks with new tasks.
- The number of tasks that Amazon ECS adds or removes from the service during a rolling update is controlled by the deployment configuration.
- The deployment configuration consists of the following values which are defined when the service is created, but can also be updated on an existing service.
  - minimumHealthyPercent and
  - maximumPercent
- The minimumHealthyPercent represents the lower limit on the number of tasks that should be running for a service during a deployment or when a container instance is draining, as a percent of the desired number of tasks for the service. This value is rounded up. For example if the minimum healthy percent is 50 and the desired task count is four, then the scheduler can stop two existing tasks before starting two new tasks. Likewise, if the minimum healthy percent is 75% and the desired task count is two, then the scheduler can't stop any tasks due to the resulting value also being two.
- The maximumPercent represents the upper limit on the number of tasks that should be running for a service during a deployment or when a container instance is draining, as a percent of the desired number of tasks for a service. This value is rounded down. For example if the maximum percent is 200 and the desired task count is four then the scheduler can start four new tasks before stopping four existing tasks.

## Kubernetes

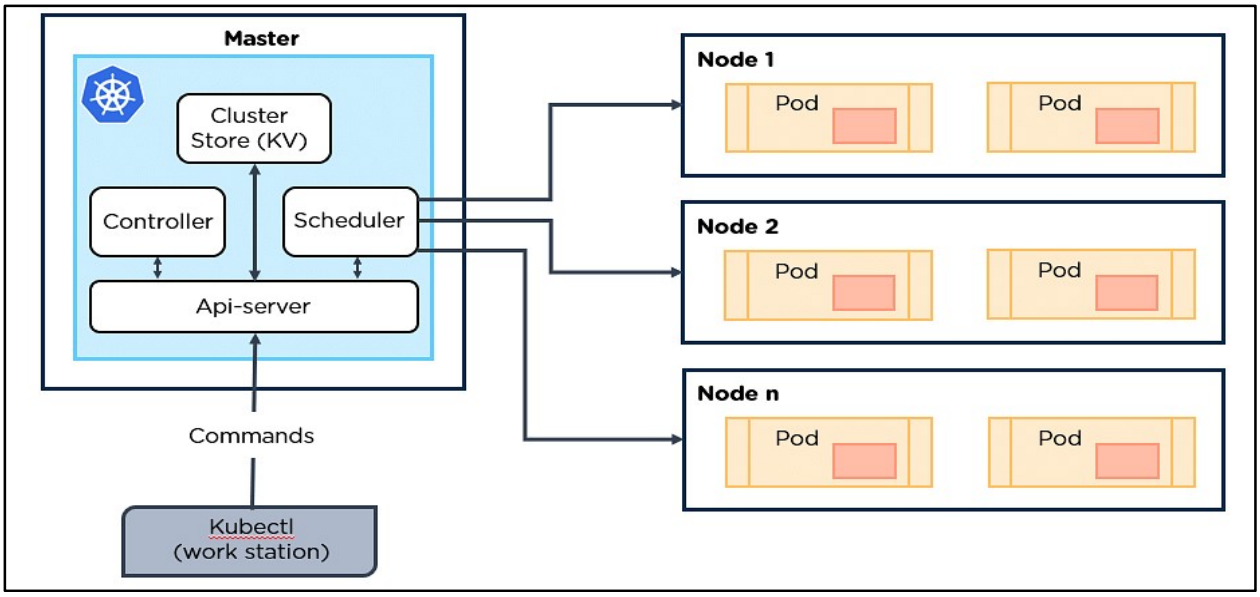
- Kubernetes is an open-source container management platform that you can use to deploy and manage containerized applications at scale.
- Kubernetes manages clusters of EC2 instances, and it runs containers on those instances with processes for deployment, maintenance, and scaling.
- By using Kubernetes, you can run any type of containerized application by using the same toolset on premises and in the cloud.
- You can use AWS services to run Kubernetes in the cloud. These services include scalable and highly available VM infrastructure, community created service integrations, and Amazon Elastic Kubernetes Service (Amazon EKS).
- Kubernetes works by managing a cluster of compute instances and scheduling containers to run on the cluster based on the available compute resources and the resource requirements of each container.
- Containers are run in logical groupings called pods and you can run and scale one or many containers together as a pod.

## Kubernetes Architecture/Components of Kubernetes

- Kubernetes has two nodes—Master Node and Server Node.




- Master Node
  - The master node is the most vital component of Kubernetes architecture.
  - It is the entry point of all administrative tasks.
  - The master node has various components, such as:
    - ETCD
    - Controller Manager
    - Scheduler
    - API Server
    - Kubectl




- Cluster Store / ETCD: This component stores the configuration details and essential values. It communicates with all other components to receive the commands to perform an action.
- Controller Manager: A daemon (server) that runs in a continuous loop and is responsible for gathering information and sending it to the API Server.
- Scheduler: The scheduler assigns the tasks to the slave nodes. It is responsible for distributing the workload and stores resource usage information on every node.
- API Server: Kubernetes uses the API server to perform all operations on the cluster. It is a central management entity that receives all REST requests for modifications, serving as a frontend to the cluster.
- Kubectl: Kubectl controls the Kubernetes cluster manager by issuing commands.
  Syntax - kubectl [flags]

- Slave Node
  - The slave node has the following components:
    - Pod: A pod is one or more containers controlled as a single application.
    - Docker: One of the basic requirements of nodes is Docker. It helps run the applications in an isolated, but lightweight operating environment.
    - Kubelet: Service responsible for conveying information to and from to the control plane service. It gets the configuration of a pod from the API server and ensures that the containers are working efficiently.
    - Kubernetes Proxy: Acts as a load balancer and network proxy to perform service on a single worker node. Manages pods on nodes, volumes, secrets, the creation of new containers, health check-ups, etc.

## Kubectl Commands

- **Kubectl get:** Use **get** to pull a list of resources you have currently on your cluster.

- **Kubectl create:** With kubectl create, you can create nearly any type of resource in a cluster.

- **Kubectl edit:** You can edit any resource in your cluster when you run this command.

- **Kubectl delete: T**o delete the whole resource. Once the resource is deleted, there's no recovering it; you will have to recreate it.

- **Kubectl apply:** The **apply** command allows you to apply configurations via files for resources within your cluster.

- **Kubectl describe: Describe** shows the details of the resource you're looking at.

- **Kubectl logs:** While the **describe** command gives you the events occurring for the applications inside a pod, **logs** offer detailed insights into what's happening inside Kubernetes in relation to the pod.

- **Kubectl exec:** Much like the **docker exec** command, you can also **exec** into a container to troubleshoot an application directly.

- **Kubectl cp:** This command is for copying files and directories to and from containers, much like the Linux **cp** command.

## AWS Elastic Kubernetes Service (EKS)

mazon Elastic Kubernetes Service (Amazon EKS) is a managed service that you can use to run Kubernetes on AWS without needing to install, operate, and maintain your own Kubernetes control plane or nodes. Kubernetes is an open-source system for automating the deployment, scaling, and management of containerized applications. Amazon EKS:

- Runs and scales the Kubernetes control plane across multiple AWS Availability Zones to ensure high availability.
- Automatically scales control plane instances based on load, detects and replaces unhealthy control plane instances, and it provides automated version updates and patching for them.
- Is integrated with many AWS services to provide scalability and security for your applications, including the following capabilities:
    - Amazon ECR for container images
    - Elastic Load Balancing for load distribution
    - IAM for authentication
    - Amazon VPC for isolation
- Runs up-to-date versions of the open-source Kubernetes software, so you can use all of the existing plugins and tooling from the Kubernetes community.