**Docker and Kubernetes – The Big Picture:-**

Docker is an open source platform that enables developers to build, deploy, and run applications in a consistent way across different environments. Docker containers are lightweight, standalone, and executable packages of software that include everything needed to run an application: code, runtime, system tools, system libraries, and settings.

**Containers:-**

Docker containers are lightweight, standalone, executable packages of software that include everything needed to run an application: code, runtime, system tools, system libraries and settings. Docker containers are isolated from each other, so they can run on the same host without interfering with each other. This makes it easy to deploy and manage multiple applications on the same server. Docker containers are also portable, so they can be easily moved from one environment to another.

**Benefits:-**

* **Ease of development:** Docker makes it easy to build and deploy applications by providing a consistent environment across different development and production environments.
* **Improved efficiency:** Docker can help organizations improve the efficiency of their IT infrastructure by reducing the need to manage and maintain multiple virtual machines.
* **Increased scalability:** Docker can help organizations increase the scalability of their applications by making it easy to deploy and manage multiple containers on the same server.

**Commands:-**

* **docker run:** This command creates and starts a new container from an image
* **docker stop:** This command stops a running container
* **docker start:** This command starts a stopped container
* **docker remove:** This command removes a stopped container
* **docker ps:** This command lists all running containers
* **docker images:** This command lists all available images

**Create a Docker Image From a Container:-**

**Step 1:** Create a Base Container

Let’s get started by creating a running container. So that we don’t get bogged down in the details of any particular container, we can use nginx.

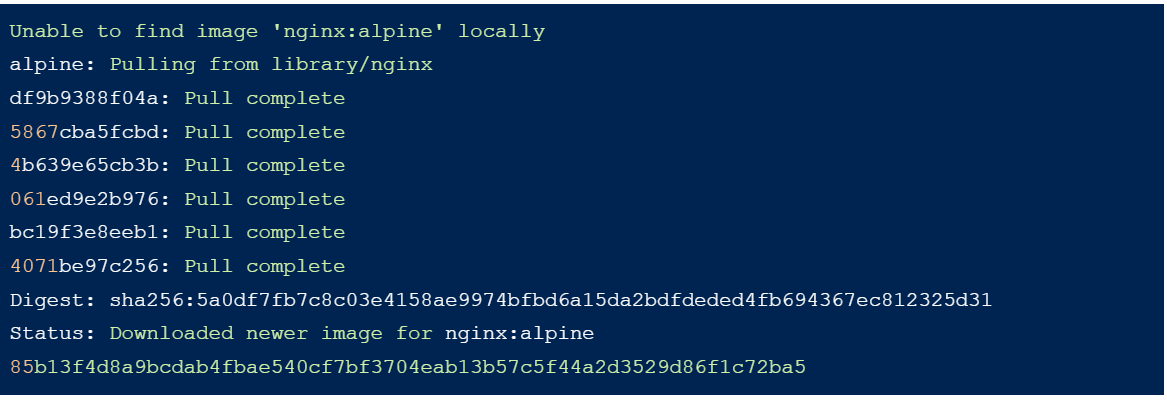
The Docker create command will create a new container for us from the command line:

**Command:**

~ docker create --name nginx\_base -p 80:80 nginx:alpine

Here we have requested a new container named nginx\_base with port 80 exposed to localhost. We are using nginx:alpine as a base image for the container.

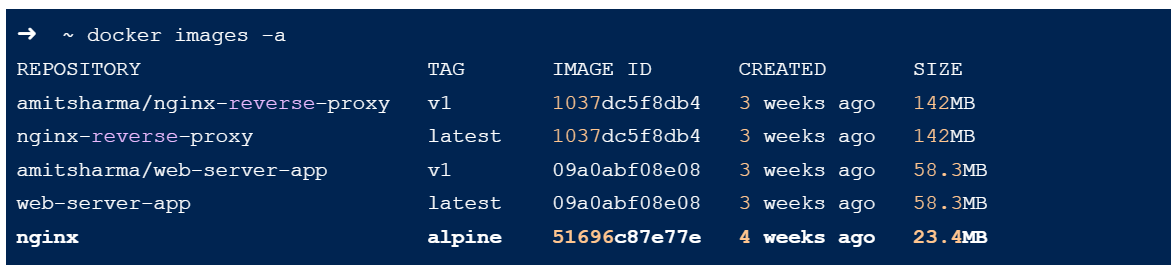
If you don’t have the nginx:alpine image in your local docker image repository, it will download automatically.



**Step 2:** Inspect Images

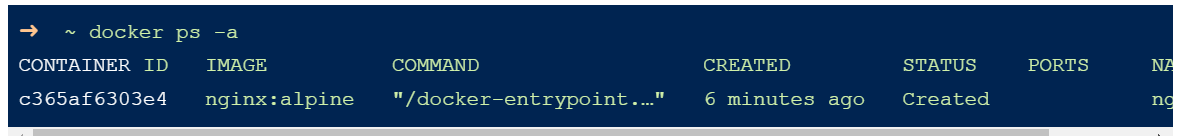
If you look at the list of images on your system, you will now see the nginx:alpine image:

**Command:** docker images –a



**Step 3:** Inspect Containers

Note here that the container is not running, so you won’t see it in the container list unless you use the -a flag (-a is for all)



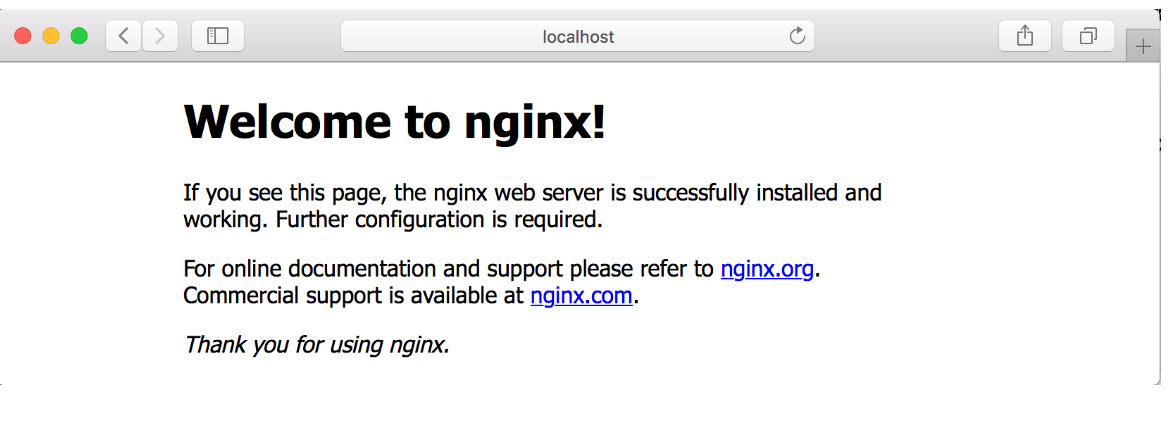
**Command:** docker ps -a

**Step 4:** Start the Container

Let’s Start the Container



**Command:** docker start nginx\_base

**Then, **

**Step 5:** Modify the Running Container

if you wanted to modify this running container so that it behaves in a specific way, there are a variety of ways to do that.

In order to keep things as simple as possible, we are just going to copy a new index.html file onto the server. You could do practically anything you wanted here.

Let’s create a new index.html file and copy it onto the running container. Using an editor on your machine, create an index.html file in the same directory that you have been running Docker commands from.

Then paste the following HTML into it:

****Then save the file and return to the command line. We will use the docker cp command to copy this file onto the running container.

**Command:-**

➜ ~ docker cp index.html nginx\_base:/usr/share/nginx/html/index.html

**Step 6:** Create an Image From a Container

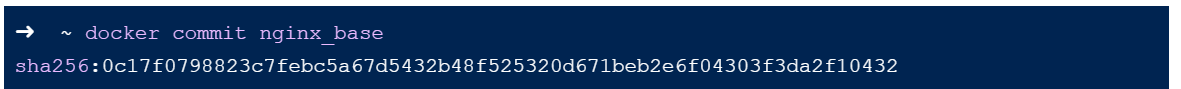
So at this point, we’ve updated the contents of a running container and as long as we keep that container around, we don’t need to do anything.

However, we want to know how to save this container as an image so we can make other containers based on this one. The Docker commands to do this are quite simple.

To save a Docker container, we just need to use the docker commit command like this:

**Command:-**

**~ docker commit nginx\_base**

****

**Docker Images List:-**

****

**Step 7:** Tag the Image

Using docker tag, we can name the image we just created. We need the image ID for the command, so given that the image ID listed above is 0c17f0798823, our command will be:

**➜ ~ docker tag 0c17f0798823 hello\_world\_nginx**

**Step 8:** Create Images With Tags

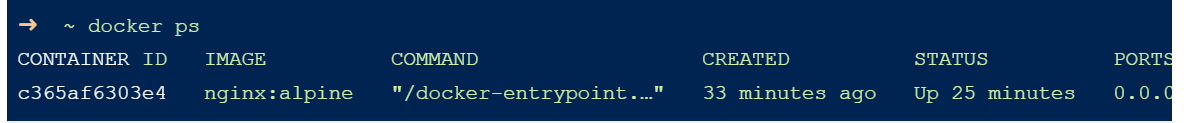
Also tag the image as it is created by adding another argument to the end of the command like this:

**Command:-**

**➜ ~ docker commit nginx\_base hello\_world\_nginx**

**Step 9:** Delete the Original Container

Earlier we started a Docker container. We can see that it is still running using the docker ps command.



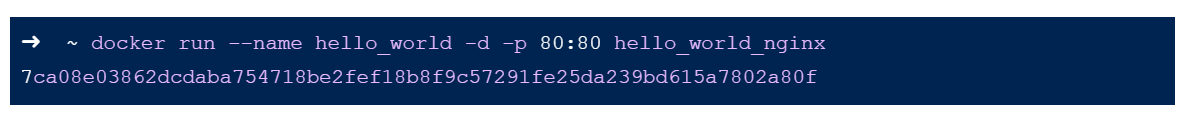
Stop and remove the Docker container that is currently running and delete it



If we list all of the Docker containers, we should have none



let’s create a new container based on the image we just created and start it

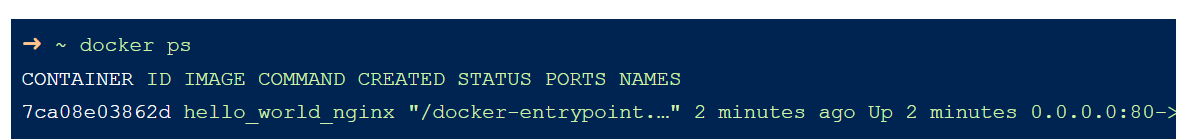


**Step 10:** Look at Running Containers

If you look at the running containers now, you will see we have one called hello\_world

**Command:-**

docker ps



Stop the container hello\_world before moving on to the next section



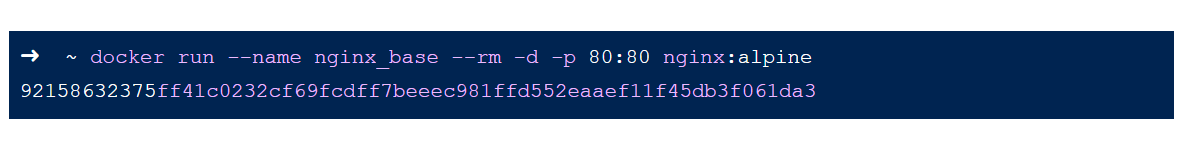
**Step 11:** Consider Your Options

There are a few optional things we can do using the commit command that will change information about our images.

For example, we might want to record who the author of our image is or capture a commit message telling us about the state of the image.

These are all controlled through optional parameters to the commit command.

Let’s go back to our original running container. We are going to use a slightly different command here to make cleanup easier:



**Docker Hub:-**

Docker Hub is a cloud-based registry that hosts Docker images. It is the world's largest repository of container images, with over 100 million images available. Docker Hub provides a number of features that make it easy to manage and share container images, including:

* **Public repositories:** Anyone can create and publish public images to Docker Hub. This makes it easy to find and use pre-built images for a wide variety of applications.
* **Private repositories:** Organizations can create private repositories to store and share images that are not intended for public use. This can be useful for storing proprietary software or for sharing images within a team.
* **Automated builds:** Docker Hub can be used to automate the build and deployment of container images. This can be done by using Dockerfiles, which are text files that describe the steps required to build an image.
* **API:** Docker Hub provides a RESTful API that can be used to programmatically manage images. This can be useful for automating tasks such as building, pushing, and pulling images.

**Benefits:-**

* **Ease of use:** Docker Hub is a very easy-to-use platform. You can easily find and download the images you need, and you can also easily create and share your own images.
* **Large repository:** Docker Hub has a large repository of images, so you can find the images you need for almost any application.
* **Security:** Docker Hub uses a number of security features to protect your images, including image signing and vulnerability scanning.

**Why Use:-**

* It's the largest repository of container images. Docker Hub has over 100 million container images, including official images from Docker, open source projects, and independent software vendors. This makes it a great place to find the images you need to build your applications.
* It's easy to use. Docker Hub is a web-based service, so you can easily browse and search for images, and then pull them down to your local machine. You can also use Docker Hub to push your own images, so that others can use them.
* It's secure. Docker Hub uses a variety of security measures to protect your images, including authentication, encryption, and vulnerability scanning.
* It's free. You can use Docker Hub for free for personal use. If you need to use Docker Hub for commercial purposes, there are a variety of subscription plans available.

**Difference Between GitHub and DockerHub:-**

**GitHub:-**

**Version Control:** GitHub is primarily a web-based hosting service for version control repositories. It allows developers to collaborate on projects, track changes, manage branches, and merge code changes. It supports Git, a distributed version control system.

**Source Code Hosting:** GitHub hosts source code repositories, which can be public or private. Developers can push code changes, create branches, open pull requests, and review and discuss code changes with other team members.

**Issue Tracking:** GitHub provides built-in issue tracking features, allowing users to create, assign, and track issues related to a project. Issues can be used for bug tracking, feature requests, or general project management.

**Collaboration and Community:** GitHub provides a platform for collaboration among developers, allowing them to contribute to open-source projects, share code, and participate in discussions. It offers features such as pull requests, code reviews, and project wikis.

**Continuous Integration/Deployment (CI/CD) Integration:** GitHub integrates with various CI/CD tools like GitHub Actions, Travis CI, CircleCI, and Jenkins, enabling developers to automate build, test, and deployment processes.

**Docker Hub:-**

**Image Hosting and Distribution:** Docker Hub is a cloud-based registry for storing and sharing Docker images. It allows developers to publish their Docker images, making them available for others to download and use.

**Docker Image Repository:** Docker Hub provides a centralized repository where developers can search for and download pre-built Docker images. It hosts images for various software applications, frameworks, and services.

**Collaboration and Image Sharing:** Docker Hub enables users to share their Docker images with others, making it easier to distribute applications or services as containers. It supports versioning, allowing multiple versions of an image to coexist.

**Automated Builds:** Docker Hub integrates with source code repositories (like GitHub) and provides automated build features. It can automatically build and update Docker images whenever changes are pushed to the associated repository.

**Official Images:** Docker Hub hosts "official" images, which are curated and maintained by the Docker team or official software vendors. These images are considered reliable and are often used as base images for other custom images.

**Cluster in VM Ware:-**

A VMScluster, originally known as a VAXcluster, is a computer cluster involving a group of computers running the OpenVMS operating system.

**Docker VMs:-**

Docker VMs have a number of warts, or drawbacks that can make them less than ideal for some use cases. These include:

**Performance overhead:** Docker VMs are typically slower than native applications, due to the overhead of running a virtual machine. This can be a significant issue for applications that require high performance.

**Resource usage:** Docker VMs can use more resources than native applications, due to the overhead of running a virtual machine. This can be a problem for environments with limited resources.

**Complexity:** Docker VMs can be more complex to manage than native applications. This is due to the fact that they are virtual machines, and as such, require additional configuration and management.

**Security:** Docker VMs can be less secure than native applications. This is due to the fact that they are virtual machines, and as such, are more vulnerable to attack.

**Docker VMs Container:-**

Docker containers are a popular way to package and run applications. They offer a number of advantages over traditional VMs, such as being more lightweight and portable. However, there are also some disadvantages to using Docker containers, which are known as "warts."

One of the biggest warts of Docker containers is that they can be difficult to debug. This is because each container is isolated from the host system, which makes it difficult to trace the source of problems.

Another wart of Docker containers is that they can be slow to start up. This is because each container needs to boot its own operating system, which can take a few seconds.

Finally, Docker containers can be difficult to secure. This is because they are running on the host system, which means that they are vulnerable to attacks on the host system.

Despite these warts, Docker containers are still a popular choice for packaging and running applications. They offer a number of advantages over traditional VMs, and the disadvantages can be mitigated with careful planning and execution.

**Mitigating the warts of Docker containers:-**

* Use a well-defined development process. This will help to ensure that applications are developed in a way that is compatible with Docker containers.
* Use a good logging system. This will help to troubleshoot problems when they occur.
* Use a good monitoring system. This will help to identify problems early on.
* Use a security scanner. This will help to identify security vulnerabilities in your applications.

**Deploying a Containerized App:-**

1. Create a Dockerfile that defines the contents of the container.
2. Build the Docker image from the Dockerfile.
3. Push the Docker image to a registry.
4. Create a deployment manifest that defines how the container should be deployed.
5. Deploy the manifest to a Kubernetes cluster.

**Wrap Speed:-**

The wrap speed for deploying a containerized app in Docker can vary depending on a number of factors, including:

* The size of the Docker image
* The number of replicas that are being deployed
* The load on the Kubernetes cluster

In general, the wrap speed for deploying a containerized app in Docker is much faster than deploying a traditional application. This is because Docker images are much smaller than traditional application packages, and because Docker containers can be started and stopped very quickly.

**Hosting on a Repository:-**

**Docker Hub:** Docker Hub is a public registry that is owned and operated by Docker. It is the most popular option for hosting Docker images.

**Quay.io:** Quay.io is a public registry that is owned and operated by Red Hat. It offers a number of features that are not available on Docker Hub, such as the ability to create private repositories.

**Amazon ECR:** Amazon ECR is a private registry that is hosted by Amazon Web Services. It is a good option for organizations that use Amazon Web Services for their infrastructure.

**Azure Container Registry:** Azure Container Registry is a private registry that is hosted by Microsoft Azure. It is a good option for organizations that use Microsoft Azure for their infrastructure.

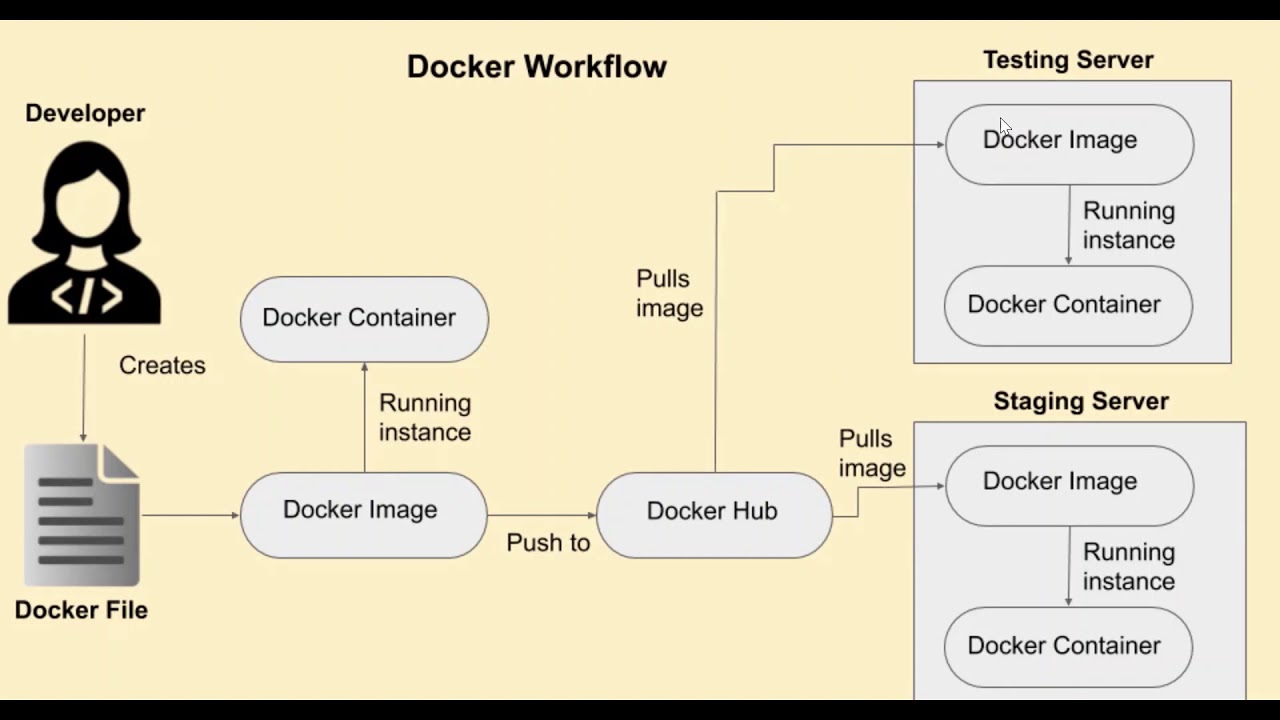
**Running and Managing Containerized in Docker:-**

**Docker CLI:** The Docker CLI is the most basic way to run and manage containerized applications. You can use the CLI to create, start, stop, and delete containers. You can also use the CLI to manage the networking and storage for your containers.

**Docker Compose:** Docker Compose is a tool that makes it easier to define and run multi-container applications. You can use Docker Compose to define a YAML file that specifies the services that make up your application. Docker Compose will then create and start the containers for you.

**Docker Swarm:** Docker Swarm is a native clustering solution for Docker. You can use Docker Swarm to create a cluster of Docker hosts and then deploy your containerized applications to the cluster.

**WorkFlow of Docker:-**



**Kubernates:-**

Kubernetes is an open-source system for automating deployment, scaling, and management of containerized applications. Kubernetes is a popular platform for running containerized applications at scale. It provides a number of features that make it easy to deploy and manage containerized applications, including:

**Service discovery and load balancing:** Kubernetes can automatically discover and load balance containers that are running on a cluster. This makes it easy to scale your application up or down as needed.

**Replication:** Kubernetes can automatically replicate your application's containers across multiple nodes in a cluster. This ensures that your application is always available, even if one or more nodes fail.

**Autoscaling:** Kubernetes can automatically scale your application up or down based on demand. This helps you to save money on resources when your application is not under heavy load.

**Health monitoring:** Kubernetes can monitor the health of your application's containers and automatically restart them if they fail. This helps to ensure that your application is always available.

**Logging and tracing:** Kubernetes can collect logs and traces from your application's containers. This helps you to troubleshoot problems and identify performance bottlenecks.

Kubernetes can be used to run a wide variety of software applications, including:

* Web applications
* Microservices
* Batch jobs
* Data processing pipelines
* Real-time applications

Kubernetes is a powerful tool that can help you to deploy and manage containerized applications at scale. If you are looking for a platform to run your containerized applications, Kubernetes is a great option.

* **Docker:** Docker is a popular containerization platform that is often used with Kubernetes.
* **Helm:** Helm is a package manager for Kubernetes that makes it easy to install and manage Kubernetes applications.
* **Prometheus:** Prometheus is a monitoring system that can be used to collect metrics from Kubernetes applications.
* **Grafana:** Grafana is a visualization tool that can be used to display metrics from Prometheus.
* **Elasticsearch:** Elasticsearch is a search engine that can be used to store and search logs from Kubernetes applications.

Kubernetes can be used with a variety of cloud providers, including:

* Amazon Web Services (AWS)
* Microsoft Azure
* Google Cloud Platform (GCP)
* IBM Cloud
* Oracle Cloud
* Alibaba Cloud
* DigitalOcean
* Vultr
* Linode

Each cloud provider offers its own Kubernetes service, which provides a managed Kubernetes environment. This means that the cloud provider takes care of the underlying infrastructure, such as nodes and storage, so that you can focus on deploying and managing your applications.

**Cloud Providers:-**

**Amazon Elastic Kubernetes Service (EKS):** EKS is a fully-managed Kubernetes service that makes it easy to deploy, manage, and scale containerized applications on AWS.

**Azure Kubernetes Service (AKS):** AKS is a fully-managed Kubernetes service that makes it easy to deploy, manage, and scale containerized applications on Azure.

**Google Kubernetes Engine (GKE):** GKE is a fully-managed Kubernetes service that makes it easy to deploy, manage, and scale containerized applications on GCP.