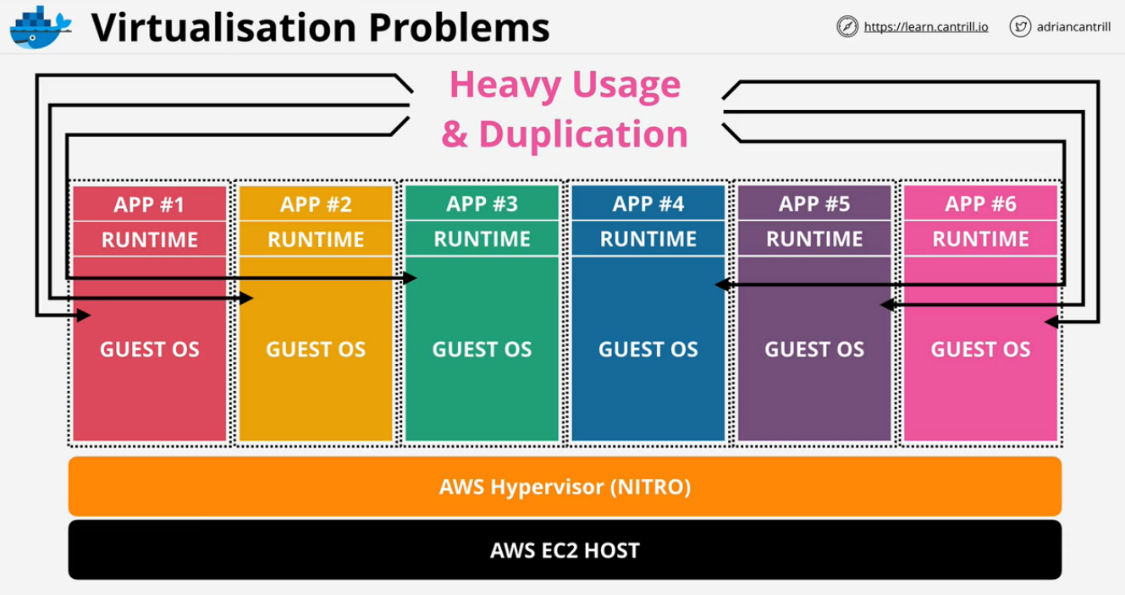
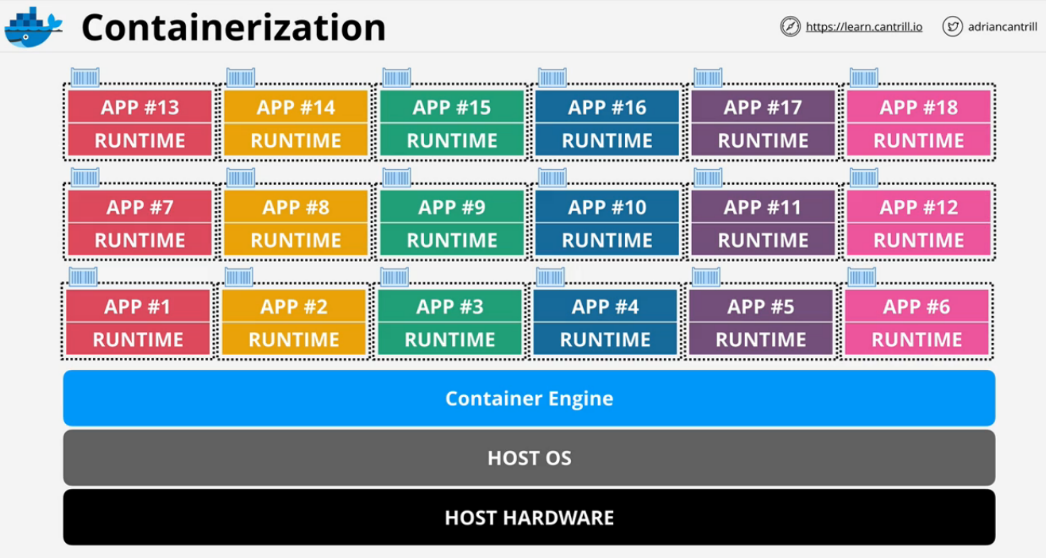
**Containers-and-ECS**

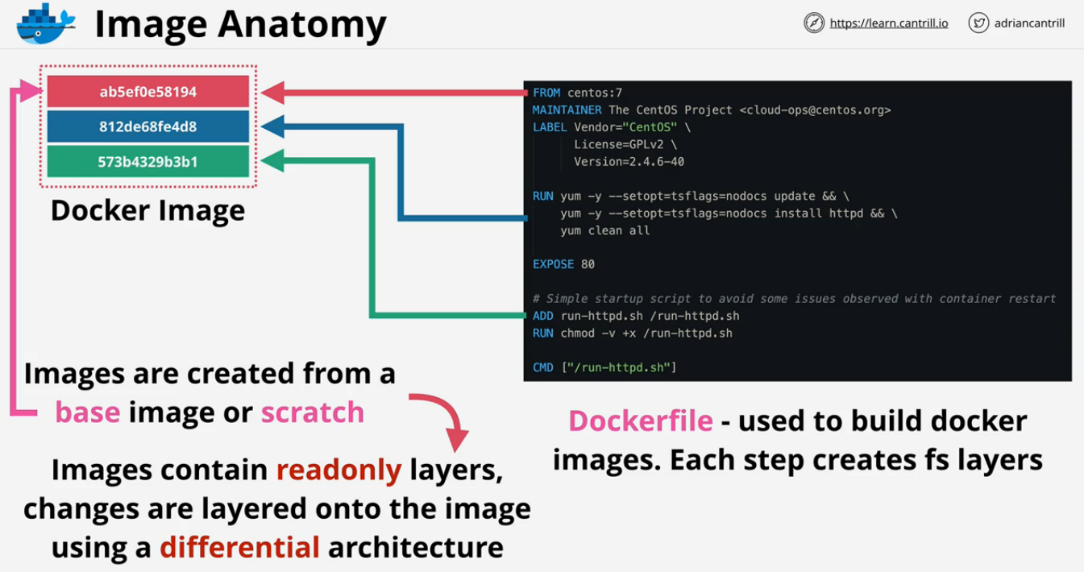
**Intro to Containers**

Virtualization Problems

Using an EC2 virtual machine with Nitro Hypervisor, 4 GB ram, and 40 GB disk, the OS can consume 60-70% of the disk and much of the available memory. Containers leverage the similarities of multiple guest OS by removing duplicate resources. This allows applications to run in their own isolated environments.

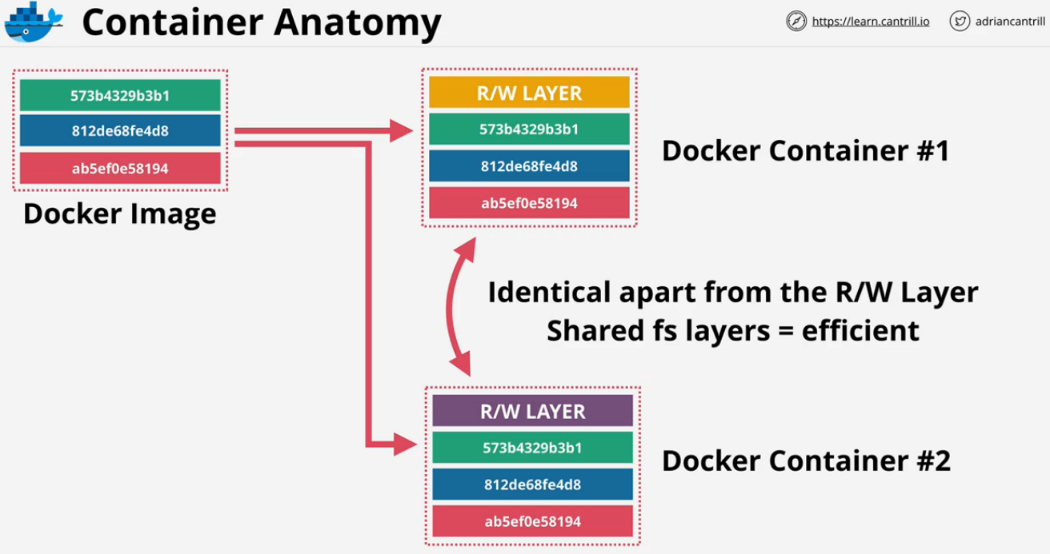


**Image Anatomy**

A Docker image is composed of multiple layers and not a monolithic disk image. Each line of a Docker image creates a new filesystem layer on top of the previous. Images are created from scratch or a base image. Images contain read only layers, images are layer onto images.

**Docker container** is the same as a **Docker image**, except it has an additional READ/WRITE layer of the container.

**Docker container is the running copy of the Docker Image.**

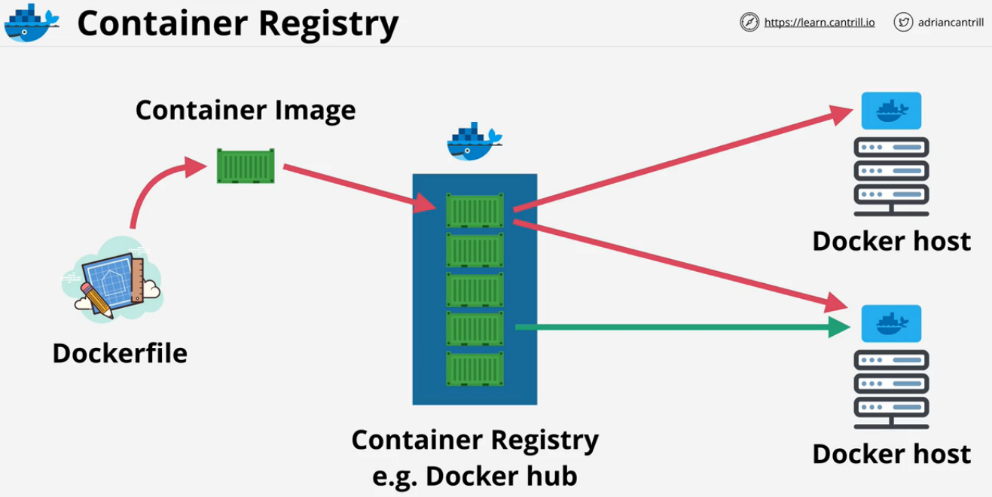
If you have **lots of containers with very similar base structures**, they will share the **parts that overlap**. The other layers are reused between containers.

* **Container** -> Running Copy of the Docker Image. Has an additional Read-Write File System Layer.
* **Docker Image** -> Stacks of the Read only File System Layers.
* **Docker Files**-> Used to build Docker Images. Each Line in a Docker File System Layer creates a new File System Layer.

**Container Registry**

Registry or hub of container images. Dockerfile can create a container image where it gets stored in the container registry.

Docker hosts can run **many containers** based on **one or more images**.

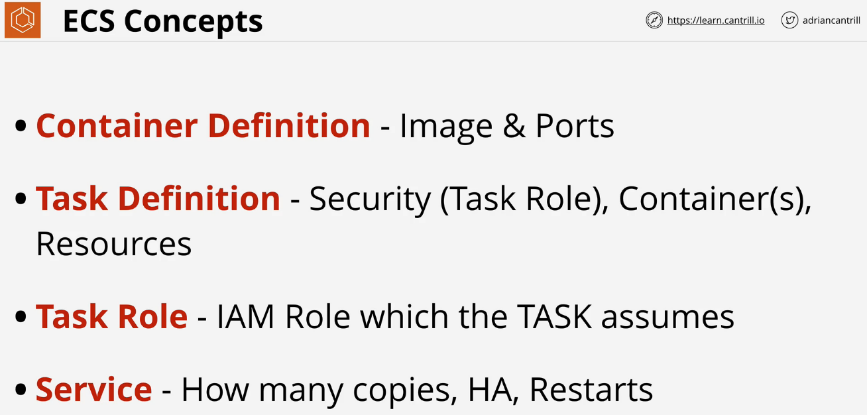
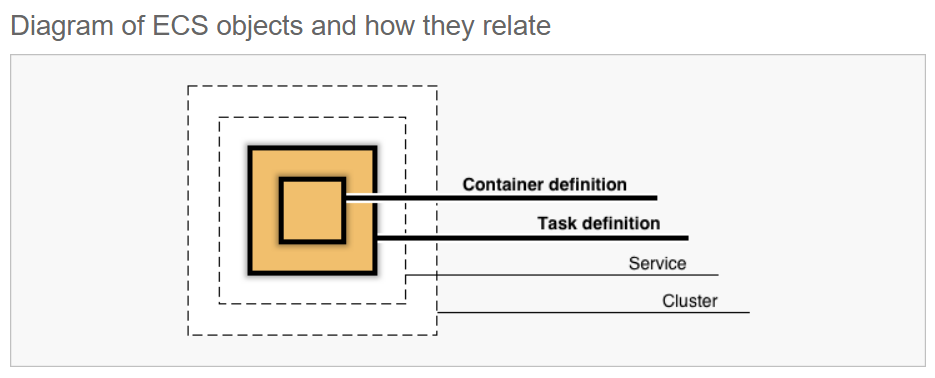
A single image can generate Containers on many different Docker hosts.

**Container Key Concepts**

* **Docker files** are used to **build Docker images**
* **Containers are portable** and **always run as expected**.
  + Anywhere there is a **compatible host, it will run exactly as you intended**.
* Containers are lightweight, use the **host OS for the heavy lifting**.
  + **File system layers are shared** when possible.
* Containers only run the application and environment it needs to run.
* Ports need to be **exposed to allow outside access from the host and beyond**.
* **Application stacks can be multi container**

**Elastic Container Service (ECS) Concepts**

* Accepts containers and instructions you provide.
* ECS allows you to **create a cluster**.
  + Clusters are **where containers run from**.
* **Container images** will be located on a registry.
  + AWS provides **ECR (elastic container registry)**
  + Dockerhub can be used as well.
* **Container definition** gives ECS just enough info about the single container.
  + **A pointer to which image to use** and **the ports that will be exposed**.
* **Task definitions** store the resources used by the task(For example **CPU & Memory, Networking mode, Compatibility, Task Role**). They apply to one or multiple containers.
  + Stores **task role**, an IAM role that allows the task access to other AWS resources.
* **Task contain Containers.** In AWS it represents a self-contained application. It can have **one or multiple containers**(May be a database container and an application container). It represents the application as a whole.
* **Task is not by itself highly available.**

ECS **Service** is configured via **Service Definition** and Service Definition **represents how many copies of a task you want to run for scaling** and HA. This adds capacity resilience. Used in Production environments to cope with increase in load.

**ECS Cluster Types**

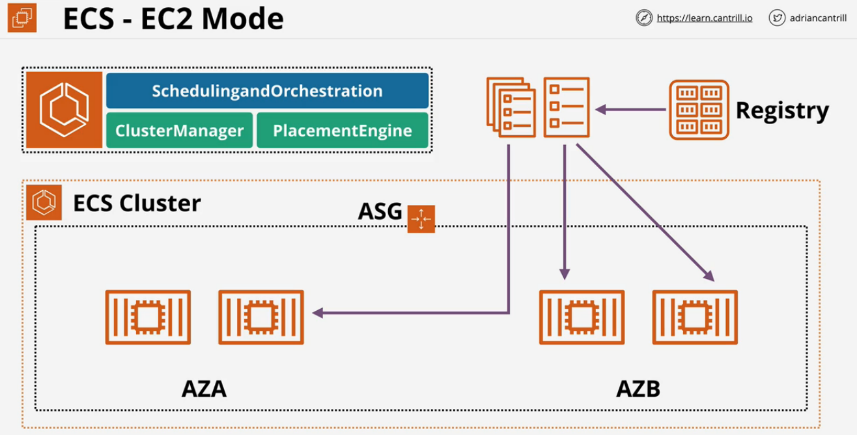
**ECS Cluster manages:**

* Scheduling and Orchestration
* Cluster manager
* Placement engine

**EC2 mode**

ECS cluster is created within a VPC. It benefits from the multiple AZs that are within that VPC. You specify an initial size which will drive an **auto scaling group**.

ECS using EC2 mode is **not a serverless solution**, you **need to worry about capacity for your cluster**.

The **container instances are not delivered as a managed service, they are managed as normal EC2 instances**. You can use spot pricing or prepaid EC2 servers.

**Fargate mode**

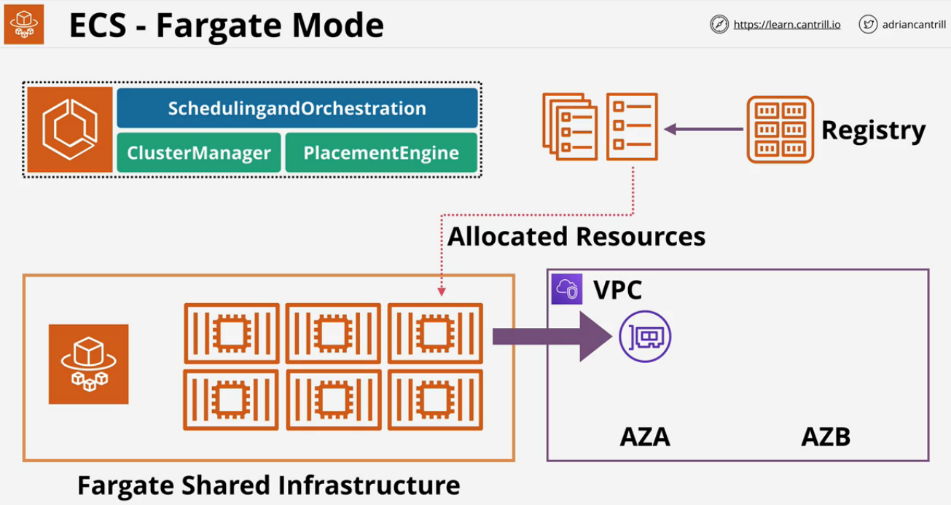
Removes more of the management overhead from ECS, no need to manage EC2.

**Fargate shared infrastructure** allows all customers to access from the same pool of resources.

Fargate deployment still **uses a cluster with a VPC where AZs are specified.**

For **ECS tasks**, they are **injected into the VPC**. Each **task is given an elastic network interface which has an IP address within the VPC**. They then run like a VPC resource.

You only **pay for the container resources** you use.



**EC2 vs ECS(EC2) vs Fargate**

If you already are using containers, use **ECS**.

**EC2 mode** is good for a **large workload if you are price conscious**. This allows for **spot pricing and prepayment**.

**Fargate** is great if you,

* Have a **large workload but are overhead conscious**.
* Have **small or burst style workloads**.
* Use **batch or periodic workloads**.