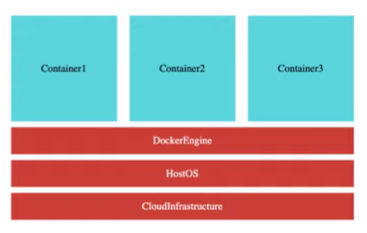
**Microservices**

* Enterprises are moving towards microservices architecture
  + Instead of building large monolithic applications providing various features, build independently deployable, small, focussed “microservices”
  + E.g: A restaurant could have a menu service, a customer feedback service, a billing service, a booking service etc.
  + This gives flexibility to build each feature in a different programming language, though this can make deployment complicated as each language is deployed differently
  + Solution? Containers, given by Docker

**Docker**

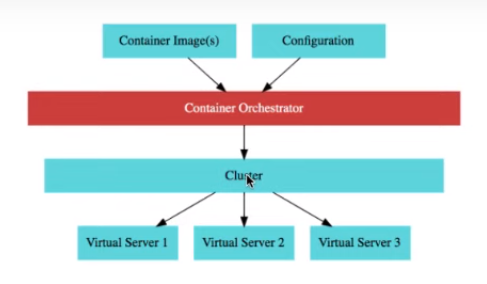
* Allows you to create an “image” for each microservice
* An “image” contains:
  + Application runtime (e.g. JDK, or Python, or NodeJS, etc.)
  + Application code
  + Dependencies
* Docker containers can be run in the same way on any infrastructure, whether it is your local machine, corporate data centre, or a cloud platform



Advantages:

* Containers are light weight as compared to VMs
* Better CPU Utilization
* Docker provides isolation for containers
* It is Cloud Neutral, i.e. it can run on any cloud platform
* Q: How do we manage thousands of containers that deal with multiple microservices? A: Container Orchestration

**Container orchestration**

* 

Our application has multiple containers for their respective microservices, and we have a configuration (e.g. we need X number of instances for Microservice A, Y instances for Microservice B, etc)

The role of the container orchestrator is to make sure that the above needs are met and the application is up and running, and it does that by deploying the application on a cluster of virtual servers (such as EC2 instances)

Features:

* Auto-scaling: Scale the number of containers based on demand
* Service discovery: Help one microservice call another
* Load Balancing: Distribute load among multiple instances of a microservice
* Self healing by performing health checks and replacement of failing instances
* Release new versions without downtime

**Container orchestration options**

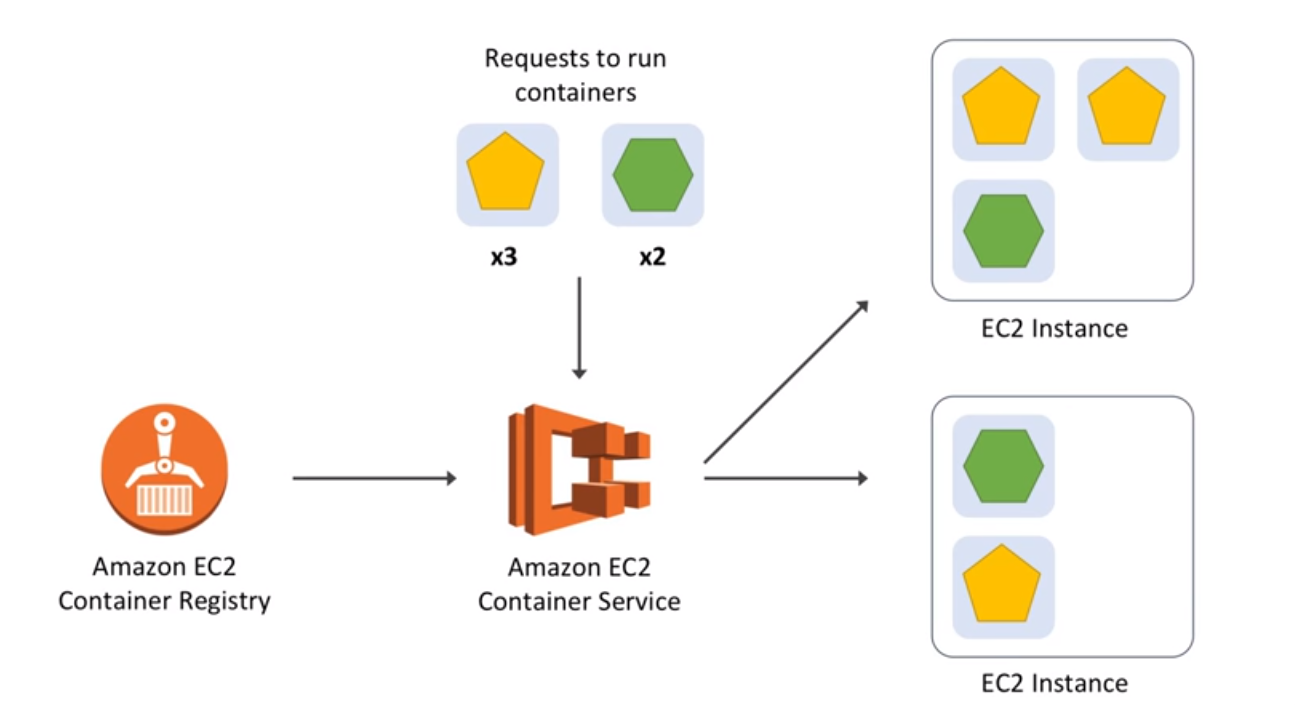
* Cloud Neutral
  + AWS EKS (Elastic Kubernetes Service) [NO FREE-TIER]
* AWS Specific
  + AWS ECS (Elastic Container Service)
  + AWS Fargate (Serverless version of ECS, so no need to worry about managing clusters) [NO FREE-TIER]

**Amazon ECS**

* Fully managed container service for container orchestration
* Can run workloads in batches using AWS Batch

Q: When to choose ECS and when to choose Fargate?

A: ECS when you want to manage clusters (and see what goes on behind in scenes in the EC2 clusters) and Fargate when you do not.



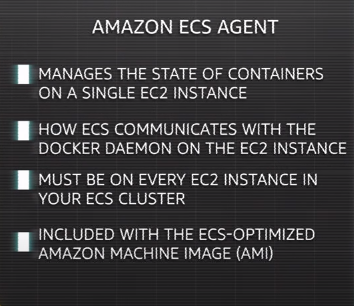
To get started with ECS, we need to create a Cluster

Q: What is an Amazon ECS Cluster?

A: Logical group of EC2 instances that each run Docker and an agent provided by ECS that we can place our container onto. EC2 instances are linked in a VPC

Q: What is ECS Agent?

A:



The cluster can be created using the Wizard in the AWS Management console

Other ways to create a cluster are:

Cloudformation

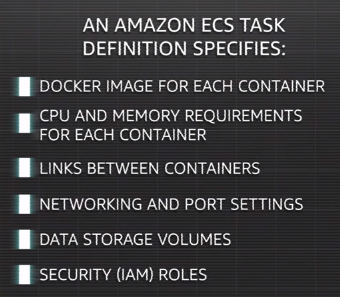
Amazon ECS CLI

ECS APIs

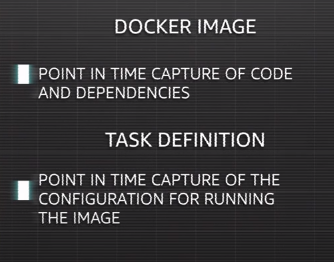
After the cluster is up and running, we need to create a task

Sample task:

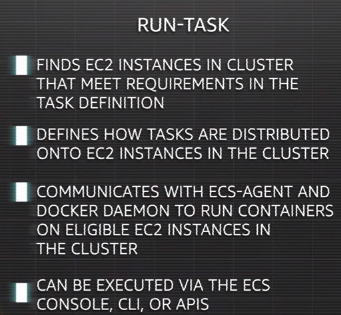




Definitions:



Q: What happens when we run a task on ECS?



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Hands-on: <https://github.com/arindamkeswani/aws/tree/main/DockerXECS>

Follow the steps in the README file