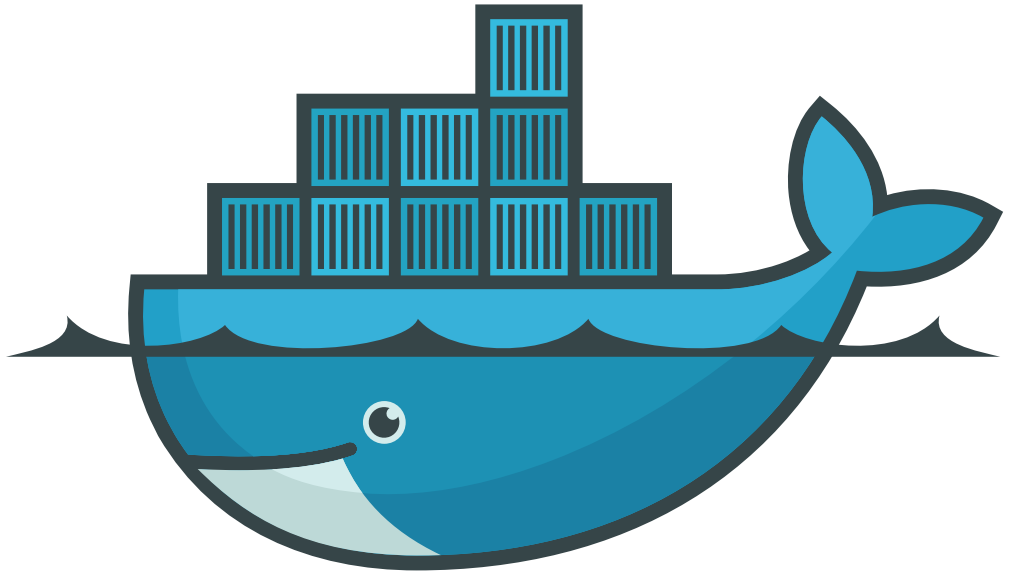
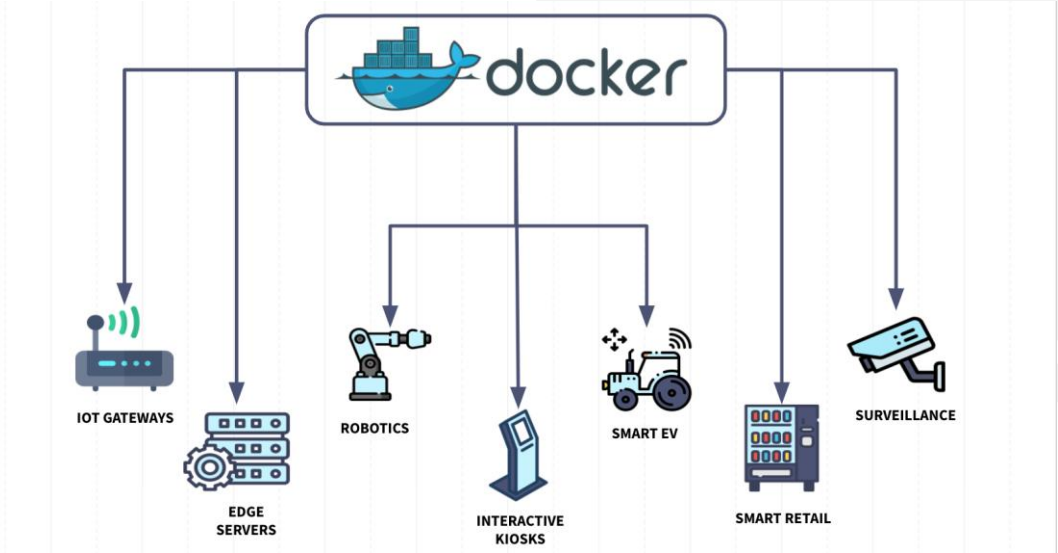


docker

Introduktion



I dagens värld måste team lansera appar snabbt om de ska skapa och behålla affärsmöjligheter.

Detta krav tvingar programvaruutvecklings- och supportteam att alltid titta på lösningar som sparar tid och minskar kostnaderna.

En idealisk lösning minskar den tid som ägnas åt att skapa och konfigurera distributionsmiljöer och förenklar programdistributionsprocessen.

Vad är Docker och varför behövs det?

Docker är en plattform för **containerisering** som gör det enkelt att skapa, leverera och köra applikationer i **containrar**.

Det hjälper utvecklare och driftteam att bygga och hantera applikationer på ett effektivt och konsekvent sätt, oavsett miljö.

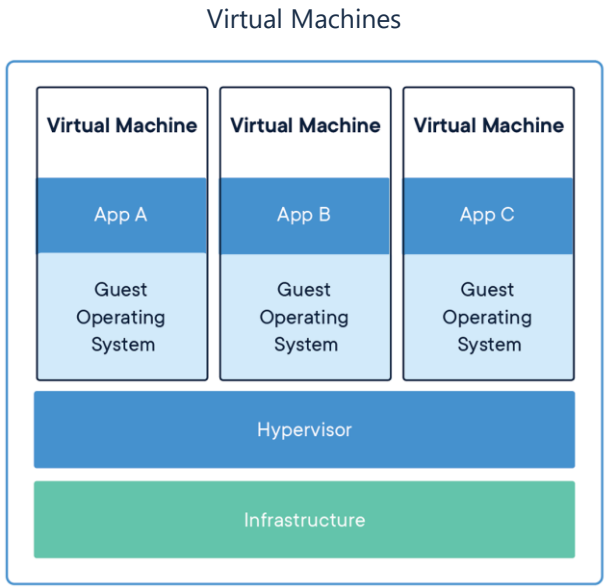
Hanterar värdmiljöer

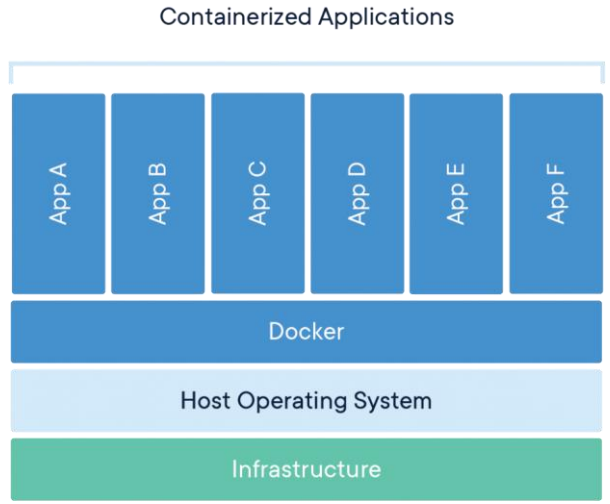
Docker gör det enkelt att hantera värdmiljöer genom att paketera programvara och dess beroenden i containrar.

Dessa containrar körs på samma sätt oavsett vilken värdmiljö de används i – vilket eliminerar skillnader mellan utvecklings-, test- och produktionsmiljöer.

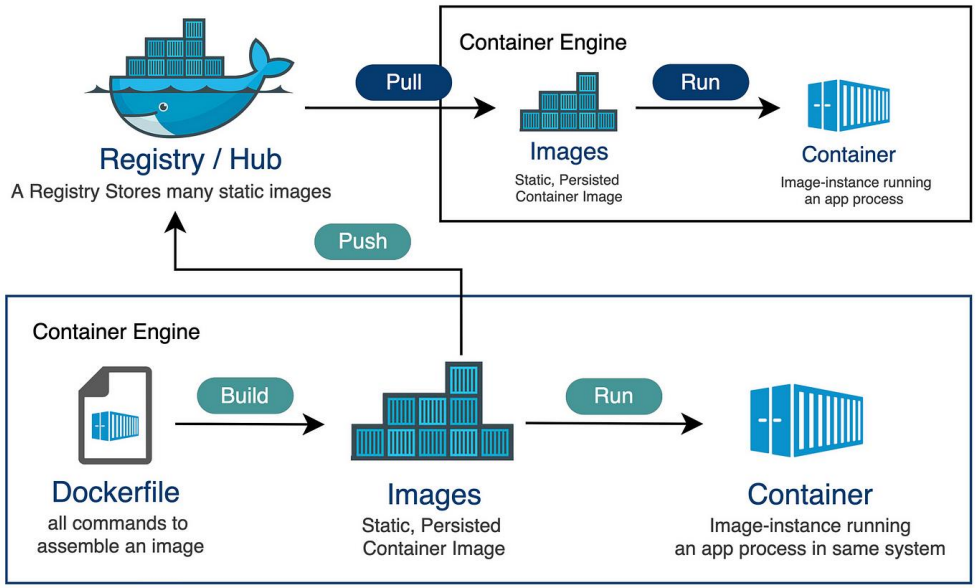
- **Konsistens mellan miljöer ”klon”**
  Alla miljöer behöver ha samma programvara installerad och maskinvara konfigurerad för att undvika problem som "det funkar på min maskin".
- **Komplex konfiguration**
  Vi måste även hantera:
  1. Nätverksåtkomst
  2. Datalagring
  3. Säkerhet

Detta måste göras på ett **konsekvent** och **reproducerbart sätt**, annars blir miljöhanteringen ineffektiv och tidskrävande.





Kontinuitet inom programvaruleverans



När vi distribuerar applikationer till olika miljöer ställs vi inför flera krav:

Kompletta distributionspaket
Varje paket måste inkludera allt som behövs för att programmet ska fungera, såsom:

1. Systempaket (t.ex OS)
2. Binärfiler (exe, dll)
3. Bibliotek (t.ex dotnet framework)
4. Konfigurationsfiler (appsettings.json)

Matchande beroenden
Alla beroenden måste stämma överens med rätt programvaruversioner och arkitektur för att undvika oväntade fel.

Effektiv maskinvaruanvändning

När vi kör flera applikationer på samma server möter vi två viktiga krav:

Isolering mellan applikationer

1. Om en applikation kraschar eller upplever hög belastning, ska detta **inte påverka andra** applikationer som körs på samma server. Genom **isolering** säkerställs att varje applikation körs som om den var på en egen maskin. Det innebär **stabilitet**.

2. För att minimera riskerna för obehörig åtkomst mellan applikationer eller att känslig information delas av misstag, måste varje applikation köras i en **isolerad** miljö vilket ökar **säkerheten**.

Effektiv resursanvändning

1. När vi försöker köra flera applikationer på samma maskinvara vill vi **optimera** användningen av CPU, RAM och lagring. Detta gör det möjligt att minska kostnader och maximera prestanda.

2. Om isoleringen är för rigid (som i fallet med virtuella maskiner) kan resursanvändningen bli ineffektiv, eftersom varje virtuell maskin behöver en fullständig uppsättning operativsystem.

Programportabilitet: Att kunna flytta utan hinder

Programportabilitet handlar om att säkerställa att våra applikationer kan köras i olika miljöer utan anpassningar.

Detta är viktigt av flera anledningar:

Redundans och skalbarhet

1. Om en värdmiljö slutar fungera kan vi behöva flytta applikationen till en annan miljö för att upprätthålla driften.
2. När belastningen ökar kan vi behöva skala ut applikationen genom att köra den på flera servrar.

Flexibilitet vid infrastrukturbyte

1. Programvaran ska kunna flyttas från en värd till en annan, även om den underliggande infrastrukturen skiljer sig åt.

Hur Docker hjälper

Docker gör det möjligt att skapa isolerade containrar som innehåller allt du behöver för din app i ett enda paket och som samtidigt är oberoende av den underliggande infrastrukturen

Detta innebär:

- **Konsistens** i alla miljöer
- **Enklare** hantering av beroenden
- **Ingen risk** för mismatch mellan versioner eller arkitekturer
- **Isolerar** applikationen från andra, som om den kördes på en separat maskin.
- **Delar resurser** (CPU, minne, nätverk) effektivt med andra containrar, vilket maximerar serverns kapacitet.
- Gör distributionen **pålitligare, mer förutsägbar och snabbare** mellan olika servrar eller molnplattformar.
- **Minimal avbrottstid** för kunderna.
- **Standardiserad miljö** som fungerar konsekvent, oavsett var den körs.

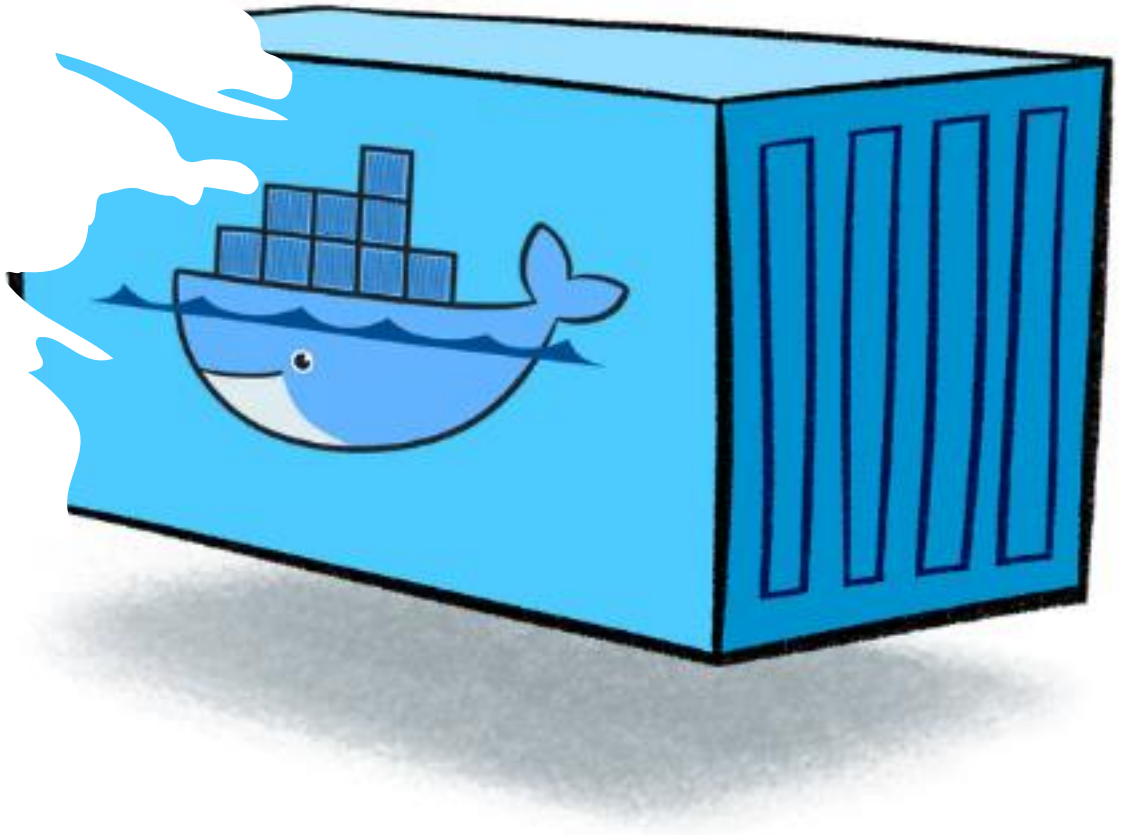
Vad är en container?

En **container** är en lättviktig och isolerad miljö som gör det möjligt att paketera och köra applikationer på ett konsekvent sätt.

Containern inkluderar:

1. Applikationens kod
2. Alla dess beroenden (bibliotek, konfigurationer osv.)

Detta säkerställer att applikationen kan köras snabbt och tillförlitligt i **vilken datormiljö som helst**.



Containeravbildningar (docker images)

Grunden för distribution

En **containeravbildning** är det paket som skapas och innehåller allt som behövs för att köra programmet.

Den blir själva enheten som vi använder för att:

- **Distribuera** applikationer till olika miljöer.
- **Starta** nya instanser av applikationen, oavsett plattform.

En **containeravbildning** är en "blåkopiering (blueprint)" av vad containern ska innehålla och hur den ska bete sig.

Den innehåller allt som behövs för att köra en applikation:

- Källkod eller binärfiler
- Operativsystemsbibliotek och beroenden
- Konfigurationer

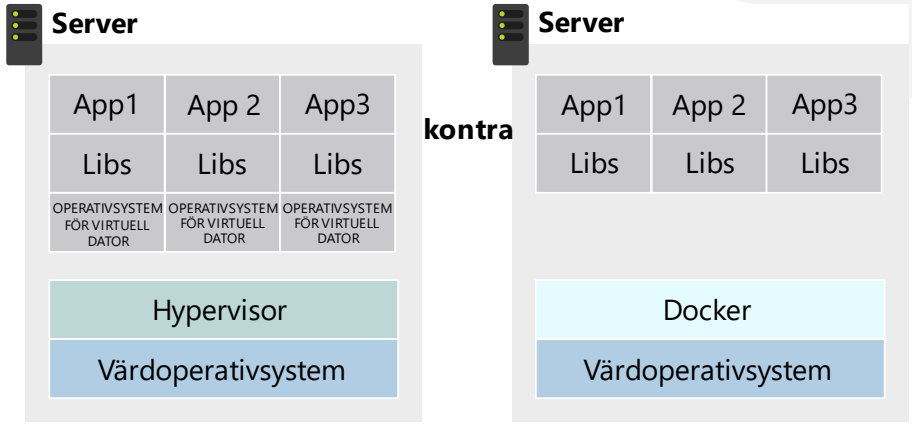
När en container startas baseras den på en avbildning, vilket gör att den alltid körs på ett förutsägbart sätt oavsett miljö.

Exempel:

Dockerfile, Docker Hub Container Registry, Azure Container Registry (ACR) mfl.

Vad är programvarucontainerisering?

Programvarucontainerisering är en metod för att isolera och köra applikationer på ett effektivt sätt genom att använda **operativsystemets virtualisering** i stället för att köra hela operativsystem i en virtuell dator (VM).



Hur fungerar det?

- En **container** är en lättviktig, fristående körmiljö som innehåller applikationens kod, dess beroenden och bibliotek.
- I stället för att virtualisera maskinvaran (som i en VM), virtualiserar containrar **operativsystemet**, vilket gör dem mycket snabbare och resurssnåla.

Var kan containrar köras?

Containrar är flexibla och kan köras:

- På **fysiska maskiner** (on-premises servrar).
- I **molntjänster** (t.ex. AWS, Azure, Google Cloud).
- På **virtuella datorer** (VM:er).
- Över flera operativsystem, så länge de stöder containermotorn (som Docker).

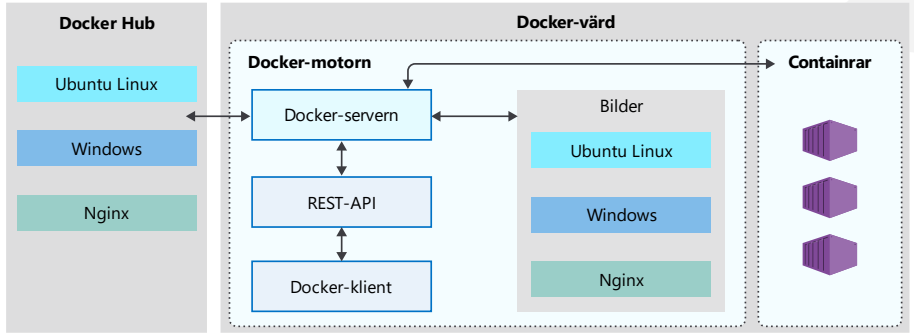
Fördelar med containerisering

- **Snabb starttid**
  Eftersom de inte behöver ladda ett helt operativsystem startar containrar nästan omedelbart.
- **Resurseffektivitet**
  Containrar delar operativsystemets kärna, vilket minskar behovet av extra resurser.
- **Portabilitet**
  Containrar körs konsekvent i vilken miljö som helst – utveckling, test, produktion – utan modifieringar.
- **Isolering**
  Varje container är isolerad från andra, vilket förbättrar säkerhet och stabilitet.

Jämförelse: Container vs. Virtuell dator (VM)

Egenskap	Container	Virtuell dator (VM)
Starttid	Sekunder	Minuter
Resursanvändning	Delar OS-kärnan, låg overhead	Kör fullständigt OS, hög overhead
Isolering	Processnivå	Komplett OS-isolering
Användningsområde	Mikrotjänster, distribuerade system	Komplett systemmiljö, äldre applikationer

Docker-arkitekturen



Docker-plattformen består av flera komponenter som vi använder för att skapa, köra och hantera våra containeriserade program.

Docker-motorn [Docker Engine]

Docker-motorn består av flera komponenter som konfigurerats som en klient-server-implementering där klienten och servern körs samtidigt på samma värd.

Klienten kommunicerar med servern med hjälp av ett REST-API, vilket gör att klienten även kan kommunicera med en fjärrserverinstans.

Vad är Docker-klienten?

Docker-klienten är det gränssnitt vi använder för att interagera med Docker-servern (dockerd). Den fungerar som en bro mellan användaren och Docker-servern och används för att hantera containrar, bilder, nätverk och volymer.

Det finns **två** alternativ för Docker-klienten:

1. Ett kommandoradsprogram med namnet **Docker CLI**.
2. Ett gui-baserat program (Grafiskt användargränssnitt) med namnet **Docker Desktop**.

Både CLI och Docker Desktop interagerar med en Docker-server.
Kommandona docker från CLI eller Docker Desktop använder Docker REST API för att skicka instruktioner till antingen en lokal server eller fjärrserver och fungera som det primära gränssnitt som vi använder för att hantera våra containrar.

Docker-servern

Docker-servern, även kallad **Docker Daemon**, är en bakgrundsprocess som hanterar all funktionalitet i Docker.

Den körs som en tjänst och ansvarar för att:

1. **Ta emot och hantera klientförfrågningar**
   Docker Daemon lyssnar på förfrågningar från **Docker-klienten** (t.ex. när du kör kommandon som ”docker run”, ”docker build” osv.) via **Docker REST API**.

2. **Hantera livscykeln för containrar**
   Den sköter allt som rör:
   - Skapande, körning och stopp av containrar.
   - Hantering av containerbilder.
   - Tilldelning av resurser som CPU och minne.

3. **Kommunicera med andra Docker Daemons**
   Docker-servern kan samarbeta med andra servrar för att hantera containrar över flera värdar, vilket är vanligt i distribuerade miljöer eller kluster.

Docker-objekt

Ett Docker-objekt är en komponent inom Docker-ekosystemet som du kan skapa, konfigurera och hantera.

1. En **container** är en körbar instans av en Docker-bild som isolerar applikationer med deras miljö.
2. En **bild** är en oföränderlig mall som innehåller allt som behövs för att köra en applikation.
3. Ett Docker-**nätverk** möjliggör kommunikation mellan containrar.
4. En **volym** används för att beständigt lagra data som genereras och används av containrar.
5. Ett **plugin-program** utökar Dockers funktionalitet, till exempel för specialiserade nätverks- eller lagringslösningar.
6. I Docker Swarm används **tjänster** för att hantera distribuerade applikationer över flera noder.

Sammanfattning:

- Docker-objekt är de fundamentala enheterna som utgör Docker-ekosystemet.
- De kan skapas, konfigureras och hanteras via Docker-kommandon.
- Genom att förstå hur dessa objekt ser ut och fungerar kan du effektivt bygga och underhålla containeriserade applikationer.

Dockerfile

En **Dockerfile** är en textfil som innehåller en uppsättning instruktioner för hur en Docker-avbildning (image) ska byggas.

Den beskriver steg-för-steg-processen för att skapa en container med alla dess beroenden, miljövariabler och kommandon.

Syfte:

1. Automatisera byggandet av Docker-avbildningar.
2. Säkerställa att avbildningar byggs konsekvent varje gång, oavsett miljö.

Docker-compose

Docker Compose är ett verktyg som används för att definiera och köra multi-container Docker-applikationer.

Med hjälp av en YAML-fil kan du konfigurera alla dina applikationstjänster, nätverk och volymer på ett ställe.

Istället för att starta varje container manuellt med docker run, kan du använda docker-compose up för att starta alla tjänster definierade i din docker-compose.yml-fil.

Exempel på Docker Compose med PostgreSQL och .NET 8

Exempel där en .NET 8-applikation interagerar med en PostgreSQL-databas.

Definera två tjänster i vår docker-compose.yml-fil:

- webb - Vår .NET 8-applikation.
- db - PostgreSQL-databasen.

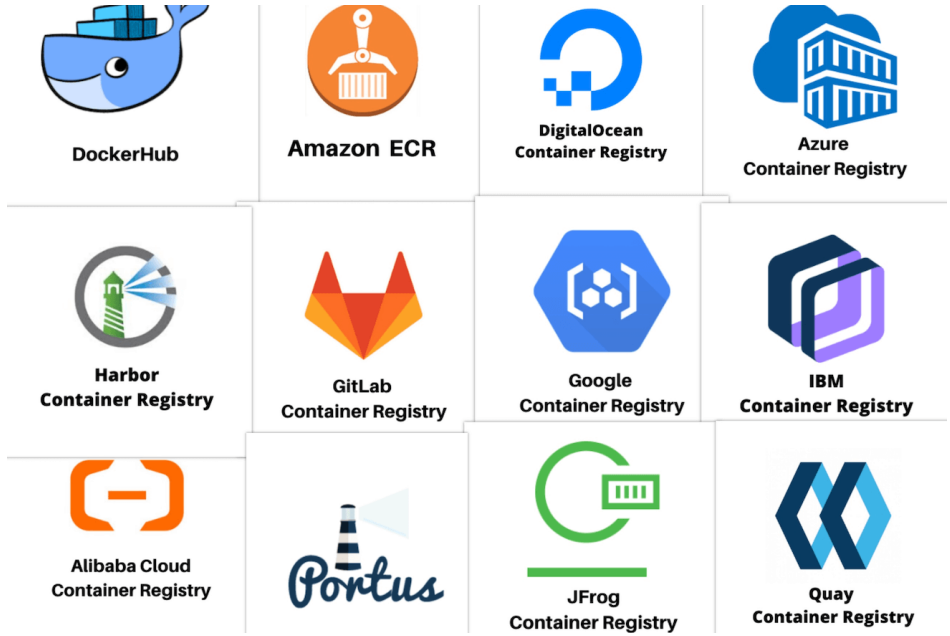
Vad är ett Docker-register?

Ett Docker-register är en lagringsplats där vi kan lagra, hantera och distribuera containeravbildningar (docker images) som vi skapar.

Det fungerar som ett bibliotek för dina applikationer i containerform.

Varför är Docker-register viktiga?

De underlättar delning och distribution av applikationer. Möjliggör snabbare och mer effektiv utveckling och distribution.



Docker Hub är ett Software as a Service (SaaS) Docker-containerregister.

Det är det standardmässiga offentliga registret som Docker använder för avbildningshantering.

Funktioner:

1. **Offentliga och privata arkiv:**
   - Offentliga arkiv där vem som helst kan komma åt dina avbildningar.
   - Möjlighet att skapa privata arkiv för känsliga eller interna avbildningar.
2. **Automatiska byggnationer:**
   - Integrera med källkodsförvar som GitHub för att automatiskt bygga avbildningar när koden uppdateras.
3. **Teamhantering:**
   - Samarbeta med andra genom att skapa organisationer och team med olika åtkomsträttigheter.

Användningsexempel:

1. **Dela avbildningar med communityn:**
   - Publicera dina applikationer så att andra kan använda dem.
2. **Åtkomst till officiella avbildningar:**
   - Hämta färdiga avbildningar för populära programvaror som Nginx, Redis och PostgreSQL.

Summering

1. **Docker Engine**: Kärnan i Docker, en klient-server-applikation som består av en Docker-daemon (dockerd), ett REST API och ett kommandoradsgräns snitt (CLI).
2. **Docker Image (Avbildning)**: En oföränderlig mall som innehåller allt som behövs för att köra en applikation, inklusive källkod, bibliotek och systemverktyg.
3. **Docker Container**: En körbar instans av en Docker-avbildning. Containrar isolerar applikationen från värdsystemet, vilket förbättrar säkerheten och stabiliteten.
4. **Dockerfile**: En textfil som innehåller instruktioner för att bygga en Docker-avbildning. Den beskriver steg för steg hur avbildningen ska skapas.
5. **Docker Compose**: Ett verktyg för att definiera och köra multi-container Docker-applikationer. Med hjälp av en YAML-fil kan du konfigurera alla dina applikationstjänster.
6. **Docker Swarm**: Dockers inbyggda orkestreringsverktyg som tillåter hantering av ett kluster av Docker-noder som en enda virtuell värd.
7. **Docker Hub**: Ett molnbaserat register där du kan lagra och dela dina Docker-avbildningar, både offentligt och privat.

Övningar

https://learn.microsoft.com/sv-se/training/modules/intro-to-docker-containers/
https://docs.docker.com/guides/dotnet/
https://www.docker.com/play-with-docker/

Referenser

https://docs.docker.com/
https://hub.docker.com/