# Lektionstillfälle 2

”Docker”

Utbildare: Robert Westin

NACKADEMIN

# Kort summering av föregående lektion/ev. lektioner

**Föregående lektion:**

- Vad är CI & CD
- Continuous Delivery vs Continuous Deployment
- Python Flask

NACKADEMIN

# Lektionstillfällets mål och metod

**Mål med lektionen:**

- Förstå grunderna i Docker
- Kunna utföra Docker kommandon

**Lektionens arbetsmetod/er:**

- Föreläsning
- Labbar

NACKADEMIN

# Begreppsgenomgång

- Docker
- Image
- Container
- Lagring icke-persistent vs persistent
- lokal lagring
- Nätverk i Docker

NACKADEMIN

# Historia SnowflakeServer & PhenixServer

- Snowflake (pets)
  - En server kan bli en unik snöflinga
  - Många arbetar och "fixar" med den
  - Uppdateringar
  - Säkerhetsfix
  - Installation av olika tjänster
  - Lever länge: månader, år
- Drift
  - Förväntat konfiguration skiljer sig från faktiskt konfiguration

Bild av: OliBac

Läs mer: https://martinfowler.com/bliki/SnowflakeServer.html

NACKADEMIN

# Historia SnowflakeServer & PhenixServer

- Phenix (cattle)
  - När en server eldas upp, reser en annan
  - Ingen gör ändringar på servern
  - När tjänsten ska uppdateras så skapar vi en ny
  - Lever kort: minuter, timmar eller dagar

Läs mer: https://martinfowler.com/bliki/PhoenixServer.html

NACKADEMIN

# Configuration Management

- Chef, Puppet, Ansible etc.
- Fokuserar på att automatisera och minimera snowflake problematik
- Automatisering installerar och konfigurerar befintliga servers
- System för att minimera drift av system, t.ex.
  - Pull via agent (puppet)
  - Push via agentless Ansible som loggar in via ssh och utför ändringar
- Configuration Management öppnade även upp för Phenix servers
  - Men var långsamt
  - Kräver systemresurser som övervakar

NACKADEMIN

# Mutable Vs Immutable Infrastructure

- Snowflake & Phenix handlade först om servrar
- Men det finns mer:
  - Virtuella maskiner
  - Proxys
  - Nätverk
  - Kommunikationsmönster
- Infrastructure as Code (IaC)
  - Möjlighet att lösa ett större problem
  - Infrastruktur som helhet
  - Cloud (Private & Public)

NACKADEMIN

Film & mer läsning: https://www.hashicorp.com/resources/what-is-mutable-vs-immutable-infrastructure

# Containers & Kubernetes

- IaC helt själv kan vara:
  - Komplext
  - Tidskrävande
  - Många kompetenser krävs
- Kubernetes & Containers
  - Ett alternativ sätt att köra applikationer och dess nätverk
  - Möjligt för utvecklare att skapa sina egna setups
  - DevOps
  - Resurssnålt
  - Från utvecklarens sida kan det ses som en PaaS

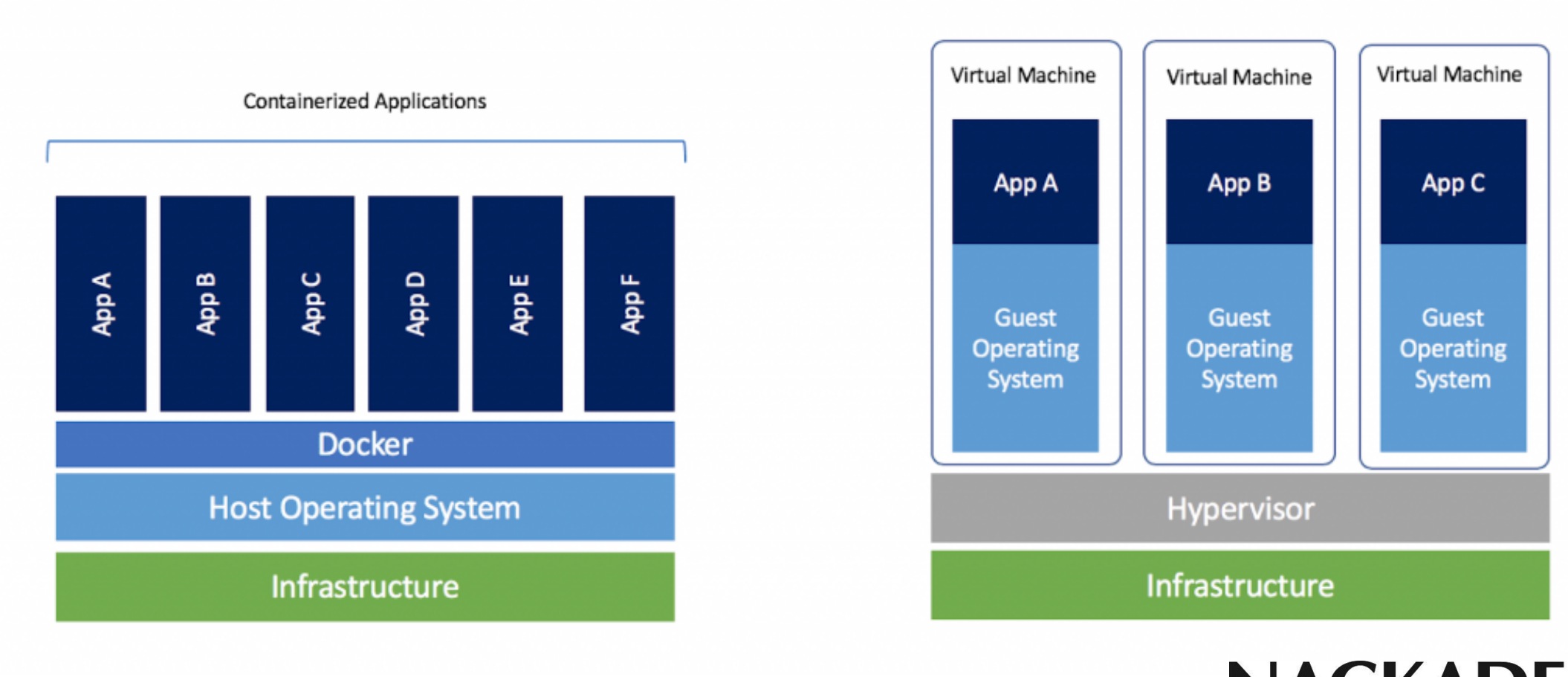
NACKADEMIN

# Vad är en container?

- https://www.docker.com/resources/what-container
- Lättviktig
- Körs ovanpå OS, till skillnad från Virtual Machine som kör hela OS
- Namespace
    - Nätverk
    - Mounts
    - Users
    - Process
    - Root directory
    - Etc.

NACKADEMIN

# Docker vs Virtual Machine




Källa: https://www.docker.com/blog/containers-replacing-virtual-machines/

NACKADEMIN

# Vad tycker utvecklare om Docker?

- Stack Overflow
  - https://insights.stackoverflow.com/survey/2020
  - https://survey.stackoverflow.co/2022/

NACKADEMIN

# Docker uppdelning historik/framtid

- Docker har gått ifrån en rörelse till en "produkt"
- Flera års arbete har gått till att bryta ut komponenter
  - t.ex Containerd för container lifecycle management är donerat till cncf https://www.cncf.io/projects/containerd/
  - Donerat/samägt via The Open Container Initiative (OCI)
    - Runc (för att köra instansen av en OCI Image)
    - OCI Image Spec
    - OCI Runtime Spec
    - OCI Distrubution Spec
    - https://opencontainers.org/about/overview/
    - https://www.docker.com/blog/open-container-project-foundation/

# Open Source alternativ

- Komplett svit för att köra kubernetes och containers
  - https://rancherdesktop.io/
- Container livscykel hantering med ctr klient
  - https://containerd.io/
- Källa att hitta andra framtida projekt av intresse
  - https://landscape.cncf.io/

NACKADEMIN

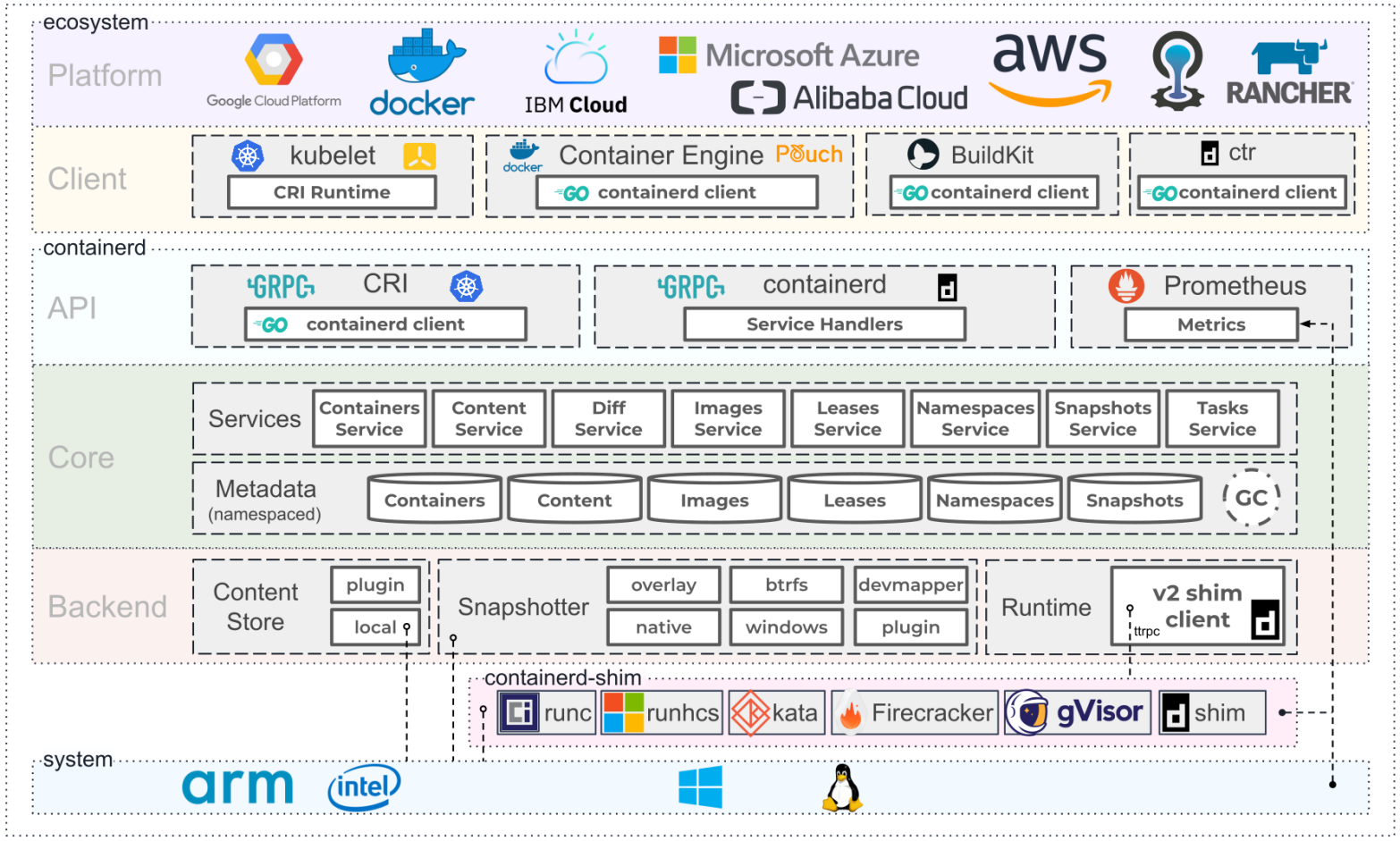
# Cloud Native Computing Foundation (CNCF)

- https://www.cncf.io/about/who-we-are/
- https://github.com/cncf/toc/blob/main/DEFINITION.md

- Cloud native technologies empower organizations to build and run scalable applications in modern, dynamic environments such as public, private, and hybrid clouds. Containers, service meshes, microservices, immutable infrastructure, and declarative APIs exemplify this approach.
- These techniques enable loosely coupled systems that are resilient, manageable, and observable. Combined with robust automation, they allow engineers to make high-impact changes frequently and predictably with minimal toil.
- The Cloud Native Computing Foundation seeks to drive adoption of this paradigm by fostering and sustaining an ecosystem of open source, vendor-neutral projects. We democratize state-of-the-art patterns to make these innovations accessible for everyone.

NACKADEMIN

# Containerd




NACKADEMIN

Läs mer på: https://containerd.io/

# Varför ska vi använda containers?

- Mutable Infrastructure vs Immutable Infrastructure
  - Ska vi ändra på serverar, eller bara skapa nytt?
- Availability traditionellt
  - MTTR (Mean time to repair, recover)
  - MTBF (Mean time between failures)
  - Uptime 99.9 eller 99.99?
- Minimera konfigurations drift (icke planerad ändring)
- Versionerat, reproducerbart, felsökning
- En container kan innehålla? Node, Python, nginx?

NACKADEMIN

# Rolig övning från wikipedia

- **Powers of 10 trick**
- **Hur länge får servern vara nere?**
- **8.64 * 10^x för n + x = 4 där n är antalet 9:or i uptime**

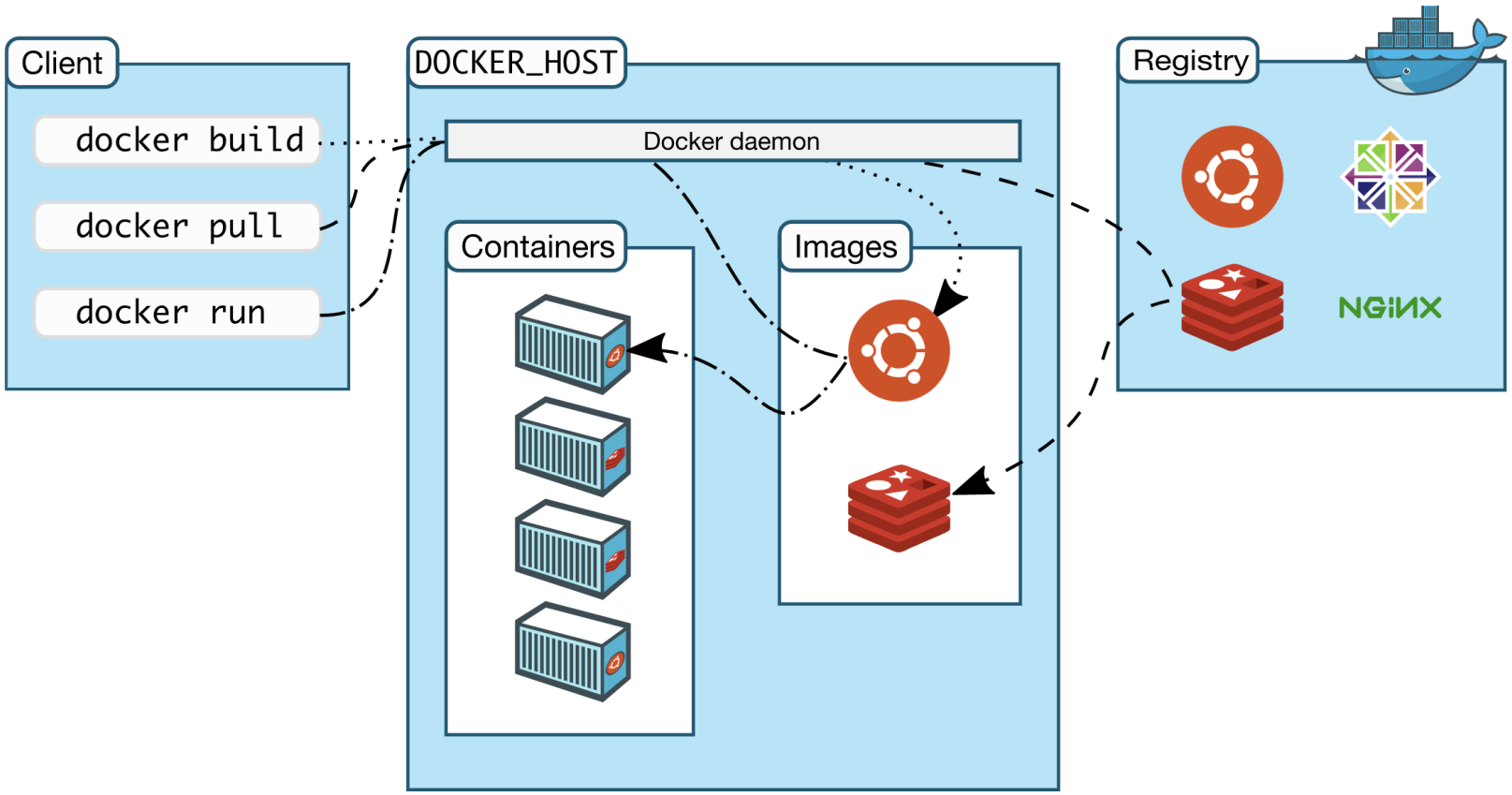
NACKADEMIN

# Varför ska vi använda containers?

- **Deployment Frequency**—How often an organization successfully releases to production
- **Lead Time for Changes**—The amount of time it takes a commit to get into production
- **Change Failure Rate**—The percentage of deployments causing a failure in production
- **Time to Restore Service**—How long it takes an organization to recover from a failure in production
- Källa: https://cloud.google.com/blog/products/devops-sre/using-the-four-keys-to-measure-your-devops-performance

NACKADEMIN

# Docker Överblick




Källa: https://docs.docker.com/get-started/overview/

NACKADEMIN

# Docker registry

För att bygga egna docker images så används docker build
Men för att kunna dela och deploya behövs ett centralt register

Det finns olika lösningar

- DockerHub (default)
- GitLab Container Registry
- Github Packages (ghcr.io)
- Artifactory
- GCE, AWS, Kubernetes Private Registry

NACKADEMIN

# Docker Kommandon

Vad finns på deras cheatsheet?

https://docs.docker.com/get-started/docker_cheatsheet.pdf

NACKADEMIN

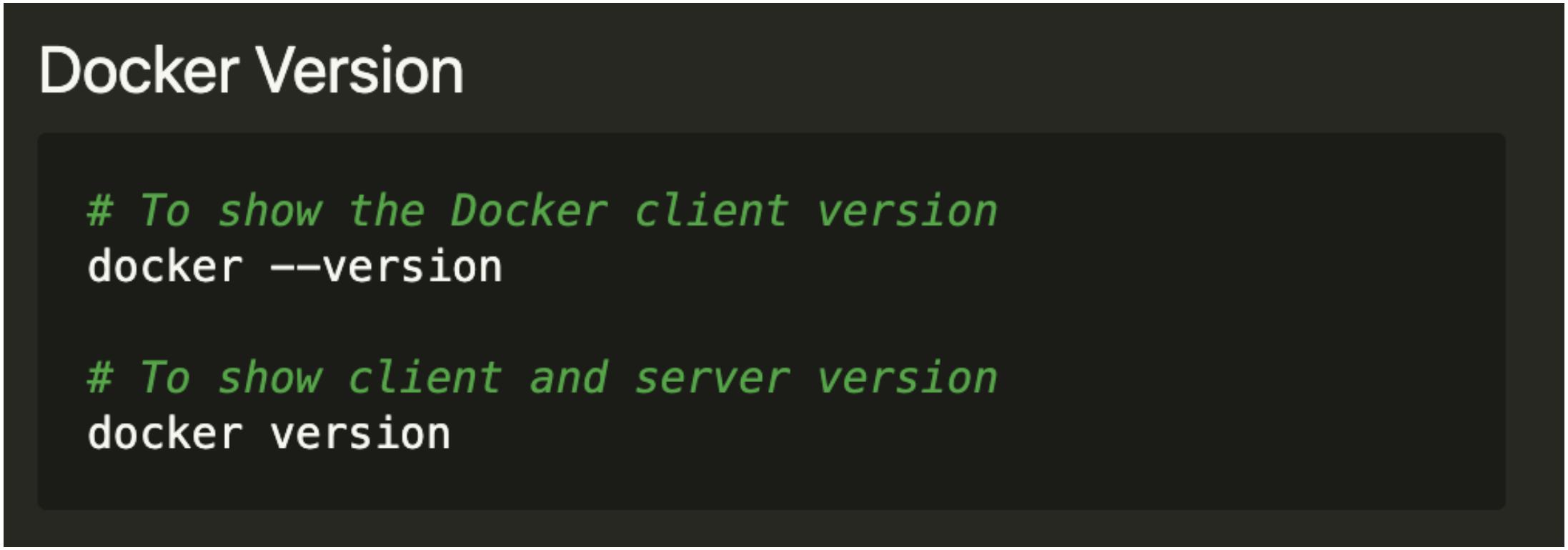
# Docker Help

```
# Docker built-in help
$ docker --help

# You can get help for a specific command with
$ docker run --help
```

NACKADEMIN

# Docker Version




NACKADEMIN

# Docker on Windows

## Docker on windows

If Docker is used with git bash and you get the error message `the input device is not a TTY`, try to run it with `winpty`.

```
# So use winpty before docker (git bash on windows only)
winpty docker run -P nginx:alpine
```

NACKADEMIN

# Docker container run

```
# Run a nginx container
$ docker container run nginx:alpine

# Run a nginx container and open a port 7777 on
the host
$ docker container run -p 7777:80 nginx:alpine

# You can try to access the server in the
browser with url: http://127.0.0.1:7777/
```

NACKADEMIN

# Docker ps

```
# List all running containers
$ docker ps

CONTAINER ID   IMAGE          COMMAND                  CREATED         STATUS         PORTS                     NAMES
2184230b4834   nginx:alpine   "/docker-entrypoint.…"   2 seconds ago   Up 2 seconds   0.0.0.0:55001->80/tcp   cool_hermann

# list all containers
$ docker ps --all

CONTAINER ID   IMAGE          COMMAND                  CREATED          STATUS                      PORTS                     NAMES
e5937e902c2f   nginx:alpine   "/docker-entrypoint.…"   16 seconds ago   Exited (0) 14 seconds ago                             great_satoshi
2184230b4834   nginx:alpine   "/docker-entrypoint.…"   3 minutes ago    Up 3 minutes                0.0.0.0:55001->80/tcp   cool_hermann
```

NACKADEMIN

# Docker container stop

Docker container stop

Usage: docker container stop [OPTIONS] CONTAINER [CONTAINER...]

Stop one or more running containers

```
# Stop a running docker container
$ docker stop CONTAINER

# Stop all running containers (linux/mac)
$ docker stop $(docker ps -q)
```

NACKADEMIN

# Docker container start

Docker container start

Usage: docker container start [OPTIONS] CONTAINER [CONTAINER...]

Start one or more stopped containers

```
# Start a stopped container
$ docker container start CONTAINER
```

NACKADEMIN

# Docker container exec

## Docker container exec

Usage: docker container exec [OPTIONS] CONTAINER COMMAND [ARG...]

Run a command in a running container

| Options | Description |
| --- | --- |
| -i, --interactive | Keep STDIN open even if not attached |
| -t, --tty | Allocate a pseudo-TTY |

```
# Open a shell on a running container
$ docker exec -it CONTAINER sh

# List a folder on a running container
$ docker exec -it CONTAINER ls /usr/bin
```

NACKADEMIN

# Docker system prune

## Docker Cleanup

```
# Basic cleanup
$ docker system prune

# To remove all unused images add -a
$ docker system prune -a

# To remove everything unused
$ docker system prune -a --volumes
WARNING! This will remove:
  - all stopped containers
  - all networks not used by at least one container
  - all volumes not used by at least one container
  - all images without at least one container associated to
them
  - all build cache
```

NACKADEMIN

# Docker attach och exec

- docker container attach alpine_1
- Vad händer om du kör exit?

- docker container exec -it alpine_1 sh
- Vad händer om du kör exit?

NACKADEMIN

# Fysisk vs Logiskt utrymme

- Logiskt utrymme är mängden information (Antal tecken)
- Fysiskt utrymme är hur mycket plats den faktiskt tar på disk

# Docker – Shared vs Unique size

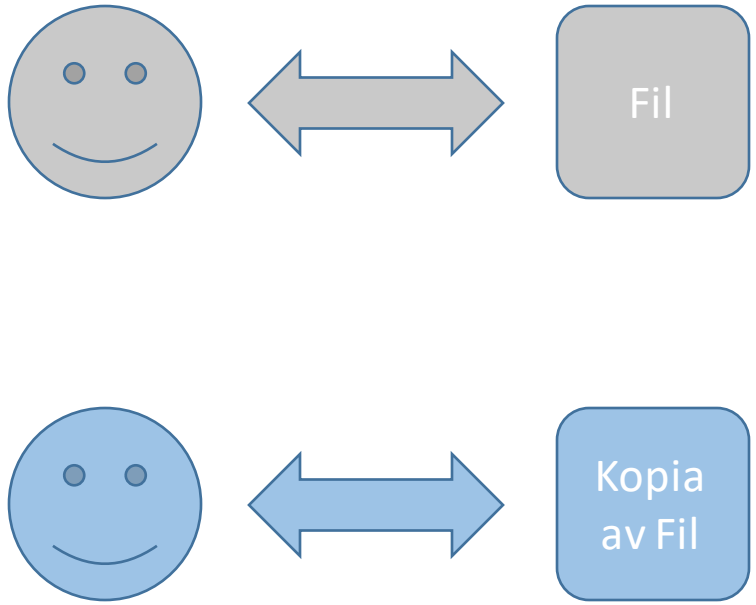
```
~/repos/devops21/devops21_contin  main +  docker system df --verbose
Images space usage:

REPOSITORY   TAG       IMAGE ID       CREATED         SIZE      SHARED SIZE   UNIQUE SIZE   CONTAINERS
nginx        alpine    b997307a58ab   2 weeks ago     23.58MB   5.544MB       18.04MB       2
alpine       latest    9c6f07244728   3 months ago    5.544MB   5.544MB       0B            1
```

- SHARED SIZE is the amount of space that an image shares with another one (i.e. their common data)
- UNIQUE SIZE is the amount of space that is only used by a given image
- SIZE is the virtual size of the image, it is the sum of SHARED SIZE and UNIQUE SIZE

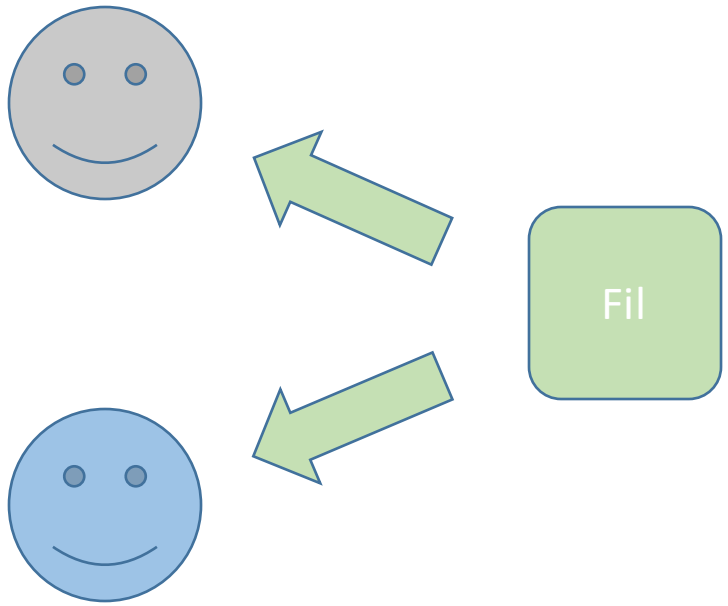
NACKADEMIN

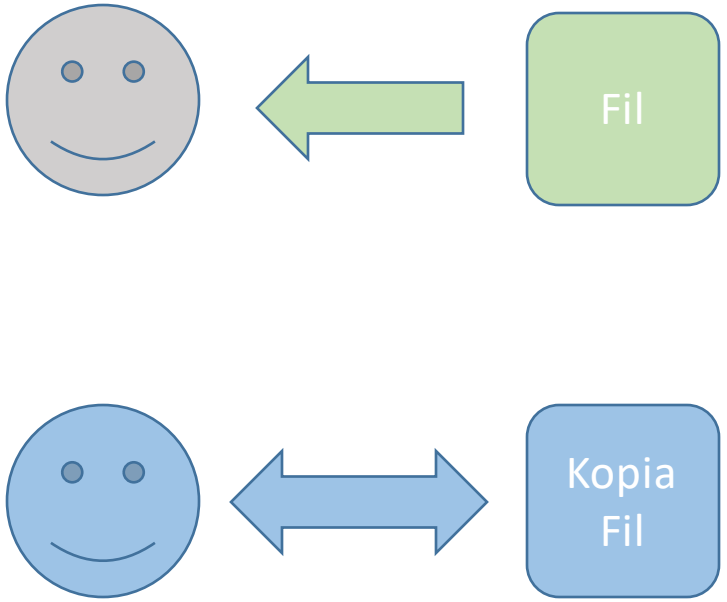
# Copy – Skriva och läsa till varsin fil




NACKADEMIN

# Copy On Write (COW)




NACKADEMIN

# Docker – image inspect

- docker image inspect alpine

```
"RootFS": {
    "Type": "layers",
    "Layers": [
        "sha256:994393dc58e7931862558d06e46aa2bb17487044f670f310dffe1d24e4d1eec7"
    ]
},
```

- docker image inspect nginx:alpine

```
"RootFS": {
    "Type": "layers",
    "Layers": [
        "sha256:994393dc58e7931862558d06e46aa2bb17487044f670f310dffe1d24e4d1eec7",
        "sha256:b96b16a53835a653cf4ba4da2bcebf8393403fd68d4f00c3f6fd56dfe92c48e8",
        "sha256:d51445d70778dc924c28175bba3c65d4da962ccf4121a17e03d0b0e896e0d256",
        "sha256:acf5e0b2cf0814a5d226d89969f0beff1e56b959bcd8af9b058f48efe7192eac",
        "sha256:6e96dd581d79dd4df16ba97f1740aff93df3e3fcfbf0ce954c10a23c4583f624",
        "sha256:0618d1e529faeab626a4f04f4245abfb8937d4caa1ecf5d9ffcd1af0324657ab"
    ]
},
```

NACKADEMIN

# Docker – Container

```
~/repos/devops21/devops21_contin  main +  docker run -it alpine sh
/ # echo "hello world!" > hello.txt
/ # exit
```

```
~/repos/devops21/devops21_contin  main +  docker system df --verbose
Images space usage:

REPOSITORY   TAG       IMAGE ID       CREATED        SIZE      SHARED SIZE   UNIQUE SIZE   CONTAINERS
nginx        alpine    b997307a58ab   2 weeks ago    23.58MB   5.544MB       18.04MB       2
alpine       latest    9c6f07244728   3 months ago   5.544MB   5.544MB       0B            2

Containers space usage:

CONTAINER ID   IMAGE          COMMAND                  LOCAL VOLUMES   SIZE      CREATED          STATUS
37a58296ee66   alpine         "/bin/sh"                0               0B        15 seconds ago   Exited (0) 14 seconds ago
6292919cdcfd   alpine         "sh"                     0               50B       2 minutes ago    Exited (0) 2 minutes ago
e81224114f81   nginx:alpine   "/docker-entrypoint.…"   0               1.09kB    36 minutes ago   Up 36 minutes
13af7926e6f2   nginx:alpine   "/docker-entrypoint.…"   0               1.09kB    38 minutes ago   Up 38 minutes
```

NACKADEMIN

# Storage Drivers vs Docker volumes

Docker uses storage drivers - to store image layers, and to store data in the writable layer of a container. The container's writable layer does not persist after the container is deleted, but is suitable for storing ephemeral data that is generated at runtime. Storage drivers are optimized for space efficiency, but (depending on the storage driver) write speeds are lower than native file system performance, especially for storage drivers that use a copy-on-write filesystem. Write-intensive applications, such as database storage, are impacted by a performance overhead, particularly if pre-existing data exists in the read-only layer.

Use Docker volumes for write-intensive data, data that must persist beyond the container's lifespan, and data that must be shared between containers. Refer to the volumes section to learn how to use volumes to persist data and improve performance.

NACKADEMIN

Källa: https://docs.docker.com/storage/storagedriver/

# Storage Drivers vs Docker volumes

Docker storage drivers -

- Image lager
- Skriv lager för container (inte persistent utan container)
- Runtime data
- Optimerat för att använda lagringsutrymmet effektivt
- Sämre skrivprestanda (COW)

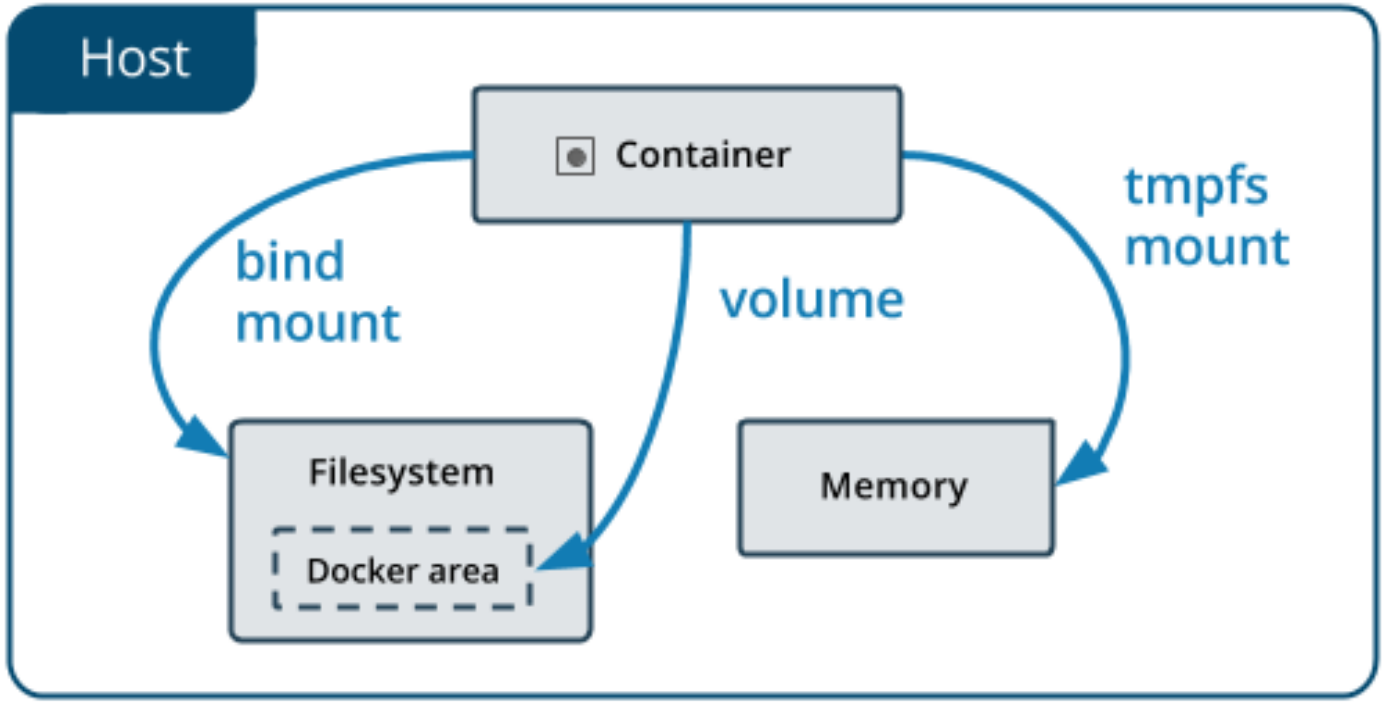
Docker volumes

- Intensivt skrivande av data
- Persistent utanför container
- Delat mellan containers

NACKADEMIN

Källa: https://docs.docker.com/storage/storagedriver/

# Docker välj rätt mount




NACKADEMIN

Källa: https://docs.docker.com/storage/

# Docker välj rätt mount

- **Volumes** are stored in a part of the host filesystem which is *managed by Docker* (/var/lib/docker/volumes/ on Linux). Non-Docker processes should not modify this part of the filesystem. Volumes are the best way to persist data in Docker.
- **Bind mounts** may be stored *anywhere* on the host system. They may even be important system files or directories. Non-Docker processes on the Docker host or a Docker container can modify them at any time.
- **tmpfs mounts** are stored in the host system's memory only, and are never written to the host system's filesystem.

NACKADEMIN

Källa: https://docs.docker.com/storage/

# Docker välj rätt mount

- **Volumes**
  - Sparas i värdens filsystem
  - Managerat av docker *Docker* (/var/lib/docker/volumes/ på Linux).
  - Låt inte andra än docker modifiera filerna
  - Bästa sättet att persistera data i docker
- **Bind mounts**
  - Filer eller foldrar som sparas varsomhelst på värdens filsystem
  - Både docker och värden kan modifiera filerna
- **tmpfs mounts**
  - Sparas i värdens minne
  - Sparas inte på värdens filsystem
  - Om containern stannar så försvinner filerna

NACKADEMIN

Källa: https://docs.docker.com/storage/

# Docker Volumes

- Sharing data among multiple running containers. If you don't explicitly create it, a volume is created the first time it is mounted into a container. When that container stops or is removed, the volume still exists. Multiple containers can mount the same volume simultaneously, either read-write or read-only. Volumes are only removed when you explicitly remove them.
- When the Docker host is not guaranteed to have a given directory or file structure. Volumes help you decouple the configuration of the Docker host from the container runtime.
- When you want to store your container's data on a remote host or a cloud provider, rather than locally.
- When you need to back up, restore, or migrate data from one Docker host to another, volumes are a better choice. You can stop containers using the volume, then back up the volume's directory (such as /var/lib/docker/volumes/<volume-name>).
- When your application requires high-performance I/O on Docker Desktop. Volumes are stored in the Linux VM rather than the host, which means that the reads and writes have much lower latency and higher throughput.
- When your application requires fully native file system behavior on Docker Desktop. For example, a database engine requires precise control over disk flushing to guarantee transaction durability. Volumes are stored in the Linux VM and can make these guarantees, whereas bind mounts are remoted to macOS or Windows, where the file systems behave slightly differently.

NACKADEMIN

Källa: https://docs.docker.com/storage/#good-use-cases-for-volumes

# Docker Bind mounts

In general, you should use volumes where possible. Bind mounts are appropriate for the following types of use case:

- Sharing configuration files from the host machine to containers. This is how Docker provides DNS resolution to containers by default, by mounting /etc/resolv.conf from the host machine into each container.
- Sharing source code or build artifacts between a development environment on the Docker host and a container. For instance, you may mount a Maven target/ directory into a container, and each time you build the Maven project on the Docker host, the container gets access to the rebuilt artifacts.

  If you use Docker for development this way, your production Dockerfile would copy the production-ready artifacts directly into the image, rather than relying on a bind mount.
- When the file or directory structure of the Docker host is guaranteed to be consistent with the bind mounts the containers require.

NACKADEMIN

Källa: https://docs.docker.com/storage/

# Docker tmpfs mounts

tmpfs mounts are best used for cases when you do not want the data to persist either on the host machine or within the container. This may be for security reasons or to protect the performance of the container when your application needs to write a large volume of non-persistent state data.

NACKADEMIN

Källa: https://docs.docker.com/storage/

# Docker Storage Drivers

https://docs.docker.com/storage/storagedriver/

Tidigare: aufs

https://docs.docker.com/storage/storagedriver/aufs-driver/

Idag: overlay2

https://docs.docker.com/storage/storagedriver/overlayfs-driver/

NACKADEMIN

# Docker Storage Drivers

Vilken storage driver använder du?

Docker info

NACKADEMIN

# Docker bind mount

- Dela filer mellan host och container (mac/linux)

```
docker run -d \
  --name devtest \
  --mount type=bind,source="$(pwd)"/target,target=/app \
  nginx:latest
```

NACKADEMIN

# Docker bind mount (gamla viset)

- Dela filer mellan host on container

```
$ docker run -d \
   --name devtest \
   -v "$(pwd)"/target:/app \
   nginx:latest
```

NACKADEMIN

# Docker volume

Docker volumes används för persistent lagring

- Filer kan sparas när containern byts ut
- Filer kan återanvändas av annan container
- Filer försvinner inte om containern slängs bort
- OBS Kan fortfarande försvinna pga. prune

Exempel:

- docker volume create my_volume # skapa volym
- docker volume help # lista kommandon
- docker volume rm my_volume # ta bort den
- docker volume inspect # inspektera volym
- docker volume ls # lista volymer

NACKADEMIN

# Docker bind volume

Se https://docs.docker.com/storage/volumes/#create-and-manage-volumes för mer exempel

Exempel:

```
docker container run --detach --name test_nginx --mount type=volume,source=html-volume,target=/usr/share/nginx/html nginx
```

```
docker volume ls
```

NACKADEMIN

# Docker container nätverk

Publish gör så att portar kan mappas mot din host

- I exempel mappas porten 8080 för mot containers port 80
- Publish används för att utveckla och testa dina egna applikationer

docker container run --detach --name test_nginx --publish 8080:80 nginx

-P ger random port

docker container run –d –name test_nginx_2 –P nginx

NACKADEMIN

# Docker container nätverk

https://docs.docker.com/network/

Bridge

Host

Overlay

Ipvlan

Macvlan

None

Plugins (https://docs.docker.com/engine/extend/plugins_services/#network-plugins)

NACKADEMIN

# Docker bridge nätverk

Bridge

- Standardval driver
- Default nätverks driver
- Bra för ensamma containers

Läs mer: https://docs.docker.com/network/bridge/

Labb: https://docs.docker.com/network/network-tutorial-standalone/

NACKADEMIN

# Docker host nätverk

- **Host networks**
  - Bäst om containern inte behöver vara isolerad från värden
  - Du delar nätverk med värden

  docker run --rm -d --network host --name my_nginx nginx

Läs mer: https://docs.docker.com/network/host/

Labb: https://docs.docker.com/network/network-tutorial-host/

NACKADEMIN

# Docker overlay nätverk

Nätverk över flera docker värdar

Läs mer: https://docs.docker.com/network/overlay/

Labb: https://docs.docker.com/network/network-tutorial-overlay/

NACKADEMIN

# Docker none nätverk

```
docker run --rm -dit \
  --network none \
  --name no-net-alpine \
  alpine:latest \
  Ash
```

```
docker exec no-net-alpine ip link show
docker stop no-net-alpine
```

Läs mer: https://docs.docker.com/network/none/

NACKADEMIN

# Docker starta automatiskt

Läs mer: https://docs.docker.com/config/containers/start-containers-automatically/

NACKADEMIN

# Summering av dagens lektion

- Docker
- Övningar från dockers officiella dokumentation

NACKADEMIN

# Framåtblick inför nästa lektion

- Vi skriver våran egen Dockerfile
- Vi bygger våran egen flask applikation

NACKADEMIN