Virtualisering och Containers

Mikael Svahnberg*

2024-02-28

1 Introduktion

- Virtuella Maskiner och Containers
- Utveckla i en Container
- Utvecklingsprocess för Kompilerade respektive Tolkade programspråk

2 Virtuella Maskiner

1. Bare Metal
   - Ingen virtualisering
   - Om mer resurser behövs → köp fler fysiska maskiner / mer minne / fler CPU:er
   - kan samhanteras i gemensamma datacenter.
2. Virtella Maskiner
   - Kör ett gäst-operativsystem på *emulerad hårdvara* ovanpå ett värd-OS.

   Värd Där du vill köra dina virtuella maskiner

   Gäst En virtuell maskin som körs på en värd.
3. Lättvikts-virtuella maskiner (Linux Containers LXC, Containers)
   - Använder stora delar av värdens operativsystem, men emulerar "sista biten"
   - Innesluten sandlåda, det som händer inuti containern stannar där.
   - I stora drag ekvivalent med `chroot` i unix.

... vidare:

4. Tolkade program ovanpå en sandlåde-runtime environment (t.ex. Java)
   - Separat minnesutrymme

*Mikael.Svahnberg@bth.se

- till synes ensam på CPUn
- Delad disk

5. Processer
6. Trådar
7. Korutiner
8. Lexografisk Bindning vs Dynamisk Bindning
9. ...

3 Anledningar till Virtuella Maskiner

- Du behöver en stabil dator att arbeta på som är skild från din testmiljö.
- Du behöver kunna återställa hela datorn (och allt installerat) till en känt ursprungsläge.
- Du vill minska gapet mellan din utvecklingsmiljö, din testmiljö, och din produktionsmiljö.
- Du vill dela upp din applikation i mindre, självständiga enheter (s.k. *microservices*).
- Du vill kunna använda olika system och programspråk för olika uppgifter.
- Du utvecklar för molnet.

4 Virtualisering och Cloud Computing

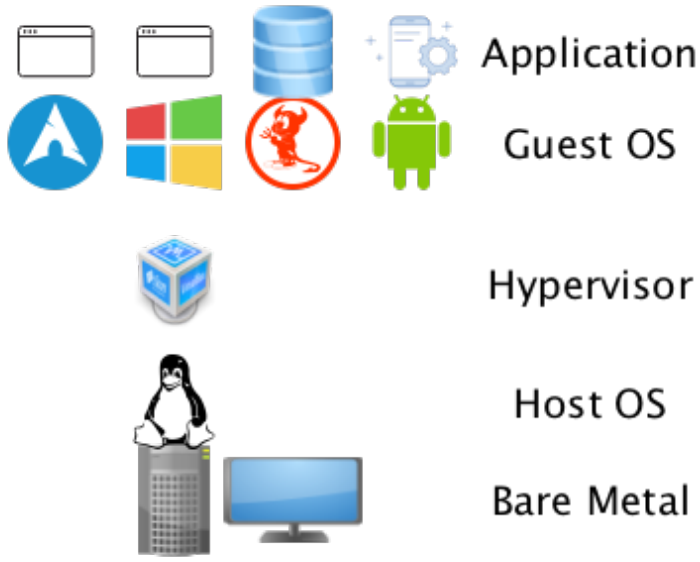
Rosenberg & Mateos (2010) listar följande stöttepelare för Cloud Computing:

1. Virtualised Computing Resources
2. Automatic creation and deletion of new VM's
3. Elastic Scaling up or down
4. Pooled Computing Resources
5. Resource usage billed only as used

Där:

- Nummer 5 beror på 1–4
- Nummer 4 beror på 3
- Nummer 3 beror på 2
- Nummer 2 beror på 1

5 Virtuella Maskiner och Hypervisor



- En *Hypervisor* agerar som ett lager mellan värd-operativsystemet och gästen.
- Exempel på hypervisors
  - VMWare - gamla i spelet, stort utbud; fungerar också (numera) på Mac M1/M2-chip.
  - VirtualBox - Oracle, lätt tillgängligt *men se upp med licensen!*
  - Qemu `https://www.qemu.org` - Open source. Lite meckigt att sätta upp, men väl värt det!
  - Hyper-V - Redan installerat på Windows, men kräver lite inställningar för att komma igång med.
  - Parallels - för Mac-datorer.
- ... men också:
  - Docker
  - Podman `https://podman.io/` (som Docker, fast open source och trevligare licens)
  - KVM
  - `chroot`

6 VM eller Container

- En VM ger dig en hel dator – om du vill även med en virtuell skärm.

- Du kan logga in, köra program, utforska filsystemet *precis som vanligt*
- En Container är (oftast) inriktad på att köra en enda uppgift
  - När uppgiften är färdig stängs containern ned.
  - Du *kan* logga in (under tiden programmet körs), men miljön är ofta väldigt begränsad.
- Idag räcker ofta en container.
- Containers är särskilt användbara för utveckling av *microservices*

7 Kom igång med Qemu OVERVIEW

- Installera program:
  - *Qemu* och *libvirt*
  - *virt-manager* `https://virt-manager.org/`
- Man kan antingen sätta upp en "tom" maskin via *virt-manager*,
  - ange ett installations-medium (en `.iso` - fil)
- Eller så kan man använda kommandoraden och göra nästan samma
- Eller så använder man ett verktyg som *Vagrant* `https://www.vagrantup.com/`

7.1 Qemu på Kommandoraden

```
# Kanske behöver starta libvirt
sudo systemctl start libvirtd

# Skapa en virtuell disk
qemu-img create -f qcow2 xfce-manjaro-disk 15G

# Installera från en iso-fil
qemu-system-x86_64 -enable-kvm -m 2G -cpu host\
                   -vga virtio -display sdl,gl=on\
                   -drive file=xfce-manjaro-disk\
                   -cdrom manjaro-xfce-21.3.7-minimal-220816-linux515.iso\
                   -boot order=d

# Kör som vanligt, när du har installerat.
qemu-system-x86_64 -enable-kvm -m 2G -cpu host\
                   -vga virtio -display sdl,gl=on\
                   -drive file=xfce-manjaro-disk
```

8 Vagrant och Automatisk ProvisioneringOverview

- *Provisionering* kallar man steget när man installerar den programvara man behöver på en maskin.
  - Steget *"Installera från ISO"* är inte automatiskt!
  - Man måste manuellt installera all programvara man vill använda.
- Målet med provisionering är att beskriva i text (som kommandon) vad som skall installeras, så att:
  - man med ett enda kommando kan starta en helt nyinstallerad maskin
  - man kan versionshantera vad som installeras på maskinen
  - man kan dela med sig av maskinkonfigurationen till andra som alla kommer köra samma installation.

8.1 Provisionering med Vagrant

- Vagrant `https://www.vagrantup.com/` är ett exempel på verktyg för att hjälpa till med provisionering.
  - Lite handpåläggning behövs för att få Vagrant att fungera med Qemu, men det är värt det.
- I en tom katalog, skapa filen `Vagrantfile`:

```
Vagrant.configure("2") do |config|
  config.vm.synced_folder ".", "/vagrant", type: "rsync"
  config.vm.boot_timeout = 999999
  config.ssh.insert_key = false
  config.vm.box_check_update = false
  config.vm.provider :libvirt do |libvirt|
    # Don't forget to create your storage pool
    libvirt.storage_pool_name="default"
    libvirt.driver="kvm"
    libvirt.uri="qemu:///system"
    libvirt.memory = 1024
    libvirt.graphics_type = "none"
    libvirt.cpus = 1
  end

  # If you are not using qemu, only the following is required:
  config.vm.box = "bento/ubuntu-18.04"

  config.vm.define "test_machine" do |node|
    node.vm.network :private_network, ip: "10.10.10.11"
    node.vm.network :forwarded_port, guest: 22, host: 24011, auto_correct: true
  end
end
```

2. Starta maskinen med `vagrant up`

- I nuläget har den här maskinen inget GUI. Det vill man oftast inte ha.
- Logga in på maskinen med `vagrant ssh` .

3. Provisionering gör du i Vagrantfilen:

```
config.vm.provision "shell", inline: <<-SHELL
  sudo apt-get update
  sudo apt-get install -y git curl wget gnupg ca-certificates
SHELL
```

(Du kan också låta Vagrant använda speciella verktyg för detta såsom `Puppet` eller `Ansible`)

8.2 Kommandon med Vagrant

```
vagrant init  # Create default configuration
vagrant up    # Start your VM or VMs
vagrant ssh   # Log in to your VM
vagrant halt  # Power down your VM
vagrant destroy # Destroy your VM so you can start from scratch
```

8.3 Vagrantfile

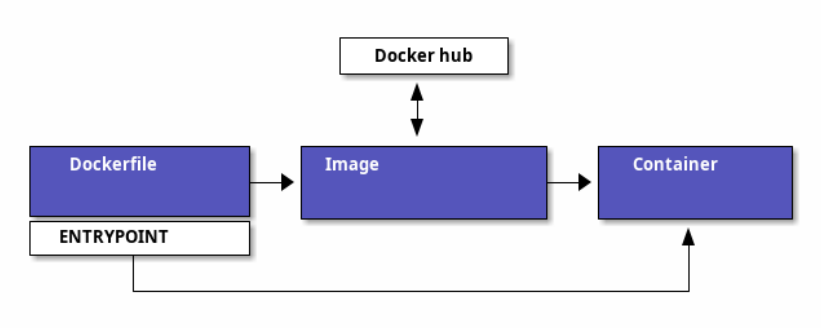
- Skriven i programspråket Ruby; du kan använda hela detta språk.
- Dokumentation: `https://www.vagrantup.com/docs`
- Några vanliga kommandon:

```
# Open up some ports between host and guest:
config.vm.network "forwarded_port", guest: 80, host: 8080, host_ip: "127.0.0.1"

# Configuration specific for a particular provider:
config.vm.provider "virtualbox" do |vb|
  vb.memory = "1024"
end

# Simple shell-based provisioning:
config.vm.provision "shell", inline: <<-SHELL
 apt-get update
 apt-get install -yy curl git
SHELL
```

9 Från Image till Container

- Lättast att komma igång med `Docker` eller `Podman`
- En `Dockerfile` beskriver vad man vill ha installerat
  - Varje kommando blir ett *lager*. För att spara plats och tid kan det här vara värt att tänka på.

- Den används för att bygga en `image`, en *mall* för hur containern skall se ut.
- från sin `image` startar man sen en eller fler `containers`, som är den körande applikationen.



9.1 Dockerfile

```
FROM node:18-alpine
RUN npm install -g nodemon
EXPOSE 3000
WORKDIR /app
COPY . .
RUN npm install
ENV DEBUG='qfapp:*'
ENTRYPOINT ["npm", "run", "dev"]
```

FROM vilken image skall man börja ifrån

RUN kör ett kommando, typiskt för att installera ytterligare program

EXPOSE den här containern kan vara tillgänglig på nätverksporten `3000`

WORKDIR När man startar containern skall den börja i den här katalogen

COPY kopiera filer från värd-datorn in till imagen

ENV sätt miljövariabler

ENTRYPOINT oftast sist i Dockerfilen. Det här är det kommando som skall köras när en container startas.

Om du kör podman så kan den här filen heta `Containerfile`

9.2 Kommandon

- `Docker` och `Podman` är i stort sett ekvivalenta. `alias docker=podman` funkar för det mesta.

```
# Help and further information
docker --help
docker -h | less
man docker
info docker

# Image management
docker build -t targetname sourcedir
docker image ls
docker image --help

# Container management
docker run
docker container ps
docker container ls
docker ps
docker container rm <container-id>
```

9.3 Docker run

- `docker run` har ett antal parametrar:

Short Parameter	Long Parameter	Description
-d	–detach	Run the container in the background without any console interaction
-p	–publish	List the ports that should be exposed from inside the container to a specific po
-e	–env	Set environment variables inside the container
-v	–volume	Mount a volume (or directory) from the host into the container
	–name	Give the running container a specific name
-i	–interactive	Run the container as an interactive console application. Usually together with
-t	–tty	Allocate a terminal (console). Usually together with -i

- Vanliga kombinationer
  - `docker run -dp 8080:3000`
  - `docker run -it`

10 Orchestration

- *Orchestration* kallar man det för när man har fler containers (eller VM) som skall arbeta tillsammans.
  - På samma vis som för Provisioning vill man komma igång automatiskt.
- `docker compose` / `podman compose` är ett sätt att starta flera containers.
  - Framför allt lokalt, för utveckling och testning
- `Kubernetes` är den mer moln-orienterade storebrodern till `compose`.

- Använder sig av konfigurationsfiler skrivna i *YAML*.
- Exempel:

```
version: "3.8"
services:
  app:
    image: qfstandalone
    ports:
      - 8080:3000
    volumes:
      - ./Containers/Version1/QFStandalone/src:/app/src
    environment:
      TEXTSTORE_HOST: textstore
  textstore:
    image: mongo
    command: --quiet --syslog
    expose:
      - "27017"
```

11 Mer om YAML syntax

```
---                          # Every new YAML node starts with three hyphens.
                             # You can often include many nodes in the same file.
Name: "Arthur Dent"          # Something with the key "Name" has the
                             # (string) value "Arthur Dent"
Age: 42                      # Integers and floats are also supported.
Active: true                 # ... as are booleans (true or false)
Friends: null                # ... and null values.
Inventory:                   # The value of the inventory key is a list.
  - Tea
  - No Tea
  - Towel
  - Babelfish
Desires: [ House, Clothes ]   # Lists can also be inlined
Address:                      # A Mapping (or a dictionary) is a set of
                              # key/value pairs, where each key is unique.
  Street: "Vogon Intergalactic Bypass #47111"
  City: "What do you mean city? We work with the whole Universe!"
  HouseNumber: 0
Contacts:                     # A list of mappings
  - Name: "Ford Prefect"
    Address: "Anywhere, really. Just wave!"
  - Name: "Trillian"
    Address: "Heart of Gold"
  - Name: "Marvin"
    Address: "Everywhere, eventually."
Appearances:                  # Inlined mappings
  - {Title: "The Hitchhiker's Guide to the Galaxy", Year: 1979}
  - {Title: "The Restaurant at the End of the Universe", Year: 1980}
```

```
  - {Title: "Life, the Universe and Everything", Year: 1982}
  - {Title: "So Long, and thanks for All the Fish", Year: 1984}
  - {Title: "Mostly Harmless", Year: 1992}
FavouritePoem: |             # Multi-line contents.
  Oh freddled gruntbuggly,
  Thy micturations are to me, (with big yawning)
  As plurdled gabbleblotchits, in midsummer morning
  On a lurgid bee,
  That mordiously hath blurted out,
  Its earted jurtles, grumbling
  Into a rancid festering confectious organ squealer.
     [drowned out by moaning and screaming]
...     # Formally, a YAML ends with three dots. Can often be left out.
```

12 Compose-filen

Services Vilka microservices (containers) skall startas, hur många, osv.

- Innehåller en lista med services, ex. "app" och "textstore"

xx/image Namnet på containern, ex. `textstore/image`

xx/ports Portar som skall öppnas från värden till gästen

xx/expose Portar som skall vara tillgängliga till andra services.

xx/volumes Filsystem som skall mappas. *Här kan man göra mycket, återkommer till det*

xx/environment Miljövariabler som skall sättas inne i containern.

xx/command Vilket kommando som skall köras

Networks Olika (privata) nätverk som skall användas. Containers behöver inte vara kopplade till samma nätverk.

Volumes Persistent lagring; dessa lever kvar från en start av systemet till nästa. Till exempel för databaser.

13 Volumes

- Kan antingen hänvisa till en särskild volym som är beskriven under `Volumes` - sektionen i din compose-fil.
  - Administreras automatiskt och internt av docker/podman
- Eller (och det här är användbart) så kan man hänvisa till en katalog på värd-datorn.
  - Kallas för en *bind mount*, och binder ihop din körande container med ditt lokala filsystem.
  - Du kan alltså utveckla lokalt
  - Containern kan startas om när du sparar; du behöver inte bygga om din `image` och starta om hela din `container`

```
version: "3.8"
services:
  app:
    image: qfapp
    ports:
      - 8080:3000
    volumes:
      - ./Containers/Version2/QFApp/src:/app/src  # This is a bind mount
    environment:
      REDIS_HOST: messagequeue
      TEXTSTORE_HOST: textstore
 textstore:
    image: mongo
    restart: always
    command: --quiet --syslog
    expose:
      - "27017"
    volumes:
      - textstore-data:/data/db                  # This mounts the volume textstore-data (
      - mongo-config:/data/configdb

volumes:
  textstore-data:
  mongo-config:
```

14 Bind Mount

- En *bind mount* gör det möjligt att mappa en viss katalog på värd-datorn till ett visst ställe i gästen
  - Vagrant: Katalogen där `Vagrantfile` ligger nås i gästen under `/vagrant`
  - Docker/Podman: Man anger en lokal katalog och en katalog i gästen.
- Speciellt användbart med tolkade språk som e.g. Node.js / JavaScript
  - `project/src` kan t.ex. mappas mot `/app/src`
- Kataloger med data i (t.ex. data om alla Ponies) kan mappas så att man kan ändra lokalt.
- Man bör undvika att ha en databas på en bind mount
  - där är det bättre att ha en volym som adminstreras av Docker/Podman

Kompilerade språk gör det krångligare

- Behöver aktivt säga till `gradle` eller `maven` att man vill bygga om projektet.
- Det här gäller även vid lokal utveckling, det gör det bara krångligare att automatisera.

15 Kompilerade eller Tolkade Programspråk

- *Kompilerade* språk vs *Interpreterade* (tolkade) språk
  - Tillkommer *Enhetstester*
- Kompilatorn kontrollerar *hela* programmet:
  - Använder du variabler som finns?
  - Använder du rätt typer i alla attribut (egentilgen en egenskap av *typade* språk, inte nödvändigtvis kompilerade)
  - Finns alla klasser tillgängliga? Når du alla paket?
  - Har du stängt alla `for`-loopar, kommit ihåg alla semikolon?
  - Returnerar alla metoder rätt saker?
- **Tolkade språk kontrollerar bara de delar som körs.**
  - Enhetstester blir *ännu* viktigare, de "motionerar" koden – där du tänkt på det.

16 Programspråk och Utvecklingsprocess

- Kompileringen tar lite tid: man vänjer sig att skriva och granska koden först, kompilera sedan.
  - Tolkad kod kan enklare utvärderas kontinuerligt.
- Kompilerad kod packas "lätt" ihop till en delbar binär. `exe` , `elf` , `lib` , `JAR`, eller en `DLL`.
  - Lätt att vänja sig vid separata utvecklinsgteam, ansvariga för sin komponent.
  - Enhetstester av varje komponent.
  - Separata repositories?
  - Kan spridas till andra team som en binär; behåller full kontroll över koden.
    - I praktiken görs detta ofta inte, man bygger om hela projektet.
    - Hel vetenskap om utvecklingsorganisationerna och vem och hur man accepterar förändringar i koden från.
- Tolkad kod
  - Lätt att ändra och direkt se förändringarna
  - Kräver mer disciplin att bara hålla sig till sin egen kod
  - Kräver mer disciplin att skriva bra testfall
  - Ofta mer lättrörlig; man behöver inte en lång kompileringscykel innan man kan testa.
    - Innebär också att man är mer benägen att testa hela tiden vilket kan sänka utvecklingsfarten.

  - Koden synlig för användaren (om de vet var de skall leta)
    * Svårare att hålla saker hemliga
    * Svårare att hindra klåfingriga användare

- Lansering av kod som *microservices*
  - Koden körs i Containers
  - Applikationen byggs upp av samarbetande containers
  - Samma idé som kompilerade binärer;
    * behåller full kontroll över koden
    * Varje team kan välja sitt eget programspråk
  - Inte lämpat för "skrivbordsapplikationer"

17 Sammanfattning

- Modulär kod == Underhållbar kod

  Modulär utveckling Metoder, Klasser, Filer, Paket, Komponenter, ...

  Modulär exekvering Komponenter, Binärer, Microservices, Applikationer, ...

- En *Virtuell Maskin* simulerar en hel dator inuti din dator.
  - Gör det möjligt att skapa en "sandlåda" för testning och utveckling
  - Gör det möjligt att ge tillgång till fler datorer än vad man fysiskt har (t.ex. Cloud Computing)
  - Gör det möjligt att dela upp en applikation i flera modulära enheter.

- En *Container* är en lättvikts-virtuell maskin.
  - Fokus på att köra en specifik komponent

- Utvecklingsprocessen påverkas bland annat av:
  - Programspråk (Kompilerat vs tolkat språk)
  - Målplatform (ex. självständig binär eller en lanserbar enhet (microservice))

18 Nästa Föreläsning: Utveckling med Microservices

- Thomas & Hunt: Kapitel 5, Bend or Break
  - Topic 28 Decoupling
  - Topic 29 Juggling the Real World
  - Topic 30 Transforming Programming
  - (Topic 31 Inheritance Tax)

  - Topic 32 Configuration
- Thomas & Hunt: Kapitel 6, Concurrency
  - Topic 33 Breaking Temporal Coupling
  - Topic 34 Shared State is Incorrect State
  - Topic 35 Actors and Processes
  - Topic 36 Blackboards
- Microservice-arkitekturer
- Kommunicerande Microservices, REST-API:er.

19 Övning: Kom igång med Containers

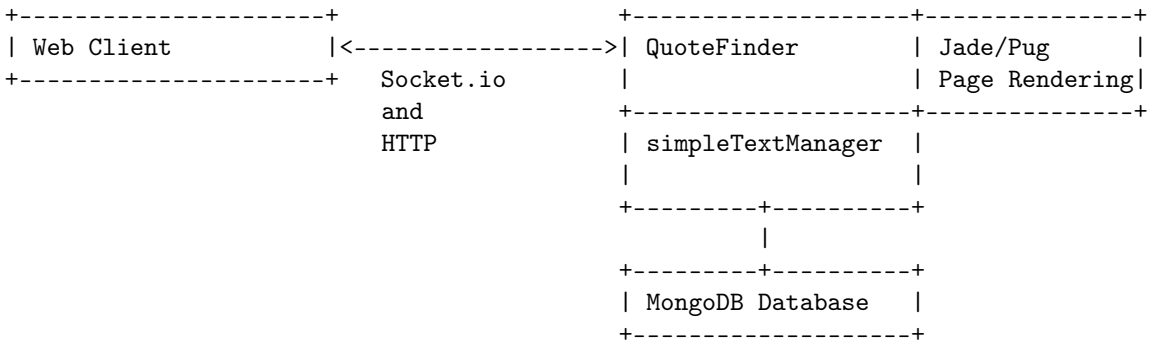
19.1 Kom igång med några tutorials

1. Docker `https://docs.docker.com/get-started/`
2. Docker with node.js `https://docs.docker.com/language/nodejs/`

19.2 Om Projektet: QuoteFinder

- Ladda ner Projektet: `https://github.com/mickesv/ProvisioningDeployment.git`
  - Är egentligen en del av en kurs i *Applied Cloud Computing and Big Data*
  - Var lugn, vi kommer inte använda allt i den här kursen.
- *QuoteFinder* letar efter citat i texter.
  - Tre versioner
    * Version 1, Letar efter en hel sträng
    * Version 2 && 3 letar efter orden nära varandra.

    Version 1 Enklare, går att köra nästan ensamt.
    * Version 2 && 3 består av ett antal kommunicerande microservices.
  - Skrivet i Javascript / node.js `https://nodejs.org/` (ett tolkat språk)
  - Express web app `http://expressjs.com/`
  - Använder också socket.io `https://socket.io/`
  - Kopplar upp sig mot en MongoDB-databas `https://www.mongodb.com/`
  - Exponerar tre websidor: `/` , `/add` , och `/list` .



19.3 Kolla koden: `Containers/Version1/QFStandalone/src/`

- Läs igenom `index.js`
  - Hur skapas en "route"?
  - Vad händer när man t.ex. begär sidan `/add` ? Vilken funktion anropas?
- **Ni behöver *inte* fördjupa er om följande**:
  - `res.render()` använder sig av *Jade/Pug* för att skapa en websida: `https://pugjs.org/`
    - Vanligt arbetssätt; "lättare" att skapa en websida med dynamiskt innehåll från något mall-system.
  - *Promises* är ett sätt att länka saker som skall hända asynkront så att de ändå sker i ordning.
    - En ledtråd är när man hittar kod `.then()`; då arbetar man troligen med en *Promise*.
- Läs igenom `simpleTextManager.js`
  - Vad gör klassen / vilka metoder / vilka ansvarsområden har den?
  - Titta lite närmre på metoden `addText()`
    - vad gör den?
    - varför tror du att den sparar texterna på det här viset?

19.4 Bygg en image

- gå till katalogen där `Dockerfile` ligger, `Containers/Version1/QFStandalone/`
- Titta på `Dockerfile`, förstår du hur den är uppbyggd och vad som kommer hända?
- Bygg en image: `docker build -t qfstandalone .`
  - Vad händer?
  - Notera hur den bygger upp lager efter lager.
- Kontrollera efteråt att den faktiskt byggdes `docker image ls`
  - Vilka fler images har du? Varför tror du att de finns där?

19.5 Starta applikationen: podman/docker

1. Applikationen använder `MongoDB`, så vi behöver hämta den: `docker pull mongo`
2. Vi behöver ett nätverk för att qfstandalone skall kunna prata med databasen:
   - `docker network create qfstandalone-net`
3. Starta databasen: `docker run -d --network qfstandalone-net --network-alias textstore --name textstore mongo`
4. Starta applikationen: `docker run -it --network qfstandalone-net -e TEXTSTORE_HOST=textstore -w /app -v ./src:/app/src --name qfstandalone -p 8080:3000 qfstandalone`

Förklaring: Starta Databasen

```
docker run                  # Start a Container
-d                          # In detached mode (in the background)
--network qfstandalone-net  # Connect to the virtual network we just created
--network-alias textstore   # Make this container accessible
                            # on the network using this name
--name textstore            # Use this name when we access
                            # the container with docker
mongo                       # Use this image as base for the container
```

Förklaring: Starta Applikationen

```
docker run                    # Start a container
-it                           # In interactive mode, and attach
                              # a terminal so we can also type into it
--network qfstandalone-net    # Same virtual network
-e TEXTSTORE_HOST=textstore   # Set the environment variable to the
                              # network alias of our MongoDB database
-w /app                       # Set the working directory inside the container
-v ./src:/app/src             # Attach the host directory ./src
                              # to the guest under /app/src
--name qfstandalone           # Container name
-p 8080:3000                  # Connect host port 8080 to
                              # port 3000 in the container
qfstandalone                  # Use this image (the tag we previously set)
```

19.6 Testa

1. Lägg till en bok, gå till: `http://localhost:8080/add`
   - Använd förslagsvis en bok från Gutenberg-projektet `https://www.gutenberg.org/`
   - Om du inte skriver in något så kommer du lägga till en textversion av Leo Tolstoy's *Krig och Fred*
2. Gå till `http://localhost:8080/` och sök efter något, till exempel 'prince'.

Att göra:

- Håll ett öga på din terminal. Vad skrivs ut? Vad händer?
- Eftersom vi startade med flaggorna `-it` så kan vi kontrollera appen i terminalen:
  - Prova skriv `rs` och tryck på `<enter>` , vad händer?
  - Det här är för att vi kör programmet med hjälp av `nodemon` : `https://nodemon.io/`
- Vi startade också programmet med en *bind mount* : `-v ./src:/app/src`
  - Öppna filen `src/index.js` och leta rätt på metoden `startPage()`
  - Byt ut return-raden mot `return listTextsPage(req, res);`
  - Vad händer i terminalen?
  - Ladda om startsidan i webläsaren; du bör nu också se en lista med alla tillgängliga texterna.

19.7 Avbryt, Stoppa, och Städa upp

- Avbryt den körande applikationen genom att trycka `Ctrl-C` i terminalen.
  - Det här stoppar den körande containern `qfstandalone`
  - Databas-containern `textstore` fortsätter köra i bakgrunden
  - Nätverket finns fortfarande tillgängligt
  - Kolla vad som finns kvar: `docker ps -a`
- Dags att rensa:

```
docker rm -f textstore qfstandalone
docker network rm qfstandalone-net
docker network prune -f
```

19.8 Starta applikationen: podman/docker compose

- Vi har redan introducerat en `docker compose` -fil för att starta applikationen.
- Öppna och studera filen `docker-compose-v1.yml`
- Starta applikationen med `docker compose -f docker-compose-v1.yml up`
- Testa som innan med `http://localhost:8080/` och `http://localhost:8080/add`

Att göra:

- Notera hur utskrifterna i terminalen skiljer sig.
- Vad händer om du skriver `rs` som innan i terminalen?

- Vad händer när du avbryter med `Ctrl-C` ? Kontrollera med `docker ps -a`

1. Överkurs: kommunicera med applikationen
   (a) Uppdatera docker compose-filen (Se nedan)
   (b) I en separat terminal, koppla på dig på den körande containern: `docker compose -f docker-compose-v1.yml attach app`

```
version: "3.8"
services:
  app:
    image: qfstandalone
    stdin_open: true # docker run -i
    tty: true        # docker run -t
    ports:
      - 8080:3000
    volumes:
      - ./Containers/Version1/QFStandalone/src:/app/src
    environment:
      TEXTSTORE_HOST: textstore
  textstore:
    image: mongo
    command: --quiet --syslog
    expose:
      - "27017"
```

19.9 Sammanfattning

1. Bygg en `Image`
2. Starta en `Container`
   - Starta endast en container åt gången
   - Starta flera containrar med ett enda kommando
3. Redigera filer lokalt och se dem ändras i en körande container

Fördelar:

- Kan köra vilka program och programspråk du vill i en container.
- Upprepningsbar deployment.

Nackdelar:

- Kan köra vilka program och programspråk du vill i en container; inklusive malware.
- Det borde vara, men är inte, helt transparent att ta nästa steg ut på molnet.
- Databasen är inte persistent än...