Linux 2

ITINF 2021

Lektion 10

Idag

- Andra sätt att virtualisera
- Docker
- Infrastructure as code
  - Ansible
  - Terraform

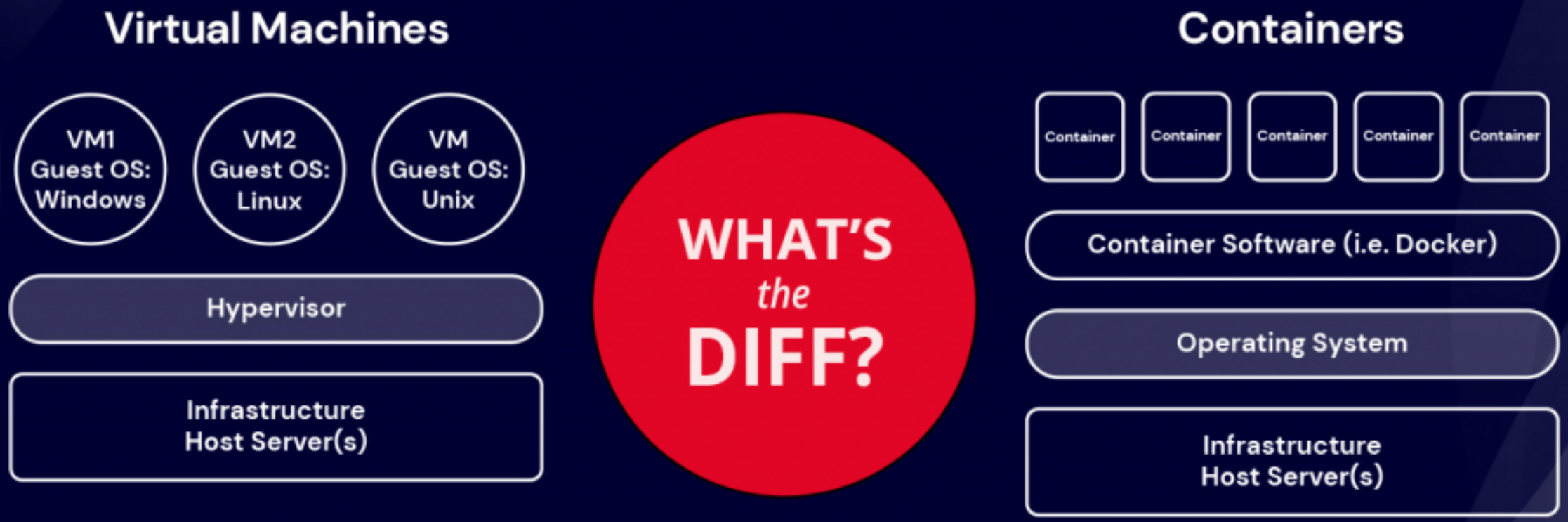
Containers

Containers

- Inte en hel virtuell dator, utan bara en viss miljö i ett virtuellt operativsystem
- Körs på ett operativsystem och delar oftast detta operativsystems kärna, samt bibliotek och en del binärer
- Exempel: Docker, LXC, Solaris Containers

Containers

> Containers may look like real computers from the point of view of programs running in them. A computer program running on an ordinary operating system can see all resources (connected devices, files and folders, network shares, CPU power, quantifiable hardware capabilities) of that computer. However, programs running inside of a container can only see the container's contents and devices assigned to the container.


Virtual Machines
VM1
Guest OS:
Windows
VM2
Guest OS:
Linux
VM
Guest OS:
Unix
Hypervisor
Infrastructure
Host Server(s)
WHAT'S
the
DIFF?
Containers
Container
Container
Container
Container
Container
Container Software (i.e. Docker)
Operating System
Infrastructure
Host Server(s)

VMs vs containers

VMs	Containers
Heavyweight.	Lightweight.
Limited performance.	Native performance.
Each VM runs in its own OS.	All containers share the host OS.
Hardware-level virtualization.	OS virtualization.
Startup time in minutes.	Startup time in milliseconds.
Allocates required memory.	Requires less memory space.
Fully isolated and hence more secure.	Process-level isolation, less secure.
Can virtualize most things (e g USB)	Can only virtualize fixed set of things

baserad på tabell från backblaze.com

Övning 1

- Ni har en fysisk server som ni vill använda till att snabbt sätta upp och köra testmiljöer för er applikation som körs under Linux.
- Lista fördelar med att köra containers respektive virtuella servrar på denna server.

Övning 1, exempel

- Containers:
  - Behöver mindre kraft för varje container än för en virtuell server
  - Går fortare att få igång testkörning för applikationer
  - Kräver mindre minne
- Virtuella servrar:
  - Tillgång till hela operativsystemet
  - Kan ha helt olika operativsystem

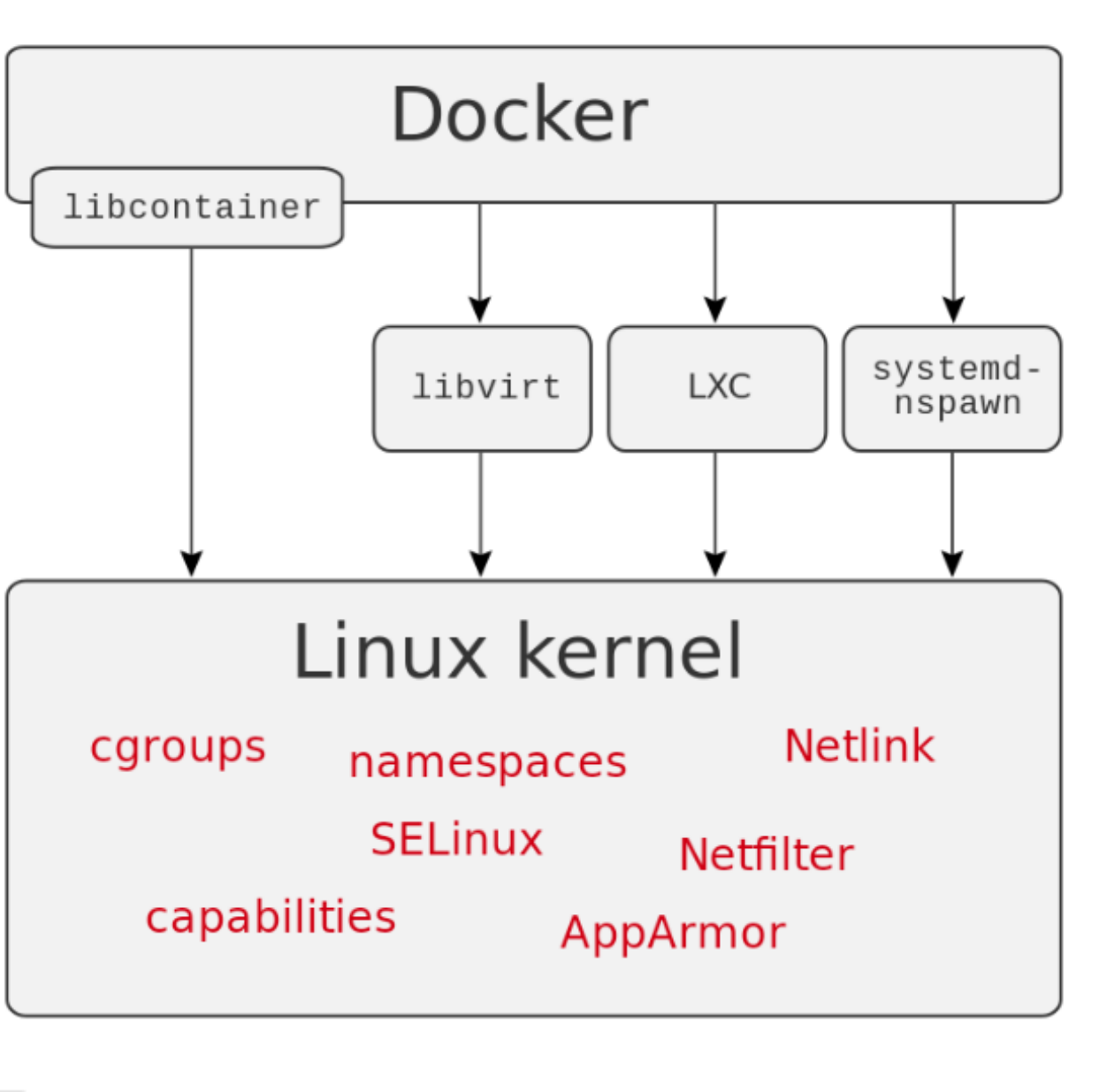
Docker

Docker

- Docker container engine
- Kan köra mjukvara i containers
- ”OS level virtualization”
- Containers är fristående och i princip isolerade från varandra
  - Specifik kommunikation mellan containrar kan konfigureras
- Alla containers kör ovanpå din existerande Linux-kernel, men kan ha olika Linux-distribution/userspace

Inuti Docker

- "`chroot` jails"
- "kernel namespaces" för att isolera varje container från andra och från host
- virtuella nätverksinterface
- etc...
- https://www.codementor.io/blog/docker-technology-5x1kilcbow



Installera Docker

- https://docs.docker.com/engine/install/ubuntu/
- https://www.youtube.com/watch?v=Vplj9b0L_1Y

Installera Docker, i korthet:

```
$ sudo apt-get update
$ sudo apt-get install ca-certificates curl gnupg lsb-release
$ sudo mkdir -p /etc/apt/keyrings
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg
$ echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/ubuntu \
  $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
$ sudo apt-get update
$ sudo apt-get install docker-ce docker-ce-cli containerd.io docker-compose-plugin
$ sudo systemctl status docker # kolla så den funkar

$ sudo docker run hello-world
```

Övning 2

- installera docker
- Kör imagen "hello world"
  `docker run hello-world`

Docker-koncept: Images

Som en disk image av en VM, eller ett apt-paket: packeterad mjukvara som går att köra som en enhet. Finns tusentals publicerade under https://hub.docker.com/search.

Heter ofta `publicerare/produkt:version`. Till exempel `bitnami/redis:7.0` är Bitnamis distribution av Redis, version 7.0.

De flesta images är baserade på en annan: `bitnami/redis` är t ex baserad på `minideb`, minimal Debian.

- `docker image ls`
- `docker image rm` eller `docker rmi`, `docker image prune`
- `docker image pull`, `docker image push`
- etc... `docker help image`

Docker-koncept: Container

Varje gång du kör igång en image så skapas en ny container, dvs en instans av denna image. T ex om du kör igång apache-imagen två gånger så får du två containrar som kör varsin webbserver.

- `docker ps` eller `docker container ls` -- vilka containrar är igång?
- `docker ps -a` vilka finns på systemet, inkl stoppade?
- `docker container attach` -- hoppa in interaktivt i en aktiv container
- `docker kill` -- stäng av en container
- etc... `docker help container`

Docker-koncept: Run

Kör igång en container från en image!

- Interaktivt: `docker run -it bash` -- starta imagen "bash" och kör den:
  - `-i`: interactive, keep stdin open
  - `-t`: allocate a pseudo-TTY (dvs tillåt tangentbords-input)
- Kör med ett kommando: `docker run -t bash echo hello`
- Kör som daemon: `docker run -d -p 80:80 httpd`
  - `-d`: Detach, receive no input nor output
  - `-p cport:hport`: Exponera containers TCP-port till hostens TCP-port
- Ta bort containern efter körning: `docker run --rm`
- etc... `docker help run`

Övning 3

Använd `bash` i en docker-container och skriv ut "hello world" med echo

Övning 3

```
docker run -t bash echo "hello world"
```

Docker, lite viktiga kommandon

- `docker run`
- `docker info`
- `docker ps`
- `docker config`
- `docker container`
- `docker images`
- `docker pull`

Docker

- Förbindelse mellan host och container
- Containernamnet hittar man med `docker ps`
- Kopiera filer till container:
  `docker cp <fil> <container>:<path>`
- Kopiera filer från container:
  `docker cp <container>:<path/fil> <fil>`

Övning 4

- Gör ett enkelt skript som räknar från 1 till 10 med en sekunds paus för varje steg
- Starta bash i en docker-container
- Kopiera scriptet till din container (lägg det t ex under /tmp)
- Kör scriptet i din container
- Observera intressanta saker kring sökvägar och att scriptet i containtern bara finns så länge som containern finns

Övning 4

```
$ cat > mycounter.sh
#!/usr/bin/env bash
for i in {1..10}
do
    echo $i
    sleep 1
done
$ docker run -it bash
$ docker cp ./mycounter.sh fa381cde7ee1:/tmp # separat terminal
```

Detta är bara ett exempel: det här är inte ett bra sätt att använda Docker. Använd inte `docker cp`.

Bättre: `Dockerfile` & `docker build`

- Dockerfile beskriver innehållet i en docker image
- Börjar nästan alltid med en `FROM` -- din bas-image
- Varje rad blir ett eget filsystem! Bara förändringar leder till ombygge.
- Byggs med `docker build`
- Läggs då till i ditt lokala image-bibliotek

`Dockerfile`, exempel 1

```
FROM alpine
CMD ["echo", "hello world!"]
```

`Dockerfile`, exempel 2

```
FROM python:3
# set a directory for the app
WORKDIR /usr/src/app
# define the port number the container should expose
EXPOSE 5000

# copy deps as a separate cache layer
COPY requirements.txt
# install dependencies
RUN pip install --no-cache-dir -r requirements.txt
# Copy rest of app
COPY *.py .

# Run this when starting the container
CMD ["python", "./app.py"]
```

`docker build`

Bygg en image från den `Dockerfile` som finns i en mapp. Anropas vanligen som:

```
docker build -t <imagenamn> <mapp som har Dockerfile>
```

e g: `sudo docker build -t helloworld .` för att bygga imagen från nuvarande mapp (`.`).

Om du inte anger version så kommer du skapa/omdefinera `latest` (dvs ovanstående blir `helloworld:latest`).

Övning 5

- Plocka hem `alpine` att använda som bas för en egen enkel image
- Testa att köra `alpine` i docker
- Skapa en Dockerfile för en image `hello` som är din egen implementation av ett ”hej världen”
- Bygg och kör din image

Övning 5

```
$ cat > Dockerfile
FROM alpine
CMD ["echo", "hello world"]
```

Övning 6

- Skapa nu en Dockerfile för en image `counter` som kör ditt räkneskript (det som räknar från 1 till 10) under `bash`
- Bygg och kör din image

Övning 6

```
$ cat > Dockerfile
FROM bash
COPY mycounter.sh .
CMD ["bash", "./mycounter.sh"]
$ sudo docker build -t counter .
$ sudo docker run counter
```

Docker: addendum

- Finns många grunduppsättningar (images) att utgå från
- Deploy av egna program genom egna images som använder sig av dessa som bas
- Väldigt snabb deploy när konfigurationen väl är gjord
- Använd layers i era images!

```
# Exempel från https://github.com/alloverse/alloplace2
FROM ubuntu:20.04 # was gcc:9, is ubuntu just for libretro

WORKDIR /alloplace2
EXPOSE 21337/udp

ENV DEBIAN_FRONTEND=noninteractive
RUN apt-get update && apt-get install -y build-essential \
    cmake curl libgme-dev libcairo2 libpoppler-glib-dev \
    retroarch libretro-nestopia libretro-genesisplusgx libretro-snes9x \
    && apt-get clean \
    && rm -rf /var/lib/apt/lists/* /tmp/* /var/tmp/*

# build and compile server with dependencies
COPY deps /alloplace2/deps
COPY src /alloplace2/src
COPY CMakeLists.txt /alloplace2

RUN cd build; cmake ..; make

CMD cd /alloplace2; ./build/alloplace2
```

IaC

IaC: Infrastructure as Code

- All infrastruktur provisioneras och konfigureras genom källkod: en uppsättning definitionsfiler.
- Genom att bara köra ett kommando så kan man skapa och konfigurera hela miljöer.
- Mycket mindre administration och lättare att maintaina flera miljöer
- Ersätter att klicka runt i webb-interface
- Exempel: Chef, Puppet, Ansible, Terraform

IaC: Infrastructure as Code

> Infrastructure as code (IaC) is the process of managing and provisioning computer data centers through machine-readable definition files, rather than physical hardware configuration or interactive configuration tools. The IT infrastructure managed by this process comprises both physical equipment, such as bare-metal servers, as well as virtual machines, and associated configuration resources.

IaC: Infrastructure as Code

- Deskriptiv modell för att hantera på infrastruktur
  - Fysiska och virtuella maskiner
  - Nätverk och nätverkstopologi
  - Lastbalansering
- Versionshantering för den uppbyggda infrastrukturen
- Egentligen samma principer som för att hantera applikationer inom DevOps

IaaS: Infrastructure as a Service

- Abstraherar tillgång till fysiska resurser
- Ger tillgång till nätverk, servrar, etc
- APIer av olika slag för att allokera ("provisionera") och använda resurser
- med andra ord: aws, gcp, azure och dylika.

> Infrastructure as a service (IaaS) are online services that provide high-level APIs used to dereference various low-level details of underlying network infrastructure like physical computing resources, location, data partitioning, scaling, security, backup etc.

IaC vs IaaS

- IaC är instruktionerna
- IaaS tar emot instruktionerna och levererar servicen

Ansible

Ansible

- Verktyg för IaC
- Konfigurationshantering
- Automatisering av återkommande uppgifter
- Öppen källkod + RedHat-moduler
- https://www.ansible.com/

Ansible

> Ansible is an open-source software provisioning, configuration management, and application-deployment tool enabling infrastructure as code. It runs on many Unix- like systems, and can configure both Unix-like systems as well as Microsoft Windows. It includes its own declarative language to describe system configuration.

Ansible

- Inventory i textfiler
- ssh som normal inloggning
- Köra kommandon på flera noder parallellt
- Playbooks för orkestrering
- I teorin deklarativt ("min backend ska se ut SÅHÄR"), i praktiken ofta imperativt ("GÖR DETHÄR med min backend")

```
$ cat inventory.ini
[webservers]
foo.example.com
bar.example.com

[dbservers]
one.example.com
two.example.com
three.example.com
```

```
$ cat playbook.yml
- name: Update web servers
  hosts: webservers
  remote_user: root

  tasks:
  - name: Ensure apache is at the latest version
    ansible.builtin.yum:
      name: httpd
      state: latest
  - name: Write the apache config file
    ansible.builtin.template:
      src: /srv/httpd.j2
      dest: /etc/httpd.conf

- name: Update db servers
  hosts: dbservers
  remote_user: root

  tasks:
  - name: Ensure postgresql is at the latest version
    ansible.builtin.yum:
      name: postgresql
      state: latest
  - name: Ensure that postgresql is started
    ansible.builtin.service:
      name: postgresql
      state: started
```

```
$ ansible-playbook playbook.yml -f 10
```

Exempel från
https://docs.ansible.com/ansible/latest/user_guide/index.html

Ansible för att provisiona AWS

Exempel från https://www.redhat.com/sysadmin/ansible-provisioning-aws-cloud, bara för att visa hur meckigt det är jämfört med Terraform ;)

```yaml
- name: Launching EC2 instances
  community.aws.ec2_instance:
    #aws_access_key: "{{ec2_access_key}}"
    #aws_secret_key: "{{ec2_secret_key}}"
    profile: "{{ aws_boto_profile }}"
    key_name: "{{ aws_demo_key }}"
    security_group: "{{ aws_security_group }}"
    instance_type: "{{ item.value.instance_type }}"
    image_id: "{{ aws_ami_id }}"
    state: present
    wait: yes
    wait_timeout: 300
    region: "{{ aws_region }}"
    tags:
       Name: "{{ item.value.name }}"
    detailed_monitoring: no
    vpc_subnet_id: "{{ vpc_subnet_list | random }}"
    network:
      assign_public_ip: yes
  loop: "{{ lookup('dict', ec2_new_list, wantlist=True) }}"
```

```yaml
- name: Create Security group
  amazon.aws.ec2_group:
    profile: "{{ aws_boto_profile }}"
    name: "{{ aws_security_group }}"
    description: 'Security Group with SSH and HTTP rules'
    vpc_id: "{{ aws_vpc_id }}"
    region: "{{ aws_region }}"
    rules:
      - proto: tcp
        ports:
        - 80
        cidr_ip: 0.0.0.0/0
        rule_desc: allow all on port 80
      - proto: tcp
        ports:
        - 22
        cidr_ip: 0.0.0.0/0
        rule_desc: allow all on port 22
```

Terraform

Terraform av HashiCorp

- Verktyg för IaC
- Ett provisioning-verktyg snarare än ett konfigurationshanterings-verktyg
  - dvs, "skapa servrar" snarare än "gör sak X på alla servrar"
  - Kan kombineras med konfigurationshantering
  - Men om man använder docker behövs knappt det
- Beskriv hur din infrastruktur ser ut, deklarativt
- https://www.terraform.io/

> Configuration management tools such as Chef, Puppet, Ansible, and SaltStack typically default to a mutable infrastructure paradigm. For example, if you tell Chef to install a new version of OpenSSL, it'll run the software update on your existing servers and the changes will happen in-place. Over time, as you apply more and more updates, each server builds up a unique history of changes. This often leads to a phenomenon known as **configuration drift**, where each server becomes slightly different than all the others, leading to subtle configuration bugs that are difficult to diagnose and nearly impossible to reproduce.

Deklarativt vs imperativt

```
- ec2:
    count: 10
    image: ami-v1
    instance_type: t2.micro
```

```
resource "aws_instance" "example" {
  count         = 10
  ami           = "ami-v1"
  instance_type = "t2.micro"
}
```

Vad händer om du ändrar count till 15? Ansible: 25, Terraform 15.

```
$ terraform plan
+ aws_instance.example.11
    ami:                          "ami-v1"
    instance_type:                "t2.micro"
+ aws_instance.example.12
    ami:                          "ami-v1"
    instance_type:                "t2.micro"
+ aws_instance.example.13
    ami:                          "ami-v1"
    instance_type:                "t2.micro"
    ...

Plan: 5 to add, 0 to change, 0 to destroy.
```

https://blog.gruntwork.io/why-we-use-terraform-and-not-chef-puppet-ansible-saltstack-or-cloudformation-7989dad2865c

Terraform: kom igång

```
$ aws configure # sätter AWS_ACCESS_KEY_ID och AWS_SECRET_ACCESS_KEY
$ mkdir myinfra; cd myinfra
$ terraform init
```

syntax:

```
resource {typ av resurs} {namn på resurs} {
    {nyckel} = {värde}
    {annan nyckel} = {annan typ}.{annat namn}.{nyckel i den resursen}
}
```

```
$ pico main.tf
provider "aws" {
  region = "us-east-2"
}
resource "aws_instance" "example" {
  ami           = "ami-0c55b159cbfafe1f0"
  instance_type = "t2.micro"
  security_group = aws_security_group.example.id
}
resource "aws_security_group" "example" {
  ingress {
    from_port = 80
    to_port = 80
    cidr_blocks = ["0.0.0.0/0"]
  }
  egress {
    from_port = 0
    to_port = 65535
    cidr_blocks = ["0.0.0.0/0"]
  }
}
$ terraform apply
```

```
$ terraform apply
Terraform will perform the following actions:
  # aws_instance.example will be created
  + resource "aws_instance" "example" {
      + ami                          = "ami-0c55b159cbfafe1f0"
      + arn                          = (known after apply)
...
      (...)
  }

  Plan: 1 to add, 0 to change, 0 to destroy.

  Do you want to perform these actions?

  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.
  Enter a value: yes

aws_instance.example: Creating…
aws_instance.example: Creation complete after 38s

Apply complete!
Resources: 1 added, 0 changed, 0 destroyed.
```

Terraform: addendum

- Versionshantera dina tf-filer i Git
- Men använd "terraform backend s3" så att state-filen ligger i S3, inte i Git
- `terraform taint` för att tvinga omskapande = uppdatera till senaste versionen (dvs fuska och låtsas att terraform är konfigurationshantering)
- finns många sätt att skapa miljöer (workspaces, moduler, etc) -- se i demo av `allo-infra`

Tillbakablick, reflektion, kommentarer ...