Mockexamen

Vraag 1

Welk van de volgende is de beste manier om root te worden?

De Opties:

1. sudo su
2. su --login
3. su
4. sudo su --login

Het Juiste Antwoord:

sudo su --login

Waarom zijn de andere opties minder goed (fout)?

Optie 3: su (Slecht)

- **Wat het doet:** Het probeert te switchen naar de "Substitute User" (standaard root).
- **Waarom fout:**
    1. Je hebt het **wachtwoord van de root-gebruiker** nodig. In veel moderne systemen (zoals Ubuntu) heeft root geen wachtwoord en is dit account vergrendeld.
    2. Het laadt de **omgeving van root niet**. Je blijft in jouw eigen mappen staan en behoudt jouw eigen instellingen (PATH-variabelen), wat voor fouten kan zorgen bij het uitvoeren van commando's.

Optie 2: su --login (Beter, maar niet best)

- **Wat het doet:** Het switcht naar root en laadt wél de volledige omgeving van de root-gebruiker (alsof je echt inlogt).
- **Waarom fout:** Je hebt nog steeds het **root-wachtwoord** nodig. In een zakelijke omgeving wil je niet dat iedereen het master-wachtwoord kent. Het is veiliger om je eigen wachtwoord te gebruiken via sudo.

Optie 1: sudo su (Lui / Risicovol)

- **Wat het doet:** Je gebruikt sudo (jouw eigen wachtwoord) om het commando su uit te voeren.
- **Waarom fout:** Het is een halve oplossing. Je wordt wel root met je eigen wachtwoord, maar je laadt de **omgeving van root niet volledig in**. Je neemt "rommel" mee uit je eigen gebruikerssessie (zoals je HOME-directory), wat configuratiebestanden van root in jouw eigen map kan aanmaken of rechten in de war kan sturen.

Waarom is sudo su --login het juiste antwoord?

Dit commando combineert het beste van twee werelden:

1. **Beveiliging (sudo):** Je gebruikt je **eigen wachtwoord**. Het systeem logt ook *wie* er root is geworden. Dit is cruciaal voor de veiligheid (audit trail).
2. **Correctheid (--login):** De vlag --login (of het korte minteken -) zorgt ervoor dat de hele omgeving wordt schoongemaakt en opnieuw wordt opgebouwd voor root.
    - Je komt in /root terecht (de juiste thuismap).
    - De PATH variabele wordt aangepast zodat je direct bij alle beheer-tools kunt.
    - Er blijven geen resten achter van je normale gebruikersaccount die processen kunnen verstoren.

Extra tip voor je examen: In de praktijk gebruiken Linux-admins vaak het commando **sudo -i**. Dit doet exact hetzelfde als sudo su --login maar is korter om te typen. Als die optie er niet staat, is sudo su --login (of sudo su -) altijd de meest professionele en veilige keuze.

Vraag 2

Het volgende is een functie in een C++ applicatie die 24/7 moet draaien om sensordata te verwerken. Ze wordt ongeveer elke seconde opgeroepen door een bovenliggende simpele loop.

Code:

codeC++

```
void ProcessSensorData() {
  // Een grote dataset laden voor analyse
  SensorData* data = new SensorData();
  data->LoadValues();

  // Berekeningen uitvoeren
  int result = data->calcAverage();

  // Resultaat loggen (data variabele wordt hierna niet meer gebruikt)
  logResult(result);
}
```

Gebruikers klagen dat de server na ongeveer 3 dagen vastloopt (crasht) door een gebrek aan RAM-geheugen.

Wat is de technische oorzaak van de crash en in welk geheugengebied vindt dit plaats?

De Opties

1. **Dangling Pointer:** De pointer data wordt verwijderd zodra de functie eindigt, waardoor het besturingssysteem het geheugen op de Heap niet meer kan terugvinden (corruptie), maar het geheugen is wel vrij.
2. **Stack Fragmentation:** De int result wordt op de Stack opgeslagen. Omdat de functie zo vaak wordt aangeroepen, raakt de Stack gefragmenteerd en kan er geen nieuwe int meer bij.
3. **Stack Overflow:** De variabele data wordt op de Stack aangemaakt, maar omdat de functie recursief wordt aangeroepen, raakt de Stack vol voordat de variabelen worden opgeruimd.
4. **Heap Memory Leak:** Het object waar data naar wijst wordt op de Heap gealloceerd, maar nooit vrijgegeven (met delete). Hierdoor blijft het geheugen bezet, zelfs nadat de functie is afgelopen.

Het Juiste Antwoord

Optie 4: Heap Memory Leak

Volledige Uitleg

1. Waarom op de Heap?

In je cursus (Slide 14) zie je dat de **Process Address Space** is verdeeld in onder andere de **Stack** en de **Heap**.

- De **Stack** is voor lokale variabelen. Deze worden *automatisch* verwijderd als de functie klaar is.
- De **Heap** is voor dynamisch geheugen. In C++ herken je dit aan het sleutelwoord **new**. Geheugen op de Heap wordt **nooit** automatisch opgeruimd. De programmeur moet dit zelf doen.

2. Wat gaat er mis in de code?

In de regel SensorData* data = new SensorData(); gebeuren er twee dingen:

- Er wordt een klein beetje ruimte op de **Stack** gebruikt voor de pointer (het adres).
- Er wordt een **groot blok ruimte op de Heap** gereserveerd voor de eigenlijke sensordata.

Aan het einde van de functie (}) gebeurt het volgende:

- De functie stopt. De Stack wordt automatisch opgeschoond. De pointer data verdwijnt dus.

- **MAAR:** Omdat er geen delete data; in de code staat, blijft dat grote blok geheugen op de **Heap** gewoon gereserveerd staan.

3. Waarom duurt het 3 dagen?

De functie wordt elke seconde aangeroepen.

- Seconde 1: 1 MB lekt weg op de Heap.
- Seconde 2: Weer 1 MB lekt weg.
- Na 3 dagen (259.200 seconden) is er zoveel geheugen "bezet" door oude data die nergens meer voor gebruikt wordt, dat het RAM-geheugen van de server fysiek op is.

4. Waarom zijn de andere antwoorden onlogisch?

- **Dangling Pointer:** Een dangling pointer is een "loshangende" verwijzing naar geheugen dat al is vrijgegeven. Hier wordt het geheugen juist *nooit* vrijgegeven.
- **Stack Fragmentation:** De Stack is een LIFO-structuur (Last In, First Out). Die raakt niet gefragmenteerd zoals een harde schijf dat doet. Bovendien wordt de Stack elke seconde weer helemaal leeggemaakt als de functie stopt.
- **Stack Overflow:** Dit gebeurt alleen als functies elkaar blijven aanroepen zonder te stoppen (recursie). Hier roept de functie zichzelf niet aan; hij wordt aangeroepen vanuit een loop. De Stack loopt dus elke seconde vol en wordt direct weer leeggegooid. Geen probleem dus voor de Stack.

Conclusie voor het examen:
Zodra je **new** ziet in C++, zoek dan naar de bijbehorende **delete**. Ontbreekt de delete en loopt het geheugen vol? Dan is het een **Memory Leak op de Heap**.

Vraag 3

Je gebruikt een nginx webserver. Een junior developer heeft handmatig een configuratiebestand genaamd app_config.json geüpload naar de map /var/www/html/api/. De webapplicatie, die draait onder de gebruiker www-data, probeert dit bestand te updaten maar crasht met een **"Permission Denied"** foutmelding.

Je inspecteert het bestand met ls -l en ziet de volgende output:
-rw-r--r-- 1 root root 2048 Dec 6 14:00 app_config.json

Je wilt dit probleem oplossen door het **eigendom (ownership)** aan te passen, waarbij je de **minst benodigde rechten** wilt geven. Welk commando voer je uit?

De Opties

1. chown www-data:www-data /var/www/html/api/app_config.json

2. usermod -aG root www-data
3. chgrp www-data /var/www/html/api/app_config.json
4. chmod 777 /var/www/html/api/app_config.json

Het Juiste Antwoord

Optie 1: chown www-data:www-data /var/www/html/api/app_config.json

Analyse van de antwoorden

Waarom is Optie 1 juist?

- **De betekenis:** chown staat voor *Change Owner*. Met www-data:www-data maak je de gebruiker www-data zowel de eigenaar als de groep van het bestand.
- **De rechten:** In de output -rw-r--r-- zie je dat de **eigenaar** (het eerste stukje rw-) lees- en schrijfrechten heeft. Door www-data de eigenaar te maken, krijgt de webapp eindelijk de rechten om het bestand te updaten (schrijven).
- **Least Necessary Permissions:** Dit is de veiligste manier. Alleen de webserver-gebruiker krijgt toegang, de rest van het systeem niet.

Waarom is Optie 2 fout? (usermod -aG root www-data)

- **Wat het doet:** Dit voegt de webserver-gebruiker toe aan de root-groep.
- **Waarom fout:**
    1. De groep heeft momenteel alleen r-- (read-only) rechten op het bestand. De app zou dus nog steeds niet kunnen schrijven.
    2. Het is een **enorm beveiligingsrisico** om een webserver-gebruiker lid te maken van de root-groep. Dit is precies het tegenovergestelde van "minst benodigde rechten".

Waarom is Optie 3 fout? (chgrp www-data ...)

- **Wat het doet:** Dit verandert alleen de **groep** van het bestand naar www-data.
- **Waarom fout:** De eigenaar blijft root. De groep-rechten op het bestand zijn r-- (alleen lezen). De webapp (die lid is van de groep www-data) zou het bestand dus nog steeds niet kunnen updaten/schrijven.

Waarom is Optie 4 fout? (chmod 777 ...)

- **Wat het doet:** Dit geeft **iedereen** op het systeem lees-, schrijf- en uitvoerrechten op het bestand.
- **Waarom fout:**
    1. De vraag vraagt specifiek om het **eigendom (ownership)** aan te passen, niet de permissies (chmod).

2. 777 is de "nucleaire optie" en wordt in de IT gezien als zeer slecht beheer. Het is onveilig omdat elke andere gebruiker of elk ander proces op de server nu je configuratiebestand kan aanpassen of verwijderen.

Belangrijk voor je examen (De ls -l output lezen):

Kijk naar -rw-r--r--:

- rw- (Eigenaar): Mag lezen en schrijven. (Staat nu op root)
- r-- (Groep): Mag alleen lezen. (Staat nu op root)
- r-- (Anderen): Mogen alleen lezen. (Dit is de reden waarom de app nu wel kan lezen, maar niet kan schrijven).

Door de **eigenaar** te veranderen naar de gebruiker van de app, los je het probleem op de meest correcte manier op.

Vraag 4

Je hebt een reeds draaiende Docker-container genaamd payment-service. Deze applicatie leest bij het opstarten een environment variabele MAX_RETRIES uit om te bepalen hoe vaak een mislukte betaling opnieuw geprobeerd mag worden. Er is een incident en je wilt deze waarde direct verhogen zonder de code aan te passen. De container is momenteel "Up" (actief).

Welke van de onderstaande acties zal **NIET** werken om de variabele binnen de huidige, draaiende applicatie container aan te passen?

De Opties

1. **De Dockerfile aanpassen** door ENV MAX_RETRIES=5 toe te voegen, het image opnieuw bouwen (build) en de container opnieuw deployen.
2. **In de docker-compose.yml** de waarde onder environment: aanpassen en vervolgens docker-compose up -d uitvoeren.
3. **Een interactieve shell openen** in de container met docker exec -it payment-service bash, daar het commando export MAX_RETRIES=5 uitvoeren en de shell weer sluiten.
4. **De container stoppen, verwijderen en opnieuw starten** met het commando: docker run -e MAX_RETRIES=5 payment-service.

Het Juiste Antwoord (de actie die NIET werkt)

Optie 3: Een interactieve shell openen in de container met docker exec ... export ....

Waarom is Optie 3 het juiste antwoord (en werkt het dus NIET)?

Er zijn twee technische redenen waarom dit mislukt:

1. **Scope van variabelen:** Wanneer je docker exec gebruikt, start je een **nieuw, apart proces** (een bash-shell) binnen de container. Het commando export stelt de variabele alleen in voor *die specifieke shell-sessie*. Zodra je de shell sluit, is de variabele weg. Het heeft **geen enkele invloed** op het hoofdproces (de applicatie) dat al draaide.
2. **Uitlezen bij opstarten:** De vraag vermeldt specifiek dat de applicatie de variabele uitleest **bij het opstarten**. Zelfs als je de variabele op magische wijze in het geheugen van de container zou kunnen veranderen, heeft de applicatie zijn eigen kopie al gemaakt toen hij opstartte. Je moet de applicatie dus herstarten om de nieuwe waarde te zien.

Waarom werken de andere opties wel?

Deze opties werken allemaal omdat ze de container **vervangen of herstarten**, waardoor de applicatie gedwongen wordt de variabele opnieuw uit te lezen:

- **Optie 1:** Je maakt een nieuw "recept" (image). Door opnieuw te deployen, gooit Docker de oude container weg en start een nieuwe met de ingebouwde waarde.
- **Optie 2:** docker-compose is slim. Als je de .yml aanpast en up -d typt, ziet Docker dat de configuratie is veranderd. Hij stopt de oude container en start automatisch een nieuwe met de nieuwe instelling.
- **Optie 4:** Dit is de handmatige manier van optie 2. Je verwijdert de oude "bubbel" en start een nieuwe met de -e (environment) vlag.

Vraag 5

Je hebt een applicatie die bestaat uit een Python backend en een Redis database. Je gebruikt **Docker Compose** om deze lokaal te draaien. Je docker-compose.yml ziet er als volgt uit (vereenvoudigd):

```
services:
  my-backend:
    image: python:3.9
    ports:
      - "5000:5000"

  redis-cache:
    image: redis:alpine
    ports:
      - "6379:6379"
```

Je start de omgeving met docker-compose up. De Python-code in de my-backend container probeert intern verbinding te maken met Redis via de hostname **localhost** op poort 6379.

Echter, de applicatie crasht met de foutmelding:
Error: Connection refused causing failure to connect to localhost:6379.

Waarom werkt de verbinding niet en wat is de **juiste** oplossing om dit in een productie-ready situatie te fixen?

De Opties

1. Binnen een container verwijst localhost (127.0.0.1) altijd naar de container zelf. Omdat Redis in een *andere* container draait, moet je de servicenaam redis-cache als hostname gebruiken.
2. Docker containers hebben geen statische IP-adressen. Je moet het commando docker inspect gebruiken om het huidige interne IP van Redis te vinden en dit hardcoden in je Python applicatie.
3. De firewall van de backend-container blokkeert uitgaand verkeer. Je moet de netwerkmodus van de backend aanpassen naar network_mode: host om de netwerkstack van de laptop te delen.
4. De poort 6379 is wel gemapt naar de host, maar Docker containers kunnen standaard niet met elkaar praten. Je moet handmatig links: - redis-cache toevoegen aan de backend service in de YAML.

Het Juiste Antwoord

Optie 1: Binnen een container verwijst localhost altijd naar de container zelf. Gebruik de servicenaam redis-cache als hostname.

Uitgebreide Uitleg

1. Het probleem met localhost

Elke container is een **geïsoleerd proces** met zijn eigen netwerkstack.

- Wanneer de Python-app in my-backend zoekt naar localhost, kijkt hij in zijn eigen "bubbel".
- Omdat Redis niet *in* de Python-container draait, maar in zijn eigen container, vindt de app daar niets op poort 6379.

2. De oplossing: Docker DNS (Service Discovery)

Wanneer je Docker Compose gebruikt, maakt Docker automatisch een virtueel netwerk aan waar alle containers in de file lid van zijn. Docker draait in dat netwerk een eigen **DNS-server**.

- In plaats van een IP-adres of localhost, kun je de **naam van de service** uit de YAML-file gebruiken (in dit geval redis-cache).
- Docker vertaalt die naam automatisch naar het juiste interne IP-adres van de Redis-container. Dit is de schoonste en meest robuuste manier om containers met elkaar te laten praten.

3. Waarom de andere opties fout zijn:

- **Optie 2 (Hardcoden):** IP-adressen in Docker zijn dynamisch. Als je de container herstart, krijgt hij vaak een nieuw IP. Hardcoden zorgt ervoor dat je app na één herstart alweer stuk is.
- **Optie 3 (network_mode: host):** Dit heft de isolatie van de container op. Het is onveilig en wordt in productie-omgevingen bijna nooit gedaan voor web-applicaties. Bovendien lost het het fundamentele probleem van container-communicatie niet op de juiste manier op.
- **Optie 4 (links):** De links parameter is "deprecated" (verouderd). Sinds Docker netwerken heeft geïntroduceerd, kunnen containers op hetzelfde netwerk elkaar automatisch vinden via hun servicenaam. Je hoeft dit niet meer handmatig te koppelen.

Samenvatting voor je examen:
In een Docker-omgeving is localhost nooit de manier om een andere container te bereiken. Gebruik altijd de **servicenaam** die in de docker-compose.yml staat gedefinieerd.

Vraag 6

Wanneer ze zeggen 'slechter is beter' over moderne besturingssystemen die het UNIX-principe volgen, verwijzen ze naar het feit dat:

De Opties:

1. **Monolithische kernels zijn beter dan microkernels**
2. **Microkernels kernels zijn beter dan monolithcal**
3. **Microkernels zijn conceptueel beter, maar monolithische kernels zijn gemakkelijker in gebruik en onderhoud.**
4. **Het is over het algemeen prima dat gebruikersapplicaties in de kernelmodus draaien.**

Het Juiste Antwoord:

Optie 3: Microkernels zijn conceptueel beter, maar monolithische kernels zijn gemakkelijker in gebruik en onderhoud.

Uitgebreide Uitleg

Deze vraag gaat over een beroemde discussie in de informatica over hoe je het "hart" (de kernel) van een besturingssysteem ontwerpt. Op **Slide 55 van Lecture 1 ("Worse is better")** zie je de twee types naast elkaar staan.

1. Waarom is een Microkernel "conceptueel beter"?

- **Structuur:** Zoals je op de rechterkant van de slide ziet, haalt een microkernel bijna alles (drivers, bestandssysteem, netwerk) uit de gevaarlijke "Kernel Mode" en zet dit in de veiligere "User Mode".
- **Veiligheid:** Als een driver voor je printer crasht, crasht alleen dat proces en niet je hele computer. Dit is een veel eleganter en veiliger ontwerp.

2. Waarom is een Monolithische Kernel (zoals Linux) dan toch de standaard?

- **Eenvoud van implementatie:** Alles zit in één groot blok (zoals links op de slide te zien is). Programmeurs vinden dit makkelijker omdat ze niet hoeven na te denken over hoe verschillende onderdelen met elkaar praten via ingewikkelde lijntjes (IPC).
- **Snelheid (Performance):** Omdat alles in hetzelfde geheugenblok zit, kan de computer taken veel sneller uitvoeren. Bij een microkernel moet de processor constant "schakelen" tussen User Mode en Kernel Mode, wat veel tijd kost.

3. Wat betekent "Worse is Better"?

Dit is een filosofie die zegt: **Een systeem dat "slechter" is ontworpen (slordiger, alles-in-één), maar gemakkelijker te bouwen en sneller is, zal in de echte wereld altijd winnen van een "perfect" ontworpen systeem dat te ingewikkeld is.**

- De monolithische kernel is de "worse" (slordige) aanpak die in de praktijk "better" (succesvoller) bleek te zijn.

Waarom zijn de andere opties fout?

- **Optie 1 & 2:** Deze zijn te simpel. Er is niet één type "beter"; het gaat om de afweging tussen veiligheid en snelheid.
- **Optie 4 (Gevaarlijk):** Dit is een absolute "no-go" in de informatica. Als gebruikersapplicaties (zoals je browser) in **Kernel Mode (Ring 0)** zouden draaien, zou elk foutje in je browser je hele hardware kunnen slopen of je privacy volledig te grabbel gooien. Zoals we zagen op **Slide 37**, is de scheiding tussen User Mode en Kernel Mode de belangrijkste beveiliging van een computer.

Vraag 7

Een cloudserviceportaal stelt u in staat:

De Opties:

1. **Taken te automatiseren**
   (Automate Tasks)
2. **De netwerk-traffic te balanceren**
   (Load balance your traffic)
3. **Bied een kader om uw klanten te ondersteunen**
   (Provide a framework to support your customers)
4. **Beheer accounts en machtigingen van je applicatie**
   (Manage Accounts & Permissions of your application)

De Juiste Antwoorden

- **Optie 1: Taken te automatiseren**
- **Optie 4: Beheer accounts en machtigingen van je applicatie**

Uitgebreide Uitleg

1. Taken te automatiseren (JUIST)

- **Uitleg:** Een van de grootste voordelen van de cloud is automatisering. Via het portaal kun je scripts instellen, automatische back-ups plannen, of "auto-scaling" configureren (waarbij de cloud automatisch servers bijplaatst als het druk is). Dit bespaart tijd en voorkomt menselijke fouten.

2. De netwerk-traffic te balanceren (FOUT)

- **Uitleg:** Het gaat om het verschil tussen **beheer** en **uitvoering**.

  De logica van het platform: Een cloudserviceportaal is een **beheertool**. In het portaal kun je een account **beheren** (Optie 4). In het portaal kun je taken **automatiseren** (Optie 1). Maar het portaal zelf **balanceert geen verkeer**. Dat doet een specifiek hardware- of software-onderdeel: de **Load Balancer**.

  Het verschil:
  Je gebruikt het portaal om een Load Balancer te *configureren*, maar het is niet het portaal dat het verkeer daadwerkelijk staat te verdelen tussen servers. Het portaal is slechts de website waar je op knopjes drukt; de infrastructuur op de achtergrond doet het zware werk.

3. Bied een kader om uw klanten te ondersteunen (FOUT)

- **Uitleg:** Dit is een instinker. Het bieden van "klantenondersteuning" is een taak voor een helpdesk-tool of een CRM-systeem (zoals Zendesk of Salesforce). Hoewel je zulke software *op* een cloudserver kunt installeren, is het portaal van de cloudprovider zelf (zoals AWS, Azure of Google Cloud) daar niet voor bedoeld. Dat portaal is puur voor het beheren van de **technische middelen** (servers, opslag, netwerk).

4. Beheer accounts en machtigingen van je applicatie (JUIST)

- **Uitleg:** Dit verwijst naar **IAM (Identity and Access Management)**. Beveiliging is een kerntaak van elk beheerplatform. In het portaal bepaal je wie toegang heeft tot de infrastructuur en welke rechten die personen hebben (mogen ze servers alleen bekijken, of ook verwijderen?). Je beheert hier de "machtigingen" (permissions) van je hele cloud-omgeving.

Vraag 8

Stel je een Windows Server 2019-machine voor die is geconfigureerd als een webserver. Het serveert zijn inhoud via HTTPS en gebruikt een X.509-certificaat. In mijn browser wil ik dit certificaat verifiëren (zie afbeelding). Welke van de volgende uitspraken is correct:

(Imagine a Windows Server 2019 machine that is configured as a webserver. It serves its contents over HTTPS and uses a X.509 certificate. In my browser I want to verify this certificate (see image). Which of the following statements is correct:)

De Opties

1. **de uitgever van het certificaat**
   (the issuer of the certificate)
2. **De openbare sleutel van de Windows-servermachine wordt samen met dezelfde openbare sleutel, versleuteld met de privésleutel van de CA, in het certificaat opgeslagen.**
3. **The public key of the windows server machine is stored in the certificate, together with the public key of the CA**
4. **De openbare sleutel van de Windows-servermachine wordt opgeslagen in het certificaat**
   (The public key of the windows server machine is stored in the certificate)

Analyse en Antwoorden

Optie 1: de uitgever van het certificaat (JUIST)

- **Uitleg:** Bij het verifiëren van een certificaat is de eerste stap kijken wie de uitgever is (in dit geval **Amazon**).

Optie 2: De openbare sleutel... versleuteld met de privésleutel van de CA... (JUIST)

- **Uitleg:** Dit beschrijft hoe de **digitale handtekening** op het certificaat werkt. De CA (Amazon) neemt de informatie (inclusief de openbare sleutel van de server) en "ondertekent" deze door het te versleutelen met hun eigen **privésleutel**. Hierdoor weet de browser dat de gegevens niet zijn aangepast.

Optie 3: ...together with the public key of the CA (FOUT)

- **Uitleg:** Dit is een instinker. De openbare sleutel van de **CA (de uitgever)** zit **niet** in het certificaat van de server zelf. De openbare sleutel van de CA zit al in je browser of je besturingssysteem opgeslagen (het "Trusted Root" archief). Het certificaat van de server bevat alleen de openbare sleutel van de server zelf.

Optie 4: De openbare sleutel van de Windows-servermachine wordt opgeslagen in het certificaat (JUIST)

- **Uitleg:** Dit is de meest fundamentele waarheid. Een certificaat is in feite een officieel document dat zegt: "Deze openbare sleutel hoort bij deze servernaam". De openbare sleutel van de server *moet* dus in het bestand staan.

Vraag 9

Welke kosten worden geëlimineerd wanneer je overschakelt van IAAS naar PAAS?
(What costs are eliminated when you switch from IAAS to PAAS)

De Opties:

1. **OS installation and management**
2. **application management**
3. **development**
4. **middleware management**

De Juiste Antwoorden

- **Optie 1: OS installation and management**
- **Optie 4: middleware management**

Uitgebreide Uitleg

Om dit te begrijpen, moet je kijken naar wie verantwoordelijk is voor welke "laag" in de cloud.

1. IAAS (Infrastructure as a Service)

Bij IaaS huur je alleen de "kale" machine (virtueel).

- De provider regelt de hardware en de virtualisatie (de onderste lagen).
- **Jij** moet zelf het Besturingssysteem (**OS**) installeren, patchen en beveiligen.
- **Jij** moet zelf de **Middleware** (zoals webservers, databases of runtimes zoals Java/Python) installeren en beheren.

2. PAAS (Platform as a Service)

Bij Paas huur je een kant-en-klaar platform.

- De provider regelt nu ook het **Besturingssysteem**. Jij hoeft dus geen updates meer te doen voor Windows of Linux. (**Optie 1 wordt geëlimineerd**).
- De provider regelt de **Middleware**. Er staat al een database of webserver voor je klaar die automatisch wordt beheerd. (**Optie 4 wordt geëlimineerd**).
- **Jij** hoeft alleen nog maar je eigen code (applicatie) en data te uploaden.

Waarom zijn de andere opties fout?

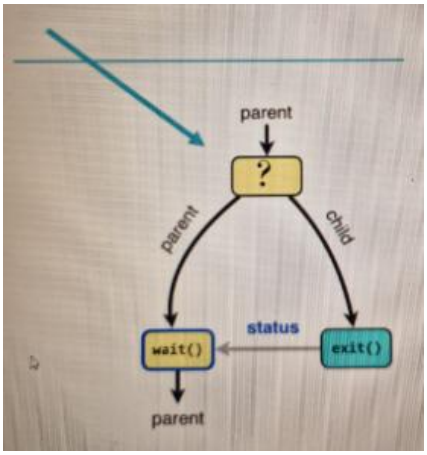
- **Application management (Optie 2):** Ook bij PaaS ben jij nog steeds verantwoordelijk voor je eigen applicatie. Als je app een bug heeft of moet worden geüpdatet, moet jij dat doen. Deze kosten blijven dus bestaan.
- **Development (Optie 3):** Je zult altijd programmeurs nodig hebben om de applicatie te bouwen, of je nu IaaS, PaaS of zelfs je eigen fysieke server gebruikt. Ontwikkelingskosten verdwijnen nooit door een overstap naar de cloud.

Samenvatting voor je examen:

- **IaaS:** Jij beheert het OS en alles daarboven.
- **PaaS:** De provider beheert het OS en de Middleware; jij beheert alleen de App.
- **SaaS:** De provider beheert alles.

Vraag 10

Welke system call/function zou op het vraagteken gezet moeten worden in de onderstaande figuur?



De Opties:

1. exit()
2. fork()
3. pthread_join()
4. pthread_create

Waarom is fork() het juiste antwoord?

Dit diagram is de klassieke weergave van **procesbeheer** in Unix-achtige besturingssystemen (zoals Linux).

1. **De Splitsing:** Het commando **fork()** wordt door een proces (de **parent**) aangeroepen. Op dat exacte moment kopieert het Besturingssysteem het proces. Er zijn nu twee identieke processen: de parent en de **child**.
2. **Het pad van de Parent:** In het diagram zie je dat de parent na de splitsing naar **wait()** gaat. Dit is een system call waarmee de vader-proces pauzeert totdat zijn kind klaar is.
3. **Het pad van de Child:** Het kind-proces voert zijn eigen taak uit en eindigt met **exit()**.
4. **De Status:** Wanneer het kind exit() aanroept, stuurt het een **statuscode** (bijv. "0" voor succes) terug naar de wait() functie van de ouder. Pas dan gaat de ouder weer verder met zijn eigen werk.

Waarom zijn de andere opties fout?

- **exit():** Dit is het *einde* van een proces, niet het begin van een splitsing.
- **pthread_create:** Dit wordt gebruikt om **threads** aan te maken, geen processen. Hoewel het principe lijkt op een splitsing, gebruikt men bij threads meestal niet de termen "parent" en "child" in combinatie met de specifieke wait() en exit() functies zoals ze hier getekend zijn.
- **pthread_join():** Dit is de "thread-versie" van wait(). Het wordt gebruikt om te wachten op een thread, niet om er een aan te maken.

Vraag 11

Welk van de volgende is waar over VMDK files in ESXi?
(Which statement is true about VMDK files in ESXi?)

De Opties:

1. **Een virtuele machine bestaat uit twee bestanden: een VMDK-bestand met de VM-gegevens en een VMC-bestand met de configuratiegegevens.**
   (A VM consists out of two files: a VMDK, which holds the VM data and a VMC which holds the configuration data.)

2. **Alle VMDK files zijn binaire files**
   (All VMDK files are binary files.)
3. **VMDK-bestanden bestaan eigenlijk uit twee bestanden: een gewoon *.vmdk-bestand en een *-flat.vmdk-bestand, maar de webinterface verbergt dit voor ons.**
   *(VMDK files actually consist of two files: a regular *.vmdk and a *-flat.vmdk, however the web GUI hides this for us.)*
4. **VMDK files hebben alle data inclusief configuratie. Ze zijn dan ook altijd meerdere gigabytes groot**
   (VMDK files hold all data including configuration: they are always several gigabytes in size.)

Het Juiste Antwoord

Optie 3: VMDK-bestanden bestaan eigenlijk uit twee bestanden: een gewoon *.vmdk-bestand en een *-flat.vmdk-bestand, maar de webinterface verbergt dit voor ons.

Uitgebreide Uitleg

Wanneer je met VMware ESXi werkt, ziet een virtuele harde schijf er voor de gebruiker uit als één bestand. Maar "onder de motorkap" zijn het er twee:

1. **De Descriptor File (.vmdk):** Dit is een heel klein **tekstbestand** (geen binair bestand!). Je kunt dit openen met Kladblok. Het bevat de "inhoudsopgave" van de schijf: hoe groot is hij, welke hardware wordt gesimuleerd, en waar staat de eigenlijke data?
2. **De Data File (-flat.vmdk):** Dit is het grote **binaire bestand**. Hierin staan de daadwerkelijke enen en nullen van de virtuele harde schijf. Dit bestand is vaak vele gigabytes groot.

Waarom zie je er maar één?
VMware heeft de webinterface (vSphere Client) zo geprogrammeerd dat deze twee bestanden als één enkel icoontje worden getoond. Dit is om het overzichtelijk te houden voor de beheerder. Alleen als je via de opdrachtregel (SSH) naar de opslag kijkt, zie je ze allebei staan.

Waarom zijn de andere opties fout?

- **Optie 1:** De configuratie van een VM staat in een **.vmx** bestand, niet een .vmc.
- **Optie 2:** Fout, want het hoofdbestand (.vmdk) is een tekstbestand (plain text), geen binair bestand.
- **Optie 4:** Fout, want VMDK-bestanden bevatten **niet** de configuratie (dat doet de .vmx). Bovendien hoeven ze niet altijd gigabytes groot te zijn; een nieuwe, lege "thin provisioned" schijf kan heel klein beginnen.