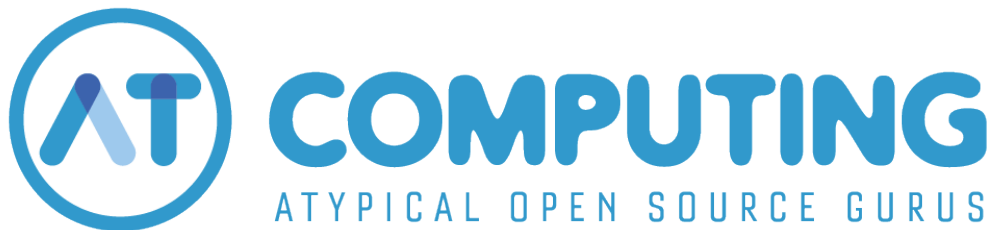


Fun with Bash & Azure



- Director @ AT Computing
- Trainer/consultant
- Open Source Enthusiast
- 10+ years in IT
- Amateur Bash-scripter

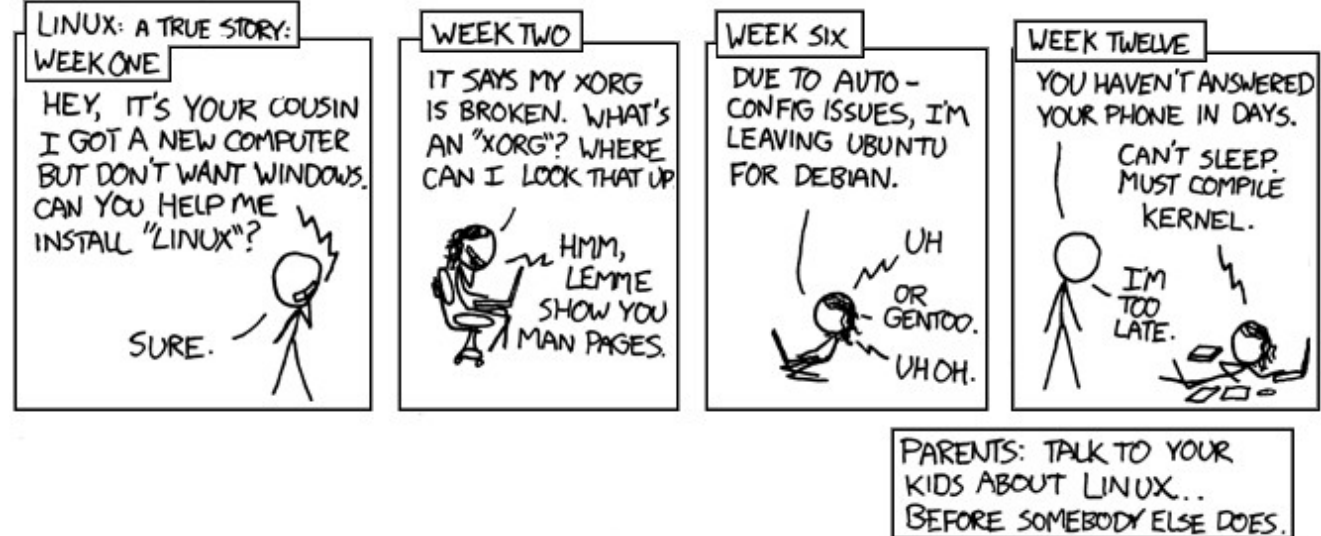
Why am I here?

- In 2019, AT Computing moved to the cloud
- The team chose Google Cloud Platform
- I firmly believe a director should keep learning
- So I decided to challenge myself
- That got out of hand a (tiny little) bit...



Why bash and Azure?

- I give demos during training courses and workshops
- I got bored of manually repeating stuff during preparation
- I wanted to give my team some resistance in the field of cloud
- Because of XKCD456



- What makes Open Source fanatics do this stuff?
- What makes mankind do this?
- What makes us, humans, so incredibly curious?

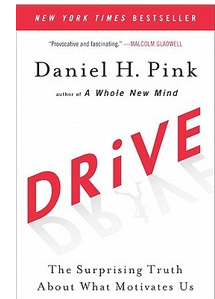
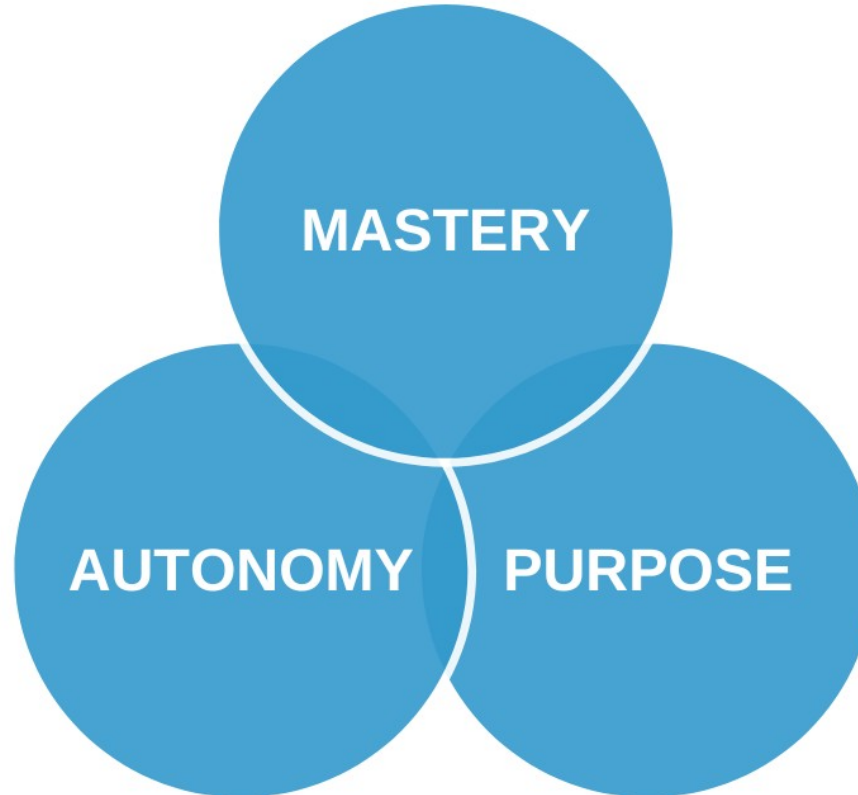
- Curiosity is a fundamental property of (intelligent) organisms
- Organisms have the innate habit of exploring the world around them
- It is at the core of our ability to learn
- Curiosity and exploration improve intelligence
- Curiosity influences decision making and our mental health
- We are not hunting for food anymore, so we have lots of time to be curious
- It seems to be (tightly) related to motivation

What is our motivation?



Why are we motivated?

INTRINSIC MOTIVATION



Why is it so important to us?

- “Curiosity is defined as a need, thirst or desire for knowledge”^π
- There is not too much research done in this area
- Various researchers contradict each other
- It’s hard to measure curiosity on it’s own (in a lab)
- We know what it is, but it’s hard to define it in a scientific way
- It might have to do with our changes of survival (because of adaptation skills)

What curiosity leads to...



Level #1: just underneath the grass

Start using a human friendly GNU/Linux distribution



Level #2: digging deeper into the crust

Use the terminal every now and then



Level #3: reaching the mantle

Using the CLI starts to feel comfortable



BASH
THE BOURNE-AGAIN SHELL

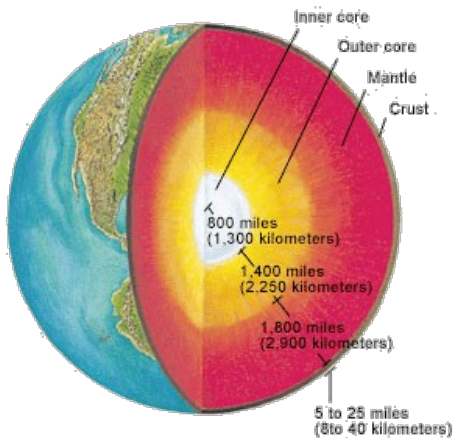
Level #4: drilling into the outer core

writing `#!/bin/bash` for the very first time

Level #5: getting burned to ashes at the inner core

```
$ declare suffering="AAAAAAAAAAAAARRRGGGHH"
```

```
$ for i in {0..∞}; do printf "%s\n" "$suffering"; done
```

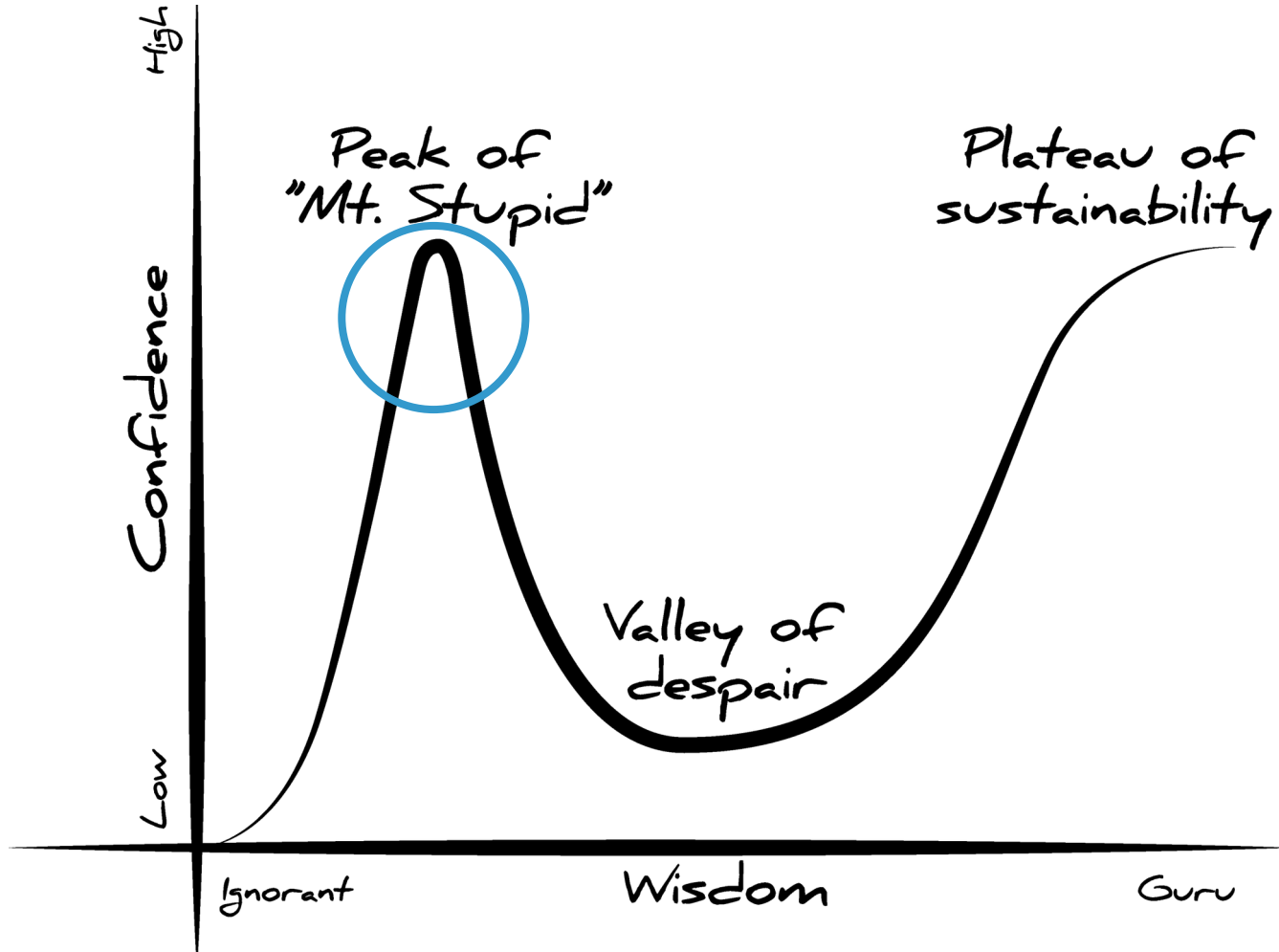


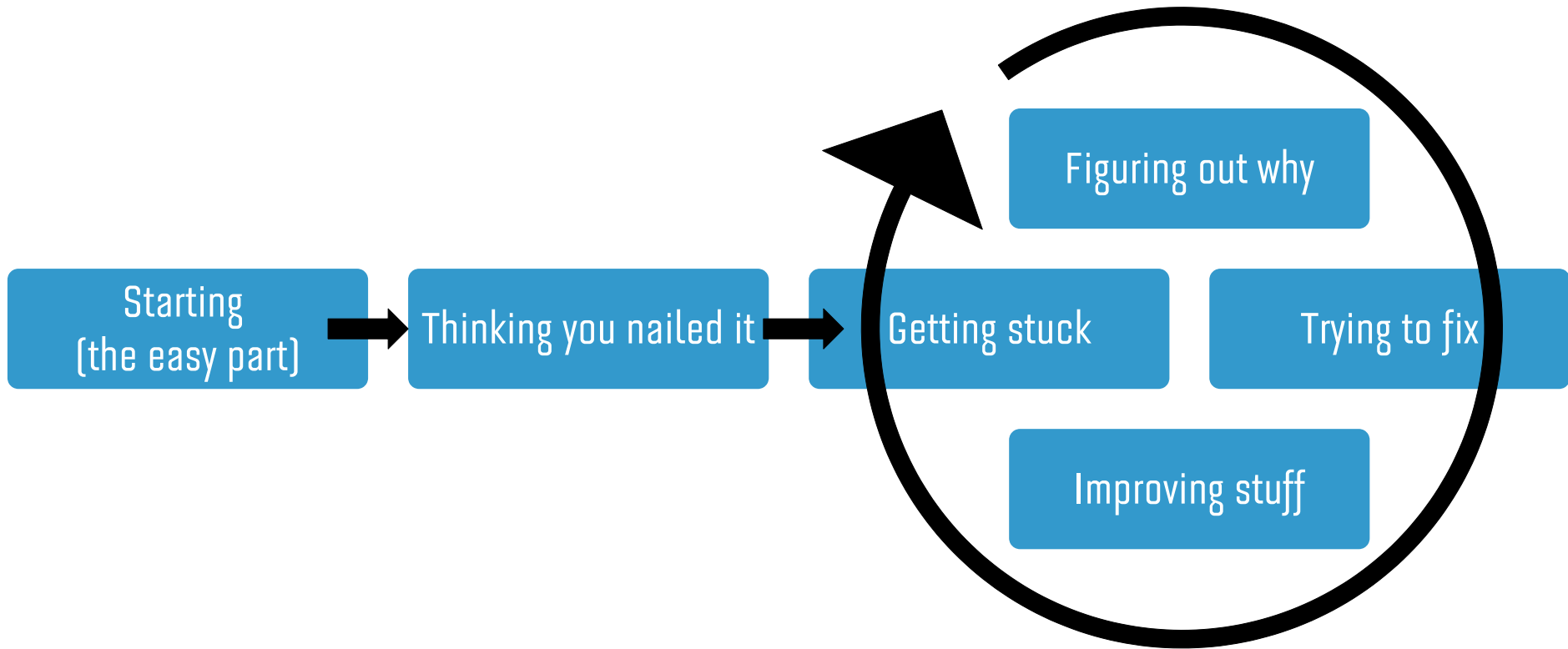
Starting
(the easy part)



Thinking you nailed it

Dunning-Kruger





Congratulations: you've discovered the infinite loop.

- Deploy a simple html/css webapp
- In a container
- On a Virtual Server
- In Microsoft Azure

"Why don't you just automate that?"
- random manager

I use Microsoft Azure for this presentation,
but my rant goes way beyond.

It also has GCP, AWS and (many) other cloud
providers in scope.

And also also automation in general...

Provisioning a virtual server in Azure using **bash** is easy.

```
$ az vm create \  
  --name "t-dose" \  
  --resource-group "t-dose" \  
  --image "Ubuntu2204" \  
  --size "Standard_b1ls"
```

```
#!/bin/bash
az vm create \
  --name "t-dose" \
  --resource-group "T-Dose" \
  --admin-username "tdose" \
  --image "Ubuntu2204" \
  --public-ip-address-dns-name "tdose" \
  --size "Standard_b1ls" \
  --ssh-key-values ~/.ssh/id_rsa.pub
az vm open-port --resource-group "T-Dose" --name "t-dose" --port 80
ssh tdose@tdose.westeurope.cloudapp.azure.com -o StrictHostKeyChecking=no <<EOF
  sudo apt update
  sudo apt-get install docker.io --assume-yes
  git clone https://github.com/Strobodov/T-Dose.git
  sudo docker build App/ --tag=mywebapp
  sudo docker run -d -p 80:80 mywebapp
EOF
firefox http://tdose.westeurope.cloudapp.azure.com
```

DEMO

Random manager was right!

- End of presentation -

Before it's easy, there are (many) prerequisites...

- Create an Azure account with sufficient privileges to...
- Create a Service Principal (because that's a best practice)
- Login with Service Principal via the CLI
- Set correct tenant and/or subscription
- Create a Resource Group
- Figure out which flags you need and which are mandatory...

Now it looks like this to create a VM...

```
$ az login --service-principal \  
  --username "<SOME GUID>" \  
  --password "<NOT GONNA TELL YOU>" \  
  --tenant "<SOME GUID>"
```

```
$ az group create \  
  --name "t-dose" \  
  --location "westeurope"
```

```
$ az vm create \  
  --name "t-dose" \  
  --resource-group "t-dose" \  
  --image "Ubuntu2204" \  
  --size "Standard_b1ls" \  
  --generate-ssh-keys \  
  --admin-username "tdose"
```

This leads to using variables pretty soon...

- Creating a variable in Bash is easy: `$ VARIABLE=$(date)`
- What about storing and using credentials in a script?
- What about secure scripting practices?
- What if you need some variable value you don't know yet?




warning: this is the automation point of no return

Example: stuff needed to automate login

- Service Principal Application ID
- Certificate for authentication
- Tenant ID

Stuff to consider at this point:

- learning more about the Bash syntax
- making use of version control
- diving deep(er) into Azure CLI documentation

- Never put credentials directly in your Bash script
- Set strict file permission(s) like **600** to sensitive stuff
- Source credentials into **readonly** variables
- When using version control software like **git**
 - Make use of **.gitignore** or..
 - Change **.git/info/index/exclude**
 - Be extra carefull with things like:
`$ git commit -am "yolo" && git push (--> github )`

```
$ chmod 600 .spCred
```

```
source .spCred
readonly SP_APPID
readonly SP_CERTNAME
readonly TENANT_ID
readonly SUBSCRIPTION
```


Now it looks like this to create a VM...

```
$ source ".spCred"  
$ readonly SP_APPID  
$ readonly SP_CERTNAME  
$ readonly TENANT_ID  
$ readonly SUBSCRIPTION
```

```
$ az login --service-principal \  
  --username "$SP_APPID" \  
  --password "$CERTNAME" \  
  --tenant "$TENANT_ID"
```

```
$ az group create \  
  --name "$GROUPNAME" \  
  --location "$LOCATION"
```

```
$ az vm create \  
  --name "$VMNAME" \  
  --resource-group "$GROUPNAME" \  
  --image "Ubuntu2204" \  
  --size Standard_b1ls \  
  --generate-ssh-keys \  
  --admin-username "$ADMIN"
```

But what if...

it doesn't work...

- Make sure you test whether execution of a command has succeeded
- **If** statements are great for this
- Make sure you write clear error messages (including line number)
- Make sure you provide exit codes if things break...

```
#login with service principal
if ! az login --service-principal \
    --username "$SP_APPID" \
    --password "$SP_CERTNAME" \
    --tenant "$TENANT_ID" \
    --output none;
then
    printf "\r\e[31mERROR - LINE %s: logging in as Service Principal failed. \e[0m\n" "$LINENO"
    exit 1;
else
    printf "\rlogging in as Service Principal = \e[32mOK \e[0m\n"
    #make sure you use the right subscription
    if ! az account set \
        --subscription "$SUBSCRIPTION"
    then
        printf "\r\e[31mERROR - LINE %s: setting subscription to %s failed. \e[0m\n" "$LINENO" "$SUBSCRIPTION"
        exit 1;
    else
        if [[ $(az account show --query name --output tsv) -eq "$SUBSCRIPTION" ]]
        then
            printf "\rsetting subscription to %s = \e[32mOK \e[0m\n" "$SUBSCRIPTION"
        else
            printf "\r\e[31mERROR - LINE %s: subscription has changed, but not to the desired value. \e[0m\n" "$LINENO"
            exit 1;
        fi
    fi
fi
```

- Check integrity of software you want to use / install
- E.g. you can use **gpg** or **sha256sum** to check/compare digests
- Make sure you use official sources where possible
- **docker** and **podman** provide flags for digest checking

```
$ az acr repository show \
  --name "$REGISTRYNAME" \
  --image "${IMAGENAME}:${IMAGETAG}" \
  --query "digest"
```

```
$ sudo docker images --digests
```

REPOSITORY	TAG	DIGEST	IMAGE ID	CREATED	SIZE
httpd	latest	sha256:a182ef2350699f04b8f8e736747104eb273e255e818cd55b6d7aa50a1490ed0c	4b7fc736cb48	10 days ago	145MB

Getting info & output formatting

- We have created some resources (resource group, virtual machine)
- We got some output containing the IP-address of the server
- We want to use that output in the **ssh** command for automated login
- By default, the **az** command spits out in JSON format and...
- **az vm get-instance-view** doesn't show the public IP address tough...

```
$ az vm get-instance-view \  
  --name "t-dose" \  
  --resource-group "t-dose"
```

```
$ PUBIP=$(az network public-ip show \  
  --name "t-dosePublicIP" \  
  --resource-group "t-dose" \  
  --output tsv \  
  --query ipAddress)
```

Back to the fun



Azure Tenant
AT Computing



Subscription "Speeltuín"



Resource
Group
"T-Dose"



podman



fedora





Azure Tenant
AT Computing



Subscription "Speeltuín"



Resource
Group
"T-Dose"

Virtual
Network



Azure
DNS

Virtual
Machine



Ubuntu
docker

Azure
Container
Registry



das globale
Internet

<http://tdose.westeurope.cloudapp.azure.com>



podman



fedora





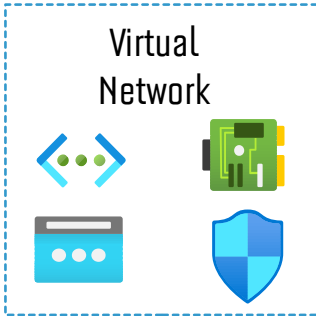
Azure Tenant
AT Computing



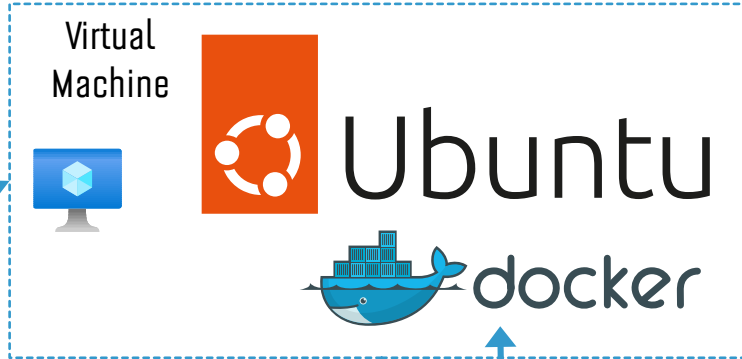
Subscription "Speeltuín"



Resource
Group
"T-Dose"



Azure
DNS



Virtual
Machine



Ubuntu
docker



Azure
Container
Registry

Pull image

Push image



das globale
Internet

<http://tdose.westeurope.cloudapp.azure.com>



podman



fedora



- In the cloud, nearly everything costs money
- Make sure you write code to tear down everything you build up
- The CLI tools of the major cloud provides help you do this

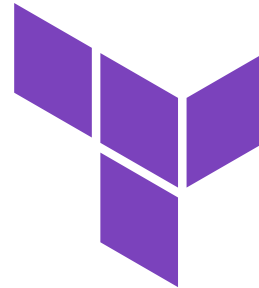
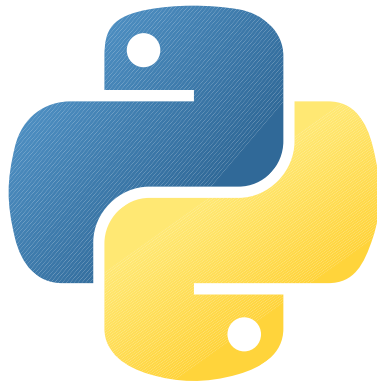
```
$ RES_ID=$(az resource list \  
  --resource-group "t-dose" \  
  --query "[]\.id" \  
  --output tsv)
```

```
$ az resource delete \  
  --id "$RES_ID"
```

```
$ az group delete \  
  --name "t-dose" \  
  --yes
```

The next level?

- It's April 23, 2023
- My Bash "program" is now over 500 lines long (and counting)
- It still doesn't work the way I would like it to...
- It has been fun, educational and it took sleep
- I've gone down the rabbit hole. Maybe now it's time to look up...



HashiCorp
Terraform

<https://github.com/Strobodov/T-Dose>

[linkedin.com/in/mkornegoor](https://www.linkedin.com/in/mkornegoor)

aHR0cHM6Ly9nb29nbGUuZ2l0aHVlLmlvL3N0eWxlZ3VpZGUvc2hlbGxndWlkZS5odG1sCmh0dHBz
Oi8vd3d3LnNoZWxsY2hlY2submV0LwpodHRwczovL2RldmhpbnRzLmlvL2Jhc2gKaHR0cHM6Ly9k
ZXZlbG9wZXJzLnJlZGhhdC5jb20vY2hlYXQtc2hlZXRzL2Jhc2gtc2hlbGwtY2hlYXQtc2hlZXQK
aHR0cHM6Ly9sZWYybi5taWNyY3NvZnQuY29tL2VuLXVzL2NsaS9henVyZS9yZWZlcmVuY2UtaW5k
ZXgKaHR0cHM6Ly9oYnIub3JnLzIwMTgvdmdkvdGhlLWZpdmdUtZGltZW5zaW9ucy1vZi1jdXJpb3Np
dHkKaHR0cHM6Ly93d3cubmNiaS5ubG0ubmloLmdvdi9wbWMvYXJ0aWNsZXNvUE1DNDYzNTQ0My8K
aHR0cHM6Ly93d3cubmNiaS5ubG0ubmloLmdvdi9wbWMvYXJ0aWNsZXNvUE1DMjUzMzU4OS8KaHR0
cHM6Ly93d3cuc2NpZW5jZW5jZWRpcmdC5jb20vc2NpZW5jZS9hcnRyY2x1L3BpaS9TMDg5NjYyNzMx
NTAwNzY3OQpodHRwOi8vd3d3LmNzdW4uZW5jZWR1L352Y3BzeTAwaC9zdHVkZW50cy9leHBsb3JlLmh0
bQo=